UNAPRIJEDENI MODEL DODJELJIVANJA PARALELNIH PROGRAMA RAZNORODNIM RASPODIJELJENIM RACUNALNIM SUSTAVIMA

Krpić, Zdravko

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET OSIJEK

Doktorska disertacija

UNAPRIJEĐENI MODEL DODJELJIVANJA PARALELNIH PROGRAMA RAZNORODNIM RASPODIJELJENIM RAČUNALNIM SUSTAVIMA

Zdravko Krpić

Osijek, 2014.

Doktorska disertacija je izrađena na Elektrotehničkom fakultetu Osijek, Sveučilišta J. J. Strossmayera u Osijeku.

Mentor: Prof. dr. sc. Goran Martinović, redoviti profesor

Sumentor: Prof. dr. sc. Ivica Crnković, redoviti profesor

Disertacija ima 178 stranica

Disertacija broj: 45

Povjerenstvo za ocjenu doktorske disertacije

1. Prof. dr. sc. Goran Martinović, Elektrotehnički fakultet Osijek
2. Prof. dr. sc. Ivica Crnković, Elektrotehnički fakultet Osijek
3. Prof. dr. sc. Željko Hocenski, Elektrotehnički fakultet Osijek
4. Prof. dr. sc. Mario Žagar, Fakultet elektrotehnike i računarstva Zagreb
5. Prof. dr. sc. Maja Štula, Fakultet elektrotehnike, strojarstva i brodogradnje Split

Povjerenstvo za obranu doktorske disertacije

1. Prof. dr. sc. Goran Martinović, Elektrotehnički fakultet Osijek
2. Prof. dr. sc. Željko Hocenski, Elektrotehnički fakultet Osijek
3. Prof. dr. sc. Mario Žagar, Fakultet elektrotehnike i računarstva Zagreb
4. Doc. dr. sc. Alfonzo Baumgartner, Elektrotehnički fakultet Osijek
5. Doc. dr. sc. Josip Job, Elektrotehnički fakultet Osijek

Datum obrane disertacije: 16. listopada 2014. godine

Mojoj obitelji

Predgovor

Ova disertacija nastala je u suradnji Elektrotehničkog fakulteta u Osijeku s odjelom programskog inženjerstva istraživačkog sjedišta za sustave stvarnog vremena Mälardalen sveučilišta u Švedskoj. Dio istraživanja na kojem se temelji ovaj rad napravljen je u okviru projekta švedske zaklade za strateška istraživanja u razvoju programske podrške za ugrađene arhitekture visokih performansi RALF3.

Zahvaljujem se mentoru prof. dr. sc. Goranu Martinoviću na podršci i strpljenju tijekom mog posijediplomskog studija te prof. dr. sc. Ivici Crnkoviću za vrijedne savjete i omogućavanju znanstvene suradnje s MDH-om (Mälardalens Högskola) u Švedskoj. Također, veliko hvala Aneti na nesebičnoj potpori prilikom istraživačkog rada na kojem se zasniva velik dio ove disertacije.

Najveće pak hvala mojoj djevojci Sandri na svoj ostaloj potpori, strpljenju, toleranciji i vjeri u mene za vrijeme kada mi je to bilo najpotrebnije.

Sadržaj

Predgovor

Popis slika **v**

Popis slika **v**

Popis tablica **ix**

Popis tablica **ix**

Popis algoritama **x**

Popis kratica **xi**

Popis simbola i oznaka **xiv**

1 Uvod **1**

1.1 Motivacija . . . 3

1.2 Ciljevi istraživanja . . . 4

1.3 Izvorni znanstveni doprinosi . . . 5

1.4 Pregled po poglavljima . . . 6

2 Osnovni pojmovi **8**

2.1 Računarstvo visokih performansi . . . 8

2.1.1 Sklopovske karakteristike računalnih sustava visokih performansi . . . 10

2.1.2 Programske karakteristike računalnih sustava visokih performansi . . . 11

2.2 Modeliranje sklopovlja i programa u domeni računarstva visokih performansi . . . 12

2.2.1 Modeliranje paralelnih programa . . . 12

2.2.2 Modeliranje sklopovlja . . . 18

2.2.3 Modeliranje pokazatelja kvalitete . . . 22

2.2.4 Energija kao kriterij raspoređivanja programa na računalne sustave visokih performansi . . . 23

2.3 Postojeći postupci raspoređivanja programa na računalne sustave visokih performansi . . . 25

2.3.1 Postupci raspoređivanja na temelju rangiranih listi jedinica programa . . . 28

2.3.2 Iterativni (metaheuristički) i bio-inspirirani postupci raspoređivanja 31
2.3.3 Hibridni i ostali postupci raspoređivanja 32
2.3.4 Nedostatci postojećih postupaka raspoređivanja 33
2.3.5 Postupak raspoređivanja HEFT 35
2.3.6 Postupak raspoređivanja LDCP 38

3 Programski okvir LOSECO **43**
3.1 Model izvršavanja programa 46
3.2 Model sklopovlja 48
3.3 Model raspoređivanja 52

4 Unaprijeđeni postupci raspoređivanja **55**
4.1 Unaprijeđeni postupak raspoređivanja RaPPaMaG 56
4.1.1 Korak 1 postupka RaPPaMaG: particioniranje miješanoga acikličkoga grafa 57
4.1.2 Korak 2 postupka RaPPaMaG: raspoređivanje faza izvršavanja 60
4.1.3 Primjer 1: izvođenje postupka RaPPaMaG 62
4.1.4 Specifičnosti ulaznih parametara za postupak RaPPaMaG 67
4.2 Unaprijeđeni postupak raspoređivanja RaNDKiP 69
4.2.1 Raznorodno rangiranje izvršnih jedinica programa 70
4.2.2 Primjer 2: raznorodno rangiranje jedinica programa postupkom RaNDKiP 71
4.2.3 Raspoređivanje komunikacije 75
4.2.4 Labavljenje vremenskih isječaka skaliranjem frekvencije 78

5 Eksperimentalna analiza **95**
5.1 Alati za generiranje i vrednovanje testnih slučajeva 96
5.1.1 TGFF 96
5.1.2 Radna opterećenja Pegasus 100
5.1.3 Zapis grafova u opisnom jeziku DOT 102
5.1.4 Razvijeni alati: generatori testnih slučajeva DAGGer, DAGGer-S i DAGGer-P . . 104
5.1.5 Razvijeni alati: simulacijsko okruženje SynGraph 108
5.2 Postavke testnih slučajeva (eksperimenta) 110
5.3 Mjerene veličine i pokazatelji kvalitete 118
5.3.1 Pokazatelji kvalitete vremenske analize 118
5.3.2 Pokazatelji kvalitete analize utroška energije 121
5.4 Rezultati vremenske analize postupka RaNDKiP 121
5.4.1 Slučaj 1: raspoređivanje programa generiranih alatom DAGGer na platforme s homogenim komunikacijskim vezama 122
5.4.2 Slučaj 2: raspoređivanje programa generiranih alatom DAGGer na platforme s raznorodnim komunikacijskim vezama 126

5.4.3 Slučaj 3: raspoređivanje programa iz repozitorija Pegasus na platforme s raznorodnim komunikacijskim vezama . . . 130
5.4.4 Slučaj 4: raspoređivanje programa generiranih alatom TGFF na platforme s raznorodnim komunikacijskim vezama . . . 135
5.5 Rezultati analize uštede energije primjenom postupka RaNDKiP . . . 140
5.5.1 Slučaj 5: energetske karakteristike programa generiranih alatom DAGGer raspoređenih na platforme s raznorodnim komunikacijskim vezama . . . 141
5.5.2 Slučaj 6: energetske karakteristike programa generiranih alatom DAGGer raspoređenih na platforme s homogenim komunikacijskim vezama . . . 144
5.5.3 Slučaj 7: energetske karakteristike programa iz repozitorija Pegasus raspoređenih na platforme s raznorodnim komunikacijskim vezama . . . 145
5.5.4 Slučaj 8: energetske karakteristike programa generiranih alatom TGFF raspoređenih na platforme s raznorodnim komunikacijskim vezama . . . 147
5.6 Rezultati analize postupka RaPPaMaG . . . 150
5.6.1 Slučaj 9: primjena postupka RaPPaMaG na programe iz repozitorija Pegasus . . . 150
5.6.2 Slučaj 10: primjena postupka RaPPaMaG na programe iz repozitorija TGFF . . . 152
5.6.3 Slučaj 11: primjena postupka RaPPaMaG na program sa sinkronom komunikacijom na miješani aciklički graf . . . 154
5.7 Skupna analiza eksperimentalnih rezultata . . . 160

6 Zaključak **164**

Bibliografija **166**

Sažetak i ključne riječi **175**

Abstract and keywords **176**

Acknowledgement **177**

Životopis **178**

Popis slika

2.1 Primjer analize traga izvršavanja paralelnog programa upotrebom alata VampirTrace. . . 13
2.2 Primjer ručno izrađenog modela DAG-a programa. 16
2.3 Primjeri modela DAG-a. 16
2.4 Primjer izrade modela TIG-a programa. 17
2.5 Prikaz dostupnih modela sklopovlja HPC-a prema razini apstraktnosti. 19
2.6 Prikaz sklopovlja koji daje alat HWLOC. 20
2.7 Najčešća hijerarhijska struktura sklopovlja sustava HPC-a prema dostupnoj literaturi. . . 21
2.8 Slojevi "ozelenjivanja" sustava HPC-a prema [1]. 24
2.9 Tehnologije "ozelenjivanja" sustava HPC-a po slojevima prema [1]. 24
2.10 Podjele postupaka raspoređivanja i dodjeljivanja prema dostupnoj literaturi. 26
2.11 Postupci raspoređivanja iz pokrivene literature. 28
2.12 Primjer rangiranja jedinica programa kod postupaka raspoređivanja. 30
2.13 Primjer ulaznih informacija za postupak HEFT. 36
2.14 Faza dodavanja jedinice programa u raspored s i bez pravila umetanja. 38
2.15 Primjer zadanog programa za postupak LDCP. 39
2.16 DAGP za procesore p_0 i p_1 ($DAGP_0$ i $DAGP_1$) primjera sa slike 2.15a. 40
2.17 $DAGP_0$ i $DAGP_1$ nakon raspoređivanja n_0, n_1 i n_2. 42

3.1 Osnovna struktura programskog okvira LOSECO. 44
3.2 Detaljna struktura programskog okvira LOSECO. 45
3.3 Primjer modela izvršavanja programa u obliku miješanoga acikličkoga grafa. 48
3.4 Matrica susjedstva modela sa slike 3.3. 48
3.5 Primjer strukture računalnog čvora dobivene alatom HWLOC. 51
3.6 Model računalnog čvora prikazanog slikom 3.5. 51
3.7 Matrica komunikacije računalnog čvora sa slike 3.5 opisanog modelom na slici 3.6. 52

4.1 Primjer particioniranja programa u obliku miješanoga acikličkoga grafa. 60
4.2 Primjer izvršavanja faza programa. 62
4.3 Model programa. 63
4.4 Model sklopovlja. 63
4.5 Program u obliku DAG-a koji treba rasporediti i njegovi DKP-ovi. 71

4.6 Platforma s homogenim komunikacijskim vezama na koju se raspoređuje program sa slike 4.5. . . . 72
4.7 Iznos DKP-ova prije raspoređivanja (homogene komunikacijske veze sklopovlja). . . . 72
4.8 Iznos DKP-ova nakon raspoređivanja v_0, v_1 i v_2 (homog. platforma). . . . 73
4.9 Platforma s raznorodnim komunikacijskim vezama. . . . 74
4.10 Iznos DKP-ova prije raspoređivanja, RaNDKiP (raznorodna platforma). . . . 74
4.11 Iznos DKP-ova nakon raspoređivanja v_0, v_1 i v_2, RaNDKiP (raznorodna platforma). . . . 75
4.12 Primjer raspoređivanja u ovisnosti o komunikaciji. . . . 76
4.13 Raspored primjera za RaNDKiP sa slike 4.5 (bez rasporeda komunikacije). . . . 77
4.14 Raspored primjera RaNDKiP sa slike 4.5 (s rasporedom komunikacije). . . . 77
4.15 Primjer skaliranja frekvencije za obradbeni isječak. . . . 81
4.16 Postupak SKO. . . . 81
4.17 Postupak DiL: primjer preklapajućeg prostora labavljenja. . . . 84
4.18 Primjer postupka RKO. . . . 89
4.19 Primjer postupka LVI: program. . . . 89
4.20 Primjer postupka LVI: dobiveni raspored programa. . . . 90
4.21 Primjer postupka LVI: raspored programa nakon postupka SKO. . . . 90
4.22 Primjer postupka LVI: raspored programa nakon postupka DiL. . . . 91
4.23 Primjer postupka LVI: raspored programa nakon postupka RKO. . . . 92

5.1 Primjer konfiguracijske datoteke *packets.tgffopt* za TGFF. . . . 97
5.2 Graf programa u *.tgff obliku dobiven TGFF-om i konfiguracijom sa slike 5.1. . . . 97
5.3 Graf opisan jezikom VCG dobiven TGFF-om i konfiguracijom "simple.tgffopt". . . . 98
5.4 Prikaz grafa programa opisanog datotekom "simple.vcg" prikazanoj na 5.3 alatom "aisee". 99
5.5 Prikaz grafa radnog opterećenja CyberShake, prema [2]. . . . 101
5.6 Prikaz sadržaja dijela datoteke DAX "CyberShake_30.xml" preuzete s [2]. . . . 102
5.7 Primjer miješanoga grafa u opisnom jeziku DOT. . . . 103
5.8 Vizualizacija grafa opisanog datotekom DOT sa slike 5.7. . . . 104
5.9 Primjer grafa programa generiranog alatom DAGGer. . . . 105
5.10 Primjer strukture platforme generirane DAGGer-P-om. . . . 107
5.11 Program raspoređen postupkom RaPPaMaG u izvođenju kroz okruženje SynGraph. . . . 110
5.12 Skica postupka DoSK (DoSK1 i DoSK2). . . . 116
5.13 Testni slučajevi generirani alatom DAGGer za različit ϑ, homogena komunikacija. . . . 123
5.14 Testni slučajevi generirani alatom DAGGer za različit ρ, homogena komunikacija. . . . 123
5.15 Testni slučajevi generirani alatom DAGGer za različit η, homogena komunikacija. . . . 125
5.16 Testni slučajevi generirani alatom DAGGer za različit *ccr*, homogena komunikacija. . . . 125
5.17 Testni slučajevi generirani alatom DAGGer za različit ϑ, raznorodna komunikacija. . . . 127
5.18 Testni slučajevi generirani alatom DAGGer za različit ρ, raznorodna komunikacija. . . . 128
5.19 Testni slučajevi generirani alatom DAGGer za različit η, raznorodna komunikacija. . . . 129

5.20 Testni slučajevi generirani alatom DAGGer za različit *ccr*, raznorodna komunikacija. . . . 130
5.21 Testni slučajevi iz repozitorija Pegasus za različit ϑ, raznorodna komunikacija. 131
5.22 Testni slučajevi iz repozitorija Pegasus za različit ρ, raznorodna komunikacija. 132
5.23 Testni slučajevi iz repozitorija Pegasus za različit η, raznorodna komunikacija. 133
5.24 Testni slučajevi iz repozitorija Pegasus za različit *ccr*, raznorodna komunikacija. 133
5.25 Testni slučajevi Pegasus, različite strukture programa, raznorodna komunikacija. 134
5.26 Testni slučajevi generirani alatom TGFF za različit ϑ, raznorodna komunikacija. 136
5.27 Testni slučajevi generirani alatom TGFF za različit ρ, raznorodna komunikacija. 137
5.28 Testni slučajevi generirani alatom TGFF za različit η, raznorodna komunikacija. 138
5.29 Testni slučajevi generirani alatom TGFF za različit *ccr*, raznorodna komunikacija. 139
5.30 Testni slučajevi generirani alatom TGFF, različite strukture programa, raznorodna komunikacija. 139
5.31 DVFS karakteristike različitih procesora, prema [3, 4]. 140
5.32 Ušteda energije kod testnih slučajeva generiranih alatom DAGGer, platforma s raznorodnom komunikacijom. 142
5.33 Utrošak energije kod testnih slučajeva generiranih alatom DAGGer, platforma s raznorodnom komunikacijom. 143
5.34 Ušteda energije kod testnih slučajeva generiranih alatom DAGGer, platforma s homogenom komunikacijom. 144
5.35 Ušteda energije kod testnih slučajeva iz repozitorija Pegasus, platforma s raznorodnom komunikacijom. 146
5.36 Parametri energije u sustavu s raznorodnom komunikacijom platforme za različite strukture programa iz repozitorija Pegasus. 147
5.37 Ušteda energije kod testnih slučajeva generiranih alatom TGFF, platforma s raznorodnom komunikacijom. 148
5.38 Parametri energije u sustavu s raznorodnom komunikacijom platforme za različite strukture programa generiranih alatom TGFF. 149
5.39 Programi malih razmjera iz repozitorija Pegasus particionirani postupkom RaPPaMaG. . 151
5.40 Programi srednjih razmjera iz repozitorija Pegasus particionirani postupkom RaPPaMaG. 152
5.41 Programi srednjih razmjera iz repozitorija TGFF particionirani postupkom RaPPaMaG. . 153
5.42 Programi vrlo velikih razmjera iz repozitorija TGFF partic. postupkom RaPPaMaG. . . . 154
5.43 Program veličine $\vartheta = 1000$ particioniran postupkom RaPPaMaG (kseries_parallel_1000). . 155
5.44 Primjer grupno sinkronog programa u obliku miješanoga grafa. 155
5.45 Program sa slike 5.44 podijeljen na faze izvršavanja postupkom RaPPaMaG. 156
5.46 Raspored dobiven iscrpnim pretraživanjem. 158
5.47 Raspored dobiven postupkom HEFT. 159
5.48 Skupne vremenske performanse raspoređivanja programa na računalne platforme s raznorodnim komunikacijskim vezama. 161

5.49 Dobiveni utrošci i ušteda energije pri raspoređivanju programa na računalne platforme s raznorodnim komunikacijskim vezama. 162

Popis tablica

2.1 Najčešći pokazatelji kvalitete u sustavima HPC-a i ograničenja prema dostupnoj literaturi. 22

4.1 Raspored faza izvršavanja na obradbenim jedinicama sklopovlja iz primjera 1. 66
4.2 Vremena izvršavanja svih jedinica programa na svim obradbenim jedinicama sklopovlja, matrica troškova 1. 68
4.3 Utrošak energije svih jedinica programa na svim obradbenim jedinicama sklopovlja, matrica troškova 2. 68
4.4 Vremena izvršavanja jedinica programa sa slike 4.5. 72
4.5 Koraci postupka raspoređivanja programa sa slike 4.5. 73
4.6 Vremena izvršavanja programa. 76
4.7 Parametri vremenskih isječaka. 90
4.8 Parametri rasporeda za svaku obradbenu jedinicu nakon postupka LVI. 91
4.9 Parametri vremenskih isječaka nakon postupaka SKP, SKO i DiL. 91
4.10 Parametri vremenskih isječaka nakon cjelokupnog postupka LVI. 92
4.11 Parametri vremenskih isječaka za primjer 3 nakon cjelokupnog postupka LVI. 94

5.1 Različite veličine radnih opterećenja Pegasus koje se mogu preuzeti iz [2]. 101
5.2 Vrijednosti i intervali svih ulaznih parametara korištenih u eksperimentalnoj analizi. . . . 117
5.3 Postavke parametara programa iz repozitorija Pegasus za eksperimentalnu analizu. 118
5.4 Postavke parametara programa generiranih alatom TGFF za eksperimentalnu analizu. . . 118
5.5 Postavke važnijih parametara za slučaj 11. 157
5.6 Različite postavke važnosti pokazatelja kvalitete i dobiveni rezultati (mala razina raznorodnosti parametra y_j^{en}). 159
5.7 Različite postavke važnosti pokazatelja kvalitete i dobiveni rezultati (velika razina raznorodnosti parametra y_j^{en}). 160

Popis algoritama

1 RaPPaMaG, korak 1. 58
2 RaPPaMaG, korak 2. 62
3 Labavljenje vremenskih isječaka - LVI. 79
4 Skaliranje komunikacije i pripravnosti - SKP. 80
5 Stapanje komunikacije i obrade - SKO. 82
6 Dinamičko labavljenje nekritičnih jedinica programa - DiL. 86
7 Razdvajanje komunikacije i obrade - RKO. 87
8 DAGGer. 105
9 SynGraph, glavna funkcija. 109
10 Dodavanje stvarne komunikacije u raspored, prvi dio - DoSK1. 114
11 Dodavanje stvarne komunikacije u raspored, drugi dio - DoSK2. 115
12 Osnovna struktura algoritama raspoređivanja. 116

Popis kratica

ACO	*ant colony optimization*
ADL	*architecture description language*
BLAS	*basic linear algebra subprograms*
CBSE	*component based software engineering*
CCR	*communication-to-computation ratio*
CPLD	*complex programmable logic device*
CPOP	*critical path on processor*
DAG	*directed acyclic graph*
DAGGer	*DAG generator*
DAGP	DAG na obradbenoj jedinici
DAX	*DAG XML*
DCP	*dynamic critical path*
DE	*differential evolution*
DiL	dinamičko labavljenje nekritičnih jedinica programa
DKP	dinamički kritični put
DoSK	dodavanje stvarne komunikacije u raspored
DVFS	*dynamic voltage and frequency scaling*
EA	*evolutionary algorithms*
ECS	*energy-conscious scheduling*
EFP	*extra-functional properties*
EFT	*earliest finish time*
EPEAT	*electronic product environmental assessment tool*
EST	*earliest start time*
FLOPS	*floating point operations per second*
FPGA	*field programmable gate array*
GA	*genetic algorithms*
GPU	*graphic processing unit*
HCP	*heterogeneous computing platform*
HEFT	*heterogeneous estimated finish time*
HPC	*high performance computing*

HPU	*heterogeneous processing unit*
HWLOC	*hardware locality*
IaaS	*infrastructure as a service*
ILP	*integer linear programming*
IoT	*internet of things*
KP	kritični put
LAN	*local area network*
LAPACK	*linear algebra package*
LDCP	*longest dynamic critical path*
LOSECO	*allocation of parallel software to heterogeneous computing platform*
LVI	labavljenje vremenskih isječaka
MARTE	*modeling and analysis of real-time and embedded systems*
MIPS	*million instructions per second*
MoCC	*models of communication and control*
MPI	*message-passing interface*
MPSoC	*multi-processing system-on-chip*
MS	*Microsoft*
NDKP	najdulji dinamički kritični put
NoC	*network on chip*
NSL	*normalized schedule length*
OpenMP	*open multiprocessing*
PaaS	*platform as a service*
PATC	*power aware task clustering*
PE	*processing element*
PEFT	*predict earliest finish time*
POS	*partial optimal slacking*
PSO	*particle swarm optimization*
QA	*quality attribute*
RaNDKiP	raznorodni najduži dinamički kritični put
RaPPaMaG	raspoređivanje programa particioniranjem miješanoga acikličkoga grafa
RKO	razdvajanje komunikacije i obrade
RoHS	*restriction of hazardous substances directive*
SaaS	*software as a service*
SCALAPACK	*scalable linear algebra package*
SEU	*software executable unit*
SKO	stapanje komunikacije i obrade
SKP	skaliranje komunikacije i pripravnosti
SLR	*schedule length ratio*
SoC	*system-on-chip*

SSE	*streaming SIMD extensions*
TGFF	*task graphs for free*
TICG	*task interaction concurrency graph*
TIG	*task interaction graph*
TPL	*task parallel libraries*
TTIG	*temporal task interaction graph*
UML	*unified modeling language*
VCG	*visualization of compiler graphs*
VM	*virtual machine*
WCET	*worst case execution time*
WLAN	*wireless local area network*
WSN	*wireless sensor networks*
XML	*extensible markup language*

Popis simbola i oznaka

A	skup lukova grafa
a_i	luk i, može sadržavati težinu
α	čimbenik komunikacije
B^k	prostor varijabli rješenja
B^n	prostor varijabli dizajna
β	broj nezavisnih varijabli x_i
C	skup svih komunikacijskih veza računalne platforme (HEFT, LOSECO)
$c_{i,j}$	komunikacijska propusnost između obradbenih jedinica p_i i p_j (HEFT, LOSECO)
$\overline{c_{i,j}}$	prosječna vrijednost komunikacijske propusnost između obradbenih jedinica p_i i p_j
c_{max}	maksimalna propusnost komunikacijskoga kanala
$\overline{C}$	prosječna vrijednost propusnosti komunikacijskih veza sklopovlja (HEFT)
ccr	omjer komunikacijskih i obradbenih zahtjeva
χ	broj nezavisnih parametara koji opisuju jedinice programa
$DAGP_j$	DAGP obradbene jedinice p_j (LDCP)
δ	faza izvršavanja (postupak RaPPaMaG)
E	skup neusmjerenih bridova grafa
$e^{uk}(\delta)$	ukupni utrošak energije faze izvršavanja δ (primjer RaPPaMaG)
e_i^{uk}	postrošena energija za izvođenje jedinice programa v_i
$e_{i,k}^j$	iznos komunikacije među jedinicama programa v_i i v_k u $DAGP_j$
$\overline{e_G}$	prosječni komunikacijski zahtjevi grafa G
$E_{i,j}$	skup komunikacijskih faza između jedinica v_i i v_j (TIG)
$e_{i,j} = e(v_i, v_j)$	obujam komunikacije između vrhova grafa v_i i v_j (TIG)
e_i	neusmjereni brid grafa, može sadržavati težinu
ε^{din}	dinamički utrošak energije
ε_{uk}^{raz}	ukupna ušteda energije platforme postupkom LVI
ε_i^{raz}	ušteda energije za vremenski isječak v_i
ε^{stat}	statički utrošak energije
$\varepsilon^{uk} = \epsilon$	ukupni utrošak energije
η	čimbenik raznorodnosti
f_j^{max}	najveća moguća radna frekvencija obradbene jedinice p_j

f_j^{min}	najniža moguća radna frekvencija obradbene jedinice p_j (SKP)
f_j^{op}	radna frekvencija obradbene jedinice p_j (SKP)
g	funkcija dodjeljivanja programa na sklopovlje (postojeći postupci raspoređivanja)
G	graf
$g : \delta \to \lambda$	funkcija raspoređivanja faza izvršavanja (primjer RaPPaMaG)
g_j	funkcija ograničenja j u obliku nejednakosti
$h_{i,j}$	broj komunikacijskih faza između jedinica v_i i v_j (TIG)
h_l	funkcija ograničenja l u obliku jednakosti
HPU_{dost}	skup dostupnih obradbenih jedinica sklopovlja (korak 2 RaPPaMaG)
HPU_{nedost}	skup nedostupnih obradbenih jedinica sklopovlja (korak 2 RaPPaMaG)
k	broj funkcija cilja kod višekriterijske optimizacije
κ	konstanta specifična obradbenoj jedinici
KP_{min}	najkraći kritični put u grafu programa
L	vektor inicijalnog troška početka komunikacije za svaku obradbenu jedinicu (HEFT)
$L(v_i)$	neposredna okolina jedinice programa v_i u DAG-u (DiL)
$\overline{L}$	prosječna vrijednost inicijalnog troška početka komunikacije (HEFT)
λ_k	podskup skupa obradbenih jedinica za izvršavanje faze δ_k (primjer RaPPaMaG)
m	broj mogućih rješenja pri raspoređivanju postupkom RaPPaMaG
M	skup svih testiranih postupaka raspoređivanja
m_i	postupak raspoređivanja i
N	skup vrhova koji čini LDCP iz kojih se raspoređuju jedinice programa (LDCP)
n	veličina faze izvršavanja (primjer RaPPaMaG)
n_i	veličina faze izvršavanja δ_i (primjer RaPPaMaG)
o_1	broj ograničenja u obliku nejednakosti
o_2	broj ograničenja u obliku jednakosti
ω	broj nezavisnih parametara koji opisuju obradbene jedinice sklopovlja
P	skup svih obradbenih jedinica sklopovlja
p_j	obradbena jedinica sklopovlja j
PE	obradbeni element
$Per(i, j)$	permutacija bez ponavljanja odabranih j elemenata od ukupno i elemenata
φ^{din}	dinamička snaga (LVI)
$pred(v_i)$	skup neposrednih prethodnika jedinice programa v_i
Ψ_k	skup svih komunikacijskih isječaka odgovarajućeg obradbenog isječka s'_k
q	broj elemenata skupa Q (DiL)
$Q(p_j)$	skup vremenskih isječaka s preklapajućim prostorom labavljenja za obradbenu jedinicu p_j
$\mathbf{q}$	vektor varijabli dizajna (parametara koji opisuju programe i sklopovlje)
$\mathbf{q}_z$	vektor parametara jednog mogućeg rasporeda programa na sklopovlju (primjer RaPPaMaG)
r_i	broj obradbenih faza jedinice programa v_i (TIG)
$\rho = \|P\|$	broj obradbenih jedinica sklopovlja

$RRang_{i,j}$	rastući rang jedinice programa v_i u DAG-u obbradbene jedinice p_j, $DAGP_j$
S	skup vremenskih isječaka rasporeda (SKP)
s	vremenski isječak iz rasporeda (SKP)
s_{r+1}	vremenski isječak koji slijedi nakon isječka s_r
S'_j	skup vremenskih isječaka polaznog rasporeda obradbene jedinice p_j (prije postupka SKO) (RKO)
S_j	skup vremenskih isječaka rasporeda obradbene jedinice p_j (SKP)
s'_p	prvi komunikacijski isječak odgovarajućeg obradbenog isječka
$sljed(v_i)$	skup nedposrednih sljedbenika jedinice programa v_i
$sljed_j(v_i)$	skup neposrednih sljedbenika vrha v_i u $DAGP_j$
t_j^{dost}	najraniji trenutak kada je obradbena jedinica p_j dostupna za izvršavanje jedinice programa (HEFT)
t^{int}	interval protezanja (od najranijeg vremena početka prvog elementa do najkasnijeg vremena završetka zadnjeg) (LVI)
t^{izv}	vrijeme izvršavanja vremenskog isječka rasporeda (LVI)
$t_{\Psi_k}^{izv}$	ukupno trajanje komunikacije obradbenog isječka s'_k
t_i^{kom}	vrijeme komunikacije jedinice programa v_i (primjer RaPPaMaG)
t_i^{kvp}	najranije vrijeme početka izvršne jedinice v_i (LDCP)
t^{nvp}	najkasnije vrijeme početka vremenskog isječka rasporeda (LVI)
$t_{i,j}^{nvp}$	najkasnije vrijeme početka jedinice programa v_i na obradbenoj jedinici sklopovlja p_j (HEFT)
t^{nvz}	najkasnije vrijeme završetka vremenskog isječka rasporeda (LVI)
$t_{i,j}^{nvz}$	najkasnije vrijeme završetka jedinice programa na obradbenoj jedinici sklopovlja p_j (HEFT)
t_i^{ob}	vrijeme obrade jedinice programa v_i (primjer RaPPaMaG)
t^{pl}	prostor labavljenja vremenskog isječka rasporeda (LVI)
t^{rasp}	ukupna duljina rasporeda
t_i^{svp}	stvarno vrijeme početka jedinice programa v_i (HEFT), vremenskog isječka rasporeda (RaNDKiP)
t_i^{svz}	stvarno vrijeme završetka jedinice programa v_i (HEFT, LDCP), vremenskog isječka rasporeda (RaNDKiP)
$t_{izlaz,i}^{svz}$	stvarno vrijeme završetka izlaznog čvora $v_{izlaz,i}$, ako ih ima više (HEFT)
$t^{uk}(\delta)$	ukupno vrijeme izvršavanja faze δ
t_i^{uk}	ukupno vrijeme izvršavanja jedinice programa v_i
τ^{izv}	vrijeme izvršavanja isječka v_i nakon skaliranja frekvencije (LVI)
$u_i(\mathbf{x})$	i-ta funkcija cilja
$\mathbf{U}(\mathbf{x})$	vektor funkcija cilja
υ	napon obradbene jedinice
V	skup jedinica programa (vrhova grafa)
$v_{izlaz,i}$	izlazni čvor, ako ih ima više (HEFT)

V_{izlaz}	skup izlaznih čvorova grafa (HEFT, DAG)
v_{ulaz}	ulazni vrh grafa, vrh bez prethodnika (HEFT, DAG)
v_i	jedinica programa (vrh grafa) i
ϑ	broj izvršnih jedinica programa
$w(p_i, p_j)$	vrijednost matrice povezanosti za par obradbenih jedinica sklopovlja p_i, p_j
$w(t^{en})$	važnost utroška energije (primjer RaPPaMaG)
$w(t^{uk})$	važnost vremena izvršavanja (primjer RaPPaMaG)
$w(v_i)$	trag koji ostavlja vremenski isječak obradbene jedinice v_i
$w(v_i, v_j)$	vrijednost matrice susjedstva za par jedinica programa v_i, v_j (miješani aciklički graf)
$w_{i,l}$	obradbena faza l jedinice programa v_i (TIG)
w_i	obradbeni zahtjevi jedinice programa v_i
W_i	skup obradbenih faza jedinice programa v_i u modelu TIG-a programa
$\overline{x_G^{ob}}$	prosječni obradbeni zahtjevi grafa G
$\overline{x_i^{ob}}$	prosječni obradbeni zahtjevi izvršne jedinice v_i (prosjek po obradbenim jedinicama sklopovlja)
x_i^{ob}	obradbeni zahtjevi jedinice programa v_i (korak 2 RaPPaMaG)
x^i	nezavisni parametar koji opisuje jedinice programa u miješanom acikličkom grafu
x_i	nezavisna varijabla
$\mathbf{x}$	vektor nezavisnih parametara koji opisuju jedinice programa
$\mathbf{x}_i$	vektor nezavisnih parametara koji opisuju jedinicu programa v_i
y_j^{en}	utrošak energije obradbene jedinice p_j (primjer RaPPaMaG)
y_j^{ob}	obradbena sposobnost obradbene jedinice p_j (primjer RaPPaMaG)
y^i	nezavisni parametar koji opisuje obradbene jedinice sklopovlja
$\mathbf{y}$	vektor nezavisnih parametara koji opisuju jedinice sklopovlja
$\mathbf{y}_i$	vektor nezavisnih parametara koji opisuju obradbenu jedinicu sklopovlja p_i
Z	skup svih faza izvršavanja programa prema postupku RaPPaMaG

Poglavlje 1

Uvod

U današnjem računalno prožetom svijetu računalni resursi predstavljaju ne samo instrument znanosti, inženjerstva i obrazovanja, već su i pokazatelj ekonomskog potencijala pojedinih institucija, pa čak i država. Razvojem tehnologija u svezi s porastom potreba čovječanstva svjetska glad za obradbenom snagom računala neprestano raste.

Iz tog razloga dolazi do razvoja računarstva visokih performansi (eng. *high performance computing*, HPC), grane računarstva koja omogućuje upotrebu podrobnijih modela, a i podrobniji uvid u problematiku koja proizlazi iz mnogih prirodnih znanosti, poput fizike, matematike, kemije, biologije, geologije, meteorologije i dr. Na taj se način uklanjaju barijere uvjetovane računalnom snagom i pruža se pomnija analiza prirodnih pojava, a samim time omogućuje se i njihovo kvalitetnije objašnjavanje. Prema nekim izvorima, primjerice [5], analizom razvijenosti računalnih sjedišta visokih performansi pojedine države, prema tzv. Branscombovoj piramidi [6] ili u novije vrijeme kiberinfrastrukturi, može se ocrtati njezin ekonomski status te na taj način procijeniti njezina važnost u svijetu. Pojavom novih sklopovskih tehnologija računarstva visokih performansi poput mnogojezgrenih procesora (eng. *manycore*), grafičkih čipova opće namjene (eng. *graphic processing unit*, GPU), MPSoC arhitektura (eng. *multi-processing system-on-chip*) te novih oblika ujedinjavanja računalnih resursa, s jedne strane poput oblaka računala, interneta stvari (eng. *internet of things*, IoT) i s druge strane već postojećih grozdova računala (eng. *grid computing systems*) te nakupina računala, (eng. *computing clusters*) stvaraju se potencijalno snažne platforme za obradu podataka, ukoliko se pravilno upotrijebi njihov potencijal. Glavno svojstvo novih tehnologija je visoka razina raznorodnosti i paralelizma, što znači da je naglasak na razvoj tehnologije pojedinih obradbenih jedinica (eng. *processing units*), poput procesora, procesorskih jezgri, procesora toka i sl., sada usmjeren na povećanje njihovog broja i njihove bolje povezanosti (kompaktnosti). Takva struktura sklopovlja nameće novi način obrade (paralelan) zahtijeva nove pristupe u izradi programske podrške (programa). Ali, rezultat je i dalje da strahovit razvoj sklopovskih rješenja nije istom brzinom popraćen razvojem tih programa. U današnje vrijeme velik se trud ulaže u razvoj programa kako bi oni iskoristili pun potencijal računalnih platformi paralelne i raspodijeljene naravi. Posljedica toga je dizajn programa sličan dizajnu sklopovlja za njegovo izvršavanje (paralelni programi). Tako nastaju brojne programske paradigme koje s jedne strane pokušavaju naći što bolji način kako program podijeliti na

manje cjeline, a s druge kako te cjeline rasporediti na obradbene jedinice sklopovlja s ciljem povećanja učinkovitosti njihovog međudjelovanja.

S jedne strane posljedica navedenoga pojava je brojnih oblika prikaza programa poput raznih modela i matematičkih formalizama, a s druge postupaka njihovog **raspoređivanja** na sklopovlje. Prikaz programa u obliku grafova, koji predstavljaju konstitutivne jedinice programa (zadatci, poslovi, procesi, niti itd.) i njihove međuovisnosti (tijek izvođenja ili njihova međusobna komunikacija), najčešći je oblik koji koriste postupci raspoređivanja. Pretežno, to je usmjereni aciklički graf (eng. *directed acyclic graph*, DAG), zatim graf interakcije zadataka (eng. *task interaction graph*, TIG), ali i mnogi drugi. Međutim, navedeni modeli u obliku grafa ne omogućuju istovremeni opis sinkrone komunikacije i vremenskog toka među jedinicama programa, kakvog se može naći u programima koji koriste paradigmu izmjene poruka (eng. *message-passing programs*). Uz to, postojeći modeli podrazumijevaju poznavanje vremena izvršavanja svih jedinica programa na svakoj od obradbenih jedinica sklopovlja, što u slučaju velikog broja jedinica programa i sklopovlja nije lako ispitati.

U grani računarstva visokih performansi pri raspoređivanju programa zadanih u obliku grafova na sklopovlje, prvenstveno se trud usmjerava na povećanje performansi, što je najčešće sinonim za skraćivanje ukupnog vremena izvršavanja programa. Važnost performansi potvrđuje i etabliranost TOP500 rang liste svjetskih superračunala prema njihovoj obradbenoj moći, [7] kao mjerodavne. Osim naglaska na performanse, gorući problem je i velika potrošnja energije sustava HPC-a. Zbog toga se sve češće smanjenje energije postavlja kao drugi cilj odmah poslije performansi, uzrokujući sve veću popularnost GREEN500 liste energetski najučinkovitijih svjetskih superračunala iz [8]. Uz performanse i energetsku učinkovitost nameću se i drugi ciljevi u obliku pokazatelja kvalitete, primjerice ujednačavanje opterećenja računalnog sustava, (eng. *load balancing*) pri izvođenju programa, povećanje skalabilnosti programa, smanjenje memorijskih zahtjeva i povećanje ukupne podatkovne propusnosti, smanjenje troškova, sprječavanje ispada itd.

Potreba za učinkovitom raščlambom programa na manje cjeline za njihov paralelan i raspodijeljen rad na sklopovlju, uz ispunjavanje rastućeg broja ciljeva, uzrokuje razvoj metoda optimizacije programa, u prvom redu postupaka njihovog raspoređivanja. Postupci raspoređivanja programa na sklopovlje obuhvaćaju odabir odgovarajućih obradbenih jedinica sklopovlja za izvršavanje pojedinih jedinica programa, (većinom zadataka ili poslova) na njima, kako bi se postigla određena razina već spomenutih parametara kvalitete, (ukupno vrijeme izvršavanja, utrošak energije, podatkovna propusnost i sl.).

Postoji nekoliko tipova postupaka raspoređivanja. U prvom redu to su razni heuristički i metaheuristički, (većinom iterativni) postupci vrlo uskog spektra primjene, zatim su to postupci raspoređivanja pomoću rangiranih listi jedinica programa (eng. *list scheduling*) te razni hibridni postupci. Mana postojećih postupaka raspoređivanja je njihova vrlo specifična primjena, čime se ostavlja malo mogućnosti za njihovu ponovnu upotrebu u drugim situacijama (prema [9]) te za međusobnu usporedbu istih. Također, NP-složenost problema raspoređivanja, prema [10–13], onemogućuje optimalno rješavanje složenijih sustava (s velikim brojem jedinica programa i/ili obradbenih jedinica sklopovlja), nedostatna je i podrška postupaka raspoređivanja za više kriterija, njihova složenost često je prepreka razumnom trajanju raspoređivanja itd. Također, u većini postojećih postupaka izostaje i puna podrška za raznorodnost sklopovlja

prema riječima autora iz [10, 14–16], kao i mnogih drugih, posebice raznorodnosti komunikacijskih veza među obradbenim jedinicama.

Da bismo ublažili i ukinuli neke od navedenih nedostataka, u ovom radu predstavljamo naše izvorne znanstvene doprinose izvedene pomoću (i) sustavnog prikaza postupka raspoređivanja i dodjeljivanja u obliku programskog okvira LOSECO, koji obuhvaća ulazne modele glavnih entiteta raspoređivanja, a to su programi, sklopovlje i kriteriji raspoređivanja, s prikazom programa u novom obliku — miješanoga acikličkoga grafa koji opisuje sinkronu komunikaciju i vremenski tijek među jedinicama programa, (ii) unaprijeđeni postupak raspoređivanja particioniranjem programa u obliku miješanoga grafa i raspoređivanjem dobivenih particija prilagodljivom višekriterijskom optimizacijom RaPPaMaG, (iii) postupak raspoređivanja na temelju rangiranih listi jedinica programa RaNDKiP, (raznorodni dinamički kritični put) koji, uz skraćivanje rasporeda, sadrži i mehanizam labavljenja vremenskih isječaka, čime se omogućuje smanjenje utroška energije na računalnim sustavima koji podržavaju dinamičko skaliranje frekvencije i napona, (eng. *dynamic voltage and frequency scaling*, DVFS) te (iv) postupke i rezultate vrednovanja unaprijeđenih postupaka raspoređivanja.

Navedenim doprinosima omogućujemo modularnost sustava za raspoređivanje paralelnih programa na sklopovlje, detaljnije modele prikaza programa za njihovo raspoređivanje, podršku za više kriterija pri raspoređivanju, podršku za raznorodnu strukturu komunikacijskih veza sklopovlja i povećanje kvalitete dobivenih rješenja raspoređivanjem.

1.1 Motivacija

Mogućnosti računalnog sklopovlja velikim su dijelom još neistražene. Od kućnih računala pa sve do superračunalnih sjedišta nalaze se računalne tehnologije čiji potencijal se ne upotrebljava u potpunosti. Da bi problem bio izraženiji, obradbene jedinice, (procesorske jezgre, *stream* procesori, rekonfigurabilne obradbene jedinice programabilnih logičkih čipova i sl.), današnjih obradbenih elemenata (procesora, grafičkih čipova itd.), postaju raznorodne (različitih obradbenih, komunikacijskih i drugih sposobnosti). Primjerice, ugradnja grafičkog čipa u procesore opće namjene (Intel "i" s erija, AMD APU) te razvoj posebnih procesora poput Godson i Cell čine nove vrste računalnih platformi na kojima je teško predvidjeti performanse i za koje se programi trebaju temeljito prilagoditi.

Drugi veliki problem povećana je potrošnja električne energije računalnih sustava visokih performansi. Jedna od činjenica koja to pokazuje je i podatak da su 2010. godine podatkovna i obradbena računalna sjedišta bila odgovorna za čak 1,3% svjetske potrošnje električne energije, dok se 2020. godine predviđa čak 8%[1]. Zato se postavlja jedno od glavnih pitanja u području računarstva — "Kako povećati ili zadržati postojeće performanse programa na suvremenom sklopovlju i u isto vrijeme smanjiti njihov energetski trag i općenit utjecaj na okoliš?". S obzirom na veličinu, upravo su sustavi HPC-a ti koji imaju najveći utjecaj. Jedna od rečenica koja opisuje novonastalu situaciju i skoriju budućnost računarstva visokih performansi je i izjava Stevea Scotta, glavnog tehničkog direktora jedinice NVidia Tesla i bivšeg višeg potpredsjednika tvrtke Cray: *"The race to exaflop computing will be a race to energy efficiency."*.

[1] http://www.analyticspress.com/datacenters.html

Dakle, utrka za performansama superračunala pretvara se u utrku za što bolju energetsku učinkovitost takvih sustava, jer bolja energetska učinkovitost, osim smanjenja utjecaja na okoliš, smanjuje i troškove. Također, općenita primjenjivost današnjih računalnih sustava visokih performansi treba biti veća prema riječima Michaela L. Normana, privremenog direktora SDSC-a (eng. *San Diego Supercomputing Center*) — *"Usability is the most important HPC metric!"* i *"Stop idolizing peak petaflops!"*. Iz navedenih izjava nazire se i problem umjetnog povećanja performansi i njihov prikaz specijaliziranim testovima koji ne pokazuju stvarnu sliku današnjih programa. Zbog toga smatramo i smatra se općenito, da je u velikim računalnim sustavima opravdano utrošiti mnogo vremena na optimizaciju programa za izvođenje na takvim sustavima, kao što kažu autori u [16] — *"Information gathering, although time consuming, is negligible comparing to the time saved in mapping of the intensive workloads"*. Dakle, treba slijediti trend učinkovitoga računarstva i u tom smjeru razvijati programske paradigme, pa tako i postupke raspoređivanja paralelnih programa na sklopovlje.

1.2 Ciljevi istraživanja

Glavni izazov našeg istraživanja jest učinkovito korištenje paralelnog i raspodijeljenog sklopovlja od strane programske podrške. Mehanizam kojim to želimo postići, kao i glavno pitanje koje si postavljamo, jest kako učinkovito rasporediti jedinice programa paralelnom i raspodijeljenom sklopovlju i to:

- u što kraćem vremenu,
- uz ispunjavanje više kriterija optimizacije s prioritetima (prvenstveno performansi i utroška električne energije),
- uz odgovarajuće vrednovanje raznorodnosti računalne platforme (njezinih komunikacijskih kanala i računalnih sposobnosti njezinih obradbenih jedinica),
- za različite domene računarstva (računarstvo visokih performansi, ugrađeni računalni sustavi i dr.),
- uz mogućnost provjere dobivenog rješenja?

Dakle, potrebno je izraditi postupke dodjeljivanja i raspoređivanja jedinica programa paralelnom i raspodijeljenom sklopovlju koji pružaju bolja rješenja u odnosu na postojeće postupke, koji skraćuju vrijeme za postizanje tih rješenja i koji podrazumijevaju raznorodnu prirodu glavnih entiteta raspoređivanja — programa i računalne platforme (sklopovlja), posebice komunikacijskih veza među obradbenim jedinicama sklopovlja.

Da bi postupak raspoređivanja programa na paralelne i raspodijeljene računalne platforme bio sustavan, modularan i primjenjiv u različitim domenama računarstva, potrebno je osmisliti i opisati njegove ključne dijelove. Navedeni problem ima nekoliko izazova: (i) na koji način predstaviti glavne entitete raspoređivanja (programi, sklopovlje i kriterije optimizacije), (ii) koju razinu apstraktnosti koristiti kako bi sustav pokrivao više različitih domena računarstva (na primjer, računarstvo visokih performansi, ugrađeni računalni sustavi i dr.), (iii) kako dizajnirati postupke raspoređivanja koji se mogu jednostavno izmjenjivati i (iv) kako omogućiti povratni učinak dobivenih rezultata u svrhu poboljšanja postupka raspoređivanja. Navedeni problemi formiraju prvi cilj našeg istraživanja:

Cilj istraživanja 1: Izraditi univerzalni programski okvir sustava za raspoređivanje programa paralelnom i raspodijeljenom sklopovlju.

Već spomenuti problem jest i predstavljanje glavnih entiteta potrebnih postupku raspoređivanja, programa, sklopovlja i kriterija optimizacije s prioritetima u kvantificiranom, računalno čitkom obliku, formirajući naš drugi cilj istraživanja:

Cilj istraživanja 2: Definirati model izvođenja paralelnih programa koji omogućuje podršku za vremensku i prostornu ovisnost njegovih izvršnih jedinica, njihovu sinkronizaciju te model sklopovlja kao ulaznih entiteta za postupak dodjeljivanja.

Nakon konstruiranja programskog okvira sustava za dodjeljivanje i definicije ulaznih modela, potrebno je izraditi postupke raspoređivanja koji koriste navedene modele, postižu rješenja bliže optimalnima i koji zahtijevaju manje vremena i računalnih resursa. Navedeno formiramo kao treći cilj našeg istraživanja:

Cilj istraživanja 3: Dizajnirati postupke dodjeljivanja i raspoređivanja na osnovu predloženih modela, prvenstveno u svrhu poboljšanja performansi i smanjenja utrošene energije pri izvođenju programa raspoređenog na sklopovlje.

S obzirom na to da modeli sadrže samo najvažnije informacije o glavnim entitetima raspoređivanja, dobiveni rezultati možda neće predstavljati planirani ishod. Nesavršenosti modela poput zagušenja mreže, međudjelovanje operacijskog sustava i utjecaj zanemarenih veličina zajedno mogu imati značajan utjecaj na konačne performanse ukazujući na potrebu za vrednovanjem rezultata raspoređivanja. Za rješavanje tog problema definiramo i posljednji cilj našeg istraživanja:

Cilj istraživanja 4: Izraditi simulacijski okvir za vrednovanje postupaka raspoređivanja programa na paralelno i raspodijeljeno sklopovlje koji se temelje na prethodnim modelima.

1.3 Izvorni znanstveni doprinosi

Glavni doprinosi ove doktorske disertacije su postupci za raspoređivanje izvršnih jedinica programa na paralelno i raspodijeljeno sklopovlje. Kada se preslikaju na zadane ciljeve istraživanja, definiraju se kao:

D.1. **Okvir za optimalno dodjeljivanje i vrednovanje paralelnih programa temeljenih na paradigmi izmjene poruka za raznorodne raspodijeljene računalne sustave visokih performansi.** (Cilj 1)

U sklopu ove doktorske disertacije izradili smo programski okvir LOSECO modularnog sustava za raspoređivanje izvršnih jedinica programa temeljenih na paradigmi izmjene poruka na raznorodne raspodijeljene računalne sustave s pripadajućim ulaznim modelima.

D.2. **Model paralelnih programa u obliku miješanoga grafa i pripadajući algoritam raspoređivanja temeljen na particioniranju grafa.** (Cilj 2 i 3)

Definirali smo modele glavnih entiteta raspoređivanja: novi model programa u obliku miješanoga grafa, model računalnog sklopovlja u obliku potpuno povezanoga neusmjerenoga grafa i model

raspoređivanja višekriterijskom optimizacijom. Ujedno smo definirali i unaprijeđeni postupak raspoređivanja programa na sklopovlje particioniranjem miješanoga grafa — RaPPaMaG.

D.3. **Unaprijeđeni algoritam raspoređivanja programskih modela temeljenih na usmjerenim grafovima s ciljem uštede energije.** (Cilj 3 i 4)
Osmislili smo i implementirali unaprijeđeni postupak raspoređivanja programa na raznorodno raspodijeljeno računalno sklopovlje zasnovanog na rangiranju jedinica programa, koji dodatno omogućuje uštedu energije labavljenjem nekritičnih vremenskih isječaka dobivenog rasporeda — RaNDKiP.

1.4 Pregled po poglavljima

Ovaj rad podijeljen je na nekoliko tematskih cjelina u obliku poglavlja, koja pružaju opis problematike područja istraživanja koje prožima ovaj rad, postojećeg stanja u povezanoj literaturi, prikaza izvornih znanstvenih doprinosa i vrednovanja istih. Radi lakšeg snalaženja, u ovom pododjeljku dan je najosnovniji pregled za svako poglavlje.

U sljedećem, **poglavlju 2**, opisani su najvažniji koncepti potrebni za razumijevanje problema optimalnog raspoređivanja programa na sklopovlje, tj. prostora u toj problematici na koje se odgovori pružaju rezultatima istraživanja provedenog ovim radom, kao i postojećeg stanja u vezanoj literaturi. U prvom redu opisana je domena računarstva visokih performansi s pripadajućim sklopovskim i programskim paradigmama te načini modeliranja programa i sklopovlja u njoj. Prije svega, naglašeni su programi zasnovani na izmjeni poruka, i sa sklopovske strane, računalni sustavi s raspodijeljenom memorijom, gdje je navedeni problem i najizraženiji. Od modela, istaknuti su oni u obliku grafova, koji se u najvećem broju koriste u praksi. Isto poglavlje pruža pregled najčešće korištenih pokazatelja kvalitete gdje je, uz naglasak na performanse, energija istaknuta kao drugi najvažniji pokazatelj kvalitete u domeni računarstva visokih performansi. Također, navedenim poglavljem dan je presjek postojećih postupaka raspoređivanja, njihovo razvrstavanje na glavne skupine te detaljan opis onih na kojima se zasnivaju postupci predstavljeni ovim radom, ali i onih s kojima se ti postupci uspoređuju — HEFT i LDCP.

U **poglavlju 3** predstavljen je programski okvir LOSECO kao prvi izvorni znanstveni doprinos. Navedenim okvirom omogućuje se izgradnja postupaka raspoređivanja modularne strukture, primjenjivih u više domena računarstva, koji sadrže mogućnost raspoređivanja prema više korisnički naglašenih pokazatelja kvalitete. Programski okvir LOSECO zasniva se na tri glavna ulazna modela: modelu programa, modelu sklopovlja i modelu raspoređivanja, svaki od kojih je opisan zasebno. Za model programa predstavljen je novi model u obliku miješanoga acikličkoga grafa, koji omogućuje prikaz ne samo vremenskog slijeda među jedinicama programa, već i njihovu sinkronu komunikaciju. Model sklopovlja predstavljen u obliku potpuno povezanoga neusmjerenoga grafa, kao strukture koja omogućuje opis svih važnih parametara sklopovlja i komunikacijske mreže među obradbenim jedinicama. Naposljetku, dan je opis višekriterijskog modela raspoređivanja s mogućnošću pridavanja težina pojedinim kriterijima. On ujedno čini i treći ulazni model programskog okvira LOSECO.

Poglavlje 4 predstavlja unaprijeđene postupke raspoređivanja RaPPaMaG i RaNDKiPpodrobnim

opisom njihovog rada i pripadajuće primjere. U prvom redu to je postupak iz dva koraka RaPPaMaG koji slijedi strukturu programskog okvira LOSECO i omogućuje podjelu programa na nezavisne faze izvršavanja. Na taj način postupak RaPPaMaG smanjuje prostor mogućih rješenja, (mogućih rasporeda) omogućujući dobivanje optimalnih po pojedinim fazama. RaPPaMaG je odgovor na primjenu raznih postupaka vođenog lokalnog pretraživanja u praksi, čije performanse se drastično smanjuju porastom broja jedinica programa i sklopovlja. Navedeni postupak može se primijeniti na predstavljeni model programa u obliku miješanoga acikličkoga grafa, ali i na postojeće modele programa u obliku neusmjerenoga acikličkoga grafa. Drugi korak postupka RaPPaMaG koristi višekriterijsku analizu za dobivanje optimalnog rasporeda prema više pokazatelja kvalitete, za svaku pojedinu fazu izvršavanja. U slučaju predstavljenim ovim radom to su performanse i utrošak energije. S druge strane, predstavljen je postupak RaNDKiP zasnovan na rangiranju jedinica programa koji, osim primjene u visoko raznorodnim platformama, omogućuje i smanjenje utroška energije postupkom LVI. Postupak RaNDKiP koristi raznorodno rangiranje i ažuriranje nakon svakog koraka raspoređivanja omogućujući kraći i učinkovitiji raspored kod raspoređivanja programa na platforme s raznorodnom komunikacijskom mrežom. Dodatni korak navedenog postupka je i postupak LVI, koji se zasniva na labavljenju vremenskih isječaka dobivenog rasporeda, s ciljem smanjenja utroška energije programa u izvođenju smanjenjem radne frekvencije sklopovlja.

Iscrpna eksperimentalna analiza navedenih unaprijeđenih postupaka raspoređivanja, kao i njihova usporedba s ostalim postupcima predstavljena je u **poglavlju 5**. Poglavlje započinje detaljnim opisom korištenih alata, kako postojećih, tako i razvijenih u sklopu ovog rada, zatim postavki eksperimenata i testnih slučajeva korištenim u analizi, opisom programa iz repozitorija Pegasus i TGFF, ali i pokazatelja kvalitete i pripadajućih mjernih parametara. U prvom dijelu eksperimentalne analize pažnja je posvećena vremenskoj analizi rasporeda dobivenih postupkom RaNDKiP, gdje je uspoređen s njegovim inačicama RaNDKiP1 i RaNDKiP2, ali i postojećim postupcima HEFT i LDCP. Drugi dio posvećen je analizi utroška energije izvođenjem programa prema rasporedu dobivenim postupkom RaNDKiP uz primjenu razvijenog postupka LVI, ali i primjenom postupka LVI u kombinaciji s već navedenim postojećim postupcima raspoređivanja. U sljedećem dijelu poglavlja dana je analiza postupka RaPPaMaG. Ona uključuje provođenje prvog koraka postupka RaPPaMaG (particioniranja) na postojećim programima iz repozitorija Pegasus i TGFF, sa svrhom smanjenja prostora mogućih rasporeda. Analiza cjelokupnog postupka (oba koraka) provedena je kroz primjer grupno-sinkronog programa u obliku miješanoga grafa, čime je dokazana primjenjivost postupka RaPPaMaG na programe zasnovane na paradigmi izmjene poruka sa sinkronom komunikacijom među izvršnim jedinicama. Na kraju eksperimentalne analize dana je skupna analiza svih eksperimenata i dobivenih performansi unaprijeđenih postupaka raspoređivanja.

Poglavlje 6 zaključuje rad.

Poglavlje 2

Osnovni pojmovi

Osnovnim pojmovima u ovom poglavlju predstavljamo važne motivacijske, tehničke i idejne koncepte koji će se koristiti kroz ovu disertaciju. Ono sadrži temeljne informacije koje prožimaju čitavu disertaciju i koje su nužne za njezino lakše razumijevanje te kako bi omogućile dovoljnu pozadinu za shvaćanje problematike, načina i analize rješavanja iste te ponuđenih rješenja u obliku naših izvornih znanstvenih doprinosa. Ono se sastoji od opisa područja računarstva visokih performansi u kojem problematika nastaje, analize i modela osnovnih koncepata iz navedene domene računarstva (programi, sklopovlje, pokazatelji kvalitete itd.) te klasifikacije i opisa relevantnih postupaka dodjeljivanja i raspoređivanja programa na sklopovlje. U sklopu ovog poglavlja bit će opisani i formalni oblici važni za opis modela i postupaka raspoređivanja s detaljima o grafovima kao formalnim oblicima, kriteriji optimizacije koji se koriste u raspoređivanju i detaljan opis postupaka raspoređivanja čije koncepte smo koristili u izradi vlastitih unaprijeđenih postupaka.

Sljedeći pododjeljak čini kratki uvod u područje računarstva visokih performansi, najvažnije paradigme tog područja, kao i probleme u okviru ovog rada koji nastaju u tom području računarstva.

2.1 Računarstvo visokih performansi

Računarstvo visokih performansi, (eng. *high performance computing*, HPC) grana je računarstva koja obuhvaća analizu i upotrebu specifičnih, snažnijih računalnih resursa, često u obliku višestrukih računala obično zvanih superračunalima, u svrhu rješavanja zahtjevnijih programskih problema. Navedeni programski problemi često nastaju detaljnijim modeliranjem te u slučaju potrebe za obradom velike količine podataka. Oni se koriste većinom u prirodnim znanostima poput biologije, fizike, kemije, meteorologije, no i u ostalim područjima razvoja poput inženjerstva, ali i poslovnih aplikacija poput otkrivanja novih lijekova, ekonomskih prognoza, seizmologiji, obradi podataka u pozadinskim uredima (eng. *backoffice*), kao podrška u e-trgovanju, web uslugama i mnogim drugima, prema [17]. Takvi sustavi su većinom u vlasništvu jedne organizacije (resurs s vlasnikom u središtu, prema [18]), kojemu korisnici imaju udaljeni pristup. HPC često obuhvaća ne samo sklopovske, već i programske koncepte i paradigme. Ne postoji stroga definicija računarstva visokih performansi, no većinom su definicije vrlo slične, poput: *"Praksa*

nakupljanja računalne snage na način koji bi omogućio puno veće performanse nego što to mogu pružiti obična stolna računala ili radne stanice u svrhu rješavanja zahtjevnih problema u znanosti, inženjerstvu i poslovanju."[1]. Sustavi HPC-a podskup su raspodijeljenih (eng. *distributed*) i paralelnih (eng. *parallel*) računalnih sustava, koje između ostalog čine i ugrađeni računalni sustavi (eng. *embedded systems*), bežične senzorske mreže (eng. *wireless sensor networks*, WSN) i sl. Neki izvori tvrde da postoji razlika između paralelnog i raspodijeljenog računarstva, time da se pod paralelnim računalnim sustavima smatraju računalni sustavi sa zajedničkom memorijom, dok se raspodijeljenim računalnim sustavima smatraju sustavi s raspodijeljenom memorijom (tj. u kojima obradbena jedinica mora koristiti mrežu kako bi došla do podatka druge obradbene jedinice). U današnje vrijeme sve je tanja granica između navedena dva pojma zbog porasta raznorodnosti računalnih sustava, tako da ćemo u ovom radu često naizmjence koristiti oba. Prema [19], najčešći oblici današnjih sustava HPC-a su računalni grozdovi (eng. *computing grids*), nakupine računala (eng. *computing clusters*) i oblaci računala (eng. *computing clouds*), no oni mogu biti i drugačije strukture, poput sustava na čipu (eng. *system-on-chip*), više- i mnogojezgrenih procesora i sl.

S obzirom na to da se u domeni HPC-a pojedina računala često zovu **čvorovima** i mi ćemo često koristiti taj izraz u istom kontekstu. Slijedeći strukturu sustava HPC-a, čvorovi se dalje sastoje od **obradbenih elemenata** (centralnog procesora, grafičkog procesora,...), koji se dalje sastoje od **obradbenih jedinica**, što mogu biti procesori, procesorske jezgre te ostale vrste procesora, (procesori toka kod grafičkih čipova, raznorodni procesori kao dijelovi Cell i Godson procesora i sl.). **Računalni grozdovi**, prema [5, 16–20] su oni najčešće zvani superračunalima, vrlo su visokih performansi i u njemu postoje dvije glavne skupine čvorova: obradbeni čvorovi, (eng. *computing nodes*) i spremišni čvorovi, (eng. *storage nodes*), prema [21]. Također, navedeni sustavi se često sastoje od raznorodnih čvorova, tj. onih različite obradbene moći, prema [21–23] i visoko raznorodne mreže, prema [23].

Nakupine računala, s druge strane, često se vežu uz slabije povezana računala različitih karakteristika, ali s velikom razinom dostupnosti i skalabilnosti. Većinom se koriste kao računalni resursi neiskorištenih računala neke tvrtke, organizacije, ali i velikog broja kućnih računala (primjerice SETI@home[2], Asteroids@home[3]) ili kao pomoćni sustav koji rasterećuje često posjećene i korištene računalne sustave. Takvi sustavi mogu objedinjavati i manje, računalno slabije sustave, poput nakupina SoC i MPSoC sustava (eng. *multi-processing system-on-chip*) opisanih u [24].

Naposljetku, **oblaci računala** su računalni resursi koji se uz plaćanje nude krajnjem korisniku u tri glavna oblika, platforma kao usluga (eng. *platform as a service*, PaaS), infrastruktura kao usluga (eng. *infrastructure as a service*, IaaS) i program kao usluga (eng. *software as a service*, SaaS). To su računalni resursi koji, osim što su raspodijeljene naravi, daju dodatnu kontrolu nad resursima u obliku virtualnih strojeva (eng. *virtual machine*, VM), koje je moguće i migrirati kroz sustav u slučaju potrebe.

Postoje i manji oblici računalnih sustava za koje se smatra da pripadaju domeni HPC-a, a to su računala s mnogojezgrenim procesorima (s brojem procesorskih jezgri većim od 8), višeprocesorska računala, umreženi SoC i MPSoC sustavi, bežične senzorske mreže, ugrađeni računalni sustavi u razno-

[1] http://insidehpc.com/hpc-basic-training/what-is-hpc/
[2] http://setiathome.berkeley.edu/
[3] http://asteroidsathome.net/

rodnim sustavima s više obradbenih jedinica (robotika) i sl. Zajedničko im je da sadrže dvije ili više obradbenih jedinica koje rade zajedno u rješavanju programskih problema.

U idućem pododjeljku promotrit ćemo detaljnije sklopovsku strukturu navedenih sustava.

2.1.1 Sklopovske karakteristike računalnih sustava visokih performansi

U najosnovnijem obliku sklopovska struktura sustava HPC-a temelji se na umreženim obradbenim jedinicama. Same obradbene jedinice su dijelovi sklopovlja namijenjeni obradi podataka (često kao dio većih, obradbenih elemenata), npr. procesorske jezgre, *stream* procesori, vektorski procesori, jedinice rekonfigurabilnih programabilnih čipova FPGA-a (eng. *field programmable gate array*) i CPLD-a (eng. *complex programmable logic device*), mikrokontroleri i sl. Obradbene jedinice su međusobno komunikacijski povezane na različite načine, od zajedničkih registara na istom čipu, preko brzih namjenskih sabirnica, pa sve do klasičnih sabirnica, brze namjenske mreže (primjerice infiniband [25–27]), pa sve do klasične mreže (eng. *local area network*, LAN) ili bežične mreže (eng. *wireless local area network*, WLAN).

Od ostalih važnijih sklopovskih komponenti u navedenim sustavima pri modeliranju performansi eksplicitno se koristi glavna memorija (primjerice u [10,14,15,28–31]), kao i jedinice za pohranu podataka u [12, 17].

Obradbene jedinice sustava HPC-a mogu biti homogene i raznorodne, a isto vrijedi i za komunikacijske veze među njima. Homogene obradbene jedinice imaju iste ili vrlo slične karakteristike, primjerice frekvenciju, broj pričuvne memorije, proizvođača, model, karakteristike disipacije enegije i topline i sl., primjeri u [32, 33]. Raznorodne obradbene jedinice imaju karakteristike različite do te mjere da različito utječu na izvođenje programa na njima, (najčešće vrijeme izvođenja), primjeri u [11, 34–41]. Upravo su sustavi s raznorodnim obradbenim jedinicama najveći izazov današnjice u smislu kako ih najbolje upotrijebiti i iskoristiti njihov potencijal, kao što je navedeno u [14–16, 42].

S druge strane, komunikacijska mreža među obradbenim jedinicama također može biti homogena ili raznorodna. To može imati drastičan utjecaj na izvođenje paralelnih programa na toj platformi, jer će njezina učinkovitost drastično ovisiti o rasporedu tog programa na njoj, (što će pokazati i naša eksperimentalna analiza u poglavlju 5). Homogene mreže često se nalaze u manjim dijelovima grozdova i nakupina računala te u višeprocesorskim čvorovima i višejezgrenim procesorima. Pod raznorodnom mrežom smatra se ona mreža koja nema istu propusnost između svakoga para obradbenih jedinica ili je pak ovisna o smjeru u kojem se odvija prijenos podataka (asimetrični komunikacijski kanal). Ponekad jedan od pokazatelja raznorodnosti mreže osim same propusnosti, prema [23], može biti i kašnjenje (eng. *latency*), ali i njezina struktura [15,16,42–44]. Od ostalih parametara važna može biti i prilagodba mreže u slučaju istovremenih zahtjeva za istim kanalom, (eng. *network contention*) prema [43], koja je često vrlo složena za modeliranje te se u većini literature ni ne koristi ili se koristi u vrlo jednostavnom obliku.

S obzirom na navedene činjenice vezane uz sklopovsku strukturu sustava HPC-a, sa sklopovske strane mi ćemo pod sustavom HPC-a smatrati **raznorodnu umreženu strukturu skupa obradbenih jedinica**.

Drugi važan aspekt sustava HPC-a osim njihove sklopovske strukture je i programska podrška za

njih, koju opisujemo sljedećim pododjeljkom.

2.1.2 Programske karakteristike računalnih sustava visokih performansi

Iz perspektive programa namijenjenih sustavima HPC, programi temeljeni na izmjeni poruka, (eng. *message-passing*) i višenitnosti, (eng. *multithreading*) još su uvijek osnova računarstva visokih performansi. Programi namijenjeni za rad na takvim sustavima najčešće se zovu **paralelni programi**, budući da se njihovi dijelovi odvijaju istovremeno na različitim obradbenim jedinicama sklopovlja u svrhu skraćivanja vremena njihovog izvršavanja. Kao što je ranije navedeno, dva su glavna načina paraleliziranja programa za njihov rad na sustavima HPC: **paradigma izmjene poruka** i **višenitno programiranje**. Glavni standard programske paradigme izmjene poruka jest MPI (eng. *message passing interface*) iz [45], čije su implementacije OpenMPI,[4] MPICH, MPICH2,[5] MPI-MS,[6] MVAPICH, MVAPICH2[7] i dr. Druga najčešće korištena paradigma u računarstvu visokih performansi jest višenitno programiranje, od koje se najčešće koristi otvorena višeobrada (eng. *open multiprocessing*, OpenMP) iz [46] te u manjoj mjeri niti programskog jezika C++11. Navedena se paradigma najčešće veže na niti operacijskog sustava poput *Pthreads*, koje koristi i sam OpenMP. MPI i OpenMP najčešće su korišteni jer imaju veliku prenosivost između različitih računalnih platformi.

Glavne su razlike navedenih paradigmi:

- njihovi konstitutivni entiteti, **procesi** kod paradigme izmjene poruka i **niti** kod otvorene višeobrade,
- način izmjene podataka među konstitutivnim entitetima, komunikacijski pozivi, (slanje poruka) kod paradigme izmjene poruka i pristup zajedničkoj memoriji kod otvorene višeobrade,
- njihova primjena, MPI se najčešće koristi u računalnim sustavima gdje svaki čvor ima vlastitu privatnu memoriju, dok se OpenMP koristi u sustavima sa zajedničkom memorijom.

Sve je češća prisutnost hibridnih paradigmi, poput kombinacije OpenMP-ja i MPI-ja opisanoj u [47] te specijalnih programskih okruženja koje služe kao sloj između programa i postojećih paradigmi, poput StarPU-a iz [48] i OpenACC-a u [27].

Ono što veže sve navedene paradigme jest činjenica da su programi podijeljeni na manje cjeline koje se izvršavaju neovisno. Te cjeline se najčešće zovu **zadatcima** (eng. *tasks*) i **poslovima** (eng. *jobs*). Zadatci se najčešće koriste kao manje jedinice programa, dok se poslovima često nazivaju skupine zadataka, prema [17, 21, 49, 50], pa čak i cijeli programi, prema [17]. No negdje postoje i drugi odnosi, primjerice zadatci imaju podzadatke u [16, 51] ili u [28] gdje su poslovi nakupine niti.

Ukratko, programska paradigma sustava HPC-a način je kako su programi prilagođeni za rad na računalnim sustavima s većim brojem obradbenih jedinica, tj. karakteristike konstitutivnih elemenata programa i njihovi odnosi.

Ono što je također važno osim oblika samih konstitutivnih jedinica programa je i njihov međusobni odnos. On može postojati u obliku vremenske međuovisnosti (vremenski slijed izvršavanja) i prostorne međuovisnosti (podatkovne, komunikacija).

[4] http://www.open-mpi.org/
[5] http://www.mpich.org/
[6] http://msdn.microsoft.com/en-us/library/bb524831(v=vs.85).aspx
[7] http://mvapich.cse.ohio-state.edu/

Navedeni odnosi, kao i struktura jedinice programa te formalni oblici istih, bit će prikazani u sljedećem pododjeljku u kojem predstavljamo načine modeliranja sklopovskih i programskih sustava HPC-a.

2.2 Modeliranje sklopovlja i programa u domeni računarstva visokih performansi

S obzirom na to da postoji velika raznolikost i broj modela koji opisuju paralelne programe i sustave HPC-a u ovom ćemo pododjeljku napraviti presjek najvažnijih. Također ćemo detaljnije opisati modele koje smo koristili i koji su inspirirali naš rad, tj. one za koje smatramo da ponajbolje opisuju ulazne entitete za postupke dodjeljivanja i raspoređivanja opisanih kasnije te onih koji su najčešći u praksi. Za modeliranje programa i sklopovlja postoje i univerzalni opisni jezici. Primjerice, jedinstven jezik za označavanje (eng. *unified markup language*, UML) i njegova izvedenica, opisni jezik za arhitekture (eng. *architecture description language*, ADL) [52]. Uzevši u obzir da su navedeni modeli u praksi dugi niz godina, oni omogućuju detaljan opis programskih (ali i sklopovskih) arhitektura. No, oni često sa sobom povlače i vrlo kompliciran razvoj potrebnih programa, posebice u slučajevima kada su kao modeli potrebni vrlo jednostavni oblici. Zbog toga nastaje velik broj manjih i jednostavnijih modela, često s vrlo ograničenom opisnom moći i primjenjivosti. Takav slučaj nastaje i u domeni HPC-a kod problema učinkovitog raspoređivanja programa na sklopovlje, što je ujedno i motivacija za pronalaskom skupa općenitijih modela kojima bi se mogao opisati veći broj slučajeva u navedenom području. Nedostatak univerzalnih modela ujedno smanjuje mogućnosti izravne usporedbe u pomacima i doprinosima u znanstvenom području raspoređivanja programa na sklopovlje.

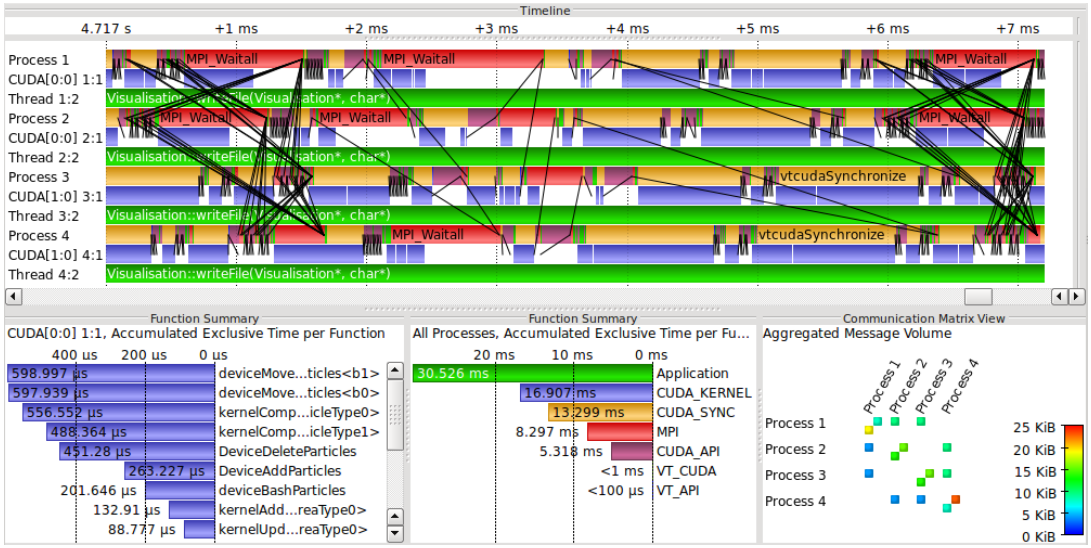
S obzirom na to da su skupine modela za opis programa (paralelnih), ali i sklopovlja HPC-a, različite, za svaku ćemo zasebno opisati karakteristike, kratku klasifikaciju te detaljnije opisati one koji su inspirirali naš rad.

2.2.1 Modeliranje paralelnih programa

Modeli programa u domeni raspoređivanja programa na sustave HPC-a u općem slučaju zahtijevaju da njihov zapis pokazuje entitete od kojih se sastoje, (koji se mogu izvršavati istovremeno, paralelno) i njihovih međusobnih odnosa. Tako da je prvi korak u izradi modela programa uzrokovan potrebom njegovog paralelnog izvršavanja, što znači da ga je potrebno podijeliti na manje dijelove. U vezanoj literaturi ti se dijelovi najčešće nazivaju zadatci (*tasks*) i poslovi (*jobs*). U većem broju slučajeva poslovi predstavljaju skupine zadataka, (primjerice u [17, 21, 49, 50]) tako da se općenito zadatci smatraju najmanjim jedinicama programa s kojima se rukuje u postupcima raspoređivanja. Naravno, postoje i mnogi drugi načini označavanja, između ostalih i oni u kojima se pod zadatkom smatra skup poslova, primjerice u [53]. Ostali autori koriste druge jedinice za prikaz sastavnih entiteta programa, primjerice komponente u [54–56], no taj pojam komponenti se razlikuje od standardiziranog pojma komponente programa korištene u domeni komponentnog razvoja programa (eng. *component-based software engineering*, CBSE) kao iz [57]. Komponentni prikaz iz [14, 53, 54] su pokušaji da se identificira skup konstitutivnih

komponenti od kojih se mogu graditi paralelni programi i koje su neovisne o samoj arhitekturi programa i sklopovlja zbog i dalje prisutne potrebe da se pojednostavi i standardizira njihova izgradnja. Drugi komponentni prikazi, primjerice [14, 55, 56], koriste komponente programa zvanih motivima, od kojih se slažu paralelni programi, ali koji predstavljaju najčešće matematičke i algebarske strukture programa korištenih u paralelnim programima, konkretno je ovdje riječ o tzv. "Berkeleyevim patuljcima" (eng. *Berkeley dwarfs*). Od ostalih sastavnih entiteta još se koriste procesi, primjerice u kojima su programi mreže procesa [30,31,58], zatim niti [28,58] itd.[8] S toga i jedan dio našeg izvornog znanstvenog doprinosa (D.1. i D.2.) daje općenitiji prikaz sastavnog entiteta programa u izvođenju.

Problem same izrade modela paralelnog programa kao ulaznog entiteta za postupak raspoređivanja je potreba za njegovom ručnom izradom. No to se mijenja pojavom i korištenjem raznih pomoćnih alata, poput besplatnih grafova zadataka (eng. *task graphs for free*, TGFF) iz [59], koji smo i sami koristili te koji je opisan kasnije. Nadalje, to su biblioteke paralelnih zadataka programskog jezika C# (eng. C# *task parallel libraries*, TPL) [60], ali i mnogi drugi. Modeliranje postojećih paralelnih programa može se izvesti korištenjem statičke analize koda, instrumentacijom (umetanjem sondi u kod i/ili *kernel* pomoću alata za profiliranje, sondiranje i analizu traga programa) i korištenjem alata za analizu nakon izvršavanja programa, prema [54]. Od alata za profiliranje i izradu traga izvršavanja programa u domeni HPC-a ističu se Score-P[9] i VampirTrace [61], nakon kojih se pomoću alata poput Periscope[10], Scalasca[11], Vampir [61] i Tau[12] vrši analiza i statistička obrada dobivenog traga. Primjer traga izvršavanja programa prikazan je i na slici 2.1, na kojoj se mogu identificirati procesi u izvođenju, komunikacijski pozivi među njima, sinkronizacija, vremenske analize pojedinih funkcija, statistika komunikacijskih poziva, kao i mnogi drugi podatci koje daju alati za profiliranje.



Slika 2.1: Primjer analize traga izvršavanja paralelnog programa upotrebom alata VampirTrace.[13]

[8] Kako bi pojednostavili raznolikost entiteta programa, pojmove zadatak, izvršna jedinica programa ili samo jedinica programa smatramo ekvivalentima.

[9] http://www.vi-hps.org/projects/score-p/

[10] http://www.lrr.in.tum.de/ periscop/

[11] http://www.scalasca.org/

[12] http://www.cs.uoregon.edu/research/tau/home.php

[13] Izvor: http://tu-dresden.de/die_tu_dresden/zentrale_einrichtungen/zih/forschung/projekte/vampirtrace/accelerator

Ukratko, mogući načini izrade modela programa su u prvom redu njegova ručna izrada, zatim korištenjem alata za njihovu pseudoslučajnu izradu poput TGFF-a ili okruženja DAGGer (opisanog kasnije) ili sintezom podataka skupljenih nekim od ranije navedenih alata za profiliranje i snimanje traga.

Pri modeliranju programa iznimno je važan odabir odgovarajuće razine zrnatosti (eng. *granularity*), tj. razine njegove rascjepkanosti na manje dijelove. Prevelika zrnatost može dovesti do velikih tranzicijskih vremena između jedinica programa, velikog ukupnog trajanja brojnih izmjena konteksta (eng. *context switching*) i sl., prema [58]. S druge strane, premala zrnatost može dovesti do ograničenosti razine paralelizma, uzrokujući velike praznine u rasporedu programa na obradbenim jedinicama sklopovlja i čekanja, dovodeći do ograničene skalabilnosti i podopterećenjem računalne platforme. Za zaključiti je kako bi veličine jedinica programa trebale biti što uniformnije, no to je ograničeno stvarnom strukturom programa, njihovom međusobnom komunikacijom, podatkovnom ovisnosti i dr.

Jedinice paralelnih programa u svrhu raspoređivanja često se formiraju u oblik raznih matematičkih formalizama koji predstavljaju njihovu međuovisnost i interakciju. U domeni HPC-a postoje dvije glavne skupine tih formalizama, a to su **nizovi nezavisnih zadataka** (eng. *queues of tasks*) i **grafovi zadataka** (eng. *task graphs*). Nizovi zadataka predstavljaju skup vremenski i prostorno nezavisnih zadataka ili poslova, poput onih u [11, 16, 32] s vrlo širokom paletom postupaka za njihovo raspoređivanje. S druge strane, grafovi zadataka predstavljaju složene strukture jedinica programa koji mogu biti vremenski ovisni, (imati određen slijed) te mogu međusobno komunicirati (prostorna ovisnost). Postoje i hibridni pristupi, poput onog u [17] gdje se raspoređuju razne strukture poslova (uključujući i nizove), od kojih se svaki sastoji od grafa zadataka. U narednim pododjeljcima opisat ćemo svakoga od njih.

Model niza nezavisnih jedinica programa

Model paralelnog programa koji se sastoji od nezavisnih jedinica programa predstavlja se kao niz $V = \{v_n : \ n = 0, ..., \vartheta - 1\}$, gdje svaki v_i predstavlja slijed operacija, (zadatak, posao) i svaki od njih se može izvršavati potpuno nezavisno od drugih (paralelno, istovremeno). Takav model često reproducira programe s vrlo velikom razinom paralelizma, (eng. *embarrassingly parallel software*) ili skup nezavisnih programskih opterećenja, (primjerice poslova u okruženju računalnih grozdova). Prije raspoređivanja niza zadataka oni se preliminarno mogu poredati prema njihovim karakteristikama, poput obujma obradbenih zahtjeva, vremena izvođenja u najgorem slučaju (eng. *worst-case execution time*, WCET), prioritetima, nužnim vremenima završetka (eng. *deadlines*), njihovom afinitetu prema određenim obradbenim jedinicama i sl. Mogu se također poredati korištenjem nekog od poznatih algoritama poput *Round-robin*, *Priority scheduling*, Min-min iz [11] itd.

Model programa u obliku grafa zadataka

Model programa zasnovan na međuovisnim jedinicama programa i onih s međusobnom interakcijom predstavlja se kao model grafa zadataka prema mnogim izvorima, primjerice [10, 17, 25, 34, 36–38, 42, 58, 62]. Najčešće su to usmjereni aciklički grafovi zadataka, (eng. *directed acyclic graph*, DAG) ili grafovi interakcije zadataka (eng. *task interaction graph*, TIG). Prema mnogim autorima model u obliku

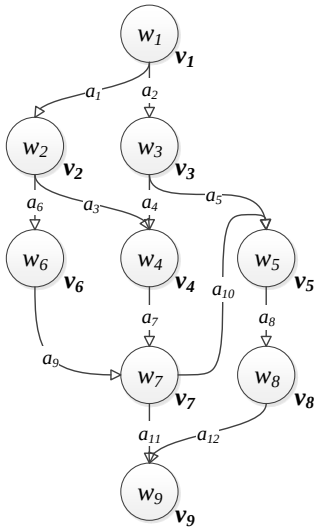
grafa zadataka ponajbolje predstavlja interakciju među jedinicama programa, komunikaciju među njima i ostale tipove njihove međuovisnosti. U općem slučaju model programa u obliku grafa $G = (V, E)$ sastoji se od skupa vrhova (čvorova) grafa $v_i \in V$ koji predstavljaju jedinice programa (zadatke) i skupa bridova $e_i \in E$ koji predstavljaju međuovisnosti među zadatcima. Vrhovi sadrže informacije o jedinicama programa poput obradbenih zahtjeva, WCET-a, nužnih vremena završetaka, prioriteta i sl. Bridovi predstavljaju tip i iznos interakcije među jedinicama programa (većinom komunikacija i vremenski slijed). Postoje i druge varijante modela programa koji se sastoje od međuovisnih jedinica, primjerice model cijevi i filtra (eng. *pipe and filter*) u [56], gdje filtar predstavlja zadatke (obrada koja se vrši nad nekim podatcima), a cijevi predstavljaju njihovu interakciju (komunikaciju između faza obrade). Zatim, to su mreže procesa u [30,31,58] koje su slične modelu u obliku DAG-a, samo s procesima umjesto zadataka i s pojavom ciklusa, zatim model programa u obliku Markovljevih lanaca u [63] itd. To mogu biti i mnogo rjeđe korišteni, relativno novi oblici, poput grafa vremenske interakcije zadataka (eng. *temporal task interaction graph*, TTIG) iz [64], skupa niti [58], lanaca zadataka [65], strojeva za označavanje (eng. *tag machines*) [33] itd.

U domeni raspoređivanja paralelnih programa na sustave HPC-a najčešći oblici modela programa, kao što je već spomenuto, su modeli u obliku usmjerenih acikličkih grafova (DAG) i grafa interakcije zadataka (TIG). S obzirom na to da je i naš model zasnovan na njima, u sljedećim pododjeljcima oni će biti predstavljeni detaljnije.

Modeli programa u obliku DAG-a

Kao što je već navedeno, najčešći oblik modela programa korišten za njegovo raspoređivanje na sklopovlje je usmjereni aciklički graf — DAG, korišten u mnogim izvorima, primjerice u [10, 12, 13, 15, 17, 25, 36, 38–42, 44, 49, 51, 66, 67]. Usmjereni aciklički graf je matematička struktura $G = (V, A)$, gdje je V skup vrhova grafa koji predstavljaju jedinice programa, dok je A skup usmjerenih bridova (lukova) koji mogu predstavljati vremenski slijed izvođenja jedinica programa, smjer toka komunikacije među njima ili oboje, [13,37,41]. U većini slučajeva taj model predstavlja vremenski tijek među jedinicama programa tako da smjer luka pokazuje slijed izvođenja. Primjer modela DAG-a prikazan je na slici 2.2 i sastoji se od devet jedinica programa $v_1, ..., v_9$, s pripadajućim obradbenim zahtjevima $w_1, ..., w_9$ i lukovima koji ih povezuju $a_1, ..., a_{12}$, predstavljajući slijed njihovog izvođenja. Kao što se vidi i iz navedene slike, jedinice programa u modelu DAG-a sadrže parametre koji ih opisuju. Većinom su to parametri u obliku troškova (obradbeni trošak, vremenski trošak, utrošak memorije itd.), primjerice obradbeni zahtjevi, tj. koliko vremena će svaka jedinica programa utrošiti na svakoj od obradbenih jedinica sklopovlja, isto tako mogu biti i parametri neovisni o sklopovlju, poput broja ciklusa/instrukcija koje trebaju izvršiti, kao u [40,51], nužnih vremena završetka, kao u [34, 68, 69] i sl.

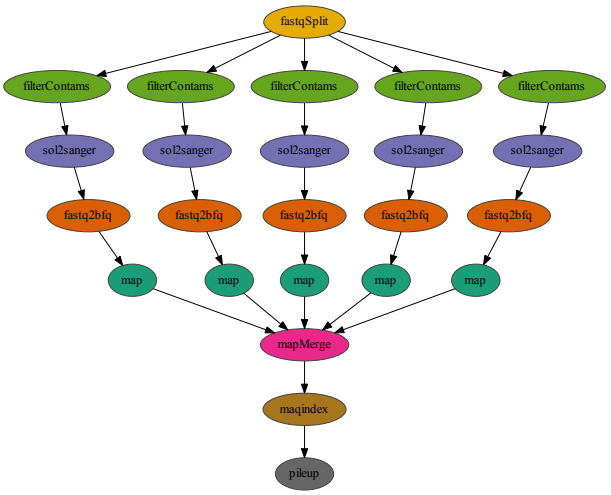
U slučaju kada lukovi modela DAG-a programa predstavljaju i vremenski slijed među jedinicama programa i komunikaciju, lukovima se dodaje i težina. Takvi se oblici DAG-a koriste kod većine postupaka raspoređivanja kojima je primarni cilj skratiti vrijeme izvršavanja programa na računalnoj platformi, poput onih u [13, 34, 37, 38, 40, 42, 44, 48], ali i mnogim drugima.



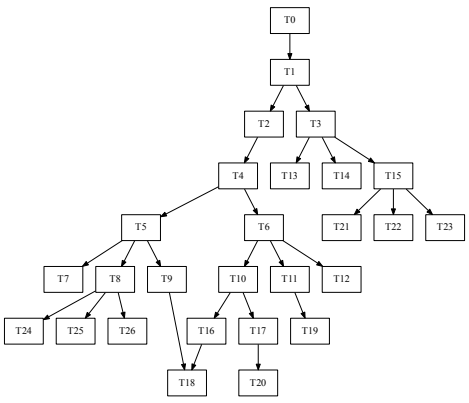
Slika 2.2: Primjer ručno izrađenog modela DAG-a programa.

Mododel programa koristi se i u drugim domenama računarstva osim onog visokih performansi, poput ugrađenih računalnih sustava u [12, 31, 40, 51, 53, 70, 71]. Detaljnije, to su sustavi na čipu (eng. *system-on-chip*, SoC) [31], višeobradbeni sustavi na čipu (eng. *multi-processing system-on-chip*, MPSoC) [12], bežične senzorske mreže (eng. *wireless sensor networks*, WSN) [51], sustavi mreže na čipu (eng. *network-on-chip*, NoC) [40] itd.

Za kreiranje modela DAG-a programa postoje i mnogi alati, primjerice već spomenuti TGFF [59], zatim niz *Random DAG generatora* korištenih u [13, 23, 39, 72], ali mogu biti i preuzeti kao dio nekog repozitorija, primjerice iz repozitorija znanstvenih programskih opterećenja Pegasus, prema [2, 49, 50]. Neki od tih programa prikazani su na slici 2.3.



(a) Graf iz repozitorija Pegasus, "Epigenomics_24".



(b) Graf generiran alatom TGFF, "kbasic_task_25".

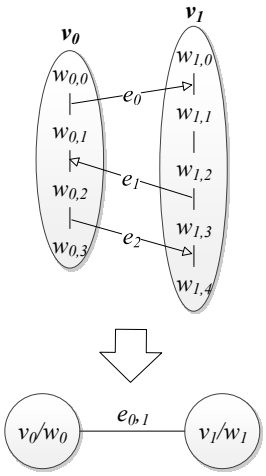
Slika 2.3: Primjeri modela DAG-a.

Kao što će biti riječi kasnije, model DAG-a sadrži i neke nedostatke. Ti su nedostatci bili i motivacija za ostvarivanje nekih od izvornih znanstvenih doprinosa predstavljenih ovim radom (doprinosi D.1. i

D.2.). Glavni nedostatak je mogućnost prikaza sinkrone i asinkrone komunikacije istovremeno, posebno sinkrone komunikacije u kojoj aktivno sudjeluju dvije ili više izvršnih jedinica programa. Zbog toga ćemo i mi predstaviti vlastiti model programa u obliku miješanoga grafa (doprinos D.2.) koji rješava taj problem. DAG oblik programa koristit ćemo kao (i) jedan od modela iz kojih je nastao i prijedlog našeg modela u obliku miješanoga grafa i kao (ii) ulazni model za unaprijeđene postupke raspoređivanja koji predstavljaju doprinos D.3. (inačice RaNDKiP), kao i njihovih konkurentnih postupaka (LDCP, HEFT).

Modeli programa u obliku TIG-a

Model programa u obliku grafa interakcije zadataka je drugi najčešći formalni opis programa koji se sastoji od zavisnih jedinica u obliku pogodnom za njegovo raspoređivanje sklopovlju, s nekim od primjera u [10, 62]. Navedeni model izvorno je nastao iz koncepta grafa interakcije i istovremenosti zadataka (eng. *task interaction concurrency graph*, TICG) opisanog 1998. godine u [73] za vrijeme začetaka paralelnog programiranja. Model TIG-a služi za opis interakcije među zadatcima (jedinicama programa), kao što mu i samo ime kaže, s tim da su interakcije predstavljene bridovima grafa. Prema [62] taj model predstavlja klasične paralelne programe, dakle većinom zasnovane na paradigmi izmjene poruka. Također, u odnosu na model DAG-a TIG predstavlja pojednostavljenje modela kada su važne performanse raspoređivanja, ili kada detaljne informacije o programu, (slijed izvođenja ili smjer komunikacije) nisu važne ili nisu poznate. Nadalje TIG nudi bolji opis programa u slučaju kada postoje višestruki komunikacijski pozivi među istim izvršnim jedinicama, što bi inače zahtijevalo višestruke usmjerene bridove u modelu DAG-a uzrokujući cikluse i negirajući sam koncept DAG-a (nije više aciklički). Model u obliku TIG-a može se promotriti kao graf $G = (V, E)$, gdje je skup V skup vrhova grafa koji predstavljaju jedinice programa, dok je skup E skup neusmjerenih bridova grafa koji predstavljaju interakciju (većinom obujam komunikacije). Slika 2.4 prikazuje primjer izrade modela TIG-a programa iz traga izvođenja programa koji se sastoji od dva procesa v_0 i v_1 te koji koristi paradigmu izmjene poruka (MPI).



Slika 2.4: Primjer izrade modela TIG-a programa.

Na toj slici vidljivo je kako se svaki proces $v_i \in V = \{v_i : \ i = 0, ..., \vartheta - 1\}$ sastoji od odgovarajućih faza

obrade $w_{i,l} \in W_i = \{w_{i,l} : l = 0, ..., r_i - 1\}$ i faza komunikacije:

$$e_k(v_i, v_j) \in E_{i,j} = \begin{cases} v_i, v_j : \ i, j = 0, ..., \vartheta - 1, \\ i \neq j, \\ k = 0, ..., h_{i,j} - 1. \end{cases} \tag{2.1}$$

Ukupan iznos obrade jedinice programa v_i dan je izrazom:

$$w_i = \sum_{l=0}^{r_i - 1} w_{i,l}, \tag{2.2}$$

dok je ukupan iznos komunikacije između jedinica programa v_i i v_j dan izrazom:

$$e_{i,j} = \sum_{k=0}^{h_{i,j} - 1} e_k(v_i, v_j). \tag{2.3}$$

Rezultirajući jednostavni model TIG-a programa zadan tragom izvođenja sa slike 2.4, primjenom (2.2) i (2.3) prikazan je ispod njega na istoj slici u obliku grafa s dva vrha (v_0, v_1) i jednim neusmjerenim bridom među njima $e_{0,1}$.

Glavni nedostatak modela TIG-a leži u njegovoj glavnoj karakteristici, a ta je da nema određen smjer bridova. Na taj način on nije u mogućnosti prikazati smjer komunikacije ili vremenski slijed izvođenja jedinica programa. No takav model može opisati sinkronu komunikaciju među jedinicama programa, što smo naveli kao nedostatak modela DAG-a u prethodnom pododjeljku. Primjerom izrade TIG-a na slici 2.4 prikazano je da je on pogodan za prikaz paralelnih programa koji koriste paradigmu izmjene poruka, koja je još uvijek vodeća u domeni HPC-a. Navedeno ne vrijedi samo za sustave s privatnom memorijom, već vrijedi i za MPI, koji postaje sve važniji i za sustave sa zajedničkom memorijom, što je navedeno i u [74]. Uz to, u izvorima [74, 75] je i dokazano da na sustavima sa zajedničkom memorijom MPI ima i bolje performanse od OpenMP okruženja. Zbog toga, glavne koncepte modela TIG-a upotrebljavamo kao dio našeg modela u obliku miješanoga grafa (doprinos D.2.) za prikaz sinkrone komunikacije među jedinicama programa. Konkretno, u slučaju višestrukih MPI poziva za komunikacijske rutine slanja i primanja podataka komunikacije od točke do točke te kolektivnih komunikacijskih rutina poput *broadcast*, *reduce*, *scatter*, *gather*, *all-to-all* i drugih, [45].

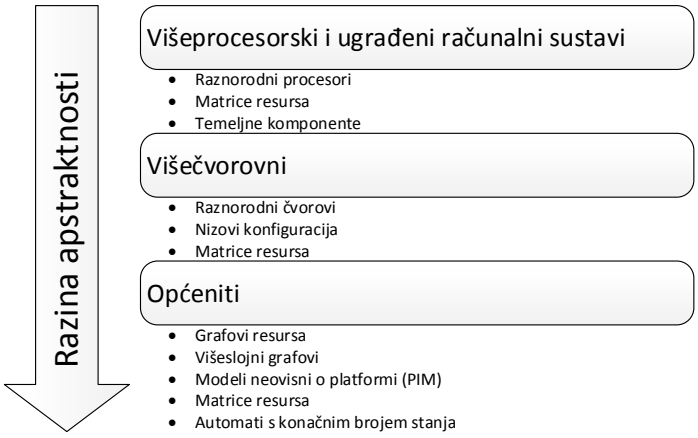
2.2.2 Modeliranje sklopovlja

Postoje mnogi primjeri modela sklopovlja HPC-a u praksi, s tim da u okviru ovog rada naglasak dajemo samo na raznorodne platforme (raznorodnost platforme opisana je u pododjeljku 2.1.1), jer je u njima izražena problematika izrade učinkovitog rasporeda. Navedeni tip računalnih platformi nazivamo **raznorodnim računalnim platformama** (eng. *heterogeneous computing platform*, HCP). Model HCP-a, kao model koji opisuje sklopovlje na koje se raspoređuju programi, treba sadržavati:

- sve relevantne informacije o obradbenim jedinicama koje sadrži,

- informacije o komunikacijskim kanalima među obradbenim jedinicama,
- informacije o energetskim karakteristikama obradbenih jedinica, a to su najčešće karakteristike dinamičkog skaliranja frekvencije i napona (eng. *dynamic voltage and frequency scaling*, DVFS) te stanja mirovanja (eng. *sleep states*) u koja sustav može ući,
- informacije i karakteristike svih parametara relevantnih za ostale pokazatelje kvalitete koji se analiziraju, primjerice:
  - vremenske karakteristike, poput vremena promjene konteksta, rekonfiguracijska kašnjenja,
  - tip obradbene jedinice,
  - dostupnost obradbene jedinice i komunikacijskoga kanala (većinom kod dinamičkog raspoređivanja) itd.

Zbog navedenih zahtjeva postoje mnogi modeli sklopovlja u relevantnom području istraživanja. Kod HCP-a s više čvorova modeli variraju od skupine raznorodnih čvorova, primjerice u [15,17,21,42,44,51,67], do nizova konfiguracija u [42, 44] i matrica resursa u [16]. Kod višeprocesorskih i ugrađenih računalnih sustava koriste se uglavnom matrice resursa [11, 76] i skupovi konstitutivnih komponenti [14, 53, 54]. Od općenitih modela platforme, koriste se apstraktne skupine raznorodnih procesora u [11, 34–41], grafovi resursa [10,31,36,62] te u manjoj mjeri modeli komunikacije i upravljanja (eng. *models of communication and control*, MoCC) [33] itd. Prikaz modela prema razini apstraktnosti prikazan je slikom 2.5.



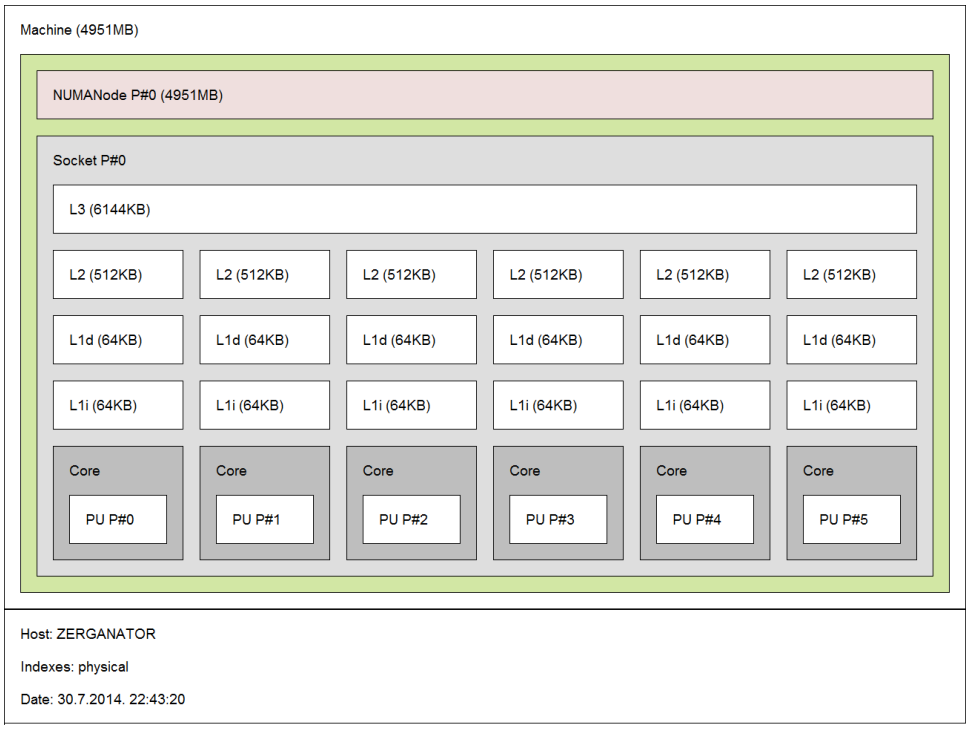
Slika 2.5: Prikaz dostupnih modela sklopovlja HPC-a prema razini apstraktnosti.

Velik broj modela koji se pojavljuju u praksi onemogućuje izravnu usporedbu različitih metoda i postupaka koji ih koriste, tako da nastaje potreba za općenitim modelima. Zatim, većina modela je usko vezana uz područje primjene, iako modeli poput skupina raznorodnih čvorova ili procesora te grafovi resursa mogu pokrivati velik dio različitog računalnog sklopovlja. Iz tog, ali i iz drugih razloga (sličnosti strukturi modela programa), mi smo koristili grafove resursa u opisu računalnih platformi na koje raspoređujemo programe i to kao dio našeg programskog okvira LOSECO (doprinos D.1.).

Model HCP-a može biti statički [62], koji ne podrazumijeva promjene stanja sklopovlja za vrijeme raspoređivanja i dinamički [16, 21], koji ažurira parametre sklopovlja za vrijeme raspoređivanja kako bi se sustav prilagodio navedenom stanju. Statički model u manjoj mjeri preslikava stanje platforme i sadrži više pojednostavljenja, no zato nudi puno bolje performanse pri raspoređivanju nego što to nude

dinamički modeli. Također, za problem raspoređivanja programa u ovom radu, pojednostavljenja koja uvodi statički model nisu značajna u toj mjeri da mogu drastično smanjiti primjenjivost rezultata koje daju postupci raspoređivanja.

Postupak modeliranja sklopovlja je također raznolik u relevantnoj literaturi, iako se kristaliziraju neka svojstva koja polako postaju standardom (model sklopovlja kao umreženi skup obradbenih jedinica). Prirodno, modeliranje sklopovlja prvenstveno ovisi o samoj platformi koja se modelira. S obzirom na to da se platforme HPC-a također dosta razlikuju, kao što je već navedeno, teško je oformiti model HCP-a koji će sadržavati sve informacije specifične za sve platforme. Postoje dva glavna pristupa modeliranju sklopovlja sustava HPC-a. To su izrada sintetičkih modela ("in silico") i preslikavanje stvarnih sustava korištenjem alata poput prijenosnog nalazišta sklopovlja[14] (eng. *portable hardware locality*, HWLOC), dostupnog kao dio OpenMPI projekta. Primjer dobivene strukture platforme pomoću alata HWLOC prikazan je slikom 2.6, na kojoj se može identificirati jedan obradbeni element sa šest obradbenih jedinica s pripadajućim parametrima pričuvne memorije, količina radne memorije, naziv računala i sl.

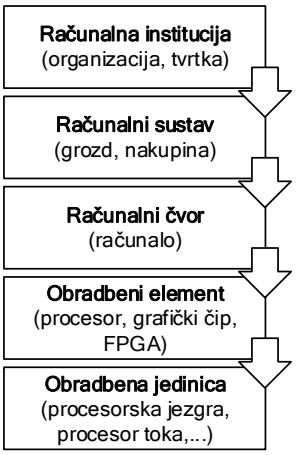


Slika 2.6: Prikaz sklopovlja koji daje alat HWLOC.

Navedeni oblik prikaza poslužio je i kao predložak za oblik platforme kakve će se moći modelirati našim programskim okvirom opisanim kasnije (LOSECO, doprinos D.1., poglavlje 3).

Obično se hijerarhijska struktura nekog sustava HPC-a prikazuje na način prikazan slikom 2.7. Većina relevantnih radova tretira najviše dvije razine takve strukture i to najčešće razinu obradbenog elementa i obradbene jedinice. Od modela obradbenih elemenata najzastupljeniji su modeli procesora, zatim grafičkog čipa (eng. *graphics processing unit*, GPU) u [14, 28, 29, 48, 66].

[14] http://www.open-mpi.org/projects/hwloc/



Slika 2.7: Najčešća hijerarhijska struktura sklopovlja sustava HPC-a prema dostupnoj literaturi.

Pri modeliranju mreže, najčešći su modeli temeljeni na stvarnim mrežnim konfiguracijama, poput onih u [15, 16, 42, 43], modeli sabirnice u [15, 34], povezanost među procesorskim jezgrama u [15, 43], modeli memorije [10,14,15,28–31,58] i spremišta podataka [12,17]. Od navedenog, naš model u obliku potpunog povezanog grafa (opisan u pododjeljku 3.2), omogućava opis obradbenih jedinica, kao i raznorodnih komunikacijskih kanala između njih.

Od parametara koji opisuju HCP-e najčešće su to njihova **obradbena moć**, koja se može izraziti u tzv. broju operacija s pomičnim zarezom u sekundi (eng. *floating point operations per second*, FLOPS) iz [77], milijuna instrukcija po sekundi (eng. *million instructions per second*, MIPS) u [16], vrijeme izvođenja za pojedine skupove operacija, kao u [13, 37–39, 41] ili neke od apstraktnih mjernih veličina koje opisuju tražene obradbene sposobnosti. **Komunikacijska mreža** među sklopovljem najčešće se opisuje mrežnom propusnosti, kao u [13, 41, 78] i/ili kašnjenjem [13, 41] i to najčešće u obliku normaliziranih vrijednosti u matricama, kao u [76] ili kao bridovi u grafovima resursa (koji su također opisani matricama). **Parametri energetskih svojstava** najčešće opisuju trend i iznos utroška električne energije i to pomoću DVFS tablica koje mogu imati diskretne (stvarna situacija) ili kontinuirane vrijednosti (pojednostavljena situacija) te brojem i opisom stanja mirovanja, (vremena prijelaznih stanja između njih i njihove energetske razine). Također, model sklopovlja može sadržavati i mnoge druge parametre, poput stupnja raznorodnosti kao u [10], sličnog stupnju raznorodnosti zadataka u modelu programa. Od navedenih parametara u našem modelu sklopovlja korišteni su parametri obradbene moći obradbenih jedinica, njihov broj te njihove energetske karakteristike, što je dano u opisu našeg modela HCP-a u obliku potpuno povezanoga neusmjerenoga grafa predstavljenoga u pododjeljku 3.2.

Osim modela programa i sklopovlja kao ulaznih entiteta za postupke raspoređivanja programa na sklopovlje, važan je i model pokazatelja kvalitete relevantnih za taj postupak. Zbog toga prikazujemo postojeće stanje literature vezano uz zajednička svojstva i nedostatke takvih modela u idućem pododjeljku.

2.2.3 Modeliranje pokazatelja kvalitete

Računalna platforma na koju se raspoređuje paralelni program treba izvršiti taj program učinkovito i brzo. Parametrizacija glavnih ulaznih sudionika raspoređivanja, programa i sklopovlja, treba biti napravljena u skladu s glavnim ciljem učinkovitog raspoređivanja. Također, važno je da platforma daje odzive kakve korisnik programa želi i da u tu svrhu isporučuje odgovarajuće pokazatelje kvalitete (eng. *quality attributes*, QA) pri izvršavanju programa. Postoje mnogi pokazatelji kvalitete koji se mogu tražiti od postupka raspoređivanja, često zadanih u obliku cilja/ciljeva koje treba postići (optimizacija više funkcija cilja) i/ili zadanih u obliku određenih ograničenja. U domeni računarstva visokih performansi najčešće su to vremenski zahtjevi u obliku skraćivanja vremena izvođenja programa (smanjivanje duljine rasporeda), od kojih se samo neki primjeri mogu naći u [10,11,14,31,34,36–39,42,44,51,62,67]. Osim toga, to može biti minimiziranje komunikacije među izvršnim jedinicama kao u [13,16,41], minimiziranje utroška električne energije [11,14,31,34,39,44,51,68,70], ujednačavanje opterećenja u [12,16,32,40,48,58,66,70], zatim povećanje skalabilnosti u [16,42,43], povećanje robusnosti [48], sprječavanje ispada [10], povećanje pouzdanosti [16] te razni drugi ciljevi specifični za primjeni poput onih u [16,30]. Ostali pokazatelji kvalitete, kao i oni na koje se često u povezanoj literaturi daju ograničenja prikazani su tablicom 2.1, gdje su crvenom bojom označeni oni koji se najčešće upotrebljavaju u literaturi, a plavom bojom oni na kojima je naglasak u ovoj disertaciji.

Tablica 2.1: Najčešći pokazatelji kvalitete u sustavima HPC-a i ograničenja prema dostupnoj literaturi.

Pokazatelji kvalitete — ciljevi	Pokazatelji kvalitete — ograničenja
Performanse	Performanse
Ujednačavanje opterećenja	Slijed izvođenja
Energija	Troškovi
Skalabilnost	Energija
Dostupnost	QoS parametri
Troškovi	Memorija (količina)
Otpornost na ispade	Mrežna propusnost
Pouzdanost	
Lokalitet podataka	
Robusnost	

Kao što je navedeno u prethodno spomenutoj tablici, osim ciljeva koji predstavljaju pokazatelje kvalitete, oni mogu biti izraženi u u obliku ograničenja. Kod vremenskih karakteristika to mogu biti nužna vremena završetka, primjerice u [34,68,69] i prioriteti, primjerice u [16,34,48]. Kod ograničenja na troškove i energiju to mogu biti maksimalne vrijednosti koje se ne smiju prekoračiti, primjerice troškovi u izradi SoC sustava u [31] ili energija u [44]. Navedeni ciljevi optimizacije u raspoređivanju pokazuju raznolikost istih u praksi. Iako postoje oni koji se češće koriste, poput minimiziranja trajanja raspoređenog programa, kriteriji kvalitete često su vrlo specifični kako bi program u izvođenju postigao rezultate primjenjive u vrlo specifičnim slučajevima. Osim samo jednog cilja, kriteriji optimizacije mogu biti i višestruki te sadržavati nekoliko ograničenja. O tome će više biti riječi u opisu postojećih postupaka raspoređivanja i njihovih nedostataka u pododjeljku 2.3. Također, s obzirom na glavnu problematiku u domeni HPC-a, a to je povećanje performansi uz smanjenje troškova (većinom izravno vezanih uz

utrošak energije). Navedeni pokazatelji kvalitete proželi su ciljeve pri raspoređivanju. Od njih, energija je podrobnije opisana u sljedećem pododjeljku.

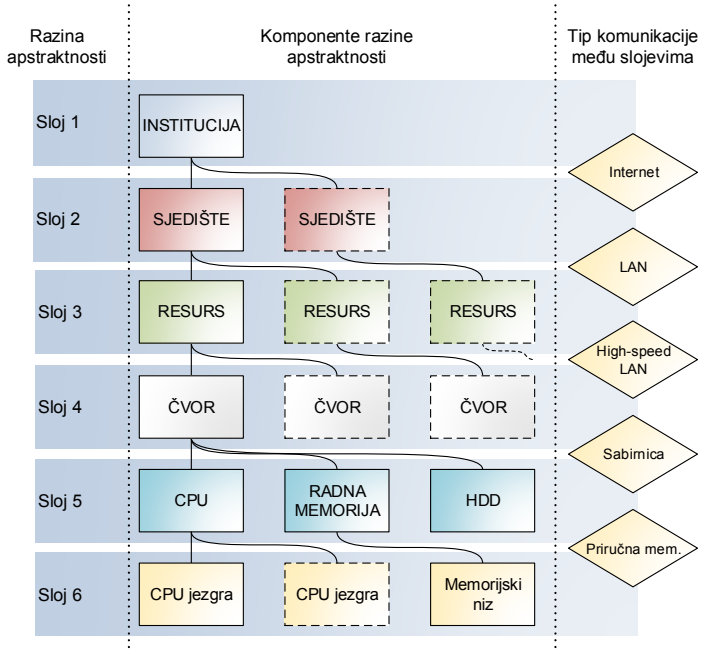
2.2.4 Energija kao kriterij raspoređivanja programa na računalne sustave visokih performansi

Pri raspoređivanju programa na sklopovlje, posebice u sustavima oblaka računala, uvođenjem eksplicitnog pojma troškova izvođenja programa otvaraju se mnogi problemi koji dosad nisu bili toliko važni kao što je pitanje utroška energije. Osim motivacije troškovima, razlog za prihvaćanje tog pokazatelja kvalitete u okružju HPC-a motiviran je i potrebom za očuvanjem okoliša, jer je višestruko dokazano da je utjecaj velikih računalnih sustava na okoliš značajan, što naglašavaju autori u [39,79], između ostalih. Primjerice, pojedina današnja podatkovna sjedišta koja troše količinu električne energije kao cijeli gradovi (primjerice Google podatkovni centar, prema [39]), ugrađuju se na posebne zemljopisne lokacije kako bi se omogućilo njihovo učinkovito hlađenje, a za njih se prave čak i zasebne elektrane. S obzirom na mnoge dokaze krhkosti zemljinog ekosustava, val "ozelenjivanja" tehnologija pogodio je i računarstvo, posebno ono visokih performansi. Početkom smanjenja utjecaja računalnih sustava visokih performansi na zemljin ekosustav smatra se tzv. Green Destiny projekt započet 2001. godine, kada je skupina istraživača pokušala napraviti računalni grozd od 240 čvorova u prostoru od jednog kubičnog metra i objavila podatke o performansama i energetskoj učinkovitosti tog sustava u [80]. Od tog trenutka, osim sirovih performansi, sve se više koriste pokazatelji energetske učinkovitosti navedenih sustava. Početak je to i tzv. GREEN500 liste energetski najučinkovitijih računalnih sjedišta visokih performansi dostupnoj na [8], kao komplement tada već dobro etabliranoj TOP500 listi najsnažnijih svjetskih superračunala dostupnoj na [7].

Danas postoje mnogi "zeleni" standardi, poput EPEAT-a (eng. *electronic product environmental assessment tool*), prema [81], Energy Star 5.0[15] i RoHS smjernice (eng. *restriction of hazardous substances directive*) prema kojima proizvođači sklopovlja u domeni HPC-a (ali i ostalim domenama računarstva) proizvode sklopovlje. Ono što je najvažnije u relevantnom području istraživanja jest da postoje tehnologije, na koje može utjecati onaj koji razvija programe, kako bi smanjio utrošak energije pri izvođenju programa, izravno ili neizravno. Izravno se to može činiti kroz dinamičko skaliranje frekvencije i napona (DVFS), ali i korištenjem različitih stanja mirovanja (eng. *sleep states, S-states*). Neizravno se utrošak energije može smanjiti učinkovitim paradigmama pri razvoju programa, optimizacijom programskog koda i sl. Razine sustava HPC-a u kojima se na različite načine može uštedjeti energija dani su u [1] i prikazani slikom 2.8. Također, različite tehnologije uštede energije, prema slojevima sustava HPC-a prikazanih slikom 2.8, dane su slikom 2.9.

Postoji mnogo povezanih radova koji koriste tehnologiju DVFS za smanjivanje energije pri raspoređivanju programa na sklopovlje, primjerice [3, 11, 34, 44, 82–84]. Većinom se ona koristi nakon već upotrijebljenog postupka raspoređivanja prema kriteriju smanjenja trajanja rasporeda (npr. [39, 85]), ali i istovremeno, primjerice u [11, 44] u obliku jednog od više kriterija optimizacije.

[15] http://www.energystar.gov/



Slika 2.8: Slojevi "ozelenjivanja" sustava HPC-a prema [1].

Također, neki autori koriste i drugačije načine modeliranja utroška energije u raspoređivanju programa na sustave HPC-a, primjerice u [70] kao potrošnju energije u prosječnom slučaju, zatim raspodjelu potrošnje energije i procijenjeni utrošak iste u [51], kao utrošak energije pri komunikaciji jedinica programa iz [86] ili izravno iz frekvencije i napona obradbenih jedinica u [87].

"Zelena" tehnologija	Sloj primjene
Tehnologija proizvodnje sklopovlja	sloj 6, sloj 5
Upravljanje energijom sustava	sloj 6, sloj 5, sloj 4
Dinamičko skaliranje frekvencije i napona procesora, ograničavanje obradbene snage	sloj 6, sloj 5
Energetski svjestan OS	sloj 4
Virtualizacija	sloj 5, sloj 4
Raspoređivanje procesa	sloj 6, sloj 5, sloj 4, sloj 3
Energetski svjesni programski prevoditelji	sloj 4
Sustavi nadzora resursa	sloj 4, sloj 3, sloj 2
Predviđanje	sloj 4, sloj 3, sloj 2
Komponentni razvoj programske podrške	sloj 4
Udio programera	sloj 4, sloj 3

Slika 2.9: Tehnologije "ozelenjivanja" sustava HPC-a po slojevima prema [1].

Ostali autori oslanjaju se na tehnologije prilagodbe stanja mirovanja računalnog sustava, poput onih u [70, 88].

Kao što smo napomenuli, energija je vrlo važan kriterij prema kojemu se provodi raspoređivanje programa na sklopovlje u sustavima HPC-a. S jedne strane to je radi smanjenja troškova, poput onih u sustavima oblaka računala (ali i zbog organizacija koje posjeduju velike sustave HPC-a) ili zbog općenite brige za okoliš. Osim tehnologija energetski učinkovitog sklopovlja, postoje i tehnologije prepuštene upotrebi od strane samog programa u izvođenju. Takve tehnologije koriste se i u sprezi s postupcima

raspoređivanja. Zato smo ih upotrijebili u oba naša unaprijeđena postupka raspoređivanja (RaPPaMaG i RaNDKiP) i to u obliku zadavanja parametara utroška energije po instrukciji za svaku obradbenu jedinicu u postupku RaPPaMaG te u obliku upotrebe DVFS karakteristike u postupku RaNDKiP. Također, energiju koristimo kao jedan od kriterija optimizacije kojemu možemo zadavati važnost prema želji korisnika (postupak RaPPaMaG), dok se u postupku RaNDKiP ušteda energije dobiva naknadnom analizom rasporeda načinjenog u svrhu smanjenja vremena izvođenja programa (kao pomoćni postupak LVI).

Nakon što smo opisali glavne entitete raspoređivanja te njihovo modeliranje, kao i raznolike mogućnosti kriterija koji postoje pri postupcima raspoređivanja, u idućem pododjeljku opisat ćemo i postojeće postupke raspoređivanja koji koriste navedeno, kako bi omogućili što bolji raspored programa na sklopovlju.

2.3 Postojeći postupci raspoređivanja programa na računalne sustave visokih performansi

Slično kao i kod modela programa i sklopovlja, u domeni HPC-a postupci raspoređivanja su također vrlo specifični situaciji za koju se koriste. Postupci raspoređivanja definiraju se na osnovu karakteristika programa i sklopovlja koji su dostupni i koji su relevantni za zahtjeve koje zadaje korisnik programa[16]. Postoji iznimno velik broj postupaka raspoređivanja programa na sklopovlje u domeni HPC-a, no mi ćemo ovim pododjeljkom pokriti one najpoznatije i najbolje te one koji su inspirirali i naš rad. Detaljno ćemo opisati samo postupke koje smo i sami koristili, bilo kao dio vlastitih postupaka raspoređivanja, bilo kao onih s kojima smo ih usporedili u eksperimentalnoj analizi.

Postupak raspoređivanja programa na sklopovlje često se zove još i dodjeljivanje (eng. *allocation*), mapiranje (eng. *mapping*) i raspoređivanje (eng. *scheduling*). Postoje razne definicije svakoga od njih, no najčešća nomenklatura jest da dodjeljivanje znači samo odabir obradbenih jedinica sklopovlja za svaku jedinicu programa, bez utvrđivanja njihovog redoslijeda na njima i bez njihovog vremenskog rasporeda. Pojam *scheduling* označava i dodjeljivanje, ali i vremenski raspored jedinica programa na sklopovlju (njihov redoslijed, njihova okvirna vremena početka i završetka). Postoji još i postupak podudaranja (eng. *matching*), za koji se smatra da je dio postupka mapiranja i koji se sastoji od podudaranja i raspoređivanja, prema [89] (vezan uz dinamičko mapiranje). U [16,34] se postupak raspoređivanja (*scheduling*) koristi zajedno s postupkom mapiranja kao odvojena faza. Primjerice, u [31] u svrhu dizajna na razini sustava (eng. *system-level design*) koriste se pojmovi dodjeljivanje (određivanje koje jedinice sklopovlja će se koristiti pri izgradnji sklopovskog sustava), vezivanje (eng. *binding*) kao ekvivalent mapiranju i naposljetku raspoređivanje (određivanje redoslijeda izvršavanja), kao različitih faza postavljanja programa na sklopovlje. Također, autori u [12] smatraju da mapiranje obuhvaća tri problema: particioniranje, dodjelu (eng. *assignment*), koja je ekvivalent pojmu *allocation* i raspoređivanje. Usprkos velikoj raznolikosti pojmova, kao i njihovih definicija, razlike između pojmova raspoređivanja (*scheduling*), dodjeljivanja (*allocation*) i mapiranja (*mapping*) sve su manje. Zbog toga se navedeni

[16] Često kroz rad koristimo i pojmove: ciljevi, kriteriji optimizacije, tražene vrijednosti pokazatelja kvalitete i zahtjevi korisnika te ih smatramo ekvivalentima.

pojmovi često isprepliću u povezanoj literaturi, ali i zbog toga što je priroda današnjih programa i sklopovlja visoko raznorodna, pa je ponekad teško jasno odrediti granice između njih.

Prema dosad navedenom i u okviru našeg rada definiramo pojam dodjeljivanja (*allocation*):

Definicija 2.1 ***Dodjeljivanje*** *programa sklopovlju je postupak odlučivanja za svaku jedinicu programa na kojoj će se obradbenoj jedinici sklopovlja izvršiti, u svrhu postizanja traženih ciljeva (najčešće performansi).*

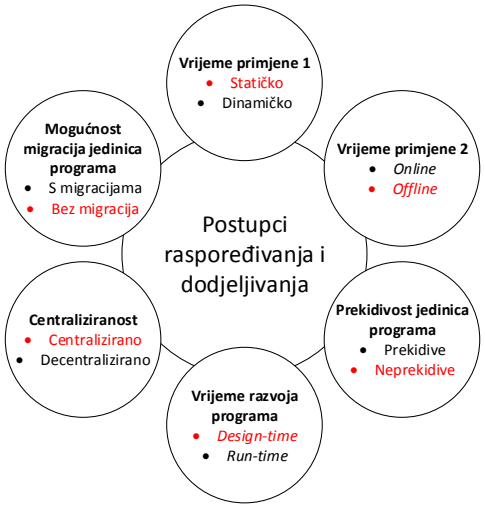
Dakle, postupak dodjeljivanja funkcija je preslikavanja skupa izvršnih jedinica na skup obradbenih jedinica sklopovlja $g : V \to P$.

S druge strane definiramo pojam raspoređivanja (*scheduling*) kao:

Definicija 2.2 ***Raspoređivanje*** *programa na sklopovlje obuhvaća pojam dodjeljivanja, ali i određivanje redoslijeda i vremenskog rasporeda jedinica programa na obradbenim jedinicama sklopovlja računalnog sustava u svrhu postizanja traženih ciljeva (najčešće performansi).*

Nakon što su definirani glavni pojmovi, opisat ćemo postojeće stanje literature u navedenom području te opisati nedostatke istih da bi pojasnili nastanak vlastitih izvornih znanstvenih doprinosa kao odgovora na njih.

U prvom redu, navodimo moguće podjele postupaka raspoređivanja[17] i dodjeljivanja. Skica nekih od glavnih podjela navedenih postupaka prikazana je na slici 2.10. Na njoj su vidljive šest glavnih podjela, dok su crvenom bojom označene postavke svake od tih podjela koje koriste i naši unaprijeđeni postupci raspoređivanja opisani u poglavlju 4.



Slika 2.10: Podjele postupaka raspoređivanja i dodjeljivanja prema dostupnoj literaturi.

Prva je podjela postupaka na **statičke** i **dinamičke**, slično kao i kod modela sklopovlja iz pododjeljka 2.2.2. Dinamički postupci podrazumijevaju trenutno stanje sustava za vrijeme raspoređivanja koje se ažurira nakon raspoređivanja svake jedinice programa. Također, kod tih postupaka potpuno stanje sustava (programa i sklopovlja), ne mora biti u postupnosti poznato prije početka raspoređivanja (primjerice

[17] S obzirom na to da u hrvatskom jeziku pojam "raspoređivanje" može značiti isto što i pojam "dodjeljivanje", kroz ovaj ćemo rad koristiti pojam "raspoređivanje" kao zajednički i za pojam dodjeljivanja i za pojam raspoređivanja, osim kada je potrebno naglasiti razliku između njih.

broj dostupnih obradbenih jedinica, zauzeće memorije, stanje mreže itd.). Dinamičko raspoređivanje ponekad se smatra ekvivalentom raspoređivanju za vrijeme izvođenja programa (eng. *run-time*), kao što je napomenuto u [13], što spada također u jednu od navedenih podjela sa slike 2.10. Statički postupci, s druge strane, podrazumijevaju da su relevantni parametri sustava poznati prije početka raspoređivanja i ne promatraju promjene sustava osim onih koje radi sam postupak raspoređivanja. Postoje i druge varijante, primjerice autori u [21] podrazumijevaju dinamički postupak, ali dijele vrijeme izvođenja na manje dijelove kako bi mogli promatrati statičko ponašanje sustava. Također, postupci poput LDCP-a iz [37] smatraju se statičkima iako se u njima odvija prilagodba za vrijeme raspoređivanja, (u ovom slučaju dinamičko rangiranje izvršnih jedinica ovisno o trenutnom rasporedu).

Podjela na već spomenute ***run-time*** postupke i postupke raspoređivanja za vrijeme dizajna programa **(eng. *design-time*)** govori u kojem razdoblju razvoja programa se on raspoređuje na sklopovlje. Kod *design-time* postupaka to se odvija pri razvoju programa u slučaju kada se unaprijed znaju parametri platforme na kojoj će on biti izvršavan (često kod izrade ugrađenih računalnih sustava, eng. *hardware-software codesign*, kao u [71]) ili barem ograničenog broja platformi na kojima će on biti pokretan (za grozd računala [17,63], MPSoC [12,40] itd.) te općenitog slučaja za velik broj platformi, poput postupaka raspoređivanja pomoću rangiranih listi zadataka [13, 38, 41]. Karakteristike *run-time* postupaka vrlo su slične dinamičkima (ponekad i jednake) i oni se često koriste kao raspoređivači poslova u grozdovima računala, poput primjera iz [19]. Za manje se zadatke rjeđe koriste zbog njihove velike složenosti ili se obično koriste u drastično pojednostavljenim oblicima, poput metode krađe poslova (eng. *work-stealing*) u [48, 58].

Slična je podjela i na ***online*** i ***offline*** postupke, gdje se za *online* postupke smatra da se prilagođavaju novonastalim stanjima sustava za vrijeme raspoređivanja, kao i da se koriste u sustavima gdje postoji konstantan pritok zadataka ili poslova, također najviše u grozdovima računala, primjerice u [17], ali i u upravljačkim aplikacijama, primjerice [90]. *Offline* postupci s druge strane, kao što im i samo ime govori, podrazumijevaju samo stanja neposredno prije početka raspoređivanja programa i neposredno nakon što je cijeli program raspoređen, također, autori u [41] tvrde da je statičko raspoređivanje ujedno i *offline*.

Postoje postupci koji raspoređuju **prekidive** jedinice programa, tj. one koje se mogu zaustaviti i ponovno pokrenuti, poput poslova na grozdovima računala ili oblacima računala, prema [91], ali i drugih, [16, 65] i kao takvi moraju podrazumijevati dodatne parametre koji povećavaju njihovu složenost, (npr. kada se jedinica programa smije prekinuti, da li se uopće smije prekinuti i sl.). Većina postupaka razmatra jedinice programa koje su monolitne, (**neprekidne**) ili ih slučaju kada su prekidne pojednostavljuju podjelom na manje zadatke.

Još jedno od svojstava jedinica programa koje zahtijeva posebne metode raspoređivanja je mogućnost njihovih **migracija**. Naime, neki izvori opisuju problem raspoređivanja jedinica programa koje nakon što su raspoređene i počele s radom mogu biti premještene na druge obradbene jedinice u svrhu postizanja određenih pokazatelja kvalitete. Najčešće su to ujednačavanje opterećenja, primjerice *Work-stealing* u [16, 58] ili postizanje sporednih ciljeva (drugih kriterija optimizacije), poput energije u [70]. Postupci raspoređivanja za prekidive jedinice programa često su dinamičke naravi (*online*, *run-time*).

Posljednja podjela tiče se samog sustava koji raspoređuje programe. Naime, on može biti **centra-**

liziran i **decentraliziran**. Centralizirani sustav za raspoređivanje programa na sklopovlje isti je za cijeli sustav na koji se raspoređuju programi i sadrži sve informacije o njemu. U slučaju kada je sustav prezahtjevan za centralno raspoređivanje, bilo zbog velikog broja obradbenih jedinica (što je čest slučaj kod golemih sustava HPC-a), bilo da je sam sustav decentraliziran, (sustavi HPC-a sastavljeni od manjih zasebnih cjelina), koriste se decentralizirani postupci raspoređivanja, kao onih koji se spominju u [18]. Decentralizirano raspoređivanje vrlo je specifičan problem ograničene primjene i zato ga ne razmatramo ovim istraživanjem.

S obzirom na to da postoji vrlo velik broj postupaka raspoređivanja u povezanoj literaturi, na slici 2.11 nalazi se popis onih koje smo pokrili vlastitim istraživanjem. Crvenom bojom označeni su najčešće korišteni postupci (samim time i najbolji), dok su plavom bojom označeni oni koje smo izravno ili neizravno koristili u ovom radu.

Prethodno spomenuta slika također pokazuje da ih mi dijelimo u četiri najvažnije skupine, od kojih će svaka biti opisana u zasebnim pododjeljcima koji slijede.

2.3.1 Postupci raspoređivanja na temelju rangiranih listi jedinica programa

Prva skupina postupaka raspoređivanja, prema slici 2.11 postupci su raspoređivanja pomoću **rangiranih listi** jedinica programa (eng. *list scheduling*), ujedno i najpopularnija skupina postupaka koji s relativno malom složenosti (prema [13, 41]) dobivaju rješenja blizu optimalnih.

Postupci rangiranjem jedinica programa	Iterativni i bio-inspirirani postupci (metaheuristika)	Hibridni postupci	Specifični postupci
• HEFT • LDCP • PEFT • DLS • CPOP • PCT • BIL • DBUS • PATC	• Genetski algoritmi (GA) • Evolucijski algoritmi (EA) • Diferencijalna evolucija (DE) • Simulirano kaljenje • Cjelobrojno linearno programiranje (ILP) • Simulator rijetkih događaja (MaTCH) • Iscrpno pretraživanje (ES) • Tabu pretraživanje • Algoritam kolonije pčela (BCO) • Algoritam kolonije mrava (ACO) • Algoritam staničnog nasljedstva	• GA + energetski svjesno raspoređivanje • HEFT + djelomično optimalno labavljenje • ACO + rangiranje jedinica programa (ETAHM) • Pohlepni algoritam + SLVL	• Particioniranje grafa • Pohlepni algoritam • Krađa posla • *MapReduce* • *Bin-packing* • *Backfilling* • Programiranje sinkronih skupina (BSP) • Problem dodjele u obliku linearnog zbroja (LSAP) • DES • Poredana stabla • Gustoća energije s namjanjim gubitkom (LLED) • Chebyshevljev leksikografski težinski postupak • Umnažanje zadataka • UDFS • KL • Min-min • Maks-min

Slika 2.11: Postupci raspoređivanja iz pokrivene literature.

Navedeni postupci koriste model programa u obliku usmjerenoga acikličkoga grafa (DAG-a) i podrazumijevaju poznate troškove (vremena) izvođenja svih jedinica programa na svim obradbenim jedinicama sklopovlja. Takvi postupci pretpostavljaju da lukovi DAG-a predstavljaju asinkronu komunikaciju, na način da pošiljatelj podatka može nastaviti s radom čim je započeo slanje podatka, dok primatelj podatka mora čekati dok se komunikacija ne završi kako bi nastavio s radom (dok ne primi podatke), što se podrazumijeva između ostalih u [13, 37, 39, 41, 42]. Isti izvori pretpostavljaju i da se svi komunikacijski pozivi

obavljaju zasebnim komunikacijskim kanalima, da je moguće preklapanje komunikacije i obrade podataka te da su poznati iznosi komunikacije u obliku težina grafa. S takvim pretpostavkama nije moguće opisati asinkronu komunikaciju i kvalitetno tretirati raznorodne komunikacijske veze među obradbenim jedinicama sklopovlja, (što će pokazati i naša eksperimentalna analiza u poglavlju 5), nedostatke koje ispravljamo našim postupkom RaNDKiP (doprinos D.3.). Postupci raspoređivanja na temelju rangiranih listi jedinica programa zasnivaju se na dva glavna koraka:

1. **rangiranje** jedinica programa,
2. **raspoređivanje** jedinica programa prema vrijednosti njihovog ranga.

Rangiranje jedinica programa provodi se na osnovu njihovih poznatih karakteristika. To su isključivo njihovi vremenski troškovi, tj. vremena $w_{i,j}$ izvođenja svake jedinice programa v_i na svim obradbenim jedinicama sklopovlja p_j i njihova međusobna komunikacija $e_{i,j}$. Postoje tri glavna načina rangiranja izvršnih jedinica i svi se zasnivaju na njihovoj udaljenosti od ulaznog ili od izlaznog čvora DAG-a:

1. ***b-level*** (eng. *bottom-level*) ili ***urank*** (eng. *upward rank*) — udaljenost od dna grafa, tj. od najdaljeg izlaznog čvora,
2. ***t-level*** (eng. *top-level*) ili ***drank*** (eng. *downward rank*) — udaljenost od vrha grafa, tj. od najdaljeg ulaznog čvora,
3. ***b-level + t-level*** — zbroj udaljenosti od najdaljeg ulaznog čvora i udaljenosti od najdaljeg izlaznog čvora.

Udaljenost *b-level*, *urank* ili rastući rang jedinice programa v_i definira se kao:

$$urank_i = w_i + \max_{v_j \in sljed(v_i)} \{c_{i,j} + urank(v_j)\}, \tag{2.4}$$

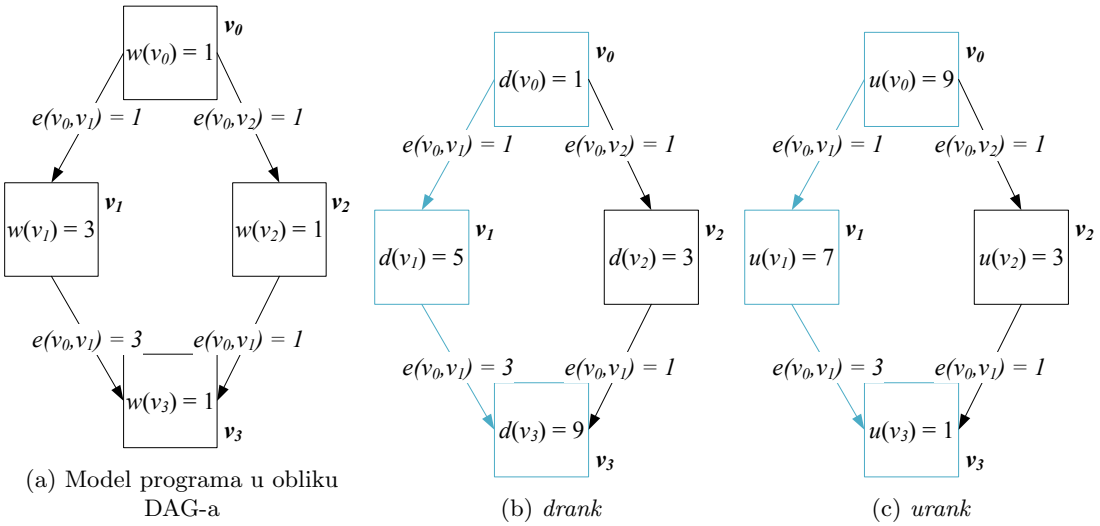
dok se *t-level*, *drank* ili padajući rang jedinice programa v_i definira kao:

$$drank_i = \max_{v_j \in pred(v_i)} \{drank(v_j) + w_j + c_{j,i}\}, \tag{2.5}$$

gdje je $sljed(v_i)$ skup neposrednih sljedbenika jedinice programa v_i, $pred(v_i)$ je skup neposrednih prethodnika vrha v_i u modelu DAG-a programa, w_i je trošak (vrijeme) izvođenja vrha v_i i $c_{i,j}$ je obujam komunikacije između vrhova v_i i v_j. Put od ulaznog do izlaznog čvora grafa (ili obratno) u kojem se biraju jedinice programa s najvećim vrijednostima *urank*-a ili *drank*-a zove se **kritični put**.

Na slici 2.12 dan je primjer rangiranja jednostavnog grafa. Izvorni graf prikazan je na slici 2.12a, odgovarajuće vrijednosti za *drank*, $d(v_i)$ prikazane su na slici 2.12b, dok su vrijednosti za *urank*, $u(v_i)$ prikazane na slici 2.12c s pripadajućim kritičnim putevima označenima plavom bojom.

Najčešće rangiranje jest pomoću *urank*-a, predstavnici kojeg su postupci HEFT [41] (eng. *heterogeneous estimated finish time*), LDCP [37] (eng. *longest dynamic critical path*), PATC [85] (eng. *power aware task clustering*) i hibridni postupci koji sadrže navedene, poput energetski svjesnog postupka u [44] ili djelomičnog optimalnog labavljenja iz [39].



(a) Model programa u obliku DAG-a (b) *drank* (c) *urank*

Slika 2.12: Primjer rangiranja jedinica programa kod postupaka raspoređivanja.

Način rangiranja broj tri (zbroj rastućeg i padajućeg ranga), koristi se u postupku CPOP [41] (eng. *critical path on processor*).

Nakon što se dobiju rangovi svih jedinica programa (rangiranje se radi rekurzivno počevši od ulaznih čvorova za *urank* odnosno od izlaznog čvora ua *drank*), one se raspoređuju redoslijedom prema padajućoj vrijednosti njihovog ranga. Raspoređivanje se vrši tako da se za svaku obradbenu jedinicu stvara vremenska linija (eng. *timeline*). Za svaku jedinicu programa bira se vremenska linija obradbene jedinice na kojoj će vrijeme njezinog završetka pri izvođenju biti minimalno. Postoji i napredniji postupak, takozvana **strategija (pravilo) umetanja** (eng. *insertion policy*), gdje se za svaku obradbenu jedinicu traži praznina u rasporedu u koju se može smjestiti izvršna jedinica koja se raspoređuje i to tako da njezino vrijeme završetka bude minimalno, pazeći na vremenski slijed (svi prethodnici moraju biti izvršeni).

Predvodnik postupaka raspoređivanja rangiranjem izvršnih jedinica je popularni HEFT (eng. *heterogeneous estimated finish time first*) iz [41], koji će biti detaljno objašnjen u pododjeljku 2.3.5 i poslužiti kao postupak za usporedbu u eksperimentalnoj analizi. Drugi je tzv. kritični put na procesoru (eng. *critical path on a processor*, CPOP) od istih autora. Zatim, to je postupak najdužeg dinamičkog kritičnog puta (eng. *longest dynamic critical path*, LDCP) iz [37], na kojem se zasniva i naš unaprijeđeni postupak raspoređivanja RaNDKiP opisan u pododjeljku 4.2, zatim postupaka s predviđanjem najranijeg vremena završetka (eng. *predict earliest finish time*, PEFT) iz [13] itd.

Nedostatci raspoređivanja rangiranjem jedinica programa su već spomenuti problemi u opisu komunikacije sustava HPC-a (sinkronizacija), zatim, potrebno je poznavati troškove izvođenja svih jedinica programa na svim obradbenim jedinicama sklopovlja (što nije uvijek slučaj u praksi) te ograničenost tih postupaka na jedan pokazatelj kvalitete — skraćivanje duljine rasporeda (eng. *schedule length*). Vlastitim postupcima raspoređivanja, kao okosnice naših izvornih znanstvenih doprinosa (D.2. i D.3.), tretirali smo te nedostatke. Prvo, sinkronu komunikaciju opisujemo u vlastitom modelu programa — miješanim acikličkim grafom, opisanom u pododjeljku 3.1, troškove izvođenja jedinica programa na sklopovlju računamo prema zasebnim podatcima za programe i sklopovlje (ne podrazumijevamo poznavanje troškova izvođenja jedinica programa na svim obradbenim jedinicama sklopovlja, postupak RaPPaMaG)

i omogućujemo proširenje na više pokazatelja kvalitete (postupci RaPPaMaG i RaNDKiP).

2.3.2 Iterativni (metaheuristički) i bio-inspirirani postupci raspoređivanja

U prethodnom pododjeljku opisani su postupci raspoređivanja koji omogućavaju postizanje jednog cilja, a to je minimiziranje vremena izvršavanja programa na sklopovlju. U slučaju kada su zahtjevi složeniji (ima ih više) navedeni postupci ne mogu odgovoriti na njih. Takva situacija sve je češća u praksi pojavom zahtjeva poput smanjenja utroška energije (time i troškova), povećanja skalabilnosti, pouzdanosti, ujednačenosti opterećenja itd. Iz tog razloga problem raspoređivanja programa na sklopovlje postaje višekriterijski (eng. *multiobjective*). Uz to, problem postaje složeniji činjenicom da su navedeni kriteriji (ciljevi) međusobno ovisni, primjerice skraćivanjem vremena izvođenja programa povećava se utrošak energije, povećanjem utroška energije smanjuje se pouzdanost i sl. Rezultat rješavanja takvog višekriterijskog problema je skup tzv. Pareto-optimalnih rješenja ili samo Pareto rješenja, prema [16,44]. Prema istim izvorima, u takvom skupu rješenja poboljšanje jednog kriterija znači pogoršavanje drugih, uzevši u obzir njihovu ovisnost. Općeniti problem višekriterijske optimizacije prema [92–94] jest:

$$\begin{aligned} \text{minimizirati}_{\mathbf{q}} \quad & \mathbf{U}(\mathbf{q}) = (u_0(\mathbf{q}), u_1(\mathbf{q}), ..., u_{k-1}(\mathbf{q}))^T \\ \text{unutar} \quad & g_j(\mathbf{q}) \leq 0, \;\; j = 0, 1, ..., o_1 - 1, \qquad (2.6) \\ & h_l(\mathbf{q}) = 0, \;\; l = 0, 1, ..., o_2 - 1, \qquad (2.7) \end{aligned}$$

gdje je k broj funkcija cilja (kriterija), o_1 je broj ograničenja u obliku nejednakosti i o_2 je broj ograničenja u obliku jednakosti. $\mathbf{q} \in B^\beta$ je vektor varijabli dizajna (varijabli odluke), gdje je β broj nezavisnih varijabli x_i. $\mathbf{U}(\mathbf{q}) \in B^k$ je vektor funkcija cilja $u_i(\mathbf{q}) : B^\beta \to B^l$. Funkcije $u_i(\mathbf{q})$ se također zovu kriteriji, ciljevi, funkcije podmirenja (eng. *payoff*), funkcije troška (eng. *cost functions*) ili funkcije vrijednosti (eng. *value functions*), prema [92].

Kao odgovor na navedeni problem, nastaje niz **metaheurističkih**, većinom **iterativnih**, postupaka raspoređivanja koji pretražuju čitav skup rješenja (mogućih rasporeda) konvergirajući prema boljem. To znači da se provjeravaju svi mogući načini na koje je moguće izvesti raspored i za svaki se provjerava vrijednost vektora funkcije cilja $\mathbf{U}(\mathbf{q})$ je li ona bolja od drugih. S obzirom na to da je takvih mnogo, obično se provjerava samo malen skup mogućih rješenja, čime se mogu izbjeći dobra. U području raspoređivanja programa na sklopovlje značilo bi da se provjerava dobiveni raspored ispunjava li on tražene zahtjeve bolje od prethodno analiziranih (vrijeme izvođenja, utrošak energije, razina pouzdanosti sustava itd.). Navedeni se postupci još zovu i **postupci vođenog lokalnog pretraživanja** (eng. *guided local search*).

Postupci raspoređivanja koji rješavaju navedeni problem većinom su bi-kriterijski (dva cilja pri raspoređivanju), primjerice [11,17,44,70], dok se rijetko nalaze izvori s više od dviju funkcija cilja, primjerice tri u [12], zatim veći broj u [36] (samo općenita formulacija problema) i u [31] u postupku mapiranja programa na višeprocesorske SoC sustave (pri njihovom dizajnu).

Metode rješavanja višekriterijskog problema raspoređivanja su također vrlo raznolike. Noviji je trend upotreba raznih algoritama inspiriranih prirodnim pojavama — bio-inspiriranima (eng. *bio-inspired*)

za rješavanje mnogih problema u računalnim znanostima, pa tako i za raspoređivanje programa na sklopovlje. Primjerice, to je optimizacija kolonije mrava (eng. *ant colony optimization*, ACO) iz [34] (u kombinaciji s rangiranjem jedinica programa). U navedenom izvoru se rješenjima daje veći značaj s manjim energetskim tragom, što je simulacija ponašanja mrava, gdje se putevi kojima je prošlo više mrava bolje prihvaćaju od strane drugih mrava. Zatim optimizacija kolonijom pčela (eng. *bee colony optimization*) iz [17], gdje se oponaša ponašanje pčela kada pronađu cvijet, one dulje plešu što je cvijet bolji, što je poslužilo kao predložak i sukladno tome se pridodaju i važnosti za duljinu rasporeda i ukupno vrijeme potrebno za prijenos podataka iz spremišnih čvorova.

Evolucijski algoritmi (eng. *evolutionary algorithm*, EA) su skupina trenutno najpopularnijih algoritama iz skupine metaheurističkog raspoređivanja i predstavlja skup algoritama koji simuliraju prirodnu selekciju u evoluciji organizama, a koriste se između ostalih u [31,36,95]. U tu skupinu spadaju algoritmi diferencijalne evolucije (eng. *differential evolution*, DE), korišteni u [16], zatim genetski algoritmi (eng. *genetic algorithms*, GA), korišteni u [16, 34, 44, 62] koji su inspirirani ponašanjem genetskog materijala te sadržavaju faze poput mutacije, selekcije i križanja u odabiru rješenja (odabere se inicijalno rješenje koje se navedenim fazama evoluira). Naposljetku se koristi još i simulirano kaljenje, npr. u [40, 96, 97], zatim optimizacija rojem čestica (eng. *particle swarm optimization*, PSO) u [51] itd.

Od iterativnih postupaka koji nisu bio-inspirirani značajni su cjelobrojno linearno programiranje (eng. *integer linear programming*, ILP) iz [11, 31], zatim simulator rijetkih događaja (eng. *rare event simulator*) iz [62], iscrpno pretraživanje u [12, 86] itd.

Glavni nedostatak metaheurističkih postupaka jest da rješenje mogu naći u lokalnom optimumu, a ono se može drastično razlikovati od globalnog optimuma (optimalnog rješenja). Navedeni se problem rješava tako da se poveća broj iteracija traženja rješenja, ali se time povećava i trajanje raspoređivanja (složenost postupka). No, to također nije jamac pronalaska boljeg rješenja. Napredak u ovom području uglavnom se odnosi na unaprjeđivanje postupaka odbacivanja nevaljalih rješenja i inteligentan odabir početnih rješenja. Manji se napori ulažu u smanjivanje prostora rješenja, čime bi se poboljšale performanse postupka (njegovo trajanje), ali i omogućio pronalazak rješenja bliže optimalnima. Upravo to čini naš unaprijeđeni postupak RaPPaMaG (doprinos D.2.) kao dio našeg programskog okvira LOSECO (doprinos D.1.) i predstavlja napredak u tom smjeru.

Idući skup postupaka su hibridni, kao i postupci specifični problemu te su opisani u narednom pododjeljku.

2.3.3 Hibridni i ostali postupci raspoređivanja

Ostali postupci raspoređivanja dijele se na vrlo **specifične postupke** ograničene primjenjivosti i **hibridne postupke** koji kombiniraju dva ili više različita pristupa pri raspoređivanju programa na sklopovlje. Kako postoji velik broj navedenih postupaka, ovdje ćemo navesti samo najvažnije te one koji su dijelom zaslužni i za naš rad. Od specifičnih postupaka neki od najzastupljenijih u literaturi su *MapReduce* u [21], koji se zasniva na dvije faze: mapiranje (eng. *map*) u kojoj se filtriraju podatci i sakupljanja (eng. *reduce*) u kojoj se vrše zbirne operacije nad odabranim podatcima. Koristi se za vrlo velike skupove podataka u nakupinama računala. Pohlepni algoritmi (eng. *greedy algorithm*) od kojih je najpoznatiji

algoritam krađe poslova (eng. *work stealing*) u [48, 58], koji radi na principu da se jedinice programa ravnomjerno rasporede na obradbene jedinice sklopovlja, a zatim se u slučaju da se na nekoj od njih red isprazni uzimaju izvršne jedinice s kraja reda na drugim obradbenim jedinicama. Postupak Min-min iz [11,35] stavlja po jednu jedinicu programa na jednu obradbenu jedinicu sklopovlja (redom), nakon što se postavi po jedna jedinica programa na svaku od jedinica sklopovlja, za idući jedinicu programa, ali i za sve naredne bira se jedinica sklopovlja takva da ukupno trajanje dosad raspoređenih jedinica programa (trenutnog rasporeda) bude minimalno (kod postupka Maks-min traži se da bude maksimalno).

Poseban je slučaj vrsta **raspoređivanja particioniranjem grafova** (eng. *graph partitioning*), koje se koristi u [10, 58, 67] te drugih postupaka grupiranja jedinica programa poput postupka *bin-packing* iz [66]. Navedeni postupci koriste strukturu programa (koja je u obliku grafa) kako bi grupirali (razrijedili) taj graf (eng. *coarsening*). Postupak svrstava vrhove težinskog grafa u n izdvojenih skupina približno jednakih težina (primjerice njihovo skupno vrijeme izvođenja) u isto vrijeme minimizirajući troškove reza (zbroj težina dodijeljenih bridovima grafa koji spajaju dobivene particije), čineći NP-težak problem prema [58, 77]. Postupci particioniranja grafova su temelj ideje za prvi korak našeg unaprijeđenog postupka RaPPaMaG, u kojem se koriste specifičnosti strukture programa u obliku grafa za njegovu podjelu na vremenski nezavisne skupine jedinica programa (doprinos D.2.).

Od hibridnih postupaka najzastupljeniji su oni koji kombiniraju postupke raspoređivanja zasnovane na rangiranju jedinica programa s drugim postupcima (najčešće vođenog lokalnog pretraživanja). Primjerice, autori u [44] kombiniraju genetski algoritam i energetski svjesno raspoređivanje (eng. *energy-conscious scheduling*), tako da genetskim algoritmom dobiju skup vremenski (pod)optimalnih rješenja u kojima tada traže one s najmanjim utroškom energije. Autori u [34] koriste postupak ETAHM, koji se sastoji od optimizacije kolonije mrava s postupkom rangiranja jedinica programa i evaluacije potrošnje energije dobivenih rješenja u drugom koraku. Od izdvojenih postupaka to su kombinacija tzv. algoritama **labavljenja vremenskih isječaka** (eng. *slacking*) i algoritama raspoređivanja, primjerice u [39] gdje autori koriste HEFT kao postupak raspoređivanja i tzv. djelomično optimalno labavljenje (eng. *partial optimal slacking*, POS). Navedeni se postupak zasniva na povećanju trajanja nekritičnih jedinica programa (kod kojih to ne utječe na duljinu rasporeda) kako bi se uštedjela energija (smanjivanjem radne frekvencije obradbenih jedinica). Navedeni oblik hibridizacije upotrebljavamo i u vlastitom postupku RaNDKiP (doprinos D.3.) opisanom kasnije.

Kroz prethodne pododjeljke opisane su karakteristike glavnih skupina raspoređivanja te su bili navedeni i neki njihovi nedostatci. U svrhu isticanja smjera i važnosti naših doprinosa, navedene ćemo nedostatke sažeti u sljedećem pododjeljku te navesti naše odgovore na njih u obliku vlastitih unaprijeđenih postupaka raspoređivanja.

2.3.4 Nedostatci postojećih postupaka raspoređivanja

Kako bi dodatno obrazložili napore u ostvarivanju naših izvornih znanstvenih doprinosa u ovom pododjeljku prikazat ćemo nedostatke postojećih analiziranih postupaka raspoređivanja te naše odgovore na njih.

Jedna od pretpostavki postupaka raspoređivanja, posebice onih zasnovanih na rangiranim listama

jedinica programa, što je i spomenuto u pododjeljku 2.3.1, jest da vremena izvršavanja individualnih jedinica programa na svim obradbenim jedinicama sklopovlja, kao i iznosi njihove međusobne komunikacije moraju biti poznati prije početka postupka. Ta pretpostavka tvori i jedan od dvaju velikih nedostataka postojećih postupaka tog tipa:

- za poznavanje svih vremena izvođenja potrebno je pokrenuti sve izvršne jedinice na svim obradbenim jedinicama sklopovlja, što je već u slučaju manjeg broja istih vremenski zahtjevan postupak.

Kao odgovor na taj nedostatak uveli smo apsolutni prikaz parametara programa i sklopovlja zasebno, u obliku obradbenih i komunikacijskih zahtjeva jedinica programa (instrukcija koje treba izvršiti) s jedne strane te obradbene i komunikacijske sposobnosti obradbenih jedinica s njihovim energetskim karakteristikama s druge. Navedeni parametri podrobnije su pokriveni u opisu našeg unaprijeđenog postupka RaPPaMaG u pododjeljku 4.1 i dio su naših izvornih znanstvenih doprinosa D.1. i D.2. Iz takvog oblika parametrizacije programa i sklopovlja mogu se po potrebi dobiti obradbeni (i drugi) troškovi na pojedinim obradbenim jedinicama sklopovlja, potrebni za postupke raspoređivanja zasnovanih na rangiranim listama jedinica programa.

Postoji jedan nedostatak takvog prikaza, naime, nije uvijek moguće procijeniti parametre koji će vrijediti u svim situacijama. Primjerice pokretanjem iste izvršne jedinice (sa zadanim brojem instrukcija koje treba obaviti) na dvama procesorima istih glavnih karakteristika (frekvencije i pričuvne memorije), ali različitih skupa instrukcija (primjerice SSE2, SSE3, SSSE3, SSE4 na Intel procesorima i 3DNow! na AMD procesorima) može dovesti do razlika u izvođenju i vremenski, ali i što se tiče utroška resursa (memorije, energije). No opet, u takvim se slučajevima može izvršiti reprezentativni skup jedinica programa na osnovu čega se tada mogu raditi procjene i za ostale jedinice (na principu kojeg su zasnovane i ideje računalnih motiva i idioma, prema [55, 56, 98]). Međutim, za dalji rad pretpostavlja se da je navedeni prikaz dovoljno točan, što će biti i potvrđeno eksperimentalnom analizom u poglavlju 5.

Drugi je nedostatak postojećih postupaka raspoređivanja:

- često se podrazumijeva da je mreža homogena, čime se smanjuje mogućnost primjene postupaka za komunikacijski zahtjevnije programe na sustavima s raznorodnom komunikacijskom strukturom.

Navedeni nedostatak prisutan je u već spomenutim postupcima raspoređivanja HEFT, LDCP, PEFT, CPOP i mnogim drugim. Postupak LDCP, prema [37], primjerice, nema podršku za raspoređivanje komunikacijskih poziva, čime raspored ne preslikava stvarnu situaciju koja se događa pokretanjem takvog programa na sklopovlju u slučaju da je iznos komunikacije značajan. To ispravljamo našim doprinosom D.3. (postupak RaNDKiP), što će pokazati i eksperimentalna analiza u poglavlju 5. Postupci HEFT, PEFT i CPOP, prema [13, 41] uzimaju u obzir prosječne vrijednosti komunikacijskih propusnosti mreže sklopovlja, čime djelomično podrazumijevaju potencijalno raznorodnu narav komunikacijske strukture sklopovlja, no još uvijek ne u dovoljnoj mjeri (što također pokazuje naša eksperimentalna analiza). Iako, navedeni postupci barem imaju podršku za raspoređivanje samih komunikacijskih poziva.

Treći je nedostatak postojećih postupaka:

- prave se optimistične pretpostavke u smislu komunikacijske strukture sklopovlja.

Navedeno je prisutno u ranije spomenutim postupcima. Naime, postupak LDCP prema [37] pretpostavlja tzv. *unity* pristup, što znači potpuno povezanu homogenu mrežu obradbenih jedinica sklopovlja. Ostali postupci podrazumijevaju, osim potpune povezanosti i činjenicu da se komunikacija može odvijati u isto vrijeme kad i obrada podataka, neovisno o njoj. To nije slučaj u nekim programskim i sklopovskim paradigmama domene HPC-a, neke od kojih smo i naveli u pododjeljku 2.1 poput MPI i OpenMP komunikacijskih i sinkronizacijskih poziva. Zato je jedan od važnijih postulata u izradi naših unaprijeđenih postupaka raspoređivanja RaPPaMaG i RaNDKiP te strukture istih u obliku programskog okvira LOSECO kao izvornih znanstvenih doprinosa D.1., D.2. i D.3. bio podrobnije preslikavanje komunikacijske strukture.

Četvrti nedostatak odnosi se na postupke raspoređivanja višekriterijskom analizom ciljeva (optimizacijom):

- prostor mogućih rješenja prevelik je da bi se mogla jamčiti kvalitetna rješenja (raspored).

Zbog navedenog nedostatka mnogi postupci raspoređivanja koriste napredne metaheurističke postupke opisane u pododjeljku 2.3.2. U tu svrhu, kao dio našeg izvornog znanstvenog doprinosa D.2., za jednu od faza unaprijeđenog postupka raspoređivanja RaPPaMaG uveli smo particioniranje grafa, kao oblik smanjivanja prostora mogućih rješenja.

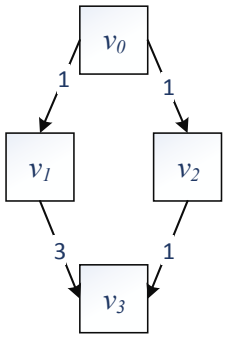
Naravno, s obzirom na to da je ovo područje istraživanja vrlo aktivno, za pretpostaviti je da ima i drugih nedostataka postojećih postupaka, no oni prema našem mišljenju izravno ili neizravno slijede iz ovdje navedenih, tako da ih nećemo detaljno opisivati.

2.3.5 Postupak raspoređivanja HEFT

S obzirom na to da je postupak HEFT reprezentativan kod postupaka raspoređivanja rangiranjem jedinica programa te ga mnogi autori smatraju jednim od najboljih postupaka u tom području, u ovom pododjeljku detaljnije ćemo ga opisati. Još jedan razlog tome jest što će nam poslužiti za usporedbu s jednim od naših unaprijeđenih postupaka raspoređivanja u eksperimentalnoj analizi (RaNDKiP i njegovih izvedbi RaNDKiP1 i RaNDKiP2 opisanih u pododjeljku 4.2), ali i za opis nekih osnovnih koncepata sadržanih u postupcima raspoređivanja rangiranjem jedinica programa, od kojih smo neke i sami koristili.

Kao što je ranije navedeno, postupak raspoređivanja HEFT je jedan od najboljih i najpoznatijih postupaka raspoređivanja programa na sklopovlje u sustavima HPC-a pomoću rangiranja jedinica programa (zadataka). Izvorno, postupak se naziva raznorodno predviđeno najranije vrijeme završetka (eng. *hetergoeneous estimated finish time*), prema [41], termin koji se odnosi na vrijeme završetka svake jedinice programa pri raspoređivanju.

Postupak HEFT, kao i većina postupaka *list scheduling*, podrazumijeva da su poznata vremena izvršavanja svih jedinica programa na svim obradbenim jedinicama sklopovlja, tzv. matrica obradbenih troškova (eng. *computation cost matrix*), prema [13, 37, 41]. Primjer takve matrice dan je na slici 2.13b za program prikazan na slici 2.13a.



(a) Program u obliku DAG-a.

Izv. jed.	p_0	p_1
v_0	1	2
v_1	3	5
v_2	1	2
v_3	1	1

(b) Matrica obradbenih troškova.

Slika 2.13: Primjer ulaznih informacija za postupak HEFT.

Glavne su faze postupka HEFT:

1. **faza rangiranja** jedinica programa, zadataka (eng. *task prioritizing phase*),
2. **faza odabira** obradbene jedinice, procesora (eng. *processor selection phase*).

Prva faza postupka HEFT rangira jedinice programa prema vrijednosti *urank*-a opisanoj pomoću (2.4), no uz neke specifičnosti koje će biti opisane u sljedećem pododjeljku.

Postupak raspoređivanja HEFT: faza rangiranja jedinica programa

Prva faza postupka HEFT rangiranje je jedinica programa, no da bi to bilo moguće, potrebno je odrediti parametre potrebne za dobivanje *urank*-a pomoću (2.4).

Na početku, za svaku jedinicu programa v_i računaju se njezini prosječni troškovi izvođenja po svim obradbenim jedinicama sklopovlja pomoću

$$w_i = \sum_{p_j \in P} w_{i,j}/\rho,\ \rho = |P|\,, \tag{2.8}$$

gdje je $w_{i,j}$ trošak izvođenja jedinice programa $v_i \in V$ na obradbenoj jedinici sklopovlja $p_j \in P$. Karakteristike komunikacijskih veza sklopovlja daju se pomoću ρ-dimenzionalnog vektora L, koji označava troškove početka komunikacije za svaku obradbenu jedinicu (eng. *communication startup cost*), prema [13, 41] i propusnosti $c_{i,j} \in C$ između svih parova obradbenih jedinica p_i i p_j u obliku matrice dimenzija $\rho \times \rho$. Trošak prijenosa podataka (komunikacije) između jedinica programa v_i i v_k definiran je pomoću prosječnih vrijednosti propusnosti komunikacijskih veza sklopovlja $\overline{C}$ i jednak je

$$\overline{c_{i,k}} = \overline{L} + \frac{e_{i,k}}{\overline{C}}, \tag{2.9}$$

gdje je $e_{i,k}$ obujam komunikacije među jedinicama programa v_i i v_k, a $\overline{L}$ je prosječna vrijednost troškova početka komunikacije za sve obradbene jedinice.

Prije nego se odredi funkcija cilja, definira se najranije vrijeme početka $t_{i,j}^{nvp}$ (eng. *earliest start time*, EST) i najranije vrijeme završetka $t_{i,j}^{nvz}$ (eng. *earliest finish time*, EFT) jedinice programa v_i na obradbenoj jedinici sklopovlja p_j. Najranije vrijeme početka ulaznih vrhova grafa v_{ulaz} (čvorovi bez

prethodnika) na obradbenoj jedinici p_j jednako je nuli:

$$t_{ulaz,j}^{nvp} = 0, \tag{2.10}$$

dok je za ostale:

$$t_{i,j}^{nvp} = \max\{t_j^{dost}, \max_{v_k \in pred(v_i)} [t_k^{svp} + c_{k,i}]\}, \tag{2.11}$$

gdje je t_j^{dost} najraniji trenutak kada je obradbena jedinica p_j dostupna za izvršavanje jedinice programa, $pred(v_i)$ je skup prethodnika jedinice programa v_i, a t_k^{svp} je stvarno vrijeme početka izvršavanja jedinice programa v_k. Najranije vrijeme završetka definira se kao

$$t_{i,j}^{nvz} = w_{i,j} + t_{i,j}^{nvp}. \tag{2.12}$$

Ako je v_k posljednja jedinica programa dodijeljena obradbenoj jedinici p_j, tada je t_j^{dost} vrijeme kada je obradbena jedinica p_j završila izvođenje jedinice programa v_k u slučaju da se koristi pravilo bez umetanja (eng. *non-insertion-based policy*), tj. da se jedinice programa uvijek dodaju na kraj rasporeda pojedinih obradbenih jedinica.

Nakon što se jedinica programa v_i dodijeli obradbenoj jedinici p_j dobiva se:

$$t_{i,j}^{nvp} = t_i^{svp} \tag{2.13}$$

$$t_{i,j}^{nvz} = t_i^{svz}, \tag{2.14}$$

gdje je t_i^{svz} stvarno vrijeme završetka jedinice programa v_i. Nakon što se rasporede sve jedinice programa, duljina rasporeda jednaka je stvarnom vremenu završetka izlazne jedinice programa:

$$t^{rasp} = t_{izlaz}^{svz} \tag{2.15}$$

(one bez sljedbenika u grafu programa). U slučaju da ima više izlaznih čvorova $v_{izlaz,i} \in V_{izlaz}$, tada je duljina rasporeda:

$$t^{rasp} = \max_{v_{izlaz,i} \in V_{izlaz}} \left(t_{izlaz,i}^{svz}\right). \tag{2.16}$$

Cilj postupka (optimizacije) raspoređivanje je jedinica programa na obradbene jedinice sklopovlja na način da se minimizira duljina rasporeda.

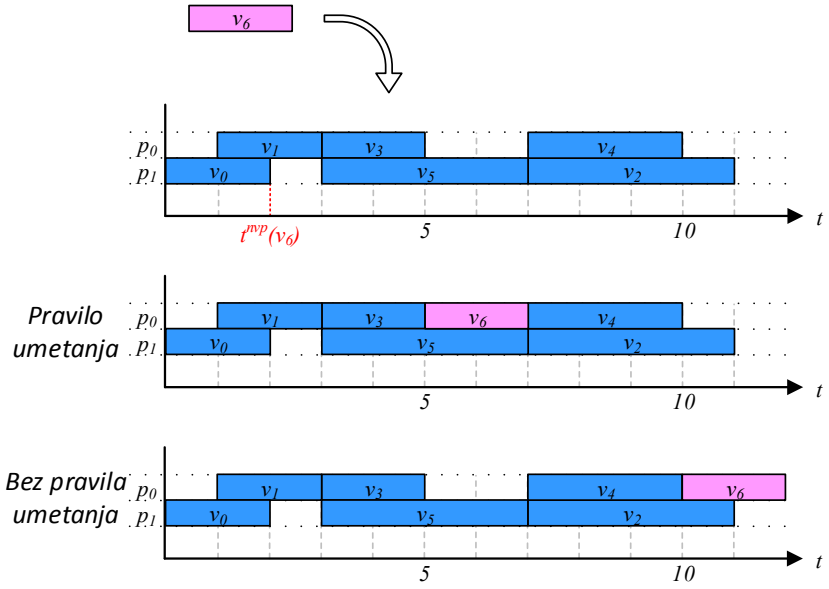
Nakon što se dobiju *urank* vrijednosti svih jedinica programa, prvo se raspoređuje ona s najvećom vrijednosti *urank*-a i to se u slučaju postupka HEFT obavlja putem pravila umetanja (eng. *insertion-based policy*), što je ujedno i druga faza postupka HEFT opisana u sljedećem pododjeljku.

Postupak raspoređivanja HEFT: faza odabira obradbene jedinice

U fazi odabira odgovarajuće obradbene jedinice za izvršavanje jedinice programa v_i traži se prostor u rasporedu takav da će njezino stvarno vrijeme završetka biti minimalno. Navedena faza sastoji se od sljedećih koraka:

1. određivanje t_i^{nvp}, tj. vremena kada su svi prethodnici jedinice programa v_i završili s izvođenjem i ona je primila sve potrebne podatke,
2. u rasporedu na svakoj obradbenoj jedinici traži se vremenski interval koji je veći ili jednak njezinom vremenu izvođenja na toj obradbenoj jedinici.

Navedeno je prikazano i na slici 2.14. U rasporedu na dvije obradbene jedinice p_0 i p_1 raspoređene su jedinice programa $v_0, ..., v_5$. Potrebno je rasporediti sljedeću jedinicu programa v_6, čije je najranije vrijeme početka jednako trenutku kada v_0 završi, $t_6^{nvp} = t_0^{svz}$.



Slika 2.14: Faza dodavanja jedinice programa u raspored s i bez pravila umetanja.

Ukoliko se koristi pravilo umetanja, jedinica programa v_6 stavlja se na obradbenu jedinicu p_0 između v_3 i v_4. Ukoliko se ne koristi pravilo umetanja, ona se stavlja također na obradbenu jedinicu p_0 ali nakon v_4 time produžujući raspored za $t_6^{svz} - t_2^{svz}$.

Složenost postupka HEFT u najgorem slučaju prema [41] je $O(\vartheta^2 \cdot \rho)$, gdje je ϑ ukupan broj izvršnih jedinica programa, a ρ ukupan broj obradbenih jedinica sklopovlja. Najgori je slučaj gust graf programa, odnosno kada se broj bridova grafa približava ϑ^2.

Sljedeći postupak (LDCP) sadržava nekoliko koncepata na kojima se zasniva naš unaprijeđeni postupak raspoređivanja RaNDKiP i koji su prema našemu mišljenju predispozicija za dalja unaprjeđenja postupaka raspoređivanja na temelju rangiranih listi jedinica programa. Iz tog razloga ga podrobno opisujemo u narednom pododjeljku.

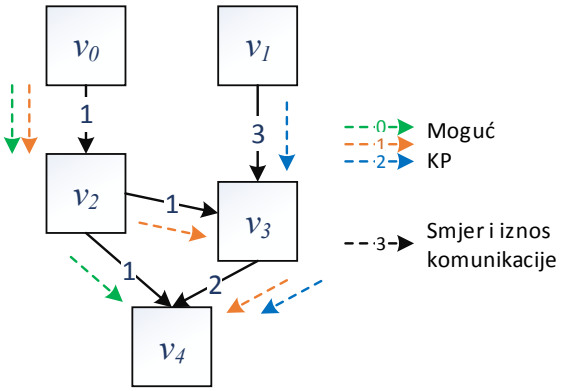
2.3.6 Postupak raspoređivanja LDCP

Postupak raspoređivanja **najdužeg dinamičkog kritičnog puta** (eng. *longest dynamic critical path*, LDCP), prema [37], još je jedan u nizu postupaka statičkog raspoređivanja pomoću rangirane liste jedinica programa. Navedeni postupak osnovni je koncept na kojem su zasnovane inačice naših unaprijeđenih postupaka raspoređivanja RaNDKiP , kao doprinosa D.3. i stoga u ovom pododjeljku dajemo podroban opis tog postupka.

Kao i kod postupka HEFT i ovdje se pretpostavlja da su vremena izvršavanja individualnih jedinica programa na svim obradbenim jedinicama sklopovlja, kao i iznosi njihove međusobne komunikacije, poznati prije početka postupka. Također, pretpostavlja se oblik programa u obliku DAG-a u kojem lukovi predstavljaju smjer i iznos komunikacije (što uvjetuje i smjer izvođenja programa), dok vrhovi predstavljaju jedinice programa (zadatke u ovom slučaju).

Kao primjer u tablici 2.15b su prikazana vremena izvršavanja jedinica programa prikazanog usmjerenim acikličkim grafom sa slike 2.15a na obradbenim jedinicama (procesorima) p_0 i p_1. Na slici 2.15a također su prikazani i podatci o komunikaciji (smjer i obujam) te mogući kritični putevi, KP-ovi (i boje kojima će biti označavani kasnije):

- KP0: $v_0 \rightarrow v_2 \rightarrow v_4$ (zeleni),
- KP1: $v_0 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4$ (narančasti),
- KP2: $v_1 \rightarrow v_3 \rightarrow v_4$ (plavi).



(a) Program u obliku DAG-a i kritični putevi (KP).

Izv. jed.	p_0	p_1
v_0	8	6
v_1	10	8
v_2	5	2
v_3	9	5
v_4	2	1

(b) Vremena izvršavanja jedinica programa.

Slika 2.15: Primjer zadanog programa za postupak LDCP.

Kritični put (KP) za određeni vrh grafa označava najdulji put kojim se od njega može doći do izlaznog vrha grafa, prema [72]. Duljina puta je zbroj obradbenih zahtjeva jedinica programa i iznosa komunikacije lukova koje sadrži. Također, za potrebe opisa navedenog postupka definira se pojam DAG-a obradbene jedinice p_j kao $DAGP_j$, prema [37].

DAGP je DAG iste strukture kao i polazni program u kojemu su svim jedinicama programa vremena izvršavanja postavljena na jednu obradbenu jedinicu (procesor). Primjerice, $DAGP_j$ je DAG kojemu su vremena izvršavanja svih vrhova jednaka vremenima izvršavanja na procesoru p_j.

Dinamički kritični put (DKP) (engl, dynamic critical path, DCP) je najdulji od kritičnih puteva ulaznih vrhova (onih bez prethodnika, ako ih ima više) u određenom DAGP-u. Iz navedenoga slijedi da svaki DAGP ima barem jedan vlastiti DKP (koji se mogu, ali i ne moraju razlikovati među DAGP-ovima).

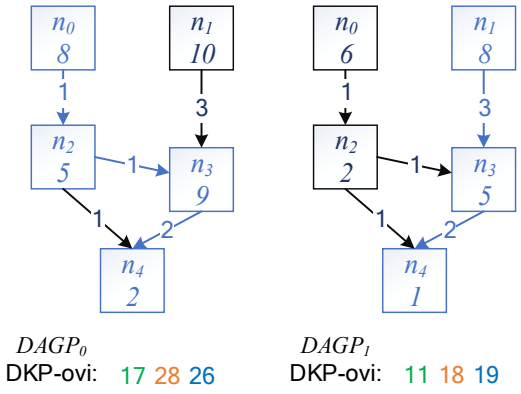
Najdulji dinamički kritični put (NDKP) jest najdulji dinamički kritični put (DKP) od onih iz svih DAGP-ova. On služi kao lista prioriteta jedinica programa koje se prve raspoređuju, što će biti opisano u narednom tekstu.

Postupak LDCP sastoji se od tri glavne faze:

1. **faza odabira zadataka** (eng. *task selection phase*),
2. **faza odabira procesora** (eng. *processor selection phase*),
3. **ažuriranje statusa** (eng. *status update phase*).

U fazi odabira zadataka formira se NDKP prema kojem se zadatci biraju za raspoređivanje. NDKP se kreira na način da se u svakom DAGP-u rangiraju jedinice programa (zadatci) prema njihovom rastućem rangu (*urank*-u), prema (2.4). Nakon toga, odabire se ulazni vrh grafa koji ima najveću vrijednost *urank*-a među svim DAGP-ovima i on se sprema kao potencijalni **ključni vrh** (eng. *key node*). Ključni vrh ujedno čini i početak NDKP-a, čiji članovi čine skup vrhova N koji se prvi razmatra u raspoređivanju. DAGP u kojem se nalazi taj vrh zove se **ključni DAGP** (eng. *key DAGP*) i u njemu se nalazi NDKP koji se razmatra. Ukoliko postoji nekoliko takvih vrhova (s jednakim *urank* vrijednostima), odabire se onaj s najvećim brojem sljedbenika, ako i takvih ima više, onda se nasumično odabire jedan od njih. Taj se postupak zove **odlučujući korak** (eng. *tie-break*). Također, vrhovi koji predstavljaju izvršne jedinice programa u DAGP-ovima predstavljaju se odgovarajućim **čvorovima**, jer se njihova vremena izvršavanja mijenjaju kako postupak napreduje. Za primjer sa slike 2.15 oni su označeni kao $n_0, ..., n_4$. Nakon što je definiran NDKP iz skupa N traži se još neraspoređeni vrh grafa (jedinica programa) s najvećom vrijednosti *urank*-a i on postaje ključni vrh. Ključni vrh je potencijalno ona jedinica programa koja će se rasporediti i ući u sljedeću fazu. Međutim, prvo se provjerava ima li taj vrh još neraspoređenih prethodnika (tzv. **ključnih roditelja**, eng. *key parents*) te ako ima, jedan od njih se odabire za raspoređivanje (ako ih ima više, koristi se odlučujući korak opisan ranije). Također, svaki od ključnih roditelja rekurzivno se provjerava ima li on također neraspoređenih prethodnika, ne bi li se oni odabrali za raspoređivanje.

Za primjer sa slike 2.15 dobivene vrijednosti *urank*-a za pojedine jedinice programa, kao i odgovarajući DAGP-ovi i DKP-ovi na njima, prikazani su slikom 2.16. Na istoj slici vidljivo je kako će odabrani NDKP činiti vrhovi, tj. čvorovi koji ih predstavljaju u DAGP-ovima n_0, n_2, n_3, n_4 i to u iznosu (udaljenosti) od 28, dok će ključni DAGP biti $DAGP_0$. Na $DAGP_1$ najduži DKP iznosi 18 te se on ne razmatra kao NDKP.



Slika 2.16: DAGP za procesore p_0 i p_1 ($DAGP_0$ i $DAGP_1$) primjera sa slike 2.15a.

U fazi odabira procesora koristi se postupak umetanja (*insertion-based policy*), kao što je to slučaj u postupku HEFT i koji je opisan u pododjeljku 2.3.5. To znači da se pri svakom koraku raspoređiva-

nja, nakon što se odabere jedinica programa za raspoređivanje, provjeravaju trenutni rasporedi na svim obradbenim jedinicama sklopovlja i traži se vremenski interval u koji je moguće staviti ključni vrh v_i na način da mu t_i^{svz} bude minimalan, uzimajući pritom u obzir vremenski slijed, tj. njegovo najranije vrijeme početka t_i^{kvp} koje je definirano kao:

$$t_i^{kvp} = \max_{v_j \in pred(v_i)} (t_j^{svz}), \tag{2.17}$$

gdje je $pred(v_j)$ skup njegovih prethodnika).

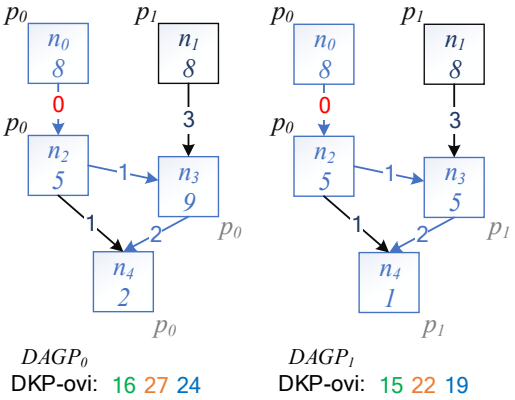
Sljedeća je faza ažuriranje statusa. Ona služi kako bi sustav osvježio informacije nakon svakoga koraka raspoređivanja, (nakon raspoređivanja pojedine jedinice programa, zadatka). Ažuriranje obuhvaća sljedeće:

- vrijeme izvršavanja prethodno raspoređene jedinice programa više nije nepoznato s obzirom na to da je određena jedinica sklopovlja na koju je raspoređena. To uvjetuje da će u sljedećim koracima raspoređivanja njezino vrijeme izvršavanja biti poznato i jednako u svim DAGP-ovima.
- Vrijednosti komunikacije između posljednje raspoređene jedinice programa i njezinih prethodnika koji su raspoređeni na istu obradbenu jedinicu postavljaju se na 0 na svim DAGP-ovima. Prema pretpostavkama, propusnost komunikacije unutar iste obradbene jedinice mnogo je veća od svih ostalih, (teoretski je beskonačna).
- Dodaje se brid grafa s iznosom 0 između posljednje raspoređene jedinice programa i jedinica programa raspoređenih prije i poslije nje na istoj obradbenoj jedinici. To se također radi za sve DAGP-ove. Navedena pretpostavka koristi se i za većinu ostalih postupaka raspoređivanja listama rangiranih jedinica programa (HEFT, CPOP, PATC, ali i našeg unaprijeđenog postupka RaND-KiP).
- Ukoliko u polaznom DAG-u programa postoji luk (vremenska ovisnost) između jedinica programa raspoređenih prije i poslije zadnje raspoređene jedinice na istoj obradbenoj jedinici, on se uklanja.
- Da bi postupak LDCP, prema riječima autora u [37], omogućio uključivanje novih jedinica programa kako napreduje postupak raspoređivanja, potrebno je dodati privremeni brid bez komunikacije (s iznosom nula) između posljednjeg raspoređenog zadatka i svih zadataka u stanju pripravnosti, tj. onih kojima su izvršeni svi prethodnici, ukoliko već između njih ne postoji komunikacija. Ti se bridovi stavljaju samo u DAGP obradbene jedinice na koju je raspoređen taj posljednji zadatak. Prije toga koraka potrebno je ukloniti sve takve bridove ukoliko ih ima, (privremeni su).
- Na kraju, ažuriraju se vrijednosti *urank*-a svih jedinica programa na svim DAGP-ovima. Ovaj korak uzrokuje problem koji ćemo opisati u tekstu koji slijedi te ga nismo razmatrali.

Sve tri faze postupka LDCP ponavljaju se dok god postoje jedinice programa koje nisu raspoređene.

Za primjer sa slike 2.15, nakon raspoređivanja jedinica programa predstavljenih čvorovima na odgovarajuće obradbene jedinice ($p_0 \leftarrow n_0$, $p_1 \leftarrow n_1$ i $p_0 \leftarrow n_2$) situacija je kao na slici 2.17. Iznos komunikacije između čvorova n_0 i n_2 postavljen je na nulu, jer su oba raspoređena na istu obradbenu jedinicu (p_0). Također, u idućem koraku raspoređivanja opet se razmatra NDKP koji čine čvorovi n_0, n_2, n_3, n_4 u iznosu

od 27, ali se traži njegov član s najvećom vrijednosti *urank*-a koji nije raspoređen, a to je n_3. S obzirom na to da su svi njegovi prethodnici već raspoređeni (n_1 i n_2) on postaje ključni vrh i raspoređuje se u trenutnom koraku.



Slika 2.17: $DAGP_0$ i $DAGP_1$ nakon raspoređivanja n_0, n_1 i n_2.

Složenost postupka LDCP u najgorem slučaju, prema [37] je $O(\vartheta^3 \cdot \rho)$, kao i kod postupka HEFT.

Postupak LDCP ima nekoliko nedostataka. Prvo, problem kod faze ažuriranja je njezin pretposljednji korak, a to je ubacivanje privremenih bridova grafa. Navedeni korak može uzrokovati stvaranje ciklusa u grafu i time onemogućiti pravilan rad postupka raspoređivanja, ali i negiranje DAG-a kao ulazne strukture programa (DAG ne sadrži cikluse). Iz tog razloga, implementacija postupka LDCP u našoj eksperimentalnoj analizi u poglavlju 5 ne sadrži navedeni korak.

Također, postupak nije u mogućnosti rasporediti nepovezane grafove (kakve primjerice daje alat TGFF, opisan u pododjeljku 5.1.1). Zbog toga smo u eksperimentalnoj analizi uveli postupak dodavanja virtualnog izlaznog čvora, što je opisano u pododjeljku 5.1.4. Isti korak spominju i drugi izvori, primjerice [38, 41].

Od nedostataka u performansama, problem postupka LDCP jest taj što ne raspoređuje komunikacijske pozive, čime dobiveni raspored gubi na primjenjivosti, posebice kod programa s visokim vrijednostima CCR-a (omjera prosječnog vremena komunikacije i obrade). Navedeni nedostatak ispravljamo pomoću inačica unaprijeđenog postupka RaNDKiP (doprinos D.3.) raspoređivanjem komunikacije, što uvelike pridonosi primjenjivosti rasporeda. To pokazuje naša eksperimentalna analiza u poglavlju 5.

Postoji još jedan nedostatak koji uspješno rješavamo našim unaprijeđenim postupcima RaNDKiP, a to je izostanak dinamičkog tretiranja komunikacije prilikom faze rangiranja jedinica programa i prilikom faze ažuriranja stanja. Naime postupak LDCP ne omogućuje analizu komunikacijskih veza među obradbenim jedinicama sklopovlja, što će biti detaljno opisano u pododjeljku 4.2 prilikom opisa inačica postupaka RaNDKiP.

Sada smo postavili temelje na kojima ćemo opisati naše izvorne znanstvene doprinose, počevši od programskog okvira LOSECO, preko unaprijeđenih postupaka raspoređivanja, pa sve do njihovog vrednovanja eksperimentalnom analizom. Ovdje predstavljeni postupci, modeli i nedostatci istih poslužit će kao okosnica oko koje ćemo opisati naredna poglavlja.

Poglavlje 3

Programski okvir LOSECO

Prvi naš doprinos D.1. temelji se na koherentnom prikazu općenitog sustava za dodjeljivanje programa na računalno sklopovlje. Kao što je spomenuto u prethodnim poglavljima, dodjeljivanje izvršnih jedinica programa (eng. *software executable unit*, SEU)[1] obradbenim jedinicama raznorodnih računalnih platformi je vrlo široka i česta tema u postojećim istraživanjima. Samim time, razvijeni su mnogi postupci mapiranja, dodjeljivanja i raspoređivanja. U najvećem broju slučajeva postoji jedan kriterij optimizacije za navedene postupke i to su većinom performanse (tj. ukupno vrijeme izvršavanja ili vrijeme odziva). Ostali primjeri kriterija optimizacije su potrošnja energije, korišteni memorijski prostor, vrijeme komunikacije, omjer obrade i komunikacije i sl. Osim postupaka s jednim kriterijem optimizacije, sve su popularniji postupci koji koriste dva ili više, poput [34, 44] i [17]. Ali podrška za više od dva kriterija još je u početnim fazama, tako da se može reći kako postojeći postupci još uvijek nemaju dovoljnu podršku za proizvoljan broj kriterija i njihovog naglašavanja, unatoč tome što za takav pristup postoji potreba u industriji, prema [99].

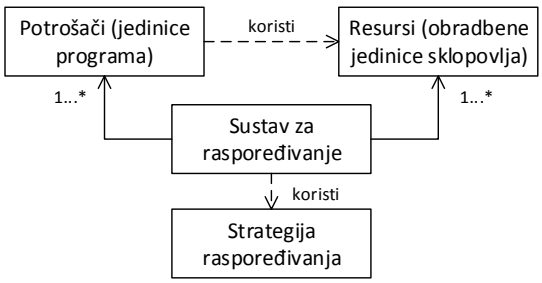
Nadalje, postojeći postupci raspoređivanja većinom su specifične primjene i koriste vrlo specifične modele za opis glavnih entiteta raspoređivanja — sklopovlja i programa. Navedeno ukazuje na potrebu povećanja prilagodljivosti postojećih postupaka raspoređivanja i to povećanjem razine apstraktnosti njegovih glavnih entiteta. Nove tehnologije sklopovlja, nove paradigme programa, nedostatak korisničke kontrole nad važnosti pojedinih kriterija raspoređivanja (pokazatelja kvalitete), podrška za veći broj pokazatelja kvalitete i odgovarajući modeli uvjetuju nove pristupe problematici. Kako bismo dali odgovor na to pitanje, razvili smo programski okvir za raspoređivanje paralelnih programa raznorodnim računalnim platformama (eng. *allocation of parallel software to hetergoeneous computing platform*, LOSECO), prema [100]. Programski okvir LOSECO omogućuje općenit opis okruženja koje gotovo optimalno raspoređuje paralelne programe raznorodnim računalnim platformama.

Programski okvir LOSECO služi metodičkom postupku prevođenja korisničkih zahtjeva nad programom u različite kriterije optimizacije, kako bi se odgovarajućim postupkom raspoređivanja pronašao gotovo optimalan raspored izvršnih jedinica programa na raznorodnoj računalnoj platformi. LOSECO se temelji na sljedećim međusobno povezanim modelima:

[1] Zajednički naziv koji koristimo za zadatke (eng. *tasks*) i poslove (eng. *jobs*).

- **modelu programa u obliku miješanoga acikličkoga grafa** koji se sastoji od raznorodnih izvršnih jedinica
- **modelu raznorodne računalne platforme u obliku potpuno povezanoga neusmjerenoga grafa** koji se sastoji od raznorodnih obradbenih jedinica
- i **korisnički definiranom modelu raspoređivanja** koji omogućuje raspoređivanje po više kriterija i eksplicitnu kontrolu nad prioritetima istih.

Općenit oblik raspoređivanja programa na sklopovlje zasniva se na problemu sa slike 3.1, prema [68], u kojem se pretpostavlja skup resursa (obradbenih jedinica sklopovlja) i skup potrošača usluženih tim resursima (izvršne jedinice programa) slijedeći određenu strategiju. Ista je slika ujedno i jezgra programskog okvira LOSECO. Ona se sastoji od sustava za raspoređivanje kao središnjeg entiteta koji koristi strategiju raspoređivanja (skup pravila i ograničenja) kako bi omogućio pristup potrošača (izvršnih jedinica programa) resursima (obradbene jedinice sklopovlja). Množnosti glavnih entiteta (resursa i potrošača) su od minimalno jednog do proizvoljnog broja i označavaju da treba postojati barem jedan resurs i barem jedan potrošač. LOSECO programski okvir koristi se za izradu *design-time* statičkog sustava za raspoređivanje. U konkretnom slučaju *design-time* označava da se postupak raspoređivanja odvija prije samog izvršavanja programa, ali samo dio koji se odnosi na njegovu paralelizaciju (ne utječe se na funkcionalnost programa). Također ovdje "statički" označava da se radi o postupku raspoređivanja koji se ne prilagođava novonastalim situacijama pri izvođenju programa, već ih predviđa unaprijed.

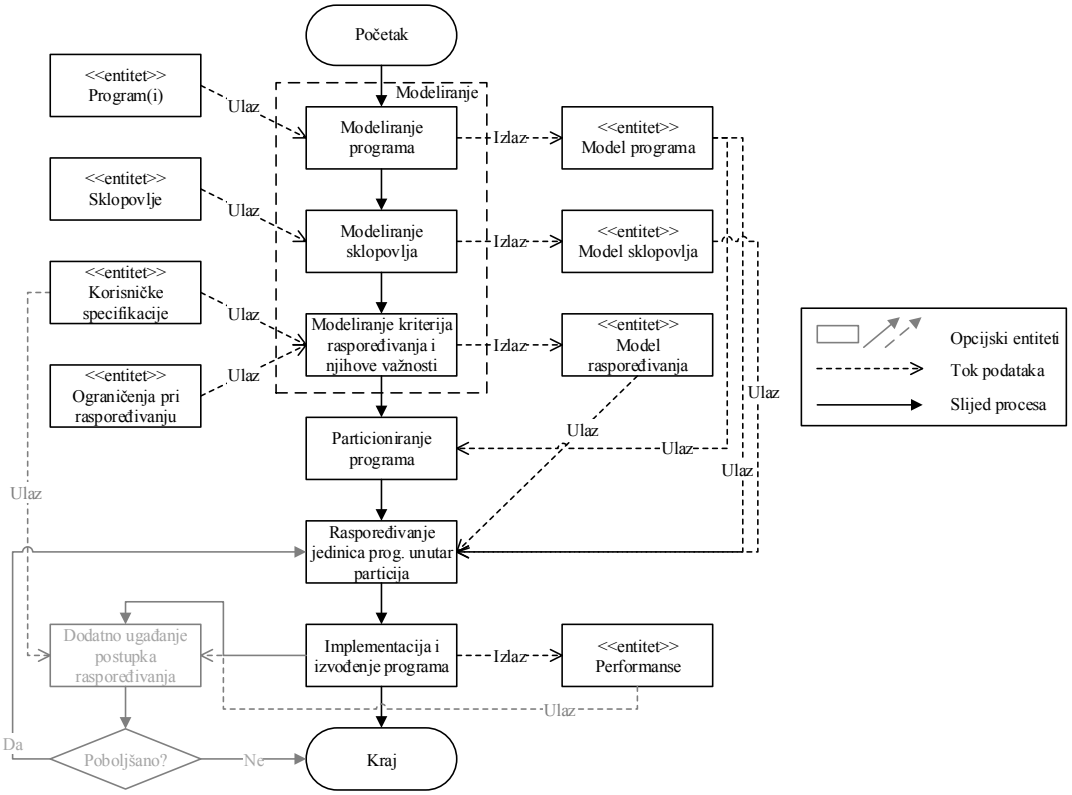


Slika 3.1: Osnovna struktura programskog okvira LOSECO.

Navedeni okvir omogućuje specifikaciju pokazatelja kvalitete ili nefunkcionalnih svojstava (eng. *extra-functional property*, EFP) pri izvođenju programa i njihove važnosti koje zadaje korisnik programa u obliku modela raspoređivanja. Omogućuje i specifikaciju druga dva glavna entiteta raspoređivanja (ulaznih modela), a to su model izvršavanja programa i model sklopovlja. Na osnovu spomenutih specifikacija programski okvir LOSECO:

- (i) analizira model izvršavanja programa i particionira ga u vremenski neovisne faze izvršavanja, smanjujući prostor mogućih rješenja i trajanje raspoređivanja (faza 1 unaprijeđenog postupka RaPPaMaG opisanog u pododjeljku 4.1),
- (ii) omogućuje raspoređivanje izvršnih jedinica programa unutar faza izvršavanja na osnovu njihovih parametara te njihove prostorne ovisnosti (komunikacija), faza 2 unaprijeđenog postupka RaPPaMaG i

- (iii) omogućuje vrednovanje dobivenih rješenja čiji se rezultati mogu iskoristiti za unaprjeđivanje postupka raspoređivanja.



Slika 3.2: Detaljna struktura programskog okvira LOSECO.

Slika 3.2 detaljno prikazuje strukturu programskog okvira LOSECO od postupka modeliranja glavnih entiteta raspoređivanja, pa sve do izvršavanja raspoređenog programa. Detaljna struktura okvira LOSECO sastoji se od tri glavna postupka:

- postupak prevođenja ulaznih entiteta u modele, koji je dijelom automatski, dijelom ručni
- postupak particioniranja i raspoređivanja
- opcijski postupak koji koristi rezultate raspoređivanja i modele predviđanja za poboljšanje cjelokupnog postupka.

Prema slici 3.2 postupak prevođenja ulaznih entiteta koristi program(e), sklopovlje te specifikacije i ograničenja postupka raspoređivanja za izradu glavnih ulaznih modela: modela programa, modela sklopovlja i modela raspoređivanja. Navedeni ulazni modeli sadrže entitete kojima se raspolaže u postupku particioniranja i raspoređivanja. Zatim, model raspoređivanja formira skup pravila, kriterija i ograničenja nad postupkom raspoređivanja, poput praga potrošnje energije, dozvoljenog vremena izvršavanja programa itd. Nakon modeliranja, program se na osnovu modela programa i modela raspoređivanja dijeli na vremenski neovisne faze izvršavanja u postupku particioniranja. Dobivene particije se tada raspoređuju na obradbene jedinice sklopovlja prema modelu raspoređivanja, nakon čega slijedi njihovo izvođenje. Nakon izvođenja, dobiveni parametri (parametri kvalitete) koriste se u postupku dodatnog ugađanja

postupka raspoređivanja, kako bi se unaprijedio cjelokupni postupak za postojeće ili buduće programe. U narednim pododjeljcima detaljno opisujemo tri glavna modela programskog okvira LOSECO.

3.1 Model izvršavanja programa

Modelom programa smatramo specifikaciju postojećeg programa pogodnu za raspoređivanje. Model programa sastoji se od izvršnih jedinica koje se mogu izvršavati serijski ili paralelno (na više obradbenih jedinica istovremeno), s poznatim vremenskim slijedom izvođenja i koje mogu izmjenjivati podatke (komunicirati). Kako bismo detaljnije opisali model programa, uvodimo definicije nekih osnovnih pojmova pomoću kojih ćemo definirati i sam model. Kao što je spomenuto ranije, općenita definicija modela programa u smislu ulaznog entiteta za raspoređivanje jest:

Definicija 3.1 *Model izvršavanja programa jest arhitektura njegovih izvršnih jedinica te njihove prostorne i vremenske međuovisnosti, tj. slijeda izvršavanja i komunikacije među njima.*

Nadalje, definiramo osnovnu jedinicu programa:

Definicija 3.2 *Izvršna jedinica programa (zadatak, posao) jest najmanji monolitni neprekidni dio programa u izvođenju koja se može rasporediti na jednu raznorodnu obradbenu jedinicu sklopovlja.*

Veličina izvršne jedinice programa ovisi o strukturi samog programa, njihovoj komunikaciji, podatkovnoj ovisnosti i prosjeku obradbene moći jedinica sklopovlja na koje će se raspoređivati. Različiti izvori navode različite razine apstraktnosti izvršnih jedinica programa. Primjerice poslovi u [17,21] i [53], komponente u [54,55,66] i [56], procesi u [30,58] i [31], računalne niti (dretve) u [28,58] itd., što je detaljnije opisano u pododjeljku 2.1.2.

Za strukturu modela programa koristimo dva najčešće korištena, gotovo standardizirana, modela programa u obliku grafa: usmjereni aciklički graf (DAG), opisan u pododjeljku 2.2.1 i graf interakcije zadataka (TIG) opisan u pododjeljku 2.2.1. U modelu DAG-a usmjereni bridovi (lukovi) grafa predstavljaju vremenski slijed izvršnih jedinica programa i/ili asinkronu komunikaciju među njima, od prethodnika prema sljedbeniku. U modelu TIG-a neusmjereni bridovi grafa predstavljaju komunikaciju među izvršnim jedinicama programa.

Kako bi naša definicija modela izvršavanja programa sadržavala prednosti oba navedena modela, kao i podršku za sinkronu komunikaciju među izvršnim jedinicama, definiramo **model izvršavanja programa u obliku miješanoga acikličkoga grafa**. Miješani aciklički graf je, prema [101, 102], formalni opis u obliku grafa koji sadržava i usmjerene i neusmjerene bridove koji povezuju vrhove grafa te ne sadrži cikluse (petlje). Prema istim izvorima u miješanom acikličkom grafu postojanje neusmjerenoga brida između para vrhova negira postojanje brida između istih. Stoga model programa u obliku takvoga grafa definiramo na sljedeći način:

Definicija 3.3 *Model izvršavanja programa u obliku miješanoga acikličkoga grafa jest uređena trojka $G = (V, E, A)$, gdje je V skup vrhova grafa $V = \{v_i : \ i = 0, ..., \vartheta - 1\}$ koje predstavljaju izvršne jedinice programa, E je skup neusmjerenih bridova koji predstavljaju komunikaciju među njima i A je skup usmjerenih bridova (lukova) koji predstavljaju njihov vremenski slijed.*

Svaka izvršna jedinica programa $v_i \in V$ opisana je vektorom parametara $\mathbf{x}_i$, gdje je $\mathbf{x} \in \mathbb{R}^\chi$, s tim da χ označava broj nezavisnih parametara x^i koji opisuju izvršne jedinice programa. Parametri iz skupa x^i, primjerice, mogu biti obradbeni zahtjevi (x^{ob}), vrijeme postavljanja, ali mogu predstavljati i informacije poput vremena izvršavanja u najgorem slučaju (eng. *worst-case execution time*, WCET), nužna vremena završetka, prioritete i sl.

Skup neusmjerenih bridova

$$E \subseteq \{(v_i, v_j) : v_i, v_j \in V,\ i \neq j\} \tag{3.1}$$

predstavlja skup komunikacijskih veza između parova izvršnih jedinica programa. Svakom neusmjerenom bridu $e_{i,j} \in E$ je dodijeljena pozitivna težina $w(e_{i,j}) > 0$ (ili ekvivalentno, $w(v_i, v_j) > 0$, dok pišemo samo $e_{i,j} > 0$) koja predstavlja obujam komunikacije među izvršnim jedinicama programa. Komunikaciju predstavljamo neusmjerenim bridovima jer smjer komunikacije ne uzimamo u obzir.

Naposljetku, skup usmjerenih bridova

$$A \subseteq \{\langle v_i, v_j \rangle : v_i, v_j \in V,\ i \neq j\} \tag{3.2}$$

predstavlja vremenski slijed izvođenja izvršnih jedinica programa, pokazujući od prethodnika prema sljedbeniku. Svakom je usmjerenom bridu dodijeljena nul-težina $w(a_{i,j}) = 0$ (ili ekvivalentno, $w(v_i, v_j) = 0$, dok pišemo samo $a_{i,j} = 0$) ako izvršavanje jedinice programa v_j ovisi o završetku izvođenja jedinice programa v_i. Ako ne postoji veza među izvršnim jedinicama programa (nema bridova ni lukova) između vrhova v_i i v_j, tada postavljamo $w(v_i, v_j) = -1$. Vrijednosti dodijeljene svim parovima vrhova grafa čine **matricu susjedstva** — oblik implementacije modela programa (eng. *adjacency matrix*). Matrica susjedstva koristi se kao ulaz za postupak raspoređivanja.

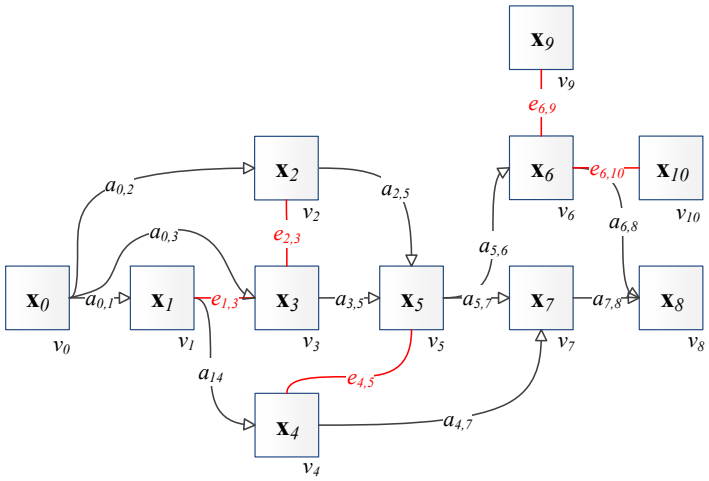
Iz [101, 102] te (3.1) i (3.2) slijedi:

$$\nexists(a_{i,i} \in A),\ \nexists(e_{i,i} \in E), \qquad \forall v_i \in V \tag{3.3}$$

$$\langle v_i, v_j \rangle \in A \Rightarrow \{v_i, v_j\} \notin E, \quad \forall v_i, v_j \in V. \tag{3.4}$$

Izraz (3.3) označava da izvršna jedinica programa ne može sama sebi biti prethodnik ili sljedbenik te ne može komunicirati sama sa sobom. U modelu to znači da nema ciklusa, što proizlazi i iz same definicije usmjerenoga acikličkoga grafa iz [101, 102]. Drugi izraz, (3.4), označava da postojanje vremenskog slijeda između para izvršnih jedinica isključuje postojanje komunikacije među njima (opisuje se sinkrona komunikacija). Na model se to odražava na način da postojanje brida između para vrhova isključuje postojanje luka između njih, kao što proizlazi i iz same definicije miješanoga grafa kao oblika formalnog opisa, prema [101, 102]. Navedena pravila važna su i za prijedlog unaprijeđenog postupka raspoređivanja particioniranjem miješanoga grafa (postupak RaPPaMaG), koji će biti opisan u pododjeljku 4.1.

Na slici 3.3 prikazan je primjer modela izvršavanja programa u obliku miješanoga acikličkoga grafa. Kvadrati označavaju vrhove grafova koji predstavljaju izvršne jedinice programa opisane vektorom parametara $\mathbf{x}_i$, spojene neusmjerenim bridovima $e_{i,j}$ koji predstavljaju komunikaciju i lukovima $a_{i,j}$ koji predstavljaju njihov vremenski slijed. Matrica susjedstva modela sa slike 3.3 predstavljena je na slici 3.4.



Slika 3.3: Primjer modela izvršavanja programa u obliku miješanoga acikličkoga grafa.

v	0	1	2	3	4	5	6	7	8	9	10
0	-1	0	0	0	-1	-1	-1	-1	-1	-1	-1
1	-1	-1	-1	$e_{1,3}$	0	-1	-1	-1	-1	-1	-1
2	-1	-1	-1	$e_{2,3}$	-1	0	-1	-1	-1	-1	-1
3	-1	$e_{1,3}$	$e_{2,3}$	-1	-1	0	-1	-1	-1	-1	-1
4	-1	-1	-1	-1	-1	$e_{4,5}$	-1	0	-1	-1	-1
5	-1	-1	-1	-1	$e_{4,5}$	-1	0	0	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	-1	0	$e_{6,9}$	$e_{6,10}$
7	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1
8	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
9	-1	-1	-1	-1	-1	-1	$e_{6,9}$	-1	-1	-1	-1
10	-1	-1	-1	-1	-1	-1	$e_{6,10}$	-1	-1	-1	-1

Slika 3.4: Matrica susjedstva modela sa slike 3.3.

Model programa, kao entiteta za raspoređivanje u programskom okviru LOSECO-a, je struktura koju on predaje postupku raspoređivanja. Drugi entitet, koji je za to potreban, model je sklopovlja i njega opisujemo u sljedećem pododjeljku.

3.2 Model sklopovlja

Kao što je već spomenuto, model sklopovlja koji se predaje postupku raspoređivanja treba sadržavati podatke o obradbenoj sposobnosti njegovih obradbenih jedinica i podatke o komunikacijskim vezama među njima. Od ostalih parametara često su to podatci o upravljanju energijom (DVFS i stanja mirovanja), tipovi obradbene jedinice (procesorska jezgra, procesor toka GPU-a i sl.), dostupnost obradbene jedinice, ostala vremenska obilježja (vrijeme za promjenu konteksta i rekonfiguracijska kašnjenja) i dr.

Postoje mnogi primjeri modela raznorodnih računalnih platformi kao ulaznih modela za postupke raspoređivanja izvršnih jedinica programa na njih. Primjerice, raznorodni čvorovi [15, 17, 21, 42, 44, 51, 67], nizovi konfiguracija [42, 44], matrice resursa [11, 16] i konstitutivne komponente [14, 53, 54]. Postoje općenitiji modeli koji se mogu primijeniti na različite računalne platforme, poput grafa resursa [10, 31,

36, 62] i modela temeljenih na opisnim jezicima UML/MARTE (eng. *modeling and analysis of real-time and embedded systems*) neovisnih o platformi poput [29]. Detalji o modeliranju sklopovlja HPC-a dani su u pododjeljku 2.2.2.

Postojanje velikog broja različitih modela sklopovlja u upotrebi dokazuje da nije trivijalno obuhvatiti sve bitne parametre raznorodnih računalnih platformi. Zbog toga u ovom pododjeljku opisujemo vlastiti model sklopovlja u obliku **potpuno povezanoga neusmjerenoga grafa**, koji pokriva osnovna obilježja velikog broja postojećih računalnih platformi potrebnih za postupke raspoređivanja programa na njih.

Da bismo opisali navedeni model, uvodimo nekoliko definicija osnovnih koncepata potrebnih za formiranje konačne definicije modela sklopovlja. Kao što smo spomenuli ranije, pod raznorodnom računalnom platformom smatramo računalni sustav koji se sastoji od raznorodnih obradbenih jedinica ili sadrži raznorodne komunikacijske veze među njima. Također, navedena platforma može sadržavati i oboje — raznorodne obradbene jedinice i raznorodne komunikacijske veze među njima. Iz navedenog formiramo definiciju:

Definicija 3.4 *Raznorodna računalna platforma je računalni sustav kojemu su barem dvije obradbene jedinice različite na način da imaju različit utjecaj na izvođenje iste izvršne jedinice programa na njima i/ili se u tom sustavu nalaze barem dva različita komunikacijska kanala.*

Na način sličan opisu modela izvršavanja programa u prethodnom pododjeljku, definiramo općenit pojam modela raznorodne računalne platforme kao ulaznog entiteta u postupak raspoređivanja:

Definicija 3.5 *Model raznorodne računalne platforme je organizacija i parametrizacija raznorodnih obradbenih jedinica u nekom računalnom sustavu.*

Sam pojam raznorodne obradbene jedinice sklopovlja, poput procesorske jezgre, procesora toka GPU-a ili mikrokontrolera je definiran kao najmanja jedinica sklopovlja:

Definicija 3.6 *Raznorodna obradbena jedinica (eng. heterogeneous processing unit, HPU) jest najmanja jedinica sklopovlja koja može nezavisno izvršiti jednu izvršnu jedinicu programa.*

S obzirom na to da je model izvršavanja programa sastavljen od njegovih izvršnih jedinica te njihove prostorne i vremenske ovisnosti u obliku miješanoga acikličkoga grafa, koristimo sličnu strukturu i za model sklopovlja (formalni opis grafa) kako bi njihovo međusobno preslikavanje bilo jednostavnije:

Definicija 3.7 *Model raznorodne računalne platforme u obliku potpuno povezanoga neusmjerenoga grafa jest uređeni par $G = (P, C)$, gdje je P skup svih vrhova grafa $P = \{p_i : i = 0, ..., \rho - 1\}$ koji predstavljaju obradbene jedinice sklopovlja, dok je $C = \{(p_i, p_j) : p_i, p_j \in P\}$ skup komunikacijskih veza među njima.*

Svaki vrh grafa $p_i \in P$ predstavlja jednu obradbenu jedinicu sklopovlja opisanu vektorom $\mathbf{y}_i$, gdje je $\mathbf{y} \in \mathbb{R}^\omega$, s tim da ω označava broj nezavisnih parametara y^i koji opisuju jedinice sklopovlja. Svakom bridu grafa dodijeljena je pozitivna težina $w(c_{i,j}) > 0$ (ili ekvivalentno $w(p_i, p_j) > 0$, dok pišemo $c_{i,j} > 0$) koja predstavlja podatkovnu propusnost između obradbenih jedinica p_i i p_j koje povezuje. Podatkovna propusnost unutar iste obradbene jedinice većinom se smatra mnogo većom od propusnosti među različitim obradbenim jedinicama, prema [13, 34, 38–41]. No navedene pretpostavke namijenjene

su programima sa asinkronom komunikacijom, tj. gdje jedna izvršna jedinica nakon završetka izvođenja šalje podatke drugoj prije nego ona počinje s radom. Stoga, u slučaju programa u obliku miješanoga grafa i podrazumijevajući definicije 3.2 i 3.6, propusnost komunikacijskoga kanala unutar iste obradbene jedinice postavlja se na 0:

$$c_{i,j} = 0,\ \forall c_{i,j} \in \{(v_i, v_j) : v_i, v_j \in V,\ i = j\}. \tag{3.5}$$

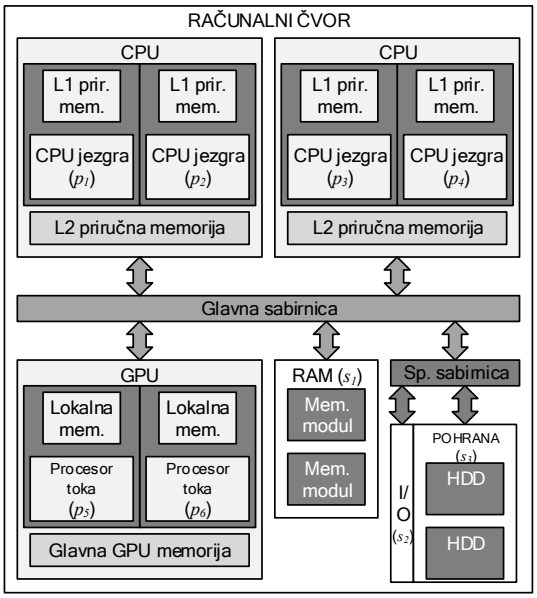
Takav način označavanja komunikacije pogodan je za postupke raspoređivanja temeljene na rangiranju jedinica programa. No, s obzirom na to da nam je s jedne strane cilj opisati programe koji sadrže sinkronu komunikaciju, gdje svaka jedinica programa u grupi koja međusobno komunicira mora biti smještena na odvojenu obradbenu jedinicu sklopovlja (što zahtijevaju izvršni dijelovi programa poput procesa u MPI okruženju), podrazumijevamo

$$c_{i,j} = \infty,\ \forall c_{i,j} \in \{(v_i, v_j) : v_i, v_j \in V,\ i = j\}. \tag{3.6}$$

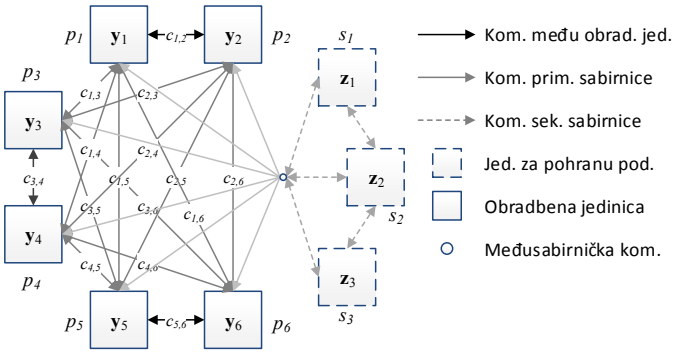
Tako je izbjegnuto istovremeno smještanje dviju ili više izvršnih jedinica programa koje međusobno komuniciraju na istu obradbenu jedinicu sklopovlja, na taj način osiguravajući njihovu sinkronu komunikaciju.

Slika 3.5 predstavlja primjer strukture jednog računalnog sustava (čvora) dobivenog alatom HWLOC, opisanom u pododjeljku 2.2.2. Navedeni čvor sastoji se od dva različita obradbena elementa — CPU-a i GPU-a. Svaki od dva CPU-a sastoji se od dvije identične procesorske jezgre koje sadržavaju vlastitu L1 predmemoriju i L2 predmemoriju, koju dijele na razini obradbenog elementa. GPU je prikazan na sličan način i sadrži dva procesora toka od kojih svaki ima vlastitu memoriju i zajedničku memoriju na razini obradbenog elementa. Glavna sabirnica je medij preko kojega međusobno komuniciraju obradbeni elementi, ali i preko kojega oni komuniciraju s glavom memorijom i sporednom sabirnicom. Sekundarna sabirnica niže je propusnosti od glavne i ona u konkretnom slučaju povezuje glavnu sabirnicu s ostalim uređajima, poput uređaja za pohranu podataka, I/O uređaja i mreže. Glavna se sabirnica računalnih sustava često naziva *Northbridge*, a sporedna *Southbridge*, kao u [15].

Na slici 3.6 prikazan je model računalnog čvora sa slike 3.5. Model je strukturiran prema definiciji 3.7 kao potpuno povezani neusmjereni graf vrhova p_i, koji predstavljaju obradbene jedinice opisane vektorom parametara $\mathbf{y}_i$. Bridovi grafa predstavljaju komunikacijske veze među obradbenim jedinicama. Brid $c_{i,j}$, primjerice, predstavlja komunikacijsku vezu među obradbenim jedinicama p_i i p_j. Poveznica između glavne i sporedne sabirnice prikazana je komunikacijskom točkom kako bi se zadržala čitkost na slici. Različita propusnost komunikacijskih kanala prikazana je različitim bojama. Nadalje, prikazani čvorovi grafa za jedinice pohrane podataka, poput glavne memorije i čvrstih diskova, predstavljaju opcijske komponente koje se mogu upotrijebiti za šira razmatranja, kao što je to lokalitet podataka u službi kriterija optimizacije u [12, 17]. Za potrebe postupaka raspoređivanja koji će biti predstavljene kasnije te se komponente neće uzeti u obzir.



Slika 3.5: Primjer strukture računalnog čvora dobivene alatom HWLOC.



Slika 3.6: Model računalnog čvora prikazanog slikom 3.5.

Iz navedenog modela uočljivo je da se obradbene jedinice različitih obradbenih jedinica tretiraju jednako, što znači da neki parametri neće biti prisutni, no to je nužno kako bi se zadržala razina apstraktnosti dovoljna za primjenu novih i postojećih postupaka raspoređivanja. Ovakav oblik apstraktnosti obradbenih elemenata neće dovesti do značajnih odstupanja u predviđenom ponašanju sustava zasnovanog na po jednom tipu obradbenih elemenata. Razina apstraktnosti u obliku obradbene jedinice čini valjan nadomjestak za različite tipove sklopovskih entiteta poput procesorskih jezgri, obradbenih jedinica ugrađenih računalnih sustava, procesora toka GPU-a i mnogih drugih.

Komunikacijska matrica, (matrica povezanosti) je matrica susjedstva čvorova grafa koji predstavlja model sklopovlja. Za primjer računalnog čvora sa slike 3.5 komunikacijska matrica prikazana je na slici 3.7. Konkretno, ako se navedena matrica interpretira za postupke raspoređivanja zasnovane na rangiranim listama jedinica programa (HEFT, LDCP, RaNDKiP), koji podrazumijevaju asinkronu komunikaciju među jedinicama programa, onda je ona prikazana slikom 3.7a.

p	1	2	3	4	5	6
1	0	$c_{1,2}$	$c_{1,3}$	$c_{1,4}$	$c_{1,5}$	$c_{1,6}$
2	$c_{1,2}$	0	$c_{2,3}$	$c_{2,4}$	$c_{2,5}$	$c_{2,6}$
3	$c_{1,3}$	$c_{2,3}$	0	$c_{3,4}$	$c_{3,5}$	$c_{3,6}$
4	$c_{1,4}$	$c_{2,4}$	$c_{3,4}$	0	$c_{4,5}$	$c_{4,6}$
5	$c_{1,5}$	$c_{2,5}$	$c_{3,5}$	$c_{4,5}$	0	$c_{5,6}$
6	$c_{1,6}$	$c_{2,6}$	$c_{3,6}$	$c_{4,6}$	$c_{5,6}$	0

(a) Asinkrona komunikacija.

p	1	2	3	4	5	6
1	∞	$c_{1,2}$	$c_{1,3}$	$c_{1,4}$	$c_{1,5}$	$c_{1,6}$
2	$c_{1,2}$	∞	$c_{2,3}$	$c_{2,4}$	$c_{2,5}$	$c_{2,6}$
3	$c_{1,3}$	$c_{2,3}$	∞	$c_{3,4}$	$c_{3,5}$	$c_{3,6}$
4	$c_{1,4}$	$c_{2,4}$	$c_{3,4}$	∞	$c_{4,5}$	$c_{4,6}$
5	$c_{1,5}$	$c_{2,5}$	$c_{3,5}$	$c_{4,5}$	∞	$c_{5,6}$
6	$c_{1,6}$	$c_{2,6}$	$c_{3,6}$	$c_{4,6}$	$c_{5,6}$	∞

(b) Sinkrona komunikacija.

Slika 3.7: Matrica komunikacije računalnog čvora sa slike 3.5 opisanog modelom na slici 3.6.

U slučaju kada se ona interpretira za postupke raspoređivanja koji podrazumijevaju sinkronu komunikaciju (poput našeg unaprijeđenog postupka RaPPaMaG), onda je ona oblika kao na slici 3.7b. Elementi matrice čine komunikacijsku propusnost među svim parovima obradbenih jedinica sklopovlja. Navedena matrica simetrična je u odnosu na glavnu dijagonalu jer se pretpostavlja da jedan komunikacijski poziv zauzima čitavu vezu za sebe dok ne završi i jednak je u oba smjera (što je dovoljno za opis većine sustava za potrebe raspoređivanja).

Nakon što su definirani modeli glavna dva entiteta raspoređivanja — programa i sklopovlja, potrebno je opisati na koji način će oni biti raspoređeni. To je opisano modelom raspoređivanja koji definiramo u sljedećem pododjeljku.

3.3 Model raspoređivanja

Model raspoređivanja određuje skup pravila i ograničenja nad postupkom raspoređivanja izvršnih jedinica programa na obradbene jedinice sklopovlja. Taj skup je dijelom nametnut strukturom glavnih ulaznih entiteta i samim postupkom raspoređivanja, a dijelom od strane korisnika programa. Model raspoređivanja najčešće je skup kriterija optimizacije s ograničenjima ili bez njih, opisanih u pododjeljcima 2.3.1, 2.3.2 i 2.3.3 za pojedine skupine postupaka raspoređivanja. Većinom su to značajke (pokazatelji) kvalitete (eng. *quality attribute*, QA) koje se identificiraju iz zahtjeva nad izvođenjem programa na sklopovlju i razlikuju se ovisno o kojoj domeni računarstva je riječ, (HPC, ugrađeni računalni sustavi, WSN itd.).[2] Npr. za računarstvo visokih performansi navedene značajke kvalitete su performanse, utrošak energije, ujednačavanje opterećenja, dostupnost, lokalitet podataka i sprječavanje ispada čvorova. Od ograničenja, to su često prag zauzeća memorije, prag troškova, prag utrošene energije, dopušteno vrijeme izvršavanja programa, broj zauzetih računalnih čvorova i sl., što je i opisano u pododjeljku 2.2.3. Svaka od značajki kvalitete ovisi o parametrima koji opisuju programe i sklopovlje, što znači da će dobivene značajke kvalitete ovisiti o rasporedu programa na sklopovlju.

Jedan od nedostataka postojećih postupaka raspoređivanja jest nemogućnost naglašavanja važnosti pojedinih kriterija optimizacije (značajki kvalitete), zbog čega mi predlažemo model raspoređivanja koji uzima u obzir više značajki kvalitete te kojima se važnost određuje od strane korisnika u obliku težina

[2] Često naizmjence koristimo termine **značajke kvalitete**, **pokazatelji kvalitete**, **odzivi** i **kriteriji optimizacije** i njih smatramo ekvivalentnima

(pondera). Višestruke značajke kvalitete označavaju višestruke korisnički-naglašene kriterije optimizacije i iz tog razloga predstavljamo postupak raspoređivanja kao optimizaciju po više kriterija, sličnu navedenoj u opisu metaheurističkih postupaka raspoređivanja opisanima u 2.3.2. Ako skup kriterija (ciljeva) predstavimo skupom pomoćnih funkcija $u_1(\mathbf{q}), ..., u_k(\mathbf{q})$, gdje svaka predstavlja jednu značajku kvalitete i ako svaka od tih značajki ovisi o $\chi + \omega$ varijabli (koje opisuju programe i sklopovlje), općenit problem raspoređivanja definiran je kao:

$$\begin{aligned} \text{minimizirati} \quad & \mathbf{u}(\mathbf{q}) = (w_0 u_0(\mathbf{q}), ..., w_{k-1} u_{k-1}(\mathbf{q})) \\ \text{unutar} \quad & \mathbf{q} \in X_f, \end{aligned} \tag{3.7}$$

gdje je $\mathbf{q}$ vektor nezavisnih parametara koji opisuju program ($\mathbf{x}$) i nezavisnih parametara koji opisuju sklopovlje ($\mathbf{y}$):

$$\mathbf{q} = \mathbf{x} \cup \mathbf{y}. \tag{3.8}$$

X_f je skup izvedivih rješenja (eng. *feasible solutions*). Vrijednost pomoćne funkcije $u_i(\mathbf{q})$ (vrijednost značajke kvalitete) ovisi o rješenju, tako da je cilj smanjiti vrijednost te funkcije kako bi se dobio što je bolje mogući raspored. Težine $w_0, ..., w_{k-1}$ predstavljaju relativne važnosti pojedinih značajki kvalitete i njih zadaje korisnik programa. Također, vrijedi:

$$\sum_i w_i = 1 \tag{3.9}$$

Pri definiranju općenitog problema raspoređivanja suočavamo se s nekoliko izazova.

- Prvi je pravilna specifikacija svake pomoćne funkcije $u_i(\mathbf{q})$ kao funkcije parametara x^i i y^j koji opisuju programe i sklopovlje. Mnoge značajke kvalitete je teško formalno opisati, npr. za mogućnost održavanja (eng. *maintainability*), sigurnost (eng. *safety*) i zaštićenost (eng. *security*) je to nemoguće. No, za mnoge značajke je to moguće, npr. za ukupno vrijeme izvršavanja (eng. *total execution time*), propusnost (eng. *throughput*) i potrošnju energije (eng. *energy consumption*), samo što je taj opis često vrlo složen.
- Drugi je izazov kako pravilno odrediti važnosti pojedinih značajki $w_0, ..., w_{k-1}$ kao rezultat analize zahtjeva na programe.
- Treći izazov vezan je uz ograničenja koja se djelomično mogu izraziti pomoćnim funkcijama, ali mogu značiti i međusobnu zavisnost rješenja, (primjerice neka rješenja mogu biti izvediva ili neizvediva, poput zabrane raspoređivanja neke izvršne jedinice programa na određenu obradbenu jedinicu sklopovlja).
- Četvrti izazov tiče se trajanja samog postupka raspoređivanja. Navedeni problem često nije moguće riješiti u polinomnom vremenu, što uzrokuje probleme u slučajevima s velikim brojem entiteta raspoređivanja i parametara koji ih opisuju. Zbog toga se rješenja najčešće traže raznim heuristikama (npr. *Greedy* algoritam, raspoređivanje pomoću rangiranih listi, *work stealing*) ili metaheuristikama (genetski algoritmi, simulirano kaljenje, diferencijalna evolucija).

Međutim, zbog činjenice da je primarni cilj ovog rada omogućiti što bolje postupke raspoređivanja, samo je posljednji od navedenih izazova onaj koji ćemo detaljno rješavati kroz nastavak ovog rada.

Programski okvir LOSECO, sa skupom modela koji opisuju glavne entitete raspoređivanja, čine predložak u koji uklapamo pretpostavke što se tiču modela i postupaka raspoređivanja i njih opisujemo u narednom poglavlju. On čini i naš prvi doprinos D.1. u smislu opisa strukture općenitog sustava za dodjeljivanje i uvođenjem više kriterija raspoređivanja (značajki kvalitete) u model raspoređivanja kojima korisnik može dodijeliti različite važnosti. Programski okvir LOSECO čini i dio našeg drugog doprinosa D.2. uvođenjem novog modela programa u obliku miješanoga acikličkoga grafa kojim možemo opisati sinkronu komunikaciju među jedinicama programa i njihov vremenski slijed. LOSECO također predstavlja strukturu naših unaprijeđenih postupaka raspoređivanja RaPPaMaG i RaNDKiP, čije opise dajemo u idućem poglavlju.

Poglavlje 4

Unaprijeđeni postupci raspoređivanja

U poglavlju 2.3 spomenuti su postojeći postupci raspoređivanja programa na sklopovlje, gdje su oni razvrstani prema tipu postupka i područjima primjene. Također su navedeni i glavni nedostatci postojećih postupaka, a to jesu:

1. vrijeme utrošeno na raspoređivanje je preveliko,
2. eksplicitna podrška za raznorodne komunikacijske veze sklopovlja gotovo da i ne postoji,
3. jedini pokazatelj kvalitete većinom ukupno je vrijeme izvođenja i
4. rješenja mogu biti daleko od optimalnih.

Kako bismo unaprijedili raspoređivanje programa na sklopovlje ispravljanjem navedenih nedostataka razvili smo dva poboljšana postupka raspoređivanja, od kojih svaki obuhvaća odgovor na dio tih nedostataka. Prvi predloženi unaprijeđeni postupak raspoređivanja je ***ra*spoređivanje *p*rograma *pa*rticioniranjem *m*iješanog *a*cikličkog *g*rafa — RaPPaMaG**, koji smanjuje prostor mogućih rješenja dijeljenjem programa u obliku miješanoga acikličkoga grafa (iz programskog okvira LOSECO prema modelu iz pododjeljka 3.1) na vremenski neovisne faze izvršavanja. Postupak RaPPaMaG dio je našeg izvornog znanstvenog doprinosa D.2. i odgovara na cilj istraživanja 3 iz pododjeljka 1.2. Navedeni postupak uz uvođenje novog prikaza programa, uvođenjem manjih pojednostavljenja i smanjivanjem prostora rješenja omogućuje i upotrebu naprednijih, vremenski zahtjevnijih postupaka raspoređivanja (poput diferencijalne evolucije, optimizacije rojem čestica, iscrpnog pretraživanja i sl.), koji daju rješenja bliže optimalnima. Postupak odgovara na nedostatak 1 svođenjem problema na manje potprobleme čime se omogućuje značajno skraćivanje vremena trajanja raspoređivanja. Nedostatak 3 postupak RaPPaMaG nadoknađuje na način da svaki potproblem za sebe rješava rafiniranim metodama optimizacije s više funkcija cilja, tako da se, osim vremena izvođenja, može vršiti optimizacija i po drugim parametrima. Utjecaj posljednjeg nedostatka smanjen je pronalaskom optimalnih rješenja za pojedine potprobleme (particije), što u određenim slučajevima može rezultirati rješenjima bližim optimalnima za cijeli program. Navedeni postupak opisan je u pododjeljku 4.1. Ukratko, postupak RaPPaMaG ima sljedeća unaprijeđena svojstva:

1. omogućuje upotrebu višestrukih kriterija kvalitete (kriterija optimizacije),

2. omogućuje zadavanje različitih važnosti pojedinim kriterijima kvalitete,
3. omogućuje raspoređivanje programa koji sadrže sinkronu komunikaciju i vremenski slijed među jedinicama programa,
4. omogućuje smanjivanje prostora rješenja particioniranjem programa na manje dijelove.

Naš drugi unaprijeđeni postupak raspoređivanja jest ***r**aznorodni **n**ajduži **d**inamički **kri**tični **put*** — **RaNDKiP** (doprinos D.3.), koji se temelji na postupku raspoređivanja pomoću rangiranih listi izvršnih jedinica programa. Navedeni postupak ima sljedeća unaprijeđena svojstva:

1. rangiranje izvršnih jedinica programa odvija se principom najdužeg dinamičkog kritičnog puta, inspirirano postupkom LDCP iz [37], ali uzimanjem u obzir raznorodnosti komunikacijskih veza među obradbenim jedinicama sklopovlja,
2. sadrži podršku za raznorodnu komunikaciju sklopovlja pomoću
   (a) raspoređivanja komunikacijskih poziva među izvršnim jedinicama programa i
   (b) ažuriranjem stanja pri svakom koraku raspoređivanja, ovisno o komunikacijskoj mreži sklopovlja,
3. omogućuje skaliranje frekvencije procesora za nekritične izvršne jedinice unaprijeđenim postupkom labavljenja vremenskih isječaka — LVI (na osnovi djelomičnog optimalnog labavljenja, prema [39]), smanjujući time utrošak energije pri izvođenju programa.

Postupak RaNDKiP raspoređuje izvršne jedinice programa na sklopovlje prema poznatim vremenima izvršavanja svake od njih na svim obradbenim jedinicama, uz eksplicitno uvođenje vremena utrošenog na komunikaciju u raspored i skaliranjem frekvencije obradbenih jedinica. Složenost inicijalnog algoritma (prvog dijela) prema [37] je $O(\rho\vartheta^3)$, gdje ρ označava broj obradbenih jedinica, a ϑ broj izvršnih jedinica programa. Drugi dio postupka ne uključuje dodatnu složenost, već samo treći dio i povećava ju na ukupno $O(e\rho + \vartheta^2)$, gdje je e broj bridova grafa. Na taj način postupak RaNDKiP odgovara na nedostatak 1. Nedostatak 2 pokriva drugim dijelom algoritma, dok je treći nedostatak smanjen uvođenjem potrošnje energije kao dodatnog kriterija optimizacije. Postupak omogućuje bolja rješenja u odnosu na postojeće postupke raspoređivanja rangiranih listi HEFT (opisanog u pododjeljku 2.3.5) i LDCP (opisanog u pododjeljku 2.3.6), što će pokazati i naša eksperimentalna analiza u poglavlju 5. Na taj način smanjuje i nedostatak 4. Postupak raspoređivanja RaNDKiP opisan je u pododjeljku 4.2.

S obzirom na to da smo u prethodnom poglavlju opisali programski okvir LOSECO koji, između ostalog, opisuje model programa u obliku miješanoga grafa, prvo ćemo dati prikaz unaprijeđenog postupka RaPPaMaG koji je vezan za taj model. Stoga njegov opis dajemo u sljedećem pododjeljku.

4.1 Unaprijeđeni postupak raspoređivanja RaPPaMaG

S obzirom na to da je problem raspoređivanja programa na sklopovlje NP-težak, što zaključuju i brojni istraživači, primjerice u [10–13, 58, 96], pronalazak optimalnih ili gotovo optimalnih rješenja postupak je koji zahtijeva značajne računalne resurse i veliko vrijeme izvođenja. Za veći broj entiteta raspoređivanja pronalazak rješenja nije izvediv što uzrokuje nastanak mnogih postupaka raspoređivanja vođenim

lokalnim pretraživanjem, poput onih opisanih u pododjeljku 2.3.2. Kako bismo odgovorili na taj problem, predstavljamo poseban postupak raspoređivanja temeljen na particioniranju miješanoga grafa iz dvaju koraka — RaPPaMaG. On omogućuje inicijalno smanjivanje prostora rješenja dijeljenjem programa na manje dijelove i zatim raspoređivanje tih dijelova zasebno. Glavni koraci navedenog postupka raspoređivanja jesu:

1. **korak 1:** particioniranje grafa programa pomoću **trivijalnog reza** uz **dodavanje odgovarajućih sinkro-vrhova** (komunicirajućih vrhova bez prethodnika),
2. **korak 2:** raspoređivanje dobivenih particija metodom **višekriterijske (višeciljne) optimizacije**.

U prvom koraku postupak RaPPaMaG dijeli program u obliku miješanoga acikličkoga grafa na manje podgrafove (particije) koji se nazivaju **fazama izvršavanja**. Faze izvršavanja predstavljaju slijed izvođenja programa i svaka za sebe sadrži vremenski neovisne izvršne jedinice. Za podjelu programa na faze izvršavanja u svrhu smanjivanja prostora rješenja koriste se informacije iz modela programa o tijeku izvođenja njegovih izvršnih jedinica (vremenska ovisnost) opisane lukovima grafa. Nakon toga se metodom trivijalnog reza grafa prave rezovi svih lukova kako bi se izolirale pojedine faze izvršavanja. Zatim slijedi ubacivanje sinkro-vrhova u pojedinu fazu koji su komunikacijski vezani s nekim od ostalih vrhova iz te faze.

Drugi je korak zasebno raspoređivanje svake od faza izvršavanja. Svaka se faza može tretirati kao zaseban graf (graf interakcije zadataka — TIG), jer unutar njega nema vremenske zavisnosti među izvršnim jedinicama kao što je to slučaj u modelu usmjerenoga acikličkog grafa (DAG-a). Na svaki takav podgraf (particiju) može se primijeniti neki od postojećih postupaka raspoređivanja programa u obliku grafa (uključujući i pretraživanje rangiranjem jedinica programa uz njihove određene prilagodbe), od kojih mi primjenjujemo iscrpno pretraživanje. Razlog korištenja iscrpnog pretraživanja dvojak je; (i) iako je vremenski zahtjevan, za manje instance problema (manji broj izvršnih jedinica programa i obradbenih jedinica sklopovlja) može dati optimalna rješenja po particijama, što će pokazati i primjer u pododjeljku 4.1.3 te eksperimentalna analiza u poglavlju 5, (ii) naglasak našeg rada nije na drugoj fazi (raspoređivanju particija), već u smanjivanju prostora rješenja (particioniranju).

Opis unaprijeđenog postupka raspoređivanja RaPPaMaG započinjemo opisom njegove prve faze — particioniranja u sljedećem pododjeljku.

4.1.1 Korak 1 postupka RaPPaMaG: particioniranje miješanoga acikličkoga grafa

Postupak RaPPaMaG podrazumijeva prikaz programa u obliku miješanoga acikličkoga grafa prema modelu programa programskog okvira LOSECO opisanom u 3.1. Prvi se korak postupka RaPPaMaG (particioniranje) sastoji od dvaju dijelova:

- trivijalni rez lukova grafa i
- ubacivanje sinkro-vrhova.

Prvi dio particioniranja jest tzv. **algoritam trivijalnog reza**, kao specijalni slučaj tzv. *Cut-clustering* algoritma iz [103]. Algoritam trivijalnog reza izdvaja sve vrhove grafa koji su međusobno vezani bilo

bridom, bilo lukom (izvorno, *Cut-clustering* algoritam iz [103] odnosi se samo na neusmjerene bridove). U slučaju predloženog postupka particioniranja miješanoga acikličkoga grafa, algoritam je prilagođen na način da izdvaja samo vrhove povezane usmjerenim bridovima (lukovima), jer oni predstavljaju vremenski slijed.

Strategija koja omogućuje primjenu algoritma na lukove jest kretanje po koracima od ulaznih vrhova grafa prema izlaznima svrstavajući razmatrane vrhove u tri skupine:

1. **skupina**: vrhovi koji su već dodijeljeni nekoj od faza izvršavanja,
2. **skupina**: vrhovi koji potencijalno mogu biti smješteni u trenutno razmatranu fazu δ^{temp},
3. **skupina**: vrhovi koji se ne mogu trenutno smjestiti ni u jednu fazu, već će biti razmatrani kasnije.

Kretanje po koracima znači da se u svakom koraku analiziraju sljedbenici svih vrhova prethodno promatrane faze (iz prethodnog koraka). Na taj se način osigurava uključivanje svih vrhova koji su dio vremenskog slijeda.

Pripadajući postupak particioniranja miješanoga grafa prikazana je pomoću algoritma 1. Algoritam sprema skup ulaznih vrhova (vrhovi koji nemaju prethodnika, a imaju sljedbenike) u prvu fazu izvršavanja programa δ^{cur} koja se analizira. Nakon toga, glavni dio algoritma (linije 2–18) ponavlja se dok god postoji novih vrhova za razmatranje, drugim riječima, dok trenutna faza izvršavanja δ^{temp} nije prazan skup. Sljedeći korak spremanje je svih sljedbenika prethodno obrađene faze u fazu koja se trenutno analizira (linija 3). U njoj se za svaki par vrhova koje sadržava provjerava postoji li vremenski slijed među njima (linije 4–8). Ako takvi vrhovi postoje, sljedbenik se izbacuje iz trenutno promatrane faze (vrhovi unutar faza moraju biti vremenski neovisni). Nakon što se u fazi δ^{temp} nalaze samo međusobno vremenski neovisni neposredni nasljednici prethodne faze δ^{cur}, započinje drugi dio algoritma. U drugom dijelu algoritma (linije 9–13) traže se sinkro-vrhovi koji komuniciraju s vrhovima iz trenutno promatrane faze (linija 10) i stavljaju se u tu fazu (linija 11). Zatim se pronađeni vrhovi spremaju u upravo obrađenu fazu (linije 14–16).

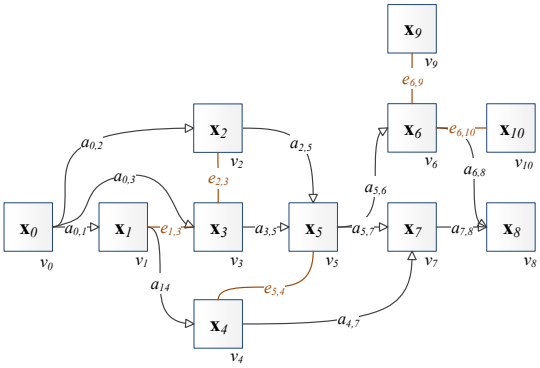
Algoritam 1 RaPPaMaG, korak 1.

1: $\delta^{cur} \leftarrow$ ulazni vrhovi
2: **ako** $\delta^{cur} \neq \emptyset$ **onda**
3: $\quad \delta^{temp} \leftarrow sljed(v_k),\ v_k \in \delta^{cur}$
4: $\quad$ **za sve** $v_i, v_j \in$ **radi**
5: $\quad\quad$ **ako** $v_i \in sljed(v_j)$ **onda**
6: $\quad\quad\quad \delta^{temp} = \delta^{temp} \setminus \{v_i\}$
7: $\quad\quad$ **završi ako**
8: $\quad$ **završi za**
9: $\quad$ **za sve** $v_i \in \delta^{temp},\ v_j \notin \delta^{temp}$ **radi**
10: $\quad\quad$ **ako** $pred(v_j) \neq \emptyset$ **i** $e(v_i, v_j) > 0$ **onda**
11: $\quad\quad\quad \delta^{temp} = \delta^{temp} + \{v_j\}$ {dodaj sinkro-vrhove u trenutnu fazu}
12: $\quad\quad$ **završi ako**
13: $\quad$ **završi za**
14: $\quad$ **ako** $\delta^{temp} \neq \emptyset$ **onda**
15: $\quad\quad$ pohrani δ^{temp}
16: $\quad$ **završi ako**
17: $\quad \delta^{cur} = \delta^{temp}$
18: **završi ako**

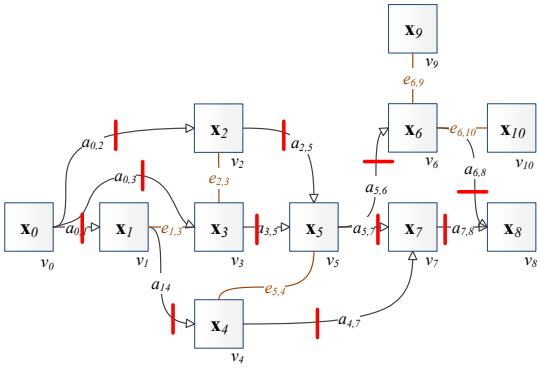
Ukoliko ih nema, znači da su svi vrhovi smješteni u pripadajuće faze izvršavanja i algoritam završava s radom.

S obzirom na to da u grafu mogu postojati vrhovi koji nemaju prethodnika iz kojih sežu neusmjereni bridovi prema drugim vrhovima (komunikacija, sinkro-vrhovi), na njih nije moguće primijeniti trivijalni rez (nisu povezani lukovima). Iz tog razloga, drugi dio prvog koraka postupka RaPPaMaG jest rekurzivno traženje takvih vrhova. Pronađeni sinkro-vrhovi smještaju se u istu fazu izvršavanja kao i vrhovi s kojima komuniciraju (ako je to moguće, tj. nema vremenske ovisnosti među njima). Razlog zbog kojeg se traže rekurzivno jest što takvi vrhovi mogu biti dalje povezani neusmjerenim bridovima s ostalim vrhovima koji nemaju drugih poveznica u grafu.

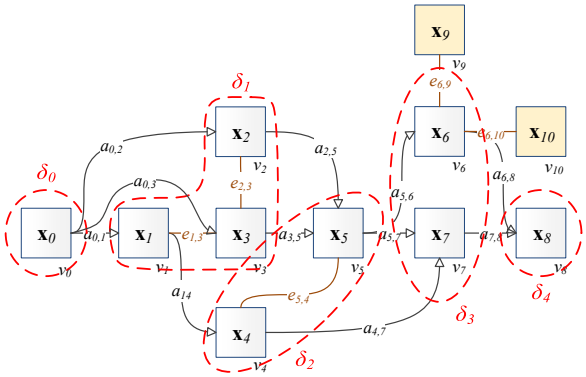
Na slici 4.1 prikazan je primjer particioniranja miješanoga acikličkoga grafa na jednostavnom modelu programa. Slika 4.1a prikazuje program u obliku miješanoga grafa koji se sastoji od 11 izvršnih jedinica $v_0, ..., v_{10}$ s pripadajućim vremenskim slijedom i komunikacijom među njima. Prvi dio particioniranja traženje je svih lukova grafa $a_{i,j}$, nakon čega se radi njihov rez (slika 4.1b). Dobiveni rezovi formiraju particije $\delta_0, ..., \delta_4$ prikazane na slici 4.1c, no bez vrhova v_9 i v_{10} jer oni sadrže samo komunikacijske veze s ostatkom programa (nema lukova koji sežu iz njih ili prema njima). Drugi dio particioniranja rekurzivno traži sinkro-vrhove i pri analizi particije δ_3 pronalazi preostala dva vrha te ih smješta u tu particiju, slika 4.1d. Na slici 4.1e prikazan je konačni oblik dobivenih particija — faza izvršavanja — s njihovim numeriranim vremenskim slijedom. Prvi korak postupka RaPPaMaG ishodi vremenski potpuno neovisne cjeline, od kojih se svaka može tretirati kao zaseban graf interakcije zadataka (TIG). Na svaki se dobiveni podgraf (fazu izvršavanja) tada mogu primijeniti vremenski zahtjevniji postupci sa svrhom dobivanja kvalitetnijih rješenja. Za particioniranje programa, ovisno o modelu, mogu se koristiti i neke druge metode grupiranja (npr. metode iz [10, 12, 58, 67, 97]), no uz njihovu prilagodbu na model miješanoga acikličkoga grafa. Važno je napomenuti da se postupci particioniranja grafa u literaturi većinom odnose na ukrupnjavanje, tj. smanjivanje broja čvorova grupiranjem, kao u [34, 44, 62, 97], što je i navedeno u pododjeljku 2.3.3. To se radi na način da su ukupni obradbeni zahtjevi unutar dobivenih particija približno jednaki, a rezovi bridova među njima minimalni. U postupku RaPPaMaG particioniranje se također koristi za grupiranje, no kriteriji za dodjelu jedinica programa particijama su već spomenute karakteristike miješanoga acikličkoga grafa — vremenski slijed i struktura sinkrone komunikacije među njima. U narednom poglavlju opisujemo drugi korak postupka RaPPaMaG, a to je raspoređivanje dobivenih particija.



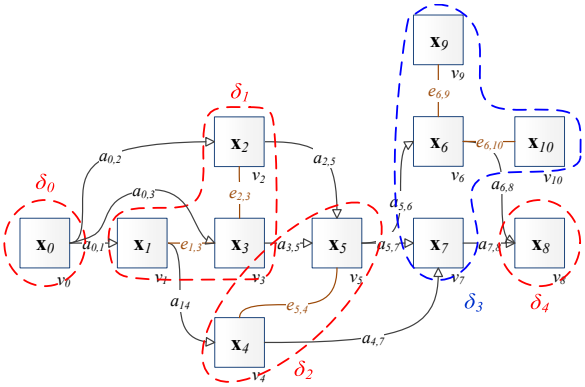
(a) Program.



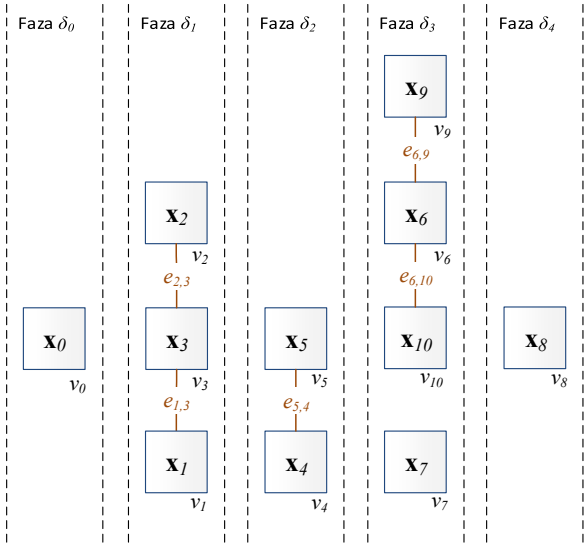
(b) Trivijalni rez lukova.



(c) Particije dobivene trivijalnim rezom.



(d) Ubacivanje vrhova bez prethodnika.



(e) Faze izvršavanja.

Slika 4.1: Primjer particioniranja programa u obliku miješanoga acikličkoga grafa.

4.1.2 Korak 2 postupka RaPPaMaG: raspoređivanje faza izvršavanja

Kao što je već spomenuto u pododjeljku 4.1, drugi korak postupka RaPPaMaG jest raspoređivanje pojedinih faza izvršavanja dobivenih u prethodnom koraku. Navedeno je također da su faze izvršavanja (particije) grafovi te da oni sadrže vremenski neovisne izvršne jedinice (koje se mogu izvršavati istovremeno — paralelno).

Kako bismo usredotočili primjenu navedenog postupka na programe koji sadrže sinkronu komunikaciju, motivirani programima poput onih iz repozitorija Pegasus spomenutih u pododjeljku 2.2.1 i MPI programa, ali bez utjecaja na optimalnost rješenja, uvodimo dvije dodatne pretpostavke:

1. pretpostavka: postojanje komunikacije među izvršnim jedinicama programa (neusmjerenoga brida) podrazumijeva njihovu sinkroniziranost,
2. pretpostavka: ukupan broj obradbenih jedinica sklopovlja raznorodne računalne platforme je veći ili jednak broju izvršnih jedinica unutar najveće faze izvršavanja (one s najvećim brojem vrhova):

$$\rho \geq \max_{i \in Z} |\delta_i|, \tag{4.1}$$

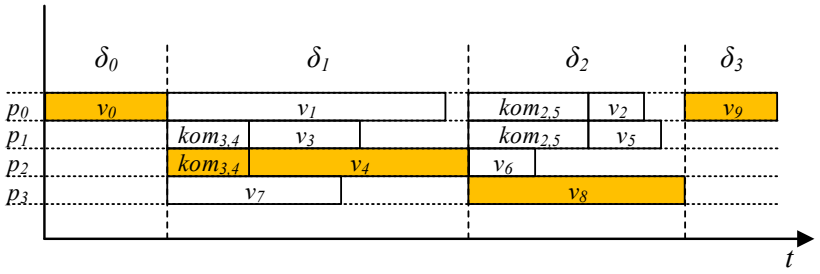
gdje je Z skup svih faza izvršavanja nekog programa.

Prva pretpostavka odražava situaciju koja se pojavljuje u MPI komunikacijskim pozivima, poput *broadcast*, *reduce*, *scatter*, *gather* i *all-to-all*, više u [45], OpenMP pozivom *Reduction* prema [46], ali i višestrukim MPI komunikacijskim pozivima slanja i primanja (eng. *send and receive*), a to je da svi sudionici komunikacije moraju aktivno sudjelovati u njoj dok ona traje. Također, različiti MPI entiteti programa u izvođenju — procesi — ne stavljaju se na istu obradbenu jedinicu. Zbog toga, podrazumijevamo da se izvršne jedinice koje međusobno komuniciraju moraju izvršavati istovremeno sa svim izvršnim jedinicama s kojima su komunikacijski vezani. Naravno, prije toga se mora omogućiti da je ispunjen vremenski slijed, tj. da nemaju nezavršenih prethodnika (korak 1 postupka RaPPaMaG). Druga pretpostavka odražava stvarnu sliku u primjeni. Dakle, postavljanje jednakog broja entiteta paralelnih programa (procesa, niti i sl.) kao i broja obradbenih jedinica (procesorskih jezgri i sl.) omogućuje najbolje iskorištavanje obradbenih resursa, tj. najveći stupanj paralelizacije (istovremenosti). U slučaju da je broj entiteta paralelnih programa veći, veća je vjerojatnost čekanja i problema s uvjetima utrkivanja (eng. *race conditions*) itd. Takav slučaj ne razmatramo postupkom RaPPaMaG.

Ukupno vrijeme izvršavanja faze $t^{uk}(\delta)$ definira se kao najveće od svih vremena izvršavanja jedinica programa unutar nje:

$$t^{uk}(\delta) = \max_{v_i \in \delta} \left(t_i^{uk} \right), \tag{4.2}$$

gdje je t_i^{uk} ukupno vrijeme izvršavanje izvršne jedinice i. Navedeno je i prikazano slikom 4.2, gdje su narančastom bojom označene izvršne jedinice koje određuju vrijeme izvršavanja pojedine faze. Za primjer sa slike 4.2 vremena izvršavanja pojedinih faza jednaka su vremenima izvršavanja jedinica v_0 za δ_0, v_4 za δ_1, v_8 za δ_2 i v_9 za δ_3. Navedeno uvjetuje da se ne može započeti s izvršavanjem faze dok njezina prethodna faza nije završila s radom. Time se treba čekati na završetak izvođenja svake od jedinica programa unutar nje, čime dolazimo do zaključka da će njezino vrijeme izvršavanja ovisiti o najsporijoj jedinici programa koju sadrži.



Slika 4.2: Primjer izvršavanja faza programa.

Drugi korak postupka RaPPaMaG prikazan je algoritmom 2. Postupak započinje tako da se sve obradbene jedinice računalne platforme spremaju kao skup dostupnih obradbenih jedinica HPU_{dost} (linija 1). Kao prva trenutno razmatrana faza izvršavanja δ^{cur} uzima se prva po vremenskom slijedu, δ_0 (linija 3). Algoritam za svaku fazu izvršavanja koristi model raspoređivanja programskog okvira LOSECO opisanog u pododjeljku 3.3 kako bi rasporedio izvršne jedinice v_i koje sadržava trenutna faza δ^{cur} (linije 5–10). To čini pomoću neke od metoda optimizacije više funkcija cilja (u našem slučaju iscrpnog pretraživanja). U istom se koraku obradbene jedinice na koje su raspoređene jedinice programa iz te faze označe nedostupnima (linija 7) i predviđa se ukupno vrijeme izvršavanja jedinica programa t^{uk} na njima. Također, dodijeljena obradbena jedinica uklanja se iz skupa dostupnih HPU_{dost}. Nakon toga, postavlja se ukupno vrijeme izvršavanja trenutne faze $t^{uk}(\delta^{cur})$ prema (4.2) (linija 11) i prelazi se na raspoređivanje sljedeće faze izvršavanja u nizu (linija 12).

Algoritam 2 RaPPaMaG, korak 2.

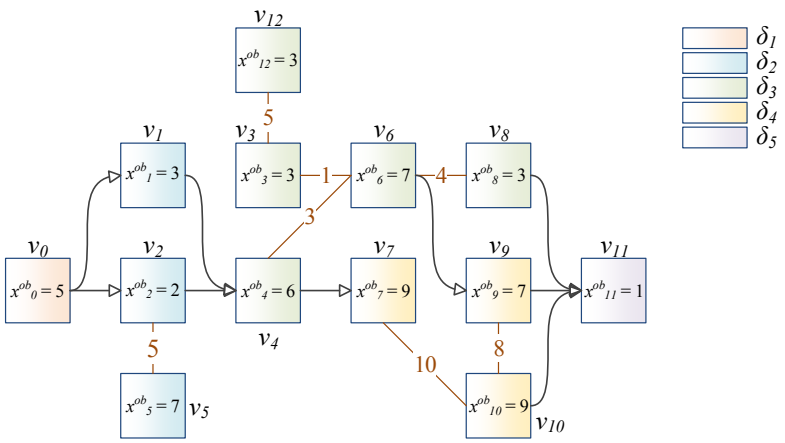
1: $HPU_{dost} \leftarrow P$ (skup svih obrabenih jedinica) {dostupne obradbene jedinice}
2: $HPU_{nedost} = \emptyset$ {nedostupne obradbene jedinice}
3: $\delta^{cur} = \delta_0$
4: **dok** ($\delta^{cur} \neq \emptyset$) **i** ($HPU_{avail} \neq \emptyset$) **radi**
5: **za sve** $v_i \in \delta^{cur}$ **radi**
6: rasporedi v_i na $HPU_j \in HPU_{dost}$
7: $HPU_{nedost} = HPU_{nedost} + \{HPU_j\}$
8: $HPU_{dost} = HPU_{dost} - \{HPU_j\}$
9: izračunaj t_i^{uk}
10: **završi za**
11: $t^{uk}(\delta^{cur}) = \max_{v_i \in \delta^{cur}} (t_i^{uk})$
12: $\delta^{cur} = \delta^{next}$
13: **završi dok**

Kako bi detaljno prikazali sve navedene karakteristike postupka RaPPaMaG, u sljedećem pododjeljku dajemo jednostavan primjer na kojem će spomenuti postupak biti primijenjen.

4.1.3 Primjer 1: izvođenje postupka RaPPaMaG

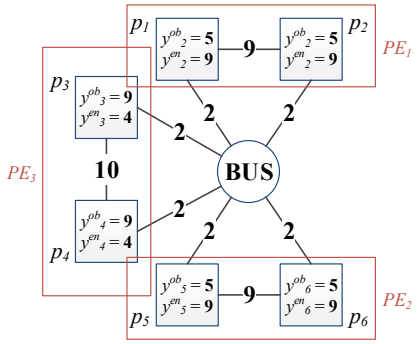
U ovom pododjeljku predstavljamo jednostavan primjer cjelovitog postupka RaPPaMaG. Za potrebe primjera napravljen je sintetički program prikazan modelom u obliku miješanoga acikličkoga grafa na slici 4.3. Broj x_i^{ob} unutar svakoga vrha $v_0, ..., v_{12}$ grafa označava računalne zahtjeve izvršne jedinice programa koju predstavlja (broj instrukcija koje treba obaviti), broj unutar smeđe označenih (neusmjerenih) bri-

dova označavaju obujam komunikacije među vrhovima koje povezuje (npr. $e_{i,j}$ megabajta), dok smjer lukova označava vremenski slijed njihovog izvođenja.



Slika 4.3: Model programa.

Također, generirana je i raznorodna računalna platforma (sklopovlje) prema modelu programskog okvira LOSECO na koju se treba rasporediti program sa slike 4.3, s pripadajućim parametrima, prikazana modelom usmjerenoga potpuno povezanog grafa na slici 4.4. Sklopovlje predstavlja računalni sustav s tri obradbena elementa (PE_1, PE_2, PE_3), od čega su obradbeni elementi PE_1 i PE_2 jednaki. Svaki od obradbenih elemenata sastoji se od dvije obradbene jedinice predstavljene vrhovima p_j, od kojih je svaka opisana s dvije vrijednosti, y_j^{ob} i y_j^{en}. Gornja vrijednost y_j^{ob} označava obradbenu sposobnost (broj instrukcija koje može izvesti u jedinici vremena) obradbene jedinice p_j, dok donja vrijednost y_j^{en} označava njezinu potrošnju energije (jedinica energije utrošenih po izvršenoj instrukciji). Brojevi na bridovima označavaju komunikacijsku propusnost $c_{j,k}$ (megabajta u sekundi) između obradbenih jedinica p_j i p_k. Dodan je poseban vrh u graf naziva BUS koji predstavlja sabirničku vezu među obradbenim elementima i uklanja potrebu za prikazom svih bridova grafa.



Slika 4.4: Model sklopovlja.

Primjenom prvog koraka algoritma postupka RaPPaMaG (pododjeljak 4.1.1) program se dijeli na pet vremenski neovisnih faza izvršavanja $\delta_0, ..., \delta_4$, označenih na slici 4.3. Postupak započinje stavljanjem ulaznog vrha grafa v_0 u fazu δ_0, nakon čega se prebacuje na traženje njegovih sljedbenika. Kada ih pronađe (vrhovi v_1 i v_2), provjerava postoji li vremenska ovisnost među njima, a s obzirom na to da

takve ovisnosti nema oni se stavljaju u sljedeću fazu, δ_1. Također, nad oba se sljedbenika (v_1, v_2) vrši provjera komuniciraju li oni s nekim od sinkro-vrhova, što je slučaj za v_2 koji komunicira sa sinkro-vrhom v_5 te se i on stavlja u fazu δ_1. Ostale faze formiraju se na sličan način. Od toga, specifična je faza δ_2 koja sadrži sinkro-vrh v_{12}, jer on komunicira samo s drugim sinkro-vrhom (v_4).

Primjenom drugog koraka postupka raspoređivanja prema algoritmu 2 raspoređuju se pojedine dobivene faze izvršavanja na obradbene jedinice sklopovlja. Drugim riječima, za svaku se bira podskup obradbenih jedinica λ_k na kojima se treba izvršiti faza kako bi se postigli korisnički definirani ciljevi.

Za potrebe primjera postavili smo sljedeće pokazatelje kvalitete, ciljeve (korisničke zahtjeve):

- cilj 1: smanjiti vrijeme izvođenja svake faze (time i ukupno vrijeme izvođenja programa)
- cilj 2: smanjiti utrošak energije pri izvođenju svake faze (time i utroška energije pri izvođenju cijelog programa).

Dakle, želimo smanjiti vrijeme izvršavanja programa na sklopovlju i pritom zadržati nisku potrošnju energije, što je vođeno stvarnim potrebama kod izvođenja programa u okruženju HPC. Problem se definira u obliku višekriterijske optimizacije prema modelu raspoređivanja LOSECO iz pododjeljka 3.3, dakle kao minimizacija pomoćne funkcije s više ciljeva (kriterija). Uzevši u obzir postavljene ciljeve, pomoćna funkcija se sastoji od dvije funkcije cilja:

- ukupno vrijeme izvršavanja faze $t^{uk}(\delta)$,
- ukupna utrošena energija faze $e^{uk}(\delta)$.

Nadalje, korisnik programa zadaje važnosti (prioritete) za oba cilja:

$$w(t^{uk}) = 0{,}8, \tag{4.3}$$

$$w(e^{uk}) = 0{,}2, \tag{4.4}$$

uzimajući u obzir da prema modelu raspoređivanja LOSECO mora vrijediti $\sum_i w_i = 1$, izraz (3.9), pododjeljak 3.3.

Prema karakteristikama postupka RaPPaMaG pretpostavljamo da je ukupno vrijeme izvršavanja pojedine faze jednako vremenu izvršavanja njezine najsporije izvršne jedinice (4.2), dok je ukupno vrijeme izvršavanja jedne jedinice programa v_i definirano s:

$$t_i^{uk} = t_i^{ob} + t_i^{kom} \tag{4.5}$$

gdje su t_i^{ob} i t_i^{kom} vremena izvršavanja i trajanja komunikacije za izvršnu jedinicu v_i. Faze se izvode slijedno, to jest, kad jedna završi druga započinje s radom.

Navodimo i ostale pretpostavke:

$$\delta \subseteq V, \qquad \lambda \subseteq P, \tag{4.6}$$

$$V = \delta_0 \cup \delta_1 \cup ... \cup \delta_n \tag{4.7}$$

$$|\delta| = |\lambda| = n \tag{4.8}$$

gdje je V skup svih vrhova grafa modela programa, dok je P skup svih vrhova grafa modela sklopovlja. Iz (4.6) vidljivo je da faza izvršavanja može biti cijeli graf (ukoliko ga nije moguće razdvojiti na faze izvršavanja) i tada je to program s potpuno vremenski neovisnim izvršnim jedinicama, $\delta = V$. Također, isti izraz definira λ kao podskup skupa obradbenih jedinica na koji se raspoređuju izvršne jedinice iz faze δ, koji također može biti i cijeli skup procesora P ukoliko je $|\delta| = \rho$. Prema (4.7) sve faze izvršavanja zajedno formiraju skup vrhova V i jedan vrh ne može biti član više od jedne faze. Izraz (4.8) označava pretpostavku 2 u pododjeljku 4.1.2, koja govori da je broj obradbenih jedinica sklopovlja veći ili jednak broju izvršnih jedinica unutar najveće faze izvršavanja (4.1) i taj je broj jednak n.

Raspoređivanje izvršnih jedinica pojedine faze δ na podskup procesora λ definirano je kao kartezijev produkt u kojem se za svaku izvršnu jedinicu v_i iz faze δ definira jedna obradbena jedinica sklopovlja p_j iz λ.

Dakle, funkcija raspoređivanja $g : \delta \to \lambda$ je podskup $\delta \times \lambda$ takav da za svaki $v_i \in \delta$ postoji jedinstveni $p_j \in \lambda$ takav da $(v_i, p_j) \in g$. Pišemo $g(v_i) = p_j$ ako $(v_i, p_j) \in g$:

$$g = \delta \times \lambda = \{(v_i, p_j) : \ v_i \in \delta, \ p_j \in \lambda\}. \tag{4.9}$$

U navedenom primjeru, a prema modelu raspoređivanja programskog okvira LOSECO, opisanom u pododjeljku 3.3, proizlazi da su članovi vektora $\mathbf{x}$ parametri $x^{ob}, e_{i,j}$ koji opisuju jedinice programa ($\chi = 2$), dok su članovi vektora $\mathbf{y}$ parametri $y^{ob}, y^{en}, c_{k,l}$ koji opisuju obradbene jedinice sklopovlja ($\omega = 3$). Ukoliko se primijeni (3.8) proizlazi da su članovi vektora $\mathbf{q}$ parametri $x^{ob}, e_{i,j}, y^{ob}, y^{en}, c_{k,l}$, pomoću kojih treba izraziti i funkcije cilja.

Pomoćnu funkciju definiramo u najjednostavnijem obliku kao težinski zbroj funkcija cilja (kriterija optimizacije), prema [93] i prema modelu raspoređivanja (3.7) programskog okvira LOSECO opisanom u pododjeljku 3.3:

$$u(\mathbf{q}) = w(t^{uk}) \cdot \mu_0(\mathbf{q}) + w(e^{uk}) \cdot \mu_1(\mathbf{q}), \tag{4.10}$$

gdje su funkcije μ_0 i μ_1 funkcije cilja dobivene normalizacijom u odnosu na njihov globalni maksimum:

$$\mu_0 = \frac{t^{uk}(\mathbf{q}_z)}{\max\limits_{\mathbf{q}}[t^{uk}(\mathbf{q}_z)]}, \tag{4.11}$$

$$\mu_1 = \frac{e^{uk}(\mathbf{q}_z)}{\max\limits_{\mathbf{q}}[e^{uk}(\mathbf{q}_z)]}, \tag{4.12}$$

gdje je $\mathbf{q}_z$ instanca vektora rasporeda $\mathbf{q}$, dakle jedan mogući raspored s pripadajućim vrijednostima parametara programa i sklopovlja. Funkcije cilja potrebno je normalizirati zbog toga da imaju jednak utjecaj na pomoćnu funkciju. Nakon ubacivanja važnosti pojedinih ciljeva $w(t^{uk})$ i $w(e^{uk})$ iz (4.3) i (4.4), pomoćna funkcija postaje:

$$u(\mathbf{q}) = 0{,}8 \cdot \mu_0(\mathbf{q}) + 0{,}2 \cdot \mu_1(\mathbf{q}), \tag{4.13}$$

Ukupno vrijeme izvršavanja $t^{uk}(\delta)$ faze izvršavanja δ definirano je kao maksimalno od zbroja vremena

utrošenih na obradu t_i^{ob} i na komunikaciju t_i^{kom} svake jedinice programa $v_i \in \delta$ prema (4.1) i (4.5):

$$t^{uk}(\delta) = \max_{v_i \in \delta} \left(t_i^{ob} + t_i^{kom} \right), \tag{4.14}$$

gdje je vrijeme obrade definirano kao

$$t_i^{ob} = \frac{x_i^{ob}}{y_j^{ob}}, \;\; p_j = g(v_i), \tag{4.15}$$

a vrijeme komunikacije kao:

$$t_i^{kom} = \sum_{\substack{v_i, v_j \in \delta,\, p_k, p_l \in \lambda \\ v_i \neq v_j \\ f(v_i)=p_k \\ f(v_j)=p_l}} \left(\frac{e_{i,j}}{c_{k,l}} + \frac{e_{j,i}}{c_{l,k}} \right), \tag{4.16}$$

gdje je $e_{i,j}$ obujam komunikacije između jedinica programa v_i i v_j, a $c_{k,l}$ propusnost komunikacijske veze između obradbenih jedinica p_k i p_l. Potrošnja energije e^{uk} pojedine faze izvršavanja δ definirana je kao:

$$e^{uk}(\delta) = \sum_{v_i \in \delta} e_i^{uk}, \tag{4.17}$$

dok je utrošak energije e_i^{uk} pri izvođenju jedne jedinice programa v_i definiran kao:

$$e_i^{uk} = \frac{x_i^{ob}}{y_j^{en}}. \tag{4.18}$$

Optimizacija se svodi na minimizaciju pomoćne funkcije:

$$\min(u(\mathbf{q})), \tag{4.19}$$

u prostoru svih $\lambda \in P$.

Za potrebe navedenog primjera upotrijebljena je metoda iscrpnog pretraživanja, kao i u [12, 82, 86], koja daje optimalno rješenje za manje instance problema, što je u slučaju navedenog primjera ispunjeno. Za veće instance problema umjesto navedene metode, mogu se koristiti naprednije metode poput metoda vođenog lokalnog pretraživanja (diferencijalna evolucija, optimizacija rojem čestica, simulirano kaljenje i dr.). Dobiveni raspored za pojedine faze prikazan je tablicom 4.1. U navedenoj tablici je, između ostaloga, vidljiv ukupan utrošak energije za izvođenje cijelog programa i on iznosi 355 jedinica energije, dok duljina rasporeda, (vrijeme izvršavanja programa) iznosi 18,02 vremenskih jedinica.

Tablica 4.1: Raspored faza izvršavanja na obradbenim jedinicama sklopovlja iz primjera 1.

	δ_0	δ_1	δ_2	δ_3	δ_4	**ukupno**
SEU-ovi (δ)	v_0	v_1, v_2, v_5	$v_3, v_4, v_6, v_8, v_{12}$	v_7, v_9, v_{10}	v_{11}	
HPU-ovi (λ)	p_2	p_0, p_2, p_3	p_0, p_2, p_3, p_1, p_4	p_2, p_0, p_3	p_2	
t^{uk}	0,56	1,28	6,08	10,00	0,11	**18,02**
e^{uk}	20	63	133	135	4	**355**

Rješenja za Primjer 1 dobivena u tablici 4.1 su optimalna rješenja za svaku od particija, koja mogu činiti

rješenje blizu optimalnog za cijeli program.

Da bi dobili bolji uvid u ovisnost performansi raspoređivanja o broju jedinica programa i jedinica sklopovlja u sljedećem pododjeljku dajemo analizu složenosti faze 2 postupka RaPPaMaG.

Analiza složenosti postupka RaPPaMaG

U slučaju da postoji k faza izvršavanja $\delta_0, ..., \delta_{k-1}$ i da je broj izvršnih jedinica u fazi δ_i dan kao n_i, ukupan broj rješenja m dan je kao zbroj uređenih n-torki izvršnih jedinica pojedine faze od ρ elemenata sklopovlja:

$$m = \sum_{i=0}^{k} Per(\rho, n_i). \tag{4.20}$$

U slučaju Primjera 1 ukupan broj mogućih rješenja je:

$$m = Per(6,1) + Per(6,3) + Per(6,5) + Per(6,3) + Per(6,1) = 967, \tag{4.21}$$

gdje je $Per(i,j)$ permutacija bez ponavljanja odabranih j elemenata od ukupno i elemenata (u navedenom slučaju broja obradbenih jedinica).

Kada bi se skupovi svih izvršnih jedinica programa V i obradbenih jedinica sklopovlja P raspoređivali u cijelosti, tj. bez razdvajanja na faze izvršavanja, problem se prema [104] definira kao:

$$m = \frac{(\vartheta + \rho - 1)!}{\vartheta!(\rho - 1)!} \cdot \rho! = (\vartheta + \rho - 1)! \cdot \rho \tag{4.22}$$

što bi za navedeni primjer bilo jednako 2.134.124.568.576.000 ($2{,}13 \times 10^{15}$) ili 2.206.954.052.301 ($2{,}2 \times 10^{12}$) puta više nego uz korištenje particioniranja kao prvog koraka postupka RaPPaMaG.

Naravno, u posebnim slučajevima postupak RaPPaMaG može također zahtijevati velik broj iteracija (kada $|\delta| \to \vartheta$), no takvi slučajevi se ionako odnose na programe s nezavisnim jedinicama programa ili programe s visokom razinom paralelizma (eng. *embarrassingly parallel programs*). Također, nedostatak navedenog postupka se događa u slučajevima programa s mnogo nezavisnih paralelnih puteva koji se međusobno uvelike razlikuju. Tada postupak može stvoriti neučinkovit raspored. Međutim, takvi su slučajevi u praksi vrlo rijetki, što je primjerice vidljivo i iz programa repozitorija Pegasus [2, 49, 50]. Za prosječan slučaj, ovakav postupak može imati primjenu za manji broj jedinica programa i sklopovlja, što će pokazati i naša eksperimentalna analiza u poglavlju 5.

Postupak RaPPaMaG, uz sam model miješanoga acikličkoga grafa kao ulaznog modela programskog okvira LOSECO (doprinos D.1.), čini naš izvorni znanstveni doprinos D.2.

4.1.4 Specifičnosti ulaznih parametara za postupak RaPPaMaG

Iz Primjera 1 u prethodnom pododjeljku ističemo da se eksplicitno zadavanje parametara odvojenih za entitete raspoređivanja — sklopovlje i program, prema programskom okviru LOSECO i njegovim modelima, može upotrijebiti i za ostale postupke traženja mogućeg rasporeda pojedine faze. To nije slučaj u većini postupaka raspoređivanja, koji koriste već poznate tablice troškova izvođenja svake izvršne jedi-

nice programa na svim obradbenim jedinicama sklopovlja. Tablice troškova mogu se dobiti zapisivanjem vrijednosti t_i^{ob} i e_i^{uk} u njih, koje su prethodno izračunate pomoću (4.15) i (4.18). Tako se za navedeni primjer dobivaju tablice 4.2 i 4.3. Matrice troškova iz tablica 4.2 i 4.3 mogu se upotrijebiti, primjerice, za postupke raspoređivanja zasnovane na rangiranju jedinica programa. Zatim, navedeno omogućuje da apsolutan zapis parametara može poslužiti kao univerzalna baza podataka za većinu postupaka raspoređivanja i većinu domena računarstva u kojima se ti postupci koriste. Primjeri u kojima se koriste su većinom pri raspoređivanju programa na WSN-e, primjerice [51] ili na NoC sustave, kao u [40].

Tablica 4.2: Vremena izvršavanja svih jedinica programa na svim obradbenim jedinicama sklopovlja, matrica troškova 1.

	p_0	p_1	p_2	p_3	p_4	p_5
t_0^{ob}	1.00	1.00	0.56	0.56	1.00	1.00
t_1^{ob}	0.60	0.60	0.33	0.33	0.60	0.60
t_2^{ob}	0.40	0.40	0.22	0.22	0.40	0.40
t_3^{ob}	0.60	0.60	0.33	0.33	0.60	0.60
t_4^{ob}	1.20	1.20	0.67	0.67	1.20	1.20
t_5^{ob}	1.40	1.40	0.78	0.78	1.40	1.40
t_6^{ob}	1.40	1.40	0.78	0.78	1.40	1.40
t_7^{ob}	1.80	1.80	1.00	1.00	1.80	1.80
t_8^{ob}	0.60	0.60	0.33	0.33	0.60	0.60
t_9^{ob}	1.40	1.40	0.78	0.78	1.40	1.40
t_{10}^{ob}	1.80	1.80	1.00	1.00	1.80	1.80
t_{11}^{ob}	0.20	0.20	0.11	0.11	0.20	0.20
t_{12}^{ob}	0.60	0.60	0.33	0.33	0.33	0.33

Tablica 4.3: Utrošak energije svih jedinica programa na svim obradbenim jedinicama sklopovlja, matrica troškova 2.

	p_0	p_1	p_2	p_3	p_4	p_5
e_0^{uk}	45	45	20	20	45	45
e_1^{uk}	27	27	12	12	27	27
e_2^{uk}	18	18	8	8	18	18
e_3^{uk}	27	27	12	12	27	27
e_4^{uk}	54	54	24	24	54	54
e_5^{uk}	63	63	28	28	63	63
e_6^{uk}	63	63	28	28	63	63
e_7^{uk}	81	81	36	36	81	81
e_8^{uk}	27	27	12	12	27	27
e_9^{uk}	63	63	28	28	63	63
e_{10}^{uk}	81	81	36	36	81	81
e_{11}^{uk}	9	9	4	4	9	9
e_{12}^{uk}	27	27	12	12	27	27

Postupak RaPPaMaG je glavni dio našeg doprinosa D.2. koji se izravno nadovezuje na pretpostavke programskog okvira LOSECO, tako da zajedno čine doprinose D.1. i D.2. On omogućuje raspoređivanje jedinica programa prema višestrukim kriterijima na jedinice sklopovlja u raznorodnim računalnim sustavima. Najveća novost tog postupka je postupak particioniranja, za koji program mora biti prikazan kao miješani aciklički graf. No, kako ćemo vidjeti u eksperimentalnoj analizi poglavlja 5, postupak RaPPa-

MaG može se primijeniti i na modele programa u obliku DAG-a, što mu dodatno povećava primjenjivost.

Naš opisani postupak RaPPaMaG pripada skupini metaheurističkih postupaka raspoređivanja, no uz prednosti koje smo naveli i manje preinake, može se lako primijeniti i umjesto drugih postupaka raspoređivanja.

Naš sljedeći unaprijeđeni postupak RaNDKiP, koji se temelji na rangiranju jedinica programa, čini doprinos D.3. i prikazan je u idućem pododjeljku.

4.2 Unaprijeđeni postupak raspoređivanja RaNDKiP

Kao što je ranije spomenuto (pododjeljak 4.1), problem raspoređivanja programa sklopovlju je NP-težak, zbog čega su se razvili mnogi heuristički i metaheuristički postupci raspoređivanja izvršnih jedinica programa na sklopovlje (navedeni u pododjeljku 2.3.2). Iako naša podjela postupaka raspoređivanja razdvaja postupke raspoređivanja rangiranjem jedinica programa (eng. *list scheduling methods*), opisanih u pododjeljku 2.3.1, od ostalih, oni su jedna od skupina heurističkih postupaka. *List scheduling* postupci postižu vrlo dobra rješenja u usporedbi s njihovim vremenom izvođenja u odnosu na sve popularnije bio-inspirirane postupke vođenog lokalnog pretraživanja, prema [67]. Navedeni postupci zasnivaju se na prvotnom rangiranju izvršnih jedinica programa prema nekim kriterijima (primjerice udaljenosti od ulaznog vrha [41], udaljenosti od izlaznog vrha [37] i dr.), a zatim ih raspoređuju prema vrijednosti dobivenog ranga. Postoji nekoliko postupaka raspoređivanja u navedenoj skupini. Najčešće je korišten postupak koji za rangiranje izvršnih jedinica koristi tzv. raznorodno procijenjeno vrijeme završetka (eng. *heterogeneous estimated finish time*, HEFT), [41, 42], opisan u pododjeljku 2.3.5. Navedeni postupak je poslužio drugim autorima ili kao jedna od metoda hibridnog pristupa, poput simuliranog kaljenja [40], bio-inspiriranih postupaka [34], metoda za uštedu energije [39, 44] i drugih, ili kao podloga za razvoj sličnih postupaka, poput LDCP [37], PEFT [13] i umnažanja izvršnih jedinica [38]. Međutim, u [13, 37] prijavljeni su bolji rezultati od navedenog postupka HEFT, kako po optimalnosti rješenja tako i po performansama (iako autori u [13] smatraju da su ti rezultati još pod upitnikom), što ukazuje na to da još uvijek postoji mogućnost unaprjeđivanja postupaka raspoređivanja na temelju rangiranih listi. Zbog toga, kao i već spomenutih prednosti, postupci raspoređivanja na temelju rangiranih listi poslužili su kao ideja na kojoj se temelji naš postupak raspoređivanja.

Vođeni ciljem poboljšanja performansi postupaka raspoređivanja, unaprjeđivanjem optimalnosti rješenja i uvođenjem dodatnih kriterija raspoređivanja osim performansi predstavljamo unaprijeđeni postupak raspoređivanja RaNDKiP (**Ra**znorodni **N**ajduži **D**inamički **Kri**tični **P**ut) koji predstavlja naš izvorni znanstveni doprinos D.3. Navedeni postupak raspoređuje programe sklopovlju na temelju rangirane liste izvršnih jedinica uz podršku za graf programa s više izlaznih vrhova, za nepovezane grafove programa, za raznorodne komunikacijske veze sklopovlja, (raspoređivanje komunikacije) te uštedu energije skaliranjem frekvencije procesora za nekritične izvršne jedinice programa.

Unaprjeđenja postupka RaNDKiP u odnosu na postojeće postupke raspoređivanja rangiranjem izvršnih jedinica programa su sljedeća:

1. **raznorodno rangiranje** izvršnih jedinica programa uzimajući u obzir

   (a) raznorodnost komunikacije
   (b) grafove programa s više izlaznih vrhova
   (c) nepovezane grafove programa

2. **detaljnija izrada rasporeda** uz eksplicitno uvođenje komunikacije u raspored (podrška za sustave s raznorodnim komunikacijskim vezama)

   (a) raspoređivanjem komunikacijskih poziva
   (b) podrazumijevanjem raznorodnosti mreže sklopovlja pri svakom ažuriranju rasporeda

3. **smanjivanje utroška energije** skaliranjem frekvencije procesora za nekritične izvršne jedinice.

Navedena unaprjeđenja ujedno su i tri glavna dijela postupka RaNDKiP. U narednim pododjeljcima detaljnije će biti opisani svaki od njih.

4.2.1 Raznorodno rangiranje izvršnih jedinica programa

Postoje različiti postupci rangiranja, (davanja prioriteta) izvršnih jedinica programa, što je i opisano u pododjeljku 2.3.1. Većinom se za računanje ranga pojedine izvršne jedinice uzima neka vrsta troška izvođenja te izvršne jedinice, primjerice prosječno vrijeme izvođenja na svim procesorima [41,67], vrijeme izvođenja na svakom od procesora zasebno [37] i drugih, poput tablice optimističnih troškova [13]. Način računanja ranga većinom se temelji na obilasku grafa, primjerice u [37] koristi se rastući rang (izlazni vrhovi grafa imaju najmanji rang koji je jednak trošku), u [41] se koristi rastući rang za postupak HEFT te kombinacija padajućeg i rastućeg za postupak CPOP. Osnovna ideja rangiranja izvršnih jedinica programa u postupku RaNDKiP zasnovana je na rangiranju postupka najduljeg dinamičkog kritičnog puta (eng. *longest dynamic critical path*, LDCP), prema [37], opisanom u pododjeljku 2.3.6. Kao što je već spomenuto, navedeni postupak rangira (tj. postavlja prioritet) izvršne jedinice programa prema njihovoj najvećoj "udaljenosti" od izlaznog čvora (pretpostavlja se jedan izlazni čvor). Ta se udaljenost rekurzivno definira za vrh v_i u pripadajućem usmjerenom acikličkom grafu procesora p_j ($DAGP_j$) (kojemu su vremena izvršavanja jedinica programa postavljena na obradbenu jedinicu p_j) kao rastući rang $RRang_j(n_i)$:

$$RRang_{i,j} = w_{i,j} + \max_{v_k \in sljed_j(v_i)} \left[e^j_{i,k} + RRang_{k,j} \right], \tag{4.23}$$

gdje je $w_{i,j}$ vrijeme izvršavanja jedinice programa v_i na obradbenoj jedinici p_j, tj. trošak vrha v_i, $sljed_j(v_i)$ je skup neposrednih sljedbenika vrha v_i u $DAGP_j$, dok je $e^j_{i,k}$ iznos komunikacije među vrhovima v_i i v_k u $DAGP_j$.

Član izraza za rangiranje izvršnih jedinica (4.23) navedenog postupka koji unosi vrijednost komunikacije u rang pojedine izvršne jedinice, $e^j_{i,k}$ podrazumijeva samo homogenu komunikacijsku mrežu sklopovlja te razlikuje samo dvije vrijednosti:

- vrijednost $e_{i,j}$: iznos komunikacije ukoliko su izvršne jedinice dodijeljene različitim obradbenim jedinicama (ekvivalentna obujmu komunikacije između izvršnih jedinica u programu, neovisan o naravi komunikacijske veze između obradbenih jedinica na koje su raspoređeni),

- vrijednost 0: iznos komunikacije ukoliko su oba sudionika (izvršne jedinice) dodijeljene istoj obradbenoj jedinici (pretpostavka kako je brzina prijenosa podataka unutar iste obradbene jedinice mnogo brža od ostalih komunikacijskih veza u sustavu).

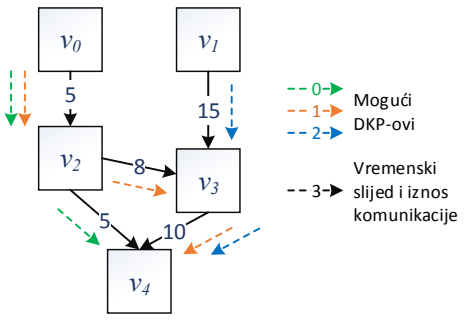
Na taj način zanemaruje se utjecaj komunikacije u sustavima s raznorodnim komunikacijskim vezama među obradbenim jedinicama, koje se mogu razlikovati i za nekoliko redova veličine i time uzrokovati drugačije rangiranje (ali i drugačiji raspored). Na primjer, takve se razlike odnose na komunikaciju između procesorskih jezgri, sabirničku komunikacije između obradbenih elemenata, različite vrste lokalne mreže itd.

Navedeno svojstvo prisutno je i u postupcima HEFT i CPOP [41], PEFT [13], DBUS [44], kao i svim ostalim postupcima raspoređivanja rangiranom listom, prema našim dosadašnjim saznanjima, što smo i naveli u pododjeljku 2.3.1.

Iz tog razloga postupak RaNDKiP koristi specijalizirani tretman komunikacije pri postupku rangiranja. Kako bi prikazali odraz rangiranja na ishod mogućeg rasporeda upotrebom postupka RaNDKiP, u sljedećim odlomcima bit će analiziran primjer raspoređivanja pomoću dvije unaprijeđene inačice postupka RaNDKiP.

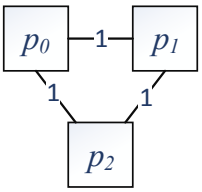
4.2.2 Primjer 2: raznorodno rangiranje jedinica programa postupkom RaNDKiP

Na slici 4.5 prikazan je primjer programa prikazanog usmjerenim acikličkim grafom. Primjer programa sastoji se od pet izvršnih jedinica prikazanim vrhovima $v_0, ..., v_4$ te pripadajućim vremenskim slijedom (smjerom lukova grafa) i obujmom komunikacije među njima, (vrijednost upisana u lukove grafa). Graf ima dva ulazna vrha v_0 i v_1 te jedan izlazni vrh v_4. Na slici 4.5 također su prikazani svi mogući dinamički kritični putevi (DKP-ovi, putevi kojima se od ulaznih vrhova može doći do izlaznog).



Slika 4.5: Program u obliku DAG-a koji treba rasporediti i njegovi DKP-ovi.

Pretpostavljamo također računalnu platformu koja se sastoji od 3 obradbene jedinice $p_0, ..., p_2$, prikazanu slikom 4.6, povezanu homogenim komunikacijskim vezama.



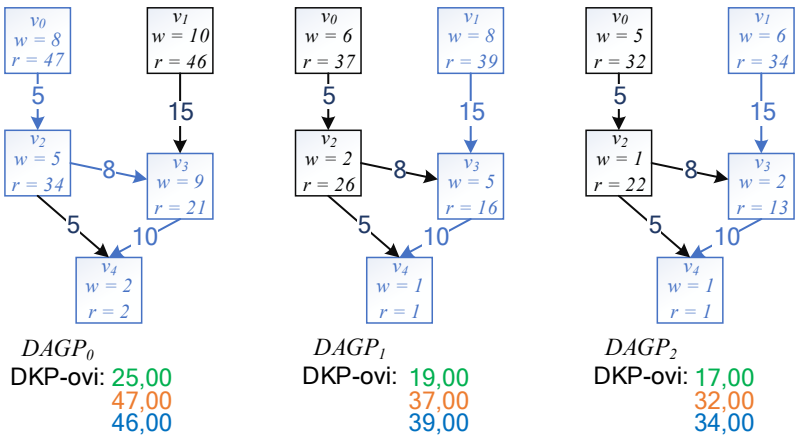
Slika 4.6: Platforma s homogenim komunikacijskim vezama na koju se raspoređuje program sa slike 4.5.

S obzirom na to da izvorni postupak LDCP ne uzima karakteristike komunikacijskih veza sklopovlja u obzir, veze su u primjeru platforme postavljene na 1. Vremena izvršavanja jedinica programa na svakoj obradbenoj jedinici (matrica troškova), dana su tablicom 4.4.

Tablica 4.4: Vremena izvršavanja jedinica programa sa slike 4.5.

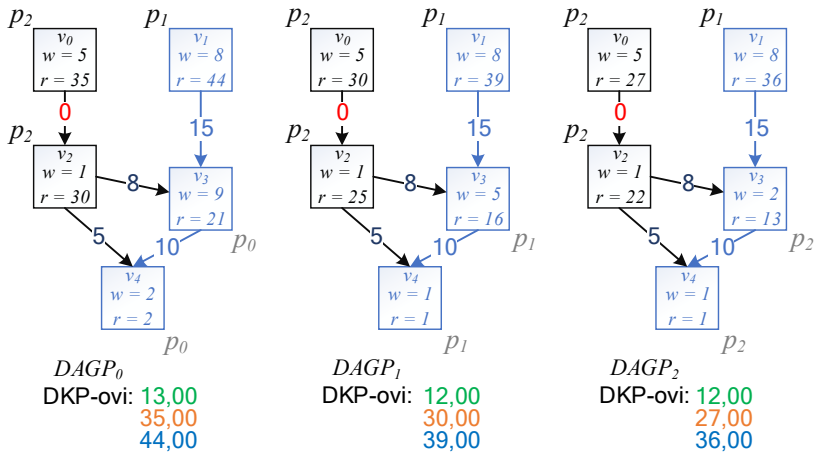
	p_0	p_1	p_2
v_0	8	6	5
v_1	10	8	6
v_2	5	2	1
v_3	9	5	2
v_4	2	1	1

Rangiranjem izvršnih jedinica pomoću (4.23), neposredno prije prvog koraka raspoređivanja, dobiveni su dinamički kritični putevi prikazani slikom 4.7, gdje w označava obradbeni zahtjev izvršne jedinice, a r njezin *RRang*. Vrhovi i bridovi najdužih dinamičkih kritičnih puteva (NDKP-ova) označeni su plavom bojom. Za p_0 ($DAGP_0$) najduži je put v_0, v_2, v_3, v_4 u iznosu od 47, dok su za p_1 i p_2 ($DAGP_1$ i $DAGP_2$) oni v_1, v_3, v_4 u iznosima od 37 i 32, tako da se globalni NDKP nalazi u DAGP-u od p_0. Prema izvornom postupku raspoređuje se vrh grafa unutar globalnog NDKP-a s najvećom vrijednosti *RRang*-a, a to je v_0, $RRang_{0,0} = 47$.



Slika 4.7: Iznos DKP-ova prije raspoređivanja (homogene komunikacijske veze sklopovlja).

Ako se prema NDKP algoritmu izvršne jedinice v_0 i v_2 rasporede obradbenoj jedinici p_2, a izvršna jedinica v_1 na p_1, pripadajući DAGP-ovi prikazani su slikom 4.8. U pripadajućim DAGP-ovima komunikacija između vrhova v_0 i v_2 postavljena je na 0, zbog toga što su pripadajuće izvršne jedinice raspoređene istom procesoru (p_1).



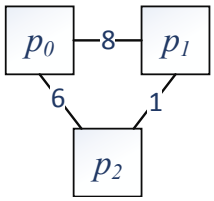
Slika 4.8: Iznos DKP-ova nakon raspoređivanja v_0, v_1 i v_2 (homog. platforma).

Čitav slijed postupka raspoređivanja primjera sa slike 4.5 prikazan je tablicom 4.5. Iz navedene tablice može se primijetiti da niti jedna izvršna jedinica nije raspoređena obradbenoj jedinici p_0. Razlog tome jest što su na njoj vremena izvršavanja najveća i uz to se ne mogu izolirati više od dvije vremenski neovisne izvršne jedinice programa prikazanog na 4.5 (koje se mogu izvršavati istovremeno). Samim time istovremeno se mogu izvršavati najviše dvije izvršne jedinice i postupak bira gdje će to za njih biti najbrže. S obzirom na to da su komunikacijske veze jednake, (simulacija ponašanja izvornog postupka LDCP koji ih podrazumijeva), one ne utječu na odabir, iako brža mreža potencijalno može nadomjestiti nedostatak obradbene snage jedinice p_0.

Tablica 4.5: Koraci postupka raspoređivanja programa sa slike 4.5.

Korak	Raspoređivanje
1.	$v_0 \rightarrow p_2$
2.	$v_1 \rightarrow p_1$
3.	$v_2 \rightarrow p_2$
4.	$v_3 \rightarrow p_2$
5.	$v_4 \rightarrow p_1$

Razmotrimo sada da su stvarne propusnosti komunikacijskih veza kao na slici 4.9. Vidi se kako je komunikacijska propusnost između p_0 i p_1 8 puta veća nego veza između p_1 i p_2 te 1,33 puta brža nego između p_0 i p_2. Kako bismo iskoristili navedenu informaciju u poboljšanju rasporeda izvršnih jedinica na sklopovlju kreirali smo modificirani pristup **raznorodnog rangiranja** koji uzima u obzir ne samo iznos komunikacije među izvršnim jedinicama programa, već i vrstu komunikacijske veze između obradbenih jedinica na koje su ti sudionici komunikacije dodijeljeni. Na taj se način vrijednost ranga $RRang_{i,j}$ može promijeniti, posebice u sustavima s iznosom parametra omjera obrade i komunikacije (eng. *communication-to-computation ratio*, CCR), većim od jedan i u sustavima s izrazito raznorodnom komunikacijskom strukturom, što će pokazati i eksperimentalna analiza postupka RaNDKiP u pododjeljku 5.4. Unaprijeđeni postupak RaNDKiP , (raznorodni najduži dinamički kritični put), temelji se na zasebnoj parametrizaciji komunikacije među izvršnim jedinicama programa i propusnosti komunikacijskih veza među obradbenim jedinicama.



Slika 4.9: Platforma s raznorodnim komunikacijskim vezama.

Komunikacija među izvršnim jedinicama v_i i v_j zadaje se kao broj informacija koje treba prenijeti (npr. $e_{i,j}$[MB]). Obradbene jedinice na koje su raspoređene izvršne jedinice su $p_k = g(v_i)$ i $p_l = g(v_j)$. Propusnost komunikacijske veze obradbenih jedinica p_k i p_l zadaje se kao broj informacija po jedinici vremena (npr. $c_{k,l}$[MB/s]). Tada je utjecaj komunikacije u DAGP-u obradbene jedinice u, $DAGP_u$ na vremenski slijed definiran kao:

$$e^u_{i,j} = \frac{e_{i,j}}{c_{k,l}}, \tag{4.24}$$

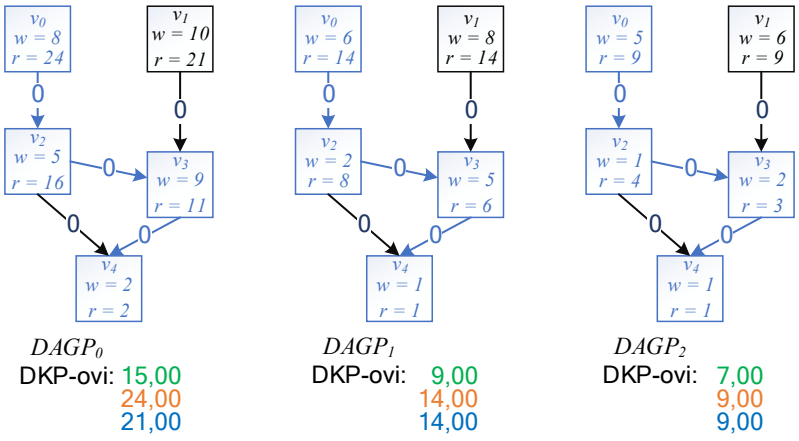
izražen u jedinici vremena (konkretno sekundama). Na taj način izraz (4.23) postaje:

$$RRang_{i,u} = w_{i,u} + \max_{v_j \in sljed_u(n_i)} \left[\frac{e_{i,j}}{c_{k,l}} \cdot \alpha + RRang_{j,u} \right], \tag{4.25}$$

$$\alpha = \begin{cases} 0, & k = l, \\ 1, & \text{inače}, \end{cases} \tag{4.26}$$

gdje α označava čimbenik komunikacije i jednak je 0 ukoliko se izvršne jedinice koje komuniciraju nalaze na istoj obradbenoj jedinici, dok je inače jednak 1.

Dobiveni DKP-ovi i DAGP-ovi primjenom modificiranog postupka prema (4.26), neposredno prije prvog koraka raspoređivanja, prikazani su na slici 4.10.

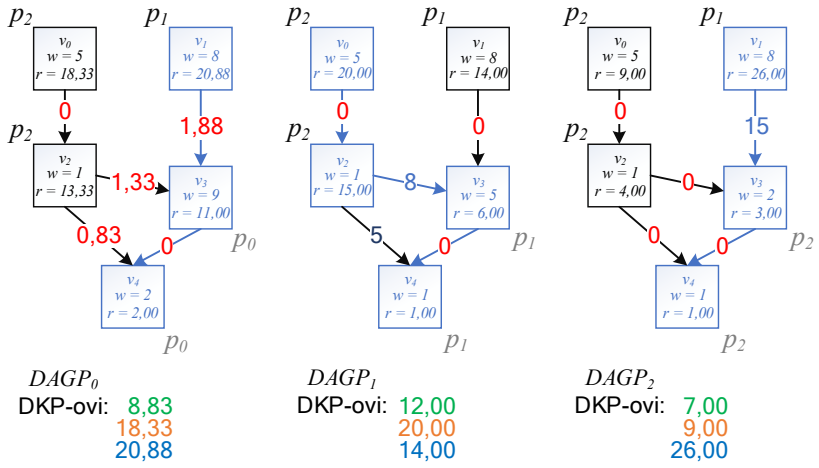


Slika 4.10: Iznos DKP-ova prije raspoređivanja, RaNDKiP (raznorodna platforma).

Vrijednosti za *RRang* sada su drugačije jer se u obzir uzimaju i propusnosti komunikacijskih veza sklopovlja. Komunikacija među svim jedinicama programa u ovom je koraku postavljena na nulu jer, prisjetimo se, smatra se da su u svakom DAGP-u sve izvršne jedinice dodijeljene istom procesoru. DKP-

ovi se također razlikuju u odnosu na izvorni postupak LDCP i za očekivati je da će RaNDKiP u takvim slučajevima birati izvršne jedinice drugačijim slijedom nego u izvornom postupku LDCP.

Nakon prva tri koraka raspoređivanja postupkom RaNDKiP ($p_2 \leftarrow v_0$, $p_2 \leftarrow v_2$ te $p_1 \leftarrow v_1$) dobiveni su DKP-ovi prikazani slikom 4.11.



Slika 4.11: Iznos DKP-ova nakon raspoređivanja v_0, v_1 i v_2, RaNDKiP (raznorodna platforma).

Vidljivo je kako su sada vrijednosti RRang-a drugačije (utjecaj komunikacije ovisi o komunikacijskoj vezi među obradbenim jedinicama), nego što je to bio slučaj kod podrazumijevanja homogene mreže među obradbenim jedinicama sklopovlja sa slike 4.8. Konkretnije, vrijednosti komunikacije koje su izmijenjene označene su crvenom bojom. Vidi se i da je DKP za $DAGP_1$ drugačiji u odnosu na slučaj s homogenom komunikacijom.

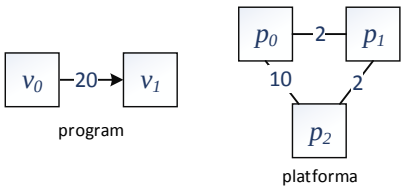
Primjer 2 prikazuje kako je uzimanjem u obzir raznorodne komunikacije među jedinicama sklopovlja (raznorodnim rangiranjem), postupak rangiranja drugačiji, čime je drugačiji i sam redoslijed odabira jedinica programa za raspoređivanje, sve to uz zadržavanje identične razine složenosti algoritma. Postupak raznorodnog rangiranja dovodi do manjih poboljšanja kvalitete postupka, pogotovo u programima u kojima dominira komunikacija (veliki CCR), što će biti prikazano u eksperimentalnoj analizi u poglavlju 5.

U idućem pododjeljku opisujemo sljedeći dio postupka RaNDKiP, a to je raspoređivanje komunikacijskih poziva.

4.2.3 Raspoređivanje komunikacije

Jedno od glavnih svojstava unaprijeđenog postupka RaNDKiP jest mogućnost raspoređivanja, ne samo izvršnih jedinica programa, već i komunikacije među njima. Naime, prilikom rasporeda izvršnih jedinica programa velik utjecaj na raspored može imati i trajanje komunikacije među njima. To je posebno važno kod postupka odabira obradbene jedinice jer se, u slučaju da se jedinica programa izvršava na brzom procesoru povezanog komunikacijskim vezama manje propusnosti, mogu dobiti lošije performanse nego u slučaju da se ona izvršava na obradbenoj jedinici manjih performansi s većom komunikacijskom propusnosti. Većina postupaka raspoređivanja, kao što je spomenuto ranije, podrazumijeva tzv. *unity*

pristup [37], gdje je platforma potpuno povezana i homogena. Drugi pak postupci (HEFT u [41] i PEFT u [13]) podrazumijevaju prosječne vrijednosti, kako za obradbene zahtjeve jedinica programa, tako i za komunikacijske veze među obradbenim jedinicama sklopovlja, (detaljnije u pododjeljku 2.3.5). Kako bismo zaobišli navedeni nedostatak, u algoritam raspoređivanja RaNDKiP, točnije, u fazu odabira obradbene jedinice, uvodi se provjera kako će utjecati raspoređivanje izvršne jedinice na taj procesor uzevši u obzir ne samo trajanje njezinog izvršavanja, već i trajanje komunikacije koja će nastati tim rasporedom. Navedeno ilustriramo primjerom sa slike 4.12. Na slici 4.12a prikazan je primjer jednostavnog programa s dvije izvršne jedinice i obujmom komunikacije među njima te platforme koja se sastoji od 3 obradbene jedinice i njihovim komunikacijskim vezama. Slika 4.12b prikazuje vremena izvršavanja obje izvršne jedinice na svakoj od obradbenih jedinica.



(a) Program i sklopovlje (platforma).

	p_0	p_1	p_2
v_0	4	8	2
v_1	2	4	1

(b) Vremena izvršavanja jedinica programa.

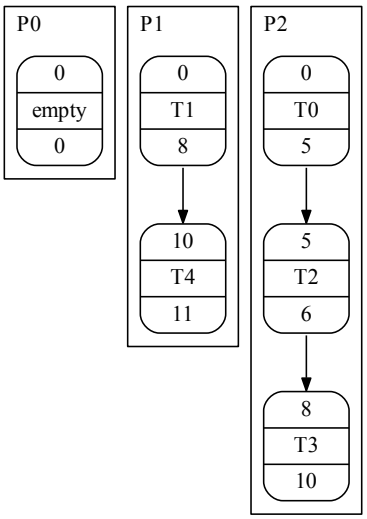
Slika 4.12: Primjer raspoređivanja u ovisnosti o komunikaciji.

Mogući rasporedi su prikazani u tablici 4.6. Kao što se vidi iz navedene tablice, postoji 6 mogućih rasporeda. Ukoliko se ne uzima u obzir raspored komunikacije među njima najmanje vrijeme izvođenja dobiva se pri rasporedu $g(v_0) = p_2$, $g(v_1) = p_0$, u iznosu od 4. Ukoliko se vrijeme potrebno za komunikaciju uzima u obzir, najboljim rasporedom pokazuje se $g(v_0) = p_2$, $g(v_1) = p_1$ u iznosu od 8. Primjerice, ako se komunikacija zanemari i kao raspored se uzme $g(v_0) = p_2$, $g(v_1) = p_0$ (sugerirano najkraćim vremenom izvršavanja), stvarno bi vrijeme izvršavanja bilo 14, što je za 29% veće nego vrijeme izvođenja koje bi se dobilo uzevši u obzir komunikaciju pri raspoređivanju.

Tablica 4.6: Vremena izvršavanja programa.

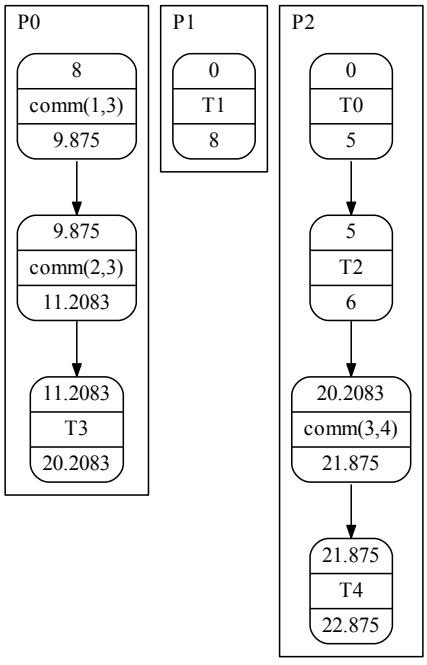
Raspored	Vrijeme bez kom.	Vrijeme s kom.
$g(v_0) = p_0$, $g(v_1) = p_1$	$4 + 4 = 8$	$4 + 4 + 2 = 10$
$g(v_0) = p_1$, $g(v_1) = p_0$	$8 + 2 = 10$	$8 + 2 + 2 = 12$
$g(v_0) = p_0$, $g(v_1) = p_2$	$4 + 1 = 5$	$4 + 1 + 10 = 15$
$g(v_0) = p_2$, $g(v_1) = p_0$	**2+2=4**	$2 + 2 + 10 = 14$
$g(v_0) = p_1$, $g(v_1) = p_2$	$8 + 1 = 9$	$8 + 1 + 2 = 11$
$g(v_0) = p_2$, $g(v_1) = p_1$	$2 + 4 = 6$	**2+4+2=8**

Ako se navedeni postupak raspoređivanja komunikacije primijeni na primjer 2 u pododjeljku 4.2.2 s vremenima izvođenja prikazanim tablicom 4.4, dobiva se raspored prikazan slikom 4.13. Stupci prikazuju vremensku crtu pripadajuće obradbene jedinice, dok svaki čvor pokazuje početno vrijeme, naziv raspoređenog entiteta (T je izvršna jedinica, $comm(i, j)$ označava komunikaciju među izvršnim jedinicama v_i i v_j) te vrijeme završetka. Ukupna duljina rasporeda $t^{rasp} = 11$.



Slika 4.13: Raspored primjera za RaNDKiP sa slike 4.5 (bez rasporeda komunikacije).

Ukoliko se za isti primjer svakoj izvršnoj jedinici dodijele pripadajuće komunikacijske rutine, duljina rasporeda se mijenja (slika 4.14), $t^{rasp} = 22,88$. Također, vidljivo je da se mijenja i sama struktura rasporeda. Sada je postupak rasporedio jedinicu programa `T3` na obradbenu jedinicu `P0` i `T4` na obradbenu jedinicu `P1` umjesto na `P2`.



Slika 4.14: Raspored primjera RaNDKiP sa slike 4.5 (s rasporedom komunikacije).

Raspored komunikacije vrlo je važan dio postupka RaNDKiP, jer se kod komunikacijski intenzivnih programa pravilnim raspoređivanjem može uštedjeti puno vremena pri izvođenju takvih programa. U eksperimentalnoj analizi će dodatno biti komentirana važnost stavljanja komunikacije u dobiveni raspored.

U sljedećem pododjeljku opisujemo posljednji dio postupka RaNDKiP, a to je skaliranje frekvencije za nekritične vremenske isječke, čime postupak RaNDKiP omogućuje uvođenje i dodatnog pokazatelja

kvalitete — utroška energije pri izvođenju programa.

4.2.4 Labavljenje vremenskih isječaka skaliranjem frekvencije

Osim performansi (većinom ukupno vrijeme izvođenja programa), jedan od najvažnijih kriterija optimizacije pri raspoređivanju programa u domeni računarstva visokih performansi je i potrošnja električne energije, što je i napomenuto u uvodnim pododjeljcima (primjerice 2.2.4). Kako bi se iskoristio potencijal postupaka raspoređivanja pomoću rangiranih listi izvršnih jedinica poput RaNDKiP-a (mala složenost i rješenja blizu optimalnima), uz istovremeno smanjenje utroška električne energije, navedene postupke moguće je kombinirati s nekima od postojećih metoda uštede energije. Neki od postupaka raspoređivanja na temelju liste s prilagodbama za uštedu energije su energetski svjesno raspoređivanje (eng. *energy-conscious scheduling*, ECS) [44], optimizacija zasnovana na koloniji mrava (eng. *ant colony optimization*, ACO) [34], labavljenje nekritičnih zadataka [85] te djelomično optimalno labavljenje (eng. *partial optimal slacking*, POS) [39]. Činjenica jest da gotovo sve današnje obradbene jedinice široke potrošnje (kakve se nalaze i u sustavima HPC) podržavaju dinamičko skaliranje frekvencije i napona obradbenih jedinica (eng. *Dynamic frequency and voltage scaling*, DVFS). Također, sama promjena frekvencije traje vrlo kratko ($23 - 50\mu$s za Intel IvyBridge arhitekturu, prema [83]) te nema velik utjecaj na vrijeme izvršavanja ako zrnatost programa nije prevelika (što je ispunjeno u primjeni). Kratko vrijeme promjene frekvencije ukazuje i na to da ni utrošak energije zbog promjene frekvencije nije značajan. Zbog navedenog, postupci raspoređivanja programa koji iskorištavaju mogućnosti DVFS tehnologije nameću se kao oni s velikim potencijalom povećanja performansi paralelnih programa, uz istovremenu uštedu energije pri njihovom izvođenju.

Jedan od takvih postupaka jest i naš unaprijeđeni postupak ***l*abavljenja *v*remenskih *i*sječaka (LVI)** koji smo razvili i upotrijebili za smanjenje frekvencije obradbenih jedinica pri izvođenju programa. Postupak LVI inspiriran je postupcima skaliranja napona za sve vremenske isječke rasporeda iz [85] i postupkom djelomičnog optimalnog labavljenja, prema [39]. Postupak LVI moguće je koristiti u sprezi sa svim postupcima raspoređivanja koji rezultiraju rasporedom u kojem se neka vremena izvršavanja jedinica programa mogu povećati bez utjecaja na ukupno trajanje programa. Za takve programe kažemo da sadrže **nekritične jedinice programa**. Isto tako, neki autori obilježavaju takvu sposobnost programa posebnom mjerom tzv. **labavosti** (eng. *slack*), tj. robusnosti rasporeda ili sposobnosti da se apsorbira labavost u rasporedu, primjerice u [13]. Važno je spomenuti da se navedeni postupak može koristiti tek nakon što je dobiven inicijalni vremenski raspored nekim od postojećih postupaka raspoređivanja.

Pod **vremenskim isječcima** smatramo sve vremenske intervale koje čine raspored svake pojedine obradbene jedinice sklopovlja (njezin vremenski slijed). Postoje tri glavna tipa isječaka:

- **obradbeni isječak** – vremenski interval u kojem se na obradbenoj jedinici izvršava jedinica programa (obradbeni dio rasporeda),
- **komunikacijski isječak** – vremenski interval u kojem se odvija komunikacija između dvije izvršne jedinice programa (pretpostavlja se da u to vrijeme obradbena jedinica ne troši energiju),
- **prazni isječak** – vremenski interval unutar kojeg ne traje ni izvršna jedinica programa, ni komu-

nikacija među njima (stanje čekanja obradbene jedinice).

Osnovni princip LVI-ja temelji se na postupku smanjenja radne frekvencije obradbenih jedinica kada se od njih ne zahtijeva puna obradbena snaga. To je ispunjeno u slučajevima kada traje komunikacija među izvršnim jedinicama programa, kada su obradbene jedinice u stanju pripravnosti, (na njima se ništa ne izvodi, "rupe" u rasporedu) te kada se na njima izvršavaju nekritične jedinice programa, tj. one o kojima ne ovisi ukupno vrijeme izvršavanja programa. LVI postupak se dijeli na četiri temeljna koraka, od kojih je svaki opisan zasebno. To su:

1. LVI korak 1: Skaliranje frekvencije na minimalnu prilikom trajanja komunikacijskih ili praznih isječaka, prema postupku iz [85], (**s**kaliranje **k**omunikacije i **p**ripravnosti — SKP).
2. LVI korak 2: Spajanje komunikacije i obrade podataka u jedinstvene isječke koji se tretiraju kao obradbeni (**s**tapanje **k**omunikacije i **o**brade — SKO).
3. LVI korak 3: Labavljenje vremenskih intervala obradbenih isječaka u kojima se izvršavaju nekritične jedinice programa, dijelom je zasnovano na postupku djelomičnog optimalnog labavljenja iz [39], (**di**namičko **l**abavljenje nekritičnih jedinica programa — DiL).
4. LVI korak 4: Vraćanje komunikacijskih isječaka u raspored nakon labavljenja vremenskih intervala obradbenih isječaka i ažuriranje njihovih vremenskih parametara, (**r**azdvajanje **k**omunikacije i **o**brade — RKO).

Algoritam 3 prikazuje cjelovit postupak LVI za svaku obradbenu jedinicu $p \in P$ (na kojima je napravljen raspored). Navedeni algoritam primjenjuje postupke SKP, SKO, DiL i RKO na vremenski raspored svake pojedine obradbene jedinice (linije 2–6).

Algoritam 3 Labavljenje vremenskih isječaka - LVI.

```
1: P ← Skup obradbenih jedinica
2: za sve p ∈ P radi
3:    Primijeni SKP
4:    Primijeni SKO
5:    Primijeni DiL
6:    Primijeni RKO
7: završi za
```

Smanjivanjem frekvencije obradbenih jedinica smanjuje se i njihov utrošak energije, što je opisano u pododjeljku 2.2.4. Ovisnost utroška energije o frekvenciji i naponu obično se daje DVFS karakteristikom obradbenih jedinica u kojoj su navedeni parovi napon – frekvencija, tj. pri kojem naponu možemo postaviti određenu frekvenciju takta.

U sljedećim pododjeljcima bit će opisan svaki od postupaka koji čine postupak LVI.

Postupak SKP

Prvi dio postupka LVI — postupak SKP — prikazan je algoritmom 4. Navedeni postupak provjerava vremenske isječke $s \in S$ rasporeda na obradbenoj jedinici p i postavlja njihovu radnu frekvenciju f^{op} na najnižu moguću f^{min} ako se radi o komunikaciji ili praznom isječku (linija 3). S_j označava skup svih vremenskih isječaka rasporeda na obradbenoj jedinici p_j, f_i^{op} označava radnu frekvenciju obradbene

jedinice na koju je raspoređena jedinica programa v_i koja se postavlja za vrijeme njezinog izvođenja[1], dok f_j^{min} označava minimalnu frekvenciju koju obradbena jedinica p_j može postići. Minimalna frekvencija obradbene jedinice određena je njezinim fizičkim sposobnostima i obično ju određuje proizvođač (može se dati u obliku DVFS karakteristike). S obzirom na to da većina obradbenih jedinica koje podržavaju DVFS automatski skaliraju frekvenciju na najnižu moguću kada ona nije potrebna, smatrat ćemo da kontrolu nad postupkom SKP ima operacijski sustav računalne platforme te da na njega ne moramo ručno utjecati. Samim time, prilikom izračuna uštede energije uzet će se u obzir samo obradbeni isječci, jer je to isključivi pokazatelj dobitaka postupka LVI prema definicijama tipova vremenskih isječaka koje smo naveli ranije, u prethodnom pododjeljku.

Algoritam 4 Skaliranje komunikacije i pripravnosti - SKP.

1: $S_j \leftarrow$ Skup vremenskih isječaka obradbene jedinice j
2: $f_i^{op} \leftarrow$ Operativna frekvencija vremenskog isječka i
3: $f_j^{min} \leftarrow$ Minimalna moguća frekvencija obradbene jedinice j
4: **za sve** $s_k \in S_j$ **radi**
5: **ako** s_k je komunikacijski isječak **ili** s_k je prazni isječak **onda**
6: $f_k^{op} = f_j^{min}$
7: **završi ako**
8: **završi za**

Definiramo obradbeni trag $w(v_i)$ koji ostavlja isječak obradbene jedinice v_i (prema [37]) kao umnožak njegove radne frekvencije f_i^{op} i vremena izvođenja t_i^{izv}:

$$w(v_i) = f_i^{op} \cdot t_i^{izv}. \tag{4.27}$$

Za potrebe opisa postupka LVI frekvencije se označavaju u simboličkom intervalu $[f^{min}, 1]$, što bi značilo da je maksimalna frekvencija f^{max} jednaka 1 u slučaju da se pri trajanju vremenskog isječka obradbena jedinica koristi punom obradbenom snagom. Zbog toga, pri maksimalnoj frekvenciji prethodni izraz postaje:

$$w(v_i) = f_i^{max} \cdot t_i^{izv} = t_i^{izv}, \tag{4.28}$$

što čini

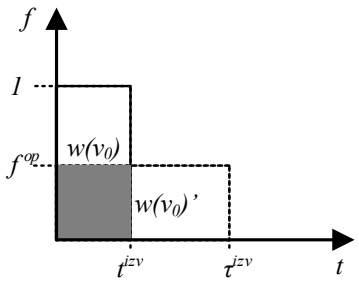
$$t_i^{izv} \sim w(v_i), \tag{4.29}$$

tako da ćemo trag koji ostavlja isječak smatrati ekvivalentnim njegovom vremenu izvršavanja, a radnu frekvenciju koeficijentom povećanja trajanja vremenskog isječka. Navedeno je i prikazano slikom 4.15, gdje je vidljivo da se frekvencija smanjuje nauštrb povećanja vremena izvođenja, dakle za $f = f^{op}$ slijedi $t^{izv} = \tau^{izv}$. Također vrijedi:

$$t^{izv} = f^{op} \cdot \tau^{izv}. \tag{4.30}$$

Nakon postupka SKP slijedi SKO, opisan u idućem pododjeljku.

[1] Zbog jednostavnosti koristimo naziv operativna frekvencija vremenskog isječka, iako se frekvencija smanjuje obradbenoj jedinici.



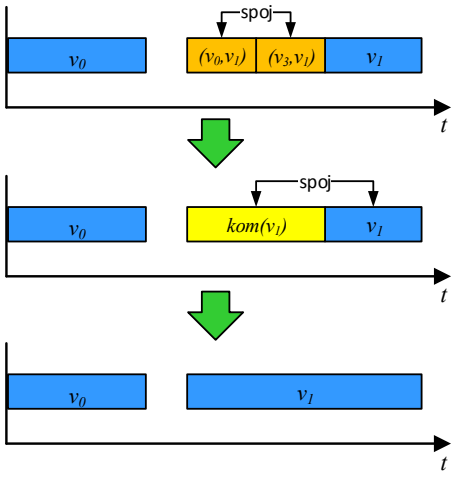
Slika 4.15: Primjer skaliranja frekvencije za obradbeni isječak.

Postupak SKO

Nakon postupka SKP, a prije nego se primijeni sljedeći korak — postupak DiL, u rasporedu se komunikacijski odsječci spajaju s pripadajućim obradbenim odsječcima (jedinicama programa). Razlog tomu je dvojak:

1. Za vrijeme komunikacije frekvencija se skalira na minimalnu jer se pretpostavlja da obradbena jedinica ne sudjeluje izravno u njoj, tako da dalje skaliranje frekvencije za nju nije potrebno (izvedeno je u prethodnom koraku, SKP-u).
2. Potpuno izuzimanje komunikacije iz rasporeda (njezino uklanjanje) značilo bi uvođenje krivih pretpostavki o stvarnoj vremenskoj ovisnosti među obradbenim jedinicama programa. Primjerice, labavljenje isječaka u kojima se izvršavaju jedinice programa odvijalo bi se preko intervala komunikacijskih isječaka, vodeći do neispravnog rasporeda.

U sljedećem dijelu postupka LVI, komunikacijski isječci prvo se spajaju sa svim komunikacijskim isječcima neposredno nakon njega te naposljetku s obradbenim isječkom koji slijedi. Međusobno spajanje uzastopnih komunikacijskih odsječaka vezano je uz činjenicu da se u rasporedu komunikacije svi komunikacijski isječci nekog obradbenog isječka stavljaju u niz ispred njega. Navedeni postupak nazivamo stapanje komunikacije i obrade — SKO. Spomenuti postupak ilustriran je slikom 4.16, na kojoj su prikazana dva obradbena isječka v_0 i v_1 te 2 komunikacijska isječka (v_0, v_1) i (v_3, v_1).



Slika 4.16: Postupak SKO.

Postupak prvo spaja dva uzastopna komunikacijska isječka (v_0, v_1) i (v_3, v_1), a zatim ukupan obujam (interval) komunikacije spaja s pripadajućim obradbenim isječkom v_1.

Detalji postupka SKO prikazani su algoritmom 5. Algoritam za svaki isječak provjerava radi li se o komunikacijskom isječku (linija 4), ako da, postavlja zastavicu *kom* na *true* i sprema vrijeme početka te izvršne jedinice t_i^{svp} (stvarno vrijeme početka) u t (linije 5 i 6). Navedeno vrijeme bit će upotrijebljeno kao vrijeme početka izvršne jedinice koja slijedi nakon komunikacije (linija 13). Nakon toga, postupak prolazi kroz sljedeće isječke sve dok ne dođe do onog koji označava obradu, tj. izvršavanje jedinice programa (linije 7–9) te odabire njega za dalju analizu (linija 10). Za obradbeni isječak algoritam provjerava nije li prije njega uslijedila komunikacija (zastavica *kom* je *true*), ako jest, postavlja vrijeme početka isječka obrade t_i^{svp} u vrijeme početka prvog odgovarajućeg komunikacijskoga odsječka t (linija 13). Izmjenom vremena početka mijenja se i vrijeme izvršavanja t_i^{izv} koje postaje jednako razlici stvarnog vremena završetka odsječka t_i^{svz} i stvarnog vremena početka t_i^{svp} isječka (linija 14). Naposljetku, zastavica *kom* ponovno se postavlja na *false* i kreće se s analizom sljedećeg isječka (linija 17).

Algoritam 5 Stapanje komunikacije i obrade - SKO.

```
1:  t^svp ← Stvarno vrijeme početka vremenskog isječka i
2:  za sve s_i ∈ S_j radi
3:      kom = false
4:      ako s_i je komunikacijski isječak onda
5:          kom = true
6:          t = t_i^svp
7:          ako s_{i+1} je komunikacijski isječak onda
8:              i = i + 1
9:          završi ako
10:         i = i + 1
11:     završi ako
12:     ako kom = true onda
13:         t_i^svp = t
14:         t_i^izv = t_i^svz − t_i^svp
15:         kom = false
16:     završi ako
17:     i = i + 1
18: završi za
```

Postupak SKO nužan je preduvjet za sljedeći, postupak DiL, kojeg opisujemo u narednom pododjeljku. Naime, na taj način zadržavamo konzistentnost rasporeda kako se labavljenje ne bi odvijalo nauštrb komunikacijskih isječaka, što bi dovelo do neizvedivog rasporeda.

Postupak DiL

Treći korak postupka LVI, ujedno i najvažniji, je postupak DiL koji služi za skaliranje radne frekvencije svih nekritičnih isječaka rasporeda. Kao što je već spomenuto, postupak je zasnovan na djelomičnom optimalnom labavljenju vremenskih isječaka prema [39]. Definiramo početne koncepte potrebne za opis postupka DiL.

Neposredna okolina izvršne jedinice programa $L(v_i)$ je skup njezinih neposrednih prethodnika i

sljedbenika, tj. izvršnih jedinica programa s kojima su spojene bridom u grafu:

$$L(v_i) \quad = \quad pred(v_i) \cup sljed(v_i), \tag{4.31}$$

gdje su:

$$pred(v_i) \quad = \quad \{(v_s) : \exists(v_s, v_i)\} \tag{4.32}$$

$$sljed(v_i) \quad = \quad \{(v_s) : \exists(v_i, v_s)\}. \tag{4.33}$$

Za svaku se izvršnu jedinicu v_i definiraju njezino **najranije vrijeme početka** t_i^{nvp} i **najkasnije vrijeme završetka** t_i^{nvz} kao:

$$t_i^{nvp} \quad = \quad \max_{j \in pred(v_i)} \left(t_j^{nvp} + t_j^{izv}\right) \tag{4.34}$$

$$t_i^{nvz} \quad = \quad \min_{j \in sljed(v_i)} \left(t_j^{nvz} - t_j^{izv}\right), \tag{4.35}$$

gdje je t_j^{izv} **vrijeme izvršavanja** jedinice programa v_j (trajanje njezinog vremenskog isječka u rasporedu). Drugim riječima, t^{nvp} jest vrijeme najranijeg početka rada izvršne jedinice kada bi bio ispunjen uvjet da su svi njezini prethodnici započeli u njihovom najranijem vremenu početka. Najkasnije vrijeme završetka, s druge strane, jest vrijeme u koje bi izvršna jedinica završila s radom kada bi svi njezini sljedbenici završili s radom u njihovim najkasnijim vremenima završetka. Vremenski isječak može imati i vremenski interval unutar kojega može povećati vrijeme izvođenja bez narušavanja vremenske ovisnosti izvršnih jedinica programa — **prostor labavljenja** (eng. *slack room*):

$$t^{pl} \quad = \quad t^{nvz} - t^{nvp} - t^{izv}. \tag{4.36}$$

Na početku postupka DiL za vremena t^{nvp}, t^{nvz} i t^{izv} postavljaju se vrijednosti prema:

$$t^{nvp} \quad = \quad t^{svp} \tag{4.37}$$

$$t^{nvz} \quad = \quad t^{svz} \tag{4.38}$$

$$t^{izv} \quad = \quad t^{svz} - t^{svp}, \tag{4.39}$$

gdje su t^{svp} i t^{svz} stvarno vrijeme početka i stvarno vrijeme završetka, koji su poznati iz ulaznog rasporeda.

Sljedeći je korak izrada **skupa isječaka s preklapajućim prostorom labavljenja** $Q(p_j)$, drugim riječima, onih kojima je najkasnije vrijeme završetka veće ili jednako najranijem vremenu početka sljedećeg isječka:

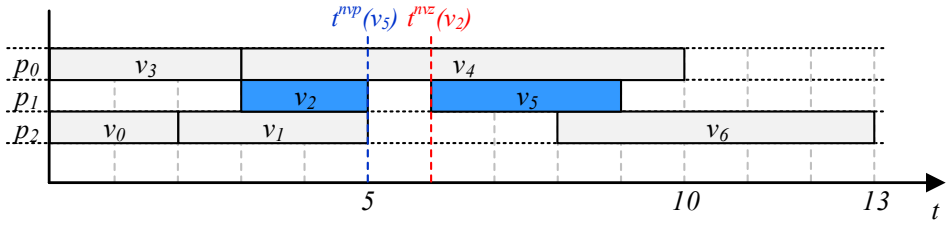
$$Q(p_j) = \left\{(v_i) : \; t_i^{nvz} \geq t_{i+1}^{nvp}\right\}. \tag{4.40}$$

Razlog tomu jest optimizacijske naravi, tj. kako postupak ne bi razmatrao isječke kojima se ionako

ne može mijenjati vrijeme izvršavanja (**kritični isječci**). Zajedničko svojstvo svih isječaka članova skupa $Q(p_j)$ jest da se njima može povećati vrijeme izvršavanja bez utjecaja na vremensku ovisnost u programu i bez produžavanja ukupnog vremena izvođenja programa. Njih nazivamo **nekritičnim isječcima**. Primjer slučaja preklapanja prostora labavljenja prikazan je na slici 4.17. Na spomenutoj slici vidi se da je za isječke v_2 i v_5 zadovoljen uvjet (4.40), jer v_5 može početi ranije (u $t_5^{nvp} = 5$), dok v_2 može završiti kasnije (u $t_2^{nvz} = 6$). Ukoliko uvjet (4.40) nije zadovoljen, isječak v_2 se ne razmatra, tj. ne dodaje se u skup isječaka s preklapajućim prostorom labavljenja $Q(p_j)$. Specijalan slučaj jest posljednji isječak. Poslije zadnjeg isječka ne postoji više ni jedan isječak, tako da uvjet (4.40) nije moguće provjeriti (vrijednost t_{i+1}^{nvp} nije poznata). U tom slučaju na kraj rasporeda postavlja se privremeni isječak s intervalom $[t^{rasp}, t^{rasp}]$ te se dobiva:

$$t_{i+1}^{nvp} = t^{rasp}, \tag{4.41}$$

gdje je t^{rasp} **vrijeme završetka programa (duljina rasporeda)** i zatim se provjerava uvjet (4.40) zbog odlučivanja hoće li se posljednji (pravi) isječak staviti u $Q(p_j)$ ili ne.



Slika 4.17: Postupak DiL: primjer preklapajućeg prostora labavljenja.

Sljedeći je korak izračun **ukupnog trajanja isječaka** unutar $Q(p_j)$ koji je jednak zbroju njihovih vremena izvršavanja:

$$t^{izv}\left[Q(p_j)\right] = \sum_{v_i \in Q(p_j)} t_i^{izv} \tag{4.42}$$

i njegovog **intervala protezanja** kao vremena od najranijeg vremena početka prvog elementa do najkasnijeg vremena završetka posljednjeg:

$$t^{int}\left[Q(p_j)\right] = t_{q-1}^{nvz} - t_0^{nvp}, \tag{4.43}$$

gdje je q broj elemenata skupa Q. Navedene vrijednosti potrebne su kako bi se dobila **radna frekvencija obradbene jedinice** p_j.

Radna frekvencija f_j^{op} obradbene jedinice p_j za njezin cijeli raspored (vremenski slijed) definirana je kao:

$$f_j^{op} = f_j^{max} \cdot \frac{t^{izv}\left[Q(p_j)\right]}{t^{int}\left[Q(p_j)\right]}, \tag{4.44}$$

a s obzirom na to da se radna frekvencija obradbene jedinice simbolički definira u intervalu $\left[f^{min}, 1\right]$,

gdje je 1 maksimalna frekvencija, (4.44) postaje:

$$f_j^{op} = \frac{t^{izv}\left[Q(p_j)\right]}{t^{int}\left[Q(p_j)\right]}. \tag{4.45}$$

Nakon toga za svaki se vremenski isječak definira njegova **radna frekvencija isječka** (radna frekvencija obradbene jedinice dok izvršava taj vremenski isječak), ovisno o tome u kakvom su odnosu vremenski parametri tih isječaka (kao i jedinice programa koje se izvršavaju unutar njih). Postoje tri moguća slučaja, kao i pripadajući izrazi za radnu frekvenciju isječka.

1. **U prvom slučaju**, vrijeme izvršavanja isječka v_i nakon skaliranja frekvencije obradbene jedinice na kojoj se izvršava (p_j) na f_j^{op}, može se dodatno povećati:

$$t_i^{izv} < f_j^{op} \cdot \left(t_{i+1}^{nvp} - t_i^{nvp}\right) \tag{4.46}$$

   te radna frekvencija isječka v_i postaje:

$$f_i^{op} = \frac{t_i^{izv}}{t_{i+1}^{nvp} - t_i^{nvp}}. \tag{4.47}$$

2. **U drugom slučaju**, kada je vrijeme izvršavanja isječka v_i skalirano na radnu frekvenciju obradbene jedinice veće ili jednako od razlike njegovog početka i početka sljedećeg isječka:

$$f_j^{op} \cdot \left(t_{i+1}^{nvp} - t_i^{nvp}\right) \quad \leq \quad t_i^{izv} \leq f_j^{op} \cdot (t_i^{nvz} - t_i^{nvp}) \tag{4.48}$$

   tada isječci v_i i v_{i+1} mogu raditi na radnoj frekvenciji f_j^{op}:

$$f_i^{op} \quad = \quad f_j^{op} = \frac{t^{izv}\left[Q(p_j)\right]}{t^{int}\left[Q(p_j)\right]}. \tag{4.49}$$

3. **U trećem slučaju**, kada je vrijeme izvršavanja isječka v_i veće od skaliranog vremena izvršavanja:

$$t_i^{izv} \quad > \quad f_j^{op} \cdot (t_i^{nvz} - t_i^{nvp}) \tag{4.50}$$

   isječak v_i produljuje vlastito vrijeme izvođenja sve do t_i^{nvz}:

$$f_i^{op} \quad = \quad \frac{t_i^{izv}}{t_i^{nvz} - t_i^{nvp}} \tag{4.51}$$

Kada se promijeni radna frekvencija isječka v_i, mijenja se i njegovo vrijeme izvođenja:

$$\tau_i^{izv} = \frac{t_i^{izv}}{f_i^{op}}. \tag{4.52}$$

Algoritam 6 prikazuje cjelokupni postupak DiL. Postupak započinje tako da postavi vremena t_i^{nvp}, t_i^{nvz} i t_i^{izv} na njihove stvarne vrijednosti iz rasporeda (linije 1–5). Zatim se ponovno za svaki vremenski isječak ažuriraju vremena t_i^{nvp} i t_i^{nvz} prema njihovoj neposrednoj okolini $L(v_i)$ (linije 6–8), kako bi se

provjerilo postoji li mogući prostor labavljenja njihovog vremenskog intervala. Nakon ažuriranja intervala svakog isječka algoritam stvara skup isječaka s preklapajućim prostorom labavljenja $Q(p_j)$ obradbene jedinice p_j (linije 9–13) provjeravajući uvjet (4.40) (linija 10). Sljedeći je korak računanje vrijednosti potrebne za dobivanje radne frekvencije f^{op} obradbene jedinice (linija 14) i u slučaju da je izračunata radna frekvencija manja od minimalne koju sklopovlje može pružiti, ona se postavlja na minimalnu vrijednost (linije 15–17). Tek tada, algoritam za svaki nekritični isječak, član skupa $Q(p_j)$, provjerava u koji od intervala (4.46), (4.48) ili (4.50) se svrstava njegovo vrijeme izvršavanja te prema tome određuje radnu frekvenciju isječka prema (4.47), (4.49) i (4.51), (linija 19 i 20). Ovdje algoritam opet provjerava da li je izračunata radna frekvencija isječka manja od minimalne koju sklopovlje može pružiti i ako jest, ona se postavlja na minimalnu vrijednost (linije 21–23). Sljedeći korak je ažuriranje intervala izvršavanja isječka (linija 25) nakon računanja novog vremena izvršavanja (linija 24). Zatim se ažurira najranije vrijeme početka njegovih sljedbenika u grafu programa, (linije 26–28) i sljedećeg isječka u rasporedu, osim ako on nije posljednji (linije 23–25). Ažuriranje vremena početka svih sljedbenika iz grafa programa, kao i isječka koji slijedi nakon njega nužan je korak, jer novonastala promjena može uzrokovati promjenu vremenske ovisnosti izvršnih jedinica programa u rasporedu.

Algoritam 6 Dinamičko labavljenje nekritičnih jedinica programa - DiL.

1: **za sve** $s_i \in S$ **radi**
2: $t_i^{nvp} \leftarrow t_i^{svp}$ (4.37)
3: $t_i^{nvz} \leftarrow t_i^{svz}$ (4.38)
4: $t_i^{izv} \leftarrow t_i^{svz} - t_i^{svp}$ (4.38)
5: **završi za**
6: **za sve** $s_i \in S$ **radi**
7: izračunaj t_i^{nvp} i t_i^{nvz}
8: **završi za**
9: **za sve** $s_i \in S$ **radi**
10: **ako** $t_i^{nvz} \geq t_{i+1}^{nvp}$ **onda**
11: dodaj v_i u $Q(p_j)$
12: **završi ako**
13: **završi za**
14: izračunaj $t^{izv}\,[Q(p_j)]$ i $t^{int}\,[Q(p_j)]$ te $f^{op}(p_j)$
15: **ako** $f^{op} < f^{min}$ **onda**
16: $f^{op} = f^{min}$
17: **završi ako**
18: **za sve** $v_i \in Q(p_j)$ **radi**
19: provjeri uvjete (4.46), (4.48) i (4.50)
20: postavi f_i^{op} prema (4.47), (4.49) i (4.51)
21: **ako** $f_i^{op} < f^{min}$ **onda**
22: $f_i^{op} = f^{min}$
23: **završi ako**
24: ažuriraj t_i^{izv} prema (4.52)
25: ažuriraj interval izvršavanja od v_i na $\left[t_i^{nvp}, t_i^{nvp} + \tau_i^{izv}\right]$
26: **za sve** $v_k \in sljed(v_i)$ **radi**
27: ažuriraj t_k^{nvp}
28: **završi za**
29: **ako** v_i nije posljednji isječak u rasporedu **onda**
30: ažuriraj t_{i+1}^{nvp}
31: **završi ako**
32: **završi za**

Nakon postupka DiL skalirane su frekvencije svih nekritičnih isječaka rasporeda, ako ih je bilo, no ti isječci su sadržavali i komunikaciju, (postupak SKO) koju je sada potrebno izdvojiti. Spomenuto se odvija putem postupka RKO kojeg opisujemo u predstojećem pododjeljku.

Postupak RKO

Posljednji korak postupka LVI jest postupak razdvajanja komunikacije i obrade — RKO, koji preslikava raspored skaliranih obradbenih isječaka, (spojenih s odgovarajućim komunikacijskim isječcima postupkom SKO) natrag u izvorni raspored. Naime, raspored dobiven nakon završetka postupka DiL (skup njegovih vremenskih isječaka S_j za obradbenu jedinicu p_j nakon postupka SKO) ne uključuje i komunikaciju prisutnu u početnom rasporedu (skup njegovih vremenskih isječaka S'_j iz polaznog rasporeda). To može uzrokovati nekonzistentnosti prilikom izvođenja takvog programa na stvarnoj platformi, ali i nekonzistentnosti analize koliko je zapravo smanjen utrošak energije pri izvođenju programa, koliko su stvarno skalirana vremena obradbenih isječaka te kolika je njihova radna frekvencija. Algoritam 7 prikazuje detalje postupka RKO.

Algoritam 7 Razdvajanje komunikacije i obrade - RKO.

$S_j \leftarrow$ Skup svih vremenskih isječaka u rasporedu nakon postupka DiL
$S'_j \leftarrow$ Skup svih vremenskih isječaka u polaznom rasporedu prije početka postupka LVI
za sve $s'_i \in S'_j$ **radi**
  ako s'_i je komunikacijski isječak **onda**
    $s'_p \leftarrow s'_i$ {Prvi komunikacijski isječak}
    $t^{izv}_{\Psi_k} = 0$ {ukupno trajanje komunikacijskih isječaka}
    dok s'_{i+1} nije obradbeni isječak **radi**
      $t^{izv}_{\Psi_k} = t^{izv}_{\Psi_k} + t^{izv\prime}_i$
      $\Psi_k \leftarrow s'_i$
      $i = i + 1$
    završi dok
    $s'_k \leftarrow s'_i$ {pronađeni obradbeni isječak u S'_j}
    $s_k \leftarrow$ odgovarajući obradbeni odsječak u S_j
    $t^{nvp\prime}_p = t^{nvp}_k$
    za sve $s'_r \in \Psi_k$ **radi**
      $t^{nvp\prime}_{r+1} = t^{nvz\prime}_r$ {svi komunikacijski isječci od s'_k}
      $t^{nvz\prime}_{r+1} = t^{nvp\prime}_{r+1} + t^{izv\,\prime}_{r+1}$
    završi za
    $t^{nvp}_k = t^{nvz}_k - \left(t^{izv}_k - t^{izv}_{\Psi_k}\right)$
    $t^{izv}_k = t^{nvz}_k - t^{nvp}_k$
  završi ako
završi za
za sve $s'_i \in S'_j$ **radi**
  ako s'_i je obradbeni odsječak **onda**
    kopiraj vrijednosti iz s_i u s'_i
    $f^{op\prime}_i = t^{izv\prime}_i / t^{izv}_i$
  završi ako
završi za

Postupak započinje traženjem svih komunikacijskih isječaka $s \in \Psi_k,\ \Psi_k \subset S'_j$ odgovarajućeg obradbenog isječka s'_k u nizu prije nailaska na taj obradbeni isječak u polaznom rasporedu S'_j te zbraja njihova vremena izvođenja $t^{izv}(s),\ s \in \Psi_k$. Na taj način dobiva se ukupno trajanje komunikacije $t^{izv}_{\Psi_k}$ obradbenoga

isječka s'_k:

$$t^{izv}_{\Psi_k} = \sum_{s \in \Psi_k} t^{izv}(s), \tag{4.53}$$

potrebno za računanje vremena izvršavanja $t^{izv\prime}_k$ tog isječka prema:

$$t^{izv\prime}_k = \tau^{izv}_k - t^{izv}_{\Psi_k}. \tag{4.54}$$

Nakon toga, algoritam ažurira vrijeme početka $t^{nvp\prime}_p$ prvog komunikacijskoga isječka s'_p odgovarajućeg obradbenog odsječka s'_k starog rasporeda S'_j u vrijeme početka t^{nvp}_k odgovarajućeg obradbenog isječka s_k u novom rasporedu S_j:

$$t^{nvp\prime}_p = t^{nvp}_k. \tag{4.55}$$

Time se određuje da je stvarni početak obradbenog isječka s'_k u trenutku završetka njegove komunikacije. Nakon toga, ako postoji još ulančanih komunikacijskih odsječaka odmah nakon prvog, ažuriraju se i njihova vremena početka $t^{nvp\prime}_{r+1}$ i završetka $t^{nvz\prime}_{r+1}$ u odnosu na vrijeme početka prethodnog u nizu $t^{nvp\prime}_r$ (slažu se jedan za drugim nakon prvog, redom iz početnog rasporeda S'_j):

$$t^{nvp}_{r+1} = t^{nvp}_r, \tag{4.56}$$

$$t^{nvz}_{r+1} = t^{nvp}_r + t^{izv}_r. \tag{4.57}$$

Sada se može ažurirati vrijeme početka odgovarajućeg obradbenog isječka u novom rasporedu prema završetku posljednjeg komunikacijskoga isječka u nizu, kako bi sva vremena u S_j bila ažurna prije njihovog kopiranja u početni raspored S'_j:

$$t^{nvp}_k = t^{nvz}_k - \left(t^{izv}_k - t^{izv}_{\Psi_k}\right) \tag{4.58}$$

te vrijeme izvođenja:

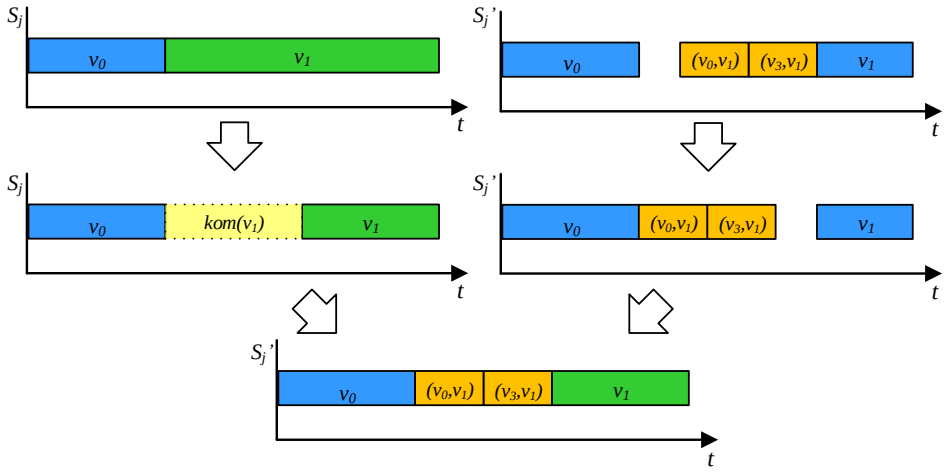
$$t^{izv}_k = t^{nvz}_k - t^{nvp}_k. \tag{4.59}$$

Pretposljednji je korak ažuriranje vremena svih obradbenih isječaka iz polaznog rasporeda $s' \in S'_j$ kopiranjem njihovih parametara iz novog rasporeda S_j. Na kraju je potrebno ažurirati radnu frekvenciju isječaka u S'_j pomoću:

$$f^{op\prime}_k = \frac{t^{izv\prime}_k}{t^{izv}_k}. \tag{4.60}$$

Postupak RKO prikazan je i slikom 4.18. Na lijevoj strani prikazan je novi raspored S_j u kojemu se od skaliranog vremena izvršavanja t^{izv}_1 obradbenoga isječka v_1 oduzima ukupno vrijeme komunikacije svih njegovih komunikacijskih isječaka prema (4.58). Na desnoj strani, u početnom se rasporedu S'_j pomiču

komunikacijski isječci s početkom u t_k^{nvp}. Nakon toga, iz novog se rasporeda S_j prebacuju parametri obradbenog isječka v_1 u početni raspored S'_j.



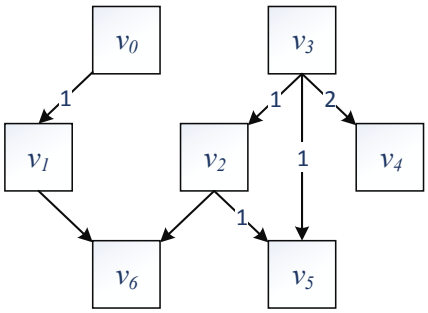
Slika 4.18: Primjer postupka RKO.

Kao što je već opisano, postupak LVI omogućuje smanjivanje frekvencije obradbenih jedinica za sve isječke za koje je to moguće, time smanjujući ukupni utrošak energije pri izvođenju programa. Navedeni postupak prikazan je i primjerom 3 u sljedećem pododjeljku, gdje je na primjeru programa i njegovog rasporeda opisano provođenje postupaka SKP, SKO, DiL i RKO te rezultati u obliku predviđene količine uštođene energije upotrebom istih.

Primjer 3: izvođenje postupka LVI

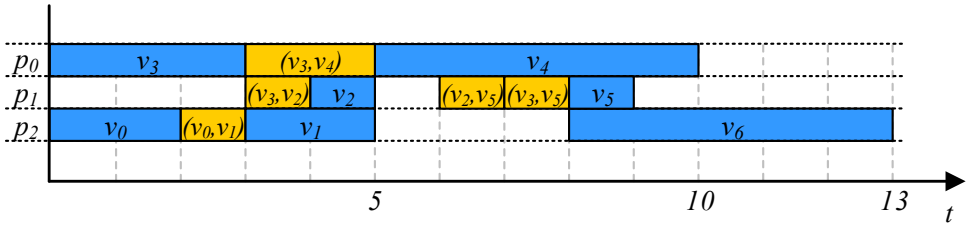
U ovom pododjeljku predstavljamo primjer kojim ćemo opisati postupak LVI s naglaskom na njegov treći korak — postupak DiL.

Zadan je program predstavljen usmjerenim acikličkim grafom prikazanim na slici 4.19.



Slika 4.19: Primjer postupka LVI: program.

Pretpostavljamo da je upotrebom nekog od postupaka raspoređivanja, (poput postupka RaNDKiP) dobiven raspored izvršavanja programa prikazan na slici 4.20.

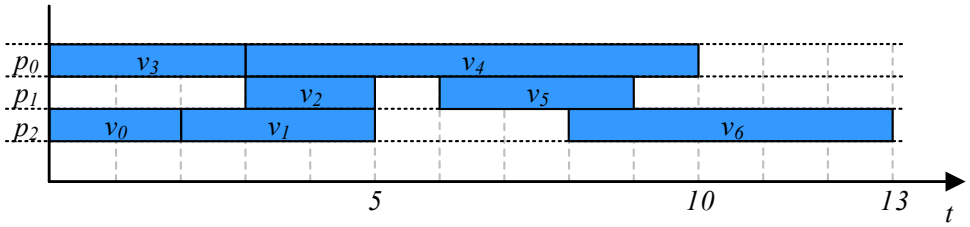


Slika 4.20: Primjer postupka LVI: dobiveni raspored programa.

Iz iste je slike također vidljivo da su izvršne jedinice raspoređene na računalnu platformu koja se sastoji od 3 obradbene jedinice $p_0, ..., p_2$, čiji je DVFS sustav jednak i može skalirati frekvenciju kontinuirano između $f^{min} = 0$ i $f^{max} = 1$. Ostali parametri platforme i njezinih obradbenih jedinica nisu bitni za provedbu ovog primjera. Kako raspored sadržava ne samo obradbene vremenske isječke $(v_0, ..., v_6)$, već i komunikacijske:

$$\{(v_0, v_1), (v_3, v_2), (v_3, v_4), (v_2, v_5), (v_3, v_5)\}, \tag{4.61}$$

prema slici 4.20, potrebno je provesti postupak SKO da bi se komunikacijski isječci spojili s odgovarajućim obradbenim isječcima prema algoritmu 5 (tj. da bi se njihovo vrijeme izvršavanja dodalo vremenu izvršavanja jedinica programa), čime se dobiva raspored prikazan na slici 4.21.



Slika 4.21: Primjer postupka LVI: raspored programa nakon postupka SKO.

Vremenski parametri isječaka prema početnom rasporedu iz primjera prikazani su tablicom 4.7, gdje su početni vremenski parametri isječaka prikazani u tablici 4.7a, dok su parametri nakon ažuriranja prikazani u tablici 4.7b. U tablici 4.7b su izmijenjeni parametri označeni crvenom bojom. Nakon što su dobivena vremena iz tablice 4.7, postupak DiL odabire vremenske isječke s preklapajućim prostorom labavljenja svake obradbene jedinice i sprema ih u skupove $Q(p_0)$, $Q(p_1)$ i $Q(p_2)$.

Tablica 4.7: Parametri vremenskih isječaka.

(a) Iz početnog rasporeda.

Isječak	t_i^{izv}	t_i^{nvp}	t_i^{nvz}	t_i^{pl}	f^{op}
v_0	2	0	2	0	1
v_1	3	2	5	0	1
v_2	2	3	5	0	1
v_3	3	0	3	0	1
v_4	7	3	10	0	1
v_5	3	6	9	0	1
v_6	5	8	13	0	1

(b) Nakon ažuriranja, prije skaliranja.

Isječak	t_i^{izv}	t_i^{nvp}	t_i^{nvz}	t_i^{pl}	f^{op}
v_0	2	0	2	0	1
v_1	3	2	8	3	1
v_2	2	3	6	1	1
v_3	3	0	3	0	1
v_4	7	3	10	0	1
v_5	3	5	9	1	1
v_6	5	5	13	3	1

U tablici 4.8 prikazani su isječci koji pripadaju navedenim skupovima, kao i radna frekvencija pojedinih obradbenih jedinica. Vidljivo je da svi isječci ispunjavaju uvjet (4.40), tako da ih sustav sve odabire za labavljenje intervala. Također, skalirane su i frekvencije svih obradbenih jedinica osim p_0 na vrijednosti prikazane tablicom 4.8.

Tablica 4.8: Parametri rasporeda za svaku obradbenu jedinicu nakon postupka LVI.

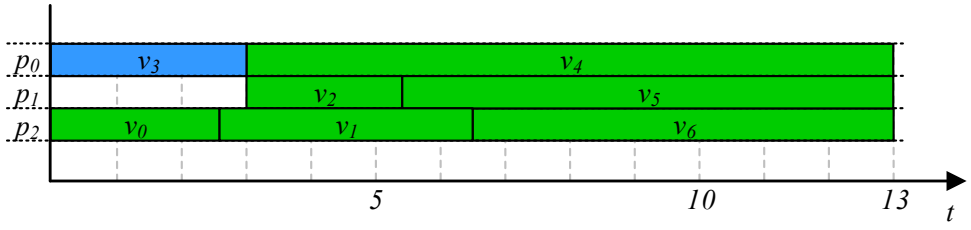
Obradbena jedinica	$Q(p)$	$f^{op}(p)$
p_0	$\{v_3,\ v_4\}$	1,00
p_1	$\{v_2,\ v_5\}$	0,83
p_2	$\{v_0,\ v_1,\ v_6\}$	0,77

Nakon što postupak DiL analizira sve vremenske isječke, dobivaju se rezultati iz tablice 4.9. U navedenoj se tablici vidi da su skalirani svi isječci osim v_3, jer je on prethodnik čak triju isječaka (v_2, v_4 i v_5), prema slici 4.19, tako da je njegov raniji završetak uvjet ranog početka velikog dijela programa.

Tablica 4.9: Parametri vremenskih isječaka nakon postupaka SKP, SKO i DiL.

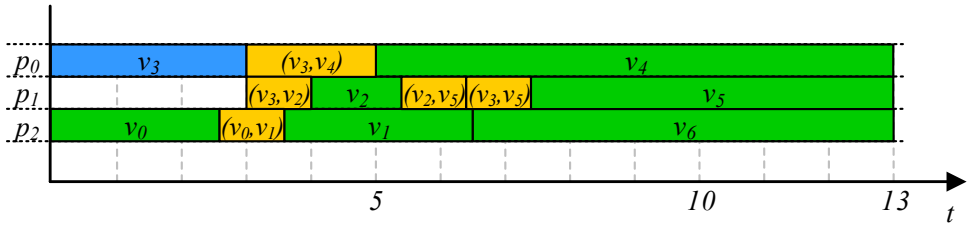
Isječak	τ_i^{izv}	t_i^{nvp}	t_i^{nvz}	t_i^{pl}	f^{op}
v_0	2,60	0,00	2,60	0	0,77
v_1	3,90	2,60	6,50	0	0,77
v_2	2,40	3,00	5,40	0	0,83
v_3	3,00	0,00	3,00	0	1,00
v_4	10,00	3,00	13,00	0	0,70
v_5	7,60	5,40	13,00	0	0,39
v_6	6,50	6,50	13,00	0	0,77

Pripadajući raspored sada je izmijenjen i prikazan je na slici 4.22. Iz navedene slike očito je da je postupak LVI gotovo u potpunosti ispunio vremenski raspored obradbenih jedinica (iznimka je prostor između trenutaka 0 i 3 na obradbenoj jedinici p_1), što navodi na to da je platforma uniformnije opterećena s manjim vršnim frekvencijama, što će zasigurno dovesti do smanjenja utroška energije.



Slika 4.22: Primjer postupka LVI: raspored programa nakon postupka DiL.

Nakon što je dobiven raspored s olabavljenim vremenima isječaka obradbenih jedinica, potrebno je ponovno uzeti u obzir komunikacijske isječke (koji su uklonjeni postupkom SKO). Samim time, vrijeme izvršavanja τ^{izv}, najranije vrijeme početka t^{nvp} i radna frekvencija f^{op} svih obradbenih isječaka koji komuniciraju moraju biti ažurirani.



Slika 4.23: Primjer postupka LVI: raspored programa nakon postupka RKO.

S obzirom na to da takve obradbene jedinice ne počinju u t^{nvp}, već kada završe svi njegovi komunikacijski isječci, primjenjuje se postupak RKO prema algoritmu 7. Nakon primjene navedenog postupka raspored je prikazan slikom 4.23.

Poslije završetka postupka LVI (i vraćanja komunikacijskih isječaka postupkom RKO) dobiveni su parametri vremenskih isječaka prikazani tablicom 4.10.

Tablica 4.10: Parametri vremenskih isječaka nakon cjelokupnog postupka LVI.

Isječak	τ_i^{izv}	t_i^{nvp}	t_i^{nvz}	t_i^{pl}	f^{op}
v_0	2,60	0,00	2,60	0	0,77
v_1	2,90	3,60	6,50	0	0,69
v_2	1,40	4,00	5,40	0	0,71
v_3	3,00	0,00	3,00	0	1,00
v_4	8,00	5,00	13,00	0	0,63
v_5	5,60	7,40	13,00	0	0,18
v_6	6,50	6,50	13,00	0	0,77

U navedenoj tablici vidljivo je da su frekvencije dodatno izmijenjene, jer je došlo do skraćivanja vremena izvršavanja τ^{izv} svih obradbenih isječaka koji imaju pripadajuće komunikacijske isječke. Također, ažurirana su im i najranija vremena početka t^{est}.

Utrošak energije

Ukupni utrošak energije prilikom izvođenja programa jednak je zbroju njezine statičke i dinamičke komponente, prema [3, 83]:

$$\varepsilon^{uk} = \varepsilon^{din} + \varepsilon^{stat}, \tag{4.62}$$

dok je dinamička energija jednaka umnošku dinamičke snage φ^{din} i vremenskog intervala u kojem djeluje Δt:

$$\varepsilon^{din} = \varphi^{din} \cdot \Delta t. \tag{4.63}$$

Snaga ρ^{din} prema [3, 34, 39] jednaka je:

$$\varphi^{din} = \kappa \cdot \upsilon^2 \cdot f, \tag{4.64}$$

gdje je κ konstanta specifična obradbenoj jedinici, υ je napon, a f frekvencija obradbene jedinice. Prema istim izvorima statička energija proporcionalna je dinamičkoj:

$$\varepsilon^{stat} \propto \varepsilon^{din}, \tag{4.65}$$

tako da ukupnu potrošenu energiju možemo aproksimirati kao

$$\varepsilon = \varphi^{din} \cdot \Delta t = \kappa \cdot \upsilon^2 \cdot f \cdot \Delta t. \tag{4.66}$$

Prema [87] napon raste s f^3, tako da za potrebe primjera napon izražavamo preko frekvencije prema tablici DVFS profila Cubieboard SoC platforme (koju smo koristili za preliminarna testiranja prije izrade ovoga rada) dobivenoj iz [4], kao primjera energetski učinkovitog obradbenog čvora s velikim potencijalom za izgradnju sustava HPC-a za manje zahtjevne aplikacije:

$$\upsilon \approx 0.2789 f^2 + 0.1401 f + 1.0143 \tag{4.67}$$

tako da izraz (4.66) postaje:

$$\begin{aligned} \varepsilon &= \kappa \cdot \upsilon^2 \cdot f \cdot \Delta t & (4.68)\\ &= \kappa \cdot \left(0.2789 f^2 + 0.1401 f + 1.0143\right)^2 \cdot f \cdot \Delta t, & (4.69) \end{aligned}$$

gdje zbog jednostavnosti postavljamo $\kappa = 1$, tako da izraz (4.69) postaje:

$$\varepsilon = \left(0.2789 f^2 + 0.1401 f + 1.0143\right)^2 \cdot f \cdot \Delta t, \tag{4.70}$$

S obzirom na to da razmatramo utrošak energije samo obradbenih isječaka (prema pretpostavkama postupka LVI), ukupna energija ε_j^{uk} obradbene jedinice p_j jednaka je zbroju utrošaka energije ε_i svih obradbenih isječaka $v_i \in (S_j \setminus \Psi_i)$ u rasporedu te obradbene jedinice:

$$\varepsilon_j^{uk} = \sum_{v_i \in (S_j \setminus \Psi_i)} \varepsilon_i. \tag{4.71}$$

Utrošak energije obradbenog isječka v_i dan je izrazom:

$$\varepsilon_i = \left(0.2789 {f_i}^2 + 0.1401 f_i + 1.0143\right)^2 \cdot f_i \cdot t_i \tag{4.72}$$

u kojem ako je

$$\begin{aligned} f_i &= f_j^{max} & (4.73)\\ t_i &= t_i^{izv} & (4.74) \end{aligned}$$

dobivamo utrošak energije obradbene jedinice v_i prije skaliranja ε_i, dok za

$$\begin{aligned} f_i &= f_i^{op} & (4.75)\\ t_i &= t_i^{izv\prime} & (4.76) \end{aligned}$$

dobivamo utrošak energije nakon skaliranja $\varepsilon_i{}'$. Razliku utroška energije računamo za svaku obradbenu

jedinicu zasebno:

$$\varepsilon_i^{raz} = \varepsilon_i - \varepsilon_i{}' \tag{4.77}$$

i dobivamo ukupno smanjenje utroška energije obradbene jedinice p_j:

$$\varepsilon_j^{raz} = \sum_{v_i \in (S_j \setminus \Psi_i)} \varepsilon_i^{raz} \tag{4.78}$$

i ukupno smanjenje utroška energije:

$$\varepsilon_{uk}^{raz} = \sum_{p_j \in P} \varepsilon_j^{raz}. \tag{4.79}$$

Za navedeni primjer dobivene su vrijednosti prikazane tablicom 4.11.

Tablica 4.11: Parametri vremenskih isječaka za primjer 3 nakon cjelokupnog postupka LVI.

Obr. jed.	ε_j	$\varepsilon_j{}'$	ε_j^{raz}	$\varepsilon_j^{raz}[\%]$
p_0	16,43	13,49	2,94	17,90
p_1	4,11	2,68	1,43	34,82
p_2	18,49	14,69	3,80	20,55
ukupno	39,03	30,86	8,17	20,94

U navedenoj tablici se vidi da je postupak LVI omogućio uštedu energije pri izvođenju izvršnih jedinica programa od gotovo 21% u odnosu na izvorni raspored, pritom zadržavajući isto ukupno vrijeme izvođenja programa. Najveća ušteda postignuta je na obradbenoj jedinici p_1 od gotovo 35% zbog toga što u izvornom rasporedu trajanje obradbenih isječaka traje vrlo kratko (2 vremenske jedinice), dok nakon skaliranja traju 10 vremenskih jedinica.

Navedenim primjerom pokazali smo prednosti postupka LVI kao dijela unaprijeđenog postupka raspoređivanja RaNDKiP u vidu procjene uštađene energije koju on omogućuje. Detaljnija analiza uslijedit će u eksperimentalnoj analizi, gdje će biti pokazano kako navedeni postupak omogućuje sličnu uštedu i za programe iz poznatih repozitorija. Uz unaprijeđene karakteristike postupka RaNDKiP (podrška raznorodnosti) omogućujemo i uštedu energije, čime uvjetno (ušteda energije ograničena je rasporedom) omogućujemo raspoređivanje prema dva kriterija — vremena i energije. Navedeno čini i naš doprinos D.3.

U sljedećem poglavlju bit će predstavljena eksperimentalna analiza naših unaprijeđenih postupaka raspoređivanja RaNDKiP i RaPPaMaG, u kojima ćemo dodatno utvrditi izvorne znanstvene doprinose koje njima postižemo.

Poglavlje 5

Eksperimentalna analiza

U ovom poglavlju prikazano je vrednovanje unaprijeđenih postupaka RaPPaMaG i RaNDKiP ispitivanjem njihove primjenjivosti, kao i usporedbom s drugim postupcima iste ili slične namjene. Na taj način postižemo i postavljeni cilj 4 našeg istraživanja u obliku dijela doprinosa D.1. (vrednovanje). S obzirom na to da su u literaturi eksperimentalne postavke za postupke raspoređivanja programa vrlo raznolike, a često i blisko vezane uz vrlo usko područje primjene, napravit ćemo presjek najvažnijih i uvesti vlastite. Vlastite pretpostavke, postavke i mjerene veličine uvodimo u slučajevima gdje postojeće ne daju potpun opis svih važnih svojstava i prednosti predloženih postupaka. Također ćemo opisati mjerne parametre pokazatelja kvalitete koje smo koristili u eksperimentalnoj analizi, tipove testnih slučajeva i njihovih parametara te alata za njihovu izradu i analizu.

Poglavlje dijelimo na sedam glavnih cjelina (pododjeljaka): u prvoj opisujemo korištene alate (postojeće i razvijene u sklopu ovog rada i vlastitog istraživanja), u drugoj opisujemo postavke eksperimenata i generiranje testnih slučajeva, sljedeća opisuje mjerene veličine pokazatelja kvalitete koji opisuju dobivene performanse, naredne dvije sadržavaju rezultate i njihovu analizu za postupke iz poglavlja 4, a to su raspoređivanje programa particioniranjem miješanoga acikličkoga grafa (postupak RaPPaMaG) i raspoređivanje programa rangiranim listama izvršnih jedinica uz raspoređivanje komunikacije i skaliranje radne frekvencije, (postupak RaNDKiP) te u posljednjoj rekapituliramo eksperimentalnu analizu zbirnim prikazom rezultata.

Alati se dijele na dvije glavne skupine: postojeći alati i alati napravljeni u okviru ovog rada. Karakteristike testnih slučajeva opisuju postavke programa koji će biti raspoređivani, platforme na koje će biti raspoređivani i postavke postupaka za njihovo raspoređivanje. Testni se slučajevi također dijele na dvije skupine. Prva su testni slučajevi generirani vlastitim prilagođenim postupcima (DAGGer, M-DAGGer i P-DAGGer), dok su u drugoj testni slučajevi generirani standardiziranim nasumičnim postupkom TGFF i instance radnih opterećenja Pegasus. Mjerne veličine dijele se na pokazatelje vremenskih karakteristika i na pokazatelje energetske učinkovitosti postupaka raspoređivanja.

Sljedećim pododjeljkom dajemo uvid u alate korištene za eksperimentalnu analizu, kako postojećih, tako i samostalno razvijenih.

5.1 Alati za generiranje i vrednovanje testnih slučajeva

U ovom pododjeljku prikazujemo alate korištene u eksperimentalnoj analizi i vrednovanju. U prvom redu, korišteni su alati za izradu testnih slučajeva, (većinom modela paralelnih programa i računalnih platformi), alati za vizualizaciju i prikaz pojedinih aspekata postupaka raspoređivanja te samih alata za raspoređivanje. Od postojećih alata za generiranje testnih slučajeva korišteni su generator "besplatnih grafova zadataka" TGFF[1] (eng. *task graphs for free*), opisan u [69], radna opterećenja Pegasus, opisana u [2,49,50] i njihov generator[2] te opisni jezik grafova DOT [105] (dostupan kroz alat **GraphViz**[3], opisan u [106]).

5.1.1 TGFF

Alat TGFF, prema [69], jest generator usmjerenih acikličkih grafova koji predstavljaju programe podijeljene na manje jedinice, (zadatke, predstavljene vrhovima grafa), koje se mogu izvršavati serijski ili paralelno te koji su međusobno ovisni, tj. povezani bridovima. Navedeni alat primarno je namijenjen generiranju testnih programa koji predstavljaju stvarne programe korištene u domeni raspodijeljenih ugrađenih računalnih sustava, primjerice multiprocesora [34] i sustava mreža na čipu [40, 107], no često se koristi i u domeni računalnih sustava visokih performansi poput spleta računala [17, 84]. S obzirom na to da je TGFF prvenstveno namijenjen generiranju modela programa namijenih za rad na ugrađenim računalnim sustavima, on sadrži mogućnost da, osim generiranja strukture samog programa, (oblik i struktura grafa s vrhovima i bridovima), definira za zadatke i njihovo nužno vrijeme završetka (eng. *deadlines*), njihova svojstva pri izvršavanju na različitim obradbenim jedinicama sklopovlja, njihove memorijske parametre i periode izvršavanja itd. Također, TGFF može generirati i višestruke grafove koji predstavljaju programe koji se sastoje od manjih potprograma, (svaki je jedan graf). Međutim, TGFF je prvenstveno zanimljiv kao generator strukture programa (oblika grafa koji predstavlja program), zbog čega se i koristi u drugim domenama računarstva, primjerice u [17, 84], jer korisnik sam može odabrati koje parametre želi vrednovati. Iz tog razloga korišten je i za generiranje strukture programa u eksperimentalnoj analizi našeg rada.

TGFF radi tako da prema konfiguracijskoj datoteci koja je tipa ***.tgffopt** i opisuje tražena svojstva programa, generira izlaznu datoteku u ***.tgff** (interni format datoteke za alat TGFF), ***.vcg** (eng. *visualization of compiler graphs*) ili ***.eps** (eng. *encapsulated postscript*) obliku. Primjer jedne takve konfiguracijske datoteke (packets.tgffopt) prikazan je slikom 5.1.

Prema [59], neki od parametara konfiguracije generiranih grafova su:

- `tg_cnt <int>`: postavlja broj grafova koji će se generirati.
- `tg_label <string>`: postavlja nazive grafova (identifikacija).
- `tg_offset <string><int>`: postavlja početnu numeričku oznaku zadataka (predefinirano je 0).

[1] dostupan na adresi http://ziyang.eecs.umich.edu/~dickrp/tgff/
[2] dostupan na https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator
[3] dostupan na http://www.graphviz.org/

- `task_cnt <int><int>`: postavlja minimalni broj zadataka po grafu zadataka, (srednja vrijednost, odstupanje).
- `start_node <int><int>`: postavlja broj ulaznih čvorova grafa, (srednja vrijednost, odstupanje).

```
task_cnt 5 2
trans_type_cnt 10

eps_write
tg_write
pe_write

table_label TRANS_MAGNITUDE
table_cnt 1
table_attrib
type_attrib magnitude 80 40
trans_write

table_label COMMUN
table_cnt 3
table_attrib price 30 10, packet_size 10 3, packet_time
10 2
misc_write
```

Slika 5.1: Primjer konfiguracijske datoteke *packets.tgffopt* za TGFF.

Osim generiranja grafova prema spomenutim parametrima, kod preuzimanja alata TGFF s internetskih stranica moguće je preuzeti i nekoliko predefiniranih struktura grafa: *bus*, *creds1*, *kbasic_tables*, *kbasic_task*, *kextended*, *kextended*, *kseries_parallel*, *packed*, *packets*, *robtst*, *simple* i *sp_rand*. Svaka od njih predstavlja strukturu specifičnu određenim tipovima programa. No, s obzirom na to da među njima postoje određene sličnosti u našoj primjeni (oblik grafa), neke od njih smo izostavili iz eksperimentalne analize. Konkretnije, strukture *kextended* i *kbasic_task* su identične, a isti slučaj je i za *packets* i *creds1* te *kseries_parallel_xover* i *robtst*. S druge strane, *kbasic_task* i *kbasic_tables* su vrlo slični. Iz svakog od navedenih parova koristili smo samo jednu strukturu (označenu podvučeno).

Primjerice, za konfiguracijsku datoteku sa slike 5.1 TGFF generira program u obliku DAG-a koji sprema u *.tgff datoteku prikazanu slikom 5.2.

```
@TASK_GRAPH 0 {
       PERIOD 466.667

       TASK t0_0    TYPE 13
       TASK t0_1    TYPE 12
       TASK t0_2    TYPE 15
       TASK t0_3    TYPE 1
       TASK t0_4    TYPE 0
       TASK t0_5    TYPE 11
       TASK t0_6    TYPE 14
       TASK t0_7    TYPE 8

       ARC a0_0     FROM t0_0  TO  t0_1 TYPE 4
       ARC a0_1     FROM t0_0  TO  t0_2 TYPE 3
       ARC a0_2     FROM t0_1  TO  t0_2 TYPE 0
       ARC a0_3     FROM t0_2  TO  t0_3 TYPE 6
       ARC a0_4     FROM t0_2  TO  t0_4 TYPE 4
       ARC a0_5     FROM t0_2  TO  t0_5 TYPE 2
       ARC a0_6     FROM t0_2  TO  t0_6 TYPE 5
       ARC a0_7     FROM t0_2  TO  t0_7 TYPE 3

       HARD_DEADLINE d0_0 ON t0_3 AT 400
       HARD_DEADLINE d0_1 ON t0_4 AT 400
       HARD_DEADLINE d0_2 ON t0_5 AT 400
       HARD_DEADLINE d0_3 ON t0_6 AT 400
       HARD_DEADLINE d0_4 ON t0_7 AT 400
}
```

Slika 5.2: Graf programa u *.tgff obliku dobiven TGFF-om i konfiguracijom sa slike 5.1.

Na toj slici je vidljivo da se jedan od grafova, graf `TASK_GRAPH 0`, sastoji od osam zadataka, (tumačimo ih kao izvršne jedinice), `TASK t0_0` do `TASK t0_7` i osam lukova, `ARC a0_0` do `ARC a0_7`. To je samo jedan od grafova iz navedene konfiguracijske datoteke (alat ih generira više). Dobiveni graf zapisan je i u *.vcg datoteku. Primjer datoteke "simple.vcg" prema konfiguraciji "simple.tgffopt" prikazan je na slici 5.3 opisnim jezikom grafova VCG iz [108, 109].

```
graph: { label: "simple.vcg"
display_edge_labels: yes
            node: { title: "t0_0" label: "t0_0 (18)"  color: lightgreen }
            node: { title: "t0_1" label: "t0_1 (2)"  color: white }
            node: { title: "t0_2" label: "t0_2 (13)"  color: white }
            node: { title: "t0_3" label: "t0_3 (12)"  color: white }
            node: { title: "t0_4" label: "t0_4 (4)"  color: white }
            node: { title: "t0_5" label: "t0_5 (19)"  color: white }
            node: { title: "t0_6" label: "t0_6 (4)"  color: white }
            node: { title: "t0_7" label: "t0_7 (16)"  color: white }
            node: { title: "t0_8" label: "t0_8 (0)"  color: white }
            node: { title: "t0_9" label: "t0_9 (10)"  color: white }
            node: { title: "t0_10" label: "t0_10 (6)"  color: white }
            node: { title: "t0_11" label: "t0_11 (14)"  color: lightred }

            edge: { thickness: 2 sourcename:"t0_0" targetname: "t0_1"  label: "(2)" }
            edge: { thickness: 2 sourcename:"t0_0" targetname: "t0_2"  label: "(35)" }
            edge: { thickness: 2 sourcename:"t0_1" targetname: "t0_3"  label: "(26)" }
            edge: { thickness: 2 sourcename:"t0_2" targetname: "t0_3"  label: "(1)" }
            edge: { thickness: 2 sourcename:"t0_2" targetname: "t0_4"  label: "(23)" }
            edge: { thickness: 2 sourcename:"t0_3" targetname: "t0_4"  label: "(34)" }
            edge: { thickness: 2 sourcename:"t0_1" targetname: "t0_4"  label: "(41)" }
            edge: { thickness: 2 sourcename:"t0_4" targetname: "t0_5"  label: "(2)" }
            edge: { thickness: 2 sourcename:"t0_4" targetname: "t0_6"  label: "(20)" }
            edge: { thickness: 2 sourcename:"t0_6" targetname: "t0_7"  label: "(37)" }
            edge: { thickness: 2 sourcename:"t0_3" targetname: "t0_7"  label: "(24)" }
            edge: { thickness: 2 sourcename:"t0_5" targetname: "t0_7"  label: "(29)" }
            edge: { thickness: 2 sourcename:"t0_7" targetname: "t0_8"  label: "(17)" }
            edge: { thickness: 2 sourcename:"t0_8" targetname: "t0_9"  label: "(5)" }
            edge: { thickness: 2 sourcename:"t0_8" targetname: "t0_10"  label: "(7)" }
            edge: { thickness: 2 sourcename:"t0_7" targetname: "t0_10"  label: "(27)" }
            edge: { thickness: 2 sourcename:"t0_9" targetname: "t0_11"  label: "(11)" }
            edge: { thickness: 2 sourcename:"t0_6" targetname: "t0_11"  label: "(31)" }
            edge: { thickness: 2 sourcename:"t0_10" targetname: "t0_11"  label: "(12)" }

 }
```

Slika 5.3: Graf opisan jezikom VCG dobiven TGFF-om i konfiguracijom "simple.tgffopt".

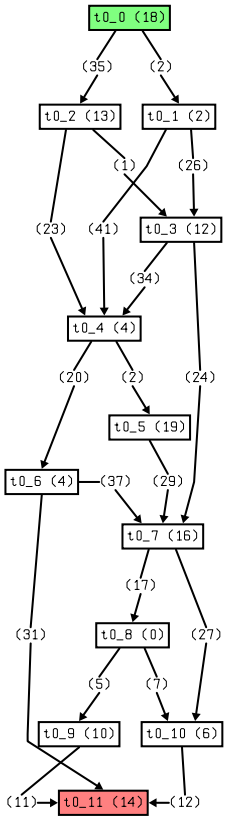
Datoteke u *.vcg obliku mogu se vizualizirati alatom "**aisee**"[4] u operacijskom sustavu MS Windows, tako da je graf prema datoteci sa slike 5.3 prikazan na slici 5.4.

S obzirom na to da grafovi dobiveni alatom TGFF ne odgovaraju u potpunosti modelima kojima se služe postupci raspoređivanja opisani u poglavlju 4, potrebno je izvršiti određene prilagodbe. Ono što nam ne odgovara u izvornim grafovima jest to što se pri transformaciji *.tgff datoteke (priloženom skriptom u Perl[5] programskom jeziku **tgff2dot.pl**) u format DOT koji koristimo za zapis grafova u eksperimentalnoj analizi (opisan kasnije), gube parametri komunikacije među izvršnim jedinicama te parametri troškova izvođenja zadataka na obradbenim jedinicama sklopovlja.

Nadalje, komunikacija među izvršnim jedinicama programa definirana TGFF-om ne odgovara modelu programa iz pododjeljka 3.1, pojedine konfiguracijske datoteke iziskuju generiranje višestrukih grafova (nama je potreban jedan), ne postoji podrška za generiranje miješanih grafova (sustav generira DAG) te nisu prisutni svi parametri izvršnih jedinica koji su nam potrebni (x^{ob}, x^{en} itd.).

[4] dostupan na http://www.absint.com/aisee/

[5] Porodica dinamičkih programskih jezika visoke razine.



Slika 5.4: Prikaz grafa programa opisanog datotekom "simple.vcg" prikazanoj na 5.3 alatom "aisee".

Iz tog razloga nad grafovima dobivenim alatom TGFF vršimo sljedeće prilagodbe:

1. zadatke smatramo ekvivalentnima izvršnim jedinicama programa,
2. izvršnim jedinicama programa pridjeljujemo parametre vlastitim postupcima iz okruženja DAG-Ger opisanim kasnije (broj instrukcija x^{ob} koje obavljaju i vremena izvođenja t^{izv} na pojedinim obradbenim jedinicama sklopovlja),
3. određenom broju bridova dajemo značaj komunikacije (20%) dok drugima pridjeljujemo značaj vremenskog slijeda — transformacija u miješani graf za postupak RaPPaMaG,
4. bridovima dajemo značaj i komunikacije i vremenskog slijeda istovremeno (asinkrona komunikacija, DAG) za postupak RaNDKiP,
5. zanemarujemo nužna vremena završetka i ostale parametre,
6. uzimamo u obzir samo prvi generirani graf (ako ih je generirano više),
7. vlastitim alatom transformiramo *.tgff datoteku u *.dot datoteku.

Navedenim prilagodbama oblikujemo programe za testne slučajeve prema većini relevantnih radova iz područja, kako bismo mogli provesti izravniju analizu i mjerenja. Također, transformacijom u *.dot oblik omogućavamo primjenu ostalih alata na navedeni graf i njihovu vizualizaciju alatom GraphViz.

Za generiranje testnih slučajeva eksperimentalne analize od postojećih alata, osim TGFF-a, koristimo i radna opterećenja Pegasus koja opisujemo u sljedećem pododjeljku.

5.1.2 Radna opterećenja Pegasus

Prikaz radnih opterećenja iz repozitorija Pegasus, prema [49], nastao je zbog potrebe za standardiziranom bibliotekom referentnih testova (eng. *benchmark*) u obliku najčešćih znanstvenih aplikacija (programa). Takva biblioteka omogućila bi bolji razvoj sustava za kontrolu opterećenja računalnih sustava visokih performansi (uključujući i postupke raspoređivanja), pomoću stvarnih programa koji se koriste u praksi. U tu svrhu, autori iz [49, 50] u suradnji sa širokom znanstvenom zajednicom iz područja astronomije, bioinformatike, seizmologije, fizike gravitacijskih valova, oceanografije, limnologije i ostalih izgradili su bazu najčešće korištenih struktura programa u obliku DAG-a, koji se koriste u tim znanstvenim disciplinama na sustavima HPC. Osim navedenog opisa, definirane su i skriptirane postavke tih opterećenja za pokretanje na zastupljenijim verzijama sustava HPC-a, poput raznih spletova i grozdova računala.

Biblioteka Pegasus konstantno se povećava novim oblicima radnog opterećenja iz istih, ali i novih područja znanosti, kao i metodama za njihovu analizu i optimizaciju. No ono čime se radna opterećenja Pegasus ističu kao izbor za generiranje testnih slučajeva u svrhu ispitivanja postupaka raspoređivanja je njihova jednostavna analiza i primjena za izradu programa koji testiraju navedene postupke. Nadalje, njihov oblik predstavlja oblik postojećih programa, pa postupci raspoređivanja koji daju dobre rezultate na njima imaju velik potencijal dati dobre rezultate i u stvarnoj upotrebi. Samim time, upotreba postupaka raspoređivanja na opterećenjima Pegasus daje uvid u to kako će se oni ponašati u praksi. Zbog toga ih neki od autora u tom području već i koriste, primjerice oni u [13, 23, 39].

Radna opterećenja Pegasus u obliku DAG-a dijele se na pet glavnih tipova:

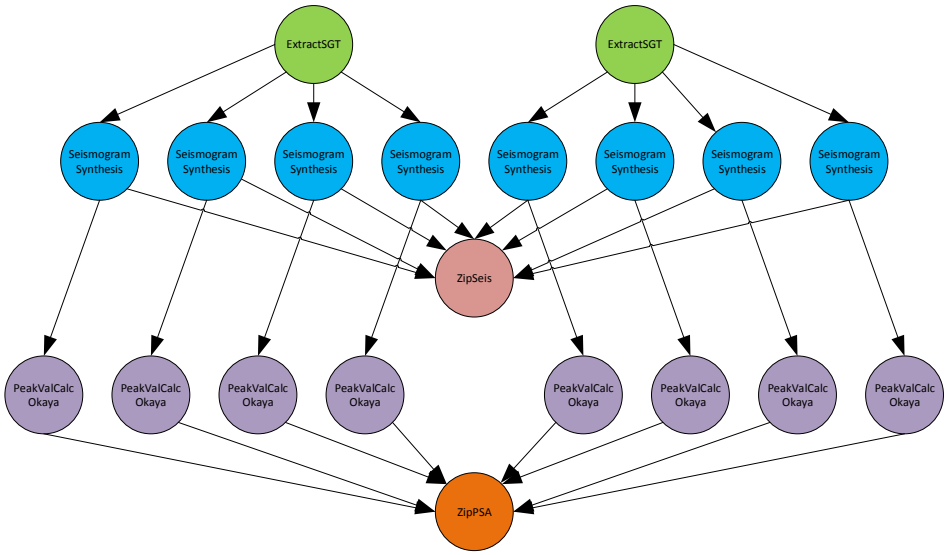
1. **Karakterizacija CyberShake** - koristi se u sjedištu za istraživanje potresa Južne Kalifornije (SAD) za analizu opasnosti od potresa u nekoj regiji.
2. **Karakterizacija Epigenomics** - kreiran je u sjedištu za automatizaciju različitih operacija prilikom obrade genskih sekvenci ESC Epigenome.
3. **Karakterizacija LIGO inspiral** - koristi se za generiranje i analizu gravitacijskih valnih oblika iz podataka prikupljenih stapanjem kompaktnih binarnih sustava.
4. **Karakterizacija Montage** - program Montage napravila je NASA/IPAC i služi za međusobno spajanje odvojeno prikupljenih slika neba.
5. **Karakterizacija SIPHT** - kreiran na projektu bionformatike sveučilišta Harvard za automatizaciju traženja neprevedenih sekvenci RNA bakterijskih replika u bazi NCBI.

Navedeni tipovi radnih opterećenja opisuju različite tipove grafa programa (njihove strukture) u spomenutim područjima primjene. Njihov dodatni parametar su i razmjeri navedenih karakteristika radnih opterećenja (broj izvršnih jedinica koje tvore program), od kojih postoje 4 različita: mali, srednji, veliki i vrlo veliki. Navedeni razmjeri prikazani su i tablicom 5.1 za svaki od pojedinih karakteristika. Primjer radnog opterećenja *CyberShake* malih razmjera prikazan je na slici 5.5, na kojoj je vidljiva glavna struktura programa te nazivi funkcionalnosti njegovih izvršnih jedinica. Svaki od spomenutih radnih opterećenja u četiri razmjera moguće je preuzeti iz [2]. Format tih radnih opterećenja je tzv. DAX (eng. *DAG XML*) ili usmjereni aciklički grafovi opisani u formatu XML (eng. *extensible markup language*).

Tablica 5.1: Različite veličine radnih opterećenja Pegasus koje se mogu preuzeti iz [2].

	Razmjeri opterećenja (broj izvršnih jedinica u programu), ϑ			
Karakteristika	Mali	Srednji	Veliki	Vrlo veliki
CyberShake	30	50	100	1000
Epigenomics	24	46	100	997
LIGO inspiral	30	50	100	1000
Montage	30	60	100	1000
SIPHT	25	50	100	1000

Dio datoteke "CyberShake_30.xml" (koja predstavlja opterećenja CyberShake malih razmjera) prikazan je na slici 5.6.



Slika 5.5: Prikaz grafa radnog opterećenja CyberShake, prema [2].

Navedena datoteka sastoji se od tri glavna dijela. Prvi dio (*part* 1), sadržava listu referenciranih datoteka za pojedine čvorove grafa (primjerice, ako se program treba fizički pokrenuti na nekom računalnom sustavu koristi se struktura grafa, gdje je svaki od vrhova izvršna datoteka iz ove liste). Drugi dio (*part* 2 u datoteci), opisuje tipove izvršnih jedinica programa te nazive datoteka koje sadržavaju njihovu funkcionalnost, primjerice "CyberShake_PSA.zip" i "FFI_0_1_fx.sgt" te njihove parametre. Sljedeći dio (*part* 3 u datoteci), opisuje veze među izvršnim jedinicama (strukturu grafa, lukove) i to na način da za svaku izvršnu jedinicu definira listu njegovih prethodnika, tj. roditelja (eng. *parent*).

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated: 2014-03-17T15:17:44+01:00 -->
<!-- generated by: Zdravko [??] -->
<adag xmlns="http://pegasus.isi.edu/schema/DAX" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://pegasus.isi.edu/schema/DAX http://
pegasus.isi.edu/schema/dax-2.1.xsd" version="2.1" count="1" index="0" name="test"
jobCount="30" fileCount="0" childCount="28">
<!-- part 1: list of all referenced files (may be empty) -->
<!-- part 2: definition of all jobs (at least one) -->
  <job id="ID00000" namespace="CyberShake" name="ZipPSA" version="1.0" runtime="0,06">
    <uses file="Cybershake_PSA.zip" link="output" register="true" transfer="true"
optional="false" type="data" size="180"/>
  </job>
  <job id="ID00001" namespace="CyberShake" name="ZipSeis" version="1.0" runtime="0,27">
    <uses file="Cybershake_Seismograms.zip" link="output" register="true" transfer="true"
optional="false" type="data" size="61417"/>
  </job>
  <job id="ID00002" namespace="CyberShake" name="ExtractSGT" version="1.0" runtime="73,35">
    <uses file="FFI_0_1_fx.sgt" link="input" register="true" transfer="true"
optional="false" type="data" size="20072103568"/>
    <uses file="FFI_0_1_fy.sgt" link="input" register="true" transfer="true"
optional="false" type="data" size="20072103568"/>
    <uses file="FFI_0_1_subfx.sgt" link="output" register="false" transfer="false"
optional="false" type="data" size="12864993 2"/>
    <uses file="FFI_0_1_subfy.sgt" link="output" register="false" transfer="false"
optional="false" type="data" size="12864993 2"/>
    <uses file="FFI_0_1_txt.variation-s00048-h06450" link="input" register="true"
transfer="true" optional="false" type="data" size="2857672"/>
  </job>

…

  <!-- part 3: list of control-flow dependencies (may be empty) -->
  <child ref="ID00000">
    <parent ref="ID00021"/>
    <parent ref="ID00012"/>
    <parent ref="ID00023"/>
    <parent ref="ID00010"/>
    <parent ref="ID00025"/>
    <parent ref="ID00006"/>
    <parent ref="ID00017"/>
    <parent ref="ID00027"/>
    <parent ref="ID00004"/>
    <parent ref="ID00015"/>
    <parent ref="ID00029"/>
    <parent ref="ID00008"/>
    <parent ref="ID00019"/>
  </child>

…

  </adag>
```

Slika 5.6: Prikaz sadržaja dijela datoteke DAX "CyberShake_30.xml" preuzete s [2].

Uz navedene instance radnih opterećenja iz izvora [2] može se preuzeti i testni virtualni stroj s cjelovitim programskim okvirom Pegasus, unutar kojeg se nalazi i skripta za pretvorbu formata DAX u opisni jezik formata DOT (**dag2dot**) koji je opisan u sljedećem pododjeljku.

5.1.3 Zapis grafova u opisnom jeziku DOT

Kako bi unificirali prikaz grafova, ali i omogućili jedinstven oblik ulaznih entiteta raspoređivanja (grafova) za novorazvijene i postojeće alate u eksperimentalnom dijelu, koristili smo jedan od najbolje prihvaćenih opisnih jezika te svrhe u praksi — opisni jezik grafova DOT. Opisni jezik DOT je jedan od najpopularnijih opisnih jezika za grafove, izvorno nastao 1991. godine, detaljno opisan i dorađen 2000. godine u [105]. Navedeni opisni jezik služi vizualizaciji i zapisu gotovo svih vrsta i oblika grafova, jednostavan je za

razvoj te je podržan od mnogih alata za razvoj i vizualizaciju, poput popularnog alata **GraphViz**[6] [106]. Alat GraphViz ima editor za format DOT, kao i mogućnost njegove vizualizacije i spremanja u mnoge popularne formate poput *.pdf, *.eps, *.emf te raznih grafičkih formata poput *.bmp, *.jpeg, *.gif itd.

Izvorno, datoteka opisnog jezika DOT ima ekstenziju ***.dot** ili ***.gv**. Graf opisan jezikom DOT obično se sastoji od tri glavna dijela, čiji je primjer dan slikom 5.7.

```
digraph dag {
  ratio=fill;
  node [style=filled, shape=box,
color=black,fillcolor=white];
  subgraph{
    edge [arrowhead=normal,arrowsize=0.8];
    "T0" -> "T1";
    "T0" -> "T2";
    "T1" -> "T4";
    "T2" -> "T4";
    "T4" -> "T7";
    "T6" -> "T9";
    "T8" -> "T11";
    "T9" -> "T11";
    "T10" -> "T11";
  }
  subgraph{
    node [style=filled,color=black,fillcolor=khaki];
    edge [arrowhead=none, color=brown];
    "T2" -> "T5" [label="13"];
    "T3" -> "T6" [label="2"];
    "T4" -> "T6" [label="4"];
    "T6" -> "T8" [label="1"];
    "T7" -> "T10" [label="10"];
    "T9" -> "T10" [label="10"];
  }
}
```

Slika 5.7: Primjer miješanoga grafa u opisnom jeziku DOT.

Prvi je dio zaglavlje u kojem se navode parametri zajednički za čitav graf. Drugi je dio zapis čvorova (vrhova) grafa primjerice linija `"T1" [shape=ellipse,fillcolor="red",label="Izvršna jedinica 1"]` definira čvor s identifikacijskom oznakom T1, elipsoidnog oblika crvene boje popune i teksta koji će biti sadržan u tom vrhu "Izvršna jedinica 1". Treći je zapis bridova koji se može prepoznati po oznaci "->", a koja označava smjer brida; primjerice `"T1" ->"T2" [arrowhead=none, color=brown, label="Brid 1"]`, označava brid od čvora T1 do čvora T2, bez strelice, smeđe boje, na kojem će pisati "Brid 1".

U potpuno povezanim grafovima deklaracija vrhova i bridova može se definirati istom linijom, kao što je to prikazano na primjeru miješanoga grafa opisanog jezikom DOT na slici 5.7. U navedenoj datoteci prikazan je miješani graf koji se sastoji od 12 vrhova (T0 do T11), devet lukova (usmjerenih bridova bez oznake, linije `"T0" ->"T1"` do `"T10" ->"T11"`) i neusmjerenih bridova koji predstavljaju sinkronu komunikaciju (`"T2" ->"T5"` do `"T9" ->"T10"`). Vizualizacija grafa opisanog datotekom sa slike 5.7 alatom GraphViz dana je slikom 5.8.

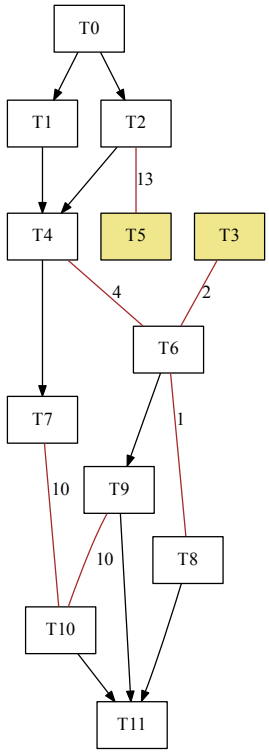
Inače, slike 5.7 i 5.8 ujedno prikazuju i predložak koji smo definirali za prikaz miješanih grafova u eksperimentalnom dijelu. Identificirani su osnovni entiteti miješanih grafova:

- **usmjereni brid crne boje** - tijek izvođenja programa (vremenski slijed jedinica programa),
- **neusmjereni brid smeđe boje** - sinkrona komunikacija između sudionika koji su njima povezani

[6] http://www.graphviz.org/

(uz prikaz iznosa te komunikacije),

- **pravokutnici žute ispune** - izvršne jedinice programa spojene samo komunikacijskim vezama s ostatkom grafa, sinkro-vrhovi,
- **pravokutnici bijele ispune** - sve ostale izvršne jedinice programa.



Slika 5.8: Vizualizacija grafa opisanog datotekom DOT sa slike 5.7.

I vrhovi i bridovi mogu imati tekstualne ili numeričke vrijednosti te razne druge parametre tekstualne naravi, koje je moguće interpretirati na željen način, tako da ih je jednostavno koristiti za obradu i za uvoz podataka postupcima raspoređivanja programa. Zbog toga smo i mi koristili opisni jezik DOT za sve eksperimente.

Osim postojećih alata, a u svrhu bolje prilagodbe testnih slučajeva i njihove analize, kreirali smo i vlastite. Prvi od njih jest okruženje za generiranje testnih slučajeva DAGGer (eng. *DAG generator*), opisano u sljedećem pododjeljku.

5.1.4 Razvijeni alati: generatori testnih slučajeva DAGGer, DAGGer-S i DAGGer-P

Kako bi opisali testne slučajeve u obliku modela programskog okvira LOSECO iz poglavlja 3, kreirali smo alat za generiranje grafova programa DAGGer (eng. ***DAG generator***). Navedeni alat služi za generiranje strukture programa u obliku usmjerenih acikličkih grafova, ali i miješanih acikličkih grafova. Algoritam 8 opisuje način rada alata DAGGer. Navedeni alat temelji se na nasumičnom generiranju

matrice susjedstva (koja predstavlja bridove grafa) izvršnih jedinica programa. S obzirom na to da DAGGer generira usmjerene acikličke grafove prema predefiniranim postavkama, svaki od bridova mora imati i smjer. Graf ne smije sadržavati petlje (cikluse, aciklički graf), tako da je u matricu susjedstva dozvoljeno stavljati vrijednosti samo iznad (ili samo ispod) glavne dijagonale, (linije 2 i 10) jer je tako najlakše generirati traženi graf. Vrijednost od -1 označava da brida nema, dok 0 u matrici na mjestu (i, j) označava brid usmjeren od vrhova v_i prema v_j (linija 5). Nadalje, od svih vrijednosti iznad glavne dijagonale generiraju se lukovi u samo 5% slučajeva (linija 3). Navedena vrijednost je odabrana da bude što sličnija strukturama koje generiraju TGFF i Pegasus, (pododjeljci 5.1.1 i 5.1.2).

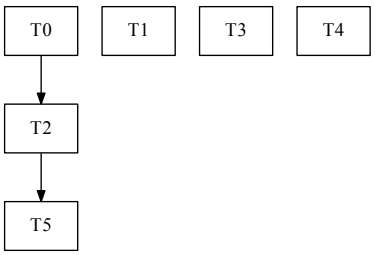
Algoritam 8 DAGGer.

1: $V \leftarrow$ Izvršne jedinice koje čine program
2: **za sve** $\{v_i, v_j \in V : \ j > i\}$ **radi**
3: $\quad e_{i,j} = random \in [0, 19]$ {1 od 20 su bridovi}
4: $\quad$ **ako** $e_{i,j} > 0$ **onda**
5: $\quad\quad e_{i,j} = 0$ {generiraj luk od v_i do v_j}
6: $\quad$ **else**
7: $\quad\quad e_{i,j} = -1$ {ne postoji brid između v_i i v_j}
8: $\quad$ **završi ako**
9: **završi za**
10: **za sve** $\{v_i, v_j \in V : \ j \leq i\}$ **radi**
11: $\quad e_{i,j} = -1$ {ne postoji brid između v_i i v_j}
12: **završi za**

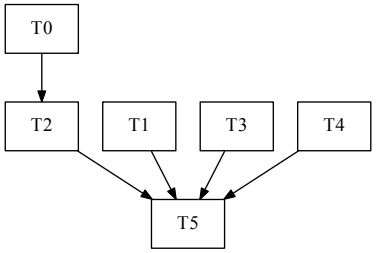
Primjer grafa generiranog alatom DAGGer prikazan je na slici 5.9. Slika 5.9a prikazuje matricu

	v_0	v_1	v_2	v_3	v_4	v_5
v_0	-1	-1	0	-1	-1	-1
v_1	-1	-1	-1	-1	-1	-1
v_2	-1	-1	-1	-1	-1	0
v_3	-1	-1	-1	-1	-1	-1
v_4	-1	-1	-1	-1	-1	-1
v_5	-1	-1	-1	-1	-1	-1

(a) Matrica susjedstva.



(b) Vizualizacija grafa.



(c) Povezivanje izlaznih čvorova virtualnim čvorom.

Slika 5.9: Primjer grafa programa generiranog alatom DAGGer.

susjedstva, gdje je nijansama sive boje označen dio koji se može mijenjati, (prebaciti vrijednost -1 u 0, tj. stvoriti luk u grafu). Oblik DOT tog grafa vidljiv je na slici 5.9b u kojem se, s obzirom na to da se bridovi generiraju nasumično u 5% slučajeva, nalaze samo dva brida. Ostali vrhovi nisu povezani, čime se program koji taj graf predstavlja može promatrati kao skup od četiri odvojena programa ($\{T0, T2, T5\}$, $\{T1\}$, $\{T3\}$ i $\{T4\}$). Takav slučaj ne analiziramo zbog toga što tada postupak LDCP ne može rasporediti sve izvršne jedinice (ostali postupci mogu), zbog toga što ne može dinamički stvarati najduži kritični put. No, taj problem je riješen u sustavu za raspoređivanje ubacivanjem **virtualnog (lažnog) čvora**, tako da graf postaje kao na slici 5.9c.

Postoje također i dvije dodatne izvedbe alata DAGGer: DAGGer-S, za generiranje parametara programa s prethodno generiranom strukturom i DAGGer-P, za generiranje parametara računalne platforme. DAGGer-S koristi pravila generiranja svih parametara grafova (osim strukture), opisanih u velikom broju radova koji opisuju raspoređivanje programa, poput [13, 37, 39–41, 85] te mnogih drugih. U većini je generiranje parametara izvršnih jedinica programa svedeno na sljedeće parametre:

- $|V| = \vartheta$ - broj izvršnih jedinica programa,
- $|P| = \rho$ - broj obradbenih jedinica računalne platforme,
- *ccr* - omjer komunikacije i obrade (eng. *communication-to-computation ratio*),
- η - čimbenik raznorodnosti (eng. *heterogeneity*),
- $\overline{x_G^{ob}}$ - prosječni obradbeni zahtjevi (grafa) programa (prosjek svih njegovih izvršnih jedinica).

Kao što je već spomenuto, ϑ predstavlja ukupan broj izvršnih jedinica koje tvore program, dok ρ predstavlja ukupan broj obradbenih jedinica sklopovlja na koje se program raspoređuje. Parametar *ccr* određuje omjer zbroja vrijednosti svih komunikacijskih poziva i zbroja obradbenih zahtjeva vrhova grafa, tj. omjer prosječnih komunikacijskih zahtjeva grafa $\overline{e_G}$ i prosječnih obradbenih zahtjeva grafa $\overline{x_G^{ob}}$:

$$ccr = \frac{\overline{e_G}}{\overline{x_G^{ob}}}. \tag{5.1}$$

Čimbenik raznorodnosti η (ili raspon vremena izvođenja na procesorima, prema [41]) jest mjera koliko vrijeme izvršavanja jedinice programa odstupa u odnosu na obradbenu jedinicu na kojoj se izvršava. Male vrijednosti η znače vrlo slična vremena izvođenja, (gotovo homogena platforma), dok veće vrijednosti označavaju značajnu razliku u tim vremenima (visoko raznorodna platforma). Srednja vrijednost obradbenih zahtjeva (vremena izvođenja i sl.) $\overline{x_i^{ob}}$ izvršne jedinice v_i se odabire nasumično uniformnom razdiobom u intervalu $\left[1,\ 2 \cdot \overline{x_G^{ob}}\right]$, gdje su $\overline{x_G^{ob}}$ prosječni obradbeni zahtjevi grafa programa definirani kao:

$$\overline{x_G^{ob}} = \sum_{v_i \in V} \frac{\overline{x_i^{ob}}}{\vartheta}. \tag{5.2}$$

Samo vrijeme izvođenja (ili obradbeni zahtjevi) $x_{i,j}^{ob}$ izvršne jedinice v_i na svakoj obradbenoj jedinici p_j odabire se nasumično iz intervala, (ovisno je o čimbeniku raznorodnosti):

$$\overline{x_i^{ob}} \cdot \left(1 - \frac{\eta}{2}\right) \leq x_{i,j}^{ob} \leq \overline{x_i^{ob}} \cdot \left(1 + \frac{\eta}{2}\right). \tag{5.3}$$

Parametri komunikacije među izvršnim jedinicama biraju se na način identičan kao i obradbeni parametri. Prvo, s obzirom na to da su *ccr* i $\overline{x_G^{ob}}$ zadani, iz (5.1) dobiva se

$$\overline{e_G} = ccr \cdot \overline{x_G^{ob}}, \tag{5.4}$$

zatim se srednja vrijednost komunikacije $\overline{e}$ između izvršnih jedinica bira nasumično iz intervala $[0,\ 2 \cdot \overline{e_G}]$, uniformnom razdiobom. Vrijednost komunikacije za svaki par izvršnih jedinica v_i i v_j između kojih postoji komunikacija (vrijednost u matrici susjedstva na (i, j) je veća od 0) bira se pomoću (5.3), supstitucijom $\overline{x_i^{ob}}$ sa $\overline{e}$ te $x_{i,j}^{ob}$ sa $e_{i,j}$:

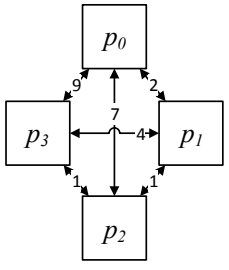
$$\overline{e} \cdot \left(1 - \frac{\eta}{2}\right) \leq e_{i,j} \leq \overline{e} \cdot \left(1 + \frac{\eta}{2}\right). \tag{5.5}$$

Dakle, raznorodnost u generiranju slučajeva alatom DAGGer utječe i na razlike u komunikaciji.

DAGGer-P je specifičan po tome što generira raznorodne komunikacijske kanale između obradbenih jedinica simulirane računalne platforme, što većina navedenih postupaka ne podrazumijeva pri konfiguraciji testnih slučajeva. Zasniva se na principu sličnom generiranju komunikacijskih zahtjeva među izvršnim jedinicama programa opisanog u algoritmu 8, osim što podrazumijeva da su komunikacijski kanali simetrični. Drugim riječima, komunikacijski kanali omogućuju istu propusnost u oba smjera i omogućuju da samo jedan smjer može biti aktivan u isto vrijeme, čime smo zadovoljili polazne pretpostavke postupka RaNDKiP dane u pododjeljku 4.2. Dodatni parametar koji uvodi DAGGer-P jest maksimalna propusnost komunikacijske veze c_{max}, dok se propusnost komunikacijskoga kanala $c_{i,j}$ između obradbenih jedinica p_i i p_j bira nasumično uniformnom razdiobom u intervalu $[1,\ c_{max}]$. Slika 5.10 prikazuje primjer parametara platforme generirane alatom DAGGer-P, gdje se na slici 5.10a vidi simetričnost komunikacijskih kanala, (simetrija u odnosu na glavnu dijagonalu). Vrijednosti na glavnoj dijagonali nisu definirane, jer se propusnost unutar iste obradbene jedinice smatra puno većom od propusnosti između različitih obradbenih jedinica (teoretski beskonačna), tako da su svi komunikacijski pozivi unutar iste obradbene jedinice jednaki nuli (što podrazumijevaju svi postupci raspoređivanja rangiranjem izvršnih jedinica programa). Prikaz platforme dan je slikom 5.10b, na kojoj se vidi i njezina struktura.

	p_0	p_1	p_2	p_3
p_0	-	2	7	9
p_1	2	-	1	4
p_2	7	1	-	1
p_3	9	4	1	-

(a) Matrica komunikacijskih kanala platforme.



(b) Vizualizacija platforme.

Slika 5.10: Primjer strukture platforme generirane DAGGer-P-om.

U slučaju generiranja miješanoga acikličkoga grafa alatom DAGGer postupak se sastoji od zamjene bridova u već generiranoj matrici susjedstva (za DAG). Za svaki se brid generira zastavica nasumično u intervalu $[0, 9]$ te u slučaju da je broj manji od 2, brid se obilježava kao komunikacijski brid (približno

80% bridova tako postaju lukovi, dok 20% njih ostaju neusmjereni bridovi). Ostali bridovi ostaju samo oznaka vremenskog slijeda (nemaju komunikaciju).

5.1.5 Razvijeni alati: simulacijsko okruženje SynGraph

U svrhu što boljeg vrednovanja rezultata dobivenih vlastitim postupcima raspoređivanja programa na računalne platforme te kako bi postigli cilj 4 našeg istraživanja, razvili smo simulacijsko okruženje **SynGraph**. Navedeno okruženje omogućuje simulaciju programa zadanog u obliku usmjerenih acikličkih grafova (DAG-ova) ili miješanih acikličkih grafova na stvarnim paralelnim i raspodijeljenim računalnim platformama koje podržavaju paradigmu izmjene poruka (MPI).

Jedno od glavnih svojstava okruženja SynGraph jest i nadopunjavanje postupka RaPPaMaG mehanizmom koji sam stvara raspored jedinica programa i odmah ih izvodi prema tom rasporedu, (zamjenjuje fazu 2 postupka, raspoređivanje faza izvršavanja)

Glavni koraci okruženja SynGraph prikazani su algoritmom 9. Postupak kao ulaz dobiva raspored dobiven nekim od postupaka raspoređivanja (inačice postupka RaNDKiP, HEFT, LDCP) ili vektor dodijeljenosti kod postupka RaPPaMaG (koja izvršna jedinica programa je dodijeljena kojoj obradbenoj jedinici sklopovlja), (linija 1). Nakon toga, okruženje stvara broj procesa jednak broju obradbenih jedinica prema pretpostavkama raspoređivanja i svaki postavlja na zasebnu obradbenu jedinicu sklopovlja (linije 3—5). Sljedeći korak je stvaranje "brave" za svaku jedinicu programa, (linije 6–10). "Brava" je skup manjih "bravica" za svakog prethodnika trenutno analizirane jedinice programa. Dakle, za određenu jedinicu programa potrebno je otključati sve bravice kako bi ona mogla nastaviti s radom, (svi prethodnici moraju završiti izvođenje). Nakon toga svaki proces prima jedinice programa koje su mu dodijeljene (koje su raspoređene na njega, tj. na odgovarajuću obradbenu jedinicu sklopovlja), (linija 13). Sada istovremeno započinje izvođenje svih jedinica programa, s tim da samo one s otključanom bravom mogu krenuti s izvršavanjem. Ukratko, faze kroz koje prolazi svaka jedinica programa su:

1. **faza:** primanje signala za otključavanje bravica (linija 16),
2. **faza:** izvršavanje obradbenih zahtjeva, izvođenje (linija 21),
3. **faza:** komunikacija (slanje i primanje potrebnih podataka, ako je potrebno), linija 22,
4. **faza:** slanje "otključaj" signala svim sljedbenicima, prema grafu programa (linija 23).

Kako jedinice programa završavaju izvođenje, tako se otključavaju brave drugih i one nastavljaju s radom (linije 15–20). Na taj se način sam program izvršava na najučinkovitiji mogući način i jedinice programa kreću s izvođenjem čim su za to preduvjeti ispunjeni.

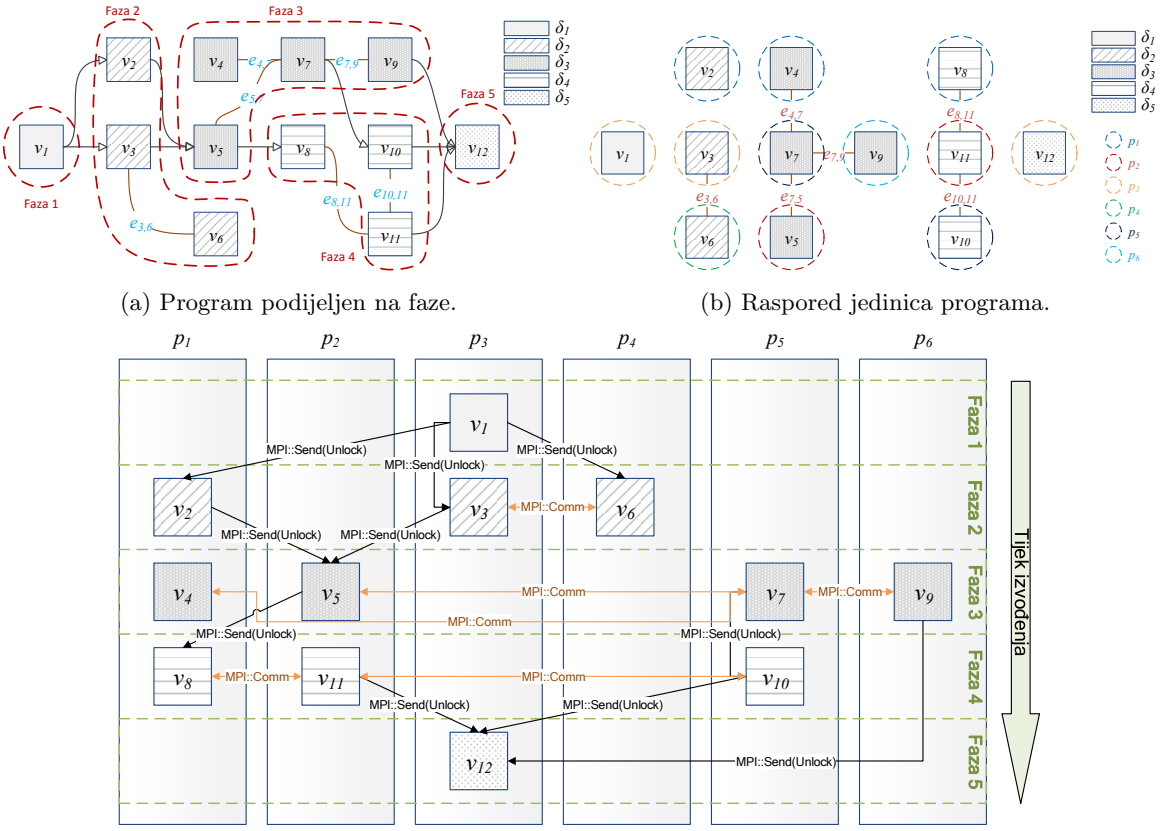
Navedeni postupak prikazan je i na slici 5.11 kroz primjer programa u obliku miješanoga grafa raspoređenog na faze izvršavanja pomoću postupka RaPPaMaG sa slike 5.11a. Pojedinim jedinicama programa dodijeljene su obradbene jedinice sklopovlja (dobiveni raspored) prema slici 5.11b. Nakon toga, ti podatci se daju simulacijskom okruženju SynGraph i program se izvodi na način prikazan slikom 5.11c. Dakle, pokreću se pojedinačni procesi na obradbenim jedinicama $p_1, ..., p_6$, svakom od kojeg se dodijele odgovarajuće jedinice programa prema ulaznom rasporedu.

Algoritam 9 SynGraph, glavna funkcija.

```
1: P ← Raspored čitavog skupa obradbenih jedinica
2: V ← skup izvršnih jedinica programa
3: za sve p ∈ P radi
4:    kreiraj proces p^virt i dodijeli ga obradbenoj jedinici p
5: završi za
6: za sve v_j ∈ V radi
7:    za sve v_k ∈ pred(v_j) radi
8:       b_j = b_j + 1 {kreiraj bravu za svakog neposrednog prethodnika}
9:    završi za
10: završi za
11: P^virt ← skup svih procesa
12: za sve p^virt ∈ P^virt radi
13:    V_i ← Skup jedinica programa dodijeljenih obradbenoj jedinici p_i
14:    za sve v_j ∈ V_i radi
15:       ponavljaj
16:          započni primanje "otključaj" signala od svih prethodnika
17:          za svaki primljeni "otključaj" signal radi
18:             b_j = b_j − 1
19:          završi za
20:       sve dok nije b_j = 0
21:       započni s radom
22:       obavi komunikacijske pozive
23:       pošalji "otključaj" signal svim sljedbenicima
24:    završi za
25: završi za
```

Tako se tvore redovi jedinica programa na svakom procesu i za svaki se simultano pokreće procedura izvršavanja. Na taj se način prvo pokreću ulazne jedinice programa (bez prethodnika u grafu) te kako one završavaju s izvođenjem, otključavaju bravice njihovih sljedbenika pomoću MPI komunikacijskih poziva, koji to čine svojim sljedbenicima itd. Komunikacija među jedinicama programa odvija se također unutar MPI komunikacijskih poziva ovisno o topologiji koja je potrebna. To mogu biti komunikacijski pozivi od točke do točke, (eng. *point to point*) i kolektivni komunikacijski pozivi, (eng. *collective communication calls*), poput *broadcast*, *reduce*, *all-to-all* itd., prema [45]. Zasad SynGraph podržava samo sinkronu komunikaciju (prema zahtjevima postupka RaPPaMaG), no u izradi je i mogućnost asinkrone komunikacije među jedinicama programa.

Još jedna od prednosti simulacijskog okruženja SynGraph je i **mogućnost inkapsulacije različitih obradbenih struktura** za izvršne jedinice programa. Naime, faza 2 kroz koju prolaze sve izvršne jedinice programa jest izvršavanje njezinih obradbenih zahtjeva. Oni mogu biti razne naravi, poput cjelobrojnih operacija (instrukcija), operacija s pomičnim zarezom (eng. *floating-point*) i raznih struktura poput osnovnih potprograma linearne algebre (eng. *basic linear algebra subprograms*, BLAS) iz biblioteke LAPACK (eng. *linear algebra package*) opisane u [110] ili njezine skalabilne verzije SCALAPACK (eng. *scalable linear algebra package*) opisane u [111]. Ono što je posebno važno jest da se sve jedinice programa izvršavaju unutar **obradbenih spremnika**, tj. strukture u koju se mogu ugraditi bilo kakvi oblici opterećenja. To svojstvo moguće je povezati s naporima da se stvore univerzalni repozitoriji komponenti za paralelne programe, poput već spomenutih **Berekeleyovih patuljaka** iz [56, 98] ili instance MICA okvira iz [14].



(a) Program podijeljen na faze.

(b) Raspored jedinica programa.

(c) Tijek i struktura izvođenja.

Slika 5.11: Program raspoređen postupkom RaPPaMaG u izvođenju kroz okruženje SynGraph.

Okruženje SynGraph koristit ćemo u slučajevima kada ne možemo destilirati apsolutne pokazatelje kvalitete, kao što je to slučaj kod raspoređivanja miješanih grafova.

Vlastitim alatom SynGraph završili smo s opisom važnijih alata korištenih u eksperimentalnoj analizi. Osim načina na koji možemo generirati testne slučajeve, važno je odrediti i njihove postavke i referentne vrijednosti, što opisujemo u dolazećem pododjeljku.

5.2 Postavke testnih slučajeva (eksperimenta)

Vođeni potrebom za što boljim preslikavanjem stvarnih primjera iz područja HPC-a u testne slučajeve, definiramo postavke eksperimenata inspiriranih postojećim istraživanjem, kao i prilagodbama tih postavki za analizu slučajeva iz našeg istraživanja. Postavke testnih slučajeva dijele se na:

- parametre paralelnih programa, (njihovih izvršnih jedinica i njihove međuovisnosti),
- parametre računalne platforme, (njezinih obradbenih jedinica i komunikacije među njima) te
- postavke postupaka raspoređivanja.

Većina parametara se odabire u intervalima koji se koriste u relevantnom području istraživanja radi lakše usporedbe, osim kada neki od njih uzrokuje određeni problem za naše testne slučajeve. Ti problemi mogu biti izvedbene naravi, zatim nedostatak mogućnosti preslikavanja stvarnih slučajeva koje želimo

pokriti našim istraživanjem na naše eksperimente ili slučaj kada ti parametri ne donose relevantne informacije. Sve ćemo važnije parametre opisati narednim pododjeljcima.

Parametri programa: broj izvršnih jedinica programa, ϑ

Broj izvršnih jedinica glavni je parametar koji obilježava testne slučajeve i izravno utječe na performanse postupaka njihovog raspoređivanja, kao i na same performanse programa u izvođenju koji se od njih sastoji. Za očekivati je da će veći broj izvršnih jedinica programa negativno utjecati na performanse postupaka raspoređivanja. No, taj utjecaj može biti smanjen u slučaju da izvršne jedinice imaju male obradbene zahtjeve i kada je komunikacija među njima slabog intenziteta i obujma. Referente vrijednosti koje se nalaze u literaturi variraju od nekoliko desetaka, pa sve do nekoliko tisuća. Kako bi eksperimenti bili konzistentni u ispitivanju vlastitih postupaka raspoređivanja RaPPaMaG i RaNDKiP, vrijednosti ϑ bit će prilagođene vrijednostima koje se za njih koriste u povezanoj literaturi.

Brojevi zadataka u povezanoj literaturi većinom se kreću od 20 do 100 [37,41,44], no ima i ekstremnijih primjera, 10 – 500 u [13], 100 – 500 u [70] ili čak i mnogo više (1024 – 131072 u [22], performansi radi) te manji broj (16 – 36 u [64]). S obzirom na to da su u intervalu $\vartheta \in [20, 200]$ i tri razmjera testnih primjera radnih opterećenja Pegasus (od 30, 50, 100 i 1000), odabrane vrijednosti u našoj eksperimentalnoj analizi su $\vartheta \in \{25, 50, 100, 200\}$. Za vrijednosti veće od 200 izvorni postupak LDCP ima vrlo veliko vrijeme izvođenja, tako da smo 200 odabrali kao gornju granicu. Međutim, kao što će rezultati eksperimenata pokazati, navedene vrijednosti za ϑ nude dobar uvid u karakteristike postupaka raspoređivanja.

Parametri programa: omjer komunikacije i obrade, *ccr*

Omjer komunikacije i obrade, kao što je već spomenuto, određuje koliki je udio komunikacije u opterećenju programa. Navedeni je parametar iznimno važan u slučajevima kada se ukupno trajanje komunikacije približava polovici ukupnog trajanja obrade u programu. Tada drastično pada učinkovitost postupaka raspoređivanja koji ne podrazumijevaju stvarni obujam komunikacije (RaNDKiP2, LDCP), ili uzimaju samo njezinu prosječnu vrijednost (HEFT). Vrijednosti za *ccr* u povezanoj literaturi kreću se između 0,1 i 10 [13, 41, 44], no postoje i manji intervali vrijednosti (u [40] između 1,2 i 3, od 1 do 2.5 u [39] ili jedna vrijednost od 5 u [64]). Zbog toga, uzeli smo interval koji pokriva većinu, $ccr \in \{0,5,\ 1,\ 5,\ 10\}$.

Parametri sklopovlja: broj obradbenih jedinica, ρ

Broj obradbenih jedinica simulirane računalne platforme na koju se raspoređuju programi drugi je najvažniji parametar koji utječe na performanse postupaka raspoređivanja, kako u vremenu izvođenja, tako i u optimalnosti rješenja. Zbog toga je i prisutan u gotovo svakoj analizi postupaka raspoređivanja programa na sklopovlje i izravni je pokazatelj skalabilnosti, kako programa koji se raspoređuju, tako i samih postupaka raspoređivanja. Broj obradbenih jedinica sklopovlja u relevantnoj literaturi se u pravilu kreće između 2 i 32, primjerice u [3, 13, 37, 39, 41, 86] i to većinom kao potencija broja dva, kako se pojavljuju i u stvarnim računalnim sustavima. Iz tih razloga i mi smo odabrali vrijednosti $\rho \in \{4, 8, 16, 32\}$. Važno je napomenuti da u povezanoj literaturi postoje i drugi intervali, primjerice od 20 do 60 u [51], 2

– 64 u [44], 2 – 128 u [64], do 1680 u [17] ili čak do 4096 u [35], no oni su često vezani uz vrlo specifične slučajeve ili slučajeve koji ispituju složenost različitih postupaka raspoređivanja.

Parametri sklopovlja: čimbenik raznorodnosti, η

Čimbenik raznorodnosti upućuje na to koliko se razlikuju vremena izvršavanja jedinica programa u odnosu na to na kojoj obradbenoj jedinici se izvršavaju. Veća vrijednost čimbenika raznorodnosti, kao što i samo ime kaže, znači veću raznorodnost računalne platforme (veće razlike u računalnoj snazi njezinih obradbenih jedinica). Za buduće računalne platforme ovakav je pristup realan (vrlo različite obradbene jedinice), kao i za velike, zemljopisno-raspoređene računalne sustave (veliki spletovi računala). Uobičajene vrijednosti u povezanoj literaturi kreću se između 0,1 i 2, kao u [13], no postoje i manje vrijednosti (0,1 do 0,8 u [37], 0,1 do 1 u [41]), ali i ekstremnije vrijednosti (100 – 200 u [44]). Stoga, za potrebe naših eksperimenata uzete su vrijednosti $\eta \in \{0,1,\ 0,5,\ 1,\ 5\}$.

Ostali parametri: $\overline{x_G^{ob}}$, c_{max}, $e_{i,j}$, x^{ob}, y^{ob}, $t_{i,j}^{izv}$

Od ostalih parametara platforme ističemo one specifične našem problemu te one koji nemaju velik značaj na performanse.

Za postupak RaPPaMaG to su obradbeni zahtjevi izvršnih jedinica programa x^{ob} (i njihovog prosjeka u cijelom grafu programa — $\overline{x_G^{ob}}$) u obliku broja instrukcija koje trebaju obaviti, čime se izravno nadovezujemo na neku od povezane literature (mega ciklusa u sekundi u bežičnim senzorskim mrežama, prema [51, 97]). Za postupke koji raspoređuju programe u obliku DAG-a (RaNDKiP, HEFT i LDCP), taj parametar je zadan u obliku vremena izvršavanja $t_{i,j}^{izv}$ jedinice programa v_i na obradbenoj jedinici p_j, no prikazujemo ga također kao obradbene zahtjeve $x_{i,j}^{ob}$.

Tako dolazimo i do obradbene snage obradbenih jedinica y^{ob} u obliku instrukcija koje mogu obaviti u sekundi. Ta se mjera može općenito promatrati kao zajednička za mjerne veličine obradbene snage računalnih sustava poput mega instrukcija po sekundi (eng. *mega instructions per second*, MIPS), prema [16] i jedne od najpopularnijih — operacija s pomičnim zarezom u sekundi (eng. *floating point operations per second*, FLOPS) — prema [77], koja se koristi i u rangiranju superračunala na listama TOP500, [7] i GREEN500, [8].

Nadalje, to su parametri komunikacije u programu u obliku količine podataka $e_{i,j}$ koja se izmjenjuje među izvršnim jedinicama programa v_i i v_j te prosječne vrijednosti komunikacije u grafu $\overline{e_G}$, opisanih u pododjeljku 5.1.4, a koji se koriste i za RaPPaMaG i za RaNDKiP te u relevantnoj literaturi [13, 37, 41] itd.

Definiramo i parametre komunikacije računalne platforme u obliku maksimalne propusnosti komunikacijskoga kanala c_{max} također opisane u pododjeljku 5.1.4. Većina ostalih parametara programa i sklopovlja nema veliki značaj, ili su usko specifični za primjenu.

Vrijednosti koje postavljamo za navedene parametre su:

- parametar prosječnih obradbenih zahtjeva grafa programa $\overline{x_G^{ob}}$ postavljen je na 50 (tako da su prosječni obradbeni zahtjevi njegovih izvršnih jedinica $x_{i,j}^{ob}$ u intervalu $[0, 100]$),

- za maksimalnu propusnost komunikacijskoga kanala c_{max} odabrana je vrijednost 100, jer na taj način vrijednost komunikacijskih zahtjeva može varirati između vremena identičnih vremenu obrade, (prema načinu generiranja alata DAGGer, pododjeljak 5.1.4), do 100 puta manjih vrijednosti (ukupno trajanje komunikacije jednako je omjeru komunikacije između izvršnih jedinica i propusnosti kanala obradbenih jedinica na koje su raspoređeni, prikazano u pododjeljku 3.3).

Većina vremenskih parametara ujedno je i navedena u pododjeljku 5.1.4.

Parametri utroška energije: y^{en}, f, v

Parametri utroška energije definiraju se samo za računalnu platformu i to u obliku simboličke mjere poput utroška energije po izvršenoj instrukciji y^{en}, čime možemo dobro opisati energetske karakteristike platforme, a što je i opisano u literaturi, primjerice [1, 75, 112, 113], (broj operacija po vatu — FLOP/W). Za postupke raspoređivanja na temelju rangiranih listi i njihovoj usporedbi s postupkom RaNDKiP upotrebljavamo parametre radne frekvencije obradbene jedinice f te njezinog radnog napona v. Navedenim parametrima dolazimo do utroška energije pomoću DVFS karakteristika obradbenih jedinica opisanih pomoću (4.66) do (4.70) u pododjeljku 4.2.4, uz poznavanje DVFS tablice poput one u [4]. Navedeni parametri su temelj izračuna utroška energije i u vezanim istraživačkim radovima koji opisuju energetski-svjesno raspoređivanje u sustavima s DVFS-om, primjerice u [3, 11, 39, 44, 84, 85].

Parametri i postavke postupaka raspoređivanja

Kod razvijenih postupaka raspoređivanja (RaPPaMaG, RaNDKiP), ali i postojećih (HEFT, LDCP), neke postavke mogu imati značajne razlike u optimalnosti pronađenih rješenja ovisno o tipu primjene. Primjerice, za već opisani postupak RaNDKiP u eksperimentalnoj analizi upotrijebljene su dvije njegove inačice. Glavna inačica RaNDKiP primjenjuje postupak **dinamičke izrade ranga** izvršnih jedinica (ovisno o komunikacijskom kanalu između obradbenih jedinica na koje su dodijeljeni, opisano u pododjeljku 4.2.1), ali i **dinamičko ažuriranje matrice susjedstva**, opisano u pododjeljku 4.2.3. No, ispitane su i njihove izvedbe bez svake od navedenih karakteristika RaNDKiP1 i RaNDKiP2. Navodimo karakteristike svake inačice RaNDKiP postupka:

- **glavna inačica RaNDKiP**: primjenjuje dinamičku izradu ranga i dinamičko ažuriranje matrice susjedstva,
- **prva izvedena inačica RaNDKiP1**: bez dinamičke izrade ranga, uz dinamičko ažuriranje matrice susjedstva i
- **druga izvedena inačica RaNDKiP2**: bez dinamičke izrade ranga, bez dinamičkog ažuriranja matrice susjedstva.

S obzirom na to da postojeći postupci raspoređivanja LDCP i HEFT (ali i mnogi drugi) ne tretiraju komunikaciju jednako kao naše inačice postupka RaNDKiP, potrebno ih je ujednačiti kako bi ih mogli izravno usporediti u eksperimentalnoj analizi. Postupak LDCP, kao što je i opisano u pododjeljku 4.2 i prema originalnom izvoru [37] nema mogućnost ubacivanja komunikacije u raspored. S druge strane, postupak HEFT, kao što je opisano u pododjeljku 2.3.5 i originalnom izvoru [41, 42] podrazumijeva

komunikaciju koja se može odvijati paralelno (ukoliko postoji više komunikacijskih poziva u nizu za jednu izvršnu jedinicu programa). S obzirom na to da se u stvarnim sustavima koje mi analiziramo komunikacija odvija bez preklapanja te u svrhu približavanja dobivenih rezultata situaciji kakva će se odvijati stvarnim programa prema dobivenom rasporedu, potrebno je taj raspored prilagoditi. Iz tih razloga, razvili smo pomoćni postupak DoSK, (**do**davanje **s**tvarne **k**omunikacije u raspored), koji se sastoji od algoritama 10 i 11 te u dobiveni raspored ubacuje komunikaciju kakva bi se odvijala pri stvarnom izvođenju programa prema tom rasporedu.

U prvom dijelu postupka DoSK, opisanog algoritmom 10, predaje se raspored dobiven bilo kojim od postupaka raspoređivanja i vrši se analiza za raspored svake od obradbenih jedinica zasebno, (linije 1–4). Zatim se uklanjaju postojeći komunikacijski isječci (ako ih ima, što je slučaj za HEFT), (linija 6). Uklanjanjem komunikacijskoga isječka mora se ažurirati i vrijeme početka i završetka odgovarajućeg obradbenog isječka, (linije 11 i 12), no prije toga treba preskočiti sve naredne komunikacijske isječke ako postoje, (linije 8–10). Ažuriranjem vremena početka i završetka odgovarajućeg obradbenog isječka potrebno je ažurirati isto i za sve njegove sljedbenike, pa se poziva funkcija $privuci\_sljedbenike(j)$ koja rekurzivno ažurira vremena početka i završetka sljedbenika, ali i svih njihovih narednih sljedbenika, (linije 13–17).

Algoritam 10 Dodavanje stvarne komunikacije u raspored, prvi dio - DoSK1.

1: $P \leftarrow$ Raspored čitavog skupa obradbenih jedinica
2: **za sve** $p \in P$ **radi**
3: $S_i \leftarrow$ Skup vremenskih isječaka rasporeda obradbene jedinice p_i
4: **za sve** $s \in S_i$ **radi**
5: **ako** s_j je komunikacijski isječak **onda**
6: ukloni s_j iz rasporeda
7: $k = j$
8: **dok** s_{k+1} nije obradbeni isječak **radi**
9: $k = k + 1$
10: **završi dok**
11: $t_k^{svp} = t_k^{svp} - t_j^{izv}$ {ažuriraj vrijeme početka odgovarajućeg obradbenog isječka}
12: $t_k^{svz} = t_k^{svz} - t_j^{izv}$ {ažuriraj vrijeme završetka odgovarajućeg obradbenog isječka}
13: $privuci\_sljedbenike(k)$:
14: **dok** $sljed(s_k) \neq \emptyset$ **radi**
15: $privuci\_sljedbenike(sljed(s_k))$
16: $t_{sljed(s_k)}^{svp} = t_{sljed(s_k)}^{svp} - t_j^{izv}$
17: **završi dok**
18: **završi ako**
19: **završi za**
20: **završi za**

Drugi dio algoritma, algoritam 11, služi za dodavanje stvarne komunikacije u raspored u kojem se sada ne nalaze komunikacijski isječci. Analizira se također raspored svake od obradbenih jedinica zasebno (linije 2 i 3) te se za svaki obradbeni isječak provjerava skup njegovih prethodnika i, ukoliko su oni dodijeljeni drugoj obradbenoj jedinici (linija 6), ažuriraju se njegova vremena početka i završetka (linije 8–10). Ažurirano vrijeme je sada stvarno vrijeme dobiveno dijeljenjem količine podataka koja se treba prenijeti između obradbenog isječka i njegovog prethodnika $e_{pred(s_j),j}$ te komunikacijske propusnosti između obradbenih jedinica na koje su dodijeljeni $c_{k,j}$ (linija 8). Sada je potrebno i rekurzivno ažurirati ta vre-

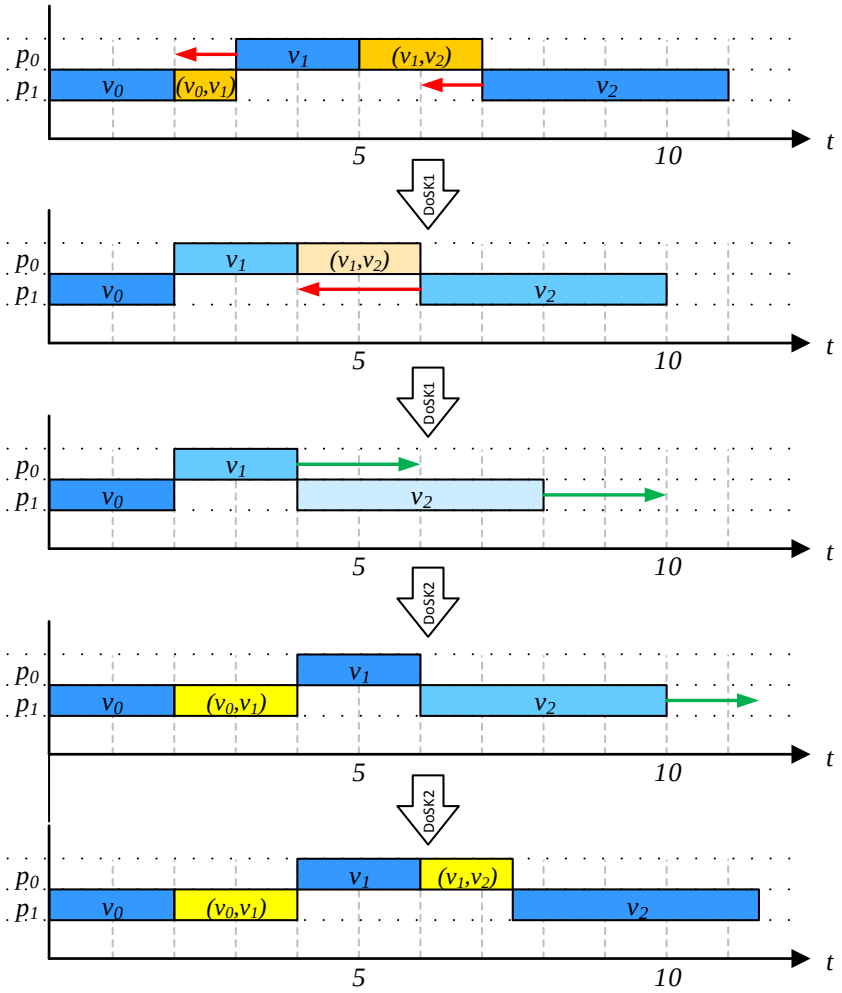
mena i za sve prethodnike analiziranog obradbenog isječka, čemu služi funkcija $poguraj\_sljedbenike(j)$ (linije 11–16).

Algoritam 11 Dodavanje stvarne komunikacije u raspored, drugi dio - DoSK2.

1: $P \leftarrow$ Raspored čitavog skupa obradbenih jedinica
2: **za sve** $p \in P$ **radi**
3: **za sve** $s \in S_i$ **radi**
4: **ako** s_j je obradbeni isječak **onda**
5: **za sve** $pred(s_j)$ **radi**
6: **ako** (p_k dodijeljen $pred(s_j)$) $\neq$ (p_l dodijeljen s_j) **onda**
7: stvori komunikacijski isječak z između $pred(s_j)$ i s_j
8: $t_z^{izv} = \frac{e_{pred(s_j),j}}{\beta_{k,l}}$ {postavi trajanje komunikacijskoga isječka}
9: $t_j^{svp} = t_j^{svp} + t_z^{izv}$ {ažuriraj vrijeme početka obradbenog isječka j}
10: $t_j^{svz} = t_j^{svz} + t_z^{izv}$ {ažuriraj vrijeme završetka obradbenog isječka j}
11: $poguraj\_sljedbenike(j)$:
12: **dok** $sljed(s_j) \neq \emptyset$ **radi**
13: $poguraj\_sljedbenike(sljed(s_j))$
14: $t_{sljed(s_j)}^{svp} = t_{sljed(s_j)}^{svp} + t_z^{izv}$
15: $t_{sljed(s_j)}^{svz} = t_{sljed(s_j)}^{svz} + t_z^{izv}$
16: **završi dok**
17: **završi ako**
18: **završi za**
19: **završi ako**
20: **završi za**
21: **završi za**

Pomoćni postupak DoSK prikazan je i slikom 5.12, na kojoj su vidljive tri izvršne jedinice programa v_0, v_1 i v_2 raspoređene na dvije obradbene jedinice p_0 i p_1. U prvom dijelu, DoSK1, uklanjaju se komunikacijski isječci iz rasporeda jedan po jedan te se svi navedeni sljedbenici primiču početku rasporeda za iznos te komunikacije. Zatim se ubacuje stvarna komunikacija (DoSK2), također svaka posebno te se ažuriraju vremena obradbenih isječaka koji predstavljaju sljedbenike obradbene jedinice za koju se ažurira komunikacija.

Općeniti algoritam za sve postupke raspoređivanja prikazan je algoritmom 12. Prvi je korak generiranje testnih slučajeva, koji uključuje generiranje parametara i programa i sklopovlja (linija 1). Zatim se za svaki generirani slučaj provode analizirani postupci raspoređivanja, što se ponavlja deset puta kako bi mjerenja bila konzistentnija (linija 3–16). Za svaki se postupak provodi DoSK, kako bi dobiveni rasporedi prikazivali stvarnu situaciju koja se može očekivati izvršavanjem programa prema njemu i bili međusobno usporedivi. Naposljetku, rade se prosjeci svih analiziranih mjerenih veličina (linija 17).



Slika 5.12: Skica postupka DoSK (DoSK1 i DoSK2).

Algoritam 12 Osnovna struktura algoritama raspoređivanja.

```
1: G ← generirani skup testnih slučajeva nekim od postupaka (DAGGer, Pegasus, TGFF)
2: za sve g ∈ G radi
3:     i ← 10
4:     ponavljaj
5:         rasporedi g postupkom RaNDKiP
6:         pozovi DoSK1 i DoSK2 za dobiveni raspored
7:         rasporedi g postupkom RaNDKiP1
8:         pozovi DoSK1 i DoSK2 za dobiveni raspored
9:         rasporedi g postupkom RaNDKiP2
10:        pozovi DoSK1 i DoSK2 za dobiveni raspored
11:        rasporedi g postupkom HEFT
12:        pozovi DoSK1 i DoSK2 za dobiveni raspored
13:        rasporedi g postupkom LDCP
14:        pozovi DoSK1 i DoSK2 za dobiveni raspored
15:        i = i − 1
16:    sve dok nije i = 0
17:    usporedi prosjek dobivenih rezultata
18: završi za
```

Zadnji dio koji opisuje postavke testnih slučajeva za eksperimentalnu analizu je odabir referentnih intervala svih korištenih parametara koje predstavljamo idućim pododjeljkom.

Referentni intervali ulaznih parametara testnih slučajeva

U ovom pododjeljku navodimo referentne vrijednosti ulaznih parametara testnih slučajeva. Neki od njih su već navedeni ranije u odgovarajućim pododjeljcima, no ovdje ih objedinjujemo tablicom 5.2 kako bi svi bili na jednom mjestu. Također, za sve su vrijednosti iz navedene tablice dana obrazloženja u odgovarajućim pododjeljcima i kroz tekst, tako da neće biti objašnjavani ovdje.

Tablica 5.2: Vrijednosti i intervali svih ulaznih parametara korištenih u eksperimentalnoj analizi.

		RaPPaMaG		RaNDKiP	
		vrijednosti	korak	vrijednosti	korak
Programi	ϑ	$\{25, 39, 50, 100, 200\}$	-	$\{25, 50, 100, 200\}$	$\times 2$
	$\vartheta_{Pegasus}$	$\{25, 50, 100\}$	$\times 2$	$\{25, 50, 100\}$	$\times 2$
	ccr	1,0	fiksno	$\{0,5,\ 1,\ 5,\ 10\}$	-
	$\overline{x_G^{ob}}$	50	fiksno	50	fiksno
	$\overline{x_i^{ob}}$	$\overline{x_G^{ob}}$	fiksno	$\left[1,\ 2 \cdot \overline{x_G^{ob}}\right]$	nasumično
	$x_{i,j}^{ob}$	$x_i^{ob} = \overline{x_i^{ob}} = \overline{x_G^{ob}}$	fiksno	prema (5.3)	nasumično
	$\overline{e_G}$	prema (5.4)	-	prema (5.4)	-
	$\overline{e}$	$[0,\ 2 \cdot \overline{e_G}]$	nasumično	$[1,\ 2 \cdot \overline{e_G}]$	nasumično
	$e_{i,j}$	prema (5.5)	nasumično	prema (5.5)	nasumično
Sklopovlje	ρ	12	fiksno	$\{4, 8, 16, 32\}$	potencija broja 2
	η	1,5	fiksno	$\{0,1,\ 0,5,\ 1,\ 5\}$	-
	c_{max}	5	fiksno	$\{1, 100\}$	-
	$c_{i,j}$	$[1,\ c_{max}]$	nasumično	$[1,\ c_{max}]$	nasumično
	$\overline{y_P^{ob}}$	$[1,\ 100]$	nasumično	-	-
	y_j^{ob}	slično (5.3)	nasumično	-	-
	y_j^{ob}	prema (5.3) uz $\overline{x_i^{ob}} \rightarrow \overline{y_P^{ob}}$ i $x_{i,j}^{ob} \rightarrow y_j^{ob}$	nasumično	-	-
	$\overline{y_P^{en}}$	4,5	fiksno	-	-
	y_P^{en}	$[3,\ 6]\,,\ [1,\ 100]$	nasumično	-	-
	f	-	-	$[0,\ 1]$	određuje LVI
	υ	-	-	prema (4.67)	određuje LVI
Važn.	$w(t^{uk})$	$\{0,8,\ 0,2\}$	-	-	-
	$w(e^{uk})$	$\{0,2,\ 0,8\}$	-	-	-

Za testne slučajeve iz repozitorija programa Pegasus i onih generiranih alatom TGFF postoji dodatni parametar, a to je tip strukture programa, oblici kojeg su navedeni u pododjeljku 5.1.2. S obzirom na to da je u istom pododjeljku navedeno da su veličine programa za preuzimanje fiksne i to $\vartheta \in \{\approx 25, \approx 50, \approx 100, \approx 1000\}$ i uz ograničenja postupaka raspoređivanja dodajemo još jednu pretpostavku. Naime, kako performanse postupka LDCP (vrijeme izvršavanja) drastično padaju porastom broja izvršnih jedinica programa, postavili smo gornju granicu na $\vartheta = 200$, tako da ne analiziramo vrlo velike razmjere programa iz repozitorija Pegasus ($\vartheta \approx 1000$). Shodno tome, ostaju vrijednosti prikazane tablicom 5.3 i one su razmatrane u eksperimentalnoj analizi.

Za programe generirane alatom TGFF postoje također predlošci po kojima su generirani, navedeni u odgovarajućem pododjeljku 5.1.1. S obzirom na to da strukture uvjetuju i neka odstupanja od broja izvršnih jedinica programa (karakteristika alata), razmjeri svakog od predložaka navedeni su tablicom

Tablica 5.3: Postavke parametara programa iz repozitorija Pegasus za eksperimentalnu analizu.

	Broj izvršnih jedinica u programu, ϑ		
Karakteristika	Mali razmjeri	Srednji razmjeri	Veliki razmjeri
CyberShake	30	50	100
Epigenomics	24	46	100
LIGO inspiral	30	50	100
Montage	30	60	100
SIPHT	25	50	100

5.4.

Tablica 5.4: Postavke parametara programa generiranih alatom TGFF za eksperimentalnu analizu.

	Broj izvršnih jedinica u programu, ϑ			
Karakteristika	Mali razmjeri	Srednji razmjeri	Veliki razmjeri	Vrlo veliki razmjeri
kbasic_task	27	52	102	203
kseries_parallel	-	41	104	208
kseries_parallel_xover	-	41	104	208
packets	27	49	100	203
simple	25	50	100	200
sp_rand	24	49	99	199

Iz navedene tablice vidi se kako za strukture programa *kseries_parallel* nije moguće generirati male razmjere programa ($\vartheta \approx 25$), tako da taj slučaj nismo analizirali.

Nakon što smo definirali postavke svih parametara važnih za izvođenje eksperimenata u analizi, predstavit ćemo i parametre kvalitete u obliku mjerenih veličina koje pokazuju performanse unaprijeđenih postupaka raspoređivanja. Njih opisujemo narednim pododjeljkom.

5.3 Mjerene veličine i pokazatelji kvalitete

Veličine mjerene u eksperimentalnoj analizi zasnovane su prvenstveno na primjerima u praksi, ali kao što je to slučaj i kod parametara postavki, oni su uvjetovani i performansama koje želimo istaknuti. Mjerene veličine ujedno nazivamo i **pokazateljima kvalitete**, jer njihove vrijednosti odlučuju da li je neki postupak dobar, tj. daje li željene rezultate te koliko je neki postupak bolji od drugoga. Pokazatelje kvalitete dijelimo ovisno o postupku koji ih polučuje na mjerene veličine za postupak RaPPaMaG i one za inačice postupka RaNDKiPte prema pojedinim kriterijima kvalitete na vremenske karakteristike (pokazatelje kvalitete) i karakteristike utroška električne energije.

5.3.1 Pokazatelji kvalitete vremenske analize

Pokazatelji kvalitete vremenske analize većinom su zasnovani na ukupnoj duljini rasporeda t^{rasp} kao glavnom pokazatelju kvalitete, od kojeg izvodimo i većinu ostalih. Mjereni vremenski parametri se dijele na:

- parametar 1: ukupna duljina rasporeda (eng. *schedule length, makespan, execution time*) — $\mathbf{t^{rasp}}$,

- parametar 2: udio duljine rasporeda (eng. *schedule length ratio*, SLR) — **SLR**,
- parametar 3: ubrzanje (eng. *speedup*) — **ubrzanje**,
- parametar 4: učinkovitost (eng. *efficiency*) — **učinkovitost**,
- parametar 5: **relativna razlika duljine rasporeda**.

Parametri t^{rasp}, SLR, ubrzanje i učinkovitost koriste se u većini literature vezane uz raspoređivanje programa na sklopovlje, posebice postupaka zasnovanih na rangiranju jedinica programa, primjerice u [13, 23, 37, 41], dok je relativna razlika duljine rasporeda pokazatelj kvalitete razvijen u sklopu ovog rada. U narednim pododjeljcima ćemo karakterizirati svaki od njih.

Ukupna duljina rasporeda t^{rasp}

Ukupna duljina rasporeda je pokazatelj koliko će ukupno trajati program ako se izvrši prema dobivenom rasporedu. Navedeni pokazatelj kvalitete često je simboličke naravi, jer su parametri pomoću kojih se dobiva također simboličke naravi. No, on je dobar pokazatelj omjera ukupnog utroška vremena i vremena izvršavanja pojedinih jedinica programa. Navedeni parametar u programima zadanima kao usmjereni aciklički graf (DAG) jednak je stvarnom vremenu završetka izlaznog vrha grafa t^{svz}_{izlaz}, (onog bez sljedbenika u grafu programa). U slučaju kada postoji više izlaznih čvorova $v_{izlaz,i} \in V_{izlaz}$, a nije korištena strategija ubacivanja virtualnog (lažnog) izlaznog čvora, ono je jednako onom s većim vremenom završetka, prema (2.16) iz pododjeljka 2.3.5. Kada se koristi strategija ubacivanja virtualnog čvora, tada je duljina rasporeda jednaka njegovom vremenu završetka. Problem kod korištenja duljine rasporeda kao pokazatelja kvalitete je njegova osjetljivost na raspon vrijednosti ulaznih parametara, posebno vremena izvršavanja jedinica programa x^{ob} na pojedinim obradbenim jedinicama sklopovlja te ukupnog broja jedinica programa ϑ. Zbog toga, primjerice, ne mogu se izravno uspoređivati rezultati raspoređivanja kod grafova različite veličine te mnogi autori uvode relativne mjere poput SLR-a koji je opisan u narednom pododjeljku.

Udio duljine rasporeda (SLR)

Udio duljine rasporeda (SLR), prema [13, 41] ili normalizirana duljina rasporeda (eng. *normalized schedule length*, NSL), prema [37], jest odgovor na problem međusobne usporedbe kvalitete rasporeda kod grafova različite veličine i/ili različitih prosječnih obradbenih zahtjeva $\overline{x^{ob}_G}$. SLR dobivenog rasporeda definiran je kao:

$$\mathrm{SLR} = \frac{t^{rasp}}{\sum\limits_{v_i \in KP_{min}} \min\limits_{p_j \in P}\{w_{i,j}\}}, \tag{5.6}$$

gdje je nazivnik jednak zbroju minimalnih obradbenih troškova jedinica programa na najkraćem kritičnom putu KP_{min}. On se dobiva tako da se izvršne jedinice programa rangiraju koristeći njihove minimalne obradbene troškove (uspoređujući ih na svim obradbenim jedinicama) i zatim se odabere najkraći kritični put (rang ulaznog vrha grafa s najmanjim rangom).

Vrijednost SLR-a dobivena bilo kojim postupkom ne može biti manja od 1, jer je nazivnik u (5.6) minimalna vrijednost duljine rasporeda koja se može dobiti. Performanse nekog postupka raspoređivanja

to su bolje što je manji SLR. U našoj eksperimentalnoj analizi korištene su prosječne vrijednosti SLR-a nekoliko različitih grafova, za koje je on izračunat kao srednja vrijednost na osnovu 10 ponavljanja.

Ubrzanje

Ubrzanje, kao najvažnija mjerna veličina pri analizi performansi paralelnih programa, koristi se i u svrhu analize performansi postupaka raspoređivanja. Naime, ono ukazuje na razliku između trajanja serijske verzije programa i njegove paralelizirane verzije, što podliježe dobro poznatim zakonima paralelizacije: Amdahlovom i Gustafsonovom, prema [114]. Navedeni omjer pokazatelj je učinkovitosti paralelizacije, kao i skalabilnosti programa, dok u konkretnom slučaju definira koliko je neki postupak raspoređivanja uspio učinkovito rasporediti program na sklopovlje.

Ubrzanje se definira kao:

$$ubrzanje = \frac{\min\limits_{p_j \in P} \left\{ \sum\limits_{v_i \in V} w_{i,j} \right\}}{t^{rasp}}, \tag{5.7}$$

gdje je brojnik razlomka serijsko (slijedno) vrijeme izvršavanja programa, dok je nazivnik paralelno vrijeme izvršavanja programa. Slijedno vrijeme izvršavanja određuje se tako da se sve jedinice programa dodijele istoj obradbenoj jedinici i to onoj na kojoj je ukupno vrijeme njihovog izvršavanja minimalno. Što je veća vrijednost ubrzanja, to je bolji raspored koji je dao postupak raspoređivanja.

Učinkovitost

Učinkovitost je pokazatelj kvalitete koji se izravno nadovezuje na ubrzanje. Učinkovitost je definirana kao omjer ubrzanja i broja procesora:

$$ucinkovitost = \frac{ubrzanje}{\rho}. \tag{5.8}$$

Učinkovitost je manja što je veći broj obradbenih jedinica i idealno bi bilo kada bi se smanjenjem procesora linearno smanjivala i učinkovitost. Učinkovitost služi za međusobnu usporedbu postupaka raspoređivanja, gdje ona s većom vrijednosti čimbenika učinkovitosti ima bolje performanse.

Relativna razlika duljine rasporeda

Posljednji pokazatelj kvalitete u vremenskoj analizi je relativna razlika duljine rasporeda. Kao što mu i samo ime kaže, on pokazuje koliko se u postocima razlikuje duljina rasporeda dobivenog određenim postupkom od dobivenih rasporeda drugim postupcima, za isti testni slučaj. Navedeni pokazatelj kvalitete uveli smo zbog toga što često korišteni pokazatelj kvalitete "broj pojavljivanja boljih rješenja" (eng. *number of occurrences of better solutions*), korišten u [13, 39, 41], ne daje na prvi pogled jasnu sliku razlike u performansama među postupcima raspoređivanja. Naime, grafički prikaz relativne razlike duljine rasporeda mnogo je intuitivniji i odmah pokazuje kako određeni postupak stoji u odnosu s uspoređenim postupcima.

Navedeni pokazatelj definiran je kao:

$$rel.\ razlika\ duljine\ rasporeda = \frac{\sum_{m_i \in M} \frac{t^{rasp}_{m_k} - t^{rasp}_{m_i} \cdot 100}{t^{rasp}_{m_k}}}{|M|}, \tag{5.9}$$

gdje je m_i postupak raspoređivanja iz skupa svih testiranih postupaka raspoređivanja M, a m_k je postupak za koji se računa relativna duljina rasporeda.

Ukoliko je vrijednost relativne duljine rasporeda postupka m_k manja od istog kod ostalih postupaka, to označava da su performanse postupka m_k bolje od njih. Smanjivanjem vrijednosti relativne duljine rasporeda nekog postupka (u negativno područje), veće su performanse tog postupka raspoređivanja u odnosu na ostale.

5.3.2 Pokazatelji kvalitete analize utroška energije

Analiza utroška energija zasniva se na dva parametra. Prvi je ukupni utrošak energije ε^{uk} koji se računa prema (4.71) i (4.72) kao zbroj utroška energije svih obradbenih jedinica sklopovlja:

$$\varepsilon^{uk} = \sum_{p_j \in P} \varepsilon^{uk}_j \tag{5.10}$$

Postupak je bolji što je utrošak energije manji, no takav prikaz, kao i duljina rasporeda kao pokazatelja kvalitete u vremenskoj analizi, ovisan je o parametrima programa (ϑ, x^{ob}_G).

Drugi je parametar ušteda energija ili postotna razlika utroška energije ε^{raz}_{uk} koja je opisana pomoću (4.79).

Okarakterizirane mjerene veličine pokazatelja kvalitete koristit ćemo kao parametre usporedbe postupaka raspoređivanja u navedenim pododjeljcima. Njihove vrijednosti upućivat će na to koji je postupak bolji od ostalih te u kojoj mjeri postupci daju željene performanse (odzive).

5.4 Rezultati vremenske analize postupka RaNDKiP

Eksperimentalna analiza svakog postupka, pa tako i postupka RaNDKiP, tematski je podijeljena na dva dijela: vremenska analiza i analiza potrošnje energije. Vremenska analiza sadržava rezultate mjernih veličina: SLR, učinkovitost, ubrzanje i relativna razlika duljine rasporeda u odnosu na druge postupke pri homogenim i raznorodnim komunikacijskim vezama među obradbenim jedinicama simulirane računalne platforme. Postupak RaNDKiP bit će uspoređen s izvedbama istog, RaNDKiP1 i RaNDKiP2 te s postupcima HEFT i LDCP (opisanim u pododjeljcima 2.3.5 i 2.3.6) kao predvodnicima postupaka zasnovanih na rangiranju jedinica programa, jer su na tom principu zasnovane i inačice postupaka RaNDKiP.

U prvom dijelu bit će prikazani rezultati vremenske analize sintetičkih programa generiranih standardiziranim nasumičnim postupcima (DAGGer, TGFF i radna opterećenja Pegasus) i to za homogene i raznorodne strukture komunikacijskih veza među obradbenim jedinicama sklopovlja. Za svaku kombinaciju slučaja bit će dani eksperimentalni rezultati, kao i njihova analiza u ovisnosti o glavnim ulaznim

parametrima: ϑ, ρ, η i ccr. Za programe iz repozitorija Pegasus, ali i one generirane alatom TGFF, bit će dodan još jedan ulazni parametar, a to je **struktura programa**.

Prvi je slučaj analiza programa generiranih vlastitim alatom — DAGGer-om, u slučaju kada se izvode na platformi s homogenim vezama među obradbenim jedinicama. U idućem pododjeljku opisujemo takav slučaj.

5.4.1 Slučaj 1: raspoređivanje programa generiranih alatom DAGGer na platforme s homogenim komunikacijskim vezama

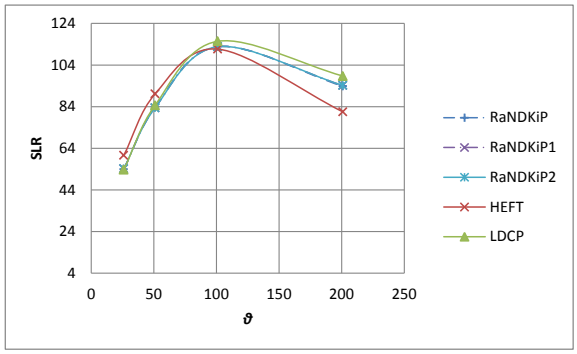
Prvi slučaj opisuje raspoređivanje programa generiranih alatom DAGGer na računalne platforme s homogenim komunikacijskim vezama. U ovom pododjeljku dani su rezultati eksperimentalne analize vremenskih parametara u ovisnosti o broju izvršnih jedinica ϑ, slika 5.13. Za svaku mjerenu veličinu izračunati su prosjeci svih testnih slučajeva za određeni broj izvršnih jedinica programa (25, 50, 100, 200) i oni su prikazani u grafovima. Komunikacija među obradbenim jedinicama je homogena, (propusnosti su jednake između bilo kojeg para obradbenih jedinica) i postavljene su na 1. Slika 5.13a prikazuje normaliziranu duljinu rasporeda za različit broj izvršnih jedinica programa. Vidljivo je kako postupak HEFT poprima veće vrijednosti od svih ostalih postupaka, čineći ga nepovoljnim za veličine do 100 izvršnih jedinica po programu. Za vrijednosti ϑ veće od 100 HEFT pokazuje bolje performanse i za pretpostaviti je da bi razlika bila i veća za veći broj izvršnih jedinica. Ostali postupci daju podjednake rezultate, osim LDCP-a, koji pri većem broju izvršnih jedinica daje veći SLR. Postupak RaNDKiP u ovom je slučaju u nepovoljnom položaju s obzirom na to da raznorodnost komunikacijskih veza sklopovlja ovdje ne igra ulogu, na čemu on temelji svoj potencijal. Sve su vrijednosti SLR-a relativno visoke, jer komunikacija igra veliku ulogu u duljini rasporeda (nema njezinog skaliranja u odnosu na komunikacijski kanal), posebice s velikim vrijednostima CCR-a, što će biti vidljivo kasnije.

Učinkovitost je prikazana na slici 5.13c i ponaša se slično kao i SLR. Razlika je u tome što HEFT pokazuje bolje vrijednosti u odnosu na ostale postupke već pri 50 izvršnih jedinica po programu. LDCP opet daje nešto slabije rezultate od inačica postupka RaNDKiP, koje su u odnosu na HEFT bolje samo za vrlo male programe. Kod svih postupaka primjetan je pad učinkovitosti poslije 100 izvršnih jedinica po programu, kao i za SLR, koji je uzrokovan malim brojem izvršnih jedinica i ograničenoj sposobnosti paralelizacije.

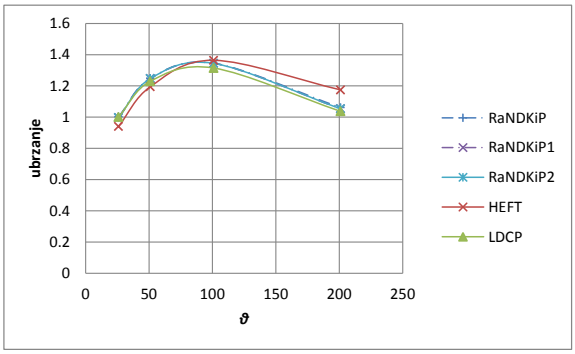
Navedeno je vidljivo i za ubrzanje, slika 5.13b, gdje su vrijednosti kod svih postupaka vrlo slične. Također, postupak HEFT daje nešto bolje rezultate za broj izvršnih jedinica po programu veći od 100, slijedi RaNDKiP, a zatim LDCP.

Relativna razlika u duljini rasporeda prikazana na slici 5.13d očekivano je u korist postupka HEFT. Međutim, vrijednosti nisu velike i kreću se do -25% (25% kraći raspored u prosjeku u odnosu na druge postupke) za HEFT, dok je prijelomna točka otprilike 75 izvršnih jedinica (prijelaz sa lošijeg od na bolje od). Ovdje je zanimljivo da izvorni postupak RaNDKiP daje bolje rezultate od njegovih izvedbi RaNDKiP1 i RaNDKiP2, što ukazuje na to da dinamičko rangiranje igra pozitivnu ulogu u tom slučaju.

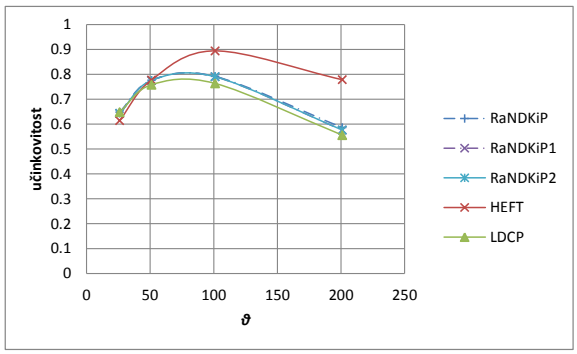
Na slici 5.14 prikazane su vrijednosti mjernih veličina u odnosu na broj obradbenih jedinica ρ.



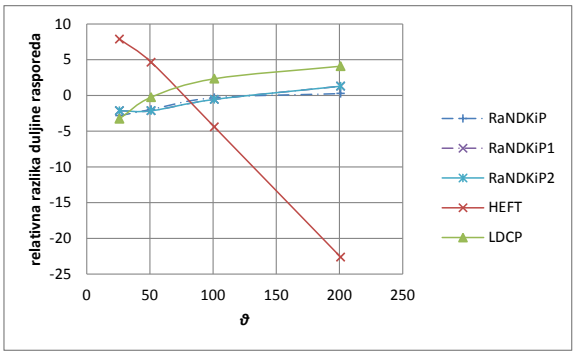
(a) SLR



(b) Ubrzanje

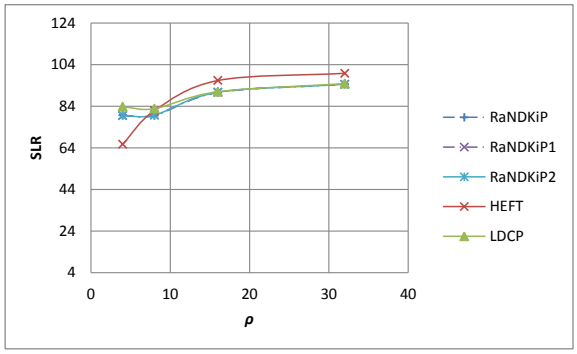


(c) Učinkovitost

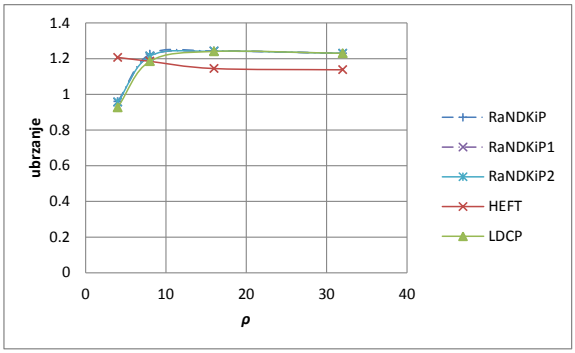


(d) Relativna razlika duljine rasporeda

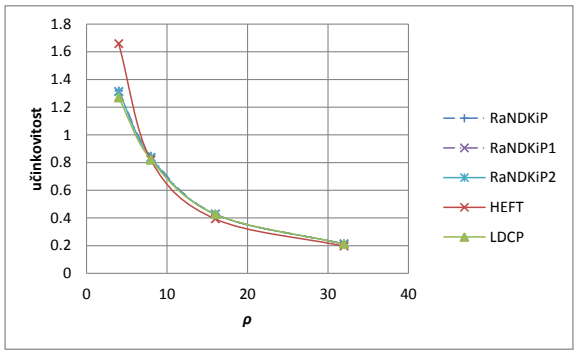
Slika 5.13: Testni slučajevi generirani alatom DAGGer za različit ϑ, homogena komunikacija.



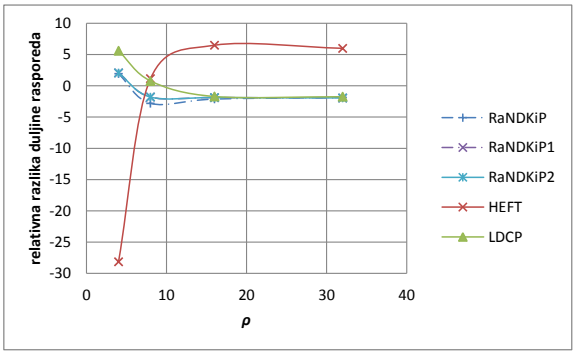
(a) SLR



(b) Ubrzanje



(c) Učinkovitost



(d) Relativna razlika duljine rasporeda

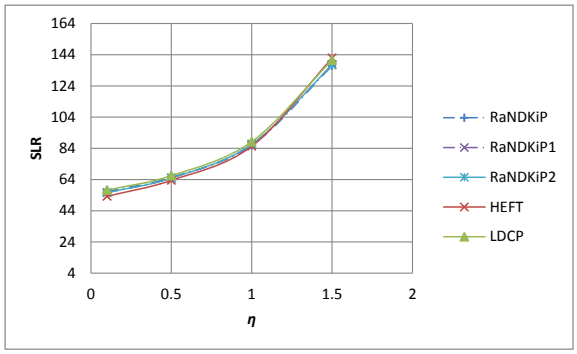
Slika 5.14: Testni slučajevi generirani alatom DAGGer za različit ρ, homogena komunikacija.

Slika 5.14a prikazuje SLR koji se ne mijenja značajno u ovisnosti o broju obradbenih jedinica sklopovlja. Također, HEFT pokazuje malo lošije rezultate u odnosu na ostale postupke i to na broju obradbenih jedinica većim od 8. Ostali postupci daju podjednake vrijednosti SLR-a. Ubrzanje ostaje gotovo konstantno s porastom broja izvršnih jedinica, osim pri vrlo malim programima, (broj izvršnih jedinica po programu je 25). Razlog tome što je ta mjerna veličina relativno neosjetljiva na broj obradbenih jedinica sklopovlja u slučajevima gdje postoji velik udio komunikacije u rasporedu jer, prema (5.6) iz pododjeljka 5.3.1, ona je odnos vremena izvršavanja svih jedinica programa na jednom procesoru i ukupnog trajanja rasporeda. Porastom broja obradbenih jedinica samo raste vjerojatnost da će jedna od njih postići minimume u nasumičnom generiranju parametara. Samim time učinkovitost pada, slika 5.14c, a njezine vrijednosti između pojedinih postupaka ostaju vrlo slične. Primjećuje se da HEFT ima nešto veću učinkovitost za manji broj obradbenih jedinica, dok se s većim vrijednostima izjednačava s ostalim postupcima. Relativna razlika u duljini rasporeda je u korist postupaka koji koriste dinamičke kritične putove (svi osim HEFT-a), no ta se razlika stabilizira za platforme brojem obradbenih jedinica većim od osam.

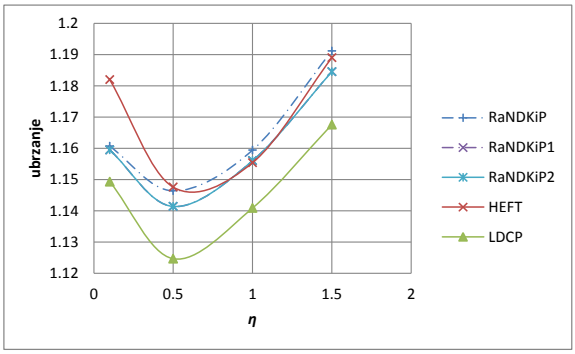
Utjecaj čimbenika raznorodnosti na dobivene rezultate vidljiv je na slici 5.15. Vrijednosti SLR-a su gotovo identične kod različitih postupaka, dok njegove vrijednosti drastično rastu porastom raznorodnosti, slika 5.15a. Ubrzanje, s druge strane, jest najmanje za iznos čimbenika raznorodnosti od 0,5 kod svih postupaka, nakon toga raste s njegovim povećanjem, slika 5.15b. Također je vidljivo da postupak RaNDKiP daje najveće vrijednosti, unatoč tome što je mreža među obradbenim jedinicama homogena. Vrijednosti ubrzanja za HEFT vrlo su bliske onima postupka RaNDKiP, dok su za LDCP nešto niže. Važno je napomenuti da su razlike vrlo male, a isti je slučaj i za učinkovitost, slika 5.15c, kao i za relativne razlike u trajanju rasporeda ($\approx$1 – 2%).

Kao što je prikazano na slici 5.15, rezultati su vrlo bliski među postupcima jer čimbenik raznorodnosti nema velik utjecaj u sustavima s homogenom mrežom računalne platforme i dominantnom komunikacijom.

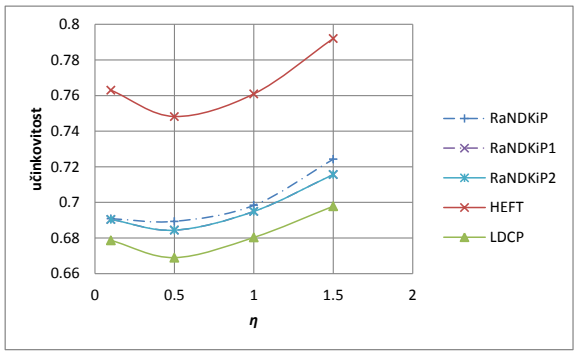
Vrijednosti CCR-a gotovo da ni ne utječu na razliku među postupcima u slučaju homogene mreže, slika 5.16, značajne su samo tendencije promjene pojedinih mjernih veličina u odnosu na CCR. Na slici 5.16a, vidi se da SLR raste linearno s porastom CCR-a, što je i za očekivati, jer je utjecaj komunikacije među izvršnim jedinicama dominantan i konstantan. S druge pak strane, vrijednosti ubrzanja drastično padaju porastom raznorodnosti (slika 5.16b), jer se razlike u vremenu izvršavanja jedinica programa velike, tako da će vremena na nekima od njih dominirati nad cijelim rasporedom. Navedeno uzrokuje i drastičan pad učinkovitosti, slika 5.16c. Relativne razlike su zanemarive, jer zbog manjka raznorodnosti svi postupci daju vrlo slične rezultate, slika 5.16d.



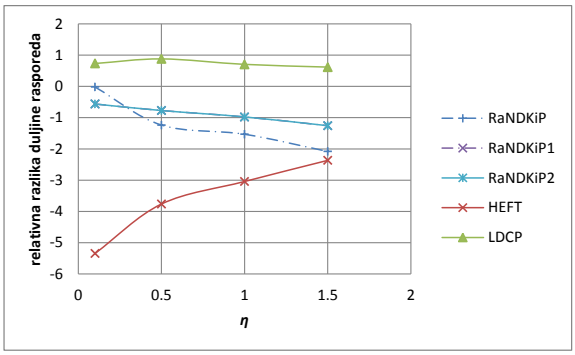
(a) SLR



(b) Ubrzanje

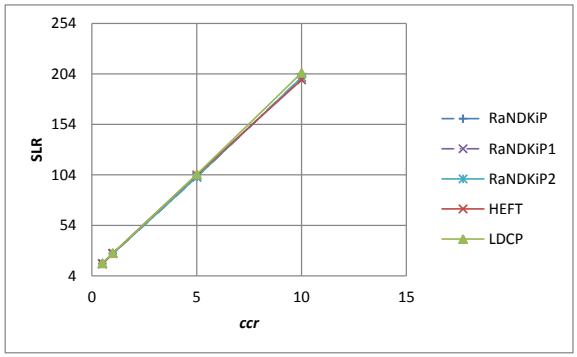


(c) Učinkovitost

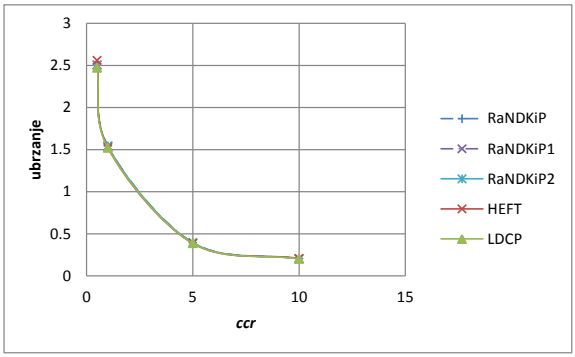


(d) Relativna razlika duljine rasporeda

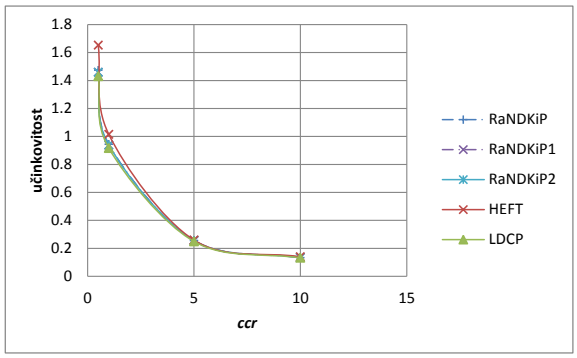
Slika 5.15: Testni slučajevi generirani alatom DAGGer za različit η, homogena komunikacija.



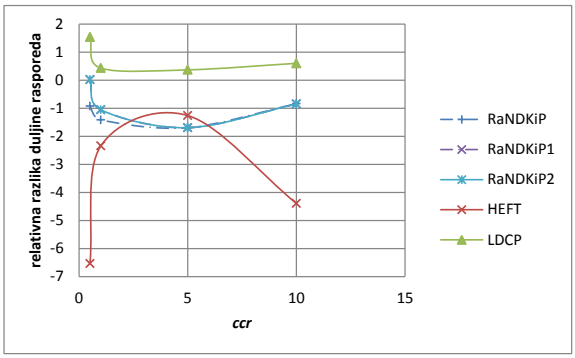
(a) SLR



(b) Ubrzanje



(c) Učinkovitost



(d) Relativna razlika duljine rasporeda

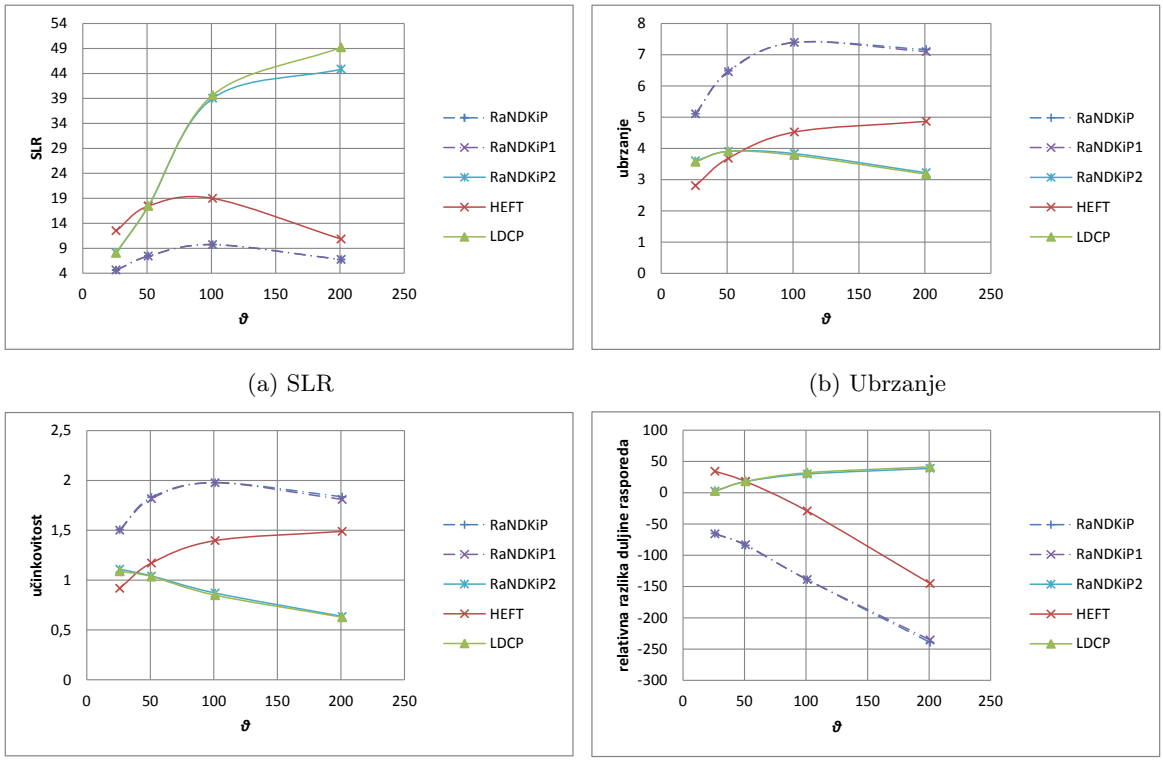
Slika 5.16: Testni slučajevi generirani alatom DAGGer za različit *ccr*, homogena komunikacija.

Dobiveni rezultati ukazuju na to da za homogenu mrežu između obradbenih jedinica sklopovlja postupak RaNDKiP daje bolje rezultate kod manjih programa (manje od 100 izvršnih jedinica po programu), gdje u svim mjernim veličinama nadmašuje ostale postupke. Pri homogenoj komunikaciji velik udio u rasporedu ima i komunikacija, jer se njezin iznos generira na isti način kao i vremena izvršavanja jedinica programa. Upravo zbog toga navedeno je posebno izraženo pri većim vrijednostima CCR-a. Sam postupak HEFT predstavlja se kao bolji odabir za programe s većim brojem izvršnih jedinica zbog toga što vrednuje prosjek komunikacije, a to je vrlo značajno kod velikog broja komunikacijskih poziva (kada komunikacija dominira). U sljedećem slučaju bit će razmatrane iste postavke eksperimenta, ali uz raznorodnu komunikaciju među obradbenim jedinicama sklopovlja. Taj slučaj opisujemo u narednom pododjeljku.

5.4.2 Slučaj 2: raspoređivanje programa generiranih alatom DAGGer na platforme s raznorodnim komunikacijskim vezama

Ovim pododjeljkom predstavljamo drugi slučaj, a to je raspoređivanje programa generiranih alatom DAGGer na računalne platforme s raznorodnim komunikacijskim vezama. Naime, slučajevi s takvim oblikom komunikacije su glavni problem na koji cilja unaprijeđeni postupak RaNDKiP u okviru našeg doprinosa D.3.

Prva situacija su također rezultati eksperimentalne analize vremenskih parametara u ovisnosti o broju izvršnih jedinica ϑ koje čine program. Za svaku mjerenu veličinu izračunati su prosjeci svih testnih slučajeva za određeni broj izvršnih jedinica programa (25, 50, 100, 200) i oni su prikazani u grafovima. Komunikacija među obradbenim jedinicama je raznorodna (propusnosti $c_{i,j}$ variraju između 1 i 100) između bilo kojeg para obradbenih jedinica. Ovisnost mjernih veličina o broju izvršnih jedinica programa prikazana je slikom 5.17. Slika 5.17a prikazuje normaliziranu duljinu rasporeda, gdje je vidljivo kako postupci LDCP i RaNDKiP2 daju najveće vrijednosti i drastično se povećavaju rastom broja izvršnih jedinica programa. Razlog tome je potpuno zanemarivanje komunikacije među obradbenim jedinicama. Također, drastičan porast SLR-a za ta dva postupka ukazuje na značaj komunikacije, jer porastom broja izvršnih jedinica raste i broj komunikacijskih poziva među njima, koje u nepovoljnom rasporedu mogu uzrokovati velika odstupanja. HEFT je u mnogo povoljnijem položaju u odnosu na LDCP i RaNDKiP, jer podrazumijeva komunikaciju među obradbenim jedinicama i pri rangiranju i pri ubacivanju vremenskih isječaka u raspored. Međutim, s obzirom na to da postupak HEFT koristi prosječne vrijednosti komunikacije, ne može u potpunosti preslikati pravu situaciju. Na tom polju je najbolji postupak RaNDKiP, koji pokazuje iznimno dobre rezultate primjenom na sustave s raznorodnom komunikacijom među obradbenim jedinicama sklopovlja. Vrijednosti SLR-a koje on postiže višestruko su veće od onih koje postiže HEFT, a posebice LDCP. Njemu identične rezultate postiže i njegova inačica RaNDKiP1 koja koristi raznorodnost u fazi rangiranja izvršnih jedinica. Vrijednosti ubrzanja, slika 5.17b, također idu u veliku korist postupka RaNDKiP i njegove inačice RaNDKiP1. Ostali postupci postižu gotovo dvostruko slabije rezultate, osim HEFT-a koji je između. Slično prikazuje i slika 5.17c, gdje opet dominira postupak RaNDKiP, no u nešto manjoj mjeri nego pri ubrzanju.



(a) SLR

(b) Ubrzanje

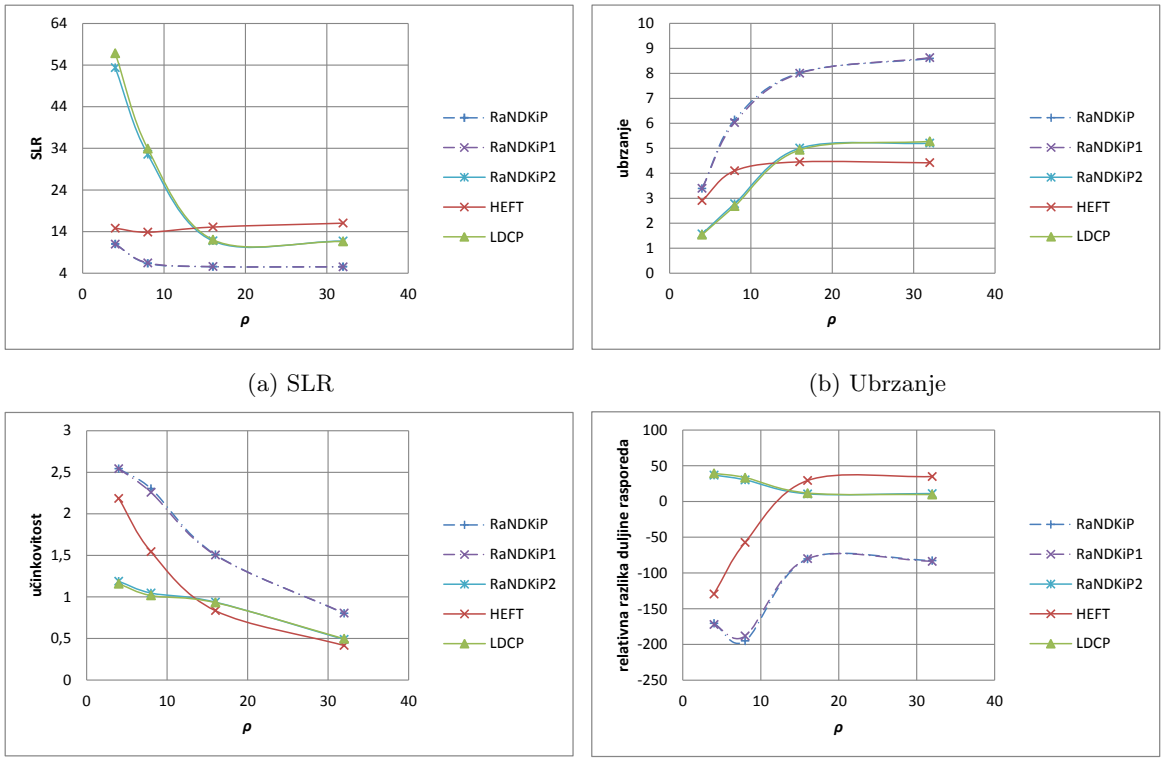
(c) Učinkovitost

(d) Relativna razlika duljine rasporeda

Slika 5.17: Testni slučajevi generirani alatom DAGGer za različit ϑ, raznorodna komunikacija.

Relativna razlika duljine rasporeda u velikoj mjeri ocrtava koliko su postupci RaNDKiP i RaNDKiP1 bolji od ostalih postupaka, gdje postižu prosječnu razliku u odnosu na njih od gotovo 250%, slika 5.17d. Ako bi se gledala razlika u odnosu na pojedinačne postupke, (LDCP, RaNDKiP2), razlika bi bila i veća. Zajednička je karakteristika da povećanjem broja izvršnih jedinica programa u slučaju raznorodne mreže među obradbenim jedinicama dolazi do lagane stagnacije većine mjernih veličina (osim relativne razlike), što dovodi do zaključka da je skalabilnost svih postupaka u ovom slučaju vrlo dobra.

Kada se promotri situacija za različit broj obradbenih jedinica sklopovlja (slika 5.18), također je vidljiva dominantnost postupaka RaNDKiP i RaNDKiP1 za sve veličine raznorodne platforme. SLR se smanjuje porastom broja obradbenih jedinica do njih 16, kada vrijednosti počinju stagnirati, slika 5.18a. Zanimljivo je da LDCP, kao i postupak RaNDKiP2 nadilaze HEFT već kod 16 obradbenih jedinica, čemu je uzrok što oba postupka rade odvojeno rangiranje za svaki procesor. Postupak RaNDKiP postiže do tri puta veće vrijednosti SLR-a nego postupak HEFT i do 2,5 puta veće od postupaka LDCP i RaNDKiP2. Kod ubrzanja je situacija slična, uz nešto manje razlike, gdje postupak RaNDKiP ostvaruje ubrzanje od 8,5 (slika 5.18b). Kod učinkovitosti je situacija nešto drugačija, slika 5.18c, no također postupak RaNDKiP ostvaruje dvostruko bolju učinkovitost od najboljeg sljedećeg postupka. HEFT je u ovom slučaju nešto bliži postupcima LDCP i RaNDKiP2. Kod relativne duljine rasporeda, slika 5.18d, razlike su nestabilne za manji broj obradbenih jedinica, no ubrzo se stabiliziraju za broj obradbenih jedinica veći od 16 i nakon toga stagniraju.



(a) SLR

(b) Ubrzanje

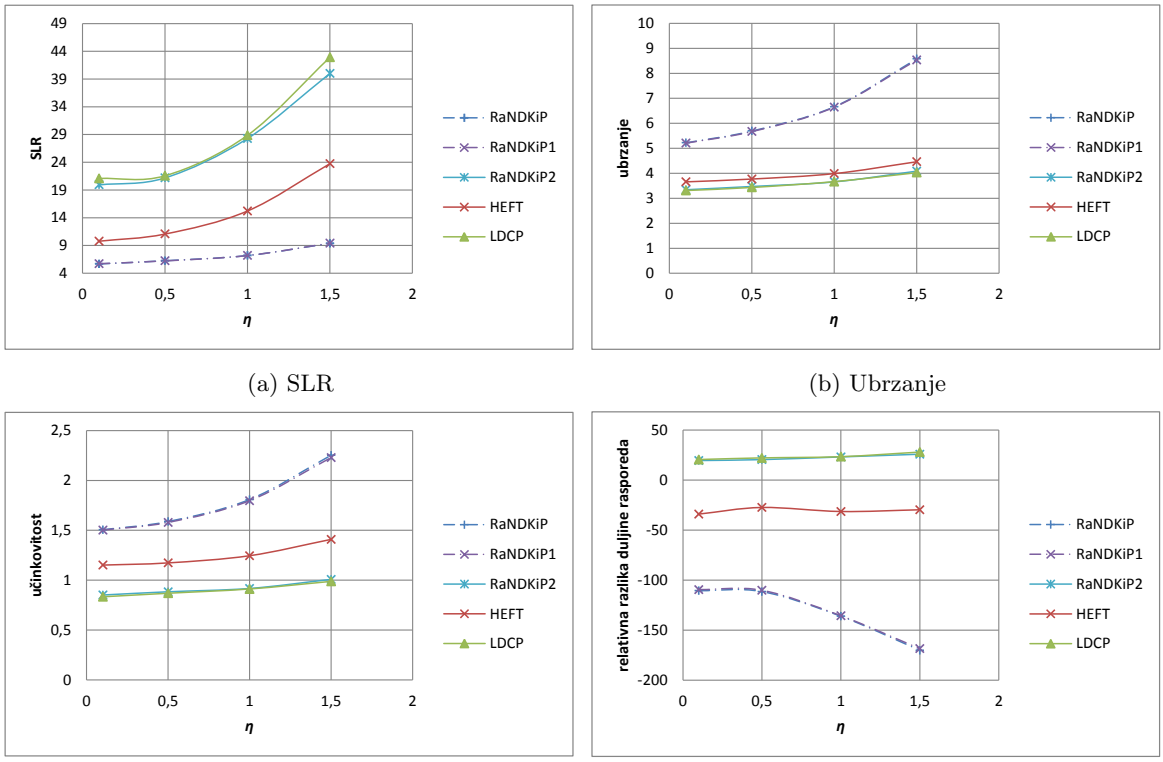
(c) Učinkovitost

(d) Relativna razlika duljine rasporeda

Slika 5.18: Testni slučajevi generirani alatom DAGGer za različit ρ, raznorodna komunikacija.

U svim mjernim veličinama i ovdje RaNDKiP donosi bolje rezultate, posebice u SLR-u, za koji se i smatra da je gotovo najbolji pokazatelj primjenjivosti postupaka raspoređivanja. Nadalje, sve vrijednosti se razlikuju kod malog broja obradbenih jedinica, no već od 16 postaju stabilne i ne ovise dalje o broju obradbenih jedinica.

Promjenom vrijednosti čimbenika raznorodnosti dobiva se situacija prikazana slikom 5.19. Najznačajnija je promjena u SLR-u koji raste povećanjem vrijednosti čimbenika raznorodnosti, slika 5.19a. Razlika je što su postupci RaNDKiP i RaNDKiP1 relativno neosjetljivi na promjenu čimbenika raznorodnosti, što je i za očekivati u odnosu na njihovu namjenu. Vidi se da oni kompenziraju porast raznorodnosti podrazumijevajući ga i u rangiranju i u ažuriranju stanja pri svakom koraku raspoređivanja. Pri rangiranju to nije toliko važno, ali pri ažuriranju jest, što se i vidi iz rezultata koje ostvaruju izvedbe RaNDKiP1 i RaNDKiP2 (prvi daje rezultate kao i RaNDKiP, dok drugi daje gotovo najslabije rezultate). HEFT je u sredini, dok je LDCP uvjerljivo najlošiji zbog njegovog zanemarivanja komunikacije. Slične su situacije za ubrzanje i učinkovitost, slike 5.19b i 5.19c. U objema postupci RaNDKiP i RaNDKiP1 pokazuju rezultate dvostruko bolje od sljedećeg najboljeg postupka (HEFT), gdje se ta razlika dalje povećava porastom raznorodnosti. Navedeno se uočava i uvidom u sliku 5.19d, gdje razlike u korist postupaka RaNDKiP i RaNDKiP1 dosežu gotovo 200% i rastu s porastom raznorodnosti.



(a) SLR

(b) Ubrzanje

(c) Učinkovitost

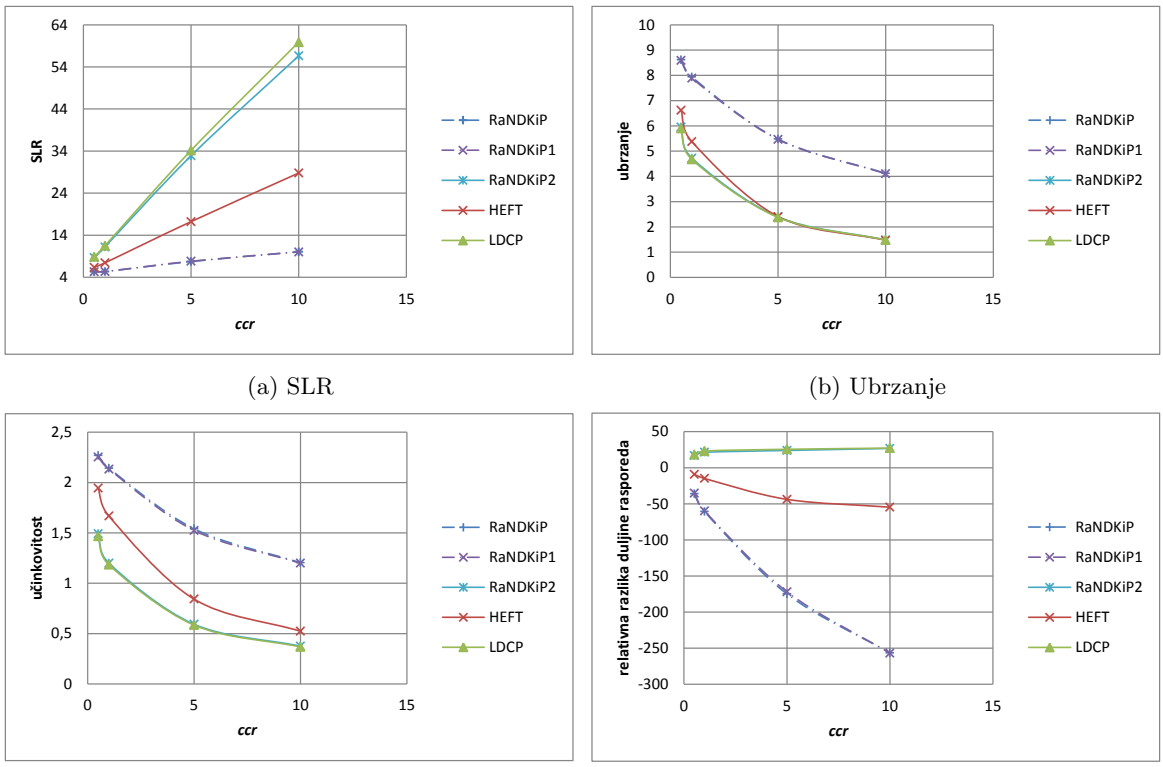
(d) Relativna razlika duljine rasporeda

Slika 5.19: Testni slučajevi generirani alatom DAGGer za različit η, raznorodna komunikacija.

Zajednička analiza pokazuje da porast vrijednosti čimbenika raznorodnosti povećava primjenjivost postupaka RaNDKiP i RaNDKiP1 i to u najvećoj mjeri zbog njihove sposobnosti uzimanja u obzir raznorodnosti pri ažuriranju rangova nakon raspoređivanja svake od izvršnih jedinica. Utjecaja raznorodnog rangiranja na rezultate nema, što je vidljivo i iz rezultata koje postižu RaNDKiP2 i LDCP.

Postupak RaNDKiP odlično se nosi i sa porastom udjela komunikacije (porastom omjera komunikacije i obrade, CCR-a), slika 5.20. Kod SLR-a, razlika je najveća i vidi se da je zapravo CCR parametar koji najviše utječe na postignuća pojedinih postupaka u slučaju raznorodne mreže među obradbenim jedinicama sklopovlja. Razlika od gotovo šest puta u odnosu na LDCP i tri puta u odnosu na HEFT ukazuje na to koliki značaj može imati pravilna interpretacija komunikacije pri raspoređivanju. Razlika je također i više nego primjetna kod ubrzanja, slika 5.20b i učinkovitosti, slika 5.20c, gdje unatoč tome što svim postupcima padaju performanse porastom CCR-a (do određene granice), postupci RaNDKiP s dinamičkom fazom ažuriranja (RaNDKiP i RaNDKiP1), donose višestruko bolju primjenjivost. To se vidi i iz slike 5.20d, gdje se za CCR od 10 dobivaju prosječne razlike od čak 250%. Također, vidljiva je i tendencija rasta, tako da je za očekivati kako bi za veći CCR razlike bile još značajnije.

Zbirnom analizom slučaja 2 možemo zaključiti kako RaNDKiP i njegova izvedba RaNDKiP1 pokazuju iznimno dobre rezultate u slučaju s raznorodnom komunikacijom sklopovlja, unatoč relativno nasumičnoj strukturi programa kakve stvara alat DAGGer. To je vidljivo po svim vremenskim pokazateljima kvalitete i za sve kombinacije ulaznih parametara. Postupak HEFT daje mnogo slabije rezultate, no ipak se zadržava negdje na polovici između postupaka RaNDKiP i RaNDKiP1 s jedne strane te postupaka RaNDKiP2 i LDCP s druge. Navedeni rezultati ukazuju da je najvažnija razlika pravilno



(a) SLR

(b) Ubrzanje

(c) Učinkovitost

(d) Relativna razlika duljine rasporeda

Slika 5.20: Testni slučajevi generirani alatom DAGGer za različit *ccr*, raznorodna komunikacija.

tretiranje raznorodnih komunikacijskih veza pri fazi ažuriranja, dok raznorodno rangiranje ne igra ulogu u performansama.

U sljedećem slučaju razmatramo programe iz repozitorija Pegasus, kao sve popularnije strukture programa za testiranje postupaka raspoređivanja.

5.4.3 Slučaj 3: raspoređivanje programa iz repozitorija Pegasus na platforme s raznorodnim komunikacijskim vezama

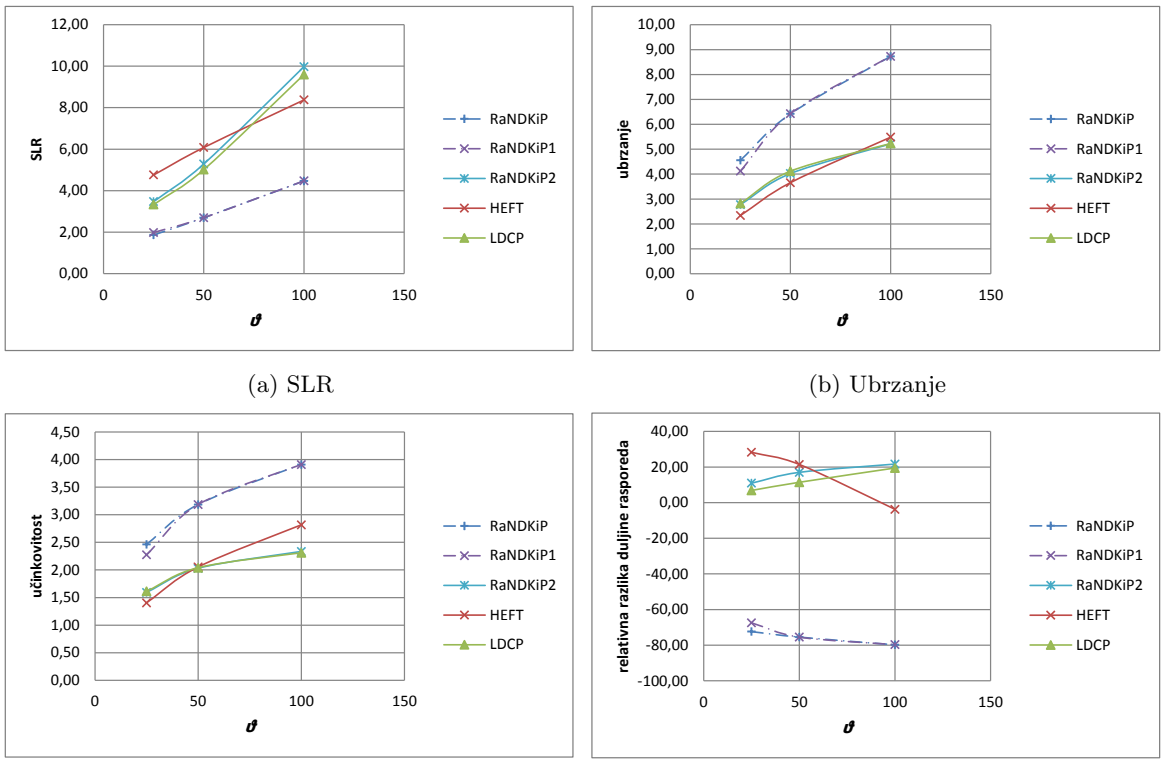
Zbog toga što su glavni ciljevi našeg rada poboljšavanje podrške za raznorodnu komunikaciju pri raspoređivanju programa na sklopovlje, ovdje nećemo dati prikaz situacije za homogenu komunikaciju. Naime, situaciju s homogenim vezama opisali smo slučajem 1 te je za očekivati da bi odnosi postupaka raspoređivanja bili vrlo slični testnim slučajevima Pegasus.

Zbog toga u ovom pododjeljku analiziramo treći slučaj, što je raspoređivanje programa iz repozitorija Pegasus na raznorodne računalne sustave s raznorodnim komunikacijskim vezama među obradbenim jedinicama.

Dobivene vrijednosti mjerenih veličina kod primjene postupaka raspoređivanja na radna opterećenja Pegasus s promjenjivim ϑ prikazani su na slici 5.21. Vrijednosti su prosječne za sve testne slučajeve koji odgovaraju kriteriju, s tim da kod opterećenja Pegasus razmjeri opterećenja određuju brojeve izvršnih jedinica programa (mali, srednji i veliki), prema pododjeljcima 5.1.2 i 5.2. Sve vrijednosti ponašaju se slično kao i kod testnih slučajeva generiranih alatom DAGGer. SLR poprima dosta manje vrijednosti,

slika 5.21a, zbog u prosjeku manjeg broja bridova grafa — manje komunikacije. Slično, time su i vrijednosti ubrzanja nešto veće u odnosu na slučajeve DAGGer, slika 5.21b, gdje RaNDKiP postiže približno 8,5 dok kod DAGGer generiranja postiže ubrzanje od 7. Učinkovitost slijedi trend ubrzanja, kao što se vidi na slici 5.21c, dok razlike nemaju toliki trend porasta kod postupka RaNDKiP, (slika 5.21d), a i nešto su manje, (80% u odnosu na gotovo 150% kod slučajeva DAGGer za postupak RaNDKiP).

Ovisnost mjerenih veličina o broju obradbenih jedinica sklopovlja za radna opterećenja Pegasus i raznorodnu komunikaciju sklopovlja prikazana je na slici 5.22. Također, kao i kod testnih slučajeva DAGGer, vrijednosti SLR-a padaju porastom broja obradbenih jedinica, no s manjim početnim vrijednostima (slika 5.22a). Pokazatelj je drastično boljih performansi postupaka nego kod testnih slučajeva DAGGer zbog manje raznorodnosti strukture programa. Odnosi među performansama su slični između pojedinih postupaka, iako LDCP pokazuje neznatan pomak u odnosu na RaNDKiP2. To toliko nije vidljivo na prikazu ubrzanja i učinkovitosti, slike 5.22b i 5.22c. Nasuprot toga, relativna razlika duljine rasporeda opet pokazuje nešto bolje performanse postupka LDCP u odnosu na postupak RaNDKiP2 i ovdje su razlike za manji broj obradbenih jedinica manje izražene nego kod testnih slučajeva DAGGer.



(a) SLR

(b) Ubrzanje

(c) Učinkovitost

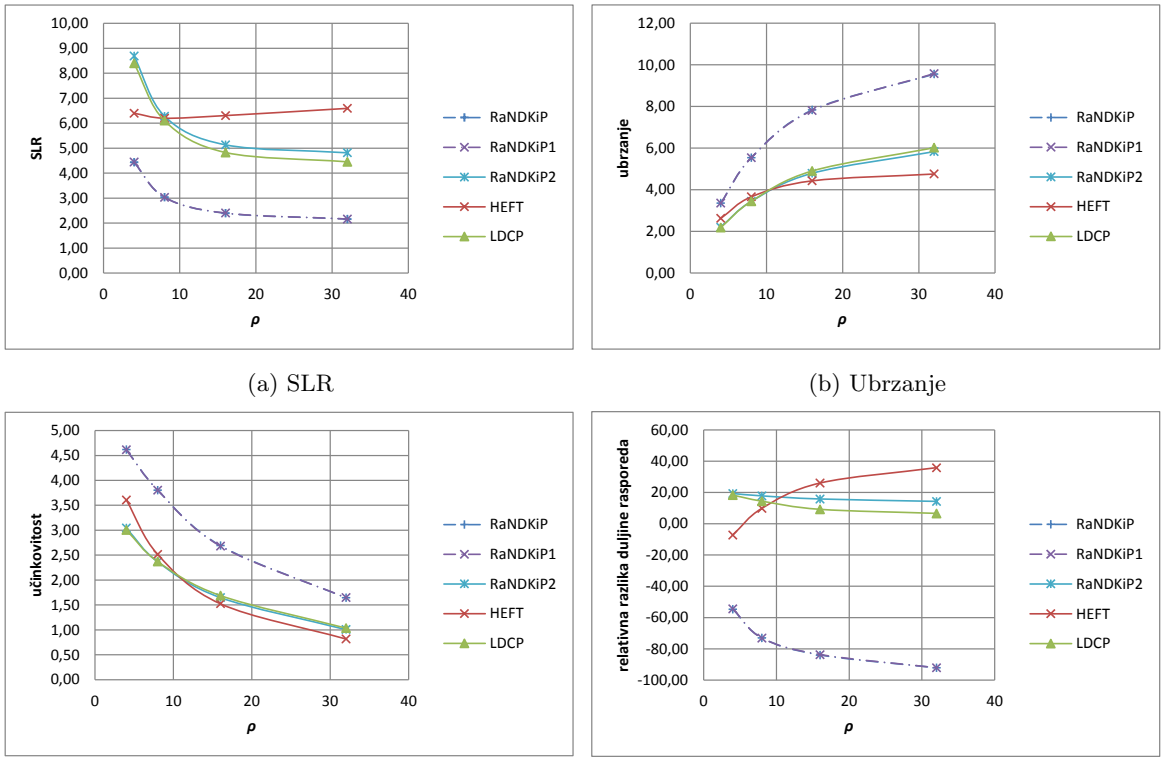
(d) Relativna razlika duljine rasporeda

Slika 5.21: Testni slučajevi iz repozitorija Pegasus za različit ϑ, raznorodna komunikacija.

Također, postupak RaNDKiPopet pokazuje odlične performanse u odnosu na ostale postupke, posebice povećanjem broja obradbenih jedinica (veći utjecaj raznorodnosti komunikacije kod njihovog većeg broja).

Za različite vrijednosti čimbenika raznorodnosti η, slika 5.23, svi postupci osim RaNDKiP-a i njegove inačice RaNDKiP1 imaju slične performanse. Performanse u smislu SLR-a smanjuju se porastom vrijed-

nosti čimbenika raznorodnosti, slika 5.23a, dok se ubrzanje i učinkovitost povećavaju za sve postupke, slike 5.23b i 5.23c, posebice za postupak RaNDKiP i njegovu inačicu RaNDKiP1. Iz navedenoga se vidi kako veća raznorodnost pogoduje tim postupcima (čime opravdavaju svoju namjenu) i kako porastom raznorodnosti rastu i njihove performanse. Kod njihove upotrebe porast SLR-a je slabije izražen, a drže i velik odmak u performansama od ostalih postupaka (više od 100%). Relativna razlika duljine rasporeda im također ide u korist, posebice povećanjem raznorodnosti, što se vidi na slici 5.23d.



(a) SLR

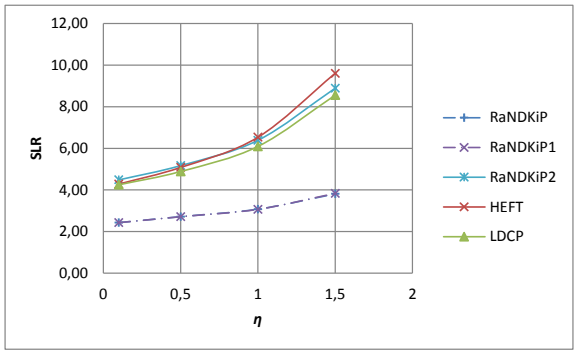
(b) Ubrzanje

(c) Učinkovitost

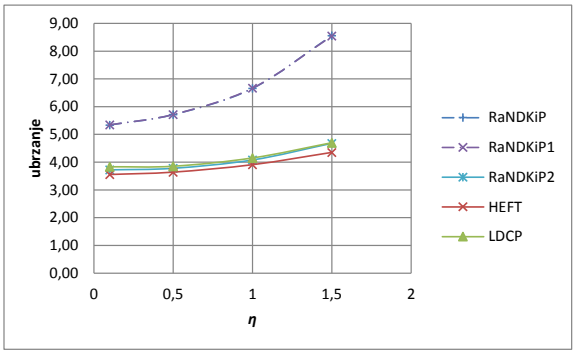
(d) Relativna razlika duljine rasporeda

Slika 5.22: Testni slučajevi iz repozitorija Pegasus za različit ρ, raznorodna komunikacija.

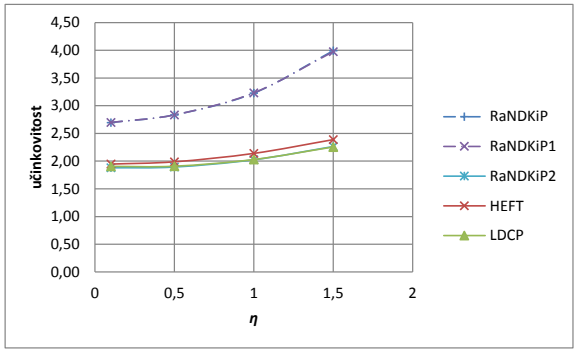
Povećanjem CCR-a smanjuju se performanse svih postupaka, prema slici 5.24, kao i kod testnih slučajeva generiranih alatom DAGGer. Za istaknuti je kako povećanje CCR-a najviše škodi postupku HEFT, što se posebno vidi na SLR-u i relativnoj razlici duljine rasporeda na slikama 5.24a i 5.24d. CCR je parametar koji i ovdje pokazuje odlične performanse inačica unaprijeđenog postupka RaNDKiP, gdje one pokazuju performanse i do 3 puta bolje od ostalih postupaka, što je osim na SLR-u vidljivo i na ubrzanju, slika 5.24b, a samim time i na učinkovitosti, slika 5.24c. Što se tiče duljine rasporeda, slika 5.24d, prednost opet imaju postupci koji koriste dinamičko rangiranje. Posebno je to slučaj kod postupka RaNDKiP, kojem je i ovdje zajamčena velika prednost za porast CCR-a.



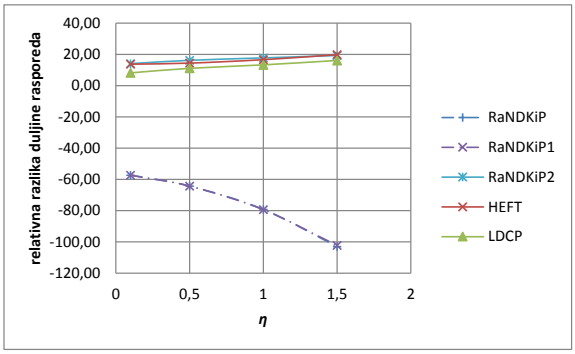
(a) SLR



(b) Ubrzanje

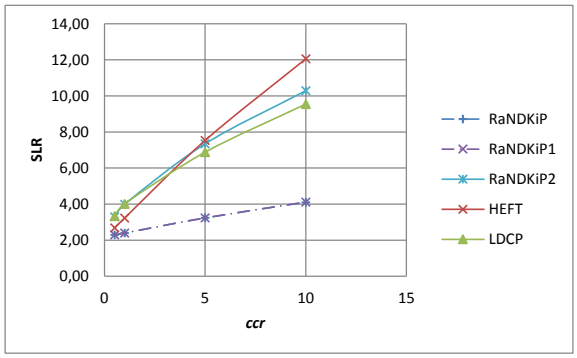


(c) Učinkovitost

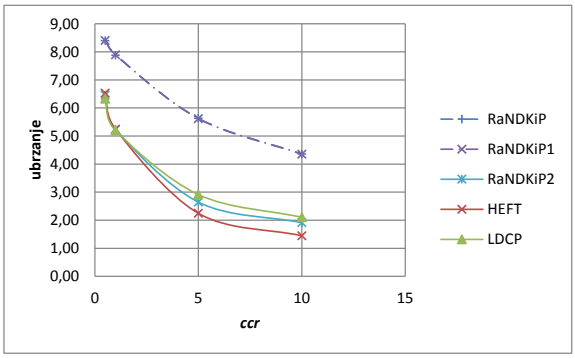


(d) Relativna razlika duljine rasporeda

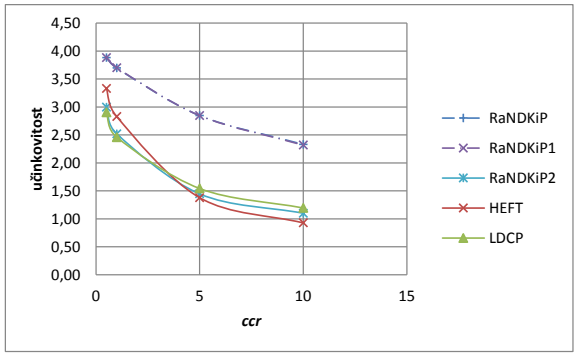
Slika 5.23: Testni slučajevi iz repozitorija Pegasus za različit η, raznorodna komunikacija.



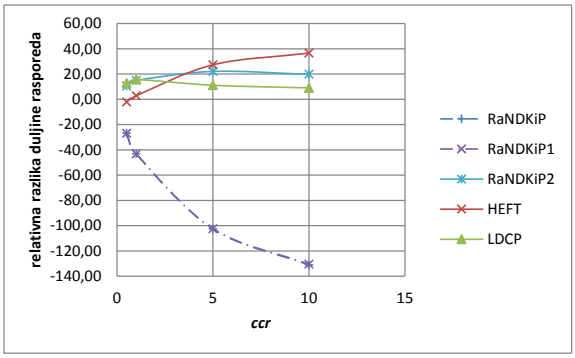
(a) SLR



(b) Ubrzanje



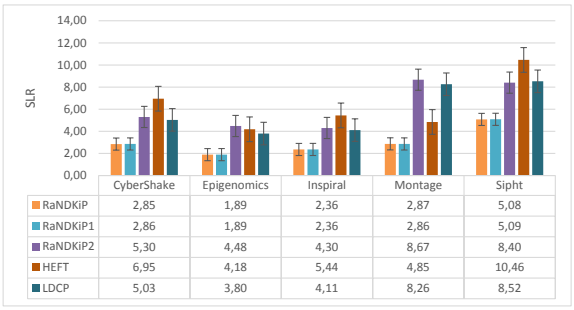
(c) Učinkovitost



(d) Relativna razlika duljine rasporeda

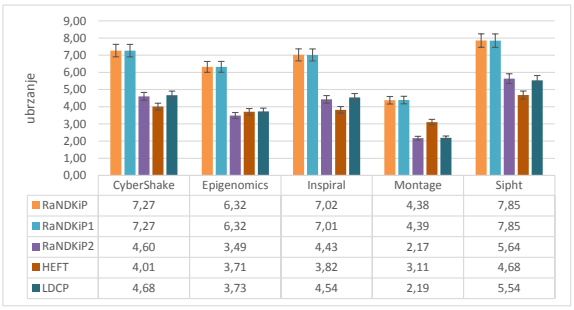
Slika 5.24: Testni slučajevi iz repozitorija Pegasus za različit *ccr*, raznorodna komunikacija.

Kao što je spomenuto u pododjeljku 5.1.2, postoje različite strukture opterećenja Pegasus. Samim time u sljedećim odlomcima bit će analiziran utjecaj strukture opterećenja na dobivene rezultate. Slika 5.25 prikazuje ovisnost već navedenih mjernih veličina o strukturi opterećenja Pegasus. Slika 5.25a prikazuje vrijednosti SLR-a na kojoj je uočljivo da se općenito najlošiji rezultati dobivaju nad opterećenjem *Sipht*, s vrijednostima do 10 kod postupka HEFT. Razlog tomu je u činjenici da se opterećenje *Sipht* sastoji od velikog broja ulaznih vrhova grafa. Samim time nastaju velike razlike u kritičnim putevima, oni su kraći i komunikacija snažno utječe na odabir. To je ujedno i razlog zašto RaNDKiP2 i LDCP postižu manje vrijednosti SLR-a od postupka HEFT. Na svim opterećenjima vidljivo je kako RaNDKiP i njegova izvedba RaNDKiP1 postižu podjednake rezultate, što navodi na to da je najvažniji parametar razlike između pojedinih postupaka raspoređivanja u načinu ažuriranja rangova nakon svakog koraka raspoređivanja. Najniže vrijednosti SLR-a postižu se za opterećenje *Epigenomics*, jer ono sadrži najjednostavniji oblik paralelizma. Na tom opterećenju uočljivo je i to da izvedba RaNDKiP2 postiže najveće vrijednosti, zatim HEFT, a tek onda LDCP. Slično je i za ubrzanje, slika 5.25b, no s nešto uniformnijim rezultatima za različit oblik opterećenja. Kod tipa opterećenja *Montage* HEFT iznenađujuće daje bolje rezultate od svih postupaka osim RaNDKiP i RaNDKiP1. Navedeno je slučaj zbog toga što je velik dio tog opterećenja serijske naravi, tako da dinamička izrada rangova izvršnih jedinica, a samim time i kritičnih puteva, nema pretjeranu svrhu. S obzirom na to da su odnosi vrijednosti dobivene za učinkovitost gotovo identične kao i za ubrzanje, one ovdje nisu razmatrane. Posljednje, kod uvida u relativnu razliku duljine rasporeda na slici 5.25c vidi se da postupci RaNDKiP i RaNDKiP1 postižu najbolje rezultate u odnosu na druge postupke kod opterećenja *Montage*.



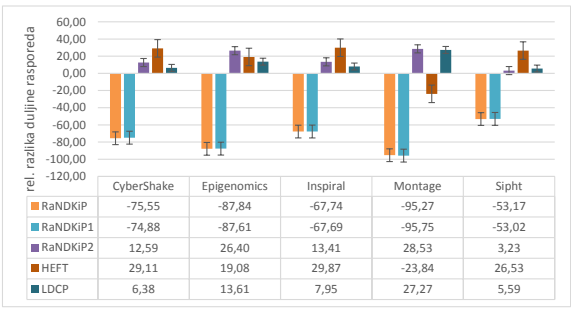
	CyberShake	Epigenomics	Inspiral	Montage	Sipht
RaNDKiP	2,85	1,89	2,36	2,87	5,08
RaNDKiP1	2,86	1,89	2,36	2,86	5,09
RaNDKiP2	5,30	4,48	4,30	8,67	8,40
HEFT	6,95	4,18	5,44	4,85	10,46
LDCP	5,03	3,80	4,11	8,26	8,52

(a) SLR



	CyberShake	Epigenomics	Inspiral	Montage	Sipht
RaNDKiP	7,27	6,32	7,02	4,38	7,85
RaNDKiP1	7,27	6,32	7,01	4,39	7,85
RaNDKiP2	4,60	3,49	4,43	2,17	5,64
HEFT	4,01	3,71	3,82	3,11	4,68
LDCP	4,68	3,73	4,54	2,19	5,54

(b) Ubrzanje



	CyberShake	Epigenomics	Inspiral	Montage	Sipht
RaNDKiP	-75,55	-87,84	-67,74	-95,27	-53,17
RaNDKiP1	-74,88	-87,61	-67,69	-95,75	-53,02
RaNDKiP2	12,59	26,40	13,41	28,53	3,23
HEFT	29,11	19,08	29,87	-23,84	26,53
LDCP	6,38	13,61	7,95	27,27	5,59

(c) Relativna razlika duljine rasporeda

Slika 5.25: Testni slučajevi Pegasus, različite strukture programa, raznorodna komunikacija.

Stoga zaključujemo kako je taj tip opterećenja prilagođen navedenim postupcima, već toga što zbog već navedenoga izraženog serijskog dijela opterećenja LDCP i RaNDKiP2 postižu slabije rezultate. Sljedeći jest *Epigenomics*, zatim *CyberShake*, *Inspiral* te naposljetku *Sipht*.

Analiza slučaja 3 pokazuje da veća strukturiranost programa uvelike povećava performanse svih postupaka raspoređivanja. Naime, kod svih pokazatelja kvalitete vidljiva su poboljšanja u odnosu na slučajeve generirane alatom DAGGer. Također se uočava da je ovdje prednost postupaka RaNDKiP i RaNDKiP1 nad ostalima nešto manja, nego što je to bio slučaj kod programa DAGGer. To ukazuje na činjenicu da se navedeni postupci bolje nose i raznorodnim strukturama programa od drugih postupaka. No usprkos tome, razlike u performansama u korist postupaka RaNDKiP i RaNDKiP1 u odnosu na ostale još su uvijek značajne i nikako ne umanjuju njihovu primjenjivost kod programa iz repozitorija Pegasus.

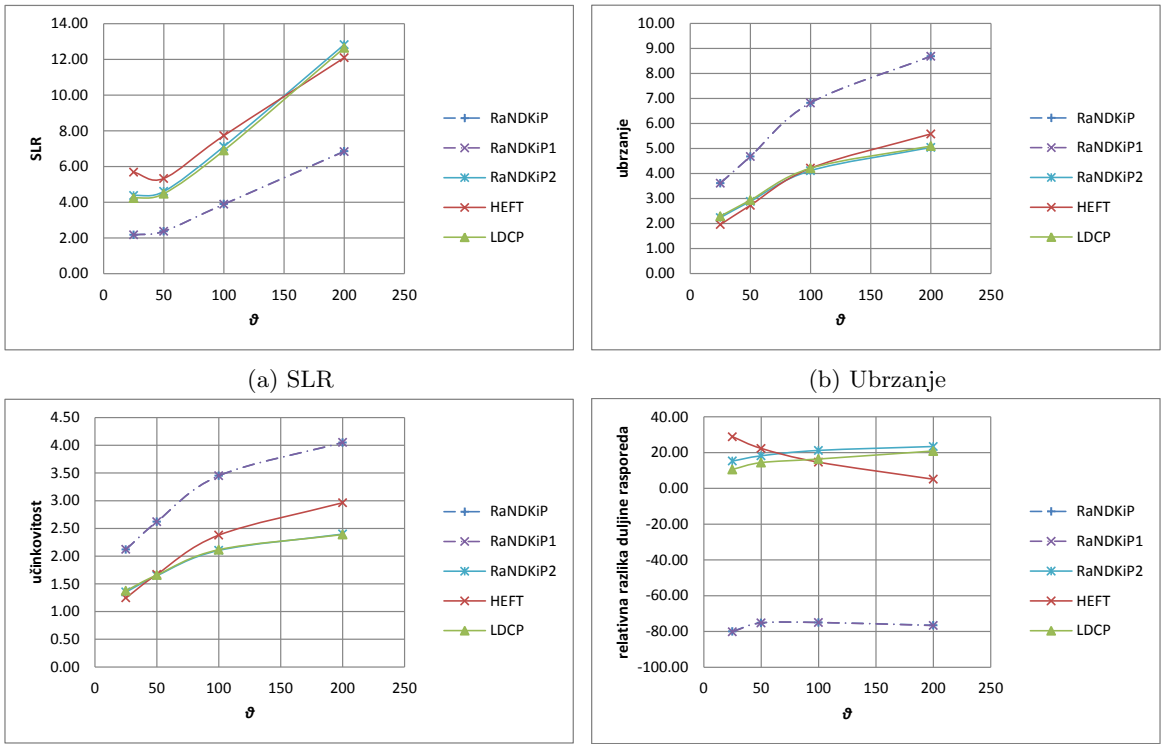
Kako bi prikazali performanse naših unaprijeđenih postupaka RaNDKiP u što većem broju slučajeva, idućim slučajem u narednom pododjeljku prikazujemo njihove rezultate nad programima generiranima alatom TGFF.

5.4.4 Slučaj 4: raspoređivanje programa generiranih alatom TGFF na platforme s raznorodnim komunikacijskim vezama

U ovom pododjeljku bit će prikazani rezultati mjernih veličina prilikom upotrebe postupaka raspoređivanja na testnim slučajevima generiranima alatom TGFF. Iz istog razloga kao i kod prethodnog slučaja, prikazat će se raspoređivanje na računalne platforme s raznorodnim komunikacijskim vezama. Uz to, kao što je već spomenuto ranije u pododjeljku 5.1.1, neće biti prikazani svi oblici grafova koji se isporučuju s navedenim alatom, već samo oni koji se u značajnijoj mjeri međusobno razlikuju. Osim ovisnosti standardnih mjernih veličina o ulaznim parametrima ϑ, ρ, η i *ccr* prikazat ćemo i njihovu ovisnost o strukturi grafa generiranog TGFF-om.

Ako se promotri ovisnost mjernih veličina o broju izvršnih jedinica programa (slika 5.26), uočava se da su im vrijednosti slične kao i kod radnih opterećenja Pegasus. S obzirom na to da je komunikacija strukturirana, postoje značajne razlike u odnosu na grafove generirane alatom DAGGer, tako da su vrijednosti SLR-a, primjerice, vrlo slične među postupcima RaNDKiP2, LDCP i HEFT, što pokazuje i slika 5.26a. Strukturiranost i manji obujam komunikacije uzrokuje također i manju razliku u prednosti postupaka RaNDKiP i RaNDKiP1 u odnosu na ostale postupke, nego što je to slučaj kod testnih slučajeva generiranih alatom DAGGer. No i dalje se radi o približno dvostrukoj razlici. Ono što je primjetno jest da SLR raste gotovo linearno povećanju broja izvršnih jedinica te je za očekivati isti trend i za $\vartheta > 200$. Kod ubrzanja i učinkovitosti, slike 5.26b i 5.26c, situacija je vrlo slična pri usporedbi performansi među različitim postupcima raspoređivanja. Porastom broja izvršnih jedinica navedene mjerne veličine ne rastu linearno ali njihov rast ukazuje na to da se postupci dobro nose s porastom izvršnih jedinica, kada su ubrzanje i učinkovitost u pitanju. Pogotovo je to izraženo, ali uz veći porast, za postupke RaNDKiP i RaNDKiP1, koji drže gotovo dvostruko bolje performanse od sljedećeg najboljeg — HEFT-a. Kod učinkovitosti je također primjetan porast performansi postupka HEFT u odnosu na RaNDKiP2 i LDCP

te je i njegova skalabilnost s porastom broja izvršnih jedinica programa gotovo identična kao i kod postupaka RaNDKiP i RaNDKiP1.



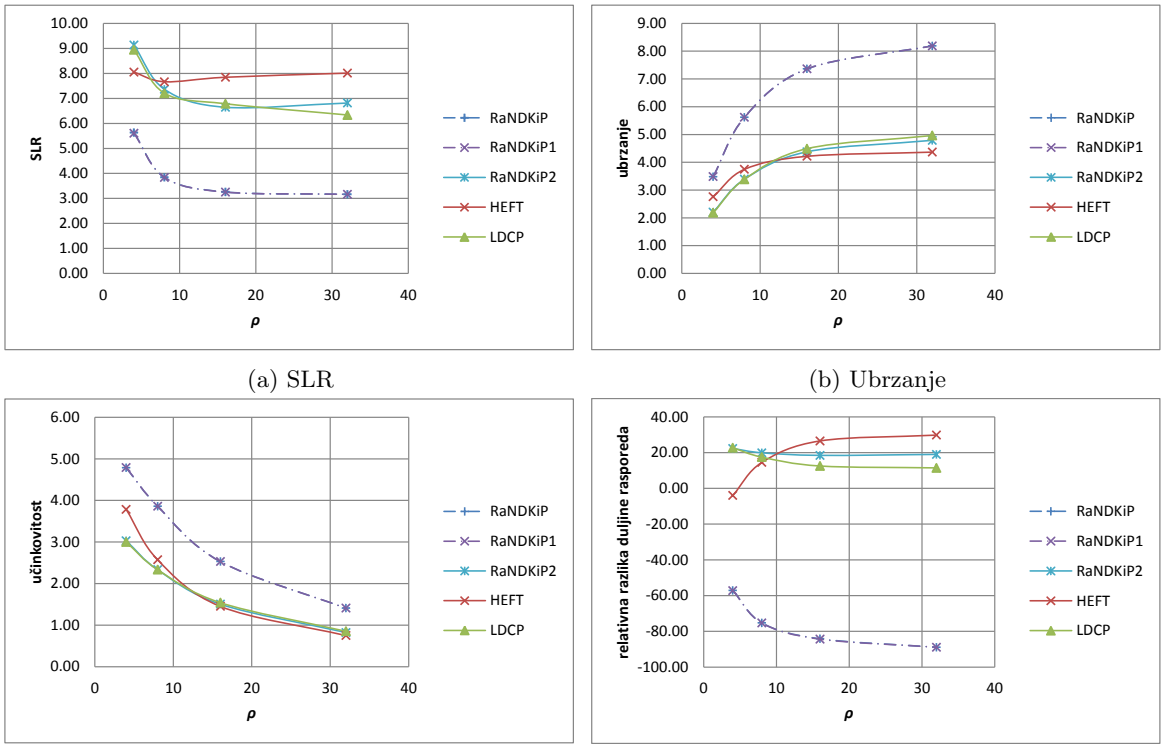
(a) SLR (b) Ubrzanje

(c) Učinkovitost (d) Relativna razlika duljine rasporeda

Slika 5.26: Testni slučajevi generirani alatom TGFF za različit ϑ, raznorodna komunikacija.

Slično je primjetno i kod relativne razlike duljine rasporeda na slici 5.26d. Naime, postupci RaNDKiP i RaNDKiP1 u navedenoj mjernoj veličini drže gotovo konstantnu razliku u odnosu na druge postupke od 80% u njihovu korist, dok se razlika postupka HEFT u odnosu na ostale povećava porastom broja izvršnih jedinica. Pokazatelj je to dobre skalabilnosti postupka HEFT, s obzirom na to da i on uzima raznorodnost komunikacije u obzir u određenoj mjeri (njezin prosjek).

Ako se promotre razlike mjerenih veličina pri upotrebi različitih postupaka uz različit broj obradbenih jedinica ρ, primjetni su slični odnosi kao i kod radnih opterećenja Pegasus, slika 5.27, osim što su općenito performanse svih postupaka nešto manje. Razlog tome je u većoj raznorodnosti programa generiranih alatom TGFF nego što je to slučaj kod programa iz repozitorija Pegasus. Kao i kod dosadašnjih mjerenja, vrijednosti mjerenih veličina stabiliziraju se i ostaju konstantne porastom parametra ρ i to već nakon njih 16, što je vidljivo za SLR na slici 5.27a te za relativnu razliku duljine rasporeda, slika 5.27d. Jedine mjerne veličine koje nastavljaju blago rasti su ubrzanje, slika 5.27b i učinkovitost, slika 5.27c. HEFT ovdje daje nešto lošije rezultate porastom broja obradbenih jedinica u odnosu na ostale postupke, jer nema potpunu podršku za raznorodnu komunikaciju.



(a) SLR

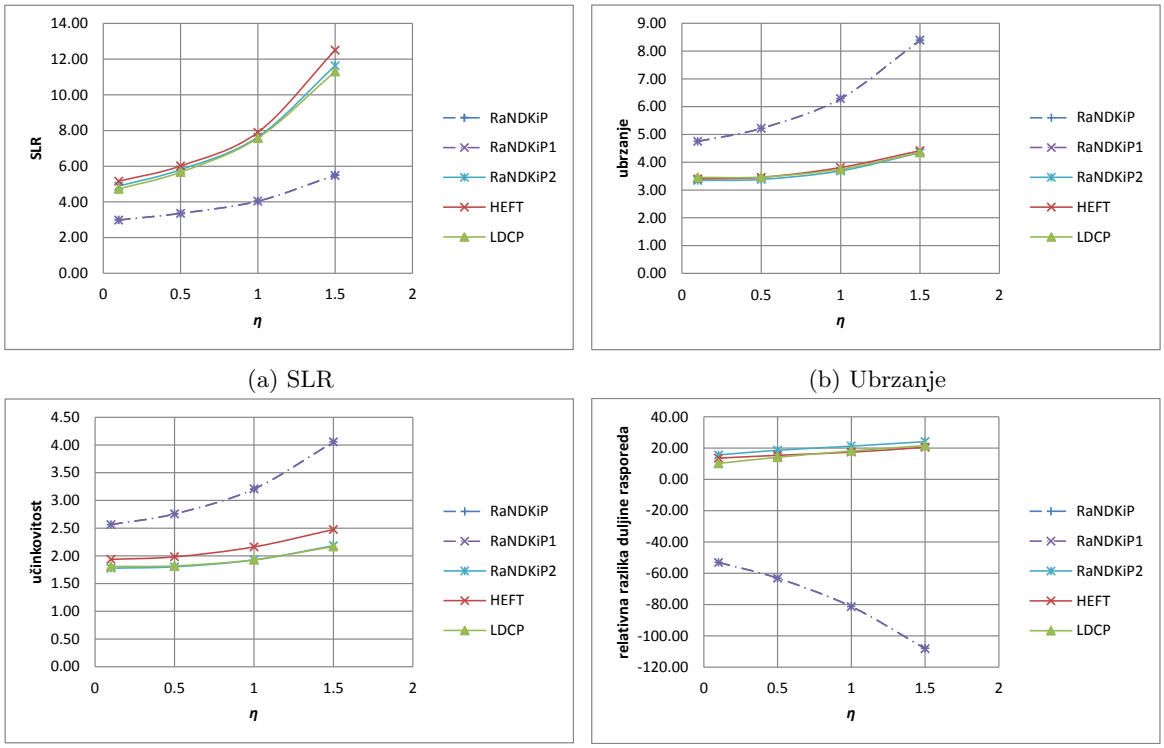
(b) Ubrzanje

(c) Učinkovitost

(d) Relativna razlika duljine rasporeda

Slika 5.27: Testni slučajevi generirani alatom TGFF za različit ρ, raznorodna komunikacija.

Kod povećanja vrijednosti čimbenika raznorodnosti, razlike među različitim postupcima raspoređivanja u odnosu na opterećenja Pegasus slične su kao i za promjenu broja obradbenih jedinica sklopovlja, slika 5.28. Prema očekivanjima, performanse svih postupaka su u padu porastom raznorodnosti, no to se ne odražava toliko za postupke RaNDKiP i RaNDKiP1. Performanse ostalih postupaka su vrlo slične, osim za učinkovitost (slika 5.28c), gdje je primjetna mala prednost postupka HEFT.



(a) SLR

(b) Ubrzanje

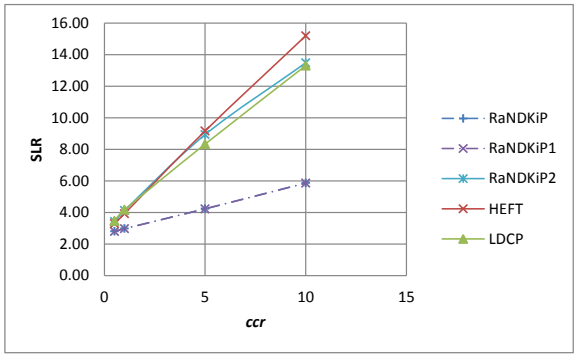
(c) Učinkovitost

(d) Relativna razlika duljine rasporeda

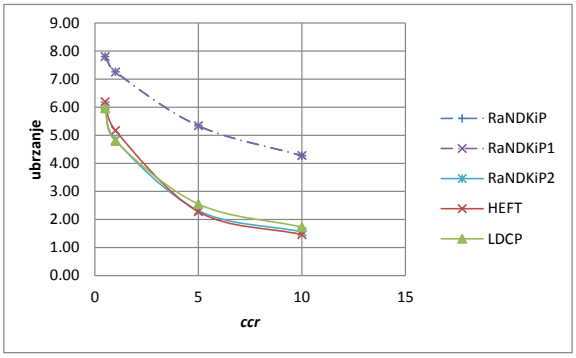
Slika 5.28: Testni slučajevi generirani alatom TGFF za različit η, raznorodna komunikacija.

Performanse u odnosu na CCR također su vrlo slične rezultatima za radna opterećenja Pegasus, slika 5.29, osim što je postupak HEFT nešto osjetljiviji na njegov porast te daje malo veće vrijednosti SLR-a, slika 5.29a. U ostalim slučajevima, slike 5.29b, 5.29c i 5.29d, svi postupci daju vrlo slične vrijednosti pokazatelja kvalitete, osim RaNDKiP i RaNDKiP1 koji daju značajno bolje performanse u odnosu na ostale postupke.

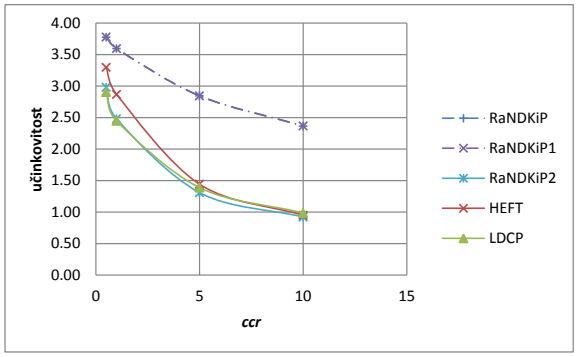
Performanse postupaka raspoređivanja u odnosu na tip grafa TGFF-a prikazane su na slici 5.30. Očekivano, svi postupci daju najlošije rezultate kod testnog slučaja *sp_rand*. To se događa zbog toga što su programi *sp_rand* visoko raznorodni, slični postupku generiranja DAGGer te sadržavaju velik broj bridova grafa (komunikacije među izvršnim jedinicama programa). Također, vidljivo je da je ta razlika najuočljivija kod SLR-a, slika 5.30a, jer je to ujedno i najosjetljivija mjerna jedinica. Postupci RaNDKiP i RaNDKiP1, osim što daju gotovo 90% bolje rezultate od ostalih postupaka, ujedno su i najmanje osjetljivi na različit tip grafa TGFF-a. Ostali postupci imaju međusobno vrlo slične performanse, uz to da HEFT pokazuje veće nedostatke kod opterećenja *sp_rand*, slika 5.30b. Relativna razlika duljine rasporeda prikazuje da je opterećenje *packets* najpogodnije postupcima RaNDKiP i RaNDKiP1, nakon kojeg slijedi *kseries_parallel*, dok su u najmanjoj prednosti kod tipa programa *sp_rand*, slika 5.30c. Za ostale postupke iz iste se slike vidi da postupcima RaNDKiP2 i LDCP najviše pogoduje opterećenje *sp_rand*, dok je to za HEFT *packets* i *kbasic_task*. U usporedbi postupaka RaNDKiP2 i LDCP, LDCP postiže nešto slabije performanse za testne slučajeve *kseries_parallel*, *kseries_parallel_xover* i *simple*, koji sadrže visoku razinu paralelizma (manja svrhovitost dinamičkog rangiranja).



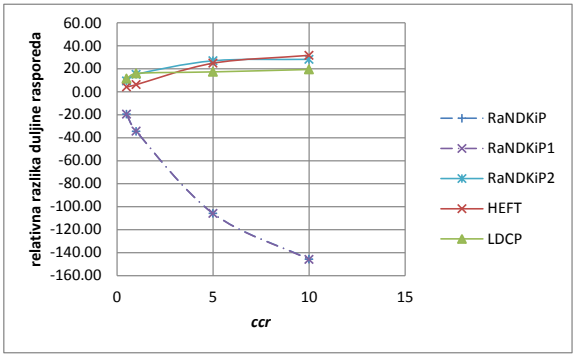
(a) SLR



(b) Ubrzanje

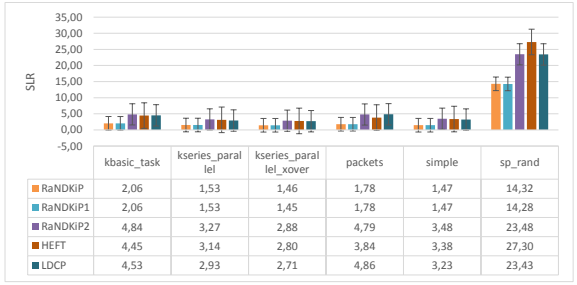


(c) Učinkovitost

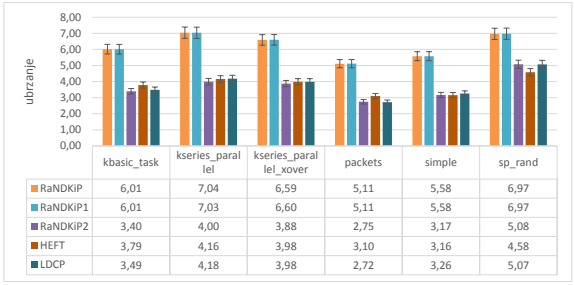


(d) Relativna razlika duljine rasporeda

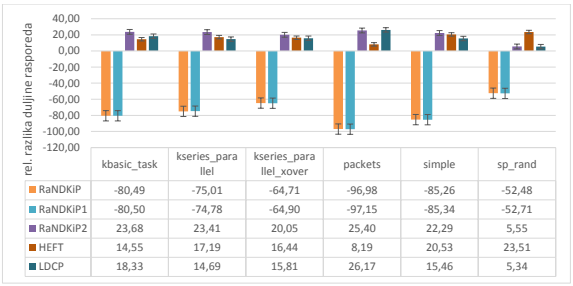
Slika 5.29: Testni slučajevi generirani alatom TGFF za različit *ccr*, raznorodna komunikacija.



(a) SLR



(b) Ubrzanje



(c) Relativna razlika duljine rasporeda

Slika 5.30: Testni slučajevi generirani alatom TGFF, različite strukture programa, raznorodna komunikacija.

Ako se promotri slučaj 4, primjetno je da su i dalje postupci RaNDKiP i RaNDKiP1 u velikoj prednosti u odnosu na druge postupke, što im opravdava primjenjivost u računalnim sustavima s raznorodnim komunikacijskim vezama. Ako se usporede rezultati svih promotrenih slučajeva, vidljivo je kako osim

glavnih ulaznih parametara ϑ, ρ, η i *ccr* važnu ulogu ima i struktura programa. Tako za pravilno strukturirane programe poput onih iz repozitorija Pegasus, svi postupci postižu bolje rezultate nego što je to slučaj kod programa generiranih alatom DAGGer i u nešto manjoj mjeri bolje nego kod programa generiranih alatom TGFF. No, u svim slučajevima s raznorodnom komunikacijom sklopovlja naši unaprijeđeni postupci RaNDKiP i RaNDKiP1 pokazuju mnogo bolje rezultate od ostalih. Na drugom je mjestu postupak HEFT, jer sadrži podršku za raznorodne komunikacijske veze sklopovlja u obliku njihovih srednjih vrijednosti.

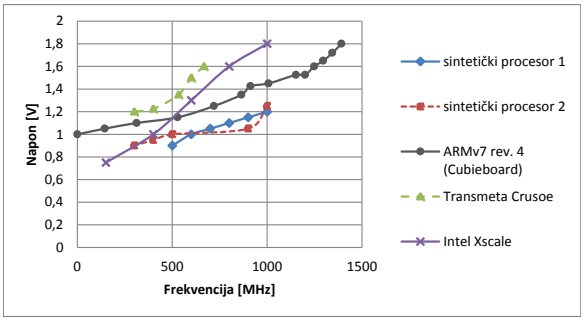
U sljedećem pododjeljku bit će prikazani parametri kvalitete analize utroška energije postupkom LVI, kao dijelom postupka RaNDKiP, zbog uvida u potencijal dobivenih rasporeda za uštedom energije.

5.5 Rezultati analize uštede energije primjenom postupka RaNDKiP

Osim vremenske analize postupka RaNDKiP, proveli smo i postupak analize uštede energije primjenom navedenog postupka. Naime, s obzirom na to da je jedan od koraka navedenog postupka LVI (labavljenje vremenskih isječaka s ciljem uštede energije), bez utjecaja na dobivenu duljinu rasporeda dobivamo određenu uštedu energije skaliranjem frekvencije obradbenih jedinica za nekritične izvršne jedinice programa. Analiza utroška energije provedena je na način:

1. da se prvo definiraju energetske karakteristike obradbenih jedinica simulirane računalne platforme,
2. zatim se procjenjuje teoretski utrošak energije za dobiveni raspored koji daju pojedini postupci raspoređivanja,
3. nakon toga se provodi postupak LVI i analizira se novodobiveni utrošak energije,
4. te se naposljetku usporedbom s početnim utroškom energije dobiva njezina potencijalna ušteda.

Energetske karakteristike obradbenih jedinica definiraju se na način opisan u pododjeljku 4.2.4. U tu svrhu na slici 5.31 predstavljamo DVFS profile za različite platforme dostupne u literaturi. Prve su karakteristike sintetički generiranih obradbenih jedinica prema [3], nazvanih "sintetički procesor 1" i "sintetički procesor 2". Također, iz istog izvora preuzete su karakteristike procesora Transmeta Crusoe i Intel Xscale. Na istoj je slici postavljena i DVFS karakteristika već korištene Cubieboard2 platforme, konkretnije, procesora koji ona sadrži — ARMv7 rev. 4 — prema podatcima iz [4].



Slika 5.31: DVFS karakteristike različitih procesora, prema [3, 4].

U ovom dijelu eksperimentalne analize koristit ćemo sljedeće pretpostavke:

- DVFS karakteristike svih obradbenih jedinica testnih slučajeva su identične i odgovaraju onoj ARMv7 rev. 4 sa slike 5.31, uz to što:
  - navedene karakteristike često nisu dostupne za sve obradbene jedinice u praksi,
  - na taj se način omogućuje jednostavnost prikaza rezultata analize i
  - ne gubi se primjenjivost rezultata na ostale platforme.
- frekvenciju svih obradbenih jedinica moguće je mijenjati kontinuirano u intervalu $[0, f^{max}]$ karakteristikom opisanom pomoću (4.70) u pododjeljku 4.2.4.

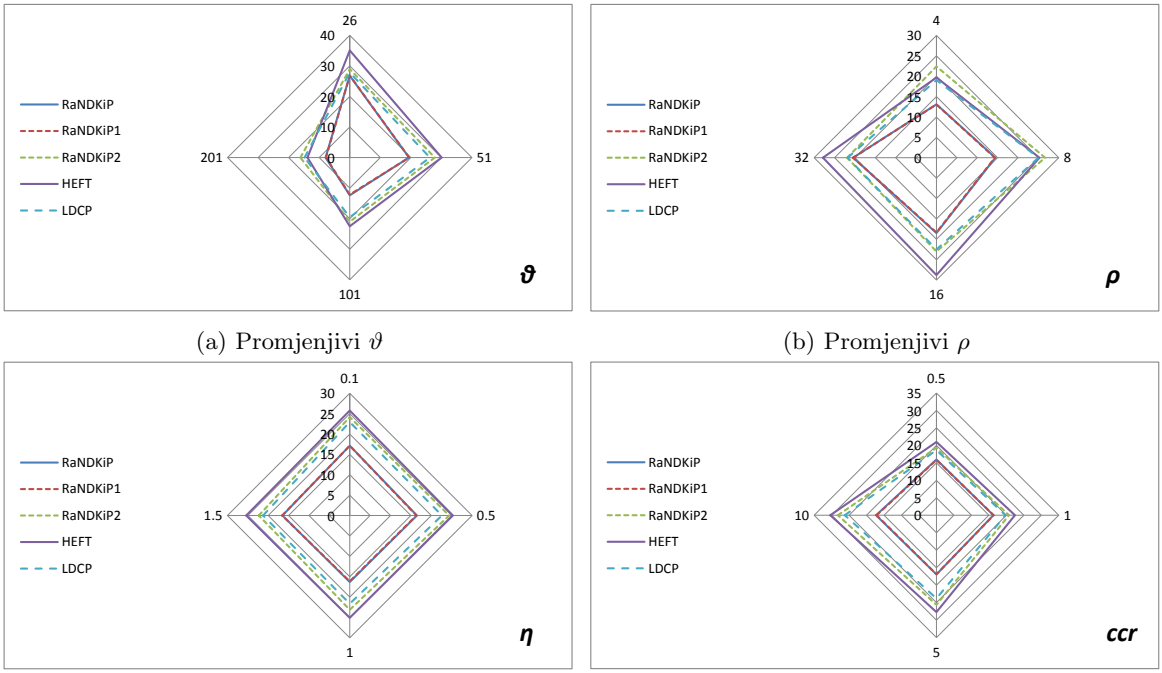
Za raspored programa upotrijebljeni su već spomenuti i ranije korišteni postupci raspoređivanja na temelju rangiranja jedinica programa: RaNDKiP , RaNDKiP1, RaNDKiP2, HEFT i LDCP. Koriste se dva glavna pokazatelja: ušteđena energija ε_{uk}^{raz}, iskazana u postocima u odnosu na polazni raspored te procijenjeni utrošak energije ε. Ostale postavke testnih slučajeva opisane su u pododjeljku 5.2.

5.5.1 Slučaj 5: energetske karakteristike programa generiranih alatom DAGGer raspoređenih na platforme s raznorodnim komunikacijskim vezama

Prvu promotrenu situaciju energije kao pokazatelja kvalitete rasporeda dobivenih različitim postupcima raspoređivanja čine testni slučajevi generirani alatom DAGGer, uz pretpostavku raznorodnih komunikacijskih veza među obradbenim jedinicama sklopovlja.

Rezultati analize uštede energije slučaja 5 prikazani su slikom 5.32. Povećanjem broja izvršnih jedinica programa opada i ušteda energije dobivena postupkom LVI, što je i vidljivo na slici 5.32a, jer velikim brojem izvršnih jedinica raspored postaje popunjeniji. Svi postupci pokazuju približno isti trend smanjenja uštede energije povećanjem ϑ, osim postupaka LDCP i RaNDKiP2 koji pokazuju nešto manji trend pada. No, to je izravno vezano uz njihove vremenske karakteristike u tom slučaju (lošiji raspored). Vrijednosti ušteđene energije kreću se od 35% za približno 25 izvršnih jedinica u programu (HEFT) pa sve do samo 10% za 200 izvršnih jedinica (RaNDKiP i RaNDKiP1), što je pokazatelj da prostora za uštedu ima te da su kvaliteta rasporeda i ušteđena energija proporcionalni. Povećanjem broja obradbenih jedinica ušteđena energija raste, jer se time razrjeđuje raspored i nastaje više prostora za labavljenje vremenskih isječaka, što pokazuje slika 5.32b. Uštede se kreću od približno 13% za 4 obradbene jedinice (RaNDKiP i RaNDKiP1) pa sve do približno 29% za 16 obradbenih jedinica. Primjetno je također da većina postupaka ostvaruje najveću uštedu na 16 obradbenih jedinica, no to bi se razlikovalo pri raspoređivanju većih programa ($\vartheta > 200$), tako da bi točka zasićenja bila postignuta tek za veći broj obradbenih jedinica sklopovlja. Kod promjene vrijednosti čimbenika raznorodnosti, slika 5.32c, nema značajnije promjene u dobitcima energije. Prosječni dobitak je približno konstantan za sve vrijednosti čimbenika raznorodnosti i iznosi oko 25%. Nadalje, na slici 5.32d vidljivo je kako se povećanjem CCR-a blago povećava i ušteda energije. To povećanje je naviše vidljivo za $ccr > 1$. Navedeno je uzrokovano većim odstupanjima duljine vremenskih isječaka zbog velikog udjela komunikacije u rasporedu, čime dolazi do većih praznina u rasporedu kod postupka LVI. No, ono što se ovdje ističe jest da su postupci RaNDKiP

i RaNDKiP1 gotovo imuni na povećanje CCR-a, što je još jedan od čimbenika koji opravdava njihovu upotrebu za visoko raznorodne sustave. Ujedno, za njih je i ušteda energije najmanja, što ukazuje na vrlo dobru popunjenost rasporeda. Ovdje se uštede kreću od 15 do 30%. Zajednička analiza otkriva da postupak HEFT prednjači po uštođenoj energiji, no to je pokazatelj kako da raspored dobiven tim postupkom ima dosta prostora te procesorsko vrijeme nije najbolje iskorišteno. Odnosi među dobitcima energije u pravilu slijede trend vremenskih performansi pojedinih postupaka. Također, uočava se kako je parametar koji najviše utječe na količinu uštođene energije broj izvršnih jedinica programa, zatim broj obradbenih jedinica sklopovlja i u manjem obujmu CCR.



(a) Promjenjivi ϑ

(b) Promjenjivi ρ

(c) Promjenjivi η

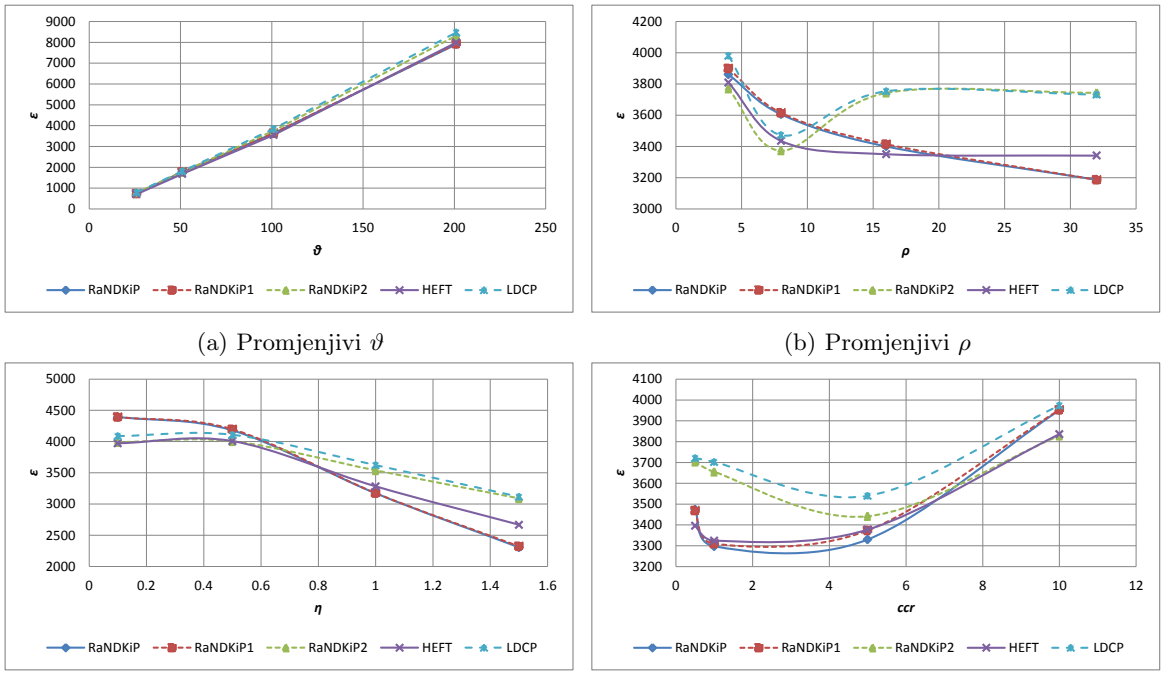
(d) Promjenjivi *ccr*

Slika 5.32: Ušteda energije kod testnih slučajeva generiranih alatom DAGGer, platforma s raznorodnom komunikacijom.

Sljedeći analizirani pokazatelj jest procijenjeni utrošak energije ε pri izvođenju programa prema dobivenom rasporedu. Mjerne jedinice korištene za navedeni pokazatelj su simboličke naravi (zbog toga što vremenska jedinica nije određena), no odnosi među postupcima dobiveni ovdje vrijede i za stvarne slučajeve, jer se uvrštavanjem stvarnih parametara trendovi ne mijenjaju (čimbenik platforme κ i mjerna jedinica vremena (primjerice sekunda)).

Dobiveni iznosi utroška energije prikazani su slikom 5.33. Primjetno je kako ukupni utrošak energije linearno raste porastom broja izvršnih jedinica programa, slika 5.33a. Razlike u utrošku za različite postupke su gotovo zanemarive, zbog toga što gustoća rasporeda ne određuje utrošak energije izravno. U rasporedu svih postupaka nalaze se sve izvršne jedinice, tako da su jedine razlike koje se mogu pojaviti u slučaju da je raspored stavio veći broj izvršnih jedinica na onu obradbenu jedinicu na kojoj će njihovo vrijeme izvršavanja biti najmanje. S druge strane, povećanjem broja obradbenih jedinica ne naziru se velike razlike u ukupnom utrošku energije, što pokazuje slika 5.33b. Za očekivati je da u slučaju kada bi broj obradbenih jedinica bio blizak broju izvršnih jedinica, navedene razlike bi bile veće, jer bi postupci

raspoređivanja tada mogli preusmjeravati izvršne jedinice na obradbene jedinice na kojima bi njihovo trajanje bilo smanjeno. Razlike među postupcima nisu značajne, no vidljivo je kako ukupan utrošak energije u manjoj mjeri ovisi o kvaliteti rasporeda (u smislu njegove duljine, *trasp*). Također, ovdje su vidljive i prednosti postupaka RaNDKiP i RaNDKiP1 koji povećanjem broja obradbenih jedinica imaju trend smanjenja utroška energije (za razliku od ostalih postupaka kojima utrošak energije ostaje konstantan). Kod povećanja vrijednosti čimbenika raznorodnosti primjetan je pad ukupnog utroška energije, slika 5.33c. Razlog tome leži u činjenici da se u sustavu porastom raznorodnosti pojavljuju obradbene jedinice na kojima je vrijeme izvršavanja jedinica programa kraće. Nadalje, pretpostavili smo jednak model utroška energije za sve obradbene jedinice u sustavu, tako da će navedeno rezultirati ukupno manjim utroškom energije (jer će vrijeme izvršavanja biti smanjeno). Ovdje su opet RaNDKiP i RaNDKiP1 u prednosti, jer porastom vrijednosti čimbenika raznorodnosti imaju veći trend smanjenja ukupnog utroška energije, time pokazujući da uspijevaju bolje popuniti raspored i prilagoditi se većim raznorodnim okruženjima. Kod povećanja CCR-a situacija je nešto drugačija, kao što se vidi na slici 5.33d. Naime, povećanjem CCR-a povećava se i ukupni utrošak energije. Navedeno se događa zbog toga što povećanjem CCR-a raste udio komunikacije u ukupnom rasporedu, što dovodi do toga da je postupcima raspoređivanja važnije trajanje komunikacije (jer on značajnije utječe na duljinu rasporeda koju oni pokušavaju smanjiti), nego trajanje samih izvršnih jedinica. Zbog toga postupci rade manje optimalan raspored po pitanju smještanja samih izvršnih jedinica (raste tendencija njihovog stavljanja na obradbene jedinice gdje im se povećava vrijeme izvršavanja) i to dovodi do povećanja njihovog ukupnog trajanja i samim time povećanjem ukupnog utroška energije.



(a) Promjenjivi ϑ

(b) Promjenjivi ρ

(c) Promjenjivi η

(d) Promjenjivi *ccr*

Slika 5.33: Utrošak energije kod testnih slučajeva generiranih alatom DAGGer, platforma s raznorodnom komunikacijom.

Zajednička su svojstva da se u manje optimalnom rasporedu po pitanju vremenskih pokazatelja

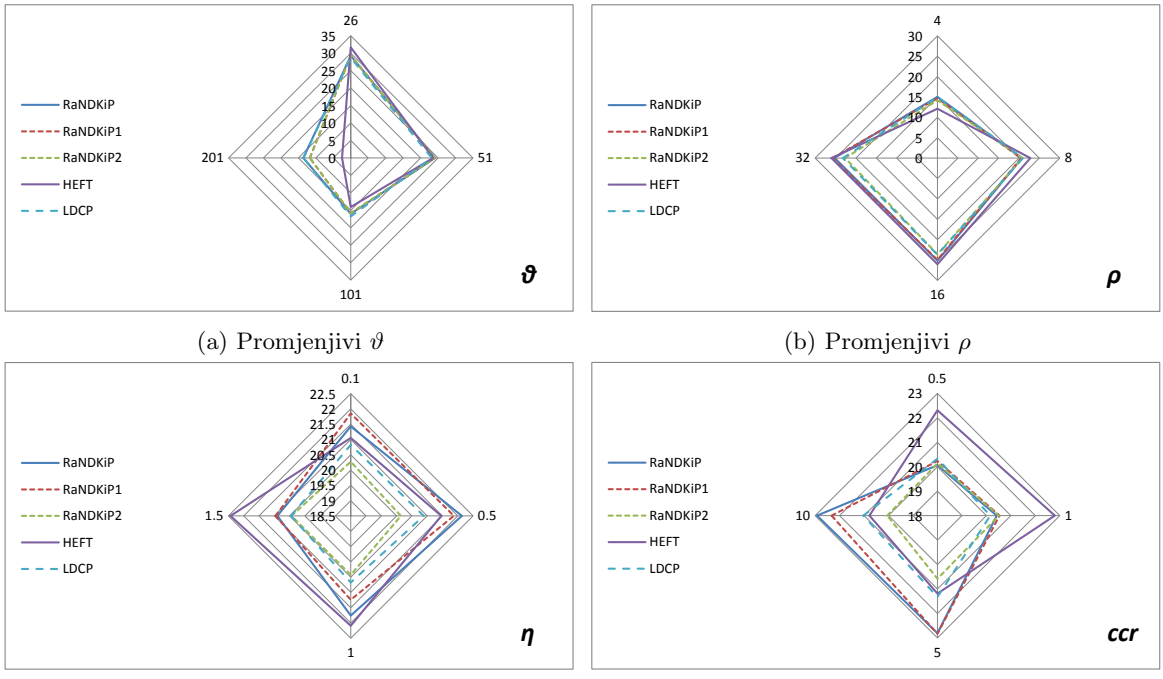
kvalitete dobivaju veće uštede energije. No, u takvim je rasporedima i inicijalni utrošak energije veći. Jedini ulazni parametar koji nema utjecaj na uštedu energije je čimbenik raznorodnosti, dok porastom ostalih trend uštede energije raste. Kod ukupnog utroška energije najveći pak utjecaj ima čimbenik raznorodnosti, čijim porastom pada i njezin utrošak.

U idućem pododjeljku predstavljamo slučaj s homogenom mrežom računalne platforme, samo kako bi prikazali osnovne trendove uštede i utroška energije u toj situaciji.

5.5.2 Slučaj 6: energetske karakteristike programa generiranih alatom DAGGer raspoređenih na platforme s homogenim komunikacijskim vezama

Iako nam primarna usredotočenost nije analiza sustava s homogenom komunikacijom, zbog potpunijeg prikaza performansi unaprijeđenog postupka raspoređivanja RaNDKiP i njegovih izvedbi RaNDKiP1 i RaNDKiP2 u ovom pododjeljku prikazujemo energetske karakteristike tog slučaja.

Na slici 5.34 prikazano je smanjenje utroška (ušteda) energije korištenjem postupka LVI pri raspoređivanju u kombinaciji sa različitim postupcima raspoređivanja. Prema slici 5.34a, kod povećanja broja izvršnih jedinica programa ušteda energije pada iz istih razloga kao i kod sustava s raznorodnom komunikacijom, (slika 5.32a). Ističe se samo drastično manja ušteda kod postupka HEFT, s obzirom na to da smo pokazali da ona postiže i bolje vremenske rezultate kod slučaja s homogenom komunikacijom platforme. Za različit broj obradbenih jedinica sklopovlja, (slika 5.34b) također je situacija slična kao i kod sustava s raznorodnom komunikacijom, time da različiti postupci daju vrlo slične vrijednosti.



(a) Promjenjivi ϑ

(b) Promjenjivi ρ

(c) Promjenjivi η

(d) Promjenjivi *ccr*

Slika 5.34: Ušteda energije kod testnih slučajeva generiranih alatom DAGGer, platforma s homogenom komunikacijom.

Ovdje su primjetne i nešto manje vrijednosti za male platforme — ušteda je manja zbog bolje is-

korištenosti vremenskog prostora.

Kod promjene vrijednosti čimbenika raznorodnosti, s druge strane, razlike su minimalne (unutar 2%), slika 5.34c, s opet vrlo ujednačenim vrijednostima za različite postupke. Opet se tome nalazi uzrok u tretiranju komunikacije različitih postupaka raspoređivanja, dakle podrška za raznorodnost iste, no ovdje je nema, tako da su postupci ujednačeni. Naposljetku, slika 5.34d prikazuje uštedu energije za različite vrijednosti CCR-a, gdje su razlike opet minimalne (unutar 3%), time da HEFT postiže nešto veću uštedu energije za male vrijednosti CCR-a, dok za veće vrijednosti to postižu RaNDKiP i RaNDKiP1. To navodi na činjenicu kako RaNDKiP i RaNDKiP1 kvalitetnije tretiraju pojavu komunikacije u rasporedu i uspijevaju tome bolje prilagoditi raspored. Zajednička je činjenica da svi postupci i kod homogene mreže među obradbenim jedinicama uspijevaju ostvariti značajnu uštedu koja se u prosjeku kreće oko 20%, tako da postupak LVI opravdava svoju upotrebu u sprezi s navedenim postupcima čak i za sustave s homogenom komunikacijom.

U ovom, ali i sljedećim slučajevima, nećemo razmatrati procijenjeni utrošak energije u ovisnosti o parametrima ϑ, ρ, η i ccr, jer u tim slučajevima on ne daje relevantne informacije. Iz istih razloga u narednim slučajevima promatramo samo rasporede na računalnim platformama s raznorodnim komunikacijskim vezama.

Uslijed spomenutog, u idućem slučaju razmatramo programe iz repozitorija Pegasus raspoređene na računalne platforme s raznorodnom komunikacijskom mrežom.

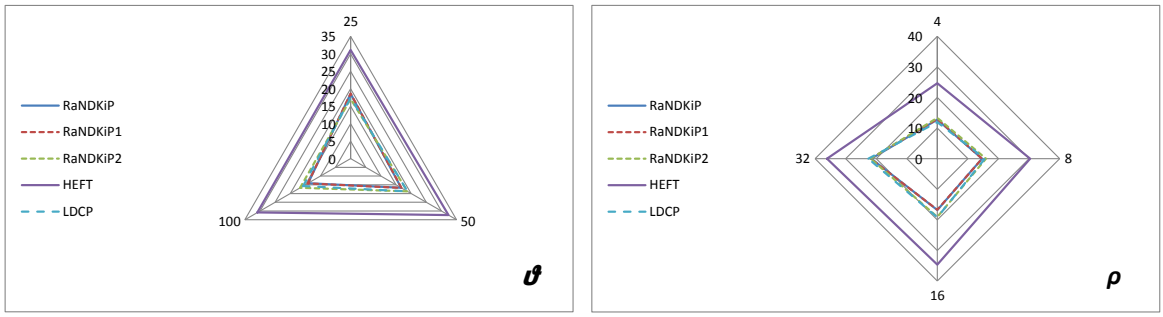
5.5.3 Slučaj 7: energetske karakteristike programa iz repozitorija Pegasus raspoređenih na platforme s raznorodnim komunikacijskim vezama

U ovom pododjeljku promatramo parametre utroška energije kod programa iz radnih opterećenja Pegasus. S obzirom na to da nam je primarni cilj analiza računalnih sustava s raznorodnom komunikacijom, a kratki prikaz parametara utroška energije kod sustava s homogenom komunikacijom dan je u pododjeljku 5.5.2, ovdje će biti razmatran samo slučaj raznorodne komunikacije unutar sklopovlja. U ovoj analizi ne provodimo postupak generiranja stvarne komunikacije, već se oslanjamo na raspored kakvog ga generira sam postupak. Time, HEFT ne prikazuje komunikaciju u rasporedu, tako da pri upotrebi postupka LVI daje privid većih ušteda energije.

Slika 5.35 prikazuje uštedu energije primjenom postupka LVI u sprezi s različitim postupcima raspoređivanja na programima Pegasus. Ovdje koristimo samo tri veličine programa (≈ 25, ≈ 50 i ≈ 100) zbog toga što je iduća veličina programa koja se nudi u repozitoriju Pegasus tek 1000, a koja je prevelika za analizu. Tako slika 5.35a daje nešto drugačiji prikaz kod utroška energije, nego što je to bio slučaj kod programa generiranih alatom DAGGer. Ovdje porast broja izvršnih jedinica programa ne prati značajnije smanjenje utroška energije te je ono najmanje primjetno za postupke RaNDKiP2 i HEFT (unutar 2%), a najviše za RaNDKiP i RaNDKiP1 (unutar 4%). Dobivene vrijednosti ukazuju na to da unaprijeđeni postupci RaNDKiP i RaNDKiP1 imaju dobru skalabilnost te da se uniformno ponašaju i bolje popunjavaju raspored porastom broja izvršnih jedinica. Kod porasta broja obradbenih jedinica sklopovlja, što prikazuje slika 5.35b, svi postupci raspoređivanja imaju isti trend povećanja

uštede energije. Taj trend najbolje opet slijede RaNDKiP i RaNDKiP1, dok ostalima ušteda energije ne raste na platformama s više od 16 obradbenih jedinica (ovdje 32).

Za različite vrijednosti čimbenika raznorodnosti promjene gotovo da i nema što je bio slučaj i kod programa generiranih alatom DAGGer u pododjeljku 5.5.1 i to prikazuje slika 5.35c. Porastom CCR-a i ovdje ušteda energije blago raste zbog većeg broja praznina u rasporedu koje se stvaraju, tj. komunikacija zauzima sve veći dio vremena, slika 5.35d. Ako se promotre sve dobivene vrijednosti vidljivo je kako primjenom postupka LVI na postupke raspoređivanja uvjerljivo najveću uštedu energije ima postupak HEFT, što dovodi do zaključka da taj postupak stvara raspored s najviše prostora među obradbenim vremenskim isječcima. Razlog tome naveden je na početku ovog pododjeljka, (nema prikaz komunikacije u rasporedu). Ostali postupci daju međusobno vrlo slične rezultate, s tim da su oni naviše dosljedni kod postupaka RaNDKiP i RaNDKiP1, (najbolje se prilagođavaju promjeni testnih parametara).



(a) Promjenjivi ϑ

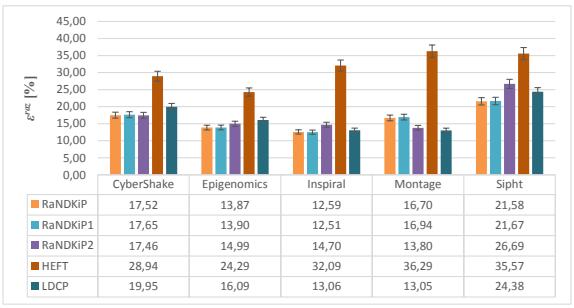
(b) Promjenjivi ρ

(c) Promjenjivi η

(d) Promjenjivi *ccr*

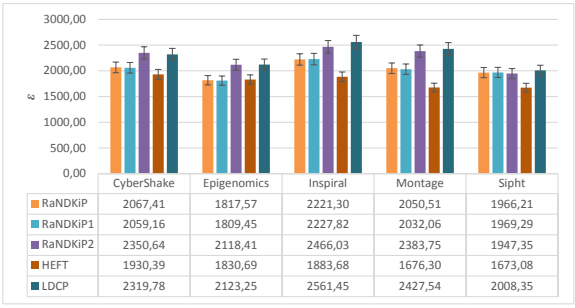
Slika 5.35: Ušteda energije kod testnih slučajeva iz repozitorija Pegasus, platforma s raznorodnom komunikacijom.

Trendovi ukupnog utroška energije slijede slučaj programa generiranih alatom DAGGer, tako da analiziramo samo razlike s obzirom na tip opterećenja. Slika 5.36 prikazuje parametre utroška energije s obzirom na strukturu programa iz repozitorija Pegasus koji se izvode na sklopovlju s raznorodnom mrežom. Ako se promotri ukupna ušteda energije, vidljivo je kako je ona najveća pri korištenju strukture programa *Sipht* iz repozitorija Pegasus, (slika 5.36a). Kao što je bila situacija i u vremenskoj analizi navedenog slučaja, *Sipht* programi imaju najveći stupanj paralelizma te svi postupci raspoređivanja pokušavaju staviti izvršne jedinice na paralelne obradbene jedinice, (da smanje njihovu komunikaciju). No, zbog toga što je obradbenih jedinica premalo da bi se svi smjestili paralelno, ostaju praznine u rasporedu, koje u ovom slučaju iskorištava postupak LVI. Primjetno je također da postupak HEFT ostvaruje najveću uštedu energije iz razloga već navedenih ranije.



	CyberShake	Epigenomics	Inspiral	Montage	Sipht
RaNDKiP	17,52	13,87	12,59	16,70	21,58
RaNDKiP1	17,65	13,90	12,51	16,94	21,67
RaNDKiP2	17,46	14,99	14,70	13,80	26,69
HEFT	28,94	24,29	32,09	36,29	35,57
LDCP	19,95	16,09	13,06	13,05	24,38

(a) Ušteda energije.



	CyberShake	Epigenomics	Inspiral	Montage	Sipht
RaNDKiP	2067,41	1817,57	2221,30	2050,51	1966,21
RaNDKiP1	2059,16	1809,45	2227,82	2032,06	1969,29
RaNDKiP2	2350,64	2118,41	2466,03	2383,75	1947,35
HEFT	1930,39	1830,69	1883,68	1676,30	1673,08
LDCP	2319,78	2123,25	2561,45	2427,54	2008,35

(b) Ukupni utrošak energije.

Slika 5.36: Parametri energije u sustavu s raznorodnom komunikacijom platforme za različite strukture programa iz repozitorija Pegasus.

Najmanje uštede ostvaruju se kod tipa programa *Inspiral*, jer kod njega postupci raspoređivanja ostvaruju najveću učinkovitost rasporeda (*fork-join* tip programa). Unaprijeđeni postupci raspoređivanja RaNDKiP i RaNDKiP1 pokazuju najmanju uštedu energije jer najbolje iskorištavaju raspored vremenskim isječcima. To nije slučaj samo kod programa *Montage*, gdje LDCP i RaNDKiP2 ostvaruju manju uštedu energije čineći učinkovitiji raspored.

Ako se promotri ukupni utrošak energije, prikazan na slici 5.36b, najveći je kod primjene postupka LDCP, zatim slijede RaNDKiP2, pa RaNDKiP i RaNDKiP1 te na kraju HEFT. Sa stajališta potrošnje energije vidljivo je kako tip programa *Sipht* odgovara postupcima LDCP i RaNDKiP2, dok im *Montage* i *Inspiral* manje odgovaraju (tip programa *fork-join*). Za njih je opet bolji izbor upotreba postupaka RaNDKiP i RaNDKiP1.

Velike razlike u utrošku i uštedi energije među različitim tipovima programa iz repozitorija Pegasus nema. No, one su ipak primjetne i daju uvid u to kojem tipu programa u primjeni odgovara koji postupak raspoređivanja, ako je drugi cilj osim skraćivanja rasporeda ušteda energije. Uštede energije su značajne i kreću se između 15 i 20%, čime se opravdava upotreba postupka LVI pri raspoređivanju.

5.5.4 Slučaj 8: energetske karakteristike programa generiranih alatom TGFF raspoređenih na platforme s raznorodnim komunikacijskim vezama

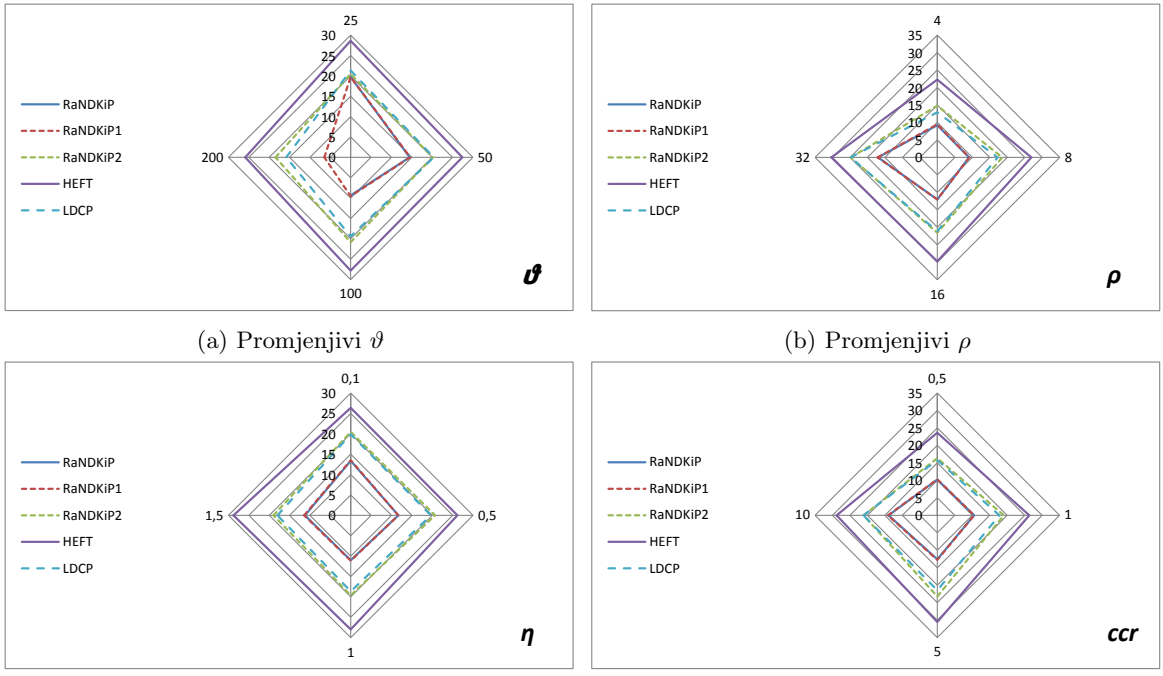
Ovim pododjeljkom dajemo još širi prikaz mogućnosti uštede energije korištenjem postupka LVI zajedno s postupcima raspoređivanja rangiranjem izvršnih jedinica programa. S obzirom na to da su slučajevi generiranja programa alatom TGFF zastupljeni u većem broju radova, što je već ranije spomenuto, iskoristit ćemo ih za prikaz energetskog aspekta raspoređivanja.

Spektar testiranja započinjemo prikazom ušteđene energije variranjem četiri glavna parametra koja smo i dosad koristili (ϑ, ρ, η i ccr), tako da možemo raditi izravnu usporedbu s prethodnim slučajevima.

Navedena situacija prikazana je slikom 5.37. Povećanjem broja izvršnih jedinica programa smanjuje se i ušteda energije (slika 5.37a), no mnogo manje nego što je to bilo kod slučajeva generiranih alatom DAGGer. Razlog tome je mnogo manja kompleksnost strukture programa generiranih alatom TGFF. Pad uštede energije najviše je vidljiv kod postupaka RaNDKiP i RaNDKiP1, što opet pokazuje bolju sposobnost navedenih postupaka da učinkovitije popunjavaju raspored. Zanimljivo je također i da se

postupak RaNDKiP2 bolje prilagođava navedenim programima nego LDCP, zbog većeg paralelizma unutar ovih programa. Nadalje, na slici 5.37b vidi se kako porastom broja obradbenih jedinica sklopovlja raste i ušteda energije. Ovdje svi postupci raspoređivanja osim RaNDKiP i RaNDKiP1 postižu točku zasićenja na 16 obradbenih jedinica, gdje se ušteda energije značajnije povećava (posebice nakon 16 obradbenih jedinica sklopovlja). Daljim porastom broja obradbenih jedinica za očekivati je da bi ta ušteda manje rasla.

Kao i dosad, za različite vrijednosti čimbenika raznorodnosti ušteda energije ostaje ista i to zbog istih razloga kao i ranije, slika 5.37c. Kod povećanja CCR-a, slika 5.37d, opet dolazi i do veće uštede, ali samo do $\eta = 5$, kada ušteda opet počinje padati. Ovom je opet uzrok manje nasumična struktura programa generiranih TGFF-om te se vidi kako svi postupci raspoređivanja imaju problema s popunjavanjem rasporeda do određene točke, kada ionako veliku većinu rasporeda počinje činiti komunikacija. Samim time i inicijalne vrijednosti energije početno su puno više (nije toliko važno trajanje obradbenih jedinica, već njihovo smještanje da bi se uštedilo na komunikaciji), tako da se postupkom LVI ne može ostvariti veća ušteda energije. Uočljivo je da kod postupaka RaNDKiP i RaNDKiP1 to nije slučaj te se ušteda nastavlja povećavati sve do $ccr = 10$, no za očekivati je da bi i navedeni postupci dosegli točku zasićenja kada bi se CCR dodatno povećao.



(a) Promjenjivi ϑ

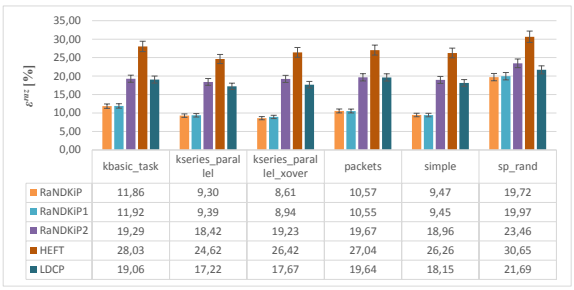
(b) Promjenjivi ρ

(c) Promjenjivi η

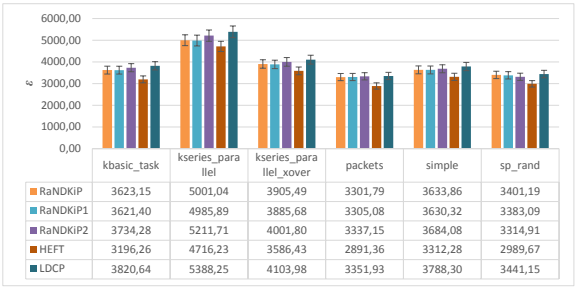
(d) Promjenjivi ccr

Slika 5.37: Ušteda energije kod testnih slučajeva generiranih alatom TGFF, platforma s raznorodnom komunikacijom.

Ako promotrimo sada ovisnost utroška i uštede energije o strukturi programa generiranih alatom TGFF (slika 5.38) primjećujemo značajno veće razlike kod različitih struktura programa.



(a) Ušteda energije.



(b) Ukupni utrošak energije.

Slika 5.38: Parametri energije u sustavu s raznorodnom komunikacijom platforme za različite strukture programa generiranih alatom TGFF.

Naime, ako se promotri ušteda energije na slici 5.38a, vidljivo je kako su za strukturu *sp_rand* uštede energije podjednake (HEFT se ovdje ističe zbog toga što ne daje pravu sliku komunikacije) i relativno visoke (programi iz repozitorija Pegasus nemaju razmjere programa od $\vartheta \approx 200$). Kao i kod vremenske analize te u sprezi s istom, dolazimo do zaključka da kod nestrukturiranih opterećenja poput *sp_rand* i slučajeva generiranih alatom DAGGer padaju performanse raspoređivanja, što je i za očekivati. No te se razlike najviše odnose na unaprijeđene postupke raspoređivanja RaNDKiP i RaNDKiP1. Oni iako daju bolje rezultate u svim dosad testiranim oblicima opterećenja, njihove performanse se izjednačavaju kod nestrukturiranih, visoko-raznorodnih programa s ostalim postupcima, gdje upotreba heuristika ionako ne može dati univerzalno dobre rezultate. Najmanja ušteda energije ostvaruje se kod struktura *kseries_parallel*, *kseries_parallel_xover* i *simple*, jer je kod njih ujedno i najveća strukturiranost te stupanj paralelizma.

Ukupni utrošak energije prikazan je slikom 5.38b. Zanimljivo je kako je on najveći upravo kod strukture *kseries_parallel*, a najmanji kod *packets* i *sp_rand*. Opet, stupanj paralelizma znači veći utrošak energije zbog toga što svi postupci pokušavaju postići što veću razinu paralelne obrade, tako da će smjestiti dosta izvršnih jedinica na obradbene jedinice koje troše više energije (na kojima je vrijeme izvođenja veće). Kod ostalih to nije slučaj. Dalje, usporedbom struktura *kseries_parallel* i *kseries_parallel_xover* vidljivo je da povećanjem broja bridova grafa, (pojave komunikacije među izvršnim jedinicama programa) utrošak energije postaje manji, jer je time uzrokovan i manji stupanj paralelizacije.

Navedena analiza pokazuje i dalje prisutnu mogućnost uštede energije za programe generirane alatom TGFF. Ono što je specifično za te programe je njihova bolja strukturiranost, posebice *kseries_parallel*, *kseries_parallel_xover* i *simple*, kod kojih unaprijeđeni postupci raspoređivanja RaNDKiP i RaNDKiP1 ostvaruju manju uštedu, no indiciraju bolje popunjen raspored. S druge strane, visoko raznorodne strukture poput *sp_rand* uzrokuju slične vrijednosti uštođene energije među različitim postupcima.

Ukratko, strukturirani programi s visokim stupnjem paralelizma uzrokuju ukupno veći utrošak energije, jer u tom slučaju postupci žrtvuju odabir jedinica sklopovlja s većom obradbenom moći za postizanje paralelizma pod svaku cijenu, posebno kod većeg broja obradbenih jedinica.

Primjenjivost postupka LVI opravdana je u svim dosad analiziranim slučajevima i uz prosječnu uštedu od približno 20% ukazuje na to da je iskorištavanjem praznina u rasporedu moguće ostvariti i neke dobitke i to u jednom od najvažnijih pokazateljem kvalitete današnjice — potrošnjom energije.

Spomenutim smo pokazali da upotrebom unaprijeđenog postupka RaNDKiP, osim što postižemo značajno kraći raspored nego s uspoređenim postupcima, omogućujemo i uštedu energije upotrebom jednog od njegovih koraka — postupka LVI. Navedeni smo postupak kombinirali i sa drugim postupcima raspoređivanja, za koje on također omogućuje uštedu energije, čak još i veću, zbog toga što navedeni postupci raspoređivanja daju manje optimalan raspored. Time smo i opravdali RaNDKiP kao naš izvorni znanstveni doprinos D.3., uz neke od pretpostavki koje čine i doprinos D.2.

U sljedećem pododjeljku analizirat ćemo naš unaprijeđeni postupak raspoređivanja RaPPaMaG, ali u nešto drugačijem okruženju zbog same njegove prirode.

5.6 Rezultati analize postupka RaPPaMaG

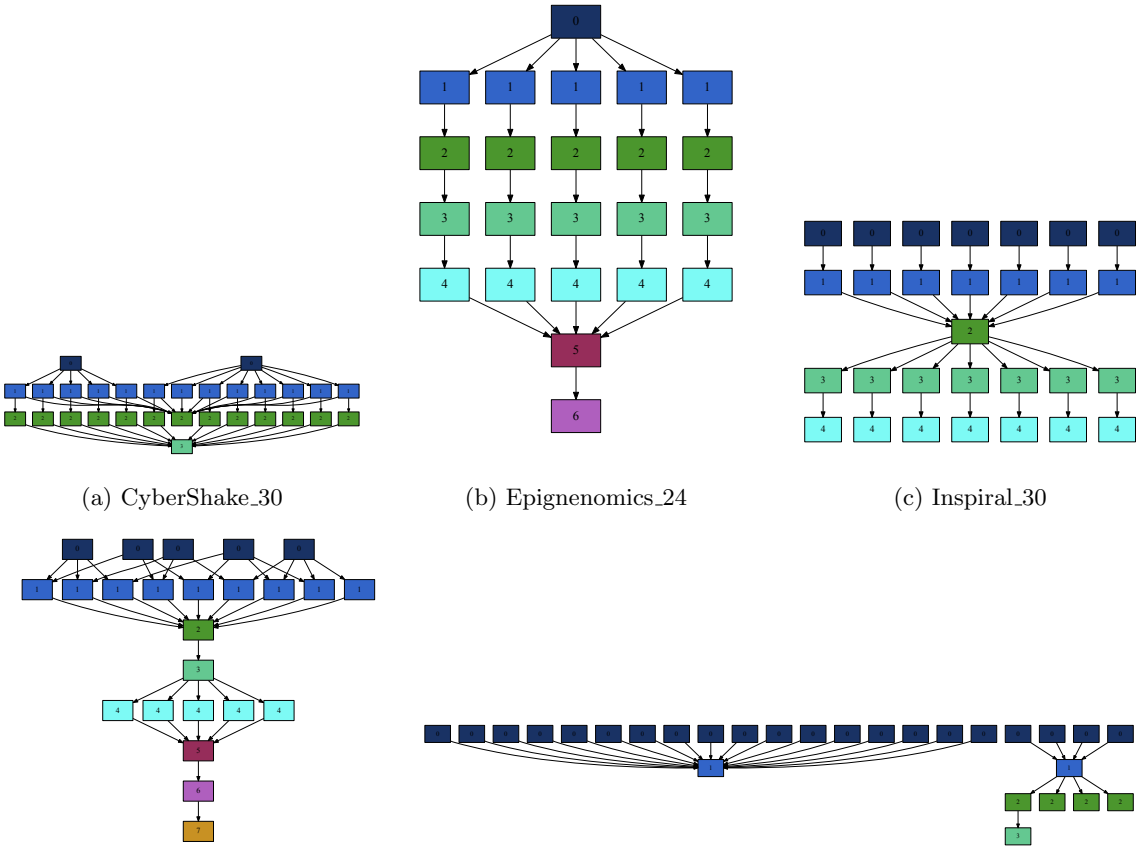
Postupak RaPPaMaG, kao što je opisano u pododjeljku 4.1, je postupak raspoređivanja programa zapisanih u novom obliku miješanoga acikličkoga grafa kakvih, prema našim saznanjima, nema u postojećoj literaturi. Također, naveli smo kako miješani graf služi opisu programa koji sadrže sinkronizaciju, bilo komunikacijsku (komunikacijske rutine iz MPI-ja i OpenMP-ja), bilo onom bez komunikacije, (sinkronizacija na barijeri, MPI, OpenMP), kakvih je mnogo u primjeni, prema [49, 50].

Navedeni postupak moguće je također primijeniti i na DAG i na TIG oblike programa. Primjena postupka RaPPaMaG prvenstveno je namijenjena situaciji kada postoji više kriterija po kojima se vrši raspoređivanje programa na sklopovlje (kriterija optimizacije, ciljeva) i kojima su dane eksplicitne važnosti u obliku korisnički zadanih parametara (težina). Višekriterijsko raspoređivanje većinom se rješava raznim metaheurističkim postupcima koji pretražuju skup rješenja promatrajući program u cjelini, što je i detaljnije opisano u pododjeljku 2.3.2. Navedeno uzrokuje veliku ograničenost pokrivanja tog prostora rješenja, tako da ona mogu ostati u lokalnom optimumu, (vrijednosti koja je blizu optimalne, ali samo na određenom dijelu prostora pretraživanja). Zato naš postupak RaPPaMaG sadrži fazu particioniranja grafa programa na manje dijelove — faze izvršavanja, čime drastično smanjuje prostor pretraživanja omogućujući optimalna, (u kombinaciji s metodama koje pretražuju čitav skup rješenja, poput iscrpnog pretraživanja) ili gotovo optimalna rješenja, (u kombinaciji s metaheurističkim metodama vođenog lokalnog pretraživanja) za pojedine faze izvršavanja. Particioniranje ima i nekoliko mana, a to je kada su sustavi visoko paralelni (nekoliko neovisnih nizova paralelnih jedinica programa) s međusobno vrlo raznorodnim vremenima izvođenja. Najsličnija struktura koja pokazuje takav tip programa je *kseries_parallel* iz repozitorija TGFF (potpoglavlje 5.1.1). No unatoč tome, primjena particioniranja omogućuje podjelu i takvih programa na neovisne faze izvršavanja, omogućujući primjenu metaheurističkih metoda na manjem prostoru pretraživanja. To ćemo pokazati i u narednim pododjeljcima.

5.6.1 Slučaj 9: primjena postupka RaPPaMaG na programe iz repozitorija Pegasus

U ovom pododjeljku analizirat ćemo programe iz repozitorija znanstvenih radnih opterećenja Pegasus pomoću postupka RaPPaMaG. Na dosad predstavljene primjerke navedenih programa primijenit ćemo fazu 1 postupka RaPPaMaG prema algoritmu 1, opisanu u pododjeljku 4.1.1.

Na slici 5.39 prikazane su faze izvršavanja programa malih razmjera, (24–30 jedinica programa) iz repozitorija Pegasus dobivene postupkom RaPPaMaG. Faze su obojane različitim bojama kako bi ih se moglo razlikovati. Od navedenih struktura problem za raspoređivanje faza izvršavanja može činiti samo *Sipht_30*, prikazan slikom 5.39e. Naime, to je opterećenje s visokim stupnjem paralelizma, tako da prva faza izvršavanja ima velik broj članova u odnosu na čitav graf. Samim time, kombinacija kojih takva faza može imati pri raspoređivanju je mnogo. No, za očekivati je da tu fazu u primjeni čine približno homogene jedinice programa (po pitanju opterećenja), što može olakšati analizu.

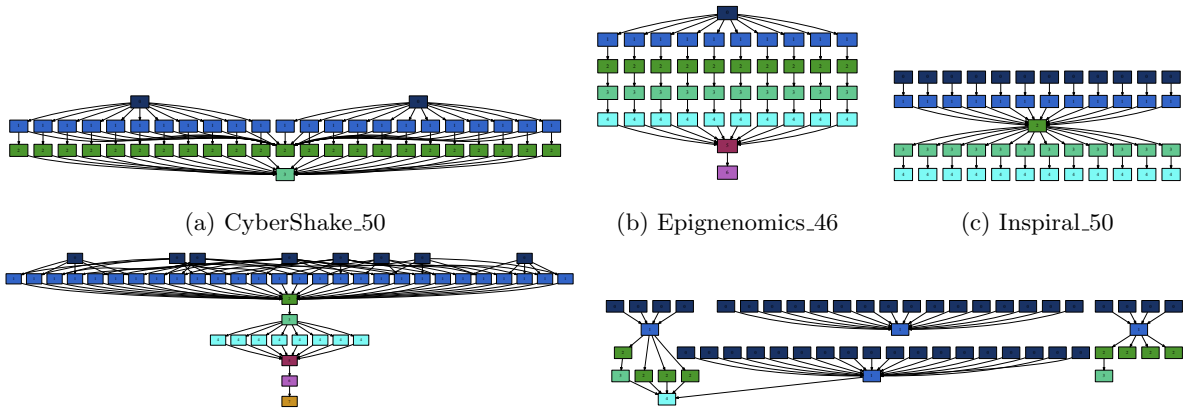


(a) CyberShake_30 (b) Epignenomics_24 (c) Inspiral_30

(d) Montage_25 (e) Sipht_30

Slika 5.39: Programi malih razmjera iz repozitorija Pegasus particionirani postupkom RaPPaMaG.

Usporedbe radi, na slici 5.40 prikazani su programi srednjih razmjera iz repozitorija programa Pegasus podijeljeni na faze izvršavanja postupkom RaPPaMaG. Vidljivo je kako je za tipove programa iz navedenog repozitorija bolje zasebno analizirati dijelove visoke razine paralelizma, nego promatrati svaku jedinicu programa zasebno kao entitet dodjeljivanja, (što čini većina današnjih postupaka raspoređivanja). Kada se programi analiziraju prema fazama izvršavanja programa mnogo je lakše iskoristiti njihovu strukturu za odabir pretpostavki pri raspoređivanju. Primjerice, kod velikih faza raspoređivanja poput faze 2 i 3 u programu *CyberShake_30* prikazanom na slici 5.40a ili faze 2 programa *Montage_25* sa slike 5.40d, zgodno bi bilo upotrijebiti homogen skup brzih obradbenih jedinica (ako su prioritet vremenski pokazatelji kvalitete).



(a) CyberShake_50 (b) Epignenomics_46 (c) Inspiral_50

(d) Montage_50 (e) Sipht_60

Slika 5.40: Programi srednjih razmjera iz repozitorija Pegasus particionirani postupkom RaPPaMaG.

Zajednička karakteristika analize programa iz repozitorija Pegasus postupkom particioniranja RaPPaMaG jest da je pomoću njihove strukture lako izdvojiti njihove glavne karakteristike, koje mogu usmjeriti postupak raspoređivanja. Primjerice, većina programa iz navedenog repozitorija ima relativno mali broj faza izvršavanja, ali s visokom razinom paralelizma. Primjena drugog koraka raspoređivanja u obliku iscrpnog pretraživanja nailazi na probleme kod velikih faza izvršavanja, no uz uzimanje u obzir navedenih karakteristika taj se korak može dodatno rafinirati.

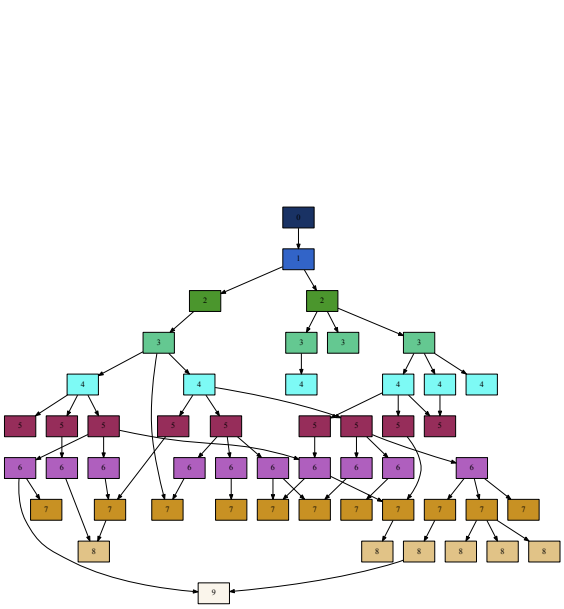
Kako bi opravdali upotrebu particioniranja kao faze unaprijeđenog postupka RaPPaMaG, u idućem pododjeljku proširit ćemo analizu i na programe generirane alatom TGFF.

5.6.2 Slučaj 10: primjena postupka RaPPaMaG na programe iz repozitorija TGFF

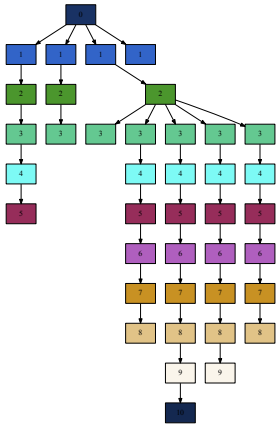
U ovom pododjeljku primjenjujemo unaprijeđeni postupak RaPPaMaG na programe generirane alatom TGFF koristeći strukture iz repozitorija TGFF opisane u pododjeljku 5.1.1. Programi srednjih razmjera navedenih karakteristika particioniranih postupkom RaPPaMaG prikazani su slikom 5.41 (male razmjere ne prikazujemo zbog prejednostavne strukture).

Primjena prvog koraka postupka RaPPaMaG (particioniranje) posebno je pogodna za uže, stepenaste grafove, poput *kseries_parallel* prikazanog na slici 5.41b i *kseries_parallel_xover* prikazanog na slici 5.41c, jer su faze izvršavanja manje i obično su u takvim programima jedinice programa u fazama podjednake (primjerice, operacije nad gustim matricama iz biblioteke SCALAPACK, prema [111]). Također, manje su faze izvršavanja u strukturama *packets* i *simple*, prikazanih slikama 5.41d i 5.41e, no zbog njihove raznorodne prirode njihovo raspoređivanje može biti vremenski zahtjevno. Konačno, u slučaju gotovo neovisnih jedinica programa, poput strukture *sp_rand*, particioniranje ne igra veliku ulogu te se u takvim slučajevima mogu koristiti specifični postupci raspoređivanja, poput serije pohlepnih algoritama kao u [34, 48, 58], algoritma Min-min poput primjera u [11, 35] i drugih.

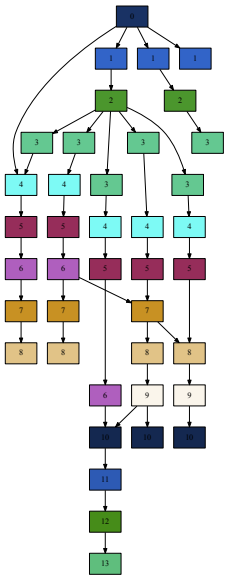
Primijenili smo također postupak particioniranja RaPPaMaG i na veće grafove iz TGFF-a ($\vartheta \approx 200$). Njihova podjela na faze izvršavanja prikazana je slikom 5.42.



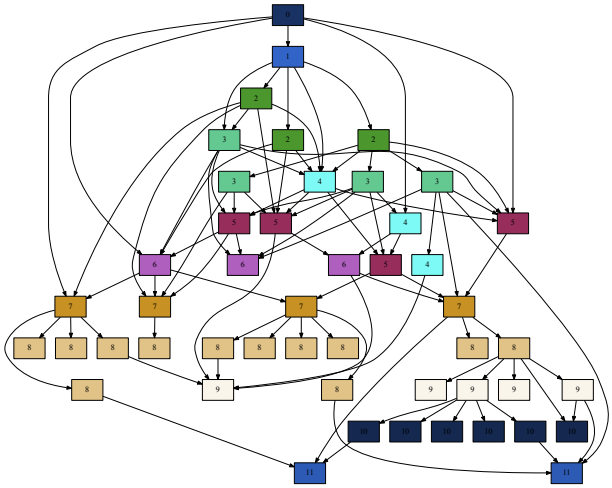
(a) kbasic_task_50



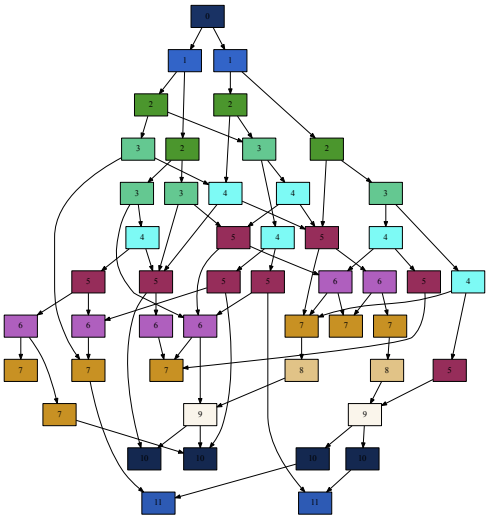
(b) kseries_parallel_50



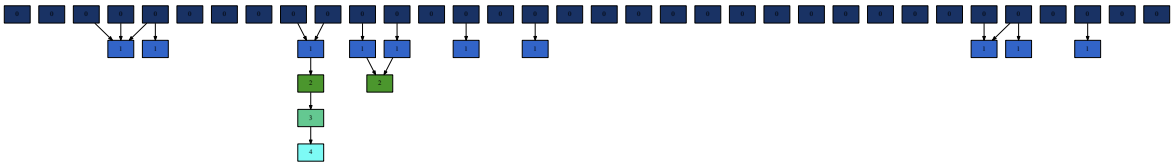
(c) kseries_parallel_xover _50



(d) packets_50



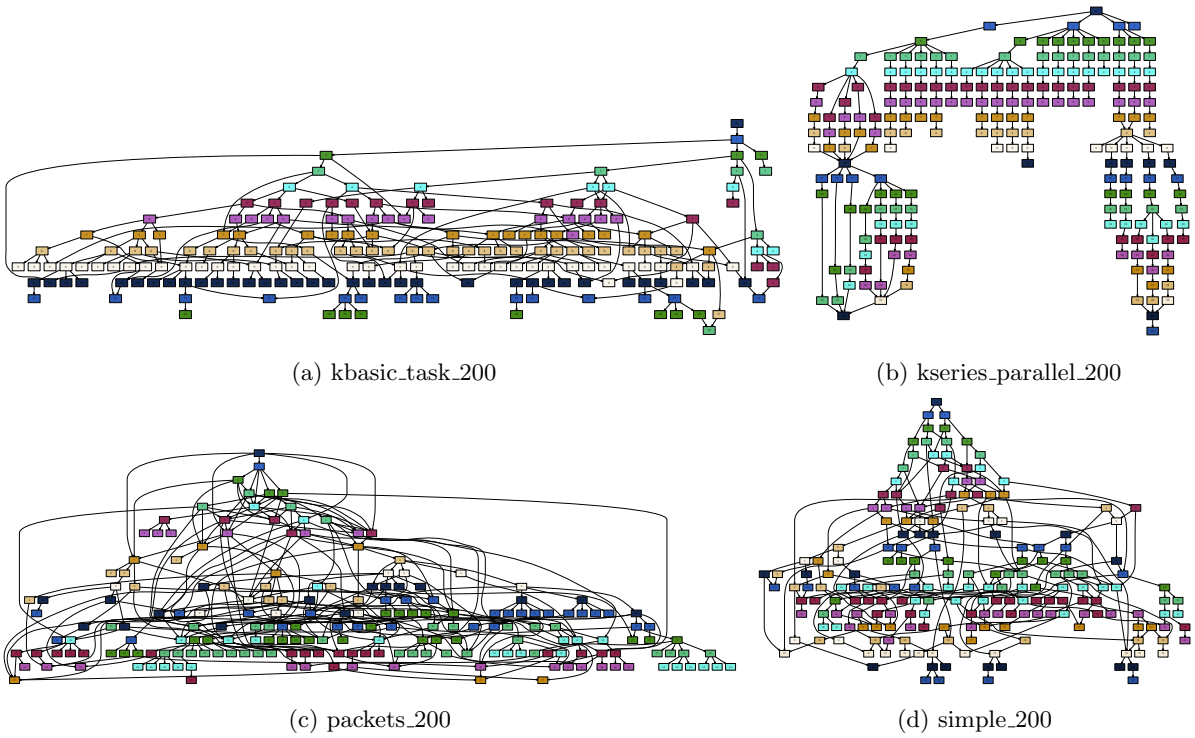
(e) simple_50



(f) sp_rand_50

Slika 5.41: Programi srednjih razmjera iz repozitorija TGFF particionirani postupkom RaPPaMaG.

Prikazane su karakteristike samo za strukture *kbasic_task* (slika 5.42a), *kseries_parallel* (slika 5.42b), *packets* (slika 5.42c) i *simple* (slika 5.42d), s obzirom na to da su ostale strukture slične i analizirane ranije. Mogu se uočiti pojedine faze izvršavanja, koje su sad mnogo veće i njihov optimalan raspored nije moguće naći pretraživanjem čitavog prostora rješenja u razumnom vremenu. No, kod povećanja faza njihova



(a) kbasic_task_200

(b) kseries_parallel_200

(c) packets_200

(d) simple_200

Slika 5.42: Programi vrlo velikih razmjera iz repozitorija TGFF partic. postupkom RaPPaMaG.

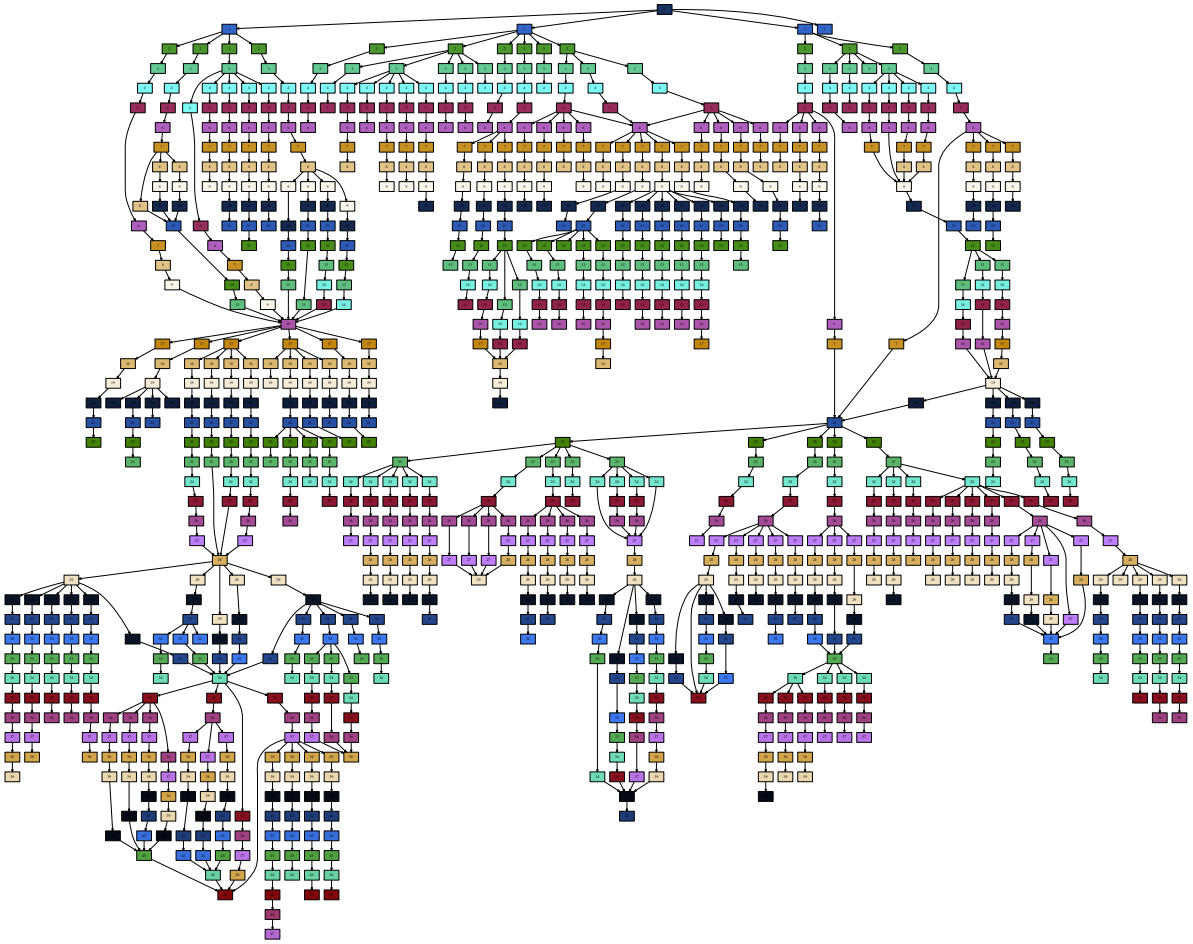
raznorodnost više ne predstavlja toliko značajnu ulogu te je moguće uvesti određena pojednostavljenja za njihovo raspoređivanje (rasporediti ih bilo kojim redoslijedom).

Vrijeme trajanja faze particioniranja je gotovo zanemarivo, ako se uzme u obzir da za graf od 1000 jedinica programa (prikazan slikom 5.43) ono traje 15ms na jednoj procesorskoj jezgri stolnog računala s procesorom AMD Phenom II radnog takta jezgre od 3,3GHz.

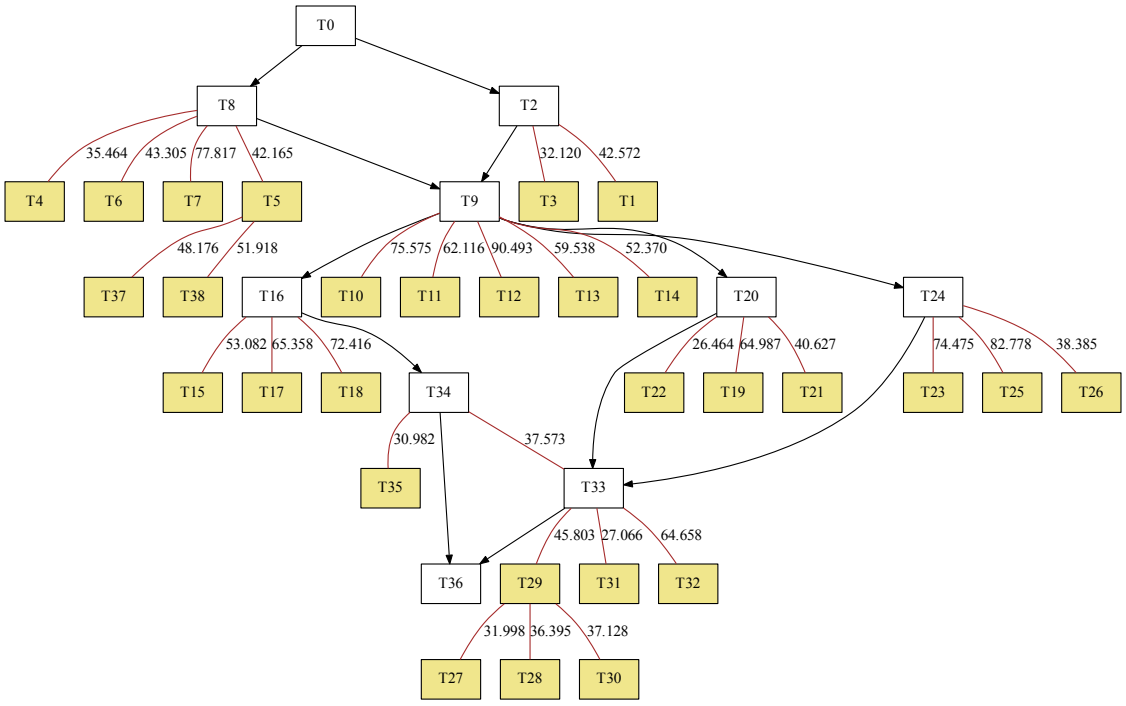
Prethodnim primjerima dana je analiza prve faze našeg unaprijeđenog postupka RaPPaMaG, a to je particioniranje programa u obliku grafa. Dosadašnji prikaz programa bio je u modelu DAG-a, za koje postupak RaPPaMaG također može izdvojiti neovisne faze izvršavanja (što smo i pokazali eksperimentalnom analizom). No, navedeni je postupak primarno namijenjen upotrebi za programe predstavljene miješanim acikličkim grafom, koji opisuje i vremenski slijed i sinkronu komunikaciju među jedinicama programa. Zbog toga, jedan takav primjer predstavljamo i analiziramo u narednom pododjeljku.

5.6.3 Slučaj 11: primjena postupka RaPPaMaG na program sa sinkronom komunikacijom na miješani aciklički graf

Već u opisu postupka RaPPaMaG u pododjeljku 4.1 spomenuto je kako je njegova glavna namjena za programe koji sadrže sinkronu komunikaciju (koja zahtijeva istovremeno sudjelovanje svih komunicirajućih jedinica programa), poput onih koji izmjenjuju poruke u MPI okruženju. Primjer jednog takvog programa dan je miješanim grafom prema modelu programa programskog okruženja LOSECO na slici 5.44. Navedeni program sadrži 39 izvršnih jedinica `T0`–`T38`, koje ćemo zbog ujednačenosti nomenklature referirati kao $v_0, ..., v_{38}$. Nadalje, struktura primjera programa je u obliku tzv. grupno sinkronih programa (eng. *bulk-synchronous programs*), sličnih onima koje opisuju autori u [28].



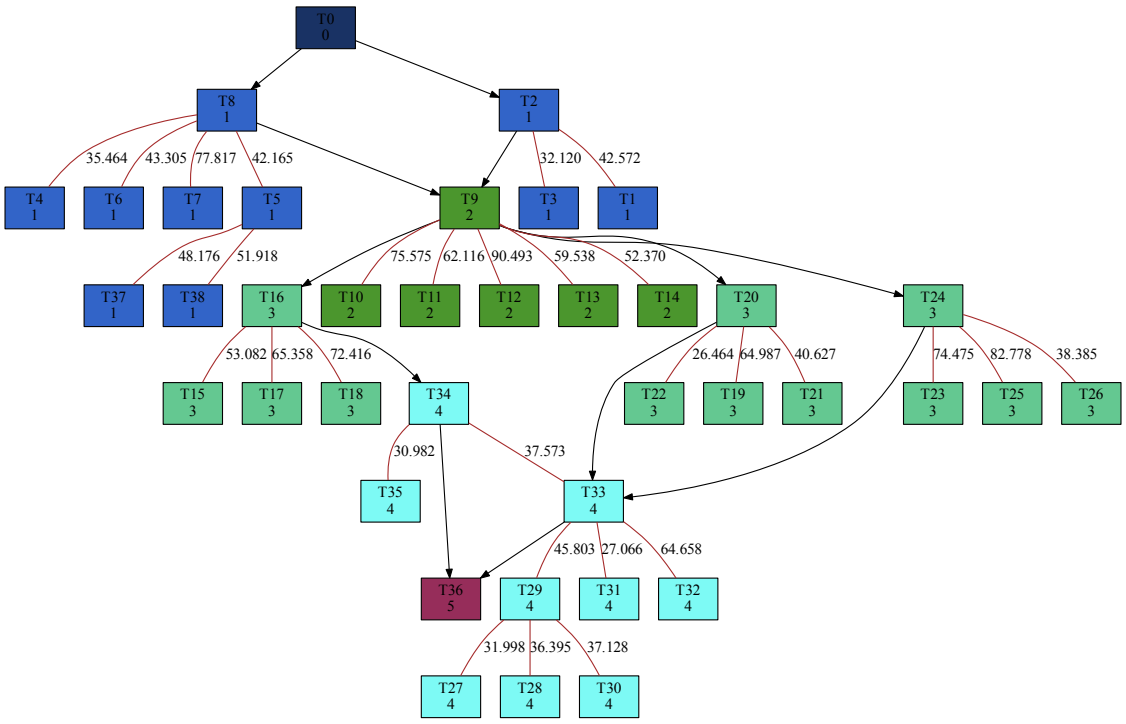
Slika 5.43: Program veličine $\vartheta = 1000$ particioniran postupkom RaPPaMaG (kseries_parallel_1000).



Slika 5.44: Primjer grupno sinkronog programa u obliku miješanoga grafa.

To znači da dio jedinica programa (obično manji broj njih), čini glavni vremenski slijed (dispečeri, eng. *master, dispatcher*), a dio čini entitete radnike (eng. *worker, slave*). U analiziranom primjeru, sa slike 5.44, dispečeri su jedinice programa označene bijelom bojom ($v_0, v_2, v_8, v_9, v_{16}, v_{20}, v_{24}, v_{33}, v_{34}, v_{36}$), dok su sve ostale jedinice programa radnici (označeni oker bojom). Struktura navedenog programa slična je i programima iz repozitorija Pegasus (pododjeljak 5.6.1), no uz veću raznorodnost i sinkronu komunikaciju. Crveni bridovi, kao i broj koji sadržavaju, su neusmjereni bridovi koji predstavljaju sinkronu komunikaciju i njezin obujam, dok crni usmjereni bridovi (lukovi), predstavljaju vremenski slijed izvođenja.

Primjenom prvog koraka postupka RaPPaMaG (particioniranje), dobivene su faze izvršavanja prikazane slikom 5.45, gdje svaka boja označava jednu fazu (kao što je to bilo kod primjera particioniranja programa iz repozitorija Pegasus, pododjeljak 5.6.1 i programa iz repozitorija TGFF, pododjeljak 5.6.2). Vidljivo je kako većinom jedinice programa dispečeri diktiraju faze (po jedan u različitim fazama).



Slika 5.45: Program sa slike 5.44 podijeljen na faze izvršavanja postupkom RaPPaMaG.

No, postoje slučajevi kada su faze manje i kada, ukoliko ima dovoljno obradbenih jedinica, nekoliko grupno sinkronih dijelova može biti stavljeno u istu fazu i biti izvršeno paralelno. To je slučaj za faze 1, 3 i 4, gdje u fazi 1 jedinice programa v_2 i v_8 čine dispečere koji se nalaze u istoj fazi, za fazu 3 su to v_{16}, v_{20} i v_{24}, dok je faza 4 specijalni slučaj. Naime, u fazi 4 postoji sinkrona komunikacija među dispečerskim jedinicama programa (v_{33} i v_{34}), koja zahtijeva njihovo paralelno izvršavanje.

Postavljeni parametri su prikazani tablicom 5.5. Parametri su podijeljeni na one koji opisuju program, one koje opisuju sklopovlje i važnosti pojedinih pokazatelja kvalitete. S obzirom na to da prikazom raspoređivanja programa postupkom RaPPaMaG nad miješanim grafom ujedno i potvrđujemo njegovu

primjenjivost, nećemo ispitivati druge postavke parametara.

Tablica 5.5: Postavke važnijih parametara za slučaj 11.

		Vrijednosti ulaznih parametara
Programi	ϑ	39
	ccr	$1,0$
	$\overline{x_G^{ob}}$	50
	$\overline{x_i^{ob}}$	$\overline{x_G^{ob}}$
	x_i^{ob}	$x_i^{ob} = \overline{x_i^{ob}} = \overline{x_G^{ob}}$
	$\overline{e_G}$	$ccr \cdot \overline{x_G^{ob}}$
	$\overline{e}$	$[0,\ 2 \cdot \overline{e_G}]$
	$e_{i,j}$	prema (5.5)
Sklopovlje	ρ	12
	η	$1,5$
	c_{max}	5
	$c_{i,j}$	$[1,\ c_{max}]$
	$\overline{y_P^{ob}}$	$[1,\ 100]$
	y_j^{ob}	$\overline{y_P^{ob}} \cdot \left(1 - \frac{\eta}{2}\right) \leq y_j^{ob} \leq \overline{y_P^{ob}} \cdot \left(1 + \frac{\eta}{2}\right)$
	$\overline{y_P^{en}}$	4,5, 50
	y_j^{en}	$[3,\ 6]\,,\ [1,\ 100]$
Važn.	$w(t^{uk})$	$\{0,8,\ 0,2\}$
	$w(e^{uk})$	$\{0,2,\ 0,8\}$

Kao što je bilo riječi o načinu postavljanja parametara za postupak RaPPaMaG u pododjeljku 4.1.4, parametri se zadaju apsolutno za programe i za sklopovlje. Zbog toga sada čimbenik raznorodnosti, primjerice, utječe samo na parametre sklopovlja. Zbog toga, a i kako bi zadržali konzistentnost postavki s testnim slučajevima za postupak RaNDKiP, fiksiramo obradbene zahtjeve jedinica programa ($\overline{x_i^{ob}} = \overline{x_G^{ob}}$) i mijenjamo samo obradbene sposobnosti sklopovlja, prema:

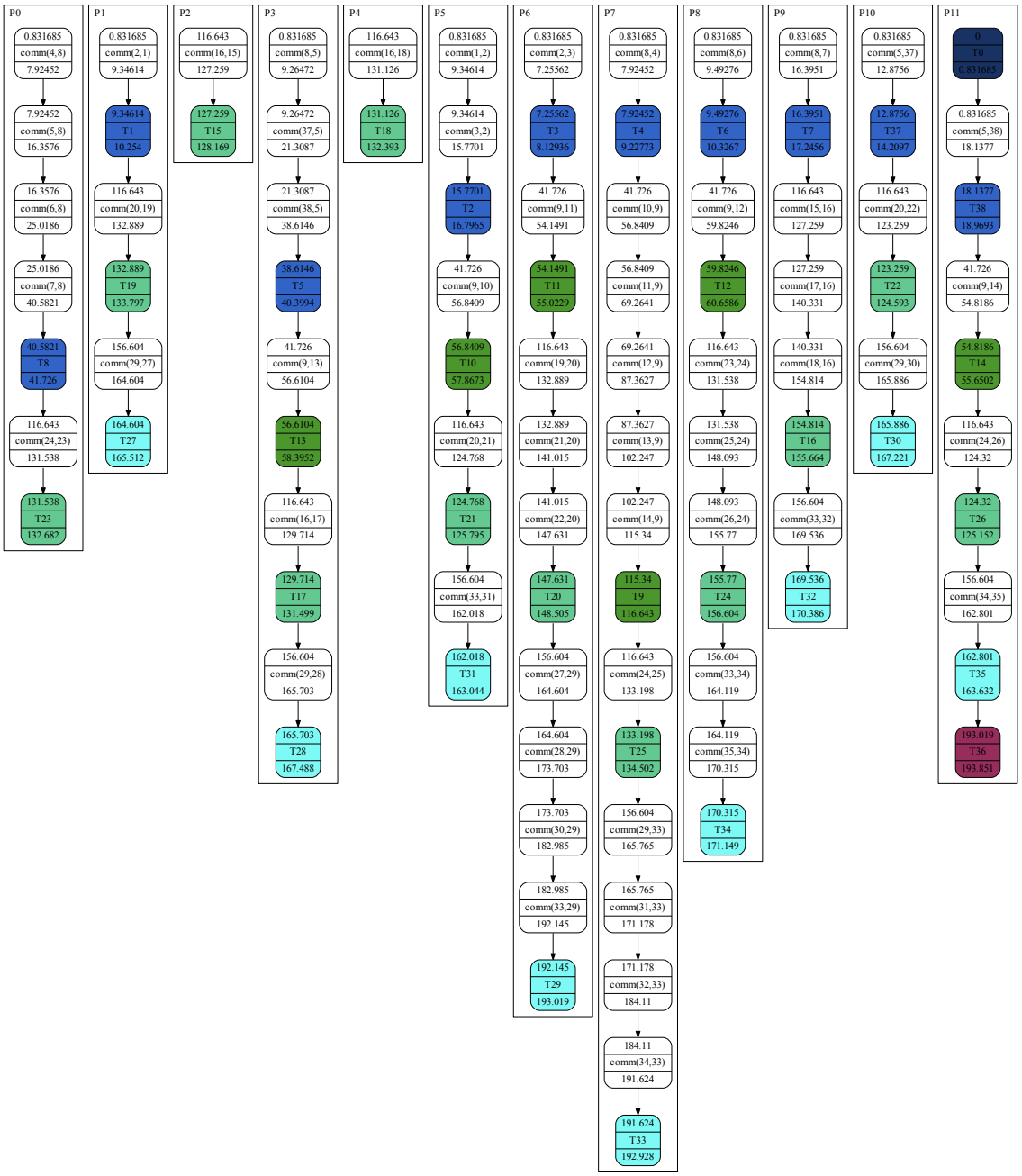
$$\overline{y_P^{ob}} \cdot \left(1 - \frac{\eta}{2}\right) \leq y_j^{ob} \leq \overline{y_P^{ob}} \cdot \left(1 + \frac{\eta}{2}\right) \tag{5.11}$$

Na taj način čimbenik raznorodnosti ima isti utjecaj kao i do sada (za postupak RaNDKiP).

Energetske karakteristike su također zadane na sličan način, uz raspon vrijednosti $y_j^{en} \in [3,\ 6]$, čime zadržavamo malu raznorodnost istih.

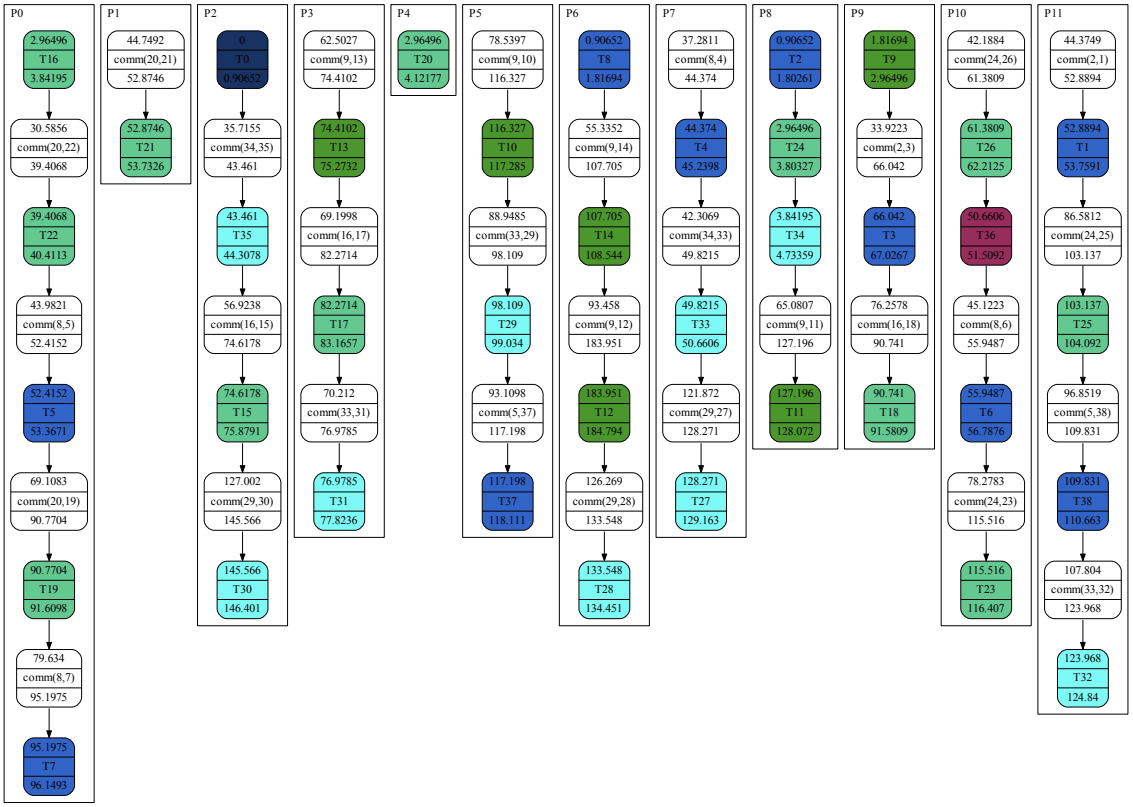
Naposljetku, važnosti pojedinih pokazatelja kvalitete zadržavamo istima kao i u primjeru iz pododjeljka 4.1.3, ukratko, omjer važnosti $0,8 : 0,2$ u korist vremena izvršavanja. Zbirne rezultate prikazat ćemo i za druge važnosti.

U slučaju da se primijeni postupak raspoređivanja dobivenih faza izvršavanja sa slike 5.45 i parametara iz tablice 5.5 u obliku iscrpnog pretraživanja (prema pododjeljku 4.1.3), dobiva se raspored sa slike 5.46. Prema modelu raspoređivanja programskog okvira LOSECO, kao i pretpostavkama postupka raspoređivanja RaPPaMaG (njegove druge faze), dobiveni je raspored optimalan po fazama izvršavanja. Vidljivo je također kako postupak ujednačeno slaže jedinice programa prema fazama uz stavljanje odgovarajućih komunikacijskih blokova prije izvršavanja obradbenih dijelova programa, čime ispunjava zahtjeve za sinkronizacijom komunikacije.



Slika 5.46: Raspored dobiven iscrpnim pretraživanjem.

Raspoređivanje pojedinih faza pokušali smo dobiti izmjenom postupka HEFT, na način da postavljamo $c_{i,j} = \infty,\ i = j$ i zadavanjem drugih uvjeta koji moraju biti ispunjeni kako bi pojedina jedinica programa bila označena kao spremna za izvršavanje (prema fazama izvršavanja). No, navedeni postupak i dalje ne daje raspored prema svim traženim postavkama i ograničenjima, s obzirom na to da ih ne može u potpunosti uzeti u obzir, što prikazuje i raspored na slici 5.47. Iako, na prvi pogled se čini da daje manju duljinu rasporeda. Naime, vidljivo je kako jedinice programa nisu sinkronizirane po izvođenju. Primjerice, u istoj fazi izvršavanja jedinica programa `T8` se izvršava u vremenskom intervalu $0,90652$–$1,81694$, dok se jedinica programa `T37` izvršava u intervalu 117,198–118,111.



Slika 5.47: Raspored dobiven postupkom HEFT.

Isto tako, pojedine jedinice programa iz narednih faza izvršavaju se prije nego prethodna završava, primjerice izlazna jedinica `T36` prije `T34` itd.

Zbirni rezultati dobiveni korakom 2 postupka RaPPaMaG (raspoređivanje faza izvršavanja) dani su tablicom 5.6. U navedenoj tablici vidljivo je kako je drastičnim povećanjem važnosti energije s 0,2 na 0,8 dobivena ušteda od samo 0,37% uz povećanje vremena izvršavanja od 8,12%.

Tablica 5.6: Različite postavke važnosti pokazatelja kvalitete i dobiveni rezultati (mala razina raznorodnosti parametra y_j^{en}).

	t^{uk}	e^{uk}
$w(t^{uk}) = 0,8,\ w(e^{uk}) = 0,2$	193,85	8681,67
$w(t^{uk}) = 0,2,\ w(e^{uk}) = 0,8$	209,59	8649,18

No, to nije znak kako postupak ne uzima važnosti postupka u obzir, nego razlog tome leži u maloj raznorodnosti parametara utroška energije ($y_j^{en} \in [3,\ 6]$), prema tablici 5.5, tako da izmjenom rasporeda ne dolazi do velike uštede energije, dok se u isto vrijeme duljina rasporeda može drastično promijeniti čime vrijednost ciljne funkcije drastično raste.

Iz tog razloga testirali smo iste postavke, ali uz $y_j^{en} \in [1,\ 100]$ odabranog nasumično uniformnom razdiobom. Spomenutim postavkama osiguravamo visoku raznorodnost karakteristike utroška energije među obradbenim jedinicama. Uz primjernu istog postupka dobivamo vrijednosti prikazane tablicom 5.7.

Tablica 5.7: Različite postavke važnosti pokazatelja kvalitete i dobiveni rezultati (velika razina raznorodnosti parametra y_j^{en}).

	t^{uk}	e^{uk}
$w(t^{uk}) = 0,8$, $w(e^{uk}) = 0,2$	187,12	70911,31
$w(t^{uk}) = 0,2$, $w(e^{uk}) = 0,8$	228,51	68597,88

Iz spomenute tablice uočavamo kako je povećanje duljine rasporeda čak 22,12%, dok je ušteda energije iznosi samo 3,26%. To se događa zbog toga što bi u slučaju izbjegavanja energetski neučinkovitih jedinica programa dobili slučajeve da nemamo dovoljno obradbenih jedinica sklopovlja za izvršavanje najveće faze izvršavanja δ_3 koja sadrži 12 jedinica programa koje treba izvršiti paralelno. Za očekivati je da bi većim brojem obradbenih jedinica situacija bila drugačija, uz veće uštede energije povećanjem njezine važnosti.

Eksperimentalnom analizom postupka RaPPaMaG dokazali smo kako ga je moguće primijeniti i na programe u obliku DAG-a i to njegove prve faze — particioniranja. Podjelom grafova na manje faze izvršavanja moguće je primijeniti raspoređivanje i analizu po manjim skupinama izvršnih jedinica programa, čime se potencijalno ubrzava postupak raspoređivanja. Spomenuto smo pokazali i na primjerima programa iz repozitorija Pegasus, ali i programa generiranih alatom TGFF. Također, kompleksnost i trajanje samog particioniranja neznatno je čak i za velike grafove ($\vartheta \approx 1000$).

S obzirom na to da je primarna uloga postupka RaPPaMaG primjena na programima koji sadržavaju sinkronu komunikaciju i vremenski slijed istovremeno, prikazali smo njegove obje faze i na primjeru grupno sinkronog programa prikazanog u obliku miješanoga acikličkoga grafa. Napomenutim primjerom dobili smo optimalna rješenja po fazama izvršavanja i pokazali kako postupak RaPPaMaG vrednuje različite važnosti pokazatelja kvalitete. Također, dali smo temelj za ugradnju novih postupaka raspoređivanja kao druge faze postupka RaPPaMaG , primjerice neke od postupaka raspoređivanja rangiranjem jedinica programa. No, takve je postupke u tom slučaju potrebno prilagoditi pretpostavkama koje podrazumijeva program predstavljen u obliku miješanoga grafa.

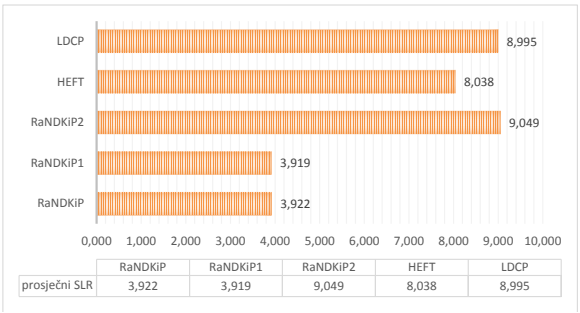
5.7 Skupna analiza eksperimentalnih rezultata

Ovim pododjeljkom dat ćemo skupni prikaz eksperimentalnih rezultata skupljenima i obrazloženima u prethodnim pododjeljcima. Navedenim ćemo dati uvid u razinu ostvarivanja izvornih znanstvenih doprinosa u smislu evaluacije unaprijeđenih postupaka raspoređivanja RaNDKiP i RaPPaMaG.

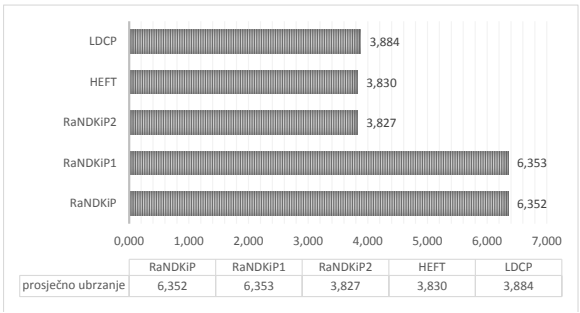
Prvim dijelom analize obuhvaćamo pokazatelje kvalitete u vremenskoj domeni upotrebom inačica postupka RaNDKiP za raspoređivanje programa na računalne sustave s raznorodnim komunikacijskim kanalima. U tu svrhu obuhvatili smo prosjek svih rezultata mjerenja za četiri glavna mjerena parametra: SLR, ubrzanje, učinkovitost i relativnu postotnu razliku duljine rasporeda. Prosjek se odnosi na sve analizirane programe, tj. one generirane alatom DAGGer i alatom TGFF te programe iz repozitorija Pegasus. Uz to treba imati na umu da se u navedenim rezultatima već nalaze prosjeci od 10 uzastopnih izvođenja. Ukupno je to 2624 ispitana slučajeva ($\times 10$), od toga se 256 odnosi na programe generirane alatom DAGGer, 1408 na programe generirane alatom TGFF i 960 na programe iz repozitorija Pegasus.

Slika 5.48 prikazuje skupne rezultate postupaka RaNDKiP u usporedbi s ostalima. Najveći prosječan

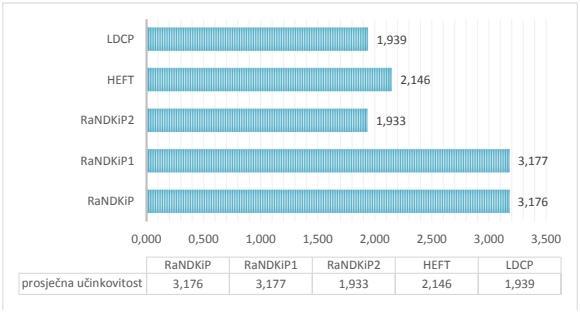
SLR postiže postupak RaNDKiP2, s gotovo identičnom vrijednosti kao i kod postupka LDCP, što se vidi na slici 5.48a. Pokazatelj je to da raznorodno rangiranje ne igra veliku ulogu pri izradi rasporeda. Drugim riječima, iako se takvim rangiranjem različitim redoslijedom biraju jedinice programa za raspoređivanje, razlike su minimalne. HEFT pokazuje nešto manje vrijednosti, no i dalje s dvostruko većom vrijednosti nego RaNDKiP i RaNDKiP1, koji ostvaruju prosječan SLR od 4. Međusobno se RaNDKiP i RaNDKiP1 gotovo i ne razlikuju, zbog toga što postupak raznorodnog rangiranja ne igra veliku ulogu u unaprjeđenju postupka, već je za to ponajviše zaslužno eksploatiranje raznorodnosti komunikacijskih veza sklopovlja pri fazi ažuriranja (nakon svakog koraka raspoređivanja). Kod ubrzanja, slika 5.48b, situacija je nešto drugačija. Naime, HEFT pokazuje gotovo iste vrijednosti kao LDCP i RaNDKiP2, što znači da ostvaruje istu razinu paralelizacije kao i oni. S druge strane, RaNDKiP i RaNDKiP1 i ovdje daju bolje rezultate (gotovo 65% bolje). Razina učinkovitosti je također specifična, što pokazuje slika 5.48c. U navedenom slučaju HEFT pokazuje veće vrijednosti učinkovitosti od LDCP i RaNDKiP2 i time ukazuje na bolju skalabilnost. No najbolju skalabilnost i dalje imaju postupci RaNDKiP i RaNDKiP1. Ukratko, slika 5.48d najbolje opisuje međusobne razlike u performansama pojedinih postupaka. Najbolje performanse pokazuju postupci RaNDKiP i RaNDKiP1 s međusobno vrlo malim razlikama, sljedeći je postupak raspoređivanja HEFT, zatim LDCP i RaNDKiP2.



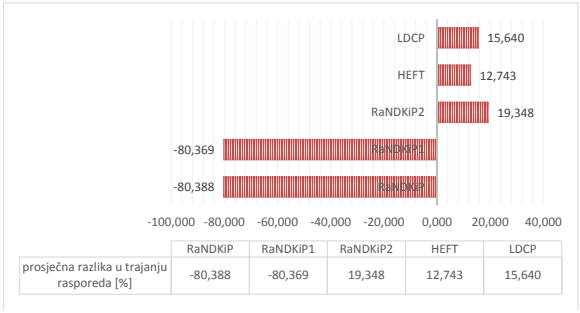
(a) SLR



(b) Ubrzanje



(c) Učinkovitost



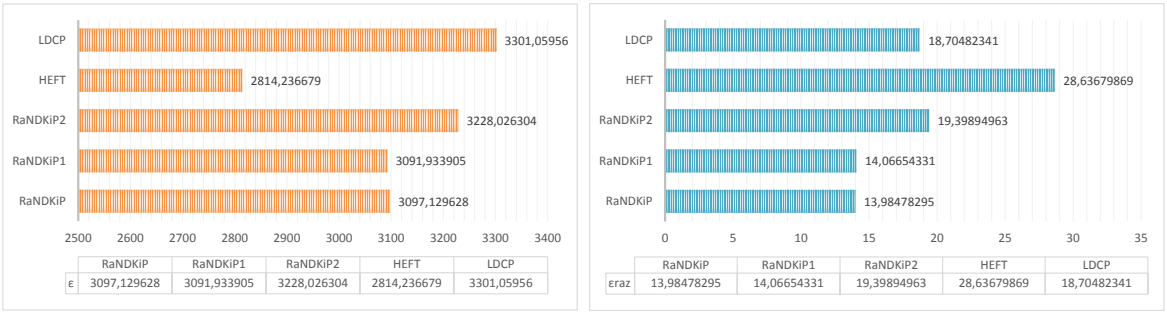
(d) Relativna razlika duljine rasporeda

Slika 5.48: Skupne vremenske performanse raspoređivanja programa na računalne platforme s raznorodnim komunikacijskim vezama.

Odnosi navedenih postupaka dokazuju kako unaprijeđeni postupci RaNDKiP i RaNDKiP1 pokazuju iznimnu primjenjivost u situacijama, gdje se raspoređivanje programa odvija na visoko raznorodne platforme, posebice pri velikim razlikama u propusnosti komunikacijskih kanala sklopovlja. To je bio i glavni cilj našega rada te jedno od svojstava naših izvornih znanstvenih doprinosa D.2. i D.3. Također, unaprjeđenje u obliku iskorištavanja raznorodnosti komunikacijskih veza sklopovlja pri raspoređivanju

pokazalo se kao iznimno važno u ostvarivanju boljih performansi raspoređivanja, čineći većinu našeg doprinosa D.3.

Za ispitane slučajeve analiziramo i energiju kao drugi pokazatelj kvalitete, za koju su prosječne vrijednosti dva glavna parametra ε i ε^{raz} prikazani slikom 5.49. Najveći procijenjeni utrošak energije daju programi u izvođenju prema rasporedu dobivenom postupkom LDCP, slika 5.49a. Razlog je tome što, unatoč velikom prostoru u rasporedu koji se iskorištava za labavljenje vremenskih isječaka postupkom LVI, raspored jedinica programa nije dovoljno učinkovit kako bi se taj nedostatak kompenzirao uštedom energije. HEFT pokazuje najmanji utrošak, što u stvarnosti nije slučaj, jer je obujam komunikacije koji daje smanjen (zbog računanja prosjeka), tako da je i utrošak energije manji nego što bi bio pri izvođenju programa. Zbog toga ga nije moguće izravno usporediti s dobivenim vrijednostima kod ostalih postupaka. Najmanji procijenjeni utrošak energije dobiva se izvršavanjem programa prema rasporedu dobivenom postupkom RaNDKiP1, nakon kojeg odmah slijedi RaNDKiP. Primjerice, razlika u utrošku energije između postupaka RaNDKiP1 i LDCP iznosi oko 6,3%, dok je ona nešto manja kod postupka RaNDKiP i iznosi 6,2%. S druge strane, ako se promotri ušteda energije primjenom postupka LVI na rasporede dobivene različitim postupcima raspoređivanja, najveći je primjetan kod postupka RaNDKiP2, što prikazuje slika 5.49b. Sljedeći po uštedi energije je postupak LDCP, s neznatno manjom vrijednosti ε^{raz}, dok je najmanja ušteda kod postupaka RaNDKiP i RaNDKiP1. Manja ušteda energije indicira i bolje vremenski popunjen raspored, što se može zaključiti i usporedbom rezultata dobivenih vremenskom analizom. Uz to, ovdje je prikazana i prednost postupka RaNDKiP2 u odnosu na LDCP jer, iako imaju gotovo istu uštedu energije, ukupni energetski trag manji je kod postupka RaNDKiP2 čineći ga boljim odabirom kod raspoređivanja.



(a) Ukupni utrošak energije ε.

(b) Ušteda energije ε^{raz}.

Slika 5.49: Dobiveni utrošci i ušteda energije pri raspoređivanju programa na računalne platforme s raznorodnim komunikacijskim vezama.

Dobiveni rezultati analize utroška i uštede energije pokazali su kako je postupak LVI kao dio unaprijeđenog postupka RaNDKiP primjenjiv i na rasporede dobivene drugim postupcima raspoređivanja, omogućavajući uštede od 14 do gotovo 20%. Sam postupak RaNDKiP unatoč boljim performansama u vremenskoj analizi pokazuje prostor za uštedu energije od 14%, čime potvrđujemo i postizanje željenih ciljeva u obliku doprinosa D.3., tj. dobivanja energetski učinkovitijeg rasporeda.

Drugi naš unaprijeđeni postupak RaPPaMaG i njegovu analizu predstavili smo u pododjeljku 5.6. Navedenom analizom prikazali smo kako možemo smanjiti prostor mogućih rasporeda podjelom pro-

grama na vremenski neovisne cjeline koje nazivamo fazama izvršavanja. Takav pristup moguće je primijeniti ne samo na programe prikazane novim modelom u obliku miješanoga acikličkoga grafa, već i programe prikazane usmjerenim acikličkim grafom kao najčešćim modelom te svrhe u praksi. Tu smo mogućnost prikazali na postojećim programima iz repozitorija Pegasus te programima generiranima alatom TGFF. Nadalje, s obzirom na to da prema našim saznanjima primjera programa u obliku miješanoga acikličkoga grafa nema, predstavili smo vlastiti primjer — grupno sinkroni program, za čiju analizu smo uspješno primijenili i drugi korak postupka RaPPaMaG, a to je raspoređivanje faza izvršavanja iscrpnim pretraživanjem. Iscrpno pretraživanje može dati optimalne rezultate za manje instance problema (dvadesetak jedinica programa i sklopovlja), dok se za ostale mogu primijeniti drugi postupci raspoređivanja, poput bioinspiriranih metaheursitičkih postupaka. Prikazanim smo primjerima pokazali kako se particioniranjem programa na faze izvršavanja olakšava njegova analiza, ali i povećavaju mogućnosti za dobivanje boljeg rasporeda. Na taj način omogućujemo alternativu postojećim iterativnim postupcima raspoređivanja, koji pretražuju čitav prostor rješenja i poboljšavamo performanse postupka, čime zaokružujemo i naš doprinos D.2.

Unaprijeđenim postupcima raspoređivanja RaPPaMaG i RaNDKiP omogućujemo višestruk napredak u razvoju paralelnih programa. Postupkom RaPPaMaG, kao dijelom doprinosa D.2., omogućujemo analizu programa visoke razine zrnatosti, izravnu upotrebu višestrukih kriterija raspoređivanja koji mogu imati različite važnosti te podršku za programe sa sinkronom komunikacijom među izvršnim jedinicama. Postupkom RaNDKiP omogućujemo izravnu podršku za raznorodnost programa, sklopovlja, ali i komunikacijskih veza među obradbenim jedinicama, skraćivanje vremena izvršavanja programa i uštedu energije.

Poglavlje 6

Zaključak

U svrhu što boljeg korištenja modernih računalnih platformi potrebno je prilagoditi postojeće programe koji se na njima izvode, ali i nove razvijati u skladu s nadolazećim tehnologijama sklopovlja. To je posebno izraženo u području računarstva visokih performansi (HPC) gdje se rukuje s velikim, raznorodnim, visoko raspodijeljenim i paralelnim sustavima goleme potrošnje energije. U navedenom području, jedan od načina kako programe prilagoditi za rad na takvim sustavima jest i kvalitetno raspoređivanje programa na sklopovlje. Pritom je važno postupak raspoređivanja provoditi u skladu s raznorodnom naravi obradbenih jedinica, kao i komunikacijskih veza među njima. Kod postojećih postupaka raspoređivanja podrazumijeva se raznorodnost obradbenih sposobnosti jedinica sklopovlja, ali ne i raznorodnost komunikacijske propusnosti unutar njega. Uz to, postojeći postupci raspoređivanja često ne podrazumijevaju više od jednog pokazatelja kvalitete, unatoč tome što je pokazatelja sve više: energija, ujednačavanje opterećenja, troškovi, dostupnost itd. U postojećim postupcima raspoređivanja nije moguće ni zadavati različite težine pokazateljima kvalitete, što primjerice može biti iznimno važno u sustavima oblaka računala gdje troškovi mogu igrati veću ulogu od performansi. S obzirom na to da struktura postupaka raspoređivanja nije ususstavljena i modularna, nemoguće je usporediti napretke koje donose različiti postupci.

Zbog navedenih razloga, u ovoj disertaciji predstavili smo programski okvir LOSECO, kojim smo definirali strukturu modularnog sustava šire upotrebe za raspoređivanje programa na raznorodne računalne platforme. U sklopu navedenoga okvira, izuzev modela sklopovlja, opisali smo i novi model programa u obliku miješanoga acikličkoga grafa, kojim se, osim vremenskog slijeda među jedinicama programa, može opisati i sinkrona komunikacija među njima. Uz ta dva modela predstavili smo i model raspoređivanja prema više pokazatelja kvalitete kojima se eksplicitno mogu zadavati težine. Da bi iskoristili nove perspektive koje nude spomenuti modeli, razvili smo i novi postupak raspoređivanja u dva koraka — RaPPaMaG. Prvim korakom spomenutog postupka omogućuje se dijeljenje programa (particioniranje) na tzv. faze izvršavanja, čime se olakšava analiza programa s velikim brojem izvršnih jedinica. Navedenim postupkom particionirali smo i programe zadane u obliku usmjerenoga acikličkoga grafa (DAG-a) iz repozitorija Pegasus te programe generirane alatom TGFF. Na taj smo način dokazali da je prvi korak postupka RaPPaMaG primjenjiv, ne samo na programe zadane u novom obliku miješanoga acikličkoga

grafa, već i na postojeće programe. Drugim korakom postupka RaPPaMaG omogućava se raspoređivanje pojedinih faza izvršavanja dobivenih particioniranjem za manje instance istih, što smo prikazali na primjeru grupno-sinkronog programa koristeći dva pokazatelja kvalitete — duljinu rasporeda i energiju.

Za programe u obliku DAG-a razvili smo poseban postupak raspoređivanja, RaNDKiP, zasnovan na rangiranju jedinica programa, kojim smo povećali podršku za raznorodnost sklopovlja, posebice komunikacijskih veza. Navedeni postupak omogućuje punu podršku za raznorodnost komunikacijskih kanala među obradbenim jedinicama u fazi rangiranja jedinica programa, ali i u fazi ažuriranja stanja nakon svakog koraka raspoređivanja. Eksperimentalnom analizom smo pokazali da navedenim svojstvima postupak RaNDKiP omogućuje značajno kraća vremena izvršavanja programa od onih koja postižu postojeći postupci HEFT i LDCP, kao jedni od najboljih te vrste prema dostupnoj literaturi. U svrhu smanjenja energije, drugog po važnosti pokazatelja kvalitete u praksi, razvili smo i postupak labavljenja vremenskih isječaka (LVI) kao jednog od koraka postupka RaNDKiP. Postupkom LVI se u kombinaciji s RaNDKiP, ali i ostalim postupcima raspoređivanja, omogućuje ušteda energije na računalnim sustavima koji podržavaju dinamičko skaliranje napona i frekvencije (DVFS). Rezultati mjerenja pokazuju da se ušteda energije primjenom postupka LVI kreće od 14 do 20% i to bez utjecaja na ukupnu duljinu rasporeda. Također, pokazali smo da se u gušćim rasporedima postupkom LVI štedi manje energije (primjerice kod RaNDKiP i RaNDKiP1), nego što je to slučaj u rjeđima (kod postupaka HEFT, LDCP i RaNDKiP2).

Svojstva modularnog prikaza sustava za raspoređivanje u obliku programskog okvira LOSECO, novi modeli prikaza programa i samog postupka raspoređivanja, kao i unaprijeđeni postupci raspoređivanja dovode do značajnih ušteda u izvođenju programa na sklopovlju sustava HPC-a. Također, navedeni će znanstveni pomaci svakako pozitivno utjecati na buduće istraživanje u tom području, ali i ukazati na nove probleme koji se do sada nisu često razmatrali.

Bibliografija

[1] G. Martinović and Z. Krpić, "Towards green HPC blueprints," in *The Second International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2011)*, 2011, pp. 113–118.

[2] Pegasus, workflow management system. Information Sciences Institute, Unviersity of Southern California. [Online]. Available: https://pegasus.isi.edu/

[3] N. B. Rizvandi, J. Taheri, and A. Y. Zomaya, "Some observations on optimal frequency selection in dvfs-based energy consumption minimization," *J. Parallel Distrib. Comput.*, vol. 71, no. 8, pp. 1154–1164, august 2011. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2011.01.004

[4] cubieforums, "CB2 overclocking," july 2014. [Online]. Available: http://www.cubieforums.com/index.php/topic,1278.0.html

[5] M. L. Norman and D. P. Wall, "Community input on the future of high performance computing," High-Performance Computing Task Force, Workshop, 2009.

[6] L. Branscomb, "National laboratories: The search for new missions and new structures," *Empowering Technology: Implementing a US Strategy*, pp. 103–134, 1993.

[7] Top500, "Top 500 supercomputer sites," http://www.top500.org/, 2014.

[8] Green500, "Ranking the world's most energy-efficient supercomputers," http://www.green500.org/, 2014.

[9] "Experimenting hpc systems with simulation," Nancy University, France. [Online]. Available: http://webloria.loria.fr/ quinson/blog/2010/0628/Tutorial_at_HPCS/

[10] R. Couturier, D. Laiymani, and S. Miquée, "Maheve: an efficient reliable mapping of asynchronous iterative applications on volatile and heterogeneous environments," in *Euro-Par 2010 Parallel Processing Workshops*. Springer, 2011, pp. 31–39.

[11] W. Sun and T. Sugawara, "Heuristics and evaluations of energy-aware task mapping on heterogeneous multiprocessors," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, 2011, pp. 599–607.

[12] Z. Jia, A. Núñez, T. Bautista, and A. Pimentel, "A two-phase design space exploration strategy for system-level real-time application mapping onto mpsoc," *Microprocessors and Microsystems*, vol. 38, no. 1, pp. 9–21, 2014.

[13] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 682–694, Mar. 2014.

[14] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, ser. IISWC '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 44–54.

[15] Z. Cao, H. Tang, Q. Li, B. Li, F. Chen, K. Wang, X. An, and N. Sun, "Design of hpc node with heterogeneous processors," in *Proceedings of the 2011 IEEE International Conference on Cluster Computing*, ser. CLUSTER '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 130–138.

[16] I. D. Falco, U. Scafuri, and E. Tarantino, "An adaptive multisite mapping for computationally intensive grid applications," *Future Generation Computer Systems*, vol. 26, no. 6, pp. 857 – 867, 2010.

[17] J. Taheri, Y. Choon Lee, A. Y. Zomaya, and H. J. Siegel, "A bee colony based optimization approach for simultaneous job scheduling and data replication in grid environments," *Comput. Oper. Res.*, vol. 40, no. 6, pp. 1564–1578, Jun. 2013.

[18] S. Sotiriadis, N. Bessis, F. Xhafa, and N. Antonopoulos, "From meta-computing to interoperable infrastructures: A review of meta-schedulers for hpc, grid and cloud," in *Proceedings of the 2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, ser. AINA '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 874–883. [Online]. Available: http://dx.doi.org/10.1109/AINA.2012.15

[19] H. Hussain, S. U. R. Malik, A. Hameed, S. U. Khan, G. Bickler, N. Min-Allah, M. B. Qureshi, L. Zhang, W. Yongji, N. Ghani, J. Kolodziej, A. Y. Zomaya, C.-Z. Xu, P. Balaji, A. Vishnu, F. Pinel, J. E. Pecero, D. Kliazovich, P. Bouvry, H. Li, L. Wang, D. Chen, and A. Rayes, "A survey on resource allocation in high performance distributed computing systems," *Parallel Computing*, vol. 39, no. 11, pp. 709 – 736, 2013.

[20] R. Sharma, V. K. Soni, M. K. Mishra, and P. Bhuyan, "A survey of job scheduling and resource management in grid computing," *World Academy of Science, Engineering and Technology*, vol. 40, pp. 461–466, 2010.

[21] Z. Guo, G. Fox, and M. Zhou, "Investigation of data locality and fairness in MapReduce," in *Proceedings of third international workshop on MapReduce and its Applications Date*, ser. MapReduce '12. New York, NY, USA: ACM, 2012, pp. 25–32.

[22] H. Cao, H. Jin, X. Wu, S. Wu, and X. Shi, "Dagmap: efficient and dependable scheduling of dag workflow job in grid," *The Journal of Supercomputing*, vol. 51, no. 2, pp. 201–223, 2010. [Online]. Available: http://dx.doi.org/10.1007/s11227-009-0284-7

[23] Y. Zhang and et al., "Relative performance of scheduling algorithms in grid environments," 2007.

[24] Z. Krpic, G. Horvat, D. Žagar, and G. Martinovic, "Towards an energy efficient soc computing cluster," in *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014 37th International Convention on.* IEEE, May 2014, pp. 178–182.

[25] G. Bosilca, A. Bouteiller, A. Danalis, T. Herault, P. Lemarinier, and J. Dongarra, "Dague: A generic distributed dag engine for high performance computing," *Parallel Computing*, vol. 38, no. 1, pp. 37–51, 2012.

[26] J. Brandt, A. Gentile, J. Mayo, P. Pebay, D. Roe, D. Thompson, and M. Wong, "Resource monitoring and management with ovis to enable hpc in cloud computing environments," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, May 2009, pp. 1–8.

[27] M. Lee, H. Jo, D. H. Choi, and S. W. Baik, "Generalized parallelization methodology for heterogeneous hpc platforms," in *Cloud Computing and Social Networking (ICCCSN), 2012 International Conference on*, 2012, pp. 1–6.

[28] B. Lisper, "Towards parallel programming models for predictability," in *Proc. 12th International Workshop on Worst-Case Execution-Time Analysis (WCET'12)*, T. Vardanega, Ed. Schloss Dagstuhl, July 2012.

[29] A. W. D. O. Rodrigues, F. Guyomarc'h, and J.-L. Dekeyser, "A modeling approach based on uml/marte for gpu architecture," *CoRR*, vol. abs/1105.4424, 2011.

[30] K. Sigdel, C. Galuzzi, K. Bertels, M. Thompson, and A. D. Pimentel, "Evaluation of runtime task mapping using the rsesame framework," *Int. J. Reconfig. Comput.*, vol. 2012, pp. 14:14–14:14, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1155/2012/234230

[31] C. Erbas, S. Cerav-Erbas, and A. D. Pimentel, "Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design," *Evolutionary Computation, IEEE Transactions on*, vol. 10, no. 3, pp. 358–374, 2006.

[32] H. Rahmawan and Y. Gondokaryono, "The simulation of static load balancing algorithms," in *Electrical Engineering and Informatics, 2009. ICEEI '09. International Conference on*, vol. 02, 2009, pp. 640–645.

[33] T. T. H. Le, R. Passerone, U. Fahrenberg, and A. Legay, "Tag machines for modeling heterogeneous systems," in *Application of Concurrency to System Design (ACSD), 2013 13th International Conference on*, 2013, pp. 186–195.

[34] P.-C. Chang, I.-W. Wu, J.-J. Shann, and C.-P. Chung, "ETAHM: an energy-aware task allocation algorithm for heterogeneous multiprocessor," in *Proceedings of the 45th annual Design Automation Conference*, ser. DAC '08. New York, NY, USA: ACM, 2008, pp. 776–779.

[35] P. Ezzatti, M. Pedemonte, and A. MartíN, "An efficient implementation of the min-min heuristic," *Comput. Oper. Res.*, vol. 40, no. 11, pp. 2670–2676, Nov. 2013.

[36] T. Blickle, J. Teich, and L. Thiele, "System-level synthesis using evolutionary algorithms," *Design Automation for Embedded Systems*, vol. 3, no. 1, pp. 23–58, 1998.

[37] M. I. Daoud and N. Kharma, "A high performance algorithm for static task scheduling in heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 68, no. 4, pp. 399 – 409, 2008.

[38] X. Tang, K. Li, G. Liao, and R. Li, "List scheduling with duplication for heterogeneous computing systems," *Journal of parallel and distributed computing*, vol. 70, no. 4, pp. 323–329, 2010.

[39] S. Su, Q. Huang, J. Li, X. Cheng, P. Xu, and K. Shuang, "Enhanced energy-efficient scheduling for parallel tasks using partial optimal slacking," *The Computer Journal*, 2014. [Online]. Available: http://comjnl.oxfordjournals.org/content/early/2014/02/06/comjnl.bxu002.abstract

[40] S. Chai, Y. Li, J. Wang, and C. Wu, "A list simulated annealing algorithm for task scheduling on network-on-chip," *Journal of Computers*, vol. 9, no. 1, pp. 176–182, 2014.

[41] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 13, no. 3, pp. 260–274, 2002.

[42] F. Suter, F. Desprez, and H. Casanova, "From heterogeneous task scheduling to heterogeneous mixed parallel scheduling," in *Euro-Par 2004 Parallel Processing*. Springer, 2004, pp. 230–237.

[43] I. Jones and C. Engelmann, "Simulation of large-scale hpc architectures," in *Parallel Processing Workshops (ICPPW), 2011 40th International Conference on*, 2011, pp. 447–456.

[44] M. Mezmaz, N. Melab, Y. Kessaci, Y. C. Lee, E.-G. Talbi, A. Y. Zomaya, and D. Tuyttens, "A Parallel Bi-objective Hybrid Metaheuristic for Energy-aware Scheduling for Cloud Computing Systems," *Journal of Parallel and Distributed Computing*, Apr 2011.

[45] M. P. Forum, "MPI 2.2: A message-passing interface standard," University of Tennessee and Oak Ridge National Laboratory, Knoxville, TN, USA, Tech. Rep., 2009.

[46] OpenMP Architecture Review Board, "OpenMP application program interface version 3.0," May 2008. [Online]. Available: http://www.openmp.org/mp-documents/spec30.pdf

[47] R. Rabenseifner, G. Hager, and G. Jost, "Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes," in *Proceedings of the 2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, ser. PDP '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 427–436. [Online]. Available: http://dx.doi.org/10.1109/PDP.2009.43

[48] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures," *Concurr. Comput. : Pract. Exper.*, vol. 23, no. 2, pp. 187–198, Feb. 2011.

[49] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on*. IEEE, 2008, pp. 1–10.

[50] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good *et al.*, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[51] J. Yang, H. Zhang, Y. Ling, C. Pan, and W. Sun, "Task allocation for wireless sensor network using modified binary particle swarm optimization," *Sensors Journal, IEEE*, vol. 14, no. 3, pp. 882–892, March 2014.

[52] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004.

[53] K. Hoste and L. Eeckhout, "Microarchitecture-independent workload characterization," *Micro, IEEE*, vol. 27, no. 3, pp. 63–72, May 2007.

[54] A. Kerr, G. Diamos, and S. Yalamanchili, "Modeling GPU-CPU workloads and systems," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, ser. GPGPU '10. New York, NY, USA: ACM, 2010, pp. 31–42.

[55] J. L. Ortega-Arjona, *Architectural Patterns for Parallel Programming: Models for Performance Estimation.* Saarbrücken, Germany, Germany: VDM Verlag, 2009.

[56] T. G. Mattson, B. A. Sanders, and B. L. Massingill, *Patterns for parallel programming.* Pearson Education, 2004.

[57] I. Crnkovic, *Building Reliable Component-Based Software Systems*, M. Larsson, Ed. Norwood, MA, USA: Artech House, Inc., 2002.

[58] Z. Vrba, H. Espeland, P. Halvorsen, and C. Griwodz, "Limits of work-stealing scheduling," in *Job Scheduling Strategies for Parallel Processing*, E. Frachtenberg and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 280–299.

[59] K. Vallerio, "Task graphs for free (tgff v3. 0)," *User's manual*, 2003.

[60] Microsoft C# task parallel library (TPL). [Online]. Available: http://msdn.microsoft.com/en-us/library/dd460717(v=vs.110).aspx

[61] Vampir - performance optimization tool. [Online]. Available: http://www.vampir.eu/

[62] S. Sanyal and S. K. Das, "MaTCH: Mapping data-parallel tasks on a heterogeneous computing platform using the cross-entropy heuristic," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers - Volume 01*, ser. IPDPS '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 64.2–.

[63] B. Song, C. Ernemann, and R. Yahyapour, "Parallel computer workload modeling with markov chains," in *Proceedings of the 10th international conference on Job Scheduling Strategies for Parallel Processing*, ser. JSSPP'04. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 47–62. [Online]. Available: http://dx.doi.org/10.1007/11407522_3

[64] C. Roig, A. Ripoll, and F. Guirado, "A new task graph model for mapping message passing applications," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 18, no. 12, pp. 1740–1753, 2007.

[65] J. Feljan, J. Carlson, and T. Seceleanu, "Towards a model-based approach for allocating tasks to multicore processors," in *Proceedings of the 2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, ser. SEAA '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 117–124.

[66] T. Li, V. K. Narayana, and T. El-Ghazawi, "A static task scheduling framework for independent tasks accelerated using a shared graphics processing unit," in *Proceedings of the 2011 IEEE 17th International Conference on Parallel and Distributed Systems*, ser. ICPADS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 88–95.

[67] M. A. Iverson, F. Özgüner, and G. J. Follen, "Parallelizing existing applications in a distributed heterogeneous environment," in *4TH HETEROGENEOUS COMPUTING WORKSHOP (HCW'95.* Citeseer, 1995.

[68] J. Warnke, S. Pawaskar, and H. Ali, "An energy-aware bioinformatics application for assembling short reads in high performance computing systems," in *High Performance Computing and Simulation (HPCS), 2012 International Conference on.* IEEE, 2012, pp. 154–160.

[69] R. Dick, D. Rhodes, and W. Wolf, "TGFF: task graphs for free," in *Hardware/Software Codesign, 1998. (CODES/CASHE '98) Proceedings of the Sixth International Workshop on*, 1998, pp. 97–101.

[70] M. A. Awan and S. M. Petters, "Energy-aware partitioning of tasks onto a heterogeneous multicore platform," in *Proceedings of the 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, ser. RTAS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 205–214.

[71] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, "Software architecture optimization methods: A systematic literature review," *Software Engineering, IEEE Transactions on*, vol. 39, no. 5, pp. 658–683, 2013.

[72] M. Qamhieh, F. Fauberteau, L. George, and S. Midonnet, "Global edf scheduling of directed acyclic graphs on multiprocessor systems," in *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, ser. RTNS '13. New York, NY, USA: ACM, 2013, pp. 287–296. [Online]. Available: http://doi.acm.org/10.1145/2516821.2516836

[73] D. L. Long and L. A. Clarke, "Task interaction graphs for concurrency analysis," in *Proceedings of the 11th international conference on Software engineering*, ser. ICSE '89. New York, NY, USA: ACM, 1989, pp. 44–52.

[74] D. Eadline, "Comparing mpi and openmp," Aug. 2009. [Online]. Available: http://www.clusterconnection.com/2009/08/comparing-mpi-and-openmp/

[75] Z. Krpic, G. Martinovic, and I. Crnkovic, "Green HPC: MPI vs. openMP on a shared memory system," in *MIPRO, 2012 Proceedings of the 35th International Convention.* IEEE, 2012, pp. 246–250.

[76] I. Švogor, I. Crnkovic, and N. Vrček, "An extended model for multi-criteria software component allocation on a heterogeneous embedded platform," *CIT. Journal of Computing and Information Technology*, vol. 21, no. 3, pp. 211–222, Jan. 2014.

[77] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, Dec. 1998. [Online]. Available: http://dx.doi.org/10.1137/S1064827595287997

[78] D. Batista and N. da Fonseca, "A brief survey on resource allocation in service oriented grids," in *Globecom Workshops, 2007 IEEE*, Nov 2007, pp. 1–5.

[79] S. K. Garg, C. S. Yeo, A. Anandasivam, and R. Buyya, "Environment-conscious scheduling of hpc applications on distributed cloud-oriented data centers," *J. Parallel Distrib. Comput.*, vol. 71, no. 6, pp. 732–749, Jun. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2010.04.004

[80] M. Warren, E. Weigle, and W. Feng, "High-density computing: A 240-node beowulf in one cubic meter," in *Supercomputing 2002*, 2002.

[81] M. Gachechiladze-Bozhesku, *Electronic Product Environmental Assessment Tool (EPEAT)*, 0th ed. SAGE Publications, Inc., 2011, pp. 153–155. [Online]. Available: http://dx.doi.org/10.4135/9781412975704

[82] C. Lee, H. Kim, H. woo Park, S. Kim, H. Oh, and S. Ha, "A task remapping technique for reliable multi-core embedded systems," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2010 IEEE/ACM/IFIP International Conference on*, Oct 2010, pp. 307–316.

[83] A. Mazouz, A. Laurent, B. Pradelle, and W. Jalby, "Evaluation of cpu frequency transition latency," *Computer Science - Research and Development*, pp. 1–9, 2013. [Online]. Available: http://dx.doi.org/10.1007/s00450-013-0240-x

[84] P. Rong and M. Pedram, "Energy-aware task scheduling and dynamic voltage scaling in a real-time system," *Journal of Low Power Electronics*, vol. 4, no. 1, pp. 1–10, 2008.

[85] L. Wang, S. U. Khan, D. Chen, J. Kołodziej, R. Ranjan, C.-z. Xu, and A. Zomaya, "Energy-aware parallel task scheduling in a cluster," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1661–1670, 2013.

[86] A. Das, A. Kumar, and B. Veeravalli, "Communication and migration energy aware task mapping for reliable multiprocessor systems," *Future Generation Computer Systems*, vol. 30, no. 0, pp. 216 – 228, 2014, special Issue on Extreme Scale Parallel Architectures and Systems, Cryptography in Cloud Computing and Recent Advances in Parallel and Distributed Systems, {ICPADS} 2012 Selected Papers. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X13001301

[87] D. M. Quan, F. Mezza, D. Sannenli, and R. Giafreda, "T-alloc: A practical energy efficient resource allocation algorithm for traditional data centers," *Future Gener. Comput. Syst.*, vol. 28, no. 5, pp. 791–800, May 2012. [Online]. Available: http://dx.doi.org/10.1016/j.future.2011.04.020

[88] E. M. Saad, M. H. Awadalla, M. Shalan, and A. Elewi, "Energy-aware task partitioning on heterogeneous multiprocessor platforms," *CoRR*, vol. abs/1206.0396, 2012.

[89] M. Maheswaran and H. J. Siegel, "A dynamic matching and scheduling algorithm for heterogeneous computing systems," in *7th HeterogeneousComputing Workshop (HCW '98*, 1998, pp. 57–69.

[90] G. Sapienza, T. Seceleanu, and I. Crnkovic, "Architectural decisions for hw/sw partitioning based on multiple extra-functional properties," in *Software Architecture (WICSA), 2014 Working IEEE/IFIP Conference on*, April 2014.

[91] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, and Z. Gu, "Online optimization for scheduling preemptable tasks on iaas cloud systems," *J. Parallel Distrib. Comput.*, vol. 72, no. 5, pp. 666–677, May 2012. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2012.02.002

[92] R. Marler and J. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and Multidisciplinary Optimization*, vol. 26, no. 6, pp. 369–395, 2004. [Online]. Available: http://dx.doi.org/10.1007/s00158-003-0368-6

[93] ——, "The weighted sum method for multi-objective optimization: new insights," *Structural and Multidisciplinary Optimization*, vol. 41, no. 6, pp. 853–862, 2010.

[94] J. Andersson, "A survey of multiobjective optimization in engineering design," 2000.

[95] R. Li, R. Etemaadi, M. T. Emmerich, and M. R. Chaudron, "An evolutionary multiobjective optimization approach to component-based software architecture design," in *Evolutionary Computation (CEC), 2011 IEEE Congress on*. IEEE, 2011, pp. 432–439.

[96] W.-S. Chang and C.-C. Chyu, "A multi-criteria decision making for the unrelated parallel machines scheduling problem." *JSEA*, vol. 2, no. 5, pp. 323–329, 2009. [Online]. Available: http://dblp.uni-trier.de/db/journals/jsea/jsea2.html#ChangC09

[97] W. Chen, "Task partitioning and mapping algorithms for multi-core packet processing systems," 2009.

[98] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The Landscape of Parallel Computing Research: A View from Berkeley," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2006-183, Dec 2006.

[99] G. Sapienza, T. Seceleanu, and I. Crnknovic, "Partitioning decision process for embedded hardware and software deployment," in *Computer Software and Applications Conference Workshops (COMPSACW), 2013 IEEE 37th Annual*, July 2013, pp. 674–680.

[100] Z. Krpić, I. Crnković, and J. Carlson, "Towards a common software-to-hardware allocation framework for the heterogeneous high performance computing," in *Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual*. IEEE, 2014.

[101] F. Harary and E. Palmer, "Enumeration of mixed graphs," *Proceedings of the American Mathematical Society*, vol. 17, no. 3, pp. 682–687, 1966.

[102] R. Gardner, B. Bobga, C. Nguyen, and G. Coker, "Some graph, digraph, and mixed graph results concerning decompositions, packings, and coverings," in *Joint Meeting of the AMS and the MAA, AMS Special Session on Design Theory and Graph Theory, I, Atlanta, Georgia*, 2005.

[103] G. W. Flake, R. E. Tarjan, and K. Tsioutsiouliklis, "Graph clustering and minimum cut trees," *Internet Mathematics*, vol. 1, pp. 385–408, 2004.

[104] E. W. Weisstein, "Multichoose," 2014. [Online]. Available: http://mathworld.wolfram.com/Multichoose.html

[105] E. R. Gansner and S. C. North, "An open graph visualization system and its applications to software engineering," *Software Practice and Experience*, vol. 30, no. 11, pp. 1203–1233, 2000.

[106] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull, "Graphviz and dynagraph—static and dynamic graph drawing tools," in *Graph drawing software*. Springer, 2004, pp. 127–148.

[107] E. Carvalho, C. Marcon, N. Calazans, and F. Moraes, "Evaluation of static and dynamic task mapping algorithms in noc-based mpsocs," in *System-on-Chip, 2009. SOC 2009. International Symposium on*, Oct 2009, pp. 087–090.

[108] G. Sander, "Visualization of compiler graphs," *Lemke, G. Sander and the COMPARE Consortium*, vol. 3, 1995.

[109] ——, "VCG-visualization of compiler graphs," *User Documentation*, 2005.

[110] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.

[111] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1997.

[112] B. Subramaniam and W. chun Feng, "Understanding power measurement implications in the green500 list," in *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, Dec 2010, pp. 245–251.

[113] C.-H. Hsu, W. chun Feng, and J. Archuleta, "Towards efficient supercomputing: a quest for the right metric," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, April 2005, pp. 8 pp.–.

[114] J. L. Gustafson, "Reevaluating amdahl's law," *Communications of the ACM*, vol. 31, pp. 532–533, 1988.

Sažetak i ključne riječi

Postupci raspoređivanja programa na sklopovlje važan su mehanizam učinkovitog korištenja novih sklopovskih tehnologija u području računarstva visokih performansi (HPC). Postoje mnoge vrste takvih postupaka, no najraširenija i najbolji su postupci rangiranja jedinica programa te (meta)heuristički postupci većinom zasnovani na vođenom lokalnom pretraživanju. Međutim, prate ih nestrukturiranost i nemogućnost da u punoj mjeri koriste raznorodnu prirodu sklopovlja, posebice komunikacijske mreže među obradbenim jedinicama. Problem stvara i rastuća potreba za praćenjem višestrukih pokazatelja kvalitete (vrijeme izvršavanja, energija, troškovi itd.), kao i manjak podrške za davanjem važnosti pojedinim pokazateljima kvalitete. U ovom radu predstavljamo LOSECO modularni programski okvir za optimalno dodjeljivanje programa na raznorodne raspodijeljene računalne sustave. U sklopu tog okvira predstavljamo i skup modela s posebnim naglaskom na modelu programa u obliku miješanoga acikličkoga grafa koji omogućuje istovremeni prikaz vremenskog slijeda među jedinicama programa i njihove sinkrone komunikacije te okvir za univerzalni model raspoređivanja pomoću više korisnički naglašenih kriterija (pokazatelja kvalitete). Uz to smo razvili i unaprijeđeni postupak raspoređivanja u dva koraka, RaPPaMaG, koji koristi navedene modele kako bi omogućio particioniranje programa na vremenski neovisne faze izvršavanja, a zatim i optimalno raspoređivanje dobivenih particija. Pokazali smo da postupak RaPPaMaG, osim na programe u obliku miješanoga grafa, možemo primijeniti i na postojeće programe iz repozitorija Pegasus te alata za njihovo stvaranje TGFF. S druge strane razvili smo još jedan unaprijeđeni postupak raspoređivanja, RaNDKiP, zasnovan na rangiranju jedinica programa, koji omogućuje iskorištavanje raznorodnosti mreže među obradbenim jedinicama kako bi unaprijedio dobiveni raspored u odnosu na postojeće postupke HEFT i LDCP. Osim skraćivanja vremena izvršavanja programa, postupak RaNDKiP omogućuje smanjivanje utroška energije labavljenjem nekritičnih vremenskih isječaka u rasporedu tzv. postupkom LVI. Postupak LVI omogućuje smanjenje utroška energije od 14% do 20% u prosječnom slučaju ovisno o tipu prethodno upotrijebljenog postupka raspoređivanja.

Ključne riječi: Energetski svjesne heuristike, model programa u obliku miješanoga acikličkoga grafa, računarstvo visokih performansi, raspoređivanje, raznorodne računalne platforme.

Abstract and keywords

In high performance computing (HPC) scheduling methods are one of the most important mechanisms that ensure the efficient use of the emergent hardware technologies. Amongst different types of scheduling, list scheduling methods and various guided local search-based (meta)heuristics are considered as the best, and therefore they are widely used. However, these methods lack structured approach as well as the ability to fully exploit the heterogeneous nature of the existing hardware, mostly the heterogeneity of its network. Another problem is the growing number of different quality attributes, such as execution time, energy consumption, costs, etc., their simultaneous use, and their prioritization. In this thesis we introduce LOSECO framework, a modular structure of the comprehensive scheduling system. Within this framework we present a set of models, with one of them being the novel mixed acyclic graph software model that allows simultaneous description of the time dependency between software units and their synchronous communication and the other being the multiple-criteria scheduling model which allows the use of the multiple user prioritized criteria (quality attributes). Furthermore, we have developed RaPPaMaG, an improved scheduling method that comprises two steps: partitioning, which exploits new structure of the software model in order to split the program into the time independent execution phases, and the allocation step which allows their optimal allocation. We have shown that RaPPaMaG can be applied to the mixed acyclic graph software model, as well as the directed acyclic graph representation of the existing workloads from Pegasus and ones generated by TGFF. Additionally, we have developed a second improved, list-based, scheduling method called RaNDKiP, which exploits network heterogeneity, enabling shorter schedule lengths in heterogeneous systems when compared to the ones obtained by existing HEFT and LDCP. In addition to reducing the schedule length, RaNDKiP also enables lowering of the power consumption by slacking the non-critical time slots via LVI procedure. Our experimental analysis has shown that in the average case LVI reduces energy consumption by 14% to 20%, depending on the previously used scheduling method.

Keywords: Enegy-aware heuristics, mixed acyclic graph software model, high performance computing, scheduling, heterogeneous computing platforms.

Acknowledgement

This thesis work was partially supported by Mälardalen University in Sweden, and by the project "RALF3 - Software for Embedded High Performance Architectures" funded by Swedish Foundation for Strategic Research.

Životopis

Zdravko Krpić rođen je 1982. u Osijeku. U studenom 2007. godine stekao je akademski stupanj diplomiranog inženjera elektrotehnike na Elektrotehničkom fakultetu Sveučilišta u Osijeku, s temom "Višekriterijsko dodjeljivanje resursa u raspodijeljenim raznorodnim računalnim sustavima". Na istoj ustanovi se zapošljava kao asistent u nastavi i istraživanju u studenom 2008.g., dok za to vrijeme radi i kao vanjski suradnik za Hrvatsku računalnu i istraživačku mrežu CARNet u području web tehnologija. U nastavi se bavi metodama paralelizacije programa upotrebom standarda izmjene poruka MPI i tehnologijom otvorene višeobrade OpenMP, zatim Linux i Unix operacijskim sustavima te modeliranjem i dizajnom programskih sustava. Tijekom rada na Elektrotehničkom fakultetu bavio se istraživanjem u području računalnih sustava visokih performansi kao korisnik Hrvatske nacionalne grid infrastrukture — CRO NGI, energetski učinkovitog računarstva, metodama višekriterijske optimizacije, CAPTCHA sustavima zaštite i postupcima paralelnog programiranja. Dobitnik je dvije ERASMUS stipendije, jednu za mobilnost nenastavnog osoblja, a drugu za mobilnost studenata poslijediplomskog studija u zbirnom trajanju od 9 mjeseci. U sklopu svog prvog istraživačkog boravka u inozemstvu, od rujna 2012., surađuje s odjelom programskog inženjerstva istraživačkog sjedišta za sustave stvarnog vremena Mälardalen sveučilišta u Švedskoj. Istraživač je i na znanstvenom projektu RALF3 švedske zaklade za strateška istraživanja u razvoju programske podrške za ugrađene arhitekture visokih performansi, od kojeg je dijelom bio i stipendiran za vrijeme svog drugog istraživačkog boravka u Švedskoj. Trenutno radi u području optimizacije paralelnih programa za rad na raznorodnim računalnim sustavima visokih performansi. Poseban mu je fokus rada povećanje vremenskih performansi programa u izvođenju na manjim raspodijeljenim računalnim sustavima, njihovo modeliranje te sintetiziranje u svrhu postizanja veće energetske učinkovitosti takvih sustava.