KOMBINOVANI ADAPTIVNI ALGORITAM ZA RASPODJELU O...

By: Luka Filipović

As of: Oct 3, 2019 2:04:54 PM
22,057 words - 65 matches - 19 sources

Similarity Index
6%

Mode: Similarity Report ▼

paper text:

UNIVERZITET CRNE GORE ELEKTROTEHNIČKI FAKULTET MSc Luka Filipović KOMBINOVANI ADAPTIVNI ALGORITAM ZA RASPODJELU OPTEREĆENJA PRI PARALELIZACIJI APLIKACIJA -DOKTORSKA DISERTACIJA- Podgorica, 2019. godine UNIVERSITY OF MONTENEGRO FACULTY OF ELECTRICAL ENGINEERING Luka Filipović, MSc. COMBINED ADAPTIVE LOAD BALANCING ALGORITHM FOR PARALLELIZATION OF APPLICATIONS DOCTORAL DISSERTATION Podgorica, 2019.

PODACI I INFORMACIJE O DOKTORANTU Ime i prezime: Luka Filipović **Datum i mjesto rođenja:** 02 **.02.** 1981. **godine, Podgorica Naziv završenog postdiplomskog studijskog programa i godina završetka:** 8

Elektrotehnički fakultet, odsjek za Elektroniku, telekomunikacije i računare, smjer Računarske nauke, 2009.

INFORMACIJE O DOKTORSKOJ DISERTACIJI Naziv doktorskih studija: Doktorske studije elektrotehnike Naziv teze: 8

Kombinovani adaptivni algoritam za Fakultet na kojem je disertacija odbranjena: raspodjelu opterećenja pri paralelizaciji aplikacija Elektrotehnički fakultet, Univerzitet Crne Gore, Podgorica

UDK, OCJENA I ODBRANA DOKTORSKE DISERTACIJE Datum prijave doktorske teze: 24.10.2014. **Datum sjednice Senata Univerziteta na kojoj je prihvaćena teza:** 26.03.2015. **Komisija za ocjenu podobnosti teze i kandidata: Mentor: Komisija za ocjenu doktorske disertacije: Komisija za odbranu** 7

doktorske disertacije: Datum odbrane: Datum promocije:

Prof. dr Igor Đurović Prof. dr Božo Krstajić Prof. dr Slobodan Đukanović Prof. dr Božo Krstajić Najsrdačnije se zahvaljujem mentoru prof. dr Božu Krstajiću, profesoru Elektrotehničkog fakulteta Univerziteta Crne Gore, na dragocjenim savjetima, pomoći i podršci tokom izrade ove disertacije, naučno-istraživačkog rada i dugogodišnje saradnje. Iskrenu zahvalnost dugujem prof. dr Milutinu Radonjiću, profesoru Elektrotehničkog fakulteta Univerziteta Crne Gore, na korisnim savjetima oko naučno-istraživačkog rada i računarskih simulacija predstavljenih u ovoj tezi. Prijateljima i kolegama dugujem zahvalnost na razumijevanju, pomoći i podršci za vrijeme izrade ove disertacije. Na kraju, najveću zahvalnost dugujem porodici na konstantnoj podršci, motivaciji za rad, napredovanje i usavršavanje. Luka Filipović PODACI O DOKTORSKOJ DISERTACIJI Naziv doktorskih studija: Naslov doktorske disertacije: Doktorske studije elektrotehnike Kombinovani adaptivni algoritam za raspodjelu opterećenja pri paralelizaciji aplikacija Datum prijave disertacije: Ključne riječi: distribuirani računarski sistemi, paralelno programiranje, load balancing algoritmi, raspodjela procesa, iskoristivost resursa, efikasnost Naučna oblast: Računarske i informacione nauke Uža naučna oblast: Distribuirani sistemi REZIME Razvoj i unapređenje efikasnih tehnika za raspodjelu procesa paralelnog programa na više jezgara procesora predstavlja jedan od problema paralelnih i distribuiranih računarskih sistema. Cilj unapređenja raspodjele procesa kod paralelnih aplikacija je povećanje performansi sistema, smanjenje vremena izvršenja aplikacije, smanjenje gubitaka i povećanje iskoristivosti resursa. U tezi su prikazani parametri za analizu performansi paralelnih aplikacija, podjela load balancing algoritama, njihove prednosti i nedostaci. Kao rezultat istraživanja, u disertaciji su prezentovana dva kombinovana load balancing algoritma koja se baziraju na domain decomposition i master-slave algoritmima. Algoritmi su kreirani radi smanjenja nedostataka u raspodjeli procesa paralelnih aplikacija koje se sastoje od više nezavisnih zadataka. Povećanje performansi i smanjenje neuravnoteženosti se vrši odabirom algoritama raspodjele u zavisnosti od segmenata u kojima su gubici najmanji i na osnovu prikupljenih parametara i prethodno definisanih uslova. Rezultati simulacija potvrdjuju bolje performanse predloženih algoritama u odnosu na standarne algoritme razmatrane u radu. INFORMATION ON DOCTORAL DISSERTATION Doctoral studies: Dissertation title: Date of dissertation registration: Key words: Scientific area: Specific scientific area: Doctoral study of electrical engineering Combined adaptive load balancing algorithm for parallelization of applications distributed computer systems, parallel processing, algorithms, load balancing algorithms, process scheduling, resource utilization, efficiency Computer sciences Distributed systems ABSTRACT Development and improvement of efficient techniques for distribution of parallel tasks on multiple processor cores is

one of the key issues encountered in parallel and distributed computer systems. The purpose of 16

process distribution improvement in parallel applications is increased system performance, reduced application execution time, reduced losses and increased resource utilization. This thesis presents parameters required for performance analysis of parallel applications, load balancing distribution algorithms and their advantages and

disadvantages. Thesis presents two

combined load balancing algorithms **based on domain decomposition and master-slave algorithms,** 14

as the research results. Algorithms were created with the purpose of reducing deficiencies in the distribution of parallel processes that include of several independent tasks. Selection of distribution algorithm, subject to segments with minimal losses, based on collected parameters and previously defined conditions, proved to deliver increased performances and reduced imbalance. Results of simulations confirm better performance of proposed algorithms compared to the standard algorithms reviewed in this paper. SADRŽAJ UVOD.....1 1. Distribuirani računarski sistemi i paralelno procesiranje .....5 1.1. Distribuirani računarski sistemi..... 5 1.1.1. Arhitektura distribuiranih sistema ..... 6 1.2. Paralelno procesiranje ..... 8 1.2.1. Distribuirani sistemi sa zajedničkom memorijom .....10 1.2.2. Višeprocesorski sistemi sa distribuiranom memorijom.....11 1.3. Metrika performansi paralelnih programa ..... 13 1.3.1. Vrijeme izvršavanja, ubrzanje i efikasnost paralelnih programa 13 1.3.2. Amdahl-ov zakon.....17 1.3.3. Skalabilnost paralelnih programa i Gustafson-Barsis-ov zakon 19 1.3.4. Neuravnoteženost opterećenja .....21 2. Algoritmi za raspodjelu opterećenja paralelnih aplikacija..... 23 2.1. Paralelni algoritmi ..... 24 2.1.1. Dizajn algoritama .....25 2.2. Raspodjela opterećenja paralelnih algoritama..... 26 2.2.1. Podjela i balansiranje opterecénja.....28 2.3. Klasifikacija load balancing algoritama ..... 29 2.3.1. Statički algoritmi raspodjele .....31 2.3.2. Dinamički algoritmi raspodjele .....33 2.3.2.1. Adaptivni load balancing algoritmi ..... 37 2.3.3. Primjeri dinamičkih algoritama.....39 2.3.3.1. Algoritmi pokrenuti od strane predajnika..... 39 2.3.3.2. Algoritmi inicirani od strane prijemnika ..... 41 2.3.3.3. Simetrično inicirani algoritmi..... 42 3. Domain decomposition i Master-slave Load balancing algoritmi 44 3.1. Dekompozicija domena (DD) ..... 44 3.1.1. Algoritam dekompozicije domena.....46 3.2 Master-slave algoritam (MS)..... 48 4. Kombinovani load balancing algoritmi ..... 52 4.1. Kombinovani load balancing algoritam (CA) ..... 52 4.2 Kombinovani adaptivni load balancing algoritam (CAA) ..... 55 5. Uporedna analiza performansi predloženih load balancing algoritama 63 5.1. Testno okruženje..... 63 5.1.1. Paralelni simulator performansi krosbar komutatora (CQ) paketa 63 5.1.2. Testni resursi .....65 5.2 Analiza performansi paralelizacije DD algoritmom..... 67 5.3. Analiza performansi kombinovanog algoritma ..... 72 5.4. Analiza performansi kombinovanog adaptivnog algoritma..... 77 ZAKLjUČAK ..... 84 LITERATURA ..... 86 POPIS SLIKA Slika 1.

Ubrzanje paralelnog programa ..... 16 Slika 2. Ubrzanje shodno Amdahlovom zakonu..... 18 Slika 3. Ubrzanje po Gustafson-Barsis-ovom zakonu..... 21 Slika 4. Primjer distribucije vremena izvršavanja po jezgrima ..... 22 Slika 5. Podjela load balancing algoritama..... 30 Slika 6. Skica dekompozicije domena u kome se domen ulaznih podataka dijeli na MxN poddomena koji se zatim dodjeljuju računarskim resursima ..... 45 Slika 7. Distribucije vremena izvršavanja - loš balans opterećenja ..... 47 Slika 8. Distribucije vremena izvršavanja - dobar balans opterećenja..... 47 Slika 9. Raspodjela procesa kod master-slave algoritma..... 49 Slika 10. Dijagram izvršavanja kombinovanog algoritma..... 54 Slika 11. Faze izvršavanja kombinovanog adaptivnog load balancing algoritma.... 56 Slika 12. Shema algoritma na osnovu koga se vrši odabir u toku druge faze ..... 60 Slika 13. Vrijeme izvršavanja paralelnog CQ simulatora uz DD algoritam za 16,32 i 64 porta i 100.000 vremenskih slotova ..... 68 Slika 14. Ubrzanje paralelnog CQ simulatora uz DD algoritam za 16, 32 i 64 porta i 100.000 vremenskih slotova..... 68 Slika 15. Efikasnost paralelnog CQ simulatora uz DD algoritam za 16, 32 i 64 porta i 100.000 vremenskih slotova..... 69 Slika 16. Vrijeme izvršavanja paralelnog simulatora komutatora

70 Slika 17.

70 Slika 18. Efikasnost

71 Slika 19. Prosječno vrijeme izvršavanja paralelne simulacije za razmatrane algoritme..... 72 Slika 20. Dužina trajanja zadataka po svakom jezgru prilikom izvršavanja simulacije koristeći DD algoritam..... 75 Slika 21. Dužina trajanja zadataka po svakom jezgru prilikom izvršavanja simulacije koristeći MS algoritam..... 75 Slika 22. Distribucija

izvršenja zadataka po jezgrima i po fazama prilikom izvršavanja simulacije koristeći CA algoritam..... 76 Slika 23. Trajanje pojedinačnih zadataka CQ simulatora ..... 78 Slika 24. Prosječno vrijeme izvršenja simulacija pomocú DD, MS i CAA algoritma na 16-128 jezgara..... 79 Slika 25. Uštede tokom izvršavanja alogitama i poredjenje izmedju kombinovanog algoritma i domain decomposition i master slave..... 81 Slika 26. Utrošeno vrijeme kombinovanog adaptivnog algoritma u zavisnosti od broja jezgara..... 82 Slika 27. Izabrani algoritam u trećoj fazi CAA ..... 83 POPIS TABELA Tabela 1. Vrijeme izvršenja paralelnog CQ simulatora na 64 jezgra ..... 73 Tabela 2. Vrijeme izvršavanja [s] DD, MS i CAA algoritama na 16-128 CPU jezgara ..... 79 POPIS AKRONIMA ASMP CA CAA CPU DD HPC IOT MIMD MISD MPI MS NUMA OpenMP POSIX RPC SISD SMP SPMD UMA Asymmetric multiprocessor Combined algorithm - Kombinovani algoritam Combined adaptive algorithm - Kombinovani adaptivni algoritam Central processing unit Domain decomposition algoritam High performance computing Internet of things

Multiple Instruction Multiple Data Multiple Instruction Single Data Message passing interface Master-slave algoritam **Nonuniform Memory Access** 12

Open Multi-Processing Portable Operating System Interface for Unix Remote Procedure call **Single Instruction** Single **Data** Symmetric multiprocessor **Single** Program Multiple **Data** 11

Uniform memory access UVOD Prvi digitalni računari izrađeni su sredinom prošlog vijeka sa zadatkom da ubrzaju kompleksne proračune u drugom svjetskom ratu [1], a kasnije olakšaju simulacije

u tehnici i nauci, da vrše **obradu velike količine podataka u** industriji, ekonomiji **i administraciji.** 5

Zahvaljujući pojavi mikroprocesora početkom 70-ih godina započela je računarska revolucija tj. napredak u tehnologiji proizvodnje računarskih elemenata, arhitekturi i performansama računara, kao i njihovoj primjeni. **Računari su** postali **manji,** 5

pouzdaniji i jeftiniji i imali su mogućnost obrade više podataka,

a oblast primjene se od proračuna i obrade podataka proširila na analizu i projektovanje procesa u tehnici, računarsku grafiku, prikupljanje i obradu mjerenja, upravljanje aparatima i procesima itd. 5

Ekspanzija umrežavanja uređaja dovela je do većeg korištenja distribuiranih računarskih resursa. Zahtjevnije operacije, kao što su kompleksni proračuni iz prirodnih nauka, analize velikih skupova podataka, simulacije u inženjeringu i industriji, se danas obavljaju metodama paralelnog procesiranja na distribuiranim računarskim resursima. Distribuirani računarski sistemi omogućavaju isporuku računarskih resursa neophodnih za rješavanje kompleksnih problema sa zahtjevima koji prevazilaze mogućnosti najsnažnijih personalnih računara. Super-računari ili računari visokih performansi (HPC), kao jedan od ključnih elemenata distribuiranih računarskih sistema, dovode do kompleksnih

rješenja pomoću računarskih simulacija omogućavajući napredak u praktično svim naučnim oblastima - od medicine i biologije, hemije, nauke o materijalima, inženjeringa, astrofizike, preko klimatologije i geologije, pa sve do ekonomije i društvenih nauka. 9

Računarske simulacije predstavljaju pažljivo dizajnirane programe koji se izvršavaju u cilju ubrzanja istraživačkog ili proizvodnog procesa ili slučajevima gdje su fizički eksperimenti skupi, opasni ili nemogući [2]. U cilju uštede vremena, mogućnosti bržeg i efikasnijeg izvršavanja kompleksnih simulacija koriste se metode paralelnog programiranja pomoću kojih više procesa i instrukcija izvršava istovremeno. Pri paralelnom procesiranju, istovremeno se koristi veći broj jezgara procesora koji uporedo rješavaju manje djelove zahtjevne aplikacije raspoređivanjem na više elemenata računarskog sistema podjelom problema ili podataka [3] [4]. Fokus mnogih istraživanja iz oblasti distribuiranih računarskih sistema i paralelnog procesiranja je pronalaženje optimalne raspodjele zadataka u cilju povećanja efikasnosti, smanjenja vrijemena izvršenja paralelnih aplikacija, smanjenja vremena komunikacije i povećanja procenta iskoristivosti računarskih resursa. Za postizanje što veće efikasnosti izvršavanja aplikacija presudno je optimizovati dodjelu zadataka djelovima distribuiranog računarskog sistema (nodovima računarskog klastera i jezgrima njihovih procesora) i nadgledati njihovo izvršenje. Postizanje paralelizma redistribucijom opterecénja paralelnih segmenata tokom izvršavanja paralelnog programa naziva se balansiranje opterecénja (load balancing) [5] [6]. Primarni cilj algoritama za balansiranje opterecénja je pronalaženje optimalnog rasporeda izvršavanja kojim se definiše početno vrijeme izvršavanja i redosled izvršavanja svih zadataka koji se pokreću na određenom resursu [7]. Load balancing paralelnih aplikacija se ostvaruje smanjenjem vremena komunikacije, vremena sinhronizacije između

procesa i vremena čekanja zbog neravnomjerne distribucije procesa. Disbalans paralelnih aplikacija se najčešće javlja usled neujednačenog opterećenja medju jezgrima, prevelike komunikacije među jezgrima ili dugotrajnog čekanja koje se pritom stvara [8]. U realnom distribuiranom okruženju

opterećenost resursa varira tokom vremena **i nije uvijek moguće** poboljšati **korištenje** resursa **koji su potpuno slobodni ili jednako opterećeni.** 1

Takođe, nije uvijek moguće odrediti ili predvidjeti dužinu trajanja procesa koji se izvršavaju na odvojenim računarima ili kašnjenja zbog komunikacije između računara, tako da dolazi do dužeg izvršenja paralelne aplikacije i pada iskoristivosti resursa. Kraj izvršavanja paralelne aplikacije ili početak faze postprocesiranja (postprocessing) direktno zavise od trajanje izvršavanja dijela aplikacije na jezgru kojem je dodijeljeno najviše procesa ili procesoru sa najnižim radnim taktom. Prvi load balancing algoritami unutar paralelnih aplikacija kreirani su prije više od četrdeset godina [9]. Neki algoritmi su razvijeni kao load balancing algoritmi opšte namjene [10], dok su neki specijalizovani za određeni vid paralelnih simulacija. Kao osnovu za odlučivanje, algoritmi mogu da koriste parametre infrastrukture kao što su karakteristike klastera i računarske mreže, opterećenje i zauzetosti resursa. Algoritmi balansiranja opterecénja, u zavisnosti od polisa i trenutka pokretanja samog algoritma, se mogu podijeliti na statičke i dinamičke. Kod naprednih dinamičkih algoritama, u zavisnosti od promjene opterećenja distribuiranog sistema tokom rada se može aktivirati adaptivna strategija unaprijed generisana šemom planiranja raspodjele zadataka. Predmet ovog istraživanja je potvrda hipoteze da se adekvatnom kombinacijom statičkih i dinamičkih load balancing algoritama može realizovati dodatno skraćenje izvršenja paralelne aplikacije uz povećanje iskoristivosti resursa i otklanjanje nedostataka postojećih algoritama. U tezi je predložen metod kombinovanog adaptivnog load balancing algoritma [11]. Analizirano je više verzija statičkih i dinamičkih load balancing algoritama, navedeni njihovi principi raspodjele procesa, prednosti i nedostaci i na osnovu pokazanih performansi izdvojeni domain decomposition i master slave algoritmi. Kombinacijom dva navedena algoritma, predložen je adaptivni algoritam balansiranja opterećenja, u cilju povećanja performasi i otklanjanja nedostataka. Predloženi algoritam vrši preraspodjelu zadataka u dijelu programa kada konstitutivni algoritmi bilježe pad performansi. Performasne predloženog algoritma verifikovane su na numerički zahtjevnom problemu, paralelnoj verziji aplikacije

simulatora performansi krosbar komutatora (CQ) **paketa sa baferima u ukrsnim tačkama komutacione matrice** 3

[12]. Doktorska disertacija je, nakon uvodnih razmatranja, izložena kroz pet poglavlja. U prvom poglavlju su predstavljeni distribuirani računarski sistemi i paralelno procesiranje, izloženi osnovni pojmovi i klasifikacija i navedena njihova primjena u savremenom svijetu. Posebna pažnja je posvećena parametrima na osnovu kojih se vrši analiza performasni distribuiranih sistema i koji su korišćeni za evaluaciju algoritama. U drugom poglavlju je izložena potreba za algoritmima za balansiranje opterećenja paralelnih aplikacija i njihova podjela. Izložena je analiza rada statičkih i dinamičkih algoritama i njihove prednosti i nedostaci. U narednom poglavlju su predstavljeni domain decomposition i master slave

algoritam, kao najčešće korišćeni algoritmi za statičku i dimaničku raspodjelu zadataka paralelnih aplikacije, karakteristike njihovog rada i njihove prednosti i nedostaci. U četvrtoj glavi je prezentovan princip rada novog kombinovanog algoritma i kombinovanog adaptivnog load balancing algoritma. U petoj glavi je prikazana komparativna analiza rezultata računarskih simulacija razmatranih algoritama. Na kraju rada su izloženi zaključci i spisak korišćene literature. Dio rezultata predstavljenih u ovoj tezi, pogotovo računarske simulacije predloženih algoritama, proistekli su i iz višegodišnjeg rada na projektu

High- Performance Computing Infrastructure for South East Europe's Research Communities (HP-SEE) 3

finansiranom od strane Evropske komisije u sklopu FP7 programa [13] . 1. Distribuirani računarski sistemi i paralelno procesiranje 1.1. Distribuirani računarski sistemi Distribuirani računarski sistemi su jedinstveni integrisani računarski sistemi koje čine skupovi autonomnih računarskih resursa koji komuniciraju međusobno u cilju izvršenja zajedničkih zadataka [2]. Oni omogućavaju isporuku računarskih resursa neophodnih za rješavanje kompleksnih problema sa zahtjevima koji prevazilaze mogućnosti njihovih konstitutivnih elemenata i obuhvataju širok opseg uređaja – od senzora do klastera visokih performansi (HPC), od kućnog računara do velikih poslovnih računarskih mreža. Njihova primjena se nalazi u svakodnevnom zivotu, poslu, nauci i industriji. Distribuirani sistemi se baziraju na dijeljenju hardverskih i softverskih resursa, modularnosti, mogućnostima korišćenja opreme različitih proizvođača, konkurentnog izvršavanja više procesa i velike otpornost na greške usled otkazivanje djelova sistema. Osnovi ciljevi prilikom projektovanja distribuiranih sistema su izrada sistema visokih performansi, pouzdanost sistema, skalabilnost, ujednačenost i sigurnost [14] [15].

Postoji više različitih tipova distribuiranih sistema, a kategorizacija se može izvršiti **na osnovu** 17

komunikacije među resursima, načinima kontrole i upravljanja distribuiranim resursima i geografske raspodjele. Elementi sistema međusobno komuniciraju razmjenom informacija putem računarske mreže. Klasifikacija distribuiranih sistema se može zasnivati na tipu arhitekture i mogućnostima hardware-a da istovremeno izvršava jedan ili više tipova instrukcija nad jednim ili više tokova podataka. Paralelni računarski sistemi, kao vrsta distribuiranih sistema, se sastoje od višeprocesorskih računara koji komuniciraju među sobom i podržavaju istovremeno (konkurentno) izvršavanje više procesa [16]. Skup višeprocesorkih računarskih nodova povezanih zajedničkim software-om (midlewarre-om) i računarskom mrežom naziva se računarski klaster i predstavlja osnovu za paralelno procesiranje [17]. Računarski klasteri predstavljaju osnovu modernih distribuiranih sistema kao što su HPC [18], Cloud [19] i Grid [20]. 1.1.1. Arhitektura distribuiranih sistema Flynn-ova taksonomija predstavlja osnovnu kategorizaciju arhitekture računara kod koje se računari klasifikuju shodno mogućnostima istovremene obrade jedne ili više instrukcija nad jednim ili više tokova podataka [21]. Taksonomija je razvijena 1966. godine [22], a njeno unapređenje i razrada 1972. godine [23]. Metodologija klasifikuje mogućnost paralelnih operacija na nivou procesora i definiše vrste paralelizama koji su podržani na

nivou hadrvera i dostupni aplikacijama. Predložena kategorizacija koristi koncepte toka instrukcija i podataka za kategorizaciju arhitekture skupa instrukcija i time izdvaja četiri kombinacije koje opisuju najpoznatije paralelne arhitekture [24]: • SISD

(Single Instruction Single Data) - Jednostruki tok instrukcija i jednostruki **tok podataka:** 2

SISD model procesora izvršava jedan tok instrukcija i nad jednim tokom podataka. Na ovoj arhitekturi se zasnivaju jednoprocesorski računari kod kojih se instrukcije izvršavaju sekvencijalno. Na SISD modelu se baziraju sekvencijalni računari po von Neumann-ovom modelu [25]. • SIMD (Single Instruction Multiple Data) - Jednostruki tok instrukcija i višestruki tok podataka: Na SIMD modelu se baziraju računari u kojima svaki od identičnih procesora izvršava isti program nad svojim lokalnim podacima.

Izvršavanje je sinhrono i **svaki procesor u isto** vrijeme **izvršava istu instrukciju za drugačiji** set **podataka.** 2

Računari bazirani

na SIMD modelu su korisni **za rad u** simulacijama **u kojima se često koriste operacije nad vektorima i matricama.** 2

• MISD

(Multiple Instruction Single Data) - Višestruki tok instrukcija i jednostruki **tok podataka:** 2

MISD je model višeprocesorskog sistema gdje više procesora dijele isti tok podataka. Svi procesori istovremeno obrađuju

jedan podatak prihvaćen iz zajedničke memorije prema instrukciji koju svaki procesor prima sa svoje upravljačke jedinice. 2

Računari ovakvog tipa se koriste za izvršavanje različitih kriptografskih algoritama, filtere za izdvajanje signala iz ulaznog niza podataka itd. • MIMD (Multiple Instruction Multiple Data) - Višestruki tok instrukcija i višestruki tok podataka: U MIMD računarskim sistemima svaki procesor posjeduje lokalnu memoriju, a računari su sposobni

da izvrše više instrukcija nad različitim skupom podataka. Instrukcije **se izvršavaju** 2

asinhrono, tj.

u istom trenutku mogu izvršavati različite instrukcije nad različitim podacima. 2

U

kategoriju MIMD modela spada najveći broj današnjih super-računara. Prednosti **MIMD modela u odnosu na ostale modele iz Flinove** podjele **su** 2

izvršavanje više različitih instrukcija u isto vreme i mogućnost nezavisnog izvršavanja operacija svakog procesora. Današnji paralelni računari se obično zasnivaju na MIMD arhitekturi ili kombinaciji SIMD i MIMD arhitektura [26] [27]. Paralelni računari bazirani na MIMD platformi su jeftiniji i dostupniji od specijalizovanih računara baziranih na SIMD platformi. Zbog zastupljenosti MIMD platforme, E. E. Johnson je 1988. godine predložio detaljniju klasifikaciju [28] mašina zasnovanih na njihovoj memorijskoj strukturi (globalnoj ili distribuiranoj) i mehanizmu koji se koristi za komunikaciju i sinhronizaciju između procesora i jezgara koristeći dijeljenu memoriju ili slanje poruka, a koja nije razmatrana prilikom Flynn-ove taksonomije. 1.2. Paralelno procesiranje Paralelno procesiranje predstavlja vid obrade podataka pri kome se više procesa i instrukcija izvršava istovremeno [3]. Zahtjevniji problemi se dijele na manje instrukcije koje se rješavaju uporedno na različitim procesorima. Glavni razlozi koji favorizuju korištenje paralelnog procesiranja su ušteda vremena, mogućnosti efikasnog izvršavanja kompleksnih simulacija, izvršavanja više problema istovremeno, korišćenja distribuiranih slobodnih resursa putem mreže i ušteda troškova. Uslijed potrebe za računarskim resursima koju personalni računari i radne stanice ne mogu da pruže, sve češća je upotreba distribuiranih i paralelnih računarskih sistema i softvera koja traženu računarsku snagu čini dostupnom. Potreba za bržim rješenjima i izvršavanju vecíh problema u što kraćem vremenskom intervalu se javlja u širokom spektru aplikacija. Neke od oblasti gdje se učestalo korsti paralelno procesiranje su vremenska prognoza, modeliranje i simulacija velikih sistema,

obrada informacija i ekstrakcija podataka, obrada slike i videa i vještačka inteligencija [29]. Računarski programi su prvenstveno osmišljeni za izvršavanje serijskih programa, koji za rješavanje određenog problema koriste algoritme koji se sastoje od seta instrukcija koje se sekvencijalno izvršavaju. S druge strane, pri paralelnom računanju istovremeno se koristi veći broj procesorskih jedinica koji uporedo rješavaju dodijeljene zadatke na više elemenata računarskog sistema podjelom problema ili podataka na niz manjih segmenata [4]. Paralelno procesiranje se uglavnom koristi na računarima visokih performansi, a interesovanje za njega je poslednjih godina poraslo usled fizičkih ograničenja koja sprečavaju drastično povećanje frekfencije procesora i povećanje potrošnje energije [30]. Fizički limiti tehnologije poluprovodnika su skoro u potpunosti dostignuti, tako da se pažnja sve više poklanja razvoju višejezgarnih procesora i distribuiranih sistema i primjene paralelnog procesiranja na njima.

Razvoj procesorskih čipova u periodu od 70tih **godina** prošlog vijeka **može** se **opisati Murovim zakonom** [31] **koji tvrdi da se broj tranzistora** 2 **po jedinici površine duplira na svakih 18 do 24** mjeseca. **Bez obzira na završetak razvoja** jednoprocesorskih **čipova Murov zakon je još** uvijek **na snazi**

i to isključivo zahvaljujući razvoju paralelnih računarskih sistema. Danas proizvođači brzinu računara uvećavaju dodavanjem dodatnih procesora ili jezgara u sklopu mikroprocesorskog paketa, umjesto poboljšanja povećanja radnog takta procesora jer se time uzrokuje visoka temperatura prilikom rada računara. Izvršavanje paralelnih programa na višeprocesorskim sistemima mora osigurati bolje performanse nego pri izvršavanju sekvencijalnih programa na jednoprocesorskom sistemu. Paralelni program mora biti tačan, riješiti složeni problem i biti značajno brži od sekvencijalnog. Da bi se osigurala brzina koju podrazumijeva upotreba paralelnih i višeprocesorskih sistema, aplikacija se mora podijeliti tako da svaki procesor istovremeno izvršava približno jednaku količinu posla, pritom pazeći da kašnjenja usled raspodjele zadataka i koordinacije procesa ne smanjuju performanse programa. Jedan od osnovnih razloga za korištenje paralelnog računarstva je smanjivanje vremena izvršavanja zahtjevnih aplikacija. Mnoga industrijska i naučna istraživanja uključuju složene i zahtjevne proračune koji bi bez paralelnih računara trajali danima ili duži vremenski period. Za određene simulacije zakasnili rezultati često predstavljaju beskorisne rezultate. Tipičan primjer je vremenska prognoza koju karakterišu kompleksni proračuni nad velikim skupovima podataka. Paralelni računari se takođe koriste u mnogim oblastima kako bi se postigla vecá tačnost proračuna ili obuhvatio veći skup ulaznih podataka. Pri numeričkim simulacijama, dok serijski računar može raditi na jednom analiziranom području, paralelni računar sa N jezgara može obraditi N područja ili postići N puta veću rezoluciju na istoj analiziranoj oblasti. Vecá rezolucija može pomocí smanjenju grešaka koje su neizbježne u numeričkim proračunima. Podjela paralelnih računara najčešće se može vršiti prema fizičkoj konfiguraciji sistema, odnosno prema nivou na kome se podržava paralelizam. Razlikujemo višeprocesorske i višejezgarne računarske sisteme koji raspolažu s nekoliko procesnih elemenata unutar istog računara, dok klasteri i grid koriste više računara za izvršavanje istog zadatka. Sa stanovišta hardware-a, paralelni računari se po Flynn-u baziraju na MIMD arhitekturi ili hibridu SIMD i MIMD arhitektura. U praksi se podjela najčešće bazira na vrsti komunikacije među procesima, odnosno tipu memorije putem koje procesi komuniciraju. Na osnovu organizacije memorije, paralelni računari se mogu podijeliti na računare sa dijeljenom memorijom i računare sa distribuiranom memorijom, kao i računari sa hibridnom

organizacijom koje objedinjavaju dva navedena tipa. U praksi su često zastupljeni računarski sistemi sa hibridnom memorijom koja kombinuje prednosti oba navedena modela [7] [32]. Na osnovu organizacije memorije računarskog sistema i načina komunikacije među procesima razlikuje se i više načina paralelizacije programa koji se na njima izvršavaju. 1.2.1. Distribuirani sistemi sa zajedničkom memorijom Distribuirani sistemi sa zajedničkom memorijom, poznati još i kao sistemi s ravnomjernim pristupom memoriji UMA (Uniform Memory Access), su vrlo popularni zbog svog jednostavnog programerskog modela koji omogućuje brzi razvoj paralelnog softvera s podrškom dijeljenja koda i podataka [33]. Svi procesori u arhitekturi sa zajedničkom memorijom mogu pristupati istom adresnom prostoru zajedničke memorije koja se koristi kao sredstvo komunikacije između procesora. U višeprocesorskom sistemu sa zajedničkom memorijom, svaki procesor može pristupiti bilo kom memorijskom modulu.

Komunikacija između procesa ostvaruje **se upisom i čitanjem podataka iz određenih memorijskih lokacija zajedničkog adresnog prostora.**	4
Prednost ovih sistema je u brzini izvršavanja pošto se komunikacija zasniva na upisu i čitanju podataka iz radne memorije.	4
Sa druge strane, predstavljeni **sistemi su ograničeni, tj. uslovljeni izvršavanjem na** jednom računaru, **dok**	4

računarski sistemi sa distribuiranom memorijom nemaju ta ograničenja. Nedostatak ove implementacije predstavlja i postojanje samo jednog pristupnog porta i zbog toga je u svakom trenutku moguće pristupiti samo jednom podatku. Razvijanje paralelnih programa za računare sa dijeljenom memorijom je jednostavnije nego kod programa za računare sa distribuiranom memorijom i neznatno se razlikuje od razvoja serijskih aplikacija [34]. Operacije komunikacije, odnosno upravljanja memorijom, obavljaju pomoćne biblioteke od kojih su najpopularnije OpenMP [35] i POSIX [36]. Distribuirani sistem sa dijeljenom memorijom je najčešće sastavljen od identičnih, homogenih procesora i glavne memorije povezane sa svim dostupnim procesorima, pa se ovaj model višeprocesorskih sistema često naziva i simetričnim višeprocesorskim sistemom (engl. symmetric (shared memory) multiprocessors – SMP) i trenutno je najpopularniji model višeprocesorskih sistema sa širokom primjenom u višejezgarnim procesorima. 1.2.2. Višeprocesorski sistemi sa distribuiranom memorijom U višeprocesorskim sistemima sa distribuiranom memorijom svaki memorijski modul je povezan sa procesorom. Procesori mogu direktno pristupiti svojim vlastitim memorijama, dok se za komunikaciju i pristup memorijskim modulima ostalih procesora koriste mehanizmi razmjene poruka.

Tipovi poruka uključuju pozive funkcija (RPC – Remote Procedure call), signale i razmjenu paketa **podataka. Svi sistemi distribuiranih objekata i RPC poziva kao što su Corba, Java RMI, SOAP, .NET Remoting, ONC RPC i slični su bazirani na modelu prenosa poruka.** 4

Pored navedenih, u naučnim simulacijama se često koristi komunikacijski protokol za prenos poruka Message passing interface (MPI). Pristup memoriji kod ovakvih sistema nije ravnomjeran jer memorija nije centralizovana, pa se ovi sistemi još nazivaju i višeprocesorskim sistemima sa neravnomjernim pristupom memoriji, tj. NUMA (Nonuniform Memory Access) multiprocesorski sistemi. Ukoliko je NUMA višeprocesorski sistem sastavljen od identičnih procesora, tada kažemo da se radi o homogenom ili simetričnom višeprocesorskom sistemu (symmetric multiprocessor, SMP), kao što je bio slučaj kod UMA sistema. Suprotno tome, ako sistem sačinjavaju različiti, tj. heterogeni procesori, tada se radi o asimetričnom višeprocesorskom sistemu, tj. ASMP-u (engl. asymmetric multiprocessor). Model NUMA najčešće se koristi u paralelnim sistemima s vrlo velikim brojem procesora, gdje se memorija decentralizuje i fizički raspodjeljuje na procesore, kako bi se zbog velikog broja procesora osigurala veća propusnost i što manja memorijska latentnost. MPI [37] predstavlja jedan od najrasprostranjenijih modela za razvoj paralelnih aplikacija za i predstavlja

standard za komunikaciju među procesima pri **paralelnom programiranju sa distribuiranim memorijskim modelom.** 4

Za specifikaciju MPI modela odgovoran je MPI forum, otvorena grupa koja ima članove iz velikog broja drugih organizacija. 1

MPI se najčešće primjenjuje u računarskim klasterima i super-računarima. **Po programskom modelu spada u modele prenosa poruka, a prema arhitekturi sistema je namenjen MIMD sistemima. MPI** 1

pruža

visoke performanse, dobru skalabilnost i prenosivost. MPI se u vrijeme **pisanja izdvaja kao dominantan model široko prihvaćen i korišćen u računarskim sistemima visokih performansi. Zvanični jezici interfejs standarda su Fortran, C i C++, a MPI implementacije se mogu naći i za C#, Java, Python, Perl i druge jezike** 1

[7]. 1.3. Metrika performansi paralelnih programa Glavni razlog paralelizacije programa predstavlja povećanje performansi programa i smanjenje vremena izvršavanja, kao najvažnije metrike, koristeći distribuirane resurse. Vrijeme izvršavanja paralelnog programa zavisi od mnogih faktora, uključujući arhitekturu platforme na kojoj se program izvršava, kompajler, operativni sistem, paralelni programski model na kojem se bazira itd. Pored vremena izvršavanja, u paralelnim aplikacijama se mogu definisati i ubrzanje, efikasnost i skalabilnost, kojima se mjeri stepen iskoristivosti pri izvršavanju na različitim distribuiranim resursima. Vremenska analiza izvršavanja paralelnog programa daje informaciju o maksimalnim performansama prilikom paralelizacije,

kao i o perfromansama programa **prilikom** promjene **broja procesorskih** jezgara **na kojima se izvršava.** 2

Predviđanje vremena izvršavanja paralelnog programa pomaže utvrđivanju da li se povećanjem broja jezgara postiže ubrzanje ili usporenje, kao i do koje granice je opravdano povećanje broja jezgara da bi se dobila opravdana efikasnost paralelne aplikacije. Koristeći Amdahl-ov [38] i Gustafson-Barsis-ov [39] zakon moguće je predvidjeti maksimalno ubrzanje i odrediti svrsishodnost paralelizacije. 1.3.1. Vrijeme izvršavanja, ubrzanje i efikasnost paralelnih programa Osnovni kriterijum za mjerenje korisnosti paralelnog programa je vrijeme njegovog izvršavanja na distribuiranom računarskom sistemu. Paralelno vrijeme izvršavanja T(n) programa predstavlja vrijeme između početka programa i kraja izvršenja na svim procesorima koji učestvuju, odnosno trenutka kada poslednji procesor završi izvršavanje paralelnog programa. U zavisnosti od arhitekture platforme za izvršavanje, paralelno izvršenje obuhvata sledecá vremena: • vrijeme izvršavanja lokalnih izračunavanja svakog učesnika procesa - izračunavanja koje svaki procesor obavlja korišćénjem podataka u svojoj lokalnoj memoriji, • • vrijeme utrošeno za komunikaciju i razmjenu podataka između procesora, vrijeme za sinhronizaciju procesora koji učestvuju prilikom pristupa dijeljenim strukturama podataka u slučaju dijeljene memorije, • vrijeme čekanja koja se javljaju zbog nejednake distribucije opterecénja procesora i • vrijeme potrošeno na dodatne operacije koje se ne izvršavaju kod sekvencijalnih programa npr. raspodjela ulaznih podataka po jezgrima [40]. Vrijeme provedeno za razmjenu podataka i sinhronizaciju, kao i vrijeme čekanja, mogu se smatrati nepotrebnim jer ne doprinose direktnom izračunavanju koje treba izvršiti i potrebno ih je smanjiti u što većoj mjeri [41]. Za analizu performansi paralelnog programa i benifita koje ono pruža koristi se upoređivanje vremena izvršenja paralelnog sekvencijalnog programa. Predviđanje ubrzavanja dobijenog paralelizacijom je važan faktor pri konverziji serijskih aplikacija u paralelne [42]. Ubrzanje S(n) paralelnog programa sa paralelnim vremenom izvršenja T(n) je

definisano kao $*S?(n?) = T? T?(n?)$, (1) gde n predstavlja broj procesora na kojima se izvršava paralelni program, a T* vrijeme izvršenja najbrže sekvencijalne implementacije. Ubrzanje paralelnog programa izražava relativnu uštedu vremena izvršenja koje se može dobiti primjenom paralelnog izvršenja na n procesora u poređenju sa najbržom sekvencijalnom implementacijom. Pojam ubrzanja se koristi i za teoretsku analizu algoritama i praktičnu evaluaciju paralelnih programa. Kako vrijeme izvršavanja paralelnog programa T(n) obuhvata vrijeme izvršavanja simulacije Tsim(n), vrijeme komunikacije između nodova Tcomm[n], sinhronizaciju između procesa Tsync[n], i vrijeme potrošeno na dodatne operacije Texop[n], ona sva utiču na ubrzanje. Stoga se ubrzanje računa po formuli $*S$

?(n?) = T? Tsim (n) + T?T?T?T?T? T ?(n?) + T? T ?T? T?T ?(n?) + T?T?T?T?T?T?(n?) . (2) 10

Da bi se dobilo veće ubrzanje potrebno je smanjiti sva navedena vremena izvršavanja aplikacije. Vrijeme potrošeno na komunikaciju među procesima se može korigovati unapređenjem računarskog klastera, tj. nodova i računarske mreže koja ih povezuje. Vrijeme sinhronizacije se može korigovati tehnikama raspodjele procesa i opterećenja. Da bi se postiglo što veće ubrzanje i ostvarili benifiti paralelne aplikacije moraju se razmotriti kompromisi između vremena dodatnih operacija, sinhronizacije i komunikacije. Definicija ubrzanja zahtijeva poređenje sa najbržim sekvencijalnim algoritmom i on se u nekim slučajevima teško može odrediti ili definisati. U nekim slučajevima najbolji sekvencijalni algoritam nije poznat jer se pretpostavlja da se optimizacijom određenih procesa može doći do kraćeg vremena izvršavanja. Takođe, postoje programi kod kojih najkraće vrijeme izvršavanja zavisi od veličine ulaznih podataka i postupka koji se za njihovu analizu koristiti. Zbog navedenih razloga, brzina se često izračunava korišćénjem sekvencijalne verzije paralelne implementacije umesto najboljeg sekvencijalnog algoritma [7] [40]. U teoriji važi S(n)≤n, jer bi za S(n)>n bilo neophodno uraditi novi sekvencijalni algoritam koji se koristi za izračunavanje ubrzanja. U praksi se ponekad može dostići superlinearno ubrzanje, tj. S(n)>n, optimizovanom podjelom podataka koji se izvršavaju na odvojenim jezgrima i korištenjem keš memorije. Superlinearno ubrzanje se može desiti i kada su operacije serijskog algoritma zahtjevnije od njegove paralelne formulacije zbog hardverskih karakteristika koje degradiraju serijsku implementaciju usled upotrebe sporijih elemenata memorije [34] . Na vrijeme izvršavanja paralelne aplikacije negativno utiče vrijeme provedeno za razmjenu podataka i sinhronizaciju i vrijeme čekanja koje se povećava rastom broja angažovanih procesora. Ovaj rast utiče na smanjenje ubrzanja, tako da rastom broja procesora može doći do zasićenja, odnosno stagnacije, ili čak i do pojave smajenja ubrzanja, prikazano na slici Slika 1. Tada je potrebno smanjiti broj procesora na kojima se program izvršava i na drugi način pokušati unaprijediti performanse programa. Slika 1. Ubrzanje paralelnog programa Efikasnost

paralelnog programa prestavlja mjeru iskoristivosti procesora. Računa se kao odnos ubrzanja S(n) koje se postiže i broja procesora n koji se koristi: 2

$E?(n?)$

= $S?(n?)$. (3) U idealnom paralelnom sistemu, ubrzanje je jednako p, a efikasnost je $n?$ jednaka jedinici. U praksi, ubrzanje je obično manje od p, a efikasnost je između nule i jedinice. U praksi su generalno prihvatljivi paralelni algoritmi sa efikasnošću većom od 50% [43] [44]. 1.3.2. Amdahl-ov zakon Amdahlov zakon definiše

ubrzanje prilikom izvršavanja paralelnih simulacija **upotrebom većeg broja procesora** i **tvrdi da je ubrzanje programa na višejezgarnim i višeprocesornim sistemima ograničeno njegovim sekvencijalnim** 2

dijelom [38] [45]. Koristi se za teorijsko određivanje maksimalnog ubrzanja paralelnog programa koje zavisi od broj procesora na kojima se izvršava program. Zakon određuje gornju granicu ubrzanja koji se može dobiti paralelizacijom. Ograničenja paralelizacije proističu od zavisnosti podataka i procesa unutar algoritma koji se implementiraju i iz djelova programa koji se moraju izvršiti sekvencijalno. Ako se pretpostavi da se paralelni algoritam sastoji od paralelnog dijela f i serijskog dijela 1-f, vrijeme potrebno za izvršavanje ove aplikacije na jednom jezgru se može predstaviti kao gdje su: $T?p?(1) = n?(1 - f?)\tau?p? + n?n?\tau?p? = n?\tau?p?$, (4) — vrijeme izvršavanja aplikacije algoritma na jednom CPU jezgru, $\tau?p?$— broj jezgara, $n?$ — vrijeme izvršavanja serijskog dijela algoritma i $n?(1 --f?)v\tau?rp?$ijeme izvršavanja paralelnog dijela algoritma. $n?n?\tau?p?$ Ukoliko bi isti zadatak bio izvršen na n paralelnih procesora, vrijeme izvršavanja se može definisati kao $T?p?(n?) = n?(1 - f?)\tau?p? + f?\tau?p?$, (5) pošto je paralelni dio distribuiran na n procesora. Tada se ubrzanje po Amdahl- ovom zakonu za n procesora može izračunati kao S

?(n?) = T? p **?(n?)** (1 − $f?)n? + f?$ **$T?p?(1)$** = $n?$ = **1 $f?$** 15

. (6) (1 − $f?$) + $n?$ Stoga je ubrzanje S(n)=1 za kompletno serijski kod (f=0), odnosno S(n)=n za kompletno paralelan kod (f=1). Da bi bilo dobijeno veće ubrzanje potrebno je da bude 1 − $f? \ll f? n?$, (7) odnosno f mora biti približno jednako 1 prilikom izvršavanja na velikom broju procesora. Za velike vrijednosti jezgara n može se aproksimirati $S?(n?) = 1$ (1 − $f?$) (8) Slika 2. Ubrzanje shodno Amdahlovom zakonu Ubrzanje po Amdahl-ovom zakonu u zavisnosti od paralelnog dijela f za različit broj procesora N prikazan je na slici 2. Puna linija prikazuje grafikon za f = 0,99; isprekidana linija je za f = 0,9; a tačkasta linija je za f = 0.5. Primjetno je značajno smanjenje brzine povećanjem broja procesora N, pogotovo za manje vrijednosti paralelnog dijela f (<0,9). Za male vrjednosti f se može desiti zasićenje ubrzanja usled povećanja broja jezgara. Shodno navedenom, serijski djelovi programa se moraju svesti na minimum prilikom projektovanja programa, analize ubrzanja i izvršavanja na velikom broju procesora. Iz svega navedenog nameće se zaključak da višeprocesorski sistemi, a pogotovo oni sa vrlo velikim brojem procesora, mogu pokazati ubrzanje i poboljšanje performansi tek ukoliko se iskoriste i primijene tehnike paralelizma unutar programa i aplikacija. Takođe, iz zakona se može zaključiti da je ubrzanje ograničeno paralelnim dijelom f, čak i kad se broj procesora N rapidno povećava. Osim navedenog, Amdahlov zakon kao osnovni cilj paralelizacije uzima minimizaciju vremena izvršavanja, te stoga zanemaruje vrijeme komunikacije između procesa i ostale

faktore koje mogu uticati na ubrzanje. 1.3.3. Skalabilnost paralelnih programa i Gustafson- Barsis-ov zakon Skalabilnost definiše mogućnost poboljšanja performansi paralelnog programa proporcionalno broju korištenih procesora. Povecánje količine ulaznih podataka kod fiksnog broja jezgara obično dovodi do povecánja ubrzanja, dok za fiksnu veličinu ulaznih podataka povećanje broja jezgara procesora može dovesti do zasicénja ili smanjenja ubrzanja i efikasnosti. U tom smislu, skalabilnost predstavlja osobinu paralelne implementacije da efikasnost održi konstantnom ako se povecáju i broj procesora i veličina problema. Stoga, skalabilnost predstavlja važnu osobinu paralelnih programa koja pokazuje da se vecí problemi mogu riješiti na isti način kao i manji ukoliko izvršavaju na dovoljno velikom broju procesora [7]. Povecánje ubrzanja usled povecánja veličine problema se ne može objasniti Amdahlovim zakonom. Umjesto toga koristi se verzija Amdahlovog zakona koja pretpostavlja da sekvencijalni programski dio nije konstantni dio f ukupne aplikacije, već zavisi i od ulaznih podataka. Gustafson [39] je dokazao da se paralelizam u aplikacijama povecáva povećanjem veličine problema, dok je Amdahlov zakon pretpostavio da je dio paralelnog koda fiksan i da ne zavisi od veličine problema. Po Gustafson-Barsis-ovom zakonu, vrijeme izvršavanja na n procesora je $T?p?(N?) = (1 - f?)\tau?p? + f?\tau?p? = \tau?p?$. Vrijeme sekvencijalne aplikacije može izraziti kao (9) $T?p?(1) = (1 - f?)\tau?p? + N?N?\tau?p?$. (10) Tada je ubrzanje T

$?p?(1) \quad S \quad ?(N?) = T \quad ?p?(N?) = (1 \quad - f?) + \quad N?N? = 1 + \quad (N? - 1)$ 18

$f?$. (11) Za postizanje ubrzanja potrebno je da bude $(N? - 1)f? \gg 1$. (12) Shodno ovom zakonu, značajna ubrzanja su moguća za male vrijednosti f, pogotovo pri izvršavanju na velikom broju procesora. Slika 3. Ubrzanje po Gustafson-Barsis-ovom zakonu Slika 3. prikazuje brzinu u skladu sa paralelnim dijelom f za različite vrednosti N. Puna linija prikazuje ubrzanje za f = 0,99; isprekidana linija je za f = 0,9; a tačkasta linija je za f = 0.5. Može se primijetiti da je ubrzanje za velike vrijednosti f skoro linearno. Primjetan je i linearan rast ubrzanja za vrlo male vrijednosti f usled povećanja broja procesora N. Gustafson-Barsisov zakon dopunjuje Amdahlov zakon koji se zasniva na pretpostavci fiksne veličine problema, tj. na obimu ulaznih podataka za paralelnu obradu koji se ne mijenja u odnosu na povećanje broja jezgara. Gustafsonov zakon redefinira efikasnost, zbog mogucńosti da ograničenja nametnuta sekvencijalnim dijelom programa mogu biti suprotna povecánjem ukupne količine računanja. Gustafsonov zakon takodje pokazuje da se zahtjevniji problemi mogu riješti u isto vrijeme kao jednostavniji uz pomoć korištenja bržeg hardware-a, odnosno da ubrzanje treba mjeriti skaliranjem problema na broj procesora, a ne određivanjem veličine problema. Uopšteno gledano, Gustafson-Barsisov zakon nudi mnogo optimističnije predviđanje ubrzanja paralelnog sistema od Amdahlovog zakona. Amdahl-ov i Gustavson-Baris-ov zakon ignorišu troškove komunikacije, pa je zbog toga ubrzanje paralelne aplikacije dobijeno u praksi nešto niže izračunatog [46]. 1.3.4. Neuravnoteženost opterećenja Tokom izvršavanja paralelnog programa može doći do neuravnoteženosti zbog nejednakog opterećenja računarkog klastera na kojima se paralelni program izvršava ili loše raspodjele zadataka unutar paralelne aplikacije. Usled toga može doći do kraja dodjeljenih zadataka procesorima u različito vrijeme kada oni prestaju sa obradom podataka i čekaju dalje instrukcije, šta rezultuje smanjenom efikasnošću različitih procesora i time i paralelnog programa u cjelini. Da bi se smanjila neuravnoteženost opterecénja potrebno je sve zadatke rasporediti što ravnomjernije, tako da svi zadaci završavaju u približno isto vrijeme i čime bi se minimiziralo vrijeme neaktivnosti procesora. Minimiziranje disbalansa opterećenja ključna je aktivnost u izradi efikasnih paralelnih programa na distribuiranim sistemima. Slika 4.

Primjer distribucije vremena izvršavanja po jezgrima Na slici 4. je prikazan primjer neuravnoteženog paralelnog programa kod kojeg CPU jezgra završavaju dodijeljene zadatke za različito vrijeme Ti. Efikasnost programa je maksimalna do vremena Tmin kada najbrže jezgro završava dodijeljene zadatke. Od ovog momenta, najbrže jezgro ili grupa jezgara su u stanju čekanja na nove zadatke ili kraj kompletnog izvršavanja, što dovodi do gubitaka, sve do Tmax trenutka kada program završi svoj rad. Sa Tavg je označeno prosječno vrijeme izvršavanja programa, tj. vrijeme kada bi paralelni program završio izvršavanje dodijeljenih zadataka u idealnom slučaju. Prosječno vrijeme izvršavanja Tavg se može izračunati ukoliko je poznato vrijeme izvršavanja na svakom jezgru Ti i iznosi $T?a?a?a?$ = $\sum i?=0$ $T?i?$ $N?-1$ $N?$, (13) gdje je N broj jezgara na kojima se izvršava paralelna aplikacija. Kao mjera neuravnoteženosti definiše se faktor neuravnoteženosti opterećenja (load balance ratio), kao jedna od mjera performansi, i računa se na sledeći način: $N?-1$ $I?(N?)$ =

$T?m?m?m? - \sum i?=$ 0 $T?i?$ $T?m?$ $m?m?$ $\sum N$ $?i?=-$ 19

01 $T?i?$ = $T?a?a?a?$ − 1 . (14) Shodno navedenoj formuli, gubici i faktor neuravnoteženosti su manji ukoliko je vrijeme kraja rada svih CPU jezgara približno jednako, odnosno mala razlika izmedju Tmax i Tavg. Za idealno raspodijeljeno opterećenje važi Tmax = Tavg, odnosno I=0. Neuravnoteženost sistema uzrokuje gubitke i degradaciju kompletnog paralelnog programa. Da bi se smanjila neuravnoteženost opterećenja aplikacije i povećala efikasnost programa primjenjuju se tehnike preraspodjele opterećenja paralelnih aplikacija što je predmet istraživanja ove teze. 2. Algoritmi za raspodjelu opterećenja paralelnih aplikacija Razvoj i unapređenje tehnika za dizajn paralelnih algoritama, njihovu optimizaciju i efikasnu raspodjelu opterećenja paralelnih aplikacija predstavljaju glavne izazove u istraživanjima iz oblasti paralelnog programiranja i paralelnih računara. Cilj unapređenja raspodjele procesa predstavlja povećanje performansi sistema, smanjenje vremena izvršavanja, minimiziranje komunikacije i poboljšanja iskoristivosti sistema. Procedure za postizanje bolje efikasnosti preraspodjelom opterećenja segmenata aplikacije u toku trajanja izvršenja simulacije nazivaju se load balancing algoritmi ili algoritmi za raspodjelu opterećenja. 2.1. Paralelni algoritmi Algoritmi predstavljaju precizno definisan postupak za rješavanje nekog računarskog problema u konačnom broju koraka. Mogu se definisati i kao niz računarskih koraka koji povezuju ulazne i izlazne skupove podataka, zasnovanih na sprovođenju niza određenih akcija [47] [48]. Paralelno računarstvo se bazira na istovremenom korišćenju više računarskih resursa u cilju rješavanja nekog problema [4]. Shodno tome, paralelni algoritmi se mogu definisati kao procedure pri kojima se dva ili više segmenata algoritma mogu izvršavati istovremeno na paralelnim ili viseprocesorskim računarskim sistemima. Dizajn paralelnog algoritma je povezan sa arhitekurom paralelnih sistema na kojima se aplikacija izvršava [49]. Specifikacije paralelnih algoritama predstavljaju više od definisanja niza postupaka za rješavanje računarskih problema zbog dodatne dimenzije konkurentnosti procesa tokom vremena. Niz procesa algoritma koji se mogu izvršavati simulatano mora biti definisan, kao i veza među njima [34]. U praksi, specifikacije paralelnih algoritama mogu sadržati: • • • • mapiranje konkurentnih djelova algoritama koji se izvršavaju istovremeno, identifikovanje podataka koji se mogu obrađivati konkurentno, upravljanje podacima koji dijele više procesora, distribuiranje ulaznih, izlaznih i međusobnih podataka na više jezgara procesora, • sinhronizacija procesora u različitim fazama izvršavanja paralelnog programa. Pored navedenog, podjela problema na manje zadatke i njihova dodjela različitim procesorima za paralelno

izvršenje predstavljaju ključne korake u dizajniranju paralelnih algoritama [50]. 2.1.1. Dizajn algoritama Računarski algoritmi moraju biti tačni, efikasni, stabilni, prenosivi i održivi [14] [43]. Kada su u pitanju paralelni algoritmi, mjerilo kvaliteta nazvano “efikasnost” se može proširiti tako da obuhvata paralelnu efikasnost i skalabilnost. U cilju postizanja paralelne efikasnosti, u razmatranje se moraju uzeti kašnjenja usled komunikacije i disbalansa opterećenja. Na osnovu parametra skalabilnosti izdvajaju se jako i slabo skalabilni algoritmi. Jako skalabilni algoritmi su oni algoritmi kod kojih, za određen problem, povećanje ili smanjenje broja CPU jezgara ne utiče na performanse. Nasuprot tome, kod slabo skalabilnih algoritama se sa povećanjem broja CPU jezgara ne dobija ušteda u vremenu i efikasnost opada. Efikasni paralelni algoritmi moraju zadovoljiti većinu ili svaki od sledećih uslova: • disbalans opterećenja jezgara je zanemarljiv, • gubici usled komunikacije su zanemarljivi, • sekvencijalni dio aplikacije je mnogo manji od paralelnog dijela. Ostvarenje svih ovih uslova osiguraće visoku efikasnost paralelnog algoritma. Ključni koraci za uspješnu paralelizaciju i rad paralelnog programa su: • dekompozicija podataka i kontrola, • planiranje procesa, • upravljanje komunikacijama (veličinom i brojem poruka), • balans opterećenja, • sinhronizacija i • analiza performansi i poboljšanje algoritma. Efikasni paralelni algoritmi, koji imaju široku primjenu u praksi, baziraju se na dekompoziciji domena (domain decomposition), dekompoziciji zadataka (control decomposition), master-slave algoritmu, paralenim podacima, SPMD (single program, multiple data) i modelu sa virtuelno dijeljenom memorijom (Virtual-shared-memory model). U praksi je uobičajeno da se pri paralelizaciji kombinuju i koriste dva ili više modela algoritama paralelizacije. 2.2. Raspodjela opterećenja paralelnih algoritama Fokus mnogih istraživanja u oblasti distribuiranog računarstva je pronalaženje optimalne raspodjele zadataka kako bi se postigla što bolja efikasnost, smanjilo vrijeme izvršenja paralelne aplikacije, smanjila komunikacija među resursima i povećala iskoristivost resursa. Da bi se navedeni ciljevi ostvarili, presudno je optimizovati dodjelu zadataka nodovima klastera i jezgrima njihovih procesora i nadgledati njihovo izvršenje. Postizanje paralelizma redistribucijom opterećenja paralelnih segmenata tokom izvršavanja paralelnog programa naziva se balansiranje opterećenja ili load balancing [6]. Osnovni cilj load balancing algoritama je pronalaženje optimalnog rasporeda kojim je definisano početno vrijeme izvršavanja i redosled izvršavanja svih zadataka koji se pokreću na određenom resursu [7]. Optimizacija performansi paralelnih aplikacija se izvršava upravljanjem i redistribucijom zadataka tokom izvršavanja load balancing algoritama i kao rezultat smanjuje vrijeme komunikacije, sinhronizacije i vremena čekanja zbog neravnomjerne distribucije procesa [51]. Dizajn load balancing algoritama distribuiranih računarskih sistemima započeo je prije više od četrdeset godina [9]. Tehnologije balansiranja opterećenja u paralelnim sistemima razvijane su na dva načina: planiranje i izvršavanje paralelnih aplikacija na distribuiranom računarskom resursu [52] i raspodjelu zadataka unutar paralelnih aplikacija. Razvijeni su različiti algoritmi raspodjele pokretanja procesa i aplikacija za računarske klastere visokih performansi, računarske grid-ove [53] [54] i za Cloud infrastrukturu [55] [56]. Cilj ovih algoritama je postizanje što veće iskoristivost resursa, uz što kraće vrijeme izvršavanja aplikacija i manjeg vremenskog razmaka između kraja izvršavanja jedne i pokretanja druge aplikacije. Rasporedom izvršavanja paralelnih aplikacija upravlja sistemski menadžer koji prilagođava opterećenje računarskog klastera, redosled pokretanja aplikacija i raspodjelu na kojim će se resursima izvršavati. Kod napredne raspodjele, u zavisnosti od veličine fluktuacija sistema, uključena je ili se može aktivirati dinamička i adaptivna strategija generisana šemom planiranja raspodjele zadataka. Automatska raspodjela se bazira na algoritmu koji može dinamički promijeniti svoje polise i spada u grupu algoritama koji koriste adaptivne distribuirane raspodjele. Najsofisticiraniji algoritmi brzo reaguju na kratkoročne promjene i kolebanja u promjenljivom okruženju. Na principu raspodjele pokretanja aplikacije na distribuiranim resursima razvijeni su i algoritmi uravnoteženja opterećenja za raspodjelu zadataka unutar paralelnih aplikacija. Određeni algoritmi su razvijeni kao load balancing algoritmi opšte namjene [57], dok su neki od njih specijalizovani za određeni vid paralelnih simulacija. Za svoj rad koriste karakteristike infrastrukture, parametare

opterećenja i trenutne zauzetosti resursa i rade na smanjivanju gubitaka usled izvršavanja paralelnih aplikacija. Disbalans paralelnih aplikacija najčešće se javlja usled prevelike komunikacije među jezgrima ili neujednačenog opterećenja medju jezgrima i velikog vremena čekanja koje se pritom stvara [8]. Mnogi load balancing algoritmi su razvijeni u nastojanju da smanje navedene nedostatke. Algoritmi se baziraju na monitoringu izvršavanja na svakom jezgru pojedinačno i raspodjeli i redistribuciji zadataka tokom izvršavanja radi postizanja veće efikasnosti. U realnom distribuiranom okruženju opterećenost resursa varira tokom vremena i nije uvijek moguće izvršavati aplikaciju na potpuno slobodnom klasteru ili klasteru sa jednako opterećenim resursima. Takođe, nije uvijek moguće odrediti ili predvidjeti dužinu trajanja procesa koji se izvršavaju na odvojenim nodovima ili kašnjenja zbog komunikacije između nodova, tako da dolazi do dužeg izvršenja paralelne aplikacije i pada iskoristivosti resursa. Kraj izvršavanja paralelne aplikacije i početak obrade rezultata paralelne aplikacije (u postprocessing fazi) direktno zavise od trajanja izvršavanja dijela aplikacije na procesoru kojem je dodijeljeno najviše procesa ili procesoru sa najnižim radnim taktom. Ova pojava je izražena na klasterima koji se sastoje od računara različitih performansi (heterogeni klasteri)

ili klasterima sa promjenljivim opterećenjem, tj. klasterima na kojima više korisnika u isto vrijeme izvršava paralelne aplikacije i time opterećuje resurse. 1

Pored toga, mnoge savremeni super-računari, kao što su višejezgarni ili SMP klasteri, koje izgledaju homogeno se zapravo ponašaju kao heterogeni i dinamički resursi. Na primjer, procesori koji se nalaze na istom serveru pristupaju dijeljenim resursima i njihova komunikacija je značajno brža od komunikacije sa procesorima drugih servera u okviru istog klastera [58]. Gubici usled izvršavanja paralelnih aplikacija se mogu dogoditi i usled različitog vremena završetka rada zadataka na pojedinačnim jezgrima i čekanja najsporijeg ili najzauzetijeg jezgra da završi dodijeljene zadatke. Ova pojava se često događa na heterogenim klasterima ili homogenim klasterima sa dinamičkom raspodjelom opterećenja (klasterima gdje više korisnika simultano izvršava paralelne aplikacije i time neravnomjerno opterećuju resurse) [4] [43] [59]. 2.2.1. Podjela i balansiranje opterećenja Algoritmi za distribuciju opterecénja mogu se klasifikovati kao algoritmi za podjelu i algoritmi za balansiranje opterećenja. Cilj algoritma za podjelu opterecénja je poboljšavanje rada distribuiranog sistema i optimizacija raspodjele zadataka u trenutku kada oni pristižu na izvršavanje. Algoritmi za podjelu opterecénja nastoje da izbjegnu dodjelu zadataka zauzetim jezgrima [60]. Ako se zadaci raspodjeljuju po pristizanju može doći do greške i zadatak se dodijeliti već zauzetom jezgru. Ovakav vid greške se može izbjeći prebacivanjem zadataka na slobodno jezgro ili jezgro kojem je dodijeljen manji broj zadataka. Transferi zadataka nisu trenutni zbog vremena potrebnog za prikupljanje zauzetosti jezgara klastera i kašnjenja usled komunikacije. Dugotrajno čekanje zadataka na pokretanje izvršavanja se može djelimično izbjecí transferom zadataka, pod pretpostavkom da cé slabije opterećena jezgra uskoro postati potpuno slobodna za izvršavanje prebačenih zadataka. Algoritmi za balansiranje, pored rasporeda opterećenja, pokušavaju da izjednače opterecénja na svim distribuiranim resursima gdje se aplikacija izvršava. Krueger i Livni [61] su pokazali da balansiranje opterecénja može smanjiti srednju vrijednost i standardnu devijaciju vremena čekanja zadataka za izvršavanje. Na distribuiranom sistemu se mogu javiti uvećani troškovi usled veće komunikacije između jezgara zbog više zahtjeva za transferom i ispitivanjem stanja pri ujednačavanju opterećenja resursa. 2.3. Klasifikacija load balancing algoritama Load balancing algoritmi se mogu podijeliti na lokalne i globalne [62] [63]. Klasifikacija se vrši na osnovu vrste resursa na kome se program izvršava i tipa informacije na osnovu koje se donosi odluka [64]. Njihova podjela je

prikazana na slici 5. Load balancing algoritmi LOKALNI GLOBALNI Load balancing algoritmi Load balancing algoritmi Statička Dinamička raspodjela procesa raspodjela procesa Optimalna Sub-optimalna Fizički Fizički distribuirana nedistribuirana Slika 5. Podjela load balancing algoritama Lokalnu raspodjelu vrši svaki procesor pojedinačno i sastoji se od slanja niza instrukcija koje se izvršavaju se bez komunikacije između nodova klastera [65]. Lokalna raspodjela predstavlja najjednostavniji način raspodjele procesa i ima male gubitke usled komunikacije. Globalna raspodjela predstavlja proces odlučivanja o načinu izvršavanja procesa u višeprocesorskom sistemu. Planiranje raspodjele ili preraspodjele procesa se može obavljati sa jednog centralnog mjesta (CPU jezgra) ili sa više jezgara koja komuniciraju međusobno. Kod globalne raspodjele, za balansiranje opterećenja se koriste informacije o performansama nodova klastera i karakteristika aplikacije koja se izvršava. Globalne metode raspodjele se mogu klasifikovati u dve grupe: statička i dinamička raspodjela procesa ili balansiranje opterecénja [10]. Njihov opis i dalja podjela će biti izloženi u nastavku rada. 2.3.1. Statički algoritmi raspodjele Pri statičkoj raspodjeli, dodjela zadataka jezgima procesora se obavlja prije početka izvođenja paralelnog dijela programa. Pretpostavlja se da su informacije o karakteristikama i zauzetosti resursima poznate u vrijeme planiranja raspodjele procesa, kao i estimacija vremena predvidjenog za izvršenje procesa. Procesi se uvijek izvršavaju na jezgrima koja su im dodijeljena i u toku izvršenja ih nije moguće mijenjati. Statički algoritami pokušavaju da smanje gubitke i vrijeme izvršavanja smanjenjem komunikacije među jezgrima [66] [67] [68] [69]. Statičke metode raspodjele prilikom generisanja redosljeda izvršavanja zadataka pokušavaju da : - procijene vrijeme izvršavanje zadataka i vrijeme neophodno za komunikaciju među njima i shodno tome naprave raspored izvršavanja i - grupišu zadatke koji više komuniciraju sa ciljem smanjenja kominikacije medju resursima (jezgrima, procesorima ili računarima klastera) u toku izvršavanja Kašnjenja usled komunikacije između jezgara događaju se samo prije izvršavanja paralelne aplikacije (u fazi preprocesinga) i manja su ili jednaka u odnosu na dinamičke metode raspoređivanja. Najčešći statički algoritmi se baziraju na podjeli zadataka ili domena i njihovoj analizi na distribuiranom sistemu (task i domain decomposition algoritmi). Statički algoritmi raspodjele se mogu podijeliti na optimalne i suboptimalne. Za planiranje i donošenje odluke o načinu alokacije resursa na osnovu optimalnih algoritama neophodni su mjerni podaci – zauzetost resursa i minimalno vrijeme obrade aplikacije ili njenih djelova. Optimalan redosled izvršavanja zadataka je moguće postići samo u slučajevima kada je vrijeme izvršavanja svih procesa jednako ili kad se raspodjela procesa vrši na dva procesora [70]. Pošto zauzetost resursa i minimalno vrijeme obrade nisu uvijek poznati, većina istraživanja i razvoja algoritama u domenu statičke raspodjele koncentrisana je na suboptimalne algoritme. Oni se zasnivaju na heuristici i aproksimativnim metodama koje ne tragaju za optimalnim rješenjem raspodjele, već se zadovoljavaju raspodjelom koja daje zadovoljavajuću efikasnost. Suboptimalni algoritmi koristite definisana pravila i trenutno stanje sistema za izbor strategije raspodjele procesa kako bi se postigle performanse najbliže optimalnoj raspodjeli procesa. Nedostatak statičkih algoritama predstavlja nepostojanje efikasnih i tačnih metoda za procjenu vremena izvršavanja zadataka i kašnjenja usled komunikacije koje može prouzrokovati nepredvidljive degradacije performansi [71] [72]. Često je nemoguće predvidjeti vrijeme trajanja paralelnog programa i njegovih djelova zbog ulaznih podataka, petlji i iteracija čiji parametri nisu poznati prije izvršavanja programa. Takođe, postojeći algoritmi raspodjele zadataka često ne razmatraju problem distribucije podataka. Ovaj propust dovodi do degradacije performansi usled kašnjenja u komunikaciji zbog različitog vremena pristupa podacima na udaljenim lokacijama. Da bi se postigla što veća efikasnost statičkih algoritama, potrebno je korištenje specijalizovanih alata za analizu performansi programa i infrastrukture i predviđanje toka izvršavanja programa prilikom definisanja pravila za raspodjelu procesa. Jedan od predloženih alata je DAG (directed acyclic graph) generator [73] i performance profiler. DAG generator kao ulazni podatak učitava paralelni program i generiše graf sa funkcionalnim zavisnostima izmedju cjelina programa, procjenama vremena izvršenja i kašnjenja

usled komunikacije. Alat pruža analizu performansi i raspodjelu procesa paralelnog programa na datoj arhitekturi i generisanje grafičkog profila očekivanih performansi dajući korisniku pregled paralelizma u programu. 2.3.2. Dinamički algoritmi raspodjele Dinamička raspodjela procesa se bazira na preraspodijeli procesa paralelnih aplikacija između jezgara računarskog klastera tokom izvršenja aplikacije. Redistribucija se vrši prenošenjem zadataka sa više opterećenih jezgara na manje opterećena jezgra procesora s ciljem poboljšanja performansi aplikacije. Dinamički algoritmi imaju potencijal da poboljšaju performanse i prevaziđu probleme statičkih algoritama korišćenjem informacija o sistemu prilikom donošenje odluka tokom izvršavanja aplikacija. Zbog toga što moraju sakupljati, čuvati i analizirati informacije o stanju sistema, dinamički algoritmi imaju više gubitaka nego statički, ali se kašnjenje usled komunikacije kompenzuje boljom raspodjelom zadataka i kraćim vremenom izvršavanja. Napredni dinamički algoritmi mogu izvršavati procese slično kao statički algoritmi, a aktivirati pravila dinamičke raspodjele samo u slučaju neravnomjerne opterećenosti klastera [74] [75] [76] [77]. Cilj load balancing algoritama je uravnoteženje opterecénja na svim jezgrima raspoloživim za izvršavanje paralelne aplikacije. Opterecénje jezgra se može procijeniti mjerljivim parametrima [78]. Različiti dinamički algoritmi za kreiranje liste izvršavanja procesa i njihove preraspodjele koriste neke od parametara: • • • dostupnost jezgara procesora za procese koji su na čekanju, redosled zadataka koji čekaju izvršenje na određenom jezgru, prosječan broj zadataka koji čeka na izvršenje u određenom vremenskom periodu, • • stopa iskoristivosti CPU-a i arhitekturu klastera (CPU, memorija, storage). Dinamički load balancing algoritmi se razlikuju u stepenu centralizacije. U zavisnosti od lokacije na kojoj se vrši odluka o balansiranju opterecénja razlikujemo: • centralizovane, • decentralizovane, • hijerarhijske ili kombinovane algoritme. Kod centralizovanog algoritma se balansiranje opterećenja izvodi na jednom jezgru, a ako se raspodjela izvodi na svim jezgrima nazivaju se decentralizovani dinamički algoritmi. Algoritmi sa centralizovanim komponentama su potencijalno manje pouzdani od decentralizovanih algoritama, s obzirom da otkazivanje sistema na mjestu odlučivanja centralne komponente može prouzrokovati propust u radu čitavog sistema i izvršavanja cijele aplikacije. Rješenje ovog problema je izrada redudanse koja se može aktivirati kada glavna komponenta odlučivanja zakaže u radu. Drugi problem leži u centralnoj komponenti koja je potencijalno usko grlo koje ograničava raspodjelu opterećenja. Hijerarhijski ili kombinovani algoritmi mogu donekle riješiti oba problema, a kao najpouzdaniji se smatraju potpuno decentralizovani algoritmi [79]. Okolnosti koje utiču na optimalni balancing se često mijenjaju tokom izvršavanja programa i shodno tome se moraju donositi odluke da li je potrebno balansiranje opterećenja i sa kojim algoritmom raspodjele. Dinamički algoritmi se oslanjaju na najnovije informacije o stanju sistema i određuju naredbe procesorima u toku rada, te su stoga atraktivniji sa stanovišta performansi. Na osnovu informacija o trenutnom stanju se vrši dinamički transfer zadataka sa preopterecénog jezgra na manje opterećeno. Sposobnost reagovanja na promjene u sistemu je glavna prednost dinamičkog pristupa balansiranja opterecénja [80] [81] [82]. Algoritmi balansiranja opterećenja se sastoje od više komponenti koje na različite načine utiču na (re)distribuciju poslova među jezgrima distribuiranog sistema sa ciljem poboljšanja performansi izvršavanja programa [83]. Tipičan algoritam za balansiranje opterećenja definisan je sa četiri strategije ili polise: polisa prenosa zadatka, odabira, lokacije i informisanja. Strategija prenosa/transfera određuje da li je jezgro u odgovarajućem stanju da učestvuje u prenosu zadataka, bilo kao predajnik ili kao prijemnik. Polise transfera se baziraju na definisanju limita prikazanih kao udio opterećenja. Ukoliko se prilikom izvršavanja zadatka utvrdi da je opterećenje jezgra veće od propisanog limita onda se jezgro definiše kao predajnik i zadatak se ne pokreće, već se inicira njegova migracija na drugo jezgro. Sa druge strane, ako je opterecénje na jezgru ispod limita prijemnika, polise prenosa odlučuju da jezgro može biti prijemnik za zadatak. Zavisno od algoritma, limiti predajnika i prijemnika mogu biti različito definisani. Za donošenje odluka se mogu koristiti i relativne polise prenosa koje razmatraju opterecénje jezgara u odnosu na druga jezgra na kojima se izvršava ista aplikacija. Relativne polise

mogu uzeti u obzir jezgro kao prijemnik ako je njegovo opterecénje niže od opterećenja drugih jezgara za neku unaprijed definisanu vrijednost. Takođe, jezgro se može smatrati prijemnikom ako je njegovo opterecénje među najnižim u sistemu. Pošto polise transfera odrede koje jezgro može biti predajnik i prijemnik, polise odabira traže odgovarajući zadatak za migraciju. Najlakši način odabira zadatka za prenos je odabir prvog neizvršenog zadatka koji definiše neko jezgro kao predajnik. Takav zadatak je relativno lako prenijeti na jezgro prijemnika. Ovo svojstvo razmatra nekoliko faktora prilikom odabira zadatka: 1. kašnjenja usled prenosa moraju biti minimalna, 2. odabrani zadatak bi trebao biti dugotrajan, tako da je vrijedno izvršiti njegov prenos 3. broj sistemskih instrukcija koje obavlja izabrani zadatak i zavise od lokacije treba da bude minimalan. Sistemske instrukcije se odnose na instrukcije izmedju zadatka i operativnog sistema resursa predajnika i odnose se na tačno definisane zadatke koji se moraju izvršiti na tim resursima [84] [85]. Odgovornost polise lokacije je pronalaženje odgovarajucég para za transfer (predajnik ili prijemnik), kada se polisama prenosa definiše određeno jezgro kao predajnik ili prijemnik. Često se koriste decentralizovane polise koja nalaze odgovarajuća jezgra putem odabira: jezgro predajnik ili prijemnik se obraća ostalim jezgrima pitanjem da li je moguć transfer zadatka. Jezgra mogu biti kontaktirana serijski ili paralelno koristeći multicast. Jezgro za transfer može biti odabrano na slučajan način [60] [86], na bazi informacija od ranijih odabira za transfer [87] [74], odabirom najbližeg jezgra ili slanjem upita svim jezgrima i traženjem bilo kog jezgra dostupnog za dijeljenje zadataka. U centralizovanoj polisi lokacije, jezgro se obraća jezgru koje je odabrano za koordinatora da locira odgovarajucé resurse za dijeljenje opterecénja. Koordinator prikuplja informacije o distribuiranom sistemu (koji je u nadležnosti informacionih polisa), a polisa prenosa koristi ove informacije kod koordinatora za odabir prijemnika. Informaciona polisa odlučuje kada se prikupljaju informacije o stanju resursa od kojih se infrormacije prikupljaju i sadržaj informacija koje se prikupljaju. Postoje tri vrste informacionih polisa. 1. Polisa usmjerena na traženje. Prema ovim pravilima jezgro sakuplja stanje drugih jezgara samo kada postane ili predajnik ili prijemnik, što ga čini pogodnim kandidatom za dijeljenje opreterećenja. Informacije mogu tražiti predajnici, prijemnici ili jezgra usled simetrično iniciranog prenosa. U smjernicama koje iniciraju predajnici, predajnici traže prijemnike na koje mogu prenijeti svoje opterecénje. U polisama koje iniciraju prijemnici, prijemnici traže prenos zadataka sa opterećenih jezgara predajnika; 2. Periodične polise. Ove polise, centralizovane ili decentralizovane, prikupljaju informacije periodično. U zavisnosti od prikupljenih informacija, pravila prenosa se mogu pokrenuti radi prenosa zadataka. Periodične informacione polise generalno ne prilagođavaju stopu aktivnosti stanju sistema. Prednosti koje proizilaze iz distribucije opterecénja su minimalne pri velikom opterecénju sistema, jer je vecína jezgara u sistemu zauzeta. Kašnjenja zbog periodičnog prikupljanja informacija još više povecávaju opterecénje sistema i time produžavaju ukupno trajanje aplikacije; 3. Polise koje upravljaju stanjima. U pravilima koja se baziraju na promjenama stanja, jezgra šalju informacije ostalim jezgrima kad god se njihovo stanje u određenoj mjeri promijeni. Polise se razlikuju od polisa usmerenih na potražnju jer distribuiraju informacije o stanju zauzetosti jezgara procesora i ne prikupljaju informacije o stanju drugih. U centralizovano organizovanim resursima jezgra šalju informacije o stanju na centralizovanu tačku prikupljanja. U decentralizovanim polisama informacije se šalju svim ostalim angažovanim resursima. U teoriji algoritama je definisan pojam stabilnosti računarskog sistema u odnosu na raspodjelu opterećenja. Prema teoriji, kada je broj zadataka koji u jedinici vremena dolazi na obradu veći od broja zadataka koje sistem može da raspodjeljuje i izvršava, redovi čekanja rastu bez ograničenja i sistem se naziva nestabilnim. S druge strane, algoritam može biti stabilan, ali to ne znači da će sistem raditi bolje od istog sistema sa statičkom raspodjelom poslova. Stoga se za procjenu stabilnosti računarskog sistema koristi efikasnost algoritma. Algoritam distribucije opterecénja je efikasan ukoliko poboljšava performanse u odnosu na sistem koji ne koristi distribuciju opterećenja. Efikasan algoritam ne može biti nestabilan, ali stabilan algoritam može biti neefikasan. Prenošenje zadatka na prijemnik može povecáti dužinu

reda čekanja prijemnika preko definisanog limita, što zahtijeva prenošenje tog zadatka na sledeći resurs u potrazi za manje opterećenim prijemnikom koji bi izvršio zadatak. Ovaj proces se može ponoviti beskonačno mnogo puta, ukoliko maksimalan broj prenošenja zadatka nije unaprijed definisan. 2.3.2.1. Adaptivni load balancing algoritmi Adaptivni algoritmi distribucije opterećenja predstavljaju posebnu klasu dinamičkih algoritama. Ovi algoritmi, poput dinamičkih algoritama, balansiraju opterecénje po dolasku svakog zadatka, ali i balansiraju opterecénje kad god se pojavljuju anomalije u radnom opterecénju sistema ili pojedinih resursa. Aktivnosti ovih algoritama se prilagođavaju opterećenju resursa tako što dinamički mijenjaju svoje parametre i pravila dinamičkih algoritama kako bi odgovarali promjenljivom stanju sistema [88] [89]. U računarskim okruženjima čije se karakteristike sistema značajno ne mjenjaju (npr. homogeni sistemi uz ujednačeno opterećenje) dinamički pristup koji koristi samo jednostavnu raspodjelu poslova može pružiti bolju efikasnost u odnosu na statičku raspodjelu i ne iziskuje aktivaciju složenijih algoritama za balansiranje opterećenja. Računarsko okruženje pogodno za adaptivne algoritme karakterišu heterogeni resursi, brza promjena opterećenja, nepredvidive karakteristike stanja sistema i promjenljivo radno opterećenje. U ovim okruženjima raspodjela opterećenja nije homogena i količina zadataka i podataka koji trebaju biti obrađeni može značajno da varira tokom vremena, što onemogućava izradu jedne strategije balansiranja opterećenja koja daje efikasne rezultate u svim okolnostima. Adaptivni algoritmi prepoznati su kao najznačajniji algoritmi koji se prilagođavaju promjenama u dinamičkom okruženju i u širokom dijapazonu ulaznih podataka i aplikacija. Među promjenama u okruženju koje se mogu dogoditi i za koje se strategije adaptivnih algoritama moraju prilagoditi su: • broj dostupnih jezgara, • varijacije u opterećenju kompletnog resursa, • raspodjela veličine procesa i obrađenih podataka, • nivo korištenja komunikacijske mreže resursa. Nabrojani parametri utiču na izbor odgovarajucég nivoa raspodjele opterecénja i strategije planiranja raspodjele. Procedure raspodjele poslova u takvom okruženju moraju automatski podesiti parametre algoritma kojim raspodjeljuje poslove ili prelaziti na druge algoritme s obzirom na trenutno stanje okruženja. Adaptivne šeme raspodjele trebaju biti prilagodljive i odmah reagovati na anomalije, omogucávajucí da distribuirani sistem uvek radi uz maksimalnu iskoristivost. Adaptivni algoritam treba uzeti u obzir dinamičke promjene opterećenja sistema i sadržati adekvatnu proceduru donošenja odluka za kontrolu ovih modifikacija. Jednostavna podešavanja parametara polisa raspodjele (npr. promjena dinamičkog praga odlučivanja) može značajno poboljšati performanse kada se opterećenje sistema fluktuira. U slučaju kada je sistem ravnomjerno toliko opterećen da se ne može dobiti poboljšanje performansi prenošenjem zadataka, neadaptivni dinamički algoritam može nastaviti sa radom. Da bi se izbjeglo preopterećenje sistema usled gubitaka koji se proizvode, adaptivnom algoritmu se može ograničiti aktivnost raspodjele opterećenja samo kada je to neophodno. 2.3.3. Primjeri dinamičkih algoritama Dinamički algoritmi se mogu klasifikovati po mjestu sa koga je raspodjela procesa inicirana. Tako ih možemo grupisati kao algoritme pokrenute od strane predajnika, prijemnika ili simetrično inicirane algoritme. 2.3.3.1. Algoritmi pokrenuti od strane predajnika Kod algoritama iniciranih od strane predajnika distribucija opterećenja aktivirana je od strane preopterećenog jezgra koje pokušava da izvrši transfer zadatka na drugo jezgro (prijemnik). Eager, Lazowska, and Zahorjan [74] su analizirali distribuirane algoritme kod kojih je transfer pokrenut od strane predajnika. Algoritmi koriste ista pravila prenosa uz polise praga koje se baziraju na dužini čekanja na izvršavanje. Jezgro se identifikuje kao predajnik ako pristigli zadatak čini dužinu čekanja većom od zadate vrijednosti T. U suprotnom, jezgro postaje prijemnik ako prihvatanje zadatka neće dovesti do dužine čekanja veće od definisane vrijednosti T. Sva tri algoritma procesuiraju samo pristigle zadatke za prenos, a razlikuju se samo u svojim lokacijskim polisama. Slučajni (random) algoritam ne koristi informacije o stanju resursa na koje se vrši prenos zadataka. Zadatak se prilikom transfera prenosi na slučajno izabrano jezgro bez razmjene informacija između jezgara koje bi pomogle u donošenju odluke. Nasumični odabir može prouzrokovati i bespotrebni prenos kod koga se zadatak prenese na jezgro gdje već čeka izvršenje

broj zadataka veći ili jednak od unesenog limita T. Ukoliko je cijeli sistem zasićen, odnosno sva jezgra imaju više od T zadataka koji čekaju izvršavanje, može se generisati beskonačno zahtjeva za transfer zadataka. U tom slučaju je neophodno uvesti limit koliko se maksimalno puta zadatak može prenijeti. Strategija nasumičnog odabira lokacije pruža bolje performance u odnosu na sisteme koji ne koriste distribuciju opterecénja [74] . Kod algoritma praga strategija lokacije izbjegava beskorisne prenose zadataka nasumičnim odabirom jezgra za izvršenje na osnovu dužine čekanja T. Ako je red čekanja manji, zadatak se prenosi na izabrano jezgro koje mora izvršiti zadatak ili ga staviti na listu čekanja za izvršavanje. U suprotnom, drugo jezgro se bira nasumično i ispituje se njegovo stanje. Da bi se snizio broj ispitivanja da li jezgro može biti primalac zadatka uvodi se parametar koji ograničava broj ispitanih jezgara. Ako se ne pronađe odgovarajucí prijemnik, tada jezgro predajnik mora izvršiti zadatak lokalno. Izbjegavajucí beskorisne prenose zadataka, polisa lokacije kod algoritma sa pragom pruža znatno poboljšanje performansi u odnosu na polisu lokacije nasumičnog algoritma. Prema pravilniku o lokaciji algoritma koji koristi najkraći red čekanja (shortest), broj jezgara se bira nasumično i ispituje se njihovo trenutno opterećenje da bi se odredio broj zadataka koji čeka na izvršenje. Jezgro sa najkracím redom čekanja je izabrano kao prijemnik, osim ako je njegova dužina čekanja vecá ili jednaka T. Upoređenje performansi algoritma je pokazalo da algoritam sa najkraćim redom čekanja daje vrlo malo poboljšanje, što ukazuje da korišćenje detaljnijih informacija o stanjima ne mora značajno poboljšati ukupne performanse paralelne aplikacije. Prilikom korištenja algoritma koji koristi red čekanja ili prag za prenos, ispitivanje stanja potencijalnih prijemnika počinje kada polisa prenosa identifikuje predajnik. Prema tome, polisa informisanja se kod ovih algoritama zasniva na potražnji informacija stanja distribuiranog sistema. Algoritmi inicirani od strane predajnika uz bilo koju od tri polise lokacije uzrokuju nestabilnost sistema kada je sistem visoko opterecén. U takvom stanju sistema postoji vjerovatnocá da neće biti određeno slobodno jezgro za prenos ili prijemnik nižeg opterećenja. Gubici usled ispitivanja stanja potencijalnih prijemnika u algoritmima iniciranim od strane predajnika se povecávaju sa povecánjem brzine pristizanja zadatka. U ekstremnom slučaju, radno opterecénje usled komunikacije prevazilazi kapacitet CPU jezgra i rezultira nestabilnostima sistema. Stoga algoritmi koje iniciraju predajnici nisu efikasni kod velikih opterecénja sistema i uzrokuju nestabilnost, jer se algoritmi ne prilagođavaju sistemskom stanju. 2.3.3.2. Algoritmi inicirani od strane prijemnika Kod algoritama čiji transfer inicira prijemnik aktivnost raspodjele opterećenja pokreće se iz manje opterećenog jezgra (prijemnika) koji pokušava da dobije zadatak iz preopterećenog jezgra (predajnika) [60] [87]. Strategija transfera praga bazira svoju odluku na dužini zadataka koji čekaju na izvršavanje na svakom jezgru. Ako je dužina reda čekanja ispod praga T, onda se to jezgro identifikuje kao prijemnik sposoban za prenos zadatka sa jezgra predajnika i koji će biti određen pravilom lokacije. Jezgro je identifikovano kao predajnik ako njegova dužina reda prelazi granicu T. Stretegija lokacije odabira jezgro nasumično i ispituje njegov status kako bi se utvrdilo da li je sposobno da bude predajnik shodno definisanom pragu. Ukoliko jezgro nije odabrano, sledeće se bira nasumično i procedura se ponavlja sve dok se ne pronađe ono koje može prenijeti zadatak, odnosno biti predajnik, ili se dostigne određeni limit broja pokušaja traženja predajnika. Problem nastaje ukoliko se ispitivanjima statusa ne može naći odgovarajući predajnik. U tom slučaju je vrijeme traženja uzalud potrošeno i nakon toga slobodni prijemnik čeka dok mu drugi zadatak ne bude dodijeljen. Problem može ozbiljno uticati na performanse u sistemima u kojima je nekoliko jezgara preopterećeno i slučajnim traženjem prijemnika se mogu preskočiti. Ukoliko sva ispitivanja ne pronađu predajnika, jezgro čeka dok ne izvrši sledeći zadatak ili u određenom periodu ponovo aktivira raspodjelu opterećenja, pod uslovom da je jezgro i dalje u statusu prijemnika [86]. Strategija informisanja je u ovom slučaju usmjerena na potražnju zadataka, pošto postupak traženja predajnika počinje tek nakon što jezgro postane prijemnik. Algoritmi čiji prenos zadataka iniciraju prijemnici ne uzrokuju nestabilnost sistema jer će pri velikim opterećenjima sistema prijemnik vjerovatno naći odgovarajućeg predajnika u nekoliko ispitivanja.

Prema tome ispitivanja statusa daju bolje rezultate pri većem opterećenju sistema i malo utiču na gubitak performansi distribuiranog resursa. U algoritmima raspodjele iniciranim od prijemnika, odabir počinje kada jezgro postaje manje opterećeno, tj. potencijalni prijemnik. Ispitivanje stanja predajnika se izvršavaju pošto su im predati novi zadaci na izvršavanje, ali prije nego što su počeli da se izvršavaju. Algoritmi koje iniciraju predajnici, s druge strane, mogu pokrenuti aktivnost distribucije opterećenja čim stigne novi zadatak i stoga su lakši za raspodjelu opterećenja. Alternativa ovom tipu algoritama predstavlja algoritam kod kojeg umesto da se ispituje stanje resursa zbog transfera već dodijeljenih zadataka prijemnik zahtijeva da se na njega prenese sljedeći zadatak koji stigne na izvršavanje. Rezervisan zadatak se po pristizanju odmah prenosi na prijemnik ako je on u to vrijeme još uvijek sposoban da primi zadatak. Iako ovaj algoritam ne zahteva prenošenje već dodijeljenih zadataka, utvrđeno je da se radi znatno lošije od algoritama koji se iniciraju od strane prijemnika. 2.3.3.3. Simetrično inicirani algoritmi Pod simetrično iniciranim algoritmima [61], i predajnik i prijemnik pokreću aktivnosti distribucije opterećenja za prenos zadataka. Ovi algoritmi imaju prednosti nad onim algoritama kod kojih transfer iniciraju ili predajnici ili prijemnici. Kod slabo opterećenih sistema, komponenta inicirana od strane predajnika je efikasnija u pronalaženju jezgara sa malim opterećenjem. Kod opterećenih sistema, komponenta koja je započela od strane prijemnika je uspješnija u pronalaženju preopterećenih jezgara. Uzrok nestabilnosti kod simetrično iniciranih algoritama predstavlja neselektivno ispitivanje resursa i nedostaci koje su naslijedili od algoritama koji ih sačinjavaju. Stabilni simetrično inicirani adaptivni algoritam [86] predstavlja jedan od najboljih predstavnika adaptivnih algoritama. Algoritam koristi informacije o opterećenosti koje su prikupljene tokom ispitivanja stanja umjesto da ih po završenoj obradi odbace, kao što to rade prethodno navedeni algoritmi. Informacije o stanju zauzetosti jezgara nisu centralizovane i osvježavaju se na svakom resursu koji se koristi u paralelnoj obradi. Ovaj algoritam pokazuje dobru skalabilnost na velikim distribuiranim sistemima. Pri velikih opterećenja sistema vjerovatnoća da je neko jezgro slabo opterećeno je vrlo mala. Prema tome ispitivanjima stanja od strane predajnika su obično neuspješna i rezultiraju uklanjanjem zauzetih jezgara iz liste prijemnika, što u ekstremnom slučaju kod opterećenog sistema dovodi do pražnjenja liste prijemnika. Navedeni algoritam sprečava buduća ispitivanja pokrenuta od strane predajnika pri velikim opterećenjima sistema i deaktivira se ostavljajući samo dijeljenje opterećenja koje je pokrenulo prijemnik. Kod malih opterećenja sistema, analize započete prijemnikom su česte i uglavnom ne rezultuju pronalaženjem predajnika. Proces ispitivanja obično ne utiče negativno na performanse, jer su resursi dostupni za izvršavanje pri niskom opterećenju sistema. Pored toga, ova ispitivanja pozitivno utiču na ažuriranje lista stanja prijemnika. Sa listama koje tačno odražavaju stanje sistema, buduća ispitivanja od strane predajnika će biti uspješna u nekoliko koraka. Koristeći sve tri raspodjele opterećenja inicirane od strane predajnika pri niskim opterećenjima, raspodjelom opterećenja na primaocima pri visokim opterećenjima i obostranoj raspodjeli opterećenja pri umerenim opterećenjem, stiče se stabilan simetrički inicirani algoritam i postižu poboljšane performanse u širokom opsegu opterećenja sistema. 3. Domain decomposition i Master- slave Load balancing algoritmi Domain decomposition i master slave algoritami su dva najčešće korištena statička i dinamička load balancing algoritma. Ovi algoritmi se koriste za raspodjelu zadataka pri izvršavanju različitih aplikacija na homogenim i heterogenim klasterima sa dijeljenom ili distribuiranom memorijom. U zavisnosti od tipa resursa na kome se izvršava i tipa paralelnog programa, algoritmi raspodjele pružaju različite rezultate balansiranja opterećenja i različite vrijednosti ubrzanja i efikasnosti. Navedeni algoritmi su osnova za razvoj kombinovanog adaptivnog load balancing algoritma koji je tema ovog rada. 3.1. Dekompozicija domena (DD) Numeričke simulacije koje obrađuju velike skupove podataka su uobičajene u inženjerskim i naučnim istraživanjima. Zbog ograničenja resursa personalnih računara, njihove memorije i procesora, a često i vremena za rješavanje problema, numeričke simulacije se ne rješavaju odjednom, već se dijele na više djelova koji se odvojeno rješavaju. Ukoliko postoji

mogućnost da se više dijelova problema može riješiti nezavisno, a zatim njihovi rezultati spojiti u rješenje cijelog problema, tada se rješavanje tog problema može realizovati na računarima visokih performansi koristeći paralelno programiranje [90]. Metode dekompozicije domena (Domain Decomposition - DD) su razvijene mnogo prije aktivnog korištenja distribuiranih sistema i paralelnog procesiranja. Definišu se kao tehnike podjele oblasti problema na manje oblasti koji se izvršavaju zasebno, a zatim se vrši integracija njihovih rezultata kako bi se dobilo globalno rješenje. U zavisnosti od problema, dekompozicija domena može biti sa preklapajućim ili nepreklapajućim poddomenima. Metode dekompozicije domena se koriste u numeričkoj analizi pri rješavanju parcijalnih diferencijalnih jednačina, pri paralelnim simulacijama strukturalnih analiza, provodjenja toplote, kretanja fluida, itd [91] [92]. One predstavljaju osnov za dizajn paralelnih programa i podjelu podataka pri analizi u fazi preprocessing-a statičkih i dinamičkih paralelnih algoritama. Zbog svoje efikasnosti i fleksibilnosti metode dekompozicije domena su predmet mnogih istraživanja i implementirane su u velikom broju paralelnih aplikacija [93] [94] [95] [96]. Sa stanovišta paralelnog procesiranja, metode dekompozicije se baziraju na podjeli domena na poddomene i njihovoj dodjeli različitim jezgrima procesora paralelnog računara na izvršavanje. Slika 6. ilustruje podjelu podataka metodama dekompozicije domena na MxN poddomena. 1,1 ... ... 1, N ... ... M, 1 ... ... M, N Slika 6. Skica dekompozicije domena u kome se domen ulaznih podataka dijeli na MxN poddomena koji se zatim dodjeljuju računarskim resursima Pored metoda dekompozicije domena, često se korisiti i dekompozicija procesa. Kod ovog metoda se procesi ili tokovi instrukcija dijele i dodjeljuju različitim jezgrima po principu sličnom podjeli domena. Pristup dekompozicije procesa je zastupljeniji kod aplikacija koje ne koriste matematičke proračune. 3.1.1. Algoritam dekompozicije domena Dekompozicije domena (DD) predstavlja osnovni statički algoritam za raspodjelu procesa kod paralelnih programa. Ovaj algoritam se koristi kod paralelnih računara sa zajedničkom i dijeljenom memorijom, kao i kod hibridnih modela paralelnih programa. U prvom koraku ovog algoritma se izvršava podjela domena na poddomene, a zatim se poddomeni šalju svim CPU jezgrima s jednakom vjerojatnoćom, uz unaprijed definisana pravila raspodjele na slučajan način ili koristeći definisani redosled raspodjele. Operacija dodjele poddommena ili zadataka jezgru procesora se izvršava prije početka izvršenja programa. Definisani zadatak se uvijek izvršava na dodijeljenom jezgru bez mogućnosti premještanja procesa na drugo jezgro. Komunikacija među jezgrima je svedena samo na komunikaciju prije i poslije izvršenja kompletnog seta dodijeljenih procesa, što je jedna od prednosti u odnosu na dinamičke metode raspodjele [97] [98] [99] [100]. Kao što je već navedeno u poglavlju 1.3.4, efikasnost DD algoritma je maksimalna od trenutka pokretanja izvršavanja paralelnog programa do trenutka kada prvo jezgro završi dodijeljene zadatke (Tmin). Od tog trenutka se javlja neuravnoteženje opterećenja programa i gubici u izvršenju se povećavaju kako ostala jezgra završavaju dodijeljene zadatke. Izvršavanje se obustavlja kad poslednje jezgro završi dodijeljene zadatke (Tmax) i svi rezultati paralelne obrade poddomena se predaju na dalje procesiranje. Neuravnoteženje paralelne aplikacije se događa jer trajanje pojedinih zadataka nije unaprijed poznato i taj parameter nije moguće uzeti u obzir prilikom planiranja raspodjele procesa. Pored toga, neuravnoteženost statičkih algoritama može biti prouzrokovana heterogenošću distribuiranog sistema i promjenom opterećenja djelova distribuiranog sistema u toku izvršenja aplikacije. Algoritam baziran na distribuciji domena je najefikasniji kada se računski problem može podijeliti na jednake djelove čija analiza podjednako traje, a računsko opterecénje je jednako raspodijeljeno između dostupnih resursa [58]. Slika 7. Distribucije vremena izvršavanja - loš balans opterećenja Slika 8. Distribucije vremena izvršavanja - dobar balans opterećenja Na slikama 7. i 8. su prikazani dijagrami izvršavanja loše i dobro uravnoteženog paralelnog programa uz pomoć DD algoritma. Plavom bojom je označen period izvršavanja procesa svakog jezgra, dok je crvenom označeno vrijeme kad su generisani gubici. Kod dobro uravnotežene raspodjele procesa primjećujemo da sva jezgra završavaju rad u kratkom vremenskom intervalu (Tmin~Tmax), dok od raspodjele sa velikim gubicima taj interval značajno veći

(Tmin << Tmax). 3.2 Master-slave algoritam (MS) Master-slave algoritam (MS) [101] predstavlja jedan od paralelnih algoritama za dinamičko balansiranje opterećenja. Raspodjelu procesa vrše dvije vrste jezgara ili procesora: master i slave jezgra procesora. Master jezgro kontroliše podjelu, distribuciju i izvršavanje zadataka na više slave jezgara ili procesora koji izvršavaju dodijeljene zadatke. Metodologija raspodjele master-slave je uspješno korištena u različitim paralelnim aplikacijama [102] [103] [104] i pokazala se kao vrlo pogodna kao programski model za aplikacije namijenjene heterogenim resursima [105] [106]. Osim paralelnih aplikacija, ovaj algoritam je pokazao dobre rezultate optimizacije u više sfera i njegova primjena je moguća u više oblasti gdje se koristi load balancing: u računarskoj industriji, bazama podataka, računarskim mrežama, industriji, transport itd. Preprocessing faza priprema ulazne podatke, vrši analizu i podjelu na domene ili procese. U ovoj fazi se mogu prikupljati podaci opterećenja distribuiranog sistema i shodno analizi opterećenja vršiti priprema za izvršavanje na slave jezgrima. Paralelnu obradu karakteriše izvršenje programa na n dodijeljenih jezgara, od kojih je jedno jezgro definisano kao master jezgro, dok se ostalih n-1 jezgara predstavljaju slave jezgra. Master jezgro sinhronizuje i vrši distribuciju zadataka na slave jezgra, koja su zadužena za izvršavanje dodijeljenih zadataka. Na početku paralelne obrade master jezgro sadrži podatke obrađene u preprocessing fazi, a slave jezgra nisu opterećena i čekaju dodjelu zadataka. Nakon toga master jezgro dodjeljuje svim slave jezgrima po jedan ili više zadataka na izvršenje (slika 9.) i ulazne podatke sa poddomena koje trebaju da analiziraju. Svako slave jezgro preuzima podatke, izvršava zadatak i po njegovom završetku informaše master jezgro. Nakon toga master jezgro dodjeljuje sledeći zadatak na izvršavanje. Ciklus se ponavlja do kraja niza procesa koji se trebaju paralelno izvršiti. Rezultati pojedinačnih procesa se mogu prosleđivati master jezru po završetku svakog procesa pojedinačno ili po okončanju rada svih procesa. Na vrijeme izvršavanja paralelnog dijela utiče pojednačno vrijeme izvršavanja zadataka i vrijeme provedeno na komunikaciju izmedju jezgara i sinhronizaciju procesa. Slave N-1 … … jezgro Slave 1 Slave 2 Slave 3 … … … Master t Slika 9. Raspodjela procesa kod master-slave algoritma Poslije paralelne obrade, u postprocessing fazi se vrši objedinjavanje i obrada dobijenih rezultata i skladištenje u željenom formatu izlaznih podataka. Prednost master-slave algoritma se ogleda u dobro koordinisanom procesu raspodjele zadataka tokom cijelog procesa izvršavanja. Podaci o opterećenju resursa se mogu prikupljati prije izvršavanja (preprocessing faza) ili tokom izvršavanja paralelnog dijela aplikacije i shodno dobijenim podacima dodjeljivati zadatke slave jezgrima sa određenim prioritetom. Podaci o resursima se mogu koristiti i pri određivanju broja zadataka koji se u nizu šalju određenim jezgrima. Algoritam ne vrši preraspodjelu tokom izvršavanja zadatka, već se zadatak izvršava na jezgru koje mu je dodijeljeno. Prednost MS algoritma predstavlja činjenica da program ne mora imati podatke o opterećenosti klastera prije i tokom izvršavanja, već se raspodjela vrši na slobodnim jezgrima shodno promjenama u opterećenjima distribuiranih resursa. Nedostatak master-slave algoritma predstavlja komunikacija između master i slave jezgara tokom cijelog procesa izvršavanja i potencijalno čekanje slave jezgara za dodjelu narednih zadataka za izvršavanje. Mana algoritma predstavlja i nemogućnost izvršavanja zadataka na master jezgru, usled čega se od početka izvršavanja aplikacije gomilaju gubici. Ovaj tip gubitaka je zanemariv kada je broj jezara velik. Usled povećanja broja jezgara se mogu povećati i gubici usled komunikacije među njima. Napredni algoritmi mogu procesom balansiranja rukovoditi sa više master jezgara prosleđujući podatke o opterećenju, ulazne podatke i o završenom poslu među jezgrima koja učestvuju u procesu balansiranja opterećenja. Problem u radu MS algoritma može predstavljati i otkazivanje rada maseter jezgra, čiju bi funkciju trebalo preuzeti neko od slave jezgara. Broj slave jezgara ne smije biti manji od broja procesa za obradu podataka, jer bi u suprotnom i slave jezgra kojima nije dodijeljen proces za izvršavanje generisala gubitke od početka izvršavanja [57] [107]. Metode raspodjele master-slave algoritmom su se pokazale kao efikasne i zastupljene u praksi, te stoga postoji mnogo istraživanja na temu poboljšanja efikasnosti navedenog algoritma. Vršeni su eksperimenti sa promjenom granularnosti master i

slave procesa kako bi uravnotežili vrijeme utrošeno na obračun i komunikaciju, različit broj i složenost zadataka dodijeljenih slave jezgrima, kao i različit broj korišćénih slave procesa [108] [109] [110]. Vršena su i istraživanja odabira master jezgra: u homogenom okruženju bilo koji procesor može biti izabran kao master ili slave pošto se svi resursi smatraju ekvivalentnim, dok u heterogenom okruženju nejednakost kapacitetima računskih i komunikacionih resursa može proizvesti veoma različita vremena izvršavanja aplikacija u zavisnosti od toga koje je jezgro izabrano za master, a koje za slave jezgro [111]. 4. Kombinovani load balancing algoritmi Domain decomposition i master-slave algoritmi imaju svoje prednosti i mane koje ispoljavaju u zavisnosti od karakteristika resursa na kome se izvršavaju, konkretne paralelne aplikacije za koju se vrši load balancing i trajanja procesa koji se izvršavaju paralelno. Ni jedan od pomenutih algoritama ne daje dobre rezultate u širokom spektru aplikacija i tipova distribuiranih sistema, pa je predmet istraživanja koja su ovdje prezentovana bila ideja da se kombinovanjem pomenutih algoritama poboljšaju performanse paralelizacije bez značajnijeg usložnjavanja algoritma. Kao rezultat su predložena dva nova algoritma za balansiranje opterećenja paralelne aplikacije na distribuiranom računarskom sistemu: kombinovani algoritam [59] i kombinovani adaptivni algoritam [11]. Oni koriste djelove domain decomposition i master-slave algoritama i kreirani su radi povećanja performansi i smanjenja nedostataka u raspodjeli procesa paralelnih aplikacija koje se sastoje od više nezavisnih zadataka. Povećanje efikasnosti i smanjenje neuravnoteženosti se vrši odabirom algoritama raspodjele zadataka u zavisnosti od segmenata u kojima su gubici najmanji i limitiranjem algoritma u trenucima kada prouzrokuje gubitke. 4.1. Kombinovani load balancing algoritam (CA) Kombinovani algoritam (CA) se sastoji od kombinacije domain decomposition i master-slave algoritama koji izvršavaju zadatke u trenucima kada osnovni algoritmi imaju maksimalni učinak. Algoritam se izvršava u tri faze. Rad algoritma započinje kao i kod prethodno opisanih algoritama podjelom ulaznih podataka po domenima ili naredbama i kreiranju lista zadataka za paralelnu obradu. Master-slave algoritam od početka rada generiše gubitke zbog neizvršavanja zadataka na master jezgru i učestale komunikacije među jezgrima. Nasuprot tome, domain decomposition algoritam od početka rada ima maksimalnu efikasnost do trenutka kada jedno od jezgara završi sve dodijeljene zadake nezavisno od heterogenosti i optrećenja distribuiranog resursa. Stoga je domain decomposition odabran za izvršavanje zadataka u prvoj fazi novog algoritma. Nakon završetka rada najbržeg jezgra se šalje signal ostalim jezgrima da obustave rad po završetku zadatka koji izvršavaju u tom trenutku. Tim postupkom se limitiraju gubici u novom algoritmu, a koje bi generisao klasični domain decomposition algoritam. Nakon obustave rada počinje druga faza u kojoj sva jezgra šalju izvještaje o izvršenim i neizvršenim zadacima unaprijed definisanom jezgru koje sumira rezultate. U trećoj fazi se pokreće master-slave algoritam zbog njegovih sposobnosti dinamičkog balansiranja opterećenja na homogenim i heterogenim resursima. Za master jezgro se bira unaprijed definisano jezgro koje sadrži informacije o neizvršenim zadacima i koje ih pokreće na ostalim (slave) jezgrima. Nakon izvršavanja svih preostalih zadataka MS algoritmom, rezultati se sumiraju i sjedinjavaju u postprocessing fazi. Dijagram izvršavanja kombinovanog algoritma je prikazan na slici 10. I faza II faza III faza Domain decomposition Pr eras pod jel a Master-slave Tmin Slika 10. Dijagram izvršavanja kombinovanog algoritma Gubici ovog algoritma se mogu posmatrati po fazama. U prvoj fazi, gdje se izvršava DD algoritam, se stvaraju gubici samo po završetku rada najbržeg jezgra. Gubici se javljaju se samo od trenutka prekida Tmin do završetka zadataka koji se izvršavaju u tom trenutku, a ne do kraja izvršavanja svih dodijeljenih zadataka (kako je kod osnovnog domain decomposition algoritma). Druga faza traje relativno kratko i gubici su zanemarljivi. Gubici u trećoj fazi su opisani kod klasičnog master- slave algoritma. Odnos trajanja prve i treće faze, kao i njihovih gubitaka, zavisi od tipa resursa na kome se paralelni program izvršava i trajanja pojedinačnih zadataka. Gubitke možemo analizirati i shodno tipu resursa i trajanju zadataka: - U idealnom slučaju, kada svi resursi klastera imaju isto opterećenje i sva jezgra istovremeno završavaju dodijeljene zadatke, sva obrada podataka je završena u prvoj

fazi. Prilikom obračuna, u drugoj fazi, se konstatuje da su svi zadaci završeni. MS algoritam u trećoj fazi počinje rad, master jezgro konstatuje da nema zadataka za izvršavanje i obustavlja rad algoritma. - U slučaju izvršavanja na homogenom resursu i kada je trajanje zadataka približno jednako, u prvoj fazi bi bio završen veći dio zadataka. Usled toga bi gubici DD ili master jezgra kod MS algoritma iz treće faze bili minimalni. - U slučaju izvršavanja aplikacije na heterogenom klasteru, homogenom sa različitim opterećenjem nodova ili ukoliko je za izvršavanje zadatka potreban različit vremenski interval, u prvoj fazi će se obraditi manji dio zadataka, a ostali prenijeti u treću fazu radi dimaničkog load balancinga. U opisanom slučaju bi klasični DD algoritam imao velike gubitke. Novi algoritam prekida klasični DD algoritam i vrši obračun i preraspodjelu. Gubici u trećoj fazi ovog algoritma usled rada master jezgra i komunikacije među jezgrima su manji nego kod klasičnog MS algoritma zbog kraćeg trajanja izvršavanja treće faze. 4.2 Kombinovani adaptivni load balancing algoritam (CAA) Kombinovani adaptivni algoritam [11] je poboljšana verzija, prethodno definisanog, CA algoritma kod koje je uveden adaptivni model odlučivanja koji vrši izbor adekvatnog algoritma na osnovu podataka o stanju resursa na kome se izvršava paralelna aplikacija i dužine trajanja zadataka koji se izvršavaju. Algoritam se izvršava u tri ili više faza. U preprocessing fazi se, kao i kod CA algoritma, vrši podjela ulaznih podatka i priprema zadataka za izvršavanje. Prije pokretanja paralelnih simulacija izvršava se i analiza konfiguracije distribuiranog resursa i dobijeni podaci se koriste u kasnijoj analizi. U paralelnoj obradi kombinovanog adaptivnog load balancing algoritma se ističu tri faze izvršavanja (slika 11.): I faza II faza III faza Domain decomposition Od P a rb ei r r aa sl p g o o d ri jt el aa ma Domain decomposition Master-slave Tmin Slika 11. Faze izvršavanja kombinovanog adaptivnog load balancing algoritma • U prvoj fazi kombinovanog adaptivnog algoritma se izvršava domain decomposition algoritam koji pruža najveću efikasnost i najmanje gubitake u početnoj fazi izvršavanja programa. Razlozi odabira algoritma su isti kao i kod, ranije opisanog, kombinovanog algoritma. Algoritam prekida rad kada prvo (“najbrže”) jezgro završi dodijeljeni posao (Tmin) i šalje instrukcije ostalim jezgrima da prekinu rad po završetku zadatka koji obrađuju u tom trenutku. Opisanim postupkom se smanjuju gubici prve faze izvršavanja na minimum. • U drugoj fazi algoritma se, na osnovu količine i trajanja izvršenih zadataka, konfiguracije klastera i njegovog opterećenja, adaptivnim pristupom vrši odabir algoritma za raspodjelu preostalih zadataka. Po iniciranju prekida na kraju prve faze, svako jezgro dostavlja unaprijed predefinisanom jezgru niz koji sadrži trajanje izvršenih zadataka na tom jezgru. Predefinisano jezgro prima poslate nizove i obrađuje ih, praveći niz sa brojem izvršenih zadataka po svakom jezgru i niz izvršenih i neizvršenih zadataka i vrši odabir algoritma koji će se izvršiti u trećoj fazi shodno definisanom algoritmu odlučivanja. Odluka o izboru algoritma se vrši na osnovu sledećih parametara: o homogenosti dodijeljenih resursa, o ukupnog broja dodijeljenih jezgara, o brojeva izvršenih zadataka po svakom jezgru ponaosob i o vremena izvršavanja svakog zadatka pojednačno. Homogenost dodijeljenih resursa, tj ispitivanje da li se oni mogu smatrati homogenim ili heterogenim, se vrši poređenjem vrijednosti performansi dodijeljenih nodova distribuiranih resursa. Mjera performanse pojedinačnog resursa može biti: radni takt jezgra, količina memorije noda ili mrežni protok noda. Zavisno od arhitekture distribuiranog sistema i vrste zadataka, može se uzimati jedan ili više mjera performansi nodova, a u prezentovanim istraživanjima se koristio radni takt jezgara (Hz) kao mjera performansi noda distribuiranog sistema. Ukupan broj dodijeljenih jezgara je definisan prilikom pokretanja aplikacije. Broj izvršenih zadataka po jezgru predstavlja dio ukupnog broja zadataka koji je izvršen do trenutka Tmin, kada je prvo jezgro izvršilo dodijeljene zadatke i iniciralo prekid, za svako jezgro ponaosob. Podatak je izražen kao niz čiji je broj elemenata jednak broju dodijeljenih jezgara, a elementi su brojevi izvršenih zadataka po svakom jezgru pojedinačno. Ukupan broj i vrsta zadataka zavisi od paralelne aplikacije koja se izvršava i ulaznih podataka, a podjela se vrši prije paralelne obrade. Vrijeme izvršavanja svakog pojedinačnog zadataka je matrica koja sadrži podatke na kom je jezgru zadatak izvršen i trajanje svakog zadatka (ms) koji je završen. Na osnovu gore navedenih parametara se mogu definisati uslovi koji

će se koristiti za izbor adekvatnog algoritma raspodjele u trećoj fazi. Radi implementacije u program, ovi uslovi su definisani promenljivima Ui koje imaju binarne vrijednosti. Dakle, promjenljiva Ui uzima vrijednost 1 ukoliko je i-ti uslov ispunjen, a u suprotnom Ui uzima vrijednost 0. Kao prvi i jedini eliminatorni uslov (Ue) za izbor algoritma raspodjele je uslov da je preostali broj zadataka manji ili jednak broju raspoloživih jezgara. U slučaju ispunjenosti uslova Ue (Ue=1) se bira DD algoritam za izvršavanje u trećoj fazi, odnosno svaki od preostalih zadataka se dodjeljuje po jednom jezgru na izvršavanje. Ukoliko eliminatorni uslov nije ispunjen (Ue=0), izbor algoritma se vrši na osnovu kombinacije sledećih uslova: o U1 - uslov homogenosti klastera: ovaj uslov je ispunjen (U1=1) ukoliko su dodijeljena CPU jezgra istog ili približnog radnog takta, tj. ako je standardna devijacija radnog takta svih jezgara manja od zadane vrijednosti; o U2 - uslov broja jezgara: ovaj uslov je ispunjen (U2=1) ukoliko je broj jezgara na kojima se izvršava aplikacija manji od unaprijed definisanog broja jezgara, tj ako se gubici master jezgra kod MS algoritma ne mogu zanemariti; o U3 - uslov ujednačenosti broja izvršenih zadataka: ovaj uslov je ispunjen (U3=1) ukoliko je broj izvršenih zadataka po svakom jezgru približan tj. ako je vrijednost standardne devijacije broja izvršenih zadataka po svakom jezgru manja od unaprijed zadate vrijednosti; o U4 - uslov ujednačenosti trajanja izvršenih zadataka: ovaj uslov je ispunjen (U4=1) ukoliko je trajanje izvršenih zadataka po jezgru približno, tj. vrijednost standardne devijacije vremena izvršenja svakog zadatka po jezgru manja od unaprijed zadate vrijednosti. Algoritam odluke provjerava ispunjenost uslova koji zavise od vrijednosti parametara pa se i sami izbor algoritma prilagođava (adaptira) trenutnim performansama dodijeljenih resursa i stanju izvršenih zadataka u I fazi. Dakle, predloženi adaptivni algoritam određuje da li će se u narednoj fazi izvršavati DD ili MS algoritam na osnovu ispunjenosti definisanih uslova po principu: što je više uslova ispunjeno to determiniše izbor DD algoritma u trećoj fazi i obrnuto. Da bi omogućili dodatno prilagođenje algoritma odluke konkretnoj aplikaciji i distibuiranom sistemu, svaki od uslova se može ponderisati sa realnim koeficijentima Ki, Ki∈[0,1], i tako omogućiti isključivanje nekih uslova ili dodjelu većeg ili manjeg značaja nekim od uslova. Ovo se ne odnosi na eliminatorni uslov koji se nikad ne isključuje i koji se razmatra nezavisno od ostalih uslova. Koeficijentima Ki se dodjeljuje maksimalna vrijednost 1 ukoliko je u potpunosti uzet u obzir taj uslov, dok se sa Ki=0 isključuje uticaj tog uslova iz uticaja na izbor algoritma. Koeficijente je moguće i potrebno definisati posebno za svaku aplikaciju i distribuirani resurs u zavisnosti od prethodno dobijenih rezultata i iskustava. Konačno, na osnovu navedenih uslova možemo definisati funkciju odluke na osnovu koje vršimo izbor algoritma u trećoj fazi: Treba definisati i “pre$U$?lom=n∑ui?”=v1rK?ije$K$?d∗nU?oU?st. funkcije odluke U na osnovu(k1o5je) 4 se vrši odabir jednog ili drugog algoritma za treću fazu (DD ili MS). Obzirom da se maksimum funkcije U postiže ispunjenostima uslova i da to determiniše izbor DD algoritma onda se za prelomnu vrijednKois∗t U(“iprag”) uzima polovina maksimalne vrijednosti funkcije U, tj. ∑i?=1 Ki 4 P = Stoga, ukoliko je zadovoljeno 2 . (16) U? ≥ P? (17) potrebno je odabrati DD algoritam u trećoj fazi, odnosno MS algoritam ukoliko uslov (17) nije zadovoljen. Slika 12. prikazuje shemu algoritma odlučivanja koji vrši odabir u drugoj fazi. Kao sto je već navedeno, na osnovu predstavljenih parametera, definisanih uslova i koeficijenata se vrši odabir algoritma za raspodjelu zadataka u trećoj fazi. START Ulazni parametri za odlučivanje, koeficijenti Ki Ue Ue=1 NE U1, U2, U3, U4 D A 4 U = ? Ki ∗ Ui i=1 P= ∑i?=1 Ki 4 2 DD u III fazi DA U≥P NE MS u III fazi END Slika 12. Shema algoritma na osnovu koga se vrši odabir u toku druge faze • Odabrani algoritam (DD ili MS) se izvršava u trecój fazi. U slučaju odabira DD algoritma, svako jezgro prima dio liste nedovršenih zadataka. Svakom jezgru se dodjeljuje po jedan od preostalih zadataka na rješavanje ukoliko je preostali broj zadataka manji ili jednak dostupnom broju jezgara (uslov Ue). U protivnom, broj dodijeljenih zadataka za svako jezgro se određuje srazmjerno broju zadataka završenih u prvoj fazi na svakom jezgru ponaosob. U slučaju izbora MS algoritma se kao master jezgro određuje jezgro koje je vršilo analizu u drugoj fazi. Ono sadrži informacije sa listom svih nedovršenih zadataka koja se dodjeljuju slave jezgrima na izvršenje u trecój fazi algoritma. Shodno prekidu izvršavanja prve faze,

analizi stanja resursa, adaptaciji iz druge faze i preraspodjeli zadataka, predloženi CAA algoritam će povecáti efikasnost i skratiti vrijeme izvršenja djelova paralelne aplikacije u trecój fazi. Efikasnost CAA algoritma je poboljšana zbog preraspodjele procesa, smanjenog vremena čekanja jezgara na nove instrukcije i poboljšanja korištenja resursa prilagođenjem raspodjele arhitekturi distribuiranog sistema i konkretnoj aplikaciji. Stoga će vrijeme izvršenja predloženog algoritma biti kracé nego vrijeme izvršenja standardnog DD algoritma ukoliko je mjereno u istim uslovima. CAA algoritam je sličan CA algoritmu u slučaju donošenja odluke da je u trećoj fazi potrebna dinamička raspodjela procesa uz MS algoritam. Nedostatke predloženog CAA algoritma predstavljaju prekid izvršavanja zadataka na kraju prve faze i trajanje adaptacije u drugoj fazi. Prekid izvršavanja zadataka u prvoj fazi može povecáti trajanje ove faze ukoliko postoji jedan ili više zadataka čije je trajanje značajno duže od trajanja ostalih zadataka. Ova pojava bi prouzrokovala povecánje trajanja prve faze koje može uticati na performanse čitavog algoritma. U tom slučaju bi efikasnost bila ista kao i kod klasičnog DD algoritma. Druga faza, zbog kratkog trajanja, ne može značajnije uticati na ukupnu efikasnost paralelne aplikacije. Predloženi CAA radi kao DD algoritam u toku perioda njegove maksimalne efikasnosti i prekida rad kada njegova efikasnost počinje opadati. Predloženi adaptivni algoritam će imati značajno bolji učinak od domain decomposition algoritma u slučaju kada osnovni algoritam ima nisku efikasnost usled prekida i preraspodjele zadataka. CAA algoritam će imati bolje performanse od MS algoritma jer MS algoritam ne izvršava zadatke na master jezgru i generiše više gubitaka usled komunikacije nego predloženi CAA algoritam. MS algoritam će imati slabiju efikasnost od predloženog algoritma jer isti počinje kao DD algoritam i vrši preraspodjelu i odabir algoritma za izvršavanje na osnovu parametara radi postizanja bolje upotrebe resursa i efikasnosti. U slučaju potrebe zbog velikih gubitaka, u trećoj fazi je moguće je ponovo inicirati prekid i ponavljanje algoritma odlučivanja, odnosno adaptacije na osnovu novih parametara, ponovnog odabira algoritma i njegovo pokretanje da bi se dobilo što bolje iskorištenje resursa. 5. Uporedna analiza performansi predloženih load balancing algoritama U ovom poglavlju je predstavljena analiza performansi kombinovanog (CA) i kombinovanog adaptivnog (CAA) load balancing algoritma i komparacija performansi novonastalih algoritama sa performansama algoritama koji ih sačinjavaju. 5.1. Testno okruženje Za potrebe istraživanja i testiranja predmetnih algoritama korišćena je paralelna verzija simulatora performansi krosbar komutatora (CQ), kao numerički zahtjevnog primjera paralelne aplikacije sa više nezavisnih procesa. Algoritmi su testirani na različitim distribuiranim računarskim okuženjima i pokretani pod različitim opterećenjima resursa. Svaka simulacija je izvršena deset ili više puta, a ovdje su predstavljeni usrednjeni rezultati vremena izvršavanja. 5.1.1. Paralelni simulator performansi krosbar komutatora (CQ) paketa Komutatori paketa predstavljaju važan segment u savremenim računarskim komunikacijama. Krosbar arhitektura komutatora paketa je zastupljena prilikom dizajna modernih komunikacionih uređaja projektovanih za prenose podataka velikim brzinama.

Privremeno smještanje paketa koji čekaju **da budu** isporučeni **na odredište** se **vrši na ulazu komutatora, na njegovom izlazu ili u kombinaciji ulaza i izlaza.** Mogućnost **da se paketi skladište unutar samog komutatora paketa** u prošlosti **nije** 6

razmatrana, jer su tehnološka ograničenja onemogućavala implementaciju velikih bafera na istom čipu sa komutatorom. Početkom 21. vijeka je dokazano da se korišćenjem moderne tehnologije mogu implementirati baferi većeg kapaciteta na istom čipu sa komutatorom.

Komutator kod kojeg se baferovanje pristiglih paketa vrši samo u ukrsnim tačkama **komutacione matrice** naziva **se CQ (Crosspoint Queued) komutator.** 6

Analiza performansi komutatora paketa se procjenjuje sledećim parametarima: propusnost, srednje kašnjenje i vjerovatnoća gubitka paketa. U radovima [112] [113] [114]

su analizirane performanse CQ komutatora paketa sa više algoritama raspodjele, **za** različite **dužine bafera u ukrsnim** tačkama i **u** različitim **uslovima dolaznog** 6

saobraćaja. Analiza performansi je izvršena uz pomoć simulatora za različite karakteristike

CQ komutatora [12]. **Aplikacija za analizu performansi CQ komutatora sastoji se iz dva dijela – generatora uniformnog saobracaja i simulatora CQ komutatora.** 3

Generator saobraćaja generiše uniformni saobraćaj za različita ulazna opterećenja, odnosno vjerovatnoće pojavljivanja paketa. Simulator vrši analizu za 12 različitih veličina bafera, 1

8 algoritama (LQF, RR, ERR, FBRR, EELQF, ELQF, FBLQF i RAND) i 32 fajla generisanog uniformnog saobraćaja. Time se 1

u preprocessing fazi generišu do 3072 nezavisna procesa. Paralelna aplikacija CQ simulatora se bazira na podjeli ulaznih podataka na nezavisne procese, a zatim njihovom dodjeljivanju CPU jezgrima na izvršavanje. Prva paralelna vezija se zasnivala na raspodjeli procesa DD algoritmom. Za paralelizaciju su korištene metode razmjene poruka (MPI) i

hibridni model (MPI i OpenMP). Paralelne simulacije su dale kraće vrijeme izvršavanja, čime je ostvaren glavni benefit paralelizacije. Međutim, primijećeno je odstupanje u parametrima kojima su mjerene performanse tokom izvršavanja paralelne simulacije. Mjerenje je vršeno u različitim uslovima i na različitim resursima. Pregled vremena izvršavanja pojedinačnih procesa je potvrdio zabilježena odstupanja, šta je ukazalo na mogućnost implementacije dinamičkih algoritama za raspodjelu procesa. U narednim paralenim verzijama je implementirana raspodjela MS algoritmom, a kasnije su korišćeni i predloženi CA i CAA algoritmi. Generator saobraćaja i CQ simulator imaju mogućnost generisanja različite količine ulaznih podataka i širok dijapazon simulacija različitog vremena trajanja pojedinačnih procesa koristeći različite vrijednosti parametara (dimenzija komutatora, broj zahtjeva, algoritama, dužina bafera…). Stoga se opisana paralelna simulacija pokazala kao dobar primjer za implementaciju i testiranje load balancing algoritama paralelne aplikacije koja se sastoji od većeg broja nezavisnih procesa. Paralelne simulacije su vršene za komutatore (matrice) različite dimenzije i sa različitim brojem zahtjeva (time slot-ova). U preprocessing fazi svake simulacije je generator uniformnog saobraćaja generisao ulazne fajlove. Izvođeno je više izvršavanja paralelnog CQ simulatora sa predstavljenim load balancing algoritmima pri različitim opterećenjima klastera. Testiranja su vršena za komutarore sa 16, 32 i 64 porta. U simulacijama su vršeni proračuni u matricama dimenzija komutatora, tako da je dupliranjem broja portova očekivano četiri puta duže trajanje pojedinačnih simulacija. Većina simulacija je vršena za 1.000.000 vremenskih zahtjeva (time slot) kao i u [12]. Određeni broj simulacija je vršen na manjem broju vremenskih zahtjeva usled nedostatka resursa i velikog broja ponavljanja sumulacija. 5.1.2. Testni resursi Paralelne verzije CQ simulatora sa navedenim algoritama raspodjele su testirane na resursima različite heterogenosti, sa i bez uticaja spoljnih faktora na opterećenje resursa i uz korištenje do 128 jezgara. Testiranje je realizovano na izdvojenom lokalnom serveru i na dva distribuirana računarska klastera. • Server HP Proliant BL685c G6 je imao dva šestojezgarna procesora i na njemu je testirana paralelizacija DD algoritmom. Izvršeno je poređenje dobijenih rezultata paralelizacije sa rezultatima izvršavanja serijske aplikacije. Ulazni fajlovi su se nalazili na lokalnom disku servera. Navedeni server se može smatrati kao homogeni distribuirani resurs na kojem nije bilo spoljih ili dodatnih uticaja koji bi mogli promijeniti opterećenje jezgara tokom izvršavanja, tako da se može smatrati za idealno okruženje za izvršavanje paralelne aplikacije. • HPCG klaster

se nalazi u Institutu za informacione i komunikacione tehnologije u Sofiji u Bugarskoj. **Klaster se** 1

u trenutku simulacija sastojao od 36 servera sa po dva Intel X5560 četvorojezgarna procesora, šta uz uključen hyperthreading daje 576 jezgara objedinjenih u HPC klaster [115]. Nodovi su međusobno povezani InfiniBand mrežom. Klaster pripada grupi homogenih klastera, ali se zbog različitog opterećenja nodova usled izvršavanja više različitih aplikacija istovremeno i komunikacije među njima može nekad posmatrati i kao heterogeni. Ulazni fajlovi korišteni u simulacijama su bili smješteni na storage-u klastera i njihov prenos je uračunat u izvršavanje simulacije. Testiranja performansi algoritama su vršena na 64 jezgra uz različite konfiguracije broja servera i broja angažovanih jezgara po serveru. • Paradox HPC klaster [116] se nalazi na Insitutu za Fiziku Univerziteta u Beogradu. Klaster se u trenutku vršenja simulacija sastojao od 106 računskih nodova baziranih na dva osmojezgarna Xeon 2.6GHz procesora sa 32GB RAM i NVIDIA® Tesla ™ M2090 karticama. Nodovi su međusobno povezani InfiniBand mrežom. Ulazni fajlovi za simulacije su bili

smješteni na storage-u klastera, ali je zbog primjetnog zagušenja klastera prilikom izvršavanja bilo neophodno korigovati simulacije i izvršiti prenos svih fajlova prije izvršavanja. Zbog izvršenih korekcija vrijeme prenosa fajlova nije uticalo na vrijeme izvršavanja simulacije. Testiranja performansi algoritama su vršena na 16, 32, 64 i 128 jezgara. Simulacije na HPCG i Paradox klasterima su izvršene tokom HP SEE projekta finansiranog u sklopu FP7 poziva Evropske Unije [13]. 5.2 Analiza performansi paralelizacije DD algoritmom Performanse paralelizacije korišćenjem domain decomposition algoritma mjerene su na pomenutom Blade serveru HP Proliant BL685c G6 izvršavanjem paralelne aplikacije CQ simulatora i upoređivane sa performansama serijske aplikacije. Vršena su mjerenja vremena izvršavanja paralelne aplikacije i na osnovu toga izračunate vrijednosti ubrzanja i efikasnosti. Pored toga, izvršena je analiza vremena izvršavanja paralelene aplikacje u zavisnosti od količine podataka koji su obrađivani [117].

Simulator CQ komutatora je preuzimao **fajlove generisanog uniformnog saobraćaja i** izvršavao aplikaciju **za različite dužine bafera i algoritme** 3
komutatora. **Simulator**

je u preprocessing fazi generisao 3072 nezavisna zadatka. Na početku izvršavanja paralelnog dijela je izvršena podjela DD algoritmom i svakom raspoloživom jezgru je dodijeljen isti ili približan broj zadataka na izvršavanje. Nakon toga je obustavljana komunikacija sa jezgrima do kraja izvršavanja dodijeljenih zadataka. Na kraju simulacija je mjereno vrijeme potrebno za izvršenje simulacije i vršen proračun ubrzanja i efikasnosti. U prvoj simulaciji je vršena analiza za komutatore sa N=16, 32 i 64 porta i Z=100.000 vremenskih slotova. Vrijeme izvršavanja, ubrzanje i efikasnost su prikazani na slikama 13, 14 i 15. Slika 13. Vrijeme izvršavanja paralelnog CQ simulatora uz DD algoritam za 16,32 i 64 porta i 100.000 vremenskih slotova Slika 14. Ubrzanje paralelnog CQ simulatora uz DD algoritam za 16, 32 i 64 porta i 100.000 vremenskih slotova Slika 15. Efikasnost paralelnog CQ simulatora uz DD algoritam za 16, 32 i 64 porta i 100.000 vremenskih slotova Grafikon vremena izvršavanja (slika 15) prikazuje značajano smanjenje vremena paralelne obrade usled povećanja broja jezgara na kojima se simulacija izvršava. Ubrzanje i efikasnost aplikacije rastu povećanjem količine informacija koja se analizira i približavaju se maksimumu. Efikasnost simulacije na 16 portova je manja od 90% zbog kratkog trajanja izvršavanja pojedinačnih zadataka. Nasuprot tome, efikasnost se povećava kod analize na 32 i 64 porta zbog veće količine podataka koja prouzrokuje duže trajanje simulacije. Primijećeno je manje ubrzanje pri izvršavanju na 2, 3, 4 i 6 jezgara zbog većeg opterećenja jezgara procesora na kojima se izvršava obrada i

čekanja ostalih jezgara da ona koja su više **opterećena završe svoje zadatke. Pri izvršavanju na 12 jezgara** 3

je zabilježena efikasnost veća od 98% u svim simulacijama i skoro linearno ubrzanje. Efikasnost preko 90% pokazuje male gubitke i ravnomjernu raspodjelu po jezgrima. Druga grupa simulacija je izvršena za 16-to portni

komutator sa 100.000, 200.000 i 400.000 vremenskih slotova. Rezultati 3

vremena izvršavanja, ubrzanja i efikasnosti su grafički prikazani na slikama 16, 17 i18. Slika 16. Vrijeme izvršavanja paralelnog simulatora komutatora

sa 16 portova i 100.000, 200.000 i 400.000 vremenskih slotova 3

Slika 17.

Ubrzanje paralelne apliakcije za komutator sa 16 portova i 100.000, 200.000 i 400.000 vremenskih slotova 3

Slika 18. Efikasnost

paralelne apliakcije za komutator sa 16 portova i 100.000, 200.000 i 400.000 vremenskih slotova 3

Paralelizacija simulatora 16-to portnog komutatora sa promjenljivim brojem slotova je pokazala slične performanse paralelizacije simulatora sa promjenljivim brojem portova. Kao i u prvoj grupi simulacija, primjetno je povećanje ubrzanja prilikom povećanja broja jezgara na kojima se simulacija izvršava i približavanje linearnoj skali. Takođe, primjetno je povećanje ubrzanja prilikom povećanja količine podataka koji se prosleđuju na obradu. Gubici u izvršavanju simulacije uz domain decomposition algoritam su minimalni zbog toga što su svi zadaci izvršavani na homogenom resursu na kom nije bilo spoljnih uticaja na opterećenje resursa, odnosno izdvojenom serveru i nije bilo kašnjenja prouzrokovanog zbog komunikacija preko mrežnih uređaja. 5.3. Analiza performansi kombinovanog algoritma Performanse kombinovanog algoritma (CA) su testirane na primjeru paralelnog CQ simulatora na HPCG klasteru. Simulacije su vršene za 16-portni CQ komutator i 1.000.000 vremenskih zahtjeva po ulaznom fajlu [59]. Kao i kod prethodnog primjera, u preprocessing fazi je generisano 3072 zadatka koji su tokom izvršavanja dodijeljeni jezgrima. Rezultati performansi kombinovanog algoritma su upoređeni sa rezultatima paralelizacije pomoću domain decomposition i master slave algoritma. Paralelni simulator je pokretan koristeći iste ulazne podatke na 64 CPU jezgra uz

izvršavanje na 4, 8 ili 16 jezgara po serveru. Prosječno vrijeme izvršenja simulacija za 3 razmatrana algoritma raspodjele zadataka i standardna devijacija vremena izvršavanja su prikazani na slici 19. Slika 19. Prosječno vrijeme izvršavanja paralelne simulacije za razmatrane algoritme Simulacije su pokazale da je najduže trajanje izvršavanja paralelne aplikacije pomoću DD algoritma koji pokazuje slabosti koje se javljaju prilikom statičke raspodjele zadataka. Master-slave algoritam je pokazao bolje rezultate od domain decomposition algoritma u svim segmentima. Kombinovani algoritam je pokazao bolje ili iste performanse kao MS algoritam, pogotovo pri izvršavanju na 8 ili 16 servera. Najveća devijacija vremena izvršavanja je zabilježena kod DD algoritma zbog razlike u vremenima izvršavanja najbržeg i najsporijeg jezgra, odnosno ukupnog izvršavanja zadataka koji se na njima izvršavaju. Najbolji rezultati paralelizacije uz sva tri algoritma su zabilježeni pri izvršavanju paralelne simulacije na 8 servera sa 8 jezgara. Izvršavanje na 16 servera sa po 4 jezgra je pokazalo gubitke usled veće komunikacije između servera nego u ostalim primjerima simulacija. Najvecá standardna devijacija vremena izvršenja je uočeno kod izvršenja simulacije na 16 servera zbog različite opterećenosti i potrebe za transferom velike količine ulaznih podataka. Simulacije koje su izvršena na 4 servera koristeći svih 16 jezgara po serveru su iziskivale najviše vremena izvršavanja zbog maksimalnog opterećenja na svim jezgrima servera. U okviru istraživanja je detaljno analiziran i jedan od slučajeva pokretanja simulatora sa algoritmima raspodjele na 64 jezgra, koristeći 16 servera sa po 4 jezgra.

Izvršena je analiza vremena izvršavanja zadatka **na svakom jezgru, a ne samo na** najbržem i **najsporijem** 1

koje su ključni za rad navedenih algoritama. Tabela 1. prikazuje vrijeme izvršavanja tri algoritama raspodjele i iskorišćenosti resursa na primjeru paralelnog CQ simulatora. DD algoritam je za raspodjelu i izvršenje zadataka iziskivao najduže vremena. MS i CA algoritmi su pokazali bolje rezultate i smanjili vrijeme izvršenja do 48%. Slabljenje usled angažovanja master jezgra je bilo zanemarljivo jer se izvršavanje pokrenulo na 64 jezgra. Tabela 1. Vrijeme izvršenja paralelnog CQ simulatora na 64 jezgra Algortam raspodjele Domain decomposition Master slave Kombinovani algoritam Vrijeme izvršenja [s] 1889,86 1016,85 991,69 Iskoristivost 59,10% 96,07% 96,29% Na slici 20. je prikazano vrijeme izvršavanja zadataka na svakom jezgru pojedinačno koristeći DD algoritam raspodjele. Ukupno vrijeme izvršavanja, odnosno izvršavanja procesa najsporijeg jezgra, je iznosilo 1889,86 sekundi, dok je najbrže jezgro završilo dodijeljene zadataka za 512,68 sekundi. Vrijeme izvršavanja dodijeljene grupe zadataka

na svakom jezgru je variralo u zavisnosti od trenutne opterećenosti servera na kome se jezgro nalazi, opterećenosti storage-a sa kog se učitavaju ulazni fajlovi, 1

komunikacija pri učitavanju ulaznih podataka i dužine izvršavanja samih dodijeljenih zadataka. Plavom bojom označeno je vrijeme koje svako jezgro provede

čekajući da preostala **jezgra završe** dodijeljene zadatka **i da se okonča rad** paralelne **aplikacije. Kod ovog primjera** ukupno **procesorsko vrijeme iznosi 33,57 CPU sati, dok je vrijeme korisnog rada procesora 19,84 CPU sati, tj.** iskoristivost je bila **svega 59%. Trajanje pojedinačnih** zadataka **je variralo od 9,71 s do 46,95 s.** 1

Slika 20. Dužina trajanja zadataka po svakom jezgru prilikom izvršavanja simulacije koristeći DD algoritam Na slikama 21. i 22. prikazana je raspodjela vremena izvršenja zadataka na svakom jezgru korišćénjem master-slave algoritma i kombinovanog algoritma. Slika 21. Dužina trajanja zadataka po svakom jezgru prilikom izvršavanja simulacije koristeći MS algoritam Slika 22. Distribucija izvršenja zadataka po jezgrima i po fazama prilikom izvršavanja simulacije koristeći CA algoritam Raspodjela zadataka MS algoritmom je skratila trajanje paralelnog CQ simulatora na 1016,85 sekundi ili 18,08 CPU sati sa iskoristivošću 96,07%. Master jezgro se koristilo samo za distribuciju procesa i komunikaciju sa drugim jezgrama i izazvalo je gubitak od 0,28 CPU sati ili 1.56% iskoristivosti. Preostalih 2,37% gubitka iskoristivosti je utrošeno na gubitke usled komunikacije i čekanje jezgara da najsporije jezgro završi poslednji dodijeljeni zadatak. Vrijeme izvršenja paralelnog CQ simulatora korišćénjem kombinovanog algoritma (CA) je iznosilo 991,69 sekundi ili 17,63 CPU sati. Iskoristivost klastera je povecána na

96,29%. Izvršavanje programa **je** podijeljeno **u tri faze. U prvoj fazi je** 1

simulacija pokrenuta koristecí algoritam dekompozicije domena. Najbrže jezgro je završilo dodijeljene zadatke za 515,92 sekundi i poslalo signal ostalima za prekid rada. U drugoj fazi (ilustrovano crvenom bojom) preostala jezgra su dobila signal za prekid rada, završila zadatke koji su obavljani u tom trenutku, izvršila sinhronizaciju i poslale izvještaj o nedovršenim zadacima prethodno definisanom jezgru. Prekid DD algoritma i trajanje druge faze su iznosili do 21,08 sekundi i zabilježeni su gubici od 0,23 CPU sati ili 1,30% iskorišćénosti. Mjerenja su pokazala da maksimalno trajanje prekida (faza 2) iznosi koliko trajanje najdužeg zadatka koji se odvija u vrijeme slanja signala za prekid izvršenja. Nakon toga, aplikacija je nastavila izvršavanje koristecí master-slave algoritam (faza 3) i završila sve zadatke za 454,69 sekundi. Gubici master jezgra u fazi 3

su smanjeni sa 0,28 CPU sati (1,56%) **iz** standardnog **master-slave** algoritma **na 0,12 CPU sati** 1

ili 0,68%. Ove uštede bi bile vecé u primjerima gdje dio MS algoritam (faza 3) radi u kracém vremenskom intervalu ili u primjerima sa DD algoritmom koji su se izvodili na manji broj jezgara. Preostalih 1,73% gubitaka utrošeno je na komunikaciju u trećoj fazi i čekanje jezgara da najsporije završi poslednji dodijeljeni zadatak.

Numeričkim primjerom je potvrđena ušteda u vremenu izvršavanja prilikom korištenja 1

kombinovanog algoritma zbog vršenja raspodjele

tokom kritičnog dijela aplikacije. Potvrđeno je da prekid aplikacije značajno **ne utiče na ukupno vrijeme izvršavanja.** 1

5.4. Analiza performansi kombinovanog adaptivnog algoritma Performanse kombinovanog adaptivnog algoritma su verifikovane na primjeru 16-to portnog CQ simulatora sa 1.000.000 zahtjeva i 3072 generisana zadatka. Simulacije izvršene na Paradox HPC klasteru Insituta za Fiziku u Beogradu. Performanse kombinovanog adaptivnog algoritma su poređene sa performansama algoritama koji ga sačinjavaju. U simulacijama je data analiza ukupnog trajanja simulacije uz raspodjelu CAA algoritmom i vremena izvršavanja po jezgrima pojedinačno. Utvrđena je zavisnost trajanja paralelne simulacije od trajanja pojedinačnih zadataka, pogotovo onih čije je trajanje višestruko premašivalo trajanje prosječnog zadatka. Izvršena je analiza pojedinačnog trajanja zadataka zbog razlike u trajanju izvršavanja simulacija. Rezultati su predstavljeni na slici 23: Slika 23. Trajanje pojedinačnih zadataka CQ simulatora Prosječno vrijeme izvršenja pojedinačnih zadataka je iznosilo 15,10 sekundi. Trajanje 95% zadataka je iznosilo između 12 i 18 sekundi, dok je ostalih 5% zadataka trajalo duže od 18 sekundi. Najduži zadatak se izvršavao 165 sekundi. Statistika se zasniva na uzorku od 800.000 izvršenih zadataka. Ukupno vrijeme izvršenja simulacija zavisi od trajanja svakog zadatka pojedinačno i njihovog rasporeda izvršavanja. Simulacije pomoću domain decomposition, master-slave i kombinovanog adaptivnog algoritma su izvršavane na 16, 32, 64 i 128 jezgara. Ulazni fajlovi su kopirani na nodove na kojima su pokretane simulacije u preprocessing fazi i time se smanjio uticaj komunikacije između nodova. U prezentovanim simulacijama je za uslov U1 korišena vrijednost standardne devijacije 10% od prosječne vrijednosti radnog takta jezgara. Za uslov U2 je definisan prag od 32 jezgra. Za uslove U3 i U4 je vrijednost standardne devijacije 25%. Koeficijenti korišćeni u ovim simulacijama su K1=0, K2=1, K3=1 i K4=0,5. Prioritet u odlučivanju je dat broju jezgara na kojima se simulacija izvršava i broju izvršenih zadataka po jezgru. Manji prioritet je dat dužini trajanja zadataka, a koeficijentom K1=0 je eliminisan uticaj homogenosti klastera. Prosječni rezultati izvršavanja paralelne aplikacije uz DD, MS i CAA algoritam za različit broj korišćenih jezgara prikazani su

u tabeli 2. i na slici 24. **Tabela 2. Vrijeme izvršavanja** [s] DD, MS **i** 13

CAA algoritama na 16-128 CPU jezgara Broj jezgara 16 32 64 128 DD 2786,55 1423,29 742,07 388,33 Algoritam MS 3367,09 1670,68 849,28 456,62 CAA 2735,26 1398,76 717,48 356,47 Slika 24. Prosječno vrijeme izvršenja simulacija pomoću DD, MS i CAA algoritma na 16-128 jezgara Kombinovani adaptivni algoritam je izvršio simulacije brže nego domain decomposition i master-slave algoritma u svim uslovima. Najbolji rezultati i najveći benifiti usled preraspodjele zadataka su utvrđeni u slučajevima izvršavanja simulacija na većem broju jezgara. Simulacije su pokazale najduže vrijeme izvršavanja uz master-slave algoritam, pogotovo na malom broju jezgara zbog njegovih ranije opisanih nedostataka. Na HPCG klasteru su simulacije uz master-slave algoritam izvršene brže nego uz domain decomposition algoritam raspodjele. Vrijeme transfera ulaznih fajlova na nodove je bilo uključeno u ukupno vrijeme simulacije, pa je zbog različitog vremena prenosa generisano različito opterećenje i trajanje pojedinačnih zadataka. Stoga je dinamički load balancing na HPCG klasteru pokazao bolje rezultate. Na Paradox klasteru je domain decomposition algoritam izvršavao simulacije brže nego master-slave algoritam. Transfer ulaznih podataka je vršen prije izvršavanja simulacija i većina zadataka se izvršila za približno isto vrijeme, šta je prikazano na slici 24. Zbog toga se statička raspodjela na Paradox-u pokazala kao dovoljna i domain decomposition algoritam je pokazao bolje performanse nego master-slave algoritam. Slika 25. prikazuje uštedu u vremenu izvršavanja između kombinovanog adaptivnog algoritma i algoritama koji ga sačinjavaju. Domain decomposition algoritam je zahtijevao više vremena od kombinovanog adaptivnog algoritma zbog statičke raspodjele tokom cijelog procesa izvršavanja. Razlika između kombinovanog adaptivnog i domen decomposition algoritma se krecé od 1,7% do 8,2%. Najveća razlika je zabilježena prilikom izvršenja aplikaicje na 128 jezgara. Slika 25. Uštede tokom izvršavanja alogitama i poredjenje izmedju kombinovanog algoritma i domain decomposition i master slave Razlike između kombinovanog adaptivnog algoritma i master-slave algortima su nastale zbog gubitaka master-slave algoritma usled raspodjele zadataka i komunikacije između jezgara tokom cijelog procesa izvršavanja programa. Razlika u vremenu izvršavanja između kombinovanog adaptivnog i master-slave algoritma se krecé od 15,5% do 21,9%. Nemogucńost izvršavanja zadataka na master jezgru je proizvela gubitke tokom izvršavanja na manjem broju jezgara. Povećana komunikacija između jezgara tokom čitavog izvršavanja simulacije je prouzrokovala najveću razliku između rezultata navedenih na 128 jezgara. Na slici 26. je prikazano vrijeme izvršavanja CAA algoritma po fazama u zavisnosti od broja jezgara na kojima se aplikacija izvršava i na osnovu gubitaka koji su zabilježeni. Prikazano vrijeme izvršavanja je grupisano u tri segmenta: 1. prva faza DD algoritma do završetka rada najbržeg jezgra (Tmin), 2. gubici iz rada I faze (od Tmin do kraja rada I faze) i II faza (faza odabira) i 3. trecá faza rada CAA algoritma. Gubici usled prekida prve faze, faze odabira i gubici iz trecé faze su se povecávali kako se simulacija izvršavala na vecém broju jezgara. Trajanje prekida prve faze i faze odabira je označeno crvenom bojom i najduže je trajalo pri izvršavanju na 128 jezgara jer je najbrže jezgro najduže čekalo preostala jezgra da završe zadatke koji su tada izvršavani. Uočeni su mali gubici uz raspodjelu CAA i DD algoritmom prilikom izvršavanja na 16 jezgara. Iako su zabilježeni gubici CAA algoritma pri izvršavanju na 64 i 128 jezgara, izvršene su simulacije za kraći vremenski period usled velikog broj zadataka koji su redistribuirani. Slika 26. Utrošeno vrijeme kombinovanog adaptivnog algoritma u zavisnosti od broja jezgara Slika 27. prikazuje rezultate izbora algoritma u drugoj fazi shodno primljenim i analiziranim podacima i donesenim odlukama na kraju druge faze. Domain decomposition algoritam je izabran u vecíni slučajeva kada je simulacija izvršena na 16 jezgara, jer je detektovano izvršavanje na manje od 32 jezgra i ravnomjernim brojem zadataka koji su trebali biti redistribuirani. S druge strane, master-slave je izabran u slučajevima simulacija na 32 ili više jezgara jer je algoritam odluke iz druge faze na osnovu parametara otkrio broj dostupnih jezgara, različit broj i trajanje izvršenih zadataka i odabrao ovaj dinamički algoritam za trecú fazu. Slika 27. Izabrani algoritam u trećoj

fazi CAA ZAKLjUČAK Distribuirani računarski sistemi i paralelno procesiranje omogućavaju brže izvršavanje komplikovanih računarskih zadataka i pružaju napredak u svim naučnim oblastima i oblastima industrije. U cilju izvršenja zadataka u što kraćem vremenskom intervalu, povećanja efikasnosti i što veće iskoristivosti resursa, koriste se load balancing algoritmi za raspodjelu zadataka paralelnih aplikacija. U tezi je predložen originalni adaptivni load balancing algoritam za paralelne aplikacije koji kombinuje rad statičkih i dinamičkih algortama. Kao osnov za predloženi algoritam su korišćeni domain decomposition i master slave algoritmi, kao jedni od najzastupljenijih algoritama u praksi. U radu je analiziran njihov rad i identifikovane prednosti i nedostaci. Kako ni jedan od algoritama ne daje dobre rezultate u širokom spektru aplikacija i tipova distribuiranih sistema, istraživanje se baziralo na ideji kombinovanja pomenutih algoritama u cilju poboljšanja performansi paralelizacije bez značajnijeg usložnjavanja algoritma. Na osnovu identifikovanih prednosti i nedostataka standarnih algoritma je predložen kombinovani adaptivni algoritam. Ideja kombinovanih algoritama je da rade u fazama kada sastavni algoritmi imaju najbolje performanse. Prednosti predloženog rješenja su sledeće: a) poboljšana efikasnost paralelne aplikacije i iskoristivost klastera u odnosu na osnovne algoritme zbog redistribucije zadataka i smanjenog vremena izvršavanja; b) identifikovani su parametri i uslovi za odabir algoritma koji shodno stanju resursa i stepena izvršenja aplikacije adaptivnom strategijom određuju adekvatniju statičku ili dinamičku raspodjelu procesa; c) definisani su koeficijenti za ponderisanje kojima je moguće izvršiti prilagođavanje adaptivnog load balancing algoritma u zavisnosti od infrastrukture i paralelne aplikacije; d) primjenljivost predloženog adaptivnog dijela algoritma odlučivanja je moguće u bilo kom load balancing algoritmu i e) predloženi algoritam je primjenljiv

na sve paralelne aplikacije koje se sastoje od više nezavisnih zadataka. U disertaciji **su** prikazani **rezultati izvršavanja**	1

domain decomposition, master-slave, kombinovanog i kombinovanog adaptivnog algoritma na različitim računarskim resursima uz pomoć numerički zahtjevne paralelne aplikacije CQ simulatora. Poređenje rezultata simulacija uz različito opterećenje i konfiguracije distribuiranih resursa potvrdjuje bolje performanse predloženog algoritma u odnosu na osnovne algoritme razmatrane u radu. U radu su navedene situacije u kojima predloženi algoritam pruža poboljšanja ili daje performanse uporedive sa algoritmima koji su njegov integralni dio. Buduće analize i poboljšanja algoritama je moguće nastaviti u nekoliko pravaca. Kao što je već rečeno, kombinovani algoritmi se zasnivaju na prekidu prve faze u trenutku kada počinje da stvara gubitke. Istraživanje se može dopuniti analizom da li je u slučaju izvršenja većeg dijela zadatka u prvoj fazi potrebno inicirati prekid i završiti raspodjelu zadataka po predloženim algoritmima ili je u nekim slučajevima efikasnije izbjeći dodatne operacije i izvršiti raspodjelu zadataka statičkim algoritmom. Nasuprot tome, kada je u prvoj fazi izvršen manji dio zadataka, moguće je ispitati svrsishodnost dva ili više prekida i gubitaka koji se time stvaraju u cilju promjene metoda raspodjele i boljeg balansa opterećenja. U tezi su predstavljeni parametri, uslovi i koeficijenti na osnovu kojih je moguće definisati uslov i prag odluke kombinovanog adaptivnog algoritma. Istraživanje se može dopuniti novim parameterima i uslovima za odlučivanje i izvršiti analiza povezanosti koeficijenata za odabir i ponderisanje uslova od broja zadataka i karakteristika resursa. Mogući pravac za nastavak istraživanja predstavlja mogućnost implementacije kombinovanog adaptivnog algoritma na hibridne aplikacije i aplikacije kod kojih postoji zavisnost između procesa koji su dodijeljeni različitim jezgrima. LITERATURA [1] M. V. Wilkes, Automatic Digital Computers, John Wiley & Sons, 1957.

[2] A. S. Tanenbaum and M. van Steen, Distributed Systems: Principles and Paradigms, 2nd Edition, Pearson Education. Inc., 2007. [3] G. S. Almasi and A. Gottlieb, Highly parallel computing, Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc. , 1989. [4] B. Barney, Introduction to Parallel Computing, Lawrence Livermore National, 2012. [5] H. D. Karatza and R. C. Hilzer, "Parallel Job Scheduling in Homogeneous Distributed Systems," Simulation, vol. 79, 2003. [6] J. Dinan, S. Olivier, G. Sabin, J. Prins, P. Sadayappan and C.-W. Tseng, "Dynamic Load Balancing of Unbalanced Computations Using Message Passing," in Parallel and Distributed Processing Symposium, 2007, IPDPS 2007, IEEE International, Long Beach, CA, USA, 2007. [7] T. Rauber and G. Rünger, Parallel Programming: for Multicore and Cluster Systems, Springer, 2010. [8] D. Thiébaut, Parallel Programming in C for the Transputer, 1995. [9] M. J. Atallah and M. Blanton, Algorithms and Theory of Computation Handbook, Second Edition, Chapman & Hall/CRC, 2009. [10] C. Amit, G. Singh, S. S. Waraich, B. Sidhu and G. Kumar, "Qualitative Parametric Comparison of Load Balancing Algorithms in Distributed Computing Environment," in World Academy of Science, Engineering and Technology, 2008. [11] L. Filipovic and B. Krstajic, "Combined load balancing algorithm in distributed computing environment," Information Technology and Control, vol. 45, no. 3, pp. 261-266, 2016. [12] M. Radonjic, Prilog analizi performansi CQ komutatora paketa sa stanovišta veličine i algoritama upravljanja redovima čekanja. doktorska disertacija, Podgorica: Elektrotehnički fakultet, Univerzitet Crne Gore , 2011. [13] "HP SEE projekat," [Online]. Available: https://cordis.europa.eu/project/rcn/95208/factsheet/en. [14] G. Tel, Introduction to Distributed Algorithms, 2nd Edition, Cambridge University Press, 2000. [15] H. Attiya and J. Welch, Distributed Computing: Fundamentals, Simulations, and Advanced Topics, 2 edition, Wiley-Interscience, 2004. [16] G. K. Vijay, Elements of Distributed Computing, Wiley-IEEE Press, 2002. [17] R. Buyya, High Performance Cluster Computing: Architectures and Systems, Upper Saddle River, New Jersey: Prentice Hall PTR, 1999. [18] J. Levesque and . Wagenbreth, High Performance Computing: Programming and Applications, Chapman & Hall/CRC Computational Science, 2010. [19] L. Wang, R. Ranjan, J. Chen and B. Benatallah, Cloud Computing: Methodology, Systems, and Applications, CRC Press, 2011. [20] I. Foster and C. Kesselman, The Grid 2: Blueprint for a new computing infrastructure, Elsevier, 2003. [21] D. Padua, Encyclopedia of Parallel Computing, Springer US, 2011. [22] M. J. Flynn, "Very High-Speed Computing Systems," Proc. IEEE, vol. 54, no. 12, pp. 1901-1909, 1966. [23] M. J. Flynn, "Some Computer Organizations and Their Effectiveness," IEEE Transactions on Computers, Vols. C-21, no. 9, pp. 948 - 960, 1972. [24] M. Flynn and R. Rudd, "Parallel Architectures," ACM Comput Surv, vol. 28, no. 1, p. 67–70, 1996. [25] J. v. Neumann, Probabilistic logic and the synthesis of reliable organisms from unreliable components, Princeton, N. J.: Princeton Univ. Press, 1954. [26] J. Hennessy and D. Patterson, Computer Architecture: A Quantitative Approach, Fifth Edition, Morgan Kaufmann, 2012. [27] F. Gebali, Algorithms and Parallel Computing, Wiley, 2011. [28] E. Johnson, "Completing an MIMD multiprocessor taxonomy," Computer Architecture News, vol. 16, no. 3, pp. 44-47, 1988. [29] J. JaJa , Introduction to Parallel Algorithms, Addison-Wesley Professional, 1992. [30] S. V. Adve, V. Adve, A. Gul, M. I. F. María and e. al., Parallel Computing Research at Illinois: The UPCRC Agenda, Parallel@Illinois, University of Illinois at Urbana-Champaign, 2008. [31] G. E. Moore, "Cramming more components onto integrated circuits," Electronics Magazine, vol. 38, no. 8, 1965. [32] S. Benkner and V. Sipkova, "Exploiting Distributed-Memory and Shared-Memory Parallelism on Clusters of SMPs with Data Parallel Programs," International Journal of Parallel Programming, vol. 31, no. 1, pp. 3-19 , 2003. [33] M. Tomasevic and V. Milutinovic, "Hardware approaches to cache coherence in shared-memory multiprocessors," IEEE Micro, vol. 14, no. 6, pp. 61-66, 1994. [34] A. Grama, A. Gupta, G. Karypis and V. Kumar, Introduction to Parallel Computing, 2nd ed., Reading, MA, USA: Addison Wesley, 2003. [35] B. Chapman, G. Jost and R. v. d. Pas, Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation), The MIT Press; Scientific and Engin edition, 2007. [36] D. Lewine, POSIX Programmers Guide, O'Reilly Media, 1991. [37] "MPI Forum,

MPI: A Message-Passing Interface Standard, University of Tennessee, Knoxville, Tennessee, 1994.". [38] G. M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," Proceedings of the April 18-20, 1967, Spring Joint Computer Conference, pp. 483--485, 1967. [39] J. L. Gustafson, "Reevaluating Amdahl's Law," Communications of the ACM, vol. 31, no. 5, pp. 532-533 , 1988. [40] D. Culler, J. P. Singh and A. Gupta, Parallel Computer Architecture: A Hardware/Software Approach, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998. [41] B. Prince, "Memory in the fast lane," IEEE Spectrum, vol. 31, no. 2, pp. 38-41, 1994. [42] A. Grama, A. Gupta and V. Kumar, "Isoefficiency Function: A Scalability Metric for Parallel Algorithms and Architectures," IEEE Parallel and Distributed Technology, Special Issue on Parallel and Distributed Systems: From Theory to Practice, vol. 1, no. 3, pp. 12-21, 1993. [43] Y. Deng, Applied Parallel computing, World Scientific Publishing Company, 2012. [44] J. Fredin, "Performance of Gaussian 09 on SharedMemory & Cluster," SGI ISV & Developer Technology Brief. [45] X.-H. Sun and Y. Chen, "Reevaluating Amdahl's law in the multicore era," Journal of Parallel and Distributed Computing, vol. 70, no. 2, pp. 183-188 , 2010. [46] A. H. Karp and H. Flatt, "Measuring Parallel Processor Performance," Communications of the ACM, vol. 33, no. 5, pp. 539-543 , 1990. [47] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, Introduction to Algorithms, 3rd Edition, The MIT Press, 2009. [48] J. Radatz, The IEEE Standard Dictionary of Electrical and Electronics Terms, IEEE, 1997. [49] G. Chen, G. Sun, Y. Zhang and Z. Mo, "Study on Parallel Computing," Journal of Computer Science and Technology, vol. 21, no. 5, p. 665–673, 2006. [50] J. J. Park, I. Stojmenovic, H. Y. Jeong and G. Yi, Computer Science and its Applications: Ubiquitous Information Technologies, Springer Publishing Company, 2014. [51] D. M. Abdelkader and F. Omara, "Dynamic task scheduling algorithm with load balancing for heterogeneous computing system," Egyptian Informatics Journal, vol. 13, pp. 135-145, 2012. [52] H. Kaur , G. Kaur and A. Chhabra, "Adaptive Job Scheduling in Heterogeneous Multicluster Systems," International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE), vol. 2, no. 3, pp. 22-25, 2013. [53] S. Patil and A. Gopal, "Cluster performance evaluation using load balancing algorithm," in Information Communication and Embedded Systems (ICICES), 2013 International Conference on, 2013. [54] D. G. Feitelson and L. Rudolph, Job Scheduling Strategies for Parallel Processing, Springer, 1995. [55] M. Katyal and A. Mishra, "A Comparative Study of Load Balancing Algorithms in Cloud Computing," International Journal of Distributed and Cloud Computing, vol. 1, 2013. [56] N. Shahapure and P. Jayarekha, "Time sliced and priority based load balancer," Advance Computing Conference (IACC), 2015 IEEE International , pp. 154-159, 2015. [57] A. Chhabra, G. Singh, S. S. Waraich, B. Sidhu and G. Kumar, "Qualitative Parametric Comparison of Load Balancing Algorithms in Parallel and Distributed Computing Environment," Proceedings of World Academy of Science, Engineering and Technology (PWASET), vol. 16, pp. 39-42, 2006. [58] C. Banino-Rokkones, "Domain Decomposition vs. Master-Slave in Apparently Homogeneous Systems," in Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, Long Beach, CA, USA, 2007. [59] L. Filipović and B. Krstajić, "Modified master-slave algorithm for load balancing in parallel applications," ETF Journal of Electrical Engineering, vol. 20, pp. 74-83, 2014. [60] M. Livny and M. Melman, "Load balancing in homogeneous broadcast distributed systems," ACM SIGMETRICS Performance Evaluation Review, vol. 11, no. 1, pp. 47-55, 1981. [61] P. Krueger and M. Livny, "The Diverse Objectives of Distributed Scheduling Policies," Proceedings - International Conference on Distributed Computing Systems., pp. 242-249, 1987. [62] X. Chenzhong and L. C. Francis, Load Balancing in Parallel Computers: Theory and Practice, Norwell, MA, USA: Kluwer Academic Publishers, 1997. [63] T. Casavant and J. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," IEEE Transactions on Software Engineering, vol. 14, no. 2, pp. 141 - 154, 1988. [64] M. Zaki, W. Li and S. Parthasarathy, "Customized dynamic load balancing for a network of workstations," Proceedings of 5th IEEE International Symposium on High Performance Distributed Computing, p. 282–291, 1996. [65] B. B. ZhouR, P. Brent, D. Walsh and S. K., "Job scheduling strategies for

networks of workstations," Job Scheduling Strategies for Parallel Processing, JSSPP 1998, vol. 1459, pp. 143-157, 1998. [66] V. M. Lo, "Heuristic Algorithms for Task Assignment in Distributed Systems," IEEE Transactions on Computers, vol. 37, no. 11, pp. 1384- 1397, 1988. [67] V. Sarkar and J. Hennessy, "Compile-time partitioning and scheduling of parallel programs," SIGPLAN '86 Proceedings of the 1986 SIGPLAN symposium on Compiler construction, vol. 21, no. 7, pp. 17-26, 1986. [68] B. Shirazi, M. Wang and G. Pathak, "Analysis and evaluation of heuristic methods for static task scheduling," Journal of Parallel and Distributed Computing, vol. 10, no. 3, pp. 222-232, 1990. [69] H. S. Stone, "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," IEEE Transactions on Software Engineering , Vols. SE-3, no. 1, pp. 85 - 93, 1977. [70] E. G. Coffman , Computer and Job-shop Scheduling Theory, John Wiley & Sons Inc, 1976. [71] B. Lee, A. Hurson and T. Feng, "A vertically layered allocation scheme for data flow systems," Journal of Parallel and Distributed Computing, vol. 11, no. 3, pp. 175-187, 1991. [72] M. Wang, B. Lee, B. Shirazi and A. Hurson, "Accurate communication cost estimation in static task scheduling," Proceedings of the Twenty- Fourth Annual Hawaii International Conference on System Sciences, pp. 10-16, 1991. [73] A. Olteanu and A. Marin, "Generation and Evaluation of Scheduling DAGs: How to provide similar evaluation conditions," Computer Science Master Research , vol. 1, no. 1, 2011. [74] D. L. Eager, E. D. Lazowska and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," IEEE Trans. Software Eng., Vol. 12, No. 5, pp. 662-657, 1986. [75] F. C. H. Lin and R. M. Keller, "The Gradient Model Load Balancing Method," IEEE Transactions on Software Engineering, Vols. SE-13, no. 1, 1987. [76] N. G. Shivaratri, P. Krueger and M. Singhal, "Load Distributing for Locally Distributed Systems," Computer, vol. 25, no. 12, pp. 33-44, 1992. [77] Y.-T. Wang and M. Robert, "Load Sharing in Distributed Systems," IEEE Transactions on Computers, Vols. C-34, no. 3, pp. 204-217, 1985. [78] M. Hamdi and C. K. Lee, "Dynamic load-balancing of image processing applications on clusters of workstations," Parallel Computing, vol. 22, no. 11, pp. 1477-1492, 1997. [79] H.-U. Heiss and M. Schmitz, "Decentralized Dynamic Load Balancing: The Particles Approach," Information Sciences—Informatics and Computer Science, Intelligent Systems, Applications: An International Journal, vol. 84, no. 1-2, pp. 115-128, 1995. [80] R. Mirchandaney, D. Towsley and J. Stankovic, "Adaptive load sharing in heterogeneous systems," Proc. International Conference on Distributed Computing Systems, pp. 298-306, 1989. [81] K. G. Shin and C. J. Hou, "Analytic models of adaptive load sharing schemes in distributed realtime systems," IEEE Transactions on Parallel and Distributed Systems, vol. 4, no. 9, pp. 740-761, 1993. [82] M. H. Willebeek-LeMair and A. P. Reeves, "Strategies for dynamic load balancing on highly parallel computers," IEEE Transactions on Parallel and Distributed Systems, vol. 4, no. 9, pp. 979-993, 1993. [83] D. Vidyarthi, B. Sarker, A. Tripathi and L. Yang, Scheduling in Distributed Computing Systems : Analysis, Design & Models, Springer US, 2009. [84] F. Doughs and J. Ousterhout, "Transparent Process Migration: Design Alternatives and the Sprite Implementation," Software—Practice and Experience, vol. 21, no. 8, pp. 757-785, 1991. [85] P. Krueger and R. Chawla, "The Stealth Distributed Scheduler," Proceedings. 11th International Conference on Distributed Computing Systems, pp. 336-343., 1991. [86] N. G. Shivaratri and P. Krueger, "Two Adaptive Location Policies for Global Scheduling," Proceedings.,10th International Conference on Distributed Computing Systems, pp. 502-509, 1990. [87] D. Eager, E. Lazowska and J. Zahorjan, "A Comparison of Receiver- Initiated and Sender-Initiated Adaptive Load Sharing," Performance Evaluation, vol. 6, no. 1, pp. 53-68, 1986. [88] J. A. Stankovic, "Simulations of three adaptive, decentralized controlled, job scheduling algorithms," Computer Networks, vol. 8, no. 3, pp. 199-217, 1984. [89] M. Bhandarkar, L. Kalé, E. de Sturler and J. Hoeflinger, "Adaptive Load Balancing for MPI Programs," Computational Science - ICCS 2001; Lecture Notes in Computer Science, vol 2074. , pp. 108-117, 2001. [90] A. M. M. Mukaddes and R. Shioya, "Parallel performance of domain decomposition method on distributed computing environment," IACSIT International Journal of Engineering and Technology , vol. 2, no. 1, pp. 28-34, 2010. [91] R. Shioya, M. Ogino, H. Kanayama and D. Tagami, "Large Scale Finite Element Analysis

with a Balancing Domain Decomposition Method," Key Engineering Materials, Vols. 243-244, pp. 21-26, 2003. [92] B. Smith, P. Bjorstad and W. Gropp, Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations, Cambridge University Press, 2004. [93] A. E. Chakib and A. Nachaoui, "A domain decomposition convergence for elasticity equations," Mathematics and Computers in Simulation, pp. 154-167, 2008. [94] M. P. Raju and S. Khaitan, "Domain decomposition based high performance parallel computing," International Journal of Computer Science Issuess, p. 27–32, 2009. [95] Y. Boubendir, X. Antoine and C. Geuzaine, "A quasi-optimal non- overlapping domain decompostion algorithm for the helmholtz equation," Journale of Computational Physics, pp. 262-280, 2012. [96] J. Sikora and T. Grzywacz, "Domain decomposition method for diffuse optical tomography problems," Engineering Analysis with Boundary Elements, p. 1005–1013, 2012. [97] V. Sarkar, Partitioning and Scheduling Parallel Programs for Multiprocessors, MIT Press, 1989. [98] B. Shirazi, M. Wang and G. Pathak, "Analysis and evaluation of heuristic methods for static task scheduling," Journal of Parallel and Distributed Computing, vol. 10, no. 3, pp. 222-232, 1990. [99] B. A. Shirazi, A. R. Hurson and K. M. Kavi, Scheduling and Load Balancing in Parallel and Distributed Systems, Wiley-IEEE Computer Society Press, 1995. [100] W. D. Gropp, "Parallel Computing and Domain Decomposition," in Fifth Conference on Domain Decomposition Methods for Partial Differential Equations, 1990. [101] S. Sartaj, "Scheduling Master-Slave Multiprocessor Systems," IEEE Transactions on Computers, vol. 45, no. 10, pp. 1195-1199, 1996. [102] K. Everaars and B. Koren, "Using coordination to parallelize sparse- grid methods for 3-d cfd problems," Parallel Computing, vol. 24, no. 7, p. 1081–1106, 1998. [103] A. Zăvoianu, E. Lughofer, W. Koppelstätter, G. Weidenholzer, W. Amrhein and E. Klement, "On the Performance of Master-Slave Parallelization Methods for Multi-Objective Evolutionary Algorithms," in International Conference on Artificial Intelligence and Soft Computing, 2013. [104] D. Hadka, K. Madduri and P. Reed, "Scalability Analysis of the Asynchronous, Master-Slave Borg Multiobjective," in IEEE 27th International Symposium on Parallel & Distributed Processing Workshops and PhD Forum, 2013. [105] G. Shao, F. Berman and R. Wolski, "Master/slave computing on the Grid," in Proceedings 9th Heterogeneous Computing Workshop (HCW 2000) (Cat. No.PR00556), Cancun, Mexico, 2002. [106] F. Berman, "High-performance schedulers," in Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1998. [107] S. Sahni and G. Vairaktarakis, "The master-slave paradigm in parallel computer and industrial settings," Journal of Global Optimization, vol. 9, no. 3-4, pp. 357-377, December 1996. [108] A. Clematis and A. Corana, "Performance analysis of task-based algorithms on heterogeneous systems with message passing," in Recent Advances in Parallel Virtual Machine and Message Passing Interface. EuroPVM/MPI 1998. Lecture Notes in Computer Science, vol 1497., Berlin, Heidelberg, Springer, 1998. [109] D. Gelernter, M. R. Jourdenais and K. D., "Piranha scheduling: Strategies and their implementation," International Journal of Parallel Programming, vol. 23, no. 1, p. 5–33, 1995. [110] A. S. Wagner, H. V. Sreekantaswamy and S. T. Chanson, "Performance models for the processor farm paradigm," IEEE Transactions on Parallel and Distributed Systems, vol. 8, no. 5, p. 475–489, 1997. [111] S. Mostaghim, J. Branke, A. Lewis and H. Schmeck, "Parallel multi- objective optimization using Master-Slave model on heterogeneous resources," in IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), Hong Kong, 2008. [112] M. Radonjic and I. Radusinovic, "Impact of scheduling algorithms on performance of crosspoint-queued switch," Annals of Telecommunications, vol. 66, no. 5-6, pp. 363-376, 2011. [113] I. Radusinovic, M. Radonjic, A. Simurina, I. Maljevic and Z. Veljovic, "A new analytical model for the CQ switch throughput calculation under the bursty traffic," International Journal of Electronics and Communications (AEU), vol. 66, no. 12, pp. 1038-1041, 2012. [114] M. Radonjic and I. Radusinovic, "CQ Switch Performance Analysis from the Point of Buffer Size and Scheduling Algorithms," in Proc. of 20th Telecommunication Forum TELFOR 2012, 2012. [115] A. Mishev and E. Atanassov, Infrastructure overview and assessment, HP-SEE project, Deliverable D 5.2. [116] A. Balaz, O. Prnjat, D. Vudragovic, V. Slavnic, I. Liabotis, E. Atanassov,

B. Jakimovski and M. Savic, "Development of Grid E-Infrastructure in South-Eastern Europe," Journal of Grid Computing, vol. 9, no. 2, pp. 135-154, June 2011. [117] L. Filipovic, "Optimizacija simulatora CQ komutatora paketa metodom paralelnog programiranja," Informacione tehnologije IT'12 , pp. 20-23, 2012. 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

sources:

1 326 words / 2% - Internet from 18-Apr-2019 12:00AM
www.it.ac.me

2 192 words / 1% - Internet from 29-Jan-2016 12:00AM
elibrary.matf.bg.ac.rs

3 184 words / 1% - Internet from 06-Sep-2016 12:00AM
www.scribd.com

4 94 words / < 1% match - Internet from 29-May-2015 12:00AM
default.rs

5 65 words / < 1% match - Internet from 28-May-2015 12:00AM
www.tf.uns.ac.rs

6 65 words / < 1% match - Crossref
Milutin Radonjic, Igor Radusinovic, Dusan Banovic, Ivo Maljevic. "Packet delay variation analysis of the CQ switch under uniform traffic", 2011 19thTelecommunications Forum (TELFOR) Proceedings of Papers, 2011

7 41 words / < 1% match - Internet from 13-Jul-2017 12:00AM
unimediteran.net

8 35 words / < 1% match - Internet from 18-Mar-2019 12:00AM
www.multimedia.ac.me

9 31 words / < 1% match - Internet from 03-Nov-2017 12:00AM
www.ucg.ac.me

10 17 words / < 1% match - Internet from 29-Dec-2014 12:00AM
www.cs.ru

11 14 words / < 1% match - Internet from 02-Nov-2017 12:00AM
fedorakg.kg.ac.rs

12 14 words / < 1% match - Internet from 14-Nov-2009 12:00AM
www.se.eecs.uni-kassel.de

13 11 words / < 1% match - Internet from 12-Sep-2017 12:00AM
documents.mx

14 11 words / < 1% match - Internet from 26-Nov-2016 12:00AM
www.kalbos.ktu.lt

15 11 words / < 1% match - Internet from 16-Apr-2018 12:00AM
kluedo.ub.uni-kl.de

16 11 words / < 1% match - Internet from 08-Oct-2014 12:00AM
waset.org

17 11 words / < 1% match - Internet
www.goranrakic.com

18 10 words / < 1% match - Internet from 07-Feb-2017 12:00AM
www.ds.ewi.tudelft.nl

19 10 words / < 1% match - Internet from 29-Aug-2017 12:00AM
compalg.inf.elte.hu