UNIVERSITY OF NOVI SAD
FACULTY OF SCIENCES
DEPARTMENT OF
MATHEMATICS AND INFORMATICS

Žarko Bodroški

Razvoj serijskog i paralelnog algoritma za računanje elektronske strukture materijala metodom sklapanja naelektrisanja

— Doktorska disertacija —

Development of Serial and Parallel Algorithms for Computing the Electronic Structure of Materials Using the Charge Patching Method

— PhD thesis —

Mentori/Advisors:
Dr Srđan Škrbić
Dr Nenad Vukmirović

Novi Sad, 2020

Aniti i Matiji

Sadržaj

Zahvalnica

Najveću zahvalnost za realizovanje ovog rada pre svega dugujem svojim mentorima dr Srđanu Škrbiću i dr Nenadu Vukmiroviću. Kako se radi o komentorstvu na radu u kome se prepliće više naučnih oblasti, smatram da su oni na izuzetan način uspeli da premoste sve izazove koje ovako složen proces nosi. Njihova nesebična pomoć, profesionalna, moralna podrška, kao i izuzetna sposobnost da se bez obzira na težinu problema pronađe adekvatno rešenje, su bile prisutne u svim fazama rada.

Zahvaljujem se članovima komisije, dr Milošu Rackoviću, dr Milošu Radovanoviću i dr Antunu Balažu na korisnim savetima i vrednim komentarima u završnoj fazi pisanja rada, kao i dr Vladimir Kurbaliji, dr Draganu Mašuloviću i dr Dušanki Perišić na nesebičnoj pomoći i inspiraciji za rešavanje problema u toku izrade teze.

Veliku zahvalnost dugujem Laboratoriji za primenu računara u nauci (SCL), u okviru Centra za izučavanje kompleksnih sistema Instituta za fiziku u Beogradu, gde bih posebno izdvojio Antuna Balaža koji je zaslužan za moje prve korake u svetu računarstva visokih performansi i koji me je inspirisao da mi ta oblast postane glavna preokupacija u daljem radu.

Takođe, zahvalnost dugujem timu oformljenom oko Axiom klastera na PMF-u, gde bih posebno izdvojio svoje kolege Baneta Ivoševa i Vladimira Jokića koji su dali nesebičnu podršku i inspiraciju za rešavanje mnogih problema koji su se javili na putu da stvorimo infrastrukturu za naučno računanje.

Na kraju želim da istaknem da je moja najveća inspiracija, kako za ovaj rad, tako i za sve ono što sam postigao u svom životu, moja porodica. Iskrena ljubav, srdačnost, požrtvovanje i upornost mojih roditelja je nešto uz šta sam odrastao, što je deo svakog mog uspeha i na šta sam im neizmerno zahvalan. Kako je početak rada na ovoj tezi bio i početak mog roditeljstva, tako je usklađivanje obaveza i postavljanje odgovarajućih prioriteta za mene postao veliki izazov. Smatram da imam veliku sreću što pored sebe imam osobu kao što je moja supruga, koja na sjajan način, uvek iznova, pronalazi načine kako da izazov pretvori u uspeh.

Poglavlje 1

Uvod

1.1 Računarstvo visokih performansi

Računarske simulacije već duži vremenski period u mnogim naučnim oblastima predstavljaju ključni metod za otkrivanje novih i za razumevanje postojećih fenomena. Domet upotrebe računarskih simulacija zavisi od primene adekvatnih metoda, razvoja odgovarajućih algoritama, njihove efikasnosti kao i od napretka tehnologije, odnosno hardverskog razvoja računarskih platformi na kojima se simulacije izvršavaju.

Ukoliko bismo paralelizam posmatrali kroz hijerarhijske nivoe, možemo prepoznati tri nivoa paralelizacije. Prvi nivo paralelizma je dostupan u gotovo svim modernim računarima zahvaljujući višejezgarnim procesorima (multi-core CPUs). Proširenje ovakvih arhitektura se uglavnom postiže dodavanjem procesora, a zajedničko im je da paralelni procesi dele pristup istoj operativnoj memoriji. Smatra se da je sa aplikativnog aspekta ovo najjednostavniji način paralelizacije. Jedan od najrasprostranjenijih programskih interfejsa koji se koriste za ovakve računarske platforme je OpenMP.

Drugi nivo paralelizma podrazumeva upotrebu specijalizovanih hardverskih akceleratora, kao što su grafički procesori (GPU), Intel Xeon Phi procesori i drugi. S obzirom da ovi akceleratori imaju arhitekturu koja se značajno razlikuje od CPU arhitekture i da poseduju svoju operativnu memoriju, uz pomoć njih je moguće određene grupe problema rešavati na efikasniji način. Programski interfejs koji je najčešće u upotrebi kada su u pitanju grafički akceleratori je CUDA. Takođe je moguće kombinovati prethodna dva nivoa paralelizma. Uz odgovarajuću komunikaciju među procesima koji se paralelno izvršavaju na različitim vrstama akceleratora (najčešće CPU-GPU) i odgovarajuću raspodelu posla, dobijamo hibridni algoritam, a arhitekture koje to omogućavaju se nazivaju heterogenim arhitekturama.

Čak i pored značajnog napretka u razvoju procesora, za mnoge domene primene računarske platforme koje dele računarsku memoriju nisu u mogućnosti da zadovolje potrebe. Najčešće se radi o nedostatku procesorske snage ili nedovoljnoj količini memorije. Da bi se to prevazišlo moguće je proširiti arhitekturu tako što se u nju uključi više računarskih instanci povezanih računarskom mrežom. Tako dolazimo do trećeg nivoa paralelizacije odnosno do računarskog klastera. Kako ovakav tip arhitekture ima distribuiranu memoriju, potrebno je primeniti programski interfejs koji je specijalizovan za ovakav oblik komunikacije među paralelnim procesima. Standard za ovu vrstu komunikacije je MPI (Message Passing Interface), dok su najrasprostranjenije implementacije ovog

standarda OpenMPI [1] i MPICH [2].

Jedna od podela algoritama za računarske simulacije je na serijske (ili sekvencijalne) i paralelne (u slučaju primene na računarskim klasterima – distribuirane). Kako su numeričke simulacije često resursno vrlo zahtevne, ograničenja do kojih se dolazi pri upotrebi serijskih algoritama na jednoprocesorskim sistemima se prevazilaze implementacijom odgovarajućih paralelnih algoritama. Ovakvi algoritmi su namenjeni za primenu na računarima visokih performansi sa distribuiranom arhitekturom. Za razvoj paralelnih algoritama je potrebna primena paralelne metodologije razvoja i odgovarajućih alata i programa. U slučaju distribuiranog računanja, za razliku od serijskog, kod razvoja ovakvih algoritama se uvode novi domeni koji se odnose na distribuciju podataka, komunikaciju među procesima, prilagođavanje specifičnostima arhitekture na kojoj se algoritam izvršava i drugi.

1.2 Primena računarstva visokih performansi u nauci o materijalima

Napredak u razvoju računarstva visokih performansi nesumnjivo utiče na mnoge oblasti istraživanja pa tako i na one koje se bave izučavanjem osobina materijala. U današnje vreme se, pored teorijske analize i eksperimenata, naučno računanje može smatrati trećom naučnom metodom [3]. Potvrda ove tvrdnje je ubrzani napredak u razvoju simulacija za opis materijala. Ove softverske simulacije predstavljaju sveobuhvatni sistem koji pokriva sve od istraživanja fundamentalnih fizičkih efekata, procesa, molekula, do dizajna materijala, inženjeringa sistema, procesnih i proizvodnih aktivnosti i konačno do razvoja tehnologija, kao rezultata konvergencije mnogih naučnih disciplina.

Nauka o materijalima cilja na razvoj materijala koji obuhvata sisteme veličine od jednog atoma pa sve do onih koji se mere makroskopskom skalom. Za razumevanje osobina materijala, od značaja su procesi koji se dešavaju na širokom opsegu vremenskih skala – počev od vremena reda femtosekunde, pa do dužina trajanja geoloških perioda. Kako je istraživanje osobina materijala usko povezano sa mnogim naučnim oblastima (kao što su medicina, biologija, hemija, geologija i druge), bez uske saradnje naučnih zajednica uspešan razvoj algoritama i alata za računarstvo visokih performansi je teško ostvariv.

Rezultati dobijeni simulacijama za opis materijala nalaze široku primenu. Neki od proizvoda čiji je razvoj direktno povezan sa napretkom računara visokih performansi i simulacija za opis materijala su solarne ćelije, superprovodnici, različiti tipovi senzora, memorijski moduli, procesori, kamere, pametni telefoni, baterije, slušalice, automobili, sateliti i drugi.

1.3 Teorija funkcionala gustine (DFT)

U oblastima koje se bave proučavanjem strukture materijala je poznato da se osobine materijala na nanometarskim dužinama mogu u potpunosti opisati pomoću Šredingerove jednačine, objavljene 1926. godine. Iako ova jednačina vrlo precizno opisuje kvantno stanje fizičkog sistema, njena praktična upotreba u originalnoj formi je, zbog svoje složenosti, vrlo teško izvodljiva. Za uspešan razvoj metoda za aproksimaciju ove jednačine, koji omogućavaju rešavanje netrivijalnih sistema, je bilo potrebno više od 5 decenija. Do proboja u razvoju algoritama i metodologija je došlo 1960. godine, kada su uvedena dva nova pristupa: teorija funkcionala gustine (Density Functional

Theory – DFT) i pseudopotencijali [4, 5, 6]. Uz pomoć DFT se inicijalni problem transformiše u jednačinu koja sadrži funkcije koje umesto $3N$ prostornih promenjivih iz Šredingerove jednačine za opis N-čestičnog sistema, sadrži samo tri promenljive koje odgovaraju prostornim koordinatama. Na taj način se $\mathbb{R}^{3N}$ funkcije svode na $\mathbb{R}^3$. Sledeće značajno unapređenje se dobija upotrebom pseudopotencijala. Pseudopotencijal omogućava da se u obzir uzimaju samo valentni elektroni, čime se značajno umanjuje kompleksnost računanja [7].

U početnim fazama razvoja, DFT je nalazio upotrebu u računanju jednostavnih sistema od svega nekoliko elektrona. Pri tome su korišćeni periodični granični uslovi. Nakon pionirskog rada, koji u cilju unapređenja DFT-a (najviše iz aspekta optimizacije), kombinuje molekularnu dinamiku sa DFT [8], postiže se značajno povećanje veličine sistema koje je bilo moguće rešavati. Ovome je svakako doprineo i razvoj metode za rešavanje svojstvenog problema uz upotrebu konjugovanih gradijenata [6], zatim uvođenje načina za efikasnu upotrebu skupa ravnih talasa [9], kao i metod baziran na matricama gustine [10]. Trenutno je moguće rutinsko rešavanje sistema od nekoliko stotina elektrona. Kako se otvaraju mogućnosti za rešavanje sve većih sistema, tako se povećavaju i mogućnosti proučavanja različitih klasa materijala i fenomena. Kako se uz pomoć DFT-a mogu dobiti pouzdani podaci koji mogu da zamene one dobijene sprovođenjem eksperimenata, mogućnosti primene DFT-a se značajno šire. DFT trenutno nalazi primenu u fizici, hemiji, nauci o materijalima i mnogim drugim. Ipak, ako se broj atoma približi 1000, složenost računanja DFT-a se jako povećava. Za sistem veličine N se može smatra da je složenost N^3. Da bi se ovo prevazišlo, neophodno je uvesti metode koji imaju linearno skaliranje [11, 12].

Pošto je poslednjih dvadeset godina razvoj numeričkih metoda pratio i ubrzani razvoj računarskih arhitektura, implementirani su mnogi softverski paketi koji su bazirani na DFT. Neki od njih su ABINIT [13], CASTEP [14], FHI-aims [15], Molpro [16], NWChem [17], Qbox [18], QChem [19], Quantum ESPRESSO [20], SIESTA [21], VASP [22, 9], kao i mnogi drugi.

DFT je trenutno najzastupljeniji metod za računanje elektronskih osobina materijala i nanostruktura sa relativno velikim brojem (stotine do oko hiljadu) atoma. Zbog numeričke zahtevnosti postojećih pristupa za rešavanje DFT jednačina postoji velika potreba za razvojem i implementacijom novih efikasnih metoda i algoritama. Trenutno postoji veliki broj računarskih programa i biblioteka koje implementiraju DFT jednačine u bazisu skupa ravnih talasa [23, 24, 25, 26], u bazisu Gausijana [17, 27], uz pomoć lokalizovanih numeričkih orbitala [28], zatim one koje koriste prostornu reprezentaciju [29, 30], kao i onih koje koriste druge pristupe [31]. Međutim, trenutni programi i biblioteke koje implementiraju DFT se ne mogu koristiti za velike sisteme (više od nekoliko stotina atoma), tako da i dalje postoji veliki interes za proboje u ovoj oblasti. Aktuelno je nekoliko različitih pravaca istraživanja koji se bave rešavanjem elektronske strukture materijala. Jedan od tih pravaca je posvećen razvoju metode koja daje rezultate visoke preciznosti za slučajeve gde standardne aproksimacije u DFT-u nisu uspešne. U okviru ovog pravca se ističu mnogočestična teorija u okviru GW aproksimacije [32] (gde G predstavlja Green-ovu funkciju, dok je W ekranirana Kulonova interakcija), razvoj novih funkcionala gustine [33, 34], kao i razvoj metoda baziranih na dinamičkoj teoriji srednjeg polja [35]. Drugi važan pravac uključuje razvoj visokoefikasnih numeričkih šema u okviru DFT koje omogućavaju računanje velikih sistema i efikasno korišćenje računara visokih performansi [36, 37, 38, 39, 40, 41, 42]. Treći jednako važan pravac je baziran na korišćenju dobro zasnovanih aproksimacija koje značajno ubrzavaju DFT izračunavanje i otvaraju put ka analizi velikih sistema. Metodi u okviru ovog pravca uključuju metod funkcionala gustine sa

reprezentacijom jakih veza [43, 44], molekularno-orbitalnu metodu fragmenata [45], metodu polu-empirijskih pseudopotencijala [46, 47], metodu sklapanja naelektrisanja (Charge Patching Method – CPM) [48, 49, 50], i druge.

1.4 Metod za sklapanje naelektrisanja (CPM)

Metodi koji se oslanjaju na direktno rešavanje DFT-a se za sisteme do nekoliko stotina elektrona smatraju efikasnim i veoma preciznim, međutim već kod sistema od preko 1000 atoma ovakav pristup dovodi do netrivijalnih procesa za koje je potrebno više stotina procesorskih jezgara i dugo vreme izvršavanja. Ukoliko se sistemi povećaju na nekoliko hiljada atoma, čak i za najveće računare visokih performansi ovaj pristup postaje neprimenjiv, odnosno nepraktičan iz aspekta računarskih resursa i vremena izvršavanja. Jedan od načina da se ovi problemi prevaziđu je primena Metode za sklapanje naelektrisanja (CPM), koja pored dobre efikasnosti izvršavanja daje rezultate čija je preciznost uporediva sa direktnim ab initio metodama [51].

Centralna ideja CPM je direktno konstruisanje jednočestičnog Kohn-Sham Hamiltonijana [5], bez potrebe za samosaglasnim DFT računanjem. CPM je bazirana na pretpostavci da je elektronska gustina naelektrisanja velikih sistema zbir doprinosa pojedinačnih atoma – takozvanih motiva gustine naelektrisanja. Ovi doprinosi se ekstrahuju na osnovu računanja nad malim prototip sistemima, dok se odgovarajuća gustina naelektrisanja za velike sisteme dobija dodavanjem doprinosa pojedinačnih atoma. S druge strane, kod metode polu-empirijskih pseudopotencijala [46, 47], se umesto gustine naelektrisanja koristi jednočestični potencijal i potencijal velikog sistema se dobija sabiranjem doprinosa potencijala pojedinačnih atoma.

Postojeće implementacije CPM-a koriste reprezentaciju talasnih funkcija uz pomoć bazisa skupa ravnih talasa [50]. Iako implementacije bazirane na ravnim talasima imaju brojne prednosti, kao što su relativno jednostavna implementacija relevantnih jednačina i mogućnost sistematičnog ograničavanja dimenzije bazisnog skupa na osnovu kinetičke energije ravnog talasa, takođe imaju i značajne nedostatke zbog potrebe za velikim brojem ravnih talasa neophodnih za precizan opis talasnih funkcija u neposrednoj okolini atoma. S druge strane, prednost korišćenja bazisnog skupa koji sadrži lokalizovane gausijanske funkcije, razvijene od strane zajednice kvantnih hemičara [17, 27], je ta što se talasne funkcije mogu predstaviti malim brojem Gausijana. Iako ne postoji jednostavan sistematičan način za smanjenje ili proširenje bazisnog skupa, moguće je iskoristiti široko rasprostranjene bazisne skupove razvijene tokom perioda od skoro pola veka. U numeričkom primeru u poglavlju 5 se koristi DGauss A1 DFT Coulomb Fitting bazisni skup [52, 53, 54].

Nakon nalaženja gustine naelektrisanja uz pomoć CPM metode, dobija se Hamiltonijan, koji je dalje potrebno dijagonalizovati. Kada se iskoristi činjenica da su u konkretnim fizičkim problemima od interesa samo određena stanja, i to ona na dnu provodne zone ili na vrhu valentne zone, postupak dijagonalizacije postaje značajno brži. Efikasne metode koje su razvijene za potrebe pronalaženja svojstvenih vrednosti, koje koriste prethodno navedenu osobinu, su metod preklopljenog spektra (folded spectrum method) [55] i metod preklopljenih fragmenata (the overlapping fragments method) [56]. Upotreba ovih metoda na računarima visokih performansi omogućila je računanje sistema sa više od deset hiljada atoma [57].

Do sada je CPM primenjen na različitim neorganskim i organskim poluprovodničkim sistemima, kao što su razređene nitridne smese (diluted nitrogen alloys) [48, 58], kvantne tačke i kvantne žice

(quantum dots and wires) [59, 60], primese/nečistoće [61, 62], ugljenične nanotube i fulereni [49], amorfni polimeri [63, 57, 64, 65, 66], termalno neuređeni polimeri [67], kao i interfejsi između domena u konjugovanim polimerima [68] i organskim poluprovodnicima baziranim na malim molekulima [69].

1.5 Diskretizacija Kohn-Sham jednačine

Da bi se Kohn-Sham jednačina rešila numerički, najpre je potrebno izvršiti diskretizaciju. U terminologiji numeričkog računanja se pod pojmom diskretizacija podrazumeva metod koji problem svodi na konačan broj nepoznatih. Postoje tri metode koje se smatraju najrasprostranjenijim: metod baziran na ravnim talasima, metod baziran na lokalizovanim orbitalama Gausijana i metod koji ne koristi bazisne skupove već se bazira na prostornim koordinatama, odnosno gde se primenjuje reprezentacija u realnom prostoru.

Pošto ravni talasi predstavljaju ortonormiran kompletan skup, takvim skupom bazisa se sve funkcije iz klase neprekidnih normalizovanih funkcija mogu opisati sa proizvoljnom preciznošću. Tako definisan skup se smatra univerzalnim, jer ne zavisi ni od pozicije atoma niti od njegove prirode (tipa atoma). Na taj način se izbegava uvođenje novog bazisnog skupa za svaki tip atoma u sistemu, kao i potreba da se takvi skupovi menjaju za različite tipove materijala, što je slučaj za gausijanske funkcije. Kod ravnih talasa je takođe moguće uz pomoć samo jednog parametra poboljšavati preciznost, odnosno povećavati ili smanjivati veličinu bazisa. Ova osobina je posebno važna za proračune dinamičkih sistema, gde se pozicija jezgra konstantno menja. Još jedna prednost je ta što je algebarska manipulacija ravnih talasa algoritamski jednostavna, pa je tako implementacija algoritma koji uključuju ravne talase značajno lakša od implementacija gde to nije slučaj (npr. onih koje se baziraju na gausijanima).

U slučaju diskretizacije primenom metode ravnih talasa, pored toga što je broj ravnih talasa velik, svaka dva ravna talasa imaju veliko prostorno preklapanje. Ovo rezultuje formiranjem matrice koja je velika i gusta. Ipak, u praksi se pokazalo da se umesto dijagonalizacije velike i guste matrice može primeniti elegantnije rešenje. Pogodnost kod ravnih talasa je ta što nije neophodno pamtiti celu matricu, već samo kako Hamiltonijan deluje na vektor. Hamiltonijan ima kinetički član koji je oblika Laplasijana, koji se nakon prelaska u recipročni prostor predstavlja kao obično množenje. Pri tom se pod prelaskom u recipročni prostor, podrazumeva prelazak iz reprezentacije talasne funkcije na uniformnom gridu u realnom prostoru, na reprezentaciju uz pomoć linearne kombinacije ravnih talasa. Druga komponenta koju imamo u Hamiltonijanu je potencijal koji zavisi samo od koordinate, i gde je dovoljno uraditi množenje u realnom prostoru. Kako kod ravnih talasa u algoritmu uvek imamo množenje u realnom i u recipročnom prostoru, u algoritmu se stalno prebacuje iz realnog u recipročni prostor (primenom Furijeove transformacije). Pošto postoje efikasni algoritmi za Furijeovu transformaciju, uz pomoć primene ravnih talasa je ipak moguće rešavati sisteme zadovoljavajućom brzinom.

Računanja bazirana na bazisima Gausijana sa konačnim, ograničenim brojem bazisnih funkcija nam u odnosu na ona baziranim na ravnim talasima, omogućavaju implementaciju efikasnijeg algoritma koji nam pri tom daje zadovoljavajuće precizan opis elektronskog stanja sistema. Preciznom opisu ukupne energije sistema doprinosi činjenica da je integrale dobijene u procesu računanja uz primenu bazisa Gausijana, moguće rešavati analitičkim računanjem. Iako se, kad je u pitanju brzina računanja, implementacije bazirane na gausijanskim funkcijama smatraju optimalnijim u odnosu

na implementacije bazirane na ravnim talasima, potrebno je ipak platiti određenu cenu. Pre svega, kompleksnost implementacije algoritama i softverskih rešenja je značajno veća, a osim toga gubi se univerzalnost zbog potrebe prilagođavanja bazisnog skupa konkretnim tipovima atoma [70]. Ukoliko je moguće prevazići ova dva glavna nedostatka, ovaj pristup nudi značajne prednosti i mogućnost rešavanja sistema sa velikim brojem atoma. U poređenju sa metodom baziranom na ravnim talasima, pri dijagonalizaciji sistema u slučaju primene Gausijana se formira matrica značajno manje veličine. S obzirom da je broj Gausijana po atomu mali, druga osobina ove matrice je da je retka.

Treća opcija za diskretizaciju, za razliku od prethodne dve, u potpunosti eliminiše potrebu za definisanjem bazisa već se oslanja na reprezentaciju u realnom prostoru. Pri diskretizaciji diferencijalne jednačine primenom metode u realnom prostoru, dobija se matrica kod koje su nenulti samo elementi koji odgovaraju susednim tačkama grida na kome je predstavljena talasna funkcija. Ovako dobijena matrica je, iako značajno veća od one dobijene primenom ravnih talasa, veoma retka. Prirodan odabir grida je uniforman grid, odnosno trodimenzionalna rešetka u prostoru. Međutim, kada znamo da je talasna funkcija lokalizovana u okolini atoma onda u toj okolini biramo gušći grid, dok se na drugim mestima bira ređi grid. S obzirom da formiranje grida na ovaj način nije jednostavno, postoji relativno mali broj rešenja koja implementiraju reprezentaciju u realnom prostoru. Konvergencija rešavanja svojstvenog problema se može poboljšati kada se umesto običnih konačnih razlika, primene konačne razlike višeg reda.

1.6 Dijagonalizacija Hamiltonijana

Postupak koji sledi nakon diskretizacije Kohn-Sham jednačina je dijagonalizacija matrice. Osim same veličine matrice, pri rešavanju svojstvenog problema se nailazi i na mnoge druge poteškoće. Prvi i najveći izazov je taj što je broj potrebnih svojstvenih vektora proporcionalan broju atoma u sistemu. Za veće sisteme se radi o nekoliko hiljada svojstvenih vektora, pa uzimajući u obzir memorijsku zahtevnost, postizanje ortogonalnosti tih vektora predstavlja izazov. Pored ovoga, ortogonalizacija je takođe i procesorski veoma zahtevna, pa iz aspekta brzine izvršavanja ona predstavlja jedan od najzahtevnijih postupaka u većini algoritama. Drugi problem koji se odnosi na dijagonalizaciju je što se pri povećanju matrice smanjuje relativno rastojanje između svojstvenih vrednosti. Ovaj efekat ima negativan uticaj na brzinu konvergencije i time značajno otežava postupak za rešavanje svojstvenog problema. Da bi se ovaj problem ublažio, potrebno je primeniti određene tehnike podešavanja početnih uslova, čime se postupak može značajno ubrzati [7].

Algoritmi koji se najčešće primenjuju u procesu dijagonalizacije kod ravnih talasa (u slučaju retkih matrica) su Lanczos [71] i Davidson [72]. Sa teorijske strane, Lanczos metod na efikasan način pronalazi 'korisni' podskup svojstvenih vrednosti i svojstvenih vektora, sve dok je veličina podskupa značajno manja od veličine matrice [73]. Međutim, zbog numeričke nestabilnosti, originalna formulacija ovog metoda je teško primenjiva u praksi [7]. Razlog ovome je što se već u početnim iteracijama, gubi ortogonalnost Lanczos vektora [74]. Da bi se ovaj problem prevazišao, potrebno je pravovremeno detektovati gubitak ortogonalnosti i primeniti korake koji bi omogućili uspešnu konvergenciju [75, 76, 77]. Softverske biblioteke koje uspešno implementiraju ovaj pristup su PROPACK [78] i PLAN [79].

Druga popularna tehnika za računanje podskupa svojstvenih vrednosti je primena Davidson metoda, koji je efikasan za računanje manjeg podskupa najnižih svojstvenih vrednosti za retke,

dijagonalno-dominante matrice. Ono što je od interesa za proces dijagonalizacije, koji se ovde opisuje, je modifikacija ove metode tako da ona postaje primenjiva i na simetrične matrice koje nisu obavezno dijagonalno-dominantne [80]. Za razliku od Lanczos metoda, Davidson metod umesto direktne primene Krylov subspace metoda, koristi iterativnu Rayleigh-Ritz proceduru koja se sastoji od petlje (koja pojedinačno izračunava tražene svojstvene vrednosti) i 'baze' (koja postepeno izračunava potprostor na koji se vrši projekcija). Takođe se može reći da je Davidson metod verzija Lanczos procedure sa preduslovima, gde se preduslovi ne moraju striktno odnositi na dijagonalu matrice. DFT programi koji koriste ovu iterativnu šemu su ABINIT [81], VASP [22, 9], kao i PWscf (deo ESPRESSO softvera [20]), koji za razliku od prethodna dva, kao osnovni metod za dijagonalizaciju koristi upravo Davidson metod.

Pošto je veličina bazisnog skupa Gausijana mala, matrica koja se pri tom formira je u odnosu na broj atoma relativno mala i retka. Ovo znači da je, u slučaju implementacije bazirane na Gausijanima, dijagonalizacija vremenski značajno kraća u odnosu na proces konstrukcije Hamiltonijana. S obzirom na to, moguće je primeniti standardnu LAPACK [82] dijagonalizaciju. Međutim, kako je rešavanje svojstvenog problema za velike sisteme (više od 10000 atoma) ipak resursno zahtevno, svaka dodatna optimizacija značajno doprinosi celokupnom algoritmu. Jedan od načina da se smanji vreme izvršavanja je da se iskoristi činjenica da nije neophodno računati ceo spektar svojstvenih vrednosti. Za relevantan skup svojstvenih energija se smatraju one koje predstavljaju poslednja popunjena i prva prazna stanja (HOMO i LUMO orbitale). Uzima se da su od fizičkog značaja ona stanja sa energijama u nekom unapred zadatom intervalu iznad najnižeg praznog i ispod najvišeg popunjenog stanja. Metod koji koristi činjenicu da je umesto celog spektra potreban samo određeni podskup i koji omogućava odabir odgovarajućeg dela spektra je Krylov-Schur metod [83], [84], koji se po SLEPc dokumentaciji [85] smatra najefikasnijim za rešavanje ovog tipa problema. Krylov-Schur metod predstavlja unapređenje tradicionalnog Krylov subspace metoda (kao što je pomenuti Lanczos metod), koji sistemom takozvanog mehanizma za restartovanje računanja, eliminiše računanje komponenti koje se odnose na svojstvene vrednosti koje nisu u željenom opsegu. Na ovaj način je moguće odabrati bilo koji skup vrednosti iz spektra i time proporcionalno smanjiti vreme računanja.

1.7 Ciljevi i doprinosi

Cilj implementacija predstavljenih u ovoj tezi je postizanje optimalnog rešenja za računanje elektronske strukture materijala, koje omogućava da se na klasterskim resursima može vršiti izračunavanje sistema od nekoliko desetina hiljada atoma. Ovo je moguće postići implementiranjem paralelne verzije CPM-a uz korišćenje računara visokih performansi.

Efikasnost CPM implementacije u mnogome zavisi od efikasnosti algoritama za računanje gausijanskih integrala koje je potrebno implementirati u sklopu DFT-a, pa je tako prvobitno uložen napor u odabir odgovarajućih algoritamskih rešenja a zatim je posebna pažnja posvećena optimizaciji i preciznosti rezultata. Pored uspostavljanja dobre baze implementiranjem računanja gausijanskih integrala i implementiranjem serijske verzije programa za DFT i CPM, takođe je potrebno prevazići ograničenja do kojih se neminovno dolazi upotrebnom serijskog pristupa. Prvo ograničenje je vremensko, koje se javlja usled ograničenih mogućnosti procesiranja jednog CPU jezgra. Za postojeća softverska rešenja je potrebno nekoliko desetina sati za računanje sistema od nekoliko stotina atoma.

Drugi problem je ograničenje kapaciteta radne memorije računara.

Da bi se navedeni problemi prevazišli, potrebna je paralelna verzija implementacije CPM programa i odgovarajuće rešenje za distribuciju podataka potrebnih za njegovo izvršavanje. Ovo je moguće jedino uz upotrebu paralelne računarske arhitekture gde je zbog prirode problema najpogodnija računarska arhitektura sa distribuiranom memorijom. U ovakvom okruženju je moguće pokretati paralelne programske procese koji međusobno komuniciraju u cilju razmene podataka. Ovakvu arhitekturu imaju računarski klasteri sastavljeni od pojedinačnih računara, odnosno računarskih nodova, međusobno povezanih računarskom mrežom.

Za analizu tačnosti rezultata i prikaz performansi CPM programa je neophodno sprovesti opsežno testiranje nad različitim klasama sistema. Testovi performansi CPM programa se dele u dve grupe: prva grupa testova prikazuje zavisnost vremena izvršavanja CPM programa u odnosu na veličinu ulaznog sistema, dok druga grupa prikazuje jako skaliranje (strong scaling), odnosno skaliranje broja paralelnih procesa. U prvoj grupi testova se vrši promena veličine sistema, dok se testovi izvršavaju pod istim uslovima. Druga grupa testova podrazumeva fiksnu veličinu problema koji se rešava, dok se vrši skaliranje broja paralelnih procesa. Kao referentna vrednost za određivanje efikasnosti CPM programa se koristi vreme izvršavanja testova sa najmanjim brojem paralelnih procesa. Kako na efikasnost izvršavanja mogu uticati razni faktori, potrebno je analizirati različite pristupe i formulisati način za formiranje okvirne procene koja se može koristiti kao referentna pri modeliranju simulacija. Okvirna procena se odnosi na određivanje optimalne upotrebe resursa na osnovu tačke zasićenja, odnosno unapred određenog broja paralelnih procesa, nakon čijeg povećanja performanse izvršavanja opadaju.

1.8 Pregled sadržaja teze

U tezi su predstavljene serijska i paralelna implementacija CPM-a bazirana na reprezentaciji talasne funkcije u bazisu gausijanskih funkcija. Kompletno softversko rešenje obuhvata četiri programa: serijski DFT program za mali prototip sistem, serijski program za ekstrahovanje motiva, serijski CPM program i paralelni MPI CPM program. Navedeni programi pokrivaju celokupan postupak koji uključuje učitavanja korisnički definisanih ulaznih sistema, rešavanja malog prototip sistema uz pomoć DFT-a, generisanje motiva i rešavanje elektronske strukture materijala za velike sisteme uz pomoć CPM-a. Produkcione verzije programa sadrže približno 20000 linija koda implementiranog u programskom jeziku C.

Ove implementacije, u odnosu na implementacije bazirane na ravnim talasima, daju značajno ubrzanje procesa računanja elektronske strukture materijala. Sadržaj teze je organizovan na sledeći način. Poglavlje 2 daje uvod u teoriju funkcionala gustine (DFT) i metod za sklapanje naelektrisanja (CPM). Ovo poglavlje takođe sadrži pregled CPM algoritma koji ujedno najavljuje faze razvoja koje se u više detalja opisuju u nastavku. Poglavlje 3 sadrži sveobuhvatan opis implementacije algoritama u bazisu Gausijana. Iako sa implementacionog aspekta algoritmi za DFT i CPM imaju zajedničke korake, radi boljeg pregleda je poglavlje podeljeno u dve sekcije. Najpre su opisani algoritmi koji se odnose na DFT (sekcija 3.1) a koji se delom takođe odnose i na procese korišćene u CPM implementaciji (sekcija 3.2), a zatim i koraci koji se specifično odnose na CPM implementaciju. Sledeće poglavlje (4) daje opis implementacije softverskih rešenja koja se oslanjaju na algoritme uvedene u poglavlju 3. Najpre je dat opis globalnih struktura koda, definicija softverskih

komponenata koje su sadržane u konačnoj verziji programske implementacije, kao i način alokacije memorije. Posle ovoga sledi centralno mesto teze, a to je implementacija programskih rešenja. Kako je glavni doprinos iz aspekta implementacije koda onaj koji se odnosi na paralelizaciju CPM, tako je manja pažnja posvećena implementaciji serijskih algoritama (sekcija 4.2), dok je primat stavljen na implementaciju paralelnih algoritama u sekciji 4.3. Poslednje poglavlje teze (5) prikazuje analizu performansi i rezultata serijskog DFT i CPM programa (5.1), kao i paralelne verzije CPM programa (5.2). Obe sekcije najpre uvode klase sistema koji su testirani, zatim analiziraju rezultate testova i prikazuju zahtevnost izvršavanja. U delu koji se bavi testiranjem paralelne verzije CPM-a je posebna pažnja posvećena performansama izvršavanja i odabiru optimalnog scenarija upotrebe programa, u zavisnosti od ulaznih sistema i raspoloživih resursa.

Poglavlje 2

Metode za računanje elektronske strukture materijala

Razvoj algoritma za računanje elektronske strukture materijala opisan u ovom radu se u osnovi bazira na dve metode: teorija funkcionala gustine (DFT) i metod za sklapanje naelektrisanja (CPM). Ovo poglavlje u sekciji 2.1 daje kratak teorijski uvod u DFT metodu a zatim u sekciji 2.2 uvodi CPM i daje pregled CPM algoritma koji ujedno najavljuje faze razvoja koje se u više detalja opisuju u poglavljima koja slede.

2.1 Teorija funkcionala gustine (Density Functional Theory, DFT)

Za standardan DFT proračun, potrebno je samosaglasno rešiti sledeću Kohn-Sham jednačinu [5]

$$\left(-\frac{\hbar^2}{2m_0}\nabla^2 + V_{ion} + V_H\left[\rho\right] + V_{xc}\left[\rho\right]\right)\psi_i(\mathbf{r}) = \varepsilon_i\psi_i(\mathbf{r}), \tag{2.1}$$

kao i jednačinu koja povezuje elektronsku gustinu naelektrisanja $\rho\left(\mathbf{r}\right)$ i jednočestične talasne funkcije $\psi_i\left(\mathbf{r}\right)$:

$$\rho\left(\mathbf{r}\right) = -|e|\sum_i \left|\psi_i\left(\mathbf{r}\right)\right|^2. \tag{2.2}$$

U prethodnim jednačinama m_0 je masa elektrona, $\hbar$ redukovana Plankova konstanta, e elementarno naelektrisanje, V_{ion} potencijal jezgra, $V_H\left[\rho\right]$ elektrostatički (Hartri) potencijal valentne elektronske gustine naelektrisanja $\rho\left(\mathbf{r}\right)$, $V_{xc}\left[\rho\right]$ izmensko-korelacioni (exchange-correlation) potencijal koji u okviru aproksimacije lokalne gustine (local density approximation, LDA) zavisi isključivo od elektronske gustine naelektrisanja u datoj tački u prostoru, dok je ε_i svojstvena energija jednočestičnog stanja i. Suma iz jednačine (2.2) uključuje sva popunjena energetska stanja i.

Samosaglasni postupak za rešavanje DFT-a podrazumeva da se, polazeći od početne pretpostavke za gustinu naelektrisanja $\rho\left(\mathbf{r}\right)$, rešava sistem jednačina (2.1), čime se dobijaju vrednosti talasnih funkcija $\psi_i\left(\mathbf{r}\right)$. Dalje se rešavanjem jednačine (2.2) dobiju nove vrednosti $\rho\left(\mathbf{r}\right)$, koje se porede sa vrednostima iz prethodnog koraka. Ovaj postupak se ponavlja sve dok se ne dođe do konvergencije.

2.2 Metod za sklapanje naelektrisanja (Charge Patching Method, CPM)

Osnovna ideja CPM-a [48, 50] je da se direktnom konstrukcijom gustine naelektrisanja $\rho(\mathbf{r})$ izbegne potreba za izvršavanjem samosaglasne procedure za rešavanje jednačine (2.1) i (2.2). Ovo je moguće upravo iz razloga što gustina naelektrisanja u susedstvu određenog atoma zavisi isključivo od lokalnog okruženja. Tada je najpre potrebno izvršiti pun samosaglasan DFT proračun na malom prototip sistemu (pogledati korak 1 sa slike 2.1) gde atomi imaju isto okruženje kao i u velikom sistemu. Prototip sistem se može najjednostavije predstaviti kao minimalni gradivni elemenat velikog sistema, tj. kao struktura koja sadrži sve potrebne elemente za formiranje velikog sistema. Gustina naelektrisanja $\rho_s(\mathbf{r})$ dobijena takvim proračunom se zatim razlaže na doprinose pojedinačnih atoma, takozvanih motiva (korak 2 sa slike 2.1), koristeći sledeću formulu:

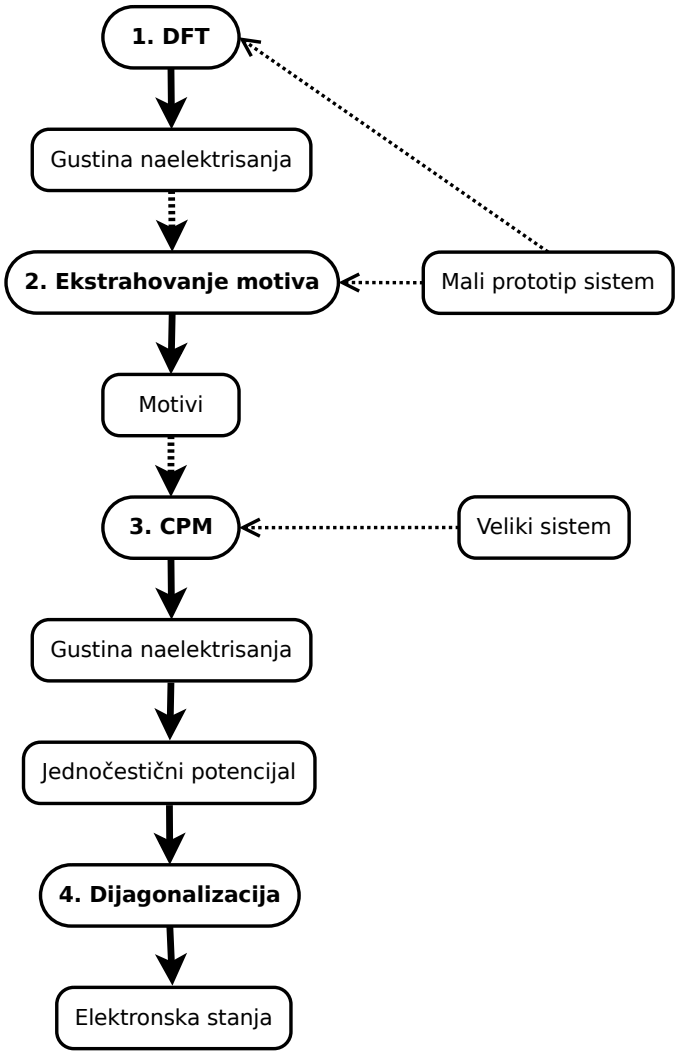
$$m_A(\mathbf{r}-\mathbf{R}_A) = \frac{w_A(\mathbf{r}-\mathbf{R}_A)}{\sum_B w_B(\mathbf{r}-\mathbf{R}_B)}\rho_s(\mathbf{r}). \tag{2.3}$$

U jednačini (2.3) $\mathbf{R}_A$ označava poziciju atoma A. Suma u deliocu obuhvata sve atome u sistemu (označeno sa B), dok su $w_A(\mathbf{r}-\mathbf{R}_A)$ funkcije lokalizovane u susedstvu atoma A i koriste se za glatko razlaganje gustine naelektrisanja $\rho_s(\mathbf{r})$ u motive $m_A(\mathbf{r}-\mathbf{R}_A)$. Dalje, w_A dobijamo množenjem elektronske gustine naelektrisanja atoma A i 'mask' funkcije $f_m(r)$, gde 'mask' funkcija ima oblik $f_m(r) = (a+br^2)/a$ kada je $r \leq r_0$, i $f_m(r) = e^{-e_p r}/a$ kada je $r > r_0$, sa $r_0 = 3\,a_0$ (gde a_0 označava Borov radijus), $e_p = 0.75\,{a_0}^{-1}$, $b = -e_p \exp(-e_p r_0)/(2r_0)$, i $a = \exp(-e_p r_0) - br_0^2$. Za klasifikovanje motiva i atoma se koristi ideja bazirana na načinu na kojem se klasifikuju atomi prilikom opisa sistema korišćenjem klasičnog potencijala interakcije, koji se najčešće koristi u simulacijama baziranim na klasičnoj molekularnoj dinamici. Na primer, u slučaju alkanskih lanaca, uvodimo dva tipa C atoma – atome na kraju lanca označavamo sa C_3, dok atome iz unutrašnjosti lanca označavamo sa C_2. Motive određujemo na osnovu tipa njegovog centralnog atoma i na osnovu njegovih suseda. U slučaju atoma vodonika je takođe potrebno uvrstiti i susede drugog reda, što nam omogućava da na pravilan način orijentišemo motive u trodimenzionalnom prostoru. Prethodno iskustvo je pokazalo da se ovakav pristup može smatrati uspešnim za široki spektar poluprovodničkih sistema. Na primer, za slučaj alkana su potrebni sledeći motivi: $C_3 - C_2HHH$, $C_2 - C_3C_2HH$, $C_2 - C_2C_2HH$, $H - C_3 - C_2HH$, $H - C_2 - C_3C_2H$ i $H - C_2 - C_2C_2H$. Primeri skupova motiva koji su korišćeni za druge sisteme se mogu pronaći u poglavlju 5 u tabeli 5.1.

Da bismo izračunali gustinu naelektrisanja za veliki sistem, potrebno je prosto za svaki pojedinačni atom dodavati doprinose motiva (korak 3 na slici 2.1)

$$\rho(\mathbf{r}) = \sum_A m_A\left[\mathcal{R}_A^{-1}\cdot(\mathbf{r}-\mathbf{R}_A)\right], \tag{2.4}$$

gde je $\mathcal{R}_A$ matrica rotacije koja uzima u obzir da se prostorna orijentacija susednog atoma u malom sistemu koji se koristi za generisanje motiva, potencijalno razlikuje u odnosu na njegovu orijentaciju u velikom sistemu koji želimo da izračunamo. Konkretno, ukoliko je $\mathbf{R}_{A_i}$ pozicija suseda atoma A a $\mathbf{R}_A^{(0)}$ i $\mathbf{R}_{A_i}^{(0)}$ pozicije atoma A i njegovih suseda u motivu, onda je $\mathcal{R}_A\cdot\left(\mathbf{R}_{A_i}^{(0)} - \mathbf{R}_A^{(0)}\right) = \mathbf{R}_{A_i} - \mathbf{R}_A$. Kada raspolažemo elektronskom gustinom naelektrisanja, moguće je direktno dobiti jednočestični Hamiltonijan [član unutar zagrade u levoj strani jednačine (2.1)], koji se nakon toga dijagonalizuje (korak 4 na slici 2.1) da bi se tako dobile energije i talasne funkcije elektronskih stanja.



Slika 2.1: CPM dijagram opisuje kompletni CPM algoritam organizovan u fazama. Prva faza implementira DFT algoritam za računanje gustine naelektrisanja za mali prototip sistem. Gustina naelektrisanja dobijena iz DFT-a se nakon toga koristi u drugoj fazi za ekstrahovanje motiva. Ovi motivi se dalje koriste u trećoj fazi za računanje elektronske strukture velikog sistema.

U slučaju implementacije uz pomoć ravnih talasa dijagonalizacija Hamiltonijana za velike sisteme može biti prilično zahtevna za izračunavanje. Razlog ovome je što imamo veliki skup ravnih talasa i veliki broj svojstvenih stanja. Pored toga, transportna i optička svojstva materijala se određuju isključivo na osnovu elektronskih stanja u okolini energetskog procepa. Iz tog razloga nije neophodno pronaći sve svojstvene vrednosti iz jednačine (2.1) već samo one koje su u spektralnoj oblasti blizu energetskog procepa. Efikasne metode koje su razvijene za potrebe pronalaženja ovih svojstvenih vrednosti su metod preklopljenog spektra (folded spectrum method) [55] i metod preklopljenih fragmenata (the overlapping fragments method) [56]. Upotreba ovih metoda na računarima visokih performansi omogućava računanje sistema sa više od deset hiljada atoma [57].

Poređenje rezultata dobijenih uz pomoć CPM-a i punog DFT računanja za više različitih poluprovodničkih sistema pokazuje odlično slaganje svojstvenih vrednosti. One su reda veličine od 10–30 meV [50]. Do sada je CPM primenjen na različitim neorganskim i organskim poluprovodničkim sistemima, kao što su razređene nitridne smese (diluted nitrogen alloys) [48, 58], kvantne tačke i kvantne žice (quantum dots and wires) [59, 60], primese/nečistoće [61, 62], ugljenične nanotube i fulereni [49], amorfni polimeri [63, 57, 64, 65, 66], termalno neuređeni polimeri [67], kao i interfejsi između domena u konjugovanim polimerima [68] i organskim poluprovodnicima baziranim na malim molekulima [69]. S druge strane, od CPM-a se ne očekuje da radi dobro u slučaju metala ili sistema gde je prisutan dugodometni prenos naelektrisanja, zato što se ne očekuje da je u ovim slučajevima zadovoljena osnovna pretpostavka da elektronska gustina naelektrisanja zavisi samo od lokalnog okruženja.

Poglavlje 3

Implementacija u bazisu Gausijana

Implementacija u bazisu Gausijana, bazirana na metodima opisanih u poglavlju 2, podrazumeva implementaciju DFT i CPM gde se talasna funkcija prikazuje kao linearna kombinacija predefinisanih bazisnih funkcija, čiju osnovu čine Gausijani. Prateći tok dijagrama na slici 2.1, da bi se dobila gustina naelektrisanja potrebna za ekstrahovanje motiva najpre se definišu algoritmi za rešavanje svih potrebnih integrala za računanje Hamiltonijana (sekcije 3.1.1-3.1.5). Dalje se u sekciji 3.1.6 opisuje kako se algoritamski unapređuje konvergencija pri rešavanju svojstvenog problema DFT algoritma. Da bi se zaokružila slika, poslednja sekcija implementacije DFT-a daje primer upotrebe Gausijana za pronalaženje energije osnovnog stanja atoma vodonika. Drugi deo ovog poglavlja opisuje CPM implementaciju, tako što najpre uvodi CPM algoritam (sekcija 3.2) a zatim se u sekciji 3.2.1 opisuje ekstrahovanje motiva na osnovu gustine naelektrisanja dobijenog iz DFT proračuna. Optimizacija algoritma uz pomoć odsecanja pri računanju integrala je opisana u sekciji 3.2.2. Na kraju u sekciji 3.2.3 je opisana dijagonalizacija Hamiltonijana.

3.1 Implementacija DFT u bazisu Gausijana

Za rešavanje svojstvenog problema datog u jednačini (2.1), talasnu funkciju $\psi_i(\mathbf{r})$ prikazujemo kao linearnu kombinaciju predefinisanih bazisnih funkcija $\varphi_{k\mu}(\mathbf{r}-\mathbf{R}_k)$

$$\psi_i(\mathbf{r}) = \sum_{k=1}^{N}\sum_{\mu=1}^{N_k} \alpha_{k\mu}^{(i)} \varphi_{k\mu}(\mathbf{r}-\mathbf{R}_k). \tag{3.1}$$

U jednačini (3.1) N je ukupan broj atoma u sistemu, N_k je broj bazisnih funkcija centriranih na atomu k, $\mathbf{R}_k$ je pozicija atoma k, dok je $\alpha_{k\mu}^{(i)}$ koeficijent ekspanzije koji je potrebno odrediti. Svaka od bazisnih funkcija je u obliku kontrahovanog Gausijana, odnosno data je kao linearna kombinacija određenog broja primitivnih Gausijana:

$$\varphi_{lmn}(\mathbf{r}) = \sum_j a_j \chi_{lmn\alpha_j}(\mathbf{r}), \tag{3.2}$$

sa predefinisanim koeficijentima a_j, gde su primitivni Gausijani u sledećem obliku

$$\chi_{lmn\alpha}(\mathbf{r}) = N(l,m,n,\alpha)\, x^l y^m z^n e^{-\alpha r^2}. \tag{3.3}$$

U jednačini (3.3) l, m i n su nenegativni celi brojevi, α je pozitivan realan broj, dok je $N(l,m,n,\alpha)$ konstanta normalizacije

$$N(l,m,n,\alpha) = \left(\frac{2\alpha}{\pi}\right)^{3/4} \sqrt{\frac{(8\alpha)^{l+m+n}\ l!\,m!\,n!}{(2l)!\ (2m)!\ (2n)!}}. \tag{3.4}$$

U numeričkom primeru u poglavlju 5 koristimo SBKJC VDZ ECP bazisni skup [86, 87, 53, 54].

Elektronsku gustinu naelektrisanja takođe predstavljamo preko bazisa kontrahovanih Gausijana centriranih na atomima:

$$\rho(\mathbf{r}) = \sum_{k=1}^{N}\sum_{\mu=1}^{n_k} \beta_{k\mu}\varphi_{k\mu}^{C}(\mathbf{r}-\mathbf{R}_k). \tag{3.5}$$

Međutim, treba imati u vidu da je skup Gausijana $\varphi_{k\mu}^{C}(\mathbf{r}-\mathbf{R}_k)$ koji se koristi u jednačini (3.5), različit od skupa koji se pojavljuje u jednačini (3.1) (ova razlika je naglašena eksponentom C). Razlog za korišćenje različitih skupova Gausijana potiče iz činjenice da je elektronska gustina naelektrisanja jednaka sumi kvadrata modula talasnih funkcija popunjenih elektronskih stanja. Iz ovoga proizilazi da je za dobar prikaz gustine naelektrisanja potrebno koristiti različite parametre Gausijana u odnosu na one koji su nam potrebni za talasnu funkciju. Stoga imamo da je broj Gausijana centriran na atomu k, u jednačini (3.5) označen sa n_k, različit od broja N_k iz jednačine (3.1). U numeričkom primeru u poglavlju 5 koristimo DGauss A1 DFT Coulomb Fitting bazisni skup [52, 53, 54]. n_k je u većini slučajeva veći od N_k. Na primer, u slučaju atoma ugljenika i bazisnog skupa koji koristimo u tom numeričkom primeru je $n_k = 34$, dok je $N_k = 8$.

Da bismo dobili svojstvene vrednosti i talasne funkcije, neophodno je rešiti generalizovani svojstveni problem:

$$\sum_{l\nu}\left(H_{k\mu,l\nu} - \varepsilon_i S_{k\mu,l\nu}\right)\alpha_{l\nu}^{(i)} = 0, \tag{3.6}$$

gde su $S_{k\mu,l\nu}$ elementi matrice preklapanja:

$$S_{k\mu,l\nu} = \int \mathrm{d}^3\mathbf{r}\,\varphi_{k\mu}(\mathbf{r}-\mathbf{R}_k)\ \varphi_{l\nu}(\mathbf{r}-\mathbf{R}_l), \tag{3.7}$$

i gde su $H_{k\mu,l\nu}$ elementi jednočestične Hamiltonijan matrice. Da bismo dobili elemente ove matrice, je potrebno izračunati kinetički integral:

$$H_{k\mu,l\nu}^{(1)} = \int \mathrm{d}^3\mathbf{r}\,\varphi_{k\mu}(\mathbf{r}-\mathbf{R}_k)\ \left(-\frac{\hbar^2}{2m_0}\nabla^2\right)\varphi_{l\nu}(\mathbf{r}-\mathbf{R}_l), \tag{3.8}$$

integral koji potiče iz potencijala jezgra i nevalentnih elektrona:

$$H_{k\mu,l\nu}^{(2)} = \int \mathrm{d}^3\mathbf{r}\,\varphi_{k\mu}(\mathbf{r}-\mathbf{R}_k)\ V_{ion}\,\varphi_{l\nu}(\mathbf{r}-\mathbf{R}_l), \tag{3.9}$$

Hartrijev integral:

$$H_{k\mu,l\nu}^{(3)} = \int \mathrm{d}^3\mathbf{r}\,\varphi_{k\mu}(\mathbf{r}-\mathbf{R}_k)\ V_H[\rho]\ \varphi_{l\nu}(\mathbf{r}-\mathbf{R}_l), \tag{3.10}$$

i izmensko-korelacioni integral:

$$H_{k\mu,l\nu}^{(4)} = \int \mathrm{d}^3\mathbf{r}\,\varphi_{k\mu}(\mathbf{r}-\mathbf{R}_k)\ V_{xc}[\rho]\ \varphi_{l\nu}(\mathbf{r}-\mathbf{R}_l). \tag{3.11}$$

Implementacija izračunavanja svih integrala pojedinačno je predstavljena u nastavku.

Algoritam za rešavanje DFT-a se sastoji od sedam koraka:

1. Gustina naelektrisanja se inicijalizuje na početnu vrednost.

2. Konstruišu se matrice H i S na osnovu jednačina (3.7), (3.8), (3.9), (3.10) i (3.11) (opisi postupaka su redom dati u sekcijama 3.1.1, 3.1.2, 3.1.3).

3. Primeni se postupak za pobošljanje konvergencije opisan u sekciji 3.1.6.

4. Reši se generalizovani svojstveni problem (3.6).

5. Provera konvergencije. Ukoliko svojstvene energije nisu konvergirale, vratiti se na korak 2.

Velika prednost predstavljanja talasne funkcije pomoću Gausijana je da se većina relevantnih integrala koji se pojavljuju u rešavanju elektronske strukture, mogu izračunati analitički. Da bismo izračunali kinetički [jednačina (3.8)] i integral preklapanja [jednačina. (3.7)], potrebno je da direktno koristimo analitičke formule, koje su date u nastavku. Za detaljan uvid u izvođenje ovih integrala pogledati [88].

3.1.1 Integral preklapanja

Najpre ćemo predstaviti analitičko rešenje integrala preklapanja (Overlap Integral) koji se pojavljuje u jednačini (3.7). Rešenje ovog integrala se sastoji od kompleksnih ugnježdenih suma. S obzirom na broj pojavljivanja ovih integrala, radi potencijalno značajnog uticaja na perfomanse programa je potrebno posvetiti pažnju efikasnoj implementacija ovih suma.

Ukoliko primenimo jednačinu (3.2), tada jednačina (3.7) ima sledeći oblik:

$$S_{k\mu,l\nu} = \int \mathrm{d}^3\mathbf{r} \sum_i a_i \chi_{k\mu_i}\left(\mathbf{r}-\mathbf{R_k}\right) \sum_j a_j \chi_{l\nu_j}\left(\mathbf{r}-\mathbf{R_l}\right). \tag{3.12}$$

Radi lakšeg prikaza, uvodimo notaciju $\chi_A\left(\mathbf{r}\right) \equiv \chi_{k\mu_i}\left(\mathbf{r}-\mathbf{R_k}\right)$. Analogno uvodimo i notaciju za $\chi_B\left(\mathbf{r}\right)$. Iz jednačine (3.12) se vidi da se svaki element matrice $S_{k\mu,l\nu}$ može dobiti rešavanjem odgovarajućeg integrala preklapanja. Ukoliko primitivne Gausijane $\chi_A\left(\mathbf{r}\right)$ i $\chi_B\left(\mathbf{r}\right)$ iz jednačine (3.3) centrirane na atomima **A** i **B** predstavimo kao

$$\chi_{\mathbf{A}}\left(\mathbf{r}\right) = N_1\left(l_1,m_1,n_1,\alpha_1\right)\left(x-\mathbf{A}_x\right)^{l_1}\left(y-\mathbf{A}_y\right)^{m_1}\left(z-\mathbf{A}_z\right)^{n_1} e^{-\alpha_1 r^2}, \tag{3.13}$$

$$\chi_{\mathbf{B}}\left(\mathbf{r}\right) = N_2\left(l_2,m_2,n_2,\alpha_2\right)\left(x-\mathbf{B}_x\right)^{l_2}\left(y-\mathbf{B}_y\right)^{m_2}\left(z-\mathbf{B}_z\right)^{n_2} e^{-\alpha_2 r^2}, \tag{3.14}$$

gde su N_1 i N_2 konstante normalizacije definisane u jednačini (3.4), tada definiciju integrala preklapanja možemo zapisati kao:

$$\langle \mathbf{A}|\mathbf{B}\rangle = \int_{-\infty}^{\infty} d^3\mathbf{r}\chi_{\mathbf{A}}\left(\mathbf{r}\right)\chi_{\mathbf{B}}\left(\mathbf{r}\right). \tag{3.15}$$

Analitičko rešenje tada ima sledeći oblik:

$$\langle \mathbf{A}|\mathbf{B}\rangle = N_1 N_2 \left(\frac{\pi}{\gamma_p}\right)^{3/2} e^{-\eta_p(\mathbf{A}-\mathbf{B})^2} \sum_{i_1,i_2,o} \mathcal{S}_x \sum_{j_1,j_2,p} \mathcal{S}_y \sum_{k_1,k_2,q} \mathcal{S}_z,$$

gde je $\gamma_p = \alpha_1 + \alpha_2$ i $\eta_p = \alpha_1\alpha_2/\gamma_p$. Dalje je

$$\begin{aligned}
&\sum_{i_1,i_2,o} \mathcal{S}_x = \frac{(-1)^{l_1} l_1! l_2!}{\gamma_p^{l_1+l_2}} \cdot \\
&\sum_{i_1}\sum_{i_2}\sum_{o} \frac{(-1)^o \Omega! \alpha_1^{l_2-i_1-2i_2-o} \alpha_2^{l_1-2i_1-i_2-o}}{4^{i_1+i_2+o} i_1! i_2! o!} \cdot \\
&\frac{\gamma_p^{2(i_1+i_2)+o} (\mathbf{A}_x-\mathbf{B}_x)^{\Omega-2_o}}{(l_1-2i_1)!(l_2-2i_2)!(\Omega-2o)!},
\end{aligned} \tag{3.16}$$

gde je $\Omega = l_1+l_2-2(i_1+i_2)$ i gde su opsezi u sumama $i_1 = 0 \to \left[\frac{1}{2}l_1\right]$, $i_2 = 0 \to \left[\frac{1}{2}l_2\right]$, $o = 0 \to \left[\frac{1}{2}\Omega\right]$. Sume $\mathcal{S}_y$ i $\mathcal{S}_z$ se za komponente y i z definišu analogno sumi $\mathcal{S}_x$.

3.1.2 Kinetički integral

Kada bazisne funkcije iz jednačine (3.8) prikažemo kao sume kontrahovanih Gausijana iz jednačine (3.3), dobijamo sledeći izraz:

$$H^{(1)}_{k\mu,l\nu} = \int \mathrm{d}^3\mathbf{r} \sum_i a_i \chi_{k\mu_i}(\mathbf{r}-\mathbf{R_k}) \left(-\frac{\hbar^2}{2m_0}\nabla^2\right) \sum_j a_j \chi_{l\nu_j}(\mathbf{r}-\mathbf{R_l}). \tag{3.17}$$

Kao i u slučaju integrala preklapanja, za primitivne Gausijane uvodimo izraze $\chi_A(\mathbf{r})$ (3.13) i $\chi_B(\mathbf{r})$ (3.14). Sada se iz jednačine (3.17) može videti da je za računanje matrice $H^{(1)}_{k\mu,l\nu}$ potrebno izračunati kinetičke integrale:

$$\langle \mathbf{A}| -\frac{1}{2}\nabla^2 |\mathbf{B}\rangle = \int_{-\infty}^{\infty} d^3\mathbf{r}\chi_{\mathbf{A}}(\mathbf{r}) \left[-\frac{1}{2}\nabla^2\chi_{\mathbf{B}}(\mathbf{r})\right]. \tag{3.18}$$

Analitično rešenje kinetičkog integrala (Kinetic Integral) se sada može predstaviti u sledećem obliku:

$$\begin{aligned}
\langle \mathbf{A}| -\frac{1}{2}\nabla^2 |\mathbf{B}\rangle &= \frac{1}{2}[\alpha_2(4(l_2+m_2+n_2)+6)\langle \mathbf{A}|\mathbf{B}\rangle \\
&- 4\alpha_2^2(\langle \mathbf{A}|\mathbf{B}, l_2+2\rangle + \langle \mathbf{A}|\mathbf{B}, m_2+2\rangle + \langle \mathbf{A}|\mathbf{B}, n_2+2\rangle) \\
&- l_2(l_2-1)\langle \mathbf{A}|\mathbf{B}, l_2-2\rangle \\
&- m_2(m_2-1)\langle \mathbf{A}|\mathbf{B}, m_2-2\rangle \\
&- n_2(n_2-1)\langle \mathbf{A}|\mathbf{B}, n_2-2\rangle].
\end{aligned} \tag{3.19}$$

Delovi jednačine koji imaju oblik $|\mathbf{R}, k-i\rangle$, označavaju Gausijan centriran na $\mathbf{R}$ čiji je stepen $k \in \{l, m, n\}$ 'pomeren' za vrednost i.

3.1.3 Integral jezgra

U ovom delu rada ćemo opisati način za rešavanje integrala pseudopotencijala (efektivni potencijal jezgra / Nuclei and Core Electron Integral), u kome nemamo eksplicitno tretiranje određenog broja nevalentnih elektrona već se njihov efekat modeluje upotrebom operatora pseudopotencijala. Doprinos atoma k operatoru V_{ion} iz jednačine (3.9) se prikazuje kao

$$V^k_{ion} = V^k_{pp} - \frac{Z_k e^2}{4\pi\varepsilon_0 |\mathbf{r}-\mathbf{R}_k|}, \tag{3.20}$$

gde $Z_k|e|$ uključuje naelektrisanje jezgra i elektrona čiji se efekat modeluje korišćenjem pseudopotencijala, dok V^k_{pp} modeluje sve ostale efekte nevalentnih elektrona ne uračunavajući standardni elektrostatički efekat. Dakle, za izračunavanje elemenata matrice datih u jednačini (3.9), potrebno je rešiti integral privlačenja jezgra:

$$N_{k\mu,l\nu} = \int \mathrm{d}^3\mathbf{r}\, \varphi_{k\mu}(\mathbf{r}-\mathbf{R}_k) \frac{1}{|\mathbf{r}-\mathbf{R}_m|} \varphi_{l\nu}(\mathbf{r}-\mathbf{R}_l), \tag{3.21}$$

i integral pseudopotencijala

$$P_{k\mu,l\nu} = \int \mathrm{d}^3\mathbf{r}\, \varphi_{k\mu}(\mathbf{r}-\mathbf{R}_k)\, V^m_{pp}\, \varphi_{l\nu}(\mathbf{r}-\mathbf{R}_l). \tag{3.22}$$

Integral jezgra rešavamo direktno koristeći analitičke formule [89, 88].

Kada bazisne funkcije iz jednačine (3.9) zamenimo sumama kontrahovanih Gausijana iz jednačine (3.3), dobijamo sledeći izraz:

$$H^{(2)}_{k\mu,l\nu} = \int \mathrm{d}^3\mathbf{r} \sum_i a_i \chi_{k\mu_i}(\mathbf{r}-\mathbf{R_k})\; V_{ion} \sum_j a_j \chi_{l\nu_j}(\mathbf{r}-\mathbf{R_l}). \tag{3.23}$$

Kao i u slučaju integrala preklapanja i kinetičkog integrala, za primitivne Gausijane uvodimo izraze $\chi_A(\mathbf{r})$ (3.13) i $\chi_B(\mathbf{r})$ (3.14). Sada se iz jednačine (3.23) može videti da je za računanje matrice $H^{(2)}_{k\mu,l\nu}$ potrebno izračunati integral privlačenja jezgra:

$$\langle \mathbf{A}| - \frac{Z_c}{|\vec{\mathbf{r}}-\vec{\mathbf{r}}_c|}|\mathbf{B}\rangle = -Z_c \int_{-\infty}^{\infty} d^3\mathbf{r} \frac{\chi_{\mathbf{A}}(\vec{\mathbf{r}})\chi_{\mathbf{B}}(\vec{\mathbf{r}})}{|\vec{\mathbf{r}}-\vec{\mathbf{r}}_c|}. \tag{3.24}$$

Analitičko rešenje ovog integrala je sledeće:

$$\begin{aligned}
&\langle \mathbf{A}| - \frac{Z_c}{|\vec{\mathbf{r}}-\vec{\mathbf{r}}_c|}|\mathbf{B}\rangle = \\
&-\frac{Z_c N_1 N_2 \pi}{\gamma_p} e^{-\eta_p(\mathbf{A}-\mathbf{B})^2} \sum_{i_1,i_2,o} \mathbf{A}_x \sum_{j_1,j_2,p} \mathbf{A}_y \sum_{k_1,k_2,q} \mathbf{A}_z \cdot \\
&2F_\nu(\gamma_p(\mathbf{P}-\vec{\mathbf{r}}_c)^2),
\end{aligned} \tag{3.25}$$

gde su $\gamma_p = \alpha_1+\alpha_2, \eta_p = \frac{\alpha_1\alpha_2}{\gamma_P}$ i $\mathbf{P} = \frac{1}{\gamma_p}(\alpha_1\mathbf{A}+\alpha_2\mathbf{B})$. Prva suma iz prethodnog izraza jednaka je

$$\begin{aligned}
&\sum_{i_1,i_2,o} \mathcal{A}_x = (-1)^{l_1+l_2} l_1! l_2! \cdot \\
&\sum_{i_1}\sum_{i_2}\sum_{o_1}\sum_{o_2}\sum_{r} \frac{(-1)^{o_2+r}(o_1+o_2)!}{4^{i_1+i_2+r} i_1! i_2! o_1! o_2! r!} \cdot \\
&\frac{\alpha_1^{o_2-i_1-r}\alpha_2^{o_1-i_2-r}(A_x-B_x)^{o_1+o_2-2r}}{(l_1-2i_1-o_1)!(l_2-2i_2-o_2)!(o_1+o_2-2r)!} \cdot \\
&\sum_u \frac{(-1)^u \mu_x! (P_x-r_{cx})^{\mu_x-2u}}{4^u u!(\mu_x-2u)!\gamma_p^{o_1+o_2-r+u}},
\end{aligned} \tag{3.26}$$

pri čemu su opsezi sumiranja:

$$\begin{aligned}
&i_1 = 0 \to \left[\frac{1}{2}l_1\right], i_2 = 0 \to \left[\frac{1}{2}l_2\right], \\
&o_1 = 0 \to l_1 - 2i_1, o_2 = 0 \to l_2 - 2i_2, \\
&r = 0 \to \left[\frac{1}{2}(o_1+o_2)\right], \\
&u = 0 \to \left[\frac{1}{2}\mu_x\right], v = 0 \to \left[\frac{1}{2}\mu_y\right], w = 0 \to \left[\frac{1}{2}\mu_z\right],
\end{aligned} \tag{3.27}$$

dok je

$$\begin{aligned}
\mu_x &= l_1 + l_2 - 2(i_1+i_2) - (o_1+o_2), \\
\mu_y &= m_1 + m_2 - 2(j_1+j_2) - (p_1+p_2), \\
\mu_z &= n_1 + n_2 - 2(k_1+k_2) - (q_1+q_2), \\
\nu &= \mu_x + \mu_y + \mu_z - (u+v+w).
\end{aligned} \tag{3.28}$$

Sume $\mathcal{A}_y$ i $\mathcal{A}_z$ su date analognim formulama, pa će njihov opis biti izostavljen.

Za računanje Bojsove funkcije (Boys function) $F_\nu(u)$ postoji više različitih metoda. Izbor odgovarajuće metode zavisi od potrebnog opsega za u i ν, kao i od potrebnog nivoa preciznosti.

Nakon parcijalne integracije osnovni izraz Bojsove funkcije dobija sledeći oblik [88]:

$$F_\nu(u) = \int_0^1 t^{2\nu} e^{-ut^2} dt = \\ \frac{(2\nu)!}{2\nu!}\left[\frac{\sqrt{\pi}}{4^\nu u^{\nu+1/2}}\mathrm{erf}\sqrt{u} - e^{-u}\sum_{k=0}^{\nu-1}\frac{(\nu-k)!}{4^k(2\nu-2k)!u^{k+1}}\right]. \tag{3.29}$$

$F_\nu(u)$ zadovoljava sledeću rekurzivnu relaciju [88]

$$F_\nu(u) = \frac{(2\nu-1)F_{\nu-1}(u) - e^{-u}}{2u}. \tag{3.30}$$

Bojsovu funkciju je takođe moguće predstaviti sledećim izrazom [90]:

$$F_\nu(u) = e^{-u}\lim_{N\to\infty}\sum_{i=0}^{N}\frac{(2u)^i}{\prod_{j=0}^{i}(2\nu+2j+1)}. \tag{3.31}$$

Nakon testiranja implementacija različitih načina rešavanja Bojsove funkcije [(3.29), (3.30), (3.31)] je utvrđeno da je najoptimalniji način korišćenja takav da se izraz (3.31) koristi samo u slučajevima gde u ima male vrednosti (za graničnu vrednost je odabrano 0.001), dok se za sve ostale slučajeve korištena implementacija izraza (3.29). Rekurzivni izraz (3.30) je zbog loše efikasnosti izvršavanja korišten samo za proveru rezultata i određivanje potrebnog faktora preciznosti za računanje (3.31).

Za rešavanje integrala pseudopotencijala, koristimo pristup iz [91]. U numeričkom primeru iz sekcije 5 primenjujemo BKJC VDZ ECP pseudopotencijal [86, 87, 53, 54].

3.1.4 Integral Hartri potencijala

Za izračunavanje doprinosa integrala Hartri potencijala (Hartree Potencial Integral) [jednačina (3.10)] su primenjena dva pristupa, koja se razlikuju po načinu izračunavanja gustine naelektrisanja $\rho(\mathbf{r})$.

Jedan od načina da se izračuna Hartrijev integral je da se gustina naelektrisanja predstavi uz pomoć jednačine (2.2)

$$\begin{aligned}
&\int \mathrm{d}^3\mathbf{r}\,\varphi_{k\mu}(\mathbf{r})V_H\left[\rho\right]\varphi_{l\nu}(\mathbf{r}) \\
&= \int \mathrm{d}^3\mathbf{r}\,\varphi_{k\mu}(\mathbf{r})\int\frac{(-1)\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}\mathrm{d}^3\mathbf{r}'\,\varphi_{l\nu}(\mathbf{r}) \\
&= \int \mathrm{d}^3\mathbf{r}\,\mathrm{d}^3\mathbf{r}'\sum_{i=1}^{N/2}\varphi_{k\mu}(\mathbf{r})\varphi_{l\nu}(\mathbf{r})\frac{2|\psi_i(\mathbf{r}')|^2}{|\mathbf{r}-\mathbf{r}'|} \\
&= \int \mathrm{d}^3\mathbf{r}\,\mathrm{d}^3\mathbf{r}'\sum_{i=1}^{N/2}\sum_{m\alpha}\sum_{n\beta}\varphi_{k\mu}(\mathbf{r})\varphi_{l\nu}(\mathbf{r})\frac{2\alpha_{m\alpha}^{(i)}\alpha_{n\beta}^{(i)}}{|\mathbf{r}-\mathbf{r}'|}\varphi_{m\alpha}(\mathbf{r}')\varphi_{n\beta}(\mathbf{r}') \\
&= 2\sum_{i=1}^{N/2}\sum_{m\alpha}\sum_{n\beta}\alpha_{m\alpha}^{(i)}\alpha_{n\beta}^{(i)}\int \mathrm{d}^3\mathbf{r}\,\mathrm{d}^3\mathbf{r}'\,\varphi_{k\mu}(\mathbf{r})\varphi_{l\nu}(\mathbf{r})\frac{1}{|\mathbf{r}-\mathbf{r}'|}\varphi_{m\alpha}(\mathbf{r}')\varphi_{n\beta}(\mathbf{r}').
\end{aligned} \tag{3.32}$$

Sada se računanje svodi na rešavanje Kulonovog integrala centriranog na četiri Gausijana:

$$(ab|\,cd) = \int \mathrm{d}^3\mathbf{r_1}\,\mathrm{d}^3\mathbf{r_2}\,\varphi_a(\mathbf{r_1})\,\varphi_b(\mathbf{r_1})\,\frac{1}{|\mathbf{r}_1-\mathbf{r}_2|}\varphi_c(\mathbf{r_2})\,\varphi_d(\mathbf{r_2}). \tag{3.33}$$

Radi lakšeg prikaza, u prethodnom izrazu je uvedena notacija $\varphi_a(\mathbf{r}) \equiv \varphi_{k_a\mu_a}(\mathbf{r} - \mathbf{R}_{k_a})$. Analogno uvodimo i notacije za $\varphi_b(\mathbf{r})$, $\varphi_c(\mathbf{r})$ i $\varphi_d(\mathbf{r})$.

U drugom načinu rešavanja predstavljamo gustinu naelektrisanja $\rho(\mathbf{r})$ u bazisu funkcija Gausijana. Da bismo ovo postigli, odabrali smo skup tačaka u prostoru $\{\mathbf{r}_\alpha\}$ i odgovarajuće težinske funkcije w_α. Zatim smo za svako α uz pomoć jednačine (2.4) izračunali vrednosti $\rho(\mathbf{r}_\alpha)$. Da bismo dobili koeficijente $\beta_{k\mu}$ iz razvoja u bazisu Gausijana

$$\rho_f(\mathbf{r}) = \sum_{k\mu} \beta_{k\mu} \varphi^C_{k\mu}(\mathbf{r} - \mathbf{R}_k), \tag{3.34}$$

potrebno je minimizirati izraz

$$D = \sum_\alpha w_\alpha \left[\rho(\mathbf{r}_\alpha) - \rho_f(\mathbf{r}_\alpha)\right]^2, \tag{3.35}$$

uz ograničenje da je integral gustine naelektrisanja jednak ukupnom naelektrisanju valentnih elektrona Q:

$$\int \mathrm{d}^3\mathbf{r}\, \rho_f(\mathbf{r}) = Q. \tag{3.36}$$

Dalje, uz pomoć prethodnog ograničenja i metode Lagranžovih množilaca, vršimo minimizaciju sledećeg funkcionala:

$$\mathcal{F}(\{\beta_{k\mu}\}, \lambda) = \sum_\alpha w_\alpha \left[\rho(\mathbf{r}_\alpha) - \rho_f(\mathbf{r}_\alpha)\right]^2 - \lambda \left[\sum_{k\mu} \beta_{k\mu} \int \mathrm{d}^3\mathbf{r}\, \varphi^C_{k\mu}(\mathbf{r} - \mathbf{R}_k) - Q\right]. \tag{3.37}$$

Koristeći uslove $\frac{\partial \mathcal{F}}{\partial \beta_{k\mu}} = 0$ i $\frac{\partial \mathcal{F}}{\partial \lambda} = 0$ dobijamo sledeći sistem jednačina (gde $\beta_{k\mu}$ i λ predstavljaju nepoznate):

$$\sum_{k\mu} q_{k\mu}\beta_{k\mu} = Q, \tag{3.38}$$

$$\sum_{l\nu} A_{k\mu,l\nu}\beta_{l\nu} - q_{k\mu}\lambda = B_{k\mu}, \tag{3.39}$$

gde je

$$q_{k\mu} = \int \mathrm{d}^3\mathbf{r} \varphi^C_{k\mu}(\mathbf{r} - \mathbf{R}_k), \tag{3.40}$$

$$A_{k\mu,l\nu} = 2\sum_\alpha w_\alpha \varphi^C_{k\mu}(\mathbf{r}_\alpha - \mathbf{R}_k)\, \varphi^C_{l\nu}(\mathbf{r}_\alpha - \mathbf{R}_l), \tag{3.41}$$

i

$$B_{k\mu} = 2\sum_\alpha w_\alpha \rho(\mathbf{r}_\alpha)\, \varphi^C_{k\mu}(\mathbf{r}_\alpha - \mathbf{R}_k). \tag{3.42}$$

Rešenje sistema jednačina (3.38)-(3.39) daje $\beta_{k\mu}$ koeficijente [jednačina (3.34)], odnosno reprezentaciju gustine naelektrisanja u bazisu Gausijana. Ovde koristimo isti skup tačkaka u prostoru $\{\mathbf{r}_\alpha\}$ i odgovarajuće težinske funkcije w_α kao i za izračunavanje izmensko-korelacionog integrala (pogledati sekciju 3.1.5).

Sledeće što ćemo razmotriti je implementacija za izračunavanje Hartri integrala uz pomoć bazisa Gausijana. Najpre je potrebno da predstavimo elektronsku gustinu naelektrisanju u obliku bazisa Gausijana iz jednačine (3.34):

$$\begin{aligned}
&\int \mathrm{d}^3\mathbf{r}\, \varphi_{k\mu}(\mathbf{r}) V_H\left[\rho\right] \varphi_{l\nu}(\mathbf{r}) \\
&= \int \mathrm{d}^3\mathbf{r}\, \varphi_{k\mu}(\mathbf{r}) \int \frac{(-1)\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} \mathrm{d}^3\mathbf{r}'\, \varphi_{l\nu}(\mathbf{r}) \\
&= \sum_{k'=1}^{M} \sum_{\mu'=1}^{L_{k'}} C_{k'\mu'} \int \mathrm{d}^3\mathbf{r}\, \mathrm{d}^3\mathbf{r}'\, \varphi_{k\mu}(\mathbf{r}) \varphi_{l\nu}(\mathbf{r}) \frac{1}{|\mathbf{r}-\mathbf{r}'|} \varphi^C_{k'\mu'}.
\end{aligned} \tag{3.43}$$

Da bismo dobili sve elemente matrice iz jednačine (3.10), neophodno je izračunati integrale iz jednačine (3.43), koje možemo zapisati u sledećem obliku:

$$(ab|\ c) = \int \mathrm{d}^3\mathbf{r_1}\, \mathrm{d}^3\mathbf{r_2}\, \varphi_a\left(\mathbf{r_1}\right) \varphi_b\left(\mathbf{r_1}\right) \frac{1}{|\mathbf{r}_1-\mathbf{r}_2|} \varphi_c^C\left(\mathbf{r_2}\right), \tag{3.44}$$

gde su

$$\begin{aligned}
\varphi_a\left(\mathbf{r}\right) &= \sum_\alpha D_{a\alpha} \phi_{a\alpha}\left(\mathbf{r}\right), \\
\varphi_b\left(\mathbf{r}\right) &= \sum_\beta D_{b\beta} \phi_{b\beta}\left(\mathbf{r}\right),
\end{aligned} \tag{3.45}$$

kontrahovani Gausijani koji se koriste kod talasnih funkcija, dok je

$$\varphi_c^C\left(\mathbf{r}\right) = \sum_\gamma D^C_{c\gamma} \phi_{c\gamma}\left(\mathbf{r}\right), \tag{3.46}$$

kontrahovani Gausijan koji se koristi kod gustine naelektrisanja. Radi lakšeg prikaza, u prethodnim jednačinama koristimo notaciju $\varphi_a\left(\mathbf{r}\right) \equiv \varphi_{k_a\mu_a}\left(\mathbf{r}-\mathbf{R}_{k_a}\right)$. Analogno uvodimo i notacije za $\varphi_b\left(\mathbf{r}\right)$ i $\varphi_c^C\left(\mathbf{r}\right)$.

Integral dat u jednačini (3.44) predstavlja specijalni slučaj Kulonovog integrala centriranog na četiri Gausijana za slučaj kada neki od Gausijana ima konstantnu vrednost. Ovaj integral se često javlja pri računanju elektronske strukture materijala. Jedan od načina da se izračuna Kulonov integral je analitički. Primer ovakvog načina računanja će biti predstavljen u nastavku [89, 88].

Analitičko rešenje

Hartrijev integral sada definišemo kao:

$$\begin{aligned}
&\langle \mathbf{A},\mathbf{C}| \frac{1}{r_{12}} |\mathbf{B},\mathbf{D}\rangle = \\
&\int_{-\infty}^{\infty} d^3\mathbf{r}_1 \int_{-\infty}^{\infty} d^3\mathbf{r}_2 \frac{\chi_{\mathbf{A}}(\mathbf{r}_1)\chi_{\mathbf{B}}(\mathbf{r}_1)\chi_{\mathbf{C}}(\mathbf{r}_2)\chi_{\mathbf{D}}(\mathbf{r}_2)}{|\mathbf{r}_1-\mathbf{r}_2|}.
\end{aligned} \tag{3.47}$$

Analitičko rešenje integrala je

$$\begin{aligned}
&\langle \mathbf{A},\mathbf{C}| \frac{1}{r_{12}} |\mathbf{B},\mathbf{D}\rangle = -\frac{N_1N_2N_3N_4\pi^{5/2}}{\gamma_p\gamma_q\sqrt{\gamma_p+\gamma_q}} \cdot \\
&e^{-\eta_p(\mathbf{A}-\mathbf{B})^2} e^{-\eta_q(\mathbf{C}-\mathbf{D})^2} \sum_{\substack{i_1,i_2,i_3,i_4,\\ o_1,o_2,o_3,o_4,\\ r_1,r_2,u}} \mathcal{J}_x \sum_{\substack{j_1,j_2,j_3,j_4,\\ p_1,p_2,p_3,p_4,\\ s_1,s_2,v}} \mathcal{J}_y \cdot \\
&\sum_{\substack{k_1,k_2,k_3,k_4,\\ q_1,q_2,q_3,q_4,\\ t_1,t_2,w}} \mathcal{J}_z 2F_\nu(\eta(\mathbf{P}-\mathbf{Q})^2),
\end{aligned} \tag{3.48}$$

gde je

$$\begin{aligned}
&\gamma_p = \alpha_1 + \alpha_2, \gamma_q = \alpha_3 + \alpha_4, \\
&\eta_p = \frac{\alpha_1\alpha_2}{\gamma_p}, \eta_q = \frac{\alpha_3\alpha_4}{\gamma_q}, \eta = \frac{\gamma_p\gamma_q}{\gamma_p + \gamma_q}, \\
&\mathbf{P} = \frac{\alpha_1\mathbf{A} + \alpha_2\mathbf{B}}{\gamma_p}, \mathbf{Q} = \frac{\alpha_3\mathbf{C} + \alpha_4\mathbf{D}}{\gamma_p}.
\end{aligned} \tag{3.49}$$

Prva od tri sume ima sledeći oblik:

$$\begin{aligned}
&\sum_{\substack{i_1,i_2,i_3,i_4,\\ o_1,o_2,o_3,o_4,\\ r_1,r_2,u}} \mathcal{J}_x = \frac{(-1)^{l_1+l_2} l_1! l_2!}{\gamma_p^{l_1+l_2}} \\
&\sum_{i_1}\sum_{i_2}\sum_{o_1}\sum_{o_2}\sum_{r_1} \frac{(-1)^{o_2+r_1}(o_1+o_2)!}{4^{i_1+i_2+r_1} i_1! i_2! o_1! o_2! r_1!} \cdot \\
&\frac{\alpha_1^{o_2-i_1-r_1}\alpha_2^{o_1-i_2-r_1}\gamma_p^{2(i_1+i_2)+r_1}(A_x - B_x)^{o_1+o_2-2r_1}}{(l_1-2i_1-o_1)!(l_2-2i_2-o_2)!(o_1+o_2-2r_1)!} \cdot \\
&\frac{l_3! l_4!}{\gamma_q^{l_3+l_4}} \sum_{i_3}\sum_{i_4}\sum_{o_3}\sum_{o_4}\sum_{r_2} \frac{(-1)^{o_3+r_2}(o_3+o_4)!}{4^{i_3+i_4+r_2} i_3! i_4! o_3! o_4! r_2!} \cdot \\
&\frac{\alpha_3^{o_4-i_3-r_2}\alpha_4^{o_3-i_4-r_2}\gamma_p^{2(i_3+i_4)+r_2}(C_x - D_x)^{o_3+o_4-2r_2}}{(l_3-2i_3-o_3)!(l_4-2i_4-o_4)!(o_3+o_4-2r_2)!} \cdot \\
&\sum_u \frac{(-1)^u \mu_x! \eta^{\mu_x-u}(P_x - Q_x)^{\mu_x-2u}}{4^u u!(\mu_x-2u)!},
\end{aligned} \tag{3.50}$$

$$\begin{aligned}
\mu_x =& l_1 + l_2 + l_3 + l_4 - 2(i_1 + i_2 + i_3 + i_4) - (o_1 + o_2 + o_3 + o_4), \\
\mu_y =& m_1 + m_2 + m_3 + m_4 - 2(j_1 + j_2 + j_3 + j_4) - (p_1 + p_2 + p_3 + p_4), \\
\mu_z =& n_1 + n_2 + n_3 + n_4 - 2(k_1 + k_2 + k_3 + k_4) - (q_1 + q_2 + q_3 + q_4), \\
\nu =& \mu_x + \mu_y + \mu_z - (u + v + w).
\end{aligned} \tag{3.51}$$

U ovoj implementaciji ponovo koristimo Bojsovu funkciju definisanu u ((3.29)).

Skup opsega prethodne sume je u nastavku:

$$\begin{aligned}
&i_1 = 0 \to \left[\frac{1}{2}l_1\right], i_2 = 0 \to \left[\frac{1}{2}l_2\right], \\
&i_3 = 0 \to \left[\frac{1}{2}l_3\right], i_4 = 0 \to \left[\frac{1}{2}l_4\right], \\
&o_1 = 0 \to l_1 - 2i_1, o_2 = 0 \to l_2 - 2i_2, \\
&o_3 = 0 \to l_3 - 2i_3, o_4 = 0 \to l_4 - 2i_4, \\
&r_1 = 0 \to \left[\frac{1}{2}(o_1 + o_2)\right], r_2 = 0 \to \left[\frac{1}{2}(o_3 + o_4)\right], \\
&u = 0 \to \left[\frac{1}{2}\mu_x\right], v = 0 \to \left[\frac{1}{2}\mu_y\right], w = 0 \to \left[\frac{1}{2}\mu_z\right].
\end{aligned} \tag{3.52}$$

Ugnježdene sume $\mathcal{J}_y$ i $\mathcal{J}_z$ se mogu definisati analogno.

Da bismo izračunali sve integrale date u jednačini (3.44), moguće je primeniti analitičku formulu i uz pomoću nje izračunati svaki integral pojedinačno.

Rekurentne relacije

S obzirom na relativno kompleksnu formu analitičkog izraza, implementacija prethodnog rešenja se nije pokazala kao zadovoljavajuća. Postoje elegantnija rešenja koja se baziraju na korišćenju rekurentnih relacija između integrala [92]. Efikasan način za rešavanje integrala centriranog na četiri Gausijana je opisan u [93]. U nastavku ćemo pokazati šemu za slučaj integrala centriranog na tri Gausijana, baziranoj na ideji korišćenoj u [93].

Želimo da izračunamo integral $(ab|c)$, dat u jednačini (3.44).

Takođe uvodimo notaciju za integral nad primitivnim Gausijanom:

$$[ab|\ c] = \int \mathrm{d}^3\mathbf{r_1}\, \mathrm{d}^3\mathbf{r_2}\, \phi_{a\alpha}\left(\mathbf{r_1}\right) \phi_{b\beta}\left(\mathbf{r_1}\right) \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \phi_{c\gamma}\left(\mathbf{r_2}\right), \tag{3.53}$$

gde koristimo skraćenu notaciju za primitivni nenormalizovani Gausijan centriran na $\mathbf{A}$

$$\phi_{a\alpha}\left(\mathbf{r}\right) = (x - A_x)^{a_x} (y - A_y)^{a_y} (z - A_z)^{a_z} e^{-\alpha(\mathbf{r}-\mathbf{A})^2}. \tag{3.54}$$

Notacije za $\phi_{b\beta}$ i $\phi_{c\gamma}$ uvodimo analogno.

Integral (3.53) predstavlja specijalni slučaj integrala:

$$[ab|\ cd] = \int \mathrm{d}^3\mathbf{r_1}\, \mathrm{d}^3\mathbf{r_2}\, \phi_{a\alpha}\left(\mathbf{r_1}\right) \phi_{b\beta}\left(\mathbf{r_1}\right) \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \phi_{c\gamma}\left(\mathbf{r_2}\right) \phi_{d\delta}\left(\mathbf{r_2}\right), \tag{3.55}$$

gde je $\delta = 0$ i $d_x = d_y = d_z = 0$.

Zato se mogu iskoristiti rekurentne relacije iz [92] za integral $[ab|\ cd]$ koje glase:

$$\begin{aligned}
[(a+1_i)\, b|\ cd]^{(m)} = {} & (P_i - A_i)\, [ab|\ cd]^{(m)} + (W_i - P_i)\, [ab|\ cd]^{(m+1)} + \\
& \frac{a_i}{2\xi}\left([(a-1_i)\, b|\ cd]^{(m)} - \frac{\zeta}{\xi+\zeta}\, [(a-1_i)\, b|\ cd]^{(m+1)} \right) + \\
& \frac{b_i}{2\xi}\left([a\,(b-1_i)|\ cd]^{(m)} - \frac{\zeta}{\xi+\zeta}\, [a\,(b-1_i)|\ cd]^{(m+1)} \right) + \\
& \frac{c_i}{2\,(\xi+\zeta)}\, [ab|\ (c-1_i)d]^{(m+1)} + \frac{c_i}{2\,(\xi+\zeta)}\, [ab|\ c(d-1_i)]^{(m+1)},
\end{aligned} \tag{3.56}$$

gde je $[ab|\ cd]^{(0)} = [ab|\ cd]$, a $[ab|\ cd]^{(m)}$ su pomoćne veličine. Simboli u jednačini (3.56) imaju sledeće značenje: $\xi = \alpha + \beta$, $\zeta = \gamma + \delta$, $P = \frac{\alpha\mathbf{A}+\beta\mathbf{B}}{\alpha+\beta}$, $Q = \frac{\gamma\mathbf{C}+\delta\mathbf{D}}{\gamma+\delta}$, $W = \frac{\xi\mathbf{P}+\zeta\mathbf{C}}{\xi+\zeta}$ a notacija $(a \pm 1_i)$ se odnosi na Gausijan gde je eksponent a_i uvećan/smanjen za 1. Takođe važi da je $[ab|\ c]^{(0)} = [ab|\ c]$. Druga rekurentna relacija glasi:

$$(a\,(b+1_i)|\ cd) = ((a+1_i)\, b|\ cd) + (A_i - B_i)\, (ab|\ cd). \tag{3.57}$$

Na osnovu prethodnih jednačina, proces rešavanja integrala centriranog na tri Gausijana je baziran na korišćenju sledećih rekurentnih relacija:

$$\begin{aligned}
[(a+1_i)\, b|\ c]^{(m)} = {} & (P_i - A_i)\, [ab|\ c]^{(m)} + (W_i - P_i)\, [ab|\ c]^{(m+1)} + \\
& \frac{a_i}{2\xi}\left([(a-1_i)\, b|\ c]^{(m)} - \frac{\zeta}{\xi+\zeta}\, [(a-1_i)\, b|\ c]^{(m+1)} \right) + \\
& \frac{b_i}{2\xi}\left([a\,(b-1_i)|\ c]^{(m)} - \frac{\zeta}{\xi+\zeta}\, [a\,(b-1_i)|\ c]^{(m+1)} \right) + \\
& \frac{c_i}{2\,(\xi+\zeta)}\, [ab|\ c-1_i]^{(m+1)}
\end{aligned} \tag{3.58}$$

i

$$(a\,(b+1_i)|\ c) = ((a+1_i)\, b|\ c) + (A_i - B_i)\, (ab|\ c). \tag{3.59}$$

Simboli u jednačinama (3.58)-(3.59) imaju sledeće značenje: $[ab|\ c]^{(m)}$ je pomoćni integral definisan kao

$$[ab|\ c]^{(m)} = \frac{2}{\sqrt{\pi}} \int_0^\infty \mathrm{d}u \left(\frac{u^2}{\rho+u^2}\right)^m \int \mathrm{d}^3\mathbf{r}_1\, \mathrm{d}^3\mathbf{r}_2\, \phi_a(\mathbf{r}_1)\, \phi_b(\mathbf{r}_1)\, e^{-u^2(\mathbf{r}_1-\mathbf{r}_2)^2} \phi_c(\mathbf{r}_2), \tag{3.60}$$

$\rho = \frac{(\alpha+\beta)\gamma}{\alpha+\beta+\gamma}$, $\xi = \alpha$, $\zeta = \gamma$, $P = \frac{\alpha\mathbf{A}+\beta\mathbf{B}}{\alpha+\beta}$, $W = \frac{\xi\mathbf{P}+\zeta\mathbf{C}}{\xi+\zeta}$ a notacija $(a \pm 1_i)$ se odnosi na Gausijan gde je eksponent a_i uvećan/smanjen za 1. Takođe važi da je $[ab|\ c]^{(0)} = [ab|\ c]$. Za jednačine (3.58) i (3.59) ćemo dalje redom koristiti izraze prva i druga rekurentna relacija, dok će se pritom podrazumevati da se radi o rekurentnim relacijama na tri Gausijana.

Evidentno je da se višestrukom primenom relacije (3.59) računanje integrala $(ab|\ c)$ svodi na računanje integrala $(a0_b|\ c)$ (simbol 0_b označava da je $b_x = b_y = b_z = 0$). Da bismo izračunali $(a0_b|\ c)$ najpre je potrebno izračunati $[a0_b|\ c]^{(0)}$, što se dalje korišćenjem jednačine (3.58), svodi na računanje integrala $[0_a0_b|\ 0_c]^{(m)}$. Poslednji integral je [93]

$$[0_a0_b|\ 0_c]^{(m)} = \frac{1}{\sqrt{\xi+\gamma}} K_{AB} K_C F_m(T), \tag{3.61}$$

gde je

$$\begin{aligned} T &= \frac{\xi\gamma}{\xi+\gamma}(\mathbf{P}-\mathbf{C})^2, \\ F_m(T) &= \int_0^1 t^{2m} e^{-Tt^2}\, \mathrm{d}t, \\ K_{AB} &= \frac{\sqrt{2}\pi^{5/4}}{\alpha+\beta} e^{-\frac{\alpha\beta}{\alpha+\beta}(\mathbf{A}-\mathbf{B})^2}. \\ K_C &= \frac{\sqrt{2}\pi^{5/4}}{\gamma}. \end{aligned} \tag{3.62}$$

Pokažimo sada kako izgleda proces računanja integrala uz pomoć navedenih rekurentnih relacija. Ukoliko želimo da izračunamo $(ab|c)$, na osnovu relacije (3.59) su nam potrebni $((a+1_x)(b-1_x)|c)$, $((a+1_y)(b-1_y)|c)$, $((a+1_z)(b-1_z)|c)$, $(a(b-1_x)|c)$, $(a(b-1_y)|c)$ i $(a(b-1_z)|c)$, za šta uvodimo skraćene izraze:

$$\begin{aligned} &((a+1)(b-1)|c), \\ &(a(b-1)|c). \end{aligned} \tag{3.63}$$

Da bismo njih izračunali, potrebni su nam $((a+2)(b-2)|c)$, $((a+1)(b-2)|c)$ i $(a(b-2)|c)$. Ovaj postupak ponavljamo sve dok drugi član ne postane 0, odnosno sve dok ne dobijemo niz izraza

$$((a+b)0|c), ((a+b-1)0|c), ..., (a0|c). \tag{3.64}$$

Svaki od ovih članova oblika $(a0|c)$ je u relaciji sa $[a0|c]$:

$$(a0|c) = \sum_{\alpha\beta\gamma} D_{a\alpha} D_{b\beta} D_{c\gamma} [a_\alpha 0_\beta | c_\gamma]. \tag{3.65}$$

Da bismo dalje izračunali $[a0|c]$, koristićemo rekurentnu relaciju (3.58), sve dok ne svedemo integral na izraz $[00|0]^m$, koji se računa uz pomoć jednačine (3.61).

Relacija (3.58) u našem slučaju glasi:

$$\begin{aligned} [a0|\ c]^{(m)} = {} & (P_i - A_i)\,[(a-1_i)0|\ c]^{(m)} + (W_i - P_i)\,[(a-1_i)0|\ c]^{(m+1)} + \\ & \frac{a_i - 1}{2\xi}\left([(a-2_i)\,0|\ c]^{(m)} - \frac{\zeta}{\xi+\zeta}\,[(a-2_i)\,0|\ c]^{(m+1)}\right) + \\ & \frac{c_i}{2(\xi+\zeta)}\,[(a-1_i)0|\ c-1_i]^{(m+1)}. \end{aligned} \tag{3.66}$$

Isti izraz možemo zapisati i kao

$$[a0|\ c]^{(m)} = [a0_b|\ c0_d]^{(m)}\,, \tag{3.67}$$

gde je

$$\begin{aligned}
a\ &\Rightarrow \phi_{a\alpha}(\mathbf{r}) = (x-A_x)^{a_x}(y-A_y)^{a_y}(z-A_z)^{a_z}\, e^{-\alpha(\mathbf{r}-\mathbf{A})^2},\\
0_b\ &\Rightarrow \phi_b(\mathbf{r}) = e^{-\beta(\mathbf{r}-B)^2},\\
c\ &\Rightarrow \phi_{c\gamma}(\mathbf{r}) = (x-C_x)^{c_x}(y-C_y)^{c_y}(z-C_z)^{c_z}\, e^{-\alpha(\mathbf{r}-\mathbf{A})^2},\\
0_d\ &\Rightarrow \phi_d(\mathbf{r}) = 1,
\end{aligned} \tag{3.68}$$

iz čega sledi da je

$$\begin{aligned}
[a0_b|\ c0_d]^{(m)} =& [c0_d|\ a0_b]^{(m)} =\\
&(Q_i - C_i)\,[(c-1_i)0_d|\ a0_b]^{(m)} + (W_i - Q_i)\,[(c-1_i)0_d|\ a0_b]^{(m+1)} +\\
&\frac{c_i-1}{2\xi}\left([(c-2_i)\,0_d|\ a0_b]^{(m)} - \frac{\zeta}{\xi+\zeta}\,[(c-2_i)\,0_d|\ a0_b]^{(m+1)}\right) +\\
&\frac{a_i}{2(\xi+\zeta)}\,[(c-1_i)0_d|\ (a-1_i)0_b]^{(m+1)} =\\
&(W_i - Q_i)\,[a0_b|\ (c-1_i)0_d]^{(m+1)} +\\
&\frac{c_i-1}{2\xi}\left([a0_b|\ (c-2_i)\,0_d]^{(m)} - \frac{\zeta}{\xi+\zeta}\,[a0_b|\ (c-2_i)\,0_d]^{(m+1)}\right) +\\
&\frac{a_i}{2(\xi+\zeta)}\,[(a-1_i)0_b|\ (c-1_i)0_d]^{(m+1)}\,.
\end{aligned} \tag{3.69}$$

Član $(Q_i - C_i)\,[(c-1_i)0_d|\ a0_b]^{(m)}$ iz prethodne jednačine se neutrališe jer važi da je za $\delta = 0$, $Q = \frac{\gamma C + \delta D}{\gamma\delta} = C$.

Dalje se izraz $[a0|\ c]^{(m)}$ takođe može napisati u sledećem obliku:

$$\begin{aligned}
[a0|\ c]^{(m)} =& (W_i - Q_i)\,[a0_b|\ c-1_i]^{(m+1)} +\\
&\frac{c_i-1}{2\xi}\left([a0_b|\ c-2_i]^{(m)} - \frac{\zeta}{\xi+\zeta}\,[a0_b|\ c-2_i]^{(m+1)}\right) +\\
&\frac{a_i}{2(\xi+\zeta)}\,[(a-1_i)0_b|\ c-1_i]^{(m+1)}\,.
\end{aligned} \tag{3.70}$$

Dakle, da bi se izračunao $[a0|\ c]^{(m)}$ potrebno je izračunati jedan od sledeća dva skupa jednačina:

$$[(a-1_i)0|\ c]^{(m)}\,,\ [(a-1_i)0|\ c]^{(m+1)}\,,\ [(a-2_i)0|\ c]^{(m)}\,,\ [(a-2_i)0|\ c]^{(m+1)}\,,\ [(a-1_i)0|\ c-1_i]^{(m+1)} \tag{3.71}$$

$$[(c-1_i)0|\ a]^{(m)}\,,\ [(c-1_i)0|\ a]^{(m+1)}\,,\ [(c-2_i)0|\ a]^{(m)}\,,\ [(c-2_i)0|\ a]^{(m+1)}\,,\ [(c-1_i)0|\ a-1_i]^{(m+1)} \tag{3.72}$$

Da li će za dati integral biti upotrebljena jednačina (3.66) ili (3.70) je moguće odrediti po sledećem kriterijumu:

a) Bira se jednačina koja sadrži veći broj članova iz skupa onih koji su već u listi članova za računanje.

b) Ukoliko a) daje isti broj članova, bira se jednačina koja ima više nula članova.

Posle ovog koraka se konačno svi integrali svode na jednačinu (3.61):

$$[00|\ 0]^{(m)}\ \text{ gde je } m \in \{0, ..., a_x + a_y + a_z + b_x + b_y + b_z + c_x + c_y + c_z\}. \tag{3.73}$$

Algoritam za rešavanje rekurentnih relacija bi mogao da se opiše sledećim koracima:

1. Podeliti integrale po grupama, tako da svaka grupa pripada jednom koraku postupka. Grupe bi trebale da budu organizovane tako da se izbegne ponavljanje računanja istih integrala.

2. U svakom koraku računati integrale po sledećem redosledu:

   (a) Prema jednačini (3.61) se računaju svi integrali koji pripadaju skupu (3.73).

   (b) Uz pomoć jednačina (3.66) i (3.70) (uz poštovanje kriterijuma za izbor) izračunati sve $[a0|\ c]^{(m)}$ integrale.

   (c) Računaju se svi $(a0|c)$ prema jednačini (3.65).

   (d) Računaju se svi integrali iz jednačine (3.59), sve dok se konačno ne dođe do integrala $(ab|c)$.

3. Da bi se izračunali svi potrebni $(ab|c)$ integrali, potrebno je ponoviti korake 1.-2. po svim atomima ulaznog sistema i svim grupama Gausijana.

Da bi se na što jednostavniji način opisao, prethodni algoritam je dat kroz niz sekvencijalnih koraka. Da bi opis implementacije rekurentnih relacija bio potpun, u sekciji 4.2.1 je uz pomoć pseudokoda i odgovarajućih obrazloženja dat opis implementacije koda uz pomoć rekurzivnih procedura.

U literaturi postoji veliki broj ideja za implementaciju efikasnih načina za rešavanje integrala datog u jednačini (3.44), sa pristupom drugačijim od našeg. Jedna takva metoda je bazirana na generalizaciji brze multipolne metode za računanje elektrostatičke energije naelektrisanja sistema [94, 95, 96]. Postoje i druge mogućnosti koje se značajno razlikuju od našeg pristupa (kompletno baziranog na korišćenju Gausijana) koje predstavljaju gustinu naelektrisanja u bazisu ravnih talasa i računaju integrale korišćenjem numeričkih efikasne Furijeove transformacije [97] ili se koriste wavelet funkcije [98].

3.1.5 Izmensko-korelacioni integral

Usled nelinearne zavisnosti izmensko-korelacionog potencijala V_{xc} od elektronske gustine naelektrisanja ρ, za izmensko-korelacioni integral (Exchange-corelation Integral) dat u jednačini (3.11) nije moguće dobiti analitičke formule. Iz tog razloga se odlučujemo za numeričku integraciju u prostoru [99], koja je bazirana na ideji opisanoj u [100]. Ukratko, ovde se prostor glatko razlaže na doprinose centara lociranih na svakom atomu. Za particionisanje koristimo funkciju iz [100]. Da bismo izračunali doprinose svih centara, vršimo numeričku integraciju u sfernom koordinatnom sistemu. Da bismo izračunali integraciju preko ugaonih koordinata koristimo Lebedevu mrežu [101], dok za radijalnu integraciju koristimo Chebyshev-Gauss mrežu transformisanu na interval $0 < r < \infty$ [99]. U numeričkom primeru u poglavlju 5 koristimo Lebedevu mrežu sa 74 tačke, radijalnu mrežu sa 40 tačaka i LDA izmensko-korelacioni potencijal.

3.1.6 Konvergencija računanja svojstvenih energija

U ovom delu će biti prikazan niz koraka koje je potrebno primeniti da bi se poboljšala konvergencije dela DFT algoritma za rešavanje svojstvenog problema. Primenom odgovarajućih metoda se dovodi do značajno manjeg broja iteracija računanja svojstvenih energija. Ovim se takođe smanjuje mogućnost dovođenja algoritma u nekonvergentno stanje. Opisani postupak je inspirisan metodom

opisanom u [102]. Ovaj metod je takođe formalno razvijen u radovima [103] i [104], sa razlikom što je stavljen u kontekst Hartri-Fok metode, dok ih mi primenjujemo na DFT.

Osnovna ideja postupka je da se umesto direktnog korišćenja gustine naelektrisanja iz prethodne iteracije, matrica H (iz jednačine (3.6)) formira na osnovu linearne kombinacije matrica H iz prethodnih iteracija.

Najpre je potrebno konstruisati matricu e:

$$e = H \cdot D \cdot S - S \cdot D \cdot H, \tag{3.74}$$

gde je

$$D = C \cdot N \cdot C^T, \tag{3.75}$$

pri čemu je C matrica koja sadrži α koeficijente (jednačina (3.1)) (u uopštenom slučaju gde vrednosti mogu biti kompleksni brojevi bi umesto C^T trebalo koristiti konjugovano transponovani oblik C^H), dok matrica N ima sledeći oblik:

$$\left[\begin{array}{c:c} I_{N/2,N/2} & 0 \\ \hdashline 0 & 0 \end{array}\right],$$

gde je $I_{N/2,N/2}$ jedinična matrica a N veličina sistema koji se rešava (broj Gausijana).

Takođe je potrebno konstruisati matricu e':

$$e' = A^T \cdot e \cdot A, \text{ gde je } A = S^{-1/2}. \tag{3.76}$$

Dalje je potrebno da izračunamo matricu $B_{i,j}$:

$$B_{ij} = Tr \cdot (e'_i \cdot e'^T_j), \tag{3.77}$$

gde $i, j \in \{1, 2, ..., k\}$ a k je broj iteracija koje učestvuju u formiranju matrice H.

Sada je potrebno rešiti sistem linearnih jednačina:

$$\begin{bmatrix} 0 & -1 & -1 & \dots & -1 \\ -1 & & & & \\ \vdots & & B & & \\ -1 & & & & \\ -1 & & & & \end{bmatrix} \begin{bmatrix} -\lambda \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Na kraju se na osnovu rešenja sistema linearnih jednačina i H_i matrica dobijenih u prethodnih k iteracija formira matrica H:

$$H = \sum_k c_k H_k. \tag{3.78}$$

Da bi se konvergencija dodatno ubrzala, pored predstavljenog procesa je takođe moguće poboljšati početni uslov za gustinu naelektrisanja (u odnosu na inicijalnu postavku gustine naelektrisanja na nula vrednost). Za početni uslov se može uzeti da su na početku talasne funkcije popunjenih stanja jednake kontrahovanim Gausijanima koji se koriste kao bazis, drugim rečima u jednačini (3.1) za dato i svi α koeficijenti su jednaki 0, osim jednog koji je jednak 1. Na ovaj način se početna gustina naelektrisanja postavlja na zbir gustina naelektrisanja pojedinačnih atoma što svakako predstavlja bolji početni uslov, čime doprinosi poboljšanju procesa konvergencije.

3.1.7 Primer upotrebe Gausijana: energija osnovnog stanja atoma vodonika

U nastavku je dat jednostavan primer upotrebe Gausijana za pronalaženje energije osnovnog stanja atoma vodonika. Šredingerova jednačina za slučaj atoma vodonika se može zapisati u sledećem obliku:

$$\left(-\frac{1}{2}\nabla^2 - \frac{1}{\mathbf{r}}\right)\psi(\mathbf{r}) = E\psi(\mathbf{r}). \tag{3.79}$$

Jednačinu (3.79) rešavamo tako što talasnu fukncij u $\psi(\mathbf{r})$ predstavimo u obliku linearne kombinacije Gausijana čija su rešenja prethodno opisana:

$$\psi(\mathbf{r}) = \sum_{\mu=1}^{2} \alpha_\mu \varphi_\mu(\mathbf{r}) = \alpha_1\varphi_1(\mathbf{r}) + \alpha_2\varphi_2(\mathbf{r}), \tag{3.80}$$

$$\left(-\frac{1}{2}\nabla^2 - \frac{1}{\mathbf{r}}\right)\sum_{\mu} \alpha_\mu \varphi_\mu(\mathbf{r}) = E\sum_{\mu} \alpha_\mu \varphi_\mu(\mathbf{r}), \tag{3.81}$$

$$\sum_{\mu} \alpha_\mu \int \mathrm{d}^3\mathbf{r}\, \varphi_\nu^*(\mathbf{r}) \left(-\frac{1}{2}\nabla^2 - \frac{1}{\mathbf{r}}\right)\varphi_\mu(\mathbf{r}) = E\sum_{\mu} \alpha_\mu \int \mathrm{d}^3\mathbf{r}\, \varphi_\nu^*(\mathbf{r})\varphi_\mu(\mathbf{r}). \tag{3.82}$$

Dalje definišemo $H_{\nu\mu}$ i $S_{\nu\mu}$:

$$H_{\nu\mu} \stackrel{\text{def}}{=} \int \mathrm{d}^3\mathbf{r}\, \varphi_\nu^*(\mathbf{r}) \left(-\frac{1}{2}\nabla^2 - \frac{1}{\mathbf{r}}\right)\varphi_\mu(\mathbf{r}), \tag{3.83}$$

$$S_{\nu\mu} \stackrel{\text{def}}{=} \int \mathrm{d}^3\mathbf{r}\, \varphi_\nu^*(\mathbf{r})\varphi_\mu(\mathbf{r}). \tag{3.84}$$

Tada se jednačina (3.79) može predstaviti sledećim izrazom:

$$\sum_{\mu} \alpha_\mu H_{\nu\mu} = E\sum_{\mu} \alpha_\mu S_{\nu\mu}, \tag{3.85}$$

što je oblik generalizovanog svojstvenog problema:

$$\left([H] - E\,[S]\right)\vec{\alpha} = 0. \tag{3.86}$$

Ukoliko primenimo unapred definisani bazisni skup SBKJC VDZ ECP [86, 87, 53, 54], u slučaju atoma vodonika, Gausijani $\varphi_1(\mathbf{r})$ i $\varphi_2(\mathbf{r})$ su sledeći;

$$\varphi_1(\mathbf{r}) = 0.033494\cdot\chi_s(18.711370) + 0.23472695\cdot\chi_s(2.8253937) + 0.81375733\cdot\chi_s(0.5401217), \tag{3.87}$$

$$\varphi_2(\mathbf{r}) = 1.0\cdot\chi_s(0.1612778), \tag{3.88}$$

gde je $\chi_s(\alpha)$ Gausijan (3.3) gde je $l = m = n = 0$ sa koeficijentom Gausijana α.

Algoritam za rešavanje energija osnovnog stanja atoma vodonika se može prikazati u dva koraka:

1. Izračunati elemente matrica $H_{\nu\mu}$ i $S_{\nu\mu}$ za sve $(\nu\mu) \in \{(1,1),(1,2),(2,1),(2,2)\}$. Primer

izračunavanja elementa $H_{1,1}$ je sledeći:

$$\begin{aligned}
H_{11} &= \int \mathrm{d}^3\mathbf{r}\, \varphi_1(\mathbf{r}) \left(-\frac{1}{2}\nabla^2 - \frac{1}{\mathbf{r}}\right) \varphi_1(\mathbf{r}) \\
&= \int \mathrm{d}^3\mathbf{r}\, (a_1^*\chi_1^* + a_2^*\chi_2^* + a_3^*\chi_3^*) \left(-\frac{1}{2}\nabla^2 - \frac{1}{\mathbf{r}}\right) (a_1\chi_1 + a_2\chi_2 + a_3\chi_3) \\
&= \int \mathrm{d}^3\mathbf{r}\, a_1^* a_1 \cdot \chi_1^* \left(-\frac{1}{2}\nabla^2 - \frac{1}{\mathbf{r}}\right) \chi_1 + \int \mathrm{d}^3\mathbf{r}\, a_1^* a_2 \cdot \chi_1^* \left(-\frac{1}{2}\nabla^2 - \frac{1}{\mathbf{r}}\right) \chi_2 + \\
&\quad \int \mathrm{d}^3\mathbf{r}\, a_1^* a_3 \cdot \chi_1^* \left(-\frac{1}{2}\nabla^2 - \frac{1}{\mathbf{r}}\right) \chi_3 + \int \mathrm{d}^3\mathbf{r}\, a_2^* a_1 \cdot \chi_2^* \left(-\frac{1}{2}\nabla^2 - \frac{1}{\mathbf{r}}\right) \chi_1 + \\
&\quad \int \mathrm{d}^3\mathbf{r}\, a_2^* a_2 \cdot \chi_2^* \left(-\frac{1}{2}\nabla^2 - \frac{1}{\mathbf{r}}\right) \chi_2 + \int \mathrm{d}^3\mathbf{r}\, a_2^* a_3 \cdot \chi_2^* \left(-\frac{1}{2}\nabla^2 - \frac{1}{\mathbf{r}}\right) \chi_3 + \\
&\quad \int \mathrm{d}^3\mathbf{r}\, a_3^* a_1 \cdot \chi_3^* \left(-\frac{1}{2}\nabla^2 - \frac{1}{\mathbf{r}}\right) \chi_1 + \int \mathrm{d}^3\mathbf{r}\, a_3^* a_2 \cdot \chi_3^* \left(-\frac{1}{2}\nabla^2 - \frac{1}{\mathbf{r}}\right) \chi_2 + \\
&\quad \int \mathrm{d}^3\mathbf{r}\, a_3^* a_3 \cdot \chi_3^* \left(-\frac{1}{2}\nabla^2 - \frac{1}{\mathbf{r}}\right) \chi_3.
\end{aligned} \tag{3.89}$$

Integrali navedeni u prethodnom izrazu predstavljaju zbir kinetičkog integrala (3.18) i integrala jezgra (3.24) čija su analitička rešenja prikazana u sekcijama 3.1.2 3.1.3. Na analogan način se računaju ostali elementi $H_{\nu\mu}$.

Da bi se formirao izraz za $S_{\nu\mu}$, potrebno je umesto kinetičkog integrala i integrala jezgra koristiti integral preklapanja čije je analitičko rešenje opisano u 3.1.1. Zbog analogije u odnosu na prethodni izraz, prikaz primera za $S_{\nu\mu}$ će biti izostavljen.

2. Nakon što se izračunaju svi elementi $H_{\nu\mu}$ i $S_{\nu\mu}$, potrebno je rešiti generalizovani svojstveni problem (3.86). Implementacija dijagonalizacija se vrši korišćenjem standardne serijske LAPACK rutine [82].

Za konačno rešenje navedenog primera se za najnižu svojstvenu energiju očekuje vrednost približno jednaka $-1/2$. Implementacija navedenog primera u programskom jeziku C daje vrednost -0.498, što pokazuje dobro slaganje sa očekivanom vrednošću.

3.2 Implementacija CPM u bazisu Gausijana

Ideja CPM je da se gustina elektronskog naelektrisanja predstavi kao zbir doprinosa pojedinačnih atoma, pri čemu je svaki od tih doprinosa dobro lokalizovan u okolini atoma. Osnovna razlika u implementaciji CPM-a u odnosu na implementaciju DFT-a, je što se u CPM algoritmu ne vrši računanje gustine naelektrisanja, već se koristi gustina naelektrisanja dobijena DFT postupkom, primenjenim na malom sistemu. Iz tog razloga se u algoritmu za rešavanje CPM ne koristi iterativni postupak kojim se ponavljaju operacije računanja jednačina [(3.7), (3.8), (3.9), (3.10) i (3.11)] radi konvergencije rezultata.

Ispostavilo se da pronalaženje odgovarajuće reprezentacije gustine naelektrisanja motiva predstavlja mnogo veći izazov nego što je to prvobitno očekivano. S obzirom da je elektronska gustina naelektrisanja prikazana kao suma Gausijana centriranih na atomu, ona je na neki način već razložena na doprinose pojedinačnih atoma. Stoga je ekstrahovanje motiva iz rezultata dobijenog nad malim sistemom, bilo logično uraditi prostim uzimanjem sume doprinosa svih Gausijana centriranih na tom atomu. U tom slučaju bi motiv predstavili preko odgovarajućih koeficijenata i na taj način bi generisanje gustine naelektrisanja za veliki sistem (iz tako formiranih motiva) bilo relativno lako.

Nažalost, nakon odgovarajućih implementacija i testiranja smo otkrili da ovakav pristup ne daje zadovoljavajući rezultat. Svojstvene energije dobijene iz CPM-a pokazuju velike razlike u odnosu na one dobijene punim DFT računanjem. Glavni razlog za ovo potiče iz činjenice da su parametri uz bazise Gausijana, namenjeni za gustinu naelektrisanja, razvijeni tako da omogućavaju odgovarajuće fitovanje za ukupnu elektronsku gustinu naelektrisanja. Gausijani koji se pritom koriste imaju mali ugaoni momenat (s, p i d). Sa druge strane, motiv koji opisuje susede datog atoma pokazuje značajan stepen anizotropije i kao posledica toga je za opis motiva potreban veći ugaoni momenat Gausijana.

Da bismo pronašli odgovarajuću proceduru za generisanje i reprezentaciju motiva, naš sledeći pokušaj je bio da upotrebimo jednačinu (2.3). Ideja je bila da izračunamo motiv za unapred definisani skup tačaka u prostoru i onda da tako dobijene funkcije predstavimo kao linearnu kombinaciju Gausijana. Ovde smo upotrebili isti skup Gausijana koji koristimo za predstavljanje elektronske gustine naelektrisanja. Gausijani koji su korišćeni za fitovanje su centrirani na centralnom atomu motiva i na njegovim susedima. Iako je rezultat koji smo na ovaj način dobili bio bolji od prethodnog pokušaja, zaključili smo da on i dalje nije zadovoljavajući. Eventualno bi moglo biti ispitano kako bi na rezultat uticala promena bazisa Gausijana. Da bi se ovo ispitalo potrebno je uložiti značajan napor za razvoj odgovarajućeg bazisnog skupa, što svakako smatramo neadekvatnim, jer bi potencijalnom korisniku CPM metode ovo bio značajan dodatni napor.

S obzirom da nismo bili zadovoljni rezultatima prethodnih pokušaja, odlučili smo da pokušamo da predstavimo motive preko njihovih vrednosti u skupu tačaka u prostoru. Izračunate vrednosti gustine naelektrisanja motiva iz (2.3) smo povezali sa tačkama raspoređenim na kubičnom gridu veličine $(2m_g + 1) \times (2m_g + 1) \times (2m_g + 1)$ unutar kocke dimenzija $a \times a \times a$. U numeričkom primeru u poglavlju 5 smo koristili vrednosti $m_g = 80$ i $a = 15\, a_0$. Centar kocke je pozicija atoma. Pri ovakvoj postavci je uz pomoć jednačine (2.3) moguće direktno izračunati gustine naelektrisanja motiva. Ono što nije trivijalno u ovom slučaju, je implementacija jednačine (2.4), odnosno dobijanje elektronske gustine naelektrisanja u bazisu Gausijana iz prostorne reprezentacije motiva. Način na koji smo rešili ovaj problem će biti detaljno objašnjen u nastavku rada.

Algoritam za rešavanje CPM-a čine sledeći koraci:

1. Konstruišu se matrice H i S na osnovu jednačina (3.7), (3.8) i (3.9).
2. Fitovanje gustine naelektrisanja dobijenog ekstrahovanjem motiva iz DFT proračuna (3.2.1).
3. Matrici H se dodaju doprinosi dobijeni rešavanjem jednačina (3.10) i (3.11) (opisi postupaka su dati u sekcijama 3.1.4 i 3.1.5).
4. Reši se generalizovani svojstveni problem (3.6) (opis dat u sekciji 3.2.3).

Za razliku od DFT algoritma, CPM algoritam ne sadrži iterativni postupak, već gustinu naelektrisanja dobija na osnovu fitavanja gustine naelekrisanja dobijenog ekstrahovanjem motiva iz DFT proračuna.

3.2.1 Ekstrahovanje motiva iz DFT proračuna

Gustina elektronskog naelektrisanja motiva se može predstaviti sledećim izrazom (jednačina (1) iz [50]):

$$m_A(\vec{r} - \vec{R_a}) = \frac{W_A(\vec{r} - \vec{R_a})}{\sum_B W_B(\vec{r} - \vec{R_b})} \rho(\vec{r}), \tag{3.90}$$

gde je $\rho(\vec{r})$ gustina elektronskog naelektrisanja dobijena DFT proračunom, koja se za svako $\vec{r}$ može izračunati preko bazisa kontrahovanih Gausijana na osnovu jednačine (3.5) ili preko jednočestičnih talasnih funkcija na osnovu jednačine (2.2).. $W_A(\vec{r})$ predstavlja sferno-simetričnu težinsku funkciju:

$$W_A(r) = \rho_A(r) M(r), \tag{3.91}$$

gde je $\rho_A(r)$ gustina elektronskog naelektrisanja izolovanog atoma A, dok je $M(r)$ dodatna funkcija oblika:

$$\begin{aligned} M(r) &= \frac{a + br^2}{a}, r \leq r_o, \\ M(r) &= \frac{e^{-\frac{E'r}{a}}}{a}, r > r_o, \end{aligned} \tag{3.92}$$

gde je $r_o = 3\ bohr$, $E' = 0.75\ \frac{1}{bohr}$, $b = -E' e^{-\frac{E'r_o}{2r_o}}$, $a = e^{-E'r_o} - br_o^2$

Izraz $\sum_B W_B(\vec{r} - \vec{R_B})$ predstavlja sumu težinskih funkcija po svim atomima iz molekula (ulaznog sistema).

Na osnovu početne jednačine (3.90) se za svako $\vec{r}$ može izračunati gustina elektronskog naelektrisanja motiva $m_A(\vec{r})$.

Dalje je poželjno izraziti $m_A(\vec{r})$ kao linearnu kombinaciju kontrahovanih Gausijana i umesto funkcije (3.90) pamtiti samo koeficijente uz Gausijane:

$$\overline{m}_A(\vec{r}) = \sum_i C_i \varphi_i(\vec{r}). \tag{3.93}$$

Koeficijenti C_i se biraju tako da je razlika između $m_A(\vec{r})$ i $\overline{m}_A(\vec{r})$ što manja, tj. da je:

$$\sum_k \gamma_K \left[m_A(\vec{r_k}) - \overline{m}_A(\vec{r_k}) \right]^2 = min, \tag{3.94}$$

gde se vektori $\vec{r_k}$ i težinski koeficijenti γ_k biraju na osnovu sledećih izraza:

$$\begin{aligned} x_k &= r_k \sin\theta_k \cos\varphi_k, \\ y_k &= r_k \sin\theta_k \sin\varphi_k, \\ z_k &= r_k \cos\theta_k, \\ \gamma_k &= r_k^2 \sin\theta_k, \\ r_k &\rightarrow\ 0...15\ bohr\ (korak\ 0.004), \\ \varphi_k &\rightarrow 0..2\pi\ (korak\ 2\pi/18), \\ \theta_k &\rightarrow 0..\pi\ (korak\ \pi/9). \end{aligned} \tag{3.95}$$

Izraz koji minimizujemo se dalje može prikazati na sledeći način:

$$\sum_k \gamma_K \left[m_A(\vec{r_k}) - \overline{m}_A(\vec{r_k})\right]^2 = \sum_k \gamma_K m_A(\vec{r_k})^2 + \sum_k \gamma_k \overline{m}_A(\vec{r_k})^2 - 2\sum_k \gamma_k m_A(\vec{r_k})\overline{m}_A(\vec{r_k})$$
$$= \sum_k \gamma_k m_A(\vec{r_k})^2 + \sum_k \gamma_k \left[\sum_i C_i\varphi_i(\vec{r_k})\right]^2 - 2\sum_k \gamma_k m_A(\vec{r_k}) \sum_i C_i\varphi_i(\vec{r_k})$$
$$= \sum_k \gamma_k m_A(\vec{r_k})^2 + \sum_k \gamma_k \sum_{ij} C_iC_j\varphi_i(\vec{r_k})\varphi_j(\vec{r_k}) - 2\sum_k \gamma_k m_A(\vec{r_k}) \sum_i C_i\varphi_i(\vec{r_k})$$
$$= \sum_k \gamma_k m_A(\vec{r_k})^2 + \sum_{ij} C_iC_j \sum_k \gamma_k\varphi_i(\vec{r_k})\varphi_j(\vec{r_k}) - \sum_i C_i 2\sum_k \gamma_k m_A(\vec{r_k})\varphi_i(\vec{r_k}). \quad (3.96)$$

Prva suma u poslednjoj jednačini je konstanta. Radi pojednostavljenog prikaza se $\sum_k \gamma_K\varphi_i(\vec{r_k})\varphi_j(\vec{r_k})$ može zameniti izrazom A_{ij}, dok se $\sum_k \gamma_K m_A(\vec{r_k})\varphi_i(\vec{r_k})$ može zameniti izrazom B_i. Sada se početni izraz može prikazati sledećom jednačinom:

$$f(C_1, C_2, ...) = \sum_{ij} C_iC_jA_{ij} - \sum_i C_iB_i. \quad (3.97)$$

Da bi funkcija f postigla minimum, potrebno je da njeni parcijalni izvodi budu jednaki nuli:

$$\frac{\partial f}{\partial C_k} = 0,$$
$$\frac{\partial f}{\partial C_k} = \sum_{ij}(A_{ij}\frac{\partial C_i}{\partial C_k}C_i + A_{ij}C_i\frac{\partial C_j}{\partial C_k}) - \sum_i B_i\frac{\partial C_i}{\partial C_k}$$
$$= \sum_j (A_{kj}C_j) + \sum_i (A_{ik}C_i) - B_k$$
$$= \sum_i (A_{ki} + A_{ik})C_i - B_k. \quad (3.98)$$

Na kraju koeficijente C_i možemo dobiti rešavanjem sistema linearnih jednačina:

$$\sum_i (A_{ki} + A_{ik})C_i = B_k. \quad (3.99)$$

Skraćeni opis algoritma za ekstrahovanje motiva bi se sada mogao opisati u tri koraka:

1. Koristeći jednačinu (3.90) izračunati gustinu elektronskog naelektrisanja $m_A(\vec{r})$ za sve $\vec{r_k}$.

2. Izračunati $\varphi_i(\vec{r_k})$ za sve $\vec{r_k}$.

3. Na osnovu jednačine (3.99) izračunati koeficijente C_i, pri čemu su A i B računaju uz pomoć formula (3.97) i (3.98).

Potrebno je napomenuti da konačna verzija implementacije CPM algoritma ne sadrži korake opisane jednačinama (3.93)–(3.99), iako se ne isključuje mogućnost njihove upotreba u budućim implementacijama koje bi primenile drugačiji pristup za reprezentaciju i ekstrahovanje motiva.

Fitovanje gustine naelektrisanja

Da bismo izračunali koeficijenate, na osnovu izračunate gustine naelektrisanja u realnom prostoru, potrebno je izvršiti fitovanje gustina naelektrisanja sa nametanjem uslova za ukupno naelektrisanje.

Ukupnu gustinu naelektrisanja želimo da izrazimo uz pomoć linearne kombinacije Gausijana:

$$\rho_{fit}(\vec{r}) = \sum_\alpha C_\alpha\phi_\alpha(\vec{r}). \quad (3.100)$$

Da bismo to uradili, potrebno je minimizirati razliku između unapred izračunate gustine naelektrisanja izračunate u realnom prostoru i fitovane gustine nalektrisanja. Da bismo problem sveli na sistem linearnih jednačina, za minimizaciju biramo sumu kvadrata razlike:

$$\sum_k W_k \left[\rho(\vec{r_k}) - \rho_{fit}(\vec{r_k})\right]^2 = min, \tag{3.101}$$

uz uslov da je

$$\int d^3r \ \rho_{fit}(\vec{r}) = Q, \tag{3.102}$$

gde je Q ukupno naelektrisanje.
Dalje uvodimo funkciju nad svim koeficijentima C_λ:

$$\begin{aligned} f(\{C_\alpha\}, \lambda) &= \sum_k W_k \left[\rho(\vec{r_k}) - \sum_\alpha C_\alpha \phi_\alpha(\vec{r_k})\right]^2 - \lambda \left[\sum_\alpha C_\alpha \int d^3r \ \phi_\alpha(\vec{r}) - Q\right] \\ &= \sum_k W_k \rho(\vec{r_k})^2 - 2\sum_k W_k \rho(\vec{r_k}) \sum_\alpha C_\alpha \phi_\alpha(\vec{r_k}) + \sum_k W_k \sum_{\alpha\beta} C_\alpha C_\beta \phi_\alpha(\vec{r_k}) \phi_\beta(\vec{r_k}) - \\ &\lambda \sum_\alpha C_\alpha \int d^3r \phi_\alpha(\vec{r}) + \lambda Q, \end{aligned} \tag{3.103}$$

gde λ predstavlja Lagranžov multiplikator.
Dalje iz $\frac{\partial f}{\partial \lambda} = 0$ i $\frac{\partial f}{\partial C_\mu} = 0$ sledi:

$$\frac{\partial f}{\partial \lambda} = 0 \ \Rightarrow \ \sum_\alpha C_\alpha \int d^3r \ \phi_\alpha(\vec{r}) = Q, \tag{3.104}$$

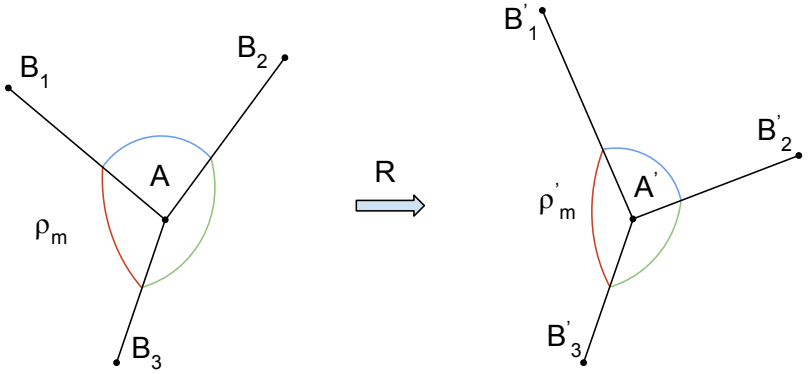
$$\begin{aligned} \frac{\partial f}{\partial C_\mu} = 0 \ \Rightarrow \ &-2\sum_k W_k \rho(\vec{r_k}) \phi_\mu(\vec{r_k}) - \lambda \int d^3r \phi_\alpha(\vec{r}) + \sum_k W_k \sum_{\alpha\beta} \delta_{\mu\alpha} C_\beta \phi_\alpha(\vec{r_k}) \phi_\beta(\vec{r_k}) + \\ &\sum_k W_k \sum_{\alpha\beta} C_\alpha \delta_{\mu\beta} \phi_\alpha(\vec{r_k}) \phi_\beta(\vec{r_k}) = 0. \\ &2\sum_k W_k \sum_\alpha C_\alpha \phi_\mu(\vec{r_k}) \phi_\alpha(\vec{r_k}) = 2\sum_k W_k \rho(\vec{r_k}) \phi_\mu(\vec{r_k}) + \lambda \int d^3r \ \phi_\mu(\vec{r}). \end{aligned} \tag{3.105}$$

Na osnovu poslednjeg izraza dobijamo sistem linearnih jednačina:

$$\sum_\alpha A_{\mu\alpha} C_\alpha - \varrho_\mu \lambda = B_\mu, \tag{3.106}$$

gde su $A_{\mu\alpha} = 2\sum_k W_k \phi_\mu(\vec{r_k}) \phi_\alpha(\vec{r_k})$, $B_\mu = 2\sum_k W_k \rho(\vec{r_k}) \phi_\mu(\vec{r_k})$ i $\varrho_\mu = \int d^3r \ \phi_\mu(\vec{r})$,
koji se može prikazati i u sledećem obliku:

$$\begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} & -\varrho_1 \\ A_{21} & A_{22} & \dots & A_{2n} & -\varrho_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ A_{n1} & A_{n2} & \dots & A_{nn} & -\varrho_n \\ \varrho_1 & \varrho_2 & \dots & \varrho_n & 0 \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_n \\ \lambda \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \\ Q \end{bmatrix},$$



Slika 3.1: Rotacija motiva u prostoru

Rotacija motiva

Kada želimo da iskoristimo motive generisane u malom prototip sistemu, prostorna orijentacija susednih atoma u velikom sistemu može biti zarotirana u odnosu na mali prototip sistem. Iz tog razloga je potrebno izvršiti odgovarajuću rotaciju motiva (slika 3.1).

Gustina naelektrisanja motiva se može predstaviti sledećim izrazom:

$$\rho_m(\vec{r}) = \sum_\mu C_\mu \varphi_\mu(\vec{r}; \vec{R_A}), \tag{3.107}$$

gde je $\varphi_\mu(\vec{r}; \vec{R})$ kontrahovan Gausijan centriran u tački A

Neka je R matrica rotacije koja preslikava:

$$\begin{aligned} R \cdot (\vec{R}_{B_1} - \vec{R}_A) &= \vec{R}_{B_1'} - \vec{R}_{A'}, \\ R \cdot (\vec{R}_{B_2} - \vec{R}_A) &= \vec{R}_{B_2'} - \vec{R}_{A'}, \\ R \cdot (\vec{R}_{B_3} - \vec{R}_A) &= \vec{R}_{B_3'} - \vec{R}_{A'}, \end{aligned} \tag{3.108}$$

gde za matricu R važi $R \cdot R^T = 1$, $R^T = R^{-1}$.

Tada je:

$$\begin{aligned} &\rho'_m(\vec{R}_{A'} + R \cdot (\vec{r} - \vec{R}_A)) = \rho_m(\vec{r}), \\ &\vec{u} = R \cdot (\vec{r} - \vec{R}_A) \;\Rightarrow\; \vec{r} = \vec{R}_A + R^{-1}\vec{u}, \\ &\rho'_m(\vec{R}_{A'} + \vec{u}) = \rho_m(\vec{R}_A + R^{-1}\vec{u}), \\ &\rho'_m(\vec{R}_{A'} + \vec{u}) = \sum_\mu C_\mu \varphi_\mu(\vec{R}_A + R^{-1}\vec{u}; \vec{R}_A) = \sum_\mu C_\mu \varphi_\mu(R^{-1}\vec{u}; 0). \end{aligned} \tag{3.109}$$

Sa druge strane, gustinu naelektrisanja zarotiranog motiva ρ'_μ želimo da predstavimo u obliku:

$$\rho'_m(\vec{R}_{A'} + \vec{u}) = \sum_\mu C'_\mu \varphi_\mu(\vec{u}; 0). \tag{3.110}$$

Iz poslednje dve jednačine sledi:

$$\sum_\mu C_\mu \varphi_\mu(R^{-1}\vec{u}; 0) = \sum_\mu C'_\mu \varphi_\mu(\vec{u}; 0). \tag{3.111}$$

1. slučaj - φ_μ je Gausijan s-tipa ($l+m+n=0$). Tada je $C'_\mu = C_\mu$.
2. slučaj - φ_μ su Gausijani p-tipa ($l+m+n=1$), gde je:

$$\begin{aligned}
&\varphi_{p_x} \text{ Gausijan gde važi } l=1, m=0, n=0.\\
&\varphi_{p_y} \text{ Gausijan gde važi } l=0, m=1, n=0.\\
&\varphi_{p_z} \text{ Gausijan gde važi } l=0, m=0, n=1.
\end{aligned} \tag{3.112}$$

Tada jednačina (3.111) glasi:

$$\begin{aligned}
&C_{p_x}\varphi_{p_x}(R^{-1}\vec{r}) + C_{p_y}\varphi_{p_y}(R^{-1}\vec{r}) + C_{p_z}\varphi_{p_z}(R^{-1}\vec{r}) = C'_{p_x}\varphi_{p_x}(\vec{r}) + C'_{p_y}\varphi_{p_y}(\vec{r}) + C'_{p_z}\varphi_{p_z}(\vec{r})\\
&C_{p_x}(R^{-1}\vec{r})_x + C_{p_y}(R^{-1}\vec{r})_y + C_{p_z}(R^{-1}\vec{r})_z = C'_{p_x}x + C'_{p_y}y + C'_{p_z}z,
\end{aligned} \tag{3.113}$$

$$\begin{aligned}
(R^{-1}\vec{r})_x &= (R^T\vec{r})_x = R_{11}x + R_{21}y + R_{31}z,\\
(R^{-1}\vec{r})_y &= (R^T\vec{r})_y = R_{12}x + R_{22}y + R_{32}z,\\
(R^{-1}\vec{r})_z &= (R^T\vec{r})_z = R_{13}x + R_{23}y + R_{33}z,
\end{aligned} \tag{3.114}$$

odatle dalje važi:

$$\begin{aligned}
C'_{p_x} &= R_{11}C_{p_x} + R_{12}C_{p_y} + R_{13}C_{p_z},\\
C'_{p_y} &= R_{21}C_{p_x} + R_{22}C_{p_y} + R_{23}C_{p_z},\\
C'_{p_z} &= R_{31}C_{p_x} + R_{32}C_{p_y} + R_{33}C_{p_z},
\end{aligned} \tag{3.115}$$

3. slučaj - φ_μ je Gausijan d-tipa ($l+m+n=2$):

$$\begin{aligned}
\varphi_{d_{x^2}} &= \left(\frac{2\alpha}{\pi}\right)^{3/4}\sqrt{\frac{1}{12}}x^2e^{-\alpha r^2} \text{ gde je } l=2, m=0, n=0,\\
\varphi_{d_{y^2}} &= \left(\frac{2\alpha}{\pi}\right)^{3/4}\sqrt{\frac{1}{12}}y^2e^{-\alpha r^2} \text{ gde je } l=0, m=2, n=0,\\
\varphi_{d_{z^2}} &= \left(\frac{2\alpha}{\pi}\right)^{3/4}\sqrt{\frac{1}{12}}z^2e^{-\alpha r^2} \text{ gde je } l=0, m=0, n=2,\\
\varphi_{d_{xy}} &= \left(\frac{2\alpha}{\pi}\right)^{3/4}\sqrt{\frac{1}{4}}xye^{-\alpha r^2} \text{ gde je } l=1, m=1, n=0,\\
\varphi_{d_{xz}} &= \left(\frac{2\alpha}{\pi}\right)^{3/4}\sqrt{\frac{1}{4}}xze^{-\alpha r^2} \text{ gde je } l=1, m=0, n=1,\\
\varphi_{d_{yz}} &= \left(\frac{2\alpha}{\pi}\right)^{3/4}\sqrt{\frac{1}{4}}yze^{-\alpha r^2} \text{ gde je } l=0, m=1, n=1.
\end{aligned} \tag{3.116}$$

Tada jednačina (3.111) glasi:

$$\begin{aligned}
&\frac{1}{\sqrt{3}}(x^2C'_{x^2} + y^2C'_{y^2} + z^2C'_{z^2}) + xyC'_{xy} + xzC'_{xz} + yzC'_{yz}\\
&= \frac{1}{\sqrt{3}}\left[(R^{-1}\vec{r})_x^2C_{x^2} + (R^{-1}\vec{r})_y^2C_{y^2} + (R^{-1}\vec{r})_z^2C_{z^2}\right] +\\
&(R^{-1}\vec{r})_x(R^{-1}\vec{r})_yC_{xy} + (R^{-1}\vec{r})_x(R^{-1}\vec{r})_zC_{xz} + (R^{-1}\vec{r})_y(R^{-1}\vec{r})_zC_{yz},
\end{aligned} \tag{3.117}$$

odakle je konačno

$$
\begin{aligned}
C'_{x^2} =&(R_{11}^2 C_{x^2} + R_{12}^2 C_{y^2} + R_{13}^2 C_{z^2}) + \sqrt{3}(R_{11}R_{12}C_{xy} + R_{12}R_{13}C_{yz} + R_{11}R_{13}C_{xz}),\\
C'_{y^2} =&(R_{21}^2 C_{x^2} + R_{22}^2 C_{y^2} + R_{23}^2 C_{z^2}) + \sqrt{3}(R_{21}R_{22}C_{xy} + R_{22}R_{23}C_{yz} + R_{21}R_{23}C_{xz}),\\
C'_{z^2} =&(R_{31}^2 C_{x^2} + R_{32}^2 C_{y^2} + R_{33}^2 C_{z^2}) + \sqrt{3}(R_{31}R_{32}C_{xy} + R_{32}R_{33}C_{yz} + R_{31}R_{33}C_{xz}),\\
C'_{xy} =&\frac{2}{\sqrt{3}}(R_{11}R_{21}C_{x^2} + R_{12}R_{22}C_{y^2} + R_{13}R_{23}C_{z^2}) +\\
&(R_{11}R_{21} + R_{21}R_{12})C_{xy} + (R_{11}R_{23} + R_{21}R_{13})C_{xz} + (R_{12}R_{23} + R_{22}R_{13})C_{yz},\\
C'_{yz} =&\frac{2}{\sqrt{3}}(R_{21}R_{31}C_{x^2} + R_{22}R_{32}C_{y^2} + R_{23}R_{33}C_{z^2}) +\\
&(R_{22}R_{31} + R_{32}R_{21})C_{xy} + (R_{23}R_{31} + R_{33}R_{21})C_{xz} + (R_{22}R_{33} + R_{32}R_{32})C_{yz},\\
C'_{xz} =&\frac{2}{\sqrt{3}}(R_{31}R_{11}C_{x^2} + R_{32}R_{12}C_{y^2} + R_{33}R_{13}C_{z^2}) +\\
&(R_{31}R_{12} + R_{11}R_{22})C_{xy} + (R_{33}R_{11} + R_{13}R_{31})C_{xz} + (R_{33}R_{12} + R_{13}R_{32})C_{yz}.
\end{aligned}
\tag{3.118}
$$

Potrebno je napomenuti da iako je u ovom delu prikazana rotacija motiva u celini, u slučaju implementacije gde se čuvanje motiva gustine naelektrisanja vrši uz pomoć prostorne reprezentacije, koraci 3.112–3.118 se mogu u potpunosti eliminisati.

3.2.2 Odsecanje pri računanju integrala

S obzirom da su funkcije Gausijana dobro lokalizovane u prostoru, integrali koji sadrže proizvod Gausijana centriranih na atomima koji su međusobno prostorno dovoljno razdvojeni, su praktično zanemarljivi. Kao posledicu toga, njihovo računanje se može izbeći, što mi i koristimo da bismo smanjili obim računanja. Ovo je posebno važno pri određivanju sistematičnog kriterijuma da li određeni integral treba izračunati li ne. U tom slučaju određujemo maksimalni poluprečnik r_g kontrahovanog Gausijana, tako da zadovoljava uslov:

$$
\sum_j a_j N\left(l,m,n,\alpha_j\right) r_g^{l+m+n} e^{-\alpha_j {r_g}^2} = \varepsilon. \tag{3.119}
$$

Takvim izborom r_g garantujemo da, kada je $r > r_g$, imamo $|\varphi_{lmn}(\mathbf{r})| < \varepsilon$, odnosno da će Gausijan izvan sfere poluprečnika r_g imati vrednost manju od ε. Dalje, za svaki par Gausijana proveravamo da li je razmak između njihovih centara veći od sume njihovih poluprečnika. Ukoliko je to slučaj, tada iz izračunavanja eliminišemo sve integrale koji učestvuju u proizvodu sa ova dva Gausijana.

Ovakvo odsecanje smanjuje broj potrebnih računanja za integrale preklapanja [Eq. (3.7)], kinetičke integrale [Eq. (3.8)] i izmensko-korelacione [Eq. (3.11)] integrale sa $O\left(N^2\right)$ na $O\left(N\right)$. Složenost izračunavanja integrala jezgra [Eq. (3.21)], integrala pseudopotencijala [Eq. (3.22)] i Hartri integrala [Eq. (3.44)] se smanjuje sa $O\left(N^3\right)$ na $O\left(N^2\right)$. Korišćenjem ove osobine lokalizacije funkcija Gausijana, ukupna složenost izračunavanja se smanjuje sa $O\left(N^3\right)$ na $O\left(N^2\right)$. Ono što još uvek nije dovoljno ispitano u našoj implementaciji je mogućnost dalje optimizacije računanja, korišćenjem činjenice da i Kulonov član $\frac{1}{|\mathbf{r}-\mathbf{R}_a|}$ opada sa povećanjem rastojanja, pa bi stoga bilo moguće izvršiti dodatna odsecanja.

U numeričkom primeru u poglavlju 5 koristimo $\varepsilon = 10^{-4}\, {a_0}^{-3/2}$. Takvo odsecanje uvodi grešku u svojstvenoj energiji manju od 0.1 mHa, što prestavlja prihvatljivu vrednost.

3.2.3 Dijagonalizacija Hamiltonijana

Nakon izračunavanja svih potrebnih integrala, potrebno je rešiti generalizovani svojstveni problem dat u jednačini (3.6). U trenutnoj jednoprocesorskoj implementaciji dijagonalizacija se vrši korišćenjem standardne jednoprocesorske LAPACK rutine [82].

Velika prednost bazisa Gausijana je što oni i sa relativno malim brojem Gausijana po atomu, precizno predstavljaju talasnu funkciju. Na primer, bazisni skup SBKJC VDZ ECP, koji koristimo u numeričkom primeru u poglavlju 5, sadrži 8 Gausijana po Si atomu i 2 Gausijana po H atomu. Tako je čak i za velike sisteme koje smo koristili za testiranje koda (kao što je $Si_{837}H_{348}$), dimenzija odgovarajućih Hamiltonijanskih matrica relativno mala (7392×7392). Iz tog razloga dijagonalizacija uz pomoć LAPACK rutine predstavlja relativno mali procenat od ukupnog vremena izvršavanja programa.

U slučaju paralelne verzije CPM programa, kojim se vrši računanje sistema značajno većih od sistema testiranih uz pomoć serijske implementacije, direktna dijagonalizacija Hamiltonijana predstavlja značajan faktor. Razlog za ovo je njeno N^3 skaliranje sa veličinom sistema. Ono što je moguće iskoristiti u našem slučaju je činjenica da se radi o dijagonalizaciji retke matrice. Neki od načina da se smanji vreme izvršavanja i dobije bolje skaliranje sa veličinom sistema, je primena metoda za dijagonalizacije kao što su metoda preklopljenog spektra [55] i metoda preklopljenih fragmenata [56]. Detalji u vezi implementacije paralelne verzije dijagonalizacije Hamiltonijana su opisani u sekciji 4.3.7.

Poglavlje 4

Softverska implementacija

U ovom poglavlju je dat opis implementacije softverskih rešenja na osnovu algoritama uvedenih u poglavlju 3. U sekciji 4.1 je najpre dat opis globalnih strukture koda, definicija softverskih komponenata koje su sadržane u konačnoj verziji programske implementacije, kao i način alokacije memorije. Centralno mesto teze je implementacija programskih rešenja koja je data kroz opis implementacije serijskih algoritama u sekciji 4.2 i paralelizacije CPM u sekciji 4.3. Kako je glavni doprinos iz aspekta implementacije koda onaj koji se odnosi na paralelizaciju CPM, tako je manja pažnja posvećena implementaciji serijskih algoritama, dok je primat stavljen na implementaciju koja se odnosi na procese paralelizacije CPM.

4.1 Struktura i opis koda

Rezultat implementacije algoritama opisanih u poglavlju 3 se iz korisničkog aspekta može prikazati kroz četiri softverska rešenja:

- Serijski DFT program za mali prototip sistem (korak 1 na slici 2.1) koji na osnovu ulaza koji čini definicija prototip sistema računa gustinu naelektrisanja. Rezultat se čuva u tekstualnom fajlu. Pošto ovaj program rešava kompletnu DFT proceduru, uz pomoć njega je takođe moguće izračunati elektronsku strukturu manjih sistema.

- Serijski program za ekstrahovanje motiva (korak 2 na slici 2.1) za ulazne podatke koristi gustinu naelektrisanja dobijenu izvršavanjem DFT programa, kao i definiciju prototip sistema na osnovu koga želimo da ekstrahujemo odgovarajuće motive. Rezultat izvršavanja ovog programa je skup fajlova u binarnoj reprezentaciji koji sadrže podatke koji opisuju motive. Motivi se mogu primeniti na celu klasu sistema koji odgovaraju prototipu na osnovu koga su ekstrahovani.

- Serijski CPM program za male sisteme (korak 3 na slici 2.1) za ulaz koristi odgovarajući skup motiva (dobijenih izvršavanjem programa za ekstrahovanje motiva) koji odgovaraju klasi sistema koji se rešava. S obzirom na algoritamsku efikasnost ovog programa, njime se mogu rešavati sistemi značajno većih dimenzija od onih koji se mogu rešiti DFT programom. Najveći sistem koji je testiran ovim programom ima 1500 atoma.

- Paralelni MPI CPM program za velike sisteme (korak 3 na slici 2.1), isto kao i serijski, za ulaz koristi odgovarajući skup motiva (dobijen određivanjem gustine naelektrisanja malog prototip sistema), kao i definiciju velikog sistema koji pripada istoj klasi sistema nad kojim su ekstrahovani motivi. Paralelna verzija programa je namenjena za izvršavanje na računarskim sistemima (klasterima) sa distribuiranom memorijom. Rezultat izvršavanja ovog programa je skup elektronskih stanja sistema. Paralelna implementacija je namenjena za rešavanje sistema od nekoliko desetina hiljada atoma.

Za implementaciju koda je korišćen programski jezik C. Produkcione verzije programa, koje su nabrojane u prethodnoj listi, sadrže približno 20000 linija koda. Deo koda koji se smatra redudantnim, u smislu da pripada delovima procesa koji su funkcionalni ali su zamenjeni optimalnijim ili rešenjima sa većom preciznošću rezultata, sadrži približno 6000 linija koda.

Upravljanje programima se vrši uz pomoć korisničkog interfejsa koji na osnovu ulaznih podataka i odabira 'moda' pokreće odgovarajući program. Od korisnika se očekuje da kao pripremu za korišćenje DFT programa najpre definiše mali prototip sistem. Rezultat DFT programa je izlazni fajl sa opisanom gustinom naelektrisanja koji bi dalje trebalo iskoristiti kao ulazni parametar programa za ekstrahovanje motiva. Tek nakon ovog koraka je moguće definisati skup sistema (koji pripadaju istoj klasi kao i prototip sistem) koji se zajedno sa odgovarajućim motivima mogu koristiti kao ulazni parametar serijskog ili paralelnog koda za rešavanje CPM-a. Prednost ovakve organizacije programa je što se priprema za računanje CPM-a, koja uključuje dobijanje gustine naelektrisanja i motiva, može izvršiti na personalnom računaru. Takođe se jednom ekstrahovan skup motiva može više puta primeniti na celu klasu sistema različitih veličina. U zavisnosti od veličine sistema CPM program se dalje može izvršiti na personalnom računaru ili na paralelnom računarskom okruženju.

4.1.1 Globalne strukture

Struktura u programskom jeziku C predstavlja kompozitni tip podataka odnosno omogućava grupisanje promenljivih u logičke celine koje se pod istim nazivom smeštaju u jedan blok memorije. Na ovaj način se uz pomoć jednog pokazivača može pristupiti različitim promenljivim definisanim unutar strukture, čime se na prirodan način omogućava pristup logičkim celinama. Osim primitivnih tipova, strukture mogu sadržati definicije nizova kao i pokazivače.

Upravo zbog navedenih karakteristika, strukture su više nego poželjne za organizovanje velikog broja promenljivih potrebnih za realizovanje DFT i CPM programa. Prirodno grupisanje promenljivih inspirisano hemijskim strukturama je dovelo do sledećih C struktura nabrojanih u listingu 4.1:

- *molecule* je osnovna struktura koja predstavlja ulazni sistem odnosno molekul. Ona je zajednička za DFT i CPM program. Kreira se prilikom inicijalizacije promenljivih na osnovu ulaznih podataka učitanih od strane korisnika i nalazi se u programskoj memoriji do kraja izvršavanja programa. Elementi strukture su niz pokazivača na strukturu *nuclei*, lista hemijskih elemenata (celobrojnog tipa) predstavljenih uz pomoć atomskog broja, kao i tri celobrojne promenljive: veličina sistema (broj atoma), broj elektrona i broj različitih hemijskih elemenata.

- *nuclei* odnosno jezgro atoma u sebi sadrži podatke o poziciju u prostoru, nazivu hemijskog elementa, atomskom broju, broju valentnih elektrona, tipu, listu suseda kao i njihov ukupan broj. Promenljiva *motif* je indentifikator koji određuje da li je *nuclei* centar motiva.

- *phi* predstavlja kontrahovani Gausijan za čiji opis su potrebni: niz struktura *term*, pozitivni celobrojni stepeni Gausijana (l, m, n), broj funkcija Gausijana kao i koeficijent u slučaju da se struktura koristi za opis motiva.
- *term* predstavlja pomoćnu strukturu za opis funkcije Gausijana; sadrži koeficijent Gausijana i eksponent orbitale Gausijana.
- *motif* je proširenje strukture *molecule* i sadrži elemente koji su potrebni da bi se predstavio motiv: naziv, molekul koji je 'centar' motiva i indentifikator o postojanju suseda drugog reda. *motif* se inicijalizuje prilikom učitavanja podataka za CPM program i nalazi se u memoriji do kraja izvršavanja programa.
- *recIntegral* je pomoćna struktura za rekurentnu proceduru koja računa Hartri integral i koja sadrži identifikator inicijalizacije vrednosti integrala kao i vrednost dobijenu računanjem integrala opisanog u 3.1.4.
- *ecp* kao i pomoćne strukture *ecpr* i *ecpterm* sadrže podatke koji opisuju efektivni potencijal jezgra za lokalne i nelokalne članove opisane u 3.1.3.
- *sph* i pomoćna struktura *sphterm* su namenjene za opis sfernih harmonika koji su takođe potrebni za deo programa za računanje pseudopotencijala 3.1.3.

Listing 4.1: Definicije C struktura

```
struct molecule{
   struct nuclei atom [numAtoms];
   int size;
   int N;
   int elementNum;
   int elements[118];
};

struct nuclei{
   int id;
   double x,y,z;
   char name[2];
   int Zc;
   int val;
   int type;
   int motif;
   int numOfNeighbors;
   int neighbors[6];
};

struct phi{
   struct term t [phiTerms];
   int l,m,n;
   int size;
   double coeff;
};

struct term{
   double a,alpha;
};

struct motif{
```

```
   char name[25];
   struct molecule mol;
   struct phi phiG [MAXRho];
   int numOfPhiG;
   int secOrderN;
};

struct recIntegral{
   short taken;
   double ivalue;
};

struct ecp{
   int L;
   struct ecpr vLocal;
   struct ecpr vNonlocalL [ecpLmax];
};

struct ecpr{
        int size;
        struct ecpterm t [ecpTerms];
};

struct ecpterm{
   double A, B, r;
   int n;
};

struct sph {
   int l,m;
   int size;
   struct sphterm t [sphTerms];
};

struct sphterm{
   int sx,sy,sz;
   double complex alpha;
};
```

4.1.2 Alokacija memorije

Pri izvršavanju procedura za rešavanje DFT-a i CPM-a je potrebno alocirati prostor za promenljive i postaviti ih na inicijalne vrednosti. Neke strukture podataka (najčešće nizovi, matrice i tenzori) u kodu imaju značajno memorijsko zauzeće, pa im je potrebno posvetiti posebnu pažnju. Neke od stvari o kojima treba voditi računa su: pravovremena realokacija memorije, korišćenje adekvatnih tipova promenljivih, balans između zauzeća memorije i efikasnosti procedura, itd. Matrice i tenzori se u memoriji alociraju kao neprekidni nizovi odgovarajućeg tipa, gde su pokazivači organizovani tako da omogućavaju pristup elementima dvodimenzionim i trodimenzionalnim indeksiranjem (pogledati listing 4.2). Ovo omogućava intuitivniji način pristupa elementima, gde se na primer elementu tenzora A, umesto jednodimenzionalnim indeksiranjem (npr. $A[i*Ny*Nz+j*Nz+k]$), pristupa sa $A[i][j][k]$. Elementi mogu biti realni brojevi, celi brojevi ili strukture *phi*, *motif* ili *recIntegral* navedene u 4.1.1.

Listing 4.2: Memorijska alokacija vektora, matrica i tenzora

```
// Double vector alokacija
double *alloc_double_vector(long Nx) {
   double *vector;

   if((vector = (double *) malloc((size_t) (Nx * sizeof(double)))) == NULL) {
      fprintf(stderr, "Failed to allocate memory for the vector.\n"); exit(EXIT_FAILURE);
   }

   return vector;
}
// Double matrix alokacija
double **alloc_double_matrix(long Nx, long Ny) {
   long i;
   double **matrix;

   if((matrix = (double **) malloc((size_t) (Nx * sizeof(double *)))) == NULL) {
      fprintf(stderr, "Failed to allocate memory for the matrix.\n"); exit(EXIT_FAILURE);
   }
   if((matrix[0] = (double *) malloc((size_t) (Nx * Ny * sizeof(double)))) == NULL) {
      fprintf(stderr, "Failed to allocate memory for the matrix.\n"); exit(EXIT_FAILURE);
   }
   for(i = 1; i < Nx; i ++)
      matrix[i] = matrix[i - 1] + Ny;

   return matrix;
}
// Double tensor alokacija
double ***alloc_double_tensor(long Nx, long Ny, long Nz) {
   long i, j;
   double ***tensor;

   if((tensor = (double ***) malloc((size_t) (Nx * sizeof(double **)))) == NULL) {
      fprintf(stderr, "Failed to allocate memory for the tensor.\n"); exit(EXIT_FAILURE);
   }
   if((tensor[0] = (double **) malloc((size_t) (Nx * Ny * sizeof(double *)))) == NULL) {
      fprintf(stderr, "Failed to allocate memory for the tensor.\n"); exit(EXIT_FAILURE);
   }
   if((tensor[0][0] = (double *) malloc((size_t)(Nx * Ny * Nz * sizeof(double)))) == NULL){
      fprintf(stderr, "Failed to allocate memory for the tensor.\n"); exit(EXIT_FAILURE);
   }
   for(j = 1; j < Ny; j ++)
      tensor[0][j] = tensor[0][j-1] + Nz;
   for(i = 1; i < Nx; i ++) {
      tensor[i] = tensor[i - 1] + Ny;
      tensor[i][0] = tensor[i - 1][0] + Ny * Nz;
      for(j = 1; j < Ny; j ++)
         tensor[i][j] = tensor[i][j - 1] + Nz;
   }

   return tensor;
}
// Struct phi vector alokacija
struct phi *alloc_phi_vector(long Nx) {
   struct phi *vector;

   if((vector = (struct phi *) malloc((size_t) (Nx * sizeof(struct phi)))) == NULL) {
      fprintf(stderr, "Failed to allocate memory for the vector.\n");
      exit(EXIT_FAILURE);
   }

   return vector;
```

```
}
// Molecule vector alokacija
struct motif *alloc_motif_vector(long Nx) {
   struct motif *vector;

   if((vector = (struct motif *) malloc((size_t) (Nx * sizeof(struct motif)))) == NULL) {
      fprintf(stderr, "Failed to allocate memory for the vector.\n");
      exit(EXIT_FAILURE);
   }

   return vector;
}
// recIngegral tensor alokacija
struct recIntegral ***alloc_recIntegral_tensor(long Nx, long Ny, long Nz) {
   long cnti, cntj;

   struct recIntegral1 ***tensor;

   if((tensor = (struct recIntegral1 ***) malloc((size_t) (Nx * sizeof(struct recIntegral1
       **)))) == NULL) {
      fprintf(stderr, "Failed to allocate memory for the tensor 1.\n");
      exit(EXIT_FAILURE);
   }
   if((tensor[0] = (struct recIntegral1 **) malloc((size_t) (Nx * Ny * sizeof(struct
       recIntegral1 *)))) == NULL) {
      fprintf(stderr, "Failed to allocate memory for the tensor 2.\n");
      exit(EXIT_FAILURE);
   }
   if((tensor[0][0] = (struct recIntegral1 *) malloc((size_t) (Nx * Ny * Nz *
       sizeof(struct recIntegral1)))) == NULL) {
      fprintf(stderr, "Failed to allocate memory for the tensor 3.\n");
      exit(EXIT_FAILURE);
   }
   for(cntj = 1; cntj < Ny; cntj ++)
      tensor[0][cntj] = tensor[0][cntj-1] + Nz;
   for(cnti = 1; cnti < Nx; cnti ++) {
      tensor[cnti] = tensor[cnti - 1] + Ny;
      tensor[cnti][0] = tensor[cnti - 1][0] + Ny * Nz;
      for(cntj = 1; cntj < Ny; cntj ++)
         tensor[cnti][cntj] = tensor[cnti][cntj - 1] + Nz;
   }

   return tensor;
}
```

Dinamički alocirane memorijske strukture navedene u listingu 4.2 ne oslobađaju alociranu memoriju po automatizmu. Da bi se to postiglo, potrebno je za svaku strukturu implementirati proceduru koja vrši oslobađanje odgovarajućih memorijskih blokova. Primer takve procedure za oslobađanje memorije alocirane trodimenzionalnom matricom je dat u listingu 4.3.

Listing 4.3: Oslobađanje memorije

```
void free_double_tensor(double ***tensor) {
   free((char *) tensor[0][0]);
   free((char *) tensor[0]);
   free((char *) tensor);
}
```

4.2 Implementacija serijskih algoritama

Implementacije serijskih verzija koda (serijski DFT program, serijski program za ekstrahovanje motiva i serijski CPM program) se u najvećem delu u potpunosti oslanjaju na algoritamske implementacije u poglavlju 3. Međutim, proces implementacije nije u svim slučajevima bio jednosmeran, u smislu da je unapred pripremljen algoritamski postupak doveo do programske implementacije koje u potpunosti odgovara očekivanjima. Postoje dva osnovna razloga za ovo. Prvi se odnosi na to što je implementacija algoritma, usled nepravilnosti u algoritamskom postupku, često dovodila do grešaka u rezultatima. Ovo je najčešće bio slučaj u implementaciji postupka za rešavanje gausijanskih integrala (sekcije 3.1.1–3.1.5). U slučajevima gde je bila moguća direktna provera rezultata, greške su relativno lako uočavane. Međutim, često je jedino bilo moguće osloniti se na intuitivne zaključke ili na posmatranje uticaja delova analiziranog koda na konačni rezultat. Drugi razlog zbog čega je implementacija koda ceo proces vraćala na unapređenje algoritma su nezadovoljavajuće performanse, odnosno nedovoljno dobra efikasnost izvršavanja koda. Unapređenja koja su bila rezultat opisanih procesa su u nekim slučajevima dovodila do redundantnih verzija delova koda. S obzirom da se radi o funkcionalnim segmentima koda oni su najčešće korišteni za poređenje tačnosti rezultata ili za poređenje performansi više različitih verzija implementacija.

Pored optimizacije algoritama, za unapređenje performanse su primenjivani i standardni postupci za optimizovanje koda. Ovi postupci uključuju upotrebu odgovarajuću tipova promenjivih i programskih struktura, odgovarajuće preciznosti zapisa promenjivih, adekvatnih računarskih operacija, optimalne alokacije memorije, itd. U implementacijama serijskih algoritama je najveća pažnja posvećena optimizaciji i preciznosti rezultata za izračunavanje integrala potrebnih za formiranje Hamiltonijan matrice iz jednačine (3.6), kao i na ubrzanje konvergencije računanja svojstvenih energija (sekcija 3.1.6), kod iterativnog DFT postupka.

Prethodno navedeno se najvećim delom odnosi na unapređenje algoritamske implementacije koja je detaljno obrazložena u poglavlju 3. Zbog toga, navođenje opisa programskih implementacija za sve pojedinačne korake ne bi doprinelo boljem razumevanje postupaka, pa će ono u najvećem delu biti izostavljeno.

Nešto drugačiji slučaj je implementacija za izračunavanja Hartri potencijala (sekcija 3.1.4), gde je implementirano nekoliko različitih verzija od kojih je najoptimalnija za izvršavanje ali algoritamski najkompleksnija ona koje se odnosi na korišćenje rekurentnih relacija. Za potpuno razumevanje implementacije ovog postupka je pored algoritamske implementacije iz sekcije 3.1.4 neophodno opisati i implementaciju koda.

4.2.1 Rekurentne relacije za implementaciju integrala Hartri potencijala

U ovom delu je opisana programska implementacija dela algoritma za rešavanje integrala Hartri potencijala uz pomoć rekurentnih relacija (sekcija 3.1.4). Iz aspekta implementacije najinteresantniji koraci su rekurzivne procedure za rekurentne relacije date u jednačinama (3.58) i (3.59), koje su redom opisane pseudokodom u listinzima 4.5 i 4.4.

Algoritamski koraci koji detaljno opisuju proceduru su dati u sekciji 3.1.4. Ukratko, redosled izvršavanja izračunavanja relacija je takav da se najpre inicira poziv rekurzivne procedure za računanje druge rekurentne relacije $(ab|c)$. Koreni rekurzivnih poziva ove procedure su izrazi $(a0|c)$, koji se dalje svode na pozive rekurzivne procedure za izračunavanje prve rekurentne relacije $[a0|c]^{(m)}$.

Dalje se rekurzivnim postupkom izraz $[ab|0]^{(m)}$ svede na izračunavanje primitivnih integrala $[00|0]^{(m)}$ (jednačina (3.61)).

Procedura $formRecRel2()$ za računanje druge rekurentne relacija (listing 4.4) za ulazne parametre ima stepene Gausijana centriranim na ova tri atoma, koordinate atoma A, B i C, Gausijane centrirane na atomima A, B i C, kao i niz unapred izračunatih primitivnih integrala. Primitivni integrali se računaju unapred da bi se eliminisalo ponavljanje njihovog računanja za iste kombinacije trojki Gausijana. Na ovaj način se dobija značajno ubrzanje algoritma, jer računanje ovih integrala u korenu prve rekurentne relacije zbog velikog broja poziva predstavlja najzahtevniji korak. U slučaju serijske verzije algoritma je ovo ponavljanje računanja moguće u potpunosti izbeći.

Procedura $formRecRel2()$ u sebi sadrži poziv ugnježdene rekurzivne procedure, koja za parametre ima promenjive komponente rekurzivnih poziva ($formRecRel2Impl()$), a to su stepeni Gausijana centrirani na atomima A i B. Uslov za koren rekurzivne relacije je da se stepeni Gausijana centrirani na atomu B svedu na nulu. Da bi se ove vrednosti svele na nulu, potrebno je vršiti rekurzivne pozive koji smanjuju (za jedan) stepene Gausijana centriranog na atomu B za sve tri prostorne koordinate. Za svaku prostornu koordinatu imamo po jedan rekurzivni poziv iz istog čvora. Istovremeno se, u svakom rekurzivnom koraku, vrednosti stepena Gausijana centriranog na atomu A, povećavaju za jedan.

Kada se postigne potrebni uslov, najpre se inicijalizuju sve potrebne promenjive i izračunaju vrednosti koje su potrebne za izračunavanje članova $(a0|c)$. Dalje se formiraju ugnježdene petlje koje 'prolaze' kroz sve kombinacije primitivnih Gausijana centriranim na atomima A, B i C, da bi se za svaku od tih trojki pozvala procedura za prvu rekurentnu relaciju $formRecRel1()$, odnosno procedura za izračunavanje $[a0|c]^{(m)}$. Rezultat svakog od ovih poziva procedure za izračunavanje prve rekurentne relacije se vraća u obliku sume. Rekurzivni pozivi koji računaju drugu rekurentnu relaciju ovu sumu koriste za izračunavanje ukupne vrednosti $(ab|c)$ koja se vraća kao konačan rezultat.

Listing 4.4: Pseudo kod implementacije druge rekurentne relacije ((3.59)) za računanje integrala Hartri potencijala

```
double formRecRel2(int ax, int ay, int az, int bx, int by, int bz, int cx, int cy, int cz,
      const struct nuclei A, const struct nuclei B, const struct nuclei C,
      const struct phi phiA, const struct phi phiB, const struct phi phiC,
      double **primInt){

   // Druga rekurentna relacija
   double formRecRel2Impl(int ax, int ay, int az, int bx, int by, int bz){
      double res = 0.;
      int primIntIndex = 0;

      // Uslov za koren druge rekurzvne relacije
      if(bx == 0 && by == 0 && bz == 0){
         //... Inicijalizacija promenjivih
         struct recIntegral1 ***recRel1 = alloc_recIntegral1_tensor(max_a, max_c, mMax);
         int setSize = max_a * max_c * mMax * sizeof(struct recIntegral1);

         int k;
         double N3[phiC.size];
         for(k = 0; k < phiC.size; k++){
            N3[k] = nCoef(phiC.l, phiC.m, phiC.n, phiC.t[k].alpha);
         }
```

```
        // Izračunavanje članova (a0|c)
        int i,j,m;
        double sum = 0.;
        for(i = 0; i < phiA.size; i++){
           double N1 = nCoef(phiA.l, phiA.m, phiA.n, phiA.t[i].alpha);
           for(j = 0; j < phiB.size; j++){
              double N2 = nCoef(phiB.l, phiB.m, phiB.n, phiB.t[j].alpha);
              for(m = 0; m < phiC.size; m++){
                 memset(**recRel1, 0, setSize);
                 // Poziv prve rekurentne relacije
                 double temp1 = formRecRel1(ax, ay, az, cx, cy, cz, A, B, C,
                       phiA.t[i].alpha, phiB.t[j].alpha, phiC.t[m].alpha,
                       recRel1, primInt, &primIntIndex, 0);
                 temp1 = N1 * N2 * N3[m] * temp1;
                 double temp2 = phiA.t[i].a * phiB.t[j].a * phiC.t[m].a * temp1;
                 sum = sum + temp2;
              }
              primIntIndex++;
           }
        }
        free_recIntegral1_tensor(recRel1);
        return sum;
     }
     // Rekurzivni pozivi
     if(bx > 0){
        res = formRecRel2Impl(ax+1, ay, az, bx-1, by, bz);
        if(A.x != B.x){
           res = res + (A.x - B.x) * formRecRel2Impl(ax, ay, az, bx-1, by, bz);
        }
     }
     else if(by > 0){/*...*/}
     else if(bz > 0){/*...*/}
     return res;
  }
  return formRecRel2Impl(ax, ay, az, bx, by, bz);
}
```

Drugi listing (4.5) opisuje proceduru koja implementira prvu rekurentnu relaciju (jednačina (3.58)). Procedura $formRecRel1()$ za ulazne parametre ima stepene Gausijana centriranih na ova tri atoma, koordinate atoma A, B i C, koeficijente primitivnih Gausijana centriranih na atomima A, B i C, tenzor strukturu $recRel1$ koji sadrži identifikator inicijalizacije vrednosti integrala kao i vrednost dobijenu računanjem integrala (opis dat u sekciji 4.1.1), kao i niz unapred izračunatih primitivnih integrala. Kao parametar se takođe prosleđuje inicijalna vrednost stepena m.

Procedura najpre inicijalizuje sve potrebne promenjive za izvršavanje rekurzivnih poziva a zatim vrši poziv ugnježdene rekurzivne procedure $formRecRel1Impl()$. Dalje se uz pomoć procedure $getRecRelIndex$ vraćaju indeksi koji uz pomoć stepena m određuju poziciju traženog integrala u tenzoru $recRel1$ koji se izračunava u datom pozivu rekurzivne funkcije. Ukoliko je vrednost već izračunata u prethodnim pozivima vraća se vrednost koja se čuva u ovoj strukturi. U drugom slučaju se dodeljuje odgovarajuća vrednost primitivnog integrala.

U poslednjem segmentu se najpre izračunavaju težišne vrednosti koje određuju koji skup jednačina se rešava u nastavku (jednačine (3.71) i (3.72)). Zatim se vrši provera da li kombinacije stepena Gausijana odgovaraju netrivijalnim rekurzivnim pozivima. Ukoliko je to slučaj, u okviru odgovarajuće grane se rekurzivnim pozivima rešava prethodno pomenuti skup jednačina. U korenu rekurzivnih poziva se dodeljuje konačna vrednost integrala akumulirana u promenjivoj res.

Listing 4.5: Pseudo kod implementacije prve rekurentne relacije ((3.58)) za računanje integrala Hartri potencijala

```
double formRecRel1(int ax, int ay, int az, int cx, int cy, int cz, const struct nuclei A,
      const struct nuclei B, const struct nuclei C, double alpha, double beta,
      double gamma, struct recIntegral1 ***recRel1, double **primInt,
      int *primIntIndex, int m){

   //... Inicijalizija promenjivih

   // Prva rekurentna relacija
   double formRecRel1Impl(int ax, int ay, int az, int cx, int cy, int cz, int m){
      double res = 0.;
      int a, c;
      a = getRecRelIndex(ax, ay, az);
      c = getRecRelIndex(cx, cy, cz);

      // Vraća se vrednost odgovarajućeg unapred izračunatog primitivnog integrala
      if(recRel1[a][c][m].taken == 0){
         if(ax == 0 && ay == 0 && az == 0 && cx == 0 && cy == 0 && cz == 0){
            double ivalue = primInt[*primIntIndex][m];
            recRel1[0][0][m].ivalue = ivalue;
            recRel1[0][0][m].taken = 1;
            return ivalue;
         }
      }else{
         return recRel1[a][c][m].ivalue;
      }

      if(ax > 0 || cx > 0){

         //... Izračunava se broj poziva funkcija: weight_ax, weight_cx
         // Vrše se odgovarajući rekurzivni pozivi
         if(ax > 0 && (weight_ax <= weight_cx || cx == 0)){
            if(b_px_Ax){
               res = res + px_Ax * formRecRel1Impl(ax-1, ay, az, cx, cy, cz, m);
            }
            if(b_wx_px){
               res = res + wx_px * formRecRel1Impl(ax-1, ay, az, cx, cy, cz, m+1);
            }
            if(ax-1 > 0){1
               res = res + ((double)(ax - 1) / (2. * alpha_beta)) *
                  (formRecRel1Impl(ax-2, ay, az, cx, cy, cz, m) -
                  (gamma / (alpha_beta + gamma)) *
                  formRecRel1Impl(ax-2, ay, az, cx, cy, cz, m+1));
            }
            if(cx > 0){
               res = res + ((double)cx / (2. *(alpha_beta + gamma))) *
                  formRecRel1Impl(ax-1, ay, az, cx-1, cy, cz, m+1);
            }
         }else if(cx > 0){
            //...
         }
      }else if(ay > 0 || cy > 0){
         //...
      }else if(az > 0 || cz > 0){
         //...
      }

      recRel1[a][c][m].ivalue = res;
      recRel1[a][c][m].taken = 1;
      return res;
```

```
    }

    return formRecRel1Impl(ax, ay, az, cx, cy, cz, m);
}
```

4.3 Paralelizacija CPM

Glavni motiv prilikom implementacije algoritma za CPM je želja da se uz pomoć njega uspešno izvršava izračunavanje za sisteme od više hiljada (do nekoliko desetina hiljada) atoma. Ukoliko težimo rešavanju CPM za velike sisteme, serijski CPM program koji je namenjen za rad na jednom procesorskom jezgru jednog personalnog računara ima dva ograničenja. Prvo je vremensko ograničenje (u sekciji koja opisuje performanse i validaciju serijskih algoritama [5.1] je prikazano da već za sisteme veće od 1000 atoma potrebno nekoliko desetina sati da bi se izvršili). Razlog za ovo je ograničenje brzine jednog CPU jezgra. Čak i ukoliko bi odlučili da prihvatimo ovaj nedostatak, drugi problem koji se ne može eliminisati je ograničenje kapaciteta radne memorije računara. Čak i pri maksimalnoj optimizaciji memorijske reprezentacije struktura podataka CPM-a, memorijski zahtevi višestruko prevazilaze raspoloživu operativnu memoriju današnjih personalnih računara.

Da bi se adresirali navedeni problemi, neophodno je da se implementira paralelna verzija koda uz odgovarajuću distribuciju podataka potrebnih za njegovo izvršavanje. Ovo je moguće jedino uz upotrebu paralelne računarske arhitekture gde je zbog prirode problema najpogodnija računarska arhitektura sa distribuiranom memorijom. U ovakvom okruženju je moguće pokretati paralelne programske procese koji međusobno komuniciraju u cilju razmene podataka. Ovakvu arhitekturu predstavljaju računarski klasteri sastavljeni od pojedinačnih računara, odnosno računarskih nodova, međusobno povezani računarskom mrežom. Alociranjem većeg broja računara se povećavaju raspoloživi procesorski i memorijski resursi ali se takođe povećava i komunikacija među procesima.

Jedan od protokola za paradigmu distribuirane memorije koji ima najrasprostranjeniju primenu je Message Passing Interface (MPI) [105]. MPI je standard koji predstavlja specifikaciju interfejsa biblioteka namenjenih za procese razmene podataka na arhitekturama sa distribuiranom memorijom. Postoji nekoliko implementacija MPI-a, od kojih su većina iz grupe softvera otvorenog koda. Implementacije koje imaju najširu upotrebu i koje se nalaze na većini računarskih klastera su Open-MPI [1] i MPICH [2].

Implementacija paralelne verzije serijskog CPM koda na distribuiranoj računarskoj arhitekturi ima i svoju cenu. Za razliku od sekvencijalnog izvršavanja, distribucija podataka među klasterskim nodovima, u svrhu prevazilaženja ograničenja memorijskih resursa, iziskuje dodatnu komunikaciju među paralelnim procesima. Iz tog razloga je posebno važno voditi računa o načinu distribucije podataka i implementaciji procesa koji imaju potrebu za pristup tim podacima. Takođe, same izmene i dopune koda kojima se implementira paralelna verzija algoritama CPM-a predstavljaju složen zadatak.

U nastavku ovog poglavlja je dat opis proširenja algoritama iz prethodnog poglavlja, koje uz odgovarajuću šemu distribucije podataka omogućava uspešnu upotrebu sistema sa distribuiranom memorijom. Implementacija ovih algoritama je urađena uz pomoć MPI-a. Najpre je u sekciji 4.3.1 opisan proces koji je doveo do optimalnog rešenja problema distribucije podataka, čime je

omogućena relativno jednostavna implementacija paralelne verzije integrala preklapanja (3.1.1), kinetičkog integrala (3.1.2) kao i procedure za računanje integrala efektivnog potencijala jezgra (3.1.3) data u poglavlju 4.3.3. Nakon toga su opisani značajno složeniji procesi procedure za učitavanje i fitovanje motiva (4.3.4), integrala Hartri potencijala (4.3.5) i izmensko-korelacionog integrala (4.3.6). Na kraju poglavlja je dat opis implementacije paralelne verzije procedure za dijagonalizaciju (4.3.7).

4.3.1 Distribucija podataka

Pri distribuciji podataka se prvenstveno vodi računa o raspoređivanju podataka koji su neophodni za izvršavanje pojedinačnih paralelnih procesa, a koji pri tom imaju značajno memorijsko zauzeće. Ovakvi podaci su u C kodu uglavnom reprezentovani uz pomoć jednodimenzionalnih, dvodimenzionalnih i trodimenzionalnih nizova za koje se najčešće redom koristi naziv vektor, matrica i tenzor. Opis i način alokacije memorije za pomenute strukture podataka se može naći u sekciji 4.1.2.

Podaci koji pri alokaciji memorije nemaju veliko zauzeće, tj. nisu memorijski zahtevni, se mogu čuvati kao kopije na svim paralelnim procesima. Ovo je najčešće mnogo optimalnije rešenje od onog koje bi zahtevalo komunikaciju među procesima radi razmene takvih podataka. Međutim, u paralelnoj implementaciji CPM postoji čitav niz različitih grupa podataka koje je neophodno distribuirati različitim paralelnim procesima. Među njima su elementi Hamiltonijana i elementi integrala preklapanja, koji formiraju matrice H i S, i čije formiranje predstavlja neophodan korak za rešavanje generalizovanog svojstenog problema predstavljenog u jednačini (3.6). Ove matrice osim što imaju posebnu važnost iz algoritamskog aspekta, takođe predstavljaju grupe podataka koji imaju veliku memorijsku zahtevnost. Alokacija memorije za ove dve matrice se vrši na samom početku izvršavanja programa. Matricu S čine vrednosti integrala preklapanja (3.7), dok svaki element matrice H čini zbir vrednosti svakog od četiri integrala opisanih jednačinama (3.8), (3.9), (3.10) i (3.11). Čuvanje vrednosti ovih matrica je neophodno do samog kraja izvršavanja programa, odnosno do dijagonalizacije sistema jednačina 3.2.3. Vizuelni prikaz jedne takve matrice je dat na slici 4.1.

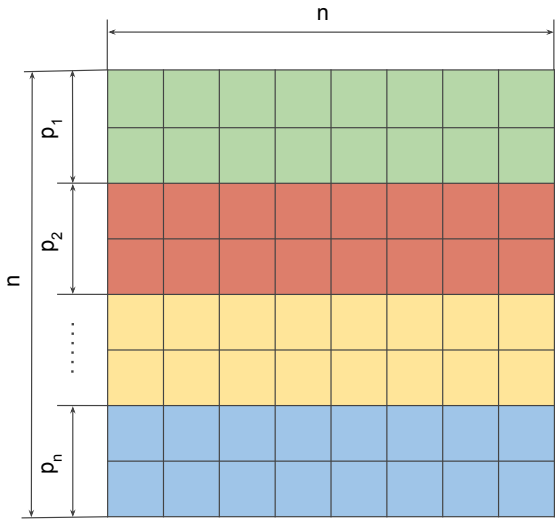
S obzirom da su vrednosti dobijene rešavanjem odgovarajućih integrala ((3.7), (3.8), (3.9), (3.10), (3.11)) realni brojevi, za njihovu reprezentaciju u matricama H i S se koristi format pokretnog zareza sa dvostrukom tačnošću (double). Tako je za zapis svakog pojedinačnog elementa ovih matrica po standardu IEEE 754 potrebno 8 bajtova (64 bita). Ovaj tip podataka svojim opsegom u potpunosti zadovoljava potrebe preciznosti zapisa realnih brojeva kako u slučaju H i S matrica tako i u slučaju svih ostalih procesa CPM-a.

Veličina ovih matrica zavisi od broja atoma koji su definisani ulaznim sistemom i od odabranog skupa Gausijana. Da bismo dali uvid u veličinu matrica, za primer ćemo uzeti klasu sistema koje čine dvodimenzionalni nizovi tiofen oligomera $(C_{4k}S_kH_{2k+2})_l$ koji sadrže atome vodonika, ugljenika i sumpora (pogledati sliku 5.1), gde k određuje veličinu jednog lanca, a l broj lanaca. Za bazisni skup Gausijana biramo SBKJC VDZ ECP [86, 87, 53, 54] koji se koristi u numeričkom primeru u poglavlju 5. Za ovu klasu se koristi prototip sistem sa tri lanca gde je svaki lanac dimenzije tri $(C_{12}S_3H_8)_3$, što daje sistem koji ima ukupno 69 atoma. U odabranom bazisu skup Gausijana atoma C i S je veličine 8, dok je skup za H veličine 2. Ukoliko sada posmatramo ceo sistem koji ukupno ima 36 C atoma, 9 S atoma i 24 H atoma, dimenzija kvadratnih matrica je 408×408 $(36 * 8 + 9 * 8 + 24 * 2)$. Dakle, u slučaju pomenutog sistema, broj elemenata matrica H i S je

166464 ($408 * 408$) gde je za svaki od elemenata potrebno 8 bajtova, što znači da je za ukupnu veličinu memorijske reprezentacije jedne matrice potrebno 1.33MB. Ako skaliramo ovaj prototip sistem na sistem veličine 38 lanaca, gde je svaki lanac dimenzije 38, dobijamo sistem sa ukupno 10184 atoma za koji su potrebne matrice dimenzije 63688×63688. Ukoliko primenimo isti račun kao za prethodni primer, za ovu matricu bi ukupno trebalo alocirati oko 32.4GB memorije. Već sa sistemom ove veličine, sa ovakvom alokacijom memorije je jasno da istovremeno postojanje obe matrice H i S u operativnoj memoriji prevazilazi današnje memorijske kapacitete jedne računarske instance. Svakako bi još trebalo uzeti u obzir da ovo nisu jedine strukture podataka u kodu koje opterećuju memorijske resurse.

Blokovi jednake veličine

U prethodnom primeru je pokazano da postoji jasna potreba da se izvrši raspodela elemenata matrice među procesima. Najčešća praksa prilikom distribucije je da se formiraju blokovi istih veličina (jednak broj elemenata u svakom bloku). Ovi blokovi mogu biti kvadratni ($m \times m, m < n$), mogu formirati vertikalnu raspodelu ($m \times n, m < n$), horizontalnu raspodelu ($n \times m, m < n$) ili neki drugi tip raspodele. U svim navedenim slučajevima m zavisi od broja procesa, odnosno $m = n \div ncore$. Za potrebe CPM algoritma je najpogodnije da blokovi sadrže kompletne redove matrica, odnosno da je najbolje matricu podeliti u horizontalne blokove kao na slici 4.1. Procenjeno je da se ovakvom raspodelom vrši minimalna komunikacija među procesima kao i da takva raspodela daje minimalno opterećenje prilikom indeksiranja elemenata kod čitanja i pisanja vrednosti. Detaljnije obrazloženje ove procene se može naći u sekcijama 4.3.3, 4.3.4, 4.3.5, 4.3.6 i 4.3.7.

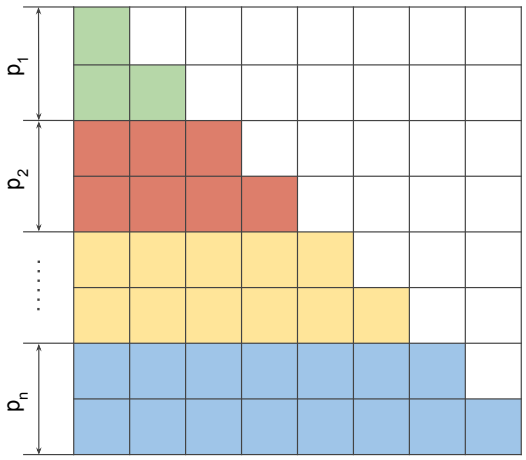


Slika 4.1: Distribucija elemenata kvadratnih matrica H i S sistemom horizontalnih blokova istih dimenzija (matrice H i S imaju iste dimenzije i njihovi elementi su podaci istog tipa).

Trougaona matrica

Sledeća važna osobina matrica H i S je da su one simetrične matrice, odnosno da za svako i i j ($0 < i < n$, $0 < j < n$) važi da je $H_{i,j} = H_{j,i}$ i $S_{i,j} = S_{j,i}$. Radi efikasnog iskorišćenja memorijskog

prostora u ovom slučaju je umesto kvadratne matrice potrebno formirati trougaonu matricu (kao što je prikazano na slici 4.2). U slučaju trougaone matrice, broj elemenata i njegov raspored se može posmatrati kroz aritmetičku progresiju, tako da ukoliko imamo matricu dimenzije $n \times n$ ukupan broj elemenata je $N = \frac{n(n+1)}{2}$.

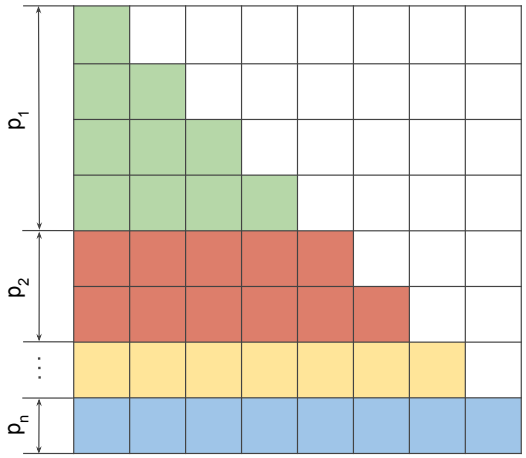


Slika 4.2: Prikaz donje trougaone matrice uz primenu distribucije elemenata sistemom horizontalnih blokova.

Ako sada pogledamo novodobijeni raspored elemenata na slici 4.2, jasno je da je narušen početni zahtev a to je da želimo brojčano jednaku raspodelu elemenata po blokovima. Da bismo postigli balans, u svakom bloku je potrebno da imamo približno $N \div ncore$, gde je $ncore$ ukupan broj procesa. Da bismo prevazišli ovaj problem, potrebno je adresirati ga na jedan od dva potencijalno pogodna načina. Prvi bi bio da svaki blok 'zauzme' odgovarajući broj redova trougaone matrice tako da zadovolji balans, uz uslov da je blok sastavljen od određenog broja uzastopnih redova matrice. Drugi način bi bio da svaki blok bude sastavljen od dve 'odvojene' grupe redova, i to tako da se za prvi blok uzastopno biraju redovi sa gornje i sa donje strane matrice, drugi sa gornje i donje strane matrice sastavljene od preostalih redova, sve dok poslednji blok ne bi 'pokrio' skup preostalih redova sa sredine matrice. Naravno, uslov je da je zadovoljen uslov da je napravljena jednaka raspodela elemenata među formiranim blokovima. Na ovaj način bi svaki blok imao isti broj redova, što se može jednostavno proveriti na osnovu aritmetičke progresije. Za oba slučaja važi da se redovi uvek dodeljuju blokovima kao celi, tj. tako da svi elementi u jednom redu matrice pripadaju istom bloku. Prvi način zahteva određene matematičke operacije koje bi formirale blokove, kao i određeno mapiranje tih blokova zbog različitog broja redova u njima. Drugi način prevazilazi oba navedena problema, međutim činjenica da se svaki blok mora razdvojiti na dve logičke a samim tim memorijske celine, predstavlja veliki nedostatak. Iz tog razloga se opredeljujemo za prvi način koji je prikazan na slici 4.3.

Blokovi matrice se formiraju postupno, počevši od prvog bloka koji započinje prvim redom matrice i završava se redom za koji uzimamo oznaku k_0. S obzirom da je zbir elemenata određen aritmetičkom progresijom kao i da se ukupan broj elemenata koji čine jedan blok određuje na osnovu broja paralelnih procesa, važi sledeća jednakost:

$$1 + 2 + 3 + ... + k_0 = \frac{n(n+1)}{2ncore}. \tag{4.1}$$



Slika 4.3: Prikaz distribucije gde su blokovi različitih dimenzija i sadrže približno jednak broj elemenata.

Na osnovu jednačine (4.1) dalje dobijamo red k_0 matrice koji čini marginu prvog bloka:

$$\frac{k_0(k_0+1)}{2} = \frac{n(n+1)}{2ncore},$$
$$k_0 \approx \frac{n}{\sqrt{ncore}}. \tag{4.2}$$

Razlog zbog čega se za k_0 se uzima približna vrednost je taj što se time pojednostavljuje računanje a greška koja se time pravi (za veće sisteme) može u najgorem slučaju dovesti do promene veličine bloka za jedan red, što predstavlja zanemarljiv faktor.

Na osnovu određenog prvog bloka matrice je moguće formirati drugi blok, tako što ćemo odrediti sledećih k_1 redova koji mu pripadaju. Najpre sa b označimo balans, odnosno broj elemenata koji težimo da imamo u svakom bloku:

$$b = \frac{n(n+1)}{2ncore}. \tag{4.3}$$

Drugi blok čine redovi k_0+1,k_0+2,... k_0+k_1, a zbir broja njihovih elemenata bi trebao da bude jednak balansu b:

$$k_1 k_0 + \frac{k_1(k_1+1)}{2} = b. \tag{4.4}$$

Dalje, rešenjem date kvadratne jednačine

$$k_1^2 + (2k_0+1)k_1 - 2b = 0, \tag{4.5}$$

dobijamo traženi broj redova drugog bloka:

$$k_1 = -\frac{1}{2} - k_0 + \sqrt{k_0^2 + k_0 + \frac{1}{4} + 2b}. \tag{4.6}$$

Na isti način formiramo i sve preostale blokove, tako što uopštimo prethodni slučaj uzimajući za k_{i-1} broj redova prethodnog bloka i za k_i broj redova i-tog bloka, gde je $1 < i <= ncore$, pa se izraz (4.4) može napisati u sledećem obliku:

$$k_i(k_0 + k_1 + \cdots + k_{i-1}) + \frac{k_i(k_i+1)}{2} = b. \tag{4.7}$$

Rešavanjem prethodne jednačine, dolazimo do konačnog rešenja za broj redova i-tog bloka, odnosno do uopštenja jednačine (4.6):

$$k_i = -\frac{1}{2} - (k_0 + k_1 + \cdots + k_{i-1}) + \sqrt{(k_0 + k_1 + \cdots + k_{i-1})^2 + (k_0 + k_1 + \cdots + k_{i-1}) + \frac{1}{4} + 2b}. \quad (4.8)$$

U listingu 4.6 je data implementacija opisanog metoda koja inicijalizuje niz *blocks*, čija veličina odgovara ukupnom broju procesa. Svaki elemenat niza predstavlja indeks poslednjeg reda matrice koji pripada odgovarajućem bloku. Prvi elemenat se dobija uz pomoć jednačine (4.2), dok se ostali elementi računaju na osnovu jednačina (4.3). Na taj način se na osnovu razlike vrednosti elementa tekućeg procesa i vrednosti prethodnog elementa niza *blocks* dobija veličina bloka koji odgovara tekućem procesu ($matrixSizeDist$).

Listing 4.6: Implementacija algoritma za podelu matrice na blokove sa jednakim brojem elemenata

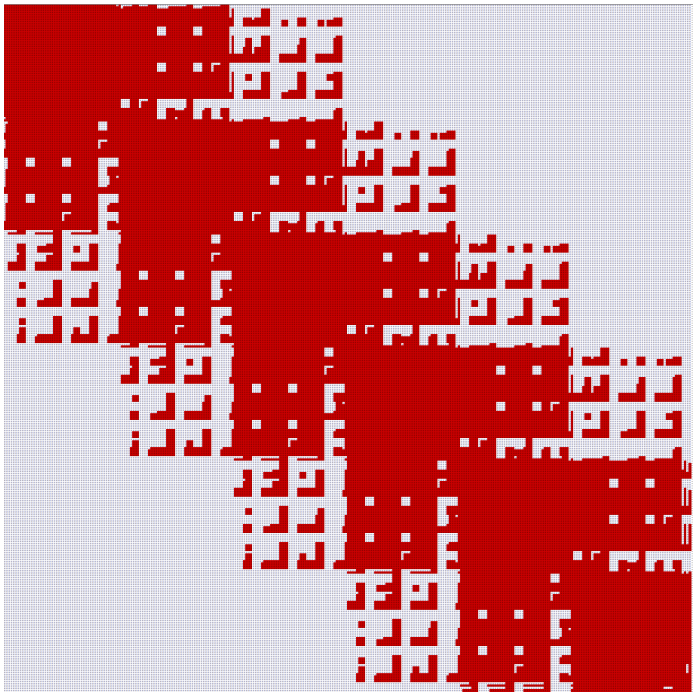
```
unsigned int blocks[ncore];
blocks[0] = (matrixSize / round(sqrt(ncore)));

if(ncore > 1){
   int i, x;
   unsigned long numEl = (unsigned long)(matrixSize)*((unsigned long)(matrixSize+1))/2;
   unsigned int balance = numEl / ncore;
   for(i = 1; i < p-1; i++){
      blocks[i] = blocks[i-1] +
      (int)(-0.5 - blocks[i-1] + sqrt(blocks[i-1] * blocks[i-1] +
      blocks[i-1] + 0.25 + 2 * balance)) + 1;
   }
   blocks[ncore-1] = matrixSize;
}
int matrixSizeDist;
if(my_rank == 0){
   matrixSizeDist = blocks[my_rank];
}else{
   matrixSizeDist = blocks[my_rank] - blocks[my_rank-1];
}
```

Struktura elemenata matrice

Sledeće što je potrebno uraditi je analiza samih elemenata koji čine blokove matrica. Pošto je svaki element matrice određen indeksima odgovarajućih Gausijana i da su funkcije Gausijana dobro lokalizovane u prostoru, potrebno je primeniti proceduru odsecanja pri računanju integrala opisanu u sekciji 3.2.2. Na osnovu ovoga je moguće eliminisati računanje određenog broja integrala koji se odnose na parove Gausijana koji su dovoljno udaljeni jedni od drugih, čime bi određen broj elemenata matrice umesto zanemarljivo malih vrednosti imao nule. Kriterijum za udaljenost se dobija unapred definisanim maksimalnim poluprečnikom Gausijana. S obzirom na fizičku osobinu sistema, povećanjem broja atoma se povećava i broj parova za koje važi uslov da su međusobno dovoljno udaljeni. Ovo znači da se povećanjem sistema povećava i efikasnost odsecanja, čime dolazimo do još jedne važne osobine matrica H i S, a to je da su one retke.

Na slici 4.4 je dat primer primene odsecanja pri računanju integrala za tiofen oligomer. Radi se o relativno malom sistemu od 69 atoma za koji se formiraju matrice H i S dimenzije 408×408. Crvenom bojom su obeleženi svi elementi za koje je potrebno računanje odgovarajućih integrala, dok

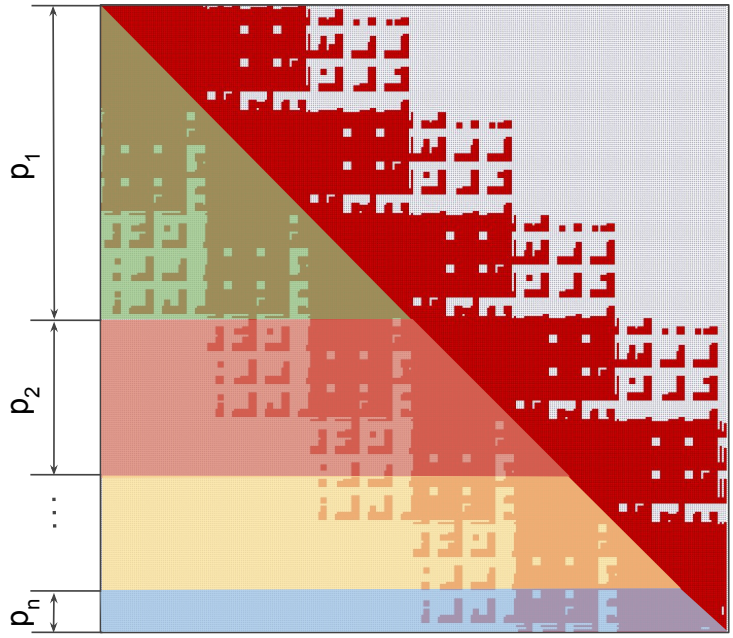
su belom bojom označeni nula elementi matrice gde se računanje može eliminisati. Ono što se može zapaziti je da se ne-nula elementi formiraju oko dijagonale matrice i da postoji određeno ponavljanje njihovog rasporeda. Razlog ovome leži u načinu na koji su raspoređeni atomi u ulaznom sistemu i u redosledu po kome se oni učitavaju u programu. U navedenom primeru redosled Gausijana koji formira matricu odgovara prostornom rasporedu atoma u ulaznom fajlu i to tako što je najpre unet jedan lanac počevši od atoma sa njegovog kraja, pa su zatim redom unošeni najbliži susedi sve do poslednjeg atoma u lancu. Nakon toga su po istom principu uneta preostala dva lanca. Otuda je razlog grupisanja oko dijagonale taj što su susedi učitani po redosledu koji odgovara prostornim koordinatama, dok su ponavljajuće grupe duž dijagonale zapravo određene oligomerskim lancima sa istim prostornim rasporedom atoma. Povećanjem sistema bi efekat grupisanja oko dijagonale postajao sve više izražen a izgled tog 'pojasa' bi bio određen odnosom veličine lanaca u odnosu na ukupan broj atoma u sistemu.



Slika 4.4: Primena odsecanja računanja integrala nad tiofen oligomerom $(C_{12}S_2H_8)_3$ za koje se formira matrica dimenzije 408×408. Crvenom bojom su označeni elementi matrice za koje se računaju odgovarajući integrali, dok su belom označeni nula elementi matrice.

Ukoliko mapiramo matricu odsecanja na odgovarajuće elemente matrice H i S i pri tom uzmemo u obzir prethodno opisanu distribuciju gde se postiže balans pri rasporedu elemenata (slika 4.3), dobijamo blokove kao na slici 4.5. Ono što se može videti je da postoji značajna razlika u popunjenosti blokova. Iako postoji dobar balans elemenata matrice, njime nije uzet u obzir raspored ne-nula elemenata dobijen na osnovu odsecanja.

Ovde je važno napomenuti da ukupan broj ne-nula elemenata zavisi od klase ulaznog sistema, a da njihov raspored u matricama zavisi isključivo od načina na koji je formiran redosled atoma u ulaznom fajlu. Dakle, prikaz iz primera bi se mogao značajno razlikovati odabirom druge klase sistema ali bi se takođe mogao značajno razlikovati ukoliko bismo drugačije rasporediti učitane atome, odnosno Gausijane koji su na njima centrirani. Ono što svakako nije poželjno je da efikasnost



Slika 4.5: Primer distribucije (slika 4.3) nad elementima matrice H i S sa kriterijumom odsecanja računanja integrala (slika 4.4) kod tiofen oligomera $(C_{12}S_2H_8)_3$.

izršavanja programa zavisi od načina na koji se organizuju ulazni podaci. Naravno, postoji i opcija da se odgovornost prebaci na korisnika time što bi se definisala pravila za pripremu ulaznih podataka radi povećanja optimalnosti izvršavanja programa. Prva negativna strana ovakve eventualne odluke je što definisanje univerzalnog pristupa za sve klase sistema predstavlja zahtevan zadatak, a drugo je što smatramo da bi korisnik trebalo da bude oslobođen svih pripremnih operacija koje se mogu rešiti implementacijom.

Randomizacija ulaznih podataka

S obzirom da je neophodno adresirati ovaj problem, potrebno je razmotriti nekoliko opcija. Jedan od načina bi bio da se matrice H i S i sve programske strukture koje zavise od njihovih indeksa čuvaju kao guste, a da se indeksi elemenata guste matrice koji ih mapiraju na početne matrice čuvaju u posebnim vektorima. Pod gustim matricama, i strukturama podataka generalno, se podrazumeva samo čuvanje popunjenih (ne-nula) elemenata. Ovo predstavlja dobro rešenje iz aspekta memorijskog zauzeća, međutim indeksiranje i pristupanje tako organizovanim podacima iziskuje velike izmene kako u implementaciji odgovarajućih algoritama tako i u implementaciji koda. Pored toga, takav način indeksiranja značajno povećava složenost izvršavanja zahtevnih delova koda, čime se gubi na efikasnosti pojedinih procesa. Drugi način bi podrazumevao implementaciju algoritma za preraspodelu elemenata matrice prema uslovu za eliminaciju nepotrebnih parova indeksa. Ovo bi podrazumevalo novo-formiranu matricu sa dobro raspoređenim elementima, koja bi morala da ima i propratne vektore sa indeksima koji bi mapirali početnu matricu. U ovom slučaju bi pored procesa koji bi raspoređivao elemente matrice, ponovo trebalo imati i propratne programske strukture za mapiranje, što prenosi problem iz prvog rešenja. Treći način bi bio da se uradi određeno preprocesi-

ranje učitanog ulaznog sistema i u zavisnosti od njegove klase napravi takav raspored Gausijana koji bi doveo do izbalansirane matrice. Ovim se rešava problem operacija nad već formiranim podacima iz drugog rešenja, kao i mapiranje podataka iz oba prethodna rešenja, međutim ovo rešenje zahteva da se razvije odgovarajući algoritam koji bi bio sposoban da adekvatno raspoređuje Gausijane za sve potencijalne klase sistema, što samo po sebi predstavlja veliki izazov.

Imajući u vidu sve navedene ideje, jasno je da one dovode do umanjenja efikasnosti algoritma i da iziskuju dodatne napore za implementaciju. S druge strane, uzimajući sve navedeno u obzir, postoji rešenje koje na vrlo elegantan način prevazilazi većinu navedenih problema. Ono ne podrazumeva nikakve direktne izmene nad ulaznim podacima kao ni promene u implementaciji postojećih procesa. Jednostavno, rešenje je da se uradi randomizacija atoma, odnosno da se iz uređenog rasporeda ulazne liste atoma formira raspored slučajnim odabirom njihovog redosleda. Kako se atomi u C strukturi koja određuje molekul identifikuju na osnovu jedinstvenog broja (indeksa) dovoljno je na osnovu veličine molekula formirati slučajni redosled liste indeksa, kao u listingu 4.7.

Listing 4.7: Randomizacija liste atoma iz ulaznog sistema

```
void shuffleArray(int *array, int n){
   srand ( time(NULL) );
   if (n > 1)
   {
      int i;
      for (i = 0; i < n - 1; i++)
      {
         int j = i + rand() / (RAND_MAX / (n - i) + 1);
         int t = array[j];
         array[j] = array[i];
         array[i] = t;
      }
   }
}
```

Dalje je dovoljno da se svakom atomu, učitanom iz ulaznog fajla, redom dodeljuju elementi iz novo-formirane liste indeksa. Na taj način dobijamo raspored indeksa skupova Gausijana u matricama H i S, koji ne zavisi od redosleda atoma definisanog u ulaznom sistemu. Slučajni izbor broja se oslanja na *rand*() (C funkciju iz stdlib.h biblioteke) koja vraća pseudo-slučajni broj iz opsega od 0 do maksimalnog broja *RAND_MAX* koji u implementaciji C99 standarda ima vrednost 2147483647. S obzirom da je veličina sistema značajno manja od *RAND_MAX*, pseudo-slučajni broj dobijen ovom procedurom ima zadovoljavajući rezultat.

Treba napomenuti da ovo ne podrazumeva randomizaciju Gausijana, već da grupa Gausijana centrirani nad istim atomom ostaje u 'paketu'. Tako se može reći da je novo-formirana lista atoma zapravo lista grupa Gausijana formirana slučajnim odabirom redosleda, gde jednu grupu povezuje prostorna pripadnost atomu. Ovo potencijalno ograničenje je moguće prevazići randomizacijom samih Gausijana (bez obzira na njihovu vezu sa atomima), međutim to dovodi do dodatnih izmena u implementaciji a pri tome bi to imalo značaj samo za manje sisteme. Kod velikih sistema, veličina skupa Gausijana koji su centrirani nad jednim atomom je značajno manja u odnosu na dimenzije matrice, pa uticaj na efikasnost usled ovakvog načina raspodele elemenata u tom slučaju nije značajan.

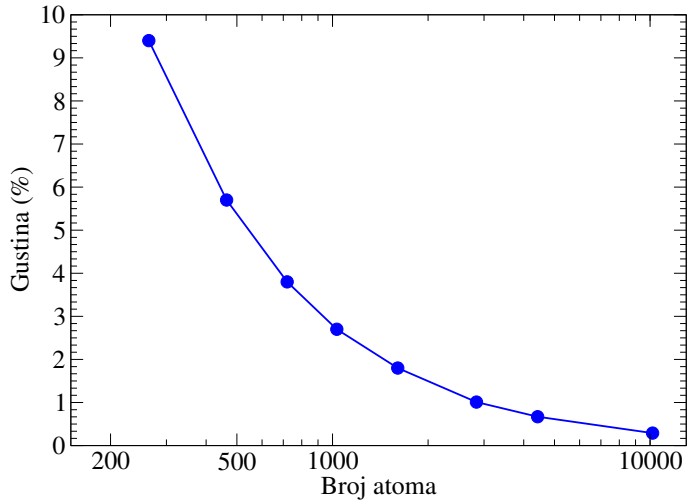
4.3.2 Retke matrice

Osobina matrica H i S da su retke je posledica kriterijuma za odsecanje računanja integrala opisanog u 3.2.2 i ta osobina je u prethodnoj sekciji obrađena iz aspekta ravnomernog balansa ne-nula elemenata među paralelnim procesima. Ovo znači da matrice H i S sadrže određeni broj elemenata koji čine sumu rešenja odgovarajućih integrala, kao i elemente koji sadrže inicijalne (nula) vrednosti i koji nemaju ulogu u programskim procesima. Da bi se odredilo kojoj od dve grupe svaki od elemenata pripada, potrebno je proveriti kriterijum za odsecanje integrala. Da bismo izbegli proveru uslova (3.119) pri svakom ponovljenom pristupu ovim elementima, potrebno je uvesti posebnu matricu koja za svaki elemenat matrica H i S sadrži logičku vrednost koja predstavlja rezultat zadovoljenja pomenutog uslova. U daljem tekstu će se koristiti izraz matrica odsecanja. Elementi matrice odsecanja koji odgovaraju odgovarajućim elementima matrica H i S imaju iste indekse i može se reći da matrica odsecanja mapira elemente matrica H i S.

Ono što je još potrebno adresirati je činjenica da se u memoriji čuvaju svi elementi koji imaju nula vrednosti, što svakako ne predstavlja optimalan način rada sa podacima. Treba napomenuti da matrice H i S nisu jedine u CPM algoritmu koje imaju ovu osobinu ali s obzirom da one utiču na gotovo sve procese, njima je posvećena cela sekcija. Druge strukture podataka koje su retke a pri tom imaju značajno memorijsko zauzeće će biti opisane u narednim sekcijama. U ovu grupu takođe spada i matrica odsecanja, jer i sama takođe predstavlja retku matricu.

Elementi matrica H i S čije se računanje može eliminisati svakako nisu neophodni za procese CPM-a, pa bi trebalo pronaći adekvatno rešenje za njihovu eliminaciju. Za veće sisteme se može reći da je broj ne-nula elemenata približno jednak $N \cdot k$ (gde je N broj atoma a k konstanta koja ne zavisi od veličine sistema). Skaliranje opadanja gustine u odnosu na veličinu sistema bi se moglo predstaviti sa $\sim k/N$. Na slici 4.6 je prikazan uticaj veličine sistema na gustinu matrica. Za primer je uzeta klasa sistema koju čine nizovi tiofen oligomera $(C_{4n}S_nH_{2n+2})_m$ gde je evidentno da sa povećanjem broja atoma dolazi do većeg broja parova Gausijana koji su dovoljno međusobno udaljeni i nad kojima bi računanje integrala moglo biti eliminisano. Tako za sistem tiofen oligomera dimenzije (n,m)=(6,6) sa 264 atoma imamo popunjenost od 9.4%, dok je popunenost sistema (n,m)=(38,38) sa 10184 atoma svega 0.3%. Za različite klase sistema se koeficijent krive smanjenja gustine elemenata može značajno razlikovati. Ovo prvenstveno zavisi od dimenzionalnosti sistema, tako što bi najveću gustinu zbog prostorne raspoređenosti atoma imali trodimenzionalni sistemi (npr. silicijumski nanokristali pod (f) na slici 5.1), dok bi najmanju gustinu formirali jednodimenzionalni sistemi koji imaju strukturu lanca (npr. (a)–(e) na slici 5.1).

Dalje bi trebalo analizirati koji su adekvatni načini čuvanja podataka retkih struktura za potrebe CPM-a, kolike su memorijske uštede koje bi se postigle njihovom primenom i koliko bi to uticalo na efikasnost algoritma. Optimalno rešenje bi podrazumevalo minimalno moguće memorijsko zauzeće svih navedenih struktura uz maksimalno očuvanje postojećeg načina implementacije procesa i uz posebnu brigu o performansama prilikom njihovog izvršavanja. Postoje najčešće dva razloga zbog čega se podaci čuvaju u gustim strukturama: ušteda memorijskog prostora i optimizacija vremena računanja. U našem slučaju zahvaljujući postojanju matrice odsecanja, brzina pristupa ne-nula elementima je već maksimalno optimizovana i ne postoji prostor za poboljšanje efikasnosti, jer se pristup svodi na jednu logičku proveru vrednosti za odgovarajuće indekse. Ono što nas zanima i što predstavlja glavno ograničenje za testiranje zaista velikih sistema je ušteda u memorijskom prostoru. U prethodnoj sekciji je kod opisa matrica H i S dat primer gde je izračunato memorijsko



Slika 4.6: Zavisnost gustine matrica H i S od veličine sistema kod nizova tiofen oligomera $(C_{4n}S_nH_{2n+2})_m$.

zauzeće matrice za tiofen oligomer sa 10184 atoma i ono iznosi 32.4GB. Ukoliko bismo umesto svih elemenata čuvali samo ne-nula elemente, uzimajući u obzir izračunatu popunjenost za ovaj sistem, koja iznosi 0.3%, zauzeće globalne matrice bi se svelo na svega 97.2MB. Ovo se naravno odnosi na idealan slučaj, gde bi se čuvali samo podaci bez pratećih struktura. Odnosi u memorijskom zauzeću retkih i gustih matrica za sisteme različitih veličina su dati u tabeli 4.1.

$(C_{4n}S_nH_{2n+2})_m$ (n,m)	(6,6)	(8,8)	(10,10)	(12,12)	(15,15)	(20,20)	(25,25)	(30,30)	(38,38)
broj atoma	264	464	720	1032	1605	2840	4425	6360	10184
gustina (%)	9.5	5.7	3.8	2.7	1.8	1.0	0.7	0.5	0.3
guste H,S	20.7	64.9	157.7	327.4	793.6	2500	6094	12621	32449
retke H,S (MB)	1.9	3.7	6.0	8.9	14.1	25.5	40.9	60.4	96.4
(1) (MB)	2.6	8.1	19.7	40.9	99.2	312.6	761.7	1577	4056
(2) (MB)	0.2	0.5	0.7	1.1	1.8	3.2	5.1	7.5	12.1

Tabela 4.1: Uporedni prikaz gustina, zauzeća memorije retkih i gustih H, S matrica kao i guste (1) i retke (2) matrice odsecanja za različite dimenzije tiofen oligomer lanaca.

U daljem procesu bi matrice H i S trebalo posmatrati uvek u odnosu na matricu odsecanja, jer način na koji se formiraju guste strukture za H i S zavise upravo od matrice odsecanja. Najpre ćemo posmatrati matricu odsecanja u neizmenjenom obliku, gde imamo veliku retku matricu sa indeksima svih ne-nula elemenata matrica H i S.

Počnimo od idealnog slučaja, a to je da za H i S čuvamo samo podatke bez ikakvih pratećih vektora. Očigledno da u tom slučaju nije moguće direktno mapiranje indeksa iz matrice odsecanja elementima H i S. Podsetimo se sada da su elementi matrica H i S rezultat pojedinačnih procesa računanja integrala. Ono što je pogodno je da redosled njihovog računanja, u smislu dodeljivanja njihovih vrednosti elementima matrica H i S, nije bitan. Ovo je posebno važno u procesu distribucije računanja integrala. Ukoliko bi se odrekli ovog komoditeta, prilagođavanjem odgovarajućih procesa se može dovesti do toga da redosled pristupa elementima bude svuda isti. Ako se ovo postigne, to

bi značilo da mapiranje indeksa postaje suvišno, i da se zaista može postići idealan slučaj, gde bi se čuvale samo vrednosti. Ovo bi zaista i bilo moguće ukoliko je jedina operacija sa elementima ove matrice smeštanje odgovarajućih vrednosti nakon računanja pojedinačnih integrala. Međutim, u kasnijem opisima procesa paralelizacije računanja integrala, biće jasno da je ovaj proces složeniji i da ovako koncipirane strukture podataka ne zadovoljavaju potrebe. Ono što je potrebno je da se dođe do kompromisnog rešenja koje bi bilo univerzalno primenjivo u svim procesima.

Iz aspekta optimizacije memorijskog prostora, logično rešenje bi bilo zadržati matrice H i S u predloženom obliku, tako da sadrže samo vrednosti u gustoj strukturi. U ovom slučaju nam je potrebna tako organizovana struktura matrice odsecanja, koja se oslanja jedino na redosled elemenata u gustim matricama H i S i koja mapira indekse početnih retkih matrica, odnosno odgovarajućih Gausijana koji učestvuju u rešavanju integrala.

Optimizacija matrice odsecanja

Analizirajmo sada osobine matrice odsecanja. Ona ima iste dimenzije kao i matrice H i S, sa tim da umesto elemenata tipa double ima tip bool koji zauzima jedan bajt. Ponovo ćemo iskoristiti primer sistema veličine 10184 atoma, gde bi za alociranje memorije bilo potrebno približno 4GB. Iako je u početnoj postavci ovo zauzeće bilo značajno manje od zauzeća matrica H i S, sada je ono u odnosu na njih dominantno (pogledati tabelu 4.1). Dalje bi trebalo pronaći način da se eliminišu nula elementi ove matrice a da se pri tom sačuva mehanizam indeksiranja Gausijana.

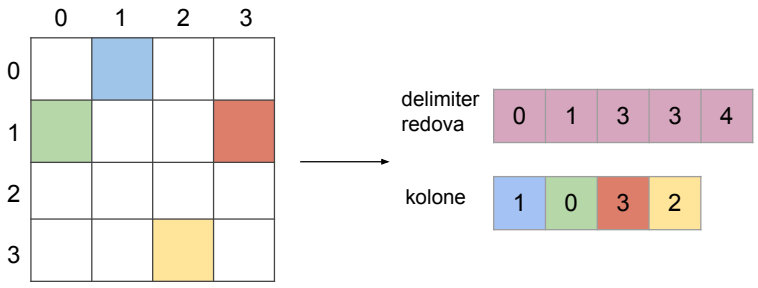
Postoje brojni načini predstavljanja retkih matrica koji najčešću primenu nalaze u rešavanju problema linearne algebre [106]. Teško je odrediti univerzalno dobar način, već je potrebno analizirati specifičnosti konkretnog problema koji se rešava. Početni motiv za slučaj matrice odsecanja je ušteda u memorijskom zauzeću, međutim važno je voditi računa i o ukupnom uticaju na performanse. Grupa rešenja koja se razmatra je svakako ona koja podrazumeva mogućnost mapiranja elemenata guste strukture sa indeksima retke matrice, što omogućava dvosmernu vezu između matrica H i S i matrice odsecanja.

Metoda koja maksimalno smanjuje potreban memorijski prostor za alociranje retke matrice a pritom zadovoljava navedene uslove je CSR (compressed sparse row, komprimovani retki redovi). Ova metoda podrazumeva postojanje tri vektora. Vektor delimiter redova sadrži elemente koji određuju broj popunjenih vrednosti retke matrice i to tako da indeks elementa odgovara indeksu reda retke matrice a vrednost predstavlja ukupni broj elemenata u svim redovima koji mu prethode. Veličina ovog vektora odgovara dimenziji retke kvadratne matrice uvećane za jedan, gde poslednji elemenat ima brojčanu vrednost svih popunjenih vrednosti retke matrice. Drugi vektor sadrži redom indekse vrsta svih popunjenih vrednosti retke matrice. Kombinacijom ova dva vektora se određuje indeks svake popunjene vrednosti retke matrice. Treći vektor sadrži same ne-nula vrednosti. Indeksi trećeg vektora odgovaraju indeksima drugog vektora, na osnovu čega se mapira vrednost sa indeksima retke matrice. Prva dva elementa su vektori celobrojnih tipova, dok tip elemenata trećeg vektora odgovara tipu vrednosti elemenata retke matrice. CSR metod dovodi do uštede u memoriji tek za slučajeve gde je broj ne-nula elemenata manji od $(n(n-1)-1)/2)$ gde je n dimenzija kvadratne retke matrice.

Kao što je već navedeno, elementi matrice odsecanja su logičke vrednosti. Dakle moguća su dva stanja, postojanje određenog elementa čiji indeksi odgovaraju indeksima retkih matrica H i S, ili njegovo izuzeće. Na osnovu prethodnog obrazloženja CSR metoda, može se zaključiti da je treći

vektor sa vrednostima elemenata u slučaju matrice odsecanja suvišan. Ovo dovodi do modifikovanog CSR metoda (nazovimo ga RCSR (Reduced CSR)), za koje važi sve kao i za CSR, sa tom razlikom što se eliminiše treći vektor. Primer upotrebe RCSR metoda je dat na slici 4.7.

U slučaju RCSR metoda, gde se za retku matricu odsecanja za čuvanje elemenata koristi C tip bool koji zauzima 1 bajt memorije, ušteda je smislena kada je broj ne-nula elemenata manji od $n(n-1)$. S obzirom da procenat zauzeća prikazan u tabeli 4.1, memorijske uštede su svakako značajne. Navedeni primer tiofen oligomera veličine od 10184 atoma gde je retka matrica odsecanja veličine približno 4GB, bi primenom CSR memorijsko zauzeće svelo na svega 48MB.



Slika 4.7: Primer sa RCSR formatom smeštanja podataka za retku matricu odsecanja 4×4 koja indeksira ne-nula elemente retkih matrica H i S.

Listing 4.8 implementira pretragu koja vraća informaciju o postojanju elementa određenog indeksima retke matrice (i, j), koji redom predstavljaju indeks vrste i kolone. Vektor *offset* je delimiter kolona, dok je *col* vektor indeksa kolona. Na osnovu indeksa i i $i+1$ je moguće odrediti broj elemenata koji se nalaze u zadatom redu matrice, kao i indekse koji ovaj vektor povezuju sa *col*. Ukoliko u zadatom opsegu vektora *col* postoji tražena vrednost, vraća se pozivan rezultat pretrage. Razlika između prikazane implementacije za RCSR u odnosu na CSR je u tome što bi za CSR bilo potrebno, na osnovu lociranog indeksa iz *col*, vratiti odgovarajući element trećeg vektora koji sadrži vrednosti elemenata. U oba slučaja je za lociranje odgovarajućeg elementa potrebno u najgorem slučaju izvršiti onoliko provera koliko ima elemenata u vrsti koja odgovara indeksu i. Ovo je i glavni nedostatak ovog načina čuvanja podataka, što predstavlja cenu maksimalne optimizacije skladišnog prostora.

Listing 4.8: Vraćanje vrednosti elementa RCSR matrice uz pomoć indeksa retke matrice

```
bool getValueRCSR(int i, int j, const unsigned *offset, const unsigned *col){
   bool res = 0;
   int it;
   for(it = offset[i]; it < offset[i+1]; it++){
      if(col[it] == j){
         res = 1;
         break;
      }
   }
   return res;
}
```

Da bi se cena pretrage i velikog broja logičkih provera ublažila, u delovima koda gde se vrši pristupanje većem skupu elemenata (bloku matrice) je moguće primeniti nešto drugačiji pristup;

prikazan pseudokodom u listingu 4.9. Za razliku od prethodnog rešenja koristi se činjenica da svakako postoji petlja koja prolazi kroz sve elemente (ili grupu elemenata) matrice nad kojom se vrše operacije, pa nije potrebno ponavljati isti proces pozivom metode $geteValueRCSR$. Na ovaj način se takođe izostavlja logička provera, jer se ne traži lociranje određenog elementa, već je dovoljno dobiti informaciju o indeksu j. Dalje se operacije vrše kao da se radi o retkoj matrici, tj. u slučaju prelaska sa 'klasičnog' načina rada sa elementima retkih struktura, nisu potrebna dodatna prilagođavanja postojeće implementacije.

Listing 4.9: Optimizovan pristup elementima RCSR matrice

```
for(i = 0; i < matrixSize; i++) {
   for(k = offsets[i]; k < offset[i+1]; k++){
      j = col[k];
      ... // Operacije nad gausijanima sa indeksima i,j
   }
}
```

Upotrebom RCSR metode smeštanja podataka smo očuvali mogućnost izvršavanja svih operacija nad matricama koje su primenjivane nad retkim strukturama. Dobijena je značajna ušteda memorije čime je omogućeno rešavanje daleko većih sistema u odnosu na rešenje sa retkim, neoptimizovanim matricama. Sa druge strane, došlo je do neminovnog gubitka na performansama u procesima koji se ne oslanjaju na redosled vrednosti. Razlog je što se umesto direktnog pristupa elementima na osnovu indeksa, potrebno mapiranje između matrica H i S i matrice odsecanja. Adekvatnom primenom pristupa memorijskim lokacijama matrica iz listinga 4.8 i 4.9 se ovaj nedostatak značajno ublažuje.

4.3.3 Paralelizacija integrala preklapanja, kinetičkog integrala i integrala jezgra

Zajedničko za integral preklapanja (3.1.1), kinetički integral (3.1.2) i integral jezgra (3.1.3) je to što se računanje doprinosa za pojedinačne elemente matrica H i S može organizovati tako da se u slučaju paralelizacije procesa ono izvršava bez komunikacije među procesima. Ukoliko se pri tome postigne i dobar balans računanja, takav scenario predstavlja idealno rešenje distribucije računanja ova tri integrala.

Za računanje ova tri integrala su potrebne prostorne koordinate jezgara molekula kao i podaci o kontrahovanim Gausijanima koji odgovaraju atomima za koje se računa integral. Ovi podaci su redom definisani promenljivom *mol* koja implementira strukturu molekula i sadrži podatke o ulaznom sistemu i vektorom *psi* koji predstavlja implementaciju niza struktura *phi* (opis globalnih struktura pogledati u 4.1.1).

Svi navedeni podaci su deo programskih struktura koje se bezmalo koriste u svim procesima računanja integrala i prirodno je da svaki proces sadrži njihovu kopiju. Drugačiji način distribucije bi bio logičan jedino ukoliko bi njihovo memorijsko zauzeće bilo značajno u odnosu na ostale memorijske strukture. Kako promenljiva *mol* sadrži niz definisan strukturama *nuclei* koji odgovara veličini ulaznog sistema a vektor *psi* sadrži definicije svih kontrahovanih Gausijana, njihova memorijska reprezentacija je bar dva reda veličine manje od memorijskog zauzeća matrica H i S (pogledati tabelu 4.1), pa ne postoji nikakvo ograničenje da ove strukture ne budu distribuirane kao kopija svim procesima. Procena je, da ukoliko bi ulazni sistem ipak bio dovoljno velik, tako da ove strukture

postanu problematične iz aspekta memorijskog zauzeća, veličina takvog sistema bi prouzrokovala probleme mnogo većih razmera kod struktura potrebnih za rešavanje Hartri integrala (4.3.5) kao i u procesu dijagonalizacije (4.3.7). Otuda razmatranje ove opcije prevazilazi okvire ovog rada.

Upravo ovakva organizacija podataka nam omogućava da se doprinosi ova tri integrala koji se računaju za svaki od elemenata matrice H i S, računaju nezavisno na paralelnim programskim procesima. Prikaz dela implementacije koja povezuje serijski algoritam sa procesom paralelizacije je dat u listingu 4.10. S obzirom da je ovo prva grupa integrala čiji doprinosi se računaju, najpre je potrebno inicijalizovati guste matrice H_dist_dense i S_dist_dense. Veličina ovih matrica je određena unapred izračunatom veličinom $denseNumEl\_dist$, koja predstavlja ukupan broj ne-nula elemenata matrica H i S (pogledati 4.3.2). Kako je ovo deo MPI procesa, matrice inicijalizovane na ovaj način odgovaraju distribuciji podataka koja je opisana u sekciji 4.3.1 tako da svaki proces ima odgovarajući blok globalnih matrica H i S. $matrixSizeDist$ i $matrixSize$ su redom veličina bloka distribuirane matrice i veličina globalne matrice (u ovom slučaju broj elemenata u svakom redu bloka). Promenljiva $iPosStart$ sadrži indeks elementa guste matrice koji određuje početak bloka (pogledati listing 4.6). Na osnovu dimenzija bloka matrice procedure $getValueRCSR()$ (4.8) se formiraju petlje kroz sve elemente matrica. Dalje je poziv odgovarajućih metoda kao i njihova implementacija deo koji je neizmenjen u odnosu na serijski program. Procedura $solveH()$ implementira izračunavanje kinetičnog integrala (3.1.2) i integrala jezgra (3.1.3), dok procedura $solveS()$ implementira integral preklapanja opisan u sekciji (3.1.1).

Listing 4.10: Distribucija računanja integrala preklapanja, kinetičkog integrala i integrala jezgra

```
double *H_dist_dense = alloc_double_vector(denseNumEl_dist);
double *S_dist_dense = alloc_double_vector(denseNumEl_dist);

int iPosStart = 0;
if(my_rank > 0){
   iPosStart = blocks[my_rank-1];
}

int i, j, iPos;
int countEl = 0;
for(i = 0; i < matrixSizeDist; i++) {
   iPos = iPosStart + i;
   for(j = 0; j < matrixSize; j++) {
      if(iPos >= j && getValueRCSR(i, j, withinRadiusDist_CSR_offsets,
          withinRadiusDist_CSR_col)) {
         H_dist_dense[countEl] = solveH(mol->atom[molIndex[iPos]], psi[iPos],
                   mol->atom[molIndex[j]], psi[j], mol);
         S_dist_dense[countEl] = solveS(mol->atom[molIndex[iPos]], psi[iPos],
                   mol->atom[molIndex[j]], psi[j]);
         countEl++;
      }
   }
}
```

Ovakvom implementacijom i načinom distribucija matrice u blokove se postiže relativno jednak raspored elemenata po procesima (brojčano). Ono što prostim rasporedom elemenata nije zagarantovano je razlika u smislu složenosti računanja pojedinačnih integrala za različite parove Gausijana. Međutim, imajući u vidu da je ovakav raspored nastao randomizacijom liste atoma (4.7) ulaznog sistema, znači da je odabir redosleda Gausijana (definisanim indeksima u blokovskim strukturama) slučajan. Samim tim bi računanje grupa integrala nad tako raspoređenim Gausijanima, trebalo da

bude dobro balansirano. Kao što je već obrazloženo u sekciji 4.3.1, pozitivan efekat ovog načina distribucije podataka se poboljšava povećanjem ulaznog sistema.

Treba još napomenuti da nakon računanja ovih integrala, svaki paralelni proces sadrži njihove doprinose samo za elemente matrica H i S koji su deo bloka koji im pripada. Sinhronizacija ovih podataka će biti opisana u daljim sekcijama.

Kao glavni doprinos pristupa opisanog u ovoj sekciji bi trebalo istaći eliminaciju potrebe za komunikacijom među procesima radi sinhronizovanja podataka, što samo po sebi dovodi do dobrog skaliranja paralelnog koda.

4.3.4 Paralelizacija procedure za učitavanje i fitovanje motiva

U ovom delu će biti opisana paralelna implementacija algoritma za ekstrahovanje motiva iz DFT proračuna (sekcija 3.2.1). Ova procedura učitava prethodno izračunate motive u realnom prostoru, izračunava gustinu elektronskog naelektrisanja i predstavlja (fituje) gustinu naelektrisanja kao linearnu kombinaciju Gausijana (jednačina (3.100)), gde se kao rezultat dobijaju koeficijenti Gausijana. Za implementaciju je primenjeno nekoliko pristupa koji se u osnovi razlikuju po načinu distribucije podataka. Podaci koje je potrebno distribuirati su oni potrebni kao ulazne strukture za rešavanje sistema linearnih jednačina datih u (3.106). Za razliku od serijske implementacije, paralelna verzija sadrži različite napredne mehanizme koji osim adekvatne distribucije podataka obezbeđuju optimalno izvršavanje i daju zadovoljavajući rezultat.

U daljem tekstu će biti opisana tri različita pristupa za fitovanje motiva kojima su zajednički sledeći koraci:

- učitavanje motiva,
- inicijalizacija matrice odsecanja za Kulonove Gausijane,
- računanje matrice A i vektora B potrebnih za rešavanje sistema linearnih jednačina (3.106),
- rešavanje sistema linearnih jednačina.

Učitavanje motiva je proces koji učitava motive, definisane preko njihovih vrednosti na uniformnom gridu u realnom prostoru, iz binarnih fajlova dobijenih izvršavanjem serijskog programa za ekstrahovanje motiva iz DFT proračuna. Na osnovu podataka dobijenih iz fajlova, ovaj proces kreira odgovarajuće C strukture podataka za potrebe procedure za fitovanje motiva.

Matrica odsecanja za Kulonove Gausijane omogućava implementaciju guste memorijske strukture za matricu A. Analogan sistem je korišćen za retke matrice H, S kao i za matricu odsecanja za 'standardne' Gausijane, opisane u sekciji 4.3.2, pa će detaljniji opis za guste strukture koje se primenjuju na deo za fitovanje motiva u ovom delu biti izostavljen.

Algoritam za izračunavanje matrice A i vektora B je u osnovi isti kao i u serijskoj verziji. Razlike se odnose na deo koji implementira distribuciju podataka za potrebe paralelnog izvršavanja kao i za optimizaciju procesa radi prilagođavanja paralelnom algoritmu.

Za razliku od serijskog algoritma koji za rešavanje sistema jednačina koristi LAPACK [82] rutinu *dgesv*, u paralelnoj implementaciji se koriste rutine kao i tipovi podataka PETSc biblioteke [107], [108]. PETSc je paralelna numerička biblioteka prvobitno namenjena za rešavanje parcijalnih diferencijalnih jednačina nad retkim strukturama podataka koja podržava MPI standard za komunikaciju. Biblioteke u PETSc-u su organizovane hijerarhijski, tako da omogućavaju apstrakciju

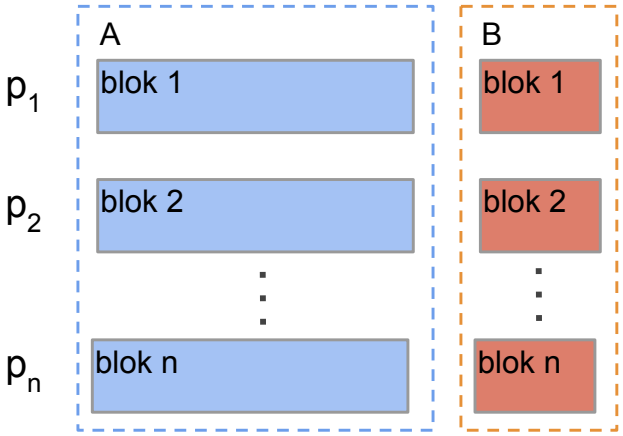
potrebnu za odgovarajuće prilagođavanje okruženja konkretnom problemu koji se rešava. Takođe su implementirani mehanizmi koji podržavaju definisanje struktura podataka (matrice i vektori) koje uz određene apstrakcije omogućavaju optimizaciju upotrebe tih struktura za različite tipove metoda za rešavanje numeričkih problema.

Procena je da je broj nenultih elemenata ove matrice jednak broju Kulonovih Gausijana pomnožen sa prosečnim brojem Kulonovih Gausijana koji su "blizu" datom Kulonovom Gausijanu. Za sistem od N atoma je broj Kulonovih Gausijana $30N$, a prosečan broj "bliskih" Kulonovih Gausijana 100, što dalje dovodi do broja od ukupno $3000 * N$ elemenata. Za procenu zauzeća memorije je potrebno uzeti u obzir dodatne memorijske strukture, pa se za npr. sistem od deset hiljada atoma dolazi do približnog memorijskog zauzeća od oko 500MB, što svakako predstavlja veoma značajan udeo u ukupnom zauzeću memorije CPM programa.

U nastavku će biti opisani različiti pristupi za implementaciju, sa naglaskom na način distribucije podataka.

Prvi način: Distribuirani blokovi podataka

Prvi način implementacije za fitovanje motiva za ideju ima da se podaci potrebni za rešavanje sistema linearnih jednačina distribuiraju po procesima u blokovima koji sadrže približno jednak broj elemenata (kao na slici 4.8). Matrica A i vektor B su ulazni podaci za rešavanje sistema linearnih jednačina datog u jednačini (3.106). Cilj je da se dobije minimalno opterećenje operativne memorije i da se time omogući rešavanje sistema sa velikim brojem atoma.



Slika 4.8: Distribuirani blokovi globalne matrice A i vektora B

Prva verzija implementacije sa distribuiranim blokovima je za matricu A imala retku strukturu podeljenu na blokove čija je veličina određivana na osnovu približnog broja ne-nula elemenata. Međutim, pošto su rešenja gustih memorijskih struktura za matrice H i S (4.3.2) otvorile put za rešavanje značajno većih sistema, u mnogim delovima koda je dovedena u pitanje održivost retkih matrica i vektora (uvođenjem gustih struktura (CSR) opis prve verzije implementacije gubi na značaju, tako da će biti izostavljen).

Imajući u vidu prethodno obrazloženje memorijskog zauzeća, logičan korak je da se na matricu A i vektor B primeni isti princip čuvanja podataka kao i za matrice H i S. Dakle, najpre je potrebno

odredili parove Gausijana za koje se može eliminisati računanje integrala (pogledati sekciju 3.2.2) i čiji indeksi određuju uslov za eliminaciju tih elemenata iz guste matrice A. Način na koji se implementira inicijalizacija CSR matrice za odsecanje za Kulonove Gausijane je dat u listingu 4.11 (isti princip se primenjuje i na CSR matricu za odsecanje za 'standardne' Gausijane ali se u ovom delu prvi put uvode detalji implementacije). Dalje se uz pomoć procedure *radius_gaussian2*() računa poluprečnik za ceo skup Kulonovih Gausijana, koji se zatim smešta u niz *gaussian_radii*[]. U daljem delu koda se za svaki par Gausijana proverava da li se njihove sfere (koje su određene izračunatim poluprečnicima) preklapaju. Ukoliko je to slučaj, odgovarajući indeks se dodeljuje CSR nizu *withinRadiusCoulombDist_CSR_col*[], dok se na kraju svakog reda evidentira ukupan broj elemenata u delimiter redova *withinRadiusCoulombDist_CSR_col*[]. Ova procedura se izvršava nad blokom podataka, odnosno skupom Gausijana, koji odgovaraju bloku matrice A, tako da svaki proces sadrži matricu odsecanja samo za segment podataka nad kojim vrši operacije.

Listing 4.11: Inicijalizacija CSR matrice odsecanje za Kulonove Gausijane

```
void determineCoulombCutOff_CSR_dist(int matrixSize, int matrixSizeDist, int iPosStart,
        const struct molecule *mol, const struct phi *psi, const int *molIndex,
        unsigned *withinRadiusCoulomb_CSR_offsets, unsigned
            *withinRadiusCoulombDist_CSR_col){

   double *gaussian_radii = alloc_double_vector(matrixSize);
   double distance,distance2;
   int my_rank, iPos, i, j;

   for (i=0;i<matrixSize;i++){
      gaussian_radii[i]=radius_gaussian2(psi[i],1e-4,0.001);
   }

   unsigned count = 0;
   for (i=0;i<matrixSizeDist;i++){
      iPos =iPosStart + i;
      withinRadiusCoulomb_CSR_offsets[i] = count;
      for (j=0;j<matrixSize;j++){
            distance = 0.0;
            distance += (mol->atom[molIndex[iPos]].x - mol->atom[molIndex[j]].x) *
               (mol->atom[molIndex[iPos]].x - mol->atom[molIndex[j]].x);
            distance += (mol->atom[molIndex[iPos]].y - mol->atom[molIndex[j]].y) *
               (mol->atom[molIndex[iPos]].y - mol->atom[molIndex[j]].y);
            distance += (mol->atom[molIndex[iPos]].z - mol->atom[molIndex[j]].z) *
               (mol->atom[molIndex[iPos]].z - mol->atom[molIndex[j]].z);
            distance2 = (gaussian_radii[iPos]+gaussian_radii[j]) *
               (gaussian_radii[iPos]+gaussian_radii[j]);
            if ( distance <= distance2 ){
               withinRadiusCoulombDist_CSR_col[count] = j;
               count++;
            }
      }
   }

   withinRadiusCoulomb_CSR_offsets[matrixSizeDist] = count;

   free_double_vector(gaussian_radii);
}
```

Sledeći korak u implementaciji je računanje elemenata matrice A i vektora B. U pseudokodu u listingu 4.12 je prikazan niz operacija koje zahtevaju višestruku sinhronizaciju podataka uz pomoć

MPI mehanizama za komunikaciju. Najpre je potrebno distribuirano računanje težinskih funkcija, potrebnih za blok nad kojim se vrši računanje, i sumiranje njihovih vrednosti koje se zatim dodeljuju tekućem procesu.

Kod sumiranja vrednosti za težinske funkcije se u MPI rutini $MPI\_Allreduce$ koristi parametar MPI_IN_PLACE. Ovaj parametar se učestalo koristi i na drugim mestima u implementaciji jer on omogućava korišćenje istog bafera za slanje i primanje podataka (sendbuf, recbuf). U slučaju rutine $MPI\_Allreduce$, MPI_IN_PLACE zamenjuje argument bafera za slanje, čime bafer koji prima podatke preuzima funkciju oba bafera. Na ovaj način se u slučaju slanja velikih paketa podataka dobija značajna ušteda u memorijskom prostoru.

Dalje se kroz petlju inicira računanje elemenata svih blokova matrice A i vektora B i to tako što se za svaki blok elementi paralelno računaju na svim raspoloživim procesima. Paralelno računanje zahteva distribuciju svih potrebnih parametara i lokalnih CSR struktura svim procesima. Tek nakon tog koraka su moguće operacije računanja, nakon čega se rezultat, uz pomoć MPI rutine MPI_Reduce, konačno vraćanja tekućem procesu.

Listing 4.12: Pseudokod implementacije računanja matrice A i vektora B potrebnih za rešavanja sistema linearnih jednačina (3.106)

```
// Memorijski blokovi matrica A i vektora B distribuirani po procesima
double *aa_dist_dense = alloc_double_vector(aa_dist_dense_size);
double *bb_dist=alloc_double_vector(coulombMatrixSize_dist[icore]);

double *weight_k = alloc_double_vector(weight_k_size);

...// Paralelno računanje weight_k[] elemenata (težinskih funkcija)

// Sumiranje svih vrednosti weight_k sa razlicitih procesa
MPI_Allreduce(MPI_IN_PLACE, &weight_k[0], weight_k_size, MPI_DOUBLE, MPI_SUM,
    MPI_COMM_WORLD);
MPI_Barrier(MPI_COMM_WORLD);

for(ic = 0; ic < ncore; ic++){

   ...// ic proces priprema parametre

   // ic proces šalje parametre svim procesima
   MPI_Bcast(&paramsC[0], 4, MPI_INT, ic, MPI_COMM_WORLD);

   // Privremene lokalne strukture
   double *aa_dist_dense_temp = alloc_double_vector(paramsC[3]);
   double *bb_dist_temp = alloc_double_vector(coulombMatrixSizeDist_temp);

   // ic proces šalje lokalne CSR matrice odsecanja za Kulonove gausijane svim procesima
   MPI_Bcast(&withinRadiusCoulombDist_CSR_offsets_temp[0], coulombMatrixSizeDist_temp+1,
       MPI_UNSIGNED, ic, MPI_COMM_WORLD);
   MPI_Bcast(&withinRadiusCoulombDist_CSR_col_temp[0], paramsC[0], MPI_UNSIGNED, ic,
       MPI_COMM_WORLD);

   ...// Paralelno računanje elemenenata aa_dist_dense_temp[] i bb_dist_temp[]

   ...// Sumiranje svih elemenata nazad procesu ic
   MPI_Reduce(&aa_dist_dense_temp[0], &aa_dist_dense[0], paramsC[3], MPI_DOUBLE, MPI_SUM,
       ic, MPI_COMM_WORLD);
   MPI_Reduce(&bb_dist_temp[0], &bb_dist[0], coulombMatrixSizeDist_temp, MPI_DOUBLE,
       MPI_SUM, ic, MPI_COMM_WORLD);
}
```

Na kraju je potrebno rešiti sistem linearnih jednačina uz pomoć PETSc rutina (listing 4.13). Najpre je potrebno inicijalizovati PETSc globalne strukture i prilagoditi njihov raspored blokova sa blokovima matrice A i vektore B (izračunatim u prethodnom procesu). Dalje, svaki proces vrši dodelu elemente svog bloka podataka odgovarajućim PETSc globalnim strukturama.

Za rešavanje sistema linearnih jednačina se koristi Krylov subspace metod (KSP) koji u osnovnoj postavci za paralelno računanje koristi kombinaciju GMRES (Generalized Minimal Residual method) [109] algoritma, sa sistemom za restartovanje računanja, u kombinaciji sa Block Jacobi prekondicionisanjem (preconditioner) – PCBJACOBI. Block Jacobi algoritam kreira blokove tako da svaki blok ima svoj KSP objekat gde se vrši ILU (Incomplete LU prekondicioner) faktorizacija. Uobičajena postavka (koja je i ovde korišćena) je da svaki proces ima tačno jedan blok. Više detalja o pomenutim algoritmima i njihovoj primeni se mogu pronaći u dokumentaciji PETSc biblioteke [110].

PCLU je prekondicioner koji koristi direktni metod za LU faktorizaciju. Ovaj metod vraća 'tačno' rešenje u jednoj iteraciji i nema potrebu za izvršavanjem Krylov-ljevog metoda. Prednost ovog metoda je što je za sve testirane sisteme bila značajno efikasnija u brzini izvršavanja od ostalih metoda. Osnovni problem i razlog zbog čega se ne koristi u produkcionoj verziji programa je što u procesu rešavanje vrši dinamičku alokaciju memorije koja nekoliko puta prevazilazi veličinu struktura koje koristi za ulazne parametre. Iz tog razloga se u našoj implementaciji PCLU prekondicioner koristi isključivo za proveru tačnosti rezultata kod malih sistema.

Listing 4.13: Pseudokod implementacije rešavanja sistema linearnih jednačina ((3.106)) uz pomoć PETSc-a

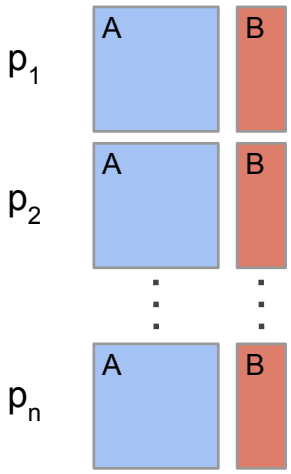
```
// Inicijalizacija PETSc i matrice A
ierr = PetscInitialize(&argcT,&argvT,(char*)0,help);if (ierr) return ierr;
ierr = MatCreate(PETSC_COMM_WORLD,&A);CHKERRQ(ierr);
ierr = MatSetSizes(A,coulombMatrixSize_dist[my_rank],
   coulombMatrixSize_dist[my_rank],coulombMatrixSize,coulombMatrixSize);
   CHKERRQ(ierr);
ierr = MatSetUp(A);CHKERRQ(ierr);

// Odredjuje se indeks prvog reda bloka koji pripada tekućem procesu
if(my_rank > 0){
  iPosStart = coulombMatrixSize_dist[my_rank-1] * my_rank;
}

// Inicijalizacija globalne PETSc matrice vrednostima iz tekućeg bloka matrice aa
for(i = 0; i < coulombMatrixSize_dist[my_rank]; i++){
  iPos = iPosStart + i;
  for(j = 0; j < coulombMatrixSize-1; j++){
     if(getValueCSR(i, j, withinRadiusCoulombDist_CSR_offsets,
          withinRadiusCoulombDist_CSR_col)){
        ierr = MatSetValue(A,iPos,j,aa[countEl],INSERT_VALUES);CHKERRQ(ierr);
        countEl++;
     }
  }
}

...// Inicijalizacija ostalih elemenata matrice A koji nisu sadržani u blokovima

ierr = MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
ierr = MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
```

```
// Kreiranje vektora
ierr = VecCreate(PETSC_COMM_WORLD,&b_vec);CHKERRQ(ierr);
ierr = VecSetSizes(b_vec,coulombMatrixSize_dist[my_rank],coulombMatrixSize);CHKERRQ(ierr);
ierr = VecSetFromOptions(b_vec);CHKERRQ(ierr);
ierr = VecCreate(PETSC_COMM_WORLD,&x_vec);CHKERRQ(ierr);
ierr = VecSetSizes(x_vec,coulombMatrixSize_dist[my_rank],coulombMatrixSize);CHKERRQ(ierr);
ierr = VecSetFromOptions(x_vec);CHKERRQ(ierr);

// Inicijalizacija vektora b
for(i = 0; i < coulombMatrixSize_dist[my_rank]; i++){
  iPos = iPosStart + i;
  VecSetValue(b_vec, iPos, bb[i], INSERT_VALUES);
}

ierr = VecAssemblyBegin(b_vec);CHKERRQ(ierr);
ierr = VecAssemblyEnd(b_vec);CHKERRQ(ierr);

// Kreiranje konteksta za Gram-Schmidt ortogonalizaciju
ierr = KSPCreate(PETSC_COMM_WORLD,&ksp);CHKERRQ(ierr);

ierr = KSPSetOperators(ksp,A,A);CHKERRQ(ierr);

// Opciono postavljanje direktnog solvera baziranog na LU faktorizaciji
#ifdef DEBUG_MODE
ierr = KSPGetPC(ksp,&pc);CHKERRQ(ierr);
ierr = PCSetType(pc,PCLU);CHKERRQ(ierr);
#endif
ierr =
    KSPSetTolerances(ksp,1.e-10,PETSC_DEFAULT,PETSC_DEFAULT,PETSC_DEFAULT);CHKERRQ(ierr);

// Rešavanje linearnog sistema
ierr = KSPSolve(ksp,b_vec,x_vec);CHKERRQ(ierr);

// Distribucija vektora sa rešenjem sistema svim procesima
VecScatterCreateToAll(x_vec,&ctx,&x_local);
VecScatterBegin(ctx,x_vec,x_local,INSERT_VALUES,SCATTER_FORWARD);
VecScatterEnd(ctx,x_vec,x_local,INSERT_VALUES,SCATTER_FORWARD);

PetscScalar *array;
ierr = VecGetArray(x_local,&array);CHKERRQ(ierr);
for (i=0; i<coulombMatrixSize; i++) xx[i] = array[i];
ierr = VecRestoreArray(x_local,&array);CHKERRQ(ierr);

..// Dealokacija memorije

ierr = PetscFinalize();
```

Drugi način: Distribuirane kopije podataka

Cilj drugog načina implementacije je da za razliku od prvog, koji ima značajan nedostatak zbog potrebe za velikim brojem operacija za sinhronizaciju podataka, omogući povećanje efikasnosti izvršavanja. Da bi se to postiglo, jedno od rešenja je da svaki proces sadrži kopiju svih potrebnih podataka (matrica A i vektor B) za rešavanje sistema linearnih jednačina (pogledati sliku 4.9). Cena ovoga je dodatno opterećenje memorije ali s obzirom na maksimalnu uštedu koja je po tom pitanju dobijena dobrom implementacijom gustih struktura, time se otvara mogućnost za rešavanje sistema sa značajno većim brojem atoma. Na ovaj način bi se obezbedio vrlo efikasan program

za rešavanje sistema manjih do srednjih veličina, dok bi za velike sisteme bio neophodan drugačiji način distribucija podataka. Npr. za već pomenuti sistem od 10000 atoma bi ukupno memorijsko zauzeće za matricu A i prateće strukture bio 500MB po procesu, što bi za računarski nod sa jednim CPU-om koji ima 16 jezgara, opterećenje operativne memorije bilo oko 8GB. Imajući u vidu i ostale memorijske strukture, koje se istovremeno čuvaju u operativnoj memoriji, za raspoložive resurse za testiranje ovo predstavlja granični slučaj (više o ovome u sekciji 5.2).



Slika 4.9: Svaki proces poseduje kopiju matrice A i vektora B

Razlika u implementaciji sa distribuiranim kopijama podataka u odnosu na prethodno rešenje sa blokovima je u tome što se ovde umesto blokova svim procesima distribuiraju cele (globalne) matrice. Otuda se prilagođavanja procedura svode uglavnom na eliminaciju blokova i učitavanja kompletnih struktura u svim procesima. Iz tog razloga nije potrebno navoditi detalje implementacije.

Treći način: MPI deljena memorija

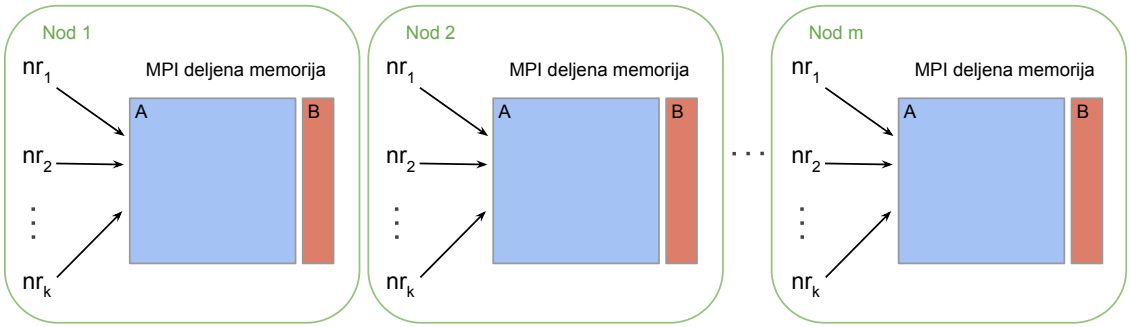
S obzirom da u paralelnom procesu za fitovanje motiva svi procesi pristupaju istom skupu podataka, potrebno je razmotriti i načine implementacije koji uključuju deljenu memoriju. Razmotrimo nekoliko opcija. Prvi je implementacija hibridnog algoritma (MPI + OpenMP). Iako ovo svakako predstavlja efikasan način rešavanja problema ovog tipa, dovodi se u pitanje izvodljivost, jer bi uvođenje hibridnog algoritma zahtevalo značajne izmene organizacije podataka i time zahtevalo opsežne izmene mnogih programskih procesa. Verzija algoritma koja bi uključila hibridno izvršavanje ovog dela procesa je moguća opcija za buduća unapređenja postojećeg CPM programa.

Sledeći mogući pristup je primena standarnog RMA (Remote Memory Access – udaljeni memorijski pristup) koji predstavlja proširenje MPI mehanizma za komunikaciju i koji uz pomoć skupa MPI rutina (MPI_Get/MPI_Put) i odgovarajuće sinhronizacije podataka omogućava udaljeni pristup memorijskim lokacijama. Iako se na ovaj način izbegava višestruko ponavljanje istih struktura podataka, obzirom na veliki broj potrebnih sihnronizacija ovaj mehanizam ne predstavlja poboljšanje u odnosu na standardnu MPI komunikaciju implementiranu na prvi način.

Ono što je takođe moguće, je da se primeni sistem koji u osnovi ima RMA pristup koji je dizajniran tako da može da iskoristi činjenicu da grupa procesora koji su definisani nad jednim računarskim nodom dele istu fizičku memoriju. Za razliku od standardnog MPI pristupa za komunikaciju, gde svi

procesi imaju isti tretman, ovde se pravi razlika koju određuje raspored procesa po fizički različitim računarskim nodovima. Ovaj pristup koji se još naziva MPI model sa deljenom memorijom (MPI Shared Memory Model) je uveden sa MPI 3.0 specifikacijom [105] i on predstavlja podskup MPI RMA rutina koji na nivou jednog računarskog noda mogu na vrlo efikasan način da obezbede direktan pristup deljenim memorijskim blokovima (MPI memorijski prozori). Vizija ovog pristupa je da se u nekim budućim specifikacijama MPI-a hibridna arhitektura, koja u praksi najčešće kombinuje MPI sa OpenMP pristupom, integriše u samu MPI specifikaciju.

Primena ovog načina distribucije podataka je prikazana na slici 4.10, gde svaki računarski nod sadrži kopiju matrica A i vektora B tako da svi procesi iz grupe procesa definisani nad jednim nodom pristupaju istim memorijskim lokacijama, koje se nalaze na istoj fizičkoj memoriji. Na taj način se isključuje potreba za sinhronizacijom podataka koja bi se vršila komunikacijom procesa sa različitih nodova. Uz pomoć interfejsa MPI modela sa deljenom memorijom, podacima unutar jedne grupe procesa se pristupa na isti način kao da se radi o implementaciji sa hibridnom arhitekturom (MPI+OpenMP). Drugim rečima, lokalnim procesima su deljeni podaci direktno dostupni.



Slika 4.10: Pristup globalnoj matrici A i vektoru B uz pomoć MPI deljene memorije (MPI Memory Window)

Proces implementacije procedure za fitovanje motiva sa MPI deljenom memorijom je dat u listingu 4.16 i on po fazama izgleda ovako:

1. Procese iz globalne grupe paralelnih procesa (MPI_COMM_WORLD) je najpre potrebno organizovati u grupe komunikatora koje čini procesi unutar jednog fizičkog noda (MPI rutina $MPI\_Comm\_split\_type()$). Sada svaki proces pored globalnog identifikatora ima i identifikator ($noderank$) na nivou jednog računarskog noda ($nodecomm$).

2. Definiše se MPI memorijski prozor (MPI memory window), tako da svaki $nodecomm$ ima pristup deljenom 'komadu' memorije (MPI rutina $MPI\_Win\_allocate\_shared()$). U našem slučaju su to matrica A i vektor B (promenljive aa_dense i bb koje implementiraju jednačinu (3.106)) kao i prateće strukture (pogledati MPI_Win instance i njihovu upotrebu u listingu (4.16)). Sada umesto da svaki proces pojedinačno alocira navedene memorijske strukture, svi procesi unutar jednog noda, iz aspekta programskog interfejsa, imaju mogućnost direktnog pristupa memorijskim lokacijama (pogledati sliku 4.10).

3. Svaki proces nezavisno računa elemente matrice A ($evaluate\_bb\_all\_and\_aa\_parallel\_v5()$, listing 4.14), tako da svi procesi unutar jedne grupe procesa ($nodecomm$) upisuju vrednosti u isti memorijski blok. MPI rutina $MPI\_Accumulate()$ koristeći RMA, dodaje vrednost (operacija

MPI_SUM) odgovarajućem elementu memorijskog prozora win_aa_dense. Da bi se izvršila adekvatna sinhronizacija i kompletirao RMA poziv, nakon svake operacije je potrebno pozvati rutinu $MPI\_Win\_flush()$. Na ovaj način se simulira atomska operacija nad elementima deljene memorijske strukture.

Listing 4.14: Pseudo kod implementacije procedure za fitovanje motiva sa MPI deljenom memorijom

```
...// Računanje weight_k i elemenata vektora ga[]
for(i = 0; i < coulombMatrixSize - 1; i++) {
   ...// Formiranje indeksa za pristup odgovarajućim elementima

   res = 2 * ga[i] * ga[j] * weight_k;

   #ifdef USE_RMA_ATOMICS
   MPI_Accumulate(&res, 1, MPI_DOUBLE, 0, countEl, 1, MPI_DOUBLE, MPI_SUM, win_aa_dense);
   MPI_Win_flush(0, win_aa_dense);
   #endif
}
```

4. U procesu dijagonalizacije ($linearSysSolver\_v4()$) je potrebno formirati globalnu PETSc matricu A (listing 4.15). Proces formiranja elemenata matrice je takav, da svaka grupa komunikatora ($nodecomm$) ima deo ukupne sume globalne matrice. Da bismo formirali globalnu PETSc matricu, potrebno je da se saberu sve matrice iz pojedinačnih grupa komunikatora. Pošto svaki proces iz jedne grupe ima pristup istom skupu podataka (memorijski prozor), dovoljno je da se odabere proizvoljni proces koji će vršiti dodelu doprinosa (npr. $noderank$ sa indeksom 0). Dodela vrednosti se vrši uz pomoć PETSc rutine $MatSetValue()$, tako što se za tip operacije bira ADD_VALUES. Ovakvim načinom organizovanja podataka se izbegava sinhronizacija na nivou RMA.

Listing 4.15: Dodela vrednosti globalnim PETSc matricama u delu procedure za rešavanje sistema linearnih jednačina ($linearSysSolver\_v4()$)

```
if(noderank == 0){
  long countEl = 0;
  for(i = 0; i < coulombMatrixSize-1; i++){
     for(j = 0; j < coulombMatrixSize-1; j++){
        if(i >= j && getValueCSR(i, j, withinRadiusCoulomb_CSR_offsets,
             withinRadiusCoulomb_CSR_col)){
           ierr = MatSetValue(A,i,j,aa[countEl],ADD_VALUES);CHKERRQ(ierr);
           if(i != j){
              ierr = MatSetValue(A,j,i,aa[countEl],ADD_VALUES);CHKERRQ(ierr);
           }
           countEl++;
        }
     }
  }
}
```

5. Izvršava se proces paralelnog rešavanja sistema linearnih jednačina na isti način kao i u prethodnim načinima implementacije.

Listing 4.16: Pseudo kod implementacije procedure za fitovanje motiva sa MPI deljenom memorijom

```
// Podela globalnog komunikatora na grupe,
// gde svaka grupa predstavlja procese jednog računarskog noda
MPI_Comm nodecomm;
int nodesize, noderank;
```

```
MPI_Comm_split_type(MPI_COMM_WORLD, MPI_COMM_TYPE_SHARED, icore,
     MPI_INFO_NULL, &nodecomm);

MPI_Comm_size(nodecomm, &nodesize);
MPI_Comm_rank(nodecomm, &noderank);

// Samo root proces vrši alokaciju memorije
int local_aa_dense_size = 0;
if (noderank == 0){
   local_aa_dense_size = aa_dense_size;
}

// Inicijalizacija memorijskog prozora (deljena memorija na nivou računarskog noda)
MPI_Win win_aa_dense;
MPI_Aint winsize;
int windisp;
double *aa_dense, *local_aa_dense;

MPI_Win_allocate_shared(local_aa_dense_size*sizeof(double), sizeof(double),
     MPI_INFO_NULL, nodecomm, &local_aa_dense, &win_aa_dense);

// Postavljanje lokalnog pokazicača na početnu lokaciju memorije root procesa
aa_dense = local_aa_dense;
if (noderank != 0){
   MPI_Win_shared_query(win_aa_dense, 0, &winsize, &windisp, &aa_dense);
}

// Memorijski prozor za withinRadiusCoulomb_CSR_col
int local_rc_size = 0;
if (noderank == 0){
   local_rc_size = coulombNumEl;
}
MPI_Win win_rc;
MPI_Aint winsize_rc;
int windisp_rc;
unsigned *withinRadiusCoulomb_CSR_col, *local_withinRadiusCoulomb_CSR_col;

MPI_Win_allocate_shared(local_rc_size*sizeof(unsigned), sizeof(unsigned),
     MPI_INFO_NULL, nodecomm, &local_withinRadiusCoulomb_CSR_col, &win_rc);

withinRadiusCoulomb_CSR_col = local_withinRadiusCoulomb_CSR_col;
if (noderank != 0){
   MPI_Win_shared_query(win_rc, 0, &winsize_rc, &windisp_rc, &withinRadiusCoulomb_CSR_col);
}

...// Sličan proces inicijalizacije memorijskog prozora
...// ponoljen za withinRadiusCoulomb_CSR_offsets i bb

MPI_Barrier(MPI_COMM_WORLD);
if(noderank == 0){
   determineCoulombCutOff_CSR(coulombMatrixSize-1, mol, rhoCoulomb,
        molIndexCoulomb, withinRadiusCoulomb_CSR_offsets,
        withinRadiusCoulomb_CSR_col, ncore, icore, coulombNumEl);
}

evaluate_bb_all_and_aa_parallel_v5(bb, aa_dense, mol, coulombMatrixSize, molIndexCoulomb,
     rhoCoulomb, coul_gaussian_radii, Rall, mm1, mm2, mm3, box1, box2, box3, mmotif,
     whichMotif,radialGridSize,ncore,icore,aa_dense_size,
     withinRadiusCoulomb_CSR_offsets, coulombNumEl, withinRadiusCoulomb_CSR_col,
     win_aa_dense, nodecomm, noderank, local_aa_dense_size);
MPI_Barrier(MPI_COMM_WORLD);
```

```
bb[coulombMatrixSize-1] = -(mol->N);

ierr = linearSysSolver_v4(aa_dense, bb, coulombMatrixSize, icore, ncore, xx,
     withinRadiusCoulomb_CSR_offsets, withinRadiusCoulomb_CSR_col,
     coulombNumEl, aa_dense_size, noderank);

MPI_Win_free(&win_aa_dense);
MPI_Win_free(&win_rc);
MPI_Win_free(&win_rc_off);
```

4.3.5 Paralelizacija računanja integrala Hartri potencijala

Paralelna implementacija računanja integrala Hartri potencijala (u daljem tekstu Hartri) je u osnovi slična implementaciji paralelizacije integrala preklapanja, kinetičkog integrala i integrala jezgra (4.3.3). U ovom slučaju je takođe moguće računanje doprinosa koji se dodaju pojedinačnim elementima matrice H, bez razmene podataka među paralelnim procesima. Za razliku od implementacije pomenuta tri integrala, u slučaju Hartija je potrebno izvršiti određene izmene i prilagođavanja serijskog procesa računanja integrala. Pritom se izmene prvenstveno odnose na proces optimizacije računanja osnovnih integrala definisanih jednačinom (3.61).

U serijskom kodu je sve potrebne osnovne integrale moguće izračunati unapred, čime se uz adekvatnu implementaciju pristupa elementima strukture koja čuva njihove vrednosti, eliminiše potreba za ponovljenim operacijama računanja integrala za iste parove Gausijana. U paralelnoj implementaciji su stvari nešto složenije. S obzirom na veličinu tenzora koji bi sadržao sve potrebne integrale, nije moguće čuvati njegovu kopiju za sve paralelne procese. Takođe ni distribucija takvog tenzora ne predstavlja optimalno rešenje, jer bi obim komunikacije radi pristupa odgovarajućim elementima značajno uticao na ukupne performanse.

Rešenje koje se nameće je da se, umesto kompletnog skupa osnovnih integrala, izračunaju samo grupe koje predstavljaju uzastopne primitivne Gausijane koji se nalaze u lokalnoj memorijskoj strukturi. Dva osnovna integrala imaju istu vrednost rešenja za sve kombinacije primitivnih Gausijana ((3.54)) koji imaju istu sumu svojih stepena kao i iste vrednosti koeficijenata. Pseudokod koji opisuje proces poziva procedura za računanje Hartrija je prikazan u listingu 4.17. Petlje indeksiraju sve elemente bloka matrice koje pripadaju odgovarajućem MPI procesu, dok je procedura $GetValueRCSR()$ (4.8) zadužena za eliminaciju svih elemenata čiji doprinosi se ne računaju. U svakoj iteraciji se radi provera da li je u prethodnom koraku već izračunata grupa osnovnih integrala koji su potrebni za računanje Hartri potencijala. Ukoliko to nije slučaj, inicijalizuje se matrica $primInt[][]$ i računaju se svi potrebni osnovni integrali, nakon čega se poziva procedura rekurentnih relacija $formRecRel2()$. Ukoliko se računanje ponavlja za istu grupu primitivnih Gausijana, poziva se procedura sa $formRecRel2()$ gde se kao parametar prosleđuje matrica $primIntPrev[][]$ koja sadrži prethodno izračunate vrednosti osnovnih integrala.

Listing 4.17: Pseudokod implementacije paralelne verzije za računanje integrala Hartri potencijala

```
countEl = 0;
for(i2 = 0; i2 < matrixSizeDist; i2++) {
   iPos = iPosStart + i2;
   for(j2 = 0; j2 < matrixSize; j2++) {
      if(iPos >= j2 && getValueRCSR(i2, j2, withinRadiusDist_CSR_offsets,
          withinRadiusDist_CSR_col)){
```

```
        res = 0.;
        double **primIntPrev = alloc_double_matrix(isize,mMax);
        for(ic = 0; ic < coulombMatrixSize-1; ic++){
           if(... /* Provera da li je prethodna grupa primInt već izračunata*/){
              elRepRes = formRecRel2(primIntPrev);
           }else{
              double **primInt = alloc_double_matrix(isize,mMax);

              ... // Računanje grupe primitivnih integrala primInt[][]

              elRepRes = formRecRel2(primInt);
              free_double_matrix(primInt);
           }
           res = res + (-1.) * initRhoCoeff[ic] * elRepRes;
        }
        free_double_matrix(primIntPrev);
        H_dist_dense[countEl] = H_dist_dense[countEl] + res;
        countEl++;
     }
  }
}
```

Na ovaj način gde svaki MPI proces nezavisno računa svoj blok matrice, postiže se paralelno računanje doprinosa Hartri potencijala. Kao i u slučaju 4.3.3 i ovde se postiže dobar efekat dobijen randomizacijom atoma ulaznog sistema (4.7). Iako se složenost rešenja rekurentnih relacija nad različitom kombinacijom ulaznih parametara može značajno razlikovati, zbog slučajnog odabira rasporeda atoma se kod većih sistema ovaj efekat značajno umanjuje.

Osim predloženog rešenja za optimizaciju osnovnih integrala, je moguće uzeti u obzir da je sam oblik rekurzivnih relacija takav da bi najprirodnije bilo da se Hartri potencijali računaju u paketima. Pod paketom se podrazumevaju svi integrali $(ab|c)$ (3.44), takvi da su a recimo p Gausijani (px, py, pz) na atomu A, b je recimo s Gausijan na atomu B, a c su recimo p Gausijani na atomu C. Ovi integrali imaju zajedničke osnovne integrale i do njih se moze doći zajedničkim rekurzivnim postupkom ((3.58) i (3.59)). Računanje integrala bi trebalo biti organizovano tako da jedan proces sadrži sve potrebne podatke tako da bez komunikacije računa sve integrale koji 'pripadaju' jednom paketu. Ravnomernom raspodelom paketa bi bio postignut balans računanja. Osnovni problem ovog rešenja je što bi on zahtevao drugačiji način distribucije podataka, što bi rezultovalo u obimnim izmenama svih programskih procedura koje se oslanjaju na postojeći način organizovanja podataka.

4.3.6 Paralelizacija računanja izmensko-korelacionog integrala

Iz aspekta paralelizacije, algoritam za računanje izmensko-korelacionog integrala deli isti princip za distribuciju podataka kao i algoritam za fitovanje motiva koji je opisan u sekciji 4.3.4. Matrica koja sadrži rešenja izmensko-korelacionih integrala je po strukturi i određivanju rasporeda ne-nula elemenata guste matrice ista kao i matrica A iz 4.3.4.

U zavisnosti od načina distribucije podataka, kao i u slučaju 4.3.4, postoje tri različita načina implementacije: organizovanje podataka u blokovima koji se distribuiraju paralelnim procesima (slika 4.8), kopiranje kompletne matrice doprinosa izmensko-korelacionih integrala svim raspoloživim procesima (slika 4.9) i primena MPI deljene memorije za pristup memorijskog prozora koji se definiše za svaki računarski nod (slika 4.10).

Zajednički algoritam za sva tri načina implementacije ima sledeće korake:

- Inicijalizacija matrice odsecanja za 'standardne' Gausijane,
- Računanje matrice doprinosa izmensko-korelacionih integrala,
- Dodavanje doprinosa matrici H.

Inicijalizacija matrice odsecanja za 'standardne' Gausijane se u odnosu od matricu za odsecanje Kulonovih Gausijana razlikuje po skupu Gausijana nad kojima se ona formira ali se primenjuje isti kriterijum za eliminaciju članova koji ne učestvuju u računanju.

Računanje izmensko-korelacionih integrala se u velikoj meri oslanja na serijsku verziju algoritma nad kojom se primenjuju isti principi paralelizacije kao i u sekciji 4.3.4, tako da će detalji u vezi implementacije biti izostavljeni.

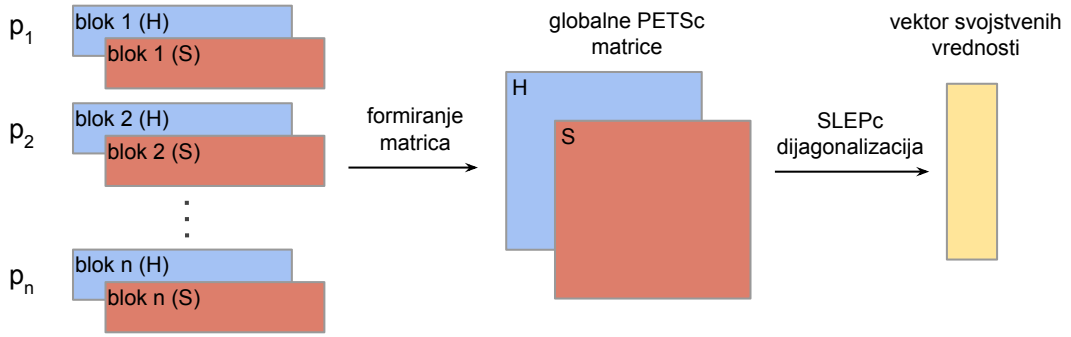
U poslednjem koraku se izračunati doprinosi dodeljuju matrici H, nakon čega se vrši dijagonalizacija opisana u sekciji 4.3.7.

4.3.7 Paralelizacija dijagonalizacije Hamiltonijana

Nakon izračunavanja doprinosa svih potrebnih integrala, poslednja procedura koju je potrebno izvršiti je rešavanje generalizovanog svojstvenog problema (dijagonalizacija Hamiltonijana) datog u jednačini (3.6). U jednoprocesorskoj implementaciji, gde programski proces ima pristup svim elementima matrica H i S, dijagonalizacija se vrši korišćenjem standardne jednoprocesorske LAPACK rutine [82].

Pri implementaciji paralelne verzije CPM programa je najpre potrebno uzeti u obzir tri činjenice. Prva se odnosi na način na koji se distribuiraju podaci koji su potrebni za rešavanje svojstvenog problema. Podaci koji su potrebni za ovaj proces su elementi matrica H i S, kao i skup pomoćnih struktura koji indeksira ove matrice. Kao što je u prethodnim sekcijama objašnjeno, matrice su organizovane u blokovima i svaki proces ima deo elemenata globalnih matrica. Drugo je činjenica da su matrice H i S retke strukture. Uzimajući ovo u obzir, potrebno je pronaći rešenje koje ima mogućnost rada sa distribuiranim podacima na paralelnoj arhitekturi i koje bi trebalo da bude posebno optimizovano za rad sa retkim matricama. Pored ovoga je neophodno da takav sistem ima opciju za rešavanja generalizovanog svojstvenog problema, tj. da ima podršku za rešavanje svojstvenog problema sa dve matrice. S obzirom da postoji širok spektar već razvijenih biblioteka koji su namenjene rešavanju problema linearne algebre, najpre je potrebno izvršiti analizu postojećih rešenja. Kao referentan repozitorijum softverskih rešenja za linearnu algebru se smatra NetLib-ova lista [111]. Uvidom u grupu softvera namenjenog za rešavanje svojstvenog problema nad retkim matricama iz NetLib repozitorijuma, je zaključeno da softverska biblioteka SLEPCs [112], [113] po svojoj specifikaciji jedina u potpunosti zadovoljava potrebne uslove.

Softverska biblioteka SLEPc (Scalable Library for Eigenvalue Problem Computations), razvijena na Universidad Politécnica de Valencia (Španija), je proširenje PETSc [107], [108] biblioteke i namenjena je za rešavanje svojstvenih problema za velike retke matrice na paralelnim računarskim platformama. Uz pomoć SLEPc je moguće rešavati kako standardne tako i generalizovane svojstvene probleme, sa operacijama nad kompleksnim ili realnim brojevima. Takođe postoji mogućnost rešavanja nelinearnih svojstvenih problema kao i neke druge mogućnosti koje nam ovde nisu od značaja. Pošto je SLEPc baziran na PETSc, iz PETSc-a se nasleđuju sve strukture podataka kao i podrška za MPI standard za komunikaciju.



Slika 4.11: Rešavanje svojstvenog problema uz pomoć SLEPc/PETSc rutina

U listingu 4.18 je dat pseudokod koji opisuje implementaciju procesa dijagonalizacije. Standardna procedura za korišćenje SLEPc-a je da se najpre inicijalizuju parametri koji određuju tip problema koji se rešava, strukture podataka koje se koriste kao i metod za rešavanje iz grupe metoda koji odgovaraju tipu problema. U našem slučaju je potrebno kreirati dve globalne PETSc matrice H i S. Ove matrice imaju globalne dimenzije, međutim s obzirom da se radi o distribuiranoj arhitekturi i da su elementi matrica raspoređeni u blokovima po odgovarajućim procesima, globalna struktura je samo interfejs koji opisuje distribuirane podatke. Da bi izvršavanje SLEPc algoritama nad ovim matricama bilo efikasno, memorijske strukture je potrebno inicijalizovati tako da svaki proces mapira elemente globalnih struktura koji odgovaraju podacima bloka koji 'poseduje'. Ovaj proces je prikazan na slici 4.11, kao i u listingu 4.18, gde se za svaki elemenat odgovarajućih blokova matrica poziva PETSc rutina $MatSetValue()$.

Listing 4.18: Pseudokod implementacije algoritma za rešavanje generalizovanog svojstvenog problema

```
... // Inicijalizacija SLEPc
... // Kreiranje globalnih PETSc matrica H i S

if(my_rank == 0){
  start = 0;
}else{
  start = blocks[my_rank-1];
}
end = blocks[my_rank];
int index = 0;
int countEl = 0;
for (i=start;i<end;i++) {
   int j;
   for(j = 0; j <= i; j++){
      if(getValueCSR(index, j, withinRadiusDist_CSR_offsets, withinRadiusDist_CSR_col)){
            ierr = MatSetValue(A,i,j,H[countEl],INSERT_VALUES);CHKERRQ(ierr);
            ierr = MatSetValue(A,j,i,H[countEl],INSERT_VALUES);CHKERRQ(ierr);
            ierr = MatSetValue(B,i,j,S[countEl],INSERT_VALUES);CHKERRQ(ierr);
            ierr = MatSetValue(B,j,i,S[countEl],INSERT_VALUES);CHKERRQ(ierr);
            countEl++;
      }
   }
   index++;
```

```
}

double target = -0.115;
int dim = 30;

if(!flg && molSize > 1000){ // Linearna zavisnost od veličine sistema
   dim = (double)((double)molSize / 1000) * 30;
}

if(flg){ // Argumenti su postavljeni od strane korisnika
   target = arg_target;
   dim = arg_dim;
}

ierr = EPSSetTarget(eps,target);CHKERRQ(ierr);
ierr = EPSSetDimensions(eps,dim,PETSC_DEFAULT,PETSC_DEFAULT); CHKERRQ(ierr);
ierr = EPSSetType(eps,EPSKRYLOVSCHUR);
... // Postavljanje ostalih parametara za SLEPs

ierr = EPSSolve(eps);CHKERRQ(ierr);

... // Stampanje izlaznih vrednosti
```

Rešavanje svojstvenog problema (naročito za velike sisteme) može biti resursno zahtevno. Jedan od načina da se smanji vreme izvršavanja je da se iskoristi činjenica da nije neophodno računati ceo spektar svojstvenih vrednosti. Za relevantan skup svojstvenih energija se smatraju one koje predstavljaju poslednja popunjena i prva prazna stanja (HOMO i LUMO orbitale). Standardna procedura podrazumeva da odabir odgovarajućeg opsega definiše korisnik. Za sisteme koji su testirani u ovom radu se uzima da su od fizičkog značaja ona stanja sa energijama u intervalu do $0.5eV$ (odnosno 0.02 Hartrija) iznad najnižeg praznog i do $0.5eV$ ispod najvišeg popunjenog. Pošto specificiranje opsega na ovaj način nije pravolinijsko, za osnovnu postavku se uzima da se za sve sisteme manje od 1000 atoma očekuje 30 vrednosti na izlazu, dok se za veće sisteme ova vrednost povećava linearno sa brojem atoma. Za postavljanje broja željenih vrednosti na izlazu se koristi SLEPCs rutina $EPSSetDimensions()$, dok se za postavljanje srednje vrednosti rezultata na izlazu koristi rutina $EPSSetTarget()$.

Sada kada se umesto celog spektra može definisati odgovarajući podskup vrednosti, za dijagonalizaciju je moguće upotrebiti Krylov-Schur metod [83], [84], koji se po SLEPc dokumentaciji smatra najefikasnijim za rešavanje ovog tipa problema [85]. Krylov-Schur metod predstavlja unapređenje tradicionalnog Krylov subspace metoda, koji sistemom takozvanog mehanizma za restartovanje računanja, eliminiše računanje komponenti koje se odnose na svojstvene vrednosti koje nisu u željenom opsegu. Na ovaj način je moguće odabrati bilo koji skup vrednosti iz spektra i time proporcionalno smanjiti vreme računanja. Da bi upotreba ovog metoda bila moguća, pored SLEPs i PETSc biblioteka je neophodan SuperLUDIST [114] paket za LU faktorizaciju.

Više o performansama dela algoritma za dijagonalizaciju kao i o uticaju različitih konfiguracija SLEPc i PETSc parametara na efikasnost izvršavanja programa će biti dato u poglavlju 5.

Poglavlje 5

Performanse i validacija rezultata

U ovom poglavlju je data analiza performansi i rezultata serijskog DFT i CPM programa (5.1), kao i paralelne verzije CPM programa (5.2). Obe sekcije najpre uvode klase sistema koji su testirani, zatim analiziraju rezultate testova i prikazuju zahtevnost izvršavanja. U delu koji se bavi testiranjem paralelne verzije CPM-a je posebna pažnja posvećena performansama izvršavanja (5.2.4) i to kroz dve grupe testova: zavisnost vremena izvršavanja CPM programa u odnosu na veličinu ulaznog sistema i jako skaliranje (strong scaling). Na kraju su uz analizu efikasnosti date smernice kako odabirati optimalni scenario upotrebe programa, u zavisnosti od ulaznih sistema i raspoloživih resursa (sekcija 5.2.5).

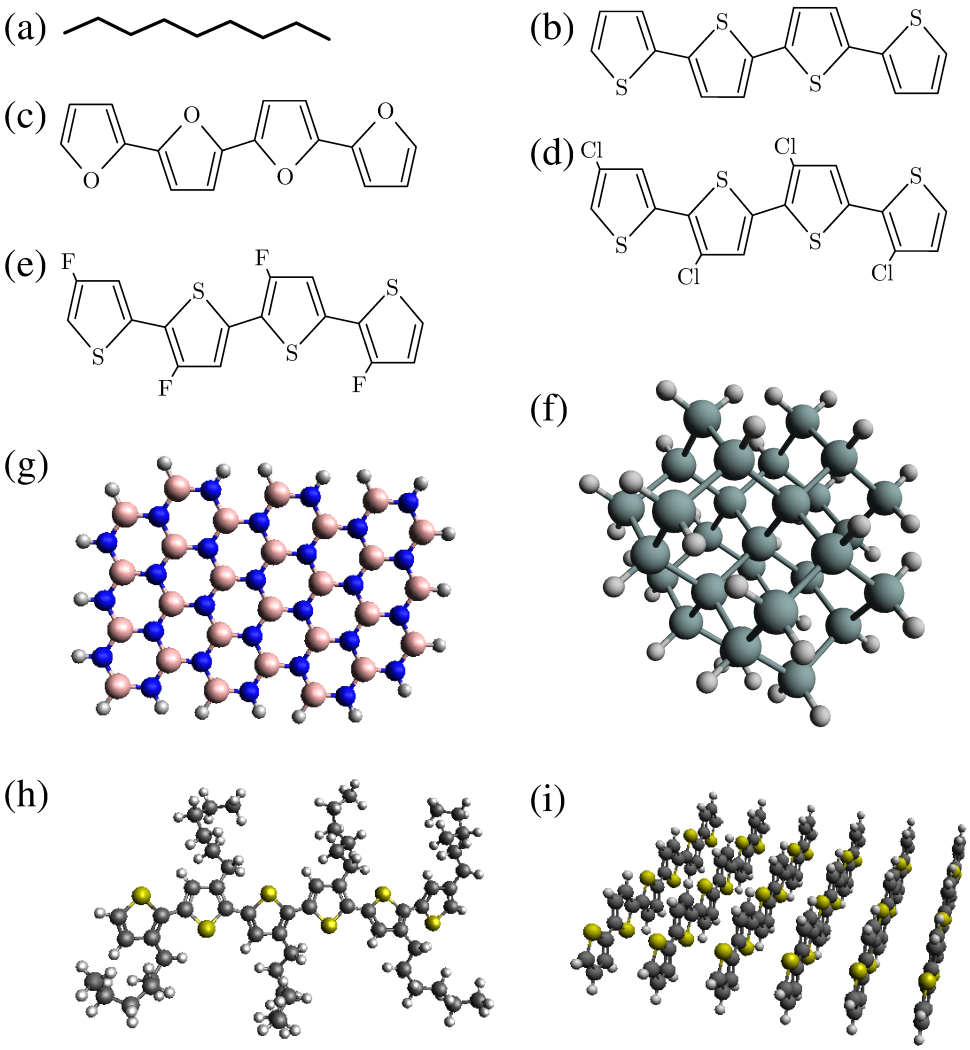
5.1 Performanse i validacija serijskih DFT i CPM programa

Serijski algoritam opisan u sekcijama 3.1 i 3.2 je implementiran u programskom jeziku C. Njegove performanse su testirane na računaru sa šestojezgarnim Intel Core i7-5820K procesorom i 64 GB DDR4 RAM-a. C kod je kompajliran na operativnom sistemu Linux uz pomoć GNU C kompajlera (GCC).

5.1.1 Klase testiranih sistema

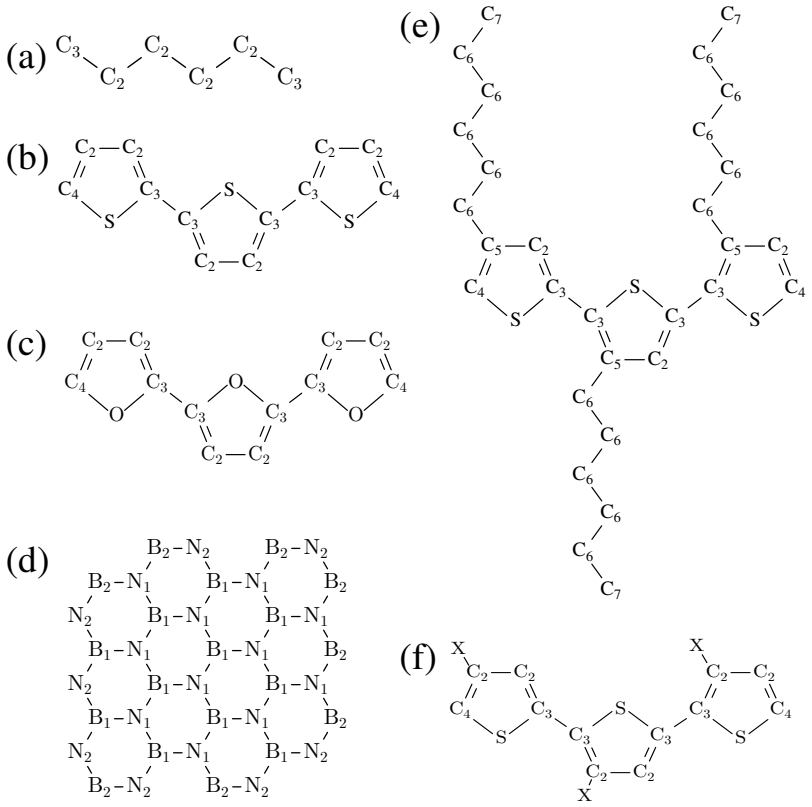
Testiranje je urađeno nad različitim fizičkim sistemima kao što su alkanski oligomeri (slika 5.1a), tiofen oligomeri [bez (slika 5.1b) i sa zamenom atoma vodonika atomom fluora (slika 5.1e) ili atomom hlora (slika 5.1d)], silicijumski nanokristali (slika 5.1f), furanski oligomeri (slika 5.1c), neuređeni poly(3-hexylthiophene) (P3HT) lanci (slika 5.1h) i nizovi politiofenskih lanaca (slika 5.1i). Ovakav skup sistema je izabran da bi se omogućilo testiranje algoritma na jednodimenzionalnim, dvodimenzionalnim i trodimenzionalnim sistemima. Ovakvim izborom su obuhvaćeni organski i neorganski sistemi, kako uređeni tako i neuređeni, koji su sastavljeni od hemijskih elementa iz različitih grupa iz periodnog sistema elemenata.

Da bismo potvrdili našu pretpostavku da se svojstvene vrednosti dobijene CPM programom dobro slažu sa rezultatima dobijenim uz pomoć DFT-a, najpre ćemo izvršiti analizu rezultata dobijenih navedenim metodama. Rezultati analize će ujedno biti i potvrda zaključaka vezanih za implementaciju CPM uz pomoć ravnih talasa, gde je pokazano dobro slaganje DFT i CPM rezultata [50]. Urađeno je poređenje za nekoliko različitih klasa izolatora i poluprovodničkih sistema –



Slika 5.1: Strukturna/atomska reprezentacija fizičkih sistema korišćenih za testiranje programa: (a) alkanski oligomeri; (b) tiofenski oligomeri; (c) furanski oligomeri; (d) tiofenski oligomeri sa fluorom kao zamenskim atomom; (e) tiofenski oligomeri sa hlorom kao zamenskim atomom ; (f) silicijumski nanokristali; (g) Bor nitridni nanosloj; (h) neuređeni P3HT; (i) niz politiofenskih lanaca.

alkanskih oligomera C_nH_{2n+2} gde su $n = 10$, $n = 20$ i $n = 40$, tiofen oligomera $C_{4n}S_nH_{2n+2}$ gde su $n = 3$, $n = 4$ i $n = 7$, niza tiofen oligomera $(C_{4n}S_nH_{2n+2})_m$ gde je $(n, m) = (3, 3); (3, 4); (4, 3); (4, 4)$, silicijumskih nanokristala $Si_{29}H_{36}$, $Si_{35}H_{36}$ i $Si_{59}H_{60}$, furan oligomera $C_{4n}O_nH_{2n+2}$ gde su $n = 3$, $n = 4$, $n = 5$, $n = 6$ i $n = 8$, tiofen oligomera gde je jedan atom vodonika zamenjen atomom hlora ($C_{4n}S_nCl_nH_{n+2}$ gde su $n = 3$, $n = 4$, i $n = 6$) ili fluora ($C_{4n}S_nCl_nH_{n+2}$ gde su $n = 3$, $n = 4$, i $n = 6$), bor nitridni nanosloj $B_{19}N_{19}H_{16}$, $B_{27}N_{27}H_{20}$ i $B_{34}N_{34}H_{22}$, i neuređenih P3HT sa 77, 102 i 152 atoma. Lista svih atoma i motiva korišćenih za svaki od navedenih sistema su predstavljeni u tabeli 5.1, dok je klasifikacija tipova atoma data u slici 5.2.



Slika 5.2: Klasifikacija tipova atoma za: (a) alkanske oligomere; (b) tiofenske oligomere; (c) furanske oligomere; (d) bor nitridni nanosloj; (e) P3HT; (f) tiofenske oligomere sa supstitucijom halogenim elementom.

5.1.2 Rezultati testova

Rezultati dobijeni rešavanjem alkanskih oligomera (C_nH_{2n+2}) gde su $n = 10$, $n = 20$ i $n = 40$ su navedeni u Tabeli 5.2. Radi jednostavnijeg prikaza je navedeno samo deset najviših popunjenih elektronskih stanja. Motivi koji su najpre generisani uz pomoć DFT-a na molekulu $C_{10}H_{22}$, su zatim korišćeni za računanje CPM-a za sisteme različitih veličina. Rezultati predstavljeni u tabeli 5.2 pokazuju da je razlika između svojstvenih energija dobijenih uz pomoć DFT-a i onih dobijenih uz pomoć CPM-a manja od 1 mHa. Da bismo proverili preciznost rezultata dobijenih uz pomoć CPM-a za širu klasu sistema, za svaki od njih računamo srednju kvadratnu razliku svojstvenih energija popunjenih elektronskih stanja dobijenih iz CPM-a i onih dobijenih iz DFT-a. Dobijeni rezultati su prikazani u tabeli 5.3 i oni pokazuju da je greška u svojstvenim energijama reda veličine 1 mHa.

sistem	tip atoma	motivi	sistem za generisanje motiva
alkanski oligomeri	C_2, C_3, H	$C_3 - C_2HHH$, $C_2 - C_3C_2HH$, $C_2 - C_2C_2HH$, $H - C_3 - C_2HH$, $H - C_2 - C_3C_2H$, $H - C_2 - C_2C_2H$	$C_{10}H_{22}$
silicijumski nanokristali	Si, H	$Si - SiSiSiSi$, $Si - SiSiSiH$, $Si - SiSiHH$, $H - Si - SiSiSi$, $H - Si - SiSiH$	$Si_{29}H_{36}$
bor nitridni nanosloj	B_1, B_2, H N_1, N_2	$B_1 - N_1N_1N_1$, $B_1 - N_2N_1N_1$, $B_1 - N_2N_2N_1$, $B_2 - N_1N_1H$, $B_2 - N_2N_1H$, $B_2 - N_2N_2H$, $H - B_2 - N_1N_1$, $H - B_2 - N_2N_1$, $H - B_2 - N_2N_2$, $H - N_2 - B_1B_1$, $H - N_2 - B_2B_1$, $H - N_2 - B_2B_2$, $N_1 - B_1B_1B_1$, $N_1 - B_2B_1B_1$, $N_1 - B_2B_2B_1$, $N_2 - B_1B_1H$, $N_2 - B_2B_1H$, $N_2 - B_2B_2H$	$B_{19}N_{19}H_{16}$
tiofenski oligomeri	C_2, C_3, C_4, H, S	$C_2 - C_3C_2H$, $C_3 - C_3C_2S$, $C_2 - C_4C_2H$, $C_4 - C_2SH$, $S - C_3C_3$, $S - C_4C_3$, $H - C_2 - C_3C_2$, $H - C_2 - C_4C_2$, $H - C_4 - C_2S$	$C_{12}S_3H_8$
furanski oligomeri	C_2, C_3, C_4, H, O	$C_2 - C_3C_2H$, $C_3 - C_3C_2O$, $C_2 - C_4C_2H$, $C_4 - C_2OH$, $O - C_3C_3$, $O - C_4C_3$, $H - C_2 - C_3C_2$, $H - C_2 - C_4C_2$, $H - C_4 - C_2S$	$C_{12}O_3H_8$
tiofeni sa substitucijom halogenim elementom	C_2, C_3, C_4, H, S, X	$C_2 - C_3C_2H$, $C_3 - C_3C_2S$, $C_2 - C_4C_2X$, $C_4 - C_2SH$, $S - C_3C_3$, $S - C_4C_3$, $H - C_2 - C_3C_2$, $X - C_2 - C_4C_2$, $X - C_2 - C_3C_2$, $H - C_4 - C_2S$, $C_2 - C_3C_2X$	$C_{12}S_3X_3H_5$
neuređeni P3HT	C_2, C_3, C_4, C_5, C_6, C_7, H, S	$C_5 - C_6C_4C_2$, $C_2 - C_5C_3H$, $C_3 - C_3C_2S$, $C_4 - C_5SH$, $C_3 - C_5C_3S$, $C_5 - C_6C_3C_2$, $C_4 - C_2SH$, $C_2 - C_5C_4H$, $S - C_3C_3$, $S - C_4C_3$, $C_6 - C_6C_5HH$, $C_6 - C_6C_6HH$, $C_7 - C_6HHH$, $C_6 - C_7C_6HH$, $H - C_2 - C_5C_3$, $H - C_4 - C_5S$, $H - C_2 - C_5C_4$, $H - C_4 - C_2S$, $H - C_6 - C_6C_5H$, $H - C_6 - C_6C_6H$, $H - C_7 - C_6HH$, $H - C_6 - C_7C_6H$	neuređeni P3HT sa 77 atoma

Tabela 5.1: Lista tipova atoma i motiva koji se koriste u CPM-u. Atomima se dodeljuju odgovarajući tipovi na način kao što je prikazano na slici 5.2.

Na osnovu toga, kada uporedimo rezultate dobijene uz pomoć CPM-a sa onim dobijenim uz pomoć DFT-a, možemo zaključiti da CPM daje dovoljno precizne vrednosti svojstvenih energija.

$C_{10}H_{22}$		$C_{20}H_{42}$		$C_{40}H_{82}$	
CPM	DFT	CPM	DFT	CPM	DFT
−0.309603	−0.309024	−0.278819	−0.278651	−0.275738	−0.276441
−0.290277	−0.289713	−0.278804	−0.278263	−0.275696	−0.276399
−0.290144	−0.289644	−0.278742	−0.277949	−0.275376	−0.275761
−0.281836	−0.281425	−0.278170	−0.277590	−0.275333	−0.275743
−0.279675	−0.279398	−0.276962	−0.276825	−0.267867	−0.268255
−0.279312	−0.278751	−0.276230	−0.275420	−0.252784	−0.253185
−0.278589	−0.277850	−0.275932	−0.275314	−0.239152	−0.239529
−0.276855	−0.276513	−0.261474	−0.260857	−0.227590	−0.227898
−0.275733	−0.275079	−0.235925	−0.235267	−0.218769	−0.218922
−0.230841	−0.229995	−0.218384	−0.217236	−0.213411	−0.213216

Tabela 5.2: Poređenje svojstvenih energija (izraženim u Ha) $C_{10}H_{22}$, $C_{20}H_{42}$ i $C_{40}H_{82}$ sistema, dobijenim uz pomoć CPM i DFT programa implementiranih u bazisu Gausijana.

sistem	$C_{10}H_{22}$	$C_{20}H_{42}$	$C_{40}H_{82}$	$C_{12}S_3H_8$	$C_{16}S_4H_{10}$	$C_{28}S_7H_{16}$
$\delta\varepsilon$ (mHa)	0.58	0.58	0.64	0.58	0.36	0.55
sistem	$Si_{29}H_{36}$	$Si_{35}H_{36}$	$Si_{59}H_{60}$	$C_{12}S_3Cl_3H_5$	$C_{16}S_4Cl_4H_6$	$C_{24}S_6Cl_6H_8$
$\delta\varepsilon$ (mHa)	1.8	0.50	2.1	1.0	1.4	1.8
sistem	$C_{12}O_3H_8$	$C_{16}O_4H_{10}$	$C_{20}O_5H_{12}$	$C_{24}O_6H_{14}$	$C_{32}O_8H_{18}$	$(C_{12}S_3H_8)_3$
$\delta\varepsilon$ (mHa)	1.4	1.6	2.0	1.8	2.0	0.69
sistem	$(C_{12}S_3H_8)_4$	$(C_{16}S_4H_{10})_3$	$(C_{16}S_4H_{10})_4$	$C_{12}S_3F_3H_5$	$C_{16}S_4F_4H_6$	$C_{24}S_6F_6H_8$
$\delta\varepsilon$ (mHa)	0.74	0.89	1.3	0.65	0.82	1.5
sistem	$B_{19}N_{19}H_{16}$	$B_{27}N_{27}H_{20}$	$B_{34}N_{34}H_{22}$	d-P3HT-77	d-P3HT-102	d-P3HT-152
$\delta\varepsilon$ (mHa)	1.1	1.1	1.3	3.5	3.6	3.4

Tabela 5.3: Srednja kvadratna razlika svojstvenih energija $\delta\varepsilon = \sqrt{\left\langle (\Delta\varepsilon)^2 \right\rangle}$ popunjenih elektronskih stanja dobijenih CPM i DFT programom implementiranim u bazisu Gausijana, za različite sisteme.

5.1.3 Memorijska zahtevnost

Sledeće što je potrebno razmotriti u vezi sa našom implementacijom CPM-a je alokacija memorijskih resursa. U serijskoj implementaciji CPM algoritma kompletne matrice H i S se čuvaju u operativnoj memoriji. Najveću dimenziju ovih matrica, koje smo imali u našim testiranim primerima, je 7392×7392 i to za slučaj nanokristala $Si_{837}H_{348}$, kada imamo i najveće zauzeće operativne memorije. Čak i u ovom slučaju, matrice se mogu skladištiti u RAM-u jednog računarskog noda, zbog čega u ovoj implementaciji nije postojala potreba za njihovom distribucijom. Ono što bi svakako trebalo iskoristiti prilikom distribucije ovih matrica u paralelnoj implementaciji CPM algoritma, je da se radi

o retkim matricama. Za razliku od H i S matrica, zbog velikog broja Hartri integrala, memorijsku strukturu koja čuva njihove vrednosti zauzima značajnu količinu memorije (pogledati opis u sekciji 3.1.4). U slučaju $(ab|\ c)$ [Eq. (3.44)] imamo $N_b^2 N_c$ integrala, gde je N_b veličina bazisnog skupa za talasne funkcije a N_c veličina bazisnog skupa za gustinu naelektrisanja, što u slučaju većih sistema svakako prevazilazi okvire raspoložive RAM memorije prosečnog računara. Olakšavajuća okolnost u ovom slučaju je činjenica da se doprinos jednom izračunatog Hartri integrala može odmah dodati odgovarajućem elementu H matrice, pa ga zbog toga nije potrebno čuvati u memoriji. S druge strane, da ne bi dolazilo do velikog broja ponovljenih operacija, umesto pojedinačnog računanja, potrebno je razmotriti drugačiji pristup. U našoj implementaciji je ovo rešeno tako što se unapred računaju grupe ovih integrala, gde se broj elemenata u grupi određuje na osnovu veličine matrice i neke unapred određene maksimalne veličine (imajući u vidu memorijske kapacitete). Tako se za slučaj malih matrica odmah izvršavaju sva potrebna računanja i vrednosti svih integrala čuvaju u privremenoj strukturi podataka, čime se omogućava višestruki pristup podacima bez ponovljenih izračunavanja. U slučaju sistema sa velikim matricama, računa se samo određeni podskup vrednosti, koji se odmah zatim 'pridodaje' matrici H. Nakon toga je moguće osloboditi memoriju, preuzeti novu grupu integrala za računanje i ponoviti operaciju, sve dok se ne izračunaju sve vrednosti.

5.1.4 Performanse

Sledeće što analiziramo je skaliranje vremena izvršavanja programa u odnosu na veličinu sistema. Analiza je urađena uzimajući u obzir doprinose različitih delova koda, koji značajno utiču na ukupno vreme izvršavanja. U skladu sa tim ćemo razmotriti sledeće programske celine:

(a) računanje kinetičkog, integrala preklapanja i integrala jezgra;

(b) računanje integrala pseudopotencijala;

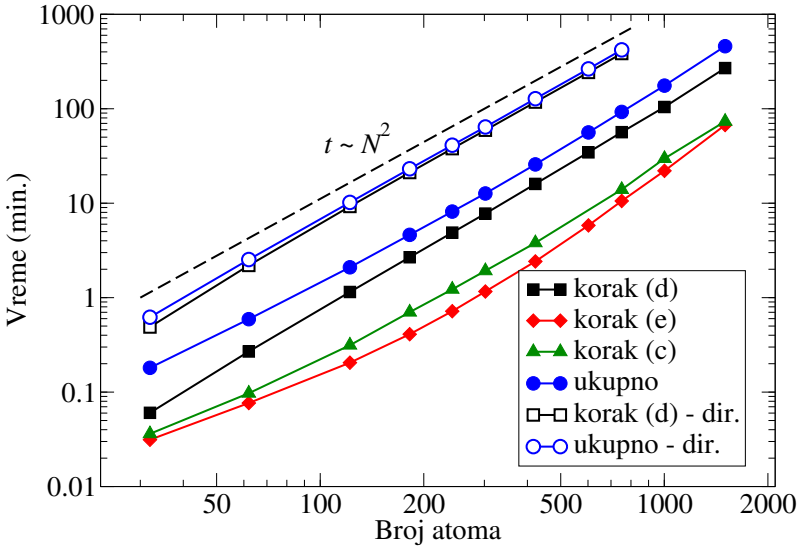
(c) učitavanje motiva i računanje reprezentacije gustine naelektrisanja u bazisu Gausijana;

(d) računanje integrala Hartri potencijala;

(e) računanje izmensko-korelacionog integrala;

(f) dijagonalizacija Hamiltonijana.

Za implementaciju (d) su uzeta u obzir dva različita pristupa – direktni pristup baziran na analitičkim formulama i pristup baziran na rekurentnim formulama koji je opisan u sekciji 3.1.4, pa će tako za prikaz vremena izvršavanja biti navedena oba slučaja. Testiranjem je utvrđeno da su tri vremenski najzahtevnija dela koda (c), (d), (e), pa smo u slučaju računanja alkalnih oligomera upravo njih odabrali za prikaz na slici 5.3. Kod ukupnog vremena izvršavanja, kao i kod pojedinačnih vremena za (c), (d) i (e), za velike vrednosti N, imamo skaliranje $t \sim N^2$ (prikaz vremena izvršavanja i broja atoma je u svim dijagramima dat na logaritamskoj skali). Za slučaj integrala Hartri potencijala (korak (d)), ovo skaliranje potiče iz činjenice da se broj integrala potreban za računanje skalira $O(N^2)$. Kod koraka (c) i (e) je potrebno izračunati elektronsku gustinu naelektrisanja i Gausijane u svakoj tački iz prostorne mreže, pa otuda i $t \sim N^2$ skaliranje. Na slici 5.3 se vidi da je vreme koraka (d) dominantno. Upravo iz tog razloga smo uložili značajan napor da bismo optimizovali izvršavanje ovog dela i time smanjili vreme izvršavanja. Detalji su opisani

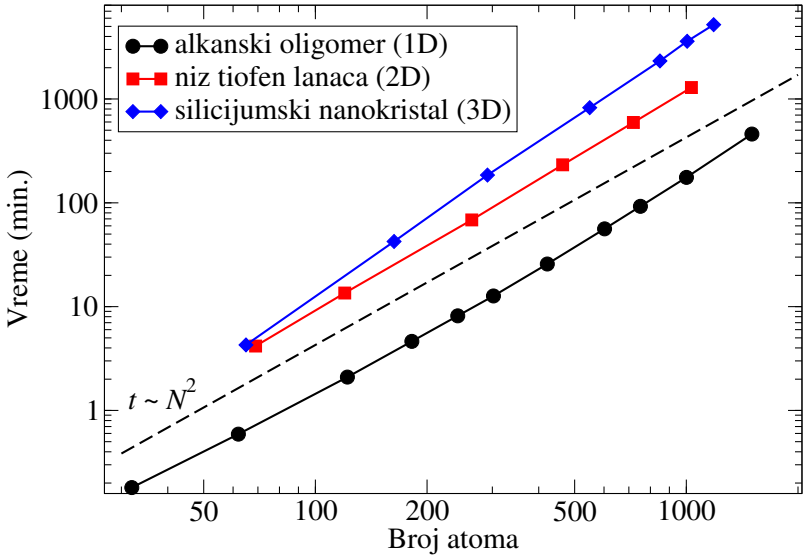


Slika 5.3: Zavisnost vremena izvršavanja od broja atoma za vremenski najzahtevnije delove programa (za alkanske oligomere): korak (c) – čitanje motiva i računanje gustine naelektrisanja u reprezentaciji bazisa Gausijana (popunjeni trouglovi); korak (d) – računanje integrala Hartri potencijala u dva slučaja - korišćenjem rekurentnih formula (puni kvadrati) i direktnih analitičkih formula (prazni kvadrati); korak (e) – računanje izmensko-korelacionog integrala (dijamanti); i ukupno vreme za oba slučaja (puni krugovi i prazni krugovi). Isprekidana linija koja prikazuje $t \sim N^2$ je data kao vizuelni orijentir.

u sekciji 3.1.4. Pozitivan rezultat ovih optimizacija se može videti u značajnoj razlici vremena za računanje Hartri integrala primenom direktnih analitičkih formula i vremena za izvršavanje rekurentnih relacija 5.3.

Na slici 5.4 je prikazano skaliranje vremena izvršavanja u odnosu na veličinu sistema različitih dimenzija: jednodimenzionalni alkanski oligomeri, dvodimenzionalni niz tiofen oligomera i trodimenzionalni silicijumski nanokristal. Rezultati pokazuju da se složenost računanja sistema povećava sa povećanjem dimenzije. Ova zavisnost potiče iz činjenice da u višedimenzionalnim sistemima atomi imaju više suseda i samim tim se povećava i broj integrala koji bi trebali da budu izračunati. Ipak i pored povećanja kompleksnosti, na računaru prosečnih performansi, u okvirima od nekoliko sati, je moguće izračunati sisteme sa više od hiljadu atoma. U slučaju jednodimenzionalnih i dvodimenzionalnih sistema imamo skaliranje od $\sim N^2$, dok je u slučaju trodimenzionalnog silicijumskog nanokristala (za slučajeve koje smo testirali) skaliranje nešto lošije. Ovakvo lošije skaliranje potiče od činjenice da za ispitivanje vrednosti N broj integrala koji treba da se računa u slučaju trodimenzionalnih sistema još uvek nije dostigao $\sim N^2$ skaliranje koje se očekuje za veliko N.

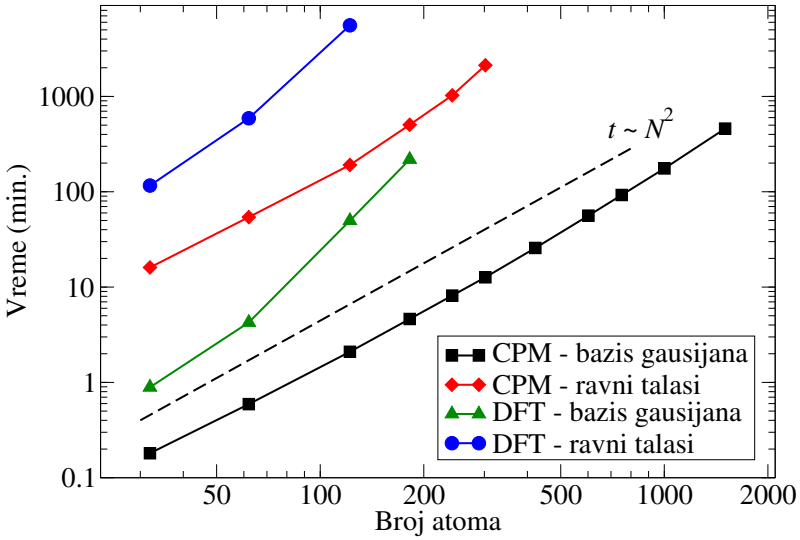
Na slici 5.5 poredimo vreme izvršavanja CPM-a baziranog na bazisima Gausijana sa vremenom izvršavanja prethodne implementacije CPM-a baziranog na ravnim talasima (za alkanske oligomere). U implementaciji sa ravnim talasima veći deo vremena izvršavanja se utroši na dijagonalizaciju Hamiltonijana, dok konstrukcija elektronske gustine naelektrisanja i jednočestičnog potencijala uzima zanemarljiv deo ukupnog vremena. Vremena prikazana na slici 5.5 su dobijena izvršavanjem dijagonalizacije korišćenjem algoritma baziranog na konjugovanim gradijentima, implementiranog u ESCAN [55] programu. Nasuprot tome, u slučaju implementacije bazirane na bazisu Gausijana je



Slika 5.4: Zavisnost vremena izvršavanja CPM programa u odnosu na broj atoma za alkanske oligomere, dvodimenzionalni niz politiofen lanaca i trodimenzionalni silicijumski nanokristal. Isprekidana linija koja prikazuje $t \sim N^2$ je data kao vizuelni orijentir.

najviše vremena potrebno za konstrukciju Hamiltonijana (posebno za doprinos Hartri potencijala, kao što je već pomenuto), dok je za dijagonalizaciju potrebno zanemarljivo malo vremena u odnosu na ukupan proces. Razlog za ovo je mala veličina bazisnog skupa Gausijana, što je upravo glavni razlog superiornih performansi implementacije bazirane na bazisu Gausijana. Kao što se može videti na slici 5.5, ovo nam omogućava da za samo nekoliko sati rešimo alkanski sistem sa više od 1500 atoma, koristeći samo jedno jezgro procesora. Takođe treba napomenuti da je rezultat koji se odnosi na vreme računanja u skladu sa prethodno navedenom činjenicom da konstruisanje Hamiltonijana, odnosno računanje gausijanskih integrala, predstavlja ograničavajući faktor u računanju elektronske strukture u bazisu Gausijana [93, 96, 94, 97], dok u slučaju računanja Hamiltonijana baziranom na ravnim talasima ograničavajući faktor predstavlja dijagonalizacija tehnikom konjugovanih gradijenata [6]. Dalje, pošto smo ustanovili da imamo dobro slaganje rezultata CPM-a i DFT-a baziranih na bazisu Gausijana, kao i što znamo da imamo dobro slaganje rezultata DFT-a i CPM-a baziranih na ravnim talasima, možemo pretpostaviti da eventualne male razlike između CPM-a baziranog na bazisu Gausijana i CPM-a baziranog na ravnim talasima mogu da potiču iz činjenice da su za različite implementacije upotrebljeni različiti bazisni skupovi. Ovo su upravo i potvrdile razlike u rezultatima svojstvenih energija, testiranjem navedenih implementacija koje su prikazane u tabeli 5.4. Takođe, pored razlike u bazisima, pokazalo se da na razliku u rezultatima takođe može da utiče i razlika u korišćenim pseudopotencijalima u dve različite implementacije.

Na slici 5.5 je prikazano poređenje vremena izvršavanja DFT-a i CPM-a za implementacije bazirane na bazisu Gausijana i na ravnim talasima. Testiranje je urađeno za sisteme alkanskih oligomera. Za računanje DFT-a je korišćen PETOT [26] baziran na ravnim talasima i naša implementacija DFT-a bazirana na bazisima Gausijana. Glavna prednost CPM-a u odnosu na DFT izračunavanje je u tome što CPM implementacija eliminiše potrebu za iteracijama potrebnim za samosaglasno rešavanje jednačina (2.1) i (2.2). Stoga možemo grubo proceniti da bi DFT program trebao biti onoliko puta sporiji od CPM-a koliko imamo iteracija u DFT-u. Ovo nije u potpunosti



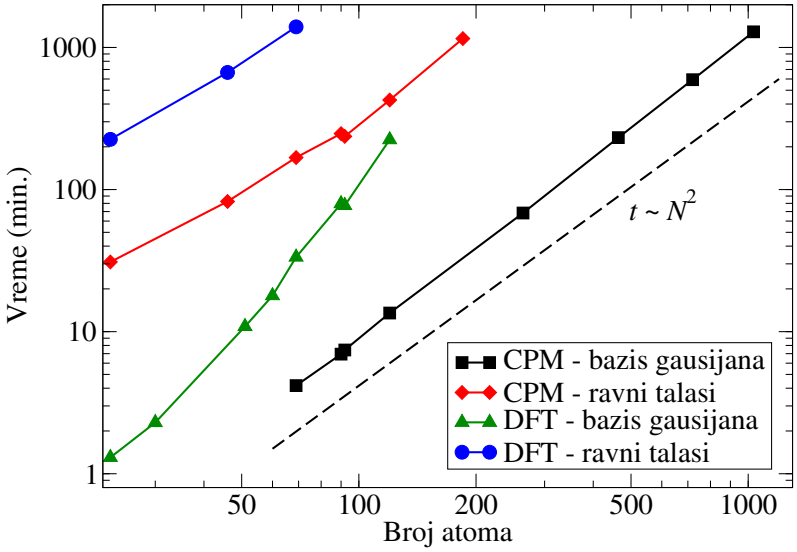
Slika 5.5: Poređenje zavisnosti vremena izvršavanja od broja atoma (alkanski oligomeri) za implementacije CPM-a i DFT-a baziranim na bazisima Gausijana i ravnim talasima. Isprekidana linija koja prikazuje $t \sim N^2$ je data kao vizuelni orijentir.

sistem	$C_{10}H_{22}$	$C_{20}H_{42}$	$C_{40}H_{82}$
$\delta\varepsilon$ (mHa)	1.6	1.7	1.7
sistem	$Si_{29}H_{36}$	$Si_{59}H_{60}$	$Si_{87}H_{76}$
$\delta\varepsilon$ (mHa)	3.4	3.1	3.3
sistem	$C_{12}S_3H_8$	$(C_{12}S_3H_8)_3$	$(C_{16}S_4H_{10})_4$
$\delta\varepsilon$ (mHa)	2.1	2.1	2.0

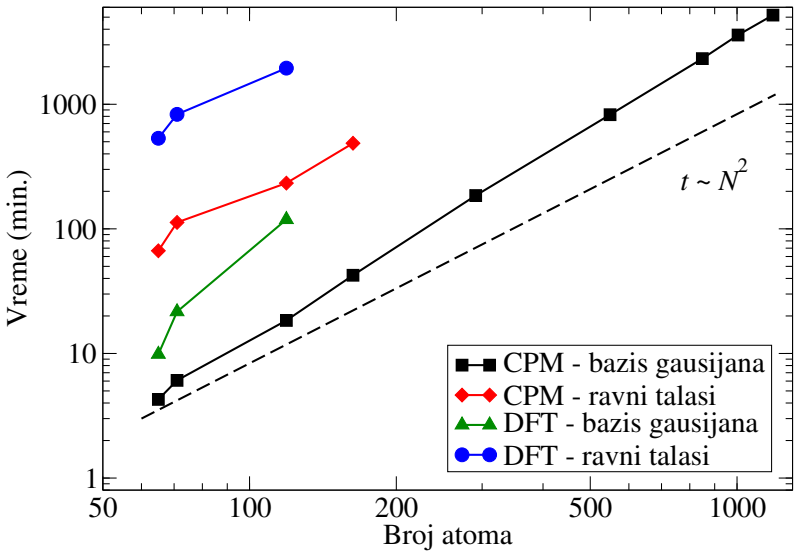
Tabela 5.4: Srednja vrednost kvadrata razlike svojstvenih energija $\delta\varepsilon = \sqrt{\left\langle (\Delta\varepsilon)^2 \right\rangle}$ popunjenih elektronskih stanja dobijenih izvršavanjem CPM programa implementiranim u bazisu Gausijana i CPM programom implementiranim u ravnim talasima.

slučaj za implementaciju baziranu na bazisima Gausijana, jer za računanje DFT-a postoje operacije koje ne moraju biti ponovljene u svakoj iteraciji. Tako na primer računanje Hartri integrala koji spada među vremenski najzahtevnije operacije, nije potrebno ponavljati u svakoj DFT iteraciji, jer se radi o istim integralima. Sa druge strane, računanje izmensko-korelacionog integrala zahteva ponovljena računanja, jer je gustina naelektrisanja različita u svakoj iteraciji. Pored ovoga, DFT program takođe sadrži implementaciju jednačine (2.2), koja oduzima značajan deo vremena za slučaj baziran na bazisima Gausijana. Kao posledica svih ovih razmatranja, možemo reći da je CPM implementacija bazirana na bazisu Gausijana bar pet puta brža od DFT programa (pogledati sliku 5.5), dok se za testirane sisteme broj DFT iteracija kreće između 20 i 35. Za veće sisteme (između 100 i 200 atoma) ova razlika vremena izračunavanja postaje čak i veća i to uglavnom zbog vremena potrebnog za računanje jednačine (2.2). Pritom, za velike sisteme nije moguće čuvati sve Hartri integrale zbog ograničenih memorijskih resursa, pa ih je tako u ovom slučaju potrebno iznova računati u svakoj DFT iteraciji. Za takve sisteme bi svakako važilo da bi CPM u bazisu Gausijana zaista bio brži onoliko puta koliko imamo iteracija u DFT-u. Na slikama 5.6 i 5.7 je prikazano

poređenje performansi DFT-a i CPM-a baziranih na bazisima Gausijana i na ravnim talasima za slučajeve sa nizovima lanaca politiofena i za silicijumske nanokristale koji se redom mogu smatrati za predstavnike dvodimenzionalnih i trodimenzionalnih sistema. Na osnovu rezultata dobijenih na tim primerima možemo potvrditi zaključke dobijene na osnovu testiranja alkanskih oligomera.



Slika 5.6: Poređenje zavisnosti vremena izvršavanja od broja atoma za slučaj dvodimenzionalnih nizova politiofen lanaca u implementaciji CPM-a i DFT-a bazirane na bazisima Gausijana i na ravnim talasima. Isprekidana linija koja prikazuje $t \sim N^2$ je data kao vizuelni orijentir.



Slika 5.7: Poređenje zavisnosti vremena izvršavanja od broja atoma za silicijumske nanokristale u implementaciji CPM-a i DFT-a baziranim na bazisima Gausijana i na ravnim talasima. Isprekidana linija koja prikazuje $t \sim N^2$ je data kao vizuelni orijentir.

5.2 Performanse i validacija paralelnog CPM programa

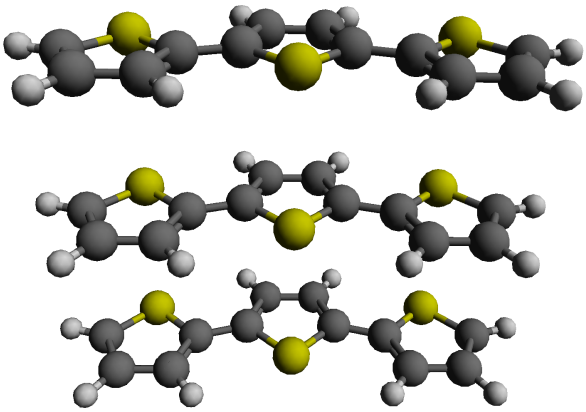
Testiranje sistema je izvršeno na dva računarska sistema visokih performansi: PARADOX-IV i AXIOM. U svrhu merenja performansi i validacije paralelnog CPM programa je korićen PARADOX-IV klaster Laboratorije za primenu računara u nauci, u okviru Centra za izučavanje kompleksnih sistema Instituta za fiziku u Beogradu. PARADOX-IV klaster se sastoji od 106 računarskih nodova povezanih u InfiniBand QDR mrežu (40 GB/s), od kojih svaki ima dva Sandy Bridge Xeon ES-2670 2.6 GHz procesora sa po 8 jezgara (16 jezgara po nodu), 32 GB DDR3 RAM memorije i Nvidia Tesla M2090 grafičku karticu sa 6 GB GPU RAM-a. Operativni sistem klastera je Scientific Linux 6.4 (Carbon). Testirani MPI CPM program je kompajliran sa Open MPI (verzija 3.0.0), dok je od programskih modula na klasteru korišćen OpenBlas (verzija 0.2.8), PETSC (verzija 3.10.3) i SLEPC (verzija 3.10.1) kao i GSL (verzija 1.13). Ukupno CPU vreme računarskog klastera PARADOX-IV, koje je bilo potrebno za sprovođenje testova, prikazanih u ovom poglavlju, je približno 80000 CPU sati (CPU vreme se odnosi na ukupno vreme alokacije pojedinačnih CPU jezgara u toku izvršavanja programa).

Testiranje manjih sistema i poređenje performansi različitih verzija implementacija u toku razvoja CPM programa je vršeno na AXIOM klasteru Istraživačke grupe za primenu računara u nauci (SCORG/SPRUN) na Prirodno-matematičkom fakultetu, Univerziteta u Novom Sadu. AXIOM klaster se sastoji od 16 računarskih nodova povezanih u 10 Gbps računarsku mrežu, od kojih svaki ima Intel i7 (8×5820k 3.3GHz, 8×8700k 3.2 GHz) procesor sa 6 jezgara i 16 GB DDR4 RAM-a. 8 računarskih nodova ima Nvidia GTX960 grafičku karticu sa 2 GB GPU DDR5 RAM-a. Operativni sistem AXIOM klastera je Scientific Linux 7.5 (Nitrogen). MPI CPM program je na AXIOM klasteru kompajliran sa Open MPI (verzija 3.1.1), dok je od programskih biblioteka korišćen OpenBlas (verzija 0.3.3), PETSC (verzija 3.9.3) i SLEPC (verzija 3.9.2) kao i GSL (verzija 1.15).

5.2.1 Klase testiranih sistema

Za testiranje paralelne verzije CPM-a su korišćeni nizovi politiofenskih lanaca $(C_{4n}S_nH_{2n+2})_k$ veličina od 69 do blizu 20000 atoma. Najmanji sistem sa 69 atoma $(C_{12}S_3H_8)_3$, čija je atomska reprezentacija prikazana na slici 5.8, je ujedno i sistem na kome su generisani motivi za CPM program. Iako se za prototip sistem može uzeti proizvoljno odabrani $C_{12}S_3H_8$ iz proizvoljno odabranog sistema dimenzije (n, k), nakon analize rezultata dobijenih nad nizovima različitih dimenzija sa politiofenskim lancima različitih dužina, zaključeno je da se najprecizniji rezultat (u odnosu na rezultat dobijen uz pomoć DFT programa) dobija kada se skup motiva formira na srednjem lancu sistema $(C_{12}C_3H_8)_3$. Način formiranja motiva se sprovodi na isti način kao i za tiofen oligomer, prikazan u tabeli 5.1

Za razliku od testova koji su rađeni za serijski DFT i serijski CPM, gde su korišćeni sistemi iz više različitih klasa, za potrebe testiranja paralelne verzije CPM-a se opredeljujemo za jednu klasu organskih polimera. Razlog tome je što je, u slučaju paralelnog CPM-a, osnovni motiv da se prikažu mogućnosti skaliranja i efikasnost CPM algoritma, dok smatramo da se on može efikasno primeniti nad bilo kojom klasom koja je testirana serijskim programima. Ovaj zaključak se može obrazložiti na osnovu pojedinačne analize svih algoritamskih izmena paralelne verzije koda (u odnosu na serijsku), u cilju procene njihovog uticaja na rezultate i performanse kad se primene na različite klase ulaznih sistema. S druge strane, potencijalno odstupanje tačnosti rezultata (uzrokovano algorita-



Slika 5.8: Atomska reprezentacija niza od tri politiofenska lanca sa po tri prstena $(C_{12}S_3H_8)_3$.

mskih nepravilnostima) usled skaliranja, bilo veličine sistema ili broja procesa, bi se podjednako ispoljilo nezavisno od odabranog sistema. Još jedan razlog zbog čega smatramo da testovi treba da budu sprovedeni za pažljivo odabran skup ulaznih sistema je što oni iziskuju značajne računarske resurse, te bi testiranje svih klasa sistema, na način koji je to urađeno za serijske programe, predstavljalo neadekvatno korišćenje istih.

5.2.2 Konfiguracija testova

Da bi se izvršio program na računarskom klasteru, potrebno je napisati skriptu koja sadrži spisak direktiva koji definišu zahtev za alokaciju potrebnih resursa, spisak biblioteka koje su potrebne za izvršavanje programa, kao i komandu, odnosno skup direktiva, kojima se pokreće izvršna verzija programa. Skript se zatim prosleđuje klasterskom sistemu za obradu zahteva (batch system). Primer skripte za pokretanje programa na PARADOX-IV klasteru je data u listingu 5.1. U listingu je najpre prikazan skup PBS (Portable Batch System) direktiva koji alocira po 16 CPU jezgara na 48 klasterskih nodova sa maksimalnim trajanjem izvršavanja od 4 sata i 30 minuta. Standardni ispis, kao i ispis grešaka se preusmerava na fajlove koji se čuvaju na lokaciji odakle je pokrenut izvršni program. Dalje se definišu putanje ka lokalno konfigurisanim softverskim bibliotekama i učitavaju se potrebni moduli iz liste standardne konfiguracije klastera. Poslednja komanda poziva MPI izvršni program uz dva parametra koja su prilagođena klasi sistema koji su testirani za potrebe ovog rada: prvi se odnosi na broj politiofen lanaca (i veličinu niza) a drugi na mod izvršavanja programa ((1) – distribuirani podaci/(2) – MPI deljena memorija).

Listing 5.1: Primer skripte sa PBS direktivama za pokretanje paralelnog CPM programa na računarskom klasteru PARADOX-IV

```
#!/bin/bash
#PBS -N cpm_s_25_768
#PBS -q standard
#PBS -l nodes=48:ppn=16
#PBS -l walltime=04:30:00
#PBS -e ${PBS_JOBID}.${PBS_JOBNAME}.err
#PBS -o ${PBS_JOBID}.${PBS_JOBNAME}.out
```

```
cd $PBS_O_WORKDIR
chmod +x dftGaussianParallel

export PETSC_DIR=/home/zbodroski/petsc-3.10.3
export SLEPC_DIR=/home/zbodroski/slepc-3.10.1
export PETSC_ARCH=arch-linux2-c-opt
module load c++/gnu/7.2.0
module load openmpi/3.0.0
module load openblas
module load gsl/1.13

mpirun ./cpm_parallel 25 2
```

5.2.3 Rezultati testova

Rezultati dobijeni izvršavanjem CPM programa za nizove politiofenskih lanaca $(C_{12}S_3H_8)_3$ i $(C_{16}S_4H_{10})_4$ su dati u tabeli 5.5. Razlog za odabir ova dva sistema je što su to najveći sistemi gde je testiran serijski DFT program, pa je moguće izvršiti poređenje sva tri različita načina rešavanja sistema: serijski DFT, serijski CPM i paralelni CPM program. Za prikaz je odabran opseg elektronskih stanja koji pripada skupu svojstvenih energija koje predstavljaju poslednja popunjena i prva prazna stanja (HOMO i LUMO orbitale). Ukoliko parametri koji određuju opseg nisu definisani od strane korisnika, uzima se da su od fizičkog značaja ona stanja sa energijama u intervalu do $0,5$eV (odnosno $0,02$ Hartrija) iznad najnižeg praznog i do $0,5$eV ispod najvišeg popunjenog. Kao što je obrazloženo u sekciji 4.3.7, za osnovnu postavku se uzima da se za sve sisteme manje od 1000 atoma očekuje 30 vrednosti na izlazu, dok se za veće sisteme ova vrednost povećava linearno sa brojem atoma. Radi jednostavnijeg prikaza, u tabeli 5.5 je dat podskup od 20 energija koji pripadaju odgovarajućem spektru za sisteme ovih veličina. Vrednosti odabranih elektronskih stanja pokazuju da su razlike između serijskog i paralelnog CPM-a manje od 1 μH, što svakako održava preciznost u odnosu na DFT rezultate, pa se može reći da paralelna verzija CPM programa (kao i serijska verzija) ima odstupanja u odnosu na DFT vrednosti manja od 1 mH. Razlog za razlike koje nastaju u rezultatima između dve verzije CPM-a se može tražiti u različitim implementacijama koje se tiču dijagonalizacije sistema linearnih jednačina u procesu ekstrahovanja motiva, kao i u procesu rešavanju svojstvenog problema (dijagonalizacija Hamiltonijana). Ova dva procesa se implementiraju uz pomoć PETSc i SLEPc biblioteka, gde se na tačnost rezultata može uticati podešavanjem parametara u PETSc i SLEPc rutinama, koje utiču na preciznosti izlaznih vrednosti. Međutim, kako su razlike manje od 1 μH, smatra se da ne utiču značajno na tačnost rezultata i da je preciznost na zadovoljavajućem nivou.

5.2.4 Performanse

Osnovno merilo za određivanje performansi CPM programa je vreme izvršavanja, pri čemu je potrebno uzeti u obzir doprinose različitih delova koda koji značajno utiču na ukupno vreme izvršavanja (po uzoru na analizu serijskih verzija programa). U skladu sa tim ćemo za slučaj paralelnog CPM programa razmotriti sledeće programske celine:

(a) računanje integrala preklapanja, kinetičkog integrala i integrala jezgra;

(b) efektivni potencijal jezgra (pseudopotencijal);

$(C_{12}S_3H_8)_3$			$(C_{16}S_4H_{10})_4$		
DFT	CPM ser.	CPM par.	DFT	CPM ser.	CPM par.
−0.223510	−0.223512	−0.223512	−0.220717	−0.219270	−0.219269
−0.219084	−0.219005	−0.219006	−0.219091	−0.217847	−0.217846
−0.217194	−0.217833	−0.217833	−0.214371	−0.213198	−0.213197
−0.215135	−0.214429	−0.214429	−0.211961	−0.209923	−0.209922
−0.207193	−0.206939	−0.206939	−0.204877	−0.203883	−0.203882
−0.177006	−0.176739	−0.176739	−0.201528	−0.201124	−0.201124
−0.171656	−0.172051	−0.172051	−0.194468	−0.193630	−0.193630
−0.162680	−0.162070	−0.162070	−0.187164	−0.185778	−0.185777
−0.086804	−0.086429	−0.086429	−0.170942	−0.169427	−0.169426
−0.079419	−0.079724	−0.079724	−0.168156	−0.167273	−0.167272
−0.068082	−0.067514	−0.067514	−0.161705	−0.160230	−0.160229
−0.050952	−0.050672	−0.050672	−0.154916	−0.152834	−0.152833
−0.042990	−0.043347	−0.043347	−0.092436	−0.090812	−0.090811
−0.030938	-0.030561	−0.030561	−0.088181	−0.087220	−0.087219
−0.012541	−0.012307	−0.012307	−0.079872	−0.078532	−0.078531
−0.004195	−0.004554	−0.004554	−0.071209	−0.069242	−0.069241
0.007065	0.007693	0.007693	−0.066144	−0.064928	−0.064927
0.008245	0.008474	0.008474	−0.061616	−0.060993	−0.060992
0.008868	0.009035	0.009035	−0.052926	−0.052047	−0.052046
0.009388	0.009520	0.009520	−0.043808	−0.042392	−0.042391

Tabela 5.5: Poređenje svojstvenih energija (izraženih u Ha) za nizove politiofenskih lanaca $(C_{12}S_3H_8)_3$ i $(C_{16}S_4H_{10})_4$ dobijenih uz pomoć serijskog DFT program, serijskog CPM programa i paralelnog CPM programa.

(c) ekstrahovanje motiva:

  (c-1) formiranje matrice odsecanja za Kulonove Gausijane;

  (c-2) učitavanje i fitovanje motiva;

  (c-3) rešavanje sistema linearnih jednačina;

(d) računanje integrala Hartri potencijala;

(e) izmensko-korelacioni integral:

  (e-1) formiranje matrice odsecanja za 'standardne' Gausijane;

  (e-2) računanje izmensko-korelacionog integrala;

(f) rešavanje svojstvenog problema (dijagonalizacija Hamiltonijana).

Delovi koda koji su izostavljeni iz prethodnog spiska imaju mali uticaj na ukupno vreme izvršavanja pa oni (bez obzira na funkcionalnu ulogu) neće biti uključeni u dalje razmatranje koje se tiče performansi. S druge strane, delovi koda kao što su implementacija ekstrahovanja motiva i izmensko-korelacionog potencijala, su zbog složenosti i značajnog uticaja na vreme izvršavanja podeljeni u

podgrupe, kako bi se detaljnijom analizom bolje razumeo uticaj pojedinačnih komponenti na ukupne performanse. Za grafičke prikaze će u nastavku teksta biti izdvojene tri programske celine: (c) – ekstrahovanje motiva, (d) – računanje integrala Hartri potencijala i (e) izmensko-korelacioni integral, dok će u tabelarnim prikazima biti prikazana pojedinačna vremena izvršavanja svih nabrojanih komponenti.

Merenje vremena izvršavanja MPI koda se vrši uz pomoć MPI rutine $MPI\_Wtime()$, koja ima visoku rezoluciju tačnosti i prikazuje lokalni 'wall-time' računarskog noda sa kojeg se poziva (provera sihnronizacije vremena između nodova se vrši uz pomoć rutine $MPI\_WTIME\_IS\_GLOBAL$). Rezolucija merenja vremena je u slučaju kompajlera koji je korišćen za većinu testova (Open MPI, verzija 3.0.0) $10^{-6}\,s$. Visoka preciznost $MPI\_Wtime()$, pored standardnog merenja brzine izvršavanja blokova koda, omogućava da se pojedinačni delovi koda, koji se učestalo ponavljaju i pojedinačno imaju kratko vreme izvršavanja, uspešno 'izoluju' iz programskih petlji (pogledati listing 5.2). Tako se, izdvajanjem iz programskih procesa, na vrlo precizan način mogu locirati vremenski zahtevni delovi koda.

Listing 5.2: Primena $MPI\_Wtime()$ rutine za merenje delova koda uz pomoć akululacije vremena

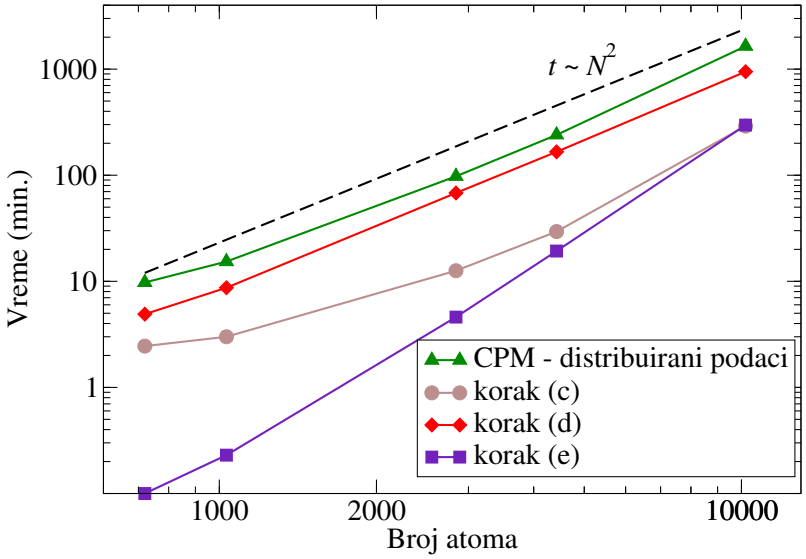
```
double counter = 0.0;
for(int i = 0; i < n; i++){
    for(int j = 0; j < m; j++){
        ...// Blok za koji se ne meri vreme izvršavanja
        counter -= MPI_Wtime();
        ...// Blok za koji se meri vreme izvršavanja
        counter += MPI_Wtime();
        ...// Blok za koji se ne meri vreme izvršavanja
    }
}
```

Testovi performansi CPM programa su podeljeni u dve grupe. Prva grupa testova ima za cilj da prikaže zavisnost vremena izvršavanja CPM programa u odnosu na veličinu ulaznog sistema. Da bi se ova zavisnost adekvatno prikazala, potrebno da svi test slučajevi budu izvršeni na istom broju paralelnih MPI procesa. S obzirom na veličinu sistema, raspoložive resurse i procenjeno vreme izvršavanja je odlučeno da testovi budu izvršeni na 16 klasterskih nodova, ondosno na 256 CPU jezgara. Druga grupa testova za cilj ima da prikaže jako skaliranje (strong scaling). Ovo podrazumeva fiksnu veličinu problema koji se rešava, dok se vrši skaliranje broja paralelnih procesa. Ova grupa testova je izvršena za dva sistema: nizovi tiofenskih polimera sa 1032 $[(C_{48}S_{12}H_{26})_{12}]$ i sa 4425 atoma $[(C_{100}S_{25}H_{52})_{25}]$. Najveći broj paralelnih procesa na kome su vršeni testovi je 768, odnosno 48 klasterskih nodova. Obe grupe testova su urađene za dva moda CPM programa: mod u kome se primenjuje sistem distribucije podataka gde se velike strukture distribuiraju kao kopije svim raspoloživim procesima i mod u kome se za iste memorijske strukture primenjuje sistem sa MPI deljenom memorijom (opisani u sekcijama 4.3.4 i 4.3.6). Iako način distribucije memorije ne utiče na sve nabrojane komponente koje se analiziraju, dve komponente gde su ovi principi primenjeni ((c) i (e)) spadaju u vremenski zahtevne delove, pa se radi pojednostavljenja ovi modovi uvode kao zasebni načini izvršavanja kompletnog CPM programa.

Prva grupa testova

Kao i u slučaju serijskog CPM programa, testiranjem je utvrđeno da su tri vremenski najzahtevnija dela koda (c), (d) i (e), pa su pored ukupnog vremena izvršavanja paralelnog CPM-a oni odabrani za prikaz na slikama 5.9 i 5.10. Detaljniji prikaz vremena izvršavanja pojedinačnih koraka je dat u tabeli 5.6. Slika 5.9 se odnosi na mod (1) sa distribuiranim podacima, dok se slika 5.10 odnosi na mod (2) sa MPI deljenom memorijom. Na slici 5.11 su upoređene vrednosti ukupnog vremena izvršavanja CPM programa za ova dva moda. Prikaz vremena izršavanja i broja atoma je u svim dijagramima dat na logaritamskoj skali. Kao što je obrazloženo u sekciji 5.1, priroda algoritma je takva da se broj operacija svodi na složenost $O(N^2)$. Otuda za velike sisteme možemo očekivati zavisnost vremena izvršavanja od veličine sistema $t \sim N^2$, pa se u slučaju paralelizacije procesa ovo skaliranje smatra referentnim. Korak koji je, kao i u serijskom CPM programu, vremenski najzahtevniji je računanje integrala Hartri potencijala (korak (d) opisan u sekciji 4.3.5). Rezultati paralelne verzije programa pokazuju da se uz odgovarajuću distribuciju podataka, dobro balansiranje računanja integrala i uz minimalan obim MPI komunikacije dobija dobro skaliranje. Iako je ovo komponenta koja je vremenski najzahtevnija, za testirane sistema do veličine od blizu 20 hiljada atoma nema značajnog odstupanja od $t \sim N^2$. Analizom raspodele vremena računanja pojedinačnih delova koda unutar ovog procesa se može zaključiti da se ovakvo skaliranje može očekivati i sa značajno većim brojem atoma od testiranih sistema. Pogoršanje ovakvog skaliranja se može očekivati jedino u slučaju gde bi usled velikog zauzeća memorije bilo potrebno smanjiti broj unapred izračunatih osnovnih integrala ((3.61)), što bi rezultovalo povećanjem broja operacija. Međutim, ovaj scenario iako moguć, zbog drugih posledica koje bi prouzrokovale matrice ove veličine, se ne treba razmatrati kao poseban slučaj. S obzirom da korak (d) ne zavisi od moda izvršavanja, razliku u ukupnom vremenu za izvršavanje CPM programa za različite modove je posledica koraka u kojim se vrši ekstrahovanje motiva (korak (c)) i računanje izmensko-korelacionog integrala (korak (e)).

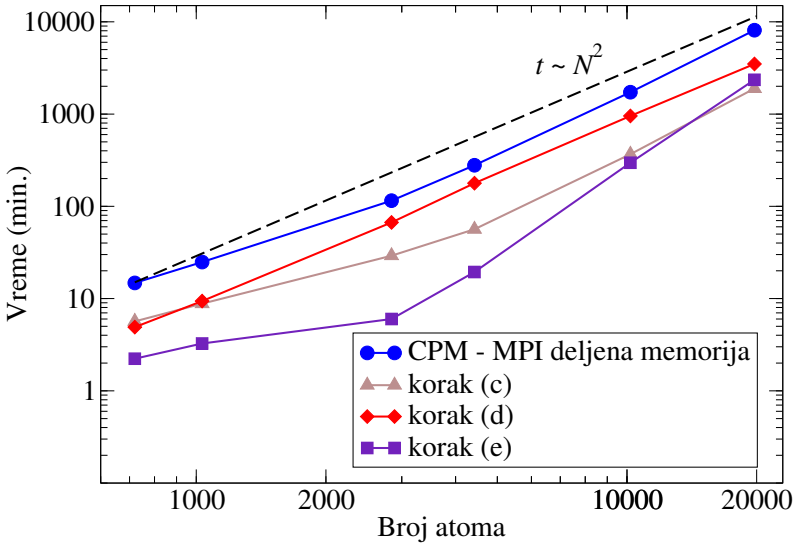
Da bismo bolje prikazali uticaj vremena kroz različite delove procesa, korak (c) ćemo posmatrati kroz tri komponente (tabela 5.6). Prva komponenta (c-1) se odnosi na vreme računanja matrice odsecanja za 'standardne' Gausijane. Pošto je vreme izršavanja koraka (c-1) zanemarljivo u odnosu na ukupno vreme izršavanja, detaljno obrazloženje razlika koje se dobijaju različitim modovima izvršavanja nije od značaja, pa će iz tog razloga biti izostavljeno. Ono što ipak treba napomenuti je da skaliranje ove komponente odgovara projektovanim vrednostima, pa se ne očekuje veliki uticaj u vremenu izršavanja ni u slučaju sistema većih od testiranih. Komponenta za učitavanje i fitovanje motiva (c-2) u slučaju moda (1) već za sistem veličine 2840 atoma predstavlja dominantnu komponentu u ukupnom vremenu za ekstrahovanje motiva, dok se za slučaj moda (2) ovo događa kasnije i vreme postaje dominantno tek za najveći testirani sistem od 19769 atoma. Ako posmatramo različite modove, brzina izvršavanja je za sve sisteme manje od 10184 u korist moda (1), dok trend pokazuje da bi za velike sisteme primat preuzeo mod (2). Razlog ovog trenda je taj što se u modu (1) sa distribuiranim podacima nakon distribuiranog računanja skupa vrednosti elemenata matrice, poziva MPI rutina $MPI\_Allreduce$ (radi sumiranja vrednosti sa svih procesa i distribucije rezultata nazad svim procesima) dok se u slučaju moda (2) sa MPI deljenim memorijskih struktura na nivou noda ove operacije vrše nad deljenim memorijskim strukturama koje su smeštene unutar iste fizičke memorije. Testiranje pokazuje da se ovaj efekat povećava skaliranjem sistema. Pretpostavka je da ovaj efekat zavisi od odnosa ukupnog broja procesa od broja procesa koji se izvršavaju na jednom klasterskom nodu. Tako bi se on povećanjem broja nodova uman-



Slika 5.9: Zavisnost vremena izršavanja za paralelni CPM program (distribuirani podaci) od broja atoma za vremenski najzahtevnije delove programa: korak (c) – ekstrahovanje motiva; korak (d) – računanje integrala Hartri potencijala; korak (e) – izmensko-korelacioni integral. Testirani sistemi su nizovi politiofenskih lanaca $(C_{4n}S_nH_{2n+2})_n$ veličina od 720 do 10184 atoma, izvršeni na 256 paralelnih procesa. Isprekidana linija koja prikazuje $t \sim N^2$ je data kao vizuelni orijentir.

jio ali bi za velike sisteme uvek bio u prednosti u odnosu na sinhronizaciju koja se obavlja nad svim procesima. Naravno, za potvrdu ovakvih pretpostavki bi trebalo opsežnije testiranje koje bi uključilo veće sisteme. Poslednji korak u ekstrahovanju motiva je rešavanje sistema jednačina (c-3). Da bi se u potpunosti razumeo uticaj (c-3) na ukupno vreme izvršavanja, potrebno je ceo proces dodatno razložiti na komponente. Međutim, da bi pojednostavili objašnjenje, dovoljno je navesti osnovnu razliku između između dva programska moda. U slučaju moda (1) kopije ulaznih podataka za dijagonalizaciju su već lokalno dostupne svim paralelnim procesima i tada je dodela elemenata globalnim PETSc strukturama trivijalna. U slučaju moda (2), da bi se formirala globalna PETSC matrica, potrebno je akumulirati vrednosti iz svih MPI deljenih struktura koje su distribuirane po računarskim nodovima. Ovaj pripremni proces je ujedno i glavni razlog razlike u vremenu u korist moda (1). Međutim, s obzirom da je složenost ovih operacija manja od složenosti operacija samog rešavanja sistema linearnih jednačina, skaliranjem veličine sistema se njihov uticaj na ukupno vreme smanjuje.

Korak (e) je u tabeli 5.6 prikazan kroz pripremni korak (e-1) (koji se odnosi na formiranje matrice odsecanja 'standardnih' Gausijana) i računanje izmensko-korelacionog integrala (e-2). Zbog manjeg broja 'standardnih' u odnosu na Kulonove Gausijane, za (e-1) (u odnosu na (c-1)) postoji manji broj operacija koje su potrebne za formiranje ove matrice. Time je vreme trajanja kraće, pa se može konstatovati da isto kao i u slučaju (c-1) ono neznatno utiče na ukupno vreme izvršavanja programa. Sam proces računanja izmensko-korelacionog integrala ima dosta zajedničkih stvari sa procesom za ekstrahovanje motiva. Ovo se posebno odnosi na deo koji se tiče distribucije podataka i komunikacije među procesima. Uvidom u grafike na slikama 5.9 i 5.10 se vidi da je skaliranje vremena izvršavanja koraka (e) nešto lošije u odnosu na korak (c). Ovde je, kao i kod koraka (c), potrebno izračunati elektronsku gustinu naelektrisanja i Gausijane u svakoj tački prostorne mreže



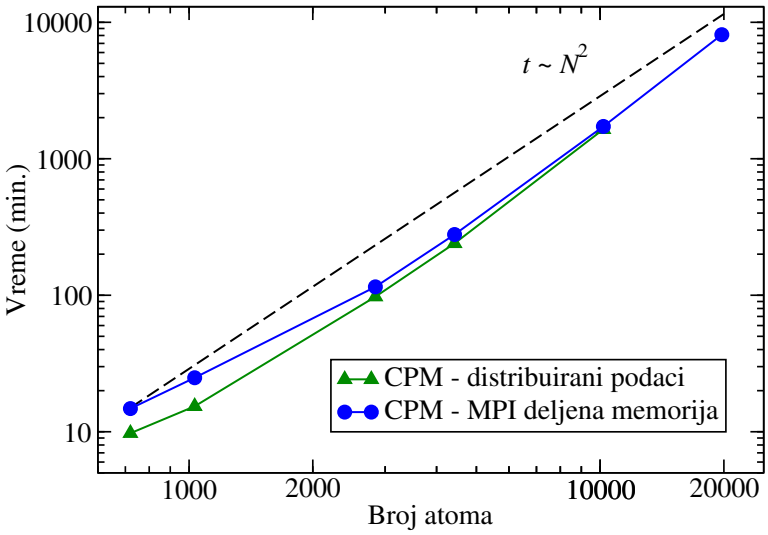
Slika 5.10: Zavisnost vremena izršavanja za paralelni CPM program (MPI deljena memorija) od broja atoma za vremenski najzahtevnije delove programa: korak (c) – ekstrahovanje motiva; korak (d) – računanje integrala Hartri potencijala; korak (e) – izmensko-korelacioni integral. Testirani sistemi su nizovi politiofenskih lanaca $(C_{4n}S_nH_{2n+2})_n$ veličina od 720 do 19769 atoma, izvršeni na 256 paralelnih procesa. Isprekidana linija koja prikazuje $t \sim N^2$ je data kao vizuelni orijentir.

pa otuda i $t \sim N^2$ skaliranje.

Koraci (b) i (f), čija vremena izvršavanja su prikazana u tabeli 5.6, imaju mali udeo u ukupnom vremenu za rešavanje CPM-a (približno 1%). Povećanjem sistema se on dodatno smanjuje, tako da se u testiranim slučajevima pokazalo da računanje efektivnog potencijala jezgra i dijagonalizacija Hamiltonijana imaju dobru efikasnost računanja i skaliranje koje je bolje od ostalih komponenti.

Ukupno vreme izvršavanja za različite modove je prikazano na slici 5.11. Kako promena moda ima direktni uticaj jedino na komponente (c) i (e), razlika u ukupnom vremenu izvršavanja je rezultat uticaja ova dva koraka. Prednost brzine izvršavanja moda (1) u odnosu na mod (2) se gubi povećanjem broja atoma. Za najveće sisteme koje smo imali priliku da testiramo se ova vremena gotovo izjednačavaju i jednako dobro prate $t \sim N^2$ skaliranje. Procenu odnosa brzine izvršavanja ova dva moda za sisteme veće od testiranih je teško napraviti. Ono što je moguće zaključiti je da bi svakako došlo do blagog odstupanja u odnosu na $t \sim N^2$. Do odstupanja bi došlo usled povećanja obima komunikacije među procesima zbog sinhronizacije procesa kod rešavanja sistema linearnih jednačina pri ekstrahovanju motiva i kod dijagonalizacije Hamiltonijana.

Ukoliko posmatramo prvu grupu testova sa fiksnim brojem računarskih nodova, rešavanje sistema sa značajno većim brojem atoma je ograničeno raspoloživom operativnom memorijom. Drugim rečima, usled odabranog načina distribucije podataka zauzeće memorije ne zavisi od ukupnog broja procesa, već od količine raspoložive memorije na pojedinačnim nodovima. Ovo ograničenje bi se najpre ispoljilo na mod (1), gde je usled čuvanja kopije dela podataka na svakom od procesa potrebno onoliko puta više memorije (u odnosu na mod (2)) koliko se ukupno procesa alocira po računarskom nodu. Ako uzmemo primer gde pomenuti podaci zauzimaju 3 GB memorije, na računarskom klasteru gde svaki od nodova ima 32 GB raspoložive memorije, za mod (1) nije moguće alocirati više od 10 CPU jezgara. U tom slučaju ako recimo imamo na raspolaganju 16 CPU jez-



Slika 5.11: Zavisnosti ukupnih vremena izvršavanja paralelnog CPM-a sa distribuiranim podacima i paralelnog CPM-a sa MPI deljenom memorijom od broja atoma. Testirani sistemi su nizovi politiofenskih lanaca $(C_{4n}S_nH_{2n+2})_n$ veličine od 720 do 19769 atoma za CPM sa distribuiranim podacima, odnosno do 10184 atoma za CPM sa MPI deljenom memorijom, izvršeni na 256 paralelnih procesa. Isprekidana linija koja prikazuje $t \sim N^2$ je data kao vizuelni orijentir.

gara, imamo 6 koji nisu upotrebljeni. Ovakav način, iako moguć, ne smatra se dobrom praksom korišćenja računarskih resursa. Iz ovog razloga, kao i na osnovu rezultata dobijenih prvom grupom testova, je preporuka da se za velike sisteme upotrebljava mod (2). Projektovano memorijsko zauzeće pokazuje da bi na raspoloživim resursima PARADOX-IV klastera, sa aspekta memorijskog zauzeća, bilo moguće rešiti sisteme do 200 hiljada atoma. Pošto je za najveći testirani sistem od 19769 atoma trebalo nešto više od 5 dana izvršavanja na svega 16 nodova na računarskom klasteru PARADOX-IV, upotrebom većeg broja nodova je svakako moguće u razumnom vremenskom roku testirati sisteme većih dimenzija. Ipak, imajući u vidu vreme izvršavanja, testiranje sistema veličina većih od 100 hiljada atoma zahteva značajno veće klasterske sisteme. Prednost moda (1) eventualno dolazi do izražaja u slučaju malih sistema gde postoji potreba za velikim brojem simulacija i gde bi ukupno vreme izvršavanja bilo značajno. Ono što bi takođe bilo interesantno je da se analizira razlika u vremenu izršavanja ova dva moda za sisteme sa većim brojem atoma, na računarskom klasteru koji ima na raspolaganju veću količinu operativne memorije (po računarskom nodu). Takva konfiguracija bi omogućila veći obim procesiranja na jednom računarskom nodu i smanjen obim komunikacije među nodovima, što bi po cenu memorijske zahtevnosti moglo dati značajnu vremensku prednost modu (1).

br. atoma	mod	(a)	ekstrahovanje motiva				Hartri		(b)	(e-1)	(e-2)	(f)	ukupno
			(c-1)	(c-2)	(c-3)	ukupno	min	max					
720	(1)	24	44	4	77	147	88	294	7	18	6	19	585
720	(2)	28	21	171	109	340	96	276	8	18	134	15	887
1032	(1)	53	64	11	83	180	213	521	11	26	14	28	921
1032	(2)	55	55	250	161	528	221	565	12	25	196	31	1493
2840	(1)	374	218	155.5	150	755	2402	4083	67	70	275	66	5858
2840	(2)	379	92	595	769	1750	2384	4013	76	70	360	100	6911
4425	(1)	870	354	623	347	1765	6365	9981	124	109	1157	86	14401
4425	(2)	1090	153	730	1679	3385	6532	10689	162	110	1165	168	16719
10184	(1)	6095	1098	9155	5187	17433	37048	56698	788	263	17841	385	98612
10184	(2)	6172	423	9164	9148	22023	36868	57255	766	257	17810	625	103495
19769	(2)	22602	1064	71409	31444	113435	148176	209495	2686	518	141077	2132	486700

Tabela 5.6: Lista vremena izršavanja za paralelni CPM program za nizove politiofenskih lanaca $(C_{4n}S_nH_{2n+2})_n$ na 256 paralelnih procesa. Oznake (1) i (2) redom označavaju mod programa gde je korišćen pristup sa distribuiranim podacima i pristup sa MPI deljenom memorijom. Za prikaz vremena su dati sledeći delovi programa: (a) - ukupno vreme za integral preklapanja, kinetički integral i integral jezgra; (c-1) - matrica odsecanja Kulonovih Gausijana; (c-2) - učitavanje i fitovanje motiva; (c-3) - sistem linearnih jednačina; minimalno i maksimalno vreme za rešavanje Hartree potencijala; (b) - efektivni potencijal jezgra; (e-1) - matrica odsecanja Gausijana; (e-2) - izmensko korelacioni integral ; (f) - dijagonalizacija Hamiltonijana. Vremena su data u sekundama.

Memorijska zahtevnost

Radi boljeg uvida u memorijske zahteve CPM programa, u tabeli 5.7 je data projekcija maksimalnog memorijskog zauzeća, kao i memorijskog zauzeća nekih struktura koje u odnosu na ostale, zahtevaju značajnu količinu RAM memorije. Maksimalno memorijsko zauzeće se postiže u procesu za ekstrahovanje motiva (4.3.4), gde se u memoriji istovremeno učitane matrice H i S, matrica odsecanja za Kulonove Gausijane, matrica A i pomoćna matrica za učitavanje motiva (mmotif). Odgovarajućim načinom distribucije podataka, opisanom u sekciji 4.3.4, je moguće postići značajnu optimizaciju korišćenja memorije. Da bi se ovo prikazalo, navedene su projekcije za mod sa distribuiranim kopijama podataka i za mod sa MPI deljenom memorijom. U oba slučaja se vidi da, prilikom alociranja različitog broja serverskih nodova, ne postoji značajna razlika u zauzeću memorije. Razlog ovome je što se u oba moda izvršavanja radi o različitim načinima distribuiranja istih podataka na nivou jednog noda. Projekcija pokazuje da iz aspekta zauzeća memorije, na raspoloživim računarskim resursima, moguće izvršiti CPM program za sisteme značajno veće od testiranih. Tako je za najveći navedeni sistem od 150000 atoma, u slučaju moda sa MPI deljenom memorijom, potrebno približno 20GB RAM memorije. Iako u ovom delu ne razmatramo vremensku komponentu, potrebno je napomenuti da na raspoloživim CPU resursima, računanje sistema ove veličine nije moguće izvršiti u razumnom vremenskom roku. S druge strane, kao što će u nastavku biti opisano, skaliranjem broja serverskih nodova se može postići zadovoljavajuće vreme računanja.

dimenzija sistema	**opseg**							
	distribuirani podaci	**MPI deljena memorija**						
veličina sistema			4425	6360	10184	50000	100000	150000
dimenzija matrica H i S			27600	39720	63688	315000	630000	945000
dimenzija matrice Kulonovih Gausijana			105201	151441	242897	1200000	2400000	3600000
dimenzija gustih matrica H I S			5110000	7552000	12055128	60000000	120000000	180000000
dimenzija guste matrice Kul. Gausijana			28077210	40155776	64299752	315690062	631380125	947070188
memorijska struktura			**memorijsko zauzeće (MB)**					
H, S matrice	globalno		41	60	96	480	960	1440
matrica odsecanja (Kulonovi Gausijani)	po procesu	po nodu	112	161	257	1263	2526	3788
A matrica	po procesu	po nodu	225	321	514	2526	5051	7577
matrica motiva (mmotif)	po procesu		300	300	300	300	300	300
ostale strukture	po procesu		7	10	16	77	154	230
distribuirane kopije podataka								
4 noda / 16 jezgarni CPU	po nodu		10634	13158	18176	70504	136200	201896
48 nodova / 16 jezgarni CPU	po nodu		10633	13156	18174	70490	136173	201855
MPI deljena memorija								
4 noda / 16 jezgarni CPU	po nodu		5274	5446	5831	9840	14872	19904
48 nodova / 16 jezgarni CPU	po nodu		5252	5445	5828	9826	14844	19863

Tabela 5.7: Projekcija maksimalnog memorijskog zauzeća i memorijskog zauzeća odabranih programskih struktura pri izvršavanju CPM programa, u odnosu na broj atoma ulaznog sistema za različite modove izvršavanja. Memorijsko zauzeće je dato u MB.

Druga grupa testova: jako skaliranje

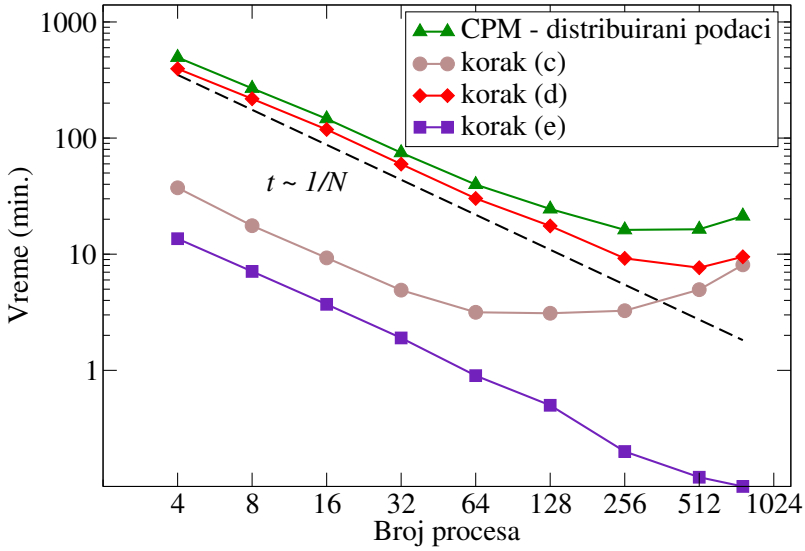
Za drugu grupu testova, kojim se prikazuje jako skaliranje, je kao u slučaju prve grupe odabran prikaz tri vremenski najzahtevnije komponente (c), (d) i (e), dok je u tabelarnom prikazu dat detaljniji pregled vremena izvršavanja. Grafici 5.12 i 5.13 prikazuju zavisnost vremena izvršavanja od broja procesa CPM programa za niz politiofenskih lanaca $(C_{48}S_{12}H_{26})_{12}$ sa 1032 atoma, gde se prvi grafik odnosi na mod (1) sa distribuiranom memorijom, a drugi na mod (2) sa MPI deljenom memorijom. Tabela 5.8 daje detaljniji prikaz za isti sistem. Grafici 5.14 i 5.15, kao i tabela 5.9, se odnose na niz politiofenskih lanaca $(C_{100}S_{25}H_{52})_{25}$ sa 4425 atoma. Poslednji grafik 5.16 poredi ukupna vremena za izvršavanje CPM programa za sva četiri navedena slučaja.

Posmatrajući grafike, rezultati testiranja pokazuju da je vremenski najzahtevniji korak (d), odnosno računanje integrala Hartri potencijala. Očekivana zavisnost vremena izvršavanja od broja procesa je $t \sim 1/N$, pa se može reći da korak (d) u svim testovima dobro prati referentno skaliranje. Izuzetak je slučaj za manji sistem, gde se za veći broj procesa ulazi u oblast zasićenja, što će biti obrazloženo kasnije. Najveće odstupanje od referentnog skaliranja se uočava za komponentu (c), iako se takvo ponašanje može pripisati ranijem ulaskom ove komponente u fazu zasićenja. Ukoliko se u slučaju testova u modu (1) odabere opseg između 4 i 64 procesa a u modu (2) između 16 i 128 procesa, može se reći da se u tim domenima korak (c) ima zadovoljavajuće skaliranje. S obzirom na procentualno učešće ove komponente na ukupno vreme CPM programa, odstupanja od referentnog skaliranja koraka (c) (van zone zasićenja) nema značajan uticaj na ukupno vreme CPM-programa. Ipak, pri modelovanju simulacija i određivanju efikasne distribucije računanja, bi ovu komponentu svakako trebalo uzeti u razmatranje. Na kraju, korak (e) u oba moda testiranja predstavlja vremenski najmanje zahtevnu komponentu i u celom opsegu pokazuje dobro skaliranje.

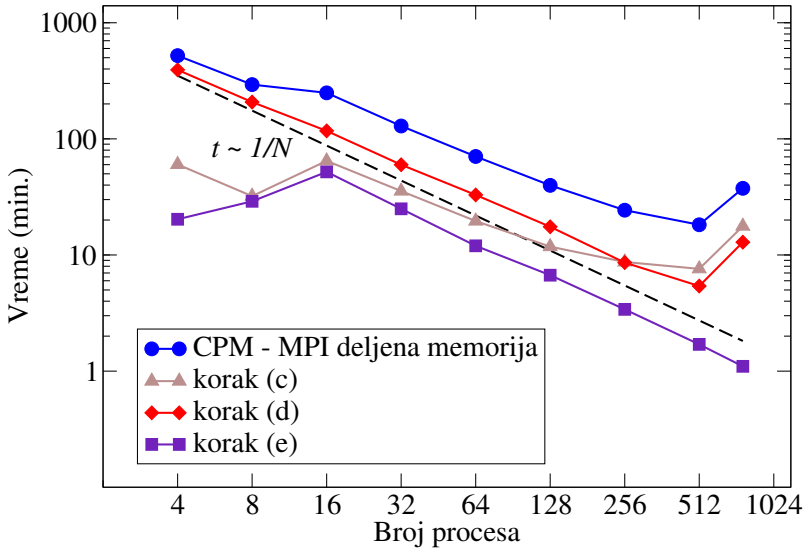
Deo rezultata koji pokazuju neočekivano ponašanje se odnosi na mod (2) za sistem od 1032 atoma (slika 5.13) u opsegu između 4 i 16 procesa. Interesantno je što se odstupanje od referentnog skaliranja ispoljava na nivou jednog klasterskog noda, dok se pri daljem skaliranju pokazuje očekivano ponašanje. Detaljnijim uvidom u rezultate prikazane u tabeli 5.8 se može videti da na ovakve rezultate nemaju uticaj pripremni postupci niti rešavanje sisteme jednačina, već je razlog lošeg skaliranja u procesu za učitavanje i fitovanje motiva (c-2) i u delu za računanje izmensko-korelacionog integrala (e-2), odnosno u delovima gde se primenjuje pristup sa MPI deljenom memorijom. Pretpostavka je da se razlog ovakvom ponašanju može tražiti u hardverskoj konfiguraciju klasterskog noda. Naime, nodovi PARADOX-IV klastera sadrže dva CPU-a koji dele komunikacionu magistralu. Sistem za alokaciju resursa, kao i operativni sistem noda, pri 'dodeli' CPU jezgra odgovarajućim paralelnim procesima, ne vodi računa o redosledu mapiranja fizičkih CPU jezgara. Posledica ovoga je da postoji mogućnost da pri alociranju 8 ili manje jezgara svi procesi budu pokrenuti na jednom CPU-u ili da budu na bilo koji drugi način proizvoljno raspoređeni na dva CPU-a. Pretpostavka je još da implementacija MPI deljene memorije pokazuje nepredvidivo ponašanje u slučaju izršavanja paralelnih procesa na ovakvoj konfiguraciji klasterskog noda. Da bi se ovo dalje ispitalo, potrebno je sprovesti niz testova koji bi dali detaljniji prikaz ponašanja izvršavanja programa na jedno računarskom nodu, kao i na veći broj nodova pod različitim brojem alociranih CPU jezgara. S obzirom da ovi marginalni slučajevi testiranja nisu od velikog značaja, smatramo da ovo prevazilazi okvire ovog rada.

Što se tiče razlike u vremenima izvršavanja koraka (c) i (e) za različite modove, posmatrajući ceo opseg, vidi se da je u oba slučaja vreme izvršavanja nekoliko puta brže u korist moda

(1). Osnovni razlog ovome je razlika u brzini pristupa memorijskim lokacijama. U modu sa distribuiranim podacima (1), svaki proces ima direktan pristup memorijskim lokacijama distribuiranih kopija podataka, dok se u slučaju MPI deljene memorije (2) pristup vrši putem MPI RMA. Iako je RMA pristup u slučaju implementacije MPI deljene memorije na nivou jednog noda, značajno brži u odnosu na eventualnu implementaciju koja bi podrazumevala komunikaciju među procesima sa različitih nodova, testovi pokazuju da i dalje značajnu prednost u vremenu izršavanja ima implementacija sa direktnim pristupom memoriji. Sledeći korak u pravcu pronalaženja optimalnog načina rešavanja distribucije velikih podataka, bi mogao biti implementacija hibridnog algoritma (MPI + OpenMP) i poređenje rezultata sa postojećim rešenjem.



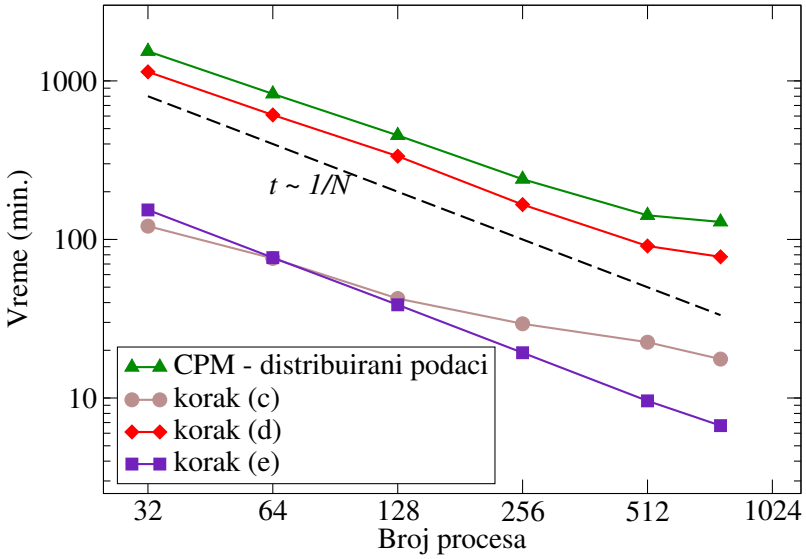
Slika 5.12: Zavisnost vremena izršavanja za paralelni CPM program (distribuirani podaci) od broja procesa za vremenski najzahtevnije delove programa: korak (c) – ekstrahovanje motiva; korak (d) – računanje integrala Hartri potencijala; korak (e) – izmensko-korelacioni integral. Testirani sistem je niz politiofenskih lanaca $(C_{48}S_{12}H_{26})_{12}$ sa 1032 atoma, gde se broj procesa menja od 4 do 768. Isprekidana linija koja prikazuje $t \sim 1/N$ je data kao vizuelni orijentir.



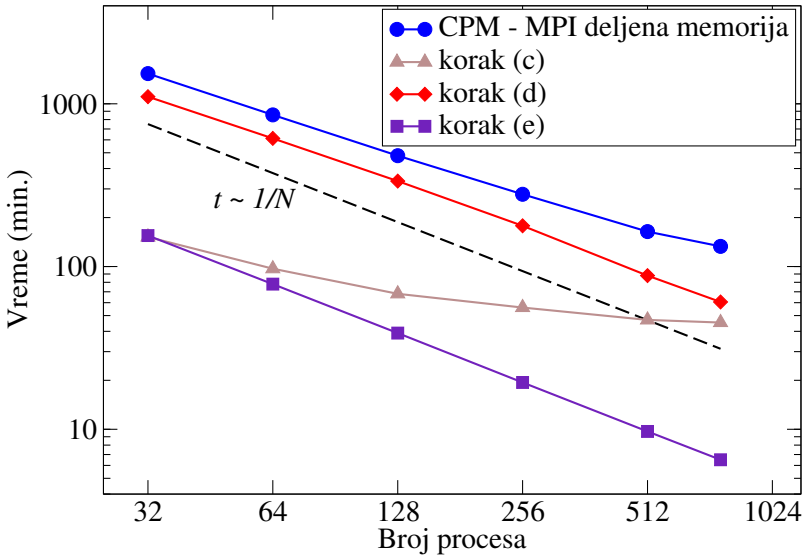
Slika 5.13: Zavisnost vremena izršavanja za paralelni CPM program (MPI deljena memorija) od broja procesa za vremenski najzahtevnije delove programa: korak (c) – ekstrahovanje motiva; korak (d) – računanje integrala Hartri potencijala; korak (e) – izmensko-korelacioni integral. Testirani sistem je niz politiofenskih lanaca $(C_{48}S_{12}H_{26})_{12}$ sa 1032 atoma, gde se broj procesa menja od 4 do 768. Isprekidana linija koja prikazuje $t \sim 1/N$ je data kao vizuelni orijentir.

br. proc.	mod	(a)	ekstrahovanje motiva					Hartri			(b)	(e-1)	(e-2)	ef.	(f)	ukupno	ubrzanje	efikasnost
			(c-1)	(c-2)	(c-3)	ukupno	ef.	min	max	ef.								
4	(1)	2309	59	554	1494	2239	**-**	22395	23723	**-**	573	23	817	-	29	29682	-	**-**
4	(2)	2343	28	1723	1635	3610	**-**	22351	23563	**-**	571	24	1217	-	37	31272	-	**-**
8	(1)	1247	62	286	542	1055	**1.06**	10912	13076	**0.91**	284	25	424	0.96	12	16080	1.85	**0.92**
8	(2)	1219	29	1121	570	1927	**0.94**	10671	12450	**0.95**	295	24	1740	0.35	15	17601	1.78	**0.89**
16	(1)	705	64	152	207	558	**1.00**	5828	7119	**0.83**	154	27	224	0.91	7	8795	3.37	**0.84**
16	(2)	698	31	3398	274	3878	**0.23**	5720	7029	**0.84**	165	26	3154	0.10	9	14940	2.09	**0.52**
32	(1)	352	66	76	75	296	**0.95**	2805	3576	**0.83**	82	26	112	0.91	5	4487	6.62	**0.83**
32	(2)	349	31	1862	137	2133	**0.21**	2715	3595	**0.82**	85	26	1510	0.10	7	7747	4.04	**0.50**
64	(1)	182	65	39	46	190	**0.74**	1240	1814	**0.82**	44	26	57	0.90	6	2389	12.42	**0.78**
64	(2)	205	31	922	112	1176	**0.19**	1187	1976	**0.75**	54	26	752	0.10	6	4229	7.39	**0.46**
128	(1)	101	65	20	62	186	**0.38**	525	1027	**0.72**	22	26	29	0.88	8	1472	20.16	**0.63**
128	(2)	113	31	450	133	707	**0.16**	542	1049	**0.70**	24	26	401	0.09	8	2387	13.10	**0.41**
256	(1)	59	68	11	86	196	**0.18**	220	553	**0.67**	12	28	14	0.91	34	975	30.44	**0.48**
256	(2)	57	31	252	149	520	**0.11**	164	521	**0.71**	12	26	203	0.09	33	1459	21.43	**0.33**
512	(1)	44	86	6	155	297	**0.06**	80	459	**0.40**	7	34	7	0.87	58	982	30.23	**0.24**
512	(2)	33	31	130	192	459	**0.06**	72	324	**0.57**	7	26	101	0.09	56	1095	28.56	**0.22**
768	(1)	52	89	4	335	485	**0.02**	45	570	**0.22**	8	35	6	0.66	57	1276	23.26	**0.12**
768	(2)	74	32	84	798	1060	**0.02**	34	775	**0.16**	18	26	66	0.10	183	2252	13.89	**0.07**

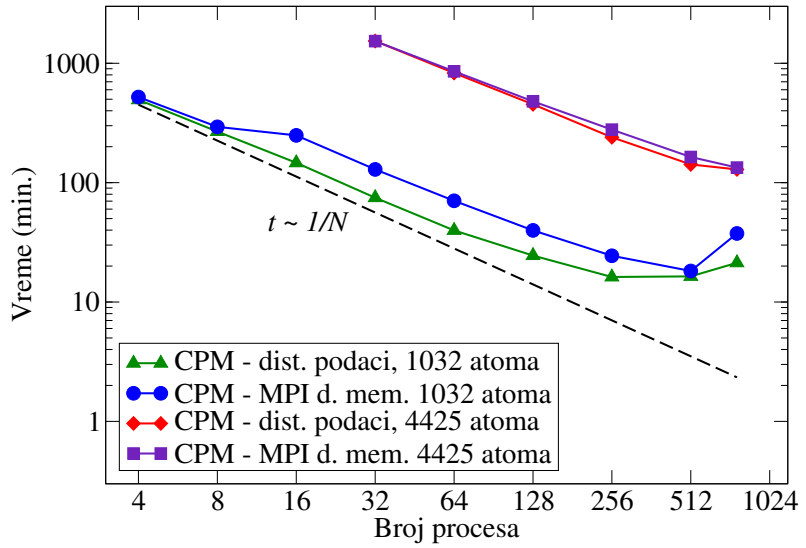
Tabela 5.8: Lista vremena izvršavanja sa koeficijentima efikasnosti za paralelni CPM program za niz politiofenskih lanaca $(C_{48}S_{12}H_{26})_{12}$ sa 1032 atoma sa različitim brojem procesa. Oznake (1) i (2) redom označavaju mod programa gde je korićen pristup sa distribuiranim podacima i pristup sa MPI deljenom memorijom. Za prikaz vremena su dati sledeći delovi programa: (a) - ukupno vreme za integral preklapanja, kinetički integral i integral jezgra; (c-1) - matrica odsecanja Kulonovih Gausijana; (c-2) - učitavanje i fitovanje motiva; (c-3) - sistem linearnih jednačina; minimalno i maksimalno vreme za rešavanje Hartri potencijala; (b) - efektivni potencijal jezgra; (e-1) - matrica odsecanja Gausijana; (e-2) - izmensko korelacioni integral ; (f) - dijagonalizacija Hamiltonijana. Vremena su data u sekundama.



Slika 5.14: Zavisnost vremena izršavanja za paralelni CPM program (distribuirani podaci) od broja procesa za vremenski najzahtevnije delove programa: korak (c) – ekstrahovanje motiva; korak (d) – računanje integrala Hartri potencijala; korak (e) – izmensko-korelacioni integral. Testirani sistem je niz politiofenskih lanaca $(C_{100}S_{25}H_{52})_{25}$ sa 4425 atoma, gde se broj procesa menja od 32 do 768. Isprekidana linija koja prikazuje $t \sim 1/N$ je data kao vizuelni orijentir.



Slika 5.15: Zavisnost vremena izršavanja za paralelni CPM program (MPI deljena memorija) od broja procesa za vremenski najzahtevnije delove programa: korak (c) – ekstrahovanje motiva; korak (d) – računanje integrala Hartri potencijala; korak (e) – izmensko-korelacioni integral. Testirani sistem je niz politiofenskih lanaca $(C_{100}S_{25}H_{52})_{25}$ sa 4425 atoma, gde se broj procesa menja od 32 do 768. Isprekidana linija koja prikazuje $t \sim 1/N$ je data kao vizuelni orijentir.



Slika 5.16: Poređenje zavisnosti ukupnih vremena izvršavanja paralelnog CPM-a sa distribuiranim podacima i paralelnog CPM-a sa MPI deljenom memorijom od broja paralelnih procesa za nizove politiofenskih lanaca $(C_{48}S_{12}H_{26})_{12}$ sa 1032 atoma i $(C_{100}S_{25}H_{52})_{25}$ sa 4425 atoma. Broj procesa se menja od 4 do 768. Isprekidana linija koja prikazuje $t \sim 1/N$ je data kao vizuelni orijentir.

br. proc.	mod	(a)	ekstrahovanje motiva					Hartri			(b)	(e-1)	(e-2)	ef.	(f)	ukupno	ubrzanje	efikasnost
			(c-1)	(c-2)	(c-3)	ukupno	ef.	min	max	ef.								
32	(1)	6415	362	4959	1303	7284	**-**	58831	68397	**-**	927	121	9212	-	64	92177	-	**-**
32	(2)	6612	152	5326	2474	9110	**-**	58862	66347	**-**	904	112	9326	-	135	92019	-	**-**
64	(1)	3765	355	2479	450	4552	**0.80**	28604	36638	**0.93**	561	113	4606	1.00	25	49695	1.85	**0.93**
64	(2)	3777	153	2792	1728	5866	**0.78**	29053	36805	**0.90**	455	112	4660	1.00	96	51349	1.79	**0.90**
128	(1)	2062	349	1248	206	2545	**0.89**	13597	20114	**0.85**	313	117	2321	0.99	23	27214	3.39	**0.85**
128	(2)	2086	154	1360	1580	4054	**0.56**	13512	20130	**0.82**	319	112	2332	1.00	93	28818	3.19	**0.80**
256	(1)	870	355	624	348	1765	**0.52**	6366	9981	**0.86**	124	110	1158	0.99	86	14401	6.40	**0.80**
256	(2)	1090	153	731	1680	3385	**0.34**	6532	10689	**0.78**	162	110	1165	1.00	168	16719	5.50	**0.69**
512	(1)	612	442	313	264	1349	**0.34**	2939	5454	**0.78**	66	149	579	0.99	159	8529	10.81	**0.68**
512	(2)	592	154	368	1662	2819	**0.20**	2782	5252	**0.79**	90	113	580	1.00	260	9870	9.32	**0.58**
768	(1)	532	442	220	26	1058	**0.29**	1529	4663	**0.61**	81	148	402	0.95	454	7750	11.89	**0.50**
768	(2)	420	154	257	1793	2719	**0.14**	1304	3637	**0.76**	65	114	389	1.00	454	8026	11.47	**0.48**

Tabela 5.9: Lista vremena izvršavanja sa koeficijentima efikasnosti za paralelni CPM program za niz politiofenskih lanaca $(C_{48}S_{12}H_{26})_{12}$ sa 4425 atoma sa različitim brojem procesa. Oznake (1) i (2) redom označavaju mod programa gde je korišćen pristup sa distribuiranim podacima i pristup sa MPI deljenom memorijom. Za prikaz vremena su dati sledeći delovi programa: (a) - ukupno vreme za integral preklapanja, kinetički integral i integral jezgra; (c-1) - matrica odsecanja Kulonovih Gausijana; (c-2) - učitavanje i fitovanje motiva; (c-3) - sistem linearnih jednačina; minimalno i maksimalno vreme za rešavanje Hartri potencijala; (b) - efektivni potencijal jezgra; (e-1) - matrica odsecanja Gausijana; (e-2) - izmensko korelacioni integral ; (f) - dijagonalizacija Hamiltonijana. Vremena su data u sekundama.

Uporedni prikaz ukupnih vremena izvršavanja za oba moda programa i oba ulazna sistema (slika 5.16) pokazuje da je prednost moda (1) u vremenu izršavanja prilikom skaliranja procesa, gotovo konstanta. Rezultati takođe pokazuju da je vremenska razlika za različite modove veća u slučaju manjeg sistema. Ovo pokazuje da efikasnost primene MPI deljene memorije dolazi do izražaja u slučajevima kada su distribuirane strukture veće, tj. kada je odnos između broja operacija na nivou noda i količine MPI komunikacije veći. Takođe se može reći da je za testirani opseg bolje skaliranje pokazala grupa testova vršena nad većim ulaznim sistemom. Tako se sa povećanjem sistema povećava i broj procesa na kome se dobija dobro skaliranje, tj. tačka zasićenja se pomera ka većem brojem procesa.

5.2.5 Efikasnost i optimizacija izršavanja paralelnog CPM programa

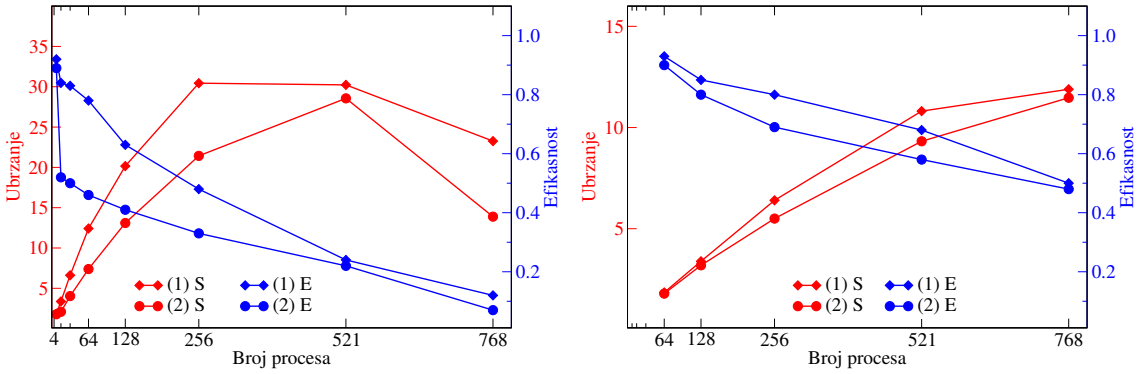
Prikaz efikasnosti CPM programa za vremenski zahtevne komponente kao i za ukupno vreme izvršavanja je dato u tabelama 5.8 i 5.9, kao i u grafičkim prikazima na slici 5.17. Kao referentna vrednost za određivanje efikasnosti se koristi vreme izvršavanja testova sa najmanjim brojem paralelnih procesa. U slučaju sistema sa 1032 atoma su uzeta referentna vremena dobijena izvršavanjem na 4, dok su u slučaju sistema sa 4425 atoma na 32 procesa. Idealno određivanje efikasnosti skaliranja bi podrazumevalo upotrebu rezultata dobijenih serijskim kodom, međutim na raspoloživim resursima za sisteme ove veličine, takvi testovi nisu mogući, pa se kao referentne vrednosti koriste one koje su dobijene izvršavanjem programa na najmanjem broju paralelnih procesa.

Ubrzanje S_p i efikasnost E_p se za CPM program izvršen na p paralelnih procesa računaju na osnovu sledeće dve jednačine:

$$S_p = \frac{T_I}{T_p} \tag{5.1}$$

$$E_p = \frac{S_p I}{p} \tag{5.2}$$

gde I predstavlja broj procesa referentnog sistema a T_I i T_p redom vremena izvršavanja CPM-a na referentnom broju procesa i na p procesa.



Slika 5.17: Ubrzanje (S) i efikasnost (E) jakog skaliranja CPM programa za mod sa distribuiranim podacima (1) i mod sa MPI deljenom memorijom (2). Leva slika se odnosi za niz politiofenskih lanaca $(C_{48}S_{12}H_{26})_{12}$ sa 1032 atoma a desna na $(C_{100}S_{25}H_{52})_{25}$ sa 4425 atoma. U slučaju sistema sa 1032 atoma se poređenje efikasnosti vrši na osnovu vremena izvršavanja dobijenih na 4 procesa, a u slučaju sistema sa 4425 astoma na 32 procesa.

Zasićenje za testirane slučajeve (slika 5.17) je očekivano i do njega dolazi iz dva razloga. Prvi

se odnosi na to što se pri povećanju broja paralelnih procesa smanjuje obim operacija po procesu a povećava obim MPI komunikacije. Drugi razlog je što se efekat randomizacije atoma ulaznog sistema smanjuje sa smanjivanjem blokova matrica koji se distribuiraju paralelnim procesima (sekcija 4.3.1). Za manji sistem (gde su matrice H i S dimenzije 6398×6398), za slučaju distribucije na 768 procesa, ovi blokovi u proseku sadrže svega 8 redova matrice. Ako uporedimo veličinu blokova sa prosečnim brojem Gausijana po atomu, svakako je jasno da dolazi do fragmentacije i disbalansa računanja integrala. U slučaju moda (1) primenjenog na manji sistem, do zasićenja dolazi već na 256 procesa, dok se za slučaj moda (2) ovo zasićenje uočava tek za test koji je pokrenut na 768 procesa. U oba slučaja do zasićenja dovode koraci (c) i (d) (pogledati vrednosti u tabeli 5.8). Za veći sistem od 4425 atoma za grupu testova koji je sproveden ne dolazi do zasićenja (pogledati vrednost u tabeli 5.9). Međutim u oba moda korak (c) daje jasan nagoveštaj da bi se za ovaj ulazni sistem ono moglo očekivati za slučajeve gde bi se testiranje vršilo na više od 1024 procesa.

Iako je određivanje tačke zasićenja za širu klasu ulaznih sistema složen proces koji zahteva analizu većeg broja faktora, moguće je napraviti okvirnu procenu koja bi mogla biti referentna pri modeliranju simulacija. Pretpostavka je da će do zasićenja doći u slučaju kada je granulacija matrica H i S takva da proces randomizacije atoma ne daje željeni efekat i kada nije moguće očekivati dobar balans računanja integrala. Odnosno, do zasićenja će sigurno doći ukoliko je srednja vrednost broja redova matrica H i S (posledica distribucije podataka) manja od srednje vrednosti broja Gausijana po atomu:

$$\frac{n}{p} < \frac{\sum_{i=1}^{k} g_i}{k}, \tag{5.3}$$

gde je p broj procesa, n veličina matrice a g_i broj Gausijana za svaki od k tipova atoma koji su deo ulaznog sistema. Međutim, da bismo dodatno relaksirali prethodni uslov i time odredili željeni broj procesa pre ulaska u zonu zasićenja, možemo uzeti uslov da je minimalna granulacija jednaka zbiru Gausijana po svim tipovima atoma.

$$\frac{n}{p} < \sum_{i=1}^{k} g_i. \tag{5.4}$$

To određujemo na osnovu pretpostavke da se pri većoj granulaciji od predložene u značajnoj meri gubi efekat randomizacije atoma. U suprotnom, randomizacija mahom vrši premeštanje blokova iste ili veoma slične strukture, što želimo da izbegnemo. Konačno, na osnovu veličine sistema koji želimo da rešavamo određujemo tačku zasićenja:

$$p_s = \frac{n}{\sum_{i=1}^{k} g_i}, \tag{5.5}$$

na osnovu čega se može reći da je za optimalno izvršavanje datog sistema potrebno da broj paralelnih procesa bude manji ili jednak p_s.

Ukoliko primenimo jednačinu (5.5) na testirane sisteme, za sistem veličine 1032 atoma koji ima matricu veličine 6398, p_s ima vrednost 355, dok za sistem veličine 4425 atoma, koji ima matricu veličine 27600, p_s ima vrednost 1533. U slučaju manjeg sistema se projektovana vrednost dobro slaže sa rezultatima testova i može se reći da jednačina (5.5) daje dobru procenu optimalnog odabira broja procesa. U slučaju većeg sistema je ovo teže utvrditi na osnovu rezultata, jer testiranje nije izvršeno na više od 768 procesa, gde program i dalje nije ušao u zonu zasićenja.

Više puta do sada je pomenut uticaj primene procedure za slučajni odabir redosleda (randomizacije) atoma u ulaznom sistemu (listing 4.7 u sekciji 4.3.1), koja je uvedena radi optimizacije balansa računanja integrala. Radi utvrđivanja odstupanja usled randomizacije atoma je izvršeno 5 ponovljenih testova za sistem veličine 1032 atoma na 256 procesa i 5 testova za sistem od 4425 atoma na 256 procesa. Rezultati ukupnog vremena izvršavanja CPM-a pokazuju da je prosečno odstupanje za navedene uzorke testiranja manje od 3%. Maksimalno odstupanje je 6%, dok je minimalno 0.4%. Ovi rezultati dovode do zaključka da za sisteme srednje veličine odstupanje usled različitog rasporeda atoma neznatno utiče na rezultat, dok se za velike sisteme ovaj faktor može zanemariti. Iako u testiranim slučajevima nije zabeleženo veće odstupanje, ne treba isključiti mogućnost slučajnog odabira rasporeda atoma koji bi za određene tipove ulaznog sistema sa distribuiranim računanjem nad određenim brojem procesa doveo do značajnijeg učinka ovog efekta.

Pored navedenog razloga, za različite rezultate vremena izvršavanja pri ponovljenom pokretanju iste konfiguracije CPM programa, postoje i drugi faktori koji ne spadaju u domen implementacije CPM-a. Jedan od njih je topologija mreže računarskog klastera. Pri definisanju skripte za izvršavanje programa se između ostalih parametara postavlja broj nodova koji je potrebno alocirati. Ova postavka prepušta klasterskom sistemu za alokaciju resursa da odabere određeni broj slobodnih klasterskih nodova, pri čemu se ne vodi računa o poziciji nodova u topologiji mreže. Dakle, ponovljenim pokretanjem programa se vrši novo alociranje resursa, kojim se dobija grupa nodova sa proizvoljno drugačijim rasporedom u odnosu na prethodno izvršavanje. Iako se na osnovu eksperimenata sprovedenim nad različitim konfiguracijama klastera može doći do optimalnih rezultata za CPM program, to svakako prevazilazi domen bazičnog testiranja performansi koji smo želeli da prikažemo. Kako efikasnost izvršavanja MPI programa značajno zavisi od brzine računarske mreže, drugi faktor koji može uticati na vreme izvršavanja je opterećenje klasterske mreže u toku trajanja testova. Iako na klasteru PARADOX-IV pri testiranju nisu evidentirana značajna odstupanja koja bi inicirala detaljnu analizu pomenuta dva faktora, ne isključuje se mogućnost da oni imaju uticaj na različita vremena ponovljenih izvršavanja.

Takođe treba napomenuti da se uticaji randomizacije i navedenih faktora koji se tiču konfiguracije klastera mogu značajno razlikovati na drugim klasterskim sistemima. Da bismo to potvrdili, sproveden je niz testova paralelnog CPM programa na klasteru AXIOM, koji nisu za cilj imali detaljnu analizu već grubu potvrdu prethodnu pretpostavke. Da bi se utvrdio uticaj opterećenja mrežnog interfejsa, najpre je izvršen test manjeg sistema na 4 noda a zatim je isti test pokrenut dva puta istovremeno (svaki na 4 noda). Iako je u ovom slučaju razlika neznatna, pri svakom ponavljanju je ona u korist testa koji je izvršen 'izolovano'. Pretpostavka je da bi pri intenzivnijoj MPI komunikaciji za veće ulazne sisteme ovaj efekat bio veći. Drugi test je uzeo u obzir topologiju mreže AXIOM-a. Ukratko, topologija je takva da su 16 nodova raspoređeni na dva mrežna čvorišta (8+8), koja međusobno ostvaruju maksimalnu brzinu od 10 Gbps. Ovde je pokazano da su vremena izršavanja programa koji su pokrenuti na nodovima istog čvorišta bolja, u odnosu na ona gde su alocirani nodovi na dva čvorišta istovremeno. Različiti rezultati u ovom slučaju ne moraju obavezno značiti zasićenje mrežnog bandwith-a, već mogu biti uzrokovani različitim latency-om unutar istog čvorišta i u komunikaciji između dva čvorišta.

Kao što je analizom prikazano, odgovarajućom alokacijom raspoloživih resursa se može postići optimalno i u odnosu na veličinu sistema, vremenski vrlo efikasno rešavanje elektronske strukture testiranih sistema. Takođe, dolazi se do zaključka da se iz aspekta performansi, programski mod (1)

može smatrati efikasnijim u slučaju manjih sistema, dok se za veće sisteme oba moda mogu smatrati podjednako efikasnim i da bi se njihovo korišćenje trebalo odrediti na osnovu potreba. Tako bi predlog za optimalnu upotrebu programskih modova bio da se mod sa distribuiranim kopijama podataka (1) koristi u slučaju kada su ulazni sistemi takvi da odgovaraju raspoloživim memorijskim resursima, a da se za veće sisteme koristi mod sa MPI deljenom memorijom (2).

Poglavlje 6

Zaključak

U tezi je predstavljena implementacija metode za sklapanje naelektrisanja (CPM) u bazisu Gausijana, koja se koristi za računanje elektronske strukture materijala. Implementacije opisane u radu obuhvataju četiri softverska rešenja: serijski DFT program za mali prototip sistem, serijski program za ekstrahovanje motiva, serijski CPM program i paralelni MPI CPM program. Ovim softverskim rešenjima se pokrivaju celokupan postupak koji uključuje učitavanja korisnički definisanih ulaznih sistema, rešavanja malog prototip sistema uz pomoć DFT-a, generisanje motiva i rešavanje elektronske strukture materijala za velike sisteme uz pomoć CPM-a. Produkcione verzije programa sadrže približno 20000 linija koda implementiranog u programskom jeziku C.

Za realizaciju navedenih rešenja su najpre implementirani algoritmi za računanje gausijanskih integrala, koji predstavljaju bazu za implementaciju DFT programa. Uz pomoć DFT programa je moguće izračunati gustinu naelektrisanja za prototip sistem, koji se dalje koristi za ekstrahovanje motiva.

Pokazano je da je najpogodniji način čuvanja motiva gustine naelektrisanja uz pomoć prostorne reprezentacije. Kao posledica toga je razvijena metoda za dobijanje odgovarajućeg bazisa Gausijana, na osnovu prostorne reprezentacije motiva. Takođe je predstavljena rekurzivna šema za integrale centrirane na četiri Gausijana. Sa takvom rekurzivnom šemom je moguće ostvariti značajno bolje perfomanse nego u slučaju direktnog računanja integrala preko analitičkih formula. Kombinacija rešenja iz serijske verzije koda, njihova adaptacija za korišćenje paralelne verzije CPM, kao i implementacija algoritama namenjenih isključivo paralelnom izvršavanju, rezultuju softverskim rešenjem za paralelno računanje CPM.

Testovi performansi serijskog algoritma pokazuju da se sistemi od nekoliko stotina atoma mogu izračunati za manje od sata, dok je sistem sa približno 1500 atoma, na jednom procesorskom jezgru, izračunat za svega nekoliko sati.

Mogućnost da se u slučaju serijske verzije uspešno izračunavaju sistemi sa više od hiljadu atoma, je dalo dobru bazu koja je dalje omogućila razvoj MPI verzije paralelnog CPM programa. Opsežnim testovima je prikazano da se sa odgovarajućim odabirom algoritma na osnovu analize ulaznog sistema i odgovarajućom alokacijom raspoloživog resursa, može postići optimalno računanje elektronske strukture materijala. Ovo je rezultovalo time da se na relativno skromnim klasterskim resursima može vršiti izračunavanje sistema od nekoliko desetina hiljada atoma.

Testovi performansi CPM programa su podeljeni u dve grupe: prva grupa testova koja prikazuje zavisnost vremena izvršavanja CPM programa u odnosu na veličinu ulaznog sistema i druga grupa

koja prikazuje jako skaliranje (strong scaling). U prvoj grupi su svi test slučajevi izvršeni na 16 klasterskih nodova PARADOX-IV klastera, odnosno na 256 CPU jezgara gde je najveći testirani sistem imao 20000 atoma. Druga grupa testova podrazumeva fiksnu veličinu problema (1032 i 4425 atoma) koji se rešava, dok se vrši skaliranje broja paralelnih procesa. Najveći broj paralelnih procesa na kome su vršeni testovi je 768, odnosno 48 klasterskih nodova. Obe grupe testova su urađene za dva moda CPU programa: mod u kome se primenjuje sistem distribucije podataka gde se velike strukture distribuiraju kao kopije svim raspoloživim procesima i mod u kome se za iste memorijske strukture primenjuje sistem sa MPI deljenom memorijom. Uporedni prikaz ukupnih vremena izvršavanja za oba moda programa i oba ulazna sistema pokazuje da je prednost prvog navedenog moda u vremenu izvršavanja prilikom skaliranja procesa, gotovo konstanta. Rezultati takođe pokazuju da je vremenska razlika za različite modove veća u slučaju manjeg sistema. Ovo dovodi do zaključka da efikasnost primene MPI deljene memorije dolazi do izražaja u slučajevima kada su distribuirane strukture veće, tj. kada je odnos između broja operacija na nivou noda i količine MPI komunikacije veći. Takođe se može reći da je za testirani opseg bolje skaliranje pokazala grupa testova vršena nad većim ulaznim sistemom. Tako se sa povećanjem sistema povećava i broj procesa na kome se dobija dobro skaliranje, tj. tačka zasićenja se pomera ka većem brojem procesa.

Kao referentna vrednost za određivanje efikasnosti CPM programa se koristi vreme izvršavanja testova sa najmanjim brojem paralelnih procesa. Na efikasnost izvršavanja mogu uticati mnogi faktori. Neki od njih su randomizacija atoma pri formiranju ulaznog sistema, konfiguracija klastera, topologija mreže i mnogi drugi. Iako je određivanje tačke zasićenja za širu klasu ulaznih sistema složen proces koji zahteva analizu većeg broja faktora, u tezi je dat algoritam za formiranje okvirne procene koja se može koristiti kao referentna pri modeliranju simulacija.

Kao što je analizom prikazano, odgovarajućom alokacijom raspoloživih resursa se može postići optimalno i u odnosu na veličinu sistema, vremenski vrlo efikasno rešavanje elektronske strukture testiranih sistema. Projektovano memorijsko zauzeće pokazuje da bi na raspoloživim resursima PARADOX-IV klastera, sa aspekta memorijskog zauzeća, bilo moguće rešiti sisteme do 200 hiljada atoma.

Vrednosti izračunatih elektronskih stanja u svim analiziranim slučajevima pokazuju dobro slaganje sa referentnim vrednostima. Pokazano je da su razlike između serijskog i paralelnog CPM-a manje od 1 μH, dok su odstupanja vrednosti paralelne i serijske verzije CPM programa u odnosu na vrednosti DFT manja od 1 mH.

Pored toga što se razvijeni CPM program može koristiti kao jedinstvena računarska simulacija za računanje elektronske strukture sistema sa više od deset hiljada atoma, on bi takođe mogao naći primenu i kao deo nekog složenije simulacije za proračune značajno većih sistema od onih koji su testirani u ovom radu. Jedan od metoda koji se može iskoristiti u tu svrhu je metod preklopljenih fragmenata [56]. U okviru te metode se fizički sistem deli na fragmente, računaju se elektronska stanja svakog od fragmenata, a zatim se jednočestični Hamiltonijan celog sistema predstavlja u bazisu tih fragmenata. Na kraju se vrši dijagonalizacija Hamiltonijana. Računski najzahtevniji korak tog postupka je proračun elektronskih stanja fragmenata. CPM se kao efikasan metod za računanje elektronskih stanja, mogao iskoristi upravo u tu svrhu.

Bibliografija

[1] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Proceedings, 11th European PVM/MPI Users' Group Meeting*, (Budapest, Hungary), pp. 97–104, 2004.

[2] MPICH2, http://www.mcs.anl.gov/mpi/mpich2, last accessed 18 April 2019.

[3] M. Guest, *PRACE: The Scientific Case for HPC in Europe*. Insight publishers, 2012.

[4] P. Hohenberg and W. Kohn, "Inhomogeneous electron gas," *Phys. Rev.*, vol. 136, p. B864, 1964.

[5] W. Kohn and L. J. Sham, "Self-consistent equations including exchange and correlation effects," *Phys. Rev.*, vol. 140, p. A1133, 1965.

[6] M. C. Payne, M. P. Teter, D. C. Allan, T. A. Arias, and J. D. Joannopoulos, "Iterative minimization techniques for ab initio total-energy calculations: molecular dynamics and conjugate gradients," *Rev. Mod. Phys.*, vol. 64, p. 1045, 1992.

[7] Y. Saad, J. R. Chelikowsky, and S. M. Shontz, "Numerical methods for electronic structure calculations of materials," *SIAM review*, no. 1, pp. 3–54, 2010.

[8] R. Car and M. Parrinello, "Unified approach for molecular dynamics and density-functional theory," *Physical review letters*, vol. 55, p. 2471, 1985.

[9] G. Kresse and J. Furthmüller, "Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set," *Physical review B*, vol. 54, p. 11169, 1996.

[10] S. Goedecker, "Linear scaling electronic structure methods," *Reviews of Modern Physics*, vol. 71, p. 1085, 1999.

[11] D. R. Bowler and T. Miyazaki, "Methods in electronic structure calculations," *Reports on Progress in Physics*, no. 3, p. 036503, 2012.

[12] L. Lin, J. Lu, and L. Ying, "Numerical methods for kohn–sham density functional theory," *Acta Numerica*, pp. 405–539, 2019.

[13] X. Gonze, F. Jollet, F. A. Araujo, D. Adams, B. Amadon, T. Applencourt, C. Audouze, J.-M. Beuken, J. Bieder, A. Bokhanchuk, *et al.*, "Recent developments in the ABINIT software package," *Computer Physics Communications*, vol. 205, pp. 106–131, 2016.

[14] S. J. Clark, M. D. Segall, C. J. Pickard, P. J. Hasnip, M. I. Probert, K. Refson, and M. C. Payne, "First principles methods using CASTEP," *Zeitschrift für Kristallographie-Crystalline Materials*, vol. 220, pp. 567–570, 2005.

[15] V. Blum, R. Gehrke, F. Hanke, P. Havu, V. Havu, X. Ren, K. Reuter, and M. Scheffler, "Ab initio molecular simulations with numeric atom-centered orbitals," *Computer Physics Communications*, vol. 180, pp. 2175–2196, 2009.

[16] H.-J. Werner, P. J. Knowles, G. Knizia, F. R. Manby, and M. Schütz, "Molpro: a general-purpose quantum chemistry program package," *Wiley Interdisciplinary Reviews: Computational Molecular Science*, vol. 2, pp. 242–253, 2012.

[17] M. Valiev, E. J. Bylaska, N. Govind, K. Kowalski, T. P. Straatsma, H. J. Van Dam, D. Wang, J. Nieplocha, E. Apra, T. L. Windus, *et al.*, "NWChem: A comprehensive and scalable open-source solution for large scale molecular simulations," *Computer Physics Communications*, vol. 181, pp. 1477–1489, 2010.

[18] F. Gygi, "Architecture of Qbox: A scalable first-principles molecular dynamics code," *IBM Journal of Research and Development*, vol. 52, pp. 137–144, 2008.

[19] Y. Shao, Z. Gan, E. Epifanovsky, A. T. Gilbert, M. Wormit, J. Kussmann, A. W. Lange, A. Behn, J. Deng, X. Feng, *et al.*, "Advances in molecular quantum chemistry contained in the Q-Chem 4 program package," *Molecular Physics*, vol. 113, pp. 184–215, 2015.

[20] P. Giannozzi, O. Andreussi, T. Brumme, O. Bunau, M. B. Nardelli, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, M. Cococcioni, *et al.*, "Advanced capabilities for materials modelling with Quantum ESPRESSO," *Journal of Physics: Condensed Matter*, vol. 29, p. 465901, 2017.

[21] J. M. Soler, E. Artacho, J. D. Gale, A. García, J. Junquera, P. Ordejón, and D. Sánchez-Portal, "The SIESTA method for ab initio order-N materials simulation," *Journal of Physics: Condensed Matter*, vol. 14, p. 2745, 2002.

[22] G. Kresse and J. Hafner, "Ab initio molecular dynamics for liquid metals," *Physical Review B*, vol. 47, no. 1, p. 558, 1993.

[23] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo, A. Dal Corso, S. de Gironcoli, S. Fabris, G. Fratesi, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A. P. Seitsonen, A. Smogunov, P. Umari, and R. M. Wentzcovitch, "QUANTUM ESPRESSO: a Modular and Open-source Software Project for Quantum Simulations of Materials," *J. Phys.: Condens. Matter*, vol. 21, p. 395502, 2009.

[24] X. Gonze, B. Amadon, P.-M. Anglade, J.-M. Beuken, F. Bottin, P. Boulanger, F. Bruneval, D. Caliste, R. Caracas, M. Côté, T. Deutsch, L. Genovese, P. Ghosez, M. Giantomassi, S. Goedecker, D. Hamann, P. Hermet, F. Jollet, G. Jomard, S. Leroux, M. Mancini, S. Mazevet, M. Oliveira, G. Onida, Y. Pouillon, T. Rangel, G.-M. Rignanese, D. Sangalli, R. Shaltaf, M. Torrent, M. Verstraete, G. Zerah, and J. Zwanziger, "Abinit: First-principles

approach to material and nanosystem properties," *Comp. Phys. Comm.*, vol. 180, p. 2582, 2009.

[25] S. J. Clark, M. D. Segall, C. J. Pickard, P. J. Hasnip, M. J. Probert, K. Refson, and M. C. Payne, "First principles methods using CASTEP," *Z. Krist.-Cryst. Mater.*, vol. 220, p. 567, 2005.

[26] PETOT, http://cmsn.lbl.gov/html/PEtot/PEtot.html, last accessed 4 April 2018.

[27] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, B. Mennucci, G. A. Petersson, H. Nakatsuji, M. Caricato, X. Li, H. P. Hratchian, A. F. Izmaylov, J. Bloino, G. Zheng, J. L. Sonnenberg, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. Bearpark, J. J. Heyd, E. Brothers, K. N. Kudin, V. N. Staroverov, R. Kobayashi, J. Normand, K. Raghavachari, A. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, N. Rega, J. M. Millam, M. Klene, J. E. Knox, J. B. Cross, V. Bakken, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, R. L. Martin, K. Morokuma, V. G. Zakrzewski, G. A. Voth, P. Salvador, J. J. Dannenberg, S. Dapprich, A. D. Daniels, O. Farkas, J. B. Foresman, J. V. Ortiz, J. Cioslowski, and D. J. Fox, "Gaussian 09." Gaussian Inc. Wallingford CT 2009.

[28] J. M. Soler, E. Artacho, J. D. Gale, A. García, J. Junquera, P. Ordejón, and D. Sánchez-Portal, "The SIESTA method for ab initio order N materials simulation," *J. Phys.: Condens. Matter*, vol. 14, p. 2745, 2002.

[29] L. Kronik, A. Makmal, M. L. Tiago, M. M. G. Alemany, M. Jain, X. Huang, Y. Saad, and J. R. Chelikowsky, "PARSEC - the pseudopotential algorithm for real-space electronic structure calculations: recent advances and novel applications to nano-structures," *Phys. Status Solidi B*, vol. 243, p. 1063, 2006.

[30] V. Michaud-Rioux, L. Zhang, and H. Guo, "RESCU: A real space electronic structure method," *J. Comp. Phys.*, vol. 307, p. 593, 2016.

[31] A. S. Banerjee, R. S. Elliott, and R. D. James, "A spectral scheme for Kohn-Sham density functional theory of clusters," *J. Comp. Phys.*, vol. 287, p. 226, 2015.

[32] J. Deslippe, G. Samsonidze, D. A. Strubbe, M. Jain, M. L. Cohen, and S. G. Louie, "BerkeleyGW: A massively parallel computer package for the calculation of the quasiparticle and optical properties of materials and nanostructures," *Comp. Phys. Comm.*, vol. 183, p. 1269, 2012.

[33] A. J. Garza and G. E. Scuseria, "Predicting band gaps with hybrid density functionals," *J. Phys. Chem. Lett.*, vol. 7, p. 4165, 2016.

[34] M. Aldegunde, J. R. Kermode, and N. Zabaras, "Development of an exchange-correlation functional with uncertainty quantification capabilities for density functional theory," *J. Comp. Phys.*, vol. 311, p. 173, 2016.

[35] G. Kotliar, S. Y. Savrasov, K. Haule, V. S. Oudovenko, O. Parcollet, and C. A. Marianetti, "Electronic structure calculations with dynamical mean-field theory," *Rev. Mod. Phys.*, vol. 78, p. 865, 2006.

[36] W. Hu, L. Lin, and C. Yang, "DGDFT: A massively parallel method for large scale density functional theory calculations," *J. Chem. Phys.*, vol. 143, p. 124110, 2015.

[37] T. V. T. Duy and T. Ozaki, "A three-dimensional domain decomposition method for large-scale DFT electronic structure calculations," *Comp. Phys. Comm.*, vol. 185, p. 777, 2014.

[38] S. Mohr, L. E. Ratcliff, L. Genovese, D. Caliste, P. Boulanger, S. Goedecker, and T. Deutsch, "Accurate and efficient linear scaling DFT calculations with universal applicability," *Phys. Chem. Chem. Phys.*, vol. 17, p. 31360, 2015.

[39] Y. Pan, X. Dai, S. de Gironcoli, X.-G. Gong, G.-M. Rignanese, and A. Zhou, "A parallel orbital-updating based plane-wave basis method for electronic structure calculations," *J. Comp. Phys.*, vol. 348, p. 482, 2017.

[40] E. Vecharynski, C. Yang, and J. E. Pask, "A projected preconditioned conjugate gradient algorithm for computing many extreme eigenpairs of a hermitian matrix," *J. Comp. Phys.*, vol. 290, p. 73, 2015.

[41] M. Lee, K. Leiter, C. Eisner, and J. Knap, "Atom-partitioned multipole expansions for electrostatic potential boundary conditions," *J. Comp. Phys.*, vol. 328, p. 344, 2017.

[42] G. Zhang, L. Lin, W. Hu, C. Yang, and J. E. Pask, "Adaptive local basis set for Kohn-Sham density functional theory in a discontinuous Galerkin framework II: Force, vibration, and molecular dynamics calculations," *J. Comp. Phys.*, vol. 335, p. 426, 2017.

[43] D. Porezag, T. Frauenheim, T. Köhler, G. Seifert, and R. Kaschner, "Construction of tight-binding-like potentials on the basis of density-functional theory: Application to carbon," *Phys. Rev. B*, vol. 51, p. 12947, 1995.

[44] M. Elstner, D. Porezag, G. Jungnickel, J. Elsner, M. Haugk, T. Frauenheim, S. Suhai, and G. Seifert, "Self-consistent-charge density-functional tight-binding method for simulations of complex materials properties," *Phys. Rev. B*, vol. 58, p. 7260, 1998.

[45] K. Kitaura, E. Ikeo, T. Asada, T. Nakano, and M. Uebayasi, "Fragment molecular orbital method: an approximate computational method for large molecules," *Chem. Phys. Lett.*, vol. 313, p. 701, 1999.

[46] L.-W. Wang and A. Zunger, "Local-density-derived semiempirical pseudopotentials," *Phys. Rev. B*, vol. 51, p. 17398, 1995.

[47] H. Fu and A. Zunger, "Local-density-derived semiempirical nonlocal pseudopotentials for InP with applications to large quantum dots," *Phys. Rev. B*, vol. 55, p. 1642, 1997.

[48] L.-W. Wang, "Charge-density patching method for unconventional semiconductor binary systems," *Phys. Rev. Lett.*, vol. 88, p. 256402, 2002.

[49] L.-W. Wang, "Generating charge densities of fullerenes," *Phys. Rev. B*, vol. 65, p. 153410, 2002.

[50] N. Vukmirović and L.-W. Wang, "Charge patching method for electronic structure of organic systems," *J. Chem. Phys.*, vol. 128, p. 121102, 2008.

[51] L.-W. Wang, "Novel computational methods for nanostructure electronic structure calculations," *Annual review of physical chemistry*, vol. 61, pp. 19–39, 2010.

[52] N. Godbout, D. R. Salahub, J. Andzelm, and E. Wimmer, "Optimization of Gaussian-type basis sets for local spin density functional calculations. Part I. Boron through neon, optimization technique and validation," *Can. J. Chem.*, vol. 70, p. 560, 1992.

[53] K. L. Schuchardt, B. T. Didier, T. Elsethagen, L. Sun, V. Gurumoorthi, J. Chase, J. Li, and T. L. Windus, "Basis set exchange: A community database for computational sciences," *J. Chem Inf. Model.*, vol. 47, p. 1045, 2007.

[54] D. Feller, "The role of databases in support of computational chemistry calculations," *J. Comp. Chem.*, vol. 17, p. 1571, 1996.

[55] A. Canning, L. W. Wang, A. Williamson, and A. Zunger, "Parallel empirical pseudopotential electronic structure calculations for million atom systems," *J. Comp. Phys.*, vol. 160, p. 29, 2000.

[56] N. Vukmirovic and L.-W. Wang, "Overlapping fragments method for electronic structure calculation of large systems," *J. Chem. Phys.*, vol. 134, p. 094119, 2011.

[57] N. Vukmirovic and L.-W. Wang, "Density of states and wave function localization in disordered conjugated polymers: a large scale computational study," *J. Phys. Chem. B*, vol. 115, p. 1792, 2011.

[58] L.-W. Wang, "Large-scale local-density-approximation band gap-corrected GaAsN calculations," *Appl. Phys. Lett.*, vol. 78, p. 1565, 2001.

[59] J. Li and L.-W. Wang, "Band-structure-corrected local density approximation study of semiconductor quantum dots and wires," *Phys. Rev. B*, vol. 72, p. 125325, 2005.

[60] D. J. Milliron, S. M. Hughes, Y. Cui, L. Manna, J. Li, L. W. Wang, and A. P. Alivisatos, "Colloidal nanocrystal heterostructures with linear and branched topology," *Nature (London)*, vol. 430, p. 190, 2004.

[61] J. Li and L.-W. Wang, "Energy levels of isoelectronic impurities by large scale LDA calculations," *Phys. Rev. B*, vol. 67, p. 033102, 2003.

[62] J. Li and L.-W. Wang, "First principles calculations of ZnS:Te energy levels," *Phys. Rev. B*, vol. 67, p. 205319, 2003.

[63] N. Vukmirović and L.-W. Wang, "Electronic structure of disordered conjugated polymers: Polythiophenes," *J. Phys. Chem. B*, vol. 113, pp. 409–415, 2009.

[64] N. Vukmirovic and L.-W. Wang, "Charge carrier motion in disordered conjugated polymers: A multiscale ab initio study," *Nano Lett.*, vol. 9, p. 3996, 2009.

[65] N. Vukmirovic and L.-W. Wang, "Carrier hopping in disordered semiconducting polymers: How accurate is the Miller–Abrahams model?," *Appl. Phys. Lett.*, vol. 97, p. 043305, 2010.

[66] J. Granadino-Roldan, N. Vukmirović, M. Fernandez-Gomez, and L.-W. Wang, "The role of disorder on the electronic structure of conjugated polymers. The case of poly-2,5-bis(phenylethynyl)-1,3,4-thiadiazole," *Phys. Chem. Chem. Phys.*, vol. 13, p. 14500, 2011.

[67] M. Mladenović and N. Vukmirović, "Effects of thermal disorder on the electronic properties of ordered polymers," *Phys. Chem. Chem. Phys.*, vol. 16, pp. 25950–25958, 2014.

[68] M. Mladenović and N. Vukmirović, "Electronic states at the interface between crystalline and amorphous domains in conjugated polymers," *J. Phys. Chem. C*, vol. 119, p. 23329, 2015.

[69] M. Mladenović, N. Vukmirović, and I. Stanković, "Electronic states at low-angle grain boundaries in polycrystalline naphthalene," *J. Phys. Chem. C*, vol. 117, p. 15741, 2013.

[70] M. Towler, "An introductory guide to Gaussian basis sets in solid-state electronic structure calculations," *European Summer School" Ab initio modelling in solid-state chemistry", Turin*, 2000.

[71] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office Los Angeles, CA, 1950.

[72] R. B. Morgan and D. S. Scott, "Generalizations of Davidson's method for computing eigenvalues of sparse symmetric matrices," *SIAM Journal on Scientific and Statistical Computing*, no. 3, pp. 817–825, 1986.

[73] Y. Saad, *Numerical methods for large eigenvalue problems*. Manchester University Press, 1992.

[74] K. Moszyński, "Bn parlett; the symmetric eigenvalue problem," *Mathematica Applicanda*, vol. 14, no. 27, pp. 218–219, 2016.

[75] H. D. Simon, "Analysis of the symmetric Lanczos algorithm with reorthogonalization methods," *Linear algebra and its applications*, vol. 61, pp. 101–131, 1984.

[76] R. M. Larsen, *Efficient algorithms for helioseismic inversion*. PhD thesis, Ph. D. thesis, University of Aarhus, Denmark, 1998.

[77] H. D. Simon, "The Lanczos algorithm with partial reorthogonalization," *Mathematics of computation*, vol. 42, no. 165, pp. 115–142, 1984.

[78] R. M. Larsen, "Lanczos bidiagonalization with partial reorthogonalization," *DAIMI Report Series*, no. 537, 1998.

[79] K. Wu and H. Simon, "A parallel Lanczos method for symmetric generalized eigenvalue problems," tech. rep., SCAN-9807131, 1997.

[80] Y. Saad, A. Stathopoulos, J. Chelikowsky, K. Wu, and S. Öğüt, "Solution of large eigenvalue problems in electronic structure calculations," *BIT Numerical Mathematics*, vol. 36, no. 3, pp. 563–578, 1996.

[81] X. Gonze, J.-M. Beuken, R. Caracas, F. Detraux, M. Fuchs, G.-M. Rignanese, L. Sindic, M. Verstraete, G. Zerah, F. Jollet, *et al.*, "First-principles computation of material properties: the ABINIT software project," *Computational Materials Science*, vol. 25, no. 3, pp. 478–492, 2002.

[82] LAPACK, http://www.netlib.org/lapack, last accessed 18 April 2019.

[83] V. Hernandez, J. E. Roman, A. Tomas, and V. Vidal, "Krylov-Schur methods in SLEPc,"

[84] G. W. Stewart, "A Krylov-Schur algorithm for large eigenproblems," *SIAM Journal on Matrix Analysis and Applications*, vol. 23, pp. 601–614, 2002.

[85] J. E. Roman, C. Campos, E. Romero, and A. Tomas, "Slepc users manual," *D. Sistemes Informàtics i Computació Universitat Politècnica de València, Valencia, Spain, Report No. DSIC-II/24/02*, 2015.

[86] J. S. Binkley, J. A. Pople, and W. J. Hehre, "Self-consistent molecular orbital methods. 21. Small split-valence basis sets for first-row elements," *J. Am. Chem. Soc.*, vol. 102, p. 939, 1980.

[87] W. J. Stevens, H. Basch, and M. Krauss, "Compact effective potentials and efficient shared-exponent basis sets for the first- and second-row atoms," *J. Chem. Phys.*, vol. 81, p. 6026, 1984.

[88] T. Petersson and B. Hellsing, "A detailed derivation of gaussian orbital-based matrix elements in electron structure calculations," *Eur. J. Phys.*, vol. 31, p. 37, 2010.

[89] Ž. Bodroški, N. Vukmirović, and S. Škrbić, *Towards the High Performance Method for Large-Scale Electronic Structure Calculations*, p. 90. Springer International Publishing, 2016.

[90] B. A. Mamedov, "On the evaluation of boys functions using downward recursion relation," *Journal of Mathematical Chemistry*, vol. 36, pp. 301–306, 2004.

[91] L. E. McMurchie and E. R. Davidson, "Calculation of integrals over ab initio pseudopotentials," *J. Comp. Phys.*, vol. 44, p. 289, 1981.

[92] P. M. W. Gill, "Molecular integrals over Gaussian basis functions," *Adv. Quantum Chem.*, vol. 25, p. 141, 1994.

[93] M. Head-Gordon and J. A. Pople, "A method for two-electron Gaussian integral and integral derivative evaluation using recurrence relations," *J. Chem. Phys.*, vol. 89, p. 5777, Nov. 1988.

[94] C. A. White, B. G. Johnson, P. M. Gill, and M. Head-Gordon, "The continuous fast multipole method," *Chem. Phys. Lett.*, vol. 230, p. 8, 1994.

[95] E. Rudberg and P. Salek, "Efficient implementation of the fast multipole method," *J. Chem. Phys.*, vol. 125, p. 084106, 2006.

[96] E. A. Toivanen, S. A. Losilla, and D. Sundholm, "The grid-based fast multipole method - a massively parallel numerical scheme for calculating two-electron interaction energies," *Phys. Chem. Chem. Phys.*, vol. 17, p. 31480, 2015.

[97] L. Füsti-Molnár and P. Pulay, "The Fourier transform Coulomb method: Efficient and accurate calculation of the Coulomb operator in a Gaussian basis," *J. Chem. Phys.*, vol. 117, p. 7827, 2002.

[98] L. Genovese, A. Neelov, S. Goedecker, T. Deutsch, S. A. Ghasemi, A. Willand, D. Caliste, O. Zilberberg, M. Rayson, A. Bergman, and R. Schneider, "Daubechies wavelets as a basis set for density functional pseudopotential calculations," *J. Chem. Phys.*, vol. 129, p. 014109, 2008.

[99] O. Treutler and R. Ahlrichs, "Efficient molecular numerical integration schemes," *J. Chem. Phys.*, vol. 102, p. 346, 1995.

[100] A. D. Becke, "A multicenter numerical integration scheme for polyatomic molecules," *J. Chem. Phys.*, vol. 88, p. 2547, 1988.

[101] V. Lebedev, "Values of the nodes and weights of ninth to seventeenth order Gauss-Markov quadrature formulae invariant under the octahedron group with inversion," *USSR Comput. Math. Math. Phys.*, vol. 15, p. 44, 1975.

[102] P. Manninen, "554017 advanced computational chemistry," pp. 66–68, 2009.

[103] P. Pulay, "Convergence acceleration of iterative sequences. The case of SCF iteration," *Chemical Physics Letters*, vol. 73, pp. 393–398, 1980.

[104] P. Pulay, "Improved SCF convergence acceleration," *Journal of Computational Chemistry*, vol. 3, pp. 556–560, 1982.

[105] Message Passing Interface Forum, "Mpi: A message-passing interface standard, version 3.1," *tech. rep., University of Tennessee, Knoxville, Tennessee*, 2015.

[106] R. W. Vuduc, *Automatic Performance Tuning of Sparse Matrix Kernels.* PhD Thesis, University of California, Berkeley, 2003.

[107] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, "PETSc Web page." `http://www.mcs.anl.gov/petsc`, 2019.

[108] S. Abhyankar, J. Brown, E. M. Constantinescu, D. Ghosh, B. F. Smith, and H. Zhang, "Petsc/ts: A modern scalable ode/dae solver library," *arXiv preprint arXiv:1806.01437*, 2018.

[109] Y. Saad and M. H. Schultz, "Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on scientific and statistical computing*, vol. 7, pp. 856–869, 1986.

[110] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. Gropp, *et al.*, "Petsc users manual," 2019.

[111] netlib, http://www.netlib.org/utk/people/JackDongarra/la-sw.html, last accessed 14 August 2019.

[112] V. Hernandez, J. E. Roman, and V. Vidal, "SLEPc: Scalable Library for Eigenvalue Problem Computations," *Lect. Notes Comput. Sci.*, vol. 2565, pp. 377–391, 2003.

[113] V. Hernandez, J. E. Roman, and V. Vidal, "SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems," *ACM Trans. Math. Software*, vol. 31, pp. 351–362, 2005.

[114] X. S. Li and J. W. Demmel, "Superlu_dist: A scalable distributed-memory sparse direct solver for unsymmetric linear systems," *ACM Transactions on Mathematical Software (TOMS)*, vol. 29, pp. 110–140, 2003.

Biografija

Žarko Bodroški je rođen 1982. godine u Srbiji. Završio je osnovnu školu "1. oktobar" u Botošu a zatim srednju elektrotehničku školu "Nikola Tesla" u Zrenjaninu, na smeru elektrotehničar računara. Odbranom diplomskog rada "ASP.net web partovi i web komponente" je diplomirao informatiku na Prirodno-matematičkom fakultetu na Univerzitetu u Novom Sadu u 2006. godini. 2009. godine brani master tezu "Razvoj web servisa informacionog sistema Prirodno-matematičkog fakulteta sa primenom open source tehnologija".

Aktuelni istraživački pravci u toku doktorskih studija računarskih nauka na Prirodno-matematičkom fakultetu su primena računara u nauci i paralelno računarstvo visokih performansi na modelovanju elektronske strukture materijala korišćenjem teorije funkcionala gustine (Density Functional Theory – DFT), uz primenu MPI i programskog jezika C. Istraživanje se obavlja u saradnji sa Laboratorijom za primenu računara u nauci, Centar za izučavanje kompleksnih sistema Instituta za fiziku u Beogradu.

Učestvovao je kao istraživač saradnik na nacionalnom naučnom projektu: inteligentne tehnike i njihova integracija u sisteme za podršku odlučivanja sa širokim poljem primena. Učestvovao je u istraživačkom SCOPES projektu: Developing Capacity for High-Productivity Large-Scale Computing. Takođe je bio aktivni učesnik u razvoju IT infrastrukture za eLTER H2020 projekat (H2020 INFRAIA) od 2015. do 2018. godine.

Trenutno je aktivan na tri H2020 projekta: "Cyber security 4.0: protecting the Industrial Internet Of Things (C4IIoT)", "A COmprehensive cyber-intelligence framework for resilient coLLABorative manufacturing Systems (COLLABS)" i "eLTER Advanced Community Project (eLTER PLUS)". U toku studija je učestvovao na nekoliko stručnih usavršavanja organizovanih od strane PRACE-a (Partnership for Advanced Computing in Europe): "HLRS Parallel Programming Workshop – MPI, OpenMP, and PETSc", HLRS, Nemačka, "Node-Level Performance Engineering LRZ Leibniz Computer Centre", Minhen, Nemačka, "Debugging and Optimization of Scientific Aplications", CINECA, Bolonja, Italija i drugim.

Takođe je član Laboratorije za razvoj informacionih sistema, kao i grupe za Primenu računara u nauci (Scientific Computing Reseach Group – SCORG) na Departmanu za matematiku i informatiku na Prirodno-matematičkom fakultetu.

Objavio je rad kategorije M21a, četiri rada kategorije M33, rad kategorije M63 i rad kategorije M51.

Univerzitet u Novom Sadu

Prirodno-matematički fakultet

Ključna dokumentacijska informacija

Redni broj:	
RBR	
Identifikacioni broj:	
IBR	
Tip dokumentacije:	Monografska dokumentacija
TD	
Tip zapisa:	Tekstualni štampani materijal
TZ	
Vrsta rada:	Doktorska disertacija
VR	
Ime i prezime autora:	Žarko Bodroški
AU	
Mentor :	dr Srđan Škrbić i dr Nenad Vukmirović
MN	
Naslov rada:	Razvoj serijskog i paralelnog algoritma za računanje elektronske strukture materijala metodom sklapanja naelektrisanja
NR	
Jezik publikacije:	Srpski
JP	
Jezik izvoda:	Srpski/Engleski
JI	
Zemlja publikovanja:	Srbija
ZP	
Uže geografsko područje:	Vojvodina
UGP	
Godina:	2020
GO	

Izdavač:
IZ Autorski reprint
Mesto i adresa: Novi Sad, Trg Dositeja Obradovića 4
MA
Fizički opis rada: 6/134/16/14/86/0/10/20
(broj poglavlja/stranica/slika/grafikona/referenci/
priloga/tabela/listinga)
FO
Naučna oblast: Informatika
NO
Naučna disciplina: Računarske nauke
ND
Predmetna odrednica/Ključne reči: Numeričke simulacije, paralelni algoritmi, parcijalne difirencijalne jednačine, DFT, CPM, MPI
PO
UDK
Čuva se:
ČU
Važna napomena:
VN

Izvod:

U tezi je predstavljena implementacija metode teorija funkcionala gustine (DFT) bazirana na metodi za sklapanje naelektrisanja (CPM) koja koristi bazise gausijanskih funkcija. Metod je baziran na pretpostavci da se elektronska gustina naelektrisanja velikih sistema, može predstaviti kao suma doprinosa pojedinačnih atoma, takozvanih motiva gustine naelektrisanja, koji se dobijaju računanjem malog prototip sistema. Talasna funkcija, kao i gustina naelektrisanja, se u našoj implementaciji reprezentuju uz pomoć bazise gausijanskih funkcija, dok se motivi opisuju korišćenjem prostornih koordinata. Uz pomoć procedure za minimizaciju se iz motiva opisanih koordinatama, dobija gustina naelektrisanja predstavljena u bazisu Gausijana. Implementacija serijskog programa pokazuje značajno poboljšanje u performansama u odnosu na prethodne implementacije bazirane na ravnim talasima. Ova implementacija rešava sistem od približno 1000 atoma na jednom procesorskom jezgru za svega nekoliko sati. Paralelna implementacija uz pomoć naprednih metoda paralelizacije i distribucije podataka omogućava rešavanje sistema od više desetina hiljada atoma. Najveći testirani sistem ima približno 20000 atoma i testiran je na 256 paralelnih procesa.

IZ

Datum prihvatanja teme od strane NN veća:	5.9.2018.
DP	
Članovi komisije:	
(Naučni stepen/ime i prezime/zvanje/fakultet)	
KO	
Predsednik:	dr Miloš Racković, redovni profesor, Prirodno-matematički fakultet, Univerzitet u Novom Sadu
Mentor:	dr Srđan Škrbić, redovni profesor, Prirodno-matematički fakultet, Univerzitet u Novom Sadu
Mentor:	dr Nenad Vukmirović, naučni savetnik, Institut za fiziku u Beogradu
Član:	dr Antun Balaž, naučni savetnik, Institut za fiziku u Beogradu
Član:	dr Miloš Radovanović, vanredni profesor, Prirodno-matematički fakultet, Univerzitet u Novom Sadu

University of Novi Sad

Faculty of Sciences

Key word documentation

Accession number: ANO	
Identification number: INO	
Document type: DT	Monograph documentation
Type of record: TR	Textual printed material
Contents code: CC	Doctoral dissertation
Author: AU	Žarko Bodroški
Mentor: MN	Dr. Srđan Škrbić and Dr. Nenad Vukmirović
Title: TI	Development of Serial and Parallel Algorithms for Computing the Electronic Structure of Materials Using the Charge Patching Method
Language of text: LT	Serbian
Language of abstract: LA	English/Serbian
Country of publication: CP	Serbia
Locality of publication: LP	Vojvodina

Publication year: PY	2020
Publisher: PU	Author's reprint
Publication place: PP	Novi Sad, Trg Dositeja Obradovića 4
Physical description: PD	6/134/16/14/86/0/10/20 (no. of chapters/pages/figures/graphs/bib. refs/ appendices/tables/listings)
Scientific field SF	Informatics
Scientific discipline SD	Computer Science
Subject, Key words: SKW	Numerical simulations, parallel algorithms, partial differential equations, DFT, CPM, MPI
UC	
Holding data: HD	
Note: N	

Abstract:

We present the implementation of the density functional theory (DFT) based charge patching method (CPM) using the basis of Gaussian functions. The method is based on the assumption that the electronic charge density of a large system is the sum of contributions of individual atoms, so called charge density motifs, that are obtained from calculations of small prototype systems. In our implementation wave functions and electronic charge density are represented using the basis of Gaussian functions, while charge density motifs are represented using a real space grid. A constrained minimization procedure is used to obtain Gaussian basis representation of charge density from real space representation of motifs. The code based on this implementation exhibits superior performance in comparison to previous implementation of the charge patching method using the basis of plane waves. It enables calculations of electronic structure of systems with around 1000 atoms on a single CPU core with computational time of just several hours. The parallel implementation enables calculations for the system with more than ten thousand atoms. The largest system tested has around 20000 atoms and was computed on 256 parallel processes.

AB

Accepted on Senate on:	September 5.2018.
AS	
Defended:	
DE	
Thesis Defend Board:	
(Degree/first and last name/title/faculty)	
KO	
President:	Dr. Miloš Racković, full professor, Faculty of Sciences, University of Novi Sad
Advisor:	Dr. Srđan Škrbić, full professor, Faculty of Sciences, University of Novi Sad
Advisor:	Dr. Nenad Vukmirović, research professor, Institute of Physics Belgrade
Member:	Dr. Antun Balaž, research professor, Institute of Physics Belgrade
Member:	Dr. Miloš Radovanović, associate professor, Faculty of Sciences, University of Novi Sad