Univerza na Primorskem
Fakulteta za matematiko, naravoslovje in informacijske tehnologije

Staš Bevc

Razvoj računalniških orodij za molekularno modeliranje

DOKTORSKA DISERTACIJA

Mentor: izred. prof. dr. Matej Praprotnik
Somentorica: prof. dr. Dušanka Janežič

Koper, september 2013

Zahvala

Rad bi se zahvalil mentorju Mateju Praprotniku in somentorici Dušanki Janežič za napotke, usmerjanje in strokovno pomoč pri izdelavi doktorske disertacije. Zahvalil bi se staršema, ker sta me spodbujala in motivirala v času doktorskega študija. Zahvalil bi se sodelavcem Janezu Koncu, Milanu Hodoščku ter Juretu Stojanu za pomoč in nasvete pri izdelavi aplikacije ENZO. Zahvala gre Laboratoriju za molekularno modeliranje L17 na Kemijskem inštitutu, ker sem uporabljal njihove prostore in opremo. Na Max Planck institute for polymer research v Mainzu v Nemčiji sem bil na štirimesečnem delovnem obisku pri teorijski skupini, za kar sem jim hvaležen. Prav tako bi se zahvalil Javni agenciji za raziskovalno dejavnost Republike Slovenije za financiranje študija ter Fakulteti za matematiko, naravoslovje in informacijske tehnologije Univerze na Primorskem, kjer sem opravljal doktorski študij. Nazadnje gre zahvala tudi sodelavcema Juriju Sabliću in Juliji Zavadlav, ki sta mi pomagala z raznimi nasveti iz področja fizike.

Povzetek: **Razvoj računalniških orodij za molekularno modeliranje**

V doktorski disertaciji smo razvili računalniška orodja za molekularno modeliranje. Vsebinsko je doktorska disertacija razdeljena na dva dela. Prvi del je posvečen razvoju algoritmov za simulacijo molekulske dinamike, drugi pa razvoju računalniških orodij za encimsko kinetiko.

Veliko procesov v molekularnih tekočinah in mehki snovi vsebuje velik razpon različnih časovnih in krajevnih skal, ki so med seboj prepletene. Uporaba večskalnih tehnik modeliranja, ki istočasno združuje različne krajevne skale, predstavlja zelo učinkovit način za obravnavo takšnih sistemov z računalniško simulacijo. Z večskalnim pristopom poenostavimo fizikalni model do največje možne stopnje, pri čemer pa zadržimo vse pomembne podrobnosti v tistih delih sistema, kjer je to potrebno.

Metodo prilagodljive ločljivosti AdResS, ki omogoča združitev dveh opisov sistema z različnima ločljivostma v eni sami simulaciji, smo implementirali v programski paket ESPResSo++, tako da omogoča poganjanje na več procesorskih jedrih. Implementacija metode AdResS temelji na dejstvu, da nam interpolacijske sheme za izračun sile ni potrebno računati preko celotnega razpona simulacijske škatle, ampak le na delcih znotraj hibridnega območja. V ta namen zgradimo tri Verletove sezname, kjer v vsakem sile računamo drugače. Pri poganjanju na več procesorskih enotah se vsi atomi nekega grobozrnatega delca zmeraj nahajajo na istem procesorju. Za prenos iz enega procesorja na drugega, se pri vpisu grobozrnatega delca v medpomnilnik sproži signal, ki poleg zapiše še pripadajoče atome. Komunikacijski klic nato prenese podatke iz medpomnilnika na drugi procesor.

Tako implementirana vzporedna metoda AdResS omogoča izvedbo simulacije raztopine soli v vodi, ki je poleg vode same eno najpomembnejših topil v simulacijah bioloških sistemov. Sočasno združimo vseatomski in grobozrnati model raztopine soli. Efektivne potenciale za interakcije med molekulami vode-vode in vode-ioni smo dobili s postopkom grobozrnenja, ki temelji na reproduciranju strukturnih lastnosti. Da zagotovimo enakomerno gostotno porazdelitev preko celotne simulacijske škatle, smo uporabili termodinamske sile. Pokažemo, da naša metoda prilagodljive ločljivosti pravilno reproducira ravnovesne strukturne ter dinamične lastnosti sistema. Vpeljali smo splošno uporaben pristop, ki ga lahko uporabimo s katerimkoli poljem sil ali vrsto ionov. Naš pristop omogoča učinkovito obravnavo bioloških sistemov, ki vključujejo raztopino soli.

Pri študiji encimske kinetike merimo hitrosti trošenja reaktantov, da določimo koncentracije posameznih vrst, ki nastopajo v kemijski reakciji. Potek reakcije opišemo z diferencialnimi enačbami. Izdelali smo spletno orodje ENZO s preprostim uporabniškim vmesnikom, ki omogoča, da zapletene reakcijske mehanizme predstavimo s shemami. ENZO iz sheme samodejno izdela diferencialne enačbe ter prilega njihove koeficiente eksperimentalnim podatkom. V primeru ujemanja sklepamo, da je predlagani kinetični model možen. Orodje uspešno preizkusimo na treh realnih primerih. Na voljo je kot prosto dostopna aplikacija na naslovu http://enzo.cmm.ki.si/.

Ključne besede: računalniška simulacija, molekularno modeliranje, simulacija molekulske dinamike, metoda prilagodljive ločljivosti, AdResS, večskalna simulacija, ESPResSo++, spletna aplikacija, kinetika encimov, ENZO

PACS: 02.70.Ns, 61.20.Ja, 61.25.Em, 02.60.Cb, 02.60.Ed

Abstract: **Development of computer tools for molecular modeling**

In this doctoral thesis we develop computer tools for molecular modeling. In terms of content, the thesis is divided into two parts. The first part is devoted to developing algorithms for molecular dynamics simulations, and the second part is devoted to developing computer software tools for enzyme kinetics.

Many processes in molecular liquids and soft matter systems contain a wide range of different length and time scales, which are interconnected. Using multiscale modeling techniques that simultaneously combine different length scales, allows for a very efficient way to deal with such systems by means of computer simulation. With a multiscale approach we simplify the physical model to the highest possible degree, while at the same time retaining all the relevant details in those parts of the system where necessary.

We implement the adaptive resolution scheme (AdResS), which allows us to couple concurrently two systems of different resolutions in one simulation, into the software package ESPResSo++ with support for parallel execution. The implementation is based on the fact, that one does not need to use the force interpolation scheme across the whole simulation box, but only for the particles located in the hybrid region. For this reason, we build three Verlet lists and calculate the forces appropriately. When running in parallel on multiple computer nodes, the implementation makes sure that all atomistic particles belonging to a certain coarse-grained particle are always located on the same node. As soon as a given coarse-grained particle is written into the communication buffer, a signal is triggered that also writes the corresponding atomistic particles into the buffer. A single communication call then transfers the contents of the buffer to another processor.

Such a parallel implementation of AdResS enables us to perform multiscale simulations of salt solution, which is besides water itself, one of the most important solvents in simulations of biological systems. We concurrently couple the atomistic and coarse-grained models of the aqueous NaCl. The effective interactions for water-water and water-ion interactions in the coarse-grained model are derived using structure-based coarse-graining approach. To assure an even distribution of molecules across the simulation box we employ thermodynamic forces. We demonstrate that the equilibrium structural and dynamical properties are correctly reproduced by our adaptive resolution method. Our multiscale approach, which is general and can be used for any classical force-field and/or types of ions, will significantly speed up biomolecular simulation involving aqueous salt.

In the enzyme kinetics study one measures reactant rates to determine the concentration of species taking part in a chemical reaction. The course of the reaction is described with differential equations. We develop a web tool ENZO with a simple user interface, which allows representing complex reaction mechanisms with graphical schemes. ENZO generates differential equations and fits their coefficients to experimental data. If convergence is reached, the proposed kinetic model is considered plausible. We successfully test the tool on three actual enzyme-kinetics scenarios. The tool is publicly available as a web application at http://enzo.cmm.ki.si/.

Keywords: computer simulation, molecular modeling, molecular dynamics, adaptive resolution scheme, AdResS, multiscale simulation, ESPResSo++, web application, enzyme kinetics, ENZO

PACS: 02.70.Ns, 61.20.Ja, 61.25.Em, 02.60.Cb, 02.60.Ed

Kazalo vsebine

Kazalo slik

Kazalo tabel

1

Uvod

Pri molekularnem modeliranju uporabljamo teoretično-računske metode za razumevanje povezav med strukturo in fizikalno-kemijskimi lastnostmi molekul ter spojin, za njihovo racionalno načrtovanje ter dodelavo eksperimentalnih strukturnih podatkov. Molekularno modeliranje temelji na kvantitativnem matematično-fizikalnem opisu odnosov med atomi in molekulami, ki ga omogočajo fizikalne zakonitosti zgradbe snovi. Tako lahko razvoj dogodkov za nek sistem zapišemo v obliki algoritmov, ki jih lahko nadalje implementiramo v računalniške programe [1].

Namen doktorske disertacije je razvoj orodij za molekularno modeliranje. Vsebinsko je doktorska disertacija razdeljena na dva dela. Prvi del je posvečen razvoju algoritmov za simulacijo molekulske dinamike, drugi del pa razvoju računalniških orodij za encimsko kinetiko.

Pri simulaciji molekulske dinamike uporabljamo računalnike za preučevanje strukture in dinamike kompleksnih molekularnih sistemov kot posledice mikroskopskih interakcij med atomi in molekulami. Simulacija molekulske dinamike dopolnjuje običajne eksperimentalne in teoretične metode. S primerjavo rezultatov simulacije in rezultatov eksperimenta lahko preverjamo pravilnost neke teorije ali modela. Poleg tega je z računalnikom možno izvajati simulacije v razmerah, ki jih je težko ali nemogoče izvesti v laboratoriju [2, 3].

V simulaciji molekulske dinamike obravnavamo atome kot točkaste delce z maso. Med njimi vladajo sile, ki opisujejo njihove interakcije. Sile izračunamo glede na medsebojne razdalje med vsemi pari delcev v sistemu. Z integriranjem Newtonovih enačb gibanja napredujemo položaje in hitrosti delcev skozi čas. Časovna zahtevnost izračuna sil je reda n^2, kjer je n število delcev. Zaradi tega je velikost sistema, ki ga lahko simuliramo v praktično uporabnem času, omejena. Prav tako smo omejeni z velikostjo časovnega koraka, saj mora biti dovolj kratek, da pravilno opišemo najvišjo frekvenco gibanja v sistemu [4–6]. Eden od načinov pohitritve simulacij je t. i. grobozrnenje, kjer skupino atomov združimo v večjo enoto, s čimer zmanjšamo število prostostnih stopenj ter posledično število podatkov, ki jih mora računalnik obdelati. Izvajanje simulacij pa tudi znatno pohitrimo z vzporednim računanjem na več procesorskih enotah. Pri tem problem simuliranja razdelimo na podprobleme tako, da simulacijski prostor razdelimo v manjše podprostore in jih dodelimo ločenim procesorjem. Število delcev na procesor se zato zmanjša.

Veliko procesov v molekularnih tekočinah in mehki snovi vsebuje velik razpon različnih časovnih in krajevnih skal, ki so med seboj prepletene. Vseatomske simulacije, ki zaobjemajo pojave na atomski skali, so včasih računsko neizvedljive ali celo nezaželene zaradi velikega števila prostostnih stopenj v teh sistemih. Grobozrnati modeli po drugi strani lahko pokrijejo veliko daljše časovne in krajevne skale, a ne morejo podati informacije na atomski

ločljivosti. Uporaba večskalnih tehnik modeliranja, ki istočasno združuje različne krajevne skale, predstavlja zelo učinkovit način za obravnavo takšnih sistemov z računalniško simulacijo [7–24]. Z večskalnim pristopom poenostavimo fizikalni model do največje možne stopnje, pri čemer pa zadržimo vse pomembne podrobnosti v tistih delih sistema, kjer je to potrebno [25]. V preteklih nekaj letih so v ta namen razvili hibridno večskalno metodo AdResS (Adaptive Resolution Scheme), ki istočasno združuje atomske in mezoskopske krajevne skale [26, 27]. Metoda omogoča združitev atomističnega in grobozrnatega opisa ter pri tem dopušča možnost prostega prehoda delcev med obema sistemoma. Dosedanja implementacija metode je razvita za potrebe metodološkega testiranja in teče le na enem procesorju [28].

Za učinkovito uporabo bomo razvili vzporedno različico metode AdResS, ki omogoča izvajanje na več procesorjih ter jo vključili v enega od programskih paketov za računanje simulacij molekulske dinamike. S tem bo postala dostopna širokemu krogu uporabnikov. Za enega najnovejših takšnih programskih paketov za simulacijo molekulske dinamike velja ESPResSo++ [29], ki ga razvijamo v skladu s sodobnimi programerskimi trendi: je odprtokoden, modularen, programiran v objektno usmerjeni tehniki, z vmesnikom Python in jedrom C++. V sklopu doktorske disertacije bomo vzporedno metodo AdResS vgradili v ESPResSo++.

Tako implementirano vzporedno metodo AdResS bomo uporabili za simulacije raztopine soli. Pri simulacijah raztopin potrebujemo velike sisteme z več deset tisoč delci zaradi nizke koncentracije ionov. Za simulacijo tipične raztopine, tj. soli v vodi, mora biti zato koda programa učinkovito programirana s podporo vzporednega izvajanja. Uporaba na sistemu soli zahteva tudi metodološko posplošitev metode, tj. vključitev posplošenega reakcijskega polja [30] za obravnavo elektrostatike. Novo razvita metodologija bo omogočila časovno daljše simulacije večjih molekularnih sistemov, kar bo znatno povečalo učinkovitost simulacije molekulske dinamike za študij strukturnih, dinamičnih in termodinamičnih lastnosti mehke snovi in molekularnih tekočin.

V drugem delu doktorske disertacije obravnavamo encimsko kinetiko, kjer z uporabo računalnika izvajamo računsko zahtevne matematične algoritme ter rešujemo zapletene sisteme diferencialnih enačb za študij encimske kinetike. V encimski kinetiki pogosto poenostavimo model tako, da uporabimo in analiziramo samo začetne hitrosti namesto celotnega časovnega poteka reakcije. S tem zanemarimo veliko število podatkov [31]. Poleg tega ni možno izpeljati analitične rešitve sistema diferencialnih enačb niti za enostavne Michaelis-Mentenove [32] reakcijske mehanizme, kaj šele za sisteme, ki vključujejo inhibitorje ali pa za sisteme, ki so alosterično regulirani. Obstajajo različne numerične metode za reševanje takšnih enačb ter vzpostavljanje časovnega poteka vsake od sodelujočih vrst. Metode morajo biti skrbno izbrane in implementirane. Za obravnavo reakcijskih mehanizmov v bioloških sistemih moramo namreč numerično metodo za reševanje povezati z algoritmom za analizo regresije [33]. Računalniških orodij, ki izpolnjujejo te pogoje ni veliko in obstoječa niso uporabniku prijazna. Najslabša stran takih orodij je priprava diferencialnih enačb, ki so specifične za vsak preučevani sistem posebej. Postopek je zamuden in ga moramo ponoviti za vsak nov reakcijski mehanizem. Pri obravnavi encimskih kinetičnih modelov so zato učinkoviti in izpiljeni uporabniški vmesniki ključnega pomena. Zato bomo zgradili orodje z uporabniku prijaznim grafičnim vmesnikom, ki bo enostavno za uporabo ter računsko učinkovito.

V naslednjem poglavju najprej na kratko v splošnem predstavimo molekularno modeliranje. V tretjem poglavju podrobneje predstavimo vejo molekularnega modeliranja in

sicer simulacijo molekulske dinamike. Nato predstavimo programski paket za simulacijo molekulske dinamike ESPResSo++. Sledi predstavitev metode AdResS ter opis vzporedne implementacije v omenjeni programski paket. Na koncu tretjega poglavja opišemo še pripravo simulacije raztopine soli v večskalni tehniki z uporabo na novo vgrajene metode. V četrtem poglavju predstavimo orodje z uporabniku prijaznim vmesnikom za analizo kinetike encimov. V petem poglavju sledi diskusija in nato v šestem zaključek.

Namen, cilji in hipoteze

V okviru doktorskega dela bomo uporabili računalniške tehnologije in pristope za razvoj novih orodij na področju molekularnega modeliranja. Z novimi orodji bomo simulirali tako biološko kot računalniško zanimive sisteme.

Postavljamo naslednje cilje:

- Implementacija večskalne metode za simulacijo mehke snovi: vgradnja metode prilagodljive ločljivosti v programski paket za molekularno modeliranje. Metoda naj omogoča vzporedno izvajanje na več procesorskih enotah.
- Simulacije biološko zanimivih sistemov z uporabo nove vzporedne večskalne metode.
- Razvoj učinkovitih računalniških vmesnikov za potrebe molekularnega modeliranja. Narava kinetičnih podatkov v encimatiki je taka, da njihova interpretacija ni enolična, zaradi česar pravega izmed konkurenčnih reakcijskih mehanizmov ni mogoče z gotovostjo izbrati. Izdelali bomo uporabniški vmesnik s katerim bo možno hitro diskriminirati med mnogimi reakcijskimi shemami in določiti reakcijski mehanizem s pripadajočimi kinetičnimi parametri.
- Aplikacije bodo dostopne širokemu krogu uporabnikov kot prosto dostopni računalniški programi ali storitve.

Postavljamo naslednje hipoteze:

- Splošno uporaben pristop večskalnega modela soli v vodi bo omogočil učinkovito obravnavo bioloških sistemov.
- Z novo razvitim vmesnikom za obravnavo encimske kinetike bomo opravili študijo titracije aktivnega mesta encima butirilholinesteraze iz organizma Torpedo californicae, obravnavo avtoaktivacije prokatepsina B ter reakcijo holinesterazebutiriltioholinom.
- Zaradi prostega dostopa, bodo novo razvita orodja uporabna široki znanstveni javnosti.

Metode raziskovanja

V okviru raziskovanja bomo razvijali programsko opremo in orodja za potrebe molekularnega modeliranja. Uporabljali bomo različne programske jezike (C++, C, Python, PHP, bash, AWK, ...), novo razvita orodja ter funkcije operacijskega sistema, v katerem bodo

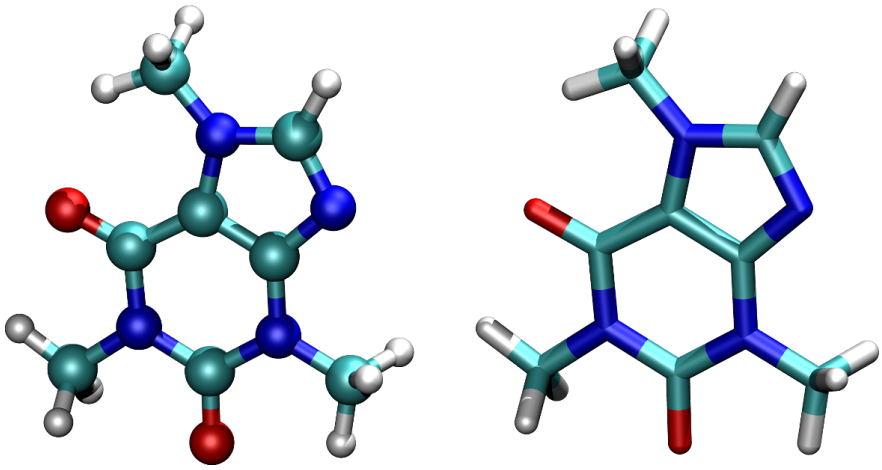
orodja delovala. Za numerično reševanje enačb gibanja delcev v simulaciji molekulske dinamike bomo uporabili Verletov integracijski algoritem. Pri sklapljanju grobozrnate in atomistične sile v večskalni simulaciji bomo uporabili metodo AdResS. Pravilnost delovanja večskalne simulacije raztopine soli bomo preverili z izračunom radialne porazdelitvene funkcije, normaliziranim profilom gostote in s povprečnim kvadratnim odmikom. Diferencialne enačbe encimskih kinetičnih reakcij bomo reševali z metodo najmanjših kvadratov.

2

Molekularno modeliranje

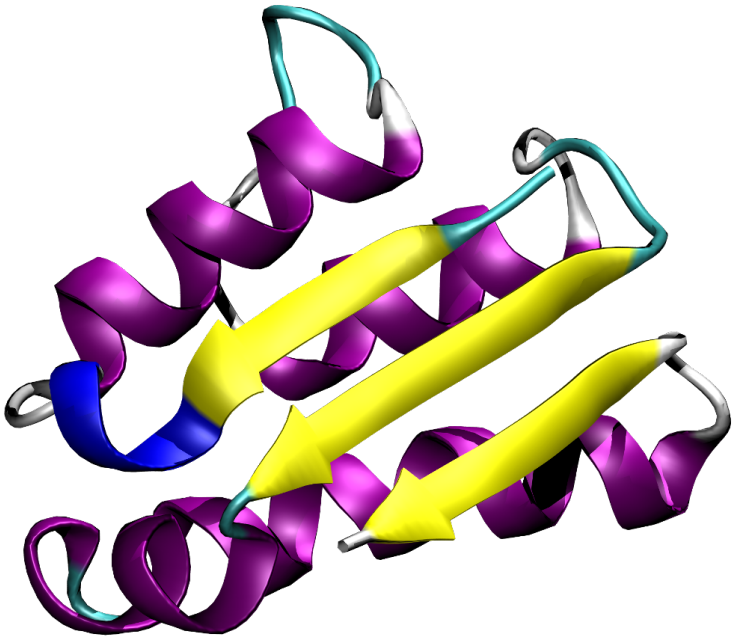
Molekularno modeliranje se ukvarja z računanjem strukturnih, dinamičnih in termodinamičnih lastnosti molekul in molekularnih sistemov. Kljub temu, da lahko res enostavne primere rešimo z mehanskimi modeli ali svinčnikom in papirjem, je molekularno modeliranje nedvomno povezano z računalniškim modeliranjem. Računalniki so povzročili pravo revolucijo na področju modeliranja do te mere, da večino dandanašnjih izračunov brez računalnika sploh ne bi mogli opraviti. To nujno ne pomeni, da so kompleksni modeli boljši od enostavnih, so pa računalniki zagotovo povečali nabor modelov, ki jih lahko obravnavamo in sistemov, na katerih jih lahko uporabimo [1].

Modele molekul najpogosteje prikažemo v obliki palic, ali kot kroglice povezane s palicami, kot so si jih zamislili Corey, Pauling in Koltun (slika 2.1). Temu modelu pogosto rečemu kar model CPK [34]. S takimi modeli prikažemo tridimenzionalno strukturo molekul, pri čemer je pomembno izpostaviti, da so interaktivni. To omogoča uporabniku, da si zastavlja vprašanja v povezavi z njimi. Strukturni modeli igrajo pomembno vlogo v poučevanju in v raziskovanju, uporablja pa se jih tudi za bolj abstraktne modele, npr. v kvantni mehaniki [1].



Slika 2.1: Modela molekul. Na levi strani je molekula prikazana kot model CPK, na desni pa ista molekula v obliki palic.

Molekularno modeliranje je dosti več kot samo prikazovanje molekul na ekranu. Interakcija med molekularno grafiko in pripadajočimi teoretičnimi metodami je izboljšala dostopnost metod molekularnega modeliranja in pripomoglo k analizi ter interpretaciji takih izračunov. Obstajajo različni načini prikazovanja molekul na ekranu, npr. barvanje



Slika 2.2: Sekundarna struktura beljakovine. Slika prikazuje beljakovino z oznako 1AY7_B [35]. Če bi beljakovino izrisali s posameznimi atomi, bi bila slika manj pregledna.

molekul glede na atomsko število, učinki sence in svetlobe ter perspektive, ki pripomorejo k realističnemu prikazu. V primerjavi z mehanskimi modeli imajo nekaj prednosti. Računalniški model lahko zlahka poda kvantitativno informacijo, od preprostih geometrijskih meritev kot npr. razdalja med dvema atomoma, do bolj kompleksnih kot npr. vrednost energije ali velikost površine. Iz mehanskih modelov težko dobimo take podatke, kljub temu pa so v določenih situacijah bolj priljubljeni, saj so intuitivnejši za uporabo in lažji za manipuliranje v treh dimenzijah (oz. v prostoru). V zadnjih nekaj letih se oba pristopa združuje z uporabo *obogatene resničnosti*, ki omogoča, da v živo rokujemo z enostavnimi predmeti (ali zgolj z referenčnimi točkami) računalnik pa jih na ekranu prikazuje obogatene [36,37]. Kakorkoli že, tudi najenostavnejši računalniški programi nudijo nekaj standardnih načinov manipulacije modelov, kot so rotacija, translacija ter povečava in pomanjšava (zoom). V primeru, da imamo opravka z velikimi molekulami, ponavadi ne želimo na ekranu prikazati prav vsakega atoma posebej, saj bi sicer bila slika nejasna in nepregledna. Z izpuščanjem določenih atomov ali s prikazom skupine atomov kot en sam delec, dobimo jasnejšo sliko. Dober primer je prikazovanje beljakovin, kjer lahko že ena majhna beljakovina vsebuje nekaj tisoč atomov. Zaradi tega jih običajno ne prikazujemo eksplicitno, ampak z uporabo sekundarne strukture (slika 2.2), kjer trakovi in vijačnice predstavljajo sekundarno strukturo beljakovine (tj. vijačnice α in trakove β) [1].

Večina študij v molekularnem modeliranju zajema tri faze. V prvi fazi izberemo model, s katerim opišemo intra- in intermolekularne interakcije v sistemu. Pogosto so takšni modeli osnovani na kvantni ali klasični mehaniki. Ta modela omogočata izračun energije za poljubno postavitev atomov ali molekul v sistemu. Raziskovalec lahko raziskuje, kako sprememba položajev atomov in molekul vpliva na energijo. Naslednja faza študija je izračun sam, kot npr. minimizacija energije, simulacija molekulske dinamike, simulacija

Monte Carlo [1] ali iskanje konformacije. V zadnji fazi analiziramo izračun, da dobimo želene lastnosti in pa tudi, da preverimo ali je bil prav izračunan [1].

Metode, ki jih uporabljamo v okviru molekularnega modeliranja zajemajo molekularno mehaniko, minimizacijo, simulacije, kvantno mehaniko, analizo konformacij in druge računske metode za razumevanje in napovedovanje obnašanja molekularnih sistemov. Kakršnakoli teoretična ali računska metoda, ki omogoča vpogled v obnašanje molekulskih sistemov, je primer molekularnega modeliranja. Poudarek je zlasti na prikazu in manipulaciji struktur molekul ter lastnostih, ki so odvisne od razporeditve v tridimenzionalnem prostoru. Računalniška grafika pri tem igra pomembno vlogo, saj skrbi za izris na ekranu [1].

3

Simulacija molekulske dinamike

Simulacija molekulske dinamike sestoji iz numeričnega reševanja klasičnih enačb gibanja za vse delce v sistemu. Izhodišče je dobro definiran mikroskopski opis fizikalnega sistema. Gibanje delcev opišemo z Newtonovimi enačbami gibanja [2, 3]:

$$m_i\ddot{\vec{r}_i} = \vec{f}_i \qquad \vec{f}_i = -\frac{\partial}{\partial r_i}\mathcal{U}$$

kjer je $\vec{f}_i$ sila, ki deluje na delec i v sistemu z maso m_i in radij vektorjem $\ddot{\vec{r}_i}$. Če je sila konservativna [38], jo izračunamo kot negativni gradient potencialne energije $\mathcal{U}$. Enačbe rešujemo numerično z uporabo računalnika, da dobimo statične in dinamične lastnosti sistema.

Obstaja več algoritmov za integriranje enačb gibanja delcev. Razlikujejo se v hitrosti, natančnosti, stabilnosti, količini potrebnega pomnilnika, zahtevnosti za programiranje, velikost časovnega koraka, ipd. Ideja integracijskih algoritmov (integratorjev) je, da časovni potek sistema razdelimo v serijo časovnih korakov, nato podatke iz prejšnjih korakov uporabimo za generiranje podatkov v prihodnosti.

Verletov integracijski algoritem

V programih za simulacijo molekulske dinamike se najpogosteje pojavlja Verletov integracijski algoritem ali ena od njegovih izpeljank. Zanj je značilno, da je simplektičen (ohranja volumen faznega prostora [38]) in da je časovno obrnljiv [4–6]. Enačba za izračun položaja v času $t + \delta t$ je sledeča [2]:

$$\vec{r}_i(t + \delta t) = 2\vec{r}_i(t) - \vec{r}_i(t - \delta t) + \delta t^2\vec{a}_i(t)$$

kjer je δt časovni korak in $\vec{a}_i = \frac{\vec{f}_i}{m_i}$ oz. pospešek delca. Čeprav za Verletov algoritem podatka o hitrosti delcev ne potrebujemo, ga ponavadi potrebujemo za izračun kinetične energije (temperature) sistema. Hitrost izračunamo iz razlike položaja delca v dveh časovnih korakih [2]:

$$\vec{v}_i(t) = \frac{\vec{r}_i(t + \delta t) - \vec{r}_i(t - \delta t)}{2\delta t}.$$

Enačba je sicer enostavna, vendar nam vrne rezultat z napako reda δt^2 [2]. To pomeni, da za natančen izračun potrebujemo kratek časovni korak. Krajši kot je korak, več jih potrebujemo, da pokrijemo želeno časovno skalo. Poleg tega moramo v pomnilniku hraniti informacijo o prejšnjem položaju delca. Hitrost v času t lahko izračunamo šele, ko je znan

položaj delcev v času $t + \delta t$, kar pomeni, da nam bo podatek o hitrosti zaostajal za en korak.

Obstaja izpeljanka Verletovega algoritma imenovana hitrostni Verlet, ki izračuna položaje, hitrosti in pospeške istočasno v času t, ter s tem tudi zmanjša napako zaradi zaokroževanja. Enačbi za položaj in hitrost sta [2]:

$$\vec{r}_i(t+\delta t) = \vec{r}_i(t) + \delta t \vec{v}_i(t) + \frac{1}{2}\delta t^2 \vec{a}(t)$$
$$\vec{v}_i(t+\delta t) = \vec{v}_i(t) + \frac{1}{2}\delta t\big[\vec{a}_i(t) + \vec{a}_i(t+\delta t)\big].$$

Algoritem hrani le $\vec{r}$, $\vec{v}$ in $\vec{a}$, ter poteka v dveh fazah, vmes pa se izvede izračun sil. Enačbe lahko preoblikujemo v sledečo obliko [39]:

$$\vec{v}_i(t+\frac{1}{2}\delta t) = \vec{v}_i(t) + \frac{1}{2}\delta t \vec{f}_i(t)$$
$$\vec{r}_i(t+\delta t) = \vec{r}_i(t) + \delta t \vec{v}_i(t+\frac{1}{2}\delta t)/m_i$$
$$\vec{v}_i(t+\delta t) = \vec{p}_i(t+\frac{1}{2}\delta t) + \frac{1}{2}\delta t \vec{f}_i(t+\delta t)$$

kar nam omogoča, da algoritem neposredno pretvorimo v računalniško psevdokodo [39]:

psevdokoda

```
1 for (1 ... st_korakov):
2     v = v + 0.5*dt*f
3     r = r + dt*v/m
4     f = sila(r)
5     v = v + 0.5*dt*f
```

kjer je `st_korakov` število integracijskih korakov simulacije, `v` je hitrost delca, `dt` je časovni korak, `f` je sila, ki deluje na delec, `r` je položaj delca v prostoru, `m` je masa delca, `sila()` pa funkcija, ki izračuna silo na delec glede na njegov položaj. Začetne hitrosti in položaji delcev so znani (določeni). V vsaki iteraciji zanke izračunamo nove hitrosti in položaje delcev.

Sile med delci

Za izračun Newtonovih enačb gibanja moramo poznati sile, ki delujejo na delce. Sile izračunamo iz potenciala. Le-tega lahko zapišemo kot člene, ki so odvisni od koordinat posameznih delcev, parov delcev, trojic delcev, itd. [2]:

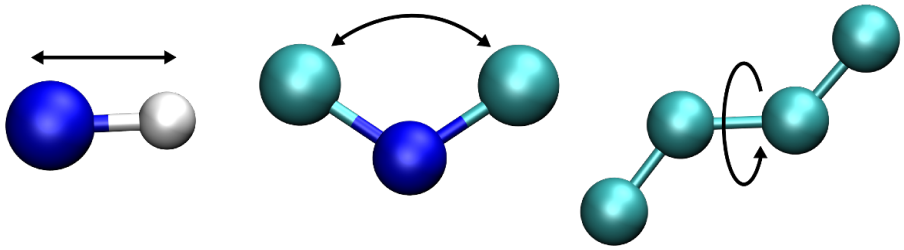
$$\mathcal{U} = \sum_i u_1(\vec{r}_i) + \sum_i \sum_{j>i} u_2(\vec{r}_i, \vec{r}_j) + \sum_i \sum_{j>i} \sum_{k>j>i} u_3(\vec{r}_i, \vec{r}_j, \vec{r}_k) + \ldots$$

Prvi člen v enačbi $u_1(\vec{r}_i)$ predstavlja učinek zunanjih sil (npr. električno polje, stene, ipd.) na sistem. Ostali členi predstavljajo interakcije med delci. Člen u_2 je parski potencial in je najbolj pomemben. Odvisen je samo od razdalje med delcema $r_{ij} = |\vec{r}_i - \vec{r}_j|$, zato ga pogosto zapišemo kar kot $u_2(r_{ij})$. Člen u_3 predstavlja potencial trojice delcev, vendar jih v računalniških simulacijah zaradi računske zahtevnosti skoraj nikoli ne računamo.

Namesto tega raje uporabimo parski potencial, ki že vsebuje približek učinka treh teles. Prispevek členov višjega reda je minimalen in ga zanemarimo.

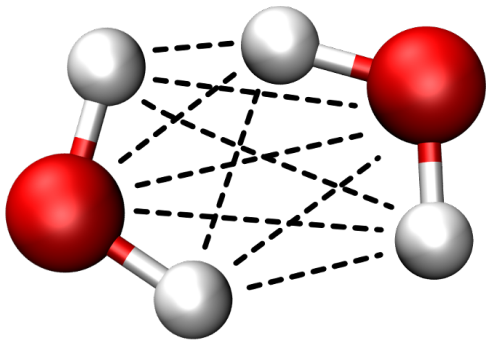
Za opis parskih neelektrostatskih interakcij delcev se pogosto uporablja Lennard-Jones potencial $u_{LJ}(r) = 4\epsilon\left((\sigma/r)^{12} - (\sigma/r)^{6}\right)$, kjer je r razdalja med dvema delcema, ϵ je negativna vrednost potenciala v minimumu in σ razdalja, pri kateri je potencial enak nič [2]. Parametri potencialov so definirani s *poljem sil* (npr. CHARMM [40–42], AMBER [43], GROMOS [44]).

Interakcije nastopajo kot vezne in nevezne. Vezne interakcije delujejo znotraj molekul kot posledica kovalentnih vezi: raztezanje in krčenje vezi, upogibanje vezi in torzija (slika 3.1). Primer vezne interakcije za raztezanje in krčenje je FENE (finite extensible nonlinear elastic) potencial $u_{FENE}(r) = -1/2kR_0^2 ln[1 - (r/R_0)^2]$ za $r \leq R_0$, oz. ∞ za $r > R_0$, pri čemer je r razdalja med dvema sosednjima delcema znotraj iste molekule, R_0 in k pa sta divergenčna razdalja in togost. V programih za simulacijo molekulske dinamike vezne interakcije ponavadi podamo kot fiksen seznam delcev.



Slika 3.1: Vezne interakcije. Raztezanje ter krčenje vezi levo, upogibanje na sredini in torzija desno.

Nevezne interakcije delujejo med molekulami kot npr. sile van der Waals (slika 3.2) in elektrostatske sile v primeru prisotnosti nabitih delcev. Za izračun neveznih sil izračunamo razdalje med vsemi pari delcev v sistemu ter izračunamo silo glede na medsebojno razdaljo med delcema [2].



Slika 3.2: Nevezne interakcije. Črtkane črte nakazujejo nevezne sile med delci dveh molekul.

V računalniški psevdokodi bi izračun nevezne sile na delce lahko zapisali kot:

psevdokoda

```
1 for (i ... N):
2     for (j ... N):
3         i.f = i.f + sila(i, j)
```

kjer je `N` število delcev v sistemu, `i.f` je sila, ki deluje na delec `i`, `sila()` pa funkcija, ki izračuna silo med delcema `i` in `j` glede na njuno razdaljo. Ob zaključku iteriranja obeh `for` zank, imamo za vse delce izračunane sile, ki delujejo na njih. Zaradi dvojne zanke je zahtevnost izračuna sil reda n^2. Največ časa simulacije se porabi prav za izračun sile.

Simulacijska škatla in periodični robni pogoji

Delce, ki jih simuliramo, moramo na nek način zadržati v sistemu [2]. To lahko storimo npr. tako, da sistem obdamo s stenami in s tem preprečimo uhajanje delcev. V tem primeru bi se delci odbijali od sten, kar bi uvedlo dodatne sile in s tem vplivalo na lastnosti sistema. Tega ponavadi ne želimo (razen v primeru, da preučujemo vpliv površine). Zato se zatečemo k drugačni rešitvi: simulacijsko škatlo obdamo s slikami same sebe (slika 3.3, škatla je siva, slike so bele). Na robovih škatle uvedemo ti. periodične robne pogoje. V trenutku, ko delec zapusti prostor škatle, ponovno vstopi vanjo na nasprotni strani (lahko si predstavljamo, da je vstopila njegova slika). Škatla je pogosto v obliki kvadra, vendar so možne tudi drugačne oblike in sicer take, s katerimi lahko zapolnimo prostor. Na tak način lahko z razmeroma majhno škatlo simuliramo velike (neskončne) sisteme, kljub temu pa se moramo zavedati, da periodičnost lahko vpliva na nekatere lastnosti dolgega dosega.

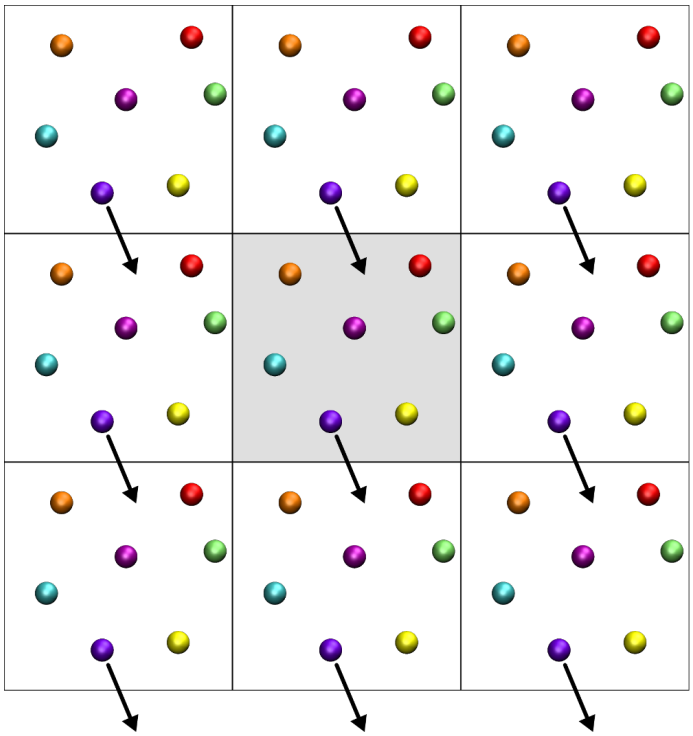
Skupaj s periodičnimi robnimi pogoji uporabimo tudi ti. konvencijo minimalne slike [2]. Ta zagotavlja, da v kolikor je nekemu delcu slika drugega delca bliže kot delec, bomo v izračunu sile uporabili njegovo sliko.

Pohitritev simulacij

Število veznih interakcij narašča linearno s številom delcev, število neveznih interakcij pa narašča kvadratično. To pomeni, da se bo tekom simulacije operacija za izračun sil neveznih interakcij največkrat izvedla. Čas potreben za izračun sil posledično narekuje hitrost izvajanja programa za simulacijo molekulske dinamike. Zato je pomembno, da kodi za izračun sile namenimo veliko pozornosti in da je čimbolj učinkovito napisana. V nadaljevanju predstavimo nekaj običajnih ter možnih prijemov za pohitritev simulacij.

Tretji Newtonov zakon

Zaradi zakona o vzajemnem učinku ni potrebno računati sile med dvema delcema za oba delca, kakor smo zapisali zgoraj. Dovolj je, če jo izračunamo za en delec, drugemu pa dodelimo nasprotno enako silo. Notranji zanki, ki gre skozi delce zato ni potrebno iti skozi vse delce:



Slika 3.3: Periodični robni pogoji. Osnovna simulacijska škatla (prikazana s sivim ozadjem) je obdana s slikami same sebe (škatle z belim ozadjem). Ko npr. vijolični delec zapusti škatlo na spodnji strani, zgoraj vstopi njegova slika. Delci se lahko prosto gibljejo preko mej škatle, vendar se njihovo število v osnovni škatli ne spreminja. Zaradi preglednosti gornja slika prikazuje periodične robne pogoje zgolj v dveh dimenzijah.

psevdokoda

```
for (i ... N-1):
    for (j=i+1 ... N):
        f = sila(i, j)
        i.f = i.f + f
        j.f = j.f - f
```

kjer je `N` je število delcev, `f` je sila med delcema `i` in `j`, `i.f` je sila, ki jo občuti delec `i` in `j.f` sila, ki jo občuti delec `j`. V primerjavi s prej zapisano psevdokodo, tokrat v notranji `for` zanki ne iteriramo skozi vse delce, ampak le skozi delce, ki v seznamu delcev sledijo `i`-ju.

Odrez potenciala

Doseg sil van der Waals je omejen, zaradi tega nam v primeru, da sta dva delca dlje od določene razdalje, ni potrebno računati sil med njima. Razdalji, preko katere sil ne računamo, rečemo odrez potenciala. S psevdokodo izračun sile z upoštevanjem odreza potenciala zapišemo kot:

psevdokoda

```
for (i ... N-1):
    for (j=i+1 ... N):
        if (razdalja(i,j) > odrez_potenciala) continue
        f = sila(i, j)
        i.f = i.f + f
        j.f = j.f - f
```

kjer je `razdalja(i,j)` razdalja med delcema `i` in `j`. V primeru, da je razdalja večja od odreza potenciala, zanka nadaljuje z iteriranjem, ne da bi izvedla vrstico s klicem funkcije `sila()`. Z odrezom potenciala tudi preprečimo, da bi delec čutil silo svoje lastne slike.

Verletov seznam delcev

Ceno iteriranja dvojne zanke lahko omilimo, če upoštevamo dejstvo, da delci čutijo silo van der Waals le v bližnji okolici. Za vsak delec zato zgradimo seznam sosedov, t. j. delcev v dosegu odreza potenciala, silo pa nato računamo samo na sosedih posameznega delca. Koda za izračun sile je sedaj takšna:

psevdokoda

```
for (i ... N):
    for (sosedi od i):
        sila(i, sosed)
```

kjer je `N` število delcev, `sosedi` je seznam delcev v bližini izbranega delca `i`, `sosed` pa trenutno izbrani delec iz seznama. Idejo je predlagal Verlet in po njem se imenuje Verletov seznam [2].

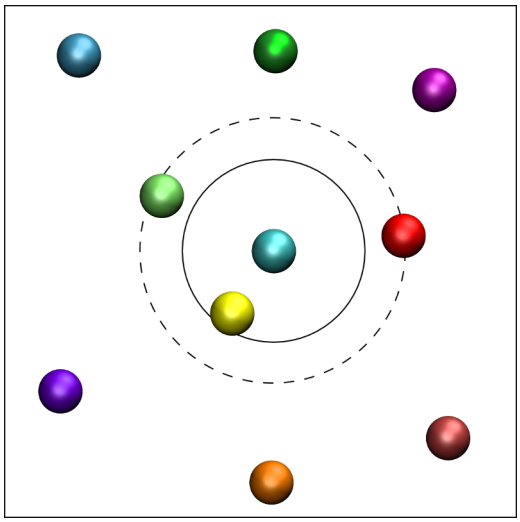
Ker se delci ves čas gibljejo, je potrebno seznam sosedov redno posodabljati. Da se izognemu posodabljanju v vsakem koraku, pri gradnji seznama vključimo tudi delce, ki so nekoliko dlje od odreza potenciala (slika 3.4). Tej dodatni razdalji rečemo koža (*angl.* skin) [2]. Če vodimo evidenco prepotovane razdalje delcev, lahko sprožimo posodobitev seznama, ko nek delec prepotuje določeno razdaljo. V nasprotnem primeru pa sicer seznam posodobimo vsakih nekaj korakov.

Razdelitev simulacijske škatle v celice

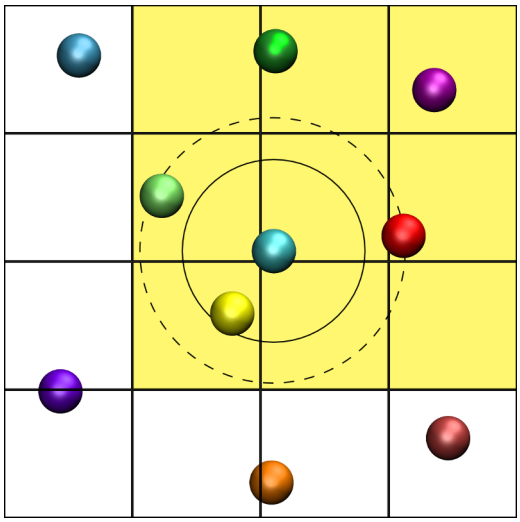
Iteriranju po delcih v dvojni zanki se na vsak način želimo izogniti. Z uporabo zgolj Verletovega seznama se temu ne izognemo v celoti, saj iteriramo po delcih, ko gradimo seznam. Zato simulacijsko škatlo razdelimo v celice, vsaka celica pa hrani seznam delcev v njej [2]. Ko sedaj gradimo Verletov seznam, se nam ni potrebno sprehoditi skozi vse delce. Dovolj je, če gremo skozi delce trenutne celice in delce sosednjih celic (slika 3.5).

Vzporedno računanje

Z uporabo vzporednega računanja problem simuliranja razdelimo na podprobleme tako, da simulacijsko škatlo razdelimo v podškatle in jih dodelimo ločenim procesorjem (ali procesorskim jedrom). Procesorji vzporedno izvajajo simulacijo za delce, ki jim pripadajo [45, 46]. Število delcev na procesor se zato zmanjša. Po vsakem integracijskem koraku je prisoten še komunikacijski korak, v katerem si procesorji izmenjajo izračunane podatke



Slika 3.4: Verletov seznam delcev. V Verletov seznam svetlomodrega delca poleg rumenega vstavimo tudi zelen in rdeč delec. Ta dva sta sicer dlje od odreza potenciala, ki je označen s krogom s polno črto. Črtkan krog predstavlja razdaljo vključno s kožo.

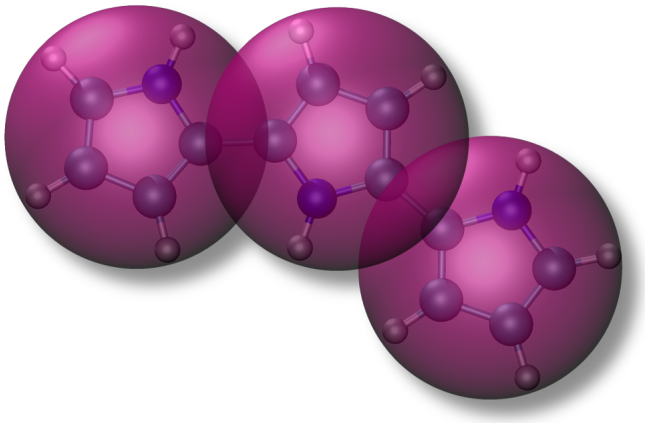


Slika 3.5: Razdelitev simulacijske škatle v celice. Pri gradnji Verletovega seznama modrega delca, iščemo sosede le v njegovi celici in v sosednjih celicah. Na sliki so celice v katerih iščemo sosede obarvane z rumeno barvo.

(ali del njih). Ker je komunikacijski korak bistveno počasnejši od ostalih korakov, ki jih izvaja procesor, je vzporedno računanje potrebno skrbno načrtovati [47].

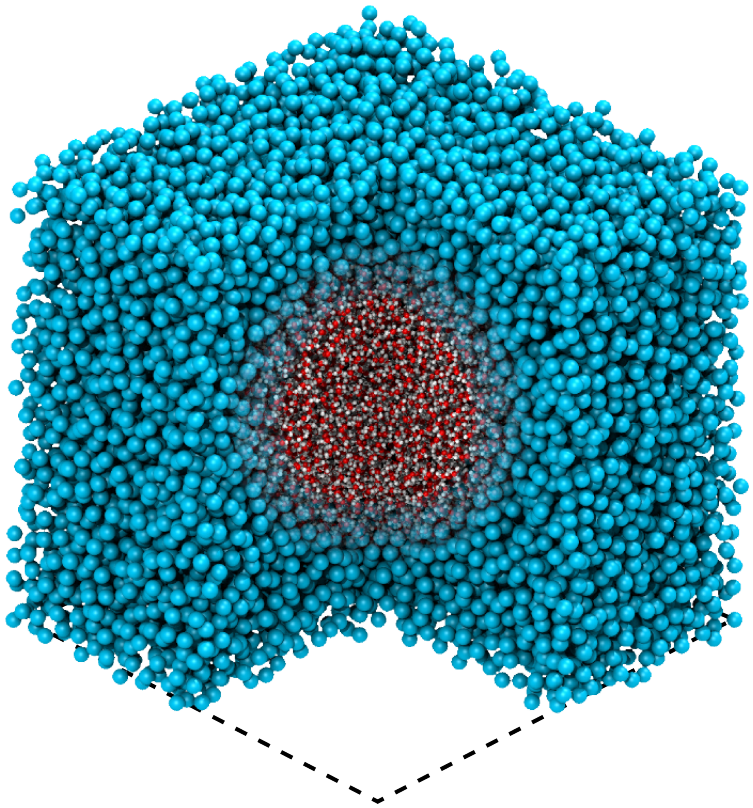
Grobozrnenje

Pri grobozrnenju (*angl.* coarse-graining) skupek atomov združimo v „večjo“ enoto, s čimer zmanjšamo število delcev in prostostnih stopenj ter posledično zmanjšamo računsko zahtevnost. Molekulo ali del molekule predstavimo kot en sam delec (slika 3.6). Grobozrnenje omogoča, da pokrijemo večje prostorske in daljše časovne skale. Slaba stran je, da s takim postopkom zmanjšamo natančnost simulacije. Grobozrnenje ponavadi uporabljamo



Slika 3.6: Grobozrnenje. Vijoličasti delci predstavljajo grobozrnate delce, ki jih uporabimo namesto devetih (oz. osmih) atomističnih delcev. Atomistični delci so na sliki narisani znotraj grobozrnatih.

pri simuliranju večjih sistemov, za katere je atomistična simulacija prepočasna, da bi bila uporabna [25].



Slika 3.7: Metoda prilagodljive ločljivosti. Simulacijska škatla z uporabo metode prilagodljive ločljivosti AdResS. Središče škatle simuliramo z atomsko ločljivostjo, okolico pa z grobozrnato ločljivostjo. Prednji vogal škatle je izrezan, da vidimo v notranjost.

Metoda prilagodljive ločljivosti

Nekatera območja simuliranja nas zanimajo bolj kot druga. Na primer makromolekule, ki jih simuliramo, so obdane s topilom, za katerega potrebujemo visoko ločljivost le v neposredni okolici makromolekule. Za manj pomembna območja ne želimo izgubljati procesorske moči in zadostuje, če uporabimo grobozrnato ločljivost (slika 3.7). Metoda prilagodljive ločljivosti AdResS [26, 27] omogoča, da združimo atomistični in grobozrnati opis, ter pri tem dopuščamo možnost prostega prehoda delcev med obema sistemoma.

V poglavju 3.2 podrobneje predstavimo delovanje metode prilagodljive ločljivosti AdResS in njeno implementacijo v programski paket ESPResSo++.

3.1 Predstavitev programa za simulacijo molekulske dinamike ESPResSo++

Že v prejšnjem poglavju smo izpostavili zahtevnost izračuna sil. Opravka imamo namreč s problemom reda n^2. Število delcev (n) v sistemu je tipično več deset tisoč, simulacije pa lahko tečejo tudi po nekaj mesecev. Zato že majhna površnost pri izvedbi izračuna sile naredi program za simulacijo molekulske dinamike popolnoma neuporaben. Vzporedno računanje sil na več procesorskih jedrih, vključno s poganjanjem na računalniških gručah, je nujen pogoj za učinkovito izvedbo simulacije.

Glede na potrebe simulacije in razpoložljive računalniške vire uporabljamo različne programske pakete za simulacijo molekulske dinamike, kot so GROMACS [48], LAMMPS [49], NAMD [50], Amber [51], DL_POLY [52], CHARMM [53], itd. Za učinkovit izračun sil uporabljajo različne metode za paralelizacijo (npr. domenska dekompozicija, atomska dekompozicija, dekompozicija osmine školjke) in knjižice (npr. MPI [54]). Metoda AdResS omogoča nadaljnjo metodološko pospešitev izvajanja simulacije, ker z uporabo grobozrnenja zmanjšamo število prostostnih stopenj in število delcev, ki nastopajo v izračunu sile. Vgradnja metode zahteva spremembo v delih kode, ki je zadolžena za izračun sile, saj sklopitev atomistične in grobozrnate sile narekuje spremenjeno shemo izračuna sil. Obstoječi programski paketi za simulacijo molekulske dinamike nimajo dovolj prožne zasnove, ki bi omogočala uporabno vgradnjo vzporedne metode prilagodljive ločljivosti AdResS. Metodo bi sicer lahko vgradili, vendar bi se prepočasi izvajala. Zahtevala bi tudi spremembo večjega dela programa oziroma znatno redundanco kode (npr. tako, da bi izdelali novo kopijo programa le, da bi namesto navadnega izračuna sil uporabili AdResS – želimo pa imeti oboje). Izhodišče izdelave novega programa ESPResSo++ je zato bila zasnova, ki omogoča učinkovito izvedbo vzporedne različice metode AdResS. Ker metoda posega v samo jedro simulacij molekulske dinamike, tj. izračun sil med delci, mora biti metoda skrbno načrtovana in previdno izvedena. Z uporabo tehnik objektno usmerjenega programiranja v kombinaciji s predlogami, dosežemo visoko stopnjo modularnosti, pri čemer ne izgubimo učinkovitosti izvajanja kode.

ESPResSo++ [29] razvijamo v sodelovanju z Max Planck Institute for Polymer Research v Nemčiji [55]. Napisan je v programskem jeziku C++ z razširitvami Python. Ker je pri postavitvah in nastavitvah parametrov za molekulsko dinamiko zmeraj potrebnega nekaj programiranja (npr., če je potrebno vpisati koordinate pol milijona delcev, jih prav gotovo ne bomo pisali na roko), ESPResSo++ združuje kombinacijo dveh programskih jezikov. Del programa, ki je izpostavljen uporabniku, je napisan v skriptnem jeziku Python, ki velja za enega uporabniku najbolj prijaznih programskih jezikov. Del ESPResSa++, ki je zadolžen za računanje, pa je napisan v C++, ker je bistveno hitrejši od Pythona [56,57].

V sklopu disertacije smo v ESPResSo++ vgradili vzporedno metodo AdResS [29]. Atomistične delce smo implementirali kot dodatno lastnost grobozrnatih delcev. Za izračun sile zgradimo tri Verletove sezname in sicer za vsako od območij (atomistično, grobozrnato, hibridno) svojega. Podrobnejši opis implementacije je v poglavju 3.2. Šele naša učinkovita izvedba nam omogoči simulacijo velikih večskalnih sistemov, kot npr. simulacija raztopine soli, ki jo predstavimo v poglavju 3.3. Sledi podrobnejši opis programa ESPResSo++.

Osnovna vodila pri izdelavi programa

Najpomembnejše vodilo pri izdelavi programa je razširljivost oz. možnost enostavnega razširjanja jedra C++ ter prav tako uporabniškega vmesnika Python. Poleg poganjanja običajnih simulacij, je ESPResSo++ zamišljen kot peskovnik za preizkušanje novih algoritmov in metod. Struktura programa je narejena s podporo za obstoječe algoritme, poleg tega pa tudi za algoritme, ki še niso bili zasnovani. Pri razvoju programske opreme v splošnem velja, da je objektno usmerjeno programiranje povsem primerno za ta namen [58, 59]. ESPResSo++ je zato napisan v programskem jeziku C++, s čimer smo pri njegovi zasnovi lahko izkoristili objetno usmerjene tehnike programiranja. Izbira programskega jezika je izražena tudi v imenu programskega paketa.

Naslednje vodilo je hitrost. To pomeni, da če si bosta hitrost in razširljivost nasprotovali, bo imela razširljivost prednost. Kljub temu lahko pričakujemo, da bo ESPResSo++ uporabljan za poganjanje simulacij, ki bodo tekle po več sto ur na dragih računalnikih visoke zmogljivosti. Zato je ključnega pomena, da je programska oprema dobro optimizirana in pripravljena za vzporedno računanje. ESPResSo++ je optimiziran za učinkovit dostop do pomnilnika in hitre algoritme ter za poganjanje na večjem številu procesorjev. Čeprav so današnji interpretirani programski jeziki [60] ali jeziki prevedeni v kodo zlogov (*angl.* bytecode) [61] (kot npr. Java [62]) računsko učinkoviti na več področjih računanja [63–67], se bo za vzporedne visoko zmogljive numerične aplikacije bistveno hitreje izvajal programski jezik, ki ga je potrebno prevesti (kot npr. C++).

Zaradi širokega razpona problemov, s katerimi se program ESPResSo++ ukvarja, in različnih zahtev po računalniški moči teh problemov, je pomembno, da lahko programski paket uporabljamo na različnih računalniških platformah, od običajnih delovnih postaj do visoko zmogljivih superračunalnikov. Zaradi čim boljše prenosljivosti ESPResSo++ ne uporablja knjižnic, ki niso na voljo ali pa so težko dostopne na ciljnih platformah. Na primer za paralelizacijo se uporablja široko dostopno knjižnico MPI-1/2 (message passing interface) [54], za programski jezik pa C++, ki je definiran s standardom ANSI/ISO ISO/IEC 14882:2003 [68] in je prenosljiv med vsemi pomembnejšimi platformami. Programski paket je oblikovan za operacijske sisteme, ki so v skladu s standardom POSIX [69].

Znanstveni problemi, s katerimi se soočamo pri uporabi programa, so raznoliki in prav tako so raznolike možne rešitve. Da omogočimo različne strategije za rokovanje s problemom, je program prožen glede načinov uporabe. ESPResSo [28], predhodnik programa ESPResSo++, uporablja skriptni jezik Tcl/Tk kot glavni uporabniški vmesnik. To se je izkazalo za uspešen pristop in zato v programu ESPResSo++ nadaljujemo na tak način, le z drugačnim jezikom. Uporabljamo namreč skriptni jezik Python, ki velja za najpopularnejši splošno namenski večparadigmatični skriptni jezik. Python se uporablja v mnogo projektih programske opreme, nudi številne znanstvene module in pakete, je prenosljiv, ima široko bazo uporabnikov in ga lahko povežemo s programskim jezikom C++. Povezavo naredimo s knjižnico *Boost.Python*.

Predpogoj, da lahko raziskovalci učinkovito delajo na novih aplikacijah in novih algoritmih s programom ESPResSo++, je, da je izvorna koda programa jasno strukturirana, da je razumljiva in dobro dokumentirana. To raziskovalcem tudi omogoča, da preverijo pravilnost programa. Berljivost olajša vzdrževanje, razširljivost in preverljivost programa.

Naslednje vodilo je robustnost. S tem je mišljeno, da program ne sme postati nestabilen, če so vhodni parametri rahlo izven fizikalno smiselnih mej ali pa če se okolje spremeni. ESPResSo++ je prav tako robusten proti spremembam parametrov med sa-

mim tekom programa, zato da lahko spreminjamo fizikalni opis po željah. Če program dovoljuje določeno kombinacijo parametrov tj., da dokumentacija tega ne izključuje, potem se s tako kombinacijo ne sme zrušiti. V primeru, da so nastavitve tehnično problematične, mora biti uporabnik opozorjen, v primeru resne napake pa mora biti izpisano smiselno sporočilo. Robustnost narekuje, da se izvede natančne in temeljite preizkuse vseh modulov programa.

ESPResSo++ je namenjen znanstvenim raziskavam in prav tako za industrijske potrebe. Program je licenciran pod GNU General Public License (GPL v2.0) [70]. Odločitev, da je ESPResSo++ prosto dostopen, zagotavlja, da bo zmeraj na voljo. To bo spodbujalo uporabniško skupnost, da razširja ESPResSo++ in pospeši razvoj novih znanstvenih raziskav in algoritmov. Na voljo je na naslovu http://www.espresso-pp.de/.

Uporabniški vmesnik

Programi za simulacijo molekulske dinamike so ponavadi bodisi v obliki glavnega programa, ki sprejme neke vhodne podatke in pripadajoče datoteke (npr. topologija sistema, začetne koordinate, parametri polja sil) ter ustvari neke izhodne podatke, bodisi kot knjižnica, ki jo lahko uporabimo v lastnem programu. Za ESPResSo++ uberemo tretjo pot, in sicer uporabimo skriptni jezik Python za izvajanje in nadzor nad pogonom za simulacije, pogon pa napišemo v programskem jeziku C++.

Tak pristop omogoča, da program uporabljamo na različne načine, dosti več kot nudijo zgolj samo vhodne datoteke brez uporabe skriptnega jezika. Poleg tega lahko uporabnik programa uporablja še vse ostale zmožnosti, ki jih nudi skriptni programski jezik. Konkretno omogoča, da uporabnik izvaja veliko opravil, ki so sicer zoprna, če jih mora izvajati z uporabo nekega zunanjega programa, kot recimo računanje spremenljivk iz nekih drugih opazovanih spremenljivk ali branje in pisanje podatkov v različnih oblikah. Nenazadnje simulacije, ki jih poganjamo preko skripte dokumentirajo „same sebe“ v smislu, da je postopek zapisan v skripti, enak postopku, preko katerega smo prišli do rezultatov.

Izbor Pythona je bil gnan z vodilom prožnosti, saj je Python sam zelo vsestranski in ker je objektno usmerjen, se povsem ujema z našim ciljem razširljivosti. Poleg tega je Python že zelo dobro uveljavljen skriptni jezik v raziskovalni skupnosti, kar odraža veliko število numeričnih in znanstvenih paketov za Python, kot npr. NumPy [71] in SciPy [72]. To sicer pomeni, da je potrebno osnovno znanje Pythona za učinkovito uporabo programa ESPResSo++, vendar to v resnici ni taka težava, saj Python velja za enega najenostavnejših programskih jezikov. Poleg tega obstaja zanj mnogo knjig in vodičev.

Čeprav je standardna knjižnica Python razširljiva, je veliko funkcionalnosti na voljo preko številnih dodatnih paketov napisanih iz strani raznih prostovoljcev ali skupnosti. Paketi so lahko kot zgolj Python ali pa kombinacija Pythona in prevedene kode, ki je tipično napisana v programskem jeziku C ali C++ ali Fortran. ESPResSo++ je primer zunanjega paketa, ki je kombinacija Pythona in C++. To pomeni, da lahko ESPResSo++ zlahka izkoristi prej omenjene Python pakete, kar vodi v tesno integriran potek dela od priprave sistema do analize. Python nudi prožen vhodno/izhodni koncept, ki ga lahko uporabijo tudi programi, ki niso zasnovani na Pythonu (npr. uporaba priključkov Unix za grafični izris v programu za vizualizacijo molekul VMD [73]).

Ker je simulacijsko jedro programa ESPResSo++ implementirano v programskem jeziku C++, omogoča uporabnikom poleg razširjanja ESPResSo++ na skriptni ravni, tudi razširjanje simulacijskega jedra samega, s pisanjem novih razredov C++. Knjižnica Bo-

ost.Python omogoča, da razrede C++ z minimalno truda izpostavimo Pythonu. Povezavo med veliko delovnimi jedri in enim jedrom, ki poganja skripto, nudi modul PMI (Parallel Method Invocation) [74]. PMI samodejno generira serijski vmesnik za vzporedne klice po modelu razširi-združi (*angl.* fork-join) [75–77].

Uporabniški vmesnik Python

Python skripta za simulacije je ponavadi sestavljena iz štirih delov: (1) uvoza vseh potrebnih modulov in paketov, (2) priprave sistema in interakcij, (3) integriranje sistema in (4) analize. V veliko primerih je uporabno, če delamo analizo in integriranje sistema istočasno ali pa če spremenimo nastavitve sistema ali interakcij tekom simulacije. Želeli bi si na primer prekiniti integriranje, če bi neka opazovana količina dosegla določeno vrednost ali pa postopno vklopiti interakcije med fazo ogrevanja, kar je pogosto potrebno za vzpostavitev začetne konfiguracije.

V nadaljevanju si bomo ogledali primer simulacijske skripte Python, ki požene preprosto simulacijo polimera [78] v programu ESPResSo++. Celotna skripta in tudi drugi primeri so na voljo v imeniku `/examples` programa. Podrobnejša razlaga osnovnih komponent programa ESPResSo++ je v poglavju o arhitekturi sistema. Vrstice s komentarji se začnejo z znakom #. Prvi del spodnje skripte naloži razne module Python iz ESPResSo++, standardne knjižnice Python in zunanje pakete:

koda python

```
1 import espresso
2 import time
3 import logging
4 import MPI
5 import numpy
```

V ESPResSo++ moramo v vsaki simulaciji definirati objekt sistema, ki predstavlja fizikalni sistem. Če bi izvajali več simulacij hkrati, potem bi v vsaki definirali svoj lasten objekt sistema. Enostaven primer, kjer je to potrebno, je simulacija vzporednega kaljenja [79]. Objekt sistema zajema glavne komponente simulacije vključno z volumnom, robnimi pogoji, dekompozicijo delcev in komunikacijsko shemo, seznam interakcij ter generator naključnih števil. V spodnjem delu kode ustvarimo objekt sistema (vrstica 6) in mu nastavimo robne pogoje, dekompozicijo ter debelino kože:

```
6 system          = espresso.System()
7 system.bc       = espresso.bc.OrthorhombicBC(system.rng, box)
8 system.storage = espresso.storage.DomainDecomposition(system, nodeGrid,
      cellGrid)
9 system.skin     = 0.3
```

Nato ustvarimo parsko interakcijo, ki jo bomo računali s pomočjo Verletovega seznama (vrstica 11). V tem primeru uporabimo potencial Lennard-Jones, ki ga prirežemo in zamaknemo, tako da dobimo samo odbojno silo. Ta potencial pogosto uporabljamo za mehko snov in predstavlja interakcije trdih objektov kot enote monomerov (imenujemo ga Weeks-Chandler-Andersen ali WCA [80]):

```
10 verletWCA = espresso.VerletList(system, cutoff=rc_max)
11 potWCA    = espresso.interaction.LennardJones(epsilon=1, sigma=1, cutoff
       =2**(1.0/6))
12 interWCA  = espresso.interaction.VerletListLennardJones(verletWCA)
13 interWCA.setPotential(type1=0, type2=0, potential=potWCA)
```

Interakcije potekajo med delci določenega tipa, ki jih označimo s števili. V zadnji vrstici gornje kode uporabimo ključne besede (npr. `type1=0, type2=0`), da jasno označimo tip obeh delcev. V tem primeru med delcema tipa nič nastavimo potencial WCA. Enote v naši skripti ne nastopajo. Enačbe gibanja ESPResSo++ rešuje v brezdimenzijski obliki, zato lahko vnesemo kakršnekoli enote, le da so si konsistentne.

Polimer sestavljen iz N monomerov s strukturo naključnega sprehoda, naredimo tako, da pokličemo metodo `polymerRW` iz `tools.topology` paketa (vrstica 14):

```
14 positions, bonds = espresso.tools.topology.polymerRW(id, startpos, N,
       bondlength)
15 polymer_chain = []
16 for i in range(N):
17     polymer_chain.append([i, positions[i], 0])
18 system.storage.addParticles(polymer_chain, 'id', 'pos', 'type')
19 system.storage.decompose()
```

Metoda `polymerRW` ne ustvari delcev, ampak zgolj vrne seznam položajev monomerov in vezi v obliki seznama n-teric ali urejenih parov. Položaje naknadno pretvorimo v seznam lastnosti delcev z enostavno `for` zanko (vrstici 16, 17), kjer nastavimo identiteto, položaj in tip. Na tem mestu lastnosti shranimo kot seznam in šele `system.storage.addParticles` interpretira te vrednosti (vrstica 18). Lahko bi dodali še eno zanko, da naredimo dodatne stranske verige. Metoda `addParticles` prav tako doda delce v sistem in jih razporedi med procesorje glede na izbrano shemo nastavljeno pri `storage` zgoraj.

Nato zgradimo objekt seznama parov (vrstica 20), da shrani vezi (vrstica 21), naredimo potencial (vrstica 22) ter nastavimo vezne interakcije (vrstica 23) in jih dodamo sistemu (vrstica 24):

```
20 fpl = espresso.FixedPairList(system.storage)
21 fpl.addBonds(bonds)
22 potFENE = espresso.interaction.FENE(K=30.0, r0=0.0, rMax=1.5)
23 interFENE = espresso.interaction.FixedPairListFENE(system, fpl, potFENE)
24 system.addInteraction(interFENE)
```

Nazadnje nastavimo še integrator (vrstica 25), nakar desetkrat poženemo sistem za 100 korakov (vrstica 27 in 28), vmes pa izračunamo tudi tlak (vrstica 29):

```
25 integrator = espresso.integrator.VelocityVerlet(system)
26 ...
27 for i in range(10):
28     integrator.run(100)
29     P = espresso.analysis.Pressure(system).compute()
30     print P
```

Zgornji izrezek kode prikazuje preprosto zanko, čeprav bi zlahka delali zahtevnejše operacije, kot na primer spreminjanje ali dodajanje oz. odstranjevanje delcev v zanki, spreminjanje interakcij, izvajanje potez Monte Carlo, spreminjanje ansamblov (verjetnostna porazdelitev), itd. Prav tako bi bilo možno prekiniti zanko, ko bi na primer tlak dosegel določeno vrednost.

Program za vizualizacijo molekul VMD [73] nudi povezavo, s katero je možno opazovati simulacijo v teku v živo. Sledeči del kode Python aktivira izvoz v program za vizualizacijo molekul VMD:

koda python

```
# povezava VMD
sock = espresso.tools.vmd.connect(system)
...
# poslji trenutne polozaje VMDju
espresso.tools.vmd.imd_positions(system, sock)
```

Ta funkcionalnost je uporabna za kratke produkcijske teke, predstavitve v razredu ali za iskanje napak. Možno bi bilo spremljati tlak ali druge opazovane količine v živo z uporabo modula `Matplotlib` [81], ki omogoča izris funkcij. S svojo prožnostjo je idealen za uporabo skupaj s programom ESPResSo++.

Z uporabo `tools.timers` razreda lahko ob zaključku izvajanja izpišemo čase izvajanja posameznih delov simulacije in velikost seznama sosedov. To funkcionalnost lahko uporabimo za odločitve v zvezi z nastavitvami parametrov, z namenom hitrejšega izvajanja simulacije.

Zasnova programskega jedra ESPResSo++

C++ se je uveljavil kot priljubljen objektno usmerjen programski jezik za programe, pri katerih sta pomembna hitrost in dostop do pomnilnika na nizkem nivoju. Nov uporabnik dosti težje pridobi osnovno razumevanje jezika C++ v primerjavi s Pythonom, kjer se je mogoče udomačiti že po nekaj dnevih. ESPResSo++ je zaradi tega zasnovan tako, da se večini uporabnikov ne bo nikoli potrebno poglabljati v jedro programa ESPResSo++. Svoje lastne simulacije najlažje pričnemo poganjati tako, da uporabimo in prilagodimo eno od obstoječih skript Python.

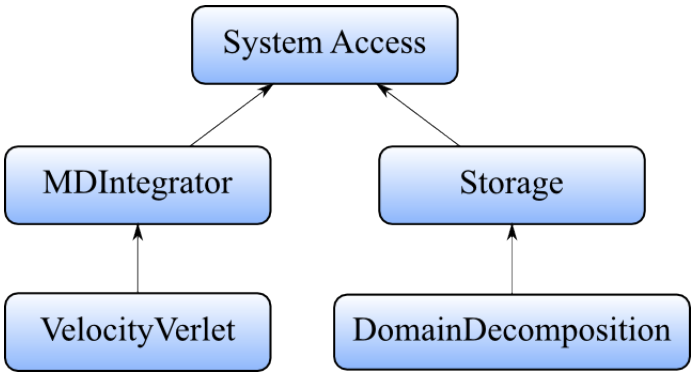
V nadaljevanju predstavimo zasnovo jedra ESPResSo++ in pokažemo, kako različni deli programskega paketa sodelujejo med seboj. V nekaterih delih predpostavljamo, da ima bralec nekaj osnovnega znanja o programiranju v programskem jeziku C++.

Dedovanje

Objektno usmerjeno programiranje omogoča, da z uporabo dedovanja večkrat uporabimo isto kodo, tako da definiramo skupne atribute in metode znotraj osnovnega razreda, te pa nato lahko uporabimo v podrazredih. Funkcije so virtualne, če jih podrazredi lahko povozijo (*angl.* override). Na primer, abstraktni razred `Storage` nudi metode za pošiljanje in prejemanje delcev iz drugih procesorjev (razred je abstrakten, če vsebuje metodo brez implementacije). Te metode uporabimo v podrazredih ter definiramo, kako naj se delci razporedijo med razpoložljivimi procesorji, npr. z uporabo prostorske domenske dekompozicije ali pa z atomsko dekompozicijo [49]. Še en primer je abstraktni osnovni razred za interakcije med delci, ki nudi metode za izračun energije, tlaka in sile. Različni potenciali nato povozijo ustrezne virtualne funkcije, da izračunajo svoje analitične izraze.

Dedovanje v ESPResSo++ pogosto uporabljamo tudi kot neke vrste funktor. Osnovni razred definira določene virtualne funkcije, ki jih nato funkcije v podrazredih povozijo. To omogoča, da npr. v integratorju `VelocityVerlet` kličemo metodo `computeForce()` razreda `Interaction` in druge metode osnovnih razredov `BC` (boundary conditions), `Storage` ter `Particle`, pri čemer je koda integratorja povsem neodvisna od potencialov interakcije. Poleg tega nam ni potrebno vedeti ničesar o tem, kako so delci razporejeni med procesorji. V integratorju preprosto uporabimo iterator, da se sprehodimo preko delcev

in posodobimo njihove položaje ter hitrosti kjer je potrebno.



Slika 3.8: Primer dedovanja razredov. Osnovni razred `SystemAccess` nudi dostop do objekta sistema, zato da lahko npr. objekt razreda `VelocityVerlet` dostopa do delcev, ki so shranjeni v objektu `Storage`.

Tak koncept dobro izraža razširljivost programa. Če npr. želimo dodati podrazred razreda `Interaction`, `Storage` ali `Particle`, nam ni potrebno spreminjati nobenih algoritmov v integratorju `VelocityVerlet`.

Predloge in virtualne funkcije

Pri klicu virtualne funkcije je potrebnih nekaj dodatnih instrukcij, ker se moramo do klicane funkcije prebiti preko tabele kazalcev. V večini primerov to ne predstavlja problema, vendar lahko v kritičnih delih kode, kot npr. v izračunu sile ali energije, kjer funkcijo mnogokrat kličemo, virtualne funkcije opazno upočasnijo izvajanje programa. Zato pri razredih za interakcije uporabljamo predloge (*angl.* templates). V primeru predloge prevajalnik za vsak potencial naredi svojo instanco zanke in se s tem izogne dodatnim klicem.

Z uporabo dedovanja in predlog se izognemo redundanci kode, ki jo lahko zasledimo v ostalih programih za simulacije (npr. LAMMPS [49]), ne da bi s tem upočasnili izvajanje programa zaradi režijskih del. S tem povečamo robustnost in enostavnost vzdrževanja programa. Običajna praksa ostalih programov, da kopirajo del kode in ga nekoliko spremenijo za nek drug namen, v programu ESPResSo++ ni zaželena.

Iteratorji

Pri programiranju v programskem jeziku C++ je pogosto in priporočljivo z iteratorji ločiti algoritme od vsebovalnikov. V programskemu paketu ESPResSo++ tak pristop omogoča spreminjanje interne organizacije vsebovalnikov, ne da bi bilo potrebno spreminjati algoritme, ki delujejo nad vsebovalniki.

Pri zankah preko parov delcev so pari lahko iz Verletovega seznama, iz seznama celic ali iz seznama fiksnih delcev. ESPResSo++ podpira vse omenjene vsebovalnike parov in nudi tudi iteratorje zanj.

Signali

Tekom simulacije lahko pride do raznih fizikalnih ali tehničnih dogodkov, ki morajo morda sprožiti programske module, da se odzovejo na te dogodke. Recimo, če se spremeni identi-

teta delca med premikom v simulaciji Monte Carlo se sproži dogodek, ki posodobi podatke o delcih. Primer tehničnega dogodka bi bil premik delcev v pomnilniku. To bi pomenilo, da moramo posodobiti reference na te delce, saj niso več veljavne. Za take dogodke ESPResSo++ uporablja signale iz knjižnice *Boost.Signals2*, ki implementira postopke za registracijo in povezavo funkcij, da se sprožijo ob določenih dogodkih.

Na primer, kadar se delci premaknejo med procesorji (tj. prevzame jih drug procesor), se v razredu `Storage` sproži signal `onParticlesChanged`. Objekt `FixedParticleList`, ki hrani informacije o vezeh med delci, je povezan na ta signal in ob sprožitvi sporoči podatke o vezeh ustreznemu procesorju, da zgradi vezi med prejetimi delci.

Signale prav tako uporablja `integrator`, da obvesti povezane metode o svojem stanju. ESPResSo++ nudi preprost vmesnik Python, da lahko odklopimo objekt od signala. To omogoča, da zamenjamo termostate ali barostate kar znotraj skripte. Termostat je del integracijskega algoritma, ki vzdržuje nastavljeno temperaturo, barostat pa vzdržuje tlak. S signali lahko integrator razširjamo (dodajamo novo funkcionalnost), ne da bi spreminjali njegovo kodo. Pomembne lokacije v kodi (npr. pred in po integraciji položajev, pred in po izračunu sile, itd.) sprožijo signal, tako da lahko na skoraj poljubno lokacijo priklopimo neko funkcijo. Termostati, barostati, omejtev SETTLE [82], termodinamska sila in deli kode AdResS so vse primeri razširitve integratorja.

Integracija C++ in Pythona

Programski paket ESPResSo++ se zanaša na tesno sodelovanje med jezikoma C++ in Python. Razrede C++ lahko izvozimo v Python, ustvarimo objekte C++ v Pythonu in kličemo njegove metode kar iz skripte simulacije.

Čeprav sta si C++ in Python zelo blizu iz vidika funkcionalnosti, je uporabniški vmesnik Python prijaznejši do uporabnika. Zato kadar izvozimo razred C++ v Python, naredimo razred Python, tako da izpeljemo nov razred iz izvoženega in mu dodelamo vmesnik. To ponavadi pomeni, da parametrom Pythonovih funkcij dodamo imena in privzete vrednosti. Pythonovi parametri so lahko strukture višjega reda, kot npr. seznami ali slovarji, Pythonove metode pa so lahko kombinacija različnih metod C++. Metode C++ za nastavljanje (*setter*) in poizvedovanje (*getter*) izvozimo kot lastnosti. Z ovojno kodo prav tako določimo dodelitev instanc razredov za izvajanje v vzporednem okolju.

Hramba delcev in domenska dekompozicija

Razred `Storage` hrani delce sistema. Ker se pri molekularni dinamiki največ procesorskega časa porabi za izračun sil in energije, morajo biti delci razporejeni tako, da se minimizira računsko breme pri računanju neveznih parskih interakcij. Za parske interakcije kratkega dosega, kot npr. Lennard-Jones, je najbolje, če delce hranimo v majhnih celicah s stranico nekoliko večjo od odreza potenciala. Tako so v interakcije vključeni le delci sosednjih celic. V predelu škatle, kjer se stikata dva procesorja, izmenjamo le manjše število celic. Taki organizacijski shemi rečemo domenska dekompozicija in jo opišemo spodaj. V primeru, da morajo vsi pari atomov sodelovati v interakciji, je bolj primerna atomska dekompozicija [49].

`Storage` hrani delce v majhnih stičnih skupinah (celicah). Vsak procesor je odgovoren za nekaj celic in delce v njih. Poleg tega procesorji hranijo tudi nekaj podatkov o nekaterih delcih sosednjih procesorjev, saj jih potrebujemo za izračun interakcij. Celicam in delcem, ki pripadajo procesorju rečemo „pravi“ delci oz. celice, medtem ko kopijam delcev iz

drugih procesorjev rečemo „duh“ delci oz. celice. Logično organizacijo celic in razporeditev delcev v njih implementirajo podrazredi razreda `Storage`. `DomainDecomposition` je en tak primer.

ESPResSo++ ima iteratorje za prave ali duh celice, ki jih lahko uporabimo za sprehajanje po celicah, za celico pa obstaja iterator, ki se sprehodi čez njene delce. Ta vrne delce neurejeno. Poleg tega obstajajo iteratorji, ki se sprehodijo čez pare delcev, kot ga npr. uporabimo pri gradnji Verletovega seznama. Sledeči del kode iz `VerletList.cpp` ponazarja sprehod čez pare delcev:

koda C++

```
1 CellList cl = getSystem()->storage->getRealCells();
2 for (CellListAllPairsIterator it(cl); it.isValid(); ++it)
3     checkPair(*it->first, *it->second);
```

`CellListAllPairsIterator` iterira preko parov delcev pravih celic, pri čemer upošteva tudi sosednje celice, ki so lahko tudi duh celice.

Do določenega izbranega delca lahko pridemo tudi z uporabo globalnega unikatnega identifikatorja, v kolikor se delec nahaja na procesorju. Preslikava med identifikatorjem delca in podatki o delcu je izvedena z uporabo podatkovne strukture `unordered_map` iz knjižnice *Boost*. To nam omogoča, da hitro pridemo do podatkov o delcu. Metode za dostop do delcev vrnejo referenco do podatkov o delcu, kar omogoča morebitne prihodnje spremembe v razporeditvi delcev v pomnilniku, ne da bi bilo potrebno spreminjati algoritme.

Ko poganjamo simulacije vzporedno na več procesorjih, je potrebno vsake toliko korakov izmenjati lastnosti delcev med procesorji. Komunikacija delcev se zgodi v naslednjih situacijah:

- Če se delec premakne več od določene razdalje in pride do prerazporeditve po celicah. Če se spremeni procesorsko lastništvo nad delcem, je potrebno podatke o delcu sporočiti procesorju, ki je sedaj odgovoren za dotični delec.
- Ko se ustvarijo duh delci na sosednjih procesorjih, je potrebno sporočiti pripadajoče podatke. Za delce duh niso potrebni vsi podatki o delcih, npr. hitrosti in sile pravih delcev se ponavadi ne izmenjajo.
- Potem ko integrator posodobi položaje delcev, se izmenjajo novi položaji, zato da se posodobijo podatki o duh delcih. V primeru, da poganjamo simulacije s termostatom DPD (dissipative particle dynamics) [83], se sporočijo tudi hitrosti.
- Po izračunu sil, se sile, ki jih občutijo duh delci, sporočijo procesorju, na katerem se nahaja pravi delec.
- Če dodamo nov delec v sistem, se med procesorji izmenjujejo podatki, dokler ustrezen procesor ne prevzame lastništva nad delcem.

Podatki o delcih se ne izmenjujejo individualno, ampak se naberejo v medpomnilnik. Nato se izvede samo en komunikacijski korak med procesorji. S tem se izognemo pošiljanju veliko majhnih paketov, kjer se veliko časa porabi zgolj za vzpostavitev povezave.

Razred `Storage` razpolaga z nekaj signali, na katere se povežejo funkcije, ki morajo vedeti kdaj se delci premaknejo. Eden takih signalov je `onParticlesChanged`, ki se sproži,

ko se delci premaknejo v pomnilniku ali med procesorji. Vsi objekti, ki hranijo kazalce na delce, morajo po sprožitvi tega signala posodobiti kazalce.

ESPResSo++ nudi razred `DomainDecomposition`, ki organizira delce na procesorjih v celice glede na prostorsko dekompozicijo domene z uporabo običajne mreže celic. To mrežo potegnemo tudi nekoliko preko mej procesorja. Meloni [84] je pokazal, da se učinkovitost simulacij molekulske dinamike izboljša, če delce shranimo blizu v pomnilniku. `Storage` razred smo napisali tako, da izkorišča to lastnost. ESPResSo++ uporablja dekompozicijsko shemo polovične školjke, ki je lažja za razumevanje in spreminjanje od naprednejših shem kot npr. shema osmine školjke ali metoda razpolovišča [85–88], ki se bolje obnese, če uporabljamo zelo veliko število procesorjev.

Z uvedbo kože [2], prihranimo nekaj komunikacijskega časa, saj delcem omogočimo, da zapustijo svojo celico ali procesor za nekaj integracijskih korakov. Šele, ko delci prepotujejo dlje od določene nastavljene vrednosti debeline kože, se prerazporedijo med procesorji ali celicami. S tem se izognemo časovno potratni izmenjavi delcev tako rekoč brez dodatnih stroškov, saj je domenska dekompozicija pogosto uporabljena v kombinaciji z Verletovim seznamom. Le-ta prav tako uporablja isti koncept kože. Debelino kože moramo pravilno nastaviti, kajti vpliva na učinkovitost izvajanja kode. ESPResSo++ ima metodo `tuneSkin`, ki nam pomaga najti optimalno debelino kože:

koda Python

```
espresso.tools.decomp.tuneSkin(system, integrator)
```

Metoda preizkusi različne vrednosti debeline kože, primerja čase izvajanja in izbere najboljšo z uporabo zlatega reza [89]. Metodo poženemo, ko sta `system` in `integrator` že povsem nastavljena.

Robni pogoji

Večina simulacij uporablja škatlo v obliki kvadra s periodičnimi robnimi pogoji v vseh smereh, obstaja pa veliko primerov, kjer potrebujemo drugačne škatle. Na primer, če preučujemo dogodke na površini, je bolje uporabiti škatlo, ki je periodična samo v dveh smereh. Triklinične škatle so uporabne pri študiji kristalov in za izvajanje neravnotežnih strižnih simulacij. Z uporabo prirezanega oktaedra minimiziramo količino topila pri študiji hidrirane biomolekule, kot npr. beljakovine.

V programu ESPResSo++ se vse metode in podatki povezani s simulacijsko škatlo nahajajo v razredu `BC` (boundary condition). To je abstraktni osnovni razred, zato za implementacijo določene škatle napišemo razred, ki deduje iz tega razreda. Objektno usmerjena zasnova omogoča, da lahko dodamo nove robne pogoje, ne da nam bi bilo potrebno spreminjati karkoli drugega v kodi.

Robne pogoje kličemo, kadar dodajamo nove delce, kadar na novo razporejamo delce, potem ko so se premaknili za večjo razdaljo od debeline kože in kadar potrebujemo parametre simulacijske škatle. Velikost škatle lahko spreminjamo tekom simulacije, kar je potrebno pri simulacijah s stalnim tlakom. Objekt sistema hrani robne pogoje za vsako simulacijo. Duh delce shranimo tako, da pri izračunu neveznih sil [90] zadostujejo evklidske razdalje. S tem se izognemo relativno dragemu klicu `getMinimumVector()`, ki vrne vektor razdalje z uporabo konvencije minimalne slike. Pri veznih interakcijah se temu klicu ne izognemo.

Integracijski algoritmi

V simulacijah molekulske dinamike je hitrostni Verlet najpogostejši algoritem za integriranje položajev in hitrosti delcev. ESPResSo++ ima razred `VelocityVerlet`, ki deduje iz osnovnega razreda `MDIntegrator`. Integrator `VelocityVerlet` uporablja razrede `Interaction`, `BC`, `Extension` in `Storage`.

Koda integratorja je povsem neodvisna od potencialov za interakcije in dekompozicije delcev na procesorje. Integrator se z uporabo iteratorja preprosto sprehodi čez delce, ki jih integrira. Dodajanje podrazredov razredu `Interaction`, `Storage` ali `Particle` ne zahteva nobenih sprememb algoritmov, ki jih uporablja `VelocityVerlet`.

Integrator dobi večino potrebnih informacij, kot je hramba delcev in robni pogoji iz objekta sistema. Ostale lastnosti, kot npr. termostati in barostati, so dodane kot razširitve integratorja. V kodi Python bi integrator poklicali tako:

koda Python

```
integrator = espresso.integrator.VelocityVerlet(system)
integrator.addExtension(langevinT)
for interval in range(100):
    integrator.run(1000)
```

Klic `integrator.run()` lahko kličemo večkrat v zanki, zato da razdelimo integracijo v intervale. V vsakem intervalu ponavadi izpišemo koordinate delcev ali kake druge opazovane spremenljivke. Gornji primer naredi 100 intervalov s 1000 koraki na interval.

n-terice delcev

V Pythonu je n-terica nespremenljiva podatkovna struktura za združevanje podatkov v eno enoto. Lahko si jo predstavljamo kot seznam, ki ga ne moremo več spreminjati, potem ko smo ga ustvarili. Delce v Verletovem seznamu hranimo kot n-terice, kjer je vsak element par delcev. Seznam n-teric prav tako uporabljamo za izključitve, tj. seznam parov delcev za katere ne računamo sil. Informacije o topologiji molekul, kot so vezi in koti, so shranjene v fiksnih seznamih n-teric.

Fiksen seznam n-teric delcev lahko ustvarimo na nivoju skripte z uporabo standardne sintakse Python. ESPResSo++ nudi `FixedPairList`, `FixedTripleList` in `FixedQuadrupleList` za hrambo n-teric, pri čemer določimo identifikatorje delcev, ki sestavljajo vez, kot ali diederski kot. N-terice dodamo v te sezname preko vmesnika Python. Na primer, vez in kot molekule vode, kjer je atom kisika delec 1, bi v skripti dodali tako:

koda Python

```
bonds = espresso.FixedPairList(system.storage)
bonds.addBonds([(1, 2), (1, 3)])
angles = espresso.FixedTripleList(system.storage)
angles.addTriples([(2, 1, 3))])
```

V naslednjem razdelku pokažemo, kako med vezi in kote dodamo interakcije.

Za razliko od ostalih programskih paketov, ESPResSo++ omogoča, da delci nastopajo v neomejenem številu vezi in kotov. To dosežemo tako, da imamo posebne sezname, v katerih hranimo podatke o vezeh in kotih, namesto, da informacijo hranimo v delce same (delec je objekt).

Ko fiksne sezname n-teric porazdelimo med razpoložljive procesorje, prevzamejo la-

stništvo glede na to, kje se nahaja prvi oz. drugi delec (v primeru kota) n-terice. Če je potrebno, za izračun interakcij uporabimo duh delce. To zagotavlja, da so potrebni podatki o delcih na razpolago za izračun veznih sil.

Cena za hrambo podatkov o topologiji molekul v seznamih namesto skupaj z delci, je v dodatni komunikaciji med procesorji, s katero sporočimo n-terice v fiksnih seznamih, potem ko se delci premaknejo preko mej procesorjev. Ker `Storage` ne vodi evidence o številu fiksnih seznamov v uporabi, uporabimo signale, da zagotovimo konsistentnost sistema. Pravočasen prenos podatkov o n-tericah dosežemo tako, da fiksne sezname povežemo na naslednje tri signale razreda `Storage`:

koda C++

```
1 void beforeSendParticles(ParticleList& pl, class OutBuffer& buf);
2 void afterRecvParticles(ParticleList& pl, class InBuffer& buf);
3 void onParticlesChanged();
```

Ob sprožitvi signala `beforeSendParticles`, se identifikatorji delcev iz lokalnega seznama, ki so se premaknili na drug procesor, vpišejo v medpomnilnik `buf` ter pripravijo za pošiljanje. Ko se sproži signal `afterRecvParticles`, se iz medpomnilnika, ki je prišel iz drugega procesorja, doda identifikatorje n-teric v lokalni seznam. Signal `onParticlesChanged` pa sproži gradnjo n-teric delcev iz lokalnih seznamov identifikatorjev.

Interakcije med delci

V Pythonu ustvarimo množico interakcij, ki se prenesejo na nivo C++. Za vsako interakcijo določimo vrsto seznama (npr. Verletov seznam, seznam celic, seznam vseh parov delcev, fiksni pari, itd.) ter potencial, ki ga želimo uporabiti na tem seznamu. V spodnjem primeru pokažemo, kako naredimo običajno nevezno interakcijo kratkega dosega z uporabo Verletovega seznama in potenciala Lennard-Jones:

koda Python

```
1 potLJ   = espresso.interaction.LennardJones(epsilon=1.0, sigma=1.0, cutoff=
      rc)
2 vl      = espresso.VerletList(system, cutoff=rc)
3 interLJ = espresso.interaction.VerletListLennardJones(vl)
4 interLJ.setPotential(type1=0, type2=0, potential=potLJ)
5 system.addInteraction(interLJ)
```

Verletov seznam ponavadi nudi najboljšo zmogljivost za medmolekularne interakcije, kljub temu pa občasno želimo uporabiti seznam celic. Naslednji primer prikaže uporabo seznama celic in potenciala Morse [91]:

koda Python

```
1 potMorse   = espresso.interaction.Morse(epsilon=10.0, alpha=2.0, rMin=0.0,
      cutoff=rc)
2 interMorse = espresso.interaction.CellListMorse(system.storage)
3 interMorse.setPotential(type1=0, type2=0, potential=potMorse)
4 system.addInteraction(interMorse)
```

Interakcije za fiksne sezname n-teric zgradimo in dodamo v sistem z uporabo potenciala FENE [92], na podoben način:

koda Python

```
bonds = [(1, 2), (2, 3), (3, 4)]
potFENE    = espresso.interaction.FENE(K=30.0, r0=0.0, rMax=1.5)
interFENE = espresso.interaction.FixedPairListFENE(system, bonds, potFENE)
system.addInteraction(interFENE)

angles = [(1, 2, 3), (2, 3, 4)]
potCosine    = espresso.interaction.Cosine(K=1.5, theta0=3.1415)
interCosine = espresso.interaction.FixedPairListCosine(system, angles,
    potCosine)
system.addInteraction(interCosine)
```

V vsakem časovnem koraku se zanka sprehodi čez te interakcije, doda ustrezen potencial ter izračuna sile in/ali energijo. Sam potencial je povsem neodvisen od seznama v uporabi. Z uporabo izključitev lahko izločimo določene interakcije, prav tako pa ESPResSo++ nudi možnost, da odstranimo celotno interakcijo.

Zgornji primer je za interakcije kratkega dosega. Za izračun elektrostatskih interakcij dolgega dosega obstaja kar nekaj dobro uveljavljenih metod: Ewaldova sumacija [93], delec-delec delec-mreža (particle-particle particle-mesh, P^3M [94]), predelana P^3M [95] in delec-mreža Ewald (particle-mesh Ewald, PME [96]). ESPResSo++ zaenkrat nudi metodo Ewaldove sumacije.

S časom, ko bodo postale računalniške gruče CPU/GPGPU bolj dostopne, bo smiselno, da določeni algoritmi tečejo na grafičnih procesorjih. Določeni deli algoritma P^3M so recimo idealni za take primere.

PMI

PMI (parallel method invocation) [74] je modul Python, ki implementira model razcepi-združi. Omogoča poganjanje vzporednih simulacij preko serijske (zaporedne) skripte. Zato jih lahko na vzporednih računalnikih poganjajo tudi uporabniki brez kakršnegakoli znanja o vzporednem računanju. PMI ima naslednjo funkcionalnost:

- Poleg kontrolnega procesa, se PMI zažene tudi na razpoložljivih delavskih procesih.
- Kontrolni proces izvaja serijsko skripto in vpokliče delavce šele, ko je ustvarjen objekt PMI ali, če je klicana metoda na objektu PMI.
- Za vsako metodo klicano na objektu PMI, se parametri razpošljejo delavcem in metoda se izvede na vseh procesorjih z istimi argumenti.
- Kontrolni proces čaka, dokler delavci ne zaključijo svojega klica.

Tak pristop je učinkovit samo, če je delo enega klica dovolj veliko za vzporedno izvajanje. Pri simulacijah molekulske dinamike je to ponavadi klic, ki požene integrator, v katerem se računajo sile. Ta klic prevlada nad vsemi ostalimi klici, ki se v glavnem uporabijo za vzpostavitev in definicijo simulacijskega sistema.

Modul PMI je zelo splošno napisan in ga je mogoče uporabiti tudi v drugih programskih paketih. Definicije objektov PMI in lastnosti lahko zlahka vgradimo v Pythonove ovoje, ki so že v uporabi za izvoz razredov C++.

Analiza

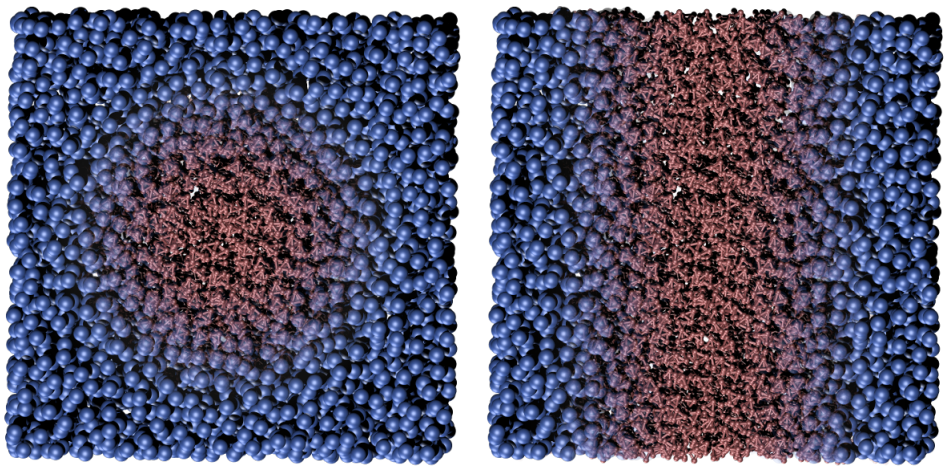
ESPResSo++ je dobro opremljen za analizo sistema tekom simulacije. Vmesnik Python poleg spremljanja temeljnih količin, kot so temperatura in tlak, omogoča tudi usmerjanje simulacij v odvisnosti od dogodkov ali merjenih količin. Primer bi bil recimo skripta, ki ustavi simulacijo, ko je konvergirala radialna porazdelitvena funkcija ali skripta, ki zamenja koeficiente polja sil, ko razdalja med dvema delcema pade pod neko določeno vrednost. Možnosti so številne, veliko pa so jih že realizirali uporabniki programa ESPResSo (predhodnika). Ta sposobnost odpira vrata naprednemu poteku dela, s čemer raziskovalci in procesor prihranijo čas.

Pri simulacijah molekulske dinamike opazovane spremenljivke izračunamo na vsakih nekaj korakov. V programu ESPResSo++ vsi razredi za analizo na nivoju C++ izhajajo iz osnovnega razreda `Observable`. Trenutno so na voljo razredi za analizo temperature, tlaka, tlačnega tenzorja, povprečnega kvadrata odmika, korelacijskih funkcij, težišča sistema in možnost izpisa konfiguracij. Prav tako so v skripti na voljo koordinate delcev in druge lastnosti. Energijo izbrane interakcije dobimo s klicem metode `computeEnergy`. Analizo, za katero niso potrebni časovno potratni izračuni, lahko opravimo kar v skripti Python, sicer pa zlahka dodamo metodo za analizo na nivoju C++.

3.2 Predstavitev metode AdResS ter opis implementacije

Z uporabo večskalnih metod v simulaciji molekulske dinamike pokrijemo večje krajevne in daljše časovne skale. Nam bolj pomembne dele sistema simuliramo z višjo, atomsko ločljivostjo, za preostali del simulacijske škatle pa zadostuje grobozrnata predstavitev. Opis sistema poenostavimo do največje možne stopnje, hkrati pa zadržimo podrobnosti. S tem se izognemo nepotrebnemu zapravljanju procesorskih virov za informacije, ki jih ne potrebujemo. Atomske simulacije čez celotno škatlo so pogosto računsko neizvedljive in tudi nezaželene zaradi velikega števila podatkov, ki jih proizvedejo.

Metoda prilagodljive ločljivosti AdResS (Adaptive Resolution Scheme) [25,26] omogoča združitev dveh različnih opisov sistema v eni sami simulaciji. Molekule prosto prehajajo med obema opisoma ter pri tem spreminjajo svojo ločljivost v odvisnosti od lege v škatli. Gladek prehod dosežemo z interpolacijo sil, ki vladajo v enem in drugem opisu. V prehodnem območju nastopajo molekule v hibridni obliki, saj čutijo delež sile iz obeh opisov. V nadaljevanju podrobneje predstavimo izračun sil v simulacijah AdResS, nato pa sledi opis implementacije vzporedne različice metode v programski paket ESPResSo++.



Slika 3.9: Geometrije atomističnega območja. Leva simulacijska škatla ima sferično geometrijo atomističnega območja v središču, desna pa ravninsko geometrijo prav tako v središču škatle.

Izračun sil

Silo med dvema molekulama izračunamo kot vsoto sil atomistične in grobozrnate predstavitve, pri čemer upoštevamo vrednost uteži. Za izračun sile uporabimo naslednjo formulo:

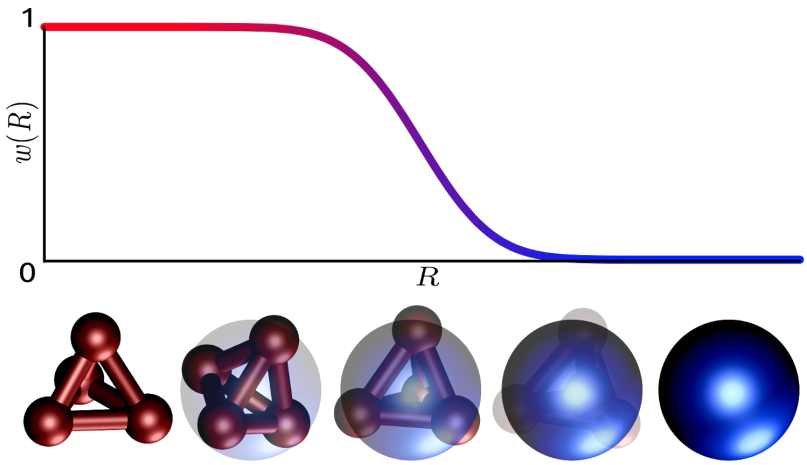
$$\boldsymbol{F}_{\alpha\beta} = w(R_\alpha)w(R_\beta)\boldsymbol{F}^{AT}_{\alpha\beta} + \left[1 - w\left(R_\alpha\right) w\left(R_\beta\right)\right] \boldsymbol{F}^{CG}_{\alpha\beta},$$

kjer α in β označujeta dve molekuli, $w(R_\alpha)$ in $w(R_\beta)$ sta prilagodljivi uteži odvisni od položajev molekul, $\boldsymbol{F}^{AT}_{\alpha\beta}$ je sila med atomi molekule α in β, $\boldsymbol{F}^{CG}_{\alpha\beta}$ je sila med grobozrnatima (*angl.* coarse-grained) predstavitvama molekule α in β. Vrednost uteži w je med 0 in 1 in je odvisna od položaja težišča molekule ali kake druge dobro definirane točke preslikave. V atomističnem območju je $w = 1$, v grobozrnatem je $w = 0$, v vmesnem ali hibridnem območju pa vrednost izpeljemo s sigmoidno funkcijo (slika 3.10), kar omogoči

gladek prehod iz enega opisa v drugega. Očitno je, da za vrednost produkta uteži 1 odpade grobozrnati del v formuli sile, za vrednost produkta uteži 0, pa odpade atomistični del v formuli sile. Pogosto se za utež uporabi neko cos^2 funkcijo:

$$w(R) = \begin{cases} 0 & : \quad R > d_{ex} + d_{hy}, \\ cos^2\left(\frac{\pi}{2d_{hy}}(R - d_{ex})\right) & : \quad d_{ex} + d_{hy} > R > d_{ex}, \\ 1 & : \quad d_{ex} > R, \end{cases}$$

kjer sta d_{ex} in d_{hy} velikosti atomističnega ter hibridnega območja, R je pa oddaljenost od središča atomističnega območja.



Slika 3.10: Utežnostna funkcija. Shematski prikaz utežnostne funkcije $w \in [0, 1]$. Os x na grafu ponazarja oddaljenost od središča atomističnega območja, os y pa vrednost uteži. Pod grafom je prikazano prehajanje molekule iz ene ločljivosti v drugo. Kjer je vrednost uteži $0 < w < 1$ molekula nastopa v hibridni obliki.

Simulacijsko škatlo lahko razdelimo v območja različnih ločljivosti ravninske ali pa sferične geometrije (slika 3.9). Kot središče atomističnega območja je možno nastaviti tudi poljubno molekulo. V tem primeru se območje premika s premikanjem molekule. Velikost posameznih območij nastavi uporabnik, vendar mora biti hibridno območje večje od odreza potenciala, da atomistični delci ne čutijo grobozrnatih neposredno. V vseh primerih je utež odvisna od razdalje težišča molekule do središča atomističnega območja ter njegove velikosti in velikosti hibridnega območja.

V preteklih letih so razvili metodo AdResS za potrebe metodološkega testiranja. Implementacija je tekla zgolj na enem procesorju, zato jo je bilo mogoče uporabiti le na majhnih sistemih. V okviru doktorske disertacije smo implementirali vzporedno različico, ki omogoča izvajanje na več procesorjih ter jo vgradili v programski paket ESPResSo++. V poglavju 3.3 predstavimo uporabo tako implementirane vzporedne različice metode za simulacijo raztopine soli. Ker je koncentracija ionov v raztopini nizka, mora biti sistem dovolj velik, da dosežemo statistično pomembno število ionov. Sisteme take velikosti pa je mogoče simulirati le z učinkovito kodo.

Opis implementacije

Nevezne interakcije so pri simulacijah AdResS drugačne kot pri navadnih simulacijah. Zato smo dodali nove razrede za Verletov seznam in izračun sil. Prav tako smo dodali podatkovne strukture in razrede za rokovanje z atomističnimi delci. V nadaljevanju podrobno opišemo implementacijo vzporedne metode AdResS v ESPResSo++.

Povezava atomističnih delcev z grobozrnatimi

Da lahko molekule spreminjajo ločljivost, kadar se premikajo skozi različne predele škatle, moramo oba opisa molekule povezati. To storimo tako, da identifikator grobozrnatega delca povežemo z identifikatorji atomističnih delcev. Uporabnik v skripti pri dodajanju delcev določi njihov tip (atomistični ali grobozrnati) in ustvari sezname identifikatorjev delcev, pri čemer prvi identifikator pripada grobozrnatemu, ostali pa atomističnim delcem. Sledi primer kode Python, v kateri povežemo grobozrnata delca 1 in 5 z atomističnimi delci 2, 3 in 4 ter 6, 7 in 8:

koda Python

```
1 povezava = [[1, 2, 3, 4], [5, 6, 7, 8]] # seznam povezav
2 ftpl = espresso.FixedTupleList(system.storage)
3 ftpl.addTuples(povezava) # dodamo povezavo
4 system.storage.setFixedTuples(ftpl)
```

Metoda `addTuples` razreda `FixedTupleList` vpiše preslikave med grobozrnatim delcem in atomističnimi delci v razpršilno tabelo ter shrani njihove identifikatorje, kar potrebujemo pri prenosu delcev med procesorji. Spodnji del kode C++ ponazarja podatkovno strukturo, v katero shranimo kazalec grobozrnatega delca in vektor kazalcev pripadajočih atomističnih delcev:

koda C++

```
1 <boost::unordered_map<Particle*, std::vector<Particle*> > > fixedTupleList;
```

Kadarkoli nato potrebujemo informacijo o atomističnih delcih, uporabimo ukaz `find()` za preslikavo na vektor atomov.

Rokovanje in hramba atomističnih delcev

Atomistični delci se v simulacijah AdResS pravzaprav vedejo kot dodatna lastnost grobozrnatih delcev. Ko se grobozrnati delec premakne iz enega procesorja na drugega, ali pa ko se njegov položaj prepogne na drugo stran škatle (zaradi periodičnih robnih pogojev), se z njim prav tako prepognejo vsi pripadajoči atomistični delci. Za to poskrbi razred `FixedTupleList`, ki je povezan na signale `onParticlesChanged`, `beforeSendParticles` in `afterRecvParticles`. Signali sprožijo metode, ki delce zapišejo v medpomnilnik in pripravijo za prenos na ustrezni procesor, preberejo medpomnilnik, ki je prispel iz nekega drugega procesorja ali prepognejo koordinate atomističnih delcev, da sledijo grobozrnatemu delcu. Ko `Storage` iz grobozrnatih delcev naredi duh delce, prav tako pretvori vse njegove atomistične delce v duhove. S tem dosežemo, da so vsi atomistični delci nekega grobozrnatega delca zmeraj ali vsi pravi delci ali pa vsi duhovi. To zagotavlja, da je katerikoli grobozrnati delec skupaj z njegovimi atomi zmeraj na istem procesorju.

Za razliko od grobozrnatih delcev (oz. zgolj delcev v primeru navadne simulacije), ki so

shranjeni v celicah, vsak procesor hrani svoje atomistične delce v enem samem vektorju. Ni nam jih potrebno razporejati v celice, ker Verletov seznam, kot opišemo v naslednjem poglavju, zgradimo na podlagi grobozrnatih delcev. Spodnji del kode ponazarja podatkovne strukture razreda `Storage`. V njih hranimo prave atomistične delce, duh atomistične delce ter razpršilno tabelo, ki preslika identifikator delca na kazalec delca:

koda C++

```
1 std::vector<Particle> AdrATParticles;  // pravi delci
2 std::vector<Particle> AdrATParticlesG; // duh delci
3 boost::unordered_map<longint, Particle*> localAdrATParticles; // preslikava
```

Metoda AdResS sicer v teoriji dovoljuje, da se število delcev v sistemu spreminja, vendar naša implementacija ohranja konstantno število delcev. V simulacijah molekulske dinamike se namreč največ časa porabi za izračun sile, v našem primeru pa v izračunu sile nastopajo dodatni atomistični delci zgolj, če so potrebni v dani lokaciji škatle.

Gradnja Verletovega seznama in izračun sil

Interpolacijsko shemo sil bi lahko uporabili čez celotno simulacijsko škatlo, vendar jo zaradi učinkovitosti uporabimo zgolj v hibridnem območju, torej tam kjer je nujno potrebna, ne pa tudi v atomističnem in grobozrnatem območju. Zato v razredu `VerletListAdress` zgradimo tri Verletove sezname (slika 3.11), in sicer enega za vsako območje:

koda C++

```
1 PairList hyPairs; // CG in AT pari v HY obmocju
2 PairList cgPairs; // CG pari v CG obmocju
3 PairList atPairs; // AT pari v AT obmocju
```

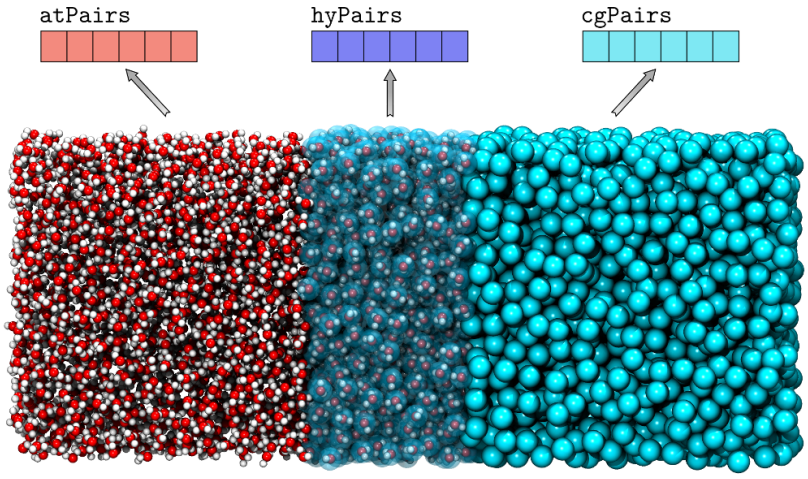
Podatkovne strukture `hyPairs`, `cgPairs` in `atPairs` hranijo sezname parov, za katere je potrebno izračunati sile. `PairList` je razred, ki hrani seznam parov delcev.

Pri gradnji Verletovega seznama se najprej sprehodimo čez vse delce po celicah, ki pripadajo posameznemu procesorju in jih glede na oddaljenost od središča atomističnega območja razvrstimo v tri množice:

koda C++

```
1 boost::unordered_set<Particle*> hyZone; // delci v HY obmocju
2 boost::unordered_set<Particle*> cgZone; // delci v CG obmocju
3 boost::unordered_set<Particle*> atZone; // delci v AT obmocju
```

Pri tem uporabimo podatkovno strukturo `unordered_set` iz boost knjižice, ki je implementirana z uporabo razpršilne tabele. Spodnji del kode prikazuje, kako se z uporabo iteratorja sprehodimo po delcih lokalnih celic in za vsakega izračunamo oddaljenost od središča atomističnega območja `ATcenter` (vrstica 3). V našem primeru uporabimo sferično geometrijo:



Slika 3.11: Verletovi seznami pri simulacijah AdResS. Pri simulacijah AdResS zgradimo tri Verletove sezname na katerih sile izračunamo na različen način. Za delce v seznamu `atPairs` silo izračunamo kot $\boldsymbol{F}_{\alpha\beta} = \boldsymbol{F}_{\alpha\beta}^{AT}$, za delce v seznamu `cgPairs` silo izračunamo kot $\boldsymbol{F}_{\alpha\beta} = \boldsymbol{F}_{\alpha\beta}^{CG}$. Za delce v `hyPairs` pa uporabimo formulo z utežmi, kot smo jo predstavili v začetku poglavja. Slika zaradi ilustrativnih razlogov namenoma ne prikazuje zveznega prehajanja molekul v hibridnem območju.

koda C++

```
1  CellList localcells = getSystem()->storage->getLocalCells();
2  for (CellListIterator it(localcells); it.isValid(); ++it) {
3      Real3D dist = it->getPos() - ATcenter;
4      real distsq = dist.sqr(); // kvadrat razdalje
5      if (distsq < dExsq)
6          atZone.insert(&(*it)); // delec vstavimo v AT obmocje
7      else if (distsq <= dExHysq)
8          hyZone.insert(&(*it)); // delec vstavimo v HY obmocje
9      else
10         cgZone.insert(&(*it)); // delec vstavimo v CG obmocje
11 }
```

Spremenljivki `dExsq` in `dExHysq` hranita kvadrat velikosti atomističnega oz. atomističnega in hibridnega območja. Razdaljo računamo v kvadratu, saj je operacija korena časovno potratna in se ji, kadar je možno, izognemo.

Nato se z iteratorjem sprehodimo čez pare delcev, ter ju vstavimo v metodo `checkPair()`:

koda C++

```
1  CellList cl = getSystem()->storage->getRealCells();
2  for (CellListAllPairsIterator it(cl); it.isValid(); ++it) {
3      checkPair(*it->first, *it->second); // par delcev
4  }
```

Iterator ustvari pare delcev z vsemi ostalimi delci znotraj iste in tudi sosednjih celic. V kolikor sta delca na seznamu izključenih delcev, vzamemo naslednji par. Za izbrani par nato preverimo ali sta delca v hibridnem območju, pri čemer uporabimo prej pripravljeno množico `hyZone`:

koda C++

```
1  checkPair(Particle& pt1, Particle& pt2) {
2      Real3D d = pt1.position() - pt2.position();
3      real distsq = d.sqr(); // kvadrat razdalje
4
5      // ali sta delca v seznamu izkljucitev?
6      if (exList.count(std::make_pair(pt1.id(), pt2.id())) == 1) return;
7      if (exList.count(std::make_pair(pt2.id(), pt1.id())) == 1) return;
8
9      // ali sta delca v HY obmocju?
10     int part1 = hyZone.count(&pt1);
11     int part2 = hyZone.count(&pt2);
```

Ukaz `count()` preveri ali delec pripada množici. Če delca ni v množici vrne 0, sicer vrne 1, saj množica ne dovoljuje dvojnikov. V kolikor je vsaj eden od delcev v hibridnem območju, se spustimo v spodnji blok kode in preverimo ali je kvadrat razdalje med delcema večji od odreza potenciala v atomističnem območju:

```
12     if (part1+part2 > 0) {
13         if (distsq > adrcutsq) return;
```

Če ni večji, koda nadaljuje z izvajanjem, kjer preverimo ali morebiti kateri od dveh delcev ni v hibridnem območju (vrstici 14 in 21). Če je delec v grobozrnatem območju, oba vstavimo v Verletov seznam parov grobozrnatih delcev `cgPairs` (vrstici 17 in 24), saj je produkt njunih uteži 0. To pomeni, da bosta molekuli čutili zgolj grobozrnato silo. Če delec ni v grobozrnatem območju, je v atomističnem. Zato par dodamo v Verletov seznam parov hibridnih delcev `hyPairs` (vrstica 28), saj bomo pri izračunu sile morali upoštevati uteži:

```
14         if (part1 == 0) {
15             if (cgZone.count(&pt1)) {
16                 if (distsq > cutsq) return;
17                 cgPairs.add(pt1, pt2); // dodaj par v seznam CG parov
18                 return;
19             }
20         }
21         else if (part2 == 0) {
22             if (cgZone.count(&pt2)) {
23                 if (distsq > cutsq) return;
24                 cgPairs.add(pt1, pt2); // dodaj par v seznam CG parov
25                 return;
26             }
27         }
28         hyPairs.add(pt1, pt2); // dodaj par v seznam HY parov
29         HYcgat.push_back(true);
```

Do sedaj smo ves čas imeli opravka le z grobozrnatimi delci. V zadnji vrstici zgornjega dela kode smo v vektor bitov `HYcgat` vpisali vrednost `true`. S tem označimo, da smo dodali par grobozrnatih delcev. Vektor uporabimo pozneje pri izračunu sil. V nadaljevanju kode dodamo v Verletov seznam parov `hyPairs` še atomistične delce (vrstica 42), ki pripadajo dodanima grobozrnatima delcema. Do njih pridemo z uporabo `find()` metode na `fixedtupleList` objektu, v katero vpišemo grobozrnata delca, metoda pa vrne vektorja z atomističnimi delci (vrstici 31 in 32). Z dvojno `for` zanko dodamo vse kombinacije parov, saj moramo izračunati sile med vsemi atomi dveh molekul. V vrstici 43 v bitni vektor `HYcgat` vpišemo `false`:

```
30          FixedTupleList::iterator it1, it2;
31          it1 = fixedtupleList->find(&pt1);
32          it2 = fixedtupleList->find(&pt2);
33
34          std::vector<Particle*> atList1, atList2;
35          atList1 = it1->second;
36          atList2 = it2->second;
37
38          for (std::vector<Particle*>::iterator itv = atList1.begin(); itv !=
                atList1.end(); ++itv) {
39              Particle &pt3 = **itv;
40              for (std::vector<Particle*>::iterator itv2 = atList2.begin();
                    itv2 != atList2.end(); ++itv2) {
41                  Particle &pt4 = **itv2;
42                  hyPairs.add(pt3, pt4); // dodaj AT par v seznam HY parov
43                  HYcgat.push_back(false);
44              }
45          }
46      }
```

V kolikor noben od dveh delcev ni v hibridnem območju (vrstica 47), moramo ugotoviti ali par sodi v Verletov seznam atomističnih parov `atPairs` ali v seznam grobozrnatih parov `cgPairs`. Pri tem je dovolj, da zgolj za en delec para pogledamo, če se nahaja v atomističnem območju. V začetku tega poglavja smo namreč napisali, da mora uporabnik nastaviti hibridno območje dovolj veliko, da atomistični delci ne vidijo grobozrnatih. Zato vemo, da sta lahko bodisi oba v atomističnem bodisi oba v grobozrnatem seznamu parov:

```
47      else {
48          if (atZone.count(&pt1)) { // pogledamo samo enega
49              if (distsq > adrcutsq) return;
50
51              atPairs.add(pt1, pt2); // dodamo CG par v seznam AT parov
52              ATcgat.push_back(true);
53
54              FixedTupleList::iterator it1, it2;
55              it1 = fixedtupleList->find(&pt1);
56              it2 = fixedtupleList->find(&pt2);
57
58              std::vector<Particle*> atList1, atList2;
59              atList1 = it1->second;
60              atList2 = it2->second;
61
62              for (std::vector<Particle*>::iterator itv = atList1.begin(); itv
                    != atList1.end(); ++itv) {
63                  Particle &pt3 = **itv;
64                  for (std::vector<Particle*>::iterator itv2 = atList2.begin()
                        ; itv2 != atList2.end(); ++itv2) {
65                      Particle &pt4 = **itv2;
66                      atPairs.add(pt3, pt4); // dodamo AT par v seznam AT
                            parov
67                  }
68              }
69          }
```

in končna možnost, par dodamo v seznam `cgPairs`:

```
70          else {
71              if (distsq > cutsq) return;
```

```
72                cgPairs.add(pt1, pt2); // dodamo CG par v seznam CG parov
73            }
74        }
75    }
```

S tem smo zaključili s pripravo treh Verletovih seznamov in jih lahko uporabimo pri izračunu sil. V primerjavi z gradnjo običajnega Verletovega seznama (pri običajnih ne-AdResS simulacijah), je koda bolj zapletena in vzame nekoliko več časa, vendar nam seznama ni potrebno obnavljati v vsakem simulacijskem koraku.

V metodi za izračun sile se sprehodimo čez grobozrnate pare v seznamu `cgPairs` in izračunamo silo med njimi povsem normalno, kakor bi jo v običajni simulaciji:

koda C++

```
1  for (PairList::Iterator it(cgPairs); it.isValid(); ++it) {
2      Particle &p1 = *it->first;
3      Particle &p2 = *it->second;
4
5      const PotentialCG &potCG = getPotentialCG(p1.type(), p2.type());
6      Real3D force(0.0, 0.0, 0.0);
7
8      if(potCG.computeForce(force, p1, p2)) {
9          p1.force() += force;
10         p2.force() -= force;
11     }
12 }
```

Metoda `getPotentialCG()` vrne grobozrnati potencial, ki vlada med danima dvema tipoma delcev. Na potencialu kličemo metodo `computeForce()`, ki vrne `false`, če sta delca dlje od odreza potenciala. Kljub temu, da smo razdaljo med delcema že preverili v času gradnje Verletovega seznama, moramo to ponovno storiti, saj sta se delca od takrat že premaknila. Silo `force` vpišemo v grobozrnata delca z nasprotnima predznakoma.

Preden izračunamo silo med pari v seznamu `hyPairs`, izračunamo uteži delcev z uporabo `hyZone` množice in jo vpišemo v razpršilno tabelo `weights`. To storimo zato, da se izognemo večkratnemu računanju uteži enega delca, saj se v `hyPairs` posamezen delec pojavi večkrat, v `hyZone` pa dvojnikov ni:

```
13 for (boost::unordered_set<Particle*>::iterator it=hyZone.begin(); it !=
       hyZone.end(); ++it) {
14     Particle &p = **it;
15
16     real d1 = p.position()[0] - ATcenter[0];
17     real d1sq = d1*d1; // kvadrat razdalje
18
19     real w;
20     if (dex2 > d1sq) w = 1;
21     else if (dexdhy2 < d1sq) w = 0;
22     else {
23         w = cos(pidhy2 * (sqrt(d1sq) - dex));
24         w *= w;
25     }
26     weights.insert(std::make_pair(&vp, w));
27 }
```

Spremenljivki `dex2` in `dexdhy2` sta kvadrata velikosti atomističnega območja d_{ex}^2 ter kvadrat vsote atomističnega in hibridnega območja $(d_{ex} + d_{hy})^2$. Spremenljivka `pidhy2` je π

ulomljeno z dvakratnikom velikosti hibridnega območja $\pi/2d_{hy}$ Omenjene spremenljivke izračunamo vnaprej in zgolj enkrat, saj se med tekom simulacije ne spreminjajo.

Sledi izračun sile med pari delcev v hibridnem območju `HYpairs`, kjer z uporabo bitnega vektorja `HYcgat` ločimo med grobozrnatimi in atomističnimi delci. Pri obojih v izračunu sile uporabimo utež:

```
28 bool skip;
29 real w12;
30 for (PairList::Iterator it(hyPairs), int hp=0; it.isValid(); ++it, ++hp) {
31     if (HYcgat[hp]) { // je CG par
32         skip = false;
33 
34         // CG delca v interakciji
35         Particle &p1 = *it->first;
36         Particle &p2 = *it->second;
37 
38         // preberi utezi
39         real w1 = weights.find(&p1)->second;
40         if (w1 != 0.0) w12 = w1 * weights.find(&p2)->second;
41         else w12 = 0.0;
```

V zgornjih treh vrsticah preberemo uteži iz razpršilne tabele in produkt shranimo v `w12`. Če je utež prvega delca enaka nič, za drugi delec ne preberemo uteži. Njun produkt bo namreč prav tako enak nič. V nadaljevanju izračunamo silo med grobozrnatima delcema, vendar le, v kolikor je produkt njunih uteži različen od ena. Kljub temu, da smo delca med gradnjo Verletovega seznama vključili v seznam parov v hibridnem območju, je možno, da sta se delca od takrat že premaknila v atomistično območje.

```
42         if (w12 != 1.0) { // izracunaj CG silo
43             const PotentialCG &potCG = getPotentialCG(p1.type(), p2.type());
44             Real3D forcevp(0.0, 0.0, 0.0);
45             if(potentialCG.computeForce(forcevp, dist, distSqr)) {
46                 forcevp *= (1.0 - w12);
47                 p1.force() += forcevp;
48                 p2.force() -= forcevp;
49             }
50         }
51     }
```

Če je vrednost bitnega vektorja `HYcgat` enaka `false` (vrstica 53) pomeni, da imamo opraviti z atomističnim prispevkom sile:

```
53     else { // je AT par
54         if (skip) continue;
55 
56         if (w12 != 0.0) { // izracunaj AT silo
57             // AT delca v interakciji
58             Particle &p1 = *it->first;
59             Particle &p2 = *it->second;
60 
61             // AT sile
62             const PotentialAT &potAT = getPotentialAT(p1.type(), p2.type());
63             Real3D force(0.0, 0.0, 0.0);
64             if(potAT.computeForce(force, p1, p2)) {
65                 force *= w12;
66                 p1.force() += force;
67                 p2.force() -= force;
```

```
68                 }
69             }
70         }
71 }
```

Preostane nam le še, da izračunamo sile atomističnih parov v `atPairs` seznamu. Silo izračunamo brez uteži, na enak način kot za delce v `cgPairs` le, da uporabimo potencial, ki vlada med atomističnimi delci:

```
72 for (PairList::Iterator it(atPairs); it.isValid(); ++it) {
73     Particle &p1 = *it->first;
74     Particle &p2 = *it->second;
75
76     // AT sile
77     const PotentialAT &potAT = getPotentialAT(p1.type(), p2.type());
78     Real3D force(0.0, 0.0, 0.0);
79     if(potAT.computeForce(force, p1, p2)) {
80         p1.force() += force;
81         p2.force() -= force;
82     }
83 }
```

S tem je izračun sile za vse delce v vseh območjih sistema zaključen. Obstaja več možnosti kaj lahko storimo s silami v integratorju. V naslednjem poglavju opišemo, kako se lotimo integriranja delcev.

Hitrostni Verletov integrator

V naši implementaciji integriramo vse atomistične delce, ne glede na to kje v škatli se nahajajo. Cena integriranja je nizka. Zato dodajanje kompleksnosti v integrator, da bi preverili, v katerem delu škatle se nahaja delec, ne bi bilo smiselno. Grobozrnatih delcev ne integriramo, pač pa jim položaje in hitrosti izračunamo iz težišč položajev in hitrosti pripadajočih atomističnih delcev. Če ponovno zapišemo integracijsko shemo s psevdokodo, kakor smo jo zapisali v 3. poglavju:

psevdokoda

```
1 for (1 ... st_korakov):
2     v = v + 0.5*dt*f
3     r = r + dt*v/m
4     f = sila(r)
5     v = v + 0.5*dt*f
```

je sedaj v primeru simulacije AdResS shema sledeča:

psevdokoda

```
1 for (1 ... st_korakov):
2     v = v + 0.5*dt*f
3     r = r + dt*v/m
4     posodobi_CG_polozaje()
5     f = sila(r)
6     razdeli_CG_sile()
7     v = v + 0.5*dt*f
8     posodobi_CG_hitrosti()
```

kjer `posodobi_CG_polozaje()` izračuna položaje grobozrnatih delcev iz novih položajev pripadajočih atomističnih delcev (izračunanih v vrstici 3), `razdeli_CG_sile()` razdeli sile grobozrnatih delcev na pripadajoče atomistične delce in `posodobi_CG_hitrosti()` izračuna hitrosti grobozrnatih delcev iz novih hitrosti pripadajočih atomističnih delcev (izračunanih v vrstici 7). Silo iz grobozrnatih na atomistične delce porazdelimo po formuli:

$$\boldsymbol{F}_i^{AT} = \boldsymbol{F}_i^{AT} + \frac{m_i}{\sum_{i\in\alpha} m_i}\boldsymbol{F}^{CG}$$

kjer je i indeks delca v molekuli α, $\boldsymbol{F}_i^{AT}$ sila, ki jo čuti atom i, m_i njegova masa, $\boldsymbol{F}^{CG}$ pa sila, ki jo čuti grobozrnati delec molekule α.

Primer skripte za simulacijo AdResS

V nadaljevanju prikažemo relevantne dele skripte Python za simulacijo prilagodljive ločljivosti molekul v obliki tetraedra (spodnji del slike 3.10). Celotna skripta se nahaja v ESPResSo++ mapi `examples`. V sistem vstavimo 40000 molekul, vsaka je sestavljena iz štirih atomov. Škatla je v obliki kocke z velikostjo stranice 54,173. Uporabljamo reducirane enote Lennard-Jones [2].

koda Python

```
1 steps = 10000
2 rc = 2.31 # odrez CG potenciala
3 rca = 1.122462048309373 # odrez AT potenciala
4 skin = 0.4
5 timestep = 0.001
6 intervals = steps/1000
```

Zgornje vrstice nastavijo spremenljivke, ki hranijo število korakov, odrez grobozrnatega in atomističnega potenciala, debelino kože, časovni korak in število intervalov. V vsakem intervalu simulacija izpiše stanje o sistemu. V našem primeru se bo to zgodilo vsakih 1000 korakov. Dalje nastavimo velikost atomističnega ter hibridnega območja. Preostali del škatle bo grobozrnat. V vrstici 11 preberemo konfiguracijo sistema (velikost škatle, seznam x, y in z koordinat delcev, hitrosti ter vezi med delci) iz datoteke `ex.txt`. Datoteka je v obliki programa ESPResSo (predhodnik) in vsebuje podatke le za atomistične delce.

```
7 ex_size = 10.0
8 hy_size = 2.5
9
10 # preberemo ESPResSo konfiguracijsko datoteko
11 Lx, Ly, Lz, x, y, z, vx, vy, vz, bonds = espresso_old.read("ex.txt")
12 num_particles   = len(x) # 20004
13 num_particlesCG = num_particles / 4.0 # stevilo CG delcev
```

V spremenljivko `num_particles` shranimo število atomističnih delcev, ki ga dobimo tako, da vrnemo dolžino seznama x koordinat. Grobozrnatih delcev je štirikrat manj kot atomističnih.

```
14 # domenska dekompozicija AdResS
15 system.storage = espresso.storage.DomainDecompositionAdress(system, nodeGrid
       , cellGrid)
16
17 # pripravimo AT delce
18 allParticlesAT = []
19 allParticles = []
```

```
20 tuples = []
21 for pidAT in range(num_particles):
22     allParticlesAT.append([pidAT, # identifikator
23                            Real3D(x[pidAT], y[pidAT], z[pidAT]),
24                            Real3D(vx[pidAT], vy[pidAT], vz[pidAT]),
25                            1, 1.0, 1]) # tip, masa, AT delec
```

Za dekompozicijo izberemo `DomainDecompositionAdress`, nato z uporabo `for` zanke ustvarimo seznam atomističnih delcev sestavljenih iz identifikatorja `pidAT`, koordinat in hitrosti, kjer uporabimo prej ustvarjene sezname `x`, `y`, `z`, `vx`, `vy`, `vz` ter tipa (1), mase (1.0) in podatka, da gre za atomistični delec. Pri grobozrnatih delcih bomo vstavili število nič.

V nadaljevanju kode ustvarimo grobozrnate delce, pri čemer njihove koordinate in hitrosti izračunamo kot težišča atomističnih delcev. Seznam delcev `allParticlesAT` ustvarimo samo zato, da si tukaj nekoliko olajšamo delo. Grobozrnate delce bi sicer lahko ustvarili tudi neposredno, brez vmesnega seznama.

```
26 # ustvarimo CG delce kot tezisca AT delcev
27 for pidCG in range(num_particlesCG):
28     cmp = [0,0,0]
29     cmv = [0,0,0]
30     tmptuple = [pidCG+num_particles]
31     # izracun tezisc
32     for pidAT in range(4):
33         pid = pidCG*4+pidAT
34         tmptuple.append(pid)
35         pos = (allParticlesAT[pid])[1]
36         vel = (allParticlesAT[pid])[2]
37         for i in range(3):
38             cmp[i] += pos[i] # mase AT delcev so 1.0, mnozenje ni potrebno
39             cmv[i] += vel[i]
40     for i in range(3):
41         cmp[i] /= 4.0 # 4.0 je masa molekule
42         cmv[i] /= 4.0
```

V spremenljivko `tmptuple` vpišemo najprej identifikator grobozrnatega delca (vrstica 30), nato pa vse štiri identifikatorje pripadajočih atomističnih delcev (vrstica 34). Težišči položajev $\boldsymbol{R}_\alpha$ (spremenljivka `cmp`) in hitrosti $\boldsymbol{V}_\alpha$ (spremenljivka `cmv`) grobozrnatega delca molekule α izračunamo po formulah:

$$\boldsymbol{R}_\alpha = \frac{\sum_{i\in\alpha} m_i \boldsymbol{r}_i}{\sum_{i\in\alpha} m_i}, \qquad \boldsymbol{V}_\alpha = \frac{\sum_{i\in\alpha} m_i \boldsymbol{v}_i}{\sum_{i\in\alpha} m_i}$$

kjer sta $\boldsymbol{r}_i$ ter $\boldsymbol{v}_i$ položaj ter hitrost delca i molekule α. V vrsticah 38 in 39 gornje kode smo množenje preskočili, saj so v našem primeru mase atomov enake ena.

Delce nato dodamo v seznam `allParticles`, pri čemer najprej vstavimo grobozrnati delec in nato štiri pripadajoče atomistične delce, ki jih preberemo iz seznama `allParticlesAT`. Tip grobozrnatega delca nastavimo na nič, maso pa štiri. V vrstici 57 v seznam `tuples` vstavimo seznam `tmptuple`.

```
43     allParticles.append([pidCG+num_particles, # najprej vstavimo CG delec
44                          Real3D(cmp[0], cmp[1], cmp[2]), # polozaj
45                          Real3D(cmv[0], cmv[1], cmv[2]), # hitrost
46                          0, 4.0, 0]) # tip, masa, CG delec
47
48     for pidAT in range(4):
49         pid = pidCG*4+pidAT
```

```
50          allParticles.append([pid, # zdaj dodamo AT delce
51                               (allParticlesAT[pid])[1], # polozaj
52                               (allParticlesAT[pid])[2], # hitrost
53                               (allParticlesAT[pid])[4], # tip
54                               (allParticlesAT[pid])[5], # masa
55                               (allParticlesAT[pid])[6]]) # AT delec
56
57      tuples.append(tmptuple)
```

Šele z naslednjim ukazom delce končno dodamo v sistem:

```
58 # dodaj delce
59 system.storage.addParticles(allParticles, "id", "pos", "v", "type", "mass",
       "adrat")
```

Dodamo še n-terice `tuples` in vezi `bonds` ter nato kličemo `decompose()`, ki ustvari duhove delcev, pripravi Verletov seznam, itd.:

```
60 # dodaj n-terice
61 ftpl = espresso.FixedTupleList(system.storage)
62 ftpl.addTuples(tuples)
63 system.storage.setFixedTuples(ftpl)
64
65 # dodaj vezi med AT delci
66 fpl = espresso.FixedPairListAdress(system.storage, ftpl)
67 fpl.addBonds(bonds)
68
69 # naredi dekompozicijo sistema
70 system.storage.decompose()
```

V nadaljevanju dodamo Verletov seznam AdResS, pri čemer podamo enak odrez potenciala (`rc`) za atomistično in grobozrnato območje, velikost atomističnega (`ex_size`) ter hibridnega območja (`hy_size`) in še središče atomističnega območja s koordinatami x, y, z v središču škatle:

```
71 # Verletov seznam AdResS
72 vl = espresso.VerletListAdress(system, cutoff=rc+skin, adrcut=rc+skin,
73                                 dEx=ex_size, dHy=hy_size,
74                                 adrCenter=[Lx/2.0, Ly/2.0, Lz/2.0])
```

Spodnje vrstice nastavijo nevezni potencial Lennard-Jones med atomističnimi delci ter efektivni potencial med grobozrnatimi delci, ki je zapisan v obliki tabele v datoteki `tabMorse.txt`. Vrstici 77 in 78 ustvarita objekt potenciala, vrstici 79 in 80 dodata potenciala v objekt nevezne interakcijo `interNB`, v vrstici 81 pa dodamo interakcijo v seznam interakcij sistema. Ko dodamo potencial, nastavimo tudi, med katerim parom tipov delcev naj se uporabi:

```
75 # nevezne interakcije
76 interNB  = espresso.interaction.VerletListAdressLennardJones(vl, ftpl)
77 potWCA   = espresso.interaction.LennardJonesCapped(epsilon=1.0, sigma=1.0,
       cutoff=rca) # AT
78 potMorse = espresso.interaction.Tabulated(itype=2, filename="tabMorse.txt",
       cutoff=rc) # GZ
79 interNB.setPotentialAT(type1=1, type2=1, potential=potWCA)    # AT
80 interNB.setPotentialCG(type1=0, type2=0, potential=potMorse) # GZ
81 system.addInteraction(interNB)
```

Vezni potencial dodamo po podobnem postopku. Med vse atome znotraj molekule nastavimo potencial FENE in Lennard-Jones:

```
82 # vezne interakcije
83 potFENE = espresso.interaction.FENE(K=30.0, r0=0.0, rMax=1.5)
84 potLJ = espresso.interaction.LennardJones(epsilon=1.0, sigma=1.0, shift=True
       , cutoff=rca)
85 interFENE = espresso.interaction.FixedPairListFENE(system, fpl, potFENE)
86 interLJ = espresso.interaction.FixedPairListLennardJones(system, fpl, potLJ)
87 system.addInteraction(interFENE)
88 system.addInteraction(interLJ)
```

Preostane nam še, da nastavimo integrator, mu dodamo razširitvi za AdResS in termostat ter poženemo simulacijo:

```
89 # hitrostni Verlet integrator
90 integrator = espresso.integrator.VelocityVerlet(system)
91 integrator.dt = timestep
92
93 # dodamo razsiritev AdResS
94 adress = espresso.integrator.Adress(system)
95 integrator.addExtension(adress)
96
97 # dodamo razsiritev termostata
98 langevin = espresso.integrator.Langevin(system)
99 langevin.gamma = 0.5
100 langevin.temperature = 1.0
101 integrator.addExtension(langevin)
102
103 # analiza sistema
104 temperature = espresso.analysis.Temperature(system)
105
106 sys.stdout.write(' step        T\n')
107 fmt = '%5d %8.4f\n'
108 nsteps = steps / intervals
109 for s in range(1, intervals + 1):
110     integrator.run(nsteps)
111     sys.stdout.write(fmt % (step, T))
```

Skripta bo v vsakem intervalu izpisala število korakov ter temperaturo sistema.

Meritve hitrosti izvajanja

Poglejmo sedaj še, kako je s časom izvajanja simulacije. V ta namen poženemo gornjo simulacijo tetraedričnih molekul dolgo 10000 korakov in sicer za navadno atomistično in grobozrnato (tj. ne-AdResS) simulacijo ter tri simulacije AdResS, pri katerih spreminjamo velikosti atomističnega in hibridnega območja. Simulacije smo poganjali na štirijedrnem Intel Core i7-2600 procesorju s frekvenco 3,4 GHz z 8192 kB predpomnilnika na operacijskem sistemu Ubuntu 12.04.1 64-bit z linux jedrom 3.2. Čas izvajanja je odvisen od nastavitve velikosti območja. Večje kot je atomistično območje, več časa potrebujemo za izračun sil. Rezultati so prikazani v tabeli 3.1.

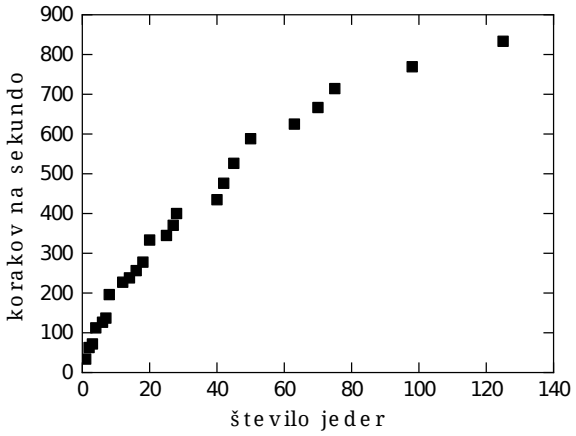
Pri eni od simulacij AdResS smo nastavili atomistično območje čez celo škatlo. V primerjavi z atomistično simulacijo je bil čas izvajanja nekoliko daljši. To je pričakovano, saj se porabi nekaj več časa za gradnjo Verletovega seznama, poleg tega pa je potrebno tudi posodabljati in med procesorji prenašati informacije o n-tericah. Pri tretji simulaciji AdResS smo čez celotno škatlo računali zgolj grobozrnate sile, kar smo dosegli tako, da smo nastavili velikost ex in hy na nič. Časa slednjih dveh simulacij služita kot okvir, znotraj katerega se gibljejo časi hibridnih simulacij. V našem primeru smo z atomističnim

tip simulacije	velikost *ex*	velikost *hy*	čas izvajanja
atomistična			108 s
AdResS	cela škatla	0	120 s
AdResS	10,0	2,5	77 s
AdResS	0	0	74 s
grobozrnata			17 s

Tabela 3.1: Časi izvajanja za 10000 korakov simulacije tetraedričnih molekul. Pognali smo pet simulacij in sicer v prvi vrstici je čas izvajanja atomistične simulacije, kjer sile računamo povsem običajno. Sledijo tri večskalne simulacije z uporabo metode prilagodljive ločljivosti in v zadnji vrstici čas navadne grobozrnate simulacije. Pri simulacijah AdResS spreminjamo velikost atomističnega (*ex*) ter hibridnega (*hy*) območja. S tem seveda vplivamo na čas izvajanja. Velikost je izražena v enotah Lennard-Jones, čas pa v sekundah, ki so pretekle od začetka do konca simulacije.

območjem velikosti 10,0 in hibridnim velikosti 2,5 dosegli čas, ki je nekoliko daljši od časa simulacije z zgolj grobozrnatimi silami, vendar bistveno krajši od navadne atomistične simulacije. Grobozrnata simulacija je potrebovala najmanj in bistveno manj časa od ostalih simulacij.

Primer nazorno pokaže, da je AdResS orodje, ki omogoča, da izberemo del simulacijske škatle, ki nam je bolj pomemben in ga simuliramo v višji ločljivosti. Za preostali, nam manj pomembni del škatle, pa s simuliranjem v nižji ločljivosti prihranimo nekaj dragocenih procesorskih virov. To pride še posebej do izraza pri večjih sistemih, kjer je nujna uporaba vzporedne različice metode AdResS.



Slika 3.12: Simulacija tetraedričnih molekul na računalniški gruči. Številko integracijskih korakov na sekundo v odvisnosti od števila procesorskih jeder [29].

Simulacijo tetraedričnih molekul s 5000 molekulami smo pognali tudi na računalniški gruči, kjer je vsak računalnik opremljen z dvema štirijedrnima Intel Xeon X5570 procesorjema s frekvenco 2,93 GHz z 8192 kB predpomnilnika. Skupno število procesorskih jeder je 128. Operacijski sistem je Suse SLES 11. Računalniki so med seboj povezani z infiniband povezavo. Velikost atomističnega območja je bila nastavljena na 12, velikost hibridnega pa 2,5 (v Lennard-Jones enotah).

Slika 3.12 prikazuje število integracijskih korakov na sekundo v odvisnosti od števila

procesorskih jeder. Število korakov narašča s številom procesorskih jeder. Do dvajsetih jeder število korakov linearno narašča, nato se pohitritev postopno manjša. Pohitritev (S_n) je izražena kot kvocient časa izvajanja na enem procesorju (T_1) in časa izvajanja na n procesorjih (T_n): $S_n = \frac{T_1}{T_n}$. Pri 30 jedrih je pohitritev približno 20x, pri 50 jedrih je 35x, pri 128 jedrih pa približno 55x. To lahko pripišemo manjšemu številu delcev na jedro in pa povečani količini komunikacije med računalniki [29].

3.3 Priprava in simulacija raztopine soli v večskalni tehniki

V tem poglavju izpeljemo hibridni atomistično-grobozrnati model za raztopino soli, tj. natrijeve (Na^+) in klorove (Cl^-) ione raztopljene v tekoči vodi pri sobnih pogojih [97]. Začnemo s predstavijo metodologije, nato opišemo podrobnosti priprave simulacije z novo razvito metodo AdResS, vključno z deli skripte Python za ESPResSo++, nakar sledi predstavitev rezultatov.

Metodologija

Najprej poženemo atomistično simulacijo, da dobimo referenčni sistem. Uporabimo običajna modela vode SPC [98] in SPC/E [99] s polji sil GROMOS [44] in AMBER [100] za ione. Kljub temu, da je molekula vode majhna, predstavlja velik izziv za računalniške simulacije. Obstaja mnogo različnih modelov vode, kar nakazuje, da noben od njih ni uspešen v kvantitativnem reproduciranju vseh lastnosti prave vode. Modela SPC in SPC/E veljata za enostavna tritočkovna modela vode in sta zaradi računske učinkovitosti pogosto uporabljena v simulaciji molekulske dinamike.

Elektrostatiko obravnavamo z metodo posplošenega reakcijskega polja [30], ki omogoča, da jo računamo kot interakcijo z omejenim dosegom. Elektrostatsko silo med delcema i in j izračunamo kot:

$$\mathbf{F}_{elek}^{AT}(\mathbf{r}_{ij}) = \frac{q_i q_j}{4\pi\epsilon_0\epsilon_1}\left[\frac{1}{r_{ij}^3} - \frac{1}{R_c^3}\frac{(2\epsilon_2 - 2\epsilon_1)(1+\kappa R_c) + \epsilon_2(\kappa R_c)^2}{(\epsilon_1 + 2\epsilon_2)(1+\kappa R_c) + \epsilon_2(\kappa R_c)^2}\right]\mathbf{r}_{ij},$$

kjer je radij odreza R_c nastavljen na 0,9 nm, relativni dielektrični konstanti $\epsilon_1 = 1$, $\epsilon_2 = 80$ in recipročna vrednost Debyeve dolžine κ= 3,25 nm^{-1}. q_i in q_j sta naboja delcev, ϵ_0 pa dielektrična konstanta vakuuma. Uporabljamo eno molarno koncentracijo raztopine soli. Taka koncentracija je višja od fiziološke koncentracije, ki je približno 0,15 molarna. To je zato, ker potrebujemo zadostno število ionov za sprejemljivo statistiko. Metoda deluje tudi pri nižjih koncentracijah, a bi morali uporabiti še večjo simulacijsko škatlo. Poleg tega je povsem običajno, da se poskuse izvaja pri eno molarni koncentraciji.

Potem, ko pridobimo referenčni atomistični sistem, nastavimo grobozrnati model. To storimo tako, da predstavimo celotno molekulo vode kot en sam delec. Za grobozrnate interakcije voda-voda in voda-ioni izpeljemo efektivni parski potencial, tako da reproduciramo strukturne lastnosti referenčnega atomističnega modela. Potencial dobimo s strategijo grobozrnenja, ki temelji na strukturi z uporabo metode iterativne Boltzmannove inverzije [101, 102]. Sile iz potencialov izračunamo na običajen način, tj. kot negativni gradient potenciala. Za interakcije ioni-ioni v grobozrnatem modelu nastavimo $\epsilon_1 = 80$ za posplošeno reakcijsko polje, zato da primerno opišemo elektrostatske interakcije med ioni. Parametri za interakcije Lennard-Jones med ioni in njihovi naboji so enaki kot v atomistični simulaciji.

Z atomističnim in grobozrnatim sistemom na mestu, vzpostavimo sistem z uporabo metode prilagodljive ločljivosti AdResS [25, 26, 103]. Skupno silo na molekulo α izrazimo kot [104, 105]:

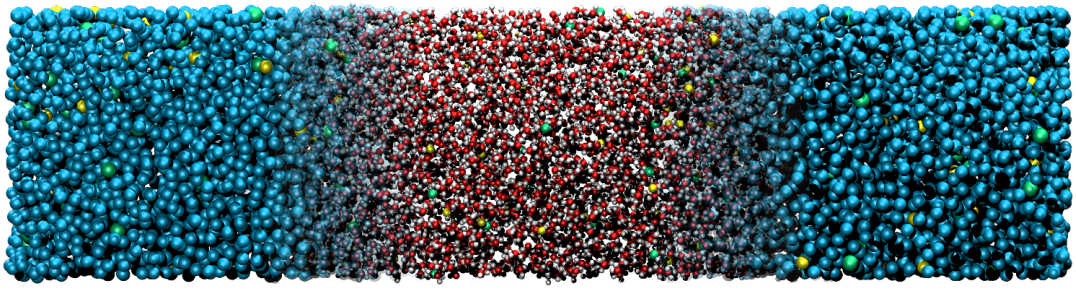
$$\mathbf{F}_\alpha = \sum_{\beta\neq\alpha}\left(w(X_\alpha)w(X_\beta)\mathbf{F}_{\alpha\beta}^{AT} + [1 - w(X_\alpha)w(X_\beta)]\,\mathbf{F}_{\alpha\beta}^{CG}\right) - \mathbf{F}_\alpha^{th}(X_\alpha),$$

kjer sta X_α in X_β koordinati x težišč položajev molekul α in β, $\mathbf{F}_{\alpha\beta}^{AT}$ in $\mathbf{F}_{\alpha\beta}^{CG}$ sta sili med molekulama α in β, ki ju občutita iz atomistične in grobozrnate predstavitve. Utežnostna funkcija $w(X) \in [0,1]$ poskrbi za gladek prehod med obema območjema. $\mathbf{F}_{\alpha}^{th}$ je termodinamska sila, ki kompenzira razliko v kemijskem potencialu zaradi različnih ločljivosti opisa in zagotavlja enakomerno gostotno porazdelitev čez celotno simulacijsko škatlo [27, 104]. Uporabimo tri termodinamske sile in sicer za molekule vode ter natrijeve in klorove ione. Molekularni indeks α določi, katero termodinamsko silo moramo uporabiti. Termodinamska sila deluje v odvisnosti od koordinate x težišča položaja molekule.

Priprava simulacije

Simulacije poganjamo s programom ESPResSo++. Poleg metode AdResS, ki smo jo predstavili v prejšnjem poglavju, smo v programski paket vgradili tudi novo nevezno interakcijo za obravnavo posplošenega reakcijskega polja, termodinamsko silo ter algoritem SETTLE [82], s katerim naredimo molekule vode rigidne.

Simuliramo dva sistema, enega z velikostjo škatle 16,20 nm x 4,05 nm x 4,05 nm z 8508 molekulami vode SPC in 160 natrijevimi ter 160 klorovimi ioni. Za drugi sistem uporabimo molekule vode SPC/E. Integriramo s hitrostnim Verletovim algoritmom s časovnim korakom 1 fs. Temperaturo nastavimo na 300 K z uporabo Langevinovega termostata [2]. Simulacije poganjamo deset milijonov korakov, kar pri naši velikosti koraka ustreza desetim nanosekundam.



Slika 3.13: Shematski prikaz simulacijske škatle. Atomistično območje je na sredi škatle in prehaja v grobozrnato vzdolž smeri x (vodoravno na sliki). Ioni so sicer v obeh ločljivostih predstavljeni z enim delcem, vendar nastopajo v različnih interakcijah.

Vseatomska sistema smo nastavili na sledeči način: za sistem z molekulami vode SPC uporabimo Lennard-Jones in elektrostatske interakcije z običajnimi parametri polja sil GROMOS [44] med atomi molekul vode in ioni. Za sistem z molekulami vode SPC/E uporabimo Lennard-Jones in elektrostatske interakcije s poljem sil AMBER99 [100] (parametri za polji sil so prikazani v tabeli 3.2). Enak večskalni model raztopine soli seveda lahko uporabimo s katerimkoli klasičnim nepolariziranim modelom vode. Ker mora biti časovni korak simulacije dovolj kratek, da zajame najhitrejše gibanje v sistemu, uporabimo algoritem SETTLE in tako preprečimo nihanja med vodikom in kisikom. V nasprotnem primeru bi moral biti časovni korak štirikrat krajši.

Efektivne potenciale obeh sistemov za interakcije med grobozrnatimi molekulami vode ter med molekulami vode in ioni smo dobili tako, da smo najprej naredili Boltzmannovo inverzijo radialnih porazdelitvenih funkcij atomističnega sistema [102]. To nam je dalo prve približke potencialov, s katerimi smo pognali grobozrnato simulacijo in izračunali radialne

parameter	model SPC	model SPC/E
ϵ_{OO}	0,6506	0,6506
σ_{OO}	0,3166	0,3166
ϵ_{NaO}	0,2004	0,0868
σ_{NaO}	0,2855	0,3247
ϵ_{NaNa}	0,0618	0,0116
σ_{NaNa}	0,2575	0,3328
ϵ_{NaCl}	0,1659	0,0696
σ_{NaCl}	0,3384	0,3865
ϵ_{ClO}	0,5383	0,5217
σ_{ClO}	0,3752	0,3783
ϵ_{ClCl}	0,4456	0,4184
σ_{ClCl}	0,4448	0,4401
q_O	-0,82	-0,85
q_H	+0,41	+0,42
q_{Na}	+1,0	+1,0
q_{Cl}	-1,0	-1,0

Tabela 3.2: Parametri za polji sil GROMOS in AMBER99 za oba modela vode. Enote za epsilon (ϵ) so kJmol^{-}1, za sigmo (σ) nm. Naboji (q) so podani v osnovnem naboju.

porazdelitvene funkcije. Z njimi popravimo efektivni potencial, tako da odštejemo Boltzmannovo inverzijo kvocienta grobozrnate in atomistične radialne porazdelitvene funkcije. Postopek ponavljamo, dokler grobozrnati model ne reproducira strukture atomističnega modela. V ta namen smo izdelali skripto Python, ki postopek v celoti avtomatizira [106].

Pri simulaciji AdResS združimo sistema obeh ločljivosti v eno samo škatlo in sicer z uporabo ravninske geometrije. Atomistični opis je v središču škatle ter prehaja po osi x v grobozrnati opis na robovih škatle. Škatla je prikazana na sliki 3.13. Velikost atomističnega območja nastavimo na 4 nm, debelino hibridnega pa na 3 nm.

Za simulacijo AdResS v skripti Python nastavimo potenciale, ki vladajo v atomističnem in grobozrnatem območju. V ta namen naredimo Verletov seznam AdResS (vrstica 2), nastavimo nevezne interakcije Lennard-Jones (vrstica 3), definiramo potenciale (vrstice 6–11) in jih nastavimo objektu interakcije. Pri tem določimo, kakšni potenciali vladajo med posameznimi tipi atomističnih delcev (vrstice 14–19):

koda Python

```
1 # nevezne interakcije
2 vl1 = espresso.VerletListAdress(system, cutoff=rc+skin, adrcut=rc+skin, dEx=
      ex_size, dHy=hy_size, adrCenter= box_cntr)
3 interNB = espresso.interaction.VerletListAdressLennardJones(vl1, ftpl)
4
5 # definicija AT potencialov
6 potOO   = espresso.interaction.LennardJones(epsilon=lj_eps_OO, sigma=
      lj_sig_OO, cutoff=rc)   # O - O
7 potNaO  = espresso.interaction.LennardJones(epsilon=lj_eps_NaO, sigma=
      lj_sig_NaO, cutoff=rc)  # Na - O
```

```
8  potNaNa = espresso.interaction.LennardJones(epsilon=lj_eps_NaNa, sigma=
       lj_sig_NaNa, cutoff=rc) # Na - Na
9  potNaCl = espresso.interaction.LennardJones(epsilon=lj_eps_NaCl, sigma=
       lj_sig_NaCl, cutoff=rc) # Na - Cl
10 potClO  = espresso.interaction.LennardJones(epsilon=lj_eps_ClO, sigma=
       lj_sig_ClO, cutoff=rc)  # Cl - O
11 potClCl = espresso.interaction.LennardJones(epsilon=lj_eps_ClCl, sigma=
       lj_sig_ClCl, cutoff=rc) # Cl - Cl
12
13 # dodamo potenciale med AT delce
14 interNB.setPotentialAT(type1=1, type2=1, potential=potOO)   #  O - O
15 interNB.setPotentialAT(type1=4, type2=1, potential=potNaO)  # Na - O
16 interNB.setPotentialAT(type1=4, type2=4, potential=potNaNa) # Na - Na
17 interNB.setPotentialAT(type1=4, type2=6, potential=potNaCl) # Na - Cl
18 interNB.setPotentialAT(type1=6, type2=1, potential=potClO)  # Cl - O
19 interNB.setPotentialAT(type1=6, type2=6, potential=potClCl) # Cl - Cl
```

Vrednosti spremenljivk `epsilon` in `sigma` nastavimo, kakor določa polje sil (glej tabelo 3.2). Interakcij z atomom vodika tukaj ne dodamo, saj jih izbrani model vode ne predvideva. S tem smo definirali potenciale Lennard-Jones, ki nastopajo v atomističnem prispevku sil. Nato določimo še grobozrnate potenciale (vrstice 20–26), jih dodamo pripadajočim grobozrnatim delcem (vrstice 29–34), nakar objekt neveznih interakcij dodamo sistemu (vrstica 37):

```
20 # definicija CG potencialov
21 potCGCG = espresso.interaction.Tabulated(itype=3, filename=tabCGCG, cutoff=
       rcCG) # CG - CG
22 potCGNa = espresso.interaction.Tabulated(itype=3, filename=tabCGNa, cutoff=
       rcCG) # CG - Na
23 potCGCl = espresso.interaction.Tabulated(itype=3, filename=tabCGCl, cutoff=
       rcCG) # CG - Cl
24 potNaNa = espresso.interaction.Tabulated(itype=3, filename=tabNaNa, cutoff=
       rcCG) # Na - Na
25 potNaCl = espresso.interaction.Tabulated(itype=3, filename=tabNaCl, cutoff=
       rcCG) # Na - Cl
26 potClCl = espresso.interaction.Tabulated(itype=3, filename=tabClCl, cutoff=
       rcCG) # Cl - Cl
27
28 # dodamo potenciale med CG delce
29 interNB.setPotentialCG(type1=3, type2=3, potential=potCGCG) # CG - CG
30 interNB.setPotentialCG(type1=5, type2=3, potential=potCGNa) # Na - CG
31 interNB.setPotentialCG(type1=5, type2=5, potential=potNaNa) # Na - Na
32 interNB.setPotentialCG(type1=5, type2=7, potential=potNaCl) # Na - Cl
33 interNB.setPotentialCG(type1=7, type2=3, potential=potCGCl) # Cl - CG
34 interNB.setPotentialCG(type1=7, type2=7, potential=potClCl) # Cl - Cl
35
36 # dodamo interakcijo v sistem
37 system.addInteraction(interNB)
```

Efektivne potenciale imamo tipično zapisane v datoteki, kjer je prvi stolpec razdalja, drugi in tretji pa potencial ter sila za dano razdaljo. Zato uporabimo „tabuliran“ tip potenciala, pri čemer moramo razdalje, ki niso podane v datoteki, interpolirati. V našem primeru uporabimo kubično interpolacijo, kar določimo s ključno besedo `itype=3`. Ime datoteke s potencialom nastavimo s `filename` parametrom.

Poleg potenciala Lennard-Jones, moramo atomističnim delcem dodati tudi elektrostatske interakcije. Ker ESPResSo++ ne omogoča neposrednega dodajanja večih potencialov

za neko kombinacijo tipov delcev, ustvarimo še en Verletov seznam (vrstica 38) in z njim naredimo nov objekt neveznih interakcij (vrstica 39). Postopek ustvarjanja potencialov in definiranja tipov je analogen prej opisanemu, s to razliko, da nastavimo odrez potenciala za grobozrnato območje `cutoff` na nič. Vse grobozrnate interakcije smo namreč že opisali s tabuliranim potencialom zgoraj. Pozoren bralec bo najbrž opazil, da so tipi ionov pri dodajanju grobozrnatih potencialov drugačni, kot kadar dodajamo atomistične potenciale. To je zato, ker med grobozrnatimi predstavitvami ionov vladajo drugačni elektrostatski potenciali, kakor smo že omenili v poglavju metodologije.

```
38 vl2 = espresso.VerletListAdress(system, cutoff=0, adrcut=rc+skin, dEx=
       ex_size, dHy=hy_size, adrCenter= box_cntr)
39 interGRF = espresso.interaction.VerletListAdressReactionFieldGeneralized(vl2
       , ftpl)
```

V elektrostatskih interakcijah nastopajo tudi vodiki, zato moramo definirati potenciale med vsemi možnimi kombinacijami tipov:

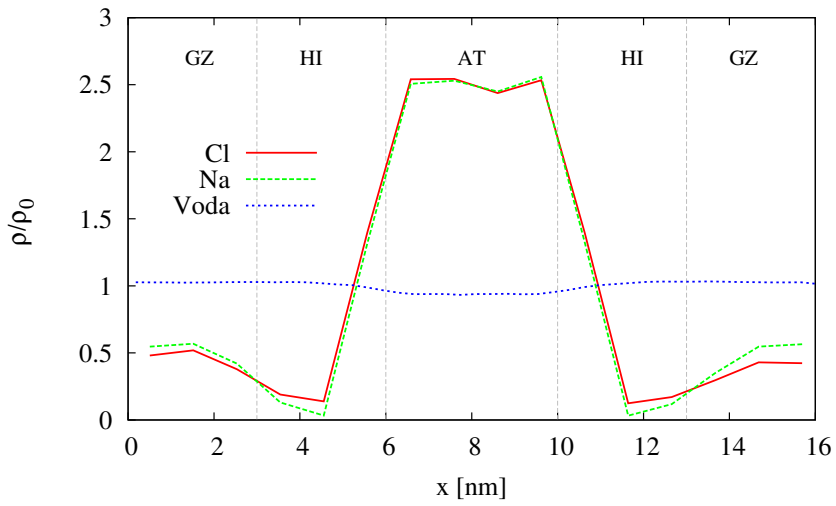
```
40 # definiramo AT potenciale
41 potOO    = espresso.interaction.ReactionFieldGeneralized(qq=eO*eO, kappa=
       kappa, epsilon1=e1, epsilon2=e2, cutoff=rc)
42 potOH    = espresso.interaction.ReactionFieldGeneralized(qq=eO*eH, kappa=
       kappa, epsilon1=e1, epsilon2=e2, cutoff=rc)
43 potHH    = espresso.interaction.ReactionFieldGeneralized(qq=eH*eH, kappa=
       kappa, epsilon1=e1, epsilon2=e2, cutoff=rc)
44 potNaNa  = espresso.interaction.ReactionFieldGeneralized(qq=eNa*eNa, kappa=
       kappa, epsilon1=e1, epsilon2=e2, cutoff=rc)
45 potNaCl  = espresso.interaction.ReactionFieldGeneralized(qq=eNa*eCl, kappa=
       kappa, epsilon1=e1, epsilon2=e2, cutoff=rc)
46 potNaO   = espresso.interaction.ReactionFieldGeneralized(qq=eNa*eO, kappa=
       kappa, epsilon1=e1, epsilon2=e2, cutoff=rc)
47 potNaH   = espresso.interaction.ReactionFieldGeneralized(qq=eNa*eH, kappa=
       kappa, epsilon1=e1, epsilon2=e2, cutoff=rc)
48 potClCl  = espresso.interaction.ReactionFieldGeneralized(qq=eCl*eCl, kappa=
       kappa, epsilon1=e1, epsilon2=e2, cutoff=rc)
49 potClO   = espresso.interaction.ReactionFieldGeneralized(qq=eCl*eO, kappa=
       kappa, epsilon1=e1, epsilon2=e2, cutoff=rc)
50 potClH   = espresso.interaction.ReactionFieldGeneralized(qq=eCl*eH, kappa=
       kappa, epsilon1=e1, epsilon2=e2, cutoff=rc)
```

Če bi bilo teh kombinacij preveč, da bi jih ročno definirali, bi seveda lahko naredili funkcijo, ki to naredi namesto nas.

```
51 # dodamo potenciale med AT delce
52 interGRF.setPotentialAT(type1=1, type2=1, potential=potOO)   #  O - O
53 interGRF.setPotentialAT(type1=1, type2=2, potential=potOH)   #  O - H
54 interGRF.setPotentialAT(type1=2, type2=2, potential=potHH)   #  H - H
55 interGRF.setPotentialAT(type1=4, type2=4, potential=potNaNa) # Na - Na
56 interGRF.setPotentialAT(type1=4, type2=6, potential=potNaCl) # Na - Cl
57 interGRF.setPotentialAT(type1=4, type2=1, potential=potNaO)  # Na - O
58 interGRF.setPotentialAT(type1=4, type2=2, potential=potNaH)  # Na - H
59 interGRF.setPotentialAT(type1=6, type2=6, potential=potClCl) # Cl - Cl
60 interGRF.setPotentialAT(type1=6, type2=1, potential=potClO)  # Cl - O
61 interGRF.setPotentialAT(type1=6, type2=2, potential=potClH)  # Cl - H
62
63 # dodamo interakcijo v sistem
64 system.addInteraction(interGRF)
```

Veznih interakcij znotraj molekul ne definiramo, saj smo se jim namenoma izognili z uporabo algoritma SETTLE.

Termodinamsko silo izračunamo z uporabo iterativnega postopka, kakor opisuje literatura [27, 104, 105]. Postopek bi v grobem opisali takole: najprej poženemo večskalno simulacijo brez termodinamske sile in po nekem izbranem številu korakov izračunamo normaliziran profil gostote. Po vsej verjetnosti ta ne bo raven in zato v hibridnem pasu (oz. nekoliko čez), izračunamo gradient, na podlagi katerega določimo popravek k termodinamski sili. Nato poženemo simulacijo z novo termodinamsko silo in postopek ponavljamo, dokler nismo zadovoljni z normaliziranim profilom gostote. Najprej iteriramo termodinamsko silo za molekule vode, dokler ne dosežemo zadovoljive porazdelitve gostote, nato pa to silo uporabimo kot izhodiščno termodinamsko silo ionov in iteriramo dalje samo termodinamsko silo ionov. Termodinamsko silo zapišemo v tabulirano datoteko, podobno kot efektivni potencial. Prvi stolpec je razdalja od središča atomističnega območja, drugi in tretji pa sta potencial in sila pri dani razdalji. Vmesne vrednosti določimo z interpolacijo.



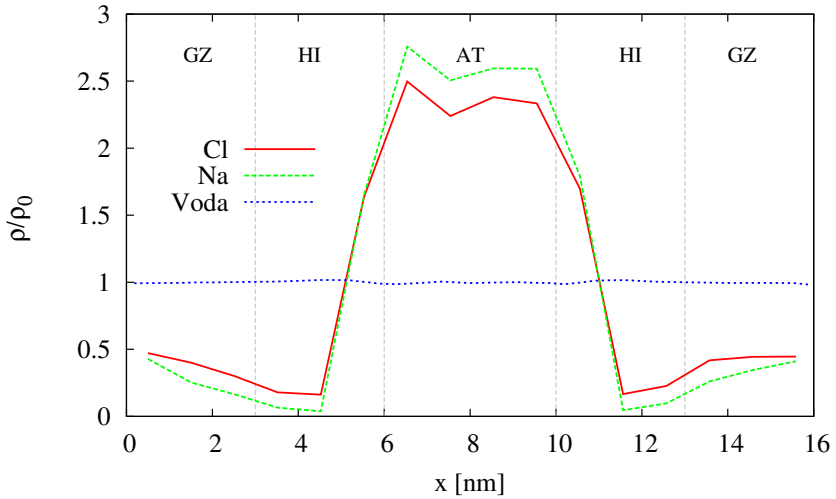
Slika 3.14: Normaliziran profil gostote sistema SPC s termodinamsko silo zgolj na molekulah vode. Normaliziran profil gostote vzdolž koordinate x sistema z molekulami vode SPC za težišča vod in ionov v simulaciji AdResS, kjer smo termodinamsko silo dodali samo na molekule vode. Navpične črte predstavljajo meje med območji različnih ločljivosti. Opazimo lahko, da je gostota natrijevih in klorovih ionov v atomističnem območju bistveno večja.

V našem primeru se brez termodinamske sile molekule vode in ioni pričnejo nabirati v atomističnem območju. Sliki 3.14 in 3.15 prikazujeta normaliziran profil gostote simulacije, pri kateri smo termodinamsko silo dodali samo na molekule vode.

Rezultati

Najprej izračunamo radialno porazdelitveno funkcijo med težišči molekul vode ter med težišči molekul vode in ionov. To storimo za atomistično, grobozrnato in večskalno simulacijo.

Radialna porazdelitvena funkcija opiše, kako so delci sistema v povprečju radialno porazdeljeni drug okrog drugega. Omogoča nam, da na učinkovit način opišemo povprečno strukturo neurejenih molekularnih sistemov, kot so tekočine. Radialno porazdelitveno



Slika 3.15: Normaliziran profil gostote sistema SPC/E s termodinamsko silo zgolj na molekulah vode. Normaliziran profil gostote vzdolž koordinate x sistema z molekulami vode SPC/E za težišča vod in ionov v simulaciji AdResS, kjer smo termodinamsko silo dodali samo na molekule vode. Navpične črte predstavljajo meje med območji različnih ločljivosti. Kakor pri SPC vodi, opazimo, da je gostota natrijevih in klorovih ionov v atomističnem območju bistveno večja.

funkcijo zgradimo tako, da izračunamo razdalje med vsemi pari delcev in jih združimo v histogram. Histogram normaliziramo glede na idealni plin, pri katerem so delci popolnoma nekorelirani. V bistvu nam funkcija poda verjetnost, da najdemo delec na neki razdalji od izbranega delca relativno na verjetnost v idealnem plinu.

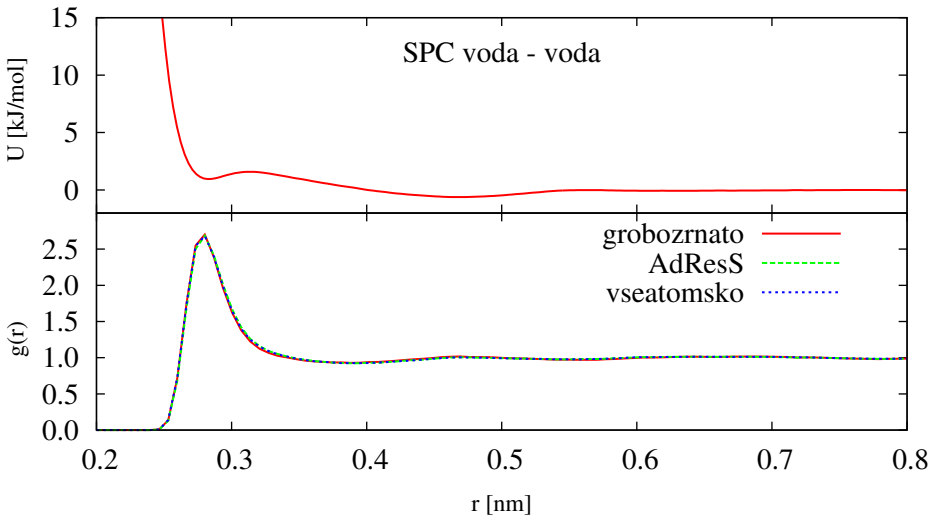
Za izračun radialne porazdelitvene funkcije vsakih 1000 korakov izpišemo koordinate atomov v datoteko. Naredimo skripto Python, ki za vsako datoteko prebere atome in se z dvojno `for` zanko sprehodi čez pare delcev ter izračuna razdalje z upoštevanjem konvencije minimalne slike (vrstice 14 do 20) (glej poglavje 3). Spremenljivka `sidehx` je polovica dolžine stranice škatle v smeri x, `sidex` pa celotna dolžina škatle v smeri x (analogno `sidehy`, `sidehz` in `sidey`, `sidez`). Spremenljivka `atoms` je seznam, ki hrani n-terice koordinat atomov. Razdalje razvrstimo v koše (vrstica 27) ter s štetjem zgradimo histogram (vrstica 29):

koda Python

```
1  ...
2      npart = len(atoms)
3      for i in range(npart):
4
5          xi = (atoms[i])[0]
6          yi = (atoms[i])[1]
7          zi = (atoms[i])[2]
8
9          for j in range(i+1, npart):
10             xx = xi - (atoms[j])[0]
11             yy = yi - (atoms[j])[1]
12             zz = zi - (atoms[j])[2]
```

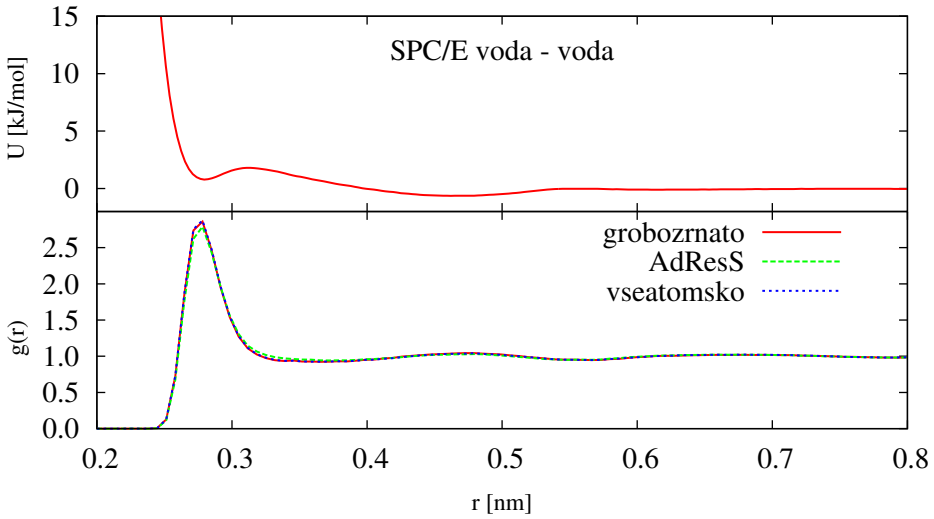
```
13              # minimalna slika
14              if  (xx < -sidehx): xx = xx + sidex
15              elif (xx > sidehx): xx = xx - sidex
16              if  (yy < -sidehy): yy = yy + sidey
17              elif (yy > sidehy): yy = yy - sidey
18              if  (zz < -sidehz): zz = zz + sidez
19              elif (zz > sidehz): zz = zz - sidez
20
21              # kvadrat razdalje med delcema i in j
22              rd  = xx * xx + yy * yy + zz * zz
23
24              if rd < cutsq:
25                  rij = sqrt(rd)
26                  bin = int(ceil(rij/dr)) # doloci kos v katerega pade delec
27                  if (bin <= maxbin):
28                      hist[bin] += 1
29  ...
```

Zahtevnost izračuna radialne porazdelitvene funkcije je, kakor za izračun sile, reda n^2. Da omilimo čas računanja, omejimo razdaljo, do katere računamo (vrstica 25). S tem se tudi izognemo dragemu izračunu funkcije korena (vrstica 26) po nepotrebnem. Poleg tega izračun pohitrimo tudi z uporabo vzporednega računanja. Datoteke razdelimo procesorjem in vsak izračuna histogram za datoteke, ki mu pripadajo. Kontrolni proces zbere rezultate, sešteje histograme in jih normalizira. Končne rezultate vpišemo v datoteko, kjer je prvi stolpec razdalja, drugi pa vrednost funkcije za to razdaljo.

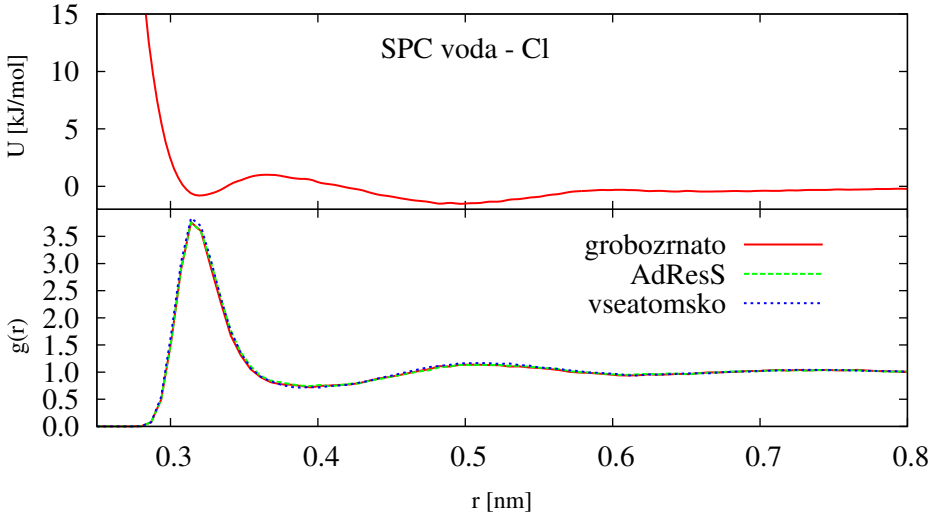


Slika 3.16: Radialne porazdelitvene funkcije in efektivni parski potencial sistema SPC za voda - voda. Radialne porazdelitvene funkcije med težišči molekul vode SPC za vse tri simulacije (spodaj) ter efektivni parski potencial (zgoraj), ki ga uporabimo za interakcije med grobozrnatimi predstavitvami molekul vode. Radialne porazdelitvene funkcije se ujemajo, kar pomeni, da dobro reproduciramo atomistično strukturo.

Na slikah 3.16 in 3.17 so prikazane radialne porazdelitvene funkcije med molekulami vode (spodaj) in efektivni parski potencial (zgoraj) za SPC in SPC/E sistema za vse tri

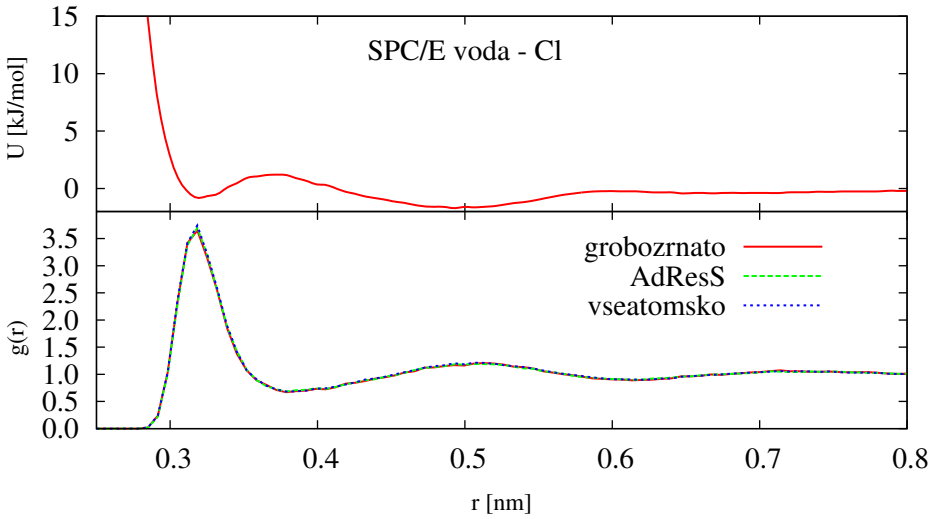


Slika 3.17: Radialne porazdelitvene funkcije in efektivni parski potencial sistema SPC/E za voda - voda. Radialne porazdelitvene funkcije med težišči molekul vode in efektivni parski potencial, kot na sliki 3.16, za sistem z molekulami vode SPC/E.

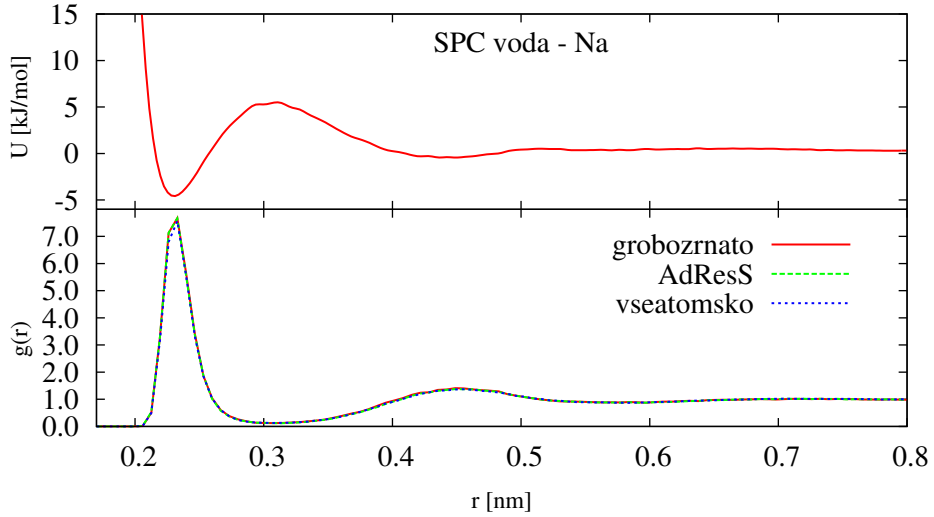


Slika 3.18: Radialne porazdelitvene funkcije in efektivni parski potencial sistema SPC za voda - klor. Radialne porazdelitvene funkcije med težišči molekul vode SPC in klorom za vse tri simulacije (spodaj) ter efektivni parski potencial (zgoraj), ki ga uporabimo za interakcije med grobozrnatimi predstavitvami molekul vode in klorom. Radialne porazdelitvene funkcije se ujemajo, kar pomeni, da dobro reproduciramo atomistično strukturo.

vrste simulacij (vseatomsko, grobozrnato in AdResS). Sliki 3.18 in 3.19 prikazujeta radialno porazdelitveno funkcijo med molekulami vode in klorom ter efektivni parski potencial za SPC in SPC/E sistema. Sliki 3.20 in 3.21 pa prikazujeta radialno porazdelitveno funkcijo med molekulami vode in natrijem ter efektivni parski potencial za SPC in SPC/E

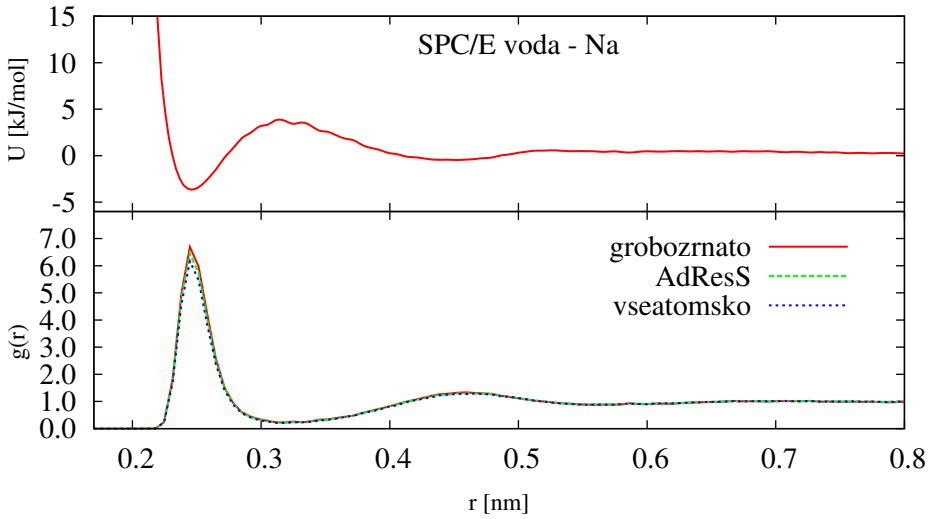


Slika 3.19: Radialne porazdelitvene funkcije in efektivni parski potencial sistema SPC/E za voda - klor. Radialne porazdelitvene funkcije med težišči molekul vode in klorom ter efektivni parski potencial, kot na sliki 3.18, za sistem z molekulami vode SPC/E.



Slika 3.20: Radialne porazdelitvene funkcije in efektivni parski potencial sistema SPC za voda - natrij. Radialne porazdelitvene funkcije med težišči molekul vode SPC in natrijem za vse tri simulacije (spodaj) ter efektivni parski potencial (zgoraj), ki ga uporabimo za interakcije med grobozrnatimi predstavitvami molekul vode in klorom. Radialne porazdelitvene funkcije se ujemajo, kar pomeni, da dobro reproduciramo atomistično strukturo.

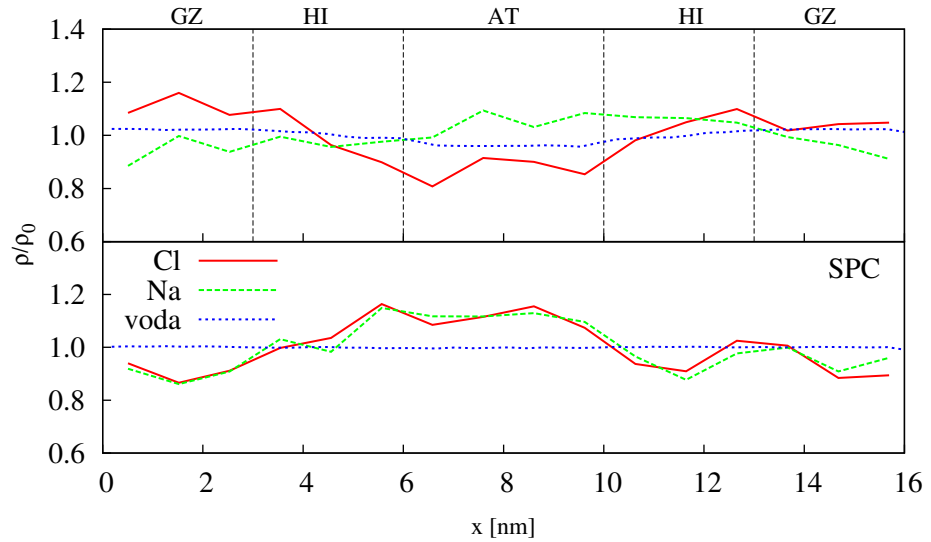
sistema. V vseatomskih in grobozrnatih simulacijah smo radialno porazdelitveno funkcijo izračunali in povprečili čez vse molekule v škatli, v simulacijah AdResS pa samo za molekule, ki se nahajajo v atomističnem območju. Na vseh slikah lahko opazimo, da efektivni potenciali izračunani z Boltzmannovo metodo, ki jih uporabimo za grobozrnate molekule,



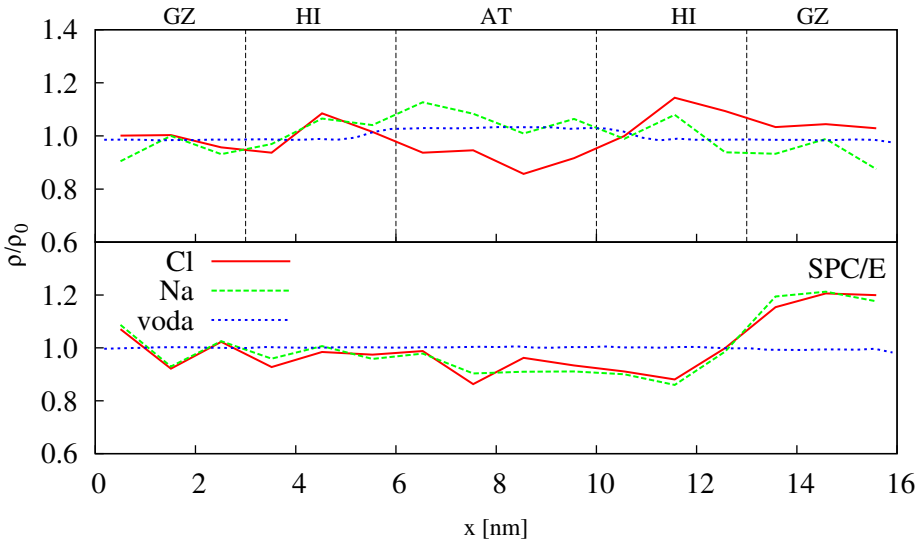
Slika 3.21: Radialne porazdelitvene funkcije in efektivni parski potencial sistema SPC/E za voda - natrij. Radialne porazdelitvene funkcije med težišči molekul vode in natrijem ter efektivni parski potencial, kot na sliki 3.20, za sistem z molekulami vode SPC/E.

dobro reproducirajo atomistično strukturo.

Želimo se tudi prepričati, da je gostota v simulacijah AdResS homogena preko različnih območij v simulacijski škatli. V ta namen izračunamo normalizirane profile gostote za težišča molekul vode in ionov vzdolž koordinate x.



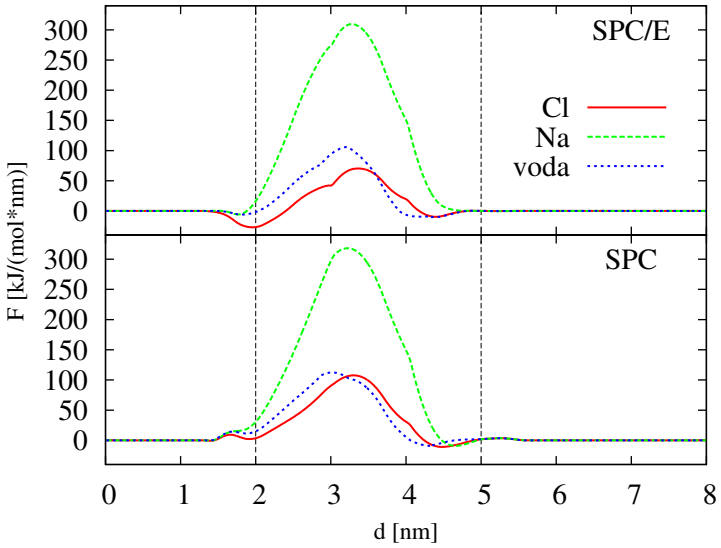
Slika 3.22: Profili normaliziranih gostot sistema SPC. Profili normaliziranih gostot vzdolž koordinate x sistema z molekulami vode SPC za težišča vod in ionov. V zgornjem delu so prikazani profili simulacija AdResS. Navpične črte ponazarjajo meje med posameznimi območji različnih ločljivosti. V spodnjem delu so prikazani profili vseatomske simulacije, kjer so prisotne fluktuacije enakega reda kot pri simulacijah AdResS.



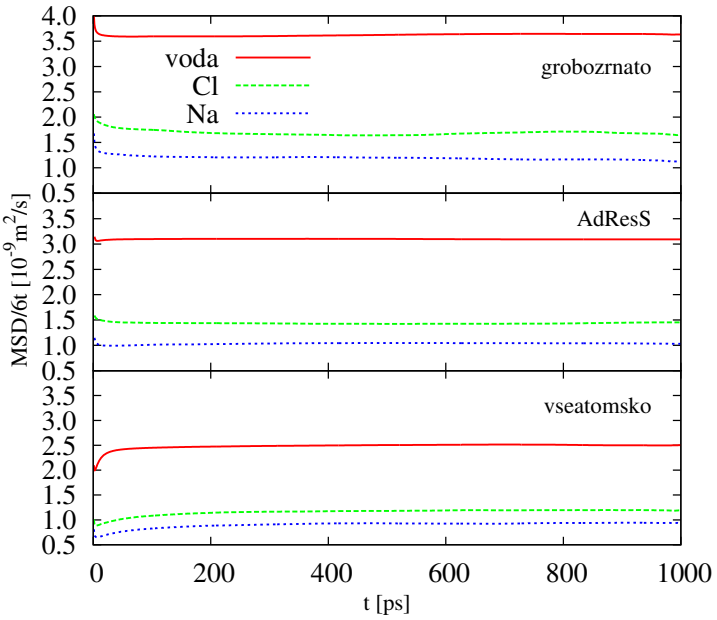
Slika 3.23: Profili normaliziranih gostot sistema SPC/E. Profili normaliziranih gostot vzdolž koordinate x sistema za težišča vod in ionov, kot na sliki 3.22, za sistem z molekulami vode SPC/E.

Podobno kakor za radialno porazdelitveno funkcijo, naredimo skripto Python, ki se sprehodi čez datoteke s koordinatami delcev ter s štetjem pojavitev delcev v koših izbrane širine vzdolž osi x zgradimo histogram. Gostoto v koših normaliziramo glede na gostoto celotne simulacijske škatle. Vrednost blizu ena pomeni, da gostota koša ne odstopa od povprečne gostote škatle. Zahtevnost izračuna je reda n. Ker pa pri dolgih simulacijah izpišemo veliko datotek, tudi tukaj, na enak način kot pri radialni porazdelitveni funkciji, uporabimo vzporedno računanje in datoteke razdelimo med razpoložljive procesorske enote. Vsak procesor izračuna histogram za svoje datoteke nato kontrolni proces sešteje rezultate in jih normalizira. Končni rezultat vpišemo v datoteko, kjer je prvi stolpec vrednost na osi x, drugi pa normalizirana gostota v tem delu škatle.

Izračun termodinamske sile za homogeno gostoto molekul vode poteka enako kot v sistemu s samo vodo [105], toda za ione je postopek zahtevnejši zaradi elektrostatike in slabše statistike. Kljub temu, da imamo relativno veliko simulacijsko škatlo, je število ionov še zmeraj majhno, zato je statistika za izračun profila normalizirane gostote slabša kot za vodo. S težavo se spopademo tako, da poganjamo daljše simulacije za posamezne iteracije. Ker med ioni vladajo tudi elektrostatske interakcije, z dodajanjem termodinamske sile na eno vrsto ionov vplivamo tudi na profil normalizirane gostote druge vrste ionov. Po drugi strani smo v postopku iteriranja termodinamske sile opazili, da so ioni z molekulami vode nepovezani. Sliki 3.22 in 3.23 prikazujeta profile normaliziranih gostot za vse tri vrste molekul za sistema z vodami SPC in SPC/E. Zgornji del slike prikazuje profil za simulacijo AdResS, spodnji pa za vseatomsko simulacijo. Gostota je prikazana vzdolž koordinate x, saj je to smer, v kateri se spreminja ločljivost molekul. Vidimo lahko, da ima profil normalizirane gostote vode le nekaj majhnih odstopanj (do 5%) od idealnega ravnega profila, medtem ko profil ionov ni tako gladek (odstopanja do 20%). Je pa na tem mestu potrebno izpostaviti, da se fluktuacije v gostoti ionov enakega reda pojavljajo tudi v vseatomskih simulacijah. Kot smo že zapisali, je razlog za to v slabi statistiki

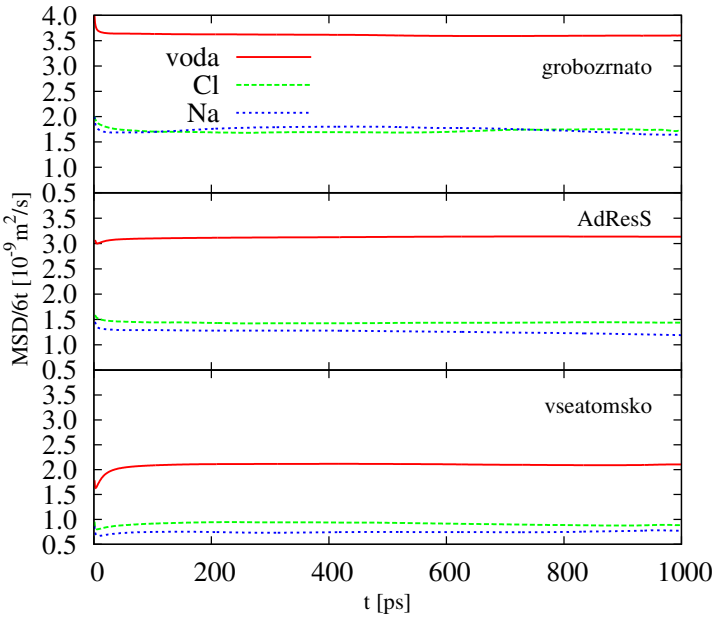


Slika 3.24: Termodinamske sile za pripadajoče vrste molekul. Os x ponazarja oddaljenost od središča simulacijske škatle v smeri x. Navpične črte ponazarjajo meje hibridnega območja. Sistem z molekulami vode SPC je spodaj, sistem z molekulami vode SPC/E pa zgoraj.



Slika 3.25: Povprečni kvadratni odmik za sistem SPC. Povprečni kvadratni odmiki preko časa za vseatomsko, AdResS in grobozrnato simulacijo za sistem z vodami SPC. Platoji definirajo pripadajoče difuzijske konstante.

zaradi majhnega števila ionov. Ne moremo pa trditi, da fluktuacije v simulaciji AdResS nastanejo zgolj zaradi slabe statistike. Lahko pa zaključimo, da z našo metodo reproduciramo profile gostote znotraj meja naših napak. Slika 3.24 prikazuje termodinamske sile za posamezen tip molekule za sistema z vodami SPC (spodaj) in SPC/E (zgoraj). V obeh



Slika 3.26: Povprečni kvadratni odmik za sistem SPC/E. Kot na sliki 3.25, le za sistem z vodami SPC/E.

sistemih sta termodinamski sili za vodo in klor približno podobne velikosti, medtem ko je termodinamska sila za natrijeve ione precej večja. Opazimo tudi, da termodinamska sila deluje v glavnem v hibridnem območju. Porazdelitev molekul vode je neodvisna od porazdelitve ionov in naša termodinamska sila vode je skoraj identična tisti, ki so jo izračunali za primer samo z vodo [105].

Dinamične lastnosti sistemov smo preverili z izračunom povprečnega kvadratnega odmika (*angl.* mean square displacement, MSD). Ta je merilo za povprečno razdaljo, ki jo molekula prepotuje. Za izračun smo izdelali skripto Python, ki izračuna kvadrat odmika za vsak delec v vsakem časovnem intervalu. Vrednost nato povprečimo čez vse delce. Če izrišemo povprečni kvadrat odmika, dobimo premico, naklon premice pa predstavlja difuzijsko konstanto [97, 107]. Na slikah 3.25 in 3.26 so prikazane difuzijske konstante pri vseatomskih, AdResS in grobozrnatih simulacijah za sistema SPC in SPC/E. Difuzijske konstante se ujemajo s tistimi navedenimi v literaturi [108]. Kot pričakovano, so difuzijske konstante grobozrnatih modelov višje od vseatomskih zaradi mehkejših interakcijskih potencialov. Ker se pri simulacijah AdResS delci gibljejo preko domen obeh ločljivosti, so vrednosti difuzijskih konstant med grobozrnatimi in vseatomskimi vrednostmi.

Tako razviti pristop je splošen in lahko ga uporabimo za katerokoli vrsto ionov in polje sil. Večskalni model je zlasti uporaben za simulacije biomolekularnih sistemov kot npr. simulacije proteinov in molekul DNK. Molekulo in njeno neposredno okolico lahko simuliramo v visoki ločljivosti, za preostalo topilo pa uporabimo nižjo ločljivost.

Pohitritev

Končno poglejmo še, kako je s pohitritvijo pri izvajanju na več procesorskih jedrih. V tabeli 3.3 so prikazani časi izvajanja simulacije z deset tisoč koraki. Simulacije smo poganjali

na računalniku z dvema šestjedrnima Intel Xeon E5-2630 procesorjema pri frekvenci 2,30 GHz z 15360 kB predpomnilnika. Operacijski sistem je Ubuntu 12.04.1 64-bit z linux jedrom 3.2. Koda smo prevajali z gcc prevajalnikom različice 4.6.3.

št. CPU	1	2	4	6	8	12
čas izvajanja	3540	1930	1220	770	665	540
pohitritev	1	1,8	3,4	4,6	5,3	6,6

Tabela 3.3: Pohitritev na več procesorskih jedrih. Časi izvajanja in pohitritev deset tisočih korakov simulacije na različnem številu procesorskih jeder. Čas je prikazan v sekundah. Pohitritev ni idealna, saj ne narašča linearno z naraščanjem števila procesorskih jeder.

Z naraščanjem števila procesorskih jeder se čas izvajanja krajša, vendar pohitritev ni idealna. Ne narašča namreč linearno z naraščanjem števila procesorskih jeder zaradi naraščanja količine dodatne komunikacije med procesorji. Kljub temu pa z vzporednim računanjem bistveno skrajšamo čas izvajanja simulacije.

4

Razvoj vmesnika za enostavno analizo kinetike encimov

V tem poglavju najprej predstavimo teoretično ozadje kinetičnih metod za preučevanje encimskih reakcij, nakar sledi opis uporabe ter predstavitev implementacije spletnega orodja ENZO [109] za hitro razločevanje med mnogimi reakcijskimi shemami encimov. Poglavje zaključimo s tremi primeri, s katerimi prikažemo zmogljivost razvitega orodja [110].

Teoretično ozadje

Sodobne biokemijske metode omogočajo proizvodnjo encimov v velikih količinah in variacijah, fizikalne metode pa nudijo dragocene podatke v zvezi z njihovo funkcijsko karakterizacijo. Najstarejše in najpogosteje uporabljane so kinetične metode, ki nadzirajo časovni potek encimske reakcije. Reakcijski mehanizem Michaelis-Menten za opis reakcij encimov z njihovimi substrati je bil predlagan že pred skoraj enim stoletjem [32]. Tudi dandanes je prvi kandidat, ki ga preizkusimo na vsakem novem preučevanem encimu.

Z uporabo novih eksperimentalnih tehnik so odkrili, da encimi njihovo aktivnost tipično prilagajajo glede na potrebe okolice. To vodi do znatnih odstopanj od klasičnih hiperboličnih kinetičnih metod [111]. Pogosto zato v kinetiki encimov poenostavimo model tako, da analiziramo zgolj začetne hitrosti namesto celotnega poteka preučevane reakcije. Pri tem seveda izgubimo veliko podatkov [31]. Poleg tega ne moremo izpeljati analitične rešitve navadnih diferencialnih enačb niti za enostavne Michaelis-Mentenove vmesne mehanizme [112], kaj šele za sisteme, ki vključujejo inhibitorje [113] ali pa so alosterično regulirani. Diferencialne enačbe, ki ustrezajo Michaelis-Mentenovi shemi:

$$E + S \underset{k_1}{\overset{k_0}{\rightleftarrows}} ES \overset{k_2}{\to} E + P$$

so:

$$\begin{aligned}
\frac{dE}{dt} &= k_0[E][S] + (k_1 + k_2)[ES], \\
\frac{dES}{dt} &= k_0[E][S] - (k_1 + k_2)[ES], \\
\frac{dS}{dt} &= -k_0[E][S] + k_1[ES], \\
\frac{dP}{dt} &= k_2[ES],
\end{aligned}$$

kjer E predstavlja prosti encim, S substrat, ES Michaelisov kompleks in P produkt, k_0 je konstanta drugega reda in k_1 ter k_2 sta konstanti prvega reda. Če določimo, da sta prosti encim E in kompleks ES v ravnovesju (določena s konstanto ravnovesja K), velja naslednja ohranitvena enačba:

$$[ES] = [E]_0[S]/(K + [S])$$

in zgornji sistem navadnih diferencialnih enačb se poenostavi v:

$$\frac{dS}{dt} = -k_2([E]_0[S]/(K + [S])),$$
$$\frac{dP}{dt} = k_2([E]_0[S]/(K + [S])).$$

Obstajajo različne numerične metode za reševanje takih diferencialnih enačb, ki vzpostavijo časovni potek vsake od sodelujočih vrst (glej npr. [114, 115]). Numerične metode moramo skrbno izbrati in implementirati za evaluacijo reakcijskih mehanizmov in pripadajočih parametrov v bioloških sistemih [33,116]. Poleg tega moramo metodo za reševanje povezati z algoritmom za analizo regresije [117]. Računalniških orodij, ki izpolnjujejo te pogoje ni veliko, pa še ta so uporabniku neprijazna. Poleg podrobnosti algoritma je največja nadloga priprava diferencialnih enačb, ki so specifične za vsak sistem in jih je potrebno znova pripravljati za vsak preučevani sistem.

Da bi rešili to težavo, smo izbrali primeren algoritem za reševanje enačb [33] in ga povezali z interaktivnim spletnim grafičnim vmesnikom. V nadaljevanju predstavimo prosto dostopno spletno orodje, ki smo ga poimenovali ENZO. ENZO samodejno ustvari diferencialne enačbe iz predlaganih shem reakcij encimske kinetike ter prileže koeficiente enačb koncentracijam krivulj poteka reakcije. To omogoča, da hitro ovrednotimo več različnih reakcijskih shem in najdemo tisto, ki je najprimernejša za dano encimsko reakcijo.

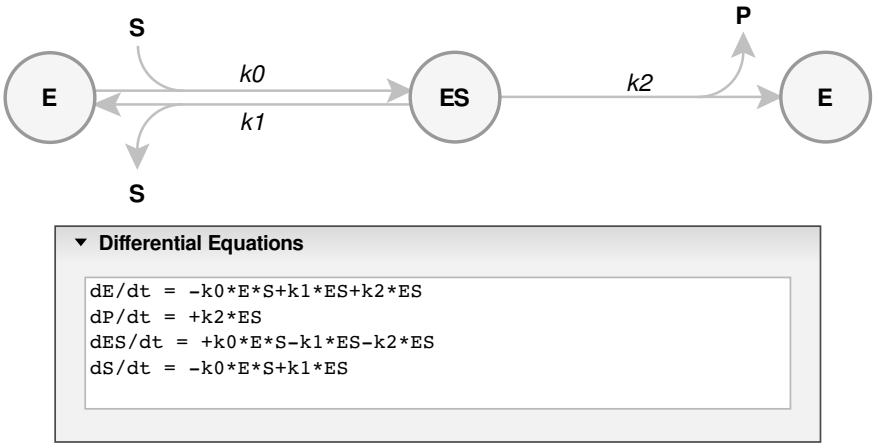
Metoda

Izmed biokemijskih metod kinetične metode najpogosteje uporabljamo za informacije v zvezi s hitrostjo ter zaporedjem, s katerim se encimi povežejo ali razvežejo z njihovimi součinkovinami ali drugimi encimi in kako predelajo svoje substrate. Na začetku predpostavimo, da obstaja več takih možnih poti. Vsako moramo posebej preizkusiti, tako da zgradimo pripadajoči kinetični model. Z njim določimo, kako se ujema z eksperimentalnimi podatki, ki nastopajo v obliki krivulj poteka reakcij. V ta namen v aplikaciji ENZO izdelamo uporabniški vmesnik, ki olajša nerodni in zamudni postopek pisanja enačb, tako da encimske reakcije prikažemo z grafičnimi diagrami. Diagrame izdelamo z miško. Na ta način lahko hitro preizkusimo in ovrednotimo različne kinetične modele. Model, ki se najbolj ujema z eksperimentalnimi podatki obdržimo, saj nam omogoča, da spoznamo korake dane encimske reakcije, kar posledično vodi do boljšega razumevanja funkcij encimov.

Spletno orodje ENZO

Za analizo časovno odvisnih kinetičnih podatkov moramo razviti bodisi analitične bodisi numerične rešitve za sistem navadnih diferencialnih enačb, ki opisujejo preučevani reakcijski mehanizem. V osnovi lahko za enačbe prvega reda ali psevdo prvega reda izpeljemo

analitično rešitev. V kolikor sledimo poteku reakcije, ki vključuje več vmesnih reaktantov do zaključka ali če je povezana z drugo reakcijo, postane trošenje reaktantov znatno in poenostavitve niso več na mestu. Da se izognemo ročni izpeljavi diferencialnih enačb, ENZO ustvari ustrezne enačbe iz shem, ki jih naredi uporabnik. Primer Michaelis-Mentenove reakcijske sheme in pripadajočih enačb je prikazan na sliki 4.1. Postopek in koda za izdelavo enačb iz sheme sta opisana v naslednjem podpoglavju.

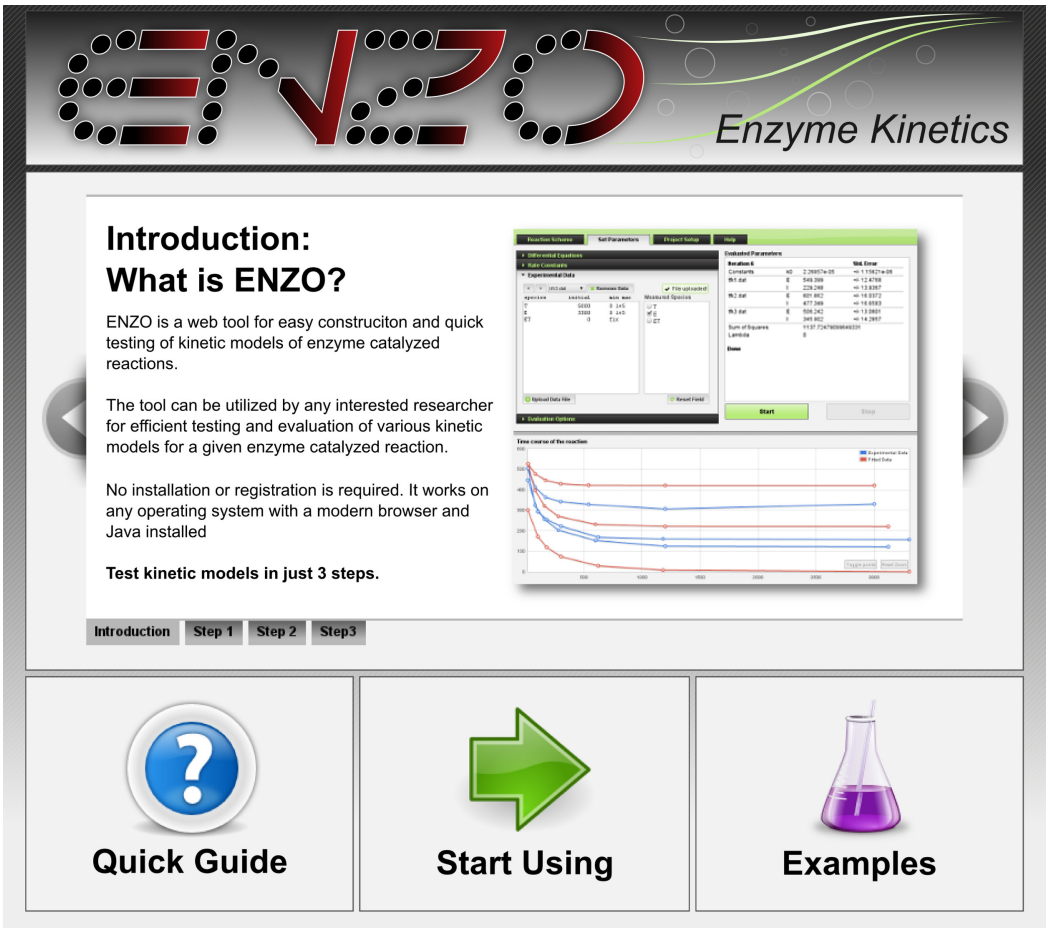


Slika 4.1: Michaelis-Mentnova reakcijska shema in pripadajoče diferencialne enačbe. Slika prikazuje Michaelis-Mentnovo reakcijsko shemo narisano s spletnim vmesnikom ENZO. Reakcije predstavimo z vozlišči in povezavami med njimi. Oznake znotraj vozlišč predstavljajo reakcijske vrste, oznake pod ali nad povezavami pa ponazarjajo kinetične konstante. E je prosti encim, S je substrat, ES je Michaelisov kompleks, P je produkt, *k0* je konstanta drugega reda, *k1* in *k2* pa sta konstanti prvega reda. Diferencialne enačbe v okvirčku pod sliko se samodejno ustvarijo.

Opis programske in strojne opreme

Z uporabo sodobnih spletnih tehnologij je možno izdelati spletne aplikacije, ki so po funkcionalnosti skoraj enakovredne namiznim aplikacijam. V primerjavi z namiznimi aplikacijami imajo pravzaprav kar nekaj prednosti, kot so neodvisnost od operacijskega sistema, ni potrebe po namestitvenem postopku, enostaven in hiter razvoj, ni problema različnih različic (vsi uporabniki imajo isto različico), enostavno kombiniranje različnih tehnologij in knjižnic, možnost razporejanja opravil na strežnik ali odjemalec, spremljanje dostopa uporabnikov, itd. Spletna aplikacija je nemudoma na razpolago vsakemu z dostopom do svetovnega spleta. ENZO je interaktivna spletna aplikacija, ki smo jo razvili z uporabo spletnih tehnologij.

Spletna stran ENZO je dostopna na naslovu http://enzo.cmm.ki.si (slika 4.2). Aplikacijo smo naredili po arhitekturi odjemalec-strežnik (slika 4.3). Koda odjemalca teče v spletnem brskalniku obiskovalca spletne strani in je odgovorna za prikaz uporabniškega vmesnika. Dinamične dele uporabniškega vmesnika smo napisali v programskem jeziku JavaScript z uporabo knjižice jQuery [118]. Del, v katerega narišemo reakcijsko shemo, smo napisali v programskem jeziku Java in teče kot applet (program Java znotraj brskalnika). Oboje vstavimo v kodo HTML, ki prav tako poskrbi za izris statičnih delov uporabniškega vmesnika. Za grafikon uporabljamo knjižico flot [119]. Z appletom komuniciramo preko javnih metod, ki jih kličemo z funkcijamo JavaScript. Spodnji primer kode prikazuje de-



Slika 4.2: Uvodna stran ENZO (http://enzo.cmm.ki.si). Sredinski del strani zaseda kratek opis orodja ter napotki za uporabo v treh korakih. Na dnu strani so povezave do hitrega vodiča (levo), ki uporabnika pelje skozi celoten postopek uporabe orodja ENZO na enostavnem primeru, povezava do pričetka uporabe (sredina) ter trije primeri na katerih predstavimo zmogljivosti orodja (desno).

lovanje gumba, ki ob pritisku napolni HTML tekstovno polje (*textarea*) z enačbami. Te ustvari applet. V HTML kodi imamo definirana elementa *textarea* in *button*:

koda html

```
1 <textarea id="equations"></textarea>
2 ...
3 <input type="button" id="show_parameters" value="Refresh">
```

Identifikator tekstovnega polja je `equations`, gumba pa `show_parameters`. Na gumb dodamo napis Refresh (osveži). S kodo JavaScript in jQuery, klik gumba povežemo s funkcijo:

koda javascript

```
$("#show_parameters").click(function(){
    $("equations").val(
            $("jgrafic").getEquations()
    );
});
```

Gumb določimo z identifikatorjem in nanj povežemo dogodek `.click()`. Ob kliku kličemo anonimno funkcijo, ki napolni vrednost tekstovnega polja. Le-tega prav tako določimo z identifikatorjem. Enačbe dobimo tako, da kličemo javno metodo appleta `getEquations()`. Ta vrne niz (enačbe), ki jih izdela na podlagi narisane reakcijske sheme. Niz se vpiše kot vrednost tekstovnega polja.

Kot smo že omenili, je reakcijska shema napisana v programskem jeziku Java. Elementi sheme (vozlišča, povezave in loki) so implementirani kot objekti z lastnostmi. Vozlišča npr. hranijo seznam pripadajočih vhodnih in izhodnih povezav, povezava ima podatek, kateri dve vozlišči povezuje, lok pa na kateri povezavi leži. Enačbe iz reakcijske sheme izdelamo z zanko preko elementov sheme. Spodnja koda prikazuje delovanje funkcije `getEquations()`:

koda java

```
public final String getEquations() {
    HashMap<String, String> map = new HashMap<String, String>();
```

Enačbe shranimo v razpršilno tabelo, kjer je ključ leva stran enačbe, vrednost pa desna stran.

```
    int nv = vertices.length; // stevilo vozlisc
    Object ov[] = vertices.elements;

    int na = arcs.length; //stevilo lokov
    Object oa[] = arcs.elements;
```

V spremenljivki `ov` in `oa` shranimo seznam vozlišč in lokov.

```
    // spremenljivke pri iteriranju skozi vozlisca, povezave in loke
    String enacba, tmp;
    int n;
    Object a[];
    Edge e;
    Arc arc;
```

S `for` zanko nato iteriramo skozi seznam vozlišč. Za vsako novo vozlišče nastane nova enačba. Če obstaja več vozlišč z enakim imenom, se bodo členi enačbe vpisali v obstoječo enačbo.

```
    //pojdi skozi vsa vozlisca
    for (int i = 0; i < nv; i++) {
        Vertex v = (Vertex) ov[i];

        //preberi trenutno vrednost enacbe iz mapa
        enacba = (String) map.get(v.label);
        if (enacba == null) enacba = "";
```

Za vsako izhodno povezavo vozlišča dodamo člen k obstoječi enačbi. Za to poskrbi funkcija `getEdgeEquations()`. Člen je sestavljen iz produkta imena konstante povezave s predznakom minus ter imenom vozlišča iz katerega izhaja povezava in vhodnih lokov. Podobno

naredimo za vhodne povezave vozlišča (vrstica 34), le da s predznakom plus. V vrstici 38 enačbo vpišemo v tabelo, kjer kot ključ uporabimo oznako vozlišča.

```
21          //pojdi skozi vse izhodne povezave
22          n = v.outedges.length;
23          a = v.outedges.elements;
24          for (int j = 0; j < n; j++) {
25              e = ((Edge) a[j]);
26              enacba = enacba+getEdgeEquations(e, na, oa, "-");
27          }
28
29          //pojdi skozi vse vhodne povezave
30          n = v.inedges.length;
31          a = v.inedges.elements;
32          for (int j = 0; j < n; j++) {
33              e = ((Edge) a[j]);
34              enacba = enacba+getEdgeEquations(e, na, oa, "+");
35          }
36
37          //vpisi enacbo v map
38          map.put(v.label, enacba);
39      }
```

Podobno kot z vozlišči, naredimo tudi z loki. Za vsak lok naredimo novo enačbo, če pa se isto ime loka pojavi večkrat, dopolnimo obstoječo enačbo. V vrsticah 52 in 56 kličemo funkcijo `getEdgeEquations()`, kjer za izhodni lok nastavimo predznak plus, za izhodni pa minus. V vrstici 60 enačbo vpišemo v tabelo.

```
41      //pojdi skozi vse loke
42      for (int l = 0; l < na; l++) {
43          arc = (Arc) oa[l];
44          e = arc.getEdge(); // povezava na kateri je lok
45
46          //preberi trenutno vrednost enacbe
47          enacba = (String) map.get(arc.label);
48          if (enacba == null) enacba = "";
49
50          // ce je izhodni lok
51          if (arc.getPuscica()) {
52              enacba = enacba+getEdgeEquations(e, na, oa, "+");
53          }
54          // ce je vhodni lok
55          else {
56              enacba = enacba+getEdgeEquations(e, na, oa, "-");
57          }
58
59          //dopisi zraven
60          map.put(arc.label, enacba);
61      }
```

Enačbe shranjene v tabeli združimo v niz `vseEnacbe` (vrstica 70) in ga vrnemo kot rezultat.

```
62      //izpisi enacbe iz mapa
63      Iterator<String> iterator = map.keySet().iterator();
64      String vseEnacbe = "";
65
66      while (iterator.hasNext()) {
67          String key = iterator.next().toString();
68          String value = map.get(key).toString();
```

```
69          if (key != " ") { //ne izpisi enacbe, ce ni oznake vozlisca
70              vseEnacbe = vseEnacbe + String.format("d%s/dt = %s\n", key,
                    value);
71          }
72      }
73      return vseEnacbe;
74  }
```

Funkcija `getEdgeEquations`, ki smo jo zgoraj že omenjali, izpiše člene enačbe za podano povezavo. Kot vhodne parametre dobi objekt povezave, število lokov, loke in predznak. Člen enačbe sestavimo iz predznaka in oznake povezave ter vozlišča, iz katerega izhaja povezava (vrstica 2). Nato iteriramo še čez vse loke povezave in dopišemo njihove oznake (vrstica 10). Vrnemo niz, ki predstavlja člen enačbe:

koda java

```
1  private String getEdgeEquations(Edge e, int na, Object oa[], String predznak
       ) {
2      String clen = predznak+e.label+"*"+e.getStart().label;
3
4      //pojdi skozi vse loke
5      for (int k = 0; k < na; k++) {
6          Arc a = (Arc) oa[k];
7          if (a.getEdge() == e) { // ce je lok na dani povezavi
8              // ce je vhodni
9              if(!a.getPuscica()) {
10                 clen = clen+"*"+a.label;
11             }
12         }
13     }
14     return clen;
15 }
```

Za komunikacijo med odjemalcem in strežnikom uporabimo ukaze knjižice jQuery, ki uporabljajo objekte XMLHttpRequest [120]. Ta tehnika, ki jo poznamo tudi pod imenom AJAX, omogoča spletnim brskalnikom, da zahtevajo podatke strežnika in jih prikažejo ali obdelajo, ne da bi bilo potrebno ponovno nalagati spletno stran. Pri tradicionalnem spletnem programiranju moramo počakati, da po izmenjavi podatkov s strežnikom, le-ta na novo naloži stran, preden jih lahko prikaže. V spletni aplikaciji ENZO je integracija s strežnikom neopazna, v smislu, da uporabnik ne more razbrati, kateri podatki so se izračunali na strežniku in kateri na odjemalcu. Zaradi tega imamo ob uporabi orodja ENZO občutek kakor, da uporabljamo samostojno namizno aplikacijo, hkrati pa obdržimo prednosti spletnih aplikacij.

Strežniško stran sestavlja računalnik, na katerem teče spletni strežnik Apache, razporejevalnik opravil in nekaj računalnikov, na katerih tečejo programi za reševanje enačb. Strežniški del aplikacije smo napisali v programskem jeziku PHP, program za reševanje enačb pa v Cju. Ob klicu funkcij PHP, brskalnik pošlje zahtevek na strežnik, ta izvede klicano funkcijo in nato vrne rezultat. Izvajamo klice AJAX.

Prikazali bomo enostaven primer brisanja datotek na strežniku. V brskalniku imamo v HTML kodi definiran spustni meni (vrstice 1 do 5 v spodnjem bloku kode) z vsemi naloženimi eksperimentalnimi datotekami (vidno npr. na sliki 4.6 v okvirčku *Experimental Data*, izbrana datoteka `tfk1.dat`). Poleg spustnega menija postavimo gumb za brisanje datotek (vrstica 7):

koda html

```
1 <select id="curve-files-select">
2     <option value="tfk1.dat">tfk1.dat</option>
3     <option value="tfk2.dat">tfk2.dat</option>
4     ...
5 </select>
6 ...
7 <input type="button" value="Remove Curve" id="remove-curve">
```

Gumbu določimo identifikator `remove-curve` in dodamo napis *Remove Curve* (odstrani krivuljo). Preko identifikatorja povežemo gumb z dogodkom klika miške. Ob kliku sprožimo anonimno funkcijo, ki prebere trenutno izbrano datoteko v meniju (vrstica 2). Ime datoteke se shrani v spremenljivko `file`. V naslednji vrstici naredimo POST zahtevek (klic AJAX), ki kliče datoteko `remove.php` na strežniku. Kot parameter podamo ime datoteke, ki jo želimo izbrisati:

koda javascript

```
1 $("remove-curve").click(function() {
2     var file = $("#curve-files-select").val();
3     $.post("remove.php", 'file='+file);
4     ...
5 }
```

Zgornja funkcija odstrani datoteko tudi iz spustnega menija in grafikona, vendar zgornja koda zaradi nazornosti tega ne prikazuje. V datoteki `remove.php` preberemo podano ime datoteke in jo izbrišemo na strežniku:

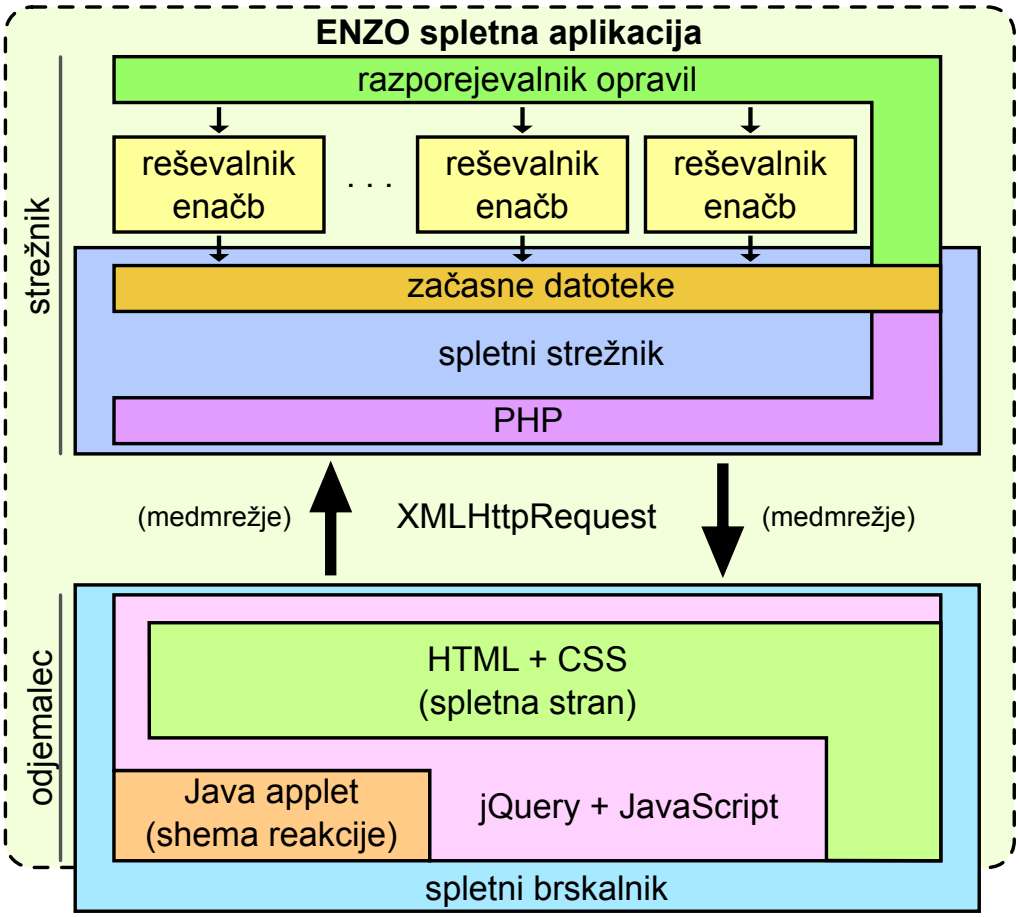
koda php

```
1 <?php
2     $datoteka = $_POST["file"];
3     unlink($datoteka);
4 ?>
```

Po podobnem postopku poteka tudi ostala komunikacija med odjemalcem in strežnikom. V mnogo primerih strežnik vrne tudi odgovor, ki ga brskalnik prikaže uporabniku ali kako drugače uporabi.

Poglejmo še, kako potekajo izračuni. Ko ustvarimo zahtevek po izračunu, ga PHP na strežniku prejme in zapiše v datoteko čakalne vrste. Razporejevalnik opravil (slika 4.3, živo zelene barve) periodično prebere datoteko in posreduje zahtevke računalnikom z reševalniki enačb (ista slika, rumene barve). Na vsakem računalniku teče zgolj en izračun hkrati. S tem zagotovimo, da ima vsak zahtevek zmeraj na razpolago vsa sredstva. Če so vsi računalniki zasedeni, zahtevek odjemalca čaka v vrsti, dokler se nek računalnik ne sprosti. Rezultat izračuna zapišemo v začasno datoteko na spletnem strežniku. Vsebino datoteke preberemo s funkcijo PHP in vrnemo brskalniku, kjer jo ustrezno prikažemo uporabniku. Namenoma smo se izognili uporabi podatkovnih baz, kajti gre za majhno količino začasnih podatkov, nad katerimi nikoli ne opravljamo poizvedb.

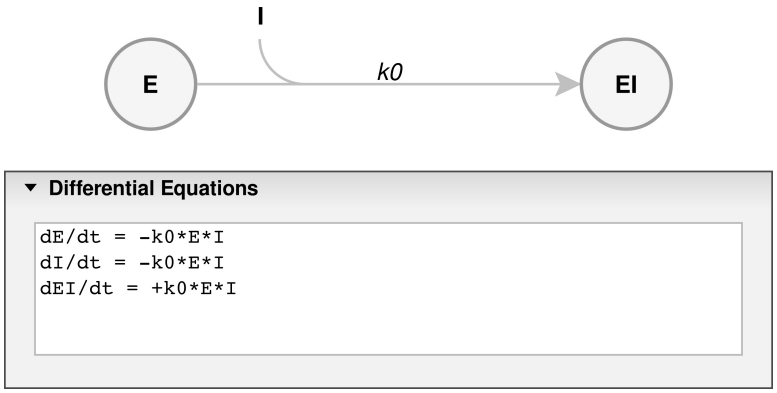
Ker spletne aplikacije ne omogočajo neposrednega branja in pisanja datotek po disku uporabnika (tj. obiskovalca spletne strani), moramo uporabiti brskalnikove zmožnosti nalaganja (*angl.* upload) in shranjevanja (*angl.* download) datotek, da zaobidemo to omejitev. Poglejmo si na primer, kako poteka prenos podatkov med odjemalcem in strežnikom,



Slika 4.3: Strežniški in odjemalčev del aplikacije ENZO. Na odjemalčevi strani uporabimo Javo v obliki appleta za del, v katerem narišemo reakcijsko shemo. Za dinamične dele uporabniškega vmesnika uporabimo jQuery. Oboje je postavljeno znotraj spletne strani HTML. S kaskadnimi slogi (CSS) definiramo stil in videz vmesnika. jQuery se prav tako uporablja za komunikacijo s strežnikom preko objektov XMLHttpRequest. Odjemalčev del aplikacije teče znotraj spletnega brskalnika obiskovalca spletne strani in je s strežnikom povezan preko medmrežja. Strežniški del aplikacije zajema spletni strežnik, na katerem teče programski jezik PHP. Ta prestreza odjemalčeva sporočila in je prav tako zadolžen za pisanje datotek. Ko se zgodi zahtevek po izračunu, se zapiše v datoteko čakalne vrste. Razporejevalnik opravil zahtevek dodeli reševalniku enačb. Ta ob zaključku zapiše rezultat na strežnik, kjer ga prebere PHP in vrne odjemalcu, da ga prikaže v brskalniku.

kadar uporabnik naloži eksperimentalne krivulje. V kodi HTML smo ustvarili obrazec za nalaganje datotek (`<input type="file">`). Brskalnik prikaže gumb, ki ob pritisku odpre okno za izbiro datotek na uporabnikovem računalniku. Po izbiri datoteke, se ta prenese na strežnik (z uporabo metode AJAX POST), kjer PHP prebere vsebino datoteke in jo vrne brskalniku nazaj v obliki JSON [121]. Brskalnik prebere strežnikov odgovor in ga prikaže v grafikonu. Vsebina datoteke torej „potuje“ od brskalnika na strežnik in nato nazaj v brskalnik. V našem primeru sicer tudi strežnik potrebuje datoteko v času prileganja krivulj, tako da bi jo vseeno morali prenesti. Če pa želimo pisati po disku uporabnika, si pomagamo tako, da naredimo povezavo na datoteko, ki se nahaja na strežniku. Ob kliku nanjo brskalnik odpre okno, s katerim uporabnik izbere lokacijo, kamor želi datoteko

shraniti. Šele prihajajoči standard HTML5 bo omogočil, da spletne aplikacije neposredno berejo in pišejo datoteke [122] po uporabnikovem disku.



Slika 4.4: Reakcijska shema titracije aktivnega mesta encima in pripadajoče diferencialne enačbe. E je encim butirilholinesteraze iz organizma *Torpedo californicae*, I je TMTFA inhibitor, EI je njun kompleks in *k0* je konstanta hitrosti drugega reda.

Vsak od računalnikov ENZO je opremljen z dvema štiri-jedrnima procesorjema. Prileganje krivulj teče vzporedno na vseh osmih jedrih enega računalnika, zato lahko v kratkem času prilegamo veliko krivulj. Zahteven izračun, kot je prikazan npr. v tretjem primeru (glej poglavje s primeri), potrebuje zgolj okoli ene minute, da se zaključi. Čas izračuna je seveda odvisen od tega, ali bo model konvergiral ali ne.

Našo aplikacijo smo si zamislili tako, da se med tekom prileganja krivulj na grafikonu izrisujejo vmesne krivulje. V primeru, da prileganje ne poteka v želeno smer, damo uporabniku možnost, da prileganje predčasno prekine. Zahtevek za pričetek prileganja pošljemo strežniku s klicem AJAX. Težava je, da le ta vrne rezultat šele, ko se izvajanje zaključi. Zaradi tega program na strežniku med prileganjem zapisuje vmesne rezultate v začasne datoteke. Hkrati z zahtevkom za pričetek izvajanja, sprožimo tudi zanko v brskalniku, ki periodično kliče funkcijo na strežniku. Le-ta bere vmesni rezultat iz datoteke, ter ga vrača brskalniku. Na ta način smo dosegli, da lahko uporabnik „v živo“ spremlja potek prileganja krivulj in trenutne parametre enačb.

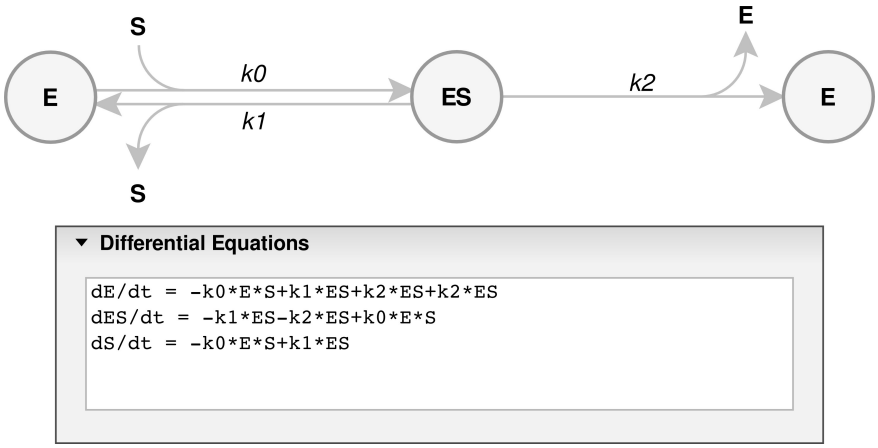
Vhodni podatki

ENZO ovrednoti predlagane kinetične modele tako, da prilega parametre pripadajočih diferencialnih enačb eksperimentalnim podatkom imenovanim *krivulje poteka*. Te predstavljajo spremembe v absorpciji, ki linearno korelira s koncentracijo zaznanih reakcijskih vrst (npr. substrati, produkti, intermediati) v diskretnih časovnih intervalih.

Eksperimentalne podatke podamo kot tekstovne datoteke z dvema stolpcema, ki sta ločena s presledkom ali tabulatorjem. V prvem stolpcu je zapisan čas meritve, v drugem stolpcu pa koncentracija reakcijske vrste v danem času. Vsaka vrstica torej predstavlja meritev v času in vsaka datoteka eno samo krivuljo poteka. V nadaljevanju opišemo potek priprave parametrov, ki definirajo začetne pogoje in približne konstante hitrosti. Podrobnejša navodila so na voljo na spletnih straneh aplikacije.

1. *Narišemo reakcijsko shemo.* Grafični vmesnik orodja ENZO omogoča, da z risbo narišemo tudi kompleksno reakcijsko shemo. V ta namen uporabimo zavihek *Re-*

action scheme na spletni strani, kjer reakcije predstavimo z vozlišči in povezavami med njimi. Oznake znotraj vozlišč predstavljajo reakcijske vrste, oznake pod ali nad povezavami pa ponazarjajo kinetične konstante. Vsaka shema določa unikatno množico enačb, ki se samodejno izdelajo iz ustvarjene sheme.



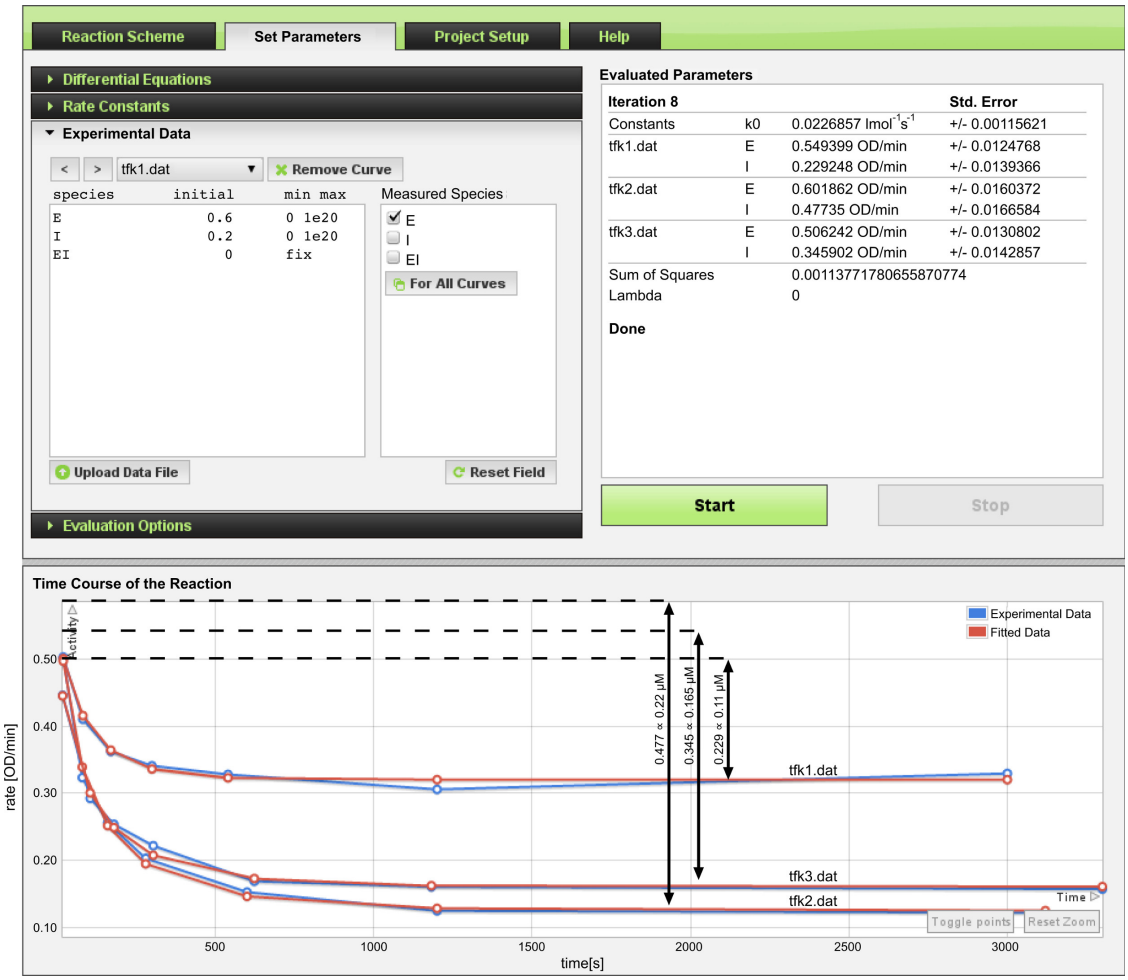
Slika 4.5: Reakcijska shema avtoaktivacije prokatepsina B in pripadajoče diferencialne enačbe. E označuje prosti encim, S encimski prekurzor, ES je njun kompleks.

2. *Nastavimo parametre.* Potem, ko smo izdelali reakcijsko shemo, se premaknemo v zavihek *Set Parameters*. Tam nastavimo začetne pogoje in približne vrednosti konstant hitrosti s smotrnimi mejami. Naložimo lahko več datotek krivulj poteka hkrati. Ker pa nekateri brskalniki tega ne omogočajo, je dodana tudi možnost, da naložimo stisnjeno .zip datoteko, ki vsebuje več datotek s krivuljami. Naložene datoteke se prikažejo kot modre krivulje v grafikonu *Time Course of the Reaction* (npr. slika 4.6 spodaj). Nato določimo začetne koncentracije in identitete vsake merjenih krivulj za vsako naloženo datoteko. Začetne vrednosti konstant hitrosti in koncentracije vrst lahko tudi nastavimo na konstantno vrednost, pri čemer bo strežnik izvedel zgolj prvi približek brez prileganja na predlagani kinetični model.

3. *Ovrednotenje parametrov.* Ko poženemo vrednotenje, se podatki posredujejo reševalniku enačb na strežniku, ki prične z iterativnim vrednotenjem in prikazuje rezultate za vsak korak iteracije. Možno je, da sočasno vrednotimo eksperimentalne krivulje poteka, ki opisujejo razne reakcijske intermediante, v kolikor so podatki na voljo.

Izhodni podatki

Izhodni podatki so sestavljeni iz množice prileganih kinetičnih parametrov, ki najbolje opisujejo ujemanje med teoretičnimi krivuljami predlaganega kinetičnega modela in eksperimentalnimi kinetičnimi podatki. Če model konvergira, smatramo, da je možen in ga potrdimo. V nasprotnem primeru moramo ponoviti izračun z drugačnimi začetnimi parametri in različnimi koncentracijami vrst. Če tudi to ne konvergira, popravimo predlagani kinetični model in ponovimo postopek.



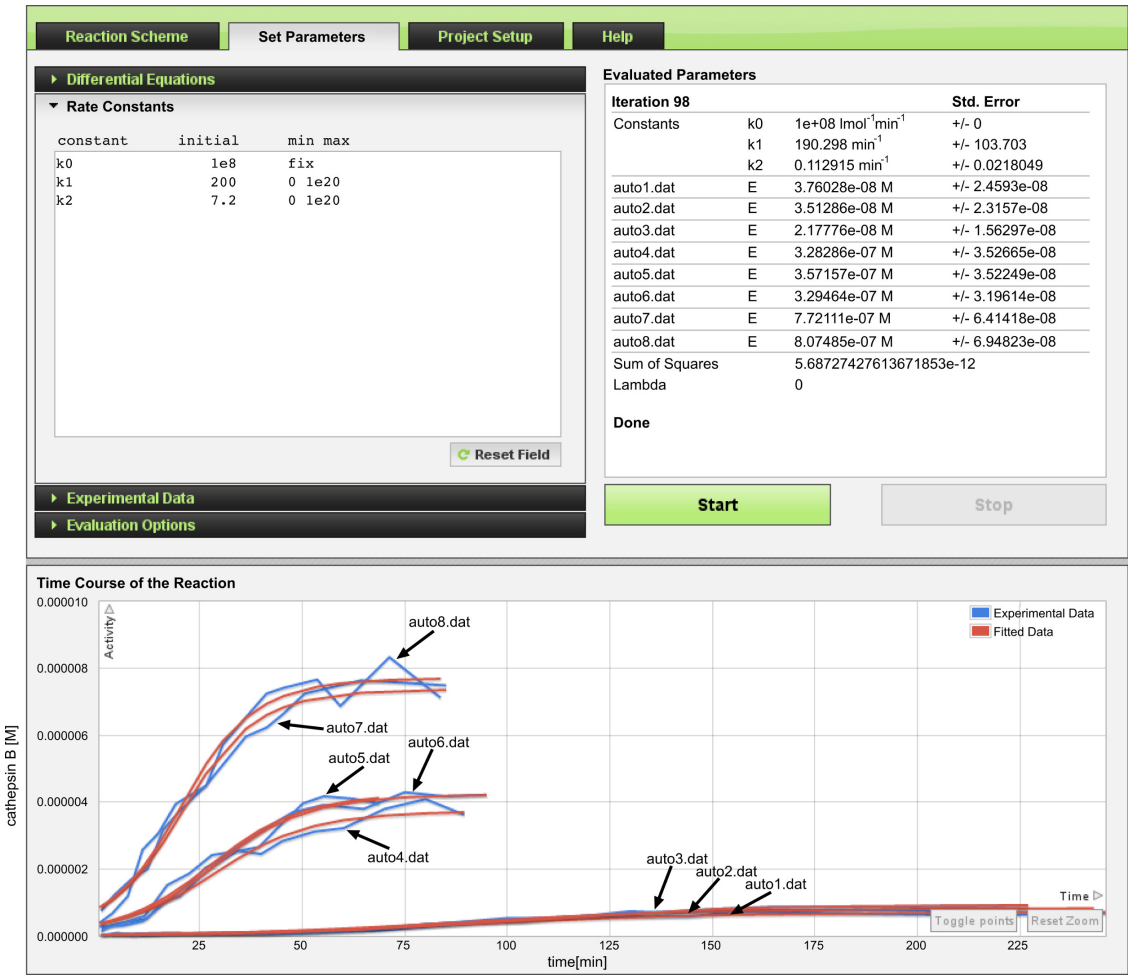
Slika 4.6: Konvergirani rezultati prileganja parametrov titracije aktivnega mesta encima. Enote na osi y so OD/min (optical density), na osi x pa sekunde. Prilegana konstanta hitrosti k_0 in začetne vrednosti E in I pri treh različnih koncentracijah I-ja so prikazane v okvirčku *Evaluated Parameters* (desno zgoraj). Eksperimentalne krivulje poteka so na grafikonu v modri barvi, prilegane pa v rdeči barvi. Puščice označujejo razliko med začetno vrednostjo in platojem.

Primeri

Zmogljivost orodja ENZO prikažemo na treh primerih, ki ponazarjajo tri prave encimsko kinetične scenarije. Tukaj jih zgolj na kratko predstavimo. Podrobneje so opisani v članku [109], lahko pa si jih tudi neposredno ogledamo in poženemo na spletni strani ENZO, v razdelku s primeri.

Vsak novi projekt uporabnika na strežniku dobi svoj imenik, v katerega se shranijo naložene datoteke eksperimentalnih meritev, narisana reakcijska shema in parametri za evaluacijo. Identifikator imenika je vpisan v naslovno vrstico brskalnika in se prenese na strežnik preko GET metode. To omogoča ustvarjanje zaznamkov v brskalniku in deljenje spletnih naslovov na poljuben projekt. Na enak način so shranjeni tudi primeri, ki jih tu predstavimo.

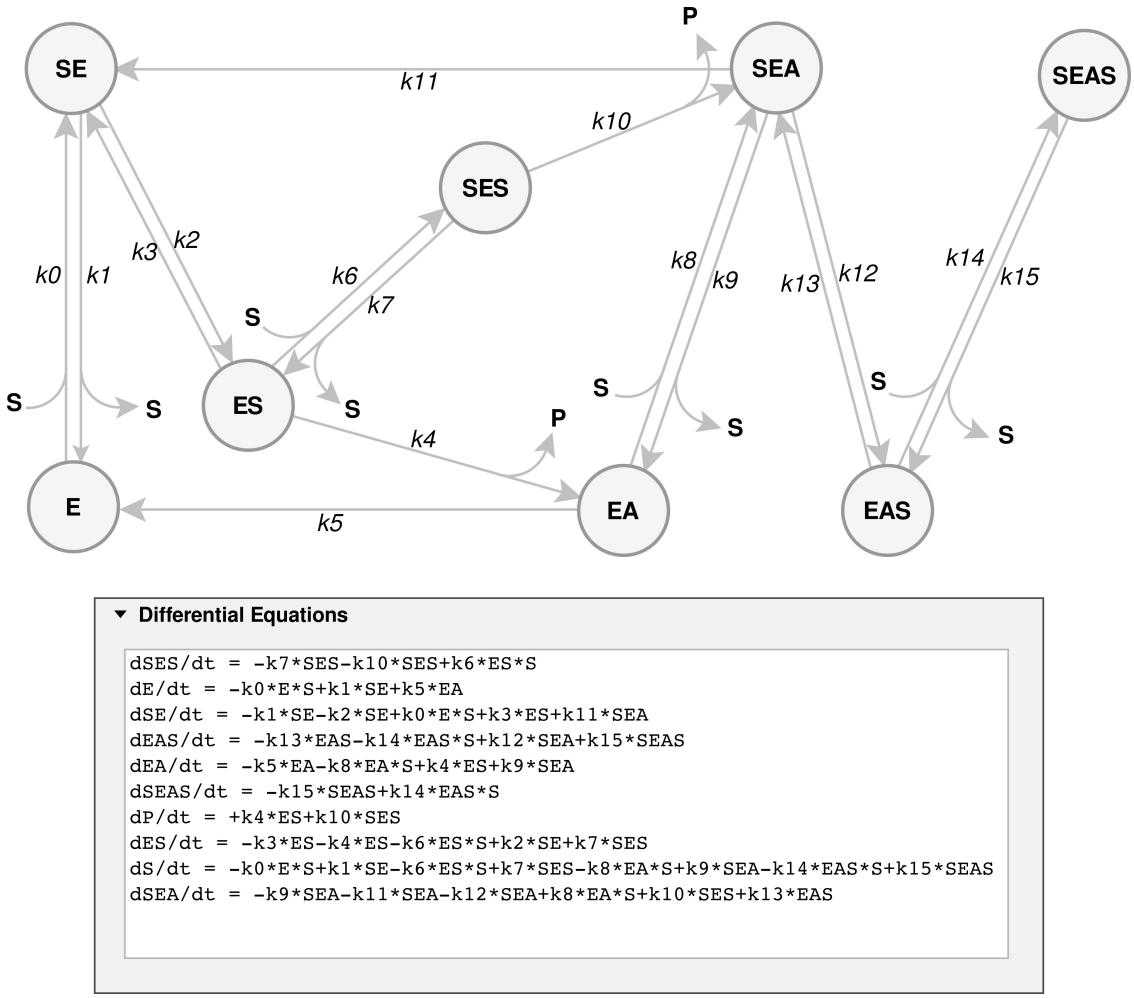
V prvem primeru obravnavamo titracijo aktivnega mesta encima butirilholinesteraze iz organizma *Torpedo californicae*. Reakcijska shema je prikazana na sliki 4.4, kjer je E



Slika 4.7: Konvergirani rezultati avtoaktivacije prokatepsina B. Na osi y je prikazana koncentracija katepsina B v enotah molarne koncentracije, os x pa prikazuje čas v minutah. Izračunane končne vrednosti konstant hitrosti in začetnih koncentracij aktivnih encimov so prikazane v okvirčku *Evaluated Parameters* (desno zgoraj) za vsako krivuljo poteka.

encim, I je TMTFA inhibitor, EI je njun kompleks in k_0 je konstanta hitrosti drugega reda. Konstanto hitrosti k_0 in začetne koncentracije E in I smo prilegali eksperimentalnim krivuljam poteka, ki smo jih naložili v treh datotekah. Na sliki 4.6 je prikazan zaključek izračuna, kjer začetne koncentracije encima E in inhibitorja I za krivulje poteka `tfk1.dat`, `tfk2.dat` in `tfk3.dat` prilegamo v intervalu $[0, 10^{20}]$ (v okvirčku *Experimental Data* je prikazana datoteka `tfk1.dat`). Začetna vrednost EI je nič in fiksirana s ključno besedo `fix`, kljukica pri črki E v okvirčku *Measured Species* označuje, da je E merjena količina. V grafikonu *Time Course of the Reaction* (v sliki 4.6 spodaj) je prikazan časovni potek dejavnosti ostanka. Vidimo, da se prilegane krivulje prekrivajo z eksperimentalnimi. V okvirčku *Evaluated Parameters* so prikazane izračunane vrednosti členov enačb za posamezne krivulje. Za izračun je bilo potrebnih osem iteracij ter približno 10 sekund.

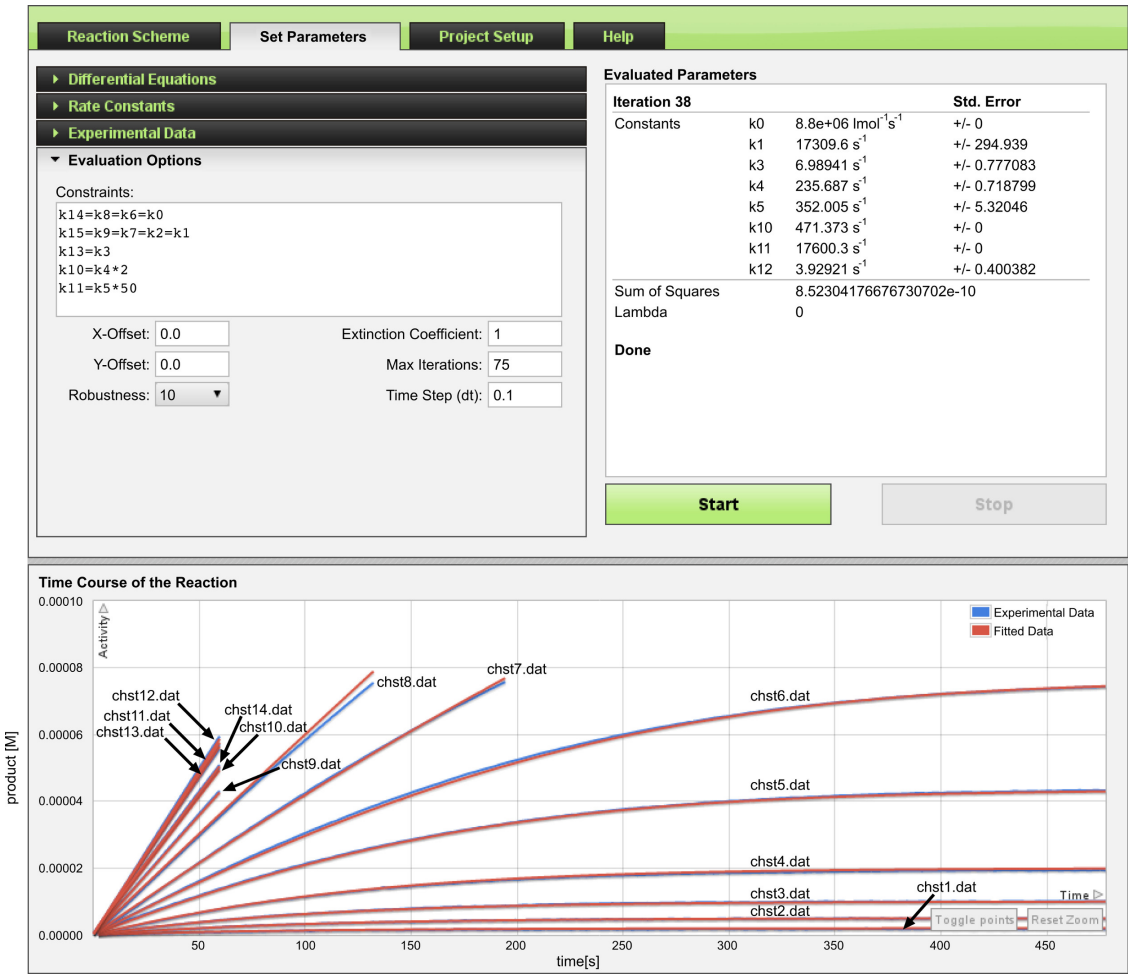
V drugem primeru obravnavamo avtoaktivacijo prokatepsina B. Reakcijska shema je prikazana na sliki 4.5, kjer E označuje prosti encim, S encimski prekurzor in ES je njun kompleks. Časovni potek takih avtoaktivacij se ponavadi začne s časovnim zamikom,



Slika 4.8: Reakcijska shema holinesteraze z butiltioholinom in pripadajoče diferencialne enačbe. Substrat butiltioholin vezan na periferno anionsko mesto je označen s črko S na levi strani imena (npr. SE, SES, SEA, SEAS). Kadar je vezan na katalitično anionsko mesto, je S postavljen na desno stran imena (npr. ES, SES, EAS, SEAS). Kovalentni acil-encim je označen z EA, P je prvi produkt (tioholin), ki se sprosti ob encimski acilaciji.

čemur sledi skok aktivnosti. Plato doseže, ko se vsi encimi pretvorijo v aktivno obliko. To je lepo vidno na sliki 4.7. Uporabili smo podatke objavljene v [123]. Naložili smo osem datotek z eksperimentalnimi podatki (`auto1.dat` do `auto8.dat`). Začetna vrednost ES je nič, E smo prilegali v intervalu $[0, 1]$. Začetna vrednost konstante hitrosti k_0 je nastavljena na difuzijsko hitrost z vrednostjo 10^8 M^{-1} min^{-1} in fiksirana. Začetni vrednosti k_1 in k_2 sta nastavljeni na 200 min^{-1} in 7.2 min^{-1} ter ju prilegamo v intervalu $[0, 10^{20}]$. Merjeni vrsti sta vsota prostega aktivnega encima E in disociativni kompleks ES. V okvirčku *Evaluated Parameters* na sliki 4.7 vidimo, da je bilo potrebnih 98 iteracij in pa izračunane vrednosti konstant k_1 in k_2 ter E za posamezne krivulje. Račun je potekal približno 45 sekund.

V tretjem primeru obravnavamo reakcijo holinesteraze z butiltioholinom. Reakcijska shema je prikazana na sliki 4.8. Gre za zelo zapleteno reakcijsko shemo, ki pa jo zlahka izdelamo z orodjem ENZO. Aktivno mesto holinesteraze je relativno veliko. Zato se mole-



Slika 4.9: Konvergirani rezultati holinesteraze z butiltioholinom. V okvirčku *Evaluated Parameters* so prikazane prilegane konstante hitrosti k_1, k_3, k_4, k_5, k_{10}, k_{11}, k_{12}. Vrednost k_0 fiksiramo na $8,8 \cdot 10^6$ s^{-1}. ENZO omogoča, da po želji omejimo nekatere hitrostne konstante in s tem poenostavimo prileganje ter pohitrimo postopek vrednotenja (vnosno polje *Constraints*). Za vse krivulje poteka fiksiramo začetne koncentracije SES, SE, EAS, ES, SEAS, P, ES, SEA na nič. E fiksiramo na 2.6 nM kar smo neodvisno določili s titracijo aktivnega mesta. Naložimo štirinajst krivulj poteka in sicer `chst1.dat` do `chst14.dat`. Na osi y je prikazan produkt v molarnih koncentracijah, v osi x pa čas v sekundah.

kula drugega substrata lahko veže na periferno anionsko mesto, preden se zgodi preobrat prvega substrata pri katalitičnem anionskem mestu. Na sliki 4.8 je substrat vezan na periferno anionsko mesto označen s črko S na levi strani črke E, npr. SE. Kadar je vezan na katalitično anionsko mesto, je S na desni strani npr. ES. Kovalentni acil-encim je označen z EA in P je prvi produkt, ki se sprosti pri encimski acilaciji. Sproščanje acilne skupine (A) ni označeno, ker ni pomembno. Kadar analiziramo zapletene reakcijske sheme, je veliko število relevantnih parametrov, ki jih je v praksi nemogoče ovrednotiti neodvisno. Težavo rešimo tako, da jih nekaj fiksiramo ali izenačimo med seboj. ENZO omogoča definiranje takih omejitev, kar je prikazano na sliki 4.9 v okvirčku *Evaluation Options*. S tem bistveno zmanjšamo število neznank. Take poenostavitve so nujne, kadar teoretično napovedani ali mehansko določeni intermediati povečajo število neznank preko

razpoložljivih eksperimentalnih podatkov. Na sliki 4.9 vidimo, da se eksperimentalne in prilegane krivulje prekrivajo. V okvirčku *Evaluated Parameters* so prikazane izračunane vrednosti konstant. Za izračun je bilo potrebnih 38 iteracij in približno minuto časa.

Z opisanimi primeri smo preizkusili naše računalniško orodje tako na osnovnih kot tudi na zahtevnih encimskih reakcijah. V vseh primerih se je ENZO izkazal za uporabno in nepogrešljivo orodje na področju encimske kinetike. Izdelava enačb z risanjem shem se odlično obnese. Ne samo, da prihranimo na času, zmanjšamo tudi možnost napak.

5

Diskusija

Programski paket ESPResSo++ s svojo kombinirano zasnovo dveh programskih jezikov hkrati omogoča visoko zmogljivost in enostavnost uporabe. Skriptni jezik Python ponuja fleksibilnost pri poganjanju simulacij molekulske dinamike, modularna zasnova jedra C++ pa poenostavi razširjanje funkcionalnosti programa. Veliko paketov za simulacijo molekulske dinamike izvira iz zgodnjega obdobja razvoja na področju računalniškega modeliranja in temelji na zastarelih programskih jezikih ali knjižnicah. ESPResSo++ z izbiro modernih programskih jezikov in konceptov ter z odprtokodno licenco predstavlja pomemben doprinos znanosti. Na široko odpira vrata raziskovalcem, da vgradijo ter preizkusijo nove algoritme in metode s področja računalništva ali računskih ved.

Nadaljnji doprinos znanosti predstavlja vgradnja metode AdResS v prostodostopni programski paket ESPResSo++, saj omogoči širšemu znanstvenemu krogu izvajanje večskalnih simulacij mehke snovi in molekularnih tekočin. Metodo prilagodljive ločljivosti AdResS smo v ESPResSo++ vgradili tako, da smo ustvarili nove razrede in v njih definirali za to potrebne metode. Implementacija temelji na dejstvu, da nam interpolacijske sheme za izračun sile ni potrebno računati preko celotnega razpona simulacijske škatle, ampak le na delcih znotraj hibridnega območja. V ta namen zgradimo tri Verletove sezname sosedov (za vsako območje svojega) in v vsakem od njih sile računamo drugače. Atomistične delce smo implementirali kot dodatno lastnost grobozrnatih delcev. Za preslikavo (oz. dostop) med grobozrnatim delcem in pripadajočimi atomističnimi delci uporabimo razpršilno tabelo. Ključ v tabeli je grobozrnati delec, vrednost pa vektor z atomi. Verletove sezname sosedov gradimo na podlagi grobozrnatih delcev. Za molekule, ki se nahajajo v hibridnem ali atomističnem območju, vstavimo v pripadajoči Verletov seznam tudi atomistične delce, saj jih potrebujemo v izračunu sile.

V fazi razvoja smo preizkusili tudi drugačen pristop, in sicer, da smo v Verletov seznam vključili zgolj grobozrnate delce in nato v samem izračunu sil po potrebi preko preslikave dostopali do atomističnih delcev. S psevdokodo bi izračun sile v hibridnem in atomističnem območju tega pristopa zapisali tako:

psevdokoda

```
for (i ... N):
    atomi1 = poglej_atome(i)
    for (sosed od i):
        sila_CG(i, sosed)
        atomi2 = poglej_atome(sosed)
        for (vse kombinacije atomov iz atomi1 in atomi2):
            sila_AT(atom1, atom2)
```

kjer je `N` število grobozrnatih delcev, `poglej_atome()` pa iz razpršilne tabele vrne atome

podanega grobozrnatega delca. Izvajanje v taki implementaciji je bilo do dvakrat počasnejše zaradi slabše izkoriščenega predpomnilnika procesorja. Procesor namreč zasega podatke iz pomnilnika v predpomnilnik v blokih in če so podatki v pomnilniku zapisani skupaj, se bodo tudi skupaj prenesli v predpomnilnik. Verletovi seznami temeljijo na vektorski podatkovni strukturi. To zagotavlja, da se podatki zapisani v njih v pomnilniku nahajajo skupaj. Zato v primeru uporabe vektorjev predpomnilnik večino časa že vsebuje delce, ki so naslednji na vrsti za izračun sile. V primeru uporabe razpršilne tabele znotraj izračuna sil, pa smo morali brati podatke atomističnih delcev iz različnih lokacij v pomnilniku in predpomnjenje ni bilo optimalno.

Pri računanju na več procesorskih enotah smo implementacijo zastavili tako, da se vsi atomi nekega grobozrnatega delca zmeraj nahajajo na istem procesorju. Ko se grobozrnati delec premakne iz enega procesorja na drugega, se sproži signal, ki zapakira pripadajoče atome v medpomnilnik skupaj z ostalimi podatki za prenos. S tem zmanjšamo število komunikacijskih klicev, ki so časovno ponavadi precej počasnejši od ostalih procesorskih operacij. Po prenosu na drugi procesor ponovno zgradimo kazalce v razpršilni tabeli preslikave.

Pri preizkušanju učinkovitosti implementacije smo dosegli opazno pohitritev v primerjavi z navadno atomistično simulacijo. Trajanje simulacije je seveda odvisno od izbora velikosti posameznih območij različnih ločljivosti in velikosti celotnega sistema. Večji kot je sistem, večja je pohitritev glede na vseatomsko simulacijo. Ker je gradnja Verletovega seznama v primeru simulacije AdResS nekoliko bolj zahtevna, se za njegovo gradnjo porabi več časa. Ker pa je izračun sil najzahtevnejši del integracijskega koraka, je povsem smotrno, da vložimo nekaj več dela v druge dele integracijskega algoritma, če nam le-to pohitri izračun sil.

Oboroženi z novo metodo, ki zdaj teče tudi na več procesorjih, smo izvedli simulacijo raztopine soli v večskalni tehniki. Zaradi nizke koncentracije ionov moramo uporabiti velik sistem, zaradi česar je nujno, da je koda učinkovito programirana. Šele na novo implementirana metoda AdResS v programskem paketu ESPResSo++, ki teče na več procesorskih jedrih, omogoča izvedbo te in podobnih biomolekularnih simulacij. Naša simulacijska škatla je dimenzij približno 16 nm x 4 nm x 4 nm in vsebuje okoli 8500 molekul vode ter skupno 320 ionov. Atomistično območje smo nastavili na sredini škatle, ki prehaja vzdolž osi x v grobozrnato ločljivost. Naša večskalna simulacija predstavlja pomemben doprinos znanosti, saj je raztopina soli v vodi zelo pomembno topilo v bioloških sistemih. Vpeljali smo splošno uporaben pristop, ki omogoča učinkovito obravnavo bioloških sistemov.

Pravilnost delovanja metode smo preverili z izračunom radialne porazdelitvene funkcije, normaliziranega profila gostote in difuzijske konstante. V primerjavi z referenčnim modelom ni bilo odstopanj. Pri računanju termodinamske sile smo morali zaradi nizkega števila ionov in posledično slabe statistike, poganjati iteracije z relativno visokim številom integracijskih korakov. V nadaljnjem razvoju programa ESPResSo++ bomo dodali tudi podporo sklopitvi grobozrnatega opisa s kontinuumom [124, 125] in s tem omogočili simulacije večskalnih tokov. Trendi na področju simulacije molekulske dinamike se nedvomno obračajo v smeri večskalnih simulacij, vendar se moramo zavedati, da priprava večskalne simulacije zahteva skrbno premišljeno in načrtovano pripravo.

Za študij kinetike encimov smo razvili spletno aplikacijo ENZO z uporabniku prijaznim uporabniškim vmesnikom. Orodje omogoča, da zapletene reakcijske mehanizme predstavimo s shemami, ki jih izdelamo z miško. Iz shem se samodejno izdelajo diferencialne enačbe, ki opisujejo potek reakcije. Orodje ENZO predstavlja doprinos znanosti iz po-

dročja molekularnega modeliranja, saj bistveno olajša študij kinetike encimov. Ročna izdelava diferencialnih enačb je sicer zelo nadležen postopek, zlasti ker moramo pri obravnavi upoštevati več možnih poti in za vsako od njih izdelati svoj kinetični model. Orodje smo uspešno preizkusili na treh realnih primerih, pri čemer je eden izmed njih zelo kompleksen.

Spletne aplikacije nudijo številne prednosti pred običajnimi namiznimi aplikacijami. Za nas je najpomembnejša takojšnja razpoložljivost aplikacije. Namestitveni postopek ni potreben, prav tako deluje v različnih operacijskih sistemih, ki so v raziskovalnih krogih lahko precej raznoliki. Spletna zasnova omogoča, da z deljenjem naslova URL (npr. preko e-pošte ali pa kar v citatu znanstvenega članka) enostavno pokažemo izdelani projekt.

S časom in z uporabo se na strežniku nabirajo eksperimentalni podatki in pripadajoče sheme ter parametri za prileganje. V nadaljnjem razvoju aplikacije bomo zato implementirali sistem, ki bo z uporabo umetne inteligence na podlagi zbranih podatkov uporabniku predlagal reakcijsko shemo in parametre, za katere je najverjetneje, da se bo model prilegal eksperimentalnim podatkom. Na ta način bomo še poenostavili uporabo orodja ENZO. Prav tako bomo v bližnji prihodnosti kodo Java za izdelavo reakcijske sheme, nadomestili s kodo JavaScript, da odpravimo potrebo po dodatnih vtičnikih brskalnika in s tem še razširimo krog uporabnikov. Del aplikacije, ki je trenutno narejen v programskem jeziku Java, pa bomo v nadaljevanju uporabili za izdelavo samostojne namizne aplikacije. Nekateri uporabniki so namreč izrazili to željo zaradi dela z zaupnimi podatki.

6

Zaključek

V okviru doktorske disertacije smo razvili računalniška orodja za molekularno modeliranje. Orodja pokrivajo dve področji molekularnega modeliranja, in sicer simulacijo molekulske dinamike ter področje kinetike encimskih reakcij.

Na kratko smo predstavili programski paket ESPResSo++ za simulacijo molekulske dinamike mehke snovi. Eno glavnih vodil pri razvoju programa je razširljivost, s čimer raziskovalcem omogočimo enostavno vgradnjo ter preizkušanje novih metod in algoritmov. ESPResSo++ je mlad program, ki uporablja moderne programerske trende. Programiran je v objektno usmerjeni tehniki s kombinacijo dveh programskih jezikov. Del, ki je izpostavljen uporabniku je napisan v programskem jeziku Python. S tem omogočimo upravljanje jedra, napisanega v programskem jeziku C++, z enostavnostjo skriptnih jezikov. Modularna zasnova jedra omogoča enostavno dodajanje novih algoritmov in spreminjanje obstoječih. Skriptni vmesnik nudi visoko stopnjo prožnosti pri pripravi ter usmerjanju poteka simulacije.

V programski paket ESPResSo++ smo vgradili vzporedno različico metode prilagodljive ločljivosti AdResS. Metoda prilagodljive ločljivosti omogoča združitev dveh opisov sistema različne ločljivosti v eni sami simulaciji, pri čemer dopušča prost prehod delcev med obema opisoma. Delcem se z gibanjem v simulacijski škatli zvezno spreminja opis v odvisnosti od položaja. Implementacija temelji na ideji, da vsak procesor zgradi tri Verletove sezname iz delcev, ki mu pripadajo: za atomistično, grobozrnato ter hibridno območje. V grobozrnatem in atomističnem območju sile računamo, kakor bi jih tudi sicer v monoskalni simulaciji, v hibridnem območju pa uporabimo interpolacijsko shemo sil z uporabo utežnostne funkcije. Metoda prilagodljive ločljivosti omogoča, da preučevani sistem poenostavimo do največje možne stopnje, hkrati pa zadržimo podrobnosti v izbranih predelih. Ker je v grobozrnatem območju število prostostnih stopenj in interakcij med delci bistveno manjše kot v atomističnem območju, lahko z metodo AdResS prihranimo dragocene procesorske vire. S tem omogočimo poganjanje simulacij, ki se raztezajo čez daljše krajevne in časovne skale.

Tako implementirano vzporedno metodo AdResS uporabimo za simulacijo raztopine soli v vodi, ki je poleg vode same eno najpomembnejših topil v simulacijah bioloških sistemov. Raztopina soli vpliva na lastnosti in funkcije bioloških molekul in je zato splošno prisotna v simulacijah molekulske dinamike. Z novo razvitim modelom soli v vodi smo vpeljali splošno uporaben pristop večskalne simulacije. Rezultati so potrdili pravilnost našega postopka. Metoda bo zlasti uporabna za nadaljnje študije, v katerih bomo v območje visoke ločljivosti vstavili biološko makromolekulo. Okoliško topilo, ki je iz vidika računalniške obravnave zelo zahtevno, pa se bo od visoke ločljivosti v okolici makromolekule raztezalo v območje nizke ločljivosti.

Za področje encimske kinetike smo razvili orodje ENZO s prijaznim uporabniškim vmesnikom, ki omogoča hitro izdelavo zapletenih kinetičnih enačb encimskih reakcij zgolj z uporabo miške. Orodje raziskovalcem omogoča hitro razlikovanje med mnogimi reakcijskimi shemami encimov, kar olajša določitev reakcijskega mehanizma, ki ustreza eksperimentalnim meritvam. Ker gre za aplikacijo izdelano z uporabo spletnih tehnologij, namestitveni postopek ni potreben. Orodje je na voljo vsakomur z delujočo povezavo v medmrežje na naslovu http://enzo.cmm.ki.si. ENZO teče na zmogljivih namenskih strežnikih, ki so optimizirani zgolj za to potrebo. Zaradi tega poteka reševanje enačb zelo hitro. Z novo razvitim orodjem za obravnavo encimske kinetike smo opravili študijo titracije aktivnega mesta encima butirilholinesteraze iz organizma Torpedo californicae, obravnavo avtoaktivacije prokatepsina B ter reakcijo holinesteraze z butiriltioholinom.

Novo razvita orodja znatno prispevajo k študiju kompleksnih molekularnih sistemov s pomočjo računalniškega modeliranja in s tem k razvoju različnih področij naravoslovnih ved. Razvita orodja so na voljo kot odprtokodni programi ali pa kot prostodostopno orodje na spletu. S tem želimo doseči kar najširšo možno publiko raziskovalcev.

Literatura

[1] A. R. Leach. *Molecular modelling: principles and applications.* Pearson Prentice Hall, London, 2001.

[2] M. P. Allen in D. J. Tildesley. *Computer simulation of liquids.* Clarendon Press, New York, 1989.

[3] M. E. Tuckerman. *Statistical mechanics: theory and molecular simulation.* Oxford graduate texts. Oxford University Press, Oxford, 2010.

[4] D. Janežič, M. Praprotnik in F. Merzel. Molecular dynamics integration and molecular vibrational theory. I. New symplectic integrators. *The Journal of Chemical Physics*, 122(17):174101, 2005.

[5] M. Praprotnik in D. Janežič. Molecular dynamics integration and molecular vibrational theory. II. Simulation of nonlinear molecules. *The Journal of Chemical Physics*, 122(17):174102, 2005.

[6] M. Praprotnik in D. Janežič. Molecular dynamics integration and molecular vibrational theory. III. The infrared spectrum of water. *The Journal of Chemical Physics*, 122(17):174103, 2005.

[7] H. Rafii-Tabar, L. Hua in M. Cross. A multi-scale atomistic continuum modelling of crack propagation in a two-dimensional macroscopic plate. *Journal of Physics: Condensed Matter*, 10:2375–2387, 1998.

[8] J. Q. Broughton, F. F. Abraham, N. Bernstein in E. Kaxiras. Concurrent coupling of length scales: methodology and application. *Physical Review B*, 60(4):2391, 1999.

[9] N. G. Hadjiconstantinou. Molecular dynamics simulation with reversible heat addition. *Physical Review E*, 59:R44–R47, 1999.

[10] J. A. Smirnova, L. V. Zhigilei in B. J. Garrison. A combined molecular dynamics and finite element method technique applied to laser induced pressure wave propagation. *Computer Physics Communications*, 118:11–16, 1999.

[11] J. Rottler, S. Barsky in M. O. Robbins. Cracks and crazes: on calculating the macroscopic fracture energy of glassy polymers from molecular simulations. *Physical Review Letters*, 89:148304, 2002.

[12] G. Csányi, T. Albaret, M. C. Payne in A. De Vita. "Learn on the fly": a hybrid classical and quantum-mechanical molecular dynamics simulation. *Physical Review Letters*, 93:175503, 2004.

[13] A. Heyden, H. Lin in D. G. Truhlar. Adaptive partitioning in combined quantum mechanical and molecular mechanical calculations of potential energy functions for

multiscale simulations. *The Journal of Physical Chemistry B*, 111(9):2231–2241, 2007.

[14] S. T. O'Connell in P. A. Thompson. Molecular dynamics - continuum hybrid computations: A tool for studying complex fluid flows. *Physical Review E*, 52:R5792–R5795, 1995.

[15] J. Li, D. Liao in S. Yip. Coupling continuum to molecular-dynamics simulation: Reflecting particle method and the field estimator. *Physical Review E*, 57:7259–7267, 1998.

[16] W. Cai, M. de Koning, V. V. Bulatov in S. Yip. Minimizing boundary reflections in coupled-domain simulations. *Physical Review Letters*, 85:3213–3216, 2000.

[17] E. G. Flekkøy, G. Wagner in J. Feder. Hybrid model for combined particle and continuum dynamics. *EPL (Europhysics Letters)*, 52(3):271, 2000.

[18] P. Koumoutsakos. Multiscale flow simulations using particles. *Annual Review of Fluid Mechanics*, 37:457–487, 2005.

[19] R. Delgado-Buscalioni, E. G. Flekkøy in P. V. Coveney. Fluctuations and continuity in particle-continuum hybrid simulations of unsteady flows based on flux-exchange. *EPL (Europhysics Letters)*, 69(6):959, 2005.

[20] E. G. Flekkøy, R. Delgado-Buscalioni in P. V. Coveney. Flux boundary conditions in particle simulations. *Physical Review E*, 72:026703, 2005.

[21] X. Nie, M. O. Robbins in S. Chen. Resolving singular forces in cavity flow: multiscale modeling from atomic to millimeter scales. *Physical Review Letters*, 96:134501, 2006.

[22] R. Delgado-Buscalioni in P. V. Coveney. Structure of a tethered polymer under flow using molecular dynamics and hybrid molecular-continuum simulations. *Physica A: Statistical Mechanics and its Applications*, 362(1):30–35, 2006.

[23] A. Dupuis, E. M. Kotsalis in P. Koumoutsakos. Coupling lattice boltzmann and molecular dynamics models for dense fluids. *Physical Review E*, 75:046704, 2007.

[24] G. De Fabritiis, R. Delgado-Buscalioni in P. V. Coveney. Multiscale modeling of liquids with molecular specificity. *Physical Review Letters*, 97:134501, 2006.

[25] M. Praprotnik, L. D. Site in K. Kremer. Multiscale simulation of soft matter: from scale bridging to adaptive resolution. *Annual Review of Physical Chemistry*, 59(1):545–571, 2008.

[26] M. Praprotnik, L. D. Site in K. Kremer. Adaptive resolution molecular-dynamics simulation: changing the degrees of freedom on the fly. *The Journal of Chemical Physics*, 123(22):224106, 2005.

[27] M. Praprotnik, S. Poblete in K. Kremer. Statistical physics problems in adaptive resolution computer simulations of complex fluids. *Journal of Statistical Physics*, 145:946–966, 2011.

[28] H. Limbach, A. Arnold, B. Mann in C. Holm. ESPResSo - an extensible simulation package for research on soft matter systems. *Computer Physics Communications*, 174(9):704–727, 2006.

[29] J. D. Halverson, T. Brandes, O. Lenz, A. Arnold, **S. Bevc**, V. Starchenko, K. Kremer, T. Stuehn in D. Reith. **ESPResSo++: a modern multiscale simulation package for soft matter systems**. *Computer Physics Communications*, 184(4):1129 – 1149, 2013.

[30] I. G. Tironi, R. Sperb, P. E. Smith in W. F. van Gunsteren. A generalized reaction field method for molecular dynamics simulations. *The Journal of Chemical Physics*, 102(13):5451–5459, 1995.

[31] H. Lineweaver in D. Burk. The determination of enzyme dissociation constants. *Journal of the American Chemical Society*, 56(3):658–666, 1934.

[32] L. Michaelis in M. L. Menten. Die kinetik der invertinwirkung. *Biochem Z*, 49:333–369, 1913.

[33] J. Stojan. Analysis of progress curves in an acetylcholinesterase reaction: a numerical integration treatment. *Journal of Chemical Information and Computer Sciences*, 37(6):1025–1027, 1997.

[34] R. B. Corey in L. Pauling. Molecular models of amino acids, peptides, and proteins. *Review of Scientific Instruments*, 24(8):621–627, 1953.

[35] J. Ševčík, L. Urbanikova, Z. Dauter in K. S. Wilson. Recognition of RNase Sa by the inhibitor Barstar: structure of the complex at 1.7Å resolution. *Acta Crystallographica Section D*, 54(5):954–963, 1998.

[36] A. Gillet, M. Sanner, D. Stoffler in A. Olson. Tangible interfaces for structural molecular biology. *Structure*, 13(3):483 – 491, 2005.

[37] G. T. Johnson, L. Autin, D. S. Goodsell, M. F. Sanner in A. J. Olson. ePMV embeds molecular modeling into professional animation software environments. *Structure*, 19(3):293 – 303, 2011.

[38] H. Goldstein. *Classical mechanics*. Addison-Wesley series in physics. Addison-Wesley Publishing Company, Boston, 1980.

[39] N. Attig, K. Binder, H. Grubmüller in K. D. Kremer, uredniki. *Computational soft matter: from synthetic polymers to proteins; NIC winter school, 29 February - 6 March 2004, Lecture notes*. John von Neumann Institute for Computing, Jülich, 2004, 1-28.

[40] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan in M. Karplus. Charmm: a program for macromolecular energy, minimization, and dynamics calculations. *Journal of Computational Chemistry*, 4(2):187–217, 1983.

[41] A. D. MacKerell, D. Bashford, M. Bellott, R. L. Dunbrack, J. D. Evanseck, M. J. Field et al. All-atom empirical potential for molecular modeling and dynamics studies of proteins. *The Journal of Physical Chemistry B*, 102(18):3586–3616, 1998.

[42] A. D. MacKerell, N. Banavali in N. Foloppe. Development and current status of the charmm force field for nucleic acids. *Biopolymers*, 56(4):257–265, 2000.

[43] W. D. Cornell, P. Cieplak, C. I. Bayly, I. R. Gould, K. M. Merz, D. M. Ferguson et al. A second generation force field for the simulation of proteins, nucleic acids, and organic molecules. *Journal of the American Chemical Society*, 117(19):5179–5197, 1995.

[44] W. F. van Gunsteren in H. J. C. Berendsen. Groningen molecular simulation (gromos) library manual. *Groningen molecular simulation (GROMOS) library manual*, (8):1–221, 1987.

[45] R. Trobec, I. Jerebic in D. Janežič. Parallel algorithm for molecular dynamics integration. *Parallel Computing*, 19(9):1029 – 1039, 1993.

[46] I. Foster. *Designing and building parallel programs: concepts and tools for parallel software engineering*. Parallel programming / scientific computing. Addison Wesley Publishing Company Incorporated, Boston, 1995.

[47] U. Borštnik, M. Hodošček in D. Janežič. Improving the performance of molecular dynamics simulations on parallel clusters. *Journal of Chemical Information and Computer Sciences*, 44(2):359–364, 2004.

[48] B. Hess, C. Kutzner, D. van der Spoel in E. Lindahl. Gromacs 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation. *Journal of Chemical Theory and Computation*, 4:435–447, 2008.

[49] S. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, 117, 1995.

[50] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa et al. Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry*, 26:1781–1802, 2005.

[51] R. Salomon-Ferrer, D. A. Case in R. C. Walker. An overview of the Amber biomolecular simulation package. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 3(2):198–210, 2013.

[52] W. Smith in I. T. Todorov. A short description of DL_POLY. *Molecular Simulation*, 32:935–943, 2006.

[53] B. R. Brooks, C. L. B. III, A. D. Mackerell, L. Nilsson, R. J. Petrella, B. Roux et al. Charmm: The biomolecular simulation program. *Journal of Computational Chemistry*, 30:1545–1615, 2009.

[54] URL: http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html, The MPI forum, MPI: a message passing interface. Dostopno na spletu: 14. 6. 2013.

[55] URL: http://www.espresso-pp.de/Developer_Team.html/. ESPResSo++ developer team. Dostopno na spletu: 14. 6. 2013.

[56] URL: http://wiki.python.org/moin/PythonSpeed. Dokumentacijca Python, hitrost. Dostopno na spletu: 25. 8. 2013.

[57] URL: http://www.python.org/doc/essays/comparisons.html. Dokumentacijca Python, primerjava z ostalimi jeziki. Dostopno na spletu: 25. 8. 2013.

[58] J. Mitchell. *Concepts in programming languages*. Cambridge University Press, Cambridge, 2003.

[59] M. Abadi in L. Cardelli. *A theory of objects*. Springer, New York, 1996.

[60] URL: http://en.wikipedia.org/wiki/Interpreted_language/. Wikipedia, interpretiran programski jezik. Dostopno na spletu: 14. 6. 2013.

[61] URL: http://en.wikipedia.org/wiki/Bytecode/. Wikipedia, bytecode. Dostopno na spletu: 14. 6. 2013.

[62] URL: http://en.wikipedia.org/wiki/Java_bytecode/. Wikipedia, Java bytecode. Dostopno na spletu: 14. 6. 2013.

[63] K.-Y. Chen, J. Chang in T.-W. Hou. Multithreading in java: performance and scalability on multicore systems. *Computers, IEEE Transactions on*, 60(11):1521–1534, 2011.

[64] Y. Zhao, J. Shi, K. Zheng, H. Wang, H. Lin in L. Shao. Allocation wall: a limiting factor of java applications on emerging multi-core platforms. *SIGPLAN Notices*, 44(10):361–376, 2009.

[65] D. Kurzyniec in V. Sunderam. Efficient cooperation between java and native codes – JNI performance benchmark. V *Proceedings of the 2001 International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, 25. – 28. junij 2001.

[66] B. Amedro, V. Bodnartchouk, D. Caromel, C. Delbe, F. Huet in L. Taboada, Guillermo. Current state of Java for HPC. Technical Report RT-0353, INRIA, Rocquencourt, 2008.

[67] M. Hertz in E. D. Berger. Quantifying the performance of garbage collection vs. explicit memory management. *SIGPLAN Notices*, 40(10):313–326, 2005.

[68] International Organization for Standardization. *ISO/IEC 14882:2003: programming languages — C++*. International Organization for Standardization, Geneva, 2003.

[69] S. R. Walli. The posix family of standards. *StandardView*, 3(1):11–17, 1995.

[70] URL: http://www.gnu.org/licenses/gpl-2.0.html, GNU General Public License. Version 2. Free Software Foundation. Dostopno na spletu: 14. 6. 2013.

[71] P. F. Dubois, K. Hinsen in J. Hugunin. Numerical python. *Computers in Physics*, 10(3), 1996.

[72] T. E. Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20, 2007.

[73] W. Humphrey, A. Dalke in K. Schulten. VMD – Visual Molecular Dynamics. *Journal of Molecular Graphics*, 14:33–38, 1996.

[74] O. Lenz. PMI - Parallel Method Invocation. V *Proceedings of the 8th Python in Science Conference*, Pasadena, 18. – 23. avgust 2009, 48–50.

[75] D. V. Lindley. The theory of queues with a single server. *Mathematical Proceedings of the Cambridge Philosophical Society*, 48:277–289, 1952.

[76] J. Kiefer in J. Wolfowitz. On the theory of queues with many servers. *Transactions of the American Mathematical Society*, 78:1–18, 1995.

[77] C. Kim in A. Agrawala. Analysis of the fork-join queue. *Computers, IEEE Transactions on*, 38(2):250–255, 1989.

[78] K. Kremer in G. S. Grest. Dynamics of entangled linear polymer melts: a molecular-dynamics simulation. *Journal of Chemical Physics*, 92(8):5057–5086, 1990.

[79] D. Frenkel in B. Smit. *Understanding molecular simulation.* Academic Press, San Diego, 2. izd., 2002.

[80] J. D. Weeks, D. Chandler in H. C. Andersen. Role of repulsive forces in determining the equilibrium structure of simple liquids. *The Journal of Chemical Physics*, 54(12):5237–5247, 1971.

[81] J. D. Hunter. Matplotlib: a 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[82] S. Miyamoto in P. A. Kollman. SETTLE: an analytical version of the SHAKE and RATTLE algorithm for rigid water models. *Journal of Computational Chemistry*, 13(8):952–962, 1992.

[83] T. Soddemann, B. Dünweg in K. Kremer. Dissipative particle dynamics: a useful thermostat for equilibrium and nonequilibrium molecular dynamics simulations. *Physical Review E*, 68:046702, 2003.

[84] S. Meloni in M. Rosati. Efficient particle labeling in atomistic simulations. *Journal of Chemical Physics*, 126:121102, 2007.

[85] K. J. Bowers, R. O. Dror in D. E. Shaw. Zonal methods for the parallel execution of range-limited N-body simulations. *Journal of Computational Physics*, 221(1):303–329, 2007.

[86] D. Shaw. A fast, scalable method for the parallel evaluation of distance-limited pairwise particle interactions. *Journal of Computational Chemistry*, 26(13):1318–1328, 2005.

[87] K. J. Bowers, R. O. Dror in D. E. Shaw. Overview of neutral territory methods for the parallel evaluation of pairwise particle interactions. *Journal of Physics: Conference Series*, 300-304, 2005.

[88] K. J. Bowers, R. O. Dror in D. E. Shaw. The midpoint method for parallelization of particle simulations. *Journal of Chemical Physics*, 124(18), 2006.

[89] D. E. Knuth. *The art of computer programming.* Addison-Wesley, Boston, 2. izd., 1997.

[90] D. C. Rapaport. Large-scale molecular dynamics simulation using vector and parallel computers. *Computer Physics Report*, 9:1–53, 1988.

[91] P. M. Morse. Diatomic molecules according to the wave mechanics. II. Vibrational levels. *Physical Review*, 34:57–64, 1929.

[92] A. Peterlin. Streaming birefringence of soft linear macromolecules with finite chain length. *Polymer*, 2(0):257–264, 1961.

[93] P. P. Ewald. Die berechnung optischer und elektrostatischer gitterpotentiale. *Annalen der Physik*, 369(3):253–287, 1921.

[94] R. Hockney in J. Eastwood. *Computer simulation using particles.* A. Hilger, Bristol, 1988.

[95] V. Ballenegger, J. J. Cerda, O. Lenz in C. Holm. The optimal P3M algorithm for computing electrostatic energies in periodic systems. *Journal of Chemical Physics*, 128(3):034109, 2008.

[96] T. Darden, L. Perera, L. Li in L. Pedersen. New tricks for modelers from the crystallography toolkit: the particle mesh ewald algorithm and its use in nucleic acid simulations. *Structure*, 7(3):R55 – R60, 1999.

[97] **S. Bevc**, C. Junghans, K. Kremer in M. Praprotnik. **Adaptive resolution simulation of salt solutions**. *New Journal of Physics*, sprejeto v objavo, 2013.

[98] H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren in J. Hermans. *Interaction models for water in relation to protein hydration.* D. Reidel Publishing Company, Dordrecht, 1981, 331-342.

[99] H. J. C. Berendsen, J. R. Grigera in T. P. Straatsma. The missing term in effective pair potentials. *The Journal of Physical Chemistry*, 91(24):6269–6271, 1987.

[100] J. Wang, P. Cieplak in P. A. Kollman. How well does a restrained electrostatic potential (RESP) model perform in calculating conformational energies of organic and biological molecules? *Journal of Computational Chemistry*, 21(12):1049–1074, 2000.

[101] A. Soper. Empirical potential monte carlo simulation of fluid structure. *Chemical Physics*, 202(2-3):295 – 306, 1996.

[102] D. Reith, M. Pütz in F. Müller-Plathe. Deriving effective mesoscale potentials from atomistic simulations. *Journal of Computational Chemistry*, 24(13):1624–1636, 2003.

[103] M. Praprotnik, L. Delle Site in K. Kremer. Adaptive resolution scheme for efficient hybrid atomistic-mesoscale molecular dynamics simulations of dense liquids. *Physical Review E*, 73:066701, 2006.

[104] S. Poblete, M. Praprotnik, K. Kremer in L. D. Site. Coupling different levels of resolution in molecular simulations. *The Journal of Chemical Physics*, 132(11):114101, 2010.

[105] S. Fritsch, S. Poblete, C. Junghans, G. Ciccotti, L. Delle Site in K. Kremer. Adaptive resolution molecular dynamics simulation through coupling to an internal particle reservoir. *Physical Review Letters*, 108:170602, 2012.

[106] **S. Bevc**, C. Junghans in M. Praprotnik. **Web interface for Coarse-Graining soft matter**. *V pripravi*, 2013.

[107] C. Junghans, M. Praprotnik in K. Kremer. Transport properties controlled by a thermostat: an extended dissipative particle dynamics thermostat. *Soft Matter*, 4:156–161, 2008.

[108] T. Koishi, S. Tamaki in T. Ebisuzaki. Large scale molecular dynamics simulation of aqueous nacl solutions. *Proceedings of the 14th International Conference on the Properties of Water and Steam*, 212-216, 2005.

[109] **S. Bevc**, J. Konc, J. Stojan, M. Hodošček, M. Penca, M. Praprotnik in D. Janežič. **ENZO: a web tool for derivation and evaluation of kinetic models of enzyme catalyzed reactions**. *PLoS ONE*, 6(7):e22265, 2011.

[110] URL: http://enzo.cmm.ki.si/. ENZO web tool. Dostopno na spletu 14. 6. 2013.

[111] R. TL. Acetylcholinesterase. *Advances in enzymology and related areas of molecular biology*, 43(891):103–218, 1975.

[112] R. G. Duggleby. Analysis of enzyme progress curves by nonlinear regression. V *Enzyme kinetics and mechanism part D: developments in enzyme dynamics*. Academic Press, Waltham, 1995, 61-90.

[113] URL: http://sl.wikipedia.org/wiki/Encimski_inhibitor. Wikipedia, encimski inhibitor. Dostopno na spletu: 25. 8. 2013.

[114] P. Kuzmič. DynaFit – a software package for enzymology. V *Methods in enzymology*. Academic Press, Waltham, 2009, 247-280.

[115] K. A. Johnson. Fitting enzyme kinetic data with kintek global kinetic explorer. V *Methods in enzymology*. Academic Press, Waltham, 2009, 601-626.

[116] K. Yamaoka in T. Nakagawa. A nonlinear least squares program based on differential equations, multi (runge), for microcomputers. *Journal of Pharmacobio-Dynamics*, 6(8):595–606, 1983.

[117] URL: http://en.wikipedia.org/wiki/Regression_analysis. Wikipedia, analiza regresije. Dostopno na spletu: 25. 8. 2013.

[118] J. Resig. URL: http://jquery.com/. jQuery: the write less, do more, JavaScript library. Dostopno na spletu: 14. 6. 2013.

[119] URL: http://www.flotcharts.org/. Flot: attractive JavaScript plotting for jQuery. Dostopno na spletu: 14. 6. 2013.

[120] URL: http://www.w3.org/TR/XMLHttpRequest/. XMLHttpRequest, W3C working draft. Dostopno na spletu: 14. 6. 2013.

[121] URL: http://www.json.org/. JSON (JavaScript Object Notation): a lightweight data-interchange format. Dostopno na spletu: 14. 6. 2013.

[122] URL: http://www.w3.org/TR/file-upload/. File API, W3C working draft. Dostopno na spletu: 14. 6. 2013.

[123] J. Rozman, J. Stojan, R. Kuhelj, V. Turk in B. Turk. Autocatalytic processing of recombinant human procathepsin b is a bimolecular process. *FEBS Letters*, 459(3):358 – 362, 1999.

[124] R. Delgado-Buscalioni, K. Kremer in M. Praprotnik. Concurrent triple-scale simulation of molecular liquids. *The Journal of Chemical Physics*, 128(11):114110, 2008.

[125] R. Delgado-Buscalioni, K. Kremer in M. Praprotnik. Coupling atomistic and continuum hydrodynamics through a mesoscopic model: application to liquid water. *The Journal of Chemical Physics*, 131(24):244107, 2009.

Izjavljam, da je disertacija nastala kot rezultat samostojnega raziskovalnega dela.

V Kopru, 6. september 2013 Staš Bevc