ELEKTROTEHNIŠKI VESTNIK 86(5): 283–289, 2019
IZVIRNI ZNANSTVENI ČLANEK

Sistem za porazdeljeno računanje simulacij dinamično trianguliranih površin in njihovo analizo

Samo Penič[1], **Miha Fošnarič**[2]

[1] *Univerza v Ljubljani, Fakulteta za elektrotehniko, Tržaška cesta 25, 1000 Ljubljana, Slovenija*
[2] *Univerza v Ljubljani, Zdravstvena fakulteta, Zdravstvena pot 5, 1000 Ljubljana, Slovenija*
E-pošta: samo.penic@fe.uni-lj.si

Povzetek. Simulacije dinamično trianguliranih površin kot modelov lipidnih dvojnih plasti so časovno zamudne. Pogosto raziskujemo fazni prostor, kjer za vsak set parametrov zaganjamo svojo simulacijo. V takšnih primerih lahko pospešimo raziskavo z vzporednim zaganjanjem simulacij. V članku predstavljamo sistem za vzporedno računanje simulacij dinamično trianguliranih površin s simulatorjem `trisurf`. Izbrali smo metodo oportunističnega računanja in tehnologijo strežnik-gost. Simulator tako postane računski del sistema, ki se izvaja na strani računalnika gosta in delovne naloge prejema od strežnika. Strežnik razdeljuje posamezne simulacije med prijavljene goste (delavce) in shranjuje vrnjene računske rezultate. Strežnik deluje tudi kot hramba rezultatov simulacij in orodje za njihovo analizo. Analizo lahko raziskovalec izvaja kar na strežniku prek spletnega brskalnika v delovnem zvezku Jupyter. Predstavljen sistem smo naredili po zgledu podobnih uveljavljenih sistemov, kjer smo upoštevali dobre prakse raziskovalnega dela za ponovljivost rezultatov.

Ključne besede: Monte Carlo simulacije, porazdeljeno računanje, paralelizacija, podatkovne baze, Jupyter, biološke membrane, lipidna dvojna plast

A System for a distributed computation of a dynamic triangulated surface simulation and its analysis

A dynamic triangulated surface simulations used in modelling lipid bilayers are time consuming. Frequently, the phase space of multiple parameters needs to be investigated by simulating each parameter set. Concurrent running of multiple simulations in parallel accelerates the investigation of the phase space. In the paper, we present a system for a parallel computation of a triangulated surface simulations using a simulator `trisurf`. A method of voluntary computing was applied, using a server-client technology. The simulator itself becomes the computing part of the system that is being executed on the client side and waits for a workload from the server. Besides, being a workload distributor, the server also acts as a simulation results collector and data archiver. The analysis of the data and metadata is performed on the server itself using a standard web browser. The technology used for the data analysis is the Jupyter notebook. The system is engineered based on experiences with similar systems and following good research practices for result reproducibility.

Keywords: Monte Carlo simulations, distributed computation, paralelization, database, Jupyter, biological membranes, lipid bilayers

1 UVOD

Simulacije dinamično trianguliranih površin so metode Monte-Carlo. Na površino, ki jo želimo simulirati, napnemo trikotniško mrežo, ki lahko v okviru izbrane verjetnostne porazdelitve in robnih pogojev spreminja tako obliko kot tudi povezanost vozlišč [1], [2]. Spremembo oblike površine dosežemo predvsem s premikanjem položaja vozlišč trikotniške mreže, spreminjanje povezav med vozlišči pa omogoči tudi lateralno mobilnost vozlišč v mreži. Tako simulirana površina se obnaša kot dvodimenzionalna tekočina, kar je pomembna lastnost membran, katerih osnova je dvojna plast lipidnih molekul [3]. Zato simulacije dinamično trianguliranih površin pogosto uporabljamo pri modeliranju lipidnih in bioloških membranskih sistemov [2], [4 - 8].

Pomemben primer membran, tankih ločnic med dvema območjema, so membrane bioloških celic. Celice različnih velikosti, oblik in funkcij so osnovni sestavni deli bioloških sistemov. Kljub raznolikosti celic, ki jih najdemo v bioloških sistemih, so osnovni gradniki in njihova kemijska sestava večine celic enaki [3], [4], [5]. Lastnosti bioloških membran, ki obdajajo celice ali njene organele, se precej razlikujejo od lastnosti makroskopskih objektov, ki smo jih vajeni iz vsakdanjega življenja. Najštevilnejši gradniki celičnih membran so lipidne molekule, med katerimi prevladujejo fosfolipidi, ki skupaj z vgrajenimi proteini sestavljajo skoraj vso maso membrane [6]. Fosfolipidne molekule so amfifilne, kar pomeni, da so sestavljene iz hidrofobnega in hidrofilnega dela. V vodnem mediju se tako samoorganizirajo v fosfolipidno dvojno plast, ki lahko tvori mehurčke oziroma vezikle (angl. vesicle). Takšna dvojna plast je lahko tako upogljiva, da je izpostavljena termičnim fluktuacijam, ki lahko pomembno vplivajo na njene fizikalne lastnosti in biološke funkcije.

Prejet 8. junij, 2019
Odobren 11. oktober, 2019

Triangulirane površine kot model lipidnih in bioloških membranski sistemov so model s skaliranjem (angl. coarse-grained model). V trikotniški mreži lahko en trikotnik ali eno vozlišče in njegova neposredna okolica modelira del membrane, ki jo zapolnjuje več molekul membrane. Interakcijo teh delov mreže z okolico opišemo s primernimi fizikalnimi modeli. Modeli s skaliranjem so nepogrešljiv del modeliranja bioloških sistemov [4], [3], omogočajo kvalitativen vpogled na dogajanje v membranah in pripomorejo k razumevanju biofizikalnih lastnosti bioloških struktur [7].

Spisati programsko kodo za simulacije dinamično trianguliranih površin je sorazmerno zahteven projekt. Zato smo simulator `trisurf`, ki je razvit prav za simulacije dinamično trianguliranih površin, izdali pod odprtokodno programsko licenco [8]. Fizikalni model, na katerem temelji simulator, je podrobneje opisan v [9], algoritem pa v [10]. Surovi rezultati simulatorja so oblika membrane, torej matematični opis trikotniške mreže – pozicije posameznih točk mreže in vezi med njimi, ter druge fizikalne lastnosti posameznih delov membrane, ki jih pripišemo mreži.

Časovno dolgotrajne simulacije pospešimo z vzporednim računanjem. Poskus vzporednega računanja, kjer več procesov izvaja premike Monte Carlo na enem samem sistemu (npr. veziklu), smo v zgodnji fazi razvoja nadomestili z učinkovitejšo in preprostejšo obliko paralelizacije. Iz izbranega začetnega stanja naredimo vzporeden zagon simulacij. Vsaka simulacija lahko obravnava sistem z drugačnim naborom parametrov ali pa z enakim. Pri slednjem praviloma zavržemo toliko začetnih korakov, da stanja med seboj niso več statistično korelirana. Tako vsak zagon simulatorja preračunava ločen sistem in odpadejo težave vzporednega računanja, kjer se lahko procesi med sabo motijo. Simulacije se lahko izvajajo na več računalnikih, ki niso nujno ves čas povezani med seboj. V praksi smo se pogosto srečevali s potrebo prečesavanja faznega prostora, torej širokega nabora parametrov [11]. Takrat smo delo praviloma razdeljevali tako, da je vsak računalnik (ali procesorsko jedro istega računalnika) izvajal simulacijo z ločenim naborom parametrov.

V članku predstavimo sistem upravljanja vzporednega računanja in dela s podatki. Sistem uporablja simulator `trisurf`, vendar je prilagodljiv in odprtokoden, zato ga je mogoče uporabiti tudi pri drugih projektih. Pri načrtovanju hranjenja podatkov je sistem zastavljen tako, da je omogočen hiter dostop do množice rezultatov simulacij. Celovitost sistema smo zagotovili tudi z uporabniškim vmesnikom, s katerim je mogoča tako analiza rezultatov kot tudi njihova dokumentacija in priprava za znanstveno objavo.

2 RAČUNANJE V GRUČAH IN PORAZDELJENO RAČUNANJE

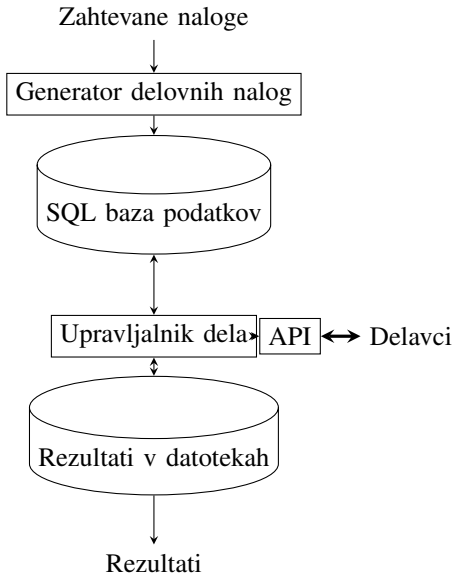
Simulacije Monte Carlo so računsko zahtevne (potrebujejo veliko procesorske računske moči), niso pa zahtevne glede pomnilnika ali prostora za shranjevanje vmesnih rezultatov. Zaradi potrebe po računski moči smo zasnovali sistem za vzporedno računanje, pri čemer smo raziskali dve tehniki, ki sta primerni za naš problem. Izognili smo se vzporednemu računanju na istem sistemu (npr. isti trikotniški mreži), saj ta pristop zahteva komunikacijo med posameznimi procesi in je skalabilen samo do neke mere. Rezultate posamezne simulacije (primera za izbran nabor fizikalnih parametrov), bi tako sicer pridelali hitreje, vendar bi del procesorskega časa porabili za upravljanje med procesi in za zagotavljanje, da se procesi med samim delom ne motijo. Izbrali smo drug pristop. Ker nas običajno zanima večji del faznega prostora, torej širok nabor parametrov, kjer potrebujemo za vsako kombinacijo parametrov svojo simulacijo, smo paralelizirali zagon več neodvisnih simulacij. Tako so posamezne simulacije samostojne in ne potrebujejo medsebojne komunikacije. Pri manjšem zahtevanem številu vzporednih simulacij, kjer sodeluje le nekaj računalnikov, je mogoč ročen zagon posameznih simulacijskih procesov, kar pa je pri večjem številu vzporednih simulacij zelo težko izvajati in spremljati. Zato smo razvili avtomatiziran sistem za zagon simulatorja.

Za hitro računanje se je zgodaj v razvoju računalnika začel razvoj metod za vzporedno računanje na več računalnikih hkrati. Prve tehnike so bile računanje v gručah (angl. cluster) in mrežnega procesiranja (angl. grid computing) [12], čedalje bolj pa postaja popularno porazdeljeno procesiranje, kjer sta popularna pristopa računanje v oblaku (angl. cloud computing) in oportunistično računalništvo (angl. volunteer computing) [13].

Za upravljanje računskih procesov programiranja v gručah sta razširjena odprtokodna projekta SLURM in PBS, namenjena podpori lokalnim gručam računalnikov in mrežnemu procesiranju. Projekt SLURM se je začel razvijati leta 2002 kot sistem za upravljanje z računskimi zmogljivosti računalnikov v gručah z nameščenim operacijskim sistemom Linux, pozneje pa so mu dodali robusten sistem za razdeljevanje dela (angl. scheduling). Danes je SLURM odprtokodna platforma za razdeljevanje računskega dela med računalnike v gručah in njihov nadzor. Razvoj sistema PBS se je začel že leta 1991 kot odprtokodna platforma za razdeljevanje dela v gručah in je del projekta OpenHPC. Omeniti velja še projekt HTCondor, ki je odprtokodni razdeljevalnik dela v mrežnem računalništvu in ga uporablja npr. Amazon EC2 [12].

Sisteme masovnega oportunističnega distribuiranega računanja je populariziral projekt Seti@Home [14]. Prostovoljci pri projektu prispevajo računske zmogljivosti svojih računalnikov za analizo signalov, prejetih iz ra-

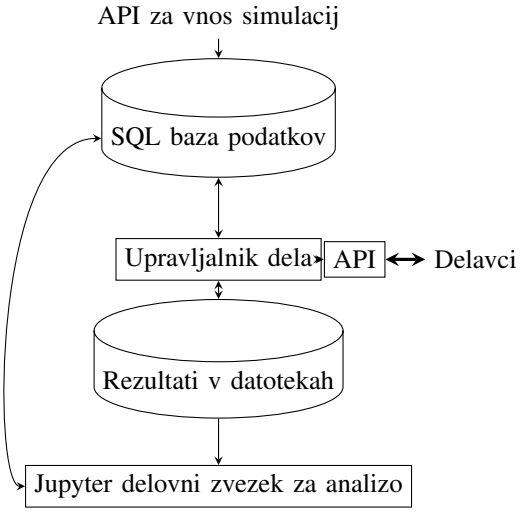
dijskih teleskopov v upanju, da bodo naleteli na signale, ki jih v vesolje pošiljajo inteligentna bitja zunaj našega osončja. Seti@Home sicer ni bil prvi tak projekt, vendar je pokazal, da je oportunistično računanje izvedljivo, in kmalu se je razvila množica podobnih znanstvenih projektov. Seti@Home je se sčasoma preselil na BOINC (kratica za Berkeley Open Infrastructure for Network Computing), ki omogoča platformo za distribucijo orodij za oportunistično računanje [15]. BOINC so začeli razvijati leta 2002 in je hitro postal vodilna platforma za oportunistično računanje. Na platformi BOINC se trenutno izvaja več kot 30 znanstvenih projektov svetovnih razsežnosti [16].



Slika 1: Poenostavljena zgradba sistema BOINC, prirejena po [17]

Razlika med pristopoma za računanje v gručah in oportunističnega procesiranja je velika. V gručah nastopajo računalniki, ki so med seboj (praviloma) enaki in povezani s hitrim računalniškim omrežjem. Računalniki delujejo konstantno in zanesljivo, zato imamo med računanjem na voljo konstantno računsko moč. Rezultati, ki jih zberemo iz posameznih računalnikov v gruči, so varni – v njihovo pravilnost lahko zaupamo, saj je cela gruča pod kontrolo administratorja ali izvajalca računskega projekta [12]. Oportunistično procesiranje se izvaja na računalnikih prostovoljcev, ki so povezani v splet in so pripravljeni darovati računske zmogljivosti svojih naprav. Te so zelo raznolike, imajo nameščene različne operacijske sisteme, strojno opremo ... Računalnikov, ki izvajajo računsko delo (delavcev), ne moremo kontrolirati in ti se lahko kadarkoli izključijo ali izgubijo povezavo s strežnikom, ki deluje kot razdeljevalec dela. Ravno tako ne moremo zaupati rezultatom oddaljenih računalnikov v lasti prostovoljcev, saj lahko ti delujejo nepravilno ali so zlonamerno prilagojeni tako, da vračajo slabe rezultate [12], [18].

Pri postavitvi našega sistema za izvajanje simulacij smo se odločili za oportunistično procesiranje, saj nismo imeli dostopa do homogenega sistema v gruči. Zaradi specifik simulatorja `trisurf` in relativno zahtevne postavitve sistema BOINC je bilo pri sistemu za upravljanje simulacij `trisurf` hitreje in pregledneje razviti lasten sistem za razdeljevanje dela in računanje na oddaljenih računalnikih prostovoljcev. Zgledovali smo se po poenostavljeni zgradbi sistema BOINC (brez določenih delov, ki spremlja računsko delo in nagrajuje prostovoljce) in po podobnem sistemu projekta Folding@Home [19]. Glavni deli sistema BOINC so predstavljeni na sliki 1. Raziskovalec sestavi naloge in sistem BOINC ima svoj lasten generator preprostih nalog, ki jih razdeli delavcem po protokolu HTTP in prirejenim API vmesnikom. Delavci dostopajo do dela, shranjenega v bazi, prek internetnega API vmesnika (odebeljena puščica na sliki), ki je neposredno povezan z razdeljevalno enoto upravljalnika dela. Računske rezultate delavec vrne na strežnik in upravljalnik dela, ki shrani datoteko z rezultati na disk. Tako zbrani neobdelani (surovi) rezultati so na voljo raziskovalcu za analizo, ki jo izvede zunaj sistema BOINC.



Slika 2: Zgradba sistema `trisurf`

Načelna poenostavljena zgradba projekta `trisurf`, ki ne vsebuje le simulatorja, ampak tudi okolje za vzporedno računanje in analizo podatkov, prikazuje slika 2. Razliki v shemah na slikah 1 in 2 sta le pri generiranju nalog in pri analizi podatkov. Pri generiranju dela je treba pri sistemu `trisurf` delo definirati izven sistema. Delovne naloge (posamezne simulacije) se v bazo vnesejo prek spletnega vmesnika, ki deluje po protokolu HTTP in REST API (angl. Representational Sate Transfer Application protocol Interface). Analizo surovih podatkov, zbranih v datotekah XML, smo vključili kar v sistem `trisurf` v obliki delovnega zvezka Jupyter. Na glavnem strežniku projekta

`trisurf` je tako nameščen program, spletni vmesnik namenjen raziskovalcem, hkrati pa shranjuje podatke in metapodatke posameznih raziskav, simulacij in surovih rezultatov, izračunanih s simulatorjem `trisurf`. Hkrati je tudi razdeljevalec računskega dela in zbiralnik rezultatov. Napisan je v programskem jeziku Python, pri čemer smo uporabili okolje Django za razvoj spletne aplikacije in upravljanje podatkovne baze. Komunikacija z računalniki, ki izvajajo računsko delo (delavci), poteka prek strani REST API in protokola HTTP.

Za računalnike, ki opravljajo simulacije, je trenutno podprt le operacijski sistem Linux. Na nje je potreba namestiti simulator `trisurf` in preprost Pythonov vmesnik, ki komunicira s strežnikom. Delavec se najprej prijavi in strežniku sporoči nameščeno različico simulatorja `trisurf`. Ta mu na podlagi sporočene različice in seznama prostih del dodeli simulacijo ter mu pošlje podatke za izvajanje (zadnji rezultat ali konfiguracijsko datoteko, če v bazi ni še nobenega rezultata). Vmesnik na delavcu zažene simulator s prejetimi podatki in spremlja napredek simulacij.

Ker je delovanje delavcev nezanesljivo (prekinitve v računanju, počasnost, nenapovedljivi izklopi delavcev), mora vmesnik na delavcu v določenih intervalih sporočati strežniku stanje delovanja in potrjevati delo na simulaciji. Ko simulator vrne rezultat (novo mikrostanje), ga vmesnik na delavcu pošlje na strežnik. Ta zabeleži transakcijo in shrani rezultat v ustrezno mapo. V primeru napake v komunikaciji ali neodzivnosti delavca strežnik simulacijo postavi v seznam prostih simulacij, ki čakajo na ponovno razdeljevanje delavcem. Zaradi izbrane tehnologije nima strežnik nad delavci nikakršne kontrole, zato je pomembno, da delavec v vnaprej določenem času posreduje podatke o delovanju. Pobudnik komunikacije je vedno delavec. Zaradi uporabe protokola HTTP je delovanje sistema mogoče tudi, če so delavci zavarovani s požarnimi zidovi ali drugimi varnostnimi tehnologijami.

3 HRANJENJE PODATKOV

Surovi podatki, ki jih dobimo iz simulacij, so mikrostanja sistema (npr. vezikla) v obliki XML (primerna tudi za pregledovanike, ki podpirajo format VTK), z metapodatki o parametrih, s katerimi je bila izvedena simulacija z različico simulatorja `trisurf`, s katerim je bilo stanje pridobljeno, ter datumom in časom nastanka. Sproti se izračuna še nekaj metrik vezikla, ki jih želimo spremljati med samim izvajanjem simulacij.

Za organizacijo surovih podatkov uporabljamo odprtokodno podatkovno bazo PostgreSQL. Vsako uspešno izračunano stanje, ki ga klient prenese nazaj k razdeljevalcu, razdeljevalec zabeleži v tabeli izmenjave podatkov, rezultat shrani na disk v obliki datoteke XML, referenco na to datoteko pa v podatkovno bazo. Raziskovalec ima tako možnost iskanja in selekcije specifičnih datotek z uporabo običajnih orodij SQL ali z njihovimi Djangovimi razširitvami. Tabele v bazi so zgrajene hierarhično. Najvišja stopnja je projekt, vsak projekt ima lahko več raziskav in vsaka raziskava več simulacij. Surovi podatki pripadajo posamezni simulaciji. Vsaka stopnja v hierarhiji je opremljena z ustreznimi metapodatki za lažje iskanje in selekcijo ustreznih podatkov za analizo.

Količina pridobljenih podatkov v nekaj letih raziskav preseže velikost sodobnih trdih diskov. Trenutno veliko količino podatkov obvladujemo z diskovnimi polji RAID, hkrati pa raziskujemo možnosti za distribuirano hranjenje podatkov in hranjenje podatkov v oblaku z odprtokodnima projektoma GlusterFS in Ceph [20].

4 OBDELAVA PODATKOV

V svetu množice podatkov je komunikacija z računalnikom le del izziva. Pomemben del je tudi komunikacija s sodelavci, kolegi iz stroke in širšo javnostjo. Zato je pregleden sistem podatkov in algoritmov ključnega pomena za efektivno delo s podatki.

Za obdelavo podatkov smo izbrali odprtokoden programski jezik Python, ki je visokonivojski, torej sorazmerno proprost za uporabo ter širokonamemben in s tem tudi dobro podprt. Python skupaj z odprtokodnimi knjižnicami, kot so `numpy`, `scipy`, `matplotlib` in druge, postavlja *de facto* standard za znanstveno računanje [21]. Python se tako postavlja ob bok programskim paketom, kot sta Matlab in Mathematica, in jih ponekod celo presega [22]. Python po svoji zasnovi sledi principom DRY (angl. Don't Repeat Yourself) in tako veščega uporabnika spodbuja k ponovni uporabi že narejenih algoritmov.

Zaradi centralizirane hrambe velike količine podatkov je obdelava na lokalnih računalnikih nepraktična, zato smo se odločili za obdelavo podatkov direktno na strežniku brez potrebe po kopiranju podatkov s strežnika na lokalni računalnik prek spleta. Raziskovalec ima dostop do uporabniškega vmesnika preko spletnega brskalnika in programskega paketa Jupyter. Tako vse razvojno in raziskovalno delo poteka na centralnem strežniku z neposrednim dostopom do podatkov.

Jupyter je odprtokodni projekt, ki se je razvil iz interaktivnega vmesnika za Python (podobnega, kot je npr. Matlab). Sprva je bil uporabniški vmesnik dostopen prek terminalskega okna na računalniku v izbranem operacijskem sistemu, Jupyter pa je prestavil uporabniški vmesnik v spletni brskalnik, dodal tehnologijo server-client in tako omogočil delo v oblaku. Sam vmesnik spominja na obliko delovnega zvezka (angl. notebook), kakršno poznamo iz programskega paketa Mathematica. Osnovni gradnik delovnega zvezka je celica, ki lahko vsebuje programsko kodo ali besedilo v jeziku markup in podpira TEXov sistem pisanja enačb. Projekt Jupyter poleg delovnega zvezka omogoča tudi vgradnjo elementov spletnih strani, zato lahko z njim naredimo tudi prototipe

uporabniške aplikacije. Tipičen primer so ponavljajoče se analize podatkov.

Raziskovalec, ki analizira podatke, si v delovnem zvezku Jupyter tako sproti, medtem ko razvija algoritem za obdelavo podatkov, ustvarja ekvivalent laboratorijskega dnevnika ali zapiskov, ki jih pozneje uporabi kot osnovo za pisanje znanstvenega članka. Dostop do spletnega vmesnika je omogočen od koderkoli, zato je mogoče delo na daljavo in s kakršnimkoli terminalom, od osebnega računalnika do mobilnega telefona. Format Jupytrovega delovnega zvezka je odprti standard, zato ga podpira že veliko pregledovalnikov in spletnih storitev. Delovni zvezek pa lahko vedno izvozimo tudi v formatu pdf ali v čisti izvorni kodi Python.

Projekt Jupyter je hitro razvijajoč se sistem, ki ima ogromno privržencev med raziskovalci, predvsem na fizikalnem področju. Pred nekaj leti so začeli razvijati naprednejši sistem delovnih zvezkov JupyterLab, ki bo omogočal tudi terminalski dostop do strežnika in do lupine Python.

Pri načrtovanju učinkovitega sistema za obdelavo rezultatov s projektom Jupyter smo se oprli na osem točk dobre prakse računalniške obdelave podatkov, povzetih po članku [23].

1) **Programi naj bodo berljivi in pisani za ljudi, ne za stroje**
   Prednost pred samo hitrostjo izvajanja je berljivost kode, saj je čas raziskovalca vreden več kot računski čas računalnika. Raziskovalcu je omogočen preprost dostop do podatkov iz celotne baze izračunanih in shranjenih mikrostranj, kot tudi dostop do podatkov o njih. Hkrati je omogočen dostop do funkcij samega simulatorja, napisanega v programskem jeziku C preko povezovalnih funkcij Python. Raziskovalec lahko tako poustvari sliko vezikla med simulacijo in pridobi dodatne podatke, ki niso del že vnaprej opravljenih analiz.
   Uporabnika/raziskovalca spodbujamo k pisanju berljive programske kode za obdelavo podatkov, saj je lahko ta ponovno uporabljena pri drugih projektih, hkrati pa služi kot dokumentacija raziskovalnega dela.

2) **Računalnik naj opravlja težaška opravila**
   Pri obdelavi podatkov, še zlasti pri statistični obdelavi, kjer se ukvarjamo s kopico podatkov, je smiselno, da ponavljajoča se opravila opravi kar računalnik sam. Raziskovalcu so na voljo pogoste rutinske obdelave podatkov, sam pa si lahko sestavi dodatne funkcije, ki presegajo osnovno zasnovo sistema. Sem spadajo tudi razni sistemi za gradnjo začasnih podatkovnih baz in dolgotrajnejših analiz podatkov.

3) **Spremembe v algoritmih naj bodo inkrementalne**
   Med delom spodbujamo raziskovalca, da razvija svoj algoritem počasi in sproti preverja pravilnost rezultatov. Zelo priporočljiva je uporaba sistemov za upravljanje različic (angl. versioning system), v katerih se zrcali celotna evolucija razvoja algoritmov obdelave podatkov.

4) **Ne ponavljaj stvari, ki si jih že naredil (oz. so jih naredili drugi)**
   Raziskovalec naj uporablja že napisane knjižnice. Smiselno je tudi, da svoje algoritme v obliki objektov ali funkcij združi v module, ki jih nato uporablja iz več raziskovalnih projektov. Funkcije, ki se pri obdelavi simulacijskih podatkov pogosteje uporabljajo, smo zapakirali v modul, ki ga preprosto vključimo v delovni zvezek.

5) **Pričakuj napake v podatkih in pri obdelavi**
   Vnos podatkov v algoritem je vedno lahko obremenjen z napako (naj bo to manjkajoč podatek ali pa podatek napačnega tipa). Programiranje raziskovalca naj bo defenzivno, kar pomeni, da naj se vsi podatki preverjajo pri prenosu med posameznimi segmenti analize. Priporočljiva je tudi uporaba t. i. unit testov, ki zagotavljajo pravilno delovanje posameznih enot/funkcij algoritma.

6) **Optimiziraj algoritme šele, ko ti delujejo pravilno**
   Izogibajmo se prehitre optimizacije programske kode. Računsko zahtevne segmente optimiziramo šele potem, ko je sestavljen celoten algoritem za analizo, če je to potrebno. Optimalna je uporaba nižjenivojskega programskega jezika (npr. C). V Jupytru sam jezik C ni podprt, lahko pa del kode sestavimo kot vgrajen program C in paket `cython`.

7) **Dokumentiraj namen algoritma, ne izvedbe**
   Ideja delovnih zvezkov je v združevanju programske kode in dokumentacije. Programska koda je poleg samega izvajanja računanja del dokumentacije. Programsko kodo dopolnjujemo z besedilom, ki opisuje namen posameznega dela kode ali interpretacijo rezultatov. Samih tehničnih detajlov algoritma ne opisujemo, če to ni nujno potrebno, saj naj bi bilo samo delovanje kode razvidno iz kode same. V okolju Jupyter lahko za dokumentacijo uporabimo jezik z značkami za opis oblike besedila (angl. markup language).

8) **Sodeluj**
Sodelovanje raziskovalcev pri raziskovalnem delu je ključnega pomena. Pogosto uporabljamo programiranje v parih, kjer dva raziskovalca skupaj razvijata algoritem za obdelavo podatkov. Pred objavo rezultatov je priporočljiv tudi neodvisen pregled algoritmov tretje osebe, ki opozori na morebitne nepravilnosti pri raziskovalnem delu, ki jih je raziskovalec spregledal.
V duhu znanstvenega sodelovanja in deljenja znanstvenih podatkov naj raziskovalec deli svoje algoritme za obdelavo z drugimi tako, da omogoči dostopnost svojih programov na javno dostopnih servisih.

5 Sklep

Avtomatiziran sistem razdeljevanja simulacij sistema `trisurf` je narejen po zgledu podobnih razdeljevalnikov sistemov oportunističnega računanja. Ker so simulacije računsko zahtevne, smo z uvedbo vzporednega računanja pospešili izvajanje raziskav. V praksi se je pokazalo, da je veliko pomembnejše, kako so programi berljivi (raziskovalcu), kot pa njihova optimizacija. Sistem je zasnovan tako, da je čim bolj približan uporabniku, torej raziskovalcu, ki pripravlja parametre za simulacije in obdeluje podatke. Pri zasnovi sistema `trisurf` smo uporabili odprtokodne tehnologije, sam sistem pa smo izdali pod odprtokodno licenco GPL različice 3 ali poznejše. Izvorna koda sistema je v celoti na voljo v repozitoriju GIT [24].

Zaradi podobnosti razdeljevalnika z razdeljevalnikom platforme BOINC raziskujemo možnosti prilagoditve simulatorja za uporabo s sistemi BOINC in SLURM, kot je opisano v [25]. Kombinirana uporaba obeh sistemov omogoča tako oportunistično računanje kot računanje v gručah. Za shranjevanje velikih količin podatkov predlagamo uporabo porazdeljenega datotečnega sistema GlusterFS.

Literatura

[1] G. Gompper and D. M. Kroll, "Triangulated-surface models of fluctuating membranes," in *Statistical Mechanics of Membranes and Surfaces*, D. Nelson, T. Piran, and S. Weinberg, Eds. World Scientific, Singapore, 2004, pp. 359–426, 2nd edition.

[2] D. M. Kroll and G. Gompper, "The conformation of fluid membranes - monte-carlo simulations," *Science*, vol. 255, no. 5047, pp. 968–971, 1992.

[3] D. Boal, *Mechanics of the Cell*, 2nd ed. Cambridge University Press, 2012.

[4] R. Phillips, J. Kondev, J. Theriot, N. Orme, and H. Garcia, *Physical biology of the cell*. Garland Science New York, 2009.

[5] L. Vodovnik, D. Miklavčič, and T. Kotnik, *Biološki sistemi*. Ljubljana: Založba FE in FRI, 1998.

[6] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter, *Molecular Biology of the Cell*, 4th ed. Garland Science, 2002.

[7] H. Noguchi, "Membrane Simulation Models from Nanometer to Micrometer scale," *Journal of Physical Society of Japan*, vol. 78, no. 4, p. 041007, 2009.

[8] S. Penič, "Gitlab: Project trisurf-ng," 2019. [Online]. Available: https://gitlab.com/Penic/trisurf-ng

[9] M. Fošnarič and S. Penič, "Trisurf – odprtokodno programsko orodje za simulacije dinamično trianguliranih površin (1. del: Model)," in *Zbornik šestindvajsete mednarodne elektrotehniške in računalniške konference ERK 2017, September 2017, Portorož, Slovenija*, 2017.

[10] S. Penič and M. Fošnarič, "Trisurf – odprtokodno programsko orodje za simulacije dinamično trianguliranih površin (2. del: algoritem)," in *Zbornik šestindvajsete mednarodne elektrotehniške in računalniške konference ERK 2017, September 2017, Portorož, Slovenija*, 2017.

[11] M. Fosnarič, S. Penič, A. Iglič, V. Kralj-Iglič, M. Drab, and N. Gov, "Theoretical study of vesicle shapes driven by coupling curved proteins and active cytoskeletal forces," *Soft Matter*, vol. 15, pp. 5319–5330, 2019.

[12] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. P. Anderson, "Cost-benefit analysis of cloud computing versus desktop grids." in *IPDPS*, vol. 9, 2009, pp. 1–12.

[13] D. P. Anderson, "Boinc: A platform for volunteer computing," *arXiv preprint arXiv:1903.01699*, 2019.

[14] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "Seti@ home: an experiment in public-resource computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[15] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*. IEEE Computer Society, 2004, pp. 4–10.

[16] BOINC project, "Choosing boinc project," 2019, [Online; accessed 17-June-2019]. [Online]. Available: https://boinc.berkeley.edu/projects.php

[17] S. Yi, D. Kondo, and D. P. Anderson, "Toward real-time, many-task applications on large distributed systems," in *European Conference on Parallel Processing*. Springer, 2010, pp. 355–366.

[18] D. P. Anderson, E. Korpela, and R. Walton, "High-performance task distribution for volunteer computing," in *First International Conference on e-Science and Grid Computing (e-Science'05)*. IEEE, 2005, pp. 8–pp.

[19] A. L. Beberg, D. L. Ensign, G. Jayachandran, S. Khaliq, and V. S. Pande, "Folding@ home: Lessons from eight years of volunteer distributed computing," in *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE, 2009, pp. 1–8.

[20] G. Donvito, G. Marzulli, and D. Diacono, "Testing of several distributed file-systems (hdfs, ceph and glusterfs) for supporting the hep experiments analysis," in *Journal of physics: Conference series*, vol. 513, no. 4. IOP Publishing, 2014, p. 042014.

[21] K. J. Millman and M. Aivazis, "Python for scientists and engineers," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 9–12, 2011.

[22] J. Somers, "The scientific paper is obsolete," *The Atlantic*, 2018.

[23] G. Wilson, D. A. Aruliah, C. T. Brown, N. P. C. Hong, M. Davis, R. T. Guy, S. H. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley *et al.*, "Best practices for scientific computing," *PLoS biology*, vol. 12, no. 1, p. e1001745, 2014.

[24] S. Penič, "Gitblit repositories," 2019. [Online]. Available: https://git.penic.eu/repositories/

[25] V. V. Kashansky and I. L. Kaftannikov, "Application of slurm, boinc, and glusterfs as software system for sustainable modeling and data analytics," in *EPJ Web of Conferences*, vol. 173. EDP Sciences, 2018, p. 05010.

Samo Penič je leta 2015 doktoriral s področja elektrotehnike na Univerzi v Ljubljani. Je asistent na Fakulteti za elektrotehniko. Njegovo področje raziskovanja obsega razvoj programske opreme za modeliranje bioloških membran, simulacijo trianguliranih mrež Monte Carlo in obdelavo pridobljenih podatkov.

Miha Fošnarič je diplomiral na Oddelku za fiziko na Fakulteti za matematiko in fiziko Univerze v Ljubljani in doktoriral na Fakulteti za elektrotehniko Univerze v Ljubljani. Med leti 2000 in 2018 je bil zaposlen kot mladi raziskovalec in nato kot asistent za fiziko na Fakulteti za elektrotehniko Univerze v Ljubljani. Od leta 2018 je zaposlen kot visokošolski učitelj za biofiziko na Zdravstveni fakulteti Univerze v Ljubljani. Njegovo raziskovanje na področju biofizike bioloških membran obsega predvsem modeliranje in statistično fiziko.