ELEKTROTEHNIŠKI VESTNIK 90(1-2): 60–68, 2023
IZVIRNI ZBNANSTVENI ČLANEK

Uporaba heterogenih računalniških sistemov za molekulsko sidranje

Davor Sluga [1], **Tine Erent** [1], **Marko Jukić** [2,3], **Črtomir Podlipnik** [4], **Nejc Ilc** [1]

[1] *Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Večna pot 113, 1000 Ljubljana, Slovenija*
[2] *Univerza v Mariboru, Fakulteta za kemijo in kemijsko tehnologijo, Smetanova ulica 17, 2000 Maribor, Slovenija*
[3] *Univerza na Primorskem, Fakulteta za matematiko, naravoslovje in informacijske tehnologije, Glagoljaška ulica 8, 6000 Koper, Slovenija*
[4] *Univerza v Ljubljani, Fakulteta za kemijo in kemijsko tehnologijo, Večna pot 113, 1000 Ljubljana, Slovenija*
E-pošta: nejc.ilc@fri.uni-lj.si

Povzetek. Razvili smo prototip vzporednega algoritma za *in silico* molekulsko sidranje, ki omogoča izrabo heterogenih računalniških arhitektur in posledično računsko moč modernih osebnih računalnikov in superračunalniških gruč. Uporabili smo programsko ogrodje OpenCL, ki podpira strojno opremo različnih proizvajalcev. Vzporedni algoritem išče optimalno pozo oziroma konformacijo molekule v receptorskem mestu beljakovine, kar imenujemo molekulsko sidranje, pri tem pa uporablja empirično cenilno funkcijo za ocenjevanje rešitev. Rešitev smo preizkusili na več beljakovinskih kompleksih in dobljene rezultate primerjali z referenčno implementacijo programa CmDock. Analizirali smo optimalnost dobljenih konformacij molekul in hitrost algoritma. Vzporedni algoritem izkazuje izrazito hitrejše delovanje na grafičnih procesnih enotah. Za učinkovitejšo izrabo strojne opreme in večje pohitritve je treba algoritem dodatno optimizirati ter izpopolniti cenilno funkcijo za pridobitev ustreznih vezavnih konformacij ligandov. Programska koda bo dostopna na https://gitlab.com/Jukic/cmdock/.

Ključne besede: molekulsko sidranje, molekulsko modeliranje, načrtovanje učinkovin, genetski algoritem, OpenCL, heterogena računalniška arhitektura, grafične procesne enote, GPE, uHTVS

Employing heterogeneous computer systems for molecular docking

We have developed a prototype parallel molecular docking algorithm that enables exploitation of heterogeneous computer architectures and thus the computational power of modern PCs and supercomputer clusters. We employed the OpenCL software framework that supports hardware from multiple vendors. The parallel algorithm searches for the optimal pose of molecules/ligands in the protein receptor site in an *in silico* experiment named molecular docking. We are using an empirical scoring function to evaluate solutions. The developed algorithm was tested on several protein complexes, and the results were compared with the reference implementation of the CmDock software. We analysed the quality of the resulting ligand poses and the speed of the algorithm to demonstrate a significantly faster performance on GPUs. The reported algorithm leaves space for additional optimizations and development of the scoring function to obtain the optimal ligand poses. The source will be made available at the https://gitlab.com/Jukic/cmdock/.

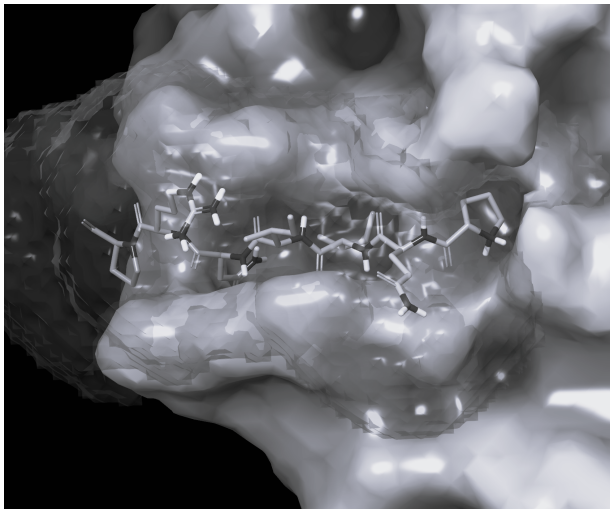
1 *In silico* MOLEKULSKO SIDRANJE

Pri razvoju novih zdravilnih učinkovin raziskovalni laboratoriji uporabljajo robotizirano opremo za visokozmogljivo rešetanje (angl. high-throughput screening), ki omogoča avtomatizirano eksperimentalno *in vitro* biološko vrednotenje različnih spojin z zmogljivostjo več sto tisoč spojin [1]. Gre za razmeroma drag in zapleten postopek, ki pa ga s hitrim razvojem računalniške znanosti dopolnjujejo in nadomeščajo računalniška orodja za molekulsko modeliranje. Podobno kot eksperimente izvajamo *in vivo* ali *in vitro*, tako govorimo o eksperimentih *in silico* oz. v siliciju. Sidranje molekul uvrščamo med metode molekulskega modeliranja in je postalo nepogrešljivo orodje *in silico* odkrivanja novih zdravil ter drugih farmacevtsko-kemijskih raziskav [2]. V kontekstu molekulskega sidranja govorimo tudi o postopkih "navideznega"visokozmogljivega rešetanja (angl. high-throughput virtual screening – HTVS), kjer se preišče ustreznost več milijonov oziroma dandanes več milijard spojin [3]. *In silico* eksperimenti nam tako že v zgodnjih fazah razvoja novih zdravil pomagajo razumeti interakcije molekul zdravil z ustreznimi terapevtskimi tarčami.

Sidranje molekule je postopek vezave molekule oz. liganda v vezavno mesto druge molekule, imenovane receptor ali tarča. Ligand je tipično majhna molekula z nekaj deset atomi, lahko pa sidramo tudi večje, kot so peptidi ali beljakovine z več tisoč atomi. Receptor je navadno makromolekula, denimo beljakovina ali nukleinska kislina [4]. Slika 1 prikazuje primer molekule liganda v vezavnem mestu receptorja.

Na voljo je bogat nabor programskih rešitev za mo-

Prejet 21. december, 2022
Odobren 24. februar, 2023



Slika 1: Primer molekule liganda v vezavnem mestu beljakovine. Aktivno vezavno mesto je prikazano s polprosojno plastjo.

lekulsko sidranje, ki za ugotavljanje optimalne poze liganda v sidrišču receptorja uporabljajo raznolike iskalne algoritme in optimizacijske postopke [5]. Tu navajamo nekaj predstavnikov, ki smo jih razdelili v skupine glede na dostopnost in z zvezdico označili tiste, ki imajo podporo za grafične procesne enote:

- odprtokodni: AutoDock Vina [6], AutoDock-GPU* [7], GeauxDock [8], EDock [9], rDock/rxDock [10], CmDock [11],
- prosti za akademsko rabo: PLANTS [12], UCSF DOCK6 [13], LeDock [14], MedusaDock* [15], AMIDE v2* [16],
- plačljivi: Glide [17], GOLD [18], MOE [19], Surflex-Dock [20].

Primerjalne študije kažejo, da so prosto dostopne rešitve primerljive s plačljivimi – predvsem to velja za sidranje majhnih ligandov [21]. Bolj očitne razlike v korist komercialnim produktom se pokažejo pri sidranju večjih molekul, kot so peptidi ali beljakovine [22].

Proces molekulskega sidranja je računsko zelo zahteven problem, saj moramo preiskati čim večji prostor možnih poz oz. konformacij liganda, poleg tega v raziskavah HTVS preverjamo na milijone/milijarde različnih ligandov. Na srečo lahko problem enostavno razbijemo na medsebojno neodvisne izračune ustreznosti konformacij in tako omogočimo izrabo porazdeljenih in vzporednih računalniških arhitektur. Eden od modelov porazdeljenega računanja, ki se je v praksi izkazal kot zelo zmogljiv za reševanje takšnih problemov, se imenuje prostovoljsko računalništvo (ang. volunteer computing), kjer računska vozlišča predstavljajo računalniške naprave v lasti prostovoljcev.

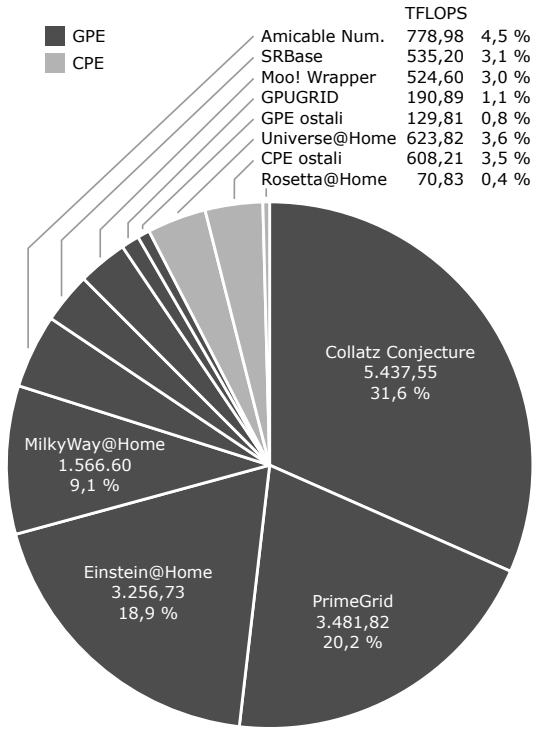
Ta model izhaja iz širšega koncepta skupnostne znanosti (angl. citizen science), ki govori o načinih in načelih sodelovanja širše javnosti v znanstveno-raziskovalni dejavnosti [23]. Posameznik ima tako priložnost, da se dejavno vključi v raziskovalne projekte in tam prispeva svojo strojno opremo, znanje, intelektualne sposobnosti in druge vire za doseganje skupnih družbenih ciljev.

”Skupnostna znanost in boj proti korona virusu – Covid.si” je aktualen projekt slovenskih znanstvenikov, ki od leta 2020 prek računskih zmogljivosti prostovoljcev izvajajo molekulsko sidranje na potencialnih terapevtskih tarčah SARS-CoV-2 [11]. Projekt je prerasel v mednarodno sodelovanje in pod imenom SiDock@home začel gostovati na platformi BOINC (Berkeley Open Infrastructure for Desktop Computing) [24], kjer ga je posvojila širša skupnost ljubiteljskih znanstvenikov. Projekt SiDock@home je tudi eden od prvih znanstvenih porazdeljenih projektov na platformah mrežnih usmernikov v kontekstu paradigme interneta stvari. Na platformi BOINC je že 20 let prisoten eden bolj znanih znanstvenih projektov Folding@home, namenjen zvijanju beljakovin, ki je znan tudi kot eden najzmogljivejših računalniških sistemov na svetu [25] – v aprilu 2020 je dosegel hitrost 2,43 eksaflopsa, kar je več kot prvih 500 najhitrejših superračunalnikov skupaj, in tako postal prvi sistem, ki je presegel mejo eksaflopsa [26].

V zadnjem desetletju se na področju visokozmogljivega računalništva vse bolj premika od klasičnih računalniških arhitektur, kjer celoten posel obdela centralna procesna enota, proti heterogenim sistemom. Moderne heterogene sisteme navadno sestavljajo večjedrne centralne procesne enote (CPE), grafične procesne enote (GPE) in drugi pospeševalniki, denimo programirljivo polje logičnih vrat (FPGA). Med najhitrejšimi 500 superračunalniki jih 170 izkorišča pospeševalnike, med katerimi izrazito prednjačijo GPE [27]. Slednje trenutno prispevajo 37 % skupne računske moči, pri čemer so energijsko bistveno učinkovitejše kot CPE. Najhitrejši in hkrati energijsko najučinkovitejši superračunalnik, ameriški Frontier, sestavljajo vozlišča z eno CPE in štirimi GPE, pri čemer slednje prispevajo več kot 99 % računske moči. Poleg tega okoli 92 % zmogljivosti porazdeljenega sistema BOINC izvira iz projektov, ki izrabljajo GPE, kar vidimo na sliki 2.

Velik potencial heterogenih računalniških arhitektur je trenutno na področju molekulskega sidranja skorajda neizkoriščen, kar velja tudi za orodje CmDock, ki se uporablja v projektu SiDock@home. Gre za programsko opremo, ki teče na enem jedru CPE in bi ji razvita podpora za izvajanje na pospeševalnikih omogočila znatne pohitritve. V članku opisujemo začetni razvoj in prve rezultate vzporednega algoritma za molekulsko sidranje, ki temelji na genetskem algoritmu.

V naslednjem poglavju predstavimo postopek sidranja z orodjem CmDock. V poglavju 3 opišemo predlagani vzporedni algoritem, v poglavju 4 ga ovrednotimo in v poglavju 5 povzamemo prispevke ter podamo izhodišča za nadaljnje delo.



Slika 2: Statistika računske moči projektov BOINC, zajeta 8. 7. 2022. Poimensko prikazujemo prvih deset projektov glede na zmogljivost v teraflopsih.

2 MOLEKULSKO SIDRANJE Z ORODJEM CMDOCK

CmDock je odprtokodno orodje za molekulsko sidranje, alternativa znanemu programu AutoDock [28]. V nasprotju z njim CmDock podpira transparentno uporabo vhodnih molekulskih podatkov, je načrtovan modularno in podpira zelo visoko stopnjo modifikacije eksperimentov ter omogoča načrtovanje protokolov navideznega visokozmogljivega rešetanja. Je edini predstavnik specifično načrtovanih orodij za izvajanje sidranja z uporabo nukleinskih kislin. Sledimo zgledu avtorjev AutoDock, ki so svojemu orodju nedavno dodali podporo za izvajanje na GPE [7].

Pri molekulskem sidranju tako konformacijo molekule določata položaj in orientacija njenih atomov v prostoru. Zaradi velikega števila možnih translacijskih, rotacijskih in konformacijskih premikov liganda in receptorja je problemski prostor velik. Pogosto si pomagamo s poenostavitvijo, kjer je receptor togo telo, katerega konformacija se ne spreminja. Ta, delno prožni model, pogosto predpostavlja tudi program CmDock. Glede na izračunano vrednost cenilne funkcije lahko sklepamo o ustreznosti interakcij med ligandom in tarčo ter postavimo izhodišča za nadaljnje laboratorijske raziskave [2].

Proces molekulskega sidranja tako razdelimo na dva osnovna koraka: 1. napoved ali vzorčenje konformacije molekule liganda (angl. sampling) in 2. ocena ustreznosti interakcije (angl. scoring). Za napoved konformacij redkeje uporabljamo sistematične pristope, pogosteje pa različne optimizacijske algoritme. Če jih naštejemo le nekaj, ti temeljijo na geometrijskem ujemanju molekul, postopni konstrukciji na podlagi fragmentov, iskanju z večkratnimi kopijami ali stohastičnem preiskovanju po metodi Monte Carlo ter z vzorci gibanja živali oziroma z uporabo genetskega algoritma [29]. Za iskanje globalnega optimuma CmDock uporablja genetski algoritem, zatem pa še fino optimizacijo konformacije liganda z metodo simuliranega ohlajanja in hevrističnim algoritmom Nelder-Mead [30]. Vezavno energijo napovedane konformacije liganda ocenjujemo s cenilno funkcijo oziroma cenilko – vsoto interakcij med ligandom in receptorjem. Upoštevamo pa lahko tudi interakcije znotraj molekule liganda in interakcije z različnimi topili.

2.1 Genetski algoritem

Genetski algoritem spada med evolucijske metode, ki iterativno izboljšujejo začetno naključno populacijo osebkov z operacijami mutacije in križanja [31]. V kontekstu molekulskega sidranja je osebek konformacija liganda. Vsak osebek je določen s kromosomom, sestavljenim iz genov. Gen, po analogiji z živim svetom, kodira eno izmed lastnosti osebka. V primeru liganda geni kodirajo njegov položaj, orientacijo in rotacijo morebitnih vrtljivih vezi. Mutacija povzroči naključno spremembo gena, križanje pa zamenjavo naključnega zaporedja genov med dvema kromosomoma. Velikost populacije osebkov je odvisna od dolžine kromosoma, ta pa od števila vrtljivih vezi. Za zapis kromosoma porabimo tri števila v plavajoči vejici za položaj, tri za orientacijo in po eno za vsako vrtljivo vez. Velikost populacije je določena kot $(6 + r_L) * 50$, pri čemer je r_L število vrtljivih vezi liganda.

V vsakem koraku genetskega algoritma se najprej na podlagi izbrane cenilke izbere množica najboljših osebkov, ki postanejo starši novi generaciji. Nad posameznim parom kromosomov izvedemo operaciji mutacije in križanja ter pridobimo dva nova osebka. Tako nastale osebke nato ocenimo in uredimo skupaj z njihovimi starši. Najboljše nato prenesemo v nov korak algoritma. Koraki se izvajajo, dokler ni izpolnjen ustavitveni pogoj – najboljša ocena osebka se ne izboljšuje več ali pa dosežemo vnaprej določeno število korakov. Rezultat genetskega algoritma je tako osebek z najboljšo oceno, ki predstavlja konformacijo liganda z najnižjo vezavno energijo.

Velika večina računskega časa genetskega algoritma se porabi za ocenjevanje osebkov, kjer je treba izračunati interakcije med atomi liganda in receptorja ter atomi znotraj liganda. Za ocenjevanje osebkov lahko uporabimo poljubno cenilno funkcijo, ki lahko identificira morebiti ugodne ali neprimerne konformacije ligandov. Cenilka orodja CmDock razdeli vezavno energijo v več členov, ki opisujejo van der Waalsov potencial, polarne interakcije in morebitne interakcije s topilom.

2.2 Cenilna funkcija na osnovi linearnega potenciala – PLP

Za lažji razvoj vzporednega algoritma smo izbrali cenilko PLANTS$_{PLP}$, ki je med drugim del orodja za sidranje PLANTS [12]. Cenilka predpostavlja model interakcije med atomi, ki za osnovo vzame odsekoma linearno funkcijo potenciala (angl. Piecewise Linear Potential – PLP). Cenilko sestavlja vsota štirih členov:

$$f_{PLANTS} = f_{plp} + f_{clash} + f_{tors} + c_{site} \,. \quad (1)$$

Prvi člen, f_{plp}, opisuje interakcije med atomi liganda in atomi receptorja. Odvisen je od razdalje med paroma atomov in vrsto interakcije. Vrednost celotnega člena je vsota prispevkov vsakega para, kar teoretično pomeni $n_L \times n_R$ izračunov, kjer sta n_L oziroma n_R število atomov liganda oziroma receptorja. Če upoštevamo predpostavko, da je receptor togo telo in se med simulacijo ne spreminja, lahko uporabimo poenostavitev izračuna [32]. Vezavno mesto vzorčimo z mrežo vnaprej določene gostote. Nato v vsaki točki mreže vnaprej izračunamo prispevke f_{plp} za vse možne vrste interakcij. Ko potem v procesu genetskega algoritma ocenjujemo osebek, njegove atome preslikamo na mrežo preko tri-linearne interpolacije in glede na razred posameznih atomov pridobimo njihov vnaprej izračunan energijski prispevek k celokupni vrednosti cenilne funkcije.

Člen f_{clash} opisuje interakcije znotraj molekule liganda in naj bi preprečeval superimpozicije atomov. Upošteva razdaljo med parom atomov liganda in vrsto interakcije med njima. Ta je odvisna od števila vezi med atomoma in od njune vrste. Vrednost celotnega člena je vsota prispevkov vsakega para. Trenutno računamo interakcije med vsemi pari atomov, ki so vsaj tri vezi narazen. Poleg tega upoštevamo samo t.i. težke atome. Računska zahtevnost je kvadratna glede na število težkih atomov liganda, $O({n_L}^2)$. Čas, potreben za izračun cenilke, je tako v veliki meri odvisen od hitrosti izračuna tega člena.

Tretji člen, f_{tors}, opisuje torzijski potencial, ki ga izračunamo za vse vrtljive vezi liganda. Uporabili smo implementacijo torzijskega potenciala, določeno v [33], ki upošteva vezi in tip atomov, ki si vez delita. Vrednost celotnega člena je vsota prispevkov vsake vezi.

Četrti člen, c_{site}, kaznuje poze/konformacije liganda, ki so zunaj vezavnega mesta receptorja. Slednji je volumen poljubne oblike. V nasprotju z [12], kjer so uporabili kroglo, smo kot približek vezavnega mesta vzeli kvader, ki v celoti vsebuje vezavno mesto. Če je neki težek atom liganda zunaj tega kvadra, celotni oceni osebka dodamo kazenski pribitek. Tukaj je možna nadaljnja optimizacija izračuna mreže in volumna vezavnega mesta.

Cilj genetskega algoritma je minimizacija funkcije f_{PLANTS}, ki predstavlja vezavno energijo liganda.

3 VZPOREDNI ALGORITEM

Naš cilj je razviti odprtokodno rešitev za sidranje molekul, ki bo omogočala učinkovito uHTVS ali ultra-visokozmogljivo rešetanje knjižnic spojin. Želimo visoko stopnjevanost, prilagodljivost in elastičnost, kar pomeni, da bo izdelana programska oprema lahko dobro izkoriščala heterogene računalniške arhitekture tudi na nivoju visokozmogljivih, superračunalniških kapacitet. Poleg tega stremimo k neodvisnosti od proizvajalcev strojne opreme in podpori različnim operacijskim sistemom. Iz teh razlogov smo kot programski vmesnik izbrali OpenCL in celoten projekt osredinili na programski jezik C++.

3.1 Standard OpenCL

OpenCL (angl. Open Computing Language) je uveljavljen odprti standard za programiranje heterogenih računalniških sistemov, ki ga razvija neprofitna organizacija Khronos Group [34]. Standard podpira večina največjih proizvajalcev strojne opreme, zato je programska oprema razvita v OpenCL prenosljiva med napravami.

Ogrodje OpenCL predpostavlja enega gostitelja, na katerem se izvaja gostiteljski program, in eno ali več naprav (soprocesorjev, pospeševalnikov), ki so z njim povezane in na katerih se izvaja eden ali več ščepcev (angl. kernel) [35]. Ščepec je torej programska koda, ki jo na napravi vzporedno izvaja množica niti. Gostitelj v vrsto na napravi vstavlja ukaze, denimo izvedi ščepec, kopiraj podatke na napravo ali na gostitelja itd., ki se nato na napravi izvedejo zaporedno. Ščepci se lahko izvajajo na raznolikih napravah, kot so: večjedrne CPE, GPE, vezja FPGA, procesne enote Tensor in procesorji za digitalno obdelavo signalov [34]. V tem delu smo se osredotočili na GPE, ki so najpogostejši pospeševalnik sodobnih visokozmogljivih računalniških sistemov.

Delo ščepca je razdeljeno v delovne skupine (angl. work-group) in delavce (angl. work-item), ki jim pravimo tudi niti (angl. threads). Določiti moramo celotno število delavcev in število delavcev v eni delovni skupini. Delavci znotraj ene delovne skupine si delijo hiter lokalni pomnilnik, med sabo pa jih lahko sinhroniziramo s pomočjo preprek.

Naš vzporedni algoritem izkorišča naravno vzporednost optimizacije z genetskim algoritmom. Slednji je namreč stohastični proces, zato izvedemo nekaj deset njegovih neodvisnih zagonov. Naša rešitev izvaja vse neodvisne zagone hkrati, pri čemer vzporedno obdeluje vse osebke v eni populaciji. Število delavcev je torej zmnožek števila zagonov genetskega algoritma in velikosti populacije. S tem zagotovimo visoko izkoriščenost GPE, ki lahko vzporedno izvaja na tisoče niti.

Za generiranje psevdo-naključnih števil smo uporabili knjižnico RandomCL [36], ki je namenjena izvajanju na GPE in je nekajkrat hitrejša od primerljivih metod.

3.2 Program na gostitelju

Gostitelj prebere in obdela vhodne datoteke, ki vsebujejo podatke o molekulah liganda in receptorja ter definicijo vezavnega mesta. Te podatke prenese na napravo. Prebere tudi datoteke s programsko kodo ščepcev in jih prevede. Nato v vrsto naprave dodaja ščepce v izvajanje in na koncu iz naprave prenese podatke o najboljšem osebku posameznega zagona genetskega algoritma. Izbere rešitev z najbolje ocenjeno konformacijo liganda in jo zapiše v standardnem formatu za shranjevanje kemijskih struktur, SDF (ang. structural data file).

3.3 Pregled ščepcev

Na napravi se najprej samo enkrat izvede skupina ščepcev, ki nastavi generator psevdo-naključnih števil, določi tipe interakcij med atomi in med receptorjevimi atomi izbere samo tiste, ki so v vezavnem mestu ali bližnji okolici. Tu upoštevamo dejstvo, da se z večanjem razdalje med atomoma vpliv med njima zmanjšuje in na neki razdalji postane zanemarljiv. Nato vnaprej izračunamo vse možne prispevke člena f_{plp} v točkah mreže kot smo opisali v poglavju 2.2. Sledi ustvarjanje in ocenjevanje začetne populacije osebkov s cenilko.

Korak genetskega algoritma je sestavljen iz ščepcev, ki izvedejo naslednje operacije: ustvarjanje novih potomcev s križanjem in mutacijo, preslikava kromosomov osebkov na prostorski model molekule liganda, ocenjevanje osebkov, njihovo urejanje in izločevanje dvojnikov ter poročanje gostitelju o najboljši oceni v populaciji. Ta podatek potrebujemo, da ugotovimo konvergenco algoritma: če se v nekaj korakih genetskega algoritma najboljša ocena ne spremeni več, ustavimo optimizacijo.

Na koncu zaženemo ščepec, ki najboljše osebke vsake populacije preslika na model liganda, da se lahko nato prenesejo na gostitelja.

4 REZULTATI

Implementacijo genetskega algoritma v kombinaciji s cenilko PLANTS$_{PLP}$ na GPE smo preizkusili na naboru beljakovinskih kompleksov različnih zahtevnosti sidranja. Na zahtevnost vplivajo število težkih atomov receptorja (n_R) in liganda(n_L) ter število vrtljivih vezi liganda (r_L). Lastnosti posameznih kompleksov so zapisane v tabeli 1. Podatkovna baza uporabljenih kompleksov je dostopna na https://pdbe.org. Referenca je kristalografsko določena eksperimentalna konformacija ligandov. Uporabljeni kompleksi so:

- 3PTB: protein tripsin in zaviralec benzamidin,
- 3CS9: tirozinska kinaza BCR-ABL1 in zaviralec nilotinib – uporablja se pri zdravljenju kronične mieloične levkemije,
- 6O0K: regulator apopotoze BCL-2 in zaviralec venetoklaks, ki je učinkovina za zdravljenje kronične limfocitne levkemije, limfocitnega limfoma ali akutne mieloične levkemije,
- 3NJG: aspartilna proteaza bakterije *Shewanella oneidensis* s peptidnim substratom.

Oznaka PDB	n_R	n_L	r_L
3PTB	1630	9	1
3CS9	2153	39	6
6O0K	1175	61	12
3NJG	5306	58	25

Tabela 1: Testni nabor makromolekulskih kompleksov, kjer je n_R število težkih atomov receptorja in n_L ter r_L število težkih atomov oziroma vrtljivih vezi liganda.

Z eksperimenti poskušamo ugotoviti, kako se algoritem na GPE primerja z referenčnim algoritmom, ki je uporabljen znotraj CmDock in zna izrabljati samo eno jedro CPE. Zanimali sta nas optimalnost konformacij in hitrost izvajanja. V ta namen smo merili odstopanje konformacije ligandov od referenčnih eksperimentalnih konformacij, ki ga izračunamo kot koren vsote kvadratov razdalj ujemajočih atomov (RMSD). Merili smo tudi čas za izvedbo enega koraka genetskega algoritma t nad eno populacijo osebkov in povprečno število korakov genetskega algoritma k, potrebnih za konvergenco konformacije liganda. Iz zadnjih dveh podatkov lahko izračunamo tudi efektivno pohitritev S_e vzporednega algoritma na GPE v primerjavi z referenčnim algoritmom CmDock na CPE:

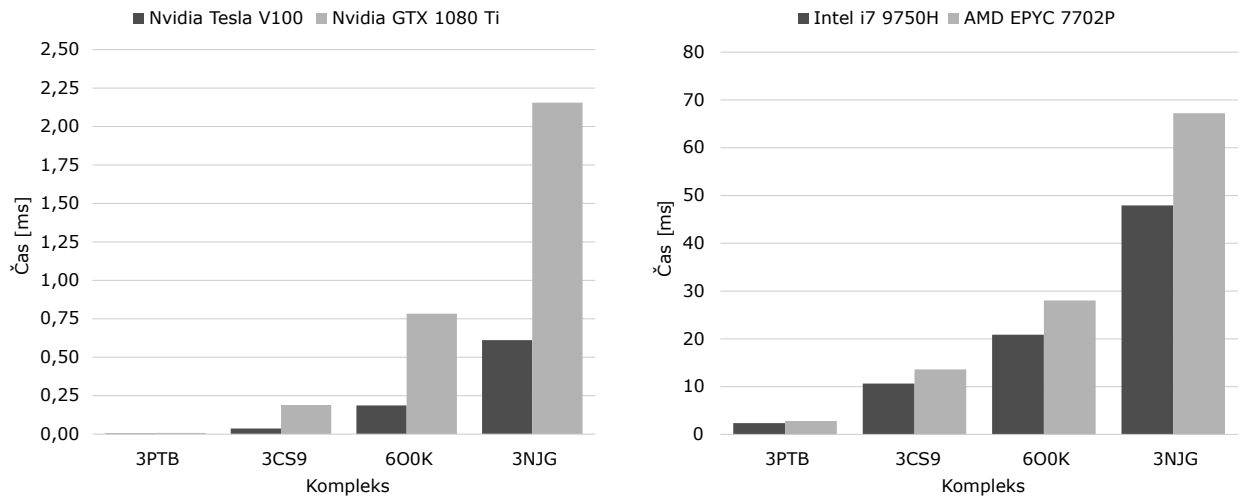
$$S_e = \frac{t_{CPE} * k_{CPE}}{t_{GPE} * k_{GPE}} . \tag{2}$$

Ta nam služi kot glavno merilo hitrosti vzporednega algoritma, saj upošteva dejstvo, da je hitrost konvergence primerjanih algoritmov različna zaradi uporabe različnih cenilk.

Eksperimente smo izvajali na različnih sistemih, da bi zaobjeli širši spekter strojne opreme, ki se lahko uporablja za poganjanje orodja CmDock. Vključili smo procesor, namenjen prenosnikom, Intel i7 9750H [37] s frekvenco ure 2,6 GHz, strežniški procesor AMD EPYC 7702P [38] s frekvenco ure 2 GHz, igričarsko grafično kartico Nvidia GTX 1080 Ti [39] in strežniško grafično kartico Nvidia Tesla V100 [40].

		3PTB	3CS9	6O0K	3NJG
RMSD$_S$	CPE	0,610	0,519	1,784	7,750
	GPE	1,752	0,898	4,385	12,429
RMSD$_{min}$	CPE	0,537	0,519	1,605	4,407
	GPE	0,371	0,898	3,466	8,111
RMSD$_{povp}$	CPE	3,506	9,439	9,635	10,963
	GPE	1,482	9,383	6,955	11,903

Tabela 2: Odstopanje konformacije liganda od referenčne, merjene kot koren vsote kvadratov razdalj ujemajočih atomov – RMSD, pri 200 zagonih algoritma na CPE in GPE. Pri tem je RMSD$_S$ odstopanje konformacije najbolje ocenjenega osebka, RMSD$_{min}$ najnižje odstopanje izmed vseh zagonov in RMSD$_{povp}$ povprečno odstopanje vseh zagonov.



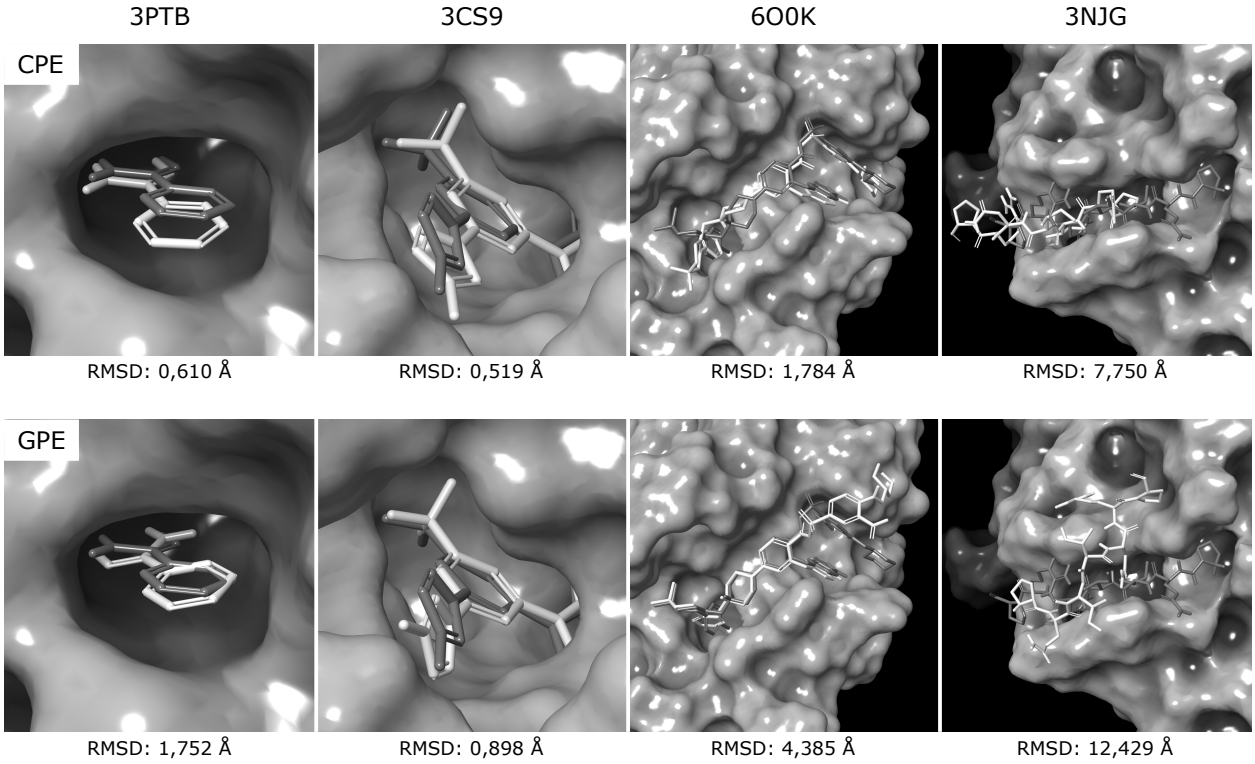
Slika 3: Povprečni časi za izvedbo enega koraka genetskega algoritma implementacije na eni populaciji za CPE (desno) in GPE (levo). Na GPE se je hkrati izvajala optimizacija nad 100 populacijami, medtem ko je bila na CPE aktivna samo ena populacija naenkrat.

Iz tabele 2 vidimo, da konformacije liganda, dobljene z vzporednim algoritmom, bolj odstopajo od referenčnih kot rezultati sidranja na CPE. RMSD_S meri odstopanje najbolje ocenjenega osebka od reference in je v primeru kompleksa 3CS9 primerljiv med izvedenkama. Najbolje ocenjene konformacije prikazuje slika 4, kjer vidimo, da je pri najenostavnejšem kompleksu, 3PTB, rešitev vzporednega algoritma zrcaljena optimalna konformacija. Razlika med referenčno konformacijo in konformacijo, dobljeno na GPE, je pri kompleksu 3NJG pričakovano največja – ligand se je sidral na popolnoma napačno mesto, ki pa je bilo sprejemljivo glede na poenostavljeno definicijo mesta sidranja cenilke PLANTS_{PLP}. Rezultati so predvsem odraz enostavnosti cenilke PLANTS_{PLP}, ki se uporablja pri algoritmu na GPE. Tako RMSD_{min} kot RMSD_{povp} kažeta podobno sliko, le da sta obe implementaciji glede na zadnjega precej bolj izenačeni.

Tabela 3 prikazuje povprečno število korakov, ki so potrebni, da genetski algoritem konvergira, torej da doseže optimalno konformacijo liganda. Konvergenca je dosežena, kadar se v šestih zaporednih korakih najboljša ocena med vsemi osebki ne izboljša več. Vidimo, da algoritem na GPE potrebuje precej več korakov za izpolnitev tega pogoja, kar lahko pripišemo preprosti cenilni funkciji, ki manj natančno opisuje interakcije med atomi.

Uporaba GPE in enostavnejše cenilne funkcije se odraža v precej krajšem času, potrebnem za izvedbo enega koraka genetskega algoritma nad eno populacijo. Spomnimo, da zaradi stohastičnosti postopka genetski algoritem zaženemo nad več populacijami. Program na GPE obdeluje osebke več populacij hkrati in tako bolje izkoristi masovno-vzporedno arhitekturo GPE. Iz končnega nabora dobljenih osebkov izberemo najboljšega. Na sliki 3 so prikazani časi za izvedbo enega koraka genetskega algoritma nad eno populacijo, pri čemer je bilo na GPE hkrati aktivnih 100 populacij, medtem ko je CPE izvajal genetski algoritem samo nad eno populacijo naenkrat. Posamezne populacije se torej na CPE optimizirajo zaporedoma. Iz primerjave časov na CPE vidimo, da se mobilni procesor Intel i7 9750H obnese bolje kot strežniški AMD EPYC 7702P, predvsem na račun višje frekvence ure. Strežniški procesorji so optimizirani za izvajanje večjega števila vzporednih niti/procesov, in zato ne blestijo v aplikacijah, ki tečejo na samo enem jedru procesorja, kot je primer pri CmDock. Primerjava časov na obeh grafičnih karticah nakazuje, da je strežniška GPE, Nvidia Tesla V100, za faktor 2- do 3-krat hitrejša od igričarske Nvidia GTX 1080 Ti. Slednja je osnovana na eno generacijo starejši arhitekturi, tako da je rezultat pričakovan. Poleg tega je tudi veliko cenejša (ob izdaji 700 USD proti 11000 USD za V100) in manj primerna za konstantno obremenitev v strežniških okoljih. Iz grafa je razvidno tudi, da čas procesiranja hitreje narašča s kompleksnostjo problema pri Nvidia GTX 1080 Ti, predvsem na račun manjše količine procesnih enot (3584 proti 5120 jeder CUDA) in nižje pasovne širine pomnilnika.

Primerjava med algoritmoma na CPE in GPE nam pokaže, da je slednji veliko hitrejši tako na račun večje zmogljivosti procesiranja kot tudi enostavnejše cenilke. Vendar je tu treba upoštevati tudi število korakov, potrebnih za konvergenco. Bolj nazorno nam to prikazuje tabela 4, v kateri najdemo efektivne pohitritve obeh grafičnih kartic v primerjavi s CPE. Efektivna pohitritev se precej spreminja v odvisnosti od zahtevnosti problema. Na najenostavnejšem kompleksu GPE doseže več kot 40-kratno pohitritev, medtem ko je ta pri najbolj kompleksnem 6,7-kratna. Tu se kažejo ozka



Slika 4: Odstopanje RMSD_S najbolje ocenjene konformacije liganda (prikazano z belo) od referenčne (prikazano s sivo) za različne komplekse. Prva vrsta prikazuje rezultat sidranja na CPE, druga pa na GPE.

grla izvedbe vzporednega algoritma, ki jih bo treba odpraviti z dodatnimi optimizacijami. Gre predvsem za razpršene dostope do pomnilnika in nezadostno izkoriščanje predpomnilnika na GPE. Kljub temu rezultati nakazujejo obetavnost uporabe GPE za pospeševanje molekulskega sidranja, saj že trenutna implementacija občutno skrajša čas procesiranja. Z izboljšavo in dopolnitvijo cenilne funkcije ter dodatnimi optimizacijami pa lahko pričakujemo še veliko boljše rezultate, tako kvalitativno kot tudi časovno.

	3PTB	3CS9	6O0K	3NJG
CPE	46,1	44,2	43,4	47,5
GPE	705,0	908,8	760,9	554,8

Tabela 3: Povprečno število korakov genetskega algoritma potrebnih za izpolnitev pogoja konvergence za posamezen kompleks.

	3PTB	3CS9	6O0K	3NJG
Tesla V100	40,5	14,1	6,4	6,7
GTX 1080 Ti	19,9	2,7	1,5	1,9

Tabela 4: Efektivna pohitritev S_e v primerjavi s CPE Intel i7 9750H. Na GPE se je hkrati izvajala optimizacija nad 100 populacijami, medtem ko je bila na CPE aktivna samo ena populacija naenkrat.

5 Zaključek

Rezultat našega dela je prototip vzporednega algoritma za izvedbo molekulskega sidranja ter uHTVS, ki temelji na genetskem algoritmu in cenilni funkciji PLANTS_{PLP}. Uporabili smo programsko ogrodje OpenCL, ki omogoča poganjanje algoritma na heterogenih računalniških sistemih in izkoriščanje grafičnih procesnih enot za pospešitev hitrosti izvajanja.

Vzporedni in zaporedni algoritem, ki je del obstoječega orodja CmDock, smo testirali na raznovrstni strojni opremi in beljakovinskih kompleksih ter primerjali rezultate kot tudi čas izvajanja obeh algoritmov. Rezultati so obetavni, saj se čas procesiranja zmanjša celo za faktor 40 v primerjavi z izvajanjem zaporednega algoritma na CPE.

Pri razvoju vzporednega algoritma je še precej prostora za izboljšave, tako pri izpopolnitvi cenilne funkcije in definiciji prostora receptorja (izračun mreže) kot pri dodatnih optimizacijah kode, ki se izvaja na GPE. Uporabljena cenilka PLANTS_{PLP} je preveč enostavna, da bi lahko učinkovito vodila iskanje optimalne konformacije liganda, sploh če gre za večje spojine. V nadaljnjem delu bomo za izvajanje na vzporednih arhitekturah prilagodili njeno izpopolnjeno različico $\text{PLANTS}_{CHEMPLP}$ [12] in cenilko, ki je del orodja CmDock. Identificirali smo ozka grla in možne rešitve za njihovo odpravo, kot so izboljšanje vzorca dostopov do globalnega pomnilnika, boljša izraba predpomnilnika na GPE in zmanjšanje

časovne zahtevnosti izračuna člena f_{clash} cenilne funkcije. Naš končni cilj je vključitev podpore za GPE in druge pospeševalnike v aplikacijo CmDock in s tem boljše izkoriščanje računskih kapacitet, ki so na voljo na platformi BOINC in v superračunalniških gručah.

ZAHVALA

Raziskavo so sofinancirali Javna agencija za raziskovalno dejavnost Republike Slovenije v okviru programa P2-0241 – Sinergetika kompleksnih sistemov in procesov in Republika Slovenija ter Evropska unija iz Evropskega socialnega sklada prek operacije Študenti UL v delovnem okolju.

LITERATURA

[1] V. Blay, B. Tolani, S. P. Ho in M. R. Arkin, "High-Throughput Screening: today's biochemical and cell-based approaches", *Drug Discovery Today*, let. 25, št. 10, str. 1807–1821, 2020. DOI: 10.1016/j.drudis.2020.07.024.

[2] L. Pinzi in G. Rastelli, "Molecular Docking: Shifting Paradigms in Drug Discovery", *International Journal of Molecular Sciences*, let. 20, št. 18, str. 4331, 2019. DOI: 10.3390/ijms20184331.

[3] N. Nikitina, M. Manzyuk, Č. Podlipnik in M. Jukić, "Volunteer Computing Project SiDock@home for Virtual Drug Screening Against SARS-CoV-2", v *Computer Science Protecting Human Society Against Epidemics*, A. Byrski, T. Czachórski, E. Gelenbe, K. Grochla in Y. Murayama, ur., zv. 616 IFIP, Cham: Springer International Publishing, 2021, str. 23–34, ISBN: 978-3-030-86582-5. DOI: 10.1007/978-3-030-86582-5_3.

[4] F. Stanzione, I. Giangreco in J. C. Cole, "Use of molecular docking computational tools in drug discovery", v *Progress in Medicinal Chemistry*, D. R. Witty in B. Cox, ur., zv. 60, Elsevier, 2021, str. 273–343. DOI: 10.1016/bs.pmch.2021.01.004.

[5] N. S. Pagadala, K. Syed in J. Tuszynski, "Software for molecular docking: a review", *Biophysical reviews*, let. 9, št. 2, str. 91–102, 2017.

[6] O. Trott in A. J. Olson, "AutoDock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading", *Journal of computational chemistry*, let. 31, št. 2, str. 455–461, 2010.

[7] D. Santos-Martins, L. Solis-Vasquez, A. F. Tillack, M. F. Sanner, A. Koch in S. Forli, "Accelerating AutoDock4 with GPUs and Gradient-Based Local Search", *Journal of Chemical Theory and Computation*, let. 17, str. 1060–1073, 2 2021. DOI: 10.1021/acs.jctc.0c01006.

[8] Y. Fang, Y. Ding, W. P. Feinstein in sod., "GeauxDock: Accelerating structure-based virtual screening with heterogeneous computing", *PLoS ONE*, let. 11, 7 2016. DOI: 10.1371/journal.pone.0158898.

[9] W. Zhang, E. W. Bell, M. Yin in Y. Zhang, "EDock: blind protein–ligand docking by replica-exchange Monte Carlo simulation", *Journal of Cheminformatics*, let. 12, št. 1, str. 37, 2020. DOI: 10.1186/s13321-020-00440-9.

[10] S. Ruiz-Carmona, D. Alvarez-Garcia, N. Foloppe in sod., "rDock: a fast, versatile and open source program for docking ligands to proteins and nucleic acids", *PLoS Comput Biol*, let. 10, št. 4, e1003571, 2014.

[11] M. Bahun, M. Jukić, D. Oblak in sod., "Inhibition of the SARS-CoV-2 3CLpro main protease by plant polyphenols", *Food Chemistry*, let. 373, str. 131 594, 2022. DOI: 10.1016/j.foodchem.2021.131594.

[12] O. Korb, T. Stutzle in T. E. Exner, "Empirical scoring functions for advanced protein-ligand docking with PLANTS", *Journal of chemical information and modeling*, let. 49, št. 1, str. 84–96, 2009.

[13] W. J. Allen, T. E. Balius, S. Mukherjee in sod., "DOCK 6: Impact of new features and current docking performance", *Journal of computational chemistry*, let. 36, št. 15, str. 1132–1156, 2015.

[14] H. Zhao in A. Caflisch, "Discovery of ZAP70 inhibitors by high-throughput docking into a conformation of its kinase domain generated by molecular dynamics", *Bioorganic & medicinal chemistry letters*, let. 23, št. 20, str. 5721–5726, 2013.

[15] M. Fan, J. Wang, H. Jiang in sod., "GPU-Accelerated Flexible Molecular Docking", *Journal of Physical Chemistry B*, let. 125, str. 1049–1060, 4 2021, MedusaDock GPU. DOI: 10.1021/acs.jpcb.0c09051.

[16] P. Darme, M. Dauchez, A. Renard in sod., "AMIDE v2: High-Throughput Screening Based on AutoDock-GPU and Improved Workflow Leading to Better Performance and Reliability", *International Journal of Molecular Sciences*, let. 22, str. 7489, 14 2021, Inverse docking with AMIDEv2 on GPU. DOI: 10.3390/ijms22147489.

[17] R. A. Friesner, R. B. Murphy, M. P. Repasky in sod., "Extra Precision Glide: Docking and Scoring Incorporating a Model of Hydrophobic Enclosure for Protein–Ligand Complexes", *Journal of Medicinal Chemistry*, let. 49, št. 21, str. 6177–6196, 2006. DOI: 10.1021/jm051256o.

[18] G. Jones, P. Willett, R. C. Glen, A. R. Leach in R. Taylor, "Development and validation of a genetic algorithm for flexible docking11Edited by F. E. Cohen", *Journal of Molecular Biology*, let. 267, št. 3, str. 727–748, 1997. DOI: 10.1006/jmbi.1996.0897.

[19] C. R. Corbeil, C. I. Williams in P. Labute, "Variability in docking success rates due to dataset preparation", *Journal of computer-aided molecular design*, let. 26, št. 6, str. 775–786, 2012.

[20] R. Spitzer in A. N. Jain, "Surflex-Dock: Docking benchmarks and real-world application", *Journal of computer-aided molecular design*, let. 26, št. 6, str. 687–699, 2012. DOI: 10.1007/s10822-011-9533-y.

[21] Z. Wang, H. Sun, X. Yao in sod., "Comprehensive evaluation of ten docking programs on a diverse set of protein–ligand complexes: the prediction accuracy of sampling power and scoring power", *Physical Chemistry Chemical Physics*, let. 18, št. 18, str. 12 964–12 975, 2016.

[22] A. S. Hauser in B. Windshügel, "LEADS-PEP: A Benchmark Data Set for Assessment of Peptide Docking Performance", *Journal of Chemical Information and Modeling*, let. 56, št. 1, str. 188–200, 2016. DOI: 10.1021/ACS.JCIM.5B00234/SUPPL_FILE/CI5B00234_SI_001.PDF.

[23] European Citizen Science Association, *Ten Principles of Citizen Science*, 2015. DOI: 10.17605/OSF.IO/XPR2N.

[24] D. P. Anderson, "BOINC: A Platform for Volunteer Computing", *Journal of Grid Computing*, let. 18, št. 1, str. 99–122, 2020. DOI: 10.1007/s10723-019-09497-9.

[25] V. Curtis, "Patterns of Participation and Motivation in Folding@home: The Contribution of Hardware Enthusiasts and Overclockers", *Citizen Science: Theory and Practice*, let. 3, št. 1, str. 1–14, 2018.

[26] M. I. Zimmerman, J. R. Porter, M. D. Ward in sod., "SARS-CoV-2 simulations go exascale to predict dramatic spike opening and cryptic pockets across the proteome", *Nature Chemistry*, let. 13, št. 7, str. 651–659, 2021. DOI: 10.1038/s41557-021-00707-0.

[27] "TOP500.org - June 2022". (7. jul. 2022), spletni naslov: https://www.top500.org/lists/top500/2022/06/.

[28] G. M. Morris, R. Huey, W. Lindstrom in sod., "AutoDock4 and AutoDockTools4: Automated docking with selective receptor flexibility", *Journal of Computational Chemistry*, let. 30, št. 16, str. 2785–2791, 2009. DOI: 10.1002/jcc.21256.

[29] T. Meza Menchaca, C. Juárez-Portilla in R. C. Zepeda, "Past, Present, and Future of Molecular Docking", v *Drug Discovery and Development - New Advances*, V. Gaitonde, P. Karmakar in A. Trivedi, ur., London: IntechOpen, 2020, pogl. 2. DOI: 10.5772/intechopen.90921.

[30] W. Press, S. Teukolsky, W. Vetterling in B. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 2007, ISBN: 9780521880688.

[31] J. Brest, V. Žumer in M. Sepesy Maučec, "Velikost populacije pri algoritmu diferencialne evolucije", *Elektrotehniški vestnik*, let. 74, št. 1/2, str. 55–60, 2007.

[32] G. Wu, D. H. Robertson, C. L. Brooks in M. Vieth, "Detailed analysis of grid-based molecular docking: A case study of CDOCKER - A CHARMm-based MD docking algorithm", *Journal of Computational Chemistry*, let. 24, št. 13, str. 1549–1562, 2003. DOI: 10.1002/jcc.10306.

[33] M. Clark, R. D. Cramer III in N. Van Opdenbosch, "Validation of the general purpose tripos 5.2 force field", *Journal of Computational Chemistry*, let. 10, št. 8, str. 982–1012, 1989. DOI: https://doi.org/10.1002/jcc.540100804.

[34] "OpenCL - The Khronos Group Inc." (7. jul. 2022), spletni naslov: https://www.khronos.org/opencl/.

[35] I. Ramovš, S. Gerkšič in U. Lotrič, "Dualna hitra gradientna metoda na grafičnih procesnih enotah", *Elektrotehniški Vestnik*, let. 86, št. 4, str. 219–224, 2019.

[36] T. Ciglarič, R. Češnovar in E. Štrumbelj, "An OpenCL library for parallel random number generators", *The Journal of Supercomputing*, let. 75, št. 7, str. 3866–3881, 2019. DOI: 10.1007/s11227-019-02756-2.

[37] Intel Corporation. "Intel Core i7 9750H". (7. jul. 2022), spletni naslov: https://www.intel.com/content/www/us/en/products/sku/191045/intel-core-i79750h-processor-12m-cache-up-to-4-50-ghz/specifications.html.

[38] Advanced Micro Devices Inc. "AMD EPYC 7702P". (7. jul. 2022), spletni naslov: https://www.amd.com/en/products/cpu/amd-epyc-7702p.

[39] NVIDIA Corporation. "GEFORCE GTX 1080". (7. jul. 2022), spletni naslov: https://www.nvidia.com/en-sg/geforce/products/10series/geforce-gtx-1080.

[40] NVIDIA Corporation. "NVIDIA V100 Tensor Core GPU". (7. jul. 2022), spletni naslov: https://www.nvidia.com/en-us/data-center/v100.

Davor Sluga je leta 2017 doktoriral na Univerzi v Ljubljani s področja računalništva. Na Fakulteti za računalništvo in informatiko Univerze v Ljubljani je zaposlen kot asistent. raziskovalno se ukvarja z vzporednim programiranjem, strojnim učenjem in visoko-zmogljivim računalništvom.

Tine Erent je leta 2022 diplomiral na Fakulteti za računalništvo in informatiko Univerze v Ljubljani.

Marko Jukić Marko Jukič je leta 2016 doktoriral in je docent s področja farmacevtske kemije. Je ustanovitelj bioinformacijskega podjetja IntelliMol. Zaposlen je na Fakulteti za matematiko, naravoslovje in informacijske tehnologije Univerze na Primorskem in Fakulteti za kemijo in kemijsko tehnologijo Univerze v Mariboru ter se ukvarja z načrtovanjem novih zdravilnih učinkovin, bioinformatiko in razvojem računalniških orodij za načrtovanje novih zdravil.

Črtomir Podlipnik je leta 2003 doktoriral na Univerzi v Ljubljani s področja teoretične kemije. Od leta 2011 je docent na Fakulteti za kemijo in kemijsko tehnologijo Univerze v Ljubljani. Raziskovalno deluje na področjih modeliranja molekul in kemoinformatike.

Nejc Ilc je leta 2016 doktoriral na Univerzi v Ljubljani s področja računalništva. Od leta 2021 dalje je docent na Fakulteti za računalništvo in informatiko Univerze v Ljubljani. Na raziskovalnem področju se zanima za strojno učenje, vzporedne računalniške sisteme in bioinformatiko.