UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matej Rus

Vzporedna simulacija problema N teles

DIPLOMSKO DELO
NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: doc. dr. Patricio Bulić
Somentor: znan. svet. dr. Milan Hodošček

Ljubljana, 2012

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00181/2011
Datum: 07.11.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATEJ RUS**

Naslov: **VZPOREDNA SIMULACIJA PROBLEMA N TELES**
PARALLEL N-BODY SIMULATION

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Problem N teles je računski problem, pri katerem je potrebno izračunati medsebojne sile med telesi v množici. Problem zahteva veliko število enakih računskih operacij. V okviru diplomske naloge se osredotočite na simulacijo Coulombove sile in Lennard-Jonesovega potenciala med atomi vode. Simulacijo implementirajte z uporabo vmesnika OpenCL, ki nam omogoča programiranje za vzporedne računalniške sisteme. Simulacijo poženite na računalniškem sistemu z več jedri in na računalniškem sistemu z grafično procesno enoto. Ocenite pri katerem številu jeder se paralelna implementacija izplača.

Mentor:

doc. dr. Patricio Bulić

Dekan:

prof. dr. Nikolaj Zimic

Somentor:

znan. svet. dr. Milan Hodošček

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani Matej Rus,

z vpisno številko 63070175,

sem avtor diplomskega dela z naslovom:

Vzporedna simulacija problema N teles

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom

  doc. dr. Patricia Bulića

  in somentorstvom

  znan. svet. dr. Milana Hodoščka

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 26.4.2012 Podpis avtorja:

Posvetilo

Diplomsko delo posvečam svojim staršem.

Kazalo

Pomen oznak

- CPE … Centralna procesna enota (ang. CPU)
- GPE … Grafična procesna enota (ang. GPU)
- *N* … Velikost problema oz. število atomov v skupini, ki jo opazujemo
- CPI … Cycles Per Instruction (število urinih ciklov, porabljenih za izvajanje enega ukaza)
- DDR3 (SDRAM) … Double Data Rate type three (3) Synchronous Dynamic RAM
- GDDR5 (SGRAM) … Graphics Double Data Rate, version 5

Dogovor

Del programa, ki nadzoruje izvajalno okolje OpenCL, se izvaja na gostitelju (ang. *host*), paralelni del pa na napravi (ang. *device*). Ker v našem primeru izvajamo paralelni del programa na CPE, sta oba, gostitelj in naprava, kar CPE. V bolj splošnem primeru velja, da je gostitelj CPE, naprava pa navadno GPE.

Povzetek

Cilj tega dela je pohitriti računanje medsebojnih interakcij med atomi vode, pri katerem se uporablja pristop z grobo silo. Pohitritev želimo doseči z uporabo programskega vmesnika OpenCL, ki nam omogoča učinkovitejšo izrabo večjedrne CPE.

OpenCL (ang. *Open Computing Language*) je programski vmesnik za paralelno programiranje, neodvisen od strojne platforme. Uporaba OpenCL bistveno izboljša hitrost izvajanja na račun učinkovite izrabe visoko paralelne arhitekture današnjih procesnih enot.

Pri računanju medsebojnih interakcij lahko napovemo gibanje skupine atomov. Ker moramo za vsak par atomov izračunati njuno medsebojno silo in razdaljo je časovna zahtevnost problema $O(N^2)$.

Ugotovimo, da je paralelizacija z uporabo OpenCL ustrezna, saj bistveno pohitri izvajanje. Prav tako ugotovimo, da je pri večjem obsegu atomov GPE še bistveno hitrejša.

Ključne besede: atom, OpenCL, problem *N* teles, centralna procesna enota

Abstract

The objective of this work is to speed up computation of interactions between atoms of water, which uses the brute force approach. We want to speed up computations using OpenCL API, which allows more efficient use of multicore CPUs.

OpenCL (Open Computing Language) is a programming interface for parallel programming, independent of the hardware platforms. Using OpenCL significantly improves performance due to efficient use of highly parallel architecture of today's CPUs and GPUs.

When calculating interactions we can predict the movement of water atoms. Since we have to calculate the force and distance for each pair of atoms time complexity of the problem is $O(N^2)$.

We find out that parallelisation using OpenCL significantly speeds up calculations on multicore CPU. We also find that with bigger sets of atoms GPU is significantly faster.

Keywords: atom, OpenCL, *N*-body problem, central processing unit

1. Uvod

Leta 1714 je angleški parlament zaradi pomorskih nesreč razpisal visoke nagrade za natančno metodo določanja zemljepisne dolžine. Nesreče so namreč bile posledica napačne navigacije, natančno navigiranje pa je tesno povezano z napovedovanjem časa [13].

Isaac Newton je leta 1687 v svoji znameniti knjigi *Philosophiae Naturalis Principia Mathematica* rešil problem dveh teles [13]. Če opazujemo več kot dve telesi se matematični opis obnašanja našega sistema oteži do te stopnje, da ga ne moremo opisati v zaključeni formi.

Pri merjenju časa na angleških ladjah je bila nihajna ura neuporabna. Newton je zato predlagal določanje časa s pomočjo položaja Lune. Newton in pomorščaki so naleteli na prvi zabeležen problem *N* teles, saj niso mogli dovolj natančno določiti obnašanja Lune pod vplivom Zemlje in Sonca (problem treh teles). Dovolj natančno rešitev so ob pomoči računalnika predstavili šele leta 1954 ob izzidu popravljenih tabel lege Lune. Nove tabele so vsebovale skupno kar 1627 popravkov in so med drugim omogočale uspešen pristanek odprave Apollo na Luni leta 1969 [13].

Danes, ko razpolagamo s tako zmogljivo računalniško opremo, napovedovanje obnašanja nekaj teles v skupini ne predstavlja več velikega izziva. Kljub skokovitemu napredku centralnih in grafičnih procesnih enot, pa se nam v znanosti vedno odpirajo novi problemi, za katere bi lahko rekli, da so podobni, kot so jih imeli Angleži leta 1714, le obseg delcev je bistveno večji.

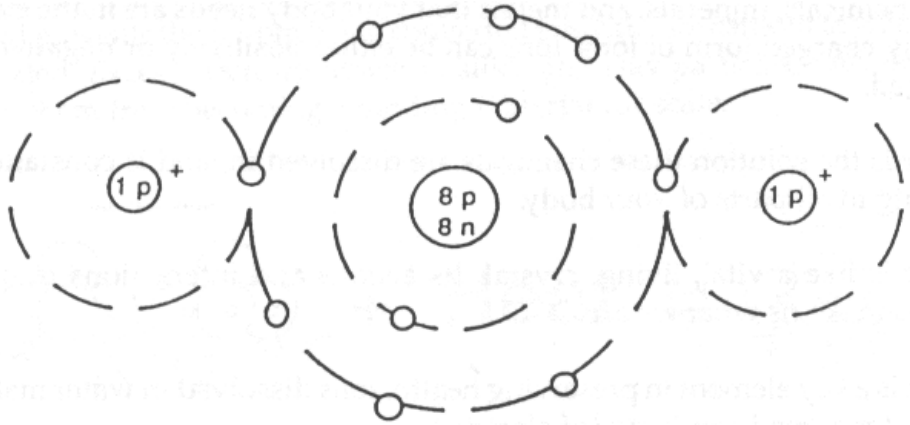
2. Voda

Voda je verjetno najbolj poznana in razširjena kemijska spojina na Zemlji. Prekriva 70.9% površine planeta. Voda je pri sobni temperaturi brezbarvna tekočina brez vonja in okusa.

2.1. Kemijske lastnosti vode

Molekulo sestavlja električno nevtralna skupina vsaj dveh atomov, ki ju povezuje kovalentna vez. Molekula vode (molekulska formula H_2O ali HOH) nastane, kot lahko vidimo na sliki 1, ko se posamezna atoma vodika (lat. *hydrogenium*) vežeta s centralnim kisikovim (lat. *oxygenium*) atomom preko para skupnih elektronov. Nastane kovalentna vez, za katero je značilno, da se elektroni porazdelijo med atomi.

Kisikov atom ima po vezavi, od skupno šestih, še vedno štiri proste elektrone v zunanji tj. drugi lupini (v prvi lupini ima kisik samo dva elektrona). Ker prosta para elektronov ostaneta v bližini kisikovega jedra, izpodrivata kovalentna para elektronov. Posledično se vez med kisikovim in vodikovima atomoma izkrivi, pri čemer znaša kot H-O-H namesto 180° samo še 104.5°. Od tod izhaja značilna "V" oblika vodne molekule [9].



Slika 1 [14]: Shematski prikaz vodne molekule.

Kisikov atom ima drugo najvišjo stopnjo elektronegativnosti (višjo ima samo še fluor - univalenten plinski halogen). Elektronegativnost je kemijska lastnost, ki označuje sposobnost atoma, da v kovalentni vezi pritegne valenčne elektrone. Posledično ima molekula asimetrično porazdeljen naboj (polarna molekula).

Območje s pozitivnim nabojem (na predelu, kjer se nahajata vodikova atoma) v eni molekuli bo zato pritegnilo ostala območja sosednjih molekul z negativnim nabojem (del s kisikovim

atomom). Več molekul vode se tako poveže preko vodikove vezi. Vsaka vodna molekula se lahko preko vodikovih vezi poveže z največ štirimi sosednjimi molekulami. Vodikove vezi so bistveno šibkejše od kovalentnih, zato se ohranijo samo nekaj milisekund, posledično se mrežna struktura vode neprestano spreminja [9].

2.2. Fizikalne lastnosti vode

Voda je edina spojina, ki ima v naravi prisotna vsa tri agregatna stanja - tekoče, trdno (led) in plinasto (para). Pri normalnem atmosferskem tlaku 1.01325 bara ima tališče pri 0°C (32°F) in vrelišče pri 99.98°C (211.97°F). Višja kot je nadmorska višina (in s tem nižji tlak), nižje je vrelišče. Molska masa vode znaša 18.01528 g/mol [6].

Voda ima vrsto anomalnih lastnosti, ki so posledica vodikove vezi. Pri 4°C doseže voda svojo najvišjo gostoto (1000 kg/m^3), ki pa pri 99.98°C znaša samo še 958.5 kg/m^3 [6]. Molekule so v trdnem agregatnem stanju bistveno bolj razmaknjene kot v tekočem. Število vodikovih vezi, ki jih lahko tvorijo molekule, se zmanjšuje z višanjem temperature. Če torej v trdnem agregatnem stanju začnemo dovajati toploto, se začnejo vodikove vezi med molekulami trgati, posledično se poveča tudi gostota. Več toplote kot dovedemo, hitreje se posamezne molekule gibljejo in s tem tvorijo vse manj novih vezi.

Ravno anomalne lastnosti so tiste, ki omogočajo življenje v vodi. V zimskem času se namreč ozračje, in s tem zemlja, ohlaja. Ker je zrak bolj hladen kot zemlja, se voda najbolj ohladi na površini. Ko se voda na površini ohladi do 4°C zaradi višje gostote potone na dno ter na površino izrine preostalo vodo z višjo temperaturo. Proces teče vse, dokler nima vsa voda 4°C. Ko se ozračje še bolj ohladi, začne voda na površini zmrzovati - nastane led, vendar zaradi nižje gostote (led ima približno 9% nižjo gostoto) ne potone na dno. Pod plastjo ledu lahko zato organizmi preživijo nizke temperature. Če bi bila gostota ledu višja od 1000 kg/m^3, bi na dno potonil led in izpodrinil preostalo vodo. Z ohlajanjem ozračja bi zato kmalu zamrznila vsa voda [11].

3. Problem *N* teles

Problemi, ki obravnavajo medsebojne interakcije med telesi v množici, so pogosto "trn v peti" v mnogih vejah znanosti. Pri *N* telesih je potrebno izračunati medsebojne sile za vsak par teles v množici (od tod tudi ime problem *N* teles). Brez sofisticiranih algoritmov, torej s pristopom z grobo silo, moramo obravnavati vsaj *N(N-1)* interakcij. Časovna odvisnost problema je zato $O(N^2)$.

Pri številu delcev, primernem za simulacijo realnih razmer, se pogosto izkaže, da je problem časovno preveč zahteven. Iz računskega stališča je zato problem simulacije nekaj tisoč atomov vode, ki skupaj zasedejo le nekaj nanometrov, enak simulaciji obnašanja planetarnega sistema.

3.1. Coulombov zakon

Coulombov zakon opisuje elektrostatsko interakcijo med električno nabitimi delci [5].

Sila med dvema naelektrenima delcema je proporcionalna produktu nabojev q_1, q_2 ter inverzno proporcionalna kvadratu razdalje med njima. Opisana je z enačbo (1), kjer sta r_1 in r_2 krajevna vektorja nabojev q_1 in q_2. V našem primeru nas zanima samo velikost, ne pa tudi smer vektorja.

$$\vec{F}_{q12} = C\frac{q_1 q_2(\vec{r_1} - \vec{r_2})}{\left|\vec{r_1} - \vec{r_2}\right|^3} \quad (1)$$

Coulombova sila je privlačna, če sta naboja q_1 in q_2 različno predznačena in odbojna, če sta predznaka enaka. Konstanta C je odvisna od izbire merskega sistema in znaša v našem primeru 332.0716. Enačba je eksaktno pravilna le, če delca obravnavamo kot točkasti telesi (njun premer je zanemarljiv v primerjavi z medsebojno razdaljo).

Razdalja med atomom A_i in A_j predstavlja dolžino daljice, ki atoma A_i, A_j povezuje. Razdaljo med atomom A_i s koordinatami (x_i, y_i, z_i) ter atomom A_j s koordinatami (x_j, y_j, z_j) lahko v kartezičnem koordinatnem sistemu izračunamo s pomočjo enačbe (2).

$$\left|\vec{r_j} - \vec{r_i}\right|^2 = (x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2 \quad (2)$$

Torej večja kot sta naboja atoma, večja bo njuna elektrostatska sila, ter večja kot bo razdalja med njima, manjša bo sila (sila pada s kvadratom razdalje). Če bi torej želeli predvideti, kam

se bo atom vode premaknil v naslednjem trenutku znotraj skupine preostalih *N-1* atomov, bi bilo potrebno izračunati podatke o interakciji izbranega atoma z vsemi preostalimi (torej *N* računskih operacij).

3.2. Lennard-Jonesov potencial

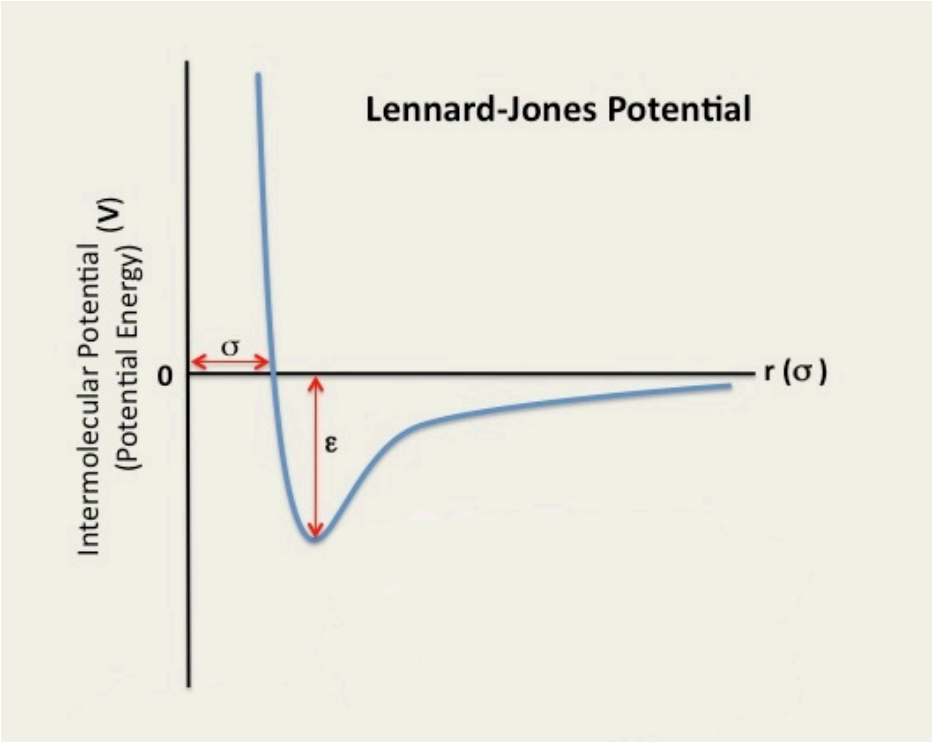
Zamislimo si, da imamo v prostoru dve majhni gumijasti kroglici. Prvotno sta razmaknjeni in ne vplivata ena na drugo (razdalja med njima je neskončna). Pri neskončni razdalji je njuna potencialna vezna energija enaka nič (poenostavljamo!). V tem primeru je potrebno dovesti le minimalno silo *F*, da se kroglici začneta pomikati bližje skupaj. Kroglici se približujeta vse dokler ne dosežeta razdalje *R,* na kateri prične delovati privlačna sila. Privlačna sila še dodatno približa telesi do njune ravnovesne lege (kroglici se sedaj dotikata ali pa sta v neposredni bližini). V kolikor bi želeli razdaljo med kroglicama še bolj zmanjšati, bi potrebovali neprimerno večjo silo. Kroglici bi se začeli prekrivati, vendar pa bi bila odbojna sila bistveno večja od privlačne. Podobno kot s kroglicami se dogaja tudi z atomi in molekulami. Dogajanje natančneje opisuje Lennard-Jonesov (L-J) potencial [10].

$$V = 4\varepsilon\left[(\frac{\sigma}{r})^{12} - (\frac{\sigma}{r})^{6}\right] \quad (3)$$

Lennard-Jonesov potencial je matematični model (3), ki opisuje obnašanje energije med dvema nepovezanima atomoma v odvisnosti od njune razdalje. Enačba upošteva privlačne (dipol-dipol, dipol-induciran dipol) in odbojne sile, ki nastopajo pri gibanju atomov.

Potencial ima dva člena. Prvi, privlačni, ki pada s šesto potenco razdalje (opisuje van der Waalsovo silo), ter drugi, odbojni, ki pada z dvanajsto potenco razdalje med atomoma in opisuje medsebojni odboj atomov. Do odboja pride, kadar sta atoma tako blizu skupaj, da se njuni elektronski ovojnici prekrivata [10].

Parameter ε, označuje minimum potencialne energije (globino oz. kako zelo se atoma privlačita). Parameter σ (tudi van der Waalsov radij) označuje razdaljo, pri kateri je potencial med dvema nepovezanima atomoma enak nič (σ je torej enak polovični razdalji med jedri dveh atomov). Pri vodi znaša ε=0.650 kJ/mol in σ=0.316 nm. Zadnji parameter *r* označuje razdaljo oz. oddaljenost prvega atoma od drugega (merjeno iz središča delca) [10].



Slika 2 [10]: Grafični prikaz Lennard-Jonesovega potenciala.

Slika 2 prikazuje, da je nemogoče, da bi se dva atoma popolnoma prekrila tj. združila, saj bi potencial presegel vse meje, kar ni mogoče. Ko jedri popolnoma prekritih atomov oddaljujemo med seboj, se potencial eksponentno zmanjšuje, dokler v ravnovesni legi pri razdalji σ ne doseže vrednosti 0. Svoj minimum doseže potencial v točki ε, kasneje pa se z večanjem razdalje med atomoma vrednost potenciala zopet približuje 0 (od tod potencial 0 pri neskončni razdalji).

3.3. III. Newtonov zakon

Pri zaporednem izvajanju, tj. izvajanje na enem samem procesorskem jedru, lahko upoštevamo III. Newtonov zakon ali zakon o akciji in reakciji. Zakaj Newtonovega zakona ne moremo uporabljati tudi pri paralelnem izvajanju, bomo razložili nekoliko kasneje, kjer spoznamo bolj podrobno pomnilniški model OpenCL.

Trije Newtonovi zakoni opisujejo gibanje teles. Tretji zakon trdi, da če deluje prvo telo na drugega s silo F, potem tudi drugo telo deluje na prvo z nasprotno enako silo $-F$ [4]. Če smo torej izračunali kako atom A_i vpliva na atom A_j, nam ni potrebno ponovno izračunati vpliva atoma A_j na atom A_i, saj bo velikost sile enaka, le obratno usmerjena. Število interakcij (N^2) se nam zato ravno razpolovi.

	A_1	A_2	A_3	A_4
A_1	0	*If*	*If*	*If*
A_2	***Ir***	0	*If*	*If*
A_3	***Ir***	***Ir***	0	*If*
A_4	***Ir***	***Ir***	***Ir***	0

Tabela 1: Prikaz računskih interakcij med posameznimi atomi.

V zgornji tabeli 1 so z oznakami A_1, A_2, A_3 in A_4 prikazani štirje atomi v skupini, med katerimi se računa Coulombova sila. V primeru, da opazujemo molekule vode, je število atomov v skupini vedno deljivo s tri, saj je posamezna molekula sestavljena iz treh atomov. Diagonalnih elementov nam ni potrebno računati, saj je razdalja in s tem sila med njimi vedno enaka nič. Interakcije *If* predstavljajo nujne računske operacije, ki jih je potrebno opraviti, da lahko predvidimo obnašanje izbranih atomov. Rdeče obarvane interakcije *Ir* označujejo dodatno redundančno računanje, ki ga je potrebno izvesti, če ne upoštevamo III. Newtonovega zakona. Iz tabele 1 lahko vidimo, da se nam obseg računanja elektrostatskih sil ob upoštevanju III. Newtonovega zakona zmanjša ravno za polovico.

4. Centralna procesna enota

Procesor je osrednji del računalniškega sistema, ki po navodilih računalniškega programa izvaja osnovne aritmetično-logične ter vhodno/izhodne operacije.

Prvi računalniki, med katere spada tudi ENIAC (ang. *Electronic Numerical Integrator and Calculator*), so bili fiksno ožičeni. ENIAC je bil desetiški stroj, programiranje pa je bilo ročno s postavljanjem stikal in povezovanjem kablov. V skupini, ki je razvijala računalnik ENIAC, se je porodila ideja o računalniku, ki bi imel program shranjen v glavnem pomnilniku. Kmalu po predstavitvi ENIACa je zato sledil EDVAC (ang. *Electronic Discrete Variable Automatic Computer*). EDVAC je bil sposoben izvajanja več različnih splošnih računskih operacij, zaporedje ukazov (program) pa je hranil v hitrem glavnem pomnilniku. V kolikor je programer želel izvajati drugačno zaporedje operacij, je moral zgolj spremeniti vsebino programa v glavnem pomnilniku namesto spreminjanja ožičenja. Danes za tovrstne stroje prevladuje oznaka von Neumannov računalnik [2]. Sledilo je obdobje proizvodnje ozko specializiranih CPE, katero pa se je kasneje z razvojem integriranih vezij (ang. *integrated circuit*) umaknilo splošno namenskim CPE. Prve generacije CPE, okoli leta 1970, so temeljile na veliki množici manjših integriranih vezij.

Mikroprocesorji temeljijo na le nekaj oz. navadno na samo enem integriranem vezju. Izvedba mikroprocesorja na enem integriranem vezju prinaša krajše preklopne čase in manjšo parazitivno kapacitivnost. Posledično lahko dosežemo frekvence nekaj GHz.

Kljub intenzivnemu razvoju in pojavu večjedrnih CPE še vedno velja von Neumannov model računanja. Žal ne moremo preprosto zmanjševati tehnologije proizvodnjega procesa (trenutno 32 nm, Ivy Bridge 25 nm). Pri zelo majhni (nekaj nm) velikosti se pojavljajo negativne posledice (ang. *electromigration* in *subthreshold leakage*). Poleg tega pa je A. P. Chandrakasan s sodelavci v svojem članku [1] pokazal, da bi z zamenjavo enega jedra, ki deluje s frekvenco *f*, z dvem jedroma, ki delujeta s frekvenco *f/2*, porabo energije zmanjšali za dobrih 60 procentov. Razvoj je zato danes usmerjen v nove tehnologije vzporednega in porazdeljenega, kot tudi kvantnega računanja.

4.1. Paralelnost v CPE

Paralelizem nam omogoča, da razbijemo večji problem v množico manjših neodvisnih delov, ki se nato paralelno izvedejo na dani strojni opremi. Pri razvoju CPE stremimo k čim manjši vrednosti CPI. V idealnem primeru bi želeli porabiti za en ukaz samo eno urino periodo.

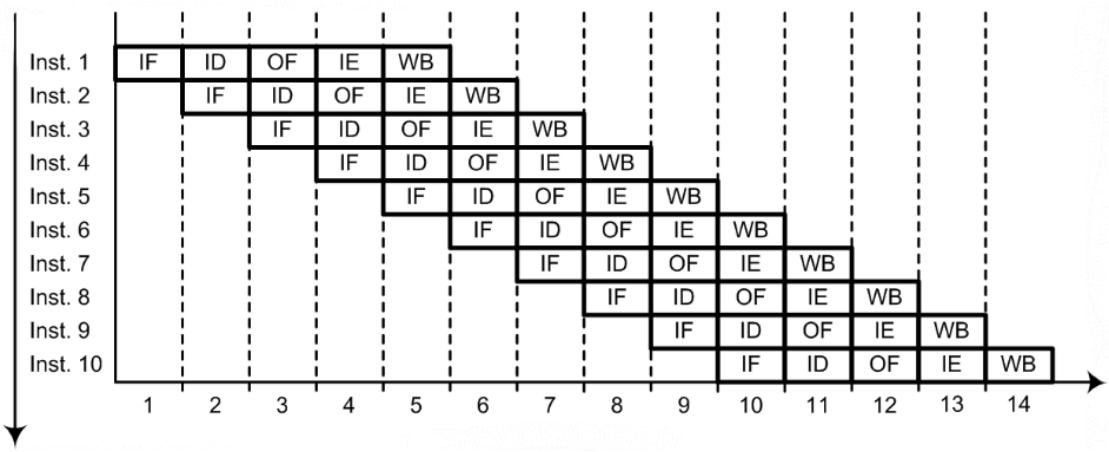
Približno do leta 1986 se je izboljševala zmogljivost s podvajanjem dolžine pomnilne besede

(ang. *bit-level parallelism*). Daljša kot je beseda, večji obseg podatkov lahko CPE obravnava v enem urinem ciklu. Tako je Intelov prvi komercialno uspešen 8-bitni procesor Intel 8080 za seštevanje dveh 8-bitnih operandov potreboval izvesti samo eno operacijo. Intel 4004, 4-bitni procesor, pa je za to potreboval dve računski operaciji [7].

CPE lahko izvaja v danem trenutku zgolj eno računsko operacijo nad enim ali dvema operandoma. Med izvrševanjem trenutnega ukaza, vse dokler se trenutni ne izvrši do konca, čaka v vrsti naslednji ukaz. Izvajanje enega ukaza navadno vzame več kot eno urino periodo (naslednji ukaz zato čaka več urinih period, preden se dejansko izvrši). Da bi vsaj delno odpravili to pomankljivost, imajo danes že vsi modernejši procesorji vgrajen cevovod (ang. *instruction pipeline*). Cevovod, shema katerega je prikazana na sliki 3, je način izkoriščanja paralelizma na nivoju ukazov (ang. *instruction-level parallelism*). Vsaka stopnja cevovoda mora opraviti svoje delo v eni urini periodi [2]. Posamezne korake pri izvrševanju ukaza pa razdelimo v posamezne stopnje cevovoda. Na sliki 3 smo izvrševanje ukaza razdelili na pet stopenj.

1) Prevzem ukaza iz pomnilnika (**IF** - instruction fetch)
2) Dekodiranje ukaza (**ID** – instruction decode)
3) Dostop do operandov (**OF** - operand fetch)
4) Izvrševanje ukaza (**IE** - instruction execute)
5) Pisanje nazaj v pomnilnik (**WB** - write back)

Te stopnje so tipične za današnje računalnike in jih srečamo tudi pri resničnih strojih [2].

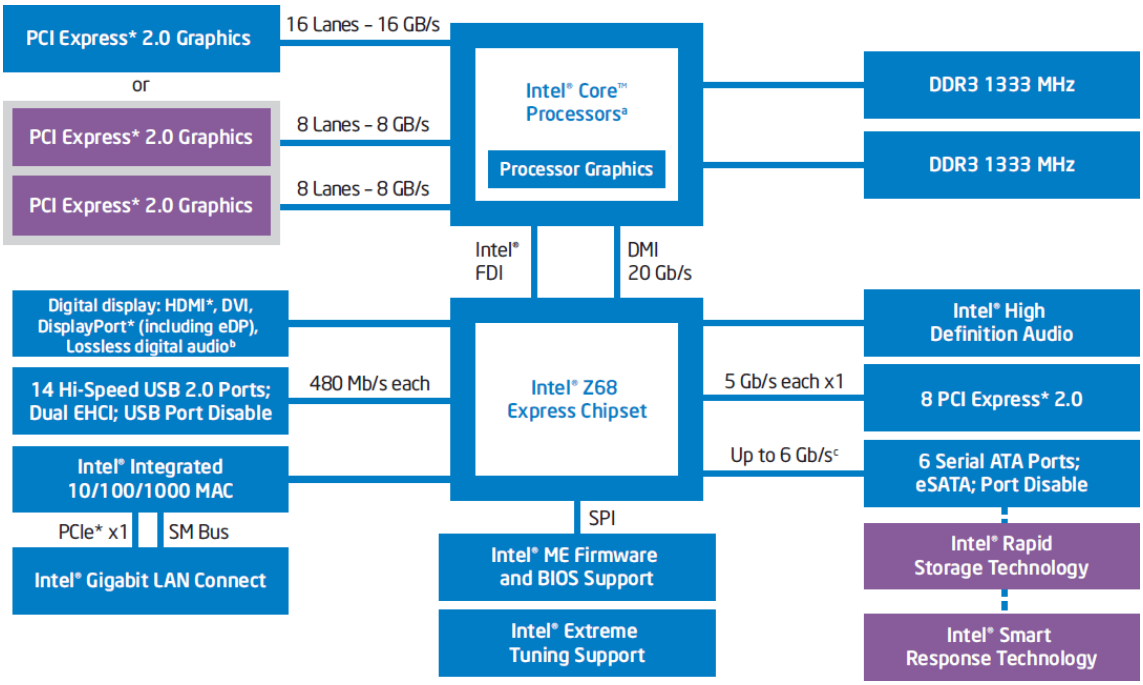


Slika 3 [12]: Shematski prikaz delovanja 5-stopenjskega cevovoda. Na osi x so prikazane urine periode, na osi y pa ukaz s svojo zaporedno številko.

Če imamo petstopenjski cevovod, imamo v vsakem trenutku lahko skupno največ pet ukazov v izvajanju na različnih stopnjah v cevovodu, kar pomeni v idealnih razmerah 5-krat nižji CPI. Za računalnike, ki uporabljajo veliko število stopenj z enostavnimi podoperacijami, se

uporablja ime supercevovodni računalniki. Eden od najbolj znanih primerov supercevovodnih računalnikov so Intelovi 80x86 procesorji, katerih prvi predstavnik je bil Pentium 4 iz leta 2000. Prve izvedbe so imele 20-stopenjski cevovod, naslednje pa celo 31-stopenjskega [2]. Večstopenjski cevovod pa nam ne zagotavlja nujno višje pohitritve, saj se pri daljših cevovodih pojavlja vedno večja verjetnost cevovodnih nevarnosti (pojav strukturnih, operandnih in kontrolnih nevarnosti).

V želji po doseganju še nižjega CPI so zato izdelali daljši superskalarni cevovod, ki je vseboval več ločenih enakih enot za izvajanje ukazov (ang. *execution unit*). Superskalarni cevovod lahko hkrati prevzame več ukazov, dodatna logika pa določi, ali so ukazi neodvisni in se jih zato lahko izvrši hkrati. Superskalarni cevovod deluje torej podobno, kot da bi imeli na razpolago več običajnih neodvisnih cevovodov.



Slika 4 [8]: Shema arhitekture Sandy Bridge.

Drugi od načinov doseganja večje zmogljivosti je vzporedno izvajanje večjega števila niti (ang. *threads*). Program se na CPE izvaja kot množica ukazov, torej kot nekakšno podatkovno zaporedje, kar lahko poimenujemo tudi nit. Aplikacija lahko generira eno, nekatere pa tudi več ločenih niti. Enojedrna CPE lahko v danem trenutku obravnava samo eno nit, torej samo en podatkovni tok. Sistem zato z veliko hitrostjo menjava med nitmi in uporabniku ustvarja vtis večopravilnosti.

V kolikor imamo več CPE znotraj enega sistema, lahko v danem trenutku obravnavamo več kot eno nit, kar seveda znatno poveča hitrost izvajanja (ang. *multiprocessing*). Če si razpoložljive CPE v računalniku delijo enak glavni pomnilnik, govorimo o simetričnem

multiprocesiranju (ang. *symmetric multiprocessing*). Ker je pri simetričnem multiprocesiranju glavni pomnilnik koherenten, si lahko posamezne CPE delijo niti enega programa. Za razliko od sistemov NUMA (ang. *Non-Uniform Memory Access*), so sistemi, ki izrabljajo prednosti simetričnega multiprocesiranja navadno omejeni na manjše število CPE. Če vgradimo na en silicijev čip več CPE, govorimo o večjedrni CPE. Večjedrna CPE ima enoten glavni pomnilnik in L3 predpomnilnik ter ločen L1 in L2 predpomnilnik.

Reševanja problema smo se lotili na večjedrnih procesorjih Intel Core. Vsi so predstavniki 32 nm arhitekture Sandy Bridge (shema prikazana na sliki 4). Vse CPE podpirajo Intelovo tehnologijo Hyper-Threading. Tehnologija Hyper-Threading omogoča enemu jedru CPE, da se le-to predstavi operacijskemu sistemu kot dvojedrna CPE. Procesor z enim fizičnim jedrom tako prikaže operacijskemu sistemu dve logični jedri, dvojedrna CPE pa prikaže operacijskemu sistemu kar štiri logična jedra. Operacijski sistem in uporabniški programi lahko med razporejanjem svojih niti izrabljajo vsa logična jedra CPE. Intelov Hyper-Threading omogoča, da se ukazi iz posameznega logičnega jedra hkrati izvajajo na enem skupnem fizičnem jedru CPE [15].

5. Sekvenčni algoritem

5.1. Specifikacije sistemov

Da bodo podani rezultati in pohitritve smiselne, podajamo na tem mestu specifikacije vseh treh sistemov, ki so se uporabljali pri računanju. Na sistemu 1 in 2 smo programsko kodo pognali samo na CPE. Sistem 3 smo uporabili izključno za primerjavo med hitrostjo izvajanja programa na CPE in GPE.

Sistem 1 (Macbook Pro 13" early 2011)

- CPE: 2.3 GHz Intel Core i5-2415M (2 jedri, 3MB L3 predpomnilnika)
- GPE: Intel HD Graphics 3000 384 MB DDR3
- Pomnilnik: 4 GB 1333 MHz DDR3

Sistem 2 (Macbook Pro 15" early 2011)

- CPE: 2.0 GHz Intel Core i7-2635QM (4 jedra, 6MB L3 predpomnilnika)
- GPE: AMD Radeon HD 6490M 256MB GDDR5/Intel HD Graphics 3000 384 MB DDR3
- Pomnilnik: 4GB 1333 MHz DDR3

Sistem 3 (iMac 21.5” mid 2011)

- CPE: 2.5 GHz Intel Core i5 (4 jedra, 6MB L3 predpomnilnika)
- GPE: AMD Radeon HD 6750 512MB GDDR5
- Pomnilnik: 4GB 1333 MHz DDR3

Na vseh treh računalnikih je bil nameščen enak operacijski sistem Mac OS X verzija 10.7.2 (11C74) in razvojno okolje Xcode 4.2 (build 4D199).

5.2. Sekvenčna rešitev

Da dobimo boljši občutek o časovni zahtevnosti problema, smo najprej primerjali izvajalne čase sekvenčnega programa na sistemu 1 in sistemu 2.

V našem primeru posamezen atom opisujejo štiri spremenljivke. Imamo torej $4N$ različnih vrednosti. Posamezen atom je v računalniškem programu, poenostavljeno, predstavljen kot struktura.

```
typedef struct{
    float x;
    float y;
    float z;
    float c;
}atom;
```

Pri čemer prve tri vrednosti predstavljajo njegovo lokacijo v kartezičnem koordinatnem sistemu (koordinate *x*, *y* in *z*), zadnja pa njegov naboj. Naboj je lahko pozitiven ali negativen in pri vodi zaseda samo dve vrednosti: 0.417 za vodikova atoma in -0.834 za kisikov atom.

Coulombovo silo in energijo, ki nastopa v vsaki interakciji med paroma atomov v računalniškem programu prav tako predstavlja struktura. Struktura, ki akumulira rezultate enega atoma, vsebuje dodatne štiri spremenljivke.

```
typedef struct{
    float r;
    float dx;
    float dy;
    float dz;
}force;
```

Računski čas se, kot lahko vidimo v tabeli 2, pri sekvenčnem izvajanju, ko koristimo samo eno jedro, ne razlikuje bistveno med obema sistemoma.

N	sistem 1 [s]	sistem 2 [s]	razlika [s]
10.125	0.47	0.47	0.0
27.783	2.70	2.63	-0.07
50.625	8.13	7.92	-0.21
107.811	34.51	35.71	+1.2

Tabela 2: Primerjava sekvenčnih izvajalnih časov na obeh sistemih.

Človeški genom, za primerjavo, sestavljajo štiri vrste nukleotidov: Adenin (A), Gvanin (G), Citozin (C) in Timin (T). Molekula DNK (deoksiribonukleinska kislina) je sestavljena iz 3 bilijonov omenjenih nukleotidnih parov. Molekula Timina, najbolj preprostega nukleotida, je sestavljena iz 34 atomov. Če bi naša DNK vsebovala samo pare Timina, bi veriga DNK obsegala približno 204 bilijonov atomov. Pri tako obsežnem naboru podatkov bi sekvenčni računski del trajal že kar nepredstavljivo dolgo.

6. Paralelizacija algoritma

6.1. OpenCL

OpenCL je nov in popularen standard za paralelno procesiranje. Za razliko od ostalih metod paralelizacije podpira bistveno širši spekter različnih naprav. Ker je OpenCL platformno neodvisen, lahko naš algoritem poganjamo tako na večjedrni CPE, kakor tudi na grafični kartici ali celo na nekaterih zmogljivejših mobilnih telefonih. Neodvisnost nam pride še posebej prav, kadar nimamo na razpolago visoko paralelnih GPE [3].

OpenCL je sestavljen iz OpenCL izvajalnega okolja (ang. API – *Application programming interface*) in posebnega programskega jezika (temelji na standardu C99) za pisanje ščepca (ang. *kernel*), ki se izvaja na razpoložljivih napravah. Ščepec, ki se izvrši na napravi, spominja na funkcije programskega jezika C, pri čemer se pred ime doda atribut `__kernel`. Pred argumente ščepca pa je potrebo dodati atribut, s katerim povemo, kje v pomnilniku naprave se bo informacija, podana preko argumenta, nahajala (z dostopnim določilom `__global`, `__local`,…). V spodnjem primeru ščepec preko klica funkcije `get_global_id(0)` zgolj vrne trenutni indeks niti.

```
__kernel void calculate_forces(const __global float4* atoms,
                               __global float4* forces,
                               const __global int* iac,
                               const int kN){

    int i=get_global_id(0);

}
```

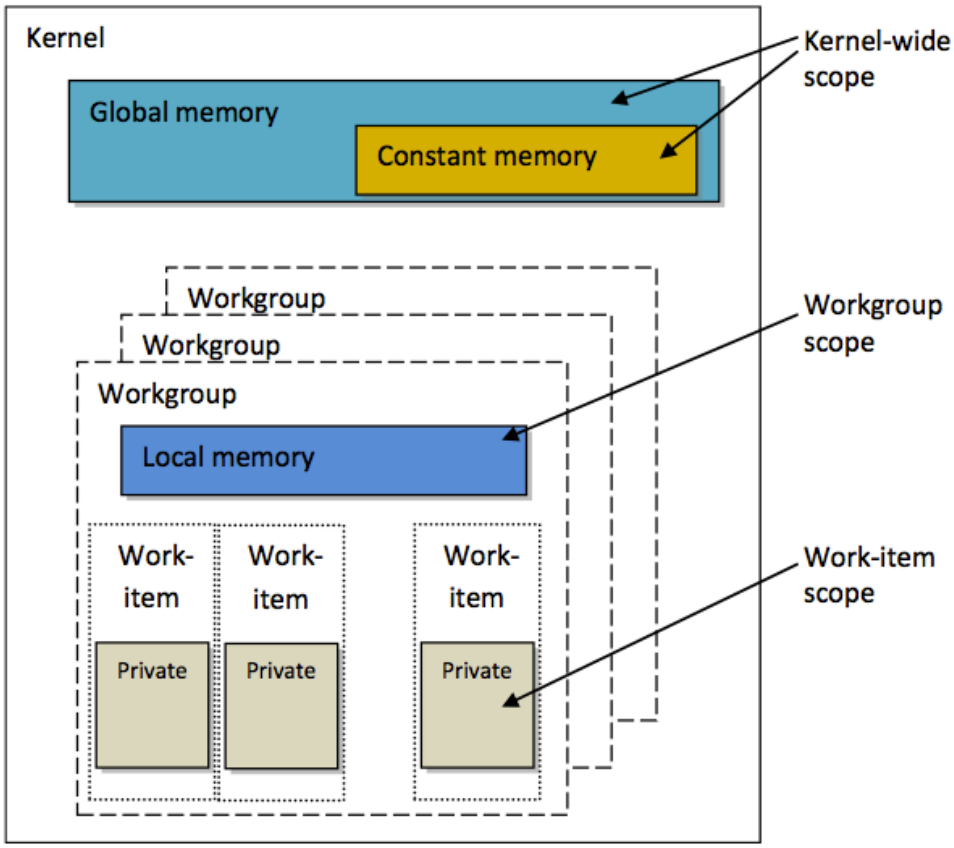
Pogosto imamo datoteko, ki predstavlja ščepec, shranjeno ločeno (navadno jo označimo s končnico ".cl"). Gre pravzaprav za tekstovno datoteko s programsko kodo, ki se izvrši na napravi in jo gostitelj preko OpenCL API med izvajanjem tudi prevede, vendar pa nas o morebitnih sintaktičnih napakah v jedru ne obvesti.

6.2. Pomnilniški model OpenCL

OpenCL ima zaradi strojne neodvisnosti definiran abstrakten večnivojski pomnilniški model, ki vsebuje štiri različne tipe pomnilniškega prostora [3].

Prvi in najmanjši v hierarhiji je privatni pomnilniški prostor (ang. *private memory*). Uporablja ga lahko ena sama računska enota (nit ang. *work item*) bodisi za bralni ali pisalni dostop. V

našem primeru ga nit uporablja za hranjenje trenutne vrednosti energije in sile. Več niti je združenih v bloke (ang. *workgroup*). Na sliki 5 imamo shematsko prikazane tri bloke. Niti združujemo v bloke zato, da se posamezni bloki lahko izvršijo neodvisno in v poljubnem vrstnem redu. To nam omogoča boljšo prilagodljivost pri izvajanju na različnih napravah. Niti znotraj enega bloka si lahko delijo njim skupen lokalni pomnilniški prostor (ang. *local memory*), na sliki 5 označen s temno modro barvo. Lokalni prostor je navadno namenjen sinhronizaciji niti in hranjenju skupnih spremenljivk. Zadnji in največji v skupini je globalni pomnilnik (ang. *global memory*). Bralno/pisalni dostop do globalnega pomnilnika je omogočen vsem enotam. Del globalnega pomnilnika je namenjen hranjenju konstant (ang. *constant memory*) in omogoča samo bralni dostop. Če je privatni pomnilniški prostor zelo majhen, drag in hiter, je globalni pomnilnik velik, cenovno ugoden ter bistveno večji.



Slika 5 [3]: Pomnilniški model OpenCL.

V globalni pomnilnik naprave smo naložili podatke o koordinatah, naboju in masi posameznih atomov ter v zadnjem koraku vanj prenesli naše rezultate. Zaradi štirih vrst pomnilniškega prostora, smo se bili primorani odreči uporabi III. Newtonovega zakona, saj je problem N teles v našem primeru samo delno podatkovno neodvisen. Težava namreč nastopi, ker niti znotraj enega bloka ne morejo komunicirati z nitmi v naslednjem bloku, poleg tega je privatni pomnilnik prve niti neviden drugi niti. Če torej nit T_i izračuna elektrostatsko silo in razdaljo med atomoma A_i in A_j, v našem primeru ne moremo zagotoviti, da bo nit T_j lahko

dostopala do rezultata niti T_i. Nit T_j mora zato ponoviti celoten računski del, pri čemer dobi enak, le obratno predznačen rezultat kot nit T_i.

6.3. Izvajalni model OpenCL

Na strani gostitelja je pred izvajanjem ščepca na napravi potreben niz klicev različnih funkcij, katere pripravijo izvajalno okolje.

Preko klica funkcije `clCreateContext()` ustvarimo kontekst, ki vsebuje informacije o eni ali več napravah, na katerih se bo izvajal ščepec. Vsaka naprava ima svojo ukazno vrsto, ki jo ustvarimo preko klica funkcije `clCreateCommandQueue()`. Ker imamo v našem primeru samo eno napravo, imamo samo eno ukazno vrsto.

```
// Create an OpenCL context
     cl_context context=clCreateContext( NULL, 1, &device_id, NULL, NULL,
&ret);

// Create a command queue
     cl_command_queue command_queue=clCreateCommandQueue(context,
device_id, 0, &ret);
```

Na napravi (večjedrni CPE) rezerviramo potreben prostor za podatke o atomih (lokacija in naboj). Rezervirati je potrebno tudi prostor za rezultat. Enodimenzionalne tabele s podatki o atomih in rezultati hranimo v globalnem pomnilniku naprave, saj mora do njih imeti bralno/pisalni dostop vsaka nit.

```
// Create memory buffers on the device for each vector
     cl_mem atoms_mem_obj=clCreateBuffer(context, CL_MEM_READ_ONLY, N *
sizeof(cl_float4), NULL, &ret);
     cl_mem iac_mem_obj=clCreateBuffer(context, CL_MEM_READ_ONLY, N *
sizeof(cl_float4), NULL, &ret);
     cl_mem forces_mem_obj=clCreateBuffer(context, CL_MEM_WRITE_ONLY, N *
sizeof(cl_float4), NULL, &ret);
```

Gostitelj programsko kodo (ščepec), ki se izvaja na napravi in je shranjena v ločeni datoteki, naloži in prevede. Izhodni status, različen od `CL_SUCCESS`, označuje, da je med prevajanjem prišlo do napake. V kolikor ne preverjamo izhodnega statusa, nam gostitelj ne vrne nobene podrobnosti o morebitni napaki med prevajanjem ščepca.

```
// Create a program from the kernel source
     cl_program program=clCreateProgramWithSource(context, 1, (const char
```

```
**)&source_str, (const size_t*)&source_size, &ret);

// Build the program
     const char * opt="-cl-fast-relaxed-math";
     ret=clBuildProgram(program, 1, &device_id, opt, NULL, NULL);
     if(ret!=CL_SUCCESS) printf("ERROR (build): %d\n",ret);

// Create the OpenCL kernel
     cl_kernel kernel = clCreateKernel(program, "calculate_forces", &ret);
     if(ret!=CL_SUCCESS) printf("ERROR (create): %d\n",ret);
```

Pred dejanskim izvajanjem ščepca na napravi je potrebno prenesti podatke o atomih v globalni pomnilnik naprave. Količino podatkov, ki jih moramo prenesti, lahko kar izračunamo. Podatkovni tip `float` nam vzame 4B, vsak atom pa je predstavljen s štirimi spremenljivkami. Prenesti moramo torej 4*N**4B = 1724976B.

```
// Copy array of atoms to their respective memory buffers
     ret=clEnqueueWriteBuffer(command_queue, atoms_mem_obj, CL_TRUE, 0, N *
sizeof(cl_float4), atoms, 0, NULL, NULL);
     ret=clEnqueueWriteBuffer(command_queue, iac_mem_obj, CL_TRUE, 0, N *
sizeof(int), iac, 0, NULL, NULL);

// Execute the OpenCL kernel
     size_t global_item_size=N;
     ret=clEnqueueNDRangeKernel(command_queue, kernel, 1, NULL,
&global_item_size, NULL , 0, NULL, NULL);
     if(ret!=CL_SUCCESS) printf("ERROR (execute): %d\n",ret);
```

Po zadnjem koraku se izvrši ščepec na napravi. Ko se izvrši, posamezna nit zapiše svoj končni rezultat, ki ga med računanjem akumulira v privatnem pomnilniku, v tabelo `forces[]` v globalnem pomnilniku naprave. V kolikor rezultata ne zapišemo v globalni pomnilnik, se le-ta v lokalnem pomnilniku izgubi. Gostitelj mora sedaj, ko so se vse niti zaključile, le še prenesti omenjeno tabelo v svoj glavni pomnilnik (RAM) in ustrezno prikazati rezultat. Rezultate prenašamo na gostitelja, ker določenih funkcij, med njimi tudi `printf()`, ne moremo uporabiti na napravi.

6.4. Niti in bloki

Ščepec se izvrši nad določenim številom niti. Odvisno od problema lahko niti organiziramo v eno- ali dvodimenzionalne bloke. Podobno kot niti, združujemo v eno- ali dvodimenzionalno mrežo tudi bloke. Število niti v bloku je bistvenega pomena predvsem pri razporejanju niti na multiprocesorjih GPE (ang. *streaming multiprocessor*). Posamezen blok se izvrši na enem

multiprocesorju. Multiprocesor običajno razdeli blok v skupine (ang. *warps*) po 32 niti. Blok, ki vsebuje 128 niti, se zato na multiprocesorju razdeli v štiri skupine po 32 niti. Z delitvijo niti v skupine prikrijemo pomnilniško latenco ter posledično zmanjšamo izvajalni čas [16].

V prvem primeru smo preko parametra `NULL` v funkciji `clEnqueueNDRangeKernel()` prepustili organizacijo niti in blokov izvajalnemu okolju. V drugem primeru smo število niti v bloku določili ročno.

```
// Execute the OpenCL kernel
      size_t global_item_size=N;
      size_t local_item_size=1; // 1 thread per block
      ret=clEnqueueNDRangeKernel(command_queue, kernel, 1, NULL,
&global_item_size, &local_item_size, 0, NULL, NULL);
```

V tem primeru uporabljamo za vsak atom svojo nit. Število niti je torej enako velikosti problema (torej *N*), posamezni bloki pa se združujejo v enodimenzionalno mrežo. Vsaka nit je neodvisna od drugih in ima svoj edinstven indeks. Ker je število niti enako številu atomov lahko indeks niti uporabimo za bralno/pisalne dostope do tabel s podatki in rezultati v globalnem pomnilniku, pri čemer se izognemo težavam s hkratnim pisalnim dostopom, saj je indeks vsake niti enolično določen.

Razlika v hitrosti izvajanja med prvim in drugim primerom je velikosti absolutne napake, torej lahko sklepamo, da tudi izvajalno okolje razdeli niti na podoben način.

Ščepec vsebuje programsko kodo, ki jo izvrši vsaka nit. Nit s svojim indeksom predstavlja en atom v skupini. Rešitev smo zasnovali z dvojno zanko. Ker je niti toliko kot je tudi atomov, nam kot zunanja zanka služi kar funkcija `get_global_id(0)`, ki nam vrne enoličen indeks. Pred vstopom v notranjo zanko pridobi nit iz globalnega pomnilnika podatke o atomu, katerega predstavlja (spremenljivka `i_atom_distance`).

```
      float4 i_atom_distance=(float4)(atoms[i].x, atoms[i].y, atoms[i].z,
0.0f);
      energy=0.0f;
      float4 d=(float4)(0.0f, 0.0f, 0.0f, 0.0f);
```

Po vstopu v notranjo zanko pridobi nit še podatke o drugem atomu (spremenljivka `n_atom_distance`). Med obema atomoma nato izračuna razdaljo, in če je potrebno, še silo. V naslednji interakciji mora nit zopet pridobiti informacijo o naslednjem atomu iz globalnega pomnilnika (podatke o atomu, katerega predstavlja, pa že hrani v spremenljivki `i_atom_distance`). Druga zanka se torej izvrši *N*-krat.

```
for(n; n<kN; n++){

     float4 n_atom_distance=(float4)(atoms[n].x, atoms[n].y, atoms[n].z,
0.0f);
     distance=i_atom_distance-n_atom_distance;
     distance2=mad(distance.x, distance.x, mad(distance.y, distance.y,
distance.z*distance.z));

     if (distance2<=cutx && i!=n){
            force=1.0f-distance2/cutx;
            cg=charge_i*atoms[n].w*332.0716f/sqrt(distance2);
            e=cg*force*force;
            energy+=e;

            tf =-e/distance2-4.0f*cg*force/cutx;
            d+=tf*distance;
        }
    }
```

V zgornjem primeru smo zaradi preglednosti namenoma izpustili računanje L-J potenciala. Spremenljivki `energy` in `d` se nahajata v privatnem pomnilniškem prostoru niti in akumulirata (prištevata) delne rezultate posamezne interakcije k skupni sili in energiji. *N* niti ponovi ta korak *N*-krat. Ko se notranja zanka zaključi, nit zapiše končen rezultat iz svojega privatnega pomnilnika v globalni pomnilnik naprave v tabelo `forces[]` na mesto, določeno s svojim indeksom.

```
     forces[i]=d;
     forces[i].w=energy;
```

6.5. Optimizacija v OpenCL

Optimizacija v OpenCL je zelo širok pojem. V našem delu se bomo zato omejili zgolj na nekaj optimizacij, ki smo jih uporabili v našem primeru.

OpenCL podpira vektorske podatkovne tipe. Namesto strukture, ki predstavlja en atom, smo zato uporabili vektorski podatkovni tip `float4`. Posamezen atom je sedaj predstavljen kot ena spremenljivka tipa `float4`. Tip `float4` je 128B velika spremenljivka, ki lahko hkrati hrani štiri običajne `float` vrednosti (torej 4*32B). Prva tri mesta vektorskega tipa sedaj uporabljamo za hranjenje lokacije v koordinatnem sistemu, zadnje pa za hranjenje naboja atoma. Vektorski tip `float4` se torej obnaša podobno kot struktura v C.

```
float4 i_atom_distance=(float4)(atoms[i].x, atoms[i].y, atoms[i].z, 0.0f);
```

Naša časovno najbolj potratna računska operacija je izračun razdalje med atomoma. V vsaki interakciji je potrebno izračunati razdaljo, pri čemer nadaljujemo z nadaljnjim računanjem le, če sta atoma dovolj skupaj, tako da njuna medsebojna elektrostatska sila ni zanemarljiva. L-J potencial in Coulombovo silo torej računamo le, če je razdalja ustrezna (manjša od 100.0).

Če uporabljamo za hranjenje podatkov o atomih strukture, potem je izračun razdalje podoben sledečemu (simbolično zapisano).

```
x=atom[i].x-atom[n].x;
y=atom[i].y-atom[n].y;
z=atom[i].z-atom[n].z;
r2=x*x+y*y+z*z;
```

Imamo torej trikratno odštevanje in množenje ter seštevanje. Pri uporabi `float4` podatkovnih tipov se operacija računanja razdalje poenostavi (spremenljivki `i_atom` in `n_atom` sta podatkovnega tipa `float4`).

```
r=i_atom-n_atom;
r2=mad(r.x, r.x, mad(r.y, r.y, r.z*r.z));
```

Pri čemer se razlike v koordinatah *x*, *y* in *z* izračuna kar hkrati v eni interakciji, tako da se odšteje istoležne komponente `float4` vektorja. Namesto treh operacij odštevanja imamo sedaj samo eno.

Pri računanju kvadrata razdalje si sedaj pomagamo s funkcijo `mad()` (ang. multiply-add). Brez uporabe omenjene funkcije bi zgornji izraz r^2 izgledal enako kot v primeru računanja s strukturami. Funkcija `mad()` izboljša hitrost izvajanja, pri čemer je natančnost še vedno sprejemljiva (polovična natančnost, večja hitrost računanja). Kljub vsemu se enemu produktu v polni natančnosti ne moremo izognit (“`r.z*r.v`” v našem primeru).

Gostitelj mora pred izvajanjem ščepca na napravi prevesti programsko kodo le-tega, shranjeno v ločeni tekstovni datoteki. Funkcija `clBuildProgram()`, ki prevede programsko kodo, sprejema kot argument tudi kazalec na niz, ki označuje vrsto optimizacije in s tem omogoči prevajalniku dodatne načine optimizacije. Z opcijo “`-cl-fast-relaxed-math`” omogočimo dodatne agresivne optimizacije, ki na račun višje hitrosti lahko kršijo določila IEEE 754.

7. Rezultati

7.1. Amdahlov zakon

Naša rešitev problema ni v celoti paralelna. Vzporedno se izvrši samo ščepec, ki se izvaja na napravi. Preostale programske kode, ki se izvede na gostitelju, ni mogoče paralelizirati.

Pri paralelnih računalnikih nam veliko število CPE ne pomaga dosti, če je problem tak, da večji del časa dela samo ena CPE [2]. Pohitritev je zato vedno omejena z delom programske kode, ki je ni mogoče paralelizirati.

Če imamo torej večji del programa paralelen in ta del označimo s *P*, potem obsega preostali sekvenčni del programa *1-P* (pri čemer predstavlja število *1* kar celotni izvajalni čas). Največja pohitritev, ki jo lahko pričakujemo, če uporabljamo *N* centralnih procesnih enot, je enaka (4).

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}} \quad (4)$$

Želimo si, da je del *1-P* čim manjši, saj izvajalni čas nikakor ne more biti nižji, kot je čas, ki ga porabi sistem za izvajanje sekvenčnega dela *1-P* (ne glede na število jedr CPE, ki jih imamo na razpolago).

Sistem 1 potrebuje za izračun na dvojedrni CPE približno 20 sekund (kar predstavlja 100% oz. 1). Sekvenčni del nam pri tem vzame 0.2 sekundi, torej del *1-P* predstavlja 1% (oz. 0.01) celotnega časa. Ne glede na število CPE bi največji dosegljivi faktor pohitritve v tem primeru znašal 100. Spremenljivka *N* predstavlja tokrat število jeder, torej 2. Iz enačbe (4) lahko izračunamo, da je največja pričakovana pohitritev za sistem 1 približno **1.98**.

7.2. Izvajanje na CPE

V spodnjih tabelah 3, 4 ter 5 so prikazani izvajalni časi. Število *N* kot navadno predstavlja število atomov. Naslednje je izvajanje algoritma na enem samem procesorskem jedru, pri čemer ne upoštevamo Newtonovih zakonov in računamo vse interakcije, torej N^2 interakcij. Sledijo računski časi, ki smo jih dosegli z uporabo OpenCL ter vseh razpoložljivih procesorskih jeder. V zadnjem stolpcu se nahaja faktor pohitritve med sekvenčnim in

paralelnim izvajanjem.

Pretečeni čas smo merili s pomočjo vgrajene funkcije `gettimeofday()`. Funkcija vrne trenutni čas v sekundah in mikrosekundah, ki je pretekel od Epohe (00:00, 1. januar 1970). Merili smo samo čas, ki ga porabimo za kopiranje podatkov o atomih na napravo, računanje elektrostatskih sil in energije na napravi ter kopiranje rezultatov z naprave na gostitelja.

N	1 jedro [s]	2 jedri [s]	faktor
10.125	0.47	0.21	2.24
27.783	2.70	1.37	1.97
50.625	8.13	4.38	1.85
107.811	34.51	19.45	1.77

Tabela 3: Računski časi na sistemu 1.

Faktor pohitritve je odvisen od števila jeder, ki jih ima CPE na razpolago. Z uporabo OpenCL, smo tako pri enakem algoritmu v primerjavi s sekvenčnim izvajanjem pridobili približno dvakrat krajši čas izvajanja pri prvem sistemu in približno štirikrat manj časa nam vzame drugi sistem. Rezultati izkazujejo lastnosti Amdahlovega zakona.

N	1 jedro [s]	4 jedra [s]	faktor
10.125	0.47	0.10	4.70
27.783	2.63	0.70	3.76
50.625	7.92	2.25	3.52
107.811	35.71	9.90	3.60

Tabela 4: Računski časi na sistemu 2.

V zakup velja vzeti dejstvo, da v vseh zgornjih primerih primerjamo izvajalne čase, pri čemer zaporedni in vzporedni algoritem opravita enako (N^2) interakcij. Ali je uporaba tehnologije OpenCL torej smiselna? Odgovor na to vprašanje lahko dobimo šele pri primerjavi realnega izvajalnega časa sekvenčnega programa z izvajanjem na dveh ali štirih jedrih CPE. Primerjave smo strnili v spodnji tabeli 5.

N	1 jedro (III. Newtonov zakon) [s]	2 jedri [s]	4 jedra [s]
10.125	0.24	0.21	0.10
27.783	1.33	1.37	0.70
50.625	4.01	4.38	2.25
107.811	17.80	19.45	9.90

Tabela 5: Primerjava realnih izvajalnih časov.

V tabeli 5 so rdeče obarvani časi, ki so kljub paralelnemu izvajanju počasnejši od sekvenčnega algoritma. Opazimo lahko, da se nam predvsem v primeru, ko imamo na razpolago zgolj dve procesorski jedri ne izplača uporaba tehnologije OpenCL. To je deloma razvidno že tudi iz tabele 2, saj je faktor pohitritve, razen v prvem primeru, vedno nižji od števila 2. Naš problem *N* teles je tako smiselno reševati kvečjemu na večjedrni CPE z več kot dvema jedroma.

7.3. Izvajanje na GPE

Ker je OpenCL neodvisen od strojne platforme, lahko programsko kodo, ki smo jo izvajali na CPE, izvedemo tudi na GPE. Za konec smo primerjali hitrosti izvajanja še na tretjem razpoložljivem sistemu, ki vsebuje ločeno grafično kartico AMD Radeon.

N	1 jedro [s]	4 jedra [s]	GPE [s]	faktor
10.125	0.40	0.10	0.09	4.44
27.783	2.28	0.74	0.50	4.56
50.625	6.87	2.35	1.31	5.24
107.811	29.01	10.27	5.17	5.61

Tabela 6: Primerjava izvajalnih časov med CPE in GPE na sistemu 3.

Zadnji sistem je pri sekvenčnem računanju nekoliko hitrejši. Faktor pohitritve predstavlja razmerje med hitrostjo sekvenčnega izvajanja (1 jedro) ter paralelnega izvajanja na GPE. Vzporedno izvajanje na vseh štirih jedrih CPE izkazuje enako obnašanje kot CPE sistema 1 in sistema 2. S povečevanjem računskega dela, torej števila atomov, se pohitritev linearno zmanjšuje (Amdahlov zakon). Obratno se GPE sistema 3 odreže bolje pri večjem naboru atomov. Torej večji kot je obseg podatkov, hitrejša je pri računanju GPE. Pri najmanjšem naboru podatkov je GPE skorajda časovno izenačena z CPE. Vzrok temu je verjetno manj obsežen paralelni del programa v primerjavi s sekvenčnim delom, ki ga ne moremo paralelizirati. S povečanjem računskega obsega pa se razmerje med sekvenčnim in paralelnim delom programa prevesi v korist paralelnega.

8. Sklepne ugotovitve

Ugotovili smo, da osnovni algoritem ni popolnoma podatkovno neodvisen. Pri sekvenčnem izvajanju na enem jedru CPE nam je v pomoč III. Newtonov zakon, ki ravno razpolovi število potrebnih računskih interakcij. Zaradi neenotnega pomnilniškega prostora v OpenCL, žal ne moremo koristno uporabiti Newtonovih zakonov, saj s tem izgubljamo delne rezultate posameznih interakcij, kar vodi v neenake končne rešitve problema pri enakih vhodnih

podatkih. Dosegli smo pohitritve, ki se skladajo z Amdahlovim zakonom.

Paralelizacija problema se nam ne izplača, v kolikor imamo na voljo zgolj CPE z dvema jedroma, pri čemer izkoriščanje tehnologije Hyper-Threading pri posameznem jedru zanemarljivo vpliva na hitrost računanja. Razpolagati moramo najmanj s štiri ali več jedrno CPE, da se nam paralelizacija časovno obrestuje.

Ker je OpenCL odprt standard, tako rekoč nismo imeli težav s poganjanjem enake programske kode na GPE sistema 3. V zadnjih primerjavah med hitrostjo izvajanja programa na CPE in GPE smo ugotovili, da že GPE srednjega hitrostnega in cenovnega razreda prekaša CPE, pri čemer se z večanjem obsega podatkov ta razlika samo še povečuje v prid GPE. Pravi smisel bi torej imela paralelizacija algoritma, če bi razpolagali z visoko zmogljivo GPE.

Prav tako smo ugotovili, da bi za tekočo realno simulacijo skupine bolj kompleksnih molekul kot je voda, potrebovali več večjedrnih CPE. Gibanje skupine teles se namreč izkaže za časovno bistveno bolj potratnega, kot bi lahko verjeli. V naravi vsak atom znotraj bolj ali manj kompleksnih molekul spremeni svojo smer v delcu sekunde. Pri simulaciji zato stremimo k čim hitrejšemu računanju novega položaja v naslednjem trenutku. Hitreje kot lahko določimo obnašanje skupine delcev, hitreje lahko napredujejo raziskave v različnih vejah znanosti.

9. Literatura in viri

[1] Chandrakasab A.P., Potkonjak M., Mehra R., Rabaey J., Brodersen R.W. Optimizing power using transformations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 14, No. 1. 1995.

[2] D. Kodek, *Arhitektura in organizacija računalniških sistemov*. Šenčur: BI-TIM, 2008. ISBN 978-961-6046-08-4.

[3] Gaster B.R., Howes L., Kaeli D., Mistry P., Schaa D., *Heterogeneous Computing with OpenCL*. MA 02451, Waltham: Morgan Kaufmann, 2011. ISBN 978-0-12-387766-6.

[4] J. Strnad, *Fizika: 1. del*. 10 natis. Ljubljana: DMFA, 2007. ISBN 978-961-212-047-4.

[5] J. Strnad, *Fizika: 2. del*. 5 natis. Ljubljana: DMFA, 1995. ISBN 961-212-048-X

SPLETNI VIRI

[6] (2011) Fizikalne lastnosti vode. Dostopno na naslovu: http://www.thermexcel.com/english/tables/

[7] (2012) Intel 8080. Dostopno na naslovu: http://www.cpu-world.com/CPUs/8080/

[8] (2012) Intel Z68 Express Chipset. Dostopno na naslovu: http://ixbtlabs.com/articles3/mainboard/intel-z68-express-chipset-p1.html

[9] (2011) Kemijske lastnosti vode. Dostopno na naslovu: http://www.chem1.com/acad/sci/aboutwater.html

[10] (2012) Lennard-Jones potential. Dostopno na naslovu: http://chemwiki.ucdavis.edu/Physical_Chemistry/Quantum_Mechanics/Intermolecular_Forces/Lennard-Jones_Potential

[11] (2012) Loresana Grabušnik, Anomalne lastnosti vode. Dostopno na naslovu: http://fizika.uni-mb.si/files/seminarji/11/AnomalneLastnostiVode.pdf

[12] (2012) Pipelines – Improving CPU performance. Dostopno na naslovu: http://mail.humber.ca/~michaud/2005-2006/CENG606/pipeline/Pipeline.htm

[13] (2012) Problem treh teles. Dostopno na naslovu: http://www.kvarkadabra.net/article.php/problem-treh-teles

[14] (2012) The magic crystal of life. Dostopno na naslovu: http://www.silkywater.com/waterintro.htm

[15] (2012) Intel Hyper-Threading. Dostopno na naslovu: http://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html

[16] (2012) Nvidia OpenCL Optimization. Dostopno na naslovu:
http://developer.download.nvidia.com/CUDA/training/NVIDIA_GPU_Computing_Webinars_Best_Practises_For_OpenCL_Programming.pdf