UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Davor Sluga

Paralelna implementacija celičnega modela dinamike tekočin

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Andrej Dobnikar

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01705/2010
Datum: 01.09.2010

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

Kandidat: **DAVOR SLUGA**

Naslov: **PARALELNA IMPLEMENTACIJA CELIČNEGA MODELA DINAMIKE TEKOČIN**

PARALLEL IMPLEMENTATION OF CELLULAR FLUID DYNAMICS MODEL

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

V nalogi se oprite na izpeljanko Navier-Stokesovih diferencialnih enačb, ki omogoča celično obravnavo (procesiranje) masnih delcev. Tako izpeljani algoritem modificirajte s pomočjo knjižnice OpenMPI, ki omogoča paralelno implementacijo na gruči računalnikov.

Rezultate paralelnega procesiranja prikažite tako, da bosta razvidni pohitritev in uspešnost procesiranja. Pri tem se osredotočite na domeno okolja v obliki 2D cevi in stopnice.

Mentor:

prof. dr. Andrej Dobnikar

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Franc Solina

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Davor Sluga,

z vpisno številko 63050108,

sem avtor diplomskega dela z naslovom:

Paralelna implementacija celičnega modela dinamike tekočin

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  prof. dr. Andreja Dobnikarja
- so elektronska oblika diplomskega dela, naslov (slov., ang.), povzetek (slov., ang.) ter ključne besede (slov., ang.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 13.9.2010 Podpis avtorja:

Zahvala

Zahvaljujem se mentorju prof. dr. Andreju Dobnikarju za njegov trud in pomoč in ideje, ki so pripomogli k nastanku te diplomske naloge.

Zahvaljujem se tudi staršem, ker so mi študij omogočili in me ves čas podpirali, tako finančno kot moralno.

Zahvalil bi se vodji raziskav na Turboinštitutu dr. Andreju Lipeju, ki je omogočil izvedbo eksperimentov na superračunalniku LSC Adria.

Zahvala gre tudi prof. dr. Andreju Likarju, za vso pomoč in potrpežljivost, ki jo je izkazal pri razlagi fizikalnega ozadja dinamike tekočin.

Kazalo

Seznam uporabljenih kratic in simbolov

MPI - Message Passing Interface
CUDA - Compute Unified Device Architecture
LAN - Local Area Network
GCC - GNU Compiler Collection
SMP - Symmetric Multi-Processing
TCP - Transmission Control Protocol

Povzetek

V diplomski nalogi smo razvili paralelno implementacijo celičnega modela dinamike tekočin. Njena prednost je v zelo učinkovitem izrabljanju paralelnih računalnikov. Najprej smo na podlagi enačb, ki modelirajo dinamiko tekočin razvili sekvenčni algoritem in ga optimizirali. Seznanili smo se z različnimi vrstami paralelnih računalnikov, metodologijo razvoja paralelnih algoritmov in metrikami s katerimi lahko ocenimo uspešnost paralelizacije. Visoko učinkovitost smo dosegli z uporabo metode Monte Carlo, ki omogoča doseganje praktično linearne pohitritve z večanjem števila procesorjev. Eksperimenti, ki smo jih izvedli na superračunalniku LSC Adria so to tudi potrdili. Ugotavljali smo kako posamezni parametri algoritma vplivajo na kvaliteto rezultata in čas procesiranja. Dobljeni rezultati so se skladali z teoretičnimi napovedmi, saj se je paralelni algoritem se je izkazal kot zelo učinkovit.

Ključne besede:

dinamika tekočin, paralelni algoritmi, metoda Monte Carlo, superračunalniki

Abstract

In this thesis we developed a parallel implementation of cellular fluid dynamics model. Its advantage lies in very efficient exploitation of parallel computers. Firstly we took the equations which govern the fluid dynamics and developed a sequential algorithm. We then optimized the algorithm and improve its performance. We described different types of parallel computers and presented a methodology for developing parallel algorithms. We also defined metrics for assessing the performance of parallel algorithms. High efficiency was achieved by using the Monte Carlo method, which enables virtually linear speedup with increasing number of processors. Experiments which were performed on LSC Adria Supercomputer confirmed this prediction. We observed how changing the algorithms parameters affects the quality of the result and processing time. The results were consistent with theoretical predictions consequently parallel algorithm proved to be very effective.

Key words:

fluid dynamics, parallel algorithms, Monte Carlo method, supercomputers

Poglavje 1

Uvod

Uporabo področja mehanike tekočin za doseganje napredka in razvoj novih tehnologij najdemo na veliko področjih, kot so gradbeništvo, letalska industrija, vesoljska tehnika, avtomobilska industrija, ladjedelništvo, vremenoslovje. Prav zaradi tako širokega spektra uporabnosti, je to področje deležno velike pozornosti. Poudarek je predvsem na čimbolj učinkovitem in natančnem modeliranju tekočin s pomočjo računalnikov. Tak pristop namreč omogoča zmanjševanje stroškov in časa potrebnega za razvoj in vpeljavo novega izdelka oz. tehnologije.

Računalniška dinamika tekočin (ang. Computational Fluid Dynamics — CFD) uporablja numerične metode ter algoritme za reševanje in analizo problemov, ki vključujejo tok tekočin. Modeliranje mehanike tekočin je matematično in časovno lahko zelo zahtevno. Posledično se na tem področju veliko uporabljajo supperračunalniki oz. na splošno paralelni računalniki. Pomembno je torej, da paralelna implementacija predstavlja učinkovit način reševanja dinamičnega modela tekočin, kar pomeni kvalitativno sprejemljiv rezultat v razumnem času. Razvitih je bilo veliko metod, ki se bolj ali manj uspešno bojujejo z visoko kompleksnostjo obnašanja tekočin in posledično zahtevnimi izračuni. Težave povzročajo predvsem turbulence na mikroskopskem nivoju, ki jih izkazujejo tekočine z nizko viskoznostjo (visokim Reynoldsovim številom).

V delu bomo predstavili nov pristop [4] k modeliranju dinamike tekočin, ki ima osnovo v celičnem načinu procesiranja in stohastični metodi Monte Carlo [11]. Zanimalo nas bo predvsem, kakšna je učinkovitost nove metode, če jo uporabimo v povezavi s paralelnim računalnikom.

Delo je razdeljeno na sedem poglavij. Po uvodu je v drugem poglavju podrobneje opisano fizikalno ozadje modeliranja dinamike tekočin. Iz osnovnih enačb dinamike tekočin je izpeljan celični model, ki služi kot iztočnica za im-

plementacijo sekvenčne in paralelne različice algoritma.

V tretjem poglavju je podan osnovni algoritem, temu pa sledijo izboljšave, ki pohitrijo izvajanje algoritma. V četrtem poglavju se seznanimo s paralelnimi računalniki in metodami razvoja paralelnih programov. V petem poglavju je prikazana implementacija paralelne različice algoritma iz tretjega poglavja. Osnovni cilj paralelne implementacije je čim večja učinkovitost in pohitritev osnovnega algoritma.

Delovanje sekvenčnega in paralelnega algoritma želimo empirično preveriti, zato v petem poglavju opišemo paralelni računalniški sistem, na katerem so bili izvedeni določeni eksperimenti. Za kakovostno analizo rezultatov eksperimentalnega dela je pomembno zagotoviti kontrolirane pogoje v zvezi s programsko in strojno opremo. Zato je podana specifikacija sistema skupaj z načini preizkušanja in izbire parametrov. V šestem poglavju so opisani rezultati eksperimentov ob uporabi sekvenčnih in paralelnih algoritmov.

Zaključek diplomske naloge je podan v sedmem poglavju, kjer so navedene tudi možne smernice za nadaljnje delo.

Poglavje 2

Fizikalno ozadje

Osnovo modeliranja dinamike tekočin opisujejo Navier-Stokesove enačbe, poimenovane po Claudeu-Louisu Navieru in Georgeu Gabrielu Stokesu. Predstavljajo sistem nelinearnih parcialnih diferencialnih enačb, ki določajo nestacionarno gibanje stisljive newtonske viskozne tekočine [6]. V vektorskem zapisu imajo obliko:

$$\rho\left(\frac{\partial\vec{v}}{\partial t}+\vec{v}\cdot\nabla\vec{v}\right)=-\nabla p+\nabla\mathbb{T}+\vec{f}\quad . \tag{2.1}$$

Nastopajoče količine so ρ, ki predstavlja gostoto tekočine, $\vec{v}$ je vektor hitrosti toka, t označuje čas, p je tlak v tekočini, $\mathbb{T}$ je deformacijski napetostni tenzor, $\vec{f}$ predstavlja preostale zunanje sile, ki delujejo na tekočino npr. gravitacija. Simbol ∇ je Hamiltonov operator vektorskega odvajanja, v dveh dimenzijah je enak $\left(\frac{\partial}{\partial x},\frac{\partial}{\partial y}\right)$.

Leva stran je proporcionalna pospešku in je sestavljena iz dveh členov, časovno odvisne spremembe hitrosti $\frac{\partial\vec{v}}{\partial t}$ in prostorsko odvisnega konvekcijskega pospeška $\vec{v}\cdot\nabla\vec{v}$. Desna stran enačbe pa je vsota vseh sil, ki delujejo na tekočino. Člen ∇p imenujemo tlačni gradient, ta vpliva na tekočino tako, da povzroči tok iz področja višjega tlaka proti nižjemu. Deformacijski napetostni tenzor $\nabla\mathbb{T}$ pa opisuje viskozne sile v tekočini.

Člen $\left(\frac{\partial \vec{v}}{\partial t} + \vec{v} \cdot \nabla \vec{v}\right)$ krajše zapišemo z uporabo materialnega odvoda $\frac{\mathbf{D}\vec{v}}{\mathbf{D}t}$, ki združuje prostorsko in časovno pogojeno spreminjanje hitrosti. V našem primeru, se bomo osredotočili na modeliranje toka vode. Zanjo velja, da je slabo stisljiva. Posledično, lahko kot približek privzamemo, da modeliramo tok nestisljive tekočine. Enačbe se zato nekoliko poenostavijo [5]. Predpostavimo tudi, da na naš sistem ne vplivajo druge zunanje sile $\vec{f} = 0$. Zaradi predpostavke o nestisljivosti, $\nabla \mathbb{T}$ postane $\mu \nabla^2 \vec{v}$, kjer je μ dinamična viskoznost tekočine. Dobimo enačbo:

$$\rho \frac{\mathbf{D}\vec{v}}{\mathbf{D}t} = -\nabla p + \mu \nabla^2 \vec{v} \quad . \tag{2.2}$$

Vpeljava substitucij:

$$a = \frac{\mu}{\rho\, U} \quad , \tag{2.3}$$

$$\tilde{\vec{x}} = \frac{\vec{x}}{a} \quad , \tag{2.4}$$

$$\tilde{p} = \frac{p}{\rho\, U^2} \quad , \tag{2.5}$$

$$\tilde{t} = \frac{Ut}{a} \quad , \tag{2.6}$$

$$\tilde{\vec{v}} = \frac{\vec{v}}{U} \quad , \tag{2.7}$$

kjer je U povprečna hitrost toka, spremeni enačbo (2.2) v brez dimenzijsko obliko [4]. Znebimo se dveh količin, gostote ρ in dinamične viskoznosti μ [4]. Brez dimenzijska oblika enačbe v vektorskem zapisu se glasi:

$$\frac{\mathbf{D}\tilde{\vec{v}}}{\mathbf{D}\tilde{t}} = -\tilde{\nabla}\tilde{p} + \frac{1}{Re}\tilde{\nabla}^2\tilde{\vec{v}} \quad , \tag{2.8}$$

kjer je $Re = \frac{U\rho a}{\mu}$ Reynoldsovo število [7] in je zaradi ustrezne izbire umeritvene konstante a enako 1 zato ga lahko v nadaljevanju zanemarimo.

Tekočino lahko obravnavamo kot množico soodvisnih masnih delcev, zato v enačbo vpeljemo pojem delca in jo na ta način diskretiziramo [4]. Člen $-\nabla\tilde{p}$ nadomestimo z $-G \cdot \vec{t}$. Koeficient gradienta tlaka G je vnaprej določena konstanta, pogojena z gostoto delcev v prostoru $\vec{t}$ pa lokalno težišče okoli trenutno opazovanega delca. Izračunamo ga po enačbi $\sum_{i=1}^{N}(\vec{x_i} - \vec{x})$, kjer $\vec{x}$ označuje položaj delca, N pa število delcev v okolici, ki je definirana kot krog z radijem δ in središčem v $\vec{x}$. Člen $\tilde{\nabla}^2\tilde{\vec{v}}$ nadomestimo s povprečjem diferenc hitrosti med opazovanim delcem in delci v njegovi okolici: $V \cdot \frac{1}{N}\sum_{i=1}^{N}(\vec{v_i} - \vec{v})$. Konstanta V je koeficient viskoznosti pogojena z radijem δ. Tako preoblikovana enačba se glasi:

$$\frac{\mathbf{D}\vec{v}}{\mathbf{D}t} = -G \cdot \vec{t} + V \cdot \frac{1}{N}\sum_{i=1}^{N}(\vec{v_i} - \vec{v}) \quad . \tag{2.9}$$

Če upoštevamo vektorsko enačbo (2.9) in vpeljemo še časovni korak T, s katerim diskretiziramo enačbo po časovni komponenti, lahko zapišemo enačbo hitrosti in položaja posameznega delca v prostoru [4]. Časovni korak T je obratno sorazmeren s številom delcev v prostoru. S tem ostanejo spremembe hitrosti in položaja delcev dovolj majhne, da ti natančno opišejo svojo pot po prostoru. Enačbi, ki definirata gibanje delca sta:

$$\vec{v}_{t+1} = \vec{v}_t + \left(-G \cdot \vec{t}_t + V \cdot \frac{1}{N}\sum_{i=1}^{N}(\vec{v_{i,t}} - \vec{v}_t)\right) \cdot T \quad , \tag{2.10}$$

$$\vec{x}_{t+1} = \vec{x}_t + \vec{v}_{t+1} \cdot T \quad . \tag{2.11}$$

Poglavje 3

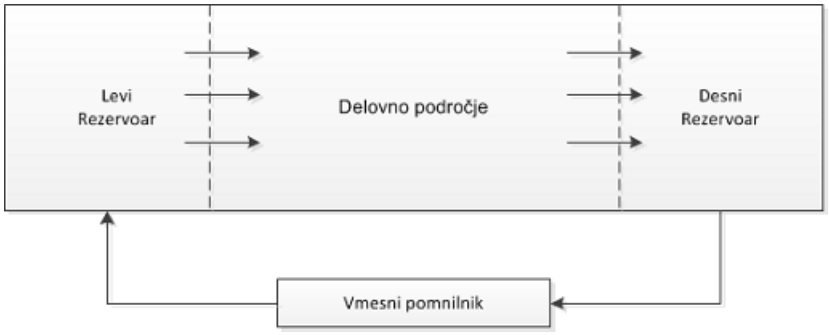
Osnovni algoritem

3.1 Ideja

Implementacija algoritma sloni na enačbah predstavljenih v prejšnjem poglavju. Tok tekočine modeliramo v dvodimenzionalnem prostoru, kar je sicer poenostavitev, vendar ne okrni splošnosti algoritma, saj je razširitev do treh dimenzij trivialna, omogoči pa nam lažjo predstavitev rezultatov.

Tok predstavljajo gibajoči se delci, ki se nahajajo v prostoru oblikovanem kot cev. Ustrezno tlačno razliko, ki poganja delce vzdolž cevi, zagotavljata dva rezervoarja in vmesni pomnilnik.

Delci, ki prečkajo konec cevi, se znajdejo desnem rezervoarju. V njem mora biti približno konstantno število delcev, s čimer simuliramo daljšo cev, kot jo imamo v resnici in se tako izognemo anomalijam, ki bi se sicer pojavile, če bi delci neovirano izstopali iz cevi. Če število delcev v desnem rezervoarju prekorači maksimalno dovoljeno mejo, se odstranijo v vmesni pomnilnik. Levi rezervoar skrbi za stalen dotok delcev v cev. Predpisano število delcev v njem se vzdržuje tako, da se v primeru pomanjkanja dodajajo iz vmesnega pomnilnika. Z $Nlevi$ označujemo spodnjo mejo delcev v levem rezervoarju, z $Ndesni$ pa zgornjo mejo delcev v desnem rezervoarju. Veljati mora neenakost: $Nlevi > Ndesni$, le v tem primeru namreč dosežemo stabilen tok delcev iz levega konca proti desnemu koncu cevi.



Slika 3.1: Shema prostora v obliki cevi z levim in desnim rezervoarjem ter prikazanim tokom delcev.

Potek modeliranja sledi naslednjim korakom:

- naključno razmeči N delcev v levi rezervoar.
- Ponavljaj za K časovnih korakov:
  - za vsak delec na podlagi tlačnega gradienta in viskoznosti izračunaj novo hitrost in položaj delca.
  - Preveri ali je delec dosegel rob prostora in ustrezno ukrepaj.
  - Posodobi polje tlaka in hitrosti na podlagi trenutnih razmer v prostoru.

Podrobna predstavitev vseh korakov modeliranja:

- **Naključno razmeči N delcev v levi rezervoar**
  Izberemo si število delcev N, ki jim priredimo naključen začetni položaj v prostoru tj. x in y koordinato. Začetne hitrosti delcev postavimo na 0. Število delcev je poljubno, vendar bomo v poglavju 7 pri analizi rezultatov ugotovili, da je kakovost rešitve pri določenem številu časovnih korakov odvisna od tega parametra.
- **Ponavljaj za K časovnih korakov**
  Število časovnih korakov algoritma mora biti dovolj veliko, da pridemo do dovolj kvalitetnega rezultata. Vpliv tega parametra na kakovost rešitve si bomo ravno tako podrobneje ogledali v poglavju 7.
- **Za vsak delec na podlagi tlačnega gradienta in viskoznosti izračunaj novo hitrost in položaj delca**
  V tem koraku uporabimo enačbi (2.10) in (2.11).

- **Preveri ali je delec dosegel rob prostora in ustrezno ukrepaj**
  Tu upoštevam robne pogoje, ki vladajo v prostoru. Trk delca s steno cevi, se izraža kot sprememba njegovega hitrostnega vektorja. Modeliramo ga tako, da ob trku komponento vektorja hitrosti vzporedno s steno postavimo na 0, medtem ko komponenti pravokotni na steno spremenimo predznak [4].

- **Posodobi polje tlaka in hitrosti na podlagi trenutnih razmer v prostoru**
  Prostor je razdeljen na ustrezno gosto mrežo. V njej beležimo polje tlaka in hitrosti tekočine. Tlak v posamezni celici mreže je definiran s številom vseh delcev, ki so se v celotnem času procesiranja znašli v njej. Podobno je tudi hitrost v celici seštevek hitrosti vseh delcev, ki so se znašli v njej. Po koncu procesiranja nam ta mreža predstavlja rešitev problema.

3.1.1 Psevdokoda osnovnega algoritma

Predstavljena je podrobna psevdokoda algoritma, ki lahko služi kot podlaga za implementacijo v poljubnem programskem jeziku.

Parametri algoritma

- $CasKonec$: število iteracij algoritma.
- $CasZacetekBelezenjaStatistike$: število iteracij po katerem začnemo beležiti statistiko polja tlaka in hitrost, uvedemo ga zato, da izpustimo začetne prehodne razmere.
- δ: radij okolice, do katerega poteka interakcija med opazovanim delcem in ostalimi delci v prostoru.
- G: koeficient tlačnega gradienta.
- V: koeficient viskoznosti.
- $Ndesni$: maksimalno število delcev v desnem rezervoarju.
- $Nlevi$: minimalno število delcev v levem rezervoarju.

Funkcije

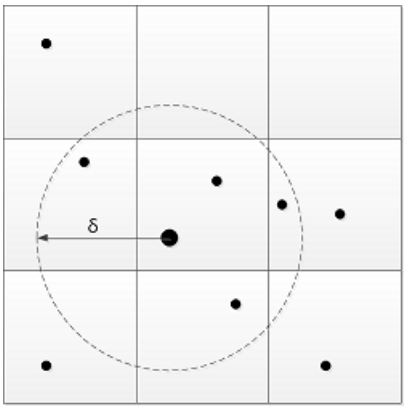
- $NakljucenPolozaj$: določi naključen položaj delcu v levem rezervoarju.
- $IzvenProstora(i)$: preveri, ali je i-ti delec že dosegel rob cevi.
- $VrniVProstor(i)$: postavi i-ti delec na ustrezno mesto v prostoru.
- $DelecVDRez(i)$: preveri ali se i-ti delec nahaja v desnem rezervoarju.
- $PrevecDelcevVDRez(Ndesni)$: preveri ali je v desnem rezervoarju že preveč delcev.
- $Odstranidelec(i)$: odstrani i-ti delec iz desnega rezervoarja v medpomnilnik.
- $PremaloDelcevVLRez(Nlevi)$: preveri ali je v levem rezervoarju premalo delcev.
- $DodajDelecVLRez$: vzame delec iz medpomnilnika in ga doda v levi rezervoar.

Algoritem 1 Osnovni algoritem

```
1: for vsak delec i do
2:    Polozaj[i] ← NakljucenPolozaj
3:    Hitrost[i] ← 0
4: end for
5: while cas < CasKonec do
6:    if cas > CasZacetekBelezenjaStatistike then
7:       for vsak delec i do
8:          PoljeHitrost[Polozaj[i]] ← PoljeHitrost[Polozaj[i]] + Hitrost[i]
9:          PoljeTlak[Polozaj[i]] ← PoljeTlak[Polozaj[i]] + 1
10:      end for
11:   end if
12:   for vsak delec i do
13:      Hitrost[i] ← Hitrost[i] − Tezisce[i] · G · T
14:      Polozaj[i] ← Polozaj[i] + Hitrost[i] · T
15:      HitrostZacasna[i] ← Hitrost[i]
16:   end for
17:   for vsak delec i do
18:      for vsak delec j do
19:         if i ≠ j then
20:            Razdalja ← |Polozaj[i] − Polozaj[j]|
21:            if Razdalja < δ then
22:               Tezisce[i] ← Tezisce[i] + Polozaj[j] − Polozaj[i]
23:               VSum ← V × (HitrostZacasna[j] − HitrostZacasna[i])
24:               Vn ← Vn + 1
25:            end if
26:         end if
27:      end for
28:      Hitrost[i] ← Hitrost[i] + VSum/Vn
29:   end for
30:   for vsak delec i do
31:      if IzvenProstora(i) then
32:         VrniVProstor(i)
33:      end if
34:      if DelecVDRez(i) then
35:         if PrevecDelcevVDRez then
36:            OdstraniDelec(i)
37:         end if
38:      end if
39:      if PremaloDelcevVLRez(Nlevi) then
40:         DodajDelecVLRez
41:      end if
42:   end for
43:   cas ← cas + T
44: end while
```

3.2 Izboljšava algoritma

Iz psevdokode algoritma 1 vidimo, da je računsko najbolj intenzivna dvojna zanka po številu delcev N, v vrsticah 17 in 18, kar nam pove da čas procesiranja narašča s kvadratno funkcijo glede na število delcev. Pohitritev dosežemo z izkoriščanjem dejstva, da na delec, ki mu posodabljamo položaj in hitrost vplivajo samo delci v njegovi neposredni okolici, ki je definirana, kot smo navedli že v poglavju 2 z radijem δ. Prostor razdelimo na mrežo sestavljeno iz kvadratnih celic z dolžino stranice enako radiju δ. V vsaki celici vodimo evidenco, kateri delci se v njej nahajajo. Tisti, ki se nahajajo v isti celici kot trenutno opazovani delec in v njenih sosedah, so možni kandidati za računanje medsebojne interakcije (3.2).



Slika 3.2: Mreža celic s katero izkoristimo lokalnost interakcij med delci.

3.2.1 Psevdokoda izboljšanega algoritma

Parametri algoritma
Parametri izboljšanega algoritma so enaki kot pri osnovnem algoritmu 1.

Funkcije
Naštete so samo dodatne funkcije, opis ostalih najdemo pri osnovnem algoritmu 1.

- $Ponastavi(Mreza)$: odstrani vse molekule iz mreže sosednosti.
- $DodajVMrezo(i, polozaj)$: i-ti delec doda v celico mreže sosednosti, ki se nahaja na mestu polozaj.
- $Okolica(polozaj)$: vrne seznam delcev, ki se nahajajo v okolici in v celici na mestu polozaj.

Algoritem 2 Izboljšan algoritem

```
1: for vsak delec i do
2:    Polozaj[i] ← NakljucenPolozaj
3:    Hitrost[i] ← 0
4: end for
5: while cas < CasKonec do
6:    if cas > CasZacetekBelezenjaStatistike then
7:       for vsak delec i do
8:          PoljeHitrost[Polozaj[i]] ← PoljeHitrost[Polozaj[i]] + Hitrost[i]
9:          PoljeTlak[Polozaj[i]] ← PoljeTlak[Polozaj[i]] + 1
10:      end for
11:   end if
12:   Ponastavi(Mreza)
13:   for vsak delec i do
14:      Hitrost[i] ← Hitrost[i] − Tezisce[i] · G · T
15:      Polozaj[i] ← Polozaj[i] + Hitrost[i] · T
16:      HitrostZacasna[i] ← Hitrost[i]
17:      DodajVMrezo(i, Mreza[Polozaj[i]/δ])
18:   end for
19:   for vsak delec i do
20:      for vsak delec j vOkolica(Mreza[Polozaj[i]/δ]) do
21:         if i ≠ j then
22:            Razdalja ← |Polozaj[i] − Polozaj[j]|
23:            if Razdalja < δ then
24:               Tezisce[i] ← Tezisce[i] + Polozaj[j] − Polozaj[i]
25:               VSum ← V × (HitrostZacasna[j] − HitrostZacasna[i])
26:               Vn ← Vn + 1
27:            end if
28:         end if
29:      end for
30:      Hitrost[i] ← Hitrost[i] + VSum/Vn
31:   end for
32:   for vsak delec i do
33:      if IzvenProstora(i) then
34:         VrniVProstor(i)
35:      end if
36:      if DelecVDRez(i) then
37:         if PrevecDelcevVDRez then
38:            OdstraniDelec(i)
39:         end if
40:      end if
41:      if PremaloDelcevVLRez(Nlevi) then
42:         DodajDelecVLRez
43:      end if
44:   end for
45:   cas ← cas + T
46: end while
```

Izboljšan algoritem prinese linearen faktor pohitritve. Delci so zaradi narave fizikalnega procesa približno enakomerno razporejeni po prostoru, kar pomeni, da jih je v vsaki celici mreže približno enako število. Skupno število iteracij dvojne zanke se iz $N \times N$ zmanjša na približno $N \times 9 \times \frac{N}{N_{celic}}$, kar pomeni $\frac{N_{celic}}{9}$-kratno zmanjšanje števila iteracij. Potrebno je tudi nekaj dodatne režije, ki jo imamo z ažuriranjem mreže celic. Natančneje $N + N_{celic}$ dodatnih operacij, ki izhajajo iz vrstic 12 in 17. Delež dodatnega procesiranja je v primerjavi z prihranjenim številom iteracij zelo majhen, zato vpeljava te izboljšave precej pohitri algoritem.

Poglavje 4

Paralelni računalniki

Paralelni računalnik je računalniški sistem z več procesorji, ki podpira paralelno programiranje. Delimo jih v dve veliki skupini, v večračunalniške sisteme in večprocesorske sisteme [1]. Večračunalniški sistem je paralelni računalnik, sestavljen iz množice računalnikov in povezovalnega omrežja. Računalniki komunicirajo med seboj s pošiljanjem in sprejemanjem sporočil [1]. Centralizirani večprocesorski sistem, je sistem v katerem si vse centralno procesne enote delijo skupni globalni pomnilnik. To omogoča komunikacijo in sinhronizacijo med procesorji [1]. Paralelno programiranje je programiranje v jeziku ki omogoča eksplicitno označevanje odsekov programa, ki se lahko izvajajo istočasno na različnih procesorjih.

Zadnja leta je mogoče opaziti poudarek pri razvoju večjedrnih procesorjev. Razlog za to so predvsem fizične omejitve pri povečevanju frekvenc procesorjev. Povečale so se možnosti za paralelno izvajanje algoritmov, saj so večjedrni procesorji dandanes dostopni vsakomur. Vse bolj se uveljavlja tudi uporaba grafičnih procesorjev za splošno namensko računanje. Ti masivno paralelni procesorji so bili v preteklosti namenjeni preračunavanju grafike, sedaj pa lahko njihovo procesorsko moč izkoristimo tudi pri reševanju drugih problemov. Uveljavili so se hibridni paralelni računalniki, ki v večračunalniškem sistemu združujejo večprocesorske sisteme, ti pa vsebujejo večjedrne procesorje in predvsem v zadnjem času še grafične procesorje [1].

Napredek na področju strojne opreme, je gnal razvoj različnih orodij za izkoriščanje potenciala paralelnih računalnikov. Pojavili so se različni vmesniki, s katerimi lahko izkoriščamo posamezne elemente paralelnega računalnik. To so npr. vmesnik MPI, ki je posebej primeren za izkoriščanje večračunalniškega sistema, vmesnik OpenMP, ki omogoča učinkovito izrabo večprocesorskih sistemov in večjedrnih procesorjev ter vmesnik CUDA, ki omogoča izrabo grafičnih procesnih enot.

4.1 Snovanje paralelnih algoritmov

Razvoj paralelnih programov se razlikuje od razvoja klasičnih zaporednih programov. Zahteva posebno vrsto razmišljanja, ki upošteva vse značilnosti ciljnih paralelnih programov. Ian Foster je v pomoč pri razvoju paralelnih programov predlagal postopek, ki vključuje vse bistvene korake razvoja takšnega programa in olajša snovanje učinkovitih paralelnih programov, posebno tistih, ki se izvajajo na paralelnih računalnikih s porazdeljenim pomnilnikom [2].

Postopek sestavljajo štirje koraki: delitev (ang. partitioning), komunikacija (ang. communication), združevanje (ang. agglomeration) in preslikava (ang. mapping).

- **Delitev**, je proces s katerim razdelimo podatke in računske operacije na majhne dele. V primeru delitve podatkov, govorimo o podatkovni delitvi, če delimo računske operacije, pa o funkcijski delitvi. Tako razdrobljene podatke in računske operacije imenujemo osnovna opravila in so predmet porazdeljevanja na paralelnem računalniku.

- **Komunikacija**, določa komunikacijski vzorec med osnovnimi opravili. Predstavlja dodaten strošek (ang. overhead) paralelnega algoritma, saj je to nekaj, česar zaporedni algoritem ne potrebuje. Minimizacija komunikacije je tako pomemben cilj pri razvoju paralelnega algoritma, saj pomembno vpliva na njegovo učinkovitost.

- **Združevanje**, osnovna opravila je potrebno združevati v večja sestavljena opravila zato, da se izboljšajo lastnosti paralelnega algoritma in poenostavi programiranje. Poglavitni cilj je zmanjšati strošek komunikacije na minimalno raven.

- **Preslikava**, je proces dodeljevanja opravil procesorjem, pri tem moramo bit pozorni, predvsem na enakomerno izkoriščenost procesorjev in zmanjševanje medprocesorske komunikacije.

Poleg samega postopka razvoja, je potrebno definirati tudi metrike, s katerimi na koncu ocenimo uspešnost paralelizacije. Zanimata nas predvsem pohitritev (ang. speedup) in učinkovitost (ang. efficiency) paralelnega algoritma.

Pohitritev označimo z S_p in definiramo kot razmerje med časom izvajanja sekvenčnega algoritma in časom izvajanja vzporednega algoritma, ki reši enak problem z uporabo p procesorjev [1]. Pri tem predpostavimo, da je vsak izmed

p procesorjev enak procesorju na katerem je bil izmerjen čas sekvenčnega algoritma. Če s T_1 označimo čas izvajanja sekvenčnega algoritma in s T_p čas izvajanja vzporednega algoritma na p procesorjih, lahko zapišemo pohitritev z enačbo:

$$S_p = \frac{T_1}{T_p} \quad . \tag{4.1}$$

Učinkovitost označimo z E_p in definiramo kot razmerje med pohitritvijo S_p in številom procesorjev p [1]. Na idealnem paralelnem računalniku je učinkovitost enaka 1. Zapišemo lahko enačbo:

$$E_p = \frac{S_p}{p} = \frac{T_1}{T_p \times p} \quad . \tag{4.2}$$

4.2 OpenMPI

Vmesnik MPI (ang. Message Passing Interface) je trenutno najbolj razširjen standard za paralelno programiranje [1]. OpenMPI je prosto dostopna odprtokodna implementacija tega programskega vmesnika [9], kot je predpisan iz strani združenja Message Passing Interface Forum [13].

Kot pove že samo ime vmesnika, poteka komunikacija med procesi z izmenjevanjem sporočil. OpenMPI v ta namen uporablja TCP protokol in leno vzpostavljanje povezave. Slednje pomeni, da se povezava vzpostavi šele, ko je to potrebno, in ne na začetku izvajanja programa, ki uporablja OpenMPI. Ko je povezava enkrat vzpostavljena, se jo do konca izvajanja ne zapre [9].

OpenMPI vsebuje podporo za povezave, hitrejše od običajnega LAN omrežja, kot na primer InfiniBand. Za prenos podatkov preko omrežja OpenMPI avtomatsko izbere najhitrejši način povezave, ki je na voljo [9]. Podprto je veliko število funkcij, ki jih prepoznamo po predponi MPI. V grobem jih ločimo v dva sklopa: elementarne funkcije in funkcije za kolektivno komuniciranje. Če naštejemo nekaj pomembnejših elementarnih: MPI_Init, MPI_Finalize, MPI_Comm_size, MPI_Comm_rank, MPI_Send, MPI_Recv. Pomembnejše funkcije za kolektivno komuniciranje pa so: MPI_Bcast, MPI_Reduce, MPI_Barrier, MPI_Scatter, MPI_Gather. Dokumentacijo in seznam vseh funkcij lahko najdemo v [12]. Nekatere izmed teh funkcij smo uporabili pri paralelni implementaciji algoritma predstavljenega v poglavju 5.

Vmesnik nudi podporo tudi heterogenim okoljem. Mogoča je komunikacija med programi napisanimi v različnih programskih jezikih: C, C++ in Fortran.

Možno je tudi izvajanje na arhitekturno heterogenih okoljih z različno strojno opremo in operacijskimi sistemi.

Poglavje 5

Paralelizacija algoritma

5.1 Ideja

Cilj, ki ga želimo doseči s paralelizacijo algoritma 2, je skrajšanje časa izvajanja algoritma, v smislu, da za dosego enako kvalitetnega rezultata porabimo manj časa kot s sekvenčnim algoritmom. To lahko storimo na več načinov.

Pohitrimo lahko izvajanje algoritma, tako da izkoristimo paralelnost, ki jo izkazuje algoritem in porazdelimo delo med več procesorjev. Pohitritev pri takšnem načinu paralelizacije opisuje Amdahlov zakon [3], ki določa največjo možno pohitritev algoritma s paralelnim računalnikom s p procesorji. Z f označimo delež zaporednih operacij, ki jih ne moremo pohitriti, velja:

$$S_p \leq \frac{1}{f + \frac{1-f}{p}} \quad . \tag{5.1}$$

Predpostavimo, da imamo na voljo neskončno število procesorjev. Poskusimo izračunati pohitritev, ki jo dosežemo v takem primeru. Limitiramo enačbo (5.1) po številu procesorjev, dobimo:

$$\lim_{p\to\infty} S_p \leq \frac{1}{f + \frac{1-f}{p}} \leq \frac{1}{f} \quad . \tag{5.2}$$

Iz enačbe (5.2) vidimo, da zgornjo mejo pohitritve omejuje delež zaporednih operacij v algoritmu.

Problema, kako pohitriti algoritem se lahko lotimo tudi iz drugega zornega kota. Namesto krajšanja časa izvajanja lahko z večanjem števila procesorjev izboljšamo kakovost rezultata, ki ga dobimo pri danih parametrih algoritma. Tak način paralelizacije nam omogoča uporaba metode Monte Carlo. Ta ni

primerna za uporabo pri paralelizaciji kakršnihkoli algoritmov, vendar se v našem primeru izkaže za učinkovito.

5.1.1 Metoda Monte Carlo

Monte Carlo metode so stohastične simulacijska metode, ki s pomočjo naključnih števil in velikega števila izračunov simulirajo kompleksne fizikalne in matematične sisteme. Pojavile so se leta 1946 pri načrtovanju termonuklearne bombe. Uporabili so jih za simulacijo gibanja nevtronov. Prvi avtorji so bili Stanislaw Ulam, John von Neumann in Nick Metropolis [11]. Poznamo jih veliko vrst, implementacija je namreč močno odvisna od same problemske domene, ki jo rešujemo, v grobem pa jih opisujejo sledeči koraki:

1. Definiraj domeno vhodnih podatkov.
2. Generiraj naključne vhodne podatke z uporabo izbrane verjetnostne porazdelitve.
3. Izvedi deterministične izračune nad generiranimi vhodnimi podatki.
4. Združi rezultate posameznih izračunov v končni rezultat.

Napaka metode Monte Carlo pada s $\sqrt{N}$, kjer je N število vhodnih podatkov, ta ugotovitev sledi iz zakona o velikih številih [10].

Metodo je mogoče enostavno uporabiti, kot ogrodje za paralelni algoritem. Vsak od p procesorjev naključno generira vhodne podatke, nad katerimi vrši procesiranje neodvisno od ostalih. Po končanem procesiranju se rezultati na posameznih procesorjih združijo v končno rešitev. Rezultat procesiranja se torej izboljšuje s kvadratnim korenom števila procesorjev, pri tem je čas procesiranja praktično konstanten, saj je komunikacija med procesorji potrebna samo na koncu pri združevanju rezultatov. Ta metoda nam posredno omogoča tudi zmanjševanje časa procesiranja. Če med napako rezultata ε in časom procesiranja T velja odvisnost $\varepsilon \propto \frac{1}{\sqrt{T}}$, kar kot bomo videli v šestem poglavju za naš algoritem drži, potem lahko z uporabo p procesorjev pridemo v p-krat krajšem času, do enakega rezultata kot sekvenčni algoritem. Teoretična pohitritev z uporabo metode Monte Carlo je torej $S_p = p$. Iz tega sledi, da je učinkovitost (4.2), vsaj teoretično v tem primeru enaka 1. Naš cilj, ki smo si ga postavili v uvodnem poglavju pa je bil ravno maksimizacija učinkovitosti paralelnega algoritma.

V nadaljevanju je prikazana implementacija paralelnega algoritma, ki se poslužuje metode Monte Carlo.

5.2 Snovanje paralelnega algoritma

Pri izdelavi paralelne inačice algoritma 2, smo uporabili Fosterjevo metodologijo opisano v poglavju 4.1.

5.2.1 Delitev

Prvi korak je delitev podatkov. Iz opisa metode Monte Carlo sledi, da moramo generirati čim večje število naključnih vhodnih podatkov, nad katerimi se nato vrši procesiranje. V našem primeru je najmanjši možen podatek lahko le celotna konfiguracija prostora z naključno postavljenimi začetnimi položaji delcev. Osnovni opravek pa celoten sekvenčni algoritem 2.

5.2.2 Komunikacija

Vsak procesor deluje neodvisno od ostalih, s tem je nivo komunikacije zmanjša na najmanjšo možno raven. Ko vsi procesorji končajo s procesiranjem, se izvede operacija redukcije, ki združi vse delne prispevke v končni rezultat.

5.2.3 Združevanje in preslikava

Ta korak je v našem primeru trivialen, saj ne potrebujemo dodatnega združevanja osnovnih opravkov in podatkov, ker so ti dovolj kompleksni. Vsaka začetna konfiguracija prostora in sekvenčni algoritem 2 lahko zasede svoj procesor.

Parametri
Parametri paralelnega algoritma so enaki kot pri izboljšanem algoritmu 2.

Funkcije
Naštete so samo funkcije, ki smo jih dodali, opis ostalih najdemo pri izboljšanem algoritmu 2.

- *Glavni*: preveri ali je proces vodja, ta na koncu zbere vse rezultate in jih združi.
- $PosljiGlavnemuPrispevek(PoljeHitrost, PoljeTlak)$: pošlje glavnemu procesu, vodji, rezultat procesiranja.
- *ZberiPrispevke*: zbere vse rezultate, ki so jih prispevali posamezni procesi in jih združi v končni rezultat.

5.3 Psevdokoda paralelnega algoritma

Algoritem 3 Paralelni algoritem

```
DO IN PARALELL
for vsak delec i do
  Polozaj[i] ← NakljucenPolozaj
  Hitrost[i] ← 0
end for
while cas < CasKonec do
  if cas > CasZacetekBelezenjaStatistike then
    for vsak delec i do
      PoljeHitrost[Polozaj[i]] ← PoljeHitrost[Polozaj[i]] + Hitrost[i]
      PoljeTlak[Polozaj[i]] ← PoljeTlak[Polozaj[i]] + 1
    end for
  end if
  Ponastavi(Mreza)
  for vsak delec i do
    Hitrost[i] ← Hitrost[i] − Tezisce[i] · G · T
    Polozaj[i] ← Polozaj[i] + Hitrost[i] · T
    HitrostZacasna[i] ← Hitrost[i]
    DodajVMrezo(i, Mreza[Polozaj[i]/δ])
  end for
  for vsak delec i do
    for vsak delec j vOkolica(Mreza[Polozaj[i]/δ]) do
      if i ≠ j then
        Razdalja ← |Polozaj[i] − Polozaj[j]|
        if Razdalja < δ then
          Tezisce[i] ← Tezisce[i] + Polozaj[j] − Polozaj[i]
          VSum ← V × (HitrostZacasna[j] − HitrostZacasna[i])
          Vn ← Vn + 1
        end if
      end if
    end for
    Hitrost[i] ← Hitrost[i] + VSum/Vn
  end for
  for vsak delec i do
    if IzvenProstora(i) then
      VrniVProstor(i)
    end if
    if DelecVDRez(i) then
      if PrevecDelcevVDRez then
        OdstraniDelec(i)
      end if
    end if
    if PremaloDelcevVLRez(Nlevi) then
      DodajDelecVLRez
    end if
  end for
```

$PosljiGlavnemuPrispevek(PoljeHitrost, PoljeTlak)$
if *Glavni* **then**
  $ZberiPrispevke$
end if
END PARLLEL DO

Psevdokoda algoritma 3 je služila, kot podlaga za implementacijo v programskem jeziku C. Uporabljen je bil prosto dostopni prevajalnik GCC. Vmesnik za komunikacijo je knjižnica OpenMPI opisana v poglavju 4.2. Zbiranje delnih rezultatov je izvedeno s pomočjo funkcije za kolektivno komunikacijo MPI_Reduce.

Poglavje 6

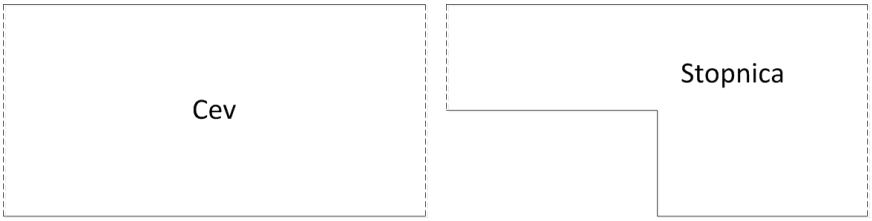
Eksperimenti

6.1 Eksperimentalno okolje

Kot eksperimentalno okolje je služil superračunalnik Ljubljana Supercomputing Center - LSC Adria, ki je v lasti podjetja Turboinštitut, d.d. in se uporablja predvsem za modeliranje dinamike tekočin, kar je tudi namen algoritma predstavljenega v tem delu. Sestavljen je iz 256 strežnikov, nameščenih v enote IBM BladeCenter H, vsak vsebuje dva štirijedrna procesorja Intel Xeon E5530 in 16GB pomnilnika. Za povezavo med strežniki skrbita dve gigabitni omrežji LAN in eno omrežje InfiniBand. Na vsakemu od strežnikov je bil nameščen Operacijski sistem Linux CentOs 4.6, arhitektura x86-64, strežniška različica s podporo za SMP. Pri prevajanju algoritmov, je bil uporabljen prevajalnik GCC 3.4.6 in knjižnica OpenMPI 1.3.2, oba že predhodno nameščena na sistemu. Superračunalnik je bil junija leta 2010 na podlagi LINPACK primerjalnega preizkusa uvrščen na 201. mesto lestvice svetovnih superračunalnikov TOP500 [8].

6.2 Parametri eksperimenta

Cilj je bil preizkusiti, tako pravilnost delovanja algoritma, kot tudi učinkovitost paralelizacije. V prvem delu eksperimenta smo algoritem pognali nad dvema konfiguracijama delovnega področja: običajna cev in stopnica. Rezultat procesiranja smo vizualizirali in ocenili ali algoritem deluje v skladu z fizikalnimi zakoni dinamike tekočin.



(a) Cev. (b) Stopnica.

Slika 6.1: Različne konfiguracije delovnega področja: a) cev, b) stopnica.

Drugi del eksperimentov, se je nanašal na ugotavljanje vpliva parametrov algoritma, to je število iteracij, število delcev v prostoru ter število procesorjev, na kvaliteto rezultata in čas procesiranja. Privzeta je bila konfiguracija prostora v obliki cevi s stopnico. Za vsak nabor vrednosti parametrov smo algoritem $N = 10$-krat pognali ter zabeležili čas procesiranja t_i in rezultat procesiranja, ki ga predstavlja polje tlaka in hitrosti. Izračunali smo povprečen čas izvajanja (6.1) in povprečni standardni odklon po vseh K količnikov ($x_{k,i} = \frac{v_{k,i}}{p_{k,i}}$), kjer sta $v_{k,i}$ in $p_{k,i}$ hitrost in tlak v k-ti celici polja. Dobljeno število služi kot metrika za oceno kakovosti rezultata. Manjši kot je povprečni standardni odklon rezultata boljša je kakovost rešitve. Enačbi obeh metrik sta:

$$T = \frac{1}{N} \sum_{i=1}^{N} t_i \quad , \tag{6.1}$$

$$\bar{\sigma} = \frac{1}{K} \sum_{k=1}^{K} \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (x_{k,i} - \bar{x_k})^2} \quad . \tag{6.2}$$

Število iteracij smo spreminjali od 20000 do 200000 s korakom 20000. Število delcev od 200 do 1200 s korakom 200 in število procesorjev od 1 do 48, najprej s korakom 1, do 8 procesorjev in nato s korakom 8 do 48.

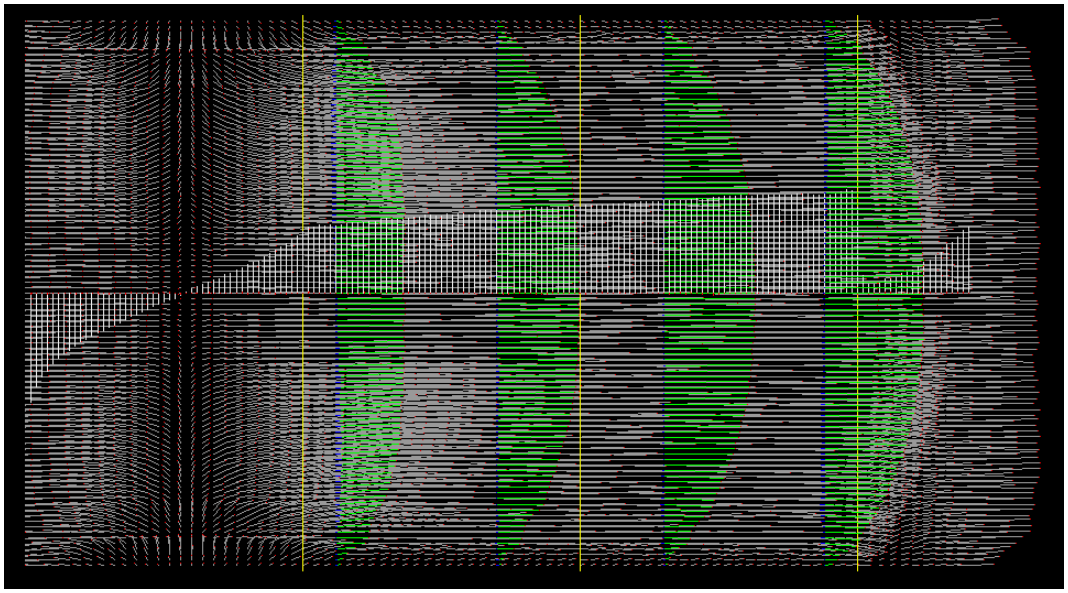
Poglavje 7

Rezultati

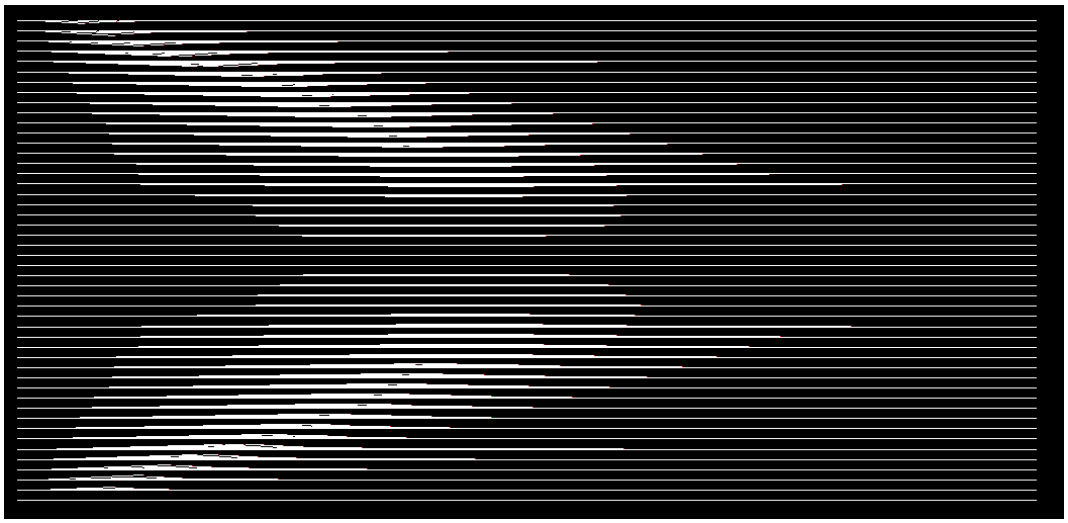
V tem poglavju so predstavljeni rezultati eksperimentov, ki smo jih pridobili z uporabo eksperimentalnega okolja in parametrov opisanih v poglavju 6. Najprej je prikazana analiza pravilnosti delovanja. Sledijo grafi odvisnosti časa procesiranja in standardnega odklona (napake) od števila delcev, števila iteracij in števila procesorjev. Na koncu je podan še graf,ki nazorno prikazuje učinkovitost paralelnega algoritma.

7.1 Rezultati procesiranja

Algoritem, ki smo ga predstavili v delu, je preveč osnoven, da bi lahko neposredno kvantitativno primerjali rezultat procesiranja z referenčno programsko opremo za računanje dinamike tekočin. Zato smo se oprli kar na vizualno primerjavo rezultatov našega algoritma in že uveljavljenega algoritma [7], za katerega vemo, da deluje pravilno. Primerjani sta izračunani polji hitrosti obeh konfiguracij delovnega področja to je, cev in cev s stopnico.

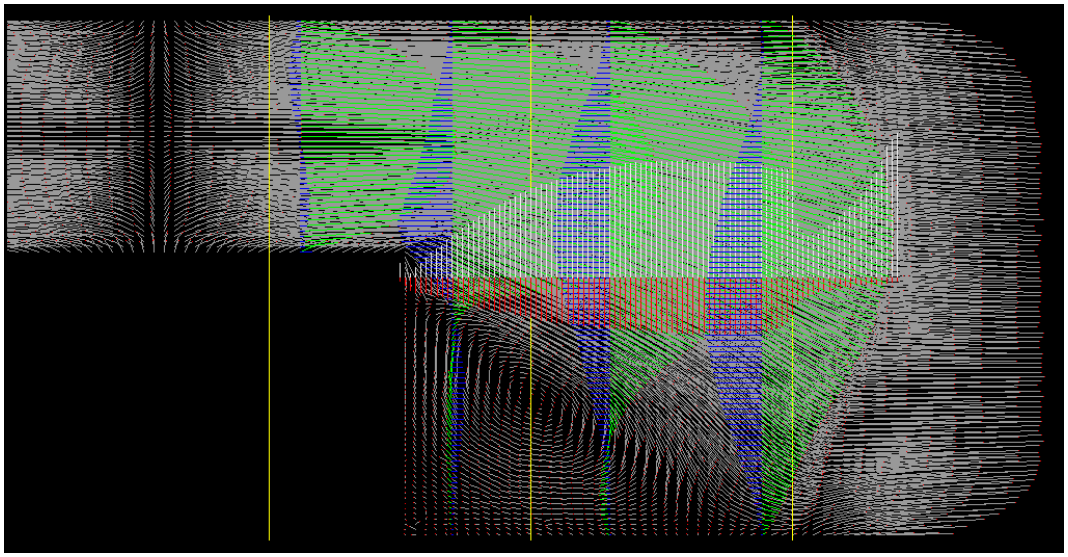


Slika 7.1: Polje hitrosti tekočine v cevi - naš algoritem.

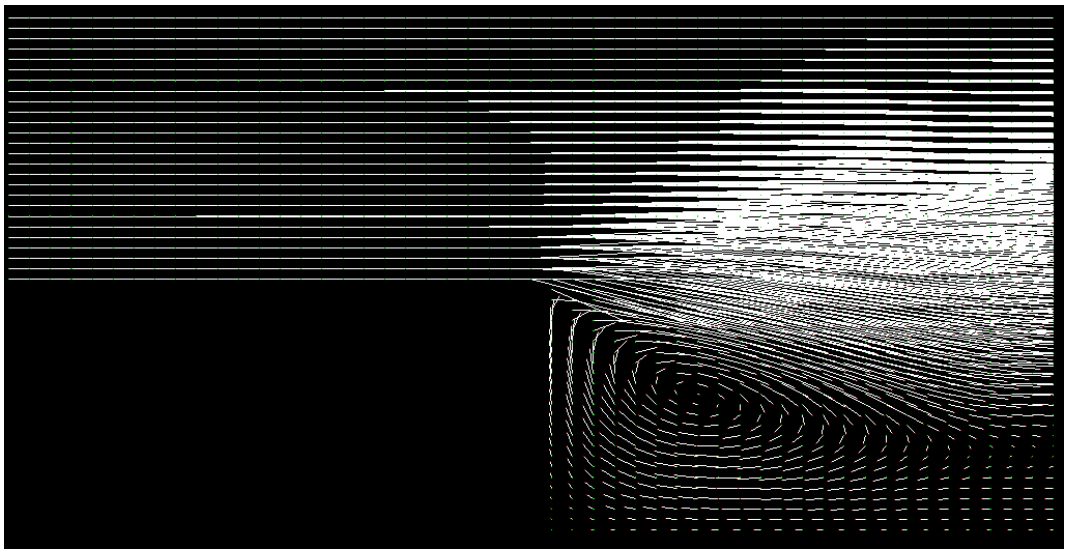


Slika 7.2: Polje hitrosti tekočine v cevi - referenčni algoritem.

Izrisani polji hitrosti v cevi sta si precej podobni, na obeh je razločno viden parabolični profil hitrostnih vektorjev, na sliki 7.1 dodatno poudarjen z zeleno barvo in se sklada z fizikalnimi zakonitostmi [4].



Slika 7.3: Polje hitrosti tekočine v cevi s stopnico - naš algoritem.



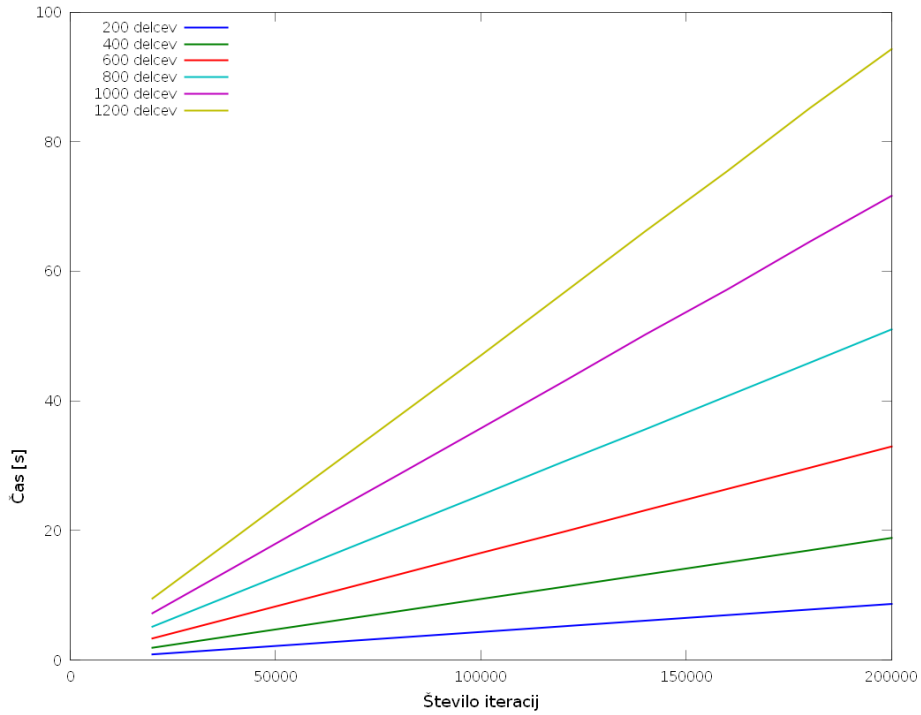
Slika 7.4: Rezultat modeliranja tekočine v cevi s stopnico - referenčni algoritem.

Tudi v primeru cevi s stopnico, je jasno vidna podobnost, merilo je predvsem vrtinčasta struktura, ki se je ustvarila ob vznožju stopnice na obeh slikah. Sklepamo lahko, da naš algoritem deluje pravilno, kljub temu pa ni

možna kvantitativna ocena pravilnosti, saj ne moremo vzpostaviti enakih robnih pogojev v obeh algoritmih, s čimer bi lahko rezultata primerjali neposredno numerično. Razlog za to je, da trenutno naš algoritem ne omogoča natančne postavitve robnih pogojev, tj. tlakov na obeh koncih cevi. Imamo le omejen nadzor z določanjem števila delcev v levem in desnem rezervoarju, tlačna razlika pa se vzpostavi samodejno, glede na to kako se delci razporedijo po prostoru.

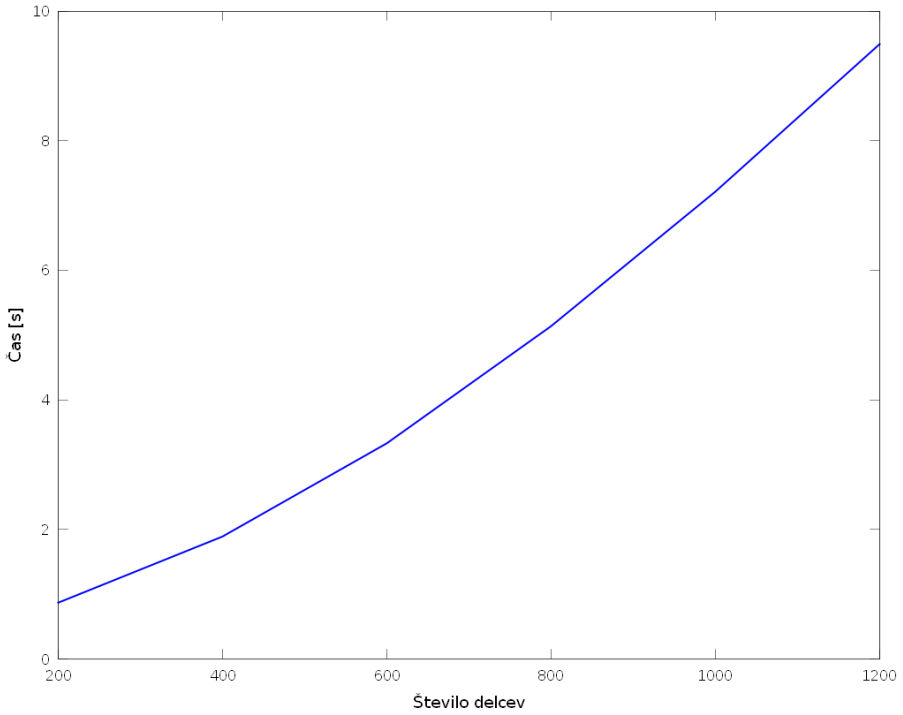
7.2 Čas procesiranja

Prikazan je čas procesiranja na enem procesorju v odvisnosti od števila iteracij. Vidimo, da čas narašča linearno z večanjem števila iteracij. Kar smo tudi pričakovali. Druga značilnost, ki jo opazimo je, da je naklon premice časovne odvisnosti proporcionalen številu delcev. Več kot je delcev, bolj strma je premica naraščanja časa.



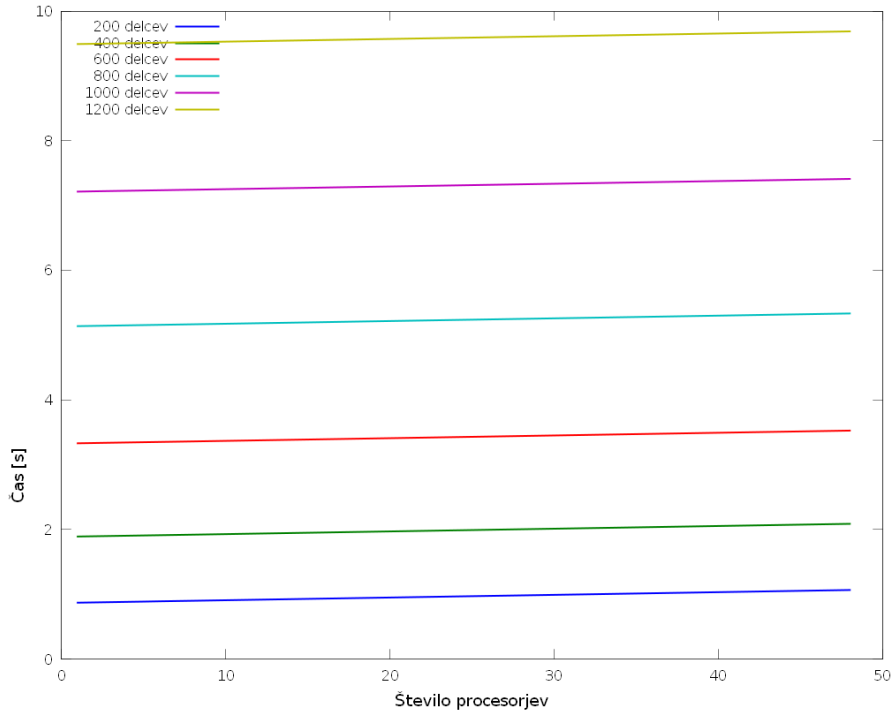
Slika 7.5: Čas procesiranja v odvisnosti od števila iteracij na enem procesorju za različna števila delcev.

Večanje števila delcev povzroči zelo položno kvadratno rast časa procesiranja. Rezultat ni presenetljiv, saj smo takšno obnašanje napovedali pri analizi izboljšanega algoritma v poglavju 3.2.



Slika 7.6: Čas procesiranja v odvisnosti od števila delcev pri 20000 iteracijah in enem procesorju za različna števila delcev.

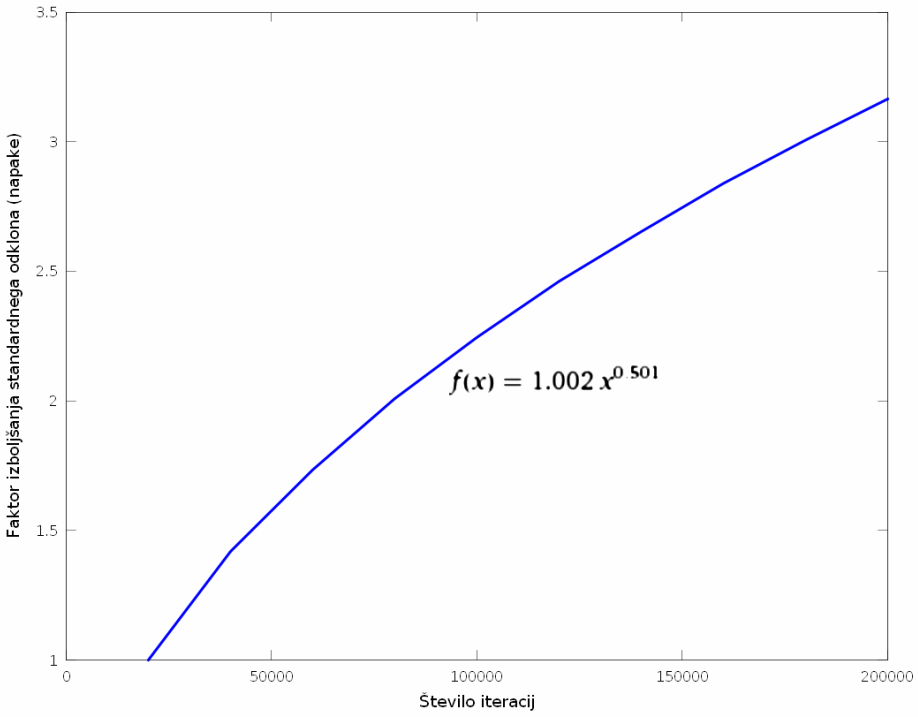
Večanje števila procesorjev praktično ne spreminja časa procesiranja pri enakem številu delcev, kar je razumljivo. Komunikacija med procesorji je prisotna le na koncu pri zbiranju rezultatov. Ta lastnost paralelnega algoritma nam omogoča visoke učinkovitosti, saj je komunikacija predstavlja praktično zanemarljiv dodatni strošek pri procesiranju.



Slika 7.7: Čas procesiranja v odvisnosti od števila procesorjev.

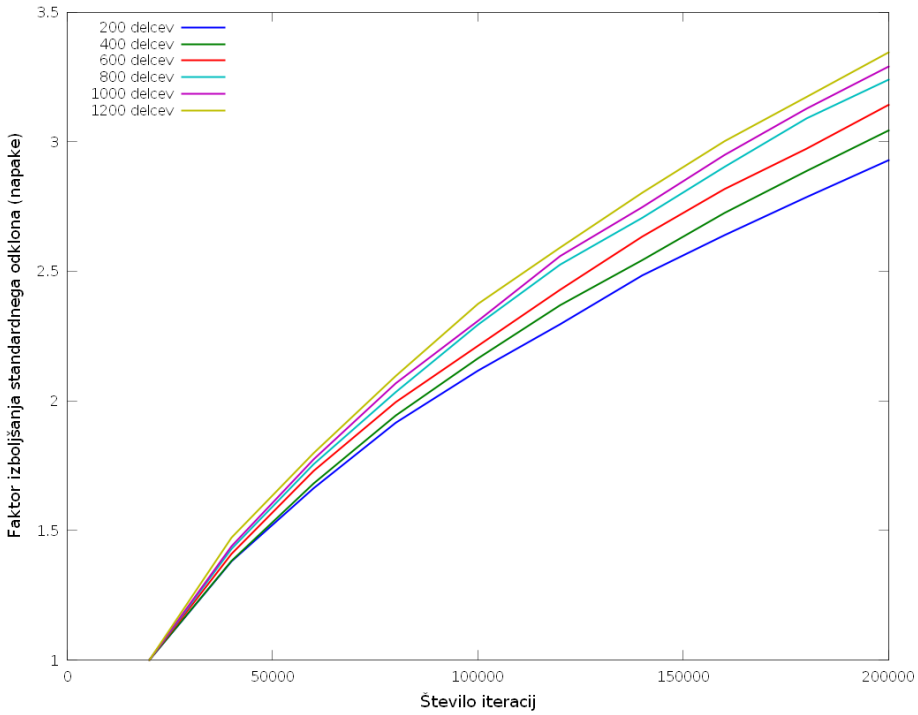
7.3 Standardni odklon

Prikazan je faktor zmanjševanja standardnega odklona z večanjem števila iteracij. Podatki so bili povprečeni po vseh vrednostih, ki jih zavzameta parametra število procesorjev in število delcev. Vidimo, da se graf zelo lepo prilega korenski funkciji. Štirikratno povečanje števila iteracij povzroči dvakratno zmanjšanje napake. Čas procesiranja se, kot smo videli na sliki 7.5 povečuje linearno, kar pomeni, da je učinkovitost povečevanja števila iteracij enaka $\frac{1}{\sqrt{I}}$, kjer je I število iteracij.



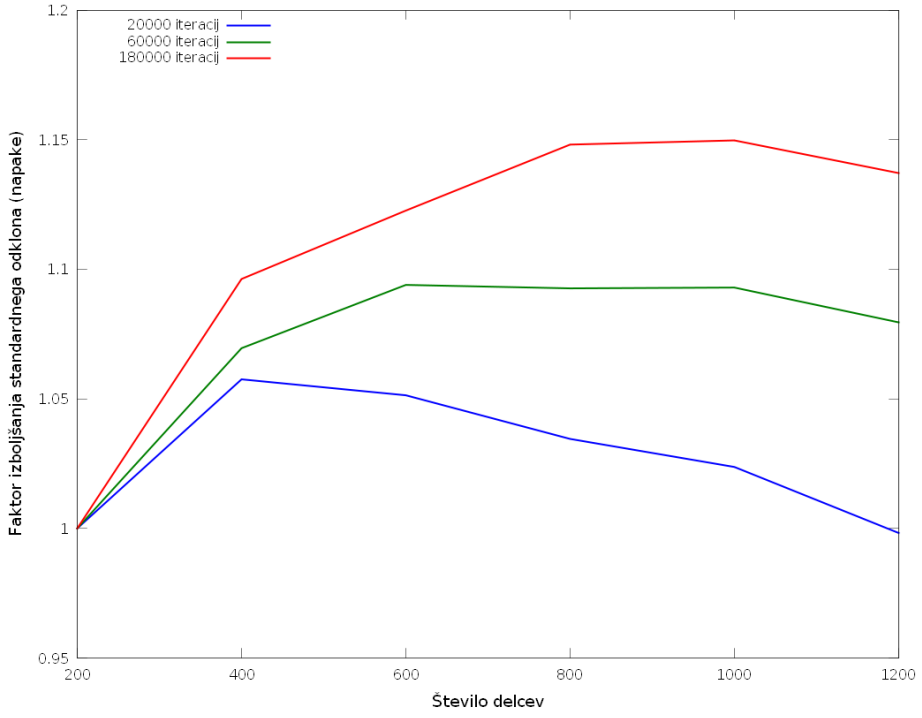
Slika 7.8: Zmanjševanje standardnega odklona s številom iteracij.

Graf prikazuje kako večanje števila iteracij vpliva na izboljševanje standardnega odklona pri različnem številu delcev. Podatki so bili povprečeni po vseh vrednostih, ki jih zavzame parameter število procesorjev. Vidimo, da je z večanjem števila delcev povečevanje števila iteracij nekoliko bolj učinkovito. Večje kot je število delcev hitreje se manjša napaka. Razlog leži v tem, da večje število delcev v vsaki iteraciji podrobneje opiše prostor, kar pripomore k hitrejši konvergenci.



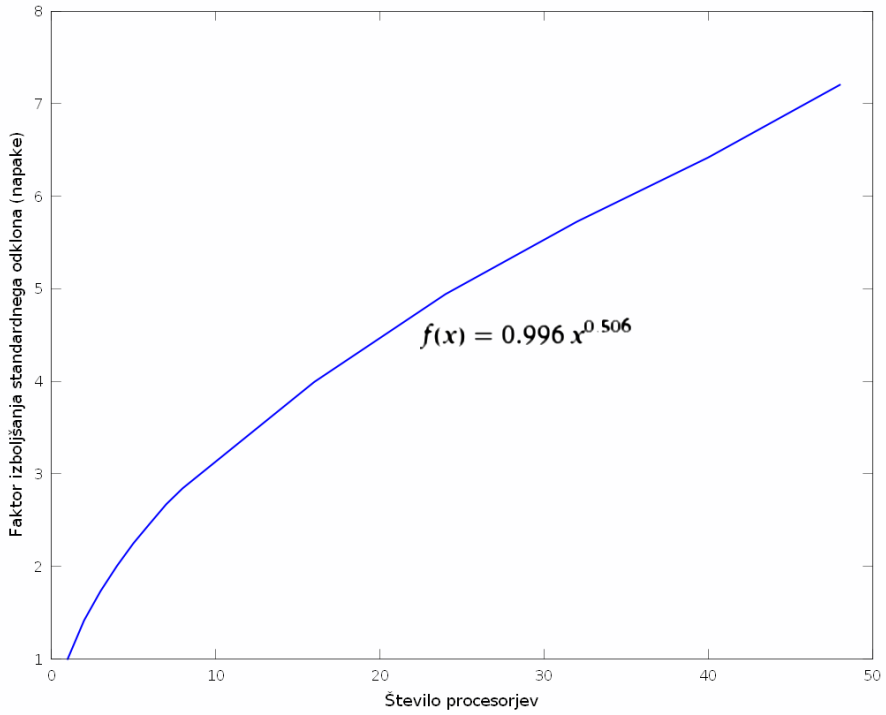
Slika 7.9: Zmanjševanje standardnega odklona s številom iteracij za različna števila delcev.

Graf prikazuje kako število delcev vpliva na izboljševanje standardnega odklona. Podatki so bili povprečeni po vseh vrednostih parametra število procesorjev. Povečevanje števila delcev je, kot vidimo, zelo neučinkovito. Zmanjšanje napake je zanemarljivo v primerjavi s povečanjem časa procesiranja. Pri štirikratnem povečanju števila delcev iz 200 na 800, se rezultat izboljša za približno 15 odstotkov, medtem ko se čas, kot vidimo iz slike 7.6 poveča za faktor 5. Opazimo tudi, da je za različna števila iteracij različno tudi optimalno število delcev, pri kateremu dosežemo najmanjšo napako. Pri 20000 iteracijah je to število okoli 400 delcev, pri 180000 iteracijah pa okoli 1000. Sklepamo lahko, da večje število delcev potrebuje več iteracij, da doseže optimalno izboljšanje, ker se z večanjem števila delcev zmanjšuje časovni korak T in s tem tudi skupen simulacijski čas pri enakem številu iteracij.



Slika 7.10: Zmanjševanje standardnega odklona s številom delcev za različna števila iteracij.

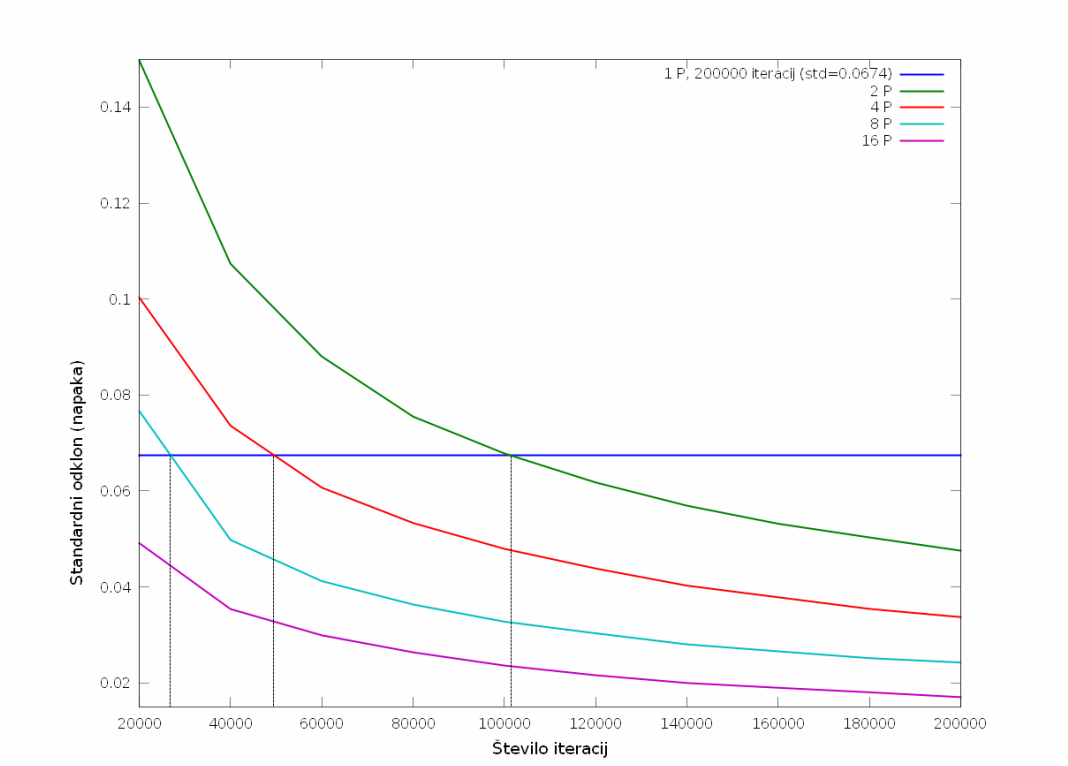
Prikazano je izboljševanje standardnega odklona z večanjem števila procesorjev. Podatki so bili povprečeni po vseh vrednostih, ki jih zavzameta parametra število iteracij in število delcev. Večanje števila procesorjev vpliva na izboljševanje standardnega odklona na enak način kot povečevanje števila iteracij. Štirikratno povečanje števila procesorjev povzroči dvakratno zmanjšanje napake. Pri tem se čas procesiranja, kot smo videli na sliki 7.7, praktično ne poveča. Učinkovitost paralelnega algoritma je zato zelo blizu idealni učinkovitosti 1.



Slika 7.11: Zmanjševanje standardnega odklona s številom procesorjev.

7.4 Učinkovitost

Vodoravna črta na spodnjem grafu označuje standardni odklon dosežen na enem procesorju pri 200000 iteracijah. Število molekul znaša 200, Krivulja zmanjševanja napake pri dveh procesorjih jo seka pri približno 100000 iteracijah, kar se sklada z teoretično napovedjo v poglavju 5.1.1. Trend se nadaljuje tudi pri štirih, osmih in šestnajstih procesorjih. Podvojitev števila procesorjev praktično razpolovi čas procesiranja v primeru, da želimo doseči neko vnaprej določeno napako.



Slika 7.12: Primerjava izboljševanja standardnega odklona pri različnem številu procesorjev glede na 1 procesor in 200000 iteracij.

Rezultati prikazani v tem poglavju potrjujejo našo napoved o učinkovitosti paralelnega algoritma. Dosežena pohitritev S_p je praktično enaka številu procesorjev.

Poglavje 8

Zaključek in nadaljnje delo

V pričujočem delu smo se na kratko seznanili z področjem dinamike tekočin. Naš glavni cilj je bil implementacija učinkovitega paralelnega algoritma za modeliranje dinamike tekočin. Najprej smo razvili sekvenčni algoritem, ki smo ga nato z vpeljavo določenih optimizacij nekoliko izboljšali.

Predstavili smo modele paralelnih računalnikov in postopek za snovanje paralelnega algoritma. Definirali smo metrike, s katerimi ocenjujemo kvaliteto paralelnih algoritmov. Predstavili smo vmesnik OpenMPI, namenjen komunikaciji med procesi, ki tečejo na paralelnem računalniku.

Paralelni algoritem smo osnovali na ideji, da lahko s paralelnim računalnikom izboljšamo kvaliteto rezultata procesiranja in s tem posredno tudi čas. Podlaga zanjo je metoda Monte Carlo s katero smo se podrobneje seznanili in opisali kako lahko z njeno pomočjo paraleliziramo algoritem. Pri paralelizaciji smo se poslužili Fosterjeve metodologije.

Predstavili smo eksperimentalno okolje na katerem smo izvajali eksperimente in opisali njihov potek. Pri analizi rezultatov, smo se posvetili tako ugotavljanju pravilnosti delovanja algoritma, kot tudi učinkovitosti paralelizacije. Pravilnost delovanja smo ugotovili kar vizualno, tako da smo sliko rezultata primerjali s sliko pridobljeno z uporabo referenčnega algoritma. Podatki pridobljeni s pomočjo eksperimentov so se skladali s predikcijami, ki smo jih postavili pri razvoju algoritma. Paralelna implementacija se je izkazala kot zelo učinkovita.

Razviti algoritem je trenutno na precej primitivni stopnji in kot tak primeren le za osnovno raziskovanje smotrnosti uporabe predstavljenega pristopa za modeliranje dinamike tekočin. Potrebnega je še veliko dela, da bi lahko metodo priredili za modeliranje realnih kompleksnih problemov. Težave povzročajo predvsem robni pogoji, ki jih trenutno še ne moremo natančno določiti. Zan-

imiva bi bila tudi paralelizacija algoritma z uporabo grafičnih procesnih enot. V tem primeru bi verjetno lahko precej pohitrili izvajanje algoritem tudi brez uporabe metode Monte Carlo. Združitev obeh pristopov bi bila idealna za uporabo na hibridnem paralelnem računalniku, ki združuje večprocesorske sisteme in grafične procesorje.

Slike

Algoritmi

Literatura

[1] A. Dobnikar, U. Lotrič, *Porazdeljeni sistemi za modeliranje, paralelno programiranje in procesiranje*, Ljubljana: Fakulteta za računalništvo in informatiko, 2008, pogl. 3, 4, 5, 6.

[2] M. J. Quinn, "Parallel Alghoritm Design" *Parallel programming in C with MPI and OpenMP*, New York: McGraw-Hill, 2004, pogl. 3.

[3] D. Kodek, *Arhitektura računalniških sistemov*, Ljubljana: Bi-tim, 1994, pogl. 3.5.

[4] A. Likar, *Osebna korespondenca*, Ljubljana, 2010.

[5] T. Toffoli, N. Margolus *Cellular automata machines*, Cambridge: MIT Press, 1988.

[6] C.A.J. Fletcher, *Computational techniques for fluid dynamics 1: fundamental and general techniques*, Berlin: Springer-Verlag, 1988.

[7] C.A.J. Fletcher, *Computational techniques for fluid dynamics 2: specific techniques for different flow categories*, Berlin: Springer-Verlag, 1988.

[8] (2010) TOP500 Supercomputing Sites. Dostopno na: http://www.top500.org/list/2010/06/

[9] (2010) OpenMPI F.A.Q. Dostopno na: http://www.open-mpi.org/faq/

[10] (2010) Monte Carlo metoda za numerično integiranje. Dostopno na: http://www-lp.fmf.uni-lj.si/plestenjak/vaje/Nm/Gradivo/SeminarMC.pdf

[11] (2010) Monte Carlo Method. Dostopno na: http://mathworld.wolfram.com/MonteCarloMethod.html

[12] (2010) Open MPI v1.3.4 documentation. Dostopno na:
http://www.open-mpi.org/doc/v1.3/

[13] (2010) MPI documents. Dostopno na:
http://www.mpi-forum.org/docs/docs.html