Perbandingan Metode Integrasi Velocity Verlet dan Predictor-Corrector pada Dinamika Molekuler Secara Serial dan Paralel

Perdana Aditya Natayuda[1], Nurul Ikhsan[2], Reza Rendian S[3]

[1,2]Fakultas Informatika, Universitas Telkom, Bandung
[3]Fakultas Teknik Elektro, Universitas Telkom, Bandung
[1]natayuda@students.telkomuniversity.ac.id,
[2]ikhsan@telkomuniversity.ac.id,
[3]zaseptiawan@telkomuniversity.ac.id

Abstrak

Dinamika molekuler adalah suatu metode simulasi komputer yang mempresentasikan interaksi antara atom dan molekul selama periode waktu tertentu. Simulasi dinamika molekuler menggunakan besaran percepatan dan kecepatan partikel untuk menentukan posisi pergerakan partikel. Pada penelitian sebelumnya, simulasi dinamika molekuler biasa dilakukan dengan menggunakan metode *Velocity Verlet* dan *Predictor-Corrector*. Simulasi dinamika molekuler memerlukan waktu komputasi yang lama karena jumlah partikel yang sangat banyak. Oleh karena itu pada penelitian tugas akhir ini, digunakan metode *Velocity Verlet* dan *Predictor-Corrector* yang dijalankan secara paralel menggunakan *Compute Unified Device Architecture* (CUDA) yang bertujuan untuk mengurangi waktu komputasi. Dengan menjalankan simulasi paralel dengan CUDA, simulasi dinamika molekuler menggunakan metode *Velocity Verlet* dan *Predictor-Corrector* berjalan lebih cepat secara paralel dibandingkan dengan dijalankan secara serial di CPU dengan total *speedup* mencapai 2.9 kali lebih cepat.

Kata Kunci: CUDA, Dinamika Molekuler, Predictor-Corrector, Velocity Verlet.

Abstract

Molecular dynamics is a computer simulation method that presents interactions between atoms and molecules over a certain period of time. Molecular dynamics simulations use the amount of acceleration and particle velocity to determine the position of the particle's movement. In previous studies, molecular dynamics simulations are usually done using the Velocity Verlet and Predictor-Corrector methods. Molecular dynamics simulation requires a long computational time because of the large number of particles. Therefore, in this final project, Velocity Verlet and Predictor-Corrector methods are run in parallel using Compute Unified Device Architecture (CUDA) which aims to reduce computing time. By running parallel simulations with CUDA, molecular dynamics simulations using the Velocity Verlet and Predictor-Corrector methods run faster in parallel than those run serially on the CPU with a total speedup reaching 2.9 times faster.

Keywords: CUDA, Molecular Dynamics, Predictor-Corrector, Velocity Verlet.

1. Pendahuluan

Dinamika molekuler merupakan metode simulasi yang menggambarkan interaksi antar molekul atom berdasarkan waktu tertentu. Seperti contohnya kelereng dalam kotak, asumsikan kelereng adalah molekul-molekul yang saling berinteraksi ketika kotak digerakkan. Metode yang biasa digunakan pada simulasi dinamika molekuler diantaranya yaitu Velocity Verlet, Runge-Kutta, Predictor-Corrector, Leapfrog, dan Ljapunov-Characteristics[4]. Untuk penelitian ini akan digunakan metode Velocity Verlet dan Predictor-Corrector. Metode Velocity Verlet digunakan untuk mendapatkan perkiraan dari pergerakan molekul menggunakan kecepatan molekul tersebut sebagai variabel. Metode Predictor-Corrector menggunakan bagian Predictor yang akan melakukan prediksi pergerakan molekul menggunakan variabel yang telah ditentukan, lalu bagian Corrector akan melakukan koreksi terhadap nilai dari metode Predictor menggunakan variabel yang telah ditentukan. Metode yang digunakan pada simulasi dinamika molekuler memiliki algortima perhitungan yang rumit dengan waktu eksekusi yang lama. Tujuan utama dari simulasi dinamika molekuler adalah, mengetahui posisi stabil dari

interaksi antar molekul yang sangat banyak dan memungkinkan untuk melakukan simulasi yang tidak bisa dilakukan dalam laboratorium.

Karena banyaknya jumlah molekul yang disimulasikan dan besarnya beban komputasi yang dikerjakan, penelitian tugas akhir ini menggunakan CUDA (*Compute Unified Device Architecture)* yang merupakan platform komputasi paralel dan model antarmuka pemrograman aplikasi yang dibuat oleh NVIDIA. Platform CUDA dirancang untuk bekerja dengan bahasa pemrograman seperti C, C ++, dan Fortran. CUDA memudahkan pemrograman paralel untuk menggunakan sumber daya GPU yang dapat mempercepat proses komputasi. CUDA dapat berjalan di sistem operasi yang umum digunakan seperti Windows, Linux, dan MacOS.

1.1 Topik dan Batasannya

Pada penelitian tugas akhir ini, simulasi dinamika molekuler akan dijalankan sebanyak 1000 iterasi. Jumlah partikel yang disimulasikan berjumlah 1024 hingga 8096 partikel. Metode yang digunakan yaitu Velocity Verlet dan Predictor Corrector. Simulasi dijalankan secara serial pada CPU dan paralel dengan CUDA pada GPU.

1.2 Tujuan

Merujuk pada perumusan masalah yang diangkat pada penelitian ini, maka tujuan pada tugas akhir ini adalah sebagai berikut:

1. Melakukan simulasi dinamika molekuler menggunakan metode Velocity Verlet dan Predictor-Corrector.
2. Melakukan analisis waktu eksekusi algoritma antara metode Velocity Verlet dan Predictor-Corrector yang dijalankan secara serial dan paralel pada GPU.
3. Membandingkan kecepatan waktu eksekusi program yang dijalankan secara serial pada CPU dan paralel pada GPU.

2. Studi Terkait

2.1 Dinamika Molekuler

Dinamika molekuler dapat didefinisikan sebagai teknik simulasi komputer untuk memahami gerakan molekul. Tujuan utama dari dinamika molekuler adalah untuk mengetahui interaksi antar atom dalam jumlah banyak sehingga memungkinkan melakukan simulasi yang tidak dapat dilakukan di laboratorium.[12]

Secara fisis, simulasi dinamika molekuler yang dilakukan pada penelitian tugas akhir ini adalah simulasi dinamika molekuler gas ideal, karena memiliki asumsi teori untuk gas ideal sebagai berikut[13]:

- Molekul terdiri dari partikel-partikel sangat kecil, dengan massa tidak nol (m=1).
- Interaksi antarmolekul diwakili oleh perhitungan potensial Lennard-Jones. Molekul tidak mengeluarkan gaya satu sama lain, kecuali saat tumbukan terjadi.
- Molekul-molekul ini bergerak secara konstan sekaligus acak.

Dalam penelitian tugas akhir ini, digunakan perhitungan potensial Lennard-Jones untuk mencari nilai gaya dari molekul. Perhitungan potensial Lennard-Jones merupakan perhitungan potensial paling sederhana untuk melakukan simulasi dinamika molekuler.

$$V(r_i) = 4\varepsilon \left[\left(\frac{\sigma}{r_i}\right)^{12} - \left(\frac{\sigma}{r_i}\right)^{6}\right] \quad (2\text{-}1)$$

Potensial Lennard-Jones terdiri dari gaya tarik-menarik dan tolak-menolak. Parameter ε adalah kekuatan interaksi molekul. Parameter σ adalah posisi ideal antar molekul.

$$\mathrm{F}_i = -\frac{\partial}{\partial r_i} V(r_i) \quad (2\text{-}2)$$

Dengan $\mathrm{F}_i = \mathrm{m}_i . \vec{a}_i$, asumsikan m = 1 maka persamaan akan berubah menjadi,

$$\vec{a}_i = -\frac{\partial}{\partial r_i} V(r_i) \quad (2\text{-}3)$$

$$\vec{a}_i(r_i) = -\frac{\partial}{\partial r_i}\left(4\varepsilon\left[\left(\frac{\sigma^{12}}{{r_i}^{12}}\right) - \left(\frac{\sigma^{6}}{{r_i}^{6}}\right)\right]\right) \quad (2\text{-}4)$$

Asumsikan ∂ = 1,

$$\vec{a}_i(r_i) = \left(24\varepsilon\left[2\left(\frac{1}{{r_i}^{13}}\right) - \left(\frac{1}{{r_i}^{7}}\right)\right]\right) \quad (2\text{-}5)$$

Untuk menyerdahanakan permasalahan interaksi antar partikel maka diasumsikan konstanta 24ε bernilai 1 persamaan maka persamaan yang digunakan dalam program adalah,

$$\vec{a}_i(r_i) = \left(\left[2\left(\frac{1}{{r_i}^{13}}\right) - \left(\frac{1}{{r_i}^{7}}\right)\right]\right) \quad (2\text{-}6)$$

Setelah didapatkan nilai a dari satu partikel, dimiliki persamaan sebagai berikut,

$$\vec{a}_i = -\frac{d}{dr}\, v_i \quad (2\text{-}7)$$

Dari persamaan diatas nilai v dapat dihitung menggunakan persamaan berikut,

$$v_i = \int \vec{a}_i \; dr \quad (2\text{-}8)$$

Untuk mencari nilai r, persamaan didapat dari hubungan antara v_i dan r_i sebagai berikut,

$$\frac{d}{dt}\, r = v \quad (2\text{-}9)$$

$$r_i = \int v \; dt \quad (2\text{-}10)$$

2.2 Velocity Verlet

Metode integrasi Verlet adalah metode yang digunakan untuk mengintegrasikan persamaan gerak dari persamaan Newton[3]. Metode ini sering digunakan dalam permodelan simulasi dinamika molekuler. Metode integrasi verlet dibagi menjadi tiga, yaitu Basic, Leap-frog, dan Velocity. Algoritma ini memaksimalkan efisiensi memori pada komputer dan waktu eksekusi. Velocity Verlet memiliki persamaan sebagai berikut[4] :

$$\mathrm{r(t+\Delta t) = r(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2} \quad (2\text{-}11)$$

$$v(t + \Delta\mathrm{t}) = v(t) + \frac{a(t)+a(t+\Delta\mathrm{t})}{2}\Delta\mathrm{t} \quad (2\text{-}12)$$

Persamaan di atas merupakan persamaan Verlet. Implementasi pada metode Velocity Verlet adalah sebagai berikut :

1. Menghitung posisi saat (t +Δt)

$$r(t+\Delta t) = r(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 \quad (2\text{-}13)$$

2. Menghitung kecepatan pada ½ timestep

$$v\left(t+\frac{1}{2}\Delta t\right) = v(t) + \frac{1}{2}a(t)\Delta t \quad (2\text{-}14)$$

3. Menghitung kecepatan baru

$$r(t+\Delta t) = r(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 \quad (2\text{-}15)$$

Diketahui bahwa :

- r adalah posisi
- v adalah kecepatan
- a adalah akselerasi
- t adalah waktu
- Δt adalah timestep

2.3 Predictor Corrector

Predictor-Corrector memprediksikan solusi numerik dari suatu persamaan diferensial biasa orde pertama (metode Predictor) kemudian kita mengoreksinya dengan metode Corrector[9]. Predictor-Corrector memiliki persamaan sebagai berikut[4]:

1. Menghitung percepatan, kecepatan, dan posisi menggunakan metode predictor

$$r^p(t+\Delta t) = r(t) + \Delta t v(t) + 1/2\Delta t^2 a(t) \quad (2\text{-}16)$$

$$v^p(t+\Delta t) = v(t) + \Delta t a(t) \quad (2\text{-}17)$$

$$a^p(t+\Delta t) = a(t) \quad (2\text{-}18)$$

2. Menghitung gaya menggunakan nilai yang didapat dari metode predictor

3. Melakukan koreksi nilai menggunakan metode corrector

$$r^c(t+\Delta t) = r^p(t+\Delta t) + c_0\, \Delta a(t+\Delta t) \quad (2\text{-}19)$$

$$v^c(t+\Delta t) = v^p(t+\Delta t) + c_1\, \Delta a(t+\Delta t) \quad (2\text{-}20)$$

$$a^c(t+\Delta t) = a^p(t+\Delta t) + c_2\, \Delta a(t+\Delta t) \quad (2\text{-}21)$$

$$c0 = 1/5;\ c1 = 5/6;\ c2 = 1;\ c3 = 1/3$$

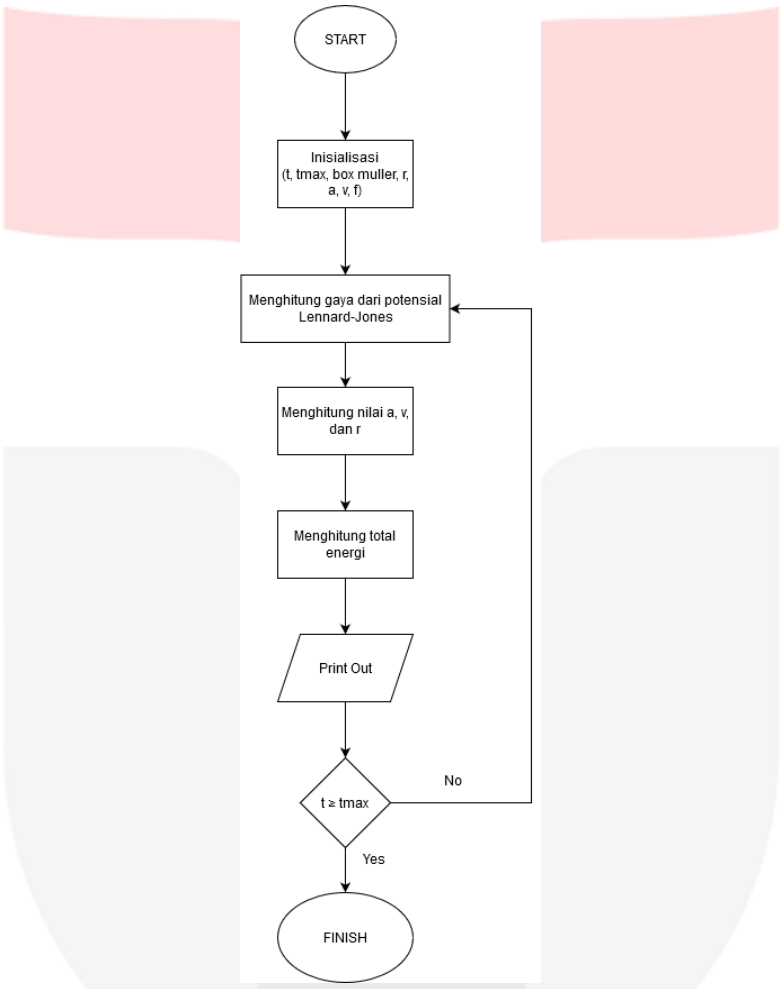
Diketahui bahwa :

- r adalah posisi
- v adalah kecepatan
- a adalah akselerasi
- t adalah waktu

- Δt adalah timestep
- c adalah koefisien terbaik untuk algoritma Predictor-Corrector

3. Sistem yang dibangun

Dalam penelitian tugas akhir ini, sistem dibangun untuk mengamati pergerakan molekul seiring berjalannya waktu. Untuk mengamati pergerakan tersebut digunakan metode Velocity Verlet dan Predictor-Corrector. Kedua metode tersebut dijalankan secara serial dengan CPU dan paralel dengan GPU dan diukur waktu eksekusinya.



Gambar 1 Flowchart program.

Program berjalan dengan melakukan inisialisasi variabel yang ingin digunakan. Setelah melakukan inisialisasi, dilakukan perhitungan potensial Lennard-Jones yang akan menghasilkan nilai F untuk digunakan menghitung a, v, dan r. Setelah mendapatkan a, v, dan r lalu variabel tersebut diperlukan untuk menghitung energi total. Program terus berjalan sampai t ≥ max.

Perhitungan secara paralel pada algoritma *Velocity Verlet* dan *Predictor-Corrector* dilakukan pada saat perhitungan Lennard-Jones yang menghasilkan nilai f.

Pseudocode untuk menghitung potensial Lennard-Jones pada program adalah sebagai berikut:

Tabel 1 Pseudocode potensial Lennard-Jones.

Algoritma: Menghitung potensial Lennard-Jones

```
Kamus:
                    i : Integer
                    j : Integer
                    n : Integer
                    l : Integer
                   dx : Float
                   dy : Float
                   dz : Float
                   dr : Float
                   rc : Float
                   fr : Float
Procedure Force(fx, fy, fz, n, l, rx, ry, rz)
rc = l/2.0

begin
  for(i = 0; i < n; i++)
    fx[i] = 0.0
    fy[i] = 0.0
    fz[i] = 0.0
end for

  for(i = 0; i < n; i++)
    for(j = 0; j < n; j++)
      if(i == j) continue
      dx = rx[i] - rx[j]
      dy = ry[i] - ry[j]
      dz = rz[i] - rz[j]

      if(dx < -rc) dx += l
      if(dy < -rc) dy += l
      if(dz < -rc) dz += l
      if(dx >  rc) dx -= l
      if(dy >  rc) dy -= l
      if(dz >  rc) dz -= l

      dr = sqrt(dx*dx + dy*dy + dz*dz)

      if(dr < rc)
        fr = 1.0/pow(dr,13) - 1.0/pow(dr,7)
        fx[i] += fr*dx/dr
        fy[i] += fr*dy/dr
        fz[i] += fr*dz/dr
      endif
    endfor
  endfor
```

4. Evaluasi

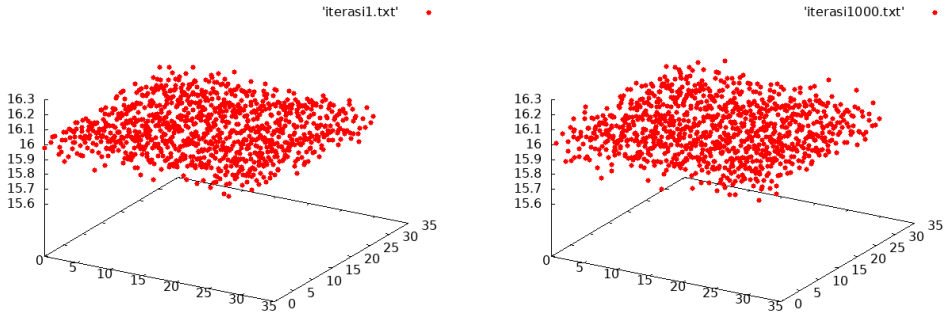
4.1 Hasil Pengujian

Penelitian tugas akhir ini dijalankan pada komputer dengan spesifikasi berikut:

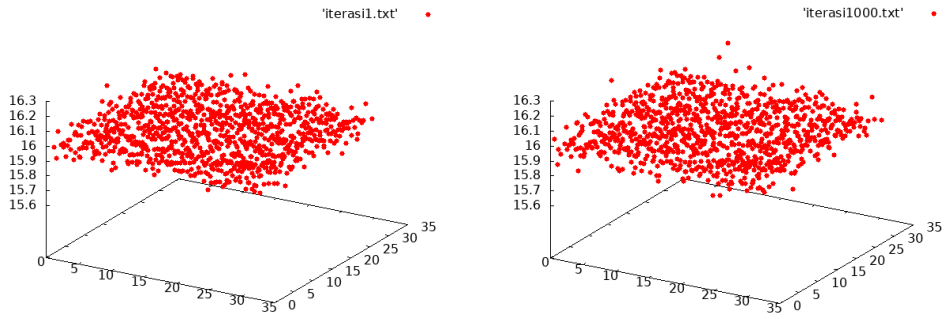
Tabel 4 Spesifikasi PC.

Processor	Intel® Core i5-9400F 4,10 GHz
Memory	16 GB DDR4 2133Mhz
Storage	1 TB
GPU	GTX 970 (1664 CUDA Core)
OS	CentOS 8

Setelah program dijalankan dengan variasi jumlah partikel 1024 sampai 8096 dengan 1000 iterasi. Didapatkan hasil sebagai berikut :



Gambar 2 Posisi simulasi Velocity Verlet pada iterasi 1 dan 1000.



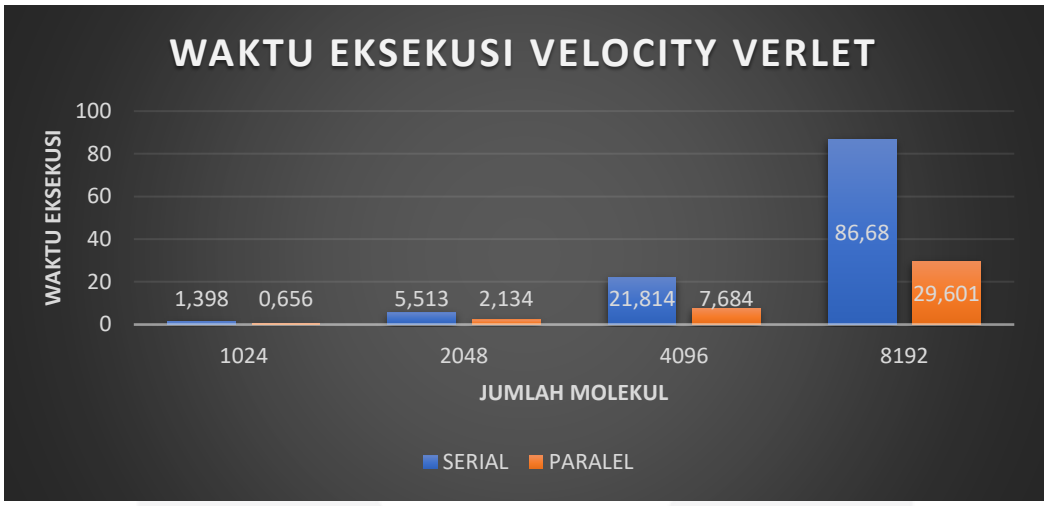
Gambar 3 Posisi simulasi Predictor Corrector pada iterasi 1 dan 1000.

Tabel 5 Speedup algoritma Velocity Verlet.

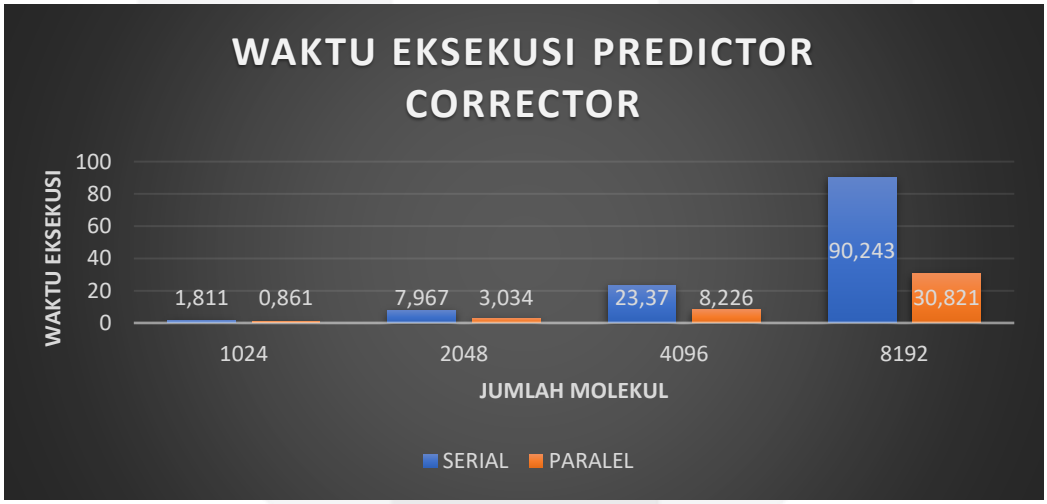
Jumlah Partikel	Waktu Eksekusi Serial	Waktu Eksekusi Paralel	Speedup
1024	1.398 s	0.656 s	2.131
2048	5.513 s	2.134 s	2.583
4096	21.814 s	7.684 s	2.827
8192	86.680 s	29.601 s	2.928

Tabel 6 Speedup algoritma Predictor Corrector.

Jumlah Partikel	Waktu Eksekusi Serial	Waktu Eksekusi Paralel	Speedup
1024	1.811 s	0.861 s	2.104
2048	7.967 s	3.034 s	2.626
4096	23.370 s	8.226 s	2.841
8192	90.243 s	30.821 s	2.928



Gambar 4 Perbandingan waktu eksekusi algoritma Velocity Verlet



Gambar 5 Perbandingan waktu eksekusi algoritma Predictor Corrector

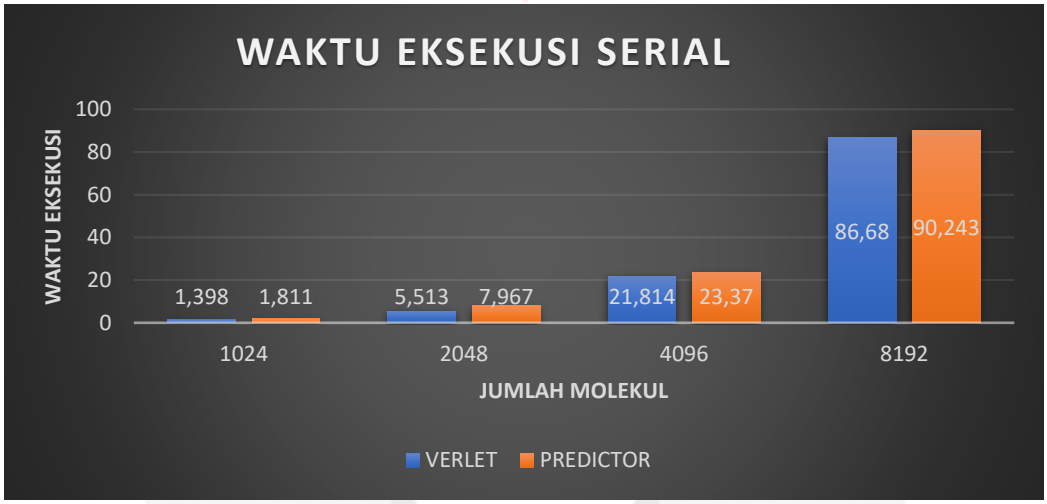
Waktu eksekusi program baik algoritma Velocity Verlet maupun Predictor Corrector terbukti lebih cepat saat dijalankan secara paralel dengan speedup rata-rata 2x lebih cepat. Ini membuktikan bahwa menjalankan simulasi dinamika molekuler pada CUDA dapat menghemat waktu.

Tabel 7 Selisih waktu algoritma Velocity Verlet dan Predictor Corrector secara serial.

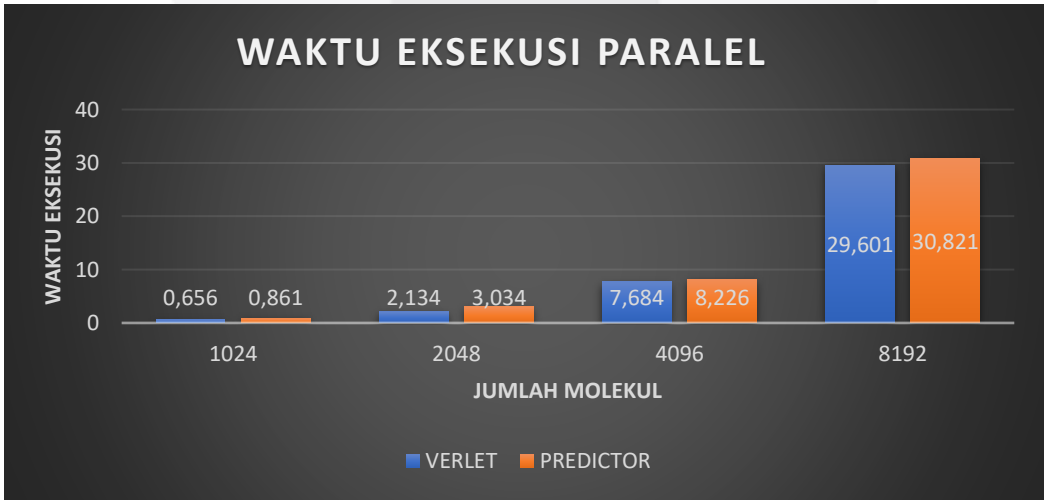
Jumlah Partikel	Waktu Eksekusi Verlet Serial	Waktu Eksekusi Predictor Serial	Selisih Waktu
1024	1.398 s	1.811 s	0.413 s
2048	5.513 s	7.967 s	2.184 s
4096	21.814 s	23.370 s	1.556 s
8192	86.680 s	90.243 s	3.563 s

Tabel 8 Selisih waktu algoritma Velocity Verlet dan Predictor Corrector secara paralel.

Jumlah Partikel	Waktu Eksekusi Verlet Paralel	Waktu Eksekusi Predictor Paralel	Selisih Waktu
1024	0.656 s	0.861 s	0.205 s
2048	2.134 s	3.034 s	0.900 s
4096	7.684 s	8.226 s	0.542 s
8192	29.601 s	30.821 s	1.220 s



Gambar 6 Perbandingan waktu eksekusi Velocity Verlet dan Predictor Corrector secara serial.



Gambar 7 Perbandingan waktu eksekusi Velocity Verlet dan Predictor Corrector secara paralel.

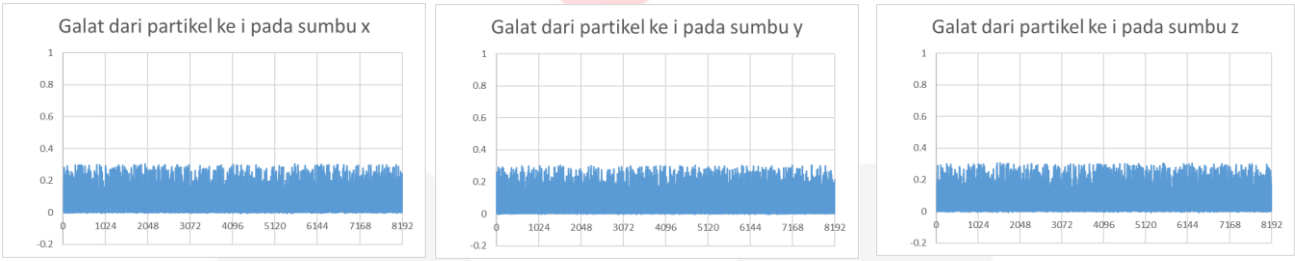
Algoritma Velocity Verlet berjalan lebih cepat dibandingkan algoritma Predictor-Corrector baik dijalankan secara serial maupun paralel.

Tabel 8 Nilai RMSE algoritma *Velocity Verlet*.

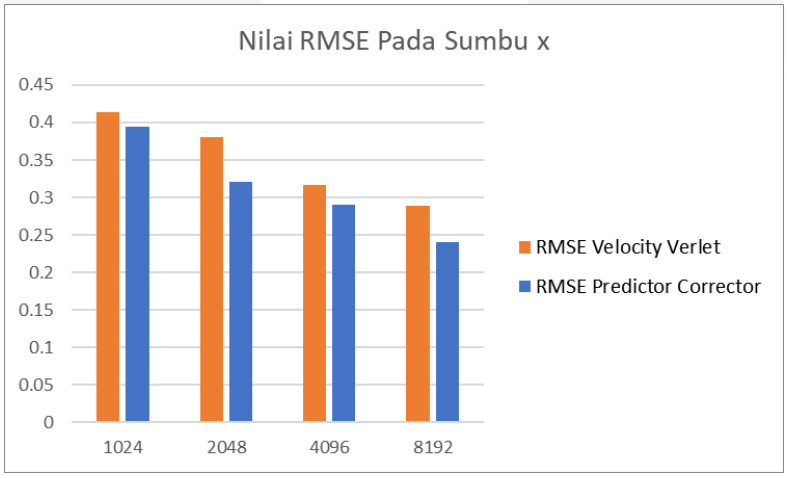
Jumlah Partikel	Nilai RMSE (x)	Nilai RMSE (y)	Nilai RMSE (z)
1024	0.4132	0.4176	0.4225
2048	0.3802	0.3880	0.3932
4096	0.3171	0.3207	0.3274
8192	0.2889	0.2885	0.2963

Tabel 9 Nilai RMSE algoritma *Predictor Corrector*.

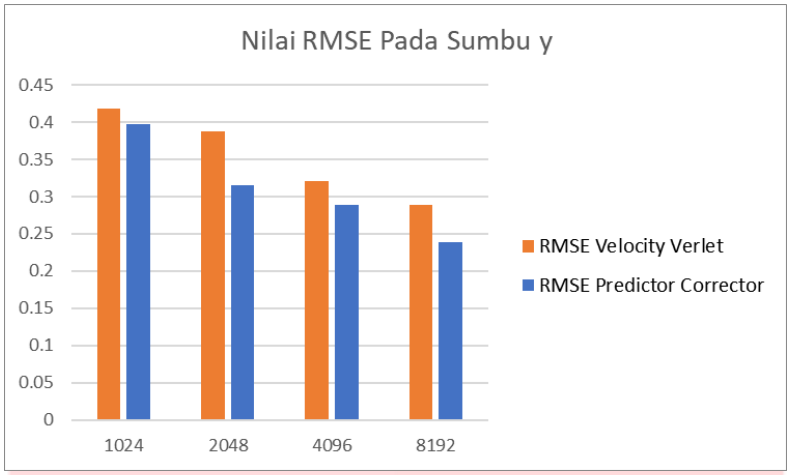
Jumlah Partikel	Nilai RMSE (x)	Nilai RMSE (y)	Nilai RMSE (z)
1024	0.3950	0.3977	0.3901
2048	0.3209	0.3154	0.3243
4096	0.2902	0.2894	0.2975
8192	0.2401	0.2391	0.2506



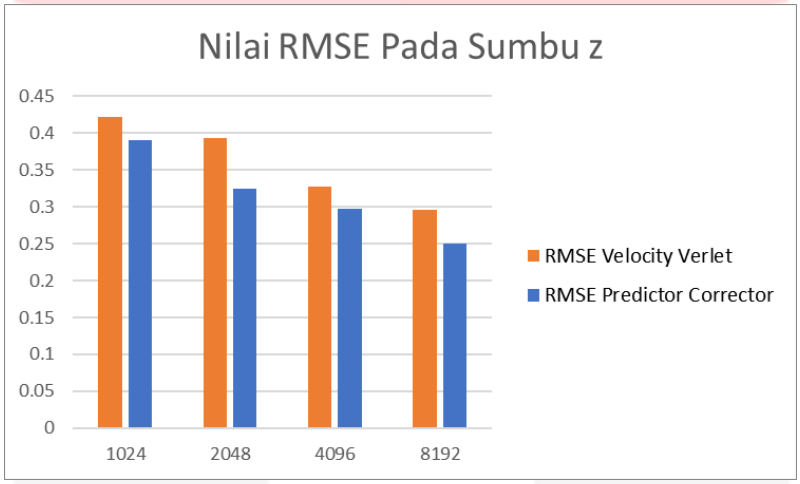
Gambar 8 Grafik galat partikel pada sumbu x, y, dan z dengan 8192 partikel.



Gambar 8 Perbandingan nilai RMSE kedua algoritma pada sumbu x.



Gambar 9 Perbandingan nilai RMSE kedua algoritma pada sumbu y.



Gambar 10 Perbandingan nilai RMSE kedua algoritma pada sumbu z.

Perhitungan error pada algoritma menggunakan RMSE dengan membandingkan nilai r yang dihasilkan oleh masing-masing program yang dijalankan serial dan paralel. Perhitungan error dihitung dari nilai maksimum error posisi yang dihasilkan per time-step dari 1024 partikel sampai 8192 partikel. Nilai yang dihasilkan lalu ditampilkan dalam bentuk grafik agar dapat diamati perbandingannya.

4.2 Analisis Hasil Pengujian

Algoritma Velocity Verlet dan Predictor-Corrector berjalan lebih cepat saat dijalankan secara paralel menggunakan CUDA. Terlihat dari grafik bahwa speedup yang dihasilkan rata-rata 2x lebih cepat ketika melakukan simulasi secara paralel dengan CUDA. Dan terlihat dari grafik bahwa semakin banyak jumlah partikel yang disimulasikan, semakin cepat juga speedup yang dihasilkan oleh program yang dijalankan secara paralel di CUDA. Hal ini disebabkan oleh hasil paralelisasi CUDA yang memanfaatkan jumlah core pada GPU sehingga CUDA dapat dengan cepat melakukan simulasi dalam jumlah partikel yang banyak berkat jumlah core dalam GPU tersebut. Ini membuktikan bahwa CUDA core bekerja dengan baik dalam mengatasi simulasi dengan jumlah banyak dan dapat menghemat waktu dibandingkan dengan melakukan eksekusi secara serial di CPU yang mempunyai jumlah core lebih sedikit dibandingkan dengan CUDA.

Algoritma Velocity Verlet berjalan lebih cepat dibandingkan algoritma Predictor-Corrector ketika dijalankan secara serial maupun paralel, terlihat dari grafik selisih waktu eksekusi kedua algoritma. Hal ini disebabkan oleh Predictor-Corrector mempunyai beban matematis yang lebih banyak dibandingkan metode Velocity Verlet.

Perhitungan error pada algoritma menggunakan RMSE dengan membandingkan nilai r pada algoritma serial dan paralel dengan mengasumsikan algoritma yang berjalan secara serial sebagai acuan. Hal ini dikarenakan

algoritma yang berjalan secara serial telah dibandingkan dengan solusi analitiknya dengan hasil error yang rendah. Hasil error menampilkan bahwa nilai error untuk jumlah molekul pada 1024 sampai 8192 berada dibawah angka 1, dan semakin banyak jumlah partikel, semakin kecil jumlah error yang dihasilkan karena semakin banyak data yang dibandingkan. Algoritma Predictor Corrector mempunyai nilai error yang lebih rendah dibandingkan algoritma Velocity Verlet. Hal ini membuktikan walaupun algoritma Predictor Corrector berjalan sedikit lebih lama dibandingkan algoritma Velocity Verlet, algoritma Predictor-Corrector memiliki akurasi yang lebih baik.

5. Kesimpulan

Algoritma Velocity Verlet dan Predictor-Corrector keduanya berjalan lebih cepat ketika dijalankan secara paralel pada GPU dibandingkan secara serial dengan CPU. Hal ini membuktikan bahwa paralelisasi pada simulasi dinamika molekuler menggunakan CUDA dapat mempercepat waktu simulasi. Grafik menunjukan bahwa semakin banyak jumlah partikel yang disimulasikan, semakin cepat speedup yang dihasilkan oleh CUDA. Hal ini disebabkan oleh CUDA core pada GPU yang memparalelkan simulasi pada banyak molekul tersebut sehingga waktu simulasi yang dihasilkan lebih cepat dibandingkan menjalankan program secara serial pada CPU.

Algoritma Velocity Verlet memiliki waktu eksekusi yang lebih cepat dibanding algoritma Predictor-Corrector, baik ketika dijalankan secara serial maupun paralel. Hal ini membuktikan beban matematis pada algoritma Predictor-Corrector lebih berat dibandingkan Velocity Verlet.

Nilai error dihitung menggunakan RMSE. Rata-rata nilai error kedua algoritma adalah dibawah angka 1 dan semakin banyak jumlah molekul, semakin kecil nilai error yang dihasilkan karena semakin banyak molekul yang dibandingkan sehingga memperkecil nilai error yang dihasilkan. Algoritma Predictor-Corrector merupakan algoritma yang lebih baik dibandingkan algoritma Velocity Verlet pada penelitian dinamika molekuler ini karena error yang dihasilkan oleh metode Predictor-Corrector lebih rendah dibandingkan dengan metode Velocity Verlet, dapat disimpulkan bahwa walaupun membutuhkan waktu eksekusi sedikit lebih lama, algoritma Predictor-Corrector menghasilkan nilai yang lebih akurat.

Daftar Pustaka

[1] Berendsen HJC, David VDS & Rudi VD. (1995). GROMACS: A Message Passing Parallel Molecular Dynamics Implementation. ComputPhys Comm.

[2] Anastasia Dwi Astuti, Prof. Dr.rer.nat. A Benny Mutiara. SIMULASI DINAMIKA MOLEKULER PROTEIN DENGAN APLIKASI GROMACS

[3] Verlet, Loup (1967). "Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard−Jones Molecules"

[4] Holm, Christian (2013). Simulation Methods in Physics 1

[5] NVIDIA® CUDA™ Architecture Introduction & Overview

[6] Inam, R. (n.d.). An Introduction to GPGPU Programming – CUDA Architecture.

[7] Zunitch, Peter (2018). CUDA vs OpenGL vs OpenCL.

[8] Dipojono, H.K. (2001), Simulasi Dinamika Molekul, Laboratorium Komputasi Material, Departemen Teknik Fisika, ITB

[9] Butcher, John C. (2003), Numerical Methods for Ordinary Differential Equations

[10] CUDA Toolkit Documentation v10.1.243

[11] Zhigilei, Leonid. University of Virginia, MSE 4270/6270: Introduction to Atomistic Simulations

[12] Saurabh Gupta and Pritish Kumar Varadwaj. (2012). A BRIEF OVERVIEW ON MOLECULAR DYNAMICS SIMULATION OF BIOMOLECULAR SYSTEM: PROCEDURE, ALGORITHMS AND APPLICATIONS

[13] Kevin M. Tenny, Jeffrey S. Cooper. (2020). Ideal Gas Behavior