

JARINGAN DAN PENGOLAHAN DATA PARALEL

LAPORAN PRAKTIKUM

Mohammad Rizka Fadhli
Ikang
20921004@mahasiswa.itb.ac.id

31 December 2021

Contents

1	INTRODUCTION	6
1.1	Definisi	6
1.2	Perbedaan <i>Serial Processing</i> dan <i>Parallel Processing</i>	6
1.3	Cara Kerja <i>Parallel Processing</i>	8
1.3.1	SISD	8
1.3.2	SIMD	9
1.3.3	MISD	9
1.3.4	MIMD	9
1.4	Tugas Praktikum	9
1.5	<i>Server</i> yang Digunakan	10
1.6	MPI	13
2	METHOD	14
2.1	Penjelasan Singkat	15
2.1.1	<i>Broadcast</i>	15
2.1.2	<i>Scatter</i>	15
2.1.3	<i>Gather</i>	15
2.1.4	<i>Point to Point</i>	15
2.2	Metode Integral Numerik	15
2.2.1	Metode Titik Tengah (<i>Midpoint</i>)	16
2.2.2	Simulasi Monte Carlo	16
2.3	Menghitung π	17
2.4	Penjumlahan dan Perkalian Matriks	17

3	<i>RESULT AND DISCUSSION</i>	17
3.1	Soal I	17
	Jawab	17
3.2	Soal II	21
	Jawab	21
3.3	Soal III	25
	Jawab	25
4	<i>CONCLUSION</i>	29
	<i>REFERENCES</i>	29

List of Figures

1	Ilustrasi Perbedaan Serial dan Parallel Processing	7
2	Ilustrasi SISD	8
3	Ilustrasi SIMD	9
4	Spesifikasi Server yang Digunakan	11
5	Tampilan Awal Setelah Login ssh	12
6	lscpu dari Server	12
7	htop dari Server	13
8	Versi MPI yang Digunakan	14
9	Flowchart Simulasi Monte Carlo untuk Integral Numerik	16
10	Rekap Runtime Soal I	20
11	Grafik 1/4 Lingkaran dengan $r=1$	21
12	Rekap Runtime Soal II	25
13	Rekap Runtime Soal III	28

List of Tables

1	Rekap Runtime dan Hasil Integral Soal I	19
2	Rekap Runtime dan Hasil Integral Soal II	24
3	Rekap Runtime dan Hasil Integral Soal III	26

1 INTRODUCTION

1.1 Definisi

Parallel processing adalah metode komputasi untuk menggunakan dua atau lebih *processors* untuk menjalankan beberapa tugas secara terpisah atau secara keseluruhan. Setiap komputer yang memiliki lebih dari satu *CPUs* atau memiliki *processor multi cores* bisa melakukan *parallel processing*.¹

1.2 Perbedaan *Serial Processing* dan *Parallel Processing*

Perbedaan mendasar dari *serial processing* dan *parallel processing* adalah dari segi bagaimana komputer melakukan proses komputasi. *Serial processing* berarti komputer melakukan tugasnya secara sekuensial (berurutan) menggunakan satu *processor*. Akibatnya adalah saat melakukan suatu proses yang kompleks, *runtime* yang diperlukan lebih lama karena *processor* harus memproses data satu-persatu.

Berbeda halnya dengan *parallel processing*. Tugas yang dilakukan komputer didistribusikan kepada sejumlah *processors* untuk diolah secara bersamaan. Konsekuensinya adalah *runtime* komputasi lebih singkat. Namun perlu diperhatikan dengan seksama bahwa tidak semua tugas bisa kita buat paralelisasinya dan cara kita menulis algoritma atau *coding* harus disesuaikan.

Kenapa tidak semua tugas bisa diparalelisasi?

Beberapa tugas sekuensial yang tidak bisa dihindari tidak bisa diparalelisasi.

Sebagai contoh:

1. *Looping* yang prosesnya tidak saling bergantung bisa diparalelisasi. Misalkan ada suatu fungsi untuk menghitung suatu *array* bisa diparalelisasi dengan cara memecah *array* tersebut untuk diproses bersamaan di beberapa *processors*.

¹<https://searchdatacenter.techtarget.com/definition/parallel-processing>

2. *Looping* yang prosesnya saling bergantung tidak bisa diparalelisasi. Misalkan suatu *looping* ke i nilainya bergantung pada proses *looping* ke $i - 1$.

Berikut adalah ilustrasi perbedaan serial dan *parallel processing*:

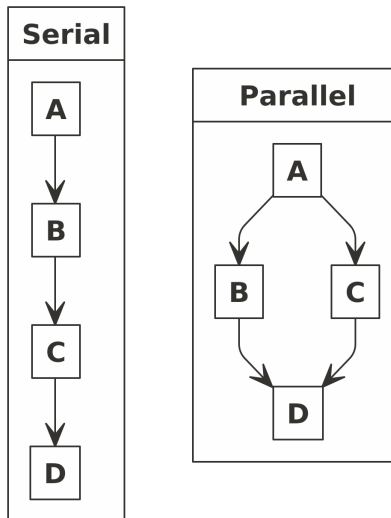


Figure 1: Ilustrasi Perbedaan Serial dan Parallel Processing

1.3 Cara Kerja *Parallel Processing*

Untuk melakukan *parallel processing*, dibutuhkan *hardware* dan *software* yang mendukung hal tersebut. Secara *hardware* dibutuhkan komputer dengan *multiple cores processors* atau dibutuhkan beberapa komputer yang digabung menjadi satu kesatuan. Secara *software* dibutuhkan tidak hanya `Python` tapi juga *middleware* bernama `Open MPI`. Bagian *hardware* dan *software* ini akan dibahas pada bagian selanjutnya.

Pada sistem *parallel processing* terdiri dari beberapa unit *processors* dan beberapa unit *memory*. Ada dua teknik berbeda yang digunakan untuk mengakses data di unit *memory*, yaitu: *shared memory address* dan *message passing*.

Berdasarkan cara mengorganisasikan memori ini komputer bisa dibedakan menjadi *shared memory parallel machine* dan *distributed memory parallel machine*.

Ada empat model komputasi yang dikenal dalam taksonomi Flynn, yaitu:

1. **SISD** (*Single Instruction, Single Data*)
2. **SIMD** (*Single Instruction, Multiple Data*)
3. **MISD** (*Multiple Instruction, Single Data*)
4. **MIMD** (*Multiple Instruction, Multiple Data*)

1.3.1 SISD

Komputer ini adalah tipikal komputer konvensional yang hanya memiliki satu *processor* dan satu instruksi yang dieksekusi secara serial. Komputer jenis ini tidak bisa melakukan *parallel processing*.

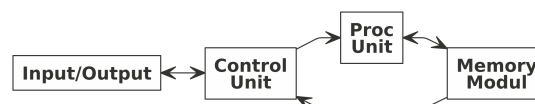


Figure 2: Ilustrasi SISD

1.3.2 SIMD

Komputer ini memiliki lebih dari satu *processor* tapi hanya mengeksekusi satu instruksi secara paralel pada data yang berbeda pada level *lock-step*. Contohnya adalah komputer vektor.

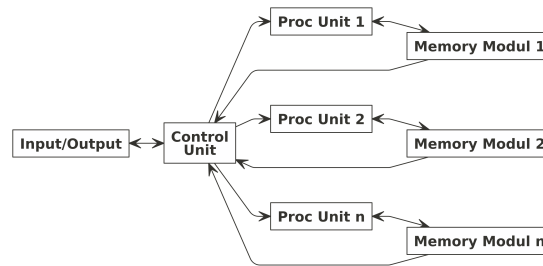


Figure 3: Ilustrasi SIMD

1.3.3 MISD

Komputer jenis ini belum diciptakan karena secara arsitekturnya tidak mudah dipahami. Secara teori komputer ini memiliki satu *processor* dan mengeksekusi beberapa instruksi secara paralel.

1.3.4 MIMD

Komputer berarsitektur ini paling banyak digunakan untuk membangun *super computer*. Komputer ini memiliki lebih dari satu *processors* dan mengeksekusi lebih dari satu instruksi secara paralel.

1.4 Tugas Praktikum

Pada praktikum ini, saya akan mengerjakan:

- Dua buah tugas terkait penyelesaian integral secara numerik memanfaatkan metode dikritisasi nilai tengah (*midpoint*) dan simulasi Monte Carlo.

- Satu buah tugas terkait penjumlahan dan perkalian matriks $n \times n$.

Ketiga tugas tersebut akan diselesaikan menggunakan serial dan *parallel processing*.

1.5 Server yang Digunakan

Pada praktikum kali ini, saya tidak bisa menggunakan *server HPC* yang disediakan oleh ITB karena masalah koneksi. Oleh karena itu, saya menggunakan *server* lain agar bisa menduplikasi apa yang seharusnya dikerjakan di *server* ITB.

Saya menggunakan *server virtual machine* milik *Google Cloud*². *Server* ini memiliki *processor Intel Xeon 8 cores*. *Hostname* dari *server* ini saya beri nama **praktikum**.

Server ini disewa menggunakan *free credit* yang kita dapatkan saat mengaktifkan layanan *Google Cloud* menggunakan akun *Google*.

²<https://ikanx101.com/blog/vm-cloud/>

Berikut adalah spesifikasinya:

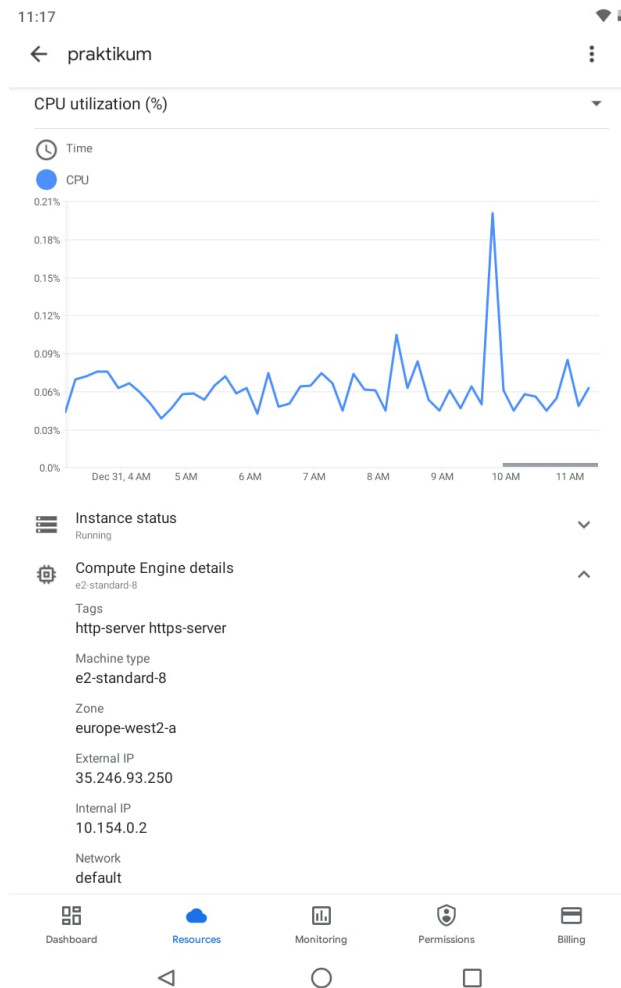


Figure 4: Spesifikasi Server yang Digunakan

Server ini bisa diakses menggunakan *command line* menggunakan `ssh` langsung ke *IP Public* yang diberikan *Google*.

```
ix@localhost:~/209_ITB/Semester I/Jaringan dan Pengolahan Data Paralel/praktikum$ ssh 35.246.93
250
Enter passphrase for key '/home/ix/.ssh/id_rsa':
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.11.0-1023-gcp x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Fri Dec 31 03:38:00 UTC 2021

System load: 0.16          Processes:            158
Usage of /:  48.4% of 9.52GB Users logged in:          0
Memory usage: 1%          IPv4 address for ens4: 10.154.0.2
Swap usage:  0%

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

   https://ubuntu.com/blog/microk8s-memory-optimisation

0 updates can be applied immediately.

Last login: Fri Dec 31 02:40:18 2021 from 111.94.196.248
ix@praktikum:~$
```

Figure 5: Tampilan Awal Setelah Login ssh

Server ini berjalan di *operating system* Ubuntu Linux 20.04 LTS.

Berikut adalah tampilan hasil `lscpu`:

```
Last login: Fri Dec 31 02:40:18 2021 from 111.94.196.248
ix@praktikum:~$ lscpu
Architecture:                x86_64
CPU op-mode(s):              32-bit, 64-bit
Byte Order:                  Little Endian
Address sizes:                46 bits physical, 48 bits virtual
CPU(s):                      8
On-line CPU(s) list:         0-7
Thread(s) per core:          2
Core(s) per socket:          4
Socket(s):                   1
NUMA node(s):                1
Vendor ID:                   GenuineIntel
CPU family:                   6
Model:                        79
Model name:                   Intel(R) Xeon(R) CPU @ 2.20GHz
Stepping:                     0
CPU MHz:                      2200.202
BogoMIPS:                     4400.40
Hypervisor vendor:           KVM
Virtualization type:         full
L1d cache:                   128 KiB
L1i cache:                   128 KiB
L2 cache:                     1 MiB
L3 cache:                     55 MiB
NUMA node0 CPU(s):           0-7
Vulnerability Itlb multihit:  Not affected
Vulnerability L1tf:           Mitigation; PTE Inversion
Vulnerability Mds:            Mitigation; Clear CPU buffers; SMT Host state unknown
Vulnerability Meltdown:       Mitigation; PTI
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1:     Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2:     Mitigation; Full generic retpoline, IBPB conditional, IBRS_FW, STIBP conditional, RSB filling
Vulnerability Srbds:          Not affected
Vulnerability Tsx async abort: Mitigation; Clear CPU buffers; SMT Host state unknown
```

Figure 6: lscpu dari Server

Berikut adalah tampilan hasil `htop`:

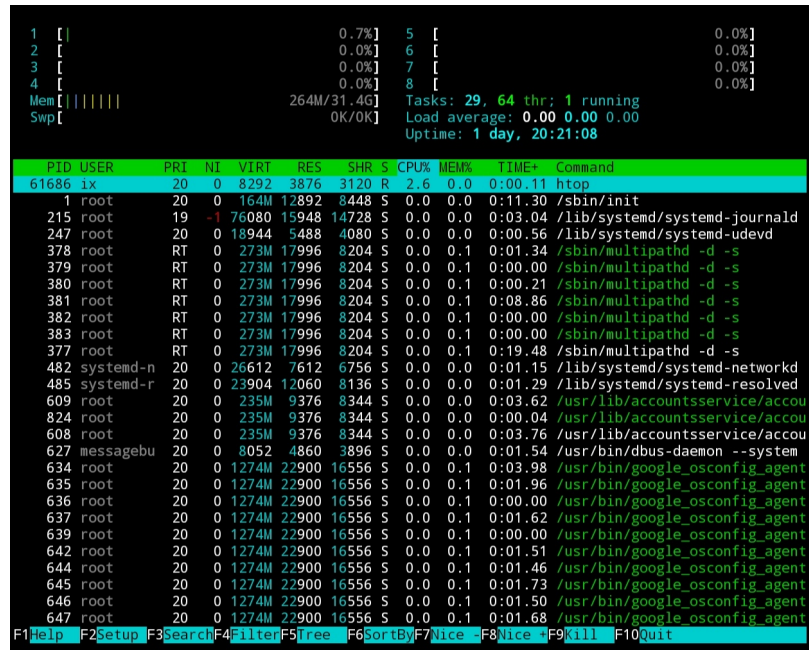


Figure 7: `htop` dari Server

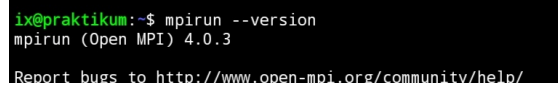
1.6 MPI

Secara *default* Python sudah ter-*install* di *server* namun tidak untuk Open MPI. Oleh karena itu, salah satu langkah yang perlu dilakukan adalah meng-*install*-nya terlebih dahulu (Jarno Rantaharju and Bennet 2019).

Pada **Ubuntu**, proses instalasinya bisa kita lakukan dengan mengetikkan perintah berikut ini di *command line*:

```
sudo apt install openmpi-bin openmpi-dev openmpi-common openmpi-doc libopenmpi-dev
```

Setelah proses instalasi selesai, kita bisa mengecek versi Open MPI yang berjalan di *server* sebagai berikut:



```
ix@praktikum:~$ mpirun --version
mpirun (Open MPI) 4.0.3
Report bugs to http://www.open-mpi.org/community/help/
```

Figure 8: Versi MPI yang Digunakan

Pada Python, saya perlu meng-*install library* `mpi4py` dengan perintah:

```
sudo apt install python3-mpi4py
```

Oleh karena saya menjalankan program Python praktikum di *server* sendiri (tanpa ada *user* lain), maka saya tidak melakukan instalasi SLURM.

2 METHOD

Pada praktikum ini, kita akan melakukan *parallel processing* menggunakan Python versi 3.8.10 di *server* berbasis Linux Ubuntu OS. Ada beberapa metode *parallel processing* yang hendak dilakukan, yakni:

1. *Broadcast*,
 - *Broadcast-gather*,
 - *Broadcast-reduce*,
2. *Scatter*,
 - *Scatter-reduce*,
 - *Scatter-gather*,
3. *Gather*,
4. *Reduce*,
5. *Multi-processing*,
6. *Multi-thread*,

7. *Point to point.*

Kemudian semua metode *parallel processing* ini akan dibandingkan *runtime*-nya dengan *serial processing*.

2.1 Penjelasan Singkat

2.1.1 *Broadcast*

Merupakan metode komunikasi kolektif di mana satu proses mengirim data yang sama ke proses lainnya.

2.1.2 *Scatter*

Secara fungsi, *broadcast* dan *scatter* hanya memiliki perbedaan pada perintah yang digunakan. Selain itu *scatter* dapat mengirim potongan data dalam *array* ke proses yang berbeda. Perlu diperhatikan bahwa jumlah *tasks* yang dibagikan tidak boleh melebihi jumlah *processors*.

2.1.3 *Gather*

Fungsi ini melakukan kebalikan dari *scatter*, yakni dengan mengumpulkan semua data yang diterima.

2.1.4 *Point to Point*

Operasi *point to point* (**P2P**) terdiri dari pertukaran pesan antara dua proses. Setiap operasi pengiriman akan disinkronkan secara sempurna dengan operasi penerimaan (*send* - *receive*).

2.2 Metode Integral Numerik

Ada beberapa metode numerik yang bisa digunakan untuk menghitung suatu integral dari fungsi kontinu. Pada praktikum ini, saya akan menggunakan metode titik tengah dan simulasi Monte Carlo untuk mengerjakan dua soal integral numerik.

2.2.1 Metode Titik Tengah (*Midpoint*)

Metode titik tengah merupakan salah satu cara perhitungan integral numerik dari fungsi kontinu melalui dikritisasi fungsi (MathLibretexts 2021). Prinsip yang digunakan adalah penjumlahan deret **Riemann**.

2.2.1.1 Definisi Misalkan $f(x)$ kontinu di selang $[a, b]$. Jika diambil suatu n bilangan bulat positif, kita bisa membagi selang tersebut menjadi partisi-partisi kecil berikut: $\Delta x = \frac{b-a}{n}$.

Tuliskan $m_i = x_i + \frac{\Delta}{2}$ sebagai titik tengah di partisi ke- i , maka: $\int_a^b f(x)dx$ bisa didekati dengan $n \sum_{i=1}^n f(m_i)\Delta x_i$.

2.2.2 Simulasi Monte Carlo

Berbeda dengan metode titik tengah, ide dasar dari simulasi Monte Carlo adalah dengan melakukan *generating* sejumlah titik secara *random* pada selang integral dan menghitung nilai fungsinya³. Berikut adalah *flowchart*-nya:

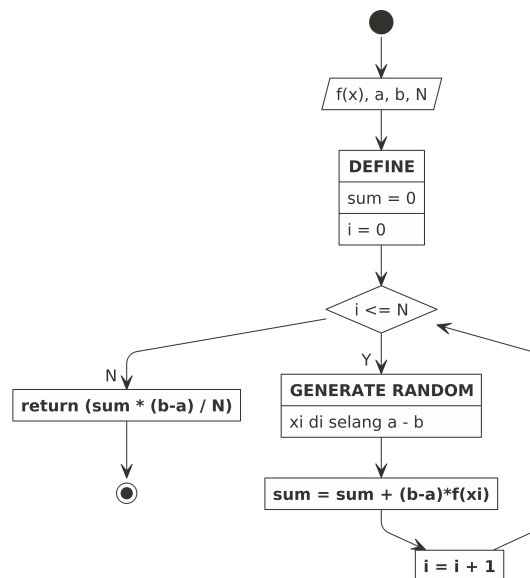


Figure 9: Flowchart Simulasi Monte Carlo untuk Integral Numerik

³<https://ikanx101.com/blog/integral-tentu/>

2.3 Menghitung π

Untuk menghitung nilai π pada soal kedua, saya akan menggunakan fungsi berikut:

$$\int_0^1 4\sqrt{1-x^2}$$

2.4 Penjumlahan dan Perkalian Matriks

Pada soal ketiga, saya menggunakan matriks berukuran $n \times n$ dengan $n = numCores^2 = 8^2 = 64$. Dengan isi dari matriks A dan B di-generate secara *random* melalui `numpy`.

3 *RESULT AND DISCUSSION*

Pada tugas praktikum ini, kita akan membandingkan *runtime* antara berbagai metode *parallel processing* dengan *serial processing*. Perlu saya ingatkan kembali bahwa *server Google Cloud* yang saya pakai memiliki **8 cores**.

3.1 Soal I

Hitung integral $\int_0^1 x^2 dx$ dengan $n = 10^8$.

Jawab

Secara analitik, kita bisa hitung bahwa:

$$\int_0^1 x^2 dx = \frac{x^3}{3} \Big|_0^1 = \frac{1}{3} \simeq 0.333333..$$

Berikut adalah program *serial processing* yang dibuat:

```
import time
```

```
# hitung waktu mulai
mulai = time.time()
# bikin fungsi f(x)
def fx (x) :
    return(x**2)
# bikin fungsi untuk integral numerik
def int_numeric (a,b,n):
    # set terlebih dahulu agar tidak error pada tipe variabel
    n = int(n)
    a = float(a)
    b = float(b)
    sum = float(0)
    # hitung selang integrasi diskritisasi
    h = (b-a) / n
    # mulai iterasi untuk menghitung penjumlahan
    for i in range(n):
        xi = a + h/2 + i*h
        sum = sum + fx(xi)
    # kalikan dengan h untuk menjadi full integral
    sum = h * sum
    return(sum)
# initial condition
a = 0
b = 1
n = 10**8
# hitung soal
nilai = int_numeric(a,b,n)
print("Nilai integral f(x) dx adalah: ",nilai)
# hitung waktu selesai
end = time.time()
```

```
waktu = end - mulai
print(waktu)
# selesai
print("DONE")
```

Hasil *runtime* program di atas adalah 27.43908143043518 detik.

Berikut adalah grafik rekap *runtime* dari semua *parallel processing* dan perbandingannya:

Table 1: Rekap Runtime dan Hasil Integral Soal I

Metode	Nilai	Runtime
Serial	0.33333	27.439
Broadcast	0.33333	56.212
Broadcast Gather	0.33333	6.989
Broadcast Reduce	0.33333	7.089
Gather	0.33333	7.081
Multi Processing	0.33333	7.526
Multi Threading	0.33333	254.073
P2P	0.33333	7.182
Reduce	0.33333	7.159
Scatter	0.33333	54.735
Scatter Gather	0.33333	7.059
Scatter Reduce	0.33333	6.877

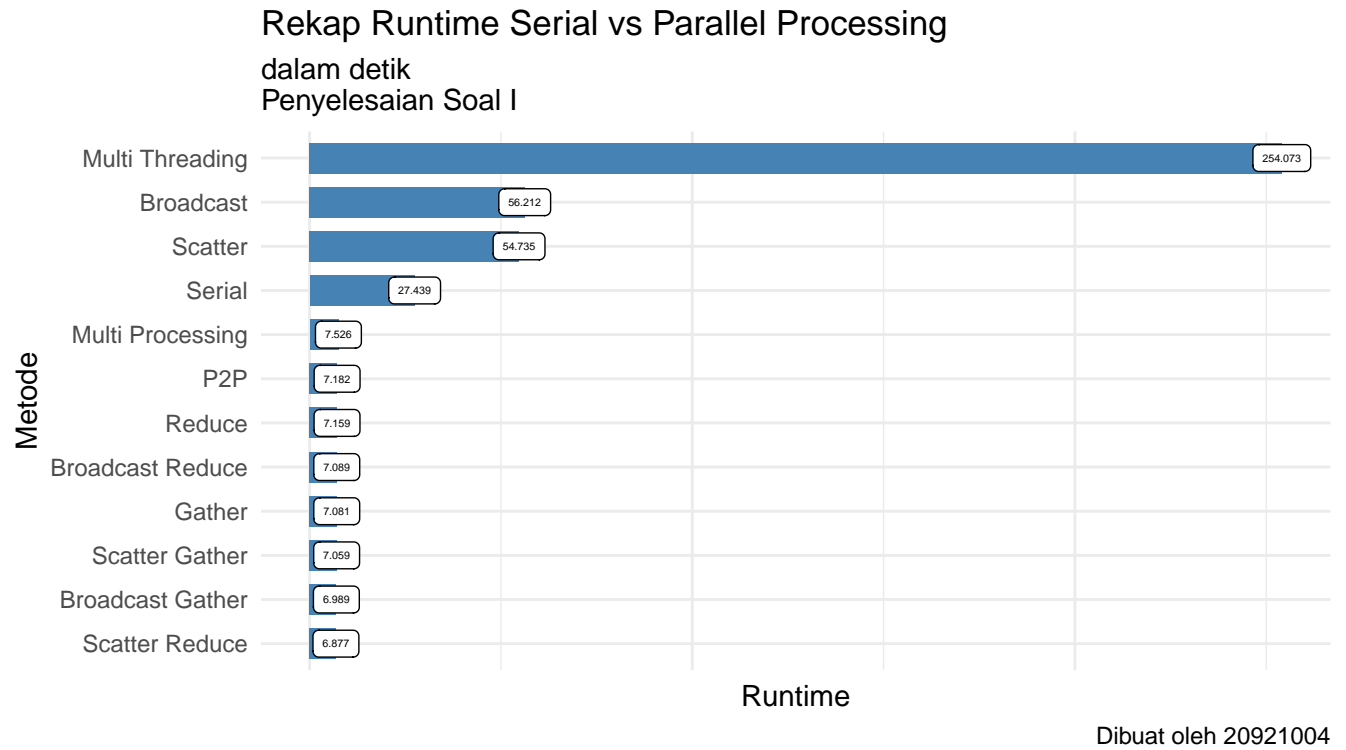


Figure 10: Rekap Runtime Soal I

3.2 Soal II

Hitung nilai π dengan menyelesaikan: $\int_0^1 4\sqrt{1-x^2}$ dengan $n = 10^8$.

Jawab

Perhatikan grafik $f(x) = \sqrt{1-x^2}, x \in [0, 1]$ berikut ini:

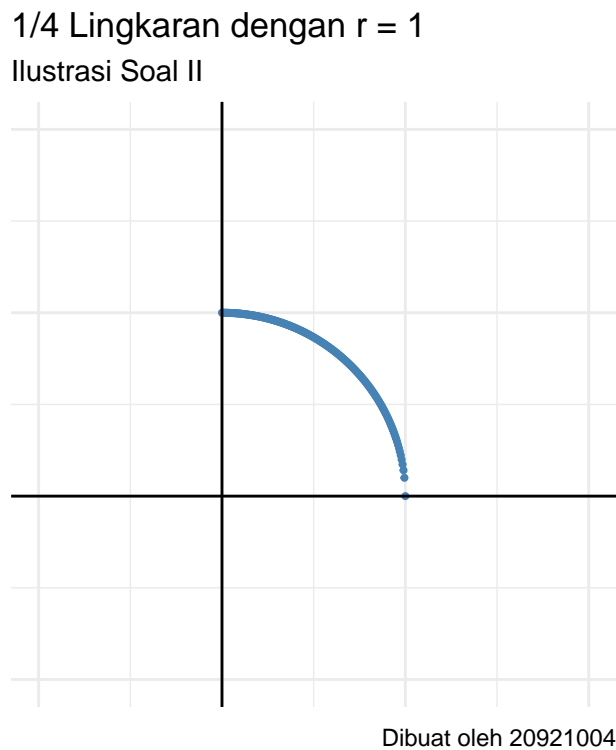


Figure 11: Grafik 1/4 Lingkaran dengan r=1

Luas area di bawah kurva dikalikan dengan 4 akan menghasilkan hampiran nilai dari π . Kali ini saya menghitung hampiran tersebut dengan metode *midpoint* dan simulasi Monte Carlo.

Berikut adalah program serial yang dibuat di Python:

```
import time
import random
import math
# bikin fungsi f(x)
```

```
def fx (x) :  
    return(math.sqrt(1 - x**2) * 4)  
# bikin fungsi untuk integral numerik  
def int_numeric (a,b,n):  
    # set terlebih dahulu agar tidak error pada tipe variabel  
    n = int(n)  
    a = float(a)  
    b = float(b)  
    sum = float(0)  
    # hitung selang integrasi diskritisasi  
    h = (b-a) / n  
    # mulai iterasi untuk menghitung penjumlahan  
    for i in range(n):  
        xi = a + h/2 + i*h  
        sum = sum + fx(xi)  
    # kalikan dengan h untuk menjadi full integral  
    sum = h * sum  
    return(sum)  
# bikin fungsi untuk monte carlo  
def monte_pi (n):  
    n = int(n)  
    monte = float(0)  
    for i in range(n):  
        xi = random.random()  
        yi = fx(xi)  
        monte = monte + yi  
    return(monte/n)  
  
# initial condition  
a = 0
```

```
b = 1
n = 10**8

# hitung soal dengan midpoint
mulai = time.time()
nilai_1 = int_numeric(a,b,n)
print("Nilai integral f(x) dx adalah: ",nilai_1)
end = time.time()
waktu_1 = end - mulai
print("Waktu yang dibutuhkan dengan midpoint: ",waktu_1)

# hitung soal dengan montecarlo
mulai = time.time()
nilai_2 = monte_pi(n)
print("Nilai integral f(x) dx adalah: ",nilai_2)
end = time.time()
waktu_2 = end - mulai
print("Waktu yang dibutuhkan dengan monte: ",waktu_2)
```

Dari program serial ini, didapatkan *runtime* sebagai berikut:

1. Menggunakan metode *midpoint* selama 37.09915375709534 detik dengan hampiran $\pi \simeq 3.1415926535908407$.
2. Menggunakan simulasi Monte Carlo selama 34.31454539299011 detik dengan hampiran $\pi \simeq 3.141628338857705$.

Berikut adalah grafik rekapan dari semua perhitungan serial dan paralel untuk metode *midpoint* dan simulasi Monte Carlo:

Table 2: Rekap Runtime dan Hasil Integral Soal II

Metode	Algo	Nilai	Runtime
Serial	Midpoint	3.14159	37.099
Serial	Monte Carlo	3.14163	34.315
Broadcast	Midpoint	3.14159	80.767
Broadcast	Monte Carlo	3.14159	72.397
Broadcast Gather	Midpoint	3.14159	10.181
Broadcast Gather	Monte Carlo	3.14157	9.693
Broadcast Reduce	Midpoint	3.14159	10.261
Broadcast Reduce	Monte Carlo	3.14145	9.937
Gather	Midpoint	3.14159	10.230
Gather	Monte Carlo	3.14145	9.260
Multi Processing	Midpoint	3.14159	10.839
Multi Processing	Monte Carlo	3.14152	9.119
Multi Threading	Midpoint	3.14159	40.604
Multi Threading	Monte Carlo	3.14146	33.762
P2P	Midpoint	3.14987	10.336
P2P	Monte Carlo	3.14141	9.111
Reduce	Midpoint	3.14159	10.358
Reduce	Monte Carlo	3.14135	9.344
Scatter	Midpoint	3.14159	81.067
Scatter	Monte Carlo	3.14156	73.132
Scatter Gather	Midpoint	3.14159	10.334
Scatter Gather	Monte Carlo	3.14156	9.324
Scatter Reduce	Midpoint	3.14159	10.311
Scatter Reduce	Monte Carlo	3.14156	9.366

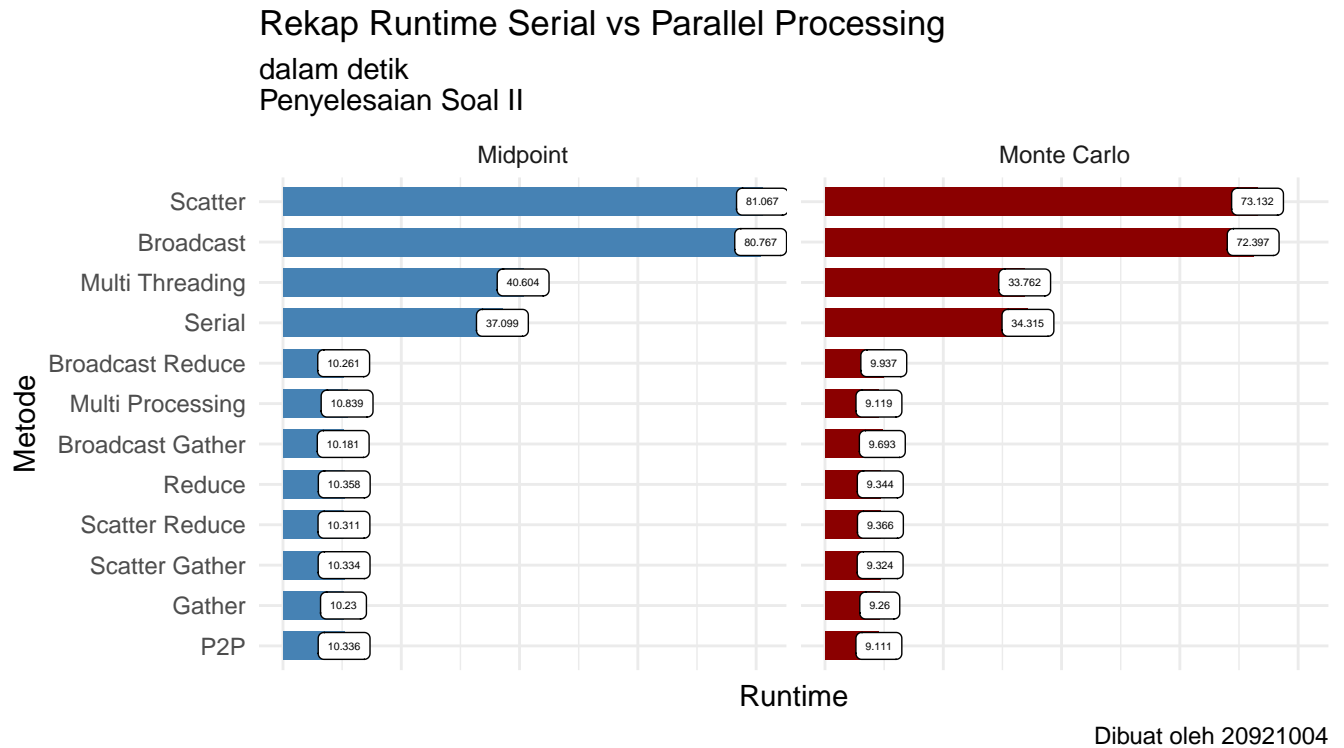


Figure 12: Rekap Runtime Soal II

3.3 Soal III

Buatlah program `Python` untuk menyelesaikan persoalan terkait dengan penjumlahan dan perkalian matriks.

Jawab

Program `Python` ini akan dibuat untuk menyelesaikan matriks 64×64 . Isi dari matriks di-generate secara *random*.

Berikut adalah programnya:

```
import numpy as np
import time
```

```

# hitung waktu mulai
mulai = time.time()

# ukuran matriks
n = 64

# penjumlahan dan perkalian
def operasi_matrix (n):
    # membuat matriks 10x10
    a = np.random.randint(n, size = (n,n))
    b = np.random.randint(n, size = (n,n))
    c = np.dot(a,b)
    d = np.add(a,b)
    return(c,d)

hitung = operasi_matrix(n)

# hitung waktu selesai
end = time.time()
waktu = end - mulai
print(waktu)

```

Untuk ukuran matriks 64×64 diperlukan *runtime* selama 0.000576019287109375 detik.

Berikut adalah grafik rekap *runtime* dari semua *parallel processing* dan perbandingannya:

Table 3: Rekap Runtime dan Hasil Integral Soal III

Metode	Runtime
Serial	0.00058
Broadcast	0.00220

Metode	Runtime
Broadcast Gather	0.01202
Broadcast Reduce	0.03328
Gather	0.01434
Multi Processing	0.02557
Multi Threading	0.01207
P2P	0.01093
Reduce	0.00561
Scatter	0.00221
Scatter Gather	0.01184
Scatter Reduce	0.01576

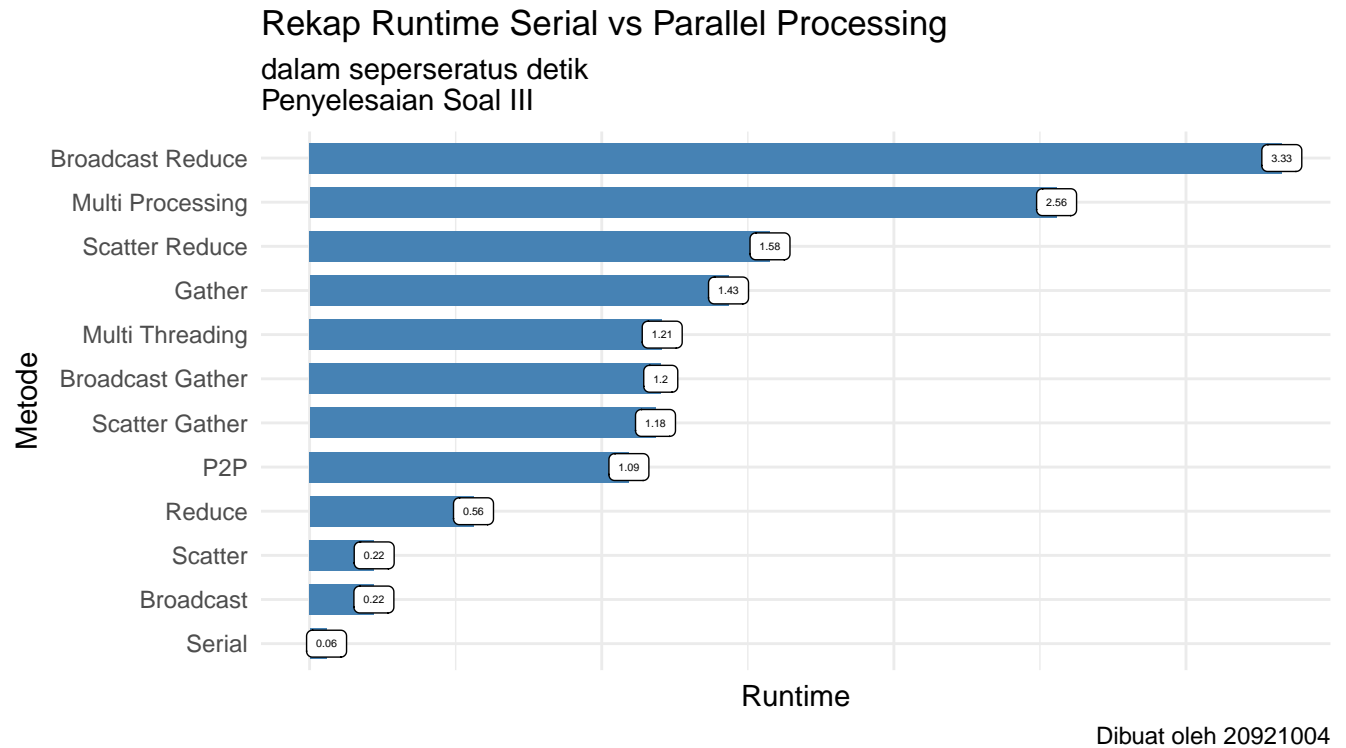


Figure 13: Rekap Runtime Soal III

4 *CONCLUSION*

Tidak semua *parallel processing* menghasilkan *runtime* yang lebih cepat dibandingkan *serial processing*. Metode seperti *scatter*, *broadcast*, dan *multi threading* menghasilkan *runtime* yang relatif lebih lama.

Metode *parallel processing* yang merupakan gabungan dari dua metode relatif menghasilkan *runtime* yang sangat cepat.

REFERENCES

- Jarno Rantaharju, Seyong Kim, and Ed Bennet. 2019. “Introduction to MPI.” <https://github.com/rantahar/introduction-to-mpi10.5281/zenodo.3274680>.
- MathLibretexts, Contributor. 2021. “Numerical Integration - Midpoint, Trapezoid, Simpson’s Rule.” <https://math.libretexts.org/@go/page/10269>.