LAPORAN PRAKTIKUM
MESSAGE PASSING INTERFACE (MPI)

Disusun Oleh :

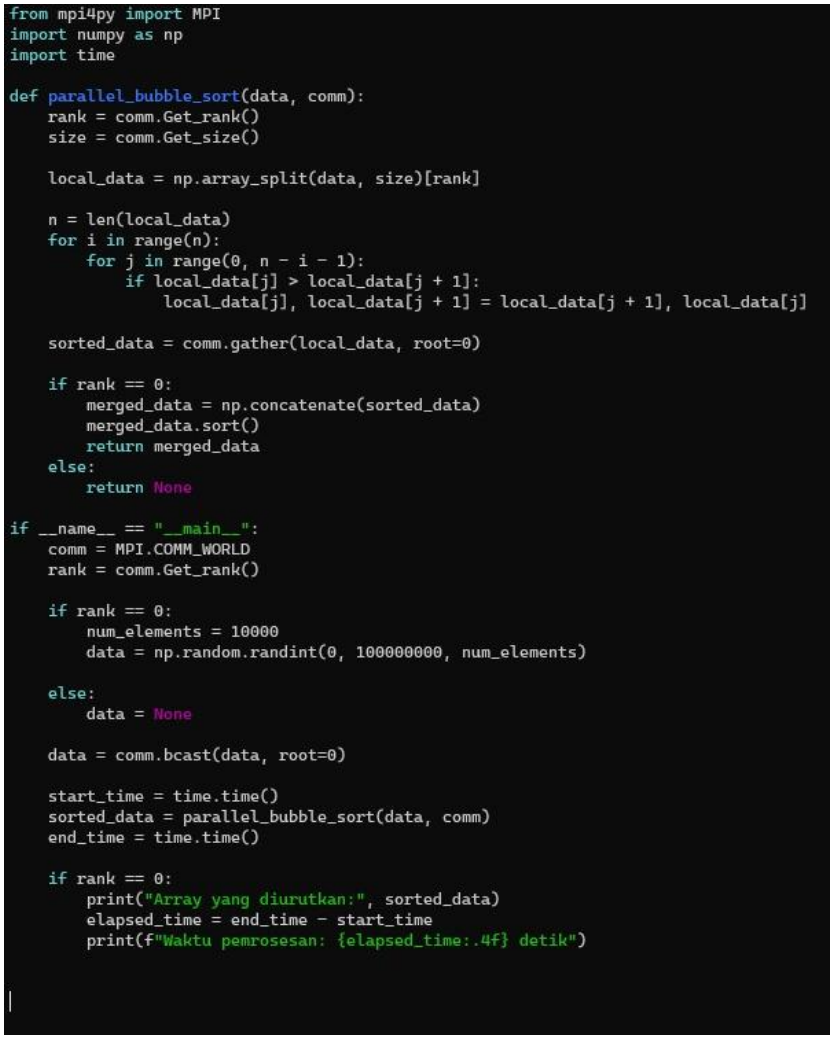
Sanvic Dicaprio (09011282227081)

Dosen Pengampu :

Adi Hermansyah, S.Kom., M.T.

Fakultas Ilmu Komputer
Jurusan Sistem Komputer
Universitas Sriwijaya Tahun 2023-2024

Setelah mempersiapkan MPI dan mencoba program bubble sort pada praktikum sebelumnya, kita akan mencoba untuk membandingkan kecepatan program yang dijalankan secara serial dan paralel menggunakan MPI. Terdapat 2 program yaitu “coding.py” yang berisi program bubble sort dan “dica.py” yang berisi program numerik untuk menghitung SPL (sistem persamaan linier) menggunakan metode eliminasi gauss. Program akan dijalankan pada 3 VM menggunakan *Host Only Adapter* agar dapat berjalan secara stabil.



```python
from mpi4py import MPI
import numpy as np
import time

def parallel_bubble_sort(data, comm):
    rank = comm.Get_rank()
    size = comm.Get_size()

    local_data = np.array_split(data, size)[rank]

    n = len(local_data)
    for i in range(n):
        for j in range(0, n - i - 1):
            if local_data[j] > local_data[j + 1]:
                local_data[j], local_data[j + 1] = local_data[j + 1], local_data[j]

    sorted_data = comm.gather(local_data, root=0)

    if rank == 0:
        merged_data = np.concatenate(sorted_data)
        merged_data.sort()
        return merged_data
    else:
        return None

if __name__ == "__main__":
    comm = MPI.COMM_WORLD
    rank = comm.Get_rank()

    if rank == 0:
        num_elements = 10000
        data = np.random.randint(0, 100000000, num_elements)

    else:
        data = None

    data = comm.bcast(data, root=0)

    start_time = time.time()
    sorted_data = parallel_bubble_sort(data, comm)
    end_time = time.time()

    if rank == 0:
        print("Array yang diurutkan:", sorted_data)
        elapsed_time = end_time - start_time
        print(f"Waktu pemrosesan: {elapsed_time:.4f} detik")
```

Isi dari program “coding.py”

```python
from mpi4py import MPI
import numpy as np
import time

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

def generate_random_matrix(rows, cols):
    return np.random.rand(rows, cols)

def main():
    ordo = 200

    baris = ordo
    kolom = ordo + 1

    A = generate_random_matrix(baris, kolom)

    start_time = time.time()

    for i in range(baris):
        diag = A[i][i]
        for j in range(kolom):
            A[i][j] /= diag
        for k in range(i + 1, baris):
            diag1 = A[k][i]
            for j in range(0, kolom):
                A[k][j] = A[k][j] - diag1 * A[i][j]

    comm.barrier()

    if rank == 0:
        print("Eliminasi Gauss:")
        for i in range(min(5, baris)):
            for j in range(kolom):
                print(f"{A[i][j]:.2f}", end=" ")
            print("...")

        x = np.zeros(ordo, dtype=float)
        for i in range(ordo - 1, -1, -1):
            x[i] = A[i][kolom - 1]
            for j in range(i + 1, ordo):
                x[i] -= A[i][j] * x[j]

        end_time = time.time()
        elapsed_time = end_time - start_time

        for i in range(min(5, ordo)):
            print(f"x{i + 1} = {x[i]:.2f}")

        print(f"Estimasi Waktu proses: {elapsed_time:.6f} detik")

if __name__ == '__main__':
    main()
```

Isi dari program “dica.py”

A. Program Pertama

Pada program pertama, kita akan menggunakan “coding.py” dengan ukuran data mulai dari 10, 100, 1.000, dan 10.000 data. Jika program berhasil, ia akan mengeluarkan output berupa angka acak yang telah disusun mulai dari angka paling kecil hingga angka paling besar dan diikuti dengan pewaktu sebagai *benchmark*. Program akan dijalankan dengan 2 kondisi perintah. Perintah pertama yaitu “python3 <nama_program>” untuk menjalankan program dengan 1 VM dan “mpiexec -n <jumlah_komputer> -host <nama_host1>, <nama_host_selanjutnya> python3 <nama_program>” untuk dijalankan secara paralel. Berikut adalah hasil *benchmark* program yang dijalankan secara paralel maupun sendiri.

```
zorin@master:~$ python3 /home/python/coding.py
Array yang diurutkan: [39755316 43999578 69484001 70604163 87345563]
Waktu pemrosesan: 0.0001 detik
zorin@master:~$
```

Contoh output program “coding.py” jika jumlah datanya 5

Jumlah Data	Estimasi Waktu (detik)	
	Serial	Paralel
10	0,0001	0,0012
100	0,0012	0,0056
1.000	0,1093	0,0215
10.000	11,0147	1,1391

Tabel praktikum dari program “coding.py”

B. Program Kedua

Diprogram kedua, kita akan memakai program eliminasi gauss yang telah dimodifikasi agar dapat berjalan secara paralel menggunakan MPI dengan program “dica.py”. Jika berhasil, program akan mengeluarkan output berupa hasil perhitungan SPL menggunakan metode eliminasi gauss tergantung dari jumlah data program disertai pewaktu. Sama sepeti program pertama, program kedua akan dijalankan secara serial dan paralel dengan jumlah ordo data 100, 200, 300, 400 dan terakhir yaitu 500. Berikut adalah hasil *benchmark* dari program kedua.

```
zorin@master:~$ python3 /home/python/dica.py
Eliminasi Gauss:
1.00 4.20 2.98 0.61 ...
-0.00 1.00 0.58 -0.13 ...
-0.00 -0.00 1.00 2.31 ...
x1 = -0.14
x2 = -1.46
x3 = 2.31
Estimasi Waktu proses: 0.000154 detik
```

Contoh Output program “dica.py” jika jumlah ordo 3

Jumlah Ordo	Estimasi Waktu (detik)	
	Serial	Paralel
100	0,236940	0,257183
200	1,986445	1,799140
300	6,150870	5,961488
400	15,701769	14,182887
500	32.065418	27,785337

Tabel praktikum dari program “dica.py”

C. Analisa dan kesimpulan

Dari tabel praktikum pertama dan kedua, jumlah data bisa sangat berpengaruh dalam kecepatan pemrosesan data dimulai. Jika dihadapkan pada data yang banyak, pemrosesan paralel lebih efisien digunakan karena estimasi waktu yang diperlukan lebih singkat dibanding pemrosesan serial. Ini disebabkan pemrosesan paralel membagi tugas dengan VM lainnya, sehingga beban yang ditanggung akan dibagi secara merata dan dapat mengurangi waktu yang terbuang karena pemrosesan serial. Bisa dilihat pada saat jumlah data diangka 1.000 dan 10.000 dari program pertama, terlihat kecepatan

pemrosesan paralel cukup jauh melampaui kecepatan pemrosesan data secara serial. Begitu juga dengan jumlah ordo 300-500 di program kedua. Semakin banyak jumlah data yang diproses, semakin jauh jarak estimasi waktu antar serial dan paralel.

Namun hal yang sama juga akan terjadi sebaliknya. Semakin sedikit jumlah data, semakin lambat pemrosesan paralel dibanding serial. Hal ini karena pemrosesan paralel membutuhkan waktu tambahan untuk mentransfer data dan mengembalikkan output pada VM lain. Membuat pemrosesan serial dapat menampilkan output lebih cepat karena ia tidak perlu menyalurkan bebannya pada VM lain. Contohnya pada ordo 100-200 program kedua dan data 10-100 pada program pertama.

Oleh karena itu, kita harus menentukkan dan melihat berapa banyak jumlah data yang akan diproses sebelum memutuskan apakah harus menggunakan paralel atau serial. Salah satu caranya yaitu menentukkan “batas tengah” dimana jarak estimasi waktu paralel dan serial tidak jauh berbeda. Pemrosesan paralel akan sangat cocok dipakai pada kondisi jumlah data yang masuk sangat banyak. Sementara, pemrosesan serial akan menghemat waktu user untuk jumlah data yang masuk tergolong sedikit.