LAPORAN PRAKTIKUM
MESSAGE PASSING INTERFACE (MPI)

Disusun Oleh:

MUHTADIN

09011182227102

DOSEN PENGAMPU:

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

PROGRAM STUDI SISTEM KOMPUTER

FAKULTAS ILMU KOMPUTER

UNIVERSITAS SRIWIJAYA

2023

LAPORAN PRAKTIKUM

Pada laporan ini Setelah mempersiapkan MPI dan mencoba program sort bubble pada praktikum sebelumnya, kita akan mencoba untuk membandingkan kecepatan program yang dijalankan secara serial dan paralel menggunakan MPI. Dua program yang digunakan adalah "coding.py", yang berisi program sort bubble, dan "dica.py," yang berisi program numerik yang menggunakan metode eliminasi gauss untuk menghitung sistem persamaan linier (SPL). Agar program dapat berjalan secara stabil, program akan dijalankan pada tiga virtual machine (VM) menggunakan adapter Host Only.

```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")
```

Gambar 1. 1 Program python dengan file 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()
```

Gambar 1. 2 Program python dengan file dica.py

1. **Pada Program Pertama**

Kita akan menggunakan "coding.py" untuk program pertama, dengan ukuran data 10, 100, 1.000, dan 10.000. Jika program berhasil, ia akan mengeluarkan output yang terdiri dari angka acak yang disusun mulai dari yang paling rendah hingga yang tertinggi, diikuti dengan waktu sebagai standar. Untuk menjalankan program, ada dua kondisi perintah. Yang pertama adalah "python3 <nama_program>" untuk menjalankan program dengan satu virtual machine, dan yang kedua adalah "mpiexec -n <jumlah_komputer> -host

<nama_host1>, <nama_host_selanjutnya> python3 <nama_program>" untuk menjalankan program secara paralel. Hasil benchmark program yang dijalankan secara bersamaan dan secara terpisah ditunjukkan di sini.

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

Gambar 1. 3 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

Gambar 1. 4 Table run time

2. Pada Program Kedua

Di program kedua, kita akan menggunakan program eliminasi gauss yang telah diubah agar dapat berjalan secara paralel menggunakan MPI dengan program "dica.py." Jika program berhasil, itu akan mengeluarkan hasil perhitungan SPL menggunakan metode eliminasi gauss, yang bergantung pada jumlah data program yang disertai pewaktu. Program kedua, seperti program pertama, akan dijalankan secara serial dan paralel dengan jumlah ordo data 100, 200, 300, 400, dan 500, yang merupakan 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
```

Gambar 1. 5 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

Gambar 1. 6 Tabel Run Time pada file dica.py

3. Analisa dan kesimpulan pada laporan

Jumlah data dapat memengaruhi kecepatan awal pemrosesan data, seperti yang ditunjukkan dalam tabel praktikum pertama dan kedua. Karena estimasi waktu yang diperlukan lebih singkat daripada pemrosesan serial, pemrosesan paralel lebih baik digunakan ketika dihadapkan pada data yang banyak. Ini disebabkan oleh fakta bahwa pemrosesan paralel membagi tugas dengan virtual machine (VM) lainnya, sehingga beban yang ditanggung dibagi secara merata, yang memungkinkan pemrosesan serial untuk mengurangi jumlah waktu yang dihabiskan. Untuk program pertama, kecepatan pemrosesan data paralel jauh melampaui pemrosesan data serial. Ini juga berlaku untuk program kedua, dengan jumlah ordo 300-500. Estimasi waktu serial dan paralel meningkat seiring dengan jumlah data yang diproses.

Tetapi hal yang sama juga akan terjadi di sisi lain. Karena pemrosesan paralel membutuhkan waktu lebih lama untuk mentransfer data dan mengembalikan output ke virtual machine (VM) lain, pemrosesan serial dapat menampilkan output lebih cepat karena tidak perlu menyalurkan bebannya ke VM lain. Akibatnya, pemrosesan paralel lebih lambat dengan jumlah data yang lebih kecil. Sebagai contoh, program kedua memiliki data 10-100 dan program pertama memiliki ordo 100-200.

Sehingga sebelum memutuskan apakah menggunakan paralel atau serial, kita harus menentukkan dan menghitung jumlah data yang akan diproses. Salah satu cara untuk melakukan ini adalah dengan menentukkan "batas tengah", yang jarak perkiraan waktu serial dan paralel hampir sama. Pada situasi di mana jumlah data yang masuk sangat besar, pemrosesan paralel akan sangat membantu, sementara pemrosesan serial akan menghemat waktu bagi pengguna dengan jumlah data yang masuk yang sedikit.