MODUL 2
Konsep Dasar Komputasi Paralel

1.1. Tujuan

a. Memahami konsep task parallelism dan data parallelism.
b. Melihat perbedaan waktu eksekusi antara program serial dan paralel.
c. Mengidentifikasi jenis paralelisme yang sesuai untuk suatu permasalahan komputasi.

1.2. Alat & Bahan

1. Perangkat Lunak:
   - Bahasa Pemrograman: Python
   - Library: numpy, multiprocessing, time, random
2. Perangkat Keras:
   - Laptop/PC dengan RAM minimal 8 GB
   - Koneski internet

1.3. Materi

A. Simulasi

Simulasi adalah teknik untuk menirukan perilaku suatu sistem dengan model komputer. Simulasi dilakukan ketika eksperimen langsung di dunia nyata terlalu mahal, berbahaya, atau tidak praktis. Beberapa contohnya seperti simulasi lalu lintas, jaringan komputer, sistem militer, ekosistem. Keterbatasan simulasi dengan teknik serial adalah simulasi berjalan pada satu prosesor sehingga waktu komputasi lama. Hal tersebut tidak efisien untuk model skala besar misalnya jutaan entitas. Simulasi secara serial sulit menangani real-time simulation. Untuk mengatasi keterbatasan tersebut, simulasi dibagi menjadi beberapa bagian yang dijalankan bersamaan.

- Parallel Simulation menggunakan banyak prosesor dalam satu mesin (shared memory).
- Distributed Simulation menggunakan banyak mesin berbeda yang terhubung lewat jaringan (distributed memory).

B. Konsep Paralelisme

Dalam sistem paralel, ada dua pendekatan utama:

1. Task Parallelism

   Membagi program menjadi beberapa tugas berbeda yang dapat dieksekusi secara bersamaan pada prosesor yang berbeda. Karakteristiknya adalah tugas tidak selalu bekerja pada data yang sama. Task parallelism cocok untuk pekerjaan yang

independen atau relatif minim komunikasi antar tugas. Contoh nyatanya dalam sebuah restoran misal satu orang memasak, satu orang mencuci piring, satu orang melayani pelanggan, semua berjalan bersamaan. Dalam simulasi komputer proses A menghitung rata-rata dataset, sementara proses B melakukan sorting dataset.

2. Data Parallelism
   Membagi data besar menjadi beberapa potongan (subsets) lalu menjalankan operasi yang sama pada tiap potongan data secara paralel. Karakteristiknya adalah semua prosesor menjalankan instruksi yang sama, hanya berbeda bagian data. Data parallelism umumnya membutuhkan penggabungan hasil akhir (*reduce step*). Contoh nyata nya membagi 1.000 mahasiswa ke dalam 4 kelompok, lalu setiap dosen menghitung nilai rata-rata kelompok masing-masing. Hasilnya digabung untuk mendapatkan rata-rata global. Dalam komputasi komputer data 1 juta angka dibagi ke 4 prosesor masing-masing menghitung rata-rata subset lalu digabungkan.

C. Tantangan

Menurut Fujimoto, tantangan utama dalam simulasi paralel dan terdistribusi adalah:

1. Sinkronisasi Waktu (Time Synchronization)
   Setiap proses biasanya merepresentasikan bagian dari simulasi. Harus dijaga agar urutan kejadian logis tetap konsisten. Contohnya dalam simulasi lalu lintas, mobil A tidak boleh “tiba di tujuan” sebelum jalan yang dilewati terbuka.
2. Komunikasi Antar Proses
   Proses seringkali perlu bertukar informasi. Pada shared memory komunikasi yang terjadi relatif cepat. Pada distributed memory (jaringan) komunikasi bisa jadi bottleneck (latensi tinggi). Contohnya pada simulasi jaringan komputer, jika satu node mengirim paket ke node lain harus ada pertukaran pesan antar proses.
3. Konsistensi Data
   Harus dijaga agar tidak ada proses yang menggunakan data yang sudah “usang” atau bertentangan. Contohnya dalam simulasi pertempuran tidak boleh satu unit dinyatakan “aktif” di proses A sementara proses B sudah menandainya “hancur”.
4. Load Balancing (Pembagian Beban)
   Jika pembagian tugas atau data tidak seimbang maka sebagian prosesor menganggur. Contohnya satu prosesor diberi 90% data, sementara yang lain hanya 10%.
5. Harus ada Paralelisme
   Komunikasi dan sinkronisasi bisa menambah waktu eksekusi. Namun kadang jika dataset kecil eksekusi paralel malah lebih lambat daripada serial.

1.4.Langkah Praktikum

A. Simulasi Paralel dan simulasi serial/sequential

Perhatikan gambar 1, pada gambar 1 ditunjukan simulasi bagaimana beberapa pelanggan bisa dilayani secara paralel menggunakan multiprocessing di Python. Simulasi yang ditampilkan dalam gambar 1 dapat diimplementasikan untuk beberapa kasus seperti loket bank atau antrian rumah sakit, dimana beberapa pelanggan ditangani oleh petugas berbeda. Kasus lainnya server web adanya beberapa permintaan (request) yang ditangani secara bersamaan oleh worker process. Hingga ke kasus data processing, membagi pekerjaan besar menjadi potongan-potongan yang bisa diproses oleh CPU core yang berbeda.

Parameter `nomor` yang ada di dalam fungsi layani pelanggan merupakan identitas pelanggan (mis,1,2,3,dst). Kemudian fungsi `random.randint(1,3)` merupaka lama pelayanan setiap pelanggan berbeda-beda, antara 1 sampai 3 detik. Fungsi `def layani_pelanggan(nomor)` hanya melayani 1 pelanggan. Kemudian dibuat sebuah fungsi `def loket_paralel(jumlah_pelanggan=5)`, nilai 5 artinya ada lima pelanggan. Setiap pelanggan akan masing masing diproses. Jika proses selesai dan dicetak maka secara paralel setiap pelanggan selesai dilayani.

```python
from multiprocessing import Process
import time, random

def layani_pelanggan(nomor):
    waktu_layanan = random.randint(1,3)
    print(f"Pelanggan {nomor} mulai dilayani, estimasi {waktu_layanan} detik")
    time.sleep(waktu_layanan)
    print(f"Pelanggan {nomor} selesai dilayani")

def loket_paralel(jumlah_pelanggan=5):
    proses = []
    for i in range(jumlah_pelanggan):
        p = Process(target=layani_pelanggan, args=(i+1,))
        proses.append(p)
        p.start()

    for p in proses:
        p.join()
    print("Semua pelanggan selesai dilayani secara paralel.")

# wajib untuk macOS & Windows
if __name__ == "__main__":
    loket_paralel()
```

Gambar 1. Simulasi Paralel

Ilustrasi dari Jalannya Simulasi pada gambar 1 adalah ada 5 pelanggan kemudian semua pelanggan (1–5) masuk loket hampir bersamaan. Loket melayani pelanggan secara paralel, Pelanggan 1 dilayani dengan waktu tertentu, pelanggan 2 dilayani hingga pelanggan 5 dilayani, Karena paralel maka pelanggan yang waktu pelayanannya lebih singkat akan selesai lebih dulu, meskipun dia datang belakangan. Hal penting dalam source code pada gambar 1 adalah library multiprocessing.Process yang digunakan untuk membuat proses baru. Jadi setiap pelanggan akan dilayani oleh proses terpisah.

Perhatikan gambar 2 yang merupakan simulasi secara sekuensial. Terdapat fungsi `loket_sequential` yang mana fungsi tersebut bekerja satu per satu. Perhatikan bahwa dalam satu loket, pelanggan berikutnya harus menunggu pelanggan sebelumnya selesai. Kemudian total waktu menunggu dilayani sama dengan jumlah semua waktu layanan, contoh 5 pelanggan dengan waktu [2, 1, 3, 1, 2] maka total waktu layanan yaitu 9 detik. Output dari simulasi ini akan terurut, pelanggan 1 selesai dulu, baru pelanggan 2, hingga pelanggan 5. Banyangkan jika hanya ada satu kasir di minimarket sehingga antrian menunggu satu jalur.

```python
import time, random

def loket_sequential(jumlah_pelanggan=5):
    for i in range(jumlah_pelanggan):
        waktu_layanan = random.randint(1,3)
        print(f"Pelanggan {i+1} dilayani selama {waktu_layanan} detik")
        time.sleep(waktu_layanan)
    print("Semua pelanggan selesai dilayani.")

loket_sequential()
```

Gambar 2. Simulasi sekuensial

B. Analisi simulasi parallel dan simulasi serial/sequential

Jika jumlah pelanggan kecil maka perbedaan waktu tidak terasa signifikan namun jika jumlah pelanggan banyak maka paralel akan jauh lebih efisien. Misalnya 100 pelanggan untuk 2 detik setiap layanan rata-rata. Maka sekuensial membutuhkan waktu ≈200 detik dan paralel (misal 4 core CPU) dapat diselesaikan sekitar 50 detik saja. Namun paralel punya overhead (yang terjadi jika membuat proses baru, koordinasi antar proses). Jadi untuk tugas yang ringan, paralel bisa justru lebih berat.

C. Simulasi Task Parallelism

Perhatikan gambar 3, pada gambar 3 ditunjukan simulasi task parallelism bekerja. Contoh source code yang digunakan adalah, ada sebuah data berisi 20 angka acak (1–100) lalu melakukan dua pekerjaan utama terhadap data tersebut yaitu `hitung_rata(data)` tugasnya adalah menghitung rata-rata nilai dari data dan `sorting_data(data)` tugasnya adalah mengurutkan data dari kecil ke besar. Pada gambar ada 2 fungsi terpisah untuk menghitung rata-rata dan mengurutkan data dengan mensimulasikan lama proses masing masing 2 detik). Kedua fungsi atau tugas tersebut dijalankan bersamaan `(p1.start(); p2.start())`, lalu program utama menunggu keduanya selesai `(join)`. Karena keduanya memakan ≈2 detik, total wall-clock time yaitu ≈2 detik (kemungkinan overhead proses), bukan 4 detik.

```python
from multiprocessing import Process
import random, time

def hitung_rata(data):
    print("Mulai menghitung rata-rata...")
    time.sleep(2)  # simulasi waktu lama
    avg = sum(data) / len(data)
    print(f"Rata-rata = {avg}")

def sorting_data(data):
    print("Mulai sorting...")
    time.sleep(2)
    sorted_data = sorted(data)
    print(f"Hasil sorting = {sorted_data[:10]} ...")

if __name__ == "__main__":
    data = [random.randint(1,100) for _ in range(20)]

    p1 = Process(target=hitung_rata, args=(data,))
    p2 = Process(target=sorting_data, args=(data,))

    p1.start()
    p2.start()

    p1.join()
    p2.join()
    print("Task parallelism selesai.")
```

Gambar 3. Simulasi task parallel

D. Analisis Simulasi Task Parallelism

Dengan menggunakan parallel untuk kasus ini adalah task paralel maka keuntungan yang didapatkan yaitu wall-clock time akan menurun karena dua tugas yang independen dikerjakan secara bersamaan. Jika tiap tugas 2 detik maka paralel ≈ 2 detik dan sekuensial ≈ 4 detik. Namun pembuatan proses akan menimbulkan overhead, untuk tugas yang sangat singkat overhead bisa lebih besar daripada keuntungan yang didapat.

E. Simulasi Data Parallelism

Perhatikan gambar 4, pada gambar 4 ditunjukan simulasi data parallelism bekerja. Contoh source code adalah, ada sebuah data berisi angka acak yang kemudian data tersebut akan dibagi menjadi 2 subset `(data[:mid] dan data[mid:])`. Proses data parallel adalah ada proses 1 yang menghitung rata-rata subset pertama dan proses 2 yang menghitung rata-rata subset kedua dan kemudian rata-rata kedua subset dihitung dari gabungan rata-rata subset `((avg1 + avg2)/2)`. Gambar 4 merupakan contoh kasus sederhana namun untuk kasus tertentu misal bagaimana mengolah data besar dengan lebih cepat maka dapat dilakukan data paralel, yaitu data dibagi (subset), lalu olah secara paralel dengan beberapa proses. Hasil dari waktu eksekusi akan lebih singkat dibanding menghitung rata-rata subset secara berurutan (sequential).

```python
from multiprocessing import Process, Queue
import random

def rata_subset(data, q):
    q.put(sum(data)/len(data))

if __name__ == "__main__":
    data = [random.randint(1,100) for _ in range(1000)]
    mid = len(data)//2

    q = Queue()
    p1 = Process(target=rata_subset, args=(data[:mid], q))
    p2 = Process(target=rata_subset, args=(data[mid:], q))

    p1.start(); p2.start()
    p1.join(); p2.join()

    avg1 = q.get()
    avg2 = q.get()

    rata_global = (avg1 + avg2) / 2
    print(f"Rata-rata global = {rata_global}")
```

Gambar 4. Simulasi Data paralel