

MODUL 4

Algoritma Paralel Dasar

1.1. Tujuan

- a. Memahami prinsip dasar *Divide and Conquer* (membagi, menyelesaikan, menggabungkan) dalam pemecahan masalah.
- b. Mengidentifikasi kasus nyata yang dapat dimodelkan dengan pendekatan *Divide and Conquer*.
- c. Mampu membandingkan perbedaan hasil eksekusi dan efisiensi antara metode sekuensial dengan metode paralel.

1.2. Alat & Bahan

1. Perangkat Lunak:
 - Library (multiprocessing)
2. Perangkat Keras:
 - Laptop/PC dengan RAM minimal 8 GB
 - Koneksi internet

1.3. Materi

Divide and Conquer (D&C) adalah teknik pemrograman algoritmik yang memecah masalah kompleks menjadi submasalah yang lebih kecil, mirip struktur rekursif. Divide and Conquer adalah strategi pemecahan masalah dengan tiga langkah:

1. Pecah masalah besar menjadi submasalah yang lebih kecil dan relatif independen.
2. Selesaikan tiap submasalah, bisa secara paralel bila memungkinkan.
3. Satukan hasil dari submasalah menjadi solusi keseluruhan.

Divide and conquer penting dilakukan karena sebuah submasalah bisa dijalankan di prosesor berbeda. Waktu eksekusi bisa lebih cepat dibandingkan algoritma sekuensial. Contoh algoritma divide and conquer yaitu pengurutan (merge sort, quick sort), pencarian maksimum/ pencarian minimum, perkalian matriks.

Pada materi sebelumnya dalam menyelesaikan masalah dengan model komputasi paralel menggunakan library multiprosesor, library tersebut adalah sistem komputer dengan lebih dari satu CPU (core) yang dapat bekerja bersamaan. Fungsi yang ada dalam multiprosesor adalah setiap core bisa mengerjakan bagian berbeda dari suatu masalah secara paralel. Sehingga library tersebut dapat digunakan untuk algoritma Divide and Conquer

seperti *Merge Sort* atau *Quick Sort*. Beberapa hal yang perlu dipertimbang jika ingin mengerjakan sebuah algoritma paralel:

1. Kompleksitas Waktu
2. Kecepatan dan Efisiensi
3. Amdahl's Law (Batas Paralelisasi)

1.4.Langkah Praktikum

1. Analisis Kasus 1.

Algoritma Merge Sort bekerja dengan konsep Divide and Conquer. Gambar 1 merupakan source code algoritma merge sort sederhana. Pada gambar langkah yang dilakukan yaitu Divide dengan membagi array ukuran n menjadi dua subarray ukuran $n/2$. Conquer melakukan rekursi dalam menyelesaikan tiap subarray. Kemudian terakhir Combine dengan menggabungkan dua array terurut dengan kompleksitas $O(n)$. Kompleksitas waktu merge sort = $O(n \log n)$. Pada gambar hasil eksekusi untuk data kecil (8 elemen) waktu eksekusi akan hampir nol, misalnya 0.00001 detik. Untuk data lebih besar (misalnya 1 juta elemen), waktu akan naik sesuai $O(n \log n)$.

Contoh skenario:

- a. 1.000 data waktu eksekusi sekitar milidetik.
- b. 100.000 data waktu eksekusi ratusan milidetik.
- c. 1.000.000 data waktu eksekusi beberapa detik.

Jadi, semakin besar input n , waktu eksekusi akan bertambah sesuai pola $n \log n$.

```

import time

def merge(left, right):
    result, i, j = [], 0, 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])
    result.extend(right[j:])
    return result

def merge_sort(seq):
    if len(seq) <= 1:
        return seq
    mid = len(seq) // 2
    left = merge_sort(seq[:mid])      # rekursi sequential
    right = merge_sort(seq[mid:])    # rekursi sequential
    return merge(left, right)

data = [12, 3, 19, 5, 2, 45, 33, 7]

start_time = time.time() # waktu mulai
sorted_data = merge_sort(data)
end_time = time.time()   # waktu selesai

print("Data sebelum sort:", data)
print("Data sesudah sort:", sorted_data)
print("Waktu eksekusi: %.6f detik" % (end_time - start_time))

```

2. Analisis Kasus 2.

Algoritma Merge Sort bekerja dengan konsep Divide and Conquer. Gambar 2 merupakan source code algoritma merge sort sederhana. Pada gambar langkah yang dilakukan yaitu data dibagi dua kemudian masing-masing bagian diproses oleh proses terpisah (p1, p2). Proses child akan mengirim hasil ke proses parent lewat Pipe. Proses parent akan menerima hasil, lalu memanggil merge() untuk menggabungkan kedua hasil. Perbedaan

kasus pertama dan kasus ke dua terletak pada proses sorting yang dilakukan secara bersamaan oleh masing masing proses kemudian akan digabungkan. Waktu eksekusi pada gambar 2 hasilnya akan sangat cepat < 0.01 detik namun seringkali versi sequential lebih cepat dari paralel, karena overhead yang membuat proses baru lebih besar daripada kerja sorting itu sendiri. Lain halnya untuk data besar (misalnya 100_000 atau 1_000_000 elemen random) paralel akan lebih cepat daripada sequential, karena beban kerja lebih besar dibanding overhead. Dalam sequential merge sort kompleksitas waktu adalah O(nlogn). Parallel Merge Sort tetap O(nlogn), tapi ada faktor percepatan karena bagian kiri dan kanan bisa dikerjakan bersamaan. Pada komputer multi-core bisa mendekati 2 kali lebih cepat namun percepatan hanya terasa pada dataset besar.

```
import time
from multiprocessing import Process, Pipe

def merge(left, right):
    result, i, j = [], 0, 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])
    result.extend(right[j:])
    return result

def parallel_merge_sort(data, conn):
    if len(data) <= 1:
        conn.send(data)
        conn.close()
        return

    mid = len(data) // 2

    # buat pipe untuk komunikasi (shared memory)
    left_conn_parent, left_conn_child = Pipe()
    right_conn_parent, right_conn_child = Pipe()

    # spawn proses
    p1 = Process(target=parallel_merge_sort, args=(data[:mid], left_conn_child))
    p2 = Process(target=parallel_merge_sort, args=(data[mid:], right_conn_child))

    p1.start()
    p2.start()

    left = left_conn_parent.recv()
    right = right_conn_parent.recv()

    p1.join()
    p2.join()

    # merge hasil
    conn.send(merge(left, right))
    conn.close()
```