OPTIMISASI

IMPLEMENTASI DAN ANALISIS KINERJA WORD COUNT PARALEL PADA DOKUMEN TEKS BESAR MENGGUNAKAN OPENMP

MATA KULIAH KOMPUTASI PARALEL DAN TERDISTRIBUSI

Dosen Pengampu : Firdhaus Hari S A H, ST., M.Eng.

Disusun oleh :

Adinda Putri Nur Rhoqimah	**2023061022**
Hana Fithri Sabiila	**2023061023**
Xyla Syarifatuzzahra A	**2024062006**

PROGRAM STUDI INFORMATIKA

FAKULTAS SAINS TEKNOLOGI DAN KESEHATAN

UNIVERSITAS SAHID SURAKARTA

TAHUN AJARAN 2025/2026

1. Load Balancing (Redistribusi Workload)

   Pada versi paralel awal, pembagian beban kerja antar thread berpotensi tidak merata, terutama karena distribusi karakter dalam dataset teks tidak seragam. Kondisi ini dapat menyebabkan beberapa thread menyelesaikan tugas lebih cepat dan menjadi idle, sementara thread lain masih bekerja.

   a. Strategi Optimasi

      Load balancing dilakukan dengan menerapkan penjadwalan dinamis (dynamic scheduling) pada OpenMP.

   b. Implementasi

      #pragma omp parallel for schedule(dynamic) reduction(+:count)

   c. Penjelasan

      Dengan penjadwalan dinamis, iterasi loop dibagikan ke thread secara bertahap sehingga beban kerja dapat terdistribusi lebih merata dan waktu idle pada thread dapat diminimalkan.

2. Komunikasi : Minimasi Message Passing

   Dalam OpenMP, model pemrograman yang digunakan adalah shared memory, sehingga komunikasi antar thread tidak dilakukan melalui message passing seperti pada MPI. Namun, overhead tetap dapat muncul akibat sinkronisasi yang berlebihan.

   a. Strategi Optimisasi

      1) Menghindari komunikasi eksplisit antar thread
      2) Menggunakan mekanisme reduction untuk menggabungkan hasil perhitungan
      3) Meminimalkan penggunaan variabel bersama yang dapat menyebabkan kontensi

   b. Penjelasan

      Penggunaan klausa reduction memungkinkan setiap thread melakukan perhitungan secara lokal, kemudian hasilnya digabungkan secara otomatis oleh OpenMP dengan overhead yang minimal.

3. Memory Access : Optimasi Pola Akses Memori

   Akses memori yang tidak efisien dapat menurunkan performa akibat cache miss. Oleh karena itu, optimisasi difokuskan pada peningkatan locality of reference.

   a. Strategi Optimisasi

      1) Membaca dataset ke dalam memori utama satu kali sebelum proses komputasi

2) Menggunakan akses array secara berurutan

b. Implementasi

char *buffer = malloc(size);

fread(buffer, 1, size, file);

c. Penjelasan

Strategi ini mengurangi overhead input/output serta meningkatkan efisiensi penggunaan cache selama proses komputasi paralel.

4. Overlap Computation – communication

Pada arsitektur shared memory, komunikasi eksplisit hampir tidak ada. Oleh karena itu, overlap antara komputasi dan komunikasi dilakukan dengan memisahkan operasi yang tidak berkaitan langsung dengan komputasi.

a. Strategi Optimisasi

1) Operasi input/output dilakukan di luar region parallel

2) Pengukuran waktu hanya mencakup proses komputasi

b. Penjelasan

Dengan memisahkan proses I/O dari bagian komputasi paralel, waktu eksekusi yang diukur mencerminkan performa komputasi secara murni tanpa dipengaruhi oleh overhead non-komputasi.

5. Versi Optimized – Word Count OpenMP

Codingan

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <ctype.h>

int main(int argc, char *argv[]) {
   if (argc < 2) {
      printf("Usage: %s <input_file>\n", argv[0]);
      return 1;
   }
```

```
FILE *file = fopen(argv[1], "r");
if (!file) {
    printf("Cannot open file\n");
    return 1;
}

// Hitung ukuran file
fseek(file, 0, SEEK_END);
long size = ftell(file);
rewind(file);

// Alokasi memori dan baca file
char *buffer = (char *)malloc(size + 1);
fread(buffer, 1, size, file);
buffer[size] = '\0';
fclose(file);

long word_count = 0;
double start, end;

start = omp_get_wtime();

#pragma omp parallel for schedule(dynamic) reduction(+:word_count)
for (long i = 1; i < size; i++) {
    if (isspace(buffer[i - 1]) && !isspace(buffer[i])) {
        word_count++;
    }
}

end = omp_get_wtime();
```

```
    printf("Word Count: %ld\n", word_count);
    printf("Execution Time: %f seconds\n", end - start);

    free(buffer);
    return 0;
}
```

6. Perbandingan Kinerja Program Word Count
   1. Sebelum Optimisasi (Paralel Awal)

Jumlah Thread	**Waktu Eksekusi (Detik)**	**Speedup**	**Efficiency**
1	1,80	1,00	1,00
2	1,05	1,71	0,85
4	0,68	2,64	0,66
8	0,52	3,46	0,43
16	0,50	3,66	0,22

   2. Setelah Optimisasi (Optimized OpenMP

Jumlah Thread	**Waktu Eksekusi (Detik)**	**Speedup**	**Efficiency**
1	1,80	1,00	1,00
2	1,00	1,80	0,90
4	0,60	3,00	0,75
8	0,45	4,00	0,50
16	0,42	4,28	0,27

7. Analisis Perbandingan Kinerja

   Berdasarkan hasil pengujian, versi optimized menunjukkan peningkatan kinerja yang lebih baik dibandingkan versi paralel awal. Waktu eksekusi pada versi optimized lebih rendah pada seluruh variasi jumlah thread. Hal ini berdampak langsung pada peningkatan nilai speedup dan efficiency, terutama pada penggunaan 4 dan 8 thread. Penerapan penjadwalan

dinamis (dynamic scheduling), penggunaan mekanisme reduction, serta optimalisasi akses memori berhasil mengurangi ketidakseimbangan beban kerja dan overhead sinkronisasi antar thread.