

Parallel Computing

Tugas Implementasi Shared-memory Parallel Computing



Anggota Kelompok

Adrian Fathan Imama	232410103047
Muhammad Sultan Ma'arif	232410103030
Billie Surya Pratama	232410103018

UNIVERSITAS JEMBER
FAKULTAS ILMU KOMPUTER
PRODI INFORMATIKA

Tugas

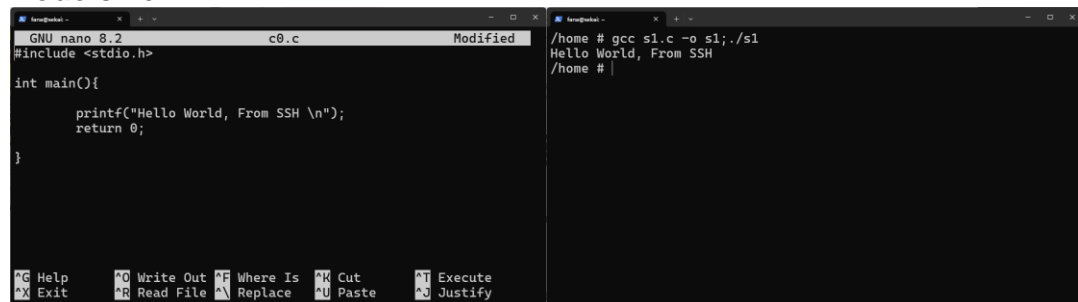
1. Program parallel computing dikerjakan menggunakan bahasa pemrograman C atau C++ dengan library openMP.
2. Implementasi program setidaknya menerapkan contoh-contoh di slide materi dengan modifikasi secukupnya.
3. Mahasiswa diharapkan mampu menjelaskan seluk beluk cara kerja program yang telah dikerjakannya.
4. Kumpulkan source code hasil implementasi satu persatu, tidak di-zip.
5. Susun juga laporan dalam bentuk PDF A4

Praktek

1. Contoh Kode #1 - Hello World

1.1. Program Serial

Kode s1.c :



```
GNU nano 8.2 c0.c Modified
#include <stdio.h>

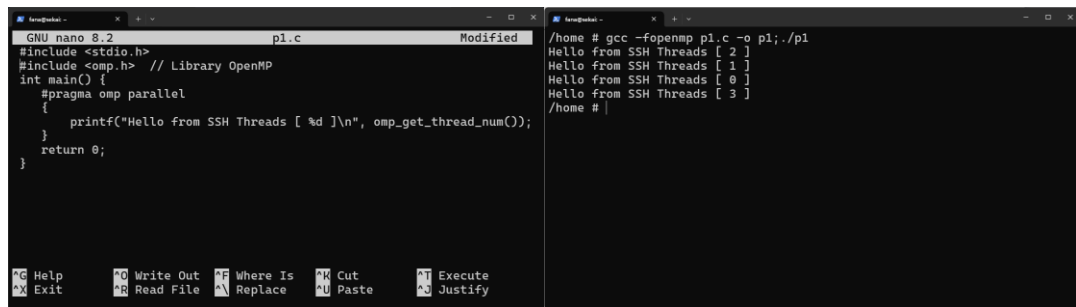
int main(){
    printf("Hello World, From SSH \n");
    return 0;
}

/home # gcc s1.c -o s1; ./s1
Hello World, From SSH
/home #
```

Diatas merupakan contoh program serial hello world di bahasa **C** baris pertama kode **#include <stdio.h>** adalah untuk mengimport standart input output library dari **C** dan function **int main** adalah untuk mendefine fungsi utama yang akan dijalankan oleh program selanjutnya mengprint "Hello World, From SSH" diikuti dengan newline dengan **\n**.

1.2. Program Parallel

Kode p1.c :



```
GNU nano 8.2 p1.c
#include <stdio.h>
#include <omp.h> // Library OpenMP
int main() {
    #pragma omp parallel
    {
        printf("Hello from SSH Threads [ %d ]\n", omp_get_thread_num());
    }
    return 0;
}
```

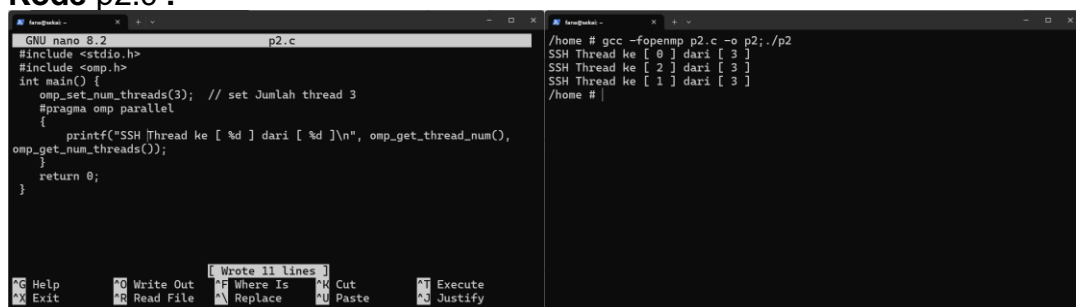
```
/home # gcc -fopenmp p1.c -o p1; ./p1
Hello from SSH Threads [ 2 ]
Hello from SSH Threads [ 1 ]
Hello from SSH Threads [ 0 ]
Hello from SSH Threads [ 3 ]
/home #
```

Disini kode program mengimport library **OpenMP** dengan syntax **#include <omp.h>** selanjutnya setelah direktif **#pragma omp parallel** runtime **OpenMP** akan membuat sekelompok thread dan menjalankan kode di dalam bloknya di setiap thread dan akan mengprint thread nomor berapa blok kode printf tersebut dijalankan menggunakan **omp_get_thread_nums()**.

2. Contoh Kode #2 – Mengatur Jumlah Thread

2.1. Program Parallel

Kode p2.c :



```
GNU nano 8.2 p2.c
#include <stdio.h>
#include <omp.h>
int main() {
    omp_set_num_threads(3); // set Jumlah thread 3
    #pragma omp parallel
    {
        printf("SSH Thread ke [ %d ] dari [ %d ]\n", omp_get_thread_num(),
omp_get_num_threads());
    }
    return 0;
}
```

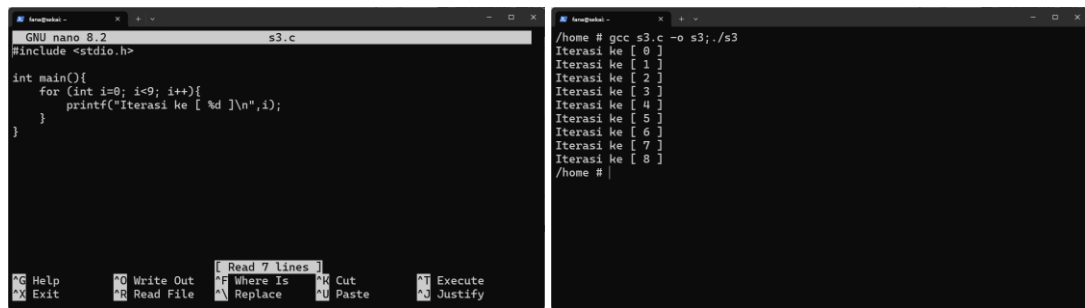
```
/home # gcc -fopenmp p2.c -o p2; ./p2
SSH Thread ke [ 0 ] dari [ 3 ]
SSH Thread ke [ 2 ] dari [ 3 ]
SSH Thread ke [ 1 ] dari [ 3 ]
/home #
```

Disini program di set agar hanya menggunakan 3 thread dari 4 thread yang tersedia di server menggunakan function **omp_set_num_threads(3)** kemudian pada block parallel kita mengprint block kode berjalan di thread nomor berapa menggunakan **omp_get_thread_num()** dan mengprint total threads yang di reserve atau digunakan oleh **OpenMP** menggunakan **omp_get_num_threads()**.

3. Contoh Kode #3 – Looping

3.1. Program Serial

Kode s3.c :



```
GNU nano 8.2 s3.c
#include <stdio.h>

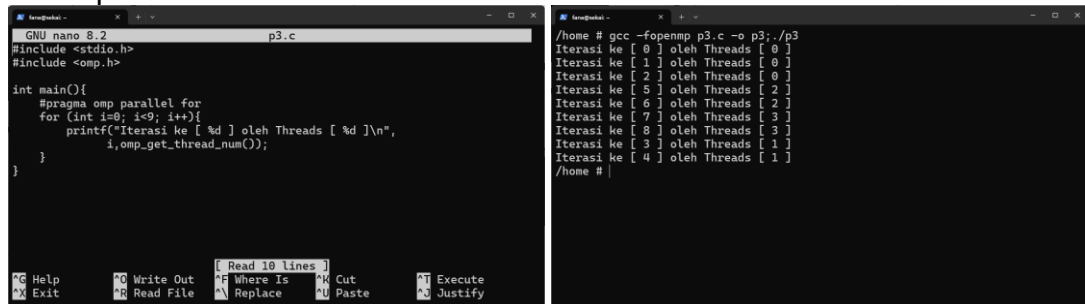
int main(){
    for (int i=0; i<9; i++){
        printf("Iterasi ke [ %d ]\n",i);
    }
}

/home # gcc s3.c -o s3; ./s3
Iterasi ke [ 0 ]
Iterasi ke [ 1 ]
Iterasi ke [ 2 ]
Iterasi ke [ 3 ]
Iterasi ke [ 4 ]
Iterasi ke [ 5 ]
Iterasi ke [ 6 ]
Iterasi ke [ 7 ]
Iterasi ke [ 8 ]
/home #
```

Di kode ini program melakukan looping sebanyak 9 kali dari 0 - 8 dan meng-print iterasi seberapa sekarang mulai dari 0 sampai 8 dan dapat dilihat bahwa outputnya berurutan dari 0 - 8.

3.2. Program Parallel

Kode p3.c :



```
GNU nano 8.2 p3.c
#include <stdio.h>
#include <omp.h>

int main(){
    #pragma omp parallel for
    for (int i=0; i<9; i++){
        printf("Iterasi ke [ %d ] oleh Threads [ %d ]\n",
            i, omp_get_thread_num());
    }
}

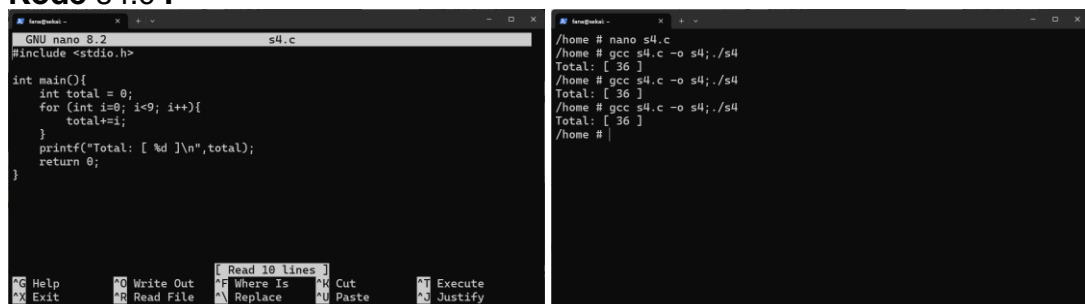
/home # gcc -fopenmp p3.c -o p3; ./p3
Iterasi ke [ 0 ] oleh Threads [ 0 ]
Iterasi ke [ 1 ] oleh Threads [ 0 ]
Iterasi ke [ 2 ] oleh Threads [ 0 ]
Iterasi ke [ 5 ] oleh Threads [ 2 ]
Iterasi ke [ 6 ] oleh Threads [ 2 ]
Iterasi ke [ 7 ] oleh Threads [ 3 ]
Iterasi ke [ 8 ] oleh Threads [ 3 ]
Iterasi ke [ 3 ] oleh Threads [ 1 ]
Iterasi ke [ 4 ] oleh Threads [ 1 ]
/home #
```

Disini program melakukan **For Loop** dengan Parallel menggunakan **#pragma omp parallel for**, Program melakukan Loop Parallel sebanyak 9 kali dari 0 - 8 dan men-outputkan iterasi saat ini dan juga Threads berapa yang menjalankannya, berbeda dengan **For Loop** serial, **For Loop** Parallel dilakukan secara bersamaan sehingga hasilnya tidak berurutan seperti gambar diatas.

4. Contoh Kode #4 – Mengakses resource yang sama

4.1. Program Serial

Kode s4.c :



```
GNU nano 8.2 s4.c
#include <stdio.h>

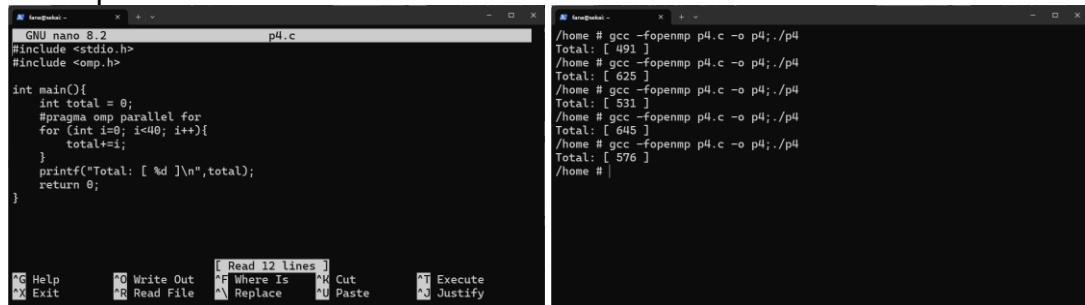
int main(){
    int total = 0;
    for (int i=0; i<9; i++){
        total+=i;
    }
    printf("Total: [ %d ]\n",total);
    return 0;
}

/home # nano s4.c
/home # gcc s4.c -o s4; ./s4
Total: [ 36 ]
/home # gcc s4.c -o s4; ./s4
Total: [ 36 ]
/home #
```

Pada gambar diatas program melakukan **Loop** sebanyak 9 kali dari 0 - 9 dan menambah variable **total** dengan **i** pada setiap iterasi kemudian mengprint totalnya dapat dilihat bahwa hasil totalnya tetap.

4.2. Program Parallel

Kode p4.c :



```
GNU nano 8.2 p4.c
#include <stdio.h>
#include <omp.h>

int main(){
    int total = 0;
    #pragma omp parallel for
    for (int i=0; i<40; i++){
        total+=i;
    }
    printf("Total: [ %d ]\n",total);
    return 0;
}
```

```
/home # gcc -fopenmp p4.c -o p4; ./p4
Total: [ 491 ]
/home # gcc -fopenmp p4.c -o p4; ./p4
Total: [ 625 ]
/home # gcc -fopenmp p4.c -o p4; ./p4
Total: [ 531 ]
/home # gcc -fopenmp p4.c -o p4; ./p4
Total: [ 645 ]
/home # gcc -fopenmp p4.c -o p4; ./p4
Total: [ 576 ]
/home #
```

Disini kita melakukan **For Loop** Secara Parallel untuk Menambahkan **Total** dengan nilai **i** pada setiap iterasi akan tetapi dapat dilihat bahwa hasilnya berbeda beda ini terjadi karena program berjalan secara parallel dan tidak menentu iterasi mana yang berjalan lebih dulu dan karena variable **Total** berada di **Main Thread** dan diakses dan dimodifikasi secara bersamaan oleh banyak **Thread** maka hasilnya berbeda beda setiap kali kita jalankan karena bisa saja semisal **total = 0** itu diakses oleh 4 Thread yang berbeda dan nilai **i** yang berbeda sehingga hasilnya akan berbeda beda setiap kali dijalankan. Hal ini biasa Disebut **Race Condition** (Situasi di mana beberapa proses mengakses dan memanipulasi data bersamaan pada saat yang bersamaan)