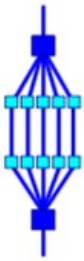
OpenMP

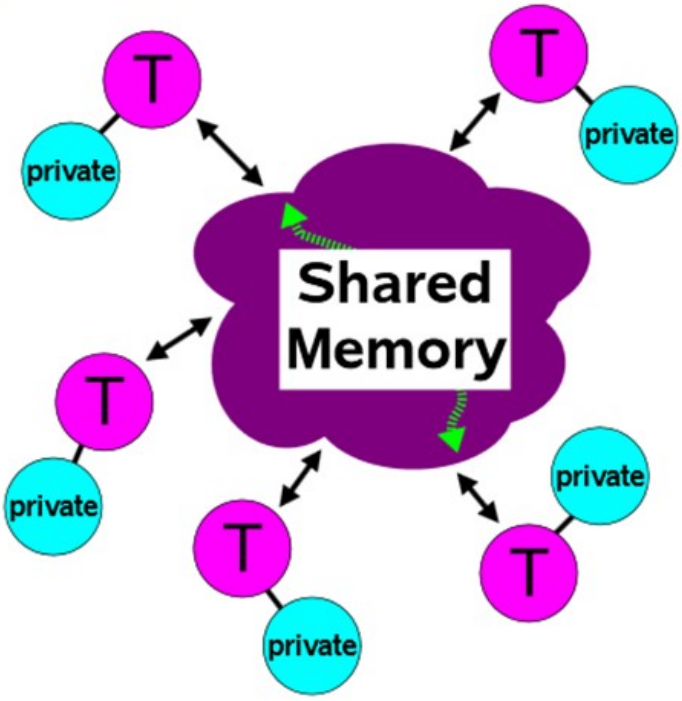
Fitriyani
fitriyani@telkomuniversity.ac.id

Persiapan Kuliah

- Laptop dengan OS Linux atau
- Install cygwin
- Compiler: gcc

Shared Memory Model





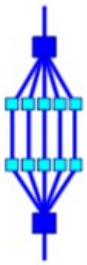
Programming Model

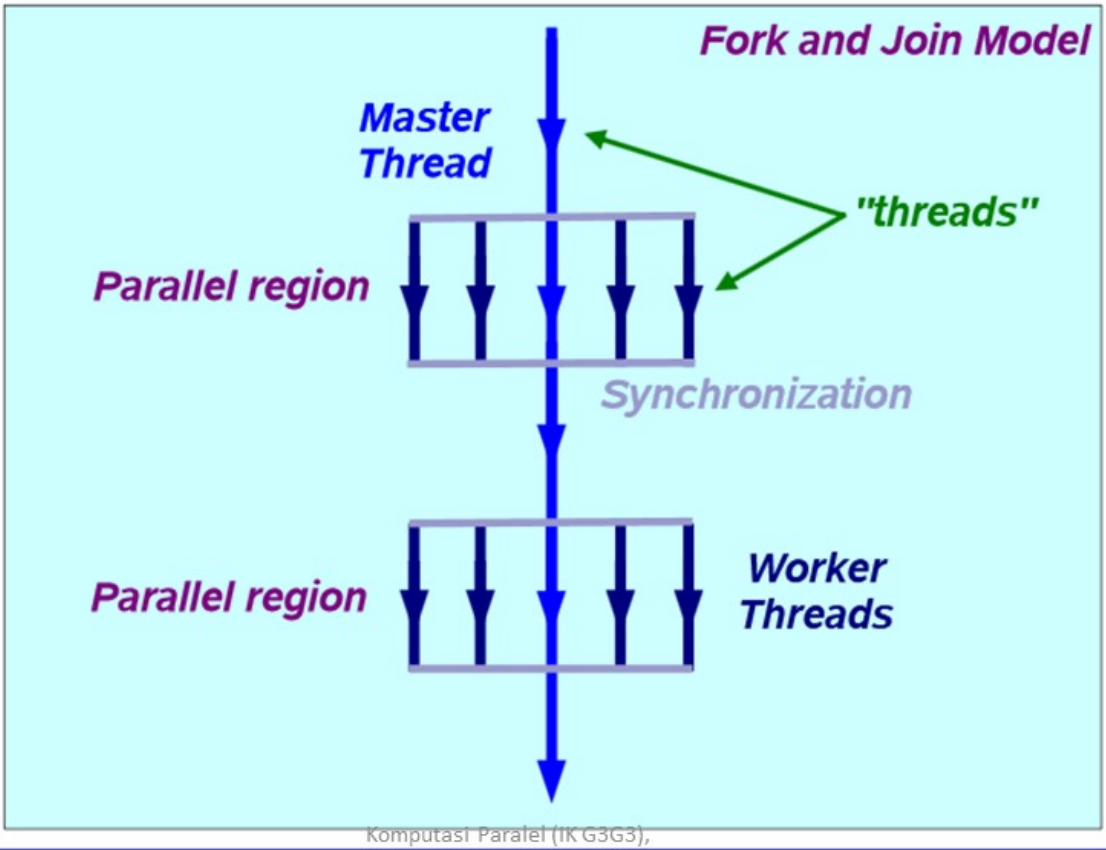
- ✔ *All threads have access to the same, globally shared, memory*
- ✔ *Data can be shared or private*
- ✔ *Shared data is accessible by all threads*
- ✔ *Private data can be accessed only by the threads that owns it*
- ✔ *Data transfer is transparent to the programmer*
- ✔ *Synchronization takes place, but it is mostly implicit*

Pengenalan Openmp

- OpenMP : **Open M**ulti **P**rocessing
- OpenMP adalah standar yang digunakan untuk pemrograman paralel *multithreading* pada arsitektur *shared memory*.
- Shared variable dan private variable
- Kompilasi program (berbasis C): `gcc -fopenmp namafile.c -0 namaoutput`

The OpenMP execution model





Sintaks Dasar

- #pragma omp construct [clause [clause] ...]
- omp_get_num_thread ( )
- omp_get_threads_num ( )
- omp_set_num_threads ( )

Contoh (1)

Bahasa C

```
#include <stdio.h>
int main()
        {
        printf("Hallo \n");
        return 0;
        }
```

OpenMP (default thread)

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

int main(){
#pragma omp parallel
//default number of thread
        {
        printf("Hallo OPenMP\n");
        }
        return 0;
        }
```

Contoh (2)

Jumlah thread (defined)

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

int main(){
omp_set_num_threads (3) ;
#pragma omp parallel
        {
        printf("Hallo OPenMP\n");
        }
        return 0;
        }
```

Jumlah thread (defined)

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

int main()
{
#pragma omp parallel num_threads (4)
{
  printf("Hallo (%d)\n");
        }
        return 0;
}
```

Contoh (3)

Thread id (default)

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

int main()
{
#pragma omp parallel
        {
int id = omp_get_thread_num();
printf("Hallo (%d)\n", id);
        }
        return 0;
}
```

Thread id (defined)

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

int main()
{
#pragma omp parallel num_threads (4)
        {
int id = omp_get_thread_num();
printf("Hallo (%d)\n", id);
        }
        return 0;
}
```

Contoh (4)

Private & shared variable

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

int main()
{
 int i= 5; int x;

#pragma omp parallel shared(i) private(x)
 {
   x = omp_get_thread_num();
   printf("x = %d, i = %d\n",x,i);
 }
}
```

Private & shared variable

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

int main()
{
 int i= 5;
#pragma omp parallel
{
   int x;
   x = omp_get_thread_num();
   printf("x = %d, i = %d\n",x,i);
 }
}
```

Worksharing (for/do loop)

- `schedule(type, chunk)`
- chunk = jumlah iterasi / jumlah *thread*
- Type:
  - static: first chunk of iterations assigned to thread 0, next one to thread 1, ... If chunk not specified : even distribution of loops among threads.
  - dynamic: iterations again assigned chunk by chunk to threads, but in any order. When a thread finishes its chunk of iterations, it can be assigned another. Default chunk = 1.

Worksharing (for/do loop)

Dynamic, Static

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

int main ()
{
int  id, i, N=5 , chunk=3;
omp_set_num_threads(2);
#pragma omp parallel private(id,i)  shared(chunk,N)
{
  id = omp_get_thread_num();
   #pragma omp for schedule (static,chunk)
   for (i=0; i<N; i++)
    printf("Loop ke: i=%d oleh thread %d \n", i, id);
}
}
```

- Eksekusi dengan jumlah N dan chunk yang variatif
- Apa bedanya static dan dynamic?

Worksharing (Section)

Section Principle

- Each section will be executed by one thread in the team.
- Lakukan running berulang-ulang, bagaimana pengamatan Anda?

Contoh

```
int id;
omp_set_num_threads(2);
#pragma omp parallel private (id)
{
 id = omp_get_thread_num();
 #pragma omp sections
  {
   #pragma omp section
    {
     printf("thread:%d doing section 1\n", id);
    }
  #pragma omp section
    {
     printf("thread:%d doing section 2\n", id);
    } } }
return 0;
}
```

Worksharing

OMP ...nowait

- Klausa agar thread jika telah selesai dari konstruk for tidak perlu menunggu thread lain selesai

Contoh

```
.....
#pragma omp sections nowait
  {
  #pragma omp section
   {
   printf("Thread %d doing section 1\n",tid);
   for (i=0; i<N; i++)
    {
    c[i] = a[i] + b[i];
    printf("Thread %d: c[%d]= %f\n",tid,i,c[i]);
    }
   }
```

Worksharing

Single-Master

- **Single**: Code pada klausa ini hanya dieksekusi oleh 1 thread
- Coba run berulang, apa yang terjadi?
- → 1 thread tidak ditentukan, tergantung sistem

Contoh

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

int main () {
 int tid;
 omp_set_num_threads(4);
#pragma omp parallel private(tid)
 {
    tid = omp_get_thread_num();
 #pragma omp single
 {
    printf("Hanya untuk thread %d saja\n",tid);
 }
     printf("Thread %d doing this part\n",tid);
}}
```

Worksharing

Single-Master

- **Master**: Code pada klausa ini hanya dieksekusi oleh thread master
- Coba run berulang, apa yang terjadi?
- → master = **thread 0**

Contoh

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

int main () {
 int tid;
 omp_set_num_threads(4);
 #pragma omp parallel private(tid)
{
    tid = omp_get_thread_num();
 #pragma omp master
 {
    printf("Hanya thread master %d saja\n",tid);
 }
    printf("Thread %d doing this part\n",tid);
}}
```

Synchronization

Barrier

- Barrier: Each thread waits for all the others before going further.
- Lakukan eksekusi program dengan barrier dan tanpa barrier, bagaimana pengamatan Anda?

Contoh

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

int main()
{
int id;
omp_set_num_threads(3);
#pragma omp parallel private (id)
        {
        id = omp_get_thread_num();
        printf("Hallo dari thread (%d)\n", id);
        #pragma omp barrier
        printf("Bye dari thread (%d)\n", id);
        }
        printf("Out of parallel region\n");
        //return 0;
```

Synchronization

Critical & Atomic

- Critical →Direktif untuk membuat wilayah kritis dalam wilayah paralel, hanya satu *thread* dalam satu waktu yang bisa masuk ke wilayah kritis ini.
- Atomic → direktif yang fungsinya sama seperti critical, tetapi hanya berlaku untuk satu *statement* aritmatika saja.

Contoh

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

int main()
{

int x=0;
omp_set_num_threads(4);
#pragma omp parallel shared (x)
                {
                #pragma omp critical
                {
                x = x+1;
                }
                }
                printf("At the end x = %d\n",x);
                return 0;
}
```

Synchronization

Reduction

- Klausa untuk mereduksi hasil dari tiap *thread* ke variabel skalar var dengan operator op: +, *, -, &, ^, |, &&, ||, min, dan max.
- Typically used with for/do loops

Contoh

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

int main()
{
   int total = 1;

   #pragma omp parallel reduction(+:total)
   total = 2;

   printf("%d\n", total);

   return 0;
}
```

Contoh2 implementasi openmp

Faktorial

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

#define  N 4

int main()
{
int i, fac= 1;
omp_set_num_threads  (2);
#pragma omp parallel for reduction(*:fac)

 for (i = 2; i <= N; i++)
#pragma omp atomic
      fac = fac*i;

   printf("%d\n", fac);

   return 0;
}
```

Perkalian matriks

```
#include<stdio.h>
#define M 4    // row
#define N 4    // column

int main()
{
   int A[M][N]  = {{2, 1, 0, 4},
                   {3, 2, 1, 1},
                   {4, 3, 1, 2},
                   {3, 0, 2, 0}};
   int b[N] = {1, 3, 4, 1};
   int c[M] = {};
   int i, j;

   // c = A*b
   for (i = 0; i < M; i++)
      for (j = 0; j < N; j++)
         c[i] += A[i][j] * b[j];

   // print c
   for (i = 0; i < M; i++)
      printf("%d\n", c[i]);

   return 0;
}
```

Referensi

- http://openmp.org/wp/resources/#Tutorials
- https://computing.llnl.gov/tutorials/openMP/
- http://cs.ipb.ac.id/~auriza/parallel/pp.html#openmp
- http://openmp.org/mp-documents/OpenMP-4.0-C.pdf
- http://www.codeproject.com/Articles/60176/A-Beginner-s-Primer-to-OpenMP
- https://www.mecs.u-picardie.fr/lib/exe/fetch.php?media=mecs:seminaireopenmp.pdf
- http://ahmadqusyairi.blogspot.co.id/2010/03/openmp-open-multi-processing.html
- http://bisqwit.iki.fi/story/howto/openmp/#ReductionClause
- https://www.ncsu.edu/hpc/Courses/8shared.html#nestedomp
- http://www.mathcs.emory.edu/~cheung/Courses/355/Syllabus/91-pthreads/openMP.html