Multicore Computing
Lecture05 - OpenMP Part II

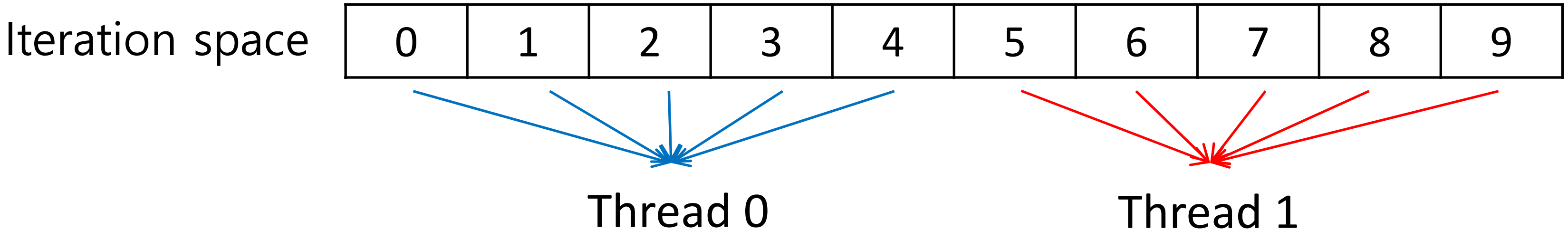
SUNG KYUN KWAN UNIVERSITY

남 범 석

bnam@skku.edu

Assigning Iterations to Threads

- `parallel for` directive
  - Basic partitioning policy → block partitioning



Iteration space

0	1	2	3	4	5	6	7	8	9

Thread 0

Thread 1

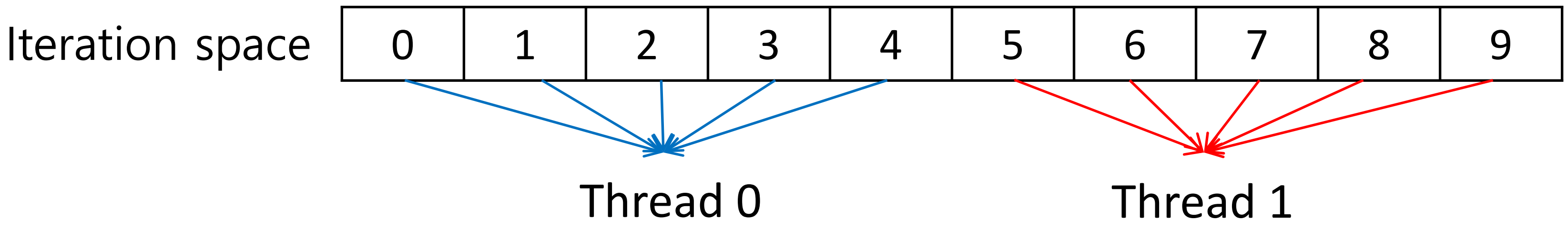
- Is this optimal?
  - **Yes**, when each iteration takes equal time
  - **No**, when each iteration takes different time
    - ex) larger index takes longer time

Assigning Iterations to Threads

- Example of f(i)
  - `f(i)` calls sin function *i* times
  - Time(`f(i)`) is linear to *i*

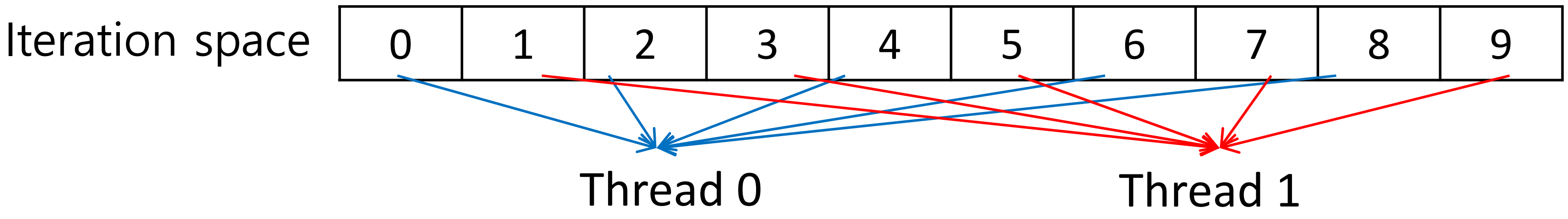
```
float A[100][100];
for(int i = 0; i < 100; i++) {
        for(int j = 0; j < i; j++) {
                A[i][j] = 1.0f;
        }
}
```

- Block partitioning



Iteration space	0	1	2	3	4	5	6	7	8	9

- Cyclic partitioning



Iteration space	0	1	2	3	4	5	6	7	8	9

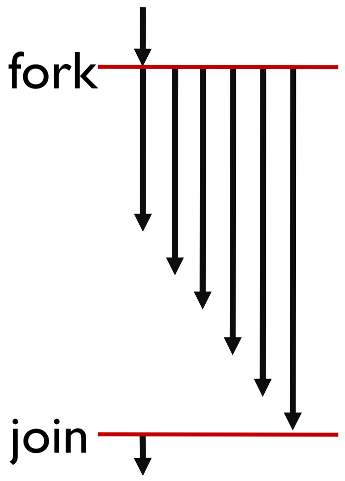
Assigning Iterations to Threads

- Example of `f(i)`
  - `f(i)` calls sin function *i* times
  - Time(`f(i)`) is linear to *i*
  - n = 10,000

```
float A[100][100];
for(int i = 0; i < 100; i++) {
        for(int j = 0; j < i; j++) {
              A[i][j] = 1.0f;
        }
}
```

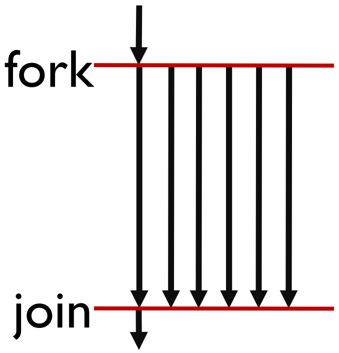
	One thread	Two threads	
		Block partitioning	Cyclic partitioning
Run-time	3.67s	2.77s	1.84s
Speed-up	1x	1.33x	1.99x

- Scheduling (load balancing) is important



Block partitioning

vs.



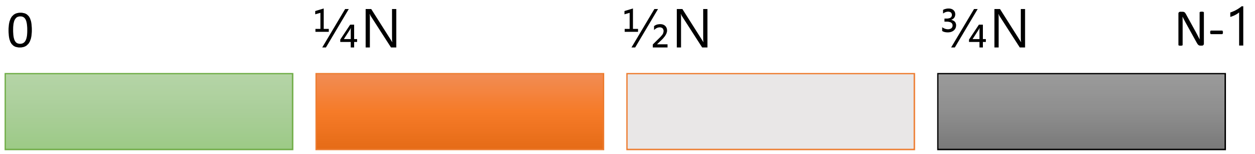
Cyclic partitioning

Schedule Clause

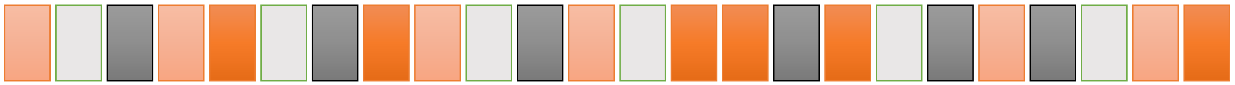
- Format
  - `#pragma omp parallel for schedule(type [, chunk])`
    - *type* := static, dynamic, guided, runtime
    - *chunk* := positive integer (# of iterations)
  - **static**
    - Divide iterations by *chunk* (near equal in size by default)
    - Statically assign threads in a round-robin fashion
  - **dynamic**
    - Divide iterations by *chunk* (1 by default)
    - Dynamically assign a chunk to an idle thread (master/worker)
  - **guided**
    - Chunk size is reduced in an exponentially decreasing manner
    - Dynamically assign a chunk to an idle thread (master/worker)
    - Minimum chunk size is specified by *chunk* (1 by default)
  - **runtime**
    - Determined at runtime with OMP_SCHEDULE environment variable
      - E.g., export OMP_SCHEDULE=“dynamic,5”

Schedule Clause (Illustration)

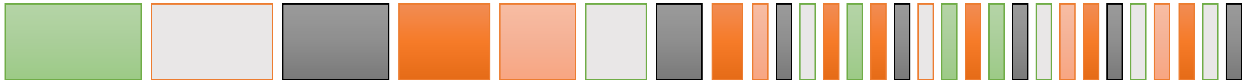
- Dividing iteration space
  - **Static** schedule on iteration space



  - **Dynamic** schedule on iteration space (master/worker)



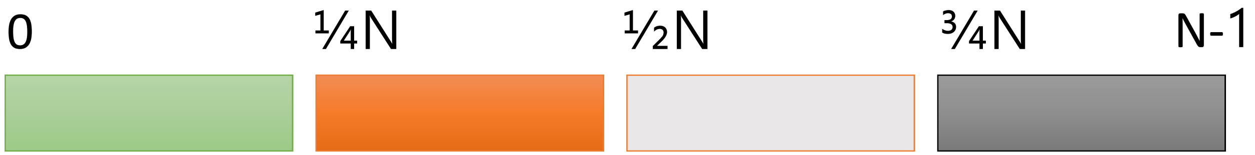
  - **Guided** schedule on iteration space (master/worker)



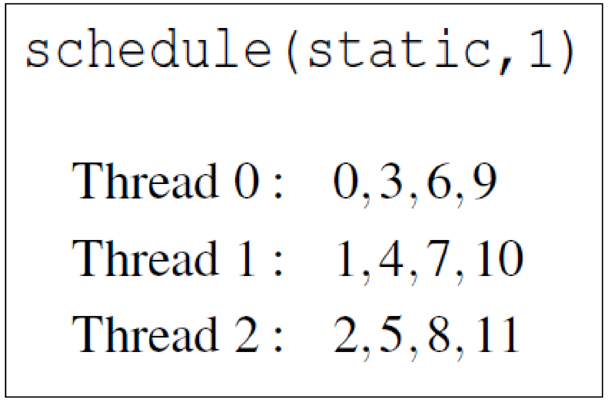
The Static Schedule Type

```
#pragma omp parallel for schedule(static, chunksize)
```

- Static schedule on iteration space



- Example
  - 12 iterations(0, 1, . . . , 11) and 3 threads

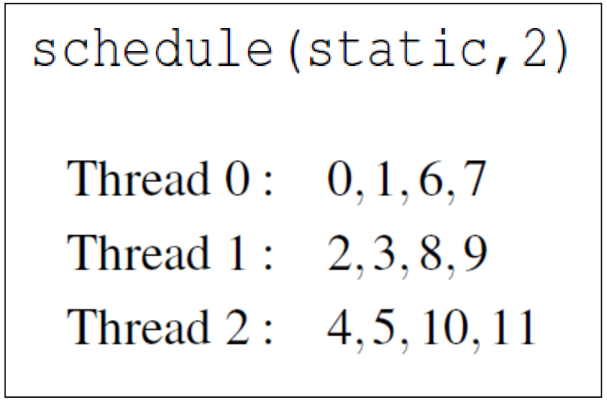


schedule(static,1)

Thread 0 : 0,3,6,9

Thread 1 : 1,4,7,10

Thread 2 : 2,5,8,11

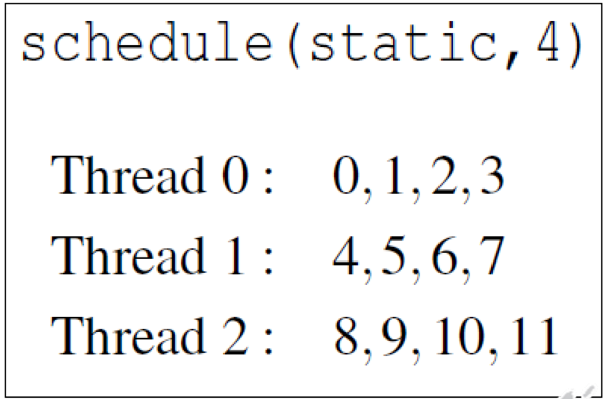


schedule(static,2)

Thread 0 : 0,1,6,7

Thread 1 : 2,3,8,9

Thread 2 : 4,5,10,11



schedule(static,4)

Thread 0 : 0,1,2,3

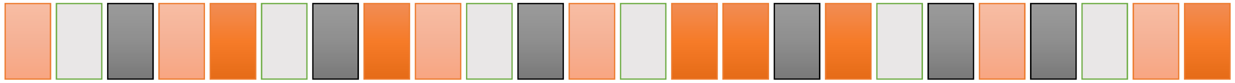
Thread 1 : 4,5,6,7

Thread 2 : 8,9,10,11

The Dynamic Schedule Type

```
#pragma omp parallel for schedule(dynamic, chunksize)
```

- **Dynamic** schedule on iteration space (master/worker)

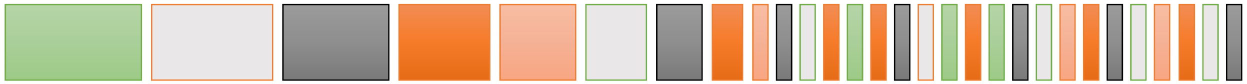


  - The iterations are broken up into chunks of chunksize consecutive iterations
  - Each thread executes a chunk
  - When a thread finishes a chunk, it requests another one from the run-time system
  - This continues until all the iterations are completed
  - The chunksize is 1 by default

The Guided Schedule Type

```
#pragma omp parallel for schedule(guided, chunksize)
```

- **Guided** schedule on iteration space (master/worker)

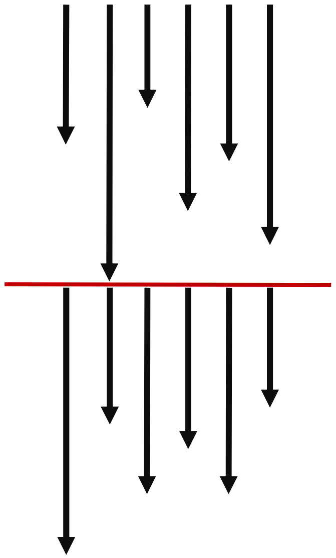


  - Initial size of chunk is smaller than # of iterations / # of threads
  - Each thread executes a chunk, and when a thread finishes a chunk, it requests another one from runtime system
  - The size of the new chunk decreases exponentially
  - If no chunksize is specified, the size of the chunks decreases down to 1
  - If chunksize is specified, it decreases down to chunksize
    - The very last chunk can be smaller than chunksize

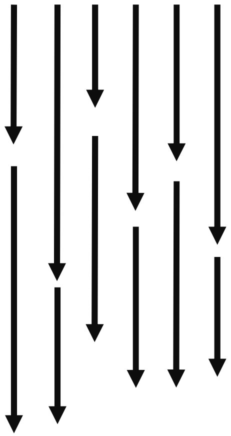
nowait Clause

- Worksharing for loops end with an implicit barrier
- Often, less synchronization is appropriate
  - When a series of for directives are in a parallel construct
- nowait clause
  - Used with a for directive
  - Avoids implicit barrier at the end of for

```
#pragma omp for
for ( …)
#pragma omp for
for (…)
```



```
#pragma omp for nowait
for ( …)
#pragma omp for
for (…)
```



sections Directive

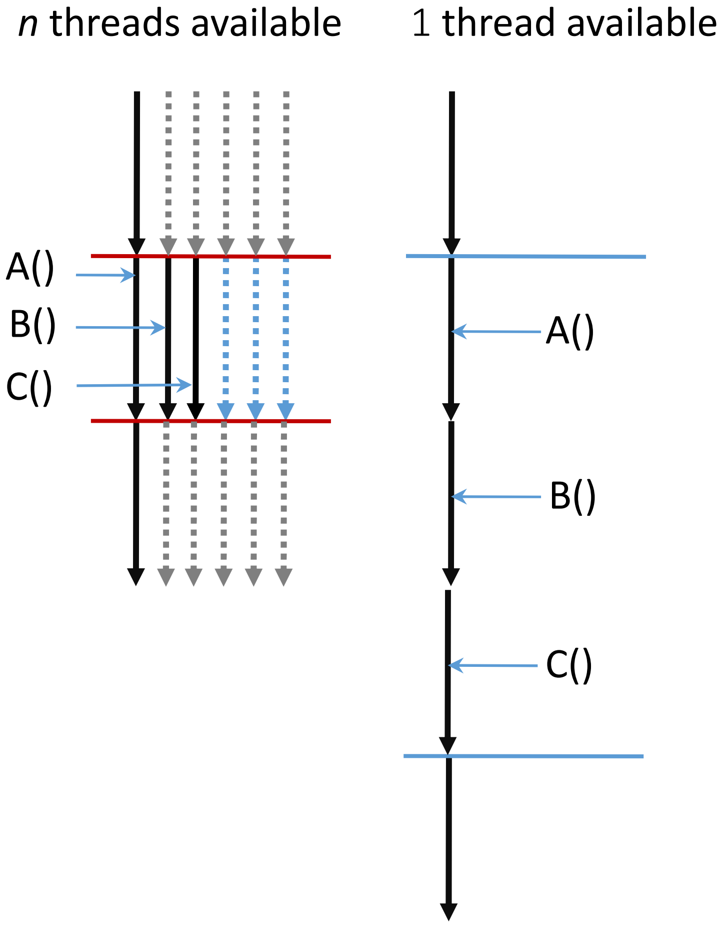
- Non-iterative parallel task assignment using the sections directive.
- General form

```
#pragma omp sections [clause list]
{
    #pragma omp section
        /* structured block */

    #pragma omp section
        /* structured block */


    ...
}
```

- Possible clauses
  - `nowait, shared, private`, …
- Implicit barrier at the end of sections

sections Directive: Example

```
#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
        {
            taskA();
        }
        #pragma omp section
        {
            taskB();
        }
        #pragma omp section
        {
            taskC();
        }
    }
}
```



Merging Directives (1)

```
#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
        {
            taskA();
        }
        #pragma omp section
        {
            taskB();
        }
        #pragma omp section
        {
            taskC();
        }
    }
}
```

→

```
#pragma omp parallel sections
{
    #pragma omp section
    {
        taskA();
    }
    #pragma omp section
    {
        taskB();
    }
    #pragma omp section
    {
        taskC();
    }
}
```

Merging Directives (2)

```
#pragma omp parallel
{
   #pragma omp for
      for (i = 0; i < mmax; i++)
         /* body of loop */
}
```

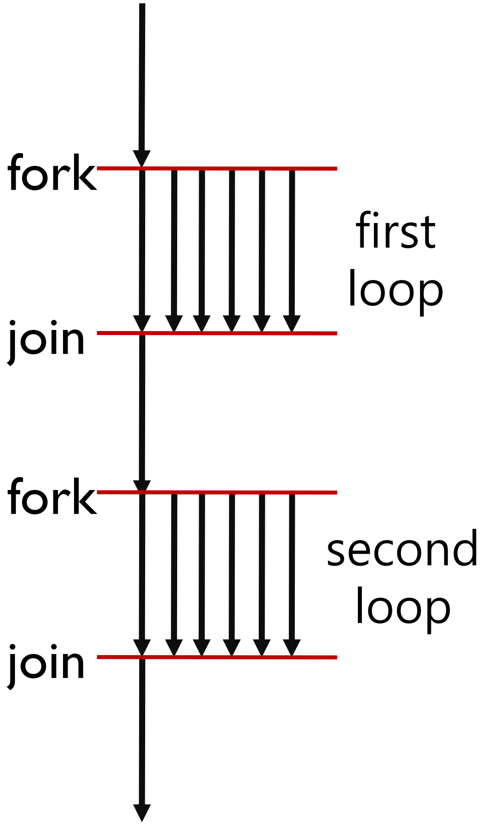
↓

```
#pragma omp parallel for
{
  for (i = 0; i < mmax; i++)
         /* body of loop */
}
```

Caution for Merging Directives (1)

- Each parallel directive forks threads
- Then, join threads after the parallel construct

```
#pragma omp parallel for
for (i=0; i<n; ++i) {
    …
}
#pragma omp parallel for
for (i=0; i<n; ++i) {
    …
}
```



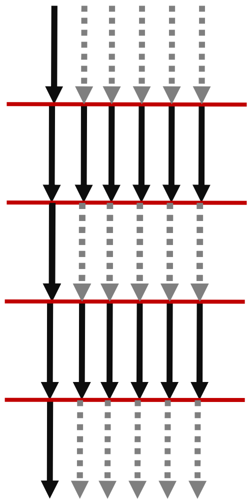
Caution for Merging Directives (2)

- Parallelize a loop using threads that are forked in advance

```
#pragma omp parallel num_threads(n)
{
    #pragma omp for
    for (i=0; i<n; ++i) {
        …
    }
    #pragma omp for
    for (i=0; i<n; ++i) {
        …
    }
}
```

fork threads for the following structured block

parallelize the following for loop using the pre-forked threads



Thread pool

Unnecessary fork&join is eliminated

Nesting `parallel` Directives

- Nested parallelism can be enabled using the `OMP_NESTED` environment variable.
  - If the `OMP_NESTED` environment variable is set to `TRUE`, nested parallelism is enabled.
  - In this case, each parallel directive creates a new team of threads.