Parallel Programming for shared memory machine

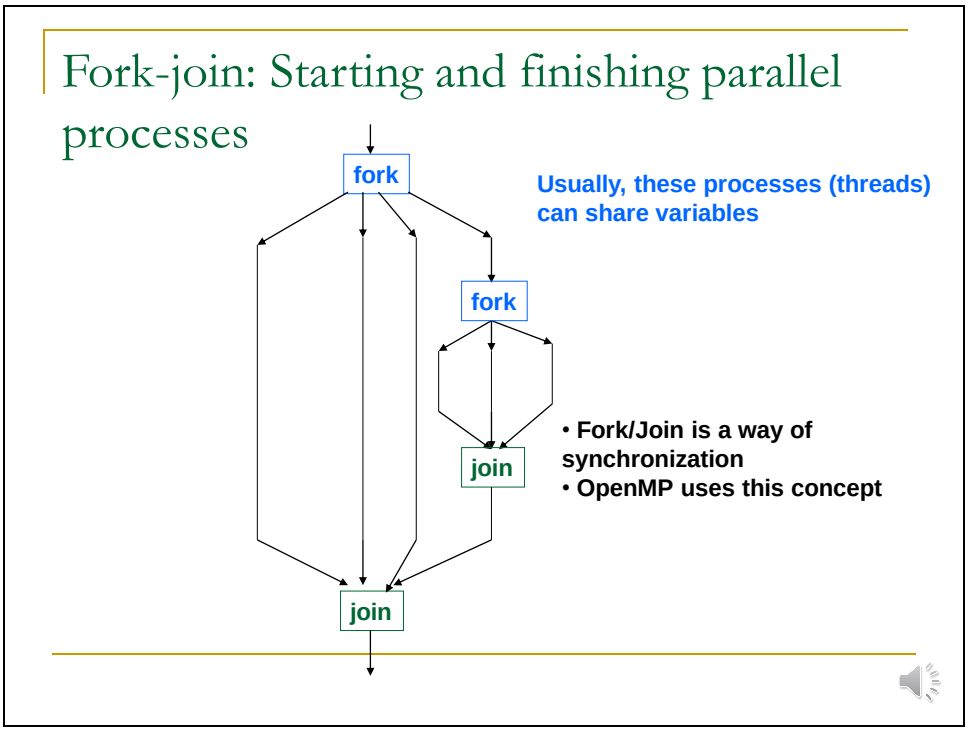
AMANO, Hideharu

Textbook pp.140－147

Parallel programming for various architectures

- UMA or NUMA with relatively small number of nodes
  - OpenMP → Today
- Cluster computer without shared memory
  - MPI → Maybe Later
- GPU
  - Cuda or OpenCL → Contest

In this class, we are going to have three types parallel programming. The first is OpenMP which is used for UMA or NUMA with relatively small number of nodes.



Let me review the fork-join parallel programming paradime. Usually, a single process starts, and when it executes fork operation to generate multiple processes. Some child-process can execute fork again. After executing in parallel, all processes execute join operation. At that time, processes except for only a process which executes the fork operation are terminated. When all processes are terminated with the join operation, the total program is finished. This join operation is a kind of synchronization. For example, OpenMP which I will explain here uses this method.

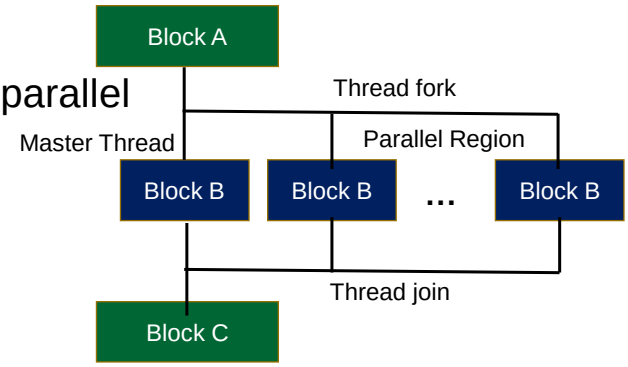
OpenMP

- Standard directives, library and environmental variables for parallelize a program.
- Shared memory is assumed, thus no data distribution is needed. ↔ MPI
- Suitable for multi-core systems within eight threads.
- For a large scale system, advanced optimization of a program is needed (by Prof. Katagiri)

OpenMP is not a language, but standard directives, library and environmental variables for parallelize a program. Shared memory is assumed and it is suitable for small systems.

The execution model of OpenMP

```
Block A
#pragma omp parallel
{
        Block B
}
Block C
```



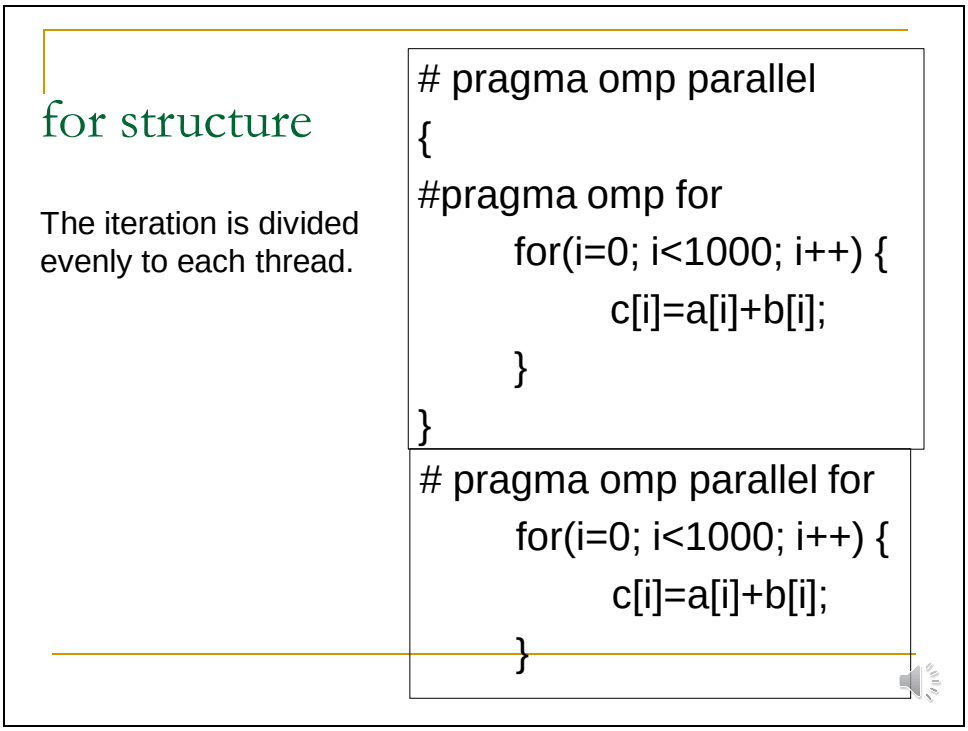
Environmental variable: OMP_NUM_THREADS represents the number of threads

This is the execution mode of OpenMP, when the directive omp parallel is used, the block B in the program structure is forked and the threads are executed in parallel. After finished all threads, the join operation is executed.

Work sharing structure

- Describe the parallel execution in the parallel region. (Used in the parallel structure)
  - for (do)
  - sections
  - single (master)
- Generate and execute
  - parallel for
  - parallel section

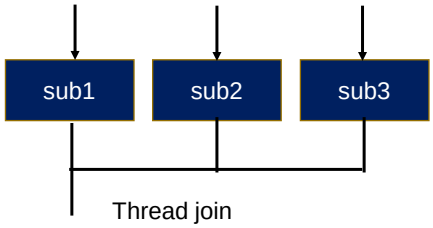
In OpenMP, the programmer must describe the structure which is executed in parallel. They are specified with for, sections and single. The generation and execution can be specified in one pragma. parallel for and parallel sections are examples.



This example shows for structure. The upper shows the standard format, while the lower uses combined format omp parallel for. First, the array elements are distributed into threads, and add operation is executed in parallel. The number of array elements executed in a thread is automatically fixed according to the number of thread which is specified by the environmental variable.

sections structure

```
#pragma omp parallel sections
{
#pragma omp section
   sub1();
#pragma omp section
   sub2();
#pragma omp section
   sub3();
}
```



Forked threads can execute completely different program block. In this example, three subroutines are executed in parallel.

private sub-directive

```
c=….;
# pragma omp parallel for firstprivate(c)
        for(i=0; i<1000; i++) {
                d[i]=a[i]+c*b[i];
        }
```

c is copied to each thread → Performance is improved.

shared: default, shared by all threads
private: variables are provided by each thread without initializing
firstprivate: private with initializing.

The directive sometimes accompanies the sub-directive. This example shows a private sub-directive called firstprivate. It copies the value in the variable, in this case ‘c’, and copied it to all threads. So, performance will be improved.

How to use private

```
# pragma omp parallel for private(j)
        for(i=0; i<100; i++) {
                for(j=0; j<100; j++)
                        a[i]=a[i]+amat[i][j]*b[j];
        }
```

Without private, j is updated by multiple threads→ Error!

private sub-directive is sometimes mandatory. In this case, if j is shared, it is updated by multiple threads and will cause the error. By coping the variable to each thread, this situation can be avoided.

reduction sub-directive

```
# pragma omp parallel for reduction(+:ddot)
        for(i=0; i<100; i++) {
            ddot+= a[i]*b[i];
        }
```

Without reduction directive, the result is not consistent.

reduction calculation is sometimes used in numerical computing. It applies an operation to all elements of an array so that the size of the array is reduced. This operation can be executed in parallel, but describing it is somehow bothering. This sub-directive solves it.

Functions

- omp_get_num_threads();
  - Getting the total number of threads.
- omp_get_thread_num();
  - Getting my thread number.
- omp_get_max_threads();
  - Getting the maximum number of threads.
- Usage:

```
#include <omp.h>
int nth, myid;
nth = omp_get_num_threads();
myid = omp_get_thread_num();
```

They are functions used for checking the number of threads or getting identifier of the thread. The thread identifier is sometimes used when it works different tasks depending the thread identifier.

Getting time: omp_get_wtime();

```
#include <omp.h>
double ts, te;
ts = omp_get_wtime();

   Processing

te = omp_get_wtime();
printf("time[sec]:%lf¥n",te-ts);
```

In order to evaluate the execution time, they are used.

Other directives

- single:

  #pragma omp single

  { blocks..... }

  Assign blocks into a single thread
- master:

  #pragma omp master

  { blocks..... }

  Assign blocks into the master thread

There are other directives, but I have no experience to use them.

Using OpenMP

- login to the ITC Linux machine
  - If you use windows 10, open command prompt
    ssh login_name@XXXX.educ.cc.keio.ac.jp
- Get the compressed file:
  - wget http://www.am.ics.keio.ac.jp/arc/open20.tar
  - tar xvf open20.tar
  - cd open

Then, let's use OpenMP. First, you should login ITC Linux machines and get the tar file.

Compile and Execution

% gcc –fopenmp hello.c –o hello

%./hello

Hello OpenMP world from 1 of 4

….

Here, the number of the thread number is set to be 4.

You can change it by setting OMP_NUM_THREADS from the command line.

Example:

$export OMP_NUM_THREADS=2

./hello

OK. So, lets, compile and try to execute the OpenMP. First, we will execute the simplest example HelloWorld. gcc can be used to compile it. The number of the maximum threads is controlled by the environmental variable OMP_NUM_THREADS. You can specify more number than physically existing cores, but of course, the performance is never improved.

reduct4k.c

An example of reduction calculation.

Compile and try to execute by changing the number of threads.

You can see the execution time is slightly changed in each execution.

→ Don’t care about it too much.

The second example is reduct4k.c, a relatively practical one.

Exercise fft.c

- **Fast Fourier Transform is a famous program for signal processing.**
- **fft.c is a sample program.**
- **If it works well, it shows the execution time, otherwise it fails.**
- **Write the openMP pragma to improve the performance.**

Today's exercise is fft.c.

Report

- Submit the followings:
  - OpenMP C source code
  - The execution results: find the number of threads which minimizes the execution time.
  - Report the number of thread and execution time.
- Submit to Keio.jp, not to hunga4125@gmail.com.

Please hand-off your report to keio.jp. Thank you.