Parallel Scientific Computation

OpenMP 1

J.-H. Parq
IPCST
Seoul National University

What is OpenMP?

- Open Multi-Processing
  - API(application programming interface) for **shared memory multi-processing**
  - Main function: **Multithreading**
  - Supported languages: Fortran, C, C++
  - Supported platforms: Linux, Windows, macOS, Solaris, AIX, HP-UX, ……
  - Managed by OpenMP ARB (Architecture Review Board)

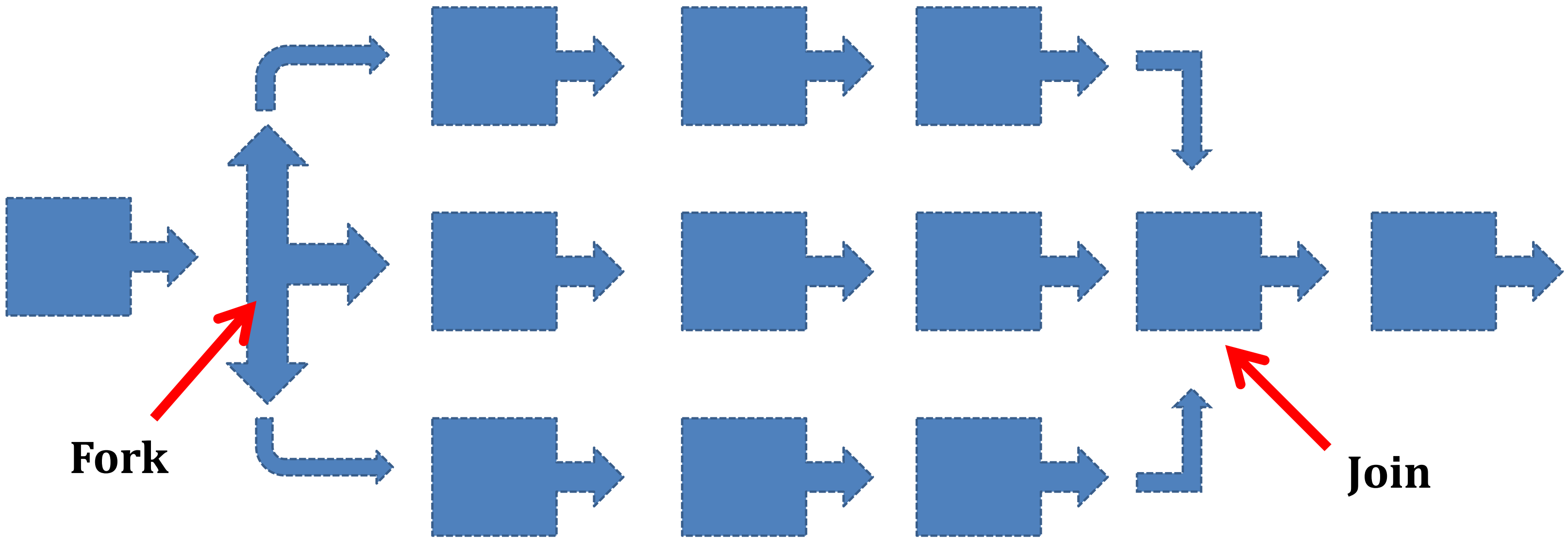
Multithreading

- Thread
  - The smallest unit of a sequence of instructions controlled by a scheduler
- In OpenMP
  - Multi-threads can exist in one process.
    - Threads share resources of the process.
  - Threads can perform concurrent execution.
  - Threads can own their private variables.

❖ Some CPUs provide hyper-threading.

Fork and Join Model

- Branching and merging
  - Fork: A master thread generates slave threads.
  - Join: Slave threads join into the master.



Components

- Compiler directives
- Runtime library routines
- Environment variables

Directives

- Commands meaningful only when the compiler is on the OpenMP mode.
  - The compiler ignores them otherwise.
  - In Fortran: the **comment** form
  - In C or C++: the **pragma** form

Directives

Fortran

- Allowed formats (sentinels)

  !$omp ……

  c$omp ……

  *$omp ……

C / C++

- Allowed formats (sentinels)

  #pragma omp ……

Directives

Fortran

- Example

```
!$omp parallel do
do I = 1, 60
    A(I) = A(I)*X(I)
end do
!$omp end parallel do
```

C / C++

- Example

```
#pragma omp parallel for
for (i = 0; i<60; i++) {
    a(i) = a(i)*x(i)
}
```

Directives

- Frequently used keywords
  - parallel (+ end)
  - do/for
  - private
  - shared
  - section(s)

Library Routines

- Functions or subroutines used in codes
- Frequently used ones
  - omp_set_num_threads
  - omp_get_num_threads
  - omp_get_thread_num
  - omp_get_num_procs
  - omp_get_max_threads

External Variables

- Variables used on OS (not in your codes)
- Used by the command setnev(csh/tcsh) or export(sh/bash)
  - Ex.) export OMP_NUM_THREADS=8
- Frequently used variables
  - OMP_NUM_THREADS
  - OMP_SCHEDULE

OMP_THREAD_LIMIT

How to Compile

- Intel

  icc –qopenmp …… icpc –qopenmp ……

  ifort –qopenmp ……

  ❖ Alternatively, -openmp option can be used although it is deprecated.

- GNU

  gcc –fopenmp …… g++ –fopenmp ……

  gfortran –fopenmp ……

- PGI

  pgcc –mp ……

  pgf90 –mp ……

Two-way Compiling

- How to enable either serial or parallel?
  - C
    - Use of a fake openmp header for serial compiling
    - Or _OPENMP macro

    Ex.)
    ```
    #ifdef _OPENMP
    id = omp_get_thread_num();
    #endif
    ```
  - Fortran
    - !$ or C$ or c$ or *$

    Ex.)
    ```
    !$   id = omp_get_thread_num()
    ```

Simple Example Program

- **hello.c**

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

main()
{
  int id, N;
  float fraction;

  N = 4;
  printf("%d threads set by me\n",N);
  omp_set_num_threads(N);
  printf("%d procs\n", omp_get_num_procs());
  printf("%d max. threads\n", omp_get_max_threads());
  printf("%d thread now\n\n",omp_get_num_threads());
```

Simple Example Program

```
  printf("Fork!\n");
#pragma omp parallel private(id,fraction) shared(N)
  {
    id = omp_get_thread_num();
    printf("Hello, I'm thread %d\n",id);
    fraction = (float)id/(float)N;
    printf("%d/%d = %f\n",id,N,fraction);
  }
  printf("Join!\n");


}
```

Simple Example Program

- **hello.cpp**

```
#include <iostream>
#include <omp.h>
using namespace std;

main() {
  int id, N;
  float fraction;

  N = 4;
  cout << N << " threads set by me" << endl;
  omp_set_num_threads(N);
  cout << omp_get_num_procs() << " procs" << endl;
  cout << omp_get_max_threads() << " max. threads" << endl;
  cout << omp_get_num_threads() << " thread now" << endl;
```

Simple Example Program

```
  cout << "Fork!" << endl;
#pragma omp parallel private(id,fraction) shared(N)
 {
   id = omp_get_thread_num();
   fraction = (float)id/(float)N;

#pragma omp critical (printing)
   {
     cout << "Hello, I'm thread " << id << endl;
     cout << id << "/" << N << " = " << fraction << endl;
   }
 }
 cout << "Join!" << endl;

}
```

Simple Example Program

- **hello.f**

```
PROGRAM HELLO
IMPLICIT NONE
INCLUDE "omp_lib.h"
INTEGER ID, N
REAL FRACTION

N = 4
PRINT '(I2,A)',N," threads set by me"
CALL OMP_SET_NUM_THREADS(N)
PRINT '(I2,A)',OMP_GET_NUM_PROCS ()," procs"
PRINT '(I2,A)',OMP_GET_MAX_THREADS()," max. threads"
PRINT *
PRINT '(I2,A)', OMP_GET_NUM_THREADS() ,  " thread now"
```

Simple Example Program

```
      PRINT '(A)',"Fork!"
C$OMP PARALLEL
C$OMP& PRIVATE(ID,FRACTION) SHARED(N)
      ID = OMP_GET_THREAD_NUM ()
      PRINT '(A,I2)',"Hello, I'm thread",ID
      FRACTION = REAL(ID)/REAL(N)
      PRINT '(I2,A,I1,A,F5.2)',ID,"/",N," = ",FRACTION
C$OMP END PARALLEL

      PRINT '(A)',"Join!"
      STOP
      END
```

Simple Example Program

- **hello.f90**

```
PROGRAM HELLO
 USE OMP_LIB
 IMPLICIT NONE
 INTEGER ID, N
 REAL FRACTION

 N = 4
 PRINT '(I2,A)',N," threads set by me"
 CALL OMP_SET_NUM_THREADS(N)
 PRINT '(I2,A)',OMP_GET_NUM_PROCS ()," procs"
 PRINT '(I2,A)',OMP_GET_MAX_THREADS()," max. threads"
 PRINT *
 PRINT '(I2,A)', OMP_GET_NUM_THREADS() ,  " thread now"
```

Simple Example Program

```
 PRINT '(A)',"Fork!"
!$OMP PARALLEL &
!$OMP PRIVATE(ID,FRACTION) SHARED(N)
  ID = OMP_GET_THREAD_NUM ()
  PRINT '(A,I2)',"Hello, I'm thread",ID
  FRACTION = REAL(ID)/REAL(N)
  PRINT '(I2,A,I1,A,F5.2)',ID,"/",N," = ",FRACTION
!$OMP END PARALLEL

  PRINT '(A)',"Join!"
  STOP
END PROGRAM HELLO
```

Simple Example Program

- **Execution**

g++ hello.cpp –fopenmp –o a.out

./a.out

4 threads set by me

4 procs

4 max. threads

1 thread now

Fork!

Hello, I'm thread 0

0/4 = 0

Hello, I'm thread 1

1/4 = 0.25

Hello, I'm thread 2

2/4 = 0.5

Hello, I'm thread 3

3/4 = 0.75

Join!

Parallel Directive

- Block structure
  - Fork at the start and join at the end
  - Synchronization at the end: implicit barrier

- Fortran (f90)

```
!$omp parallel [clauses]
    ……
!$omp end parallel
```

- C/C++

```
#pragma omp parallel [clauses]
   {
         ……
   }
```

Reduction Clause

- Results of all threads reduce to the master
- Ex.)

```
!$omp parallel reduction(+:b) private(i) shared(a)
   i = omp_get_thread_num()
   b = b + a(i)*a(i)
!$omp end parallel
```

- About the operation
  - Addition(+:…), multiplication(*:…), ……
  - 0(+) or 1(*) for the initial value of the variable which all the results reduce into

Private/Shared Clauses

- Private
  - Every thread has an **individual** variable of the same name.
  - **Different memory location**
- Shared
  - **All** thread share the variable.
  - **Same memory location**
  - *Race condition* can form.
  - Many shared variables reduce performance speed.

Race Condition

- Simultaneous access
  - Hazardous if the result depends on the sequence of processes
- Example

```
!$omp parallel private(i) shared(a,b)
   i = omp_get_thread_num()
   b = b + a(i)*a(i)
!$omp end parallel
```

Reading and writing ‘b’ are problematic!

➢ A race condition can be solved by adding variables or using directives for synchronization.

Critical Directive

- One by one (to avoid conflicts)
  – Only one thread at a time (not in number order but randomly) executes the commands under ‘critical’ directive.

- Fortran (f90)

```
!$omp critical [(name)]
    ……
!$omp end critical [(name)]
```

- C/C++

```
#pragma omp critical [(name)]
    ……
or {
        ……
   }
```

Critical Directive

- Notes
  - Use only if indispensible
    - Total computational time increases.
    - But necessary if race conditions emerge.
  - Naming is important.
    - All ‘critical’ blocks with no name are regarded as the same block.
      - If one thread enter a ‘critical’ block, the other threads **cannot** enter **any** ‘critical’ blocks until the thread comes out of the block.

Atomic Directive

- Similar to 'critical' but only for atomic operations.
- No block structure.
- Faster than 'critical'.

- Fortran (f90)

```
!$omp atomic
    ……
```

- C/C++

```
#pragma omp atomic
    ……
```

Atomic Directive

- Atomic operations in OpenMP

  x++ x-- (for C/C++)

  x 𝕓= *expre* (for C/C++)

  x = x 𝕓 *expre*

  x = *expre* 𝕓 x

  – 𝕓: binary operator (+, -, *, /, ……). Not overloaded.

  – *expre*: expression without x

  ❖ x must be a scalar variable.

Other Synchronization Directives

- Barrier
  - Threads wait until all threads reach the point.
  - Some directives already include implicit barriers and do not need barrier directive.
- Master
  - Only the master executes the command block while the other threads skip it. (No barrier)

References

- C. Evangelinos,

  Parallel Programming for Multicore Machines Using OpenMP and MPI

- B. Barney,

  OpenMP

- Wikipedia