OpenMP

John H. Osorio Ríos

Abstract (1/1)

- OpenMP is an Application Program Interface.
- Provides a portable, scalable model for developers of shared memory parallel applications.
- Supports C/C++ and Fortran.

What is OpenMP? (1/2)

- May be used to explicitly direct multi-threaded, shared memory parallelism.
- API Components:
  - Compiler Directives
  - Runtime Library Routines
  - Environment Variables

What is OpenMP? (2/2)

- An abbreviation for:
  - Short Version: **Open Multi-Processing**
  - Long Version: **Open** Specifications for **Multi-Processing** via collaborative work between interested parties from the hardware and software industry, government and academia.

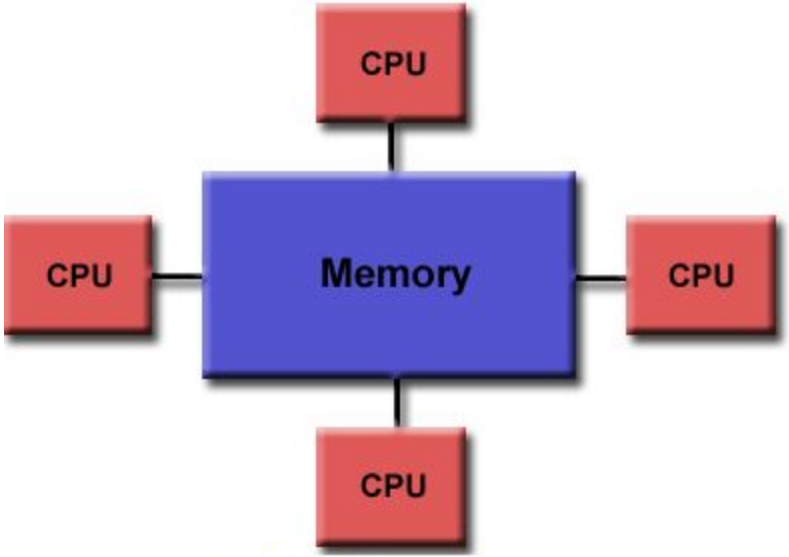
Goals of OpenMP (1/1)

- Standardization
- **Lean and Mean**
- Ease of Use
- Portability

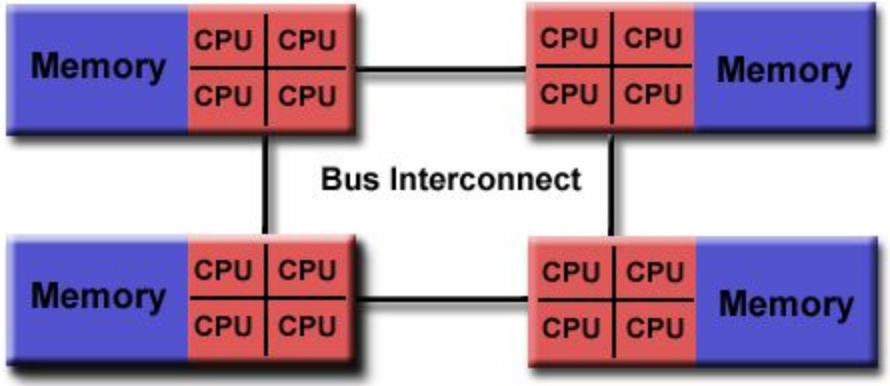
History (1/1)

Month/Year	Version
Oct 1997	Fortran 1.0
Oct 1998	C/C++ 1.0
Nov 1999	Fortran 1.1
Nov 2000	Fortran 2.0
Mar 2002	C/C++ 2.0
May 2005	OpenMP 2.5
May 2008	OpenMP 3.0
Jul 2011	OpenMP 3.1
Jul 2013	OpenMP 4.0

OpenMP Programming Model (1/4)



Uniform Memory Access



Non-Uniform Memory Access

OpenMP Programming Model (2/4)

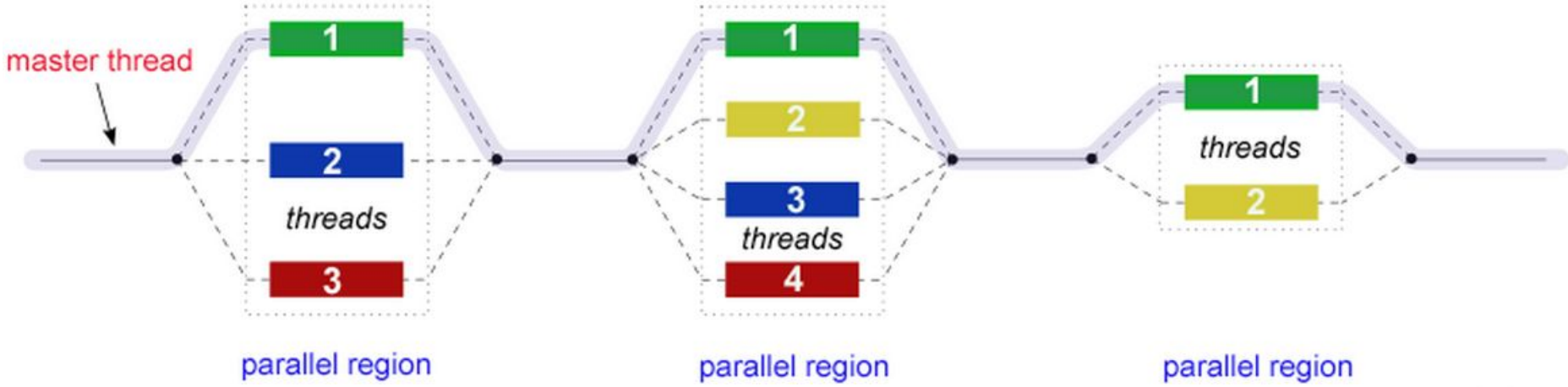
- Thread Based Parallelism:
  - Exclusively through the use of threads
  - A thread is the smallest unit of processing
  - **Number of Threads = Number of Processors**

OpenMP Programming Model (3/4)

- Explicit Parallelism:
  - Isn’t automatic programming model.
  - Offers to the programmer full control
  - Can be as simple as inserting compiler directives …
  - Or as complex as using multiple levels of parallelism, locks, even nested locks

OpenMP Programming Model (4/4)

- Fork - Join Model:



OpenMP API Overview (1/4)

- Compiler Directives:
  - Appear as comments in the source code
  - Need to use a compiler flag
  - Have the following syntax

```
#pragma omp parallel default(shared) private(beta,pi)
```

OpenMP API Overview (2/4)

- Runtime Library Routines:
  - Setting and querying the number of threads
  - Querying a thread's unique identifier
  - Setting and querying the dynamic threads feature

```
#include <omp.h>
int omp_get_num_threads(void)
```

OpenMP API Overview (3/4)

- Environment Variables:
  - Setting the number of threads
  - Binding threads to processors
  - Enabling/disabling dynamic threads

csh/tcsh	`setenv OMP_NUM_THREADS 8`
sh/bash	`export OMP_NUM_THREADS=8`

OpenMP API Overview (4/4)

- OpenMP Code Structure:

```
#include <omp.h>

main ()  {

int var1, var2, var3;

Serial code
      .
      .
      .

Beginning of parallel section. Fork a team of threads.
Specify variable scoping

#pragma omp parallel private(var1, var2) shared(var3)
  {

  Parallel section executed by all threads
            .
  Other OpenMP directives
            .
  Run-time Library calls
            .
  All threads join master thread and disband

  }

Resume serial code
      .
      .
      .

}
```

Compiling OpenMP Programs (1/1)

- gcc test.c -o test **-fopenmp**

OpenMP Directives (1/3)

- Parallel Region Construct:

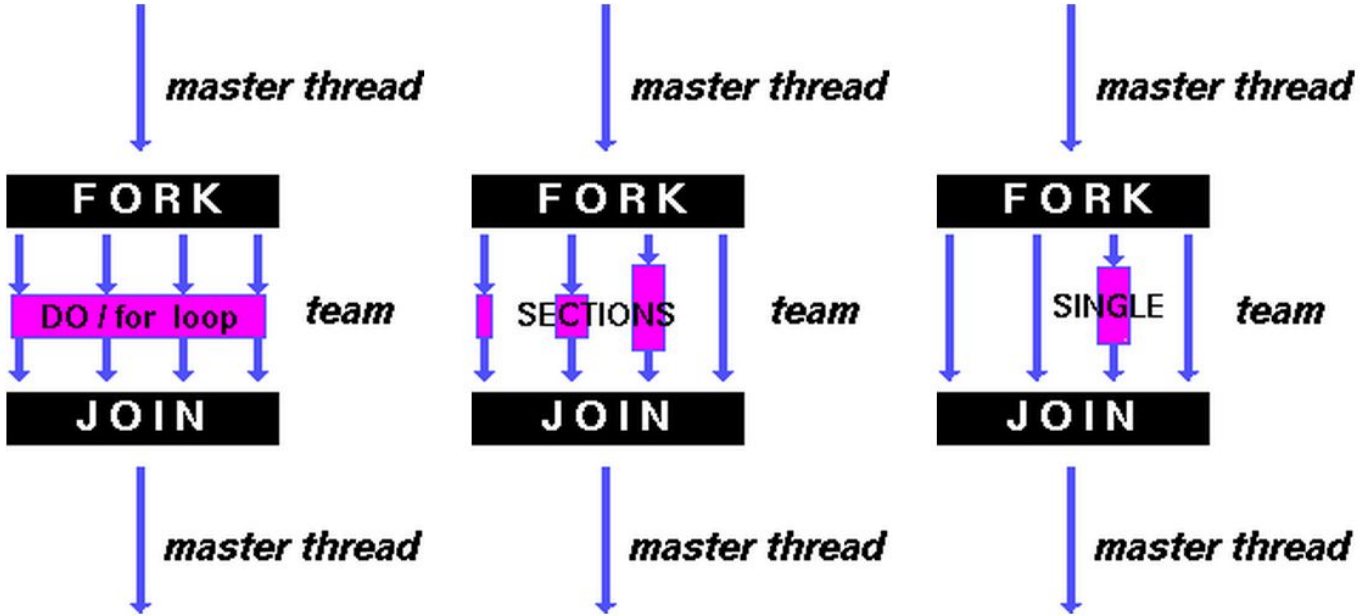
```
#pragma omp parallel [clause ...]  newline
                     if (scalar_expression)
                     private (list)
                     shared (list)
                     default (shared | none)
                     firstprivate (list)
                     reduction (operator: list)
                     copyin (list)
                     num_threads (integer-expression)

   structured_block
```

OpenMP Exercise (1/1)

- Download the source code from here
- Compile the code
- Run the code
- Change **OMP_NUM_THREADS** environment variable and run again

OpenMP Directives (2/3)

- Work-Sharing Constructs:



OpenMP Directives (3/3)

- Work-Sharing Constructs:

```
#pragma omp for [clause ...]  newline
                schedule (type [,chunk])
                ordered
                private (list)
                firstprivate (list)
                lastprivate (list)
                shared (list)
                reduction (operator: list)
                collapse (n)
                nowait

    for_loop
```

OpenMP Exercise (1/1)

- Download the source code from here
- Compile the code
- Run the code

TODO (1/1)

- Implement the matrix multiplication using **openmp**
- Make some test and try to figure out the best number of threads to use.
- Remember to measure the execution time of the program.

Bibliography (1/1)

- http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf
- http://www.openmp.org/mp-documents/OpenMP3.1.pdf
- https://computing.llnl.gov/tutorials/openMP/

THANKS

john@sirius.utp.edu.co