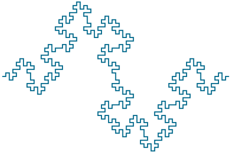
Introduction to Parallel Programing with Share Memory Open Multi-Processing (OpenMP)

Henry R. Moncada



August 20, 2020

Contenido

Intended Learning Outcomes

- Have basic understanding of Parallel programming with OpenMP
- Understand OpenMP core syntax: directives, constructs, parallel region
- Compile and run your first OpenMP code on your laptop or a supercomputer

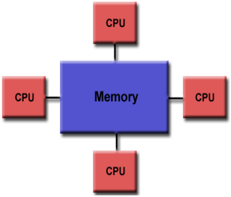
Speed

BACKGROUND

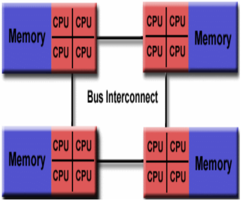
TOTALLY UNRELATED CONCEPTS

OpenMP Defined

OpenMP is a Parallel Programming Model for Shared memory and distributed shared memory multiprocessors. The underlying architecture can be shared memory UMA or NUMA.



Uniform Memory Access (UMA)



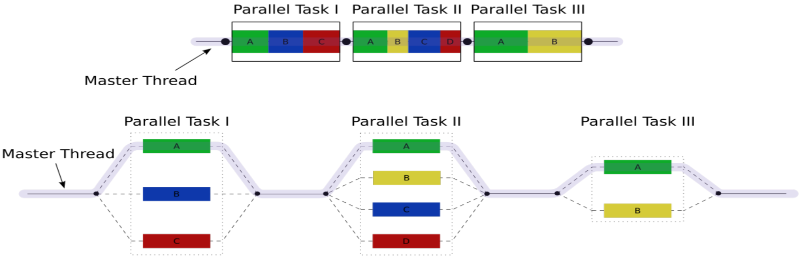
Non-Uniform Memory Access (NUMA)

- OpenMP stand for Open Multi-Processing
- OpenMP is not a computer language.
- OpenMP is an Application Programming Interface (API) that supports multi-platform shared memory multiprocessing programming.
- Works in conjuction with C/C++ or Fortran
- Applications built using hybrid model of parallel programming:
  - Runs on a computer cluster using both OpenMP and Message Passing Interface (MPI)
  - OR through the use of OpenMP extensions for non-shared memory systems.

Execution Model

OpenMP is an implementation of multithreading

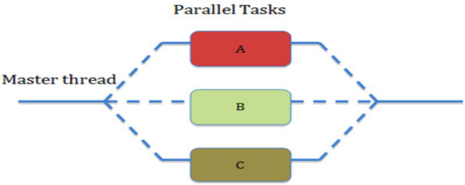
- Program begins execution as a single thread **(Master Thread)**
- Master thread executes in serial until parallel construct encountered
- Team of threads created which execute statements in **parallel region (Task)**
- After parallel region, serial execution resumes with master thread



OpenMP uses the fork-join model of parallel execution

Parallel Region

- The parallel region is the basic parallel construct in OpenMP.
- A parallel region defines a section of a program.
- Program begins execution on a single thread (the master thread).
- When the first parallel region is encountered, the master thread creates a team of threads (fork/join model).
- Every thread executes the statements which are inside the parallel region
- At the end of the parallel region, the master thread waits for the other threads to finish, and continues executing the next statements



Parallel Region

Compiler Directives and OpenMP Directives

A directive is a language construct that instructs the compiler on how to process its input

- Compiler directives
  - Example C/C++ compiler
    - `#define` substitutes a preprocessor macro
    - `#include` inserts the content of a particular file
    - `#pragma` issues special commands to the compiler
  - Examples in Fortran
    - `!DIR$` issues special commands to the compiler
- OpenMP Directives are used to express parallelism in OpenMP
  - In C/C++ directives begin with

```
#pragma omp
```

```
#pragma omp parallel
```

The `omp` keyword signals the pragma as OpenMP specific. Non OpenMP compilers will ignore.

  - In Fortran directives begin with

```
!$omp
c$omp
*$omp
```

```
!$omp parallel
```

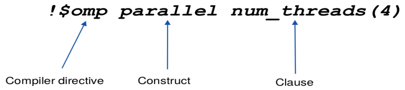
In fixed form, a line beginning with one of the above keywords and containing a space or zero in the sixth column will be treated as an OpenMP directive. It will be treated as a comment by non-OpenMP compilers.

OpenMP Construct, Clauses and Parallel Region

- A **construct** is a specific OpenMP executable directive
- OpenMP directives may include various **clauses** to provide further information on the expected behaviour of the OpenMP implementation

```
!$omp construct [clause [clause ]...]
```

Example Core Syntax in Fortran



!$omp parallel num_threads(4)

- A **parallel region** is a region executed by all threads
  - Default storage attributes are defined by a data environment (we come back to this later)
- In C/C++ a **parallel region** is included between { . . . } after a directive
- In Fortran a **parallel region** is included between a directive pair

```
#pragma omp parallel
{

 ...

}
```

```
!$omp parallel

 ...

!$omp end parallel
```

OpenMP programs require including a header file

OpenMP programs require including a header file:

Language	Header Files
Fortran 77	`INCLUDE 'omp_lib.h'`
Fortran 90	`use omp_lib`
Fortran 95	`use omp_lib`
C	`#include <omp.h>`
C++	`#include <omp.h>`

QUESTION

Which thread is going to execute the subroutine hello() ?

```fortran
 program hello_world
   use omp_lib
   implicit none

!$omp parallel
   call  hello(omp_get_thread_num()) ! obtain thread number
!$omp end parallel

end program
```

```fortran
subroutine hello(tid)
   integer  tid
   write (*,*)  'hello  world  !!!,  from thread = ',  tid
end subroutine
```

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

int  hello(int);

int  main() {
     #pragma omp parallel
     {
          hello(omp_get_thread_num());// obtain thread
                         number
     }
     return  0;
}
```

```c
int  hello(int  tid){
     printf(  "Hello,  World  !!!,  From thread %d!\n", tid);
}
```

QUESTION

Which thread is going to execute the subroutine hello() ?

```fortran
 program hello_world
   use omp_lib
   implicit  none

!$omp parallel
    call  hello(omp_get_thread_num()) ! obtain thread number
!$omp end parallel

end program
```

```fortran
subroutine hello(tid)
   integer  tid
   write (*,*)  'hello  world  !!!,  from thread = ',  tid
end subroutine
```

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

int  hello(int);

int  main() {
     #pragma omp parallel
     {
          hello(omp_get_thread_num());// obtain thread
                        number
     }
     return  0;
}
```

```c
int  hello(int  tid){
     printf( "Hello, World  !!!,  From thread %d!\n", tid);
}
```

Answer : The directives

- Fortran : `!$omp parallel` and `!$omp end parallel`
- C : `#pragma omp parallel {... }`

creates a section of code that is run from all available threads.

OpenMP Library Functions

- OpenMP function prototypes and types are defined in
  - C/C++

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

  - Fortran

    ```
    use omp_lib
    ```

- Some of the library functions we will use to:
  - Modify/check the number of threads:

    ```
    omp_set_num_threads(), omp_get_num_threads(), omp_get_thread_num(), omp_get_max_threads()
    ```

  - Check if we are in active parallel region:

    ```
    omp_in_parallel()
    ```

  - Dynamically vary the number of threads

    ```
    omp_set_dynamic, omp_get_dynamic()
    ```

  - Check the number of processors in the system

    ```
    omp_num_procs()
    ```

OpenMP Environment Variables

Important environment variables tobe used when you compiled code with OpenMP

- Set the number of threads using the environment variable `OMP_NUM_THREADS`.
  - For the `csh` or `tcsh` shell, enter:

    ```
    set OMP_NUM_THREADS= <number of threads to use>
    ```

  - For the `bash` shell, enter:

    ```
    export OMP_NUM_THREADS= <number of threads to use>
    ```

  - When executing a program (a.out)

    ```
    env OMP_NUM_THREADS=<number of threads to use> ./a.out
    ```

- Process binding is enabled if this variable is true · · · i.e. if true the runtime will not move threads around between processors.

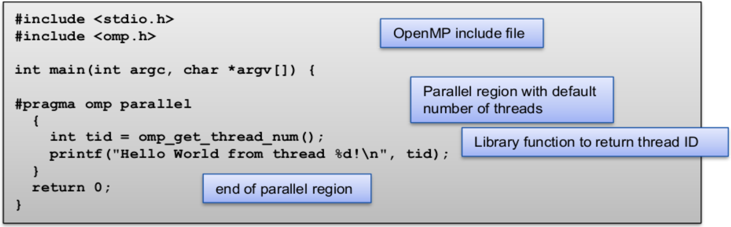
  ```
  OMP_PROC_BIND true | false
  ```

- Control how “omp for schedule(RUNTIME)” loop iterations are scheduled

  ```
  OMP_SCHEDULE schedule [chunk_size]
  ```

OPENMP HELLO WORLD

EXERCISE 1: OUR FIRST OPENMP PROGRAM

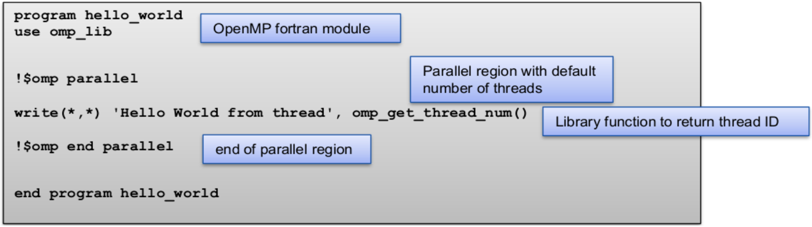


```
#include <stdio.h>
#include <omp.h>

int main(int argc, char *argv[]) {

#pragma omp parallel
  {
    int tid = omp_get_thread_num();
    printf("Hello World from thread %d!\n", tid);
  }
  return 0;
}
```

OpenMP include file
Parallel region with default number of threads
Library function to return thread ID
end of parallel region



```
program hello_world
use omp_lib


!$omp parallel

write(*,*) 'Hello World from thread', omp_get_thread_num()

!$omp end parallel


end program hello_world
```

OpenMP fortran module
Parallel region with default number of threads
Library function to return thread ID
end of parallel region

OpenMP Hello World (Fortran/C)

Hello_World.f90

```
 program hello_world
! Include OpenMP header file, invoke openmp functionality.
  use omp_lib
  implicit none
  integer nthreads, thead_id

! Fork a team of threads giving them their own copies of
          variables

!$omp parallel private(nthreads, thead_id)

   thead_id = omp_get_thread_num() ! obtain thread number
   print *, ’hello world from thread = ’, thead_id

   if (thead_id .eq. 0) then     ! only master thread does this
        nthreads = omp_get_num_threads()
        print *, ’number of threads = ’, nthreads
   end if
!$omp end parallel

! All threads join master thread and disband
end program
```

with GNU/GCC compiler

```
$ gfortran −fopenmp −o out omp_hello.f90
```

```
$ export OMP_NUM_THREADS=16
$ ./out
```

Hello_World.c

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

int main (int argc, char *argv[])  {
  int nthreads, thead_id;

/* Fork a team of threads giving them their own copies of
          variables */
  #pragma omp parallel private(nthreads, thead_id)
  {
  /* Obtain thread number */
     thead_id = omp_get_thread_num();
     printf(”Hello World from thread = %d\n”, thead_id);

  /* Only master thread does this */
     if (thead_id == 0) {
       nthreads = omp_get_num_threads();
       printf(”Number of threads = %d\n”, nthreads);
     }
   } /* All threads join master thread and disband */
  return 0;
}
```

with GNU/GCC compiler

```
$ gcc −fopenmp −o out omp_hello.c
```

with LLVN/CLANG compiler

```
$ clang −fopenmp −o out omp_hello.c
```

Compile with Fortran

- ▶ GNU (gfortran)

```
$ gfortran —fopenmp —o out omp_hello.f90
```

- ▶ Intel (ifort)

```
$ ifort —openmp —o out omp_hello.f90
```

- ▶ Portland Group (pgf77,pgf90)

```
$ pg90 —mp —o out omp_hello.f90
```

Execute

```
$ export OMP_NUM_THREADS=16
$ ./out
```

Compile with C/C++

- ▶ GNU (gcc, g++)

```
$ gcc —fopenmp —o out omp_hello.c
```

- ▶ Intel (icc)

```
$ icc —openmp —o out omp_hello.c
```

- ▶ Portland Group (pgcc,pgCC)

```
$ pgcc —mp —o out omp_hello.c
```

- ▶ LLVM

```
$ clang —fopenmp —o out omp_hello.c
```

Execute

```
$ export OMP_NUM_THREADS=8
$ ./out
```

END

QUESTIONS

REFERENCES

- OpenMP specifications:

  http://www.openmp.org/specifications/

- OpenMP summary Card:
  - C:

    http://www.openmp.org/wp − content/uploads/OpenMP − 4.0 − C.pdf

  - Fortran:

    http://www.openmp.org/wp − content/uploads/OpenMP − 4.0 − Fortran.pdf