RV College of Engineering®

Department of Computer Science & Engineering

OpenMP, MPI

Dr. Minal Moharir

Go, change the world®

What is OpenMP?

De-facto standard API for writing shared memory parallel applications in C, C++, and Fortran

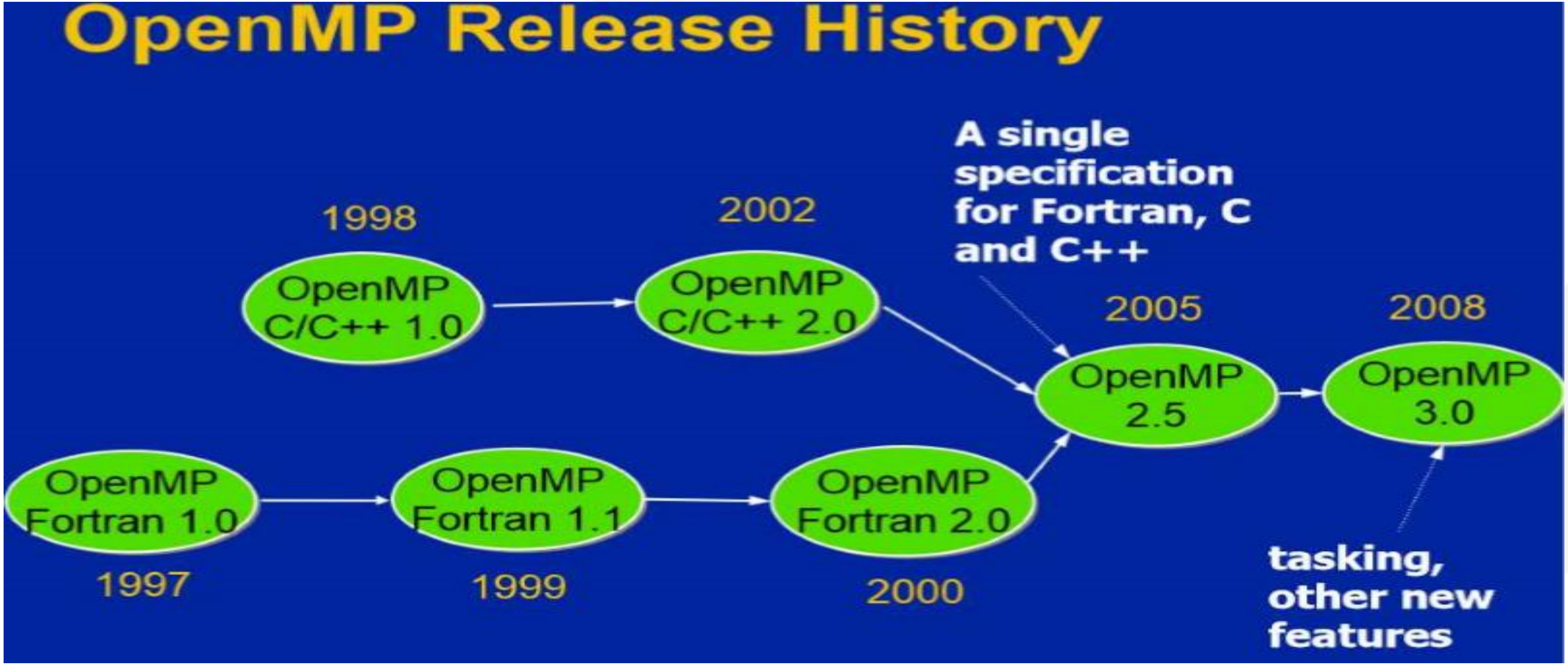
Consists of:

1. Compiler Directives
2. Run time Libraries
3. Environment Variables

Specifications are mentationed by OpenMp Architecture Review Board(http://www.openmp.org)

Version 4.5 has been released by in Nov. 2015

OpenMP?



OpenMP Release History

1998

OpenMP C/C++ 1.0

2002

OpenMP C/C++ 2.0

A single specification for Fortran, C and C++

2005

OpenMP 2.5

2008

OpenMP 3.0

OpenMP Fortran 1.0

1997

OpenMP Fortran 1.1

1999

OpenMP Fortran 2.0

2000

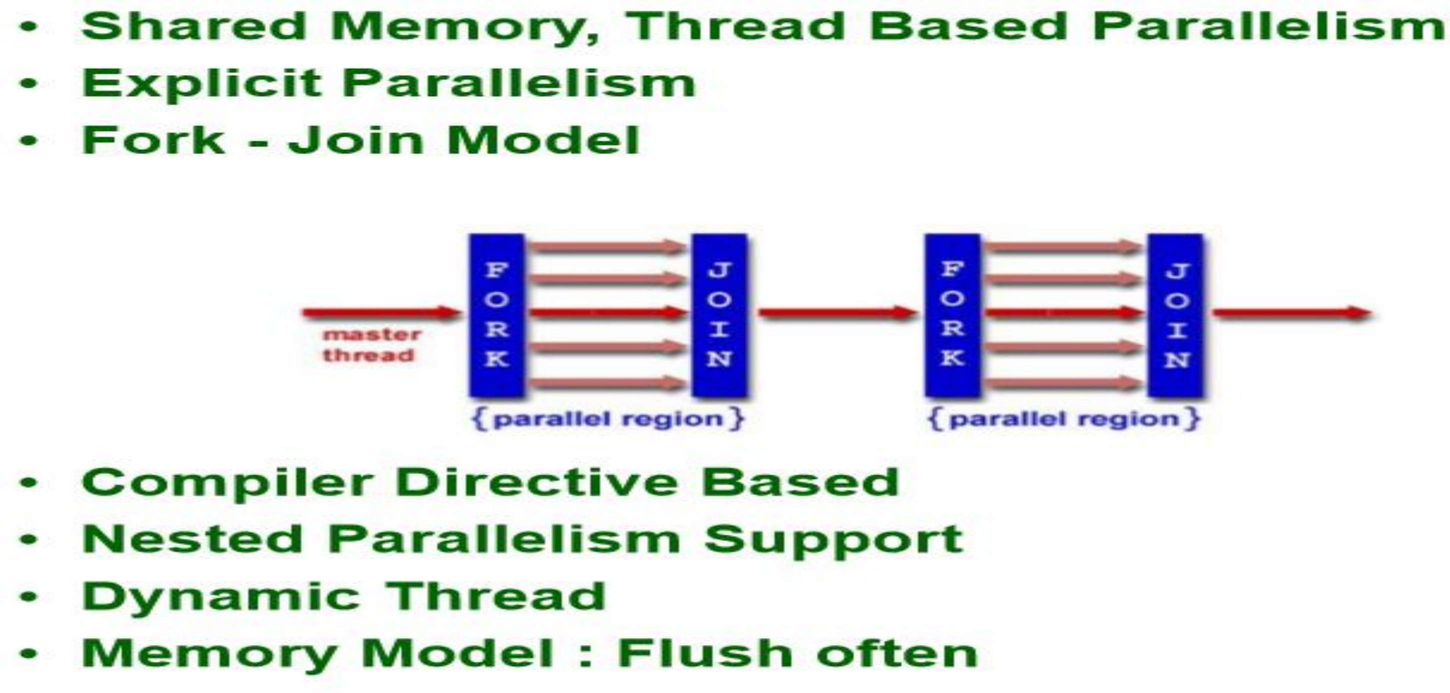
tasking, other new features

When to Consider

When Compiler cannot find parallelism

Use explicit PARALLELIZATION-OpenMp

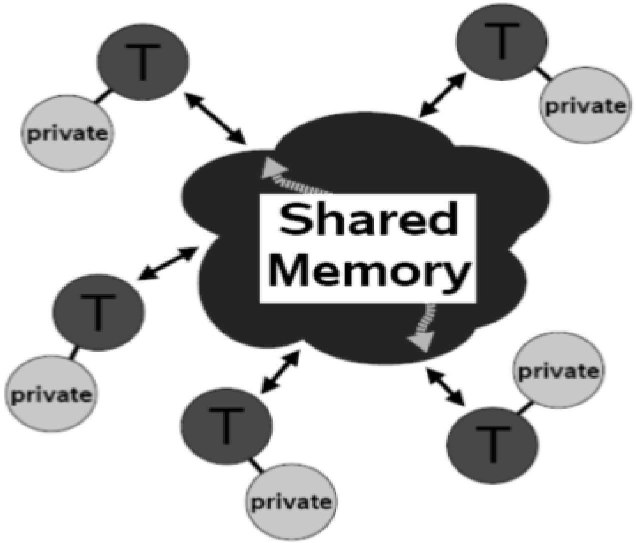
Memory Model



- Shared Memory, Thread Based Parallelism
- Explicit Parallelism
- Fork - Join Model

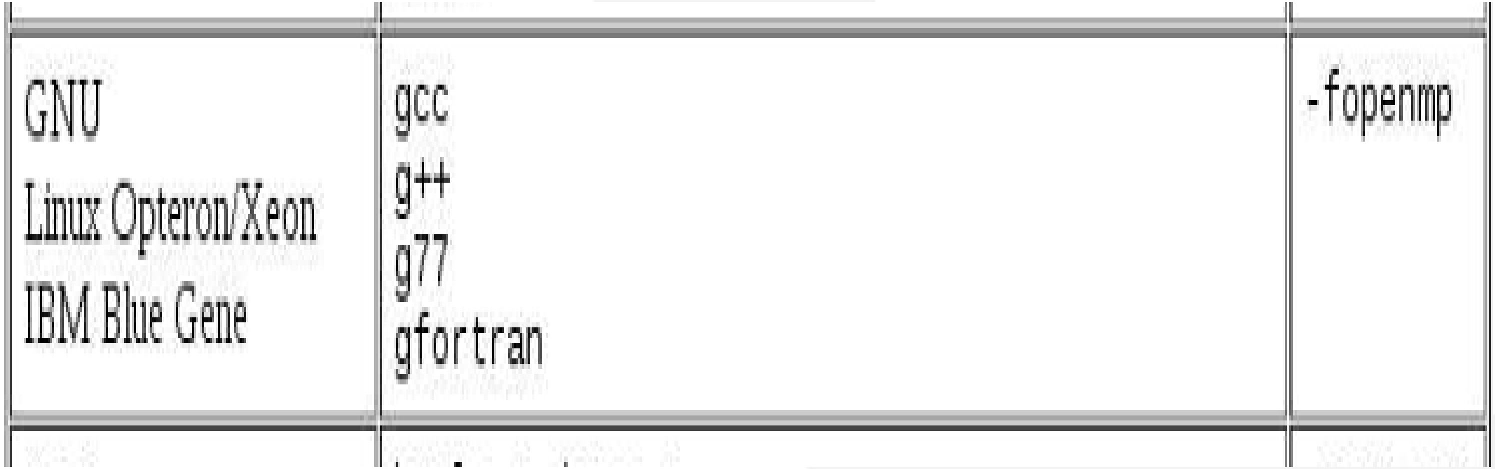
- Compiler Directive Based
- Nested Parallelism Support
- Dynamic Thread
- Memory Model : Flush often

Memory Model

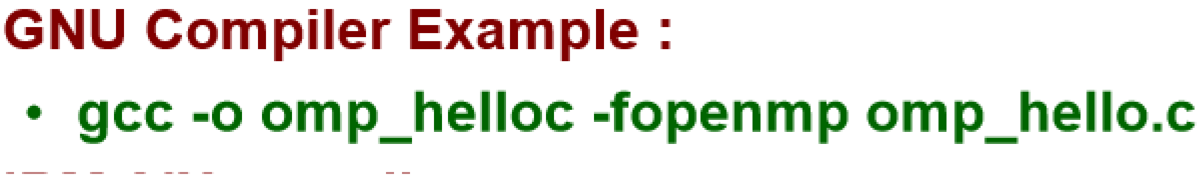


- Data is private or shared.
- All threads have access to same globally shared memory.
- Shared data accessible by all threads.
- Private accessed only by owned threads.
- Data transfer is transparent to programmer.
- Synchronization takes place, but it is almost implicit.

Compilation



GNU Linux Opteron/Xeon IBM Blue Gene	gcc g++ g77 gfortran	-fopenmp



GNU Compiler Example :

- gcc -o omp_helloc -fopenmp omp_hello.c

Advantages of OpenMP

- Good performance and scalability
  - ✓ If you do it right ....
- De-facto and mature standard
- An OpenMP program is portable
  - ✓ Supported by a large number of compilers
- Requires little programming effort

Components of OpenMP

Compiler Directives

- Parallel Construct
- Work Sharing
- Synchronization
- Data Environment
  - ✓private
  - ✓first private
  - ✓last private
  - ✓shared
  - ✓reduction

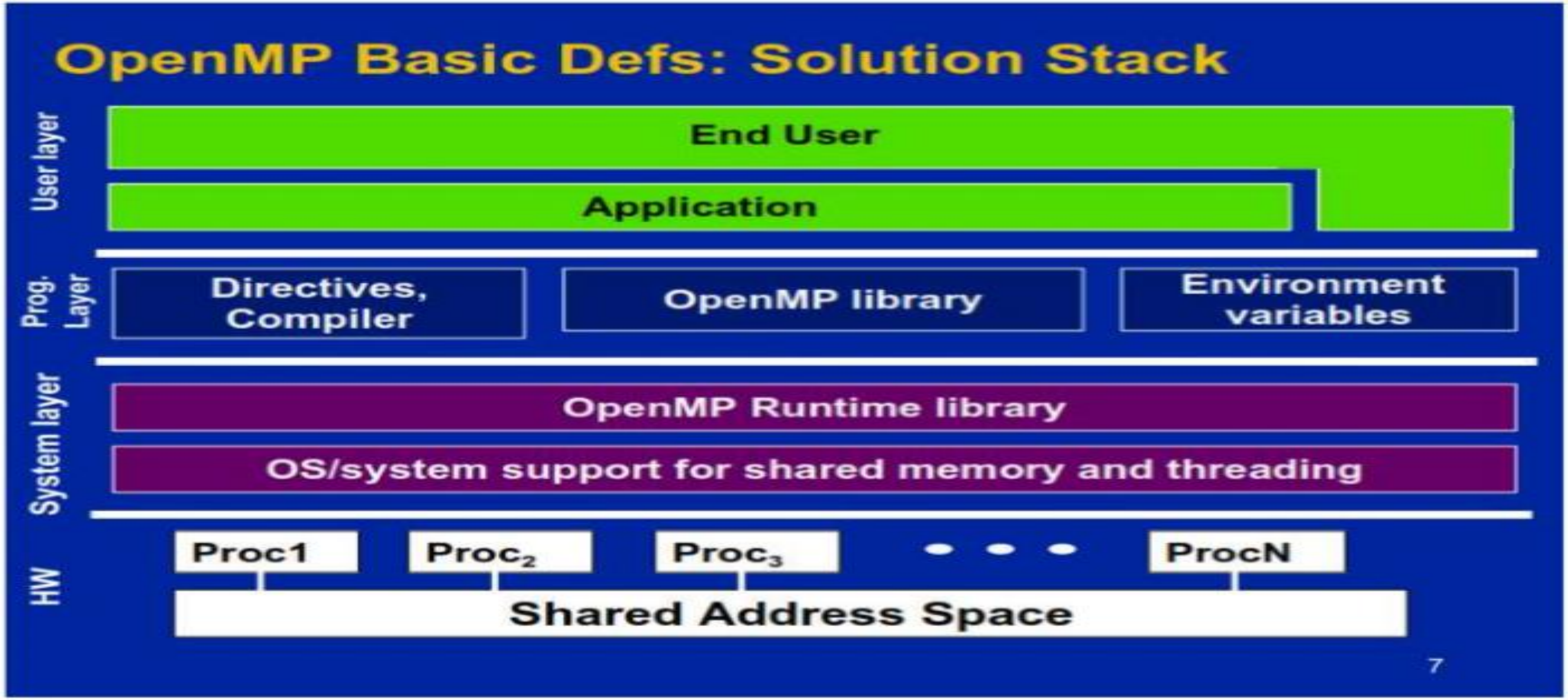
Environment Variables

- Number of threads
- Scheduling Type
- Nested parallelism
- Dynamic Thread
- Adjustment

Runtime Library routines

- Number of threads
- Thread ID
- Dynamic thread
- adjustment
- Nested parallelism

OpenMP Stack



OpenMP Basic Defs: Solution Stack

User layer

End User

Application

Prog. Layer

Directives, Compiler

OpenMP library

Environment variables

System layer

OpenMP Runtime library

OS/system support for shared memory and threading

HW

Proc1 Proc$_2$ Proc$_3$ • • • ProcN

Shared Address Space

7

OpenMp Directives

- #pragma omp directive-name [clause, clause..] new-line
  - Eg: #pragma omp parallel default(shared) private(beta,pi)

- General Rules:
  - Case sensitive
  - Compiler Directives follow C/C++ standards
  - Only one directive-name to be specified per directive
  - Each directive applies to at most one succeeding statement.
  - Use ("\") for continuing on succeeding lines.

Restrictions

- A parallel region must be a structured block that does not span multiple routines or code files
- It is illegal to branch into or out of a parallel region
- Only a single IF clause is permitted
- Only a single NUM_THREADS clause is permitted

How many Threads?

- **The number of threads in a parallel region is determined by the following factors, in order of precedence:**
  - ✓ **Evaluation of the IF clause**
  - ✓ **Setting of the NUM_THREADS clause**
  - ✓ **Use of the omp_set_num_threads() library function**
  - ✓ **Setting of the OMP_NUM_THREADS environment variable**
  - ✓ **Implementation default - usually the number of CPUs on a node, though it could be dynamic (see next bullet).**
- **Threads are numbered from 0 (master thread) to N-1**

Data Scoping

The OpenMP Data Scope Attribute Clauses are used to explicitly define how variables should be scoped. They include:

- ✓ PRIVATE
- ✓ FIRSTPRIVATE
- ✓ LASTPRIVATE
- ✓ SHARED
- ✓ DEFAULT
- ✓ REDUCTION

Data Scope Attribute Clauses are used in conjunction with several directives (PARALLEL, DO/for, and SECTIONS) to control the scoping of enclosed variables

Data Scoping

❖ private clause

- This declares variables in its list to be private to each thread
- Format
  - ✓ private (list)

```
Eg:     int B = 10;
        #pragma omp parallel private(B)
        B = … ;
```

- A private un-initialised copy of B is created before the parallel region begins
- B value is not the same within the parallel region as outside

Data Scoping

❖ **firstprivate clause**

- **Format**
  - ✓ first private (list)
  - ✓ Eg:

```
int B;
B = 10;
#pragma omp parallel firstprivate(B)
B = B + … ;
```

- **A private initialized copy of B is created before the parallel region begins**
- **The copy of each thread gets the same value**

Data Scoping

❖ **SHARED Clause**

- ✓ A shared variable exists in only one memory locatio and all threads can read or write to that address

- Format
  - ✓ shared (list)

❖ **DEFAULT Clause**

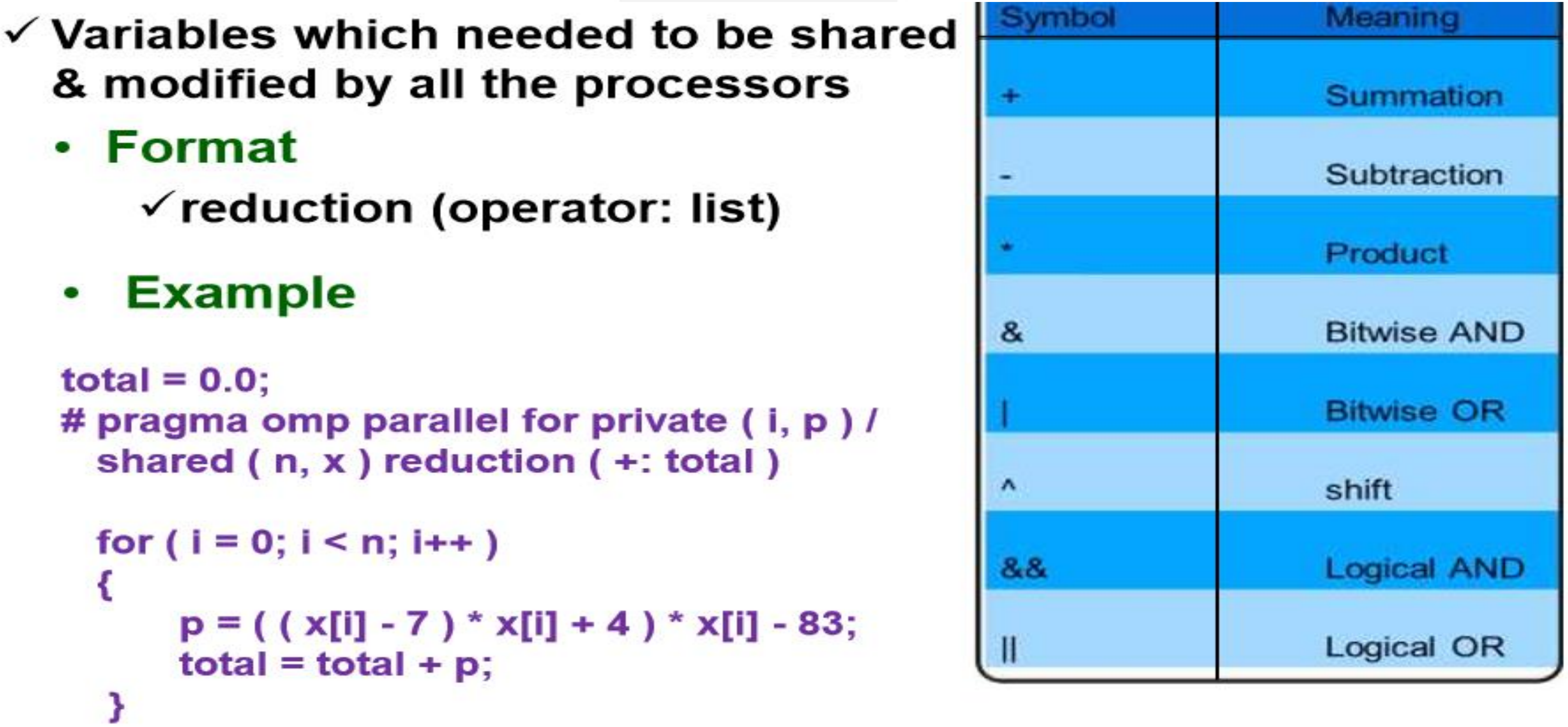
- ✓ Specify default scope for all variables in the lexical extent.

- Format
  - ✓ default (shared | none)

❖ **LASTPRIVATE Clause**

- ✓ Value from the last loop iteration assigned the original variable object.

Reduction Clause



- ✓ **Variables which needed to be shared & modified by all the processors**
  - **Format**
    - ✓ **reduction (operator: list)**
  - **Example**

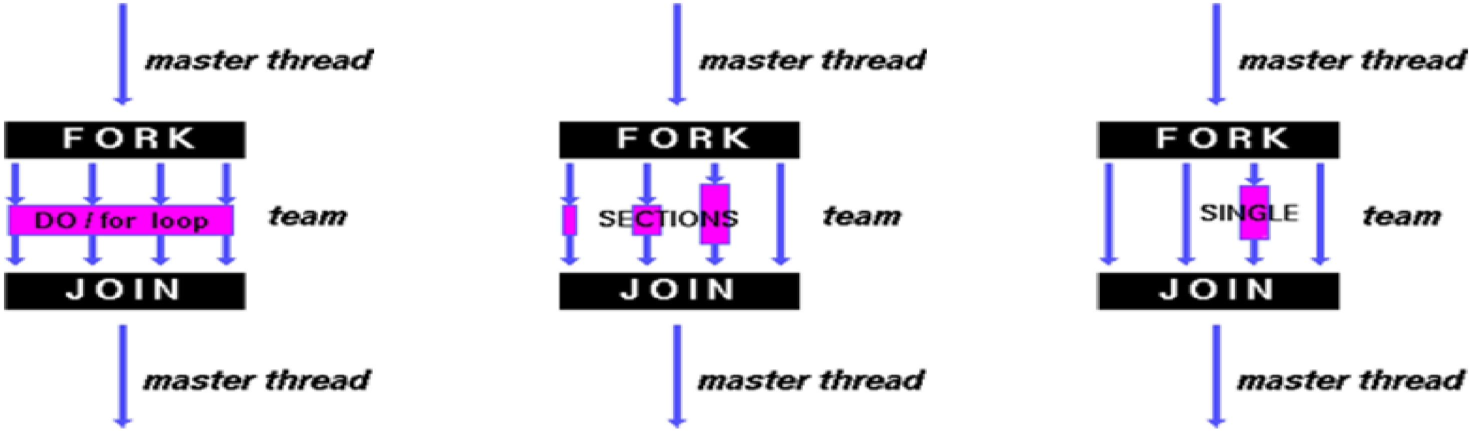
```
total = 0.0;
# pragma omp parallel for private ( i, p ) /
   shared ( n, x ) reduction ( +: total )

   for ( i = 0; i < n; i++ )
   {
          p = ( ( x[i] - 7 ) * x[i] + 4 ) * x[i] - 83;
          total = total + p;
    }
```

Symbol	Meaning
+	Summation
-	Subtraction
*	Product
&	Bitwise AND
|	Bitwise OR
^	shift
&&	Logical AND
||	Logical OR

Reduction Clause

- **FOR** - data parallelism
- **SECTIONS** - functional parallelism
- **SINGLE** - serializes a section of code



Program Programming Paradigm

```
#include <stdio.h>
   #include <mpi.h>
    main(int argc, char
**argv)
   {    int ierr;
       ierr = MPI_Init(&argc,
&argv);
      printf("Hello world\n");
               ierr =
MPI_Finalize();
}
```

```
mpicc hello.c
-o hello
```

mpirun -np 4 hello

OpenMP

```
#include <omp.h>
#include <stdio.h>
int main (int argc, char *argv[])
{
#pragma omp parallel
        printf(“Hello World”);
}
```

Compile: gcc –fopenmp hello.c

OpenMP

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

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

/* Fork a team of threads giving them their own copies of
variables */
#pragma omp parallel private(nthreads, tid)
  {

  /* Obtain thread number */
  tid = omp_get_thread_num();
  printf("Hello World from thread = %d\n", tid);

  /* Only master thread does this */
  if (tid == 0)
    {
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
    }

  }  /* All threads join master thread and disband */
```

Compile: gcc –fopenmp hello.c

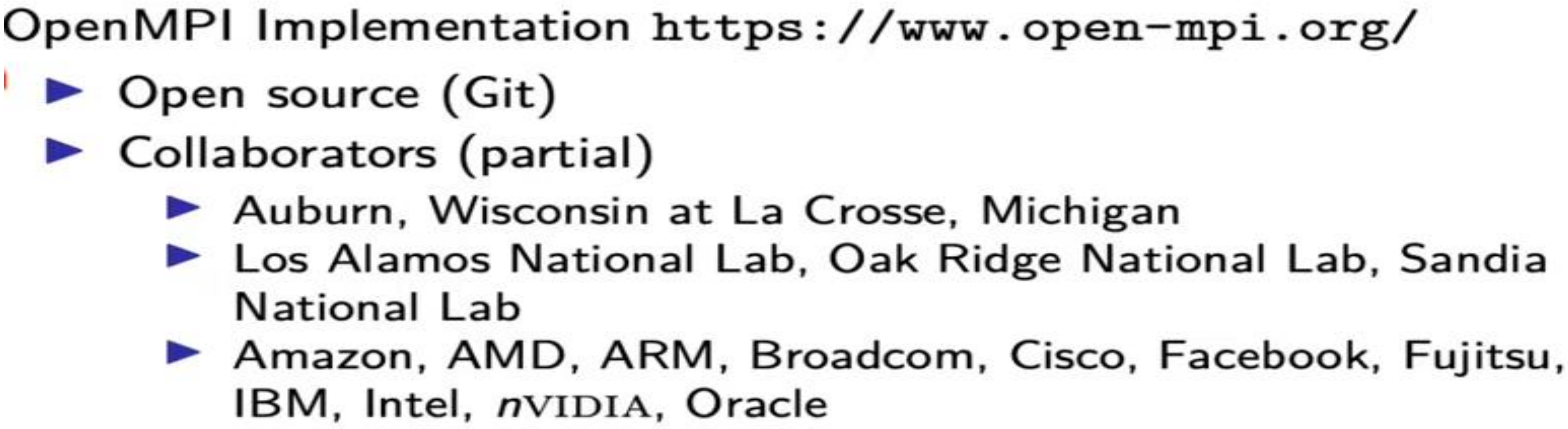
MPI: Message Passing Interface

- A message passing library specification
- Message passing among processes in parallel computing
- Meant for clusters and network of workstations
- Message Passing Standard for Parallel Programs

  -MPI Implementation left to individual vendors.

  -Most commonly supports C, Fortran, C++ programs

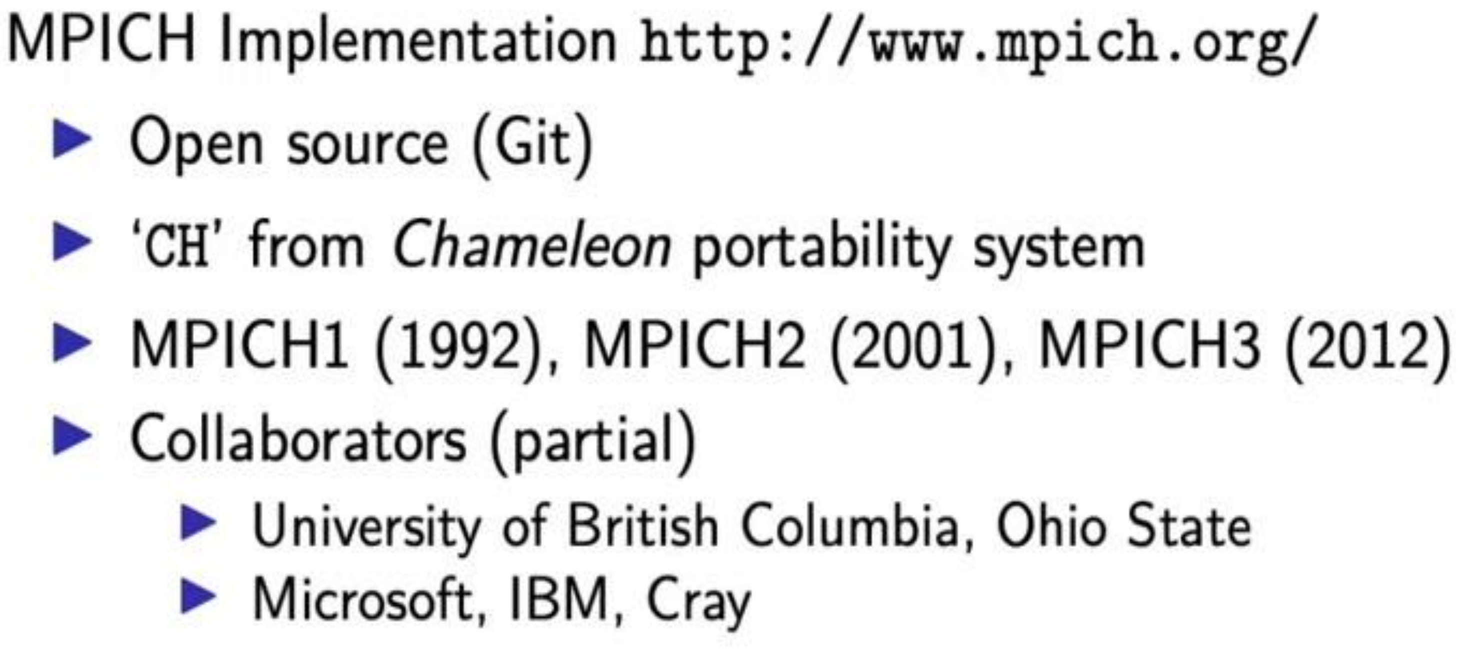
MPI: Message Passing Interface



OpenMPI Implementation `https://www.open-mpi.org/`

- Open source (Git)
- Collaborators (partial)
  - Auburn, Wisconsin at La Crosse, Michigan
  - Los Alamos National Lab, Oak Ridge National Lab, Sandia National Lab
  - Amazon, AMD, ARM, Broadcom, Cisco, Facebook, Fujitsu, IBM, Intel, *n*VIDIA, Oracle

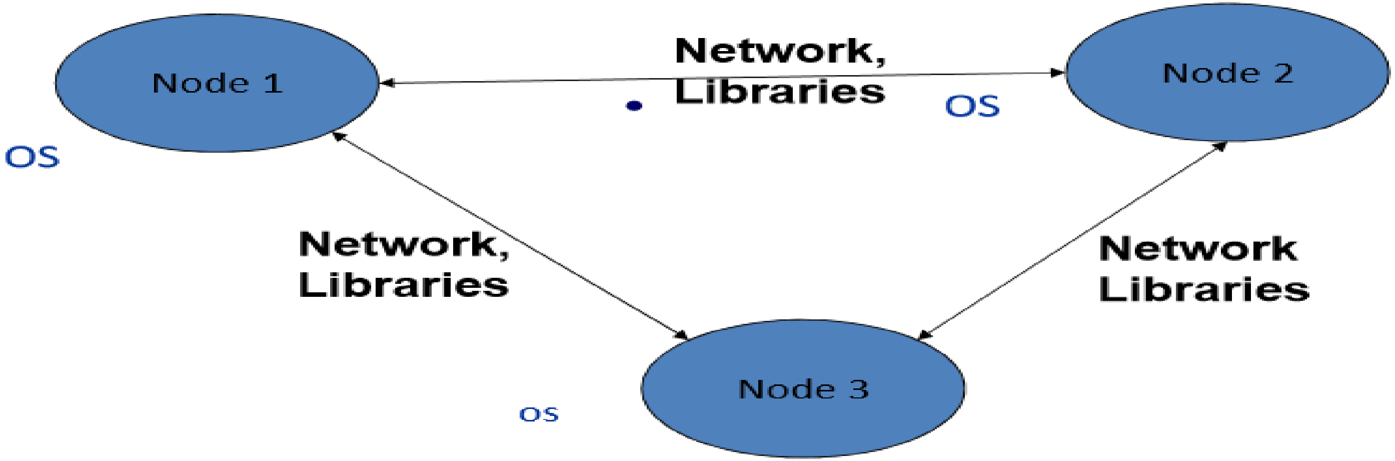
MPI: Message Passing Interface



MPICH Implementation `http://www.mpich.org/`

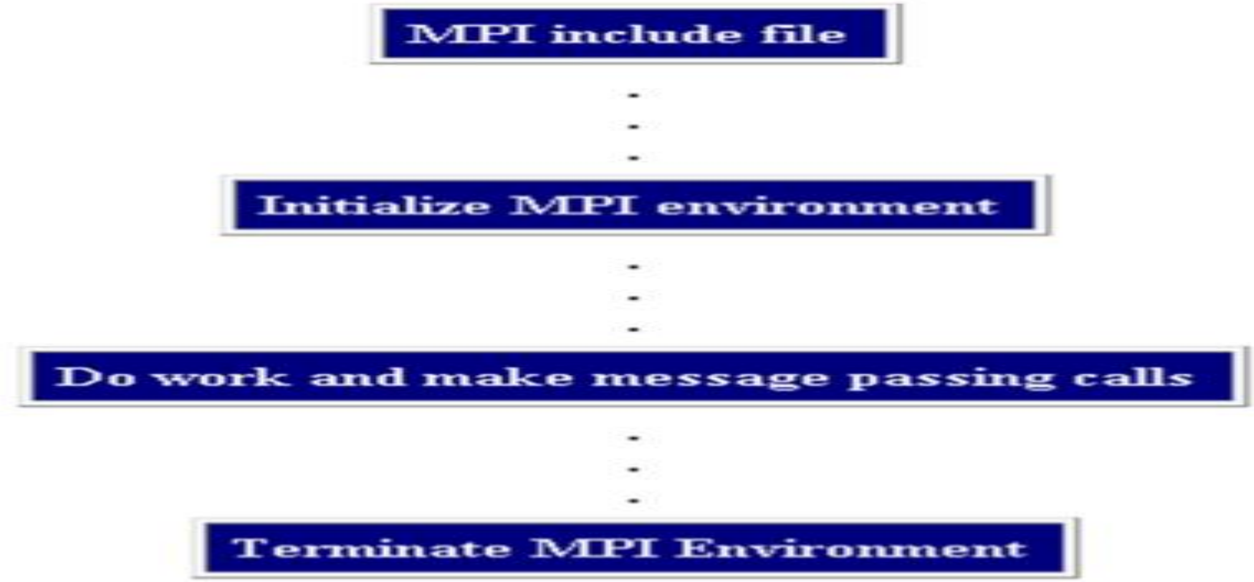
- Open source (Git)
- ‘`CH`’ from *Chameleon* portability system
- MPICH1 (1992), MPICH2 (2001), MPICH3 (2012)
- Collaborators (partial)
  - University of British Columbia, Ohio State
  - Microsoft, IBM, Cray

MPI: Message Passing Interface



- Multiple Nodes connected together to form a Computer Cluster

MPI: Message Passing Interface



MPI include file

.
.
.

Initialize MPI environment

.
.
.

Do work and make message passing calls

.
.
.

Terminate MPI Environment

OpenMP

```
#include <omp.h>
#include <stdio.h>
int main (int argc, char
*argv[])
{
#pragma omp parallel
num_threads(8)
        printf(“Hello World”);
}
```

Compile: gcc –fopenmp hello.c

Program Programming Paradigm

```
 #include <stdio.h>
   #include <mpi.h>
        main(int argc, char **argv)
   {
     int ierr, num_procs, my_id;
      ierr = MPI_Init(&argc, &argv);
/* find out MY process ID, and how
many processes were started. */
       ierr =
MPI_Comm_rank(MPI_COMM_WORL
D, &my_id);
      ierr =
MPI_Comm_size(MPI_COMM_WORL
D, &num_procs);
```

```
          printf("Hello worlc
I'm process %i out of
processes\n",
               my_id, num_prc

          ierr = MPI_Finaliz
   }
```

Program Programming Paradigm

```
 #include <stdio.h>
   #include <mpi.h>
        main(int argc, char **argv)
   {
      int ierr, num_procs, my_id;
       ierr = MPI_Init(&argc, &argv);
/* find out MY process ID, and how
many processes were started. */
       ierr =
MPI_Comm_rank(MPI_COMM_WORL
D, &my_id);
      ierr =
MPI_Comm_size(MPI_COMM_WORL
D, &num_procs);
```

```
         printf("Hello worl
I'm process %i out of
processes\n",
              my_id, num_pro

         ierr = MPI_Finaliz
   }
```

RV College of Engineering®

THANK YOU

Go, change the world®