**INTRODUCTION TO OPEN MP AND MPI**

OpenMP is an Application Program Interface (API). It provides a portable, scalable model for developers of shared memory parallel applications. The API supports C/C++ and Fortran on a wide variety of architectures.

It consists of 3 main parts –

1. Compiler directives (eg #pragma omp parallel)

2. Runtime Library Routines (eg omp\_get\_num\_threads())

3. Environment Variables (eg OMP\_NUM\_THREADS)

**Why Open MP?**

It provides a standard among a variety of shared memory architectures/platforms and establishes a simple and limited set of directives for programming shared memory machines. Significant parallelism can be implemented by using just 3 or 4 directives. It provides capability to incrementally parallelize a serial program, unlike message-passing libraries which typically require an all or nothing approach. For High Performance Computing (HPC) applications, OpenMP is combined with MPI for the distributed memory parallelism. This is often referred to as Hybrid Parallel Programming. OpenMP is used for computationally intensive work on each node. MPI is used to accomplish communications and data sharing between nodes.

**How Open MP works-**

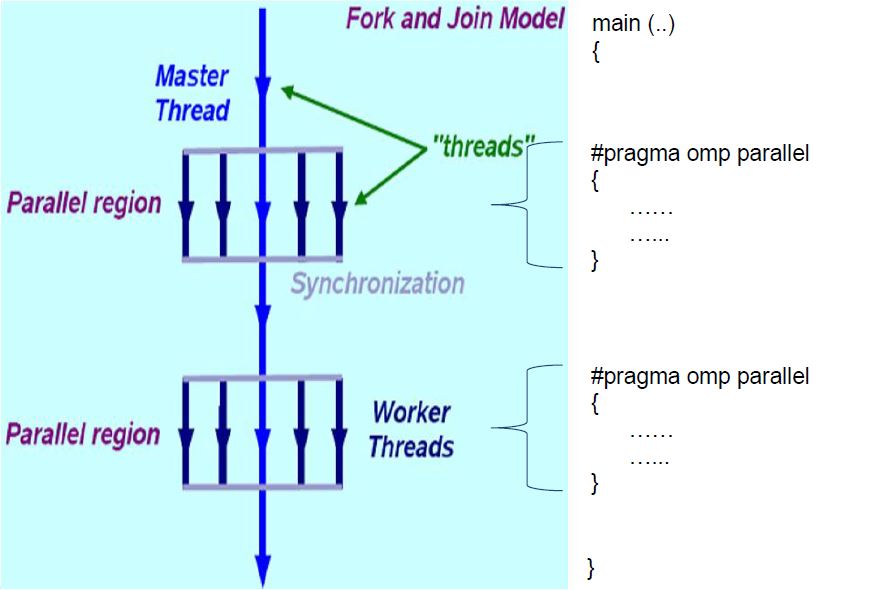
1. Provides a standard among a variety of shared memory architectures/platforms.

2. Establish a simple and limited set of directives for programming shared memory machines.

3. Significant parallelism can be implemented by using just 3 or 4 directives.

4. Provide capability to incrementally parallelize a serial program, unlike message-passing libraries which typically require an all or nothing approach

5. For High Performance Computing (HPC) applications, OpenMP is combined with MPI for the distributed memory parallelism. This is often referred to as Hybrid Parallel Programming. OpenMP is used for computationally intensive work on each node

6. MPI is used to accomplish communications and data sharing between nodes.

**What are the main parts of Open MP?**

**1. Compiler Directives-** It’s an instruction to the compiler to change how it’s compiling the code, rather than a piece of the code itself. #include and #define in C/C++ are considered directives, but they’re instructions to another program - the preprocessor. A true compiler directive might be something like a pragma, which is a compiler-specific command for changing what the compiler does, typically error handling.

OpenMP compiler directives are used for various purposes:

* 1. Spawning a parallel region
  2. Dividing blocks of code among threads
  3. Distributing loop iterations between threads
  4. Serializing sections of code
  5. Synchronization of work among threads.
  6. Syntax- #pragma omp parallel default (shared)

**2. Run time Libraries-** These routines are used for a variety of purposes:

Setting and querying the number of threads. Setting and querying the dynamic threads features and querying if in a parallel region, and at what level. for eg.

#include<omp.h>

int omp\_get\_num\_threads(void)

**3.Environment Variables-** OpenMP provides several environment variables for controlling the execution of parallel code at run-time.These environment variables can be used to control such things as:

Setting the number of threads.

Specifying how loop interations are divided

Binding threads to processors.

For eg. OMP\_GET\_THREADS, OMP\_STACKSIZE.

Schedulers -