1. What is OpenMP?

<https://www3.nd.edu/~zxu2/acms60212-40212-S12/Lec-11-01.pdf>

<http://wgropp.cs.illinois.edu/courses/cs598-s16/lectures/lecture17.pdf>

n this post, we will see What is OpenMP?, Which header file should we include in C/C++ language?, A simple example to create threads,

How to create required number of threads?, How to create multiple threads using "for" loop?, How to allocate different work to different thread?, How to synchronize threads?

**1. What is OpenMP?**

OpenMP is an API. OpenMP stands for Open Multi-Processing. 'Open' stands for Open source. Multi-Processing means Multi Threading. This means OpenMP is open source API which is used for multi-threading. It is managed by consortium OpenMP Architecture Review Board (OpenMP ARB) which is formed by several companies like AMD, Intel, IBM, HP, Nvidia, Oracle etc.

OpenMP is available for languages C, C++, Fortran. For the first time, OpenMP ARB releases OpenMP for Fortran language in 1997. In the next year, it gave OpenMP API for C/C++ languages. In this post, we will see how to use OpenMP for C/C++.

Each process starts with one main thread. This thread is call *master thread* in OpenMP. For a particular block of code, we create multiple threads alongwith this master thread. These extra threads other than master thread are called as *slave threads*.

OpenMP is called as *Shared Memory* model as OpenMP is used to create multiple threads and these multiple threads share the memory of main process.

OpenMP is called as *Fork-Join* model. It is because, all slave threads after execution get joined to the master thread. i.e. Process starts with single master thread and ends with single master thread.

**2. Which header file should we include in C/C++ language?**

For C/C++, we have to include "omp.h" header file. "omp.h" is available for gcc/g++ compilers.

**3. A simple example to create threads:**

#pragma omp parallel

{

printf("Hello World");

}

Suppose we are adding this code in a C language program. This snippet will create multiple threads. All of which are printing "Hello World". By default, number of threads created is equal to number of Processor cores.

**4. How to create required number of threads?**

#pragma omp parallel num\_threads(7)

{

printf("Hello World");

}

If you want create specific number of threads, use num\_threads() and a number indicating number of threads to be created should be passed as argument to num\_threads(). In above example, seven threads will be created. Each one will be responsible for printing "Hello World".

To understand how to create multiple threads, check example of array addition given in post [OpenMP Program for Array Addition.](http://www.comrevo.com/2016/01/openmp-array-sum-parallel.html)

**5. How to create multiple threads using "for" loop?**

We can create multiple threads for "for" block. Check following snippet:

#pragma omp parallel for

for(i=0;i<6;i++)

{

printf("Hello World");

}

In above code snippet, since we are not mentioning number of threads, number of threads will be equal to number of cores. This "for" loop will have six iterations which are done by these many number of threads.

**6. How to allocate different work to different thread?**

In OpenMP, we can allocate different work to different thread by using "sections".

Check following example:

#pragma omp parallel sections num\_threads(3)

{

#pragma omp section

{

printf("Hello World One");

}

#pragma omp section

{

printf("Hello World Two");

}

#pragma omp section

{

printf("Hello World Three");

}

}

In above example, we have created three threads by mentioning num\_threads(3). First thread will print "Hello World One". Second thread will print "Hello World Two" and third thread will print "Hello World Three".

Check example given in post [OpenMP Parallel Sections Example](http://www.comrevo.com/2016/01/openmp-parallel-sections-example.html) . You can understand thoroughly.

**7. How to synchronize threads?**

In OpenMP, we are creating multiple threads. By default, variables are shared by all the threads. A block of code where resources(variables) are shared by multiple threads is called "critical section". There is race condition i.e. competition among threads to use these resources(variables).

We can avoid such race condition by using preprocessor directive "#pragma omp critical".

Check following example:

#pragma omp parallel num\_threads(5)

{

#pragma omp critical

{

x=x+1;

}

}

Here, we are creating five threads by using num\_threads(5). As we are mentioning "#pragma omp critical", only one thread can do the operation of "x=x+1" at one instance.

Check this complete example in post [OpenMP Critical Section Example](http://www.comrevo.com/2016/01/openmp-critical-section-example.html) .