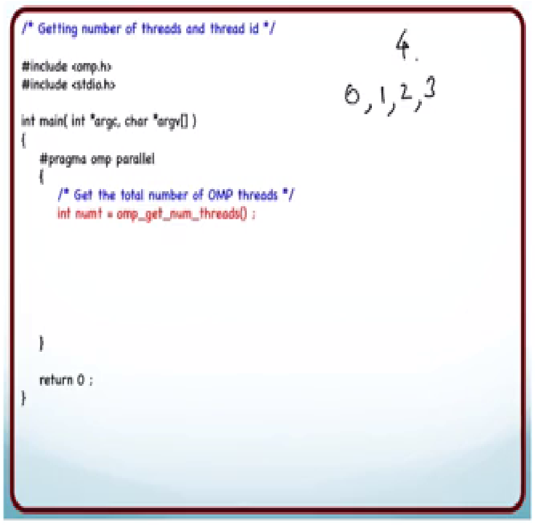
Introduction to Parallel Programming in OpenMp

Dr. Yogish Sabharwal
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

Lecture – 05
Basic OpenMP Constructs and Features

(Refer Slide Time: 00:25)



```
/* Getting number of threads and thread id */

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

int main( int *argc, char *argv[] )
{
    #pragma omp parallel
    {
        /* Get the total number of OMP threads */
        int numt = omp_get_num_threads() ;




    }

    return 0 ;
}
```

4.

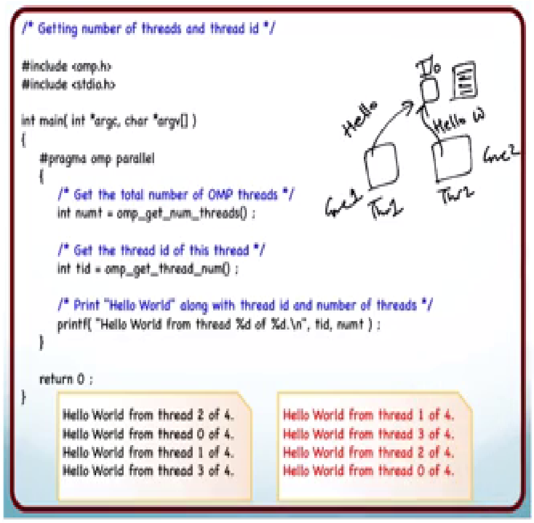
0, 1, 2, 3

So, whenever I have to do some work, right, let us say I am doing the dot product of a large director of let us say 1000 entries.

So, I decide that I liked openMP code, I will define a parallel and region and I will have each thread two one fourth the work, I will launch for threads and do one fourth the work, but how do you figure out that which thread is supposed to do what. How do you divide the work? What happens is that there are calls to figure out what is the total number of threads in the system, and there is an unique identifier given to each thread right it is called the thread id. And on the basis of the thread id and the total number of threads I can decide; what is the work I am supposed to do, right. So, if you want to do a dot product, and let us say that there are four threads, the thread will have id 0, 1, 2 and 3 can I use this information to figure out what work I am going to do?

So, each thread will know it is id right and it will not the total number of threads, now I can figure out what is the work I am going to do. I am going to start from 250 times my thread id for the next 250 elements right. I am going to do the dot product store it somewhere and in the; and we will figure out a way to accumulate these results right. Knowing the thread id and total number of threads is important for a lot of things including the distribution of work and communication. So, how do you get the total number of threads and thread id? So, this is the call OMP get num threads, which essentially returns the total number of threads.

(Refer Slide Time: 02:03)



```
/* Getting number of threads and thread id */

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

int main( int *argc, char *argv[] )
{
    #pragma omp parallel
    {
        /* Get the total number of OMP threads */
        int numt = omp_get_num_threads() ;

        /* Get the thread id of this thread */
        int tid = omp_get_thread_num() ;

        /* Print "Hello World" along with thread id and number of threads */
        printf( "Hello World from thread %d of %d.\n", tid, numt ) ;
    }

    return 0 ;
}
```

```
Hello World from thread 2 of 4.
Hello World from thread 0 of 4.
Hello World from thread 1 of 4.
Hello World from thread 3 of 4.
```

```
Hello World from thread 1 of 4.
Hello World from thread 3 of 4.
Hello World from thread 2 of 4.
Hello World from thread 0 of 4.
```

And the call OMP get thread num returns the thread number of the current thread. So, remember this code is getting executed inside a parallel region. So, different threads will be making calls to this function, and each one of them will get a different value back. So, what about these variables num t and tid where are these?

Student: (Refer Time: 02:31).

So, these are going to be in the stack of each thread, this is kind of like the first frame on the stack of is thread right. So, since the end declared and defined inside the parallel region they will go on the start. The important part is that each thread will have it is own copy of num t and tid, what do I do next? I just print hello world along with the third id and number of threads. So, what do I see what they do this? This is what I see right I run it again this is what I see, what do you observe from the output.

Student: (Refer Time: 03:17).

Print is not atomic, most of the time 99 percent of the time if you have short print statements, will find that they get printed contiguously or otherwise print is not atomic.

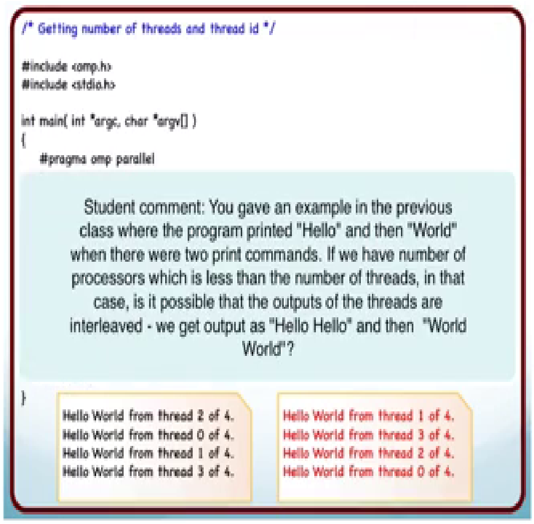
Student: (Refer Time: 03:27).

The order right you cannot be assured of what order the output will appear in or what order actually the statement executed in. The only thing you can be sure about that within one thread the statement scot executed sequentially, but our mind saids in what order did that does sequential orders get interleaved, you have absolutely no idea and if you run multiple times they make it interleaved in different manners. Because it is completely dynamic right when does the time slice expire, right. So, when the threads are initially launched which thread does the processor pick up first. If any thread does an eye operation like printers printer is an eye operation right it has to print on the screen to an IO device right it is not deterministic.

Student: (Refer Time: 04:15).

You have no idea about the order in which three statements get executed, the only thing you are sure about is that within each thread their execution takes place sequentially. So, that is why you see different orders when you run at multiple times.

(Refer Slide Time: 04:30)



```
/* Getting number of threads and thread id */

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

int main( int *argc, char *argv[] )
{
    #pragma omp parallel
```

Student comment: You gave an example in the previous class where the program printed "Hello" and then "World" when there were two print commands. If we have number of processors which is less than the number of threads, in that case, is it possible that the outputs of the threads are interleaved - we get output as "Hello Hello" and then "World World"?

```
}
```

```
Hello World from thread 2 of 4.
Hello World from thread 0 of 4.
Hello World from thread 1 of 4.
Hello World from thread 3 of 4.
```

```
Hello World from thread 1 of 4.
Hello World from thread 3 of 4.
Hello World from thread 2 of 4.
Hello World from thread 0 of 4.
```

.

Student: (Refer Time: 04:30) given example in the previous class hello and then world is two print commands (Refer Time: 04:37), we have a single processor, (Refer Time:

04:42) processor, which is less than the number of threads. So, in that case is it possible that if the outputs which we get are being hello (Refer Time: 04:48) world (Refer Time: 04:49) this.

So, that may even happen when you have separate processes, even if you have four cores and your running four threads, even then you may see that, how to put the core has to right of the output device that output devices is a common output device, it is a same screen right same display.

Student: (Refer Time: 05:05).

So, there is some memory location and then some IO port maybe where all of this is being written. So, maybe thread one this is recruiting on core 1 thread 2 executing on core 2, but when it is executing the print statement it sends some data may be it sends H e l l o.

Student: (Refer Time: 05:24).

Right, thread 2 in the meanwhile sense it is on H e l l o w.

Student: (Refer Time: 05:31).

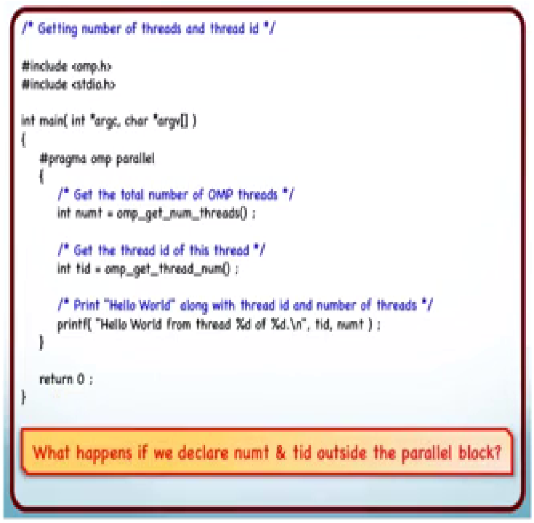
So, what will the screen print the screen will print hello followed by hello w, and then maybe something else.

Student: So, the stuff like (Refer Time: 05:39) painting, on the console is not a good idea (Refer Time: 05:43).

You know I mean you have to do it; you will have to figure out how to make those operations atomic. Atomic means what appear as if nothing else is happening within that atomic operation right. I just know the thread is getting scheduled within that operation, and you have to make it seem atomic. So, one way of doing it is maybe yeah we will talk about this maybe locks mutual exclusion there is lots of we will doing this. So, you maybe you get a lock, then you do the print then you unlock.

So, we will discuss all those and not in the context of prints in prints typical it is not such a big issue. So, what if the print came out wrong right? What is important is the file operations must be correct, and the data I mean when you working on variables that they should be managed properly that is very very important, that is the most important thing the main memory operations your profile on main memory, they must be consistent. So, all these output generated using actual programs right, but most of the time like more than 90 percent of the time you will realise that one printer statement of years together, but if you run maybe one million times is something, you may actually find that that does not happen so, how print operations are not at all.

(Refer Slide Time: 06:57)



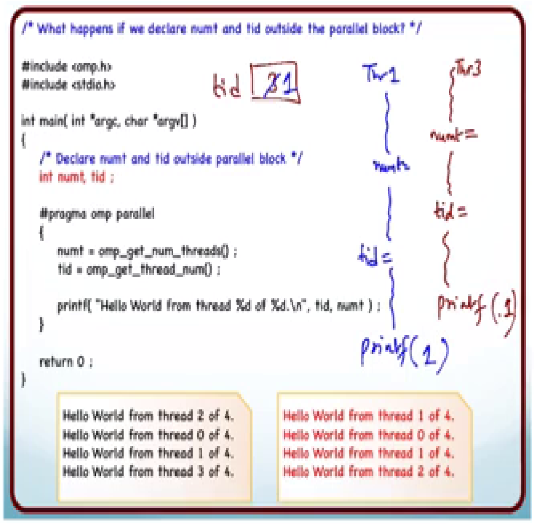
```
/* Getting number of threads and thread id */

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

int main( int *argc, char *argv[] )
{
    #pragma omp parallel
    {
        /* Get the total number of OMP threads */
        int numt = omp_get_num_threads() ;

        /* Get the thread id of this thread */
        int tid = omp_get_thread_num() ;

        /* Print "Hello World" along with thread id and number of threads */
        printf( "Hello World from thread %d of %d.\n", tid, numt ) ;
    }

    return 0 ;
}
```

What happens if we declare numt & tid outside the parallel block?

.

Now let us consider the following scenario right. So, I had declared and define the num t and tid inside the parallel region right and therefore, a separate copy was created for each of these, what happens if we declare these variables outside the parallel block.

(Refer Slide Time: 07:16)



```
/* What happens if we declare numt and tid outside the parallel block? */

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

int main( int *argc, char *argv[] )
{
   /* Declare numt and tid outside parallel block */
   int numt, tid ;

   #pragma omp parallel
   {
      numt = omp_get_num_threads() ;
      tid = omp_get_thread_num() ;

      printf( "Hello World from thread %d of %d.\n", tid, numt ) ;
   }

   return 0 ;
}
```

```
Hello World from thread 2 of 4.
Hello World from thread 0 of 4.
Hello World from thread 1 of 4.
Hello World from thread 3 of 4.
```

```
Hello World from thread 1 of 4.
Hello World from thread 0 of 4.
Hello World from thread 1 of 4.
Hello World from thread 2 of 4.
```

.

So, this is a change I have made, remember I asked you to concentrate on the red part right. So, I have not declared num t and tid outside the parallel region. So, what happens now? So, when I run this could I get the output that I expected, I run it again I get some other output.

Student: (Refer Time: 07:42).

So, why has this happened? So, thread three never printed anything and third one printed twice.

Student: (Refer Time: 07:49).

Correct now this is a shared variable. So, num t and tid are not shared variables. So, what happened in this particular case?

Student: (Refer Time: 08:01).

Thread three was let us say executing, yeah it executed num t equal to OMP get num threads, and in the meanwhile what happened was that thread one came along it executed num t equal to OMP get num threads right. So, they both of them got the number of threads and let us say then thread three then executed tid equal to OMP get thread num. So, this tid is a shared variable right. So, what got stored into tid? So, thread three stored the value three in the variable tid, but it shared by all the threads, and now before it could get to printer what happened was that thread one came and executed tid equal to OMP get thread num. So, what happened here? What happened was this actually got replaced with the value one.

So, now, tid has value one, and then thread three went ahead and called printer, and when it passed at the parameter tid what was actually passed? One got passed right and similarly thread one eventually called printef at some point and time, and what got passed one got passed that is why you see one getting print it was. So, this is essentially a race condition right when there are variables or resources being shared by multiple threads, and they try to update that variable right at the same time, then you run into race conditions.