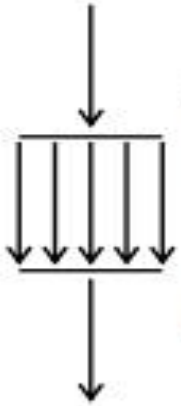
OpenMP: parallel regions

- A parallel region within a program is specified as

```
#pragma omp parallel [clause [[,] clause] …]
        Structured-block
```

- A team of threads is formed
- Thread that encountered the omp parallel directive becomes the **master thread** within this team
- **The structured-block is executed by every thread in the team**.
- At the end, there is an implicit **barrier**
- Only after **all threads have finished, the threads created by this directive are terminated and only the master resumes execution**
- A parallel region might be refined by a list of clause

OpenMP: parallel regions



```
#pragma omp parallel default(none)
{
        . . .
}
```

This creates a team of threads

Each thread then executes all lines of code in this block.

Think of it this way:

#pragma omp parallel default(none)

“Hello Word” Example/2

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

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

    #pragma omp parallel
    {
            printf("Hello World\n");

    } // End of parallel region

    return(0);
}
```

OpenMP controlling number of threads

- Once a program is compiled, the number of threads can be controlled using the following shell variables
  - **At the program level**, via the **omp_set_number_threads** function:

```
void omp_set_num_threads(int n)
```

  - At the pragma level, via the **num_threads** clause:

```
#pragma omp parallel num_threads(numThreads)
```

OpenMP controlling number of threads

- Asking how many cores this program has access to:

```
num = omp_get_num_procs( );
```

- Setting the number of available threads to the exact number of cores available:

```
omp_set_num_threads( omp_get_num_procs( ) );
```

- Asking how many OpenMP threads this program is using right now:

```
num = omp_get_num_threads( );
```

- Asking which thread number this one is:

```
me = omp_get_thread_num( )
```

Hello World in OpenMP

- Each thread has a unique integer “id”; master thread has “id” 0
- Other threads have “id” 1, 2, …
- OpenMP runtime function

`omp_get_thread_num()`

returns a thread’s unique “id”.

- What will be the programs output?

```
#include <omp.h>
void main()
{
   #pragma omp parallel
   {
      int ID = omp_get_thread_num();
      printf(“ hello(%d) ”, ID);
      printf(“ world(%d) \n”, ID);
   }
}
```

OpenMP: parallel loops

- A parallel for loops are declared as

```
#pragma omp for [clause [[,] clause] …]
        for-loops
```

Each *for loop* among ***for-loops*** associated with the ***omp for directive*** must be in the **canonical form**

- ✓ the loop variable is made **private** to each thread in the team and must be either (unsigned) **integer or a pointer**,
- ✓ the loop variable should not be modified during the execution of any iteration;
- ✓ the condition in the for loop must be a **simple relational expression**,
- ✓ the increment in the for loop must specify a change by constant additive expression;
- ✓ the number of iterations of all associated loops must be known before the start of the outermost for loop.

Parallelizing Loops with Independent Iterations

- The `omp parallel for` directive in line 6 specifies that the for loop must be executed in parallel
- The iterations of the *for loop* will be divided among the threads
- Once all iterations have been executed, all threads in the team are synchronized at the implicit barrier at the end of the parallel for loop
- All slave threads are terminated
- Finally, the execution proceeds sequentially, and the master thread terminates the program by executing return 0

```
1  #include <stdio.h>
2  #include <omp.h>
3
4  int main (int argc, char *argv[]) {
5    int max; sscanf (argv[1], "%d", &max);
6    #pragma omp parallel for
7      for (int i = 1; i <= max; i++)
8        printf ("%d: %d\n", omp_get_thread_num (), i);
9    return 0;
10 }
```

Listing 3.3 Printing out all integers from 1 to *max* in no particular order.

Program does not specify how the iterations should be divided among threads.

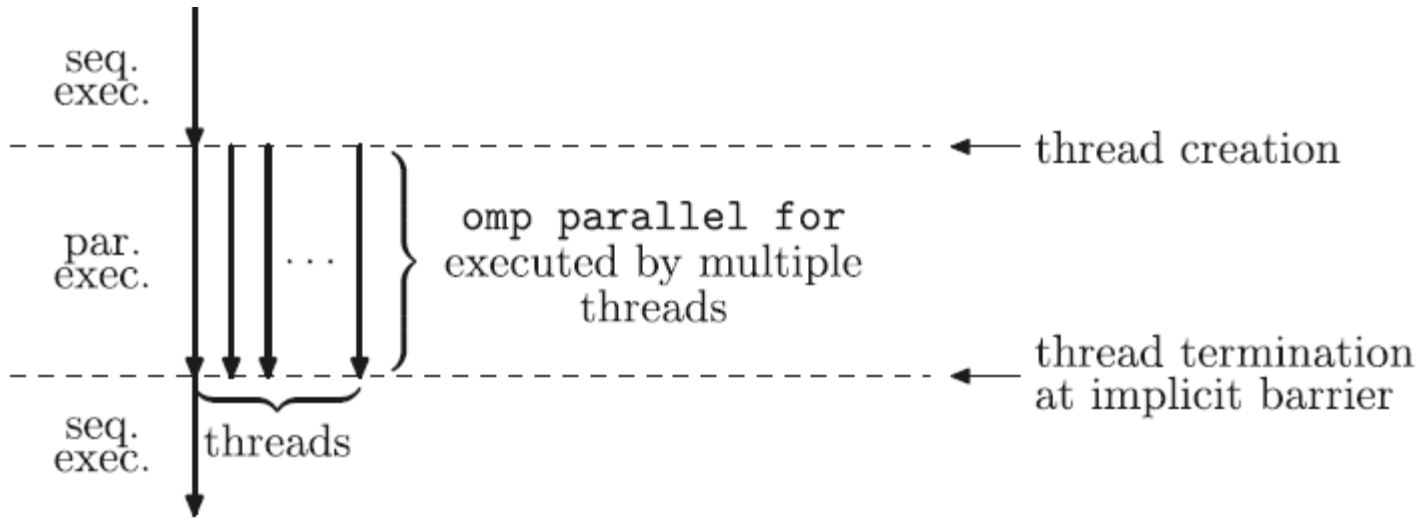


Fig. 3.5 Execution of the program for printing out integers as implemented in Listing 3.3

OpenMP: Nowait clause

```
int main()
{
#pragma omp parallel
    {
    #pragma omp for
        for (int i = 0; i <5; i++)
        {
            printf("first loop i= %d\n", i);
        }

        printf("outside\n");
    }


    return 0;
}
```

```
first loop i= 2
first loop i= 3
first loop i= 0
first loop i= 4
first loop i= 1
outside
outside
outside
outside
outside
outside
outside
outside

D:\DSCA\Parallel_Computing\Lab_Programs\Nowait_trial\x64\Debug\Now
To automatically close the console when debugging stops, enable To
le when debugging stops.
Press any key to close this window . . .
```

```
int main()
{
#pragma omp parallel
    {
    #pragma omp for nowait
        for (int i = 0; i <5; i++)
        {
            printf("first loop i= %d\n", i);
        }

        printf("outside\n");
    }


    return 0;
}
```

```
first loop i= 1
outside
first loop i= 3
outside
first loop i= 0
outside
first loop i= 2
outside
outside
first loop i= 4
outside
outside
outside

D:\DSCA\Parallel_Computing\Lab_Programs\Nowait_trial\x64\Debug
To automatically close the console when debugging stops, enabl
le when debugging stops.
Press any key to close this window . . .
```