

High Performance Computer Architectures Practical Course - Exercise 7 -

Tutorium 1

David Jordan (6260776)
Florian Rüffer (7454628)
Michael Sanjatin (7485765)

June 6, 2023

0_HelloWorldParallel

In this task we need to parallelize a HelloWorld program with OpenMP. We start of by creating 10 threads (line 3):

```
1      int main() {
2
3      #pragma omp parallel num_threads(10)
4      {
5          int id = 0;
6
7          cout << " Hello world " << id << endl;
8      }
9
10     return 0;
11 }
12
13 }
```

File 1: Step 1

This creates a more or less unexpected output, which can be traced back to the threads not being synchronized. To solve this issue, we will synchronize our threads by adding 'omp critical' to the parallel environment (line 7):

```
1      int main() {
2
3      #pragma omp parallel num_threads(10)
4      {
5          int id = 0;
6
7          #pragma omp critical
8          {
9              cout << " Hello world " << id << endl;
10             }
11         }
12
13         return 0;
14     }
```

File 2: Step 2

We still need to get the id of our threads, to know which thread computed which output. Basically which 'Hello World' is printed out by which one of the ten threads. To do this we get the thread id like this (line 5):

```
1      int main() {
2
3      #pragma omp parallel num_threads(10)
```

```

4      {
5          int id = omp_get_thread_num();
6
7          #pragma omp critical
8          {
9              cout << " Hello world " << id << endl;
10             }
11         }
12
13     return 0;
14 }

```

File 3: Step 2

At this point we can compare what happens, if we forget to properly synchronize our threads.

```

• hpc@hpc-vm:~/Desktop/Exercise/Exercises/5_OMP/0_HelloWorldParallel$
Hello world 0
Hello world 7
Hello world 8
Hello world 9
Hello world 6
Hello world 5
Hello world 4
Hello world 3
Hello world 2
Hello world 1

```

Figure 1: w/o synchronization

```

• hpc@hpc-vm:~/Desktop/Exercise/Exercises/5_OMP/0_HelloWorldParallel$
Hello world 1
Hello world 2
Hello world 3
Hello world 4
Hello world 5
Hello world 6
Hello world 7
Hello world 8
Hello world 9
Hello world 0

```

Figure 2: w/ synchronization

We can see that the id's are completely out of order, if the leave the synchronization out. By adding it the output is perfectly in order.

1_Bugs

In this task we will examine four bugs and how to fix them.

Bug1

If we run this code, we receive an error telling us, that the output array we tried to copy is not correct. This can be fixed by editing the following lines of code:

```
1  #pragma omp parallel firstprivate(N) num_threads(  
    NThreads)  
2  {  
3      #pragma omp for  
4      for(int i=0; i<N; i++)  
5          output[i] = input[i];  
6  }
```

File 4: Step 2

Especially changing line 1 to firstprivate instead of just private. This error occurs because 'private' is not initializing the values properly. With 'private' each thread has its own local copy of the value, but it is not kept while entering or exiting the private region and therefore also does not effect the value of the variable in the 'master thread'.

Bug2

Again our output array is not correct. Now the problem lies in the tmp variable of this code snippet:

```
1  tmp = 0;  
2  #pragma omp parallel num_threads(NThreads)  
3  {  
4      #pragma omp for  
5      for(int i=1; i<N; i++)  
6      {  
7          tmp += i;  
8          outputParallel[i] = float(tmp)/float(i);  
9      }  
10 }
```

File 5: Step 2

The variable tmp will be used by multiple threads, leading to multiple threads updating the same value at the same time. This will lead to a corrupted variable and incorrect results. To solve this issue we need to make the calculation independent from the iteration (independent from the results of previous iterations).

```
1  tmp = 0;  
2  #pragma omp parallel num_threads(NThreads)  
3  {  
4      #pragma omp for
```

```

5      for(int i=1; i<N; i++)
6      {
7          float tmp = (1+i)*i/2; // arithmetical
              progression
8          outputParallel[i] = float(tmp)/float(i);
9      }
10     }

```

File 6: Step 2

Bug3

Here the problem can be found in line 4. By using 'nowait' lines 12 and 13 can be executed before the sum calculation (in line 8) is finished.

```

1      sum = 0;
2      #pragma omp parallel firstprivate(N) num_threads(
          NThreads)
3      {
4          #pragma omp for nowait
5          for(int i=0; i<N; i++)
6          {
7              #pragma omp atomic
8              sum += input[i];
9          }
10
11         #pragma omp for
12         for(int i=0; i<N; i++)
13             outputParallel[i] = input[i]/sum;
14     }

```

File 7: Step 2

To fix this we can cut out the 'nowait' completely. So we calculate the sum first and then continue with filling 'outputParallel'

```

1      sum = 0;
2      {
3
4          for(int i=0; i<N; i++)
5          {
6              sum += input[i];
7          }
8
9          #pragma omp parallel firstprivate(N) num_threads
              (NThreads)
10         #pragma omp for

```

```

11     for(int i=0; i<N; i++)
12         outputParallel[i] = input[i]/sum;
13     }

```

File 8: Step 2

Bug4

By using 'nowait' in the previous bug, we didn't wait for relevant code to finish calculations properly. Now we need to add a barrier to ensure, that relevant calculations finish before other start.

```

1
2     #pragma omp atomic
3     sum += sumLocal;
4
5     #pragma omp barrier
6
7     #pragma omp for
8     for(int i=0; i<N; i++)
9         outputParallel[i] = input[i]/sum;
10    }

```

File 9: Step 2

Here we place a barrier just after the final sum calculations. It is crucial for the sum to finish, before other calculations (which use the sum) can start. As this is just an excerpt, we would need to add this line right after line 72 in the original code.

Section 3

Section 4