High Performance Computer Architectures Practical Course - Exercise 7 -

Tutorium 1

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

June 5, 2023

Section 1

Section 2

1 Pi

1.1 omp parallel and OpenMP

First, we need to specify the number of threads (in this case 2) and a variable to store the sums, that are calculated by each thread. Therefore the array has the lengt of the number of threads, initialized with the value 0:

```
    const int nThreads = 2;
    double sums[nThreads] = {0};
```

Another important task is to save the id of the threads, so that they can be addressed and saved correctly in the "cache-sum-list". Because we split these steps into 2 threads, each thread only needs to calculate one half (steps / thread_count). On last thing to pay attention is the calculation of x. Here we need to factor in the offset for the current thread. Last but not least we need to calculate the sums in the "cache-sum-list" and do the final computations.

Least amount of changes

This can quickly be done by adding one line of code before the loop:

```
    #pragma omp parallel for reduction(+ : sum)
       private(i,x)
```

The variables x and i are declared as private variables for safety reasons, so that they do not interfer with each other. In the solution sum is declared as one value, and not a list - this is probably for better data consistency and the correctnerss of the final result.
One another advantage of OpenMP is the correctness of the code even if the library is disabled.

Section 4

```
    Double_t background(Double_t *x, Double_t *par) {
   return par[0] + par[1]*x[0] + par[2]*x[0]*x[0] +
      par[3]*x[0]*x[0]*x[0];
}
```

File 1: Order 3

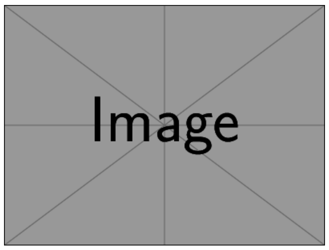

Figure 1: 6-order