UPPSALA
UNIVERSITET

Assignment 1

Elvira Brenner, Johan Paulsson, Oscar Hultmar

1 Problem

The task is to calculate a two dimensional integral using the midpoint rule. The calculations are then to be parallellized using MPI and the performance analyzed. The result was to be written to an output file together with the time it took to run the code and the number of intervals.

To numerically solve the integral it was written as a sum according to the midpoint rule. Then the MPI was initialized. The summation was done over an interval which was split into as many subintervals as there were threads. Each thread was assigned an unique interval for which the summation was to be done. A local sum was introduced and set to 0. The step size was set to the integration boundary divided by the interval, i.e 2/intervals. Then each thread calculated the sum within its interval and then all sums where summarized into a global sum which was printed as the result and written to an output file.

2 Theory

To solve this problem iteratively, the problem needed to be converted from an integral to a sum. The definition of an integral is a sum with an infinite number of iterations and an infinitely small step size. This can't be done on a computer. Instead of an infinite number of iterations the sum form has a set number of iterations, the bigger the number the more precise the result. And instead of an infinitely small step size a small number was used. Then the midpoint rule was used to define the x and y in the summation equation. When integrating, the integral calculates the area under the step dx and the midpoint rule defines the x-value as the midpoint of the area that is being calculated.

With this theory implemented, the summations were made over the interval n and the step sizes dx and dy were set to h where h=2/n. This resulted in the equation:

$$\int_0^2 (\int_0^2 (x + sin(y) + 1)dx)dy = \sum_{i=1}^{n} \sum_{j=1}^{n} h^2 * ((i - 0.5) + sin((h * (i - 0.5)) + 1) \quad (1)$$

3 Results

The exact result for the integrals $10 - 2cos(2) \approx 10.832293673094284$. The result achieved by the numerical method was 10.8322936731418178 for the interval n=100 000. This gives the error $4.8 * 10^{-11}$. This error can be explained by the fact that the numerical solution is only an approximation of the problem and the result is not expected to be exact.

The ideal behaviour of introducing threads is reducing the workload for each thread. I.e. by doubling the amount of threads and equally distributing the work on those threads, the run time will be halved. As can be seen in figure 1 the real curve does not fit the ideal behaviour perfectly. But one can see that the run time decreases as the number of threads increases. The small difference from the ideal behaviour could be explained by other processes running on the computer taking up processor power.

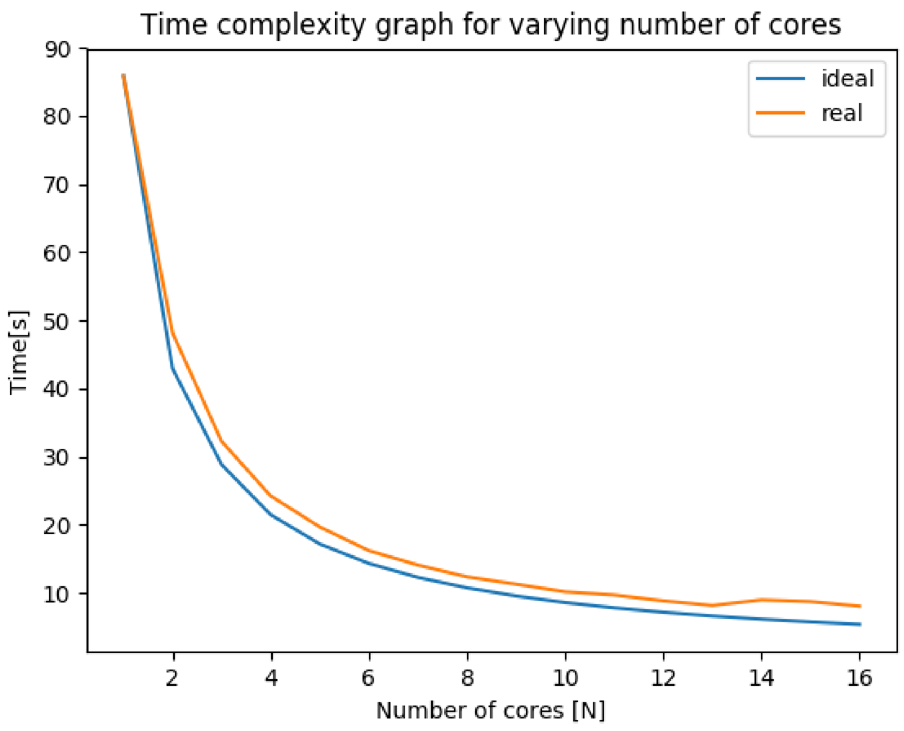

Figure 1: The run time for different number of threads

As the interval increases the result becomes more precise. However a bigger interval also increases the run time. Figure 2 shows how the time increases with the intervals. This was done with 16 cores. One can see that when the interval gets larger than 10000 the run time increases exponentially.

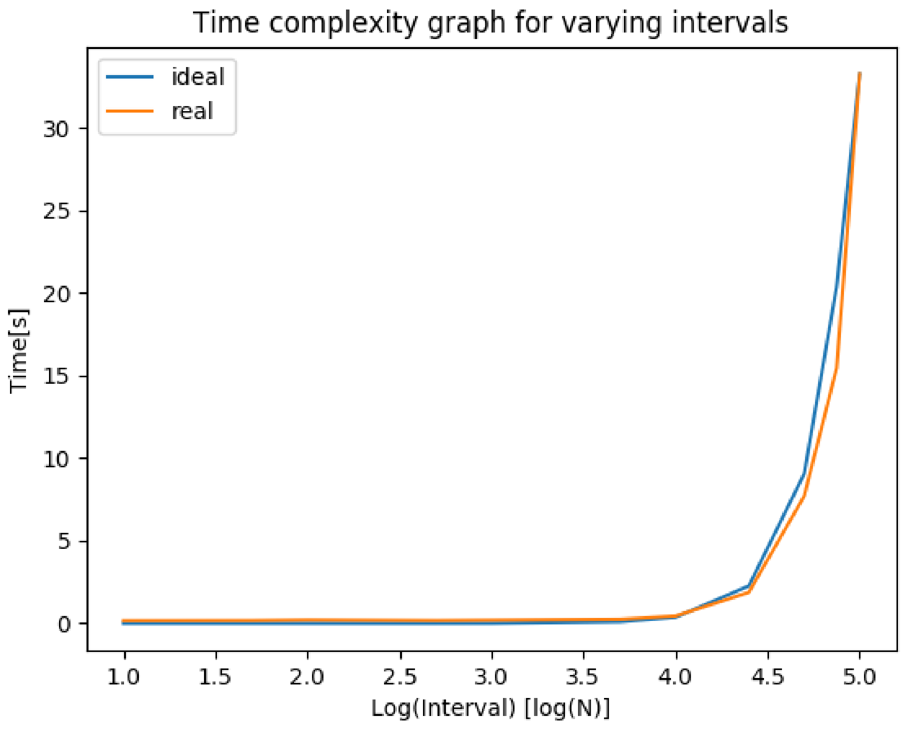

Figure 2: The run time for different problem sizes with 16 threads.