C1544657
CM3103
04/12/2017

High Performance Computing

Coursework 2

Programming with MPI

This report consists of 5 sections. These are:

1) Hardware and Software environment
2) Description of the timing experiments
3) Graphs
4) The Performance Model
5) Conclusion

Hardware:

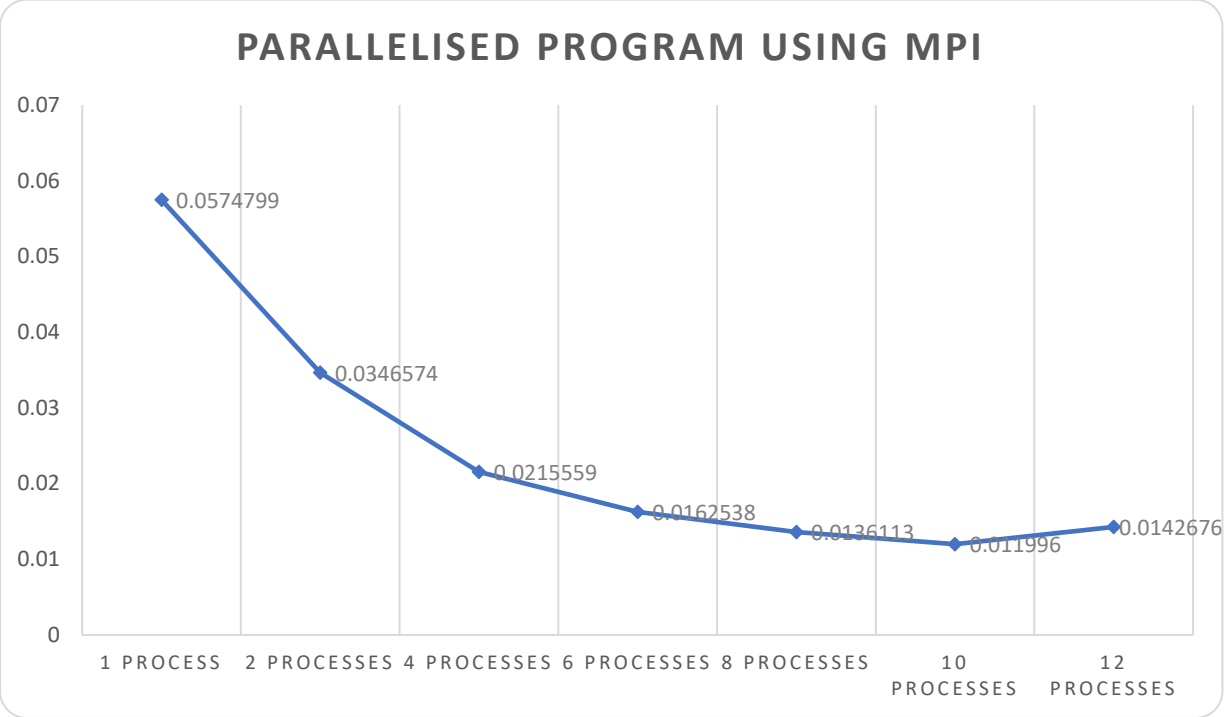
Processor: Intel Core i7 -4790 CPU @3.60GHz Quad Core, 8 threads, 8MB Smart Cache
GPU: GeForce GTX 960, 2048MB Total Memory, 128-bit Memory Interface
RAM: 16GB(2x8GB)

Software:

OS: Linux-x86_64
Editor: Sublime text
Programming Language: C
Software: MPI

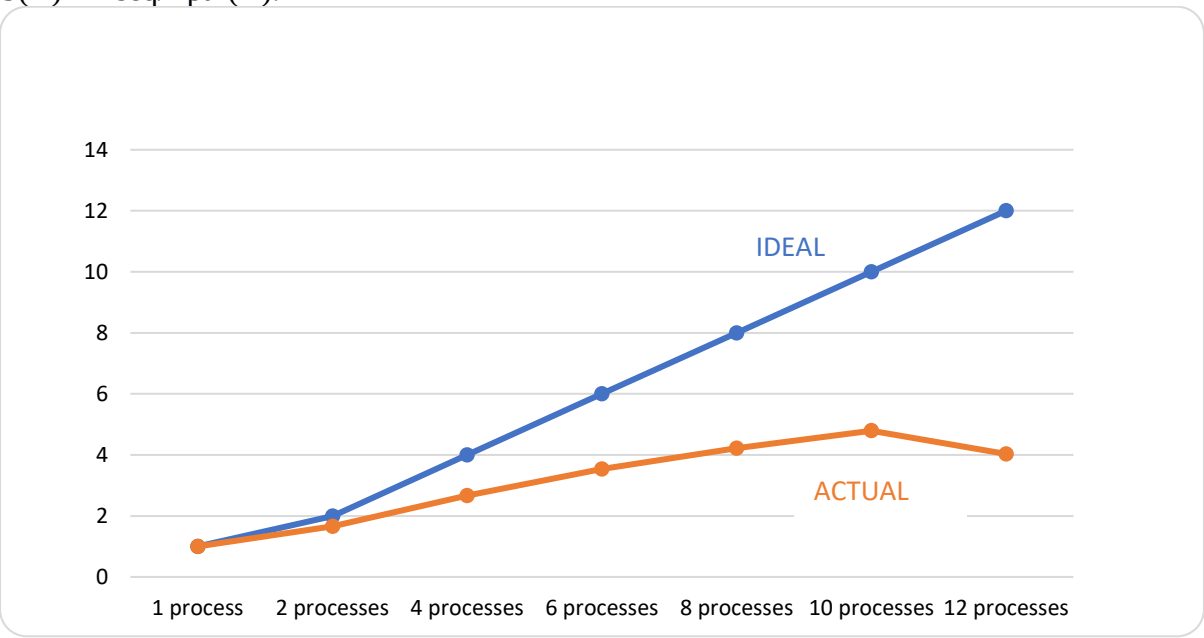
Description of the timing experiments:

Firstly, I parallelised the program, by adding the code in the two sections. I compiled and ran the program without giving it a specific number of processes to run as instructed. This was run 10 times, finding the average and standard deviation of these 10 times. The next step was to run the program again giving it different numbers of processes each time. The numbers of processes I used were 1,2,4,6,8,10 and 12. Each of these was run 10 times and the average and standard deviation was found for each number of processes. I proceeded to create a graph showing these averages. The graph can be seen below.



Speed up graph:

The graph below shows the speed-up of the code as a function of the number of processes. In this graph I have also included the ideal line, as shown in the lectures, to help me understand the difference between the actual and ideal results. The speed up is defined by the equation: S(N) = Tseq/Tpar(N).



C1544657
CM3103
04/12/2017

The Performance Model:

To create this performance model, I went through the examples we went through in the lectures. One example that was similar is the Laplace example. There are some differences too between the Laplace performance model and the one I've created. One of the differences is that the Laplace performance model only works with a square grid, while in our case we can also have a rectangular grid. The update formula requires 4 floating-point operations per grid point in both the Laplace example and our case. For the performance model to be correct, I had to multiply the formula by 3, as we have three arrays this time, unlike the Laplace example which only had 1. These are the Red, Green and Blue.

$$S(N) = 3 * \frac{(4m * n)tcalc}{(4n * m/N)tcalc + (2n/Q)tshift + (2m/P)tshift}$$

The number of grid points per processor shifted in the left/right direction (m/P)
The number of grid points per processor shifted in the up/down direction (n/Q)
Size of the grid (m*n)
Number of processors in one column of the processor mesh(P)
Number of processors in one row of the processor mesh(Q)

As we can see from the graphs, the speedup increases as the number of processes increases, but up to a point. At 12 processes the program begins to perform slower than 10 or 8 processes. The reason for that is that the work given to each process is not maximising the efficiency of the process. More time is taken to distribute the data between the processes, while the time to perform shifts does not decrease. This causes the time to run the program increase.

Conclusion:

Before this project I had minimal experience with the C programming language. This assignment has helped me develop my C programming skills and understand when a program will perform better if run in Parallel with MPI. I gained the skill to be able to convert a code to run parallel with MPI. I understood that as we increase the processes the program performs better. My most important conclusion is that the higher the number of processes does not necessarily mean we're getting the best performance. This could be different with a more process-demanding program, but in our case 10 processes were more than enough.