CM3103: High Performance Computing
Coursework – 2A

This report is for the High Performance Computing (Assignment 2A) which represents and interprets the result of the experiments.

4.
Sol - After running blurMPI.c (with just one processor (n) in a command line)

Input file – David.ps

Output file – DavidBlurMPI.ps (Blur value = 10)

With just one processor (n) in a command line

Computational Times:

1. 0.072032 seconds elapsed
2. 0.070743 seconds elapsed
3. 0.071024 seconds elapsed
4. 0.070803 seconds elapsed
5. 0.070913 seconds elapsed
6. 0.071161 seconds elapsed

Average of these computational times = **0.07111266666**

No. of computational times (N) = 6
Sum of all computational times (S) = 0.426676
mean (u) = 0.07111266666

variance (v) = $1.879642222 * 10^{-7}$

Standard deviation of these computational times = $\sqrt{(v)}$

= $\sqrt{(1.879642222 * 10^{-7})}$

After running blurMPI.c (with number of processors in a command line)

Input file – David.ps

Output file – DavidBlurMPI.ps (Blur value = 10)

No. of processor -1 **No. of processor-2** **No. of processor-6** **No. of processor-8** **No. of processor-10**

No. of processor – 14 **No. of processor - 17**

Computational Times:

1. 0.072215 seconds elapsed
2. 0.036807 seconds elapsed
3. 0.057740 seconds elapsed
4. 0.057740 seconds elapsed
5. 0.053978 seconds elapsed
6. 0.054235 seconds elapsed
7. 0.053300 seconds elapsed

Average of these computational times = **0.055145**
variance (v) = $9.2389886857143 * 10^{-5}$
Standard deviation of these computational times = √(v)
= $\sqrt{(9.2389886857143 * 10^{-5})}$

5.Plot a graph to show the speed-up of the code as a function of the number of processes.

Sol - Calculate Speed-up and Efficiency

The sequential algorithm(blur.c) takes **0.058487 seconds (**from the blur.c**)**

For processor = 1

The parallel algorithm(blurMPI.c) takes **0.072215 seconds** for the 1 processor

So, Speed-up = (0.058487 / 0.072215) = 0.80990099009

And Efficiency = (0.80990099009 / 1) = 0.80990099009

For processor = 2

The parallel algorithm(blurMPI.c) takes **0.036807 seconds** for the 2 processor

So, Speed-up = (0.058487 / 0.036807) = 1.58901839324

And Efficiency = (1.58901839324 / 2) = 0.79450919662

For processor = 6

The parallel algorithm(blurMPI.c) takes **0.057740 seconds** for the 6 processor

So, Speed-up = (0.058487 / 0.057740) = 1.01293730516

And Efficiency = (1.01293730516 / 6) = 0.16882288419

For processor = 8

The parallel algorithm(blurMPI.c) takes **0.057740 seconds** for the 8 processor

So, Speed-up = (0.058487 / 0.057740) = 1.01293730516

And Efficiency = (1.01293730516 / 8) = 0.12661716314

For processor = 10

The parallel algorithm(blurMPI.c) takes **0.053978 seconds** for the 10 processor

So, Speed-up = (0.058487 / 0.053978) = 1.08353403238

And Efficiency = (1.08353403238 / 10) = 0.10835340323

For processor = 14

The parallel algorithm(blurMPI.c) takes **0.054235 seconds** for the 14 processor

So, Speed-up = (0.058487 / 0.054235) = 1.07839955748

And Efficiency = (1.07839955748 / 14) = 0.07702853982

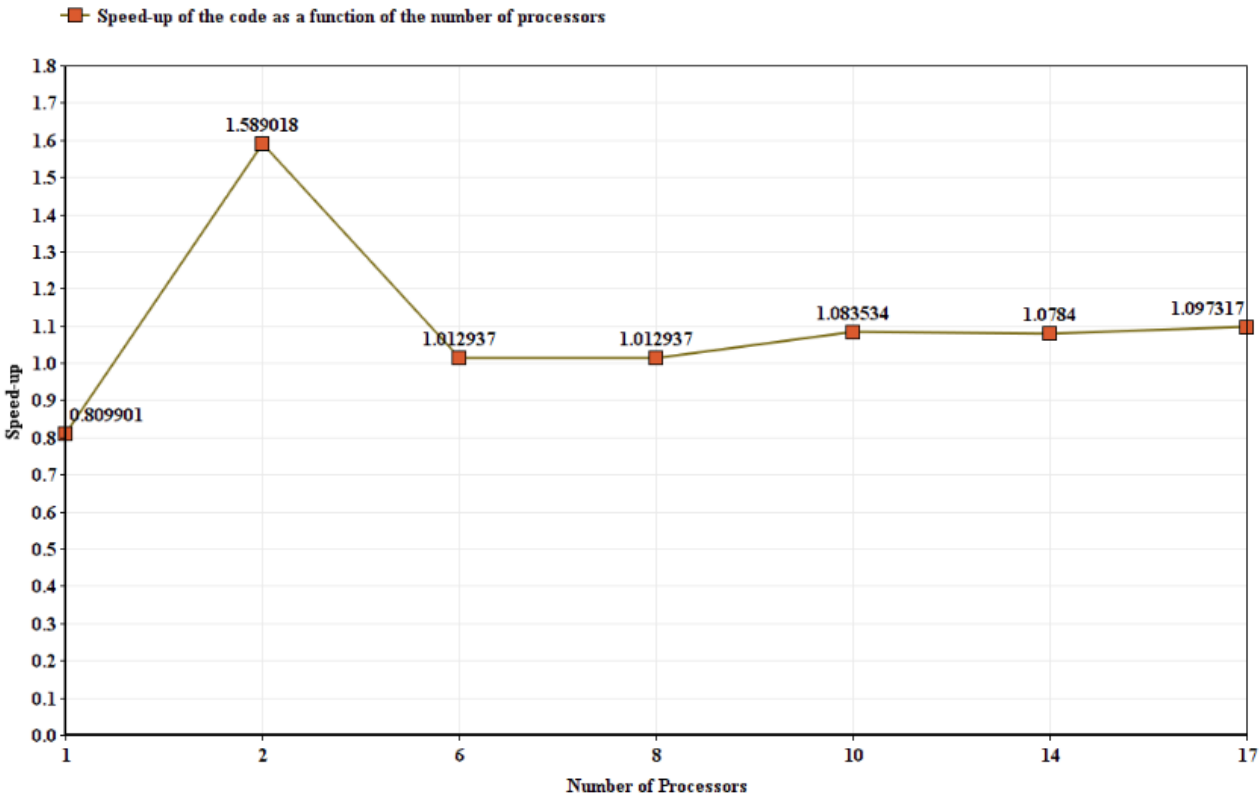
For processor = 14

The parallel algorithm(blurMPI.c) takes **0.053300 seconds** for the 17 processor

So, Speed-up = (0.058487 / 0.053300) = 1.09731707317

And Efficiency = (1.09731707317 / 17) = 0.06454806312

Graph to show the speed-up of the code as a function of the number of processes.



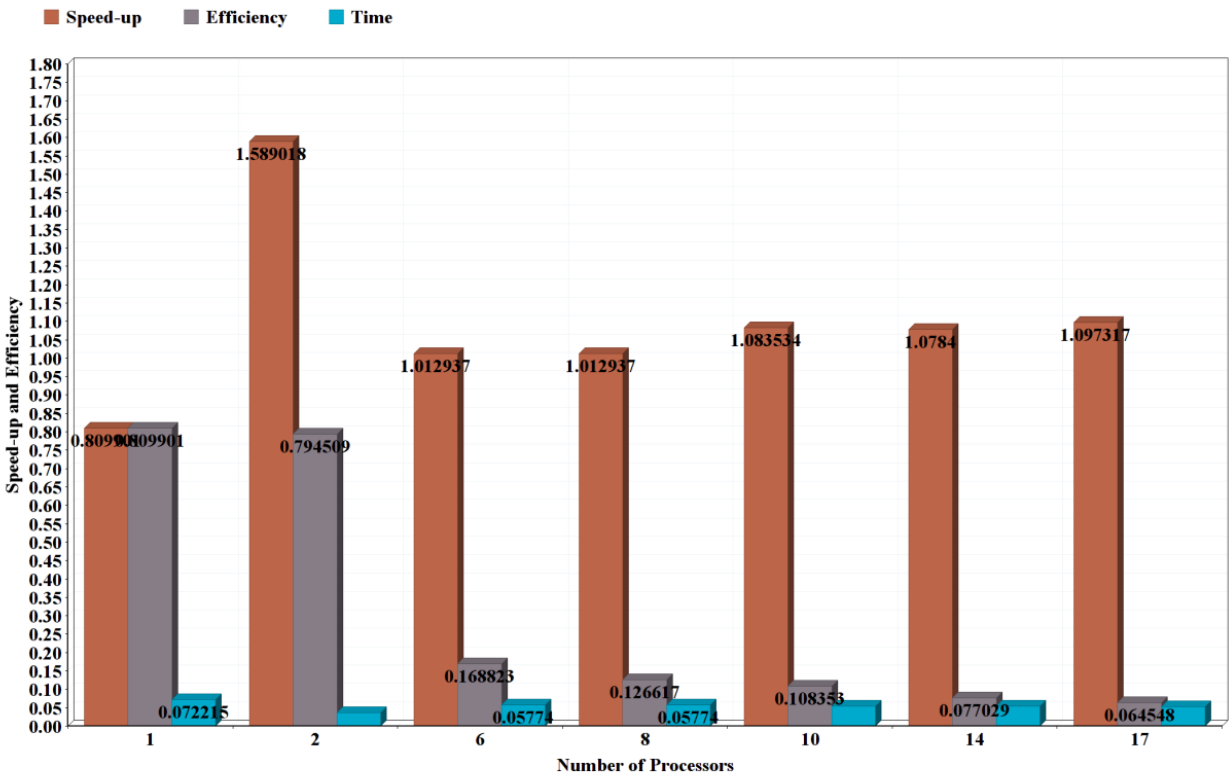
This graph represents the speed-up of the code as a function of the number of processes.
Y-Axis shows speed-up.
X-Axis shows range of the Number of Processors.
From the graph, we can tell how speed-up depends on number of processors.

---

6. Develop a simple performance model for your parallel code that expresses the speed- up as a function of the number of processes.

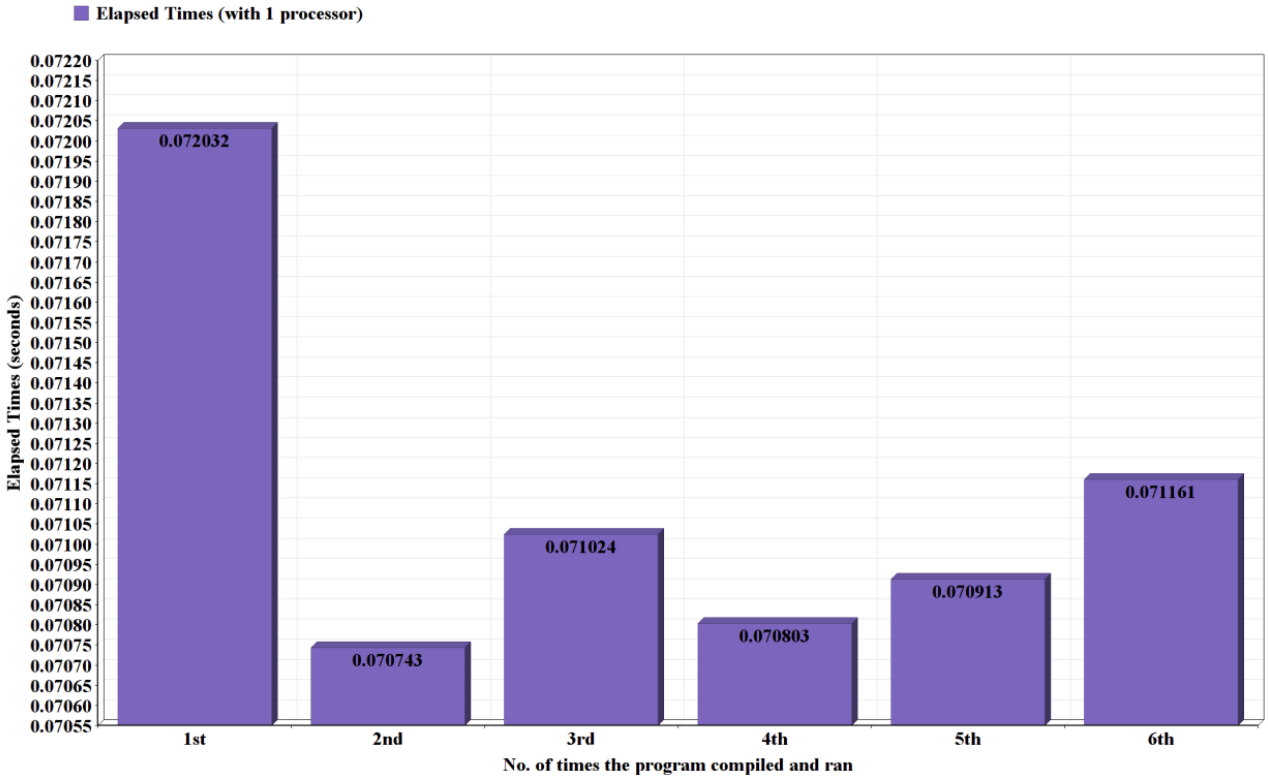
Sol - performance model for MY parallel code that expresses the speed- up as a function of the number of processes as well time taken with different number of processors.

Below graph represents the performance of the parallel application.
Orange bar shows Speed-up, Grey bar shows Efficiency and Blue bar shows Time.

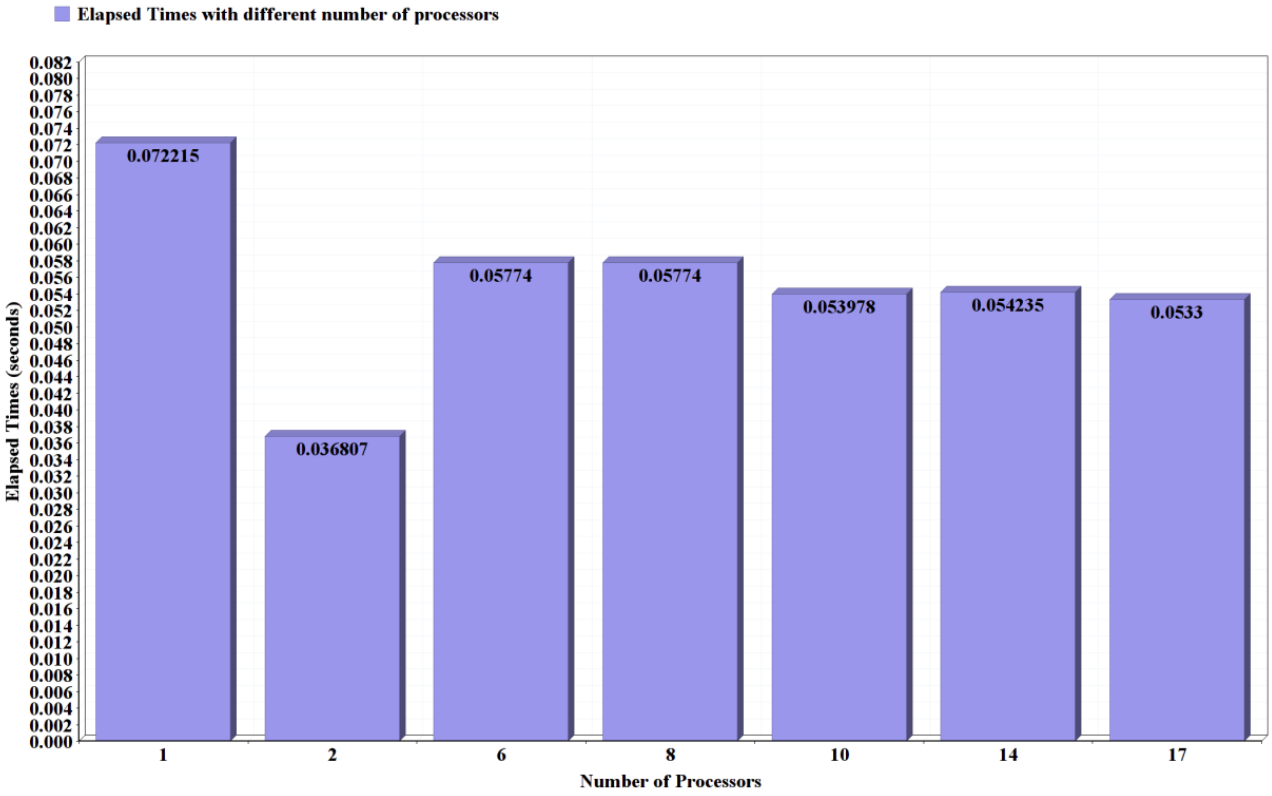


7(a). A description of the hardware and software environment.
Sol -
Hardware Environment - Intel(R) Core(TM) i5-7200U CPU 2.50GHz processor (Quad Core), RAM (8GB), SSD
Software Environment - Linux (OS), Visual Studio(IDE) and Atom(Editor)

7(b). Appropriate graphs of your timing experiments.
Sol -



This graph represents the elapsed times for the each time blurMPI.c has been compiled and ran with only 1 processors
Y-Axis shows all range of the times.
X-Axis shows the number of times program has been compiled and ran.



This graph represents the elapsed times taken against No. of Processors.
Y-Axis shows range of the elapsed times.
X-Axis shows the number of processors.

7(c). The derivation of your performance model, and a comparison of the model of the timings results that accounts for any discrepancies or unusual features.

Sol - In the blurMPI.c while running the program several times with the different number of processors the parallel program gave the output postscript file (DavidBlurMPI.c) with significant amount of changes i.e increase in the blurriness, lines and noises in the output file. The difference between the elapsed times and their speed-ups as well efficiencies because of the different processors are also significant.

7(d). A short section presenting any overall conclusions, and giving a reflection on what you have learned from this coursework.

Sol - From this coursework, I have learned about the MPI which is a library of routines that can be used to create parallel programs of the sequential programs. It is communication protocol for parallel programming. It allows the application to run in a parallel across number of processors. Also learnt, how MPI can effects the computational time of the program compare to sequential one.
In blurMPI.c, I have implemented MPI communication to check the output for different number of processors and their effect on postscript file.