**Document Type:** Logbook Document

**Course:** BEng Hons Computing for Embedded Systems

**Student Number:** 15010892

**Module:** Parallel Computing

Author: Amadeusz Szczypkowski

Date: 11/11/2018

# Logbook structure introduction

In this simple logbook, I will try to contain all of the necessary Action Log information and also share my personal experiences, opinions and difficulties I had to overcome to implement solution program for the problem provided in the Parallel Computing module assignment.  
  
The logbook is structured as follows:  
(Hyperlinks provided for quicker navigation)

* [Research and Design evaluation](#_Research_and_Design) section focuses on my research and design ideas, problems and rough estimates
* [Laboratories](#_Laboratories) section focuses on the core attribute of any logbook, which is log of actions contained in consistently structured
* [Performance Analyses](#_Performance_Analyses) section contains real measurements of how each approach (Serial, Open MP, MPI) has performed
* [Conclusion](#_Conclusion) section focuses on concluding the whole work I did to solve the given problem, and also conclude on my ‘estimates vs real-world’ results
* [References](#_References) section contains all of the relevant referenced materials I have used in this document

# Research and Design evaluation

This sector of my logbook is fully dedicated to the implementation design and research of techniques, which are recommended to use for this module assignment. Most of the technologies were covered either during the lectures or practical sessions by the module leader. The nature and main goal of this research is do deeply analyse proposed technologies, come up with design decisions and build up the confidence for using it all in a real-world coding approach.

## AES encryption and decryption with open SSL

The first focus of actual research is an AES encryption and decryption part of my algorithm. AES (Advanced Encryption Standard) is an algorithm for the symmetry of key encryption. Symmetric key algorithms are cryptographic algorithms that use the same cryptographic keys for both plaintext encryption and ciphertext decryption.  
  
According to my research, cryptographic keys used for AES usually have a fixed length (128 or 256bit keys, for example). Since people cannot easily remember long random strings, key stretching is carried out to create a long, short, fixed key.

For key search simplification purposes, I was recommended by module leader to use shouter 128-bit encryption key with padded ten padded chars, five from the most significant side and five from least significant side.   
According to the key simplification document provided by module leader actual key to be broken should be made about of ASCII charset which would exclude ‘space’ and ‘del’. So, the actual key passed in to encrypt/decrypt should look something like the one presented below.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| # | # | # | # | # | **a** | **a** | **^** | **~** | **?** | **@** | # | # | # | # | # |

## 

## Types of Brute force search

In this sector, I will be focusing at getting good understanding and choosing appropriate method of brute forcing search algorithm.  
After few codding attempts I was certain which type of Brute search should be used for my approach.   
  
I decided to use exhaustive search, which is also known as generate and test. This was probably most of the appropriate solution for my problem because this solving technique consist of systematically listing all possible candidates for the solution and it is checking whether each candidate complies with the problem statement (in case algorithm- the key).  
  
Next step was to make a sensible decision of the code design nature. I had to choose the right approach, which would then allow me later on to reuse the same code for parallelised implementation.

I decided to use functioned nested for loops to generate an alphanumeric sequence ASCII charset combination of 6 symbols long without having the same sequence repairing at different iterations. This choice allowed me having relatively simple but effective code approach which then could be parallelised.

## Open MP

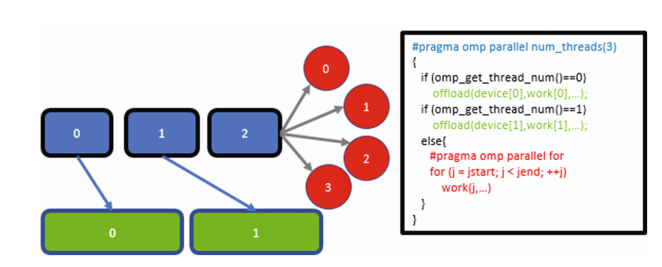
Open MP is the part of solution program, which I am mostly confident and familiar with since it was covered at multiple module tasks and tutorials that I have accomplished.   
  
My understanding of Open MP corelated directly with research and design decisions I’ve made for the sequential approach of my program.   
  
From what I have learnt OpenMP is a programming platform that lets developer to parallel code over a homogeneous shared memory system (for instance multi - core processor). For example, developer could parallel a set of operations over a multi core processor where the cores share memory among themselves. This memory contains cache, RAM, hard disk memory, etc.   
  


Fig. 1. Nested parallel regions with concurrent execution on host and devices. Outer parallel region contains three CPU threads. CPU threads 0 and 1 launch kernels on devices 0 and 1 correspondingly, while CPU thread 2 creates a parallel region with four CPU threads on a host. Research Paper of   
Sourced from: Research Paper of OpenMP [online]., pp. 17-29. [Accessed 03 December 2018].

Communication between mentioned above cores is easy and relatively cheap performance wise, so it doesn’t alter performance. According to my rough estimates it might be the quickest solution.

## MPI

MPI is also the compulsory part of solution program. Even though it was briefly covered in university module associated with this coursework MPI technology seemed quite new to me and I had to revise a little bit more than the other elements of the solution program.   
  
According to the research I have carried out for it and my further understanding of MPI, it is also a programming platform, but instead it gives developer the ability to parallel code over a distributed non-homogeneous system.

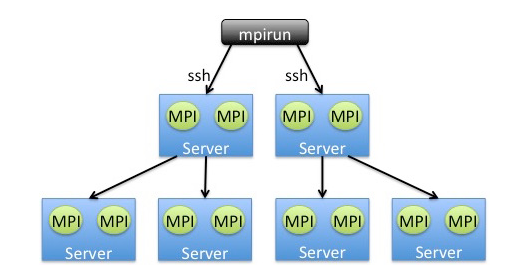


Fig.2 MPI High Level Diagram   
Sourced from: (Cisco Blogs, 2018)

According to the diagram presented above it is possible to parallelise whole solution program over network of computers, clusters or other kind of nodes, which have some processing power and communicate over the same network. Since those mentioned nodes are essentially computers, they will have their own memory layout and town set of cores capable of processing threaded data.   
  
  
My estimation and very brief pre-implementation expectations of the MPI is that it will be slightly slower than Open MP solution because of communication between nodes can be more difficult rather expensive performance wise.

All the mentioned above speculations will be faced with real life code performance analyses placed in the appropriate sector of the log book, where I will be able to reflect on and compare different technologies used it alternative solutions of the algorithm.

# Laboratories

## LAB1 (02/10/2018)

Introduction

Very first task which could be considered as a laboratory session was getting familiar with the task and assignment information on the release day release. Major part of it was dedicated to the research.

Tasks and Lab outcome

As soon as assignment has been released I started collecting information which were strictly necessary to pass the module without hesitation. Initially the main focused of this lab was directed to the development techniques and getting familiar with new technology.

Shortly after getting all of the essential assignment points covered, I focused on reading about the methods, which could allow me to crack the key, which then could be used for decrypting. I bumped in to brute force attack method and got insipid by its exhaustive search methodology. Using brute force attack to crack the key ticked almost all of the boxes mentioned in the assignment, so I decided to try it out.

I didn’t know enough about encryption/decryption, so I decided to leave it for the next lab and then I fully focused on writing sequential program in C, which iterated through ACII char set (with space and delete excluded for simplifications) and produced as many key combinations. One of my advantages was that I could operate at super high frequency by using full potential of my laptops Intel i7 processor.

**Summary**Laboratory achievements:

* Assignment covered
* Research breakdown
* Tasks breakdown
* First part of the research completed
* Simple prototype program for brute force key breaker

## LAB2 (11/11/2018)

Introduction

This laboratory focus was mainly directed at acknowledging encrypt/decrypt and methods associated with it. The goal was to cover the research area and get as much knowledge as possible about the methods and also have some prototype code accomplished as well.

Tasks and Lab outcome

The first part this laboratory was actual research and understanding of what an AES encryption and decryption is and how it could become part of my finalised algorithm in the future. AES (Advanced Encryption Standard) is an algorithm for the symmetry of key encryption and this is what I had to break using brute force attack program prototype developed in the previous lab.

After the research has been completed and I gained good understanding of what AES encryption/decryption method is I decided to use command line approach to encrypt and decrypt my plain text. This helped me to understand the whole cycle much better.

Subsequently after getting familiar with what OpenSSL open source library is and realised that it’s been created for security operations for this purpose I jumped on to coding. First of all, like in case of brute force key cracker mentioned in the previous lab I did create completely isolated separate program that would just simply encrypt and then decrypt back the ‘plain text’ using ‘key’ and ‘iv’. I shortly realised that it would be better approach to use C++ for performance and time management purposes.

**Summary**Laboratory achievements:

* Research breakdown
* Research covered
* Tasks breakdown
* Implementation of the basic encryption decryption algorithm
* Choosing C++ over C, as a main implementation language

## LAB3 (14/10/2018)

Introduction

In this laboratory I focused on merging both previously developed separate programs. I have used my full understanding of both brute force algorithm and encryption/decryption using OpenSSL library.

Tasks and Lab outcome

In the initial state of this laboratory I focused on refactoring the code of my brute force sequential program. I shortly realised that it would have to be rewritten to C++ language since I was already having containers conversions problems and I had to cast most of them to meet data requirements of decryption/encryption program. This could be very problematic in the future and could alter general performance of the problem.  
  
Once I have fully converted my brute force key breaker program to C++ I got rid of most of my problems which were currently making completing my task almost impossible.   
  
The last part of this lab was actual merging two programs together, so then I could have sequential first part of my assignment behind my belt. Unfortunately, I shortly realised that my current brute force key breaker program wasn’t meeting all of the requirements as some of the combinations were repeating and general performance of the program was rather low.

I came up with new solution of having six nested for loops iterating through ASCI char set and generating unique key combinations which then were passed to the decrypt function. Initial text and outcome from decrypt function was continuedly compared and passed back to decryptor.

**Summary**Final year project idea sealed down.

* Tasks breakdown
* C to C++ conversion
* Redesign of key breaker program
* Merging two programs together
* Finished sequential program

## LAB4 (20/10/2018)

Introduction

This lab was fully focused on OpenMP research and Parallel Implementation aspect of the program written in the previous lab.

Tasks and Lab outcome

My initial focus of this lab was directed at researching OpenMP to only a little bit more depth since I have already completed multiple tasks and attended numerous lectures dedicated to OpenMP tasks and structures.  
  
The next step was to transform my sequential solution program to Parallel one using OpenMP. There were many surprising facts I had to go across especially when it comes to performance, but it will all be mentioned in performance analyses.

**Summary**

* Tasks breakdown
* Research covered
* Finished parallel omp program

## LAB5 (03/12/2018)

Introduction

Most of this laboratory was fully dedicated to MPI research and configuring access to the University network cluster. Parallel implementation of algorithm using MPI came at the end.

Tasks and Lab outcome

In the initial state of this laboratory I was concentrating on researching relevant information and materials, which would allow me to fully understand the structure of MPI.  
  
Once I understood it I switched my focus to configuring the network cluster and making the file transfers much easier by for example setting password less ssh and nfs. After communication between nodes was established I have attempted to paralysation of my code. It was a similar approach to openMP technique however the syntax and methodology was completely different.

**Summary**

* Tasks breakdown
* Research covered
* Finished parallel mpi program

# Performance Analyses

This section is dedicated to measurements and performance Analyses of mentioned above algorithm, which has been implemented under 3 different performance technologies, however the main core code of the program remined the same.

## Sequence vs OMP vs MPI

The first criteria taken into account when analysing the performance of the parallel systems is the speed at which the number of times a parallel program works faster than a sequential program and vice versa.   
  
The main reason for parallelizing a sequential program is that it runs the program more quickly. Especially when it comes to overcoming more difficult combinations of the solution key.

According to my observations, performance analyses and execution time measurements, sequential programs might result better at solving simple solution problems, where parallelised programs might take longer time on it.   
Occurrence of this phenomena is caused by resource distribution factor where small task is unnecessarily divided in to separate threads, what in some cases may take more processing power than actual solution to the given problem.  
  
  
Fig. 3 Easy solution key chart  
  
  
As it can be observed from the fig.3 chart, ‘Sequential Run’ of the program (which is a non parallelised version of the same code), found the solution of the problem in the least amount of time and over lowest amount of combinations attempted. This happens simply because parallelised versions of code (OpenMP and MPI) must deal with hardware and resource splitting tasks where, in the same time sequential program, gets ahead since compiler don’t have to deal with additional instructions and data sharing.

On the other hand, programs with parallelised code sectors were the only ones that managed to resolve the problem of difficult solution keys. This is where the power of parallelism comes in to place and over acceleration initial acceleration advantage of the sequential program.

Fig. 4 Difficult solution key chart  
  
The chart presented in fig.4 proves my hypothesis introduced in above paragraph. Parallelised programs managed to complete introduced task and sequence program was still continuing. Solving this problem by sequential program would take extremely long time, hence why execution has been cancelled by my when the algorism reached 55 million combinations attempted.  
  
According to my estimates and previous calculations, solution to the hard key cold take up to eight days for he sequential program. Parallelised programs (openMP, MPI) took reasonable time to match the key that’s why they could continue and finish their work which is reflected on the mentioned diagram of fig.4.  
  
As a conclusion I would like to mention that in order to obtain a faster program, we need to minimize the unparalleled chunks of code, to ensure the load balance of the tasks at the processor core is reasonable. This is mainly to minimize the time dedicated to communication and synchronization at the very beginning of the program. That’s where sequential program was stepping up rapidly and parallelized copies of the same program were taking time before acceleration or didn’t have a time to fully accelerate as the problem was relatively simple.

# References

1. Grinberg, L., Bertolli, C. and Haque, R. (2017) Hands on with Openmp4.5 and Unified Memory: Developing Applications For Ibm’s Hybrid Cpu + Gpu Systems (Part Ii). *Research Paper of OpenMP* [online]., pp. 17-29. [Accessed 03 December 2018].
2. Squyres, J. / Cisco (2015) *Cisco Blogs* Available [online] from: https://blogs.cisco.com/performance/tree-based-launch-in-open-mpi [Accessed 04 December 2018].