University of the West of England
BRISTOL

MODULAR PROGRAMME

COURSEWORK ASSESSMENT SPECIFICATION

Module Details

Module Code UFCFFL-15-M	**Run**	**Module Title** PARALLEL COMPUTING
Module Leader Kun Wei	**Module Coordinator**	**Module Tutors**
Component and Element Number B:CW1		**Weighting: (% of the Module's assessment)** 60%
Element Description LOGBOOK (Logbook and Demonstration of Final Product - (1500-3000 Words))		**Total Assignment time**

Dates

Date Issued to Students 02/10/2019	**Date to be Returned to Students**
Submission Place **- BlackBoard (Logbook)** **- UWE GitLab (program code)**	**Submission Date** **29/04/2021**
	Submission Time **14:00hrs (GMT)**

Deliverables

1. A Logbook of 1500-3000 words.
2. A compressed file of program code

Module Leader Signature

Dr Kun Wei

1. Overview

This assignment assesses the following module learning outcomes:

- Critically evaluate the effectiveness of parallel computation in homogenous and heterogeneous environments;
- Develop programs for parallel systems, e.g. using OpenMP and MPI for multicores, and OpenCL for accelerators if possible.
- Understand and develop a parallel design and algorithm.

This is an *individual* assignment, worth ***60%*** of the overall mark for the module. It is split into two components, a research *logbook* and a *development project*. It requires students to design and implement a selection of parallel algorithms for computing the problem described in Section 2. In addition to implementing an optimized set of parallel implementations for the problem, it is expected that a successful completion will include a benchmarking analysis of the implemented algorithm, compared against a "golden" serial version.

20% of the 60% marks obtainable from this assignment are for the companion logbook. The logbook is expected to be between 1500 – 3000 words and is a reflective account of the development process, research into different approaches to parallelising sequential algorithms, both from material taught as part of the course, but also from research material, e.g. papers.

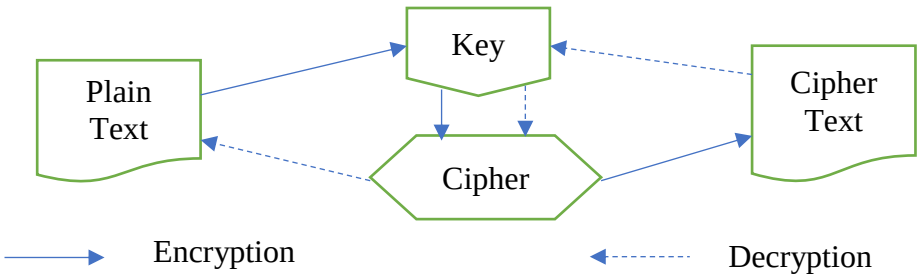
More details of the assignment is described in Section 2.

Working on this assignment will help understand modern multi-core and many-core architectures, parallelise sequential algorithms, thinking concurrently, and apply academic research in a practical context.

2. Task Specification

This section introduces the details of the problem chosen for implementing parallel algorithms as part of this assessment point. The chosen problem is how to find the most efficient way to determine the key to encrypt and decrypt textual messages passed across communication lines.

An encryption/decryption process takes a plaintext and a key expression converts into an encrypted text, which is also called as ciphertext. The process is sketched in the following diagram.



Advanced Encryption Standard (AES) is known to be one of the-state-of-the-art encryption/decryption method used by many prestigious companies e.g. NSA, Microsoft and Apple, which uses a symmetric key generating approach to achieve very highly secured messaging and access across the networks. More details on how it works can be found in the literature including the following link;

https://thebestvpn.com/advanced-encryption-standard-aes/

There are a number of tools developed to work out security operations including encryption/decryption procedures. **OpenSSL** is an open source library created for security operations for this purpose. It is developed in C programming language and interfaced with various other programming languages too. This library uses SSL and TSL protocols and includes a number of cryptographic functions. The following two links provide further details on use of **OpenSSL** and the ways how to incorporate with self-developed C programs for encryption/decryption purposes.

- https://eclipsesource.com/blogs/2017/01/17/tutorial-aes-encryption-and-decryption-with-openssl/
- https://www.openssl.org/docs/man1.0.2/crypto/EVP_EncryptInit.html

Brute-force attack is one of cipher cracking attacks, which uses an exhaustive search to find the key to crack the message encrypted. Brute force search is a global search approach relies on trial-and-error operations, where an input set is assumed and tested whether or not it produces the targeted output. There are a number of well-known brute-force search algorithms used and well-evaluated with respect to a number of performance measures including complexity. (1) Generate and test, (2) Breath-First, and (3) Depth-first search algorithms are three of these well-known Brute-force (exhaustive) search algorithms. Further details can be found in the literature including Wikipedia.

As an ethical hacking operation, a **brute-force search** algorithm can be implemented incorporating with OpenSSL library, given that both a Plaintext and a Ciphertext can be provided while the secure key used in encryption can be searched for with the developed Brute-Force algorithm.

Obviously, an exhaustive search developed with any Brute-force algorithm will take very long time to find the secure key for crack of the cipher, AES this time, using a sequentially developed Brute-force algorithm. This creates a good motivation to go for parallelisation of Brute-force search algorithms. ***The main requirement of this assignment is to design parallel Brute-force search algorithms for finding the right secure key to crack the ciphers using both shared memory and distributed memory paradigms with OpenMP and MPI parallelisation tools, respectively.***

3. Requirements

a. Problem set 1: OpenMP (20%)

Implement a parallelised version of Brute-force search, which behaves the same as the serial solution for cracking ***aes-128-cbc*** cipher. The implementation should use ***OpenMP*** for parallelization of the algorithm.

Your solution should be implemented using C/C++, compiled with ***gcc***. Your final solution should include a Makefile that can be used by makers to build and test your work.

Your solution must be committed in UWE's Gitlab. It will naturally be time stamped and you must be careful to not make commits after the submission deadline.

b. Problem set 2: MPI (20%)

Implement a parallelised version of Brute-force search, which behaves the same as the serial solution for cracking ***aes-128-cbc*** cipher. The implementation should use ***MPI*** for parallelization of the algorithm.

Your solution should be implemented using C/C++, compiled with **gcc.** ***Open MPI*** is installed on own PCs as illustrated during the lab session, or PCs in the Q block labs. Your final solution should include a Makefile that can be used by makers to build and test your work.

Your solution must be committed in UWE's Gitlab. It will naturally be time stamped and you must be careful to not make commits after the submission deadline.

c. Problem set 3: Benchmarking and performance analysis (20%)

For this component you are expected to perform an analysis of the serial solution and compare and contrast this against your solution problem set 1 and 2.

The analysis should be reproducible and you should include instructions on how to execute the solutions, including the serial one provided, to produce the performance results. Of course, it is not expected that during testing the same performance numbers will be produced, but they should be within a reasonable margin of error.

You are free to include this in your logbook, but it does not count against the word count, and it might be easier to produce a separate document with the presentation of the performance numbers and analysis of the results.

4. Deliverables

a. Each problem solution, OpenMP and MPI implementations, should be committed in its own directory in UWE's Gitlab, including a README.md documenting what the program does, how to build it, and benchmark the resulting executable(s).
b. Additionally, the Git project should include a folder with documentation for your project and its solution, including the logbook, which details the performance analysis, the approach to benchmarking, and so on.
c. Logbook of 1500 – 3000 words. The logbook is for you to document your design and development process, along with a performance analysis of your solutions to Brute-Force search for cipher cracking. It is not specified what shape your logbook should take, but it should be academically rigorous, including academic references supporting any claims made, as well as to provide context, and so on. **The logbook should be submitted as a word or PDF file to Blackboard. In addition, the log book should include a link to your Git project for this module.**

5. Marking Criteria

The assignment will be marked with the following criteria

	0-29	30-39	40-49	50-59	60-69	70-84	85-100	Mark & Advice for Improvement
Functionality 30%	Application delivers less than 30% of the required functionality	Application delivers 30-50% of required features as specified in the requirements	Application delivers 51 to 60% of required features as specified in the requirements	Application delivers 61 to 70% of required features as specified in the requirements	Application delivers 71 to 80% of required features as specified in the requirements	Application delivers over 80% of required features as specified in the requirements	Application delivers all of the required features and goes beyond the requirements in such a way as to improve on what is specified.	
Performance analysis 30%	Little or no analysis is provided .	Analysis does little to aid understanding of performance behaviour of both the provided serial program and implemented parallel variant.	Analysis presentation is not and precise. References to methodology used and background context for benchmarking is not detailed or even included.	Analysis presentation is clear and precise. References to methodology used and background context for benchmarking and so on but less than 70% complete	Analysis presentation is clear and precise. References to methodology used and background context for benchmarking and so on but less than 71% - 85% complete	Analysis presentation is clear and precise. References to methodology used and background context for benchmarking and so on and is more than 85% complete	Analysis presentation is clear and precise. References to methodology used and background context for benchmarking and so and 100% complete and to a professional standard throughout.	
Implementation 35%	Application uses example code with minimal changes OR no attempt has been made to	Programming standards, application structures, and user interface design standards	Application inconsistently applies programming standards and user interface	Application consistently applies programming standards and user interface	Application structure is clean and matches standards	Expertly uses the programming techniques and programming and interface design	Utilises programming techniques and constructs that were not explicitly taught in the module	

	use programming standards	are not applied consistently	design standards	design standards		standard that were taught in the module.		
Internal Documentation (Comments etc) 5%	No internal documentation	Little OR inconsistent internal documentation	Internal documentation does little to aid understanding of the code	Internal documentation is helpful but inconsistent with the standards taught AND insufficient	Internal documentation is helpful but inconsistent with the standards taught OR insufficient	Internal documentation is helpful and mostly consistent with the standards taught	Internal documentation is helpful and wholly consistent with the standards taught	