Parallel Computing:

CW1: Assignment 2

LOGBOOK:

Logbook and Demonstration of Final Product

**By Michael Onileowo**

**Table of contents**

1. Table of contents 2
2. Introduction 3
3. Brute force 4
4. OpenMP 5
5. MPI 6
6. References 9
7. **Introduction**.

The problem to work on is as described below.

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.

Key

Cipher Text

Plain Text

Cipher

Encryption

Decryption

The task is to write a serial program and then parallelize it using OpenMP and MPI implementations and then measure the performance achieved in all scenarios and be able to highlight the speed up (if any) that was achieved when parallelization was used.

1. **Brute force**

A brute force [1] attack is a trial-and-error method used to obtain information such as a user password or personal identification number (PIN). In a brute force attack, automated software is used to generate a large number of consecutive guesses as to the value of the desired data. Brute force attacks may be used by criminals to crack encrypted data, or by security analysts to test an organization's network security.

A brute force attack is also known as brute force cracking or simply brute force.

In our case we will be using the Brute Force algorithm to detect the key that was used to encrypt our data.

Obviously, an exhaustive search developed with any Brute-force algorithm will take very long time to find the secure key for crack the cipher, AES this time, using a sequentially developed Brute-force algorithm.

One example of a type of brute force attack is known as a dictionary attack, which might try all the words in a dictionary. Other forms of brute force attack might try commonly-used passwords or combinations of letters and numbers.

1. **OpenMP**

OpenMP [2] is a library that supports shared memory multiprocessing. The OpenMP programming model is SMP (symmetric multi-processors, or shared-memory processors): that means when programming with OpenMP all threads share memory and data.

Parallel code with OpenMP marks, through a special directive, sections to be executed in parallel. The part of the code that’s marked to run in parallel will cause threads to form. The main tread is the master thread. The slave threads all run in parallel and run the same code. Each thread executes the parallelized section of the code independently. When a thread finishes, it joins the master. When all threads finished, the master continues with code following the parallel section.

Each thread has an ID attached to it that can be obtained using a runtime library function (called omp\_get\_thread\_num()). The ID of the master thread is 0.

OpenMP supports C, C++ and Fortran.

The OpenMP functions are included in a header file called

The OpenMP parts in the code are specified using #pragmas

OpenMP has directives that allow the programmer to:

specify the parallel region (create threads)

specify how to parallelize loops

specify the scope of the variables in the parallel section (private and shared)

specify if the threads are to be synchronized

specify how the works is divided between threads (scheduling)

OpenMP hides the low-level details and allows the programmer to describe the parallel code with high-level constructs, which is as simple as it can get.

I will be using some (and possibly more) of the above solution to parallelize my serial code (from brute force) and be measuring the performance against one another to highlight efficiency of parallelism.

1. **MPI**

MPI (Message Passing Interface) is a specification for the developers and users of message passing libraries. By itself, it is NOT a library - but rather the specification of what such a library should be.

MPI primarily addresses the message-passing parallel programming model: data is moved from the address space of one process to that of another process through cooperative operations on each process.

Simply stated, the goal of the Message Passing Interface is to provide a widely used standard for writing message passing programs. The interface attempts to be:

* Practical
* Portable
* Efficient
* Flexible

The MPI standard has gone through a number of revisions, with the most recent version being MPI-3.x

Interface specifications have been defined for C and Fortran90 language bindings:

* C++ bindings from MPI-1 are removed in MPI-3
* MPI-3 also provides support for Fortran 2003 and 2008 features

I will also parallelize my serial code using MPI and measuring its performance and will also measure MPI’s performance against that of the OpenMP.

This is to help evaluate when and where parallelism is efficient and how to go about implementing them.

20/11/2018 10:20am

Took a sample program for encryption from OpenSSL Wiki website (<https://wiki.openssl.org/index.php/EVP_Symmetric_Encryption_and_Decryption>) to customise for this course work

Message to be encrypted is “This is a really really top secret!”

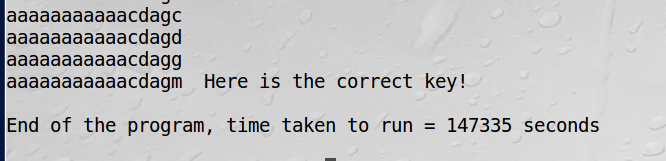
IV used is "\x01\x02\x03\x04\x05\x06\x07\x08"

Key used is "aaaaaaaaaaadagm"

I had to use a simplified key at the moment as I will like the program to run quickly so I can measure the time taken and then begin to make it more complex to find the key used.

22/11/2018 12:10pm

Key was changed to “aaaaaaaaaaacdagm” to make it a bit more complex and extend the runtime. More tiding up and error correction were done in my brute force program and it compiles well but I am unable to get the time to work correctly.



Will spend more time to correct this and then move to implement openMP and MPI.

24/11/2018 14:30pm

I found out I was having error with my code returning multiple (about 10) keys of the same value.

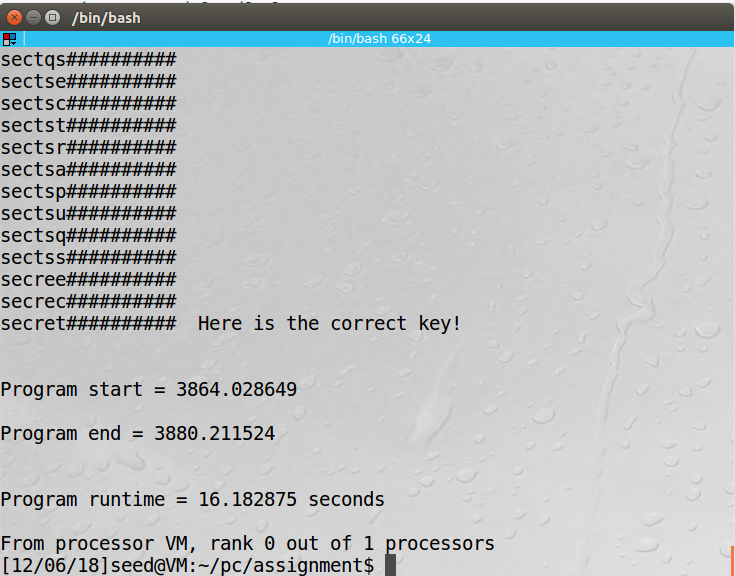
This was resolved when I found out that the 16 bytes but the alphabet used at that point in time was 5 bytes, hence my for loops should not be for 16 but for 5. This made my code even run faster as the bottleneck of repetition has been removed.

I will be increasing the number bytes in my alphabets so that I can have a longer key and the be able to measure performance better in openMP and MPI.



30/11/2018 14:23

sudo apt-get install libopenmpi-dev and sudo apt-get install openmpi.bin was used to install mpi library



**References**.

[1]. <https://www.techopedia.com/definition/18091/brute-force-attack>

[2]. <http://www.bowdoin.edu/~ltoma/teaching/cs3225-GIS/fall16/Lectures/openmp.html>