Group Project Design Document

CP-631-OC1 Pawel Pomorski

WiLFRID LAURIER UNIVERSITY

Chunxiang Zhang & Vishnu Sukumaran

  Group 2

Contents

[**1: Introduction** 1](#_Toc14266292)

[**2: Problem** 2](#_Toc14266293)

[**3: Overview of the Design** 3](#_Toc14266294)

[**4: The Scope of the Job** 3](#_Toc14266295)

[**5: System Design** 4](#_Toc14266296)

[**6: Design in Detail** 4](#_Toc14266297)

[**7: User Interface** 4](#_Toc14266298)

[**8: Non-Functional Requirements** 4](#_Toc14266299)

[**9: Explanation of Testing and Performance** 4](#_Toc14266300)

[**10: Deliverables** 4](#_Toc14266301)

Version Control History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Changes |
| 0.1 | 17/07/2019 | V Sukumaran | Initial draft of skeleton and Design definition |
|  |  |  |  |
|  |  |  |  |

## **1: Introduction**

The purpose of this document is to address the steps taken to find the best possible solution to a real world problem through parallel computing programming stacks like Message Passing Interface (MPI), Compute Unified Device Architecture (CUDA) and also OpenMP.

Message Passing Interface: It is a language-independent communications protocol. It is portable, platform independent, and is the de facto standard for parallel computing on distributed memory systems. MPI is not a new programming language. It is a collection of functions and macros, or a library that can be used in C programs (also C++, FORTRAN, Python etc.) Most MPI programs are based on SPMD model - Single Program Multiple Data. This means that the same executable runs in a number of processes, but the input data makes each copy compute different things.

OpenMP offers a way around this by offering a more restricted set of options for thread operations, controlled by compiler pragmas.  In this way, the compiler itself generates threaded code in the executable following the relatively simple pragma directive issued by the programmer. This offers less flexibility than working with threads directly, but in exchange it makes programming easier and increases program reliability.

Compute Unified Device Architecture (CUDA): Nvidia released the CUDA API for programming its Graphics Processing Units (GPU). It allows the programmer to treat the GPU as a general computing device, without any explicit reference to graphics. CUDA enables developers to speed up compute-intensive applications by harnessing the power of GPUs for the parallelizable part of the computation.

By implementing solution on parallel computing, the project team is trying to make programs more efficient and better in performance to achieve its goal.

Advantages of Parallel Computing over Serial Computing are as follows:

* It saves time and money as many resources working together will reduce the time and cut potential costs.
* It can be impractical to solve larger problems on Serial Computing.
* It can take advantage of non-local resources when the local resources are finite.
* Serial Computing ‘wastes’ the potential computing power, thus Parallel Computing makes better work of hardware.

## **2: Problem**

Five largest gaps between prime numbers.

*A prime number (or a prime) is a natural number greater than 1 that cannot be formed by multiplying two smaller natural numbers.*

For example, say the gap between consecutive prime numbers 17 and 19 is equal to 2. The gap between consecutive prime numbers 37 and 41 is equal to 4. Find the best solution to determine the largest 5 gaps between a pair of consecutive prime numbers, up to 10^9 (1,000,000,000).

## **3: Overview of the Design**

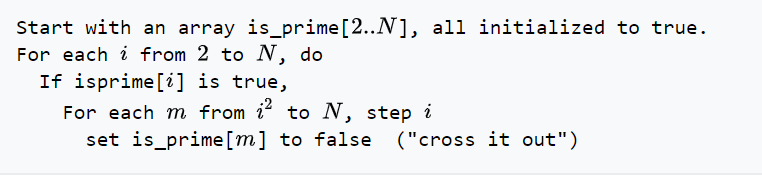
Upon research, Sieve of Eratosthenes algorithm is a simple, ancient algorithm for finding all prime numbers up to any given limit. It does so by iteratively marking as composite (i.e., not prime) the multiples of each prime, starting with the first prime number, 2.

Given a number n, print all primes smaller than or equal to n. It is also given that n is a small number. Following is the algorithm to find all the prime numbers less than or equal to a given integer n by Eratosthenes’ method:

* Create a list of consecutive integers from 2 to n: (2, 3, 4… n).
* Initially, let p equal 2, the first prime number.
* Starting from p2, count up in increments of p and mark each of these numbers greater than or equal to p2 itself in the list. These numbers will be p (p+1), p (p+2), p (p+3), etc.
* Find the first number greater than p in the list that is not marked. If there was no such number, stop. Otherwise, let p now equal this number (which is the next prime), and repeat from step 3.

When the algorithm terminates, all the numbers in the list that are not marked are prime.

Below is a sample pseudocode for the algorithm



## **4: The Scope of the Job**

The scope of this group project mainly consists of four main aspects

1. Write an MPI parallel program
2. Write a program combining MPI and OpenMP
3. Implement the above programs by using CUDA on GPU.
4. Analyze above programs and performance to decide the best way to execute this problem.

## **5: System Design**

The architecture, interaction and data structures need explaining, as does the database and approaches towards procedures.

## **6: User Interface**

Since most of the output is captured in command prompt screen, it is designed to look like the below format.

The largest gap is XXXX1 between prime number XXXXa and prime number XXXXb.

The 2nd largest gap is XXXX2 between prime number XXXXc and prime number XXXXd.

The 3rd largest gap is XXXX3 between prime number XXXXe and prime number XXXXf.

The 4th largest gap is XXXX4 between prime number XXXXg and prime number XXXXh.

The 5th largest gap is XXXX5 between prime number XXXXi and prime number XXXXj.

*Note: All the prime numbers (XXXXa, XXXXb, XXXXc, XXXXd, XXXXe, XXXXf, XXXXg, XXXXh, XXXXi, XXXXj) should be in the range [1:10^9].*

## **8: Output and Performance comparison**

|  |  |  |  |
| --- | --- | --- | --- |
| File Name | Start Time | End time | Efficiency (Compared with Serial program) |
|  |  |  |  |
|  |  |  |  |

## **9: Deliverables**

|  |  |  |  |
| --- | --- | --- | --- |
| Deliverable ID | Deliverable File Name | File Type | Steps to Compile and Run |
| D1.1 | CP631\_Final\_serial.c | .c | Compile: gcc -O2 CP631\_Final\_serial.c -o CP631\_Final\_serial.x  Run: ./CP631\_Final\_serial.x |
| D2.1 | CP631\_Final\_cuda.cu | .cu |  |
|  |  |  |  |