

**香港中文大學**

**計算機科學及工程學系**

**Department of Computer Science and Engineering,**

**The Chinese University of Hong Kong**

**Title**

Large Scale Data Mining Using Super Computer

**Name**

**Chu Pak Sum 1155068765**

Supervised By

**Prof. LEUNG Kwong-Sak**

©2016 The Chinese University of Hong Kong

The Chinese University of Hong Kong holds the copyright of this proposal.

Any person(s) intending to use a part or whole of the materials in the thesis

in a proposed publication must seek copyright release from the University.

# Acknowledgement

First and foremost my deepest gratitude to my supervisor, Prof. Leung Kwong-Sak for his supervision and guidance throughout this project.

I would like also to express my sincere appreciation to Mr. Leo Liu and Mr. Kester Lee for all their kindness and help.

# Abstract

Accelerating data mining algorithm by GPU parallel computing with super computer is renowned for its significant performance improvement. Large scale data mining is computationally intensive in nature, which is a perfect candidate for GPU acceleration.

We specifically selected P-Value, T-Test, Mutual Information and ReliefF as their data structures are good for data parallelism, and thus suitable for GPU acceleration.

With the help of CUDA platform, we can implement GPU parallel computing on NVIDIA display cards efficiently. We also conducted extensive experiments in order to find out how to accelerate our selected algorithms as well as measure how much their performance be improved.

Although the performance of GPU acceleration is not fully optimized, we found that the approach still can accelerate some of the algorithms more than 1,000 times. Showing that GPU parallel computing is promising in data mining area.

Contents

[Acknowledgement 2](#_Toc449873317)

[Abstract 3](#_Toc449873318)

[1. Introduction 7](#_Toc449873319)

[1.1. Motivation 7](#_Toc449873320)

[1.2. Objectives 8](#_Toc449873321)

[2. Background 9](#_Toc449873322)

[2.1. Large Scale Data 9](#_Toc449873323)

[2.2. Data Mining - Feature Selection 9](#_Toc449873324)

[2.3. Super Computer 9](#_Toc449873325)

[2.4. Performance of Common Data Mining Tools 10](#_Toc449873326)

[2.5. Popularity of GPU Acceleration 10](#_Toc449873327)

[2.6. CUDA framework and Programming 11](#_Toc449873328)

[2.7. NVIDIA Visual Profiler 13](#_Toc449873329)

[3. User Specifications 14](#_Toc449873330)

[3.1. GPU Accelerated Data Mining Algorithm Library 14](#_Toc449873331)

[3.2. GPU Accelerated Feature Selection Engine 15](#_Toc449873332)

[3.2.1. Framework 15](#_Toc449873333)

[3.2.2. Data set 16](#_Toc449873334)

[4. Design Specifications 17](#_Toc449873335)

[4.1. Design Principle 17](#_Toc449873336)

[4.2. GPU Accelerated Data Mining Algorithm Library 17](#_Toc449873337)

[4.3. GPU Accelerated Feature Selection Engine 18](#_Toc449873338)

[4.4. GPU Acceleration Performance Testing Tool 19](#_Toc449873339)

[5. Detail Design 20](#_Toc449873340)

[5.1. GPU Accelerated Data Mining Algorithm Library 20](#_Toc449873341)

[5.1.1. Simple Algorithm Processors 22](#_Toc449873342)

[5.1.2. GPU Accelerated Algorithm Processors 22](#_Toc449873343)

[5.1.2.1. GPU Accelerated P-Value Processor 24](#_Toc449873344)

[5.1.2.2. GPU Accelerated T-Test Processor 27](#_Toc449873345)

[5.1.2.3. GPU Accelerated ReliefF Processor 28](#_Toc449873346)

[5.2. GPU Accelerated Feature Selection Engine 29](#_Toc449873347)

[5.3. Performance Testing Tool 31](#_Toc449873348)

[6. Implementation 32](#_Toc449873349)

[6.1. Development Environment 32](#_Toc449873350)

[6.2. Profiling 33](#_Toc449873351)

[6.3. Performance Tuning 34](#_Toc449873352)

[7. Test and Validation 37](#_Toc449873353)

[7.1. P-Value 37](#_Toc449873354)

[7.2. T-Test 38](#_Toc449873355)

[7.3. Mutual Information 38](#_Toc449873356)

[7.4. ReliefF 38](#_Toc449873357)

[8. Results and Evaluation 39](#_Toc449873358)

[8.1. GPU Acceleration Data Size Limitation 39](#_Toc449873359)

[8.1.1. Big Feature Size Data 39](#_Toc449873360)

[8.1.2. Big Sample Size Data 39](#_Toc449873361)

[8.2. Performance of GPU Accelerated Algorithms in Different Thread Numbers 40](#_Toc449873362)

[8.2.1. GPU Accelerated P-Value Processor 41](#_Toc449873363)

[8.2.2. GPU Accelerated T-Test Processor 42](#_Toc449873364)

[8.2.3. GPU Accelerated Mutual Information Processor 43](#_Toc449873365)

[8.2.4. GPU Accelerated ReliefF Processor 44](#_Toc449873366)

[8.3. Performance of GPU Accelerated Algorithms in Different Sample Size and Feature Size 45](#_Toc449873367)

[8.3.1. GPU Accelerated P-Value Processor 46](#_Toc449873368)

[8.3.2. GPU Accelerated T-Test Processor 47](#_Toc449873369)

[8.3.3. GPU Accelerated Mutual Information Processor 48](#_Toc449873370)

[8.3.4. GPU Accelerated ReliefF Processor 49](#_Toc449873371)

[8.4. Performance Comparisons between 1 CPU and 1 GPU 50](#_Toc449873372)

[8.4.1. P-Value 50](#_Toc449873373)

[8.4.2. T-Test 51](#_Toc449873374)

[8.4.3. Mutual Information 52](#_Toc449873375)

[8.4.4. ReliefF 53](#_Toc449873376)

[8.5. Single GPU VS Multiple GPU 53](#_Toc449873377)

[8.5.1. GPU Accelerated P-Value Processor 54](#_Toc449873378)

[8.5.2. GPU Accelerated T-Test Processor 56](#_Toc449873379)

[8.5.3. GPU Accelerated Mutual Information Processor 58](#_Toc449873380)

[8.5.4. GPU Accelerated ReliefF Processor 60](#_Toc449873381)

[9. Discussion 62](#_Toc449873382)

[9.1. The Zig Zag Curve Shape 62](#_Toc449873383)

[9.2. Performance of GPU Acceleration among Algorithms 63](#_Toc449873384)

[9.3. Limitation 63](#_Toc449873385)

[9.4. Future Work 64](#_Toc449873386)

[10. Conclusion 65](#_Toc449873387)

[11. Schedule 66](#_Toc449873388)

[12. Reference 68](#_Toc449873389)

# Introduction

## Motivation

With the help of open source data mining tools, everyone can mine data on small dataset easily with a desktop or notebook. But when it comes to large scale data mining, their performance will be a big issue. In worst case, it can take several days for a task to complete, or even cannot complete a task. Analyzing biomedical data such as DNA is one of the examples.

To speed up such computation by separating the mining tasks to many CPUs or many machines always is the first idea come up in one's mind. However, such solution is not cost effective nowadays.

GPU parallel computing is the best solution for now. GPU acceleration is becoming popular in these few years. Lots of big IT firms like Google and Amazon also make use of GPU for data analysis. When comparing the parallelizing capability between CPU and GPU, GPU always is the winner.

In fact, the multi-threading computational power of GPUs comes from their visual processing ability. As GPU is designed for processing millions of pixels at a time, it has superb multi-tasking power. Since GPU is capable of processing many small pieces of a data mining algorithm in parallel at a time, a single piece of GPU can outrun a CPU thousands of times. This motivated us to parallelize data mining algorithms on GPU.

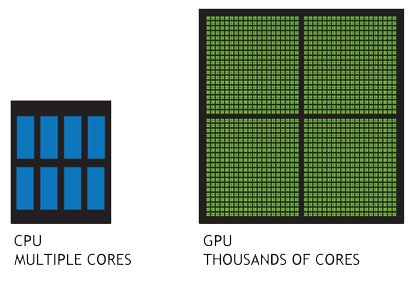


Figure : CPU cores VS GPU cores

## Objectives

The first objective of this project is to accelerate data mining algorithms on GPU by parallel computing for better mining performance. Detail comparison should be made between different configurations for their improvement measurements.

In data mining, no single algorithm is suitable for all models. It is usual for a researcher to analyze a data set again and again with different algorithms. Thus, the second objective is to accelerate the following feature selection algorithms: P-Value, T-Test, Mutual Information and ReliefF.

As the performance of an algorithm can be affected by the size of data set. The third objective is to provide the flexibility for users to configure our GPU accelerated data mining algorithms. Such as having more threads and GPUs for a large dataset for better parallelization, or having less GPU for a small for less overhead.

The forth objective is to develop an library for applications to make use of. An feature selection engine should be developed as the first application to integrate with the library.

As the number of GPUs on a machine is limited, for large scale data mining, it needs to distribute tasks to multiple machines. Hence, the last objective is to enable a framework to run our GPU accelerated algorithms on a distributed system easily.

# Background

## Large Scale Data

In Bioinformatics, one of the most common large scale biological data is generic information. Single nucleotide polymorphism (SNP), which is the genetic variation in nucleotide, is always used for analysis. As human being has 3 billion DNA pairs, although only 1 % of them are SNPs, there are still around 10 million SNPs for analysis. Which is very huge in size.

## Data Mining - Feature Selection

Not all SNPs are useful for a model construction, it needs to select relevant features before head. For example, researchers may use feature selection to target chromosomes which are mostly related to lung cancer.

One challenge in such data mining process on large scale data set is to work with lots of features or samples. For human genetic analysis, there are 10 millions of SNPs to work for, selecting useful features certainly involves lots of calculations.

## Super Computer

In this project, the super computers we used were accelerated by graphics processing unit (GPU). NVIDIA Tesla K20M is the GPU model we geared the machines up with. Which has 2,496 cores with clock rate 705.5 MHz, and the one GPU has 5 gigabytes memory.

As a GPU has thousands more cores than a CPU, GPU can accelerate calculation significantly if we can break a task into many small pieces and parallelize them efficiently among GPU cores.

Figure : NVIDIA GPU Architecture (K20M)

## Performance of Common Data Mining Tools

Nowadays, it is convenient to get open sourced tools online, such as WEKA for data mining. Their stability and their correctness are trustable as lots of users have using them extensively for many years. However, performance is not the area they are good at.

In fact, for large scale data mining, it requires hundreds to thousands times performance improvement. Those CPU based data mining tools usually cannot complete a task in a reasonable processing time.

## Popularity of GPU Acceleration

So people always want to parallel the computation with GPU's super multi-processing power. Actually, big IT firms like Google and Amazon are also making use of such technology for their data analysis as well as machine learning projects.

GPU acceleration is popular. In 2015, Google has open sourced its machine learning library TensorFlow. Which "allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API." (TensorFlow Home Page, 2015) Retrieved from https://www.tensorflow.org/

Apart from Google, Amazon's AWS also support GPUs. "If you require high parallel processing capability, you'll benefit from using GPU instances, which provide access to NVIDIA GPUs with up to 1,536 CUDA cores and 4 GB of video memory." (Linux GPU Instances - Amazon Elastic Compute Cloud). Retrieved from http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using\_cluster\_computing.html

## CUDA framework and Programming

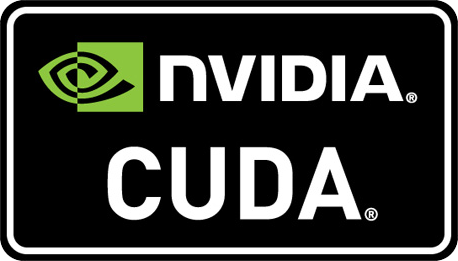


Figure : NVIDIA CUDA

CUDA is one of the most popular platform for GPU acceleration. Having a NVIDIA display card is all you need to start with. Hence, GPU acceleration can also be carried out on personal computers and notebooks.

CUDA, which stands for Compute Unified Device Architecture, is a parallel computing platform and application programming interface (API) model created by NVIDIA. "It allows software developers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing – an approach known as GPGPU. The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements." (Abi-Chahla, 2008). Retrieved from https://en.wikipedia.org/wiki/CUDA

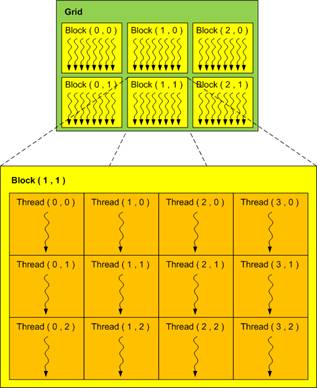


Figure : CUDA Architecture

In fact, there is a relatively long learning curve for developers to get used to GPU parallelism implementation. To accelerate an algorithm by GPU, developers must first see the problem in a multi-dimensional perspective. It is because GPUs are originally designed for calculating every pixel's effect on a screen in three dimensions (width, height and depth).

In NVIDIA display cards, the three dimensional computations are handled by grids, blocks and threads, and CUDA architecture groups threads into blocks, and blocks into grids. And CUDA assigns each thread a unique index in their block, and assigns each block a unique index in their grid. This architecture and its specific naming bring complexity to both design and implementation.

By now, CUDA can bind with the following language:

* Common Lisp
* C
* C++
* C#Fortran
* F#
* Haskell
* IDL
* Java
* Lua
* Mathematica
* MATLAB
* .NET
* CUFFT
* Perl
* Python
* Ruby
* R

(CUDA - Wikipedia, the free encyclopedia). Retrieved from https://en.wikipedia.org/wiki/CUDA#Language\_bindings

(About CUDA). Retrieved from https://developer.nvidia.com/about-cuda

## NVIDIA Visual Profiler

CUDA platform comes with a handy profiling tool, NVIDIA Visual Profiler. This profiler is a cross-platform performance tool which can deliver developers vital feedback for optimizing CUDA C/C++ applications. Apart from looking into resource usage, it can also provide developers useful suggestion for performance gain.

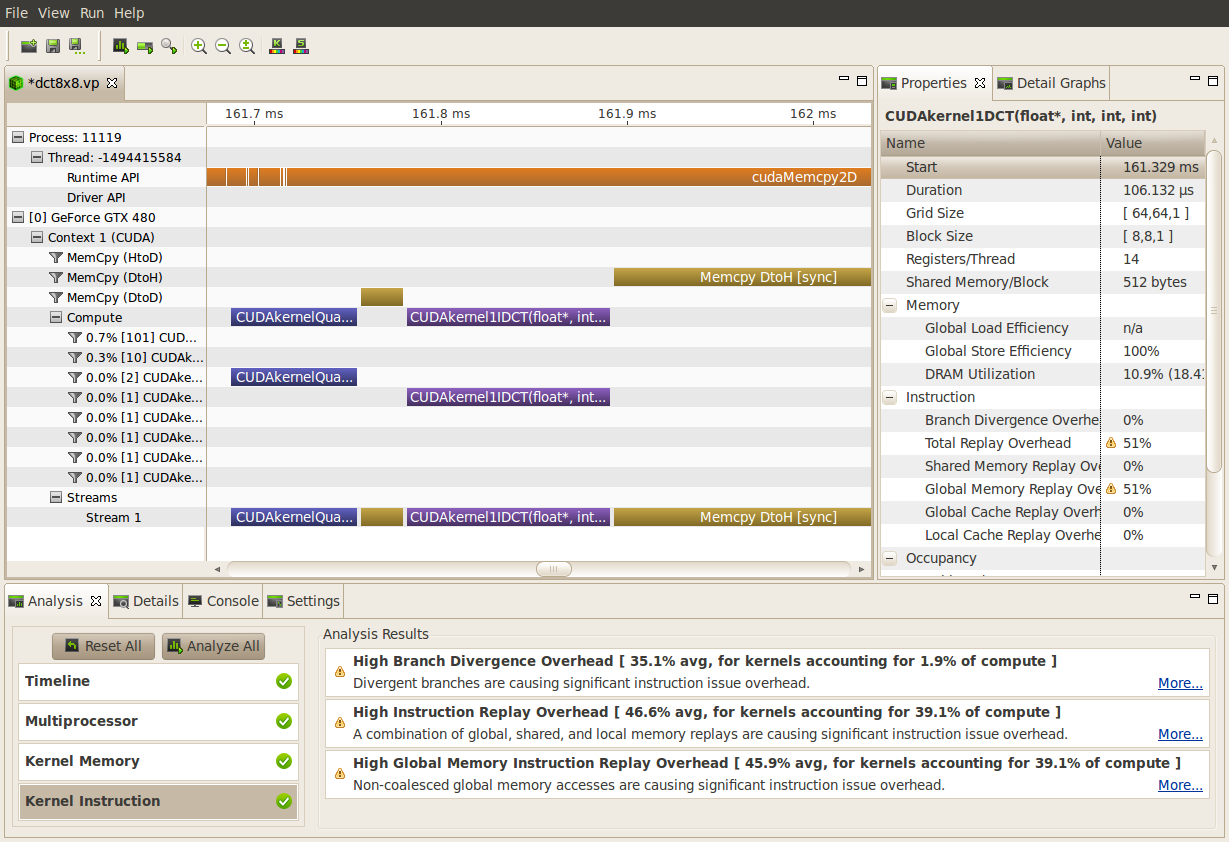


Figure : NVIDIA Visual Profiler Screenshot

# User Specifications

This project targets for two areas. One is to accelerate data mining algorithms with GPU and the other is to demonstrate the performance improvement by making use of such accelerated algorithms.

In this project, all GPU accelerated data mining algorithms should be packaged in "GPU Accelerated Data Mining Algorithm Library". Also a "GPU Accelerated Feature Selection Engine" has to be developed by making use of the GPU accelerated library.

## GPU Accelerated Data Mining Algorithm Library

This library should support the four data mining algorithms, they are P-Value, T-Test, Mutual Information and ReliefF.

Not only for bioinformatics, this library is designed for general purpose. Which should be suitable for different kinds of data mining applications.

New algorithms should be possible to add to the library easily.

This library should support both CPU and Nvidia GPU for performance comparison and resource utilization.

## GPU Accelerated Feature Selection Engine

### Framework

This engine should work as a framework by making use of the classes of "GPU Accelerated Data Mining Algorithm Library" for score calculations.

This engine is responsible for separating samples into groups by labels, and passing such pre-processed data to the libraries processors for scores calculation. And perform feature selection by picking the highest or lowest score features.

As this is the first application to integrate with "GPU Accelerated Data Mining Algorithm Library". It should be able to make use of all the functions provided by the library for finding out the library's performance bottle necks and limitations.

It should support multiple data input file formats, ARFF is the first one to start with. Note that " an ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files were developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the Weka machine learning software. " (Waikato, 2008). Retrieved from http://www.cs.waikato.ac.nz/ml/weka/arff.html

### Data set

This project used part of GWAS data set with the following parameters:

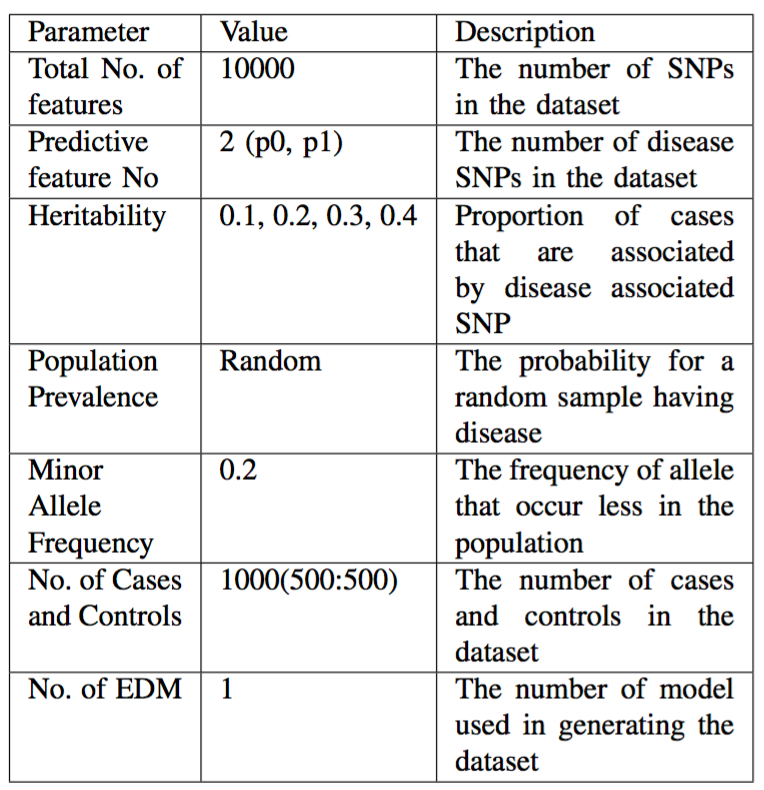


Figure : Parameters for generating the data set

Paper reference of the data set generator:   
  
Source of real GWAS data set: North American Rheumatoid Arthritis Consortium

# Design Specifications

Apart from user specifications, non-functional requirements such as design principle and a "GPU CPU Performance Comparison Tool" is introduced in this section.

## Design Principle

As this project should be implementing in iterations of experiments together with lots of bug fixes and redesigns. Modifying the same class or the same function repeatedly is expected. Hence we should keep all designs and codes simple and stupid.

Object Oriented design pattern is adopted for keeping the project architectural, scalable , understandable, software reusable, and for complexity encapsulation.

For the best code readability, every API can only serve for only 1 purpose; functions should be divided into small scopes; objects should be named in nouns; functions should be named in present tense singular verbs.

## GPU Accelerated Data Mining Algorithm Library

In order to purchase the best performance, C++ has been chosen as this projects programming language.

CUDA has been selected for GPU acceleration, because The Chinese University of Hong Kong has equipped several super computers with NVIDIA display cards. Which provides a mature development environment for this project.

This library accelerates calculations mainly by parallel computing. Not only GPU parallelization, it should also support CPU parallelization.

As a data set can be processed by multiple algorithms for data mining, the APIs of algorithms should be standardized. With this standardization, any application only needs to have one library integration to make use of all the provided algorithms. Thus, the integration effort for an applications can be minimized.

## GPU Accelerated Feature Selection Engine

As "GPU Accelerated Feature Selection Engine" should be able to make use of all the functions provided by this project's GPU library for figuring out the library's performance bottle necks and limitations. Therefore, functionality is more important than usability. Together with the limited development timeframe, command based input is adopted instead of having a control panel.

As the library only supports C++ for now, this engine should be developed in C++ also.

Processing time should be logged, such that performance can be monitored conveniently.

Two output formats should be supported for reporting. They are MATLAB script output for graph plotting and tab delimited for Excel.

An example of MATLAB script output:

|  |
| --- |
| P1=[ 795 828 737 811 766 824 748 810...]  P2=[ 801 819 756 826 703 806 815 793...] |

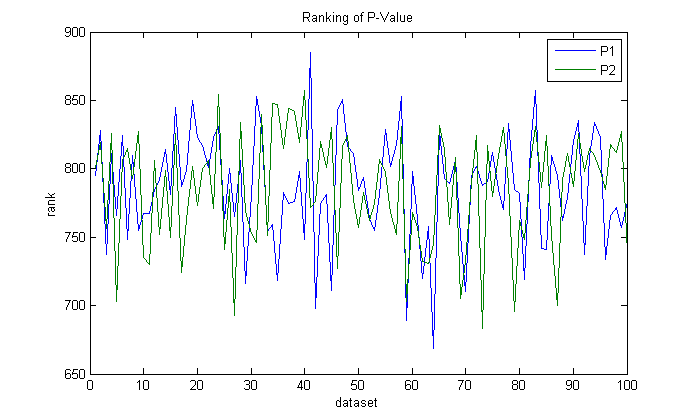


Figure : An example of plotting the score ranking by MATLAB

Note that P-Value is not the suitable model for this data set.

An example for Tab Delimited output

|  |
| --- |
| @/uac/msc/pschu/project/Algorithm/data/snp1000/h01/01\_EDM-1\_01txt.arff  N236 0.000127802 1  N660 0.00108914 2  N912 0.0018259 3  N522 0.00229444 4  N884 0.0032375 5  N622 0.00337159 6  N137 0.00384936 7  N665 0.00584463 8 |

The following parameters should be configurable:

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| Algorithm | Data mining algorithm like P-Value, Mutual Information, T-Test, ReliefF. |
| Processor type | CPU or GPU. |
| Input type | A file or files of a folder. |
| Output type | File or screen. |
| Output format | MATLAB script or tab delimited output. |
| Debug mode | Turn on debug mode or not. |

## GPU Acceleration Performance Testing Tool

Apart from using this project's GPU library, figuring out how much can an algorithm be accelerated by GPU is the utmost important. Performance changes should be measured by running lots of tests systematically. For conducting such measurements, it needs a "GPU Acceleration Performance Testing Tool" to record the processing time of all the GPU accelerated algorithms in different settings as well as the performance of their CPU counterparts for comparing.

Hence, it should be able to start performance tests with different configurations without code change.

For reporting, it should be able to export performance records in MATLAB script.

And the tool should be robust enough for running continuously in several days at least.

Again, for keeping this tool simple for implementation, it should be control by command input.

For scheduling a performance test, the following parameters should be configurable:

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| **Algorithm** | Data mining algorithm like P-Value, Mutual Information, T-Test, ReliefF. |
| **Processor Type** | CPU or GPU. |
| **Threads** | Number of threads for a CUDA block. |
| **Range of threads** | The range of threads for testing. |
| **Device** | Number of GPUs. |

# Detail Design

## GPU Accelerated Data Mining Algorithm Library

In this library, all algorithms are encapsulated in processor classes. All algorithm processor classes are either a child class of "SimpleProcessor" or "GPUAcceleratedProcessor". Subclasses of "SimpleProcessor" only make use of CPU for data mining, and subclasses of "GPUAcceleratedProcessor" accelerate their calculation by GPU parallelization. Configuration functions are implemented in the two parent classes, such that the same function won't be copied across different classes.

"SimpleProcessor" and "GPUAcceleratedProcessor" themselves are child classes of an abstract class "Processor" which is responsible for standardizing the calculation APIs for all the algorithms. Up until now, only one calculation API has been developed, but more API will be introduced to support different data structures.

The only concrete processor classes in the library are algorithm processors. Their class names carry the corresponding algorithm and the type of processing unit. Such that developers able to understand how to make use of the APIs by naming only. Eight processors: "GPUAcceleratedPValueProcessor", "GPUAcceleratedTTestProcessor", "GPUAcceleratedMutualInformationProcessor", "GPUAcceleratedReliefFProcessor", "SimplePValueProcessor", "SimpleTTestProcessor", "SimpleMutualInformationProcessor" and "SimpleReliefFProcessor" were planned to be developed for the time being. New algorithms can be added to this library by following this architecture.

Processing results are exported by a structure data type named in "Result", which carries the processing start time and end time and the scores calculated by algorithm processors.

To make use of the algorithm processors, a developer just needs to instantiate the required class, and then calls the calculation API for the processing result. Below is a simple example in pseudo code.

|  |
| --- |
| Processor processor = new GPUAcceleratedPValueProcessor  Result result = processor.calculate(data) |

Below is a class diagram with detail APIs for the processor classes.

E:\CUHK\project\final_report\Algorithm Processor.png

Figure : Class Diagram of GPU Accelerated Data Mining Algorithm Library

## Simple Algorithm Processors

If implementing the data mining algorithm from scratch, the development time would be unnecessary long. Therefore, this library is started from porting C/C++ source code of the 2 following Open Source Projects:

**MIToobox for C and MATLAB**: note that this toolbox was developed by a PhD student of The University of Manchester. "Adam Pocock", the researcher, said "the toolbox was developed to support our research into feature selection algorithms and includes some sample feature selection algorithms from the literature to illustrate its use." (Pocock) . Retrieved from http://www.cs.man.ac.uk/~pococka4/MIToolbox.html

**Rosetta Code**: "Rosetta Code is a programming chrestomathy site. The idea is to present solutions to the same task in as many different languages as possible, to demonstrate how languages are similar and different, and to aid a person with a grounding in one approach to a problem in learning another." (Rosetta Code Home Page). Retrieved from http://rosettacode.org/wiki/Rosetta\_Code

## GPU Accelerated Algorithm Processors

GPU acceleration basically is achieved by 2 kinds of parallelism. They are data parallelism and task parallelism.

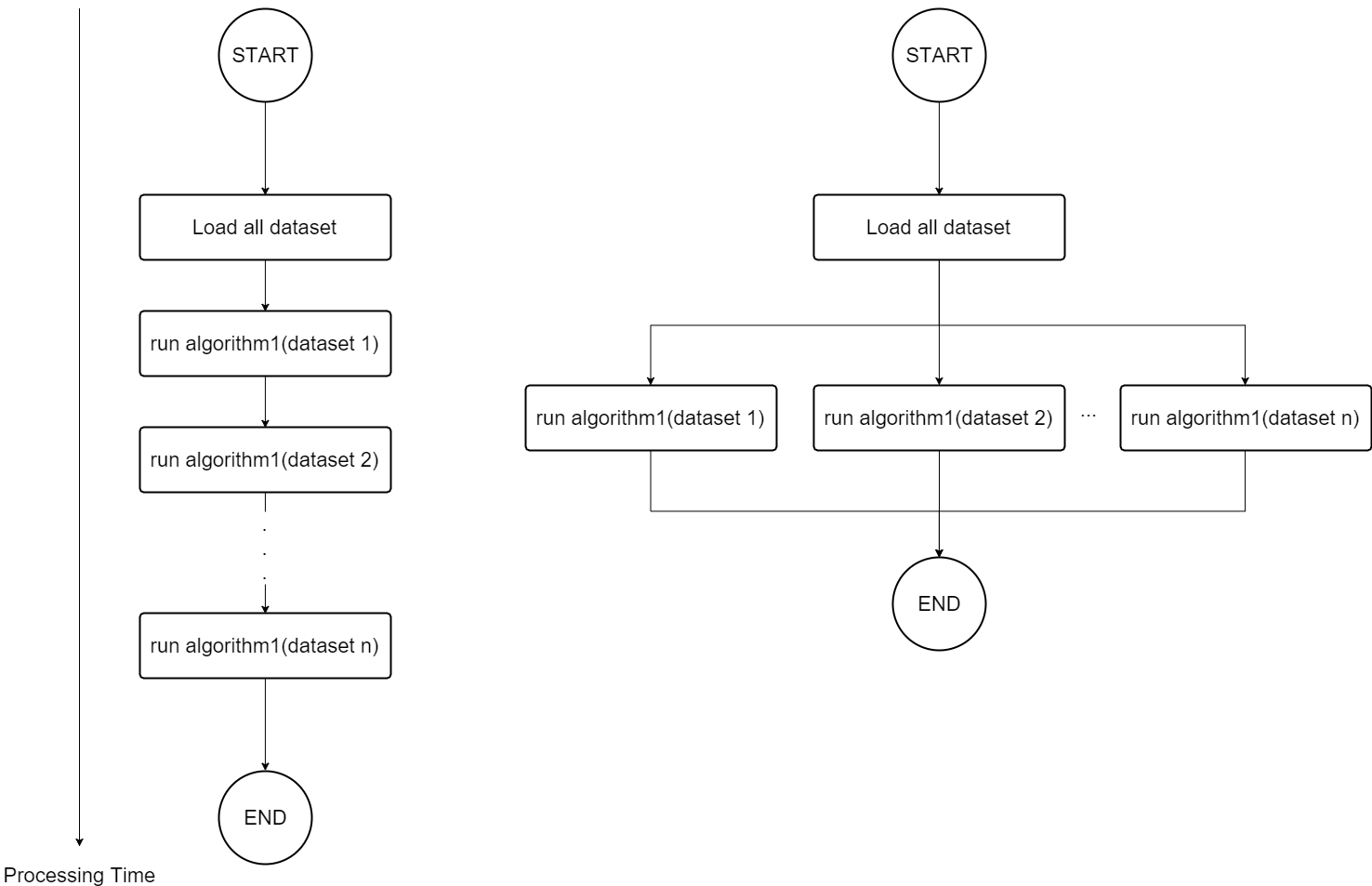
Data parallelism focuses on distributing the data across different parallel computing nodes. In order words same functions runs simultaneously across the same or different data.

Task parallelism focuses on distributing tasks—concretely performed by processes or threads—across different parallel computing nodes. In order words, different tasks runs simultaneously across the same or different data.

Apart from them, it is also possible to collaborate multiple GPUs for higher parallelizing capability.

For the calculation, as the API is designed for comparing the data between arrays, data parallelism can be implemented in the abstract class "GpuAcceleratedProcessor" by separating the calculation tasks by data.

The following flow chart shows how data parallelism make the performance difference. Performance can be increased be replacing the sequential run algorithm by running algorithms on GPU in parallel.



GPU accelerates algorithm 1 for multiple datasets by running it in parallel.

Figure : Comparison between the workflow of CPU and GPU accelerated Processor

For the "GpuAcceleratedProcessor", data are stored separately into CUDA blocks. In CUDA architecture as shown below, grid is in three dimensional, and a grid contains lots of blocks, each of them is responsible for calculating a pixel. In "GpuAcceleratedProcessor", calculations of every data set are carried out independently inside blocks.

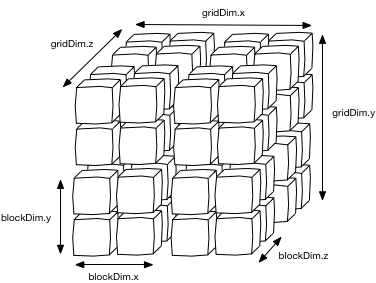


Figure : CUDA Grid and Blocks

### GPU Accelerated P-Value Processor

In P-Value algorithm, mean, variance, t-test and the corresponding probability distribution are calculated in sequence. They can be accelerated by data parallelism in GPU's blocks and threads.

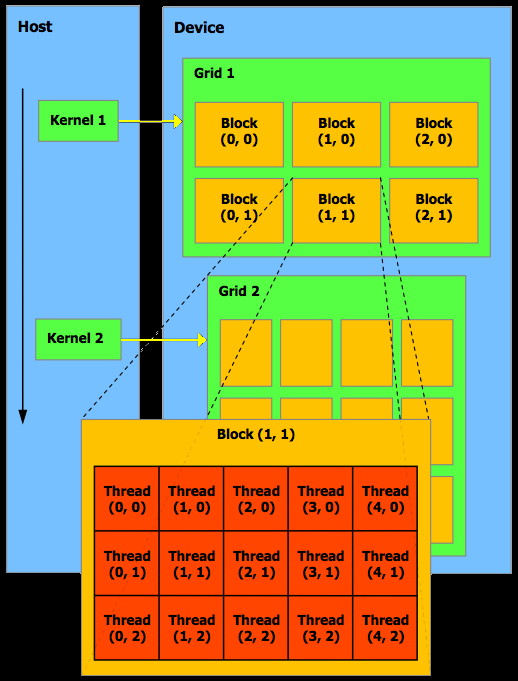
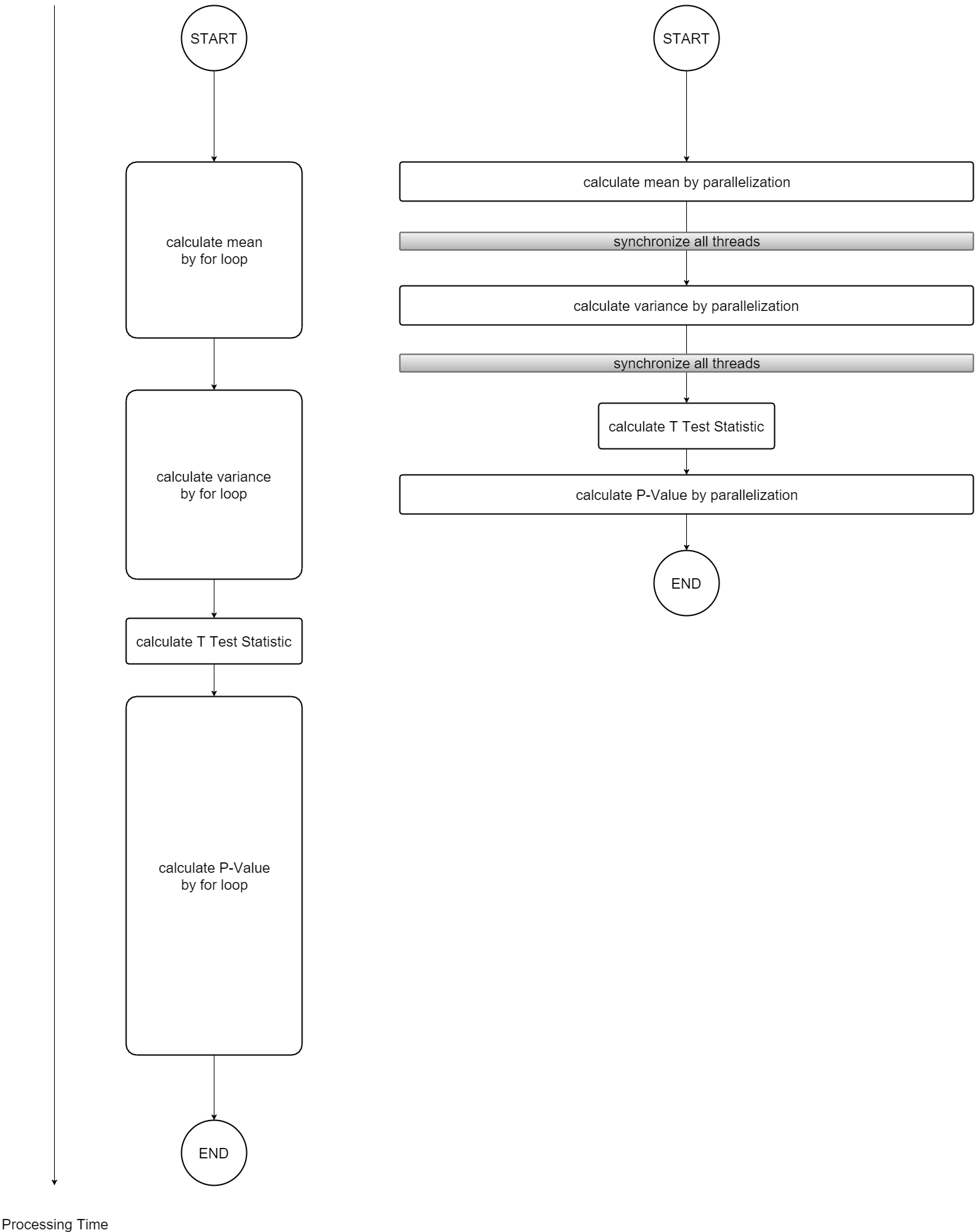


Figure : Threads are dedicated for blocks in CUDA architecture

The following flow chart shows how parallelization improves the performance.



parallelize

parallelize

parallelize

Figure : Simple VS GPU accelerated P-Value calculation

As showed in the above diagram, mean, variance and score calculation are the three parts of the algorithm running in for loops repeatedly. CUDA can further parallelizes the date by using the threads of a block.

As the execution order of calculating mean, variance and score cannot be changed. Synchronization has to be used for keeping their execution orders. In order to prevent race condition, atomic add should be used for the add operations over the shared memory of threads.

Both synchronization and atomic add need to lock threads for waiting, which can lower the GPU parallelization ability if data are separated to all the 1,024 threads. As it turn out, this P-Value GPU Processor performs best with block threads number somehow between 1 to 1,024.

Experiment result for choosing the best block threads number and discussion will be made in the "Results" and "Discussion" section.

### GPU Accelerated T-Test Processor

As T-Test value should be calculated before calculating P-Value's probability distribution, we can make use of the completed T-Test statistic code in the P-Value processor for T-Test processor.

The following flow chart shows how parallelization improves the performance.

E:\CUHK\project\final_report\TTest Parallel Data.png

parallelize

parallelize

Figure : Simple VS GPU accelerated T-Test calculation

### GPU Accelerated ReliefF Processor

ReliefF calculation can be divided into 3 stages. They are "calculate the distance between samples", "find the nearest neighbors" and "weight all features".

Similar to GPU Accelerated P-Value Processor, we parallelize ReliefF scores calculation of by blocks and threads. Hence, threads numbers can affect its performance because of the synchronization and atomic operations.

The following flow chart shows how parallelization improves the performance.

E:\CUHK\project\final_report\ReliefF Parallel Data.png

parallelize

parallelize

parallelize

Figure : Simple VS GPU accelerated ReliefF calculation

## GPU Accelerated Feature Selection Engine

The workflow of this engine is simple. Once the engine is called by a command, it will first read algorithm, input and out file path from the command. And then instantiates the required algorithm processor with suitable configuration. After then, it will read the input file and load data. Once all data of the file has been loaded, it passes the data to the processor for score calculation. If there are more files for data loading, the engine will read the next file for another round of score calculation. After all files have been read, all scores have been calculated, it will rank the features according to the score for feature selection and export the result accordingly.

Below is a flow chart of the feature selection.

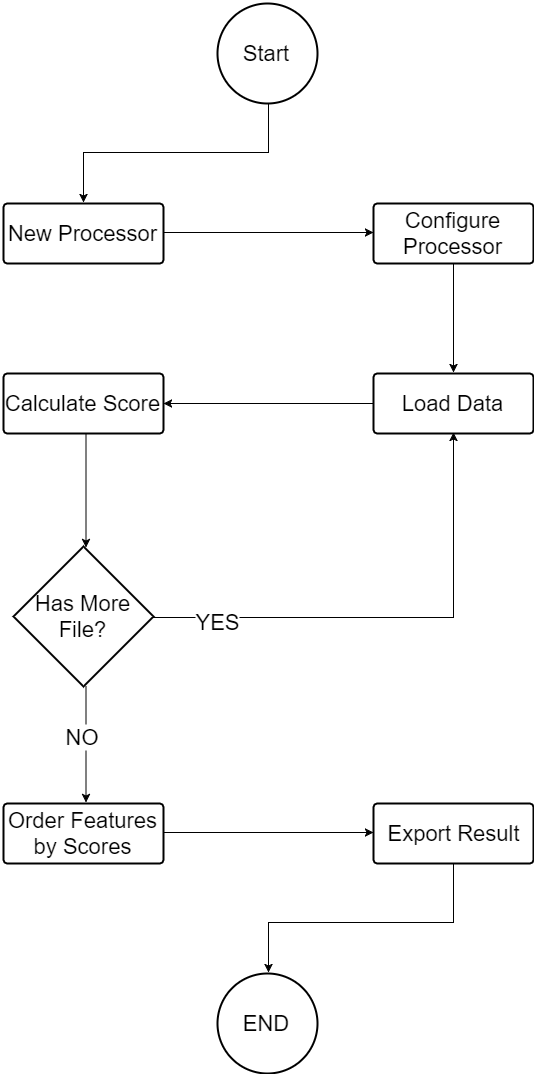


Figure : Feature Selection workflow

## Performance Testing Tool

In order to make sure each test won't affect others, "GPU Acceleration Performance Test Tool" triggers "GPU Accelerated Feature Selection Engine" only once for every action.

For example, if it needs to find out the performance of P-Value GPU Processor in different number of threads of a block from 1 to 99 by testing the 100 configurations for 100 times. This tool will call the engine 100 x 100 = 10,000 times.

Apart from testing independently, for multiple configurations test cases, configuration will be changed for every test. Together with large number of tests, noises can be averaged out to different configurations.

Below is an example for interleaving configurations.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Time** | **Configuration 1**  **64 threads** | **Configuration 2**  **128 threads** | **Configuration 3**  **196 threads** | **Configuration 4**  **256 threads** |
| **0** | RUN |  |  |  |
| **1** |  | RUN |  |  |
| **2** |  |  | RUN |  |
| **3** |  |  |  | RUN |
| **4** | RUN |  |  |  |
| **5** |  | RUN |  |  |
| **6** |  |  | RUN |  |
| **7** |  |  |  | RUN |
| **8** | RUN |  |  |  |
| **9** |  | RUN |  |  |
| **10** |  |  | RUN |  |
| **11** |  |  |  | RUN |

# Implementation

## Development Environment

This project implemented on the super computers provided by The Chinese University of Hong Kong with the following specification:

**CPU**: Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz

**GPU**: NVIDIA K20 5GB RAM

|  |  |
| --- | --- |
| **Hostname** | **GPUs** |
| hpc1.cse.cuhk.edu.hk | 3 x K20m GPU |
| hpc2.cse.cuhk.edu.hk | 1 x K20m GPU |
| hpc3.cse.cuhk.edu.hk | 6 x K20m GPU |
| hpc4.cse.cuhk.edu.hk | 6 x K20m GPU |
| hpc5.cse.cuhk.edu.hk | 8 x K20m GPU |
| hpc6.cse.cuhk.edu.hk | 8 x K20m GPU |

**C++ compiler**: gcc version 4.8.1 (GCC)

**CUDA version**: 6.5

**Operation System**: Linux version 2.6.32-431.5.1.el6.x86\_64

## Profiling

It is worth mentioning that NVIDIA Visual Profiler saved us lots of time for spotting out performance bottle neck. It always give detail information about where the time spent, how memory bytes of memory has been used. It even can suggest ideas such as collaborating both CPU and GPU for better resource utilization.

Figure : Profiling Screenshot

## Performance Tuning

It usually took us many iterations to accelerate an algorithm processing.

Take ReliefF as an example. The algorithm has three main stages, they are "calculate the distance between samples", "find the nearest neighbors" and "weight all features". Following is the profile before performance tuning.

Weight all features

Find the nearest neighbours

Calculate sample distances

Figure : Before performance tuning

**Iteration 1**

As the sample distances calculation stage took most of processing time in the first implementation, we reduce that by using bitwise operations.

Reduced by bitwise operations

Figure : Calculate distance by bitwise operations

**Iteration 2**

Then, our next target was improving the performance of "find the nearest neighbors", and the solution was to sort the distances among samples by using bucket sort algorithm.

Reduced by bucket sort

Figure : Use bucket sort to find the nearest neighbors

**Iteration 3**

However, bucket sort consumed too much GPU memory, as it needs features size \* number of nearest neighbors \* integer size memory space for a bucket.

So we replaced the sorting algorithm with heap sort, which is slower than bucket sort but just needs a little memory space with sample size \* integer size memory space for a heap.

Replaced bucket sort with heap sort

Figure : use heap sort to find the nearest neighbors

**Iteration 4**

Then we found the workload of distance calculation was parallelized among GPUs so inefficiently that one of the GPU completed around one third of the work among 5 GPUs.

Bad parallelization among GPUs

Figure : Bad multiple GPUs parallelization

Thus, we modified our ReliefF implementation to rebalance the distance calculation among GPUs fairly.

Fairly parallelized among GPUs

Figure : Fair multiple GPUs parallelization

# Test and Validation

The scores of P-Value, T-Test, Mutual Information and ReliefF are validated by the following libraries or tool:

**SciPy**: "SciPy (pronounced “Sigh Pie”) is an open source Python library used by scientists, analysts, and engineers doing scientific computing and technical computing." (SciPy Home Page). http://www.scipy.org/

**Scikit-learn**: "scikit-learn (formerly scikits.learn) is an open source machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy." (scikit-learn - Wikipedia, the free encyclopedia). Retrieved from https://en.wikipedia.org/wiki/Scikit-learn

The calculation correctness of the processors of "GPU Accelerated Data Mining Algorithm Library" was evaluated by cross checking the scores among 10,000 features with 1,000 samples. The range of the scores is [0,1].

**WEKA**: " Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a data set or called from your own Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes." (Weka 3 - Data Mining with Open Source Machine Learning Software in Java)

## P-Value

Difference between the scores calculated from this project's processor and the Python SciPy library listed above:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Score difference between implementations** | | | |
| **Score** | **Mean** | **Median** | **Max** | **Min** |
| P-Value > 0.5 \* | 1.26 x 10-4 | 5.56 x 10-7 | 2.692 x 10-3 | 0 |
| P-Value <= 0.5 | 4.19 x 10-7 | 3.2 x 10-7 | 1.75 x 10-6 | 8.53 x 10-10 |

\* As we consider small P-Value for feature selection, the difference between P-Value won't affect the correctness.

## T-Test

Difference between the scores calculated from this project's processor and the Python SciPy library listed above:

|  |  |  |  |
| --- | --- | --- | --- |
| **Score difference between implementations** | | | |
| **Mean** | **Median** | **Max** | **Min** |
| 1.27 x 10-6 | 7.22 x 10-7 | 1.04 x 10-10 | 0 |

## Mutual Information

Difference between the scores calculated from this project's processor and the Python Scikit learn library listed above:

|  |  |  |  |
| --- | --- | --- | --- |
| **Score difference between implementations** | | | |
| **Mean** | **Median** | **Max** | **Min** |
| 3.09 x 10-9 | 2.22 x 10-9 | 4.93 x 10-8 | 6.16 x 10-16 |

## ReliefF

Difference between the scores calculated from this project's processor and WEKA listed above:

|  |  |  |  |
| --- | --- | --- | --- |
| **Score difference between implementations** | | | |
| **Mean** | **Median** | **Max** | **Min** |
| 2.96 x 10-4 | 2.5 x 10-4 | 1.23x 10-3 | 0 |

# Results and Evaluation

## GPU Acceleration Data Size Limitation

In human SNPs analysis, feature size usually bigger than sample size. The biggest data size is around 1,000,000 features with 10,000 samples. The GPU we used cannot process such huge data size at once, so we found our GPU limitation with the max feature and max sample size respectively.

|  |  |
| --- | --- |
| **Testing environment** | |
| **Server** | hpc2.cse.cuhk.edu.hk |
| **GPU** | 1 x NVIDIA K20 5GB RAM |

### Big Feature Size Data

Following is the maximum number of sample can our GPU accelerated processors in the testing environment when data sets contain 1,000,000 features.

|  |  |
| --- | --- |
| **Algorithm** | **Max sample size** |
| P-Value | 2,100 |
| T-Test | 2,100 |
| Mutual Information | 2,100 |
| ReliefF | 1,100 |

### Big Sample Size Data

Following is the maximum number of features can our GPU accelerated processors in the testing environment when data sets contain 10,000 samples.

|  |  |
| --- | --- |
| **Algorithm** | **Max feature size** |
| P-Value | 210,000 |
| T-Test | 210,000 |
| Mutual Information | 210,000 |
| ReliefF | 95,000 |

## Performance of GPU Accelerated Algorithms in Different Thread Numbers

Because of atomic operation and synchronization, GPU acceleration varies in different thread sizes. Tests in this section were designed for investigating how much can algorithms running be accelerated in different thread numbers and data size permutation (samples x features).

Since all the algorithms of this project are executed in different speeds, a configuration which takes 1,000 ms for an algorithm running may takes only a few ms for another. Thus, not all algorithms were tested with the same setups.

We ran each sample size, feature size, thread size permutation for 10 times. For each algorithm, we plotted graphs to show the relationship between number of threads and the corresponding processing time. In each graph, Y-Axis is the average processing time, X-axis is the number of threads.

According to the results, we found that the increasing thread numbers can improve performance, but the improvements saturate very soon and then reduce afterward.

|  |  |
| --- | --- |
| **Testing environment** | |
| **Server** | hpc2.cse.cuhk.edu.hk |
| **GPU** | 1 x NVIDIA K20 5GB RAM |

### GPU Accelerated P-Value Processor

We found that GPU Accelerated P-Value Processor performs well with 128 threads for bigger feature sizes. As there is no big difference for using 96 threads or 128 threads for small size data sets, we suggest to set 128 threads for this processor's default configuration.

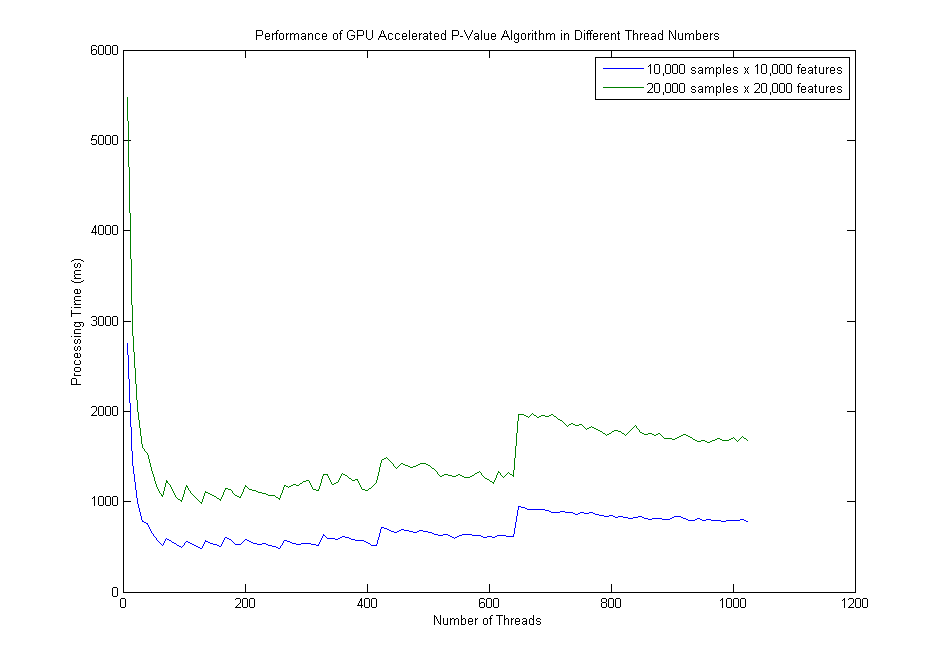


Figure : P-Value Processing Time VS Number of Threads (8 to 1024)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data Size** | | **Processing Time** | | | |
| **Sample Size** | **Feature Size** | **Min** | **Max** | **Mean** | **Median** |
| 10,000 | 10,000 | 481 ms  (128 threads) | 1,9264 ms  (1 thread) | 710 ms | 650 ms |
| 20,000 | 20,000 | 984 ms  (128 threads) | 18,776 ms  (1 thread) | 1,489 ms | 1,358 ms |
| 1,000 | 10,000 | 592 ms  (128 threads) | 15,956 ms  (1 thread) | 941 ms | 792 ms |
| 10,000 | 1,000 | 55 ms  (96 threads) | 2,094 ms  (1 thread) | 76 ms | 68 ms |

### GPU Accelerated T-Test Processor

We found that GPU Accelerated P-Value Processor performs well with 232 threads for bigger size data sets. As there is no big difference for varying the configuration between 106 threads and 232 threads for small size data sets, we suggest to set 232 threads for this processor's default configuration.

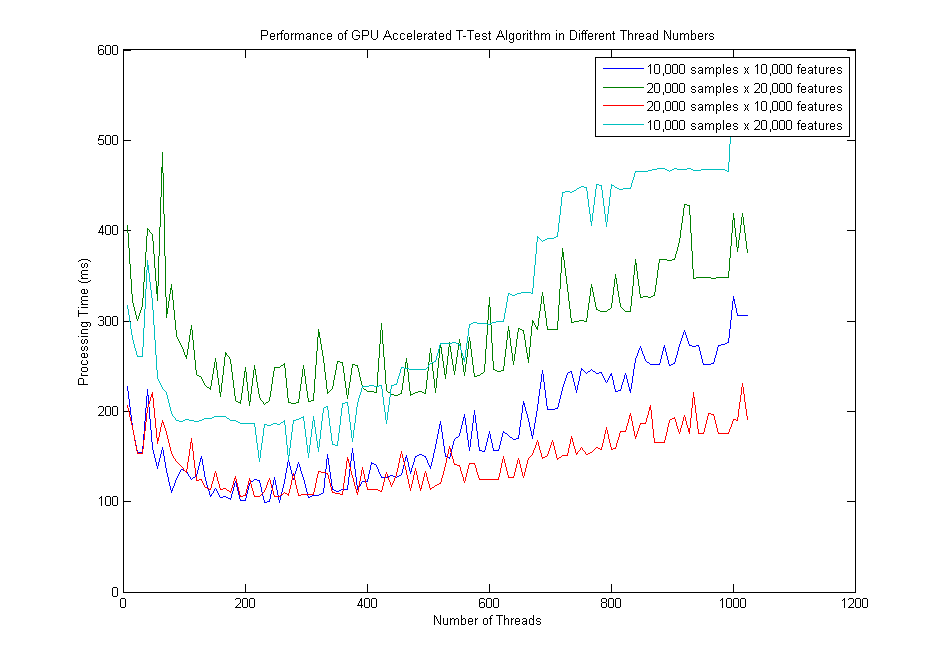


Figure : T-Test Processing Time VS Number of Threads (8 to 1024)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data Size** | | **Processing Time** | | | |
| **Sample Size** | **Feature Size** | **Min** | **Max** | **Mean** | **Median** |
| 10,000 | 10,000 | 232 ms  (232 threads) | 511 ms  (1 thread) | 179 ms | 158 ms |
| 10,000 | 20,000 | 144 ms  (224 threads) | 1,015 ms  (1 thread) | 310 ms | 275 ms |
| 20,000 | 10,000 | 106 ms  (192 threads) | 923 ms  (1 thread) | 146 ms | 142 ms |
| 20,000 | 20,000 | 207 ms  (208 threads) | 1,822 ms  (1 thread) | 286 ms | 281 ms |

### GPU Accelerated Mutual Information Processor

We found that GPU Accelerated Mutual Information Processor performs well with 272 threads for bigger size data sets. As there is no big difference for varying the configuration between 232 threads and 272 threads for small size data sets, we suggest to set 272 threads for this processor's default configuration.

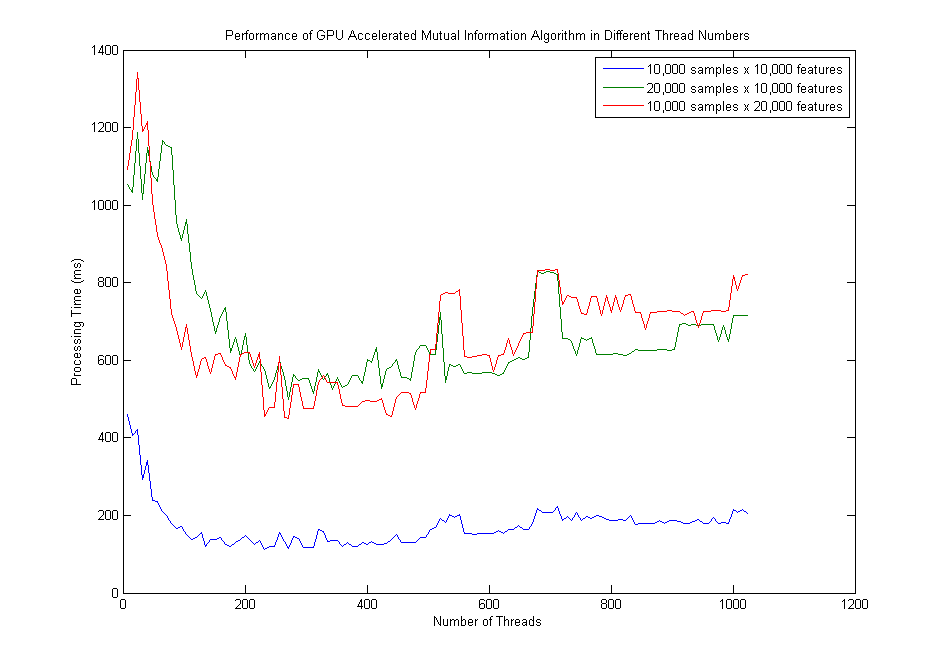


Figure : Mutual Information Processing Time VS Number of Threads (8 to 1024)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data Size** | | **Processing Time** | | | |
| **Sample Size** | **Feature Size** | **Min** | **Max** | **Mean** | **Median** |
| 10,000 | 10,000 | 112 ms  (232 threads) | 1,043 ms  (1 thread) | 173 ms | 166 ms |
| 10,000 | 20,000 | 451 ms  (272 threads) | 2,215 ms  (1 thread) | 671 ms | 662 ms |
| 20,000 | 10,000 | 499 ms  (272 threads) | 2,054 ms  (1 thread) | 672 ms | 620 ms |
| 20,000 | 20,000 | 505 ms  (232 threads) | 4,442 ms  (1 thread) | 803 ms | 675 ms |

### GPU Accelerated ReliefF Processor

We found that GPU Accelerated ReliefF Processor performs well starting from 128 threads. As there is no big difference starting from 128 threads for big size data sets, we suggest to set 256 threads for this processor's default configuration.

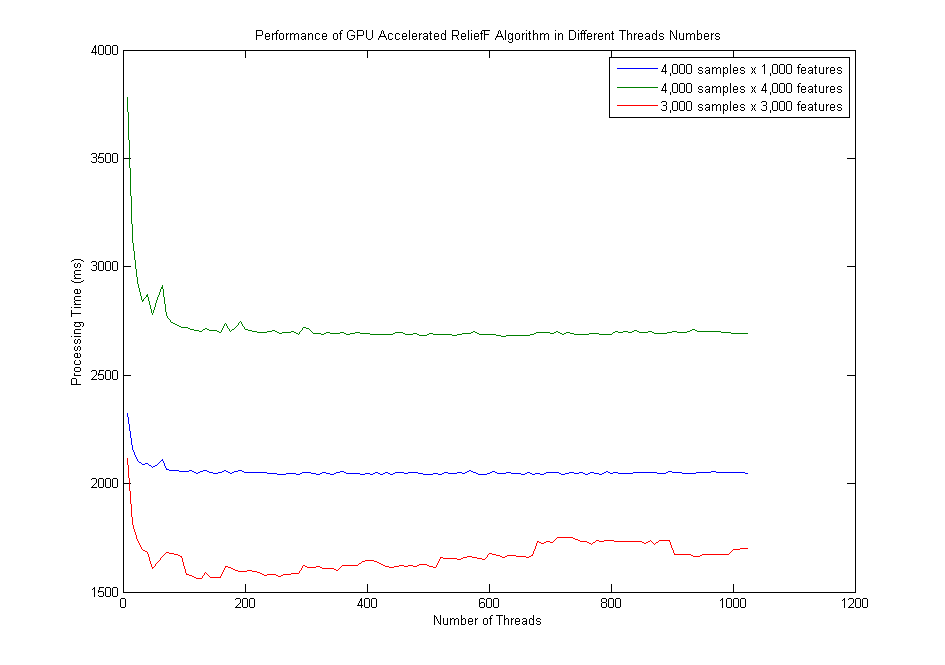


Figure : ReliefF Processing Time VS Number of Threads (8 to 1024)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data Size** | | **Processing Time** | | | |
| **Sample Size** | **Feature Size** | **Min** | **Max** | **Mean** | **Median** |
| 4,000 | 1,000 | 2,042 ms  (440 threads) | 5152 ms  (1 thread) | 2,054 ms | 2,049 ms |
| 4,000 | 4,000 | 2,679 ms  (624 threads) | 14,590 ms  (1 thread) | 2,717 ms | 2,696 ms |
| 3,000 | 3,000 | 1559 ms  (128 threads) | 6,675 ms  (1 thread) | 1,661 ms | 1,658 ms |
| 1,000 | 4,000 | 219 ms  (520 threads) | 1,016 ms  (1 thread) | 223 ms | 221 ms |

## Performance of GPU Accelerated Algorithms in Different Sample Size and Feature Size

In order to visualize the performance of under different sample size and feature size, we plotted the experiment results by 3-D shaded surface plots.

We ran sample size, feature size, thread size permutation test for 10 times each. For each algorithm, we had the 3-D plots with Z-Axis as the average processing time, X-axis as the number of samples and Y-axis as the number of features.

|  |  |
| --- | --- |
| **Testing environment** | |
| **Server** | hpc2.cse.cuhk.edu.hk |
| **GPU** | 1 x NVIDIA K20 5GB RAM |

### GPU Accelerated P-Value Processor

According to the graph below, the relationship between processing time and sample size is linear, and the processing time is insensitive to the changes of the number of features.

The result is reasonable as the most time consuming stage (calculate probability distribution) of P-Value calculation is only sensitive to the number of features but samples.

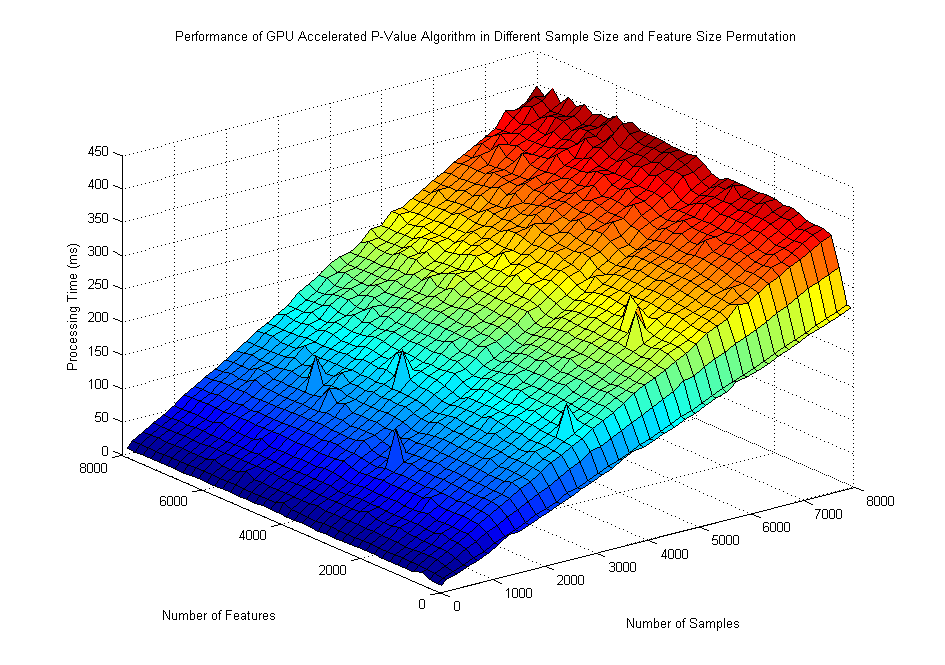


Figure : P-Value Processing Time in Different Sample Size and Feature Size Permutations

### GPU Accelerated T-Test Processor

According to the graph below, the relationship between processing time and sample size is linear, and the relationship between processing time and feature size is also linear.

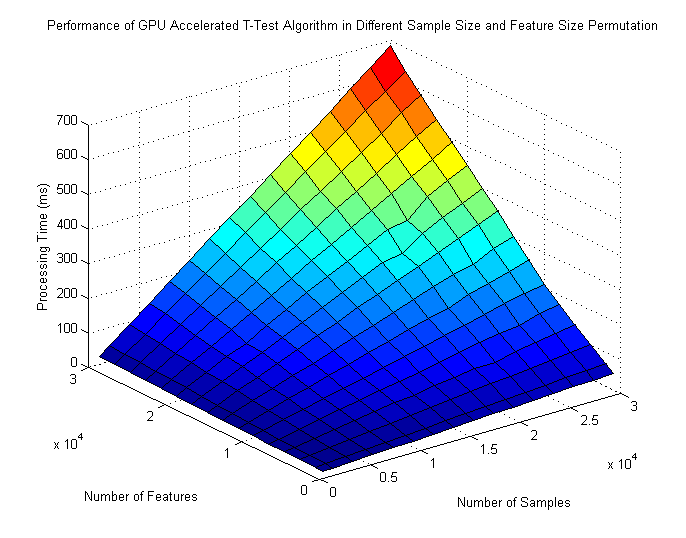


Figure : T-Test Processing Time in Different Sample Size and Feature Size Permutations

### GPU Accelerated Mutual Information Processor

According to the graph below, the relationship between processing time and sample size is linear, and the relationship between processing time and feature size is also linear.

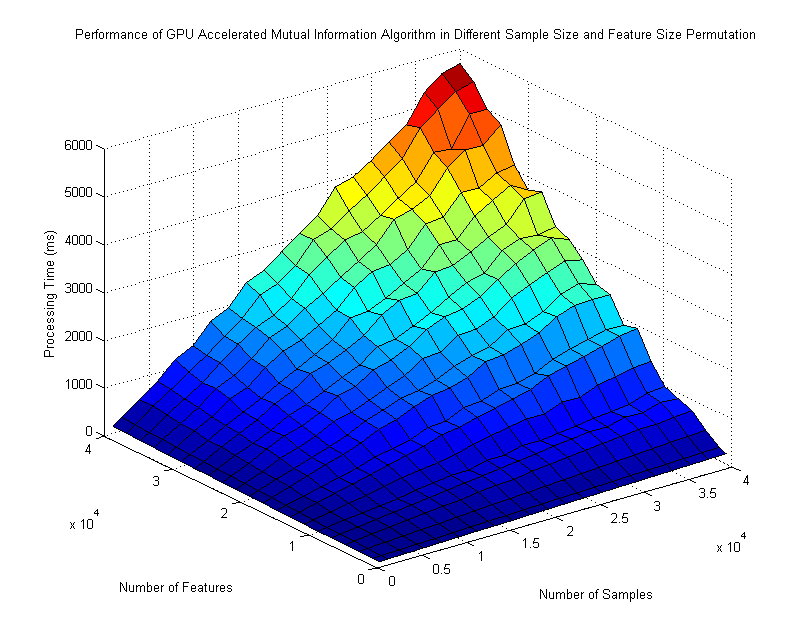


Figure : Mutual Information Processing Time in Different Sample Size and Feature Size Permutations

### GPU Accelerated ReliefF Processor

According to the graph, the relationship between processing time and sample size is non-linear, and the relationship between processing time and feature size is also non-linear.

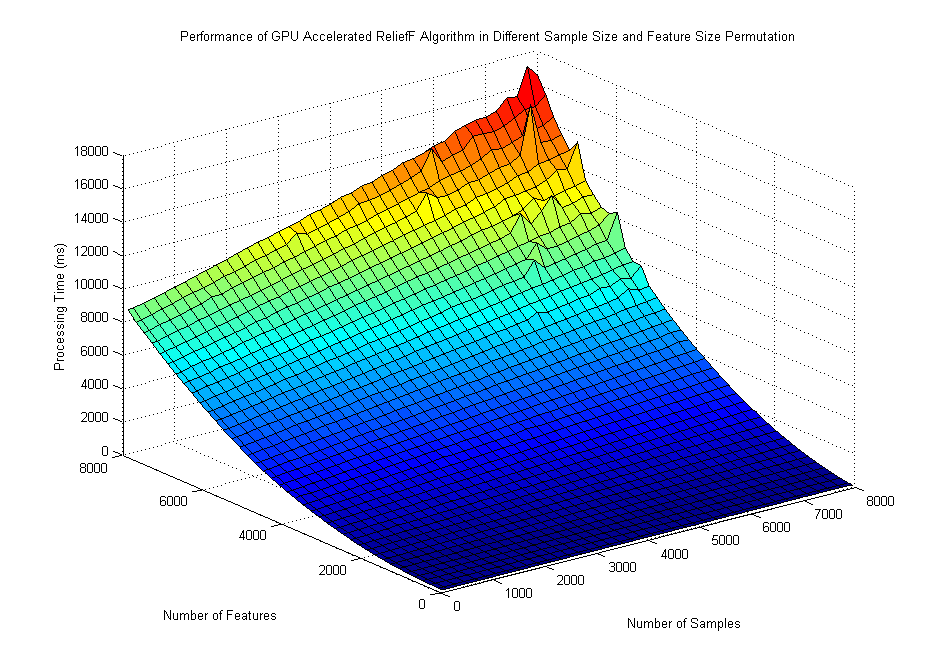


Figure : ReliefF Processing Time in Different Sample Size and Feature Size Permutations

## Performance Comparisons between 1 CPU and 1 GPU

As it can take days for a CPU to run a very large data set repeatedly, so we didn't measure the performance of running GPU accelerated algorithms in their max capability.

This test was conducted by running the different data size combinations 100 times each for CPU and GPU separately.

|  |  |
| --- | --- |
| **Testing environment** | |
| **Server** | hpc2.cse.cuhk.edu.hk |
| **GPU** | 1 x NVIDIA K20 5GB RAM |
| **CPU** | Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz |

### P-Value

The result shows that GPU can accelerate this data mining algorithm significantly for small data sizes (around 1000), which is around 200 times faster. But the acceleration drops with bigger sample sizes down to around 175 times.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Size** | | **Average Processing Time** | | |
| **Sample Size** | **Feature Size** | **CPU Processor** | **GPU Accelerated Processor** | **GPU Accelerated** |
| 1,000 | 1,000 | 12,343 ms | 62 ms | 199.1 times |
| 1,000 | 5,000 | 61, 586 ms | 296 ms | 208.1 times |
| 1,000 | 10,000 | 123,066 ms | 595 ms | 206.8 times |
| 1,000 | 15,000 | 184,663 ms | 892 ms | 207.0 times |
| 5,000 | 1,000 | 10,681 ms | 63 ms | 169.5 times |
| 5,000 | 5,000 | 52,793 ms | 314 ms | 168.1 times |
| 5,000 | 10,000 | 105,640 ms | 623 ms | 169.6 times |
| 5,000 | 15,000 | 158,306 ms | 923 ms | 171.5 times |
| 10,000 | 1,000 | 11,365 ms | 64 ms | 177.6 times |
| 10,000 | 5,000 | 56,145 ms | 320 ms | 175.5 times |
| 10,000 | 10,000 | 112,548 ms | 596 ms | 188.8 times |
| 10,000 | 15,000 | 168,896 ms | 984 ms | 171.6 times |
| 15,000 | 1,000 | 11,381 ms | 69 ms | 164.9 times |
| 15,000 | 5,000 | 56,370 ms | 322 ms | 175.1 times |
| 15,000 | 10,000 | 113,063 ms | 636 ms | 177.7 times |
| 15,000 | 15,000 | 169,233 ms | 1,020 ms | 165.9 times |

### T-Test

The result shows that GPU can accelerate this data mining algorithm slightly. And GPU acceleration perform better with bigger data size with around 10 times acceleration. However, it can just accelerate around 4 times for small data size.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Size** | | **Average Processing Time** | | |
| **Sample Size** | **Feature Size** | **CPU Processor** | **GPU Accelerated Processor** | **GPU Accelerated** |
| 1,000 | 1,000 | 14 ms | 4 ms | 3.5 times |
| 1,000 | 5,000 | 57 ms | 14 ms | 4.0 times |
| 1,000 | 10,000 | 116 ms | 27 ms | 4.3 times |
| 1,000 | 15,000 | 169 ms | 42 ms | 4.0 times |
| 5,000 | 1,000 | 59 ms | 6 ms | 9.8 times |
| 5,000 | 5,000 | 277 ms | 30 ms | 9.2 times |
| 5,000 | 10,000 | 440 ms | 45 ms | 9.8 times |
| 5,000 | 15,000 | 672 ms | 64 ms | 10.5 times |
| 10,000 | 1,000 | 108 ms | 9 ms | 12 times |
| 10,000 | 5,000 | 442 ms | 51 ms | 8.6 times |
| 10,000 | 10,000 | 860 ms | 73 ms | 11.7 times |
| 10,000 | 15,000 | 1,273 ms | 143 ms | 8.9 times |
| 15,000 | 1,000 | 163 ms | 15 ms | 10.7 times |
| 15,000 | 5,000 | 661 ms | 62 ms | 10.7 times |
| 15,000 | 10,000 | 1,277 ms | 122 ms | 10.5 times |
| 15,000 | 15,000 | 1,915 ms | 180 ms | 10.6 times |

### Mutual Information

The result shows that GPU can accelerate this data mining algorithm slightly. And GPU acceleration perform better with bigger data size with around 8 times acceleration. However, it can just accelerate around 6 times for small data size.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Size** | | **Average Processing Time** | | |
| **Sample Size** | **Feature Size** | **CPU Processor** | **GPU Accelerated Processor** | **GPU Accelerated** |
| 1,000 | 1,000 | 33 ms | 5 ms | 6.6 times |
| 1,000 | 5,000 | 137 ms | 25 ms | 5.5 times |
| 1,000 | 10,000 | 275 ms | 52 ms | 5.3 times |
| 1,000 | 15,000 | 403 ms | 76 ms | 5.3 times |
| 5,000 | 1,000 | 119 ms | 14 ms | 8.5 times |
| 5,000 | 5,000 | 512 ms | 63 ms | 8.1 times |
| 5,000 | 10,000 | 963 ms | 123 ms | 7.8 times |
| 5,000 | 15,000 | 1,430 ms | 186 ms | 7.7 times |
| 10,000 | 1,000 | 237 ms | 25 ms | 9.5 times |
| 10,000 | 5,000 | 957 ms | 115 ms | 8.3 times |
| 10,000 | 10,000 | 1,851 ms | 210 ms | 8.8 times |
| 10,000 | 15,000 | 2,772 ms | 374 ms | 7.4 times |
| 15,000 | 1,000 | 332 ms | 39 ms | 8.5 times |
| 15,000 | 5,000 | 1,400 ms | 190 ms | 7.4 times |
| 15,000 | 10,000 | 2,793 ms | 348 ms | 8.0 times |
| 15,000 | 15,000 | 4,150 ms | 567 ms | 7.3 times |

### ReliefF

The result shows that performance of GPU acceleration is highly dependent on data size. For small data size, GPU can accelerate this algorithm around 20 to 40 times. And GPU acceleration perform better with bigger data size with around 60 times acceleration.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Size** | | **Average Processing Time** | | |
| **Sample Size** | **Feature Size** | **CPU Processor** | **GPU Accelerated Processor** | **GPU Accelerated** |
| 1,000 | 1,000 | 3,782 ms | 168 ms | 22.5 times |
| 1,000 | 5,000 | 10,688 ms | 366 ms | 29.2 times |
| 1,000 | 10,000 | 18,480 ms | 614 ms | 30.1 times |
| 1,000 | 15,000 | 26,384 ms | 865 ms | 30.5 times |
| 5,000 | 1,000 | 90,790 ms | 3,817 ms | 23.8 times |
| 5,000 | 5,000 | 252,141 ms | 5,504 ms | 45.8 times |
| 5,000 | 10,000 | 435,220 ms | 7,686 ms | 56.6 times |
| 5,000 | 15,000 | 618,388 ms | 10,133 ms | 61.0 times |
| 10,000 | 1,000 | 366,154 ms | 16,056 ms | 22.8 times |
| 10,000 | 5,000 | 1,005,610 ms | 21,309 ms | 47.2 times |
| 10,000 | 10,000 | 1,729,442 ms | 28,621 ms | 60.4 times |
| 10,000 | 15,000 | 2,459,125 ms | 35,571 ms | 69.1 times |
| 15,000 | 1,000 | 828,478 ms | 37,774 ms | 21.9 times |
| 15,000 | 5,000 | 2,263,180 ms | 47,918 ms | 47.2 times |
| 15,000 | 10,000 | 3,888,577 ms | 64,963 ms | 59.9 times |
| 15,000 | 15,000 | 5,526,391 ms | 79,575 ms | 69.4 times |

## Single GPU VS Multiple GPU

The test was conducted by running the data set 100 times for different number of GPU.

Y-Axis is the average processing time in ms, X-axis is the number of devices (GPUs). It shows that the performance was improved by increasing the number of GPUs.

|  |  |
| --- | --- |
| **Testing environment** | |
| **Server** | hpc6.cse.cuhk.edu.hk |
| **Data size** | 10,000 features x 1,000 samples |
| **GPU** | 1 x NVIDIA K20 5GB RAM |

### GPU Accelerated P-Value Processor

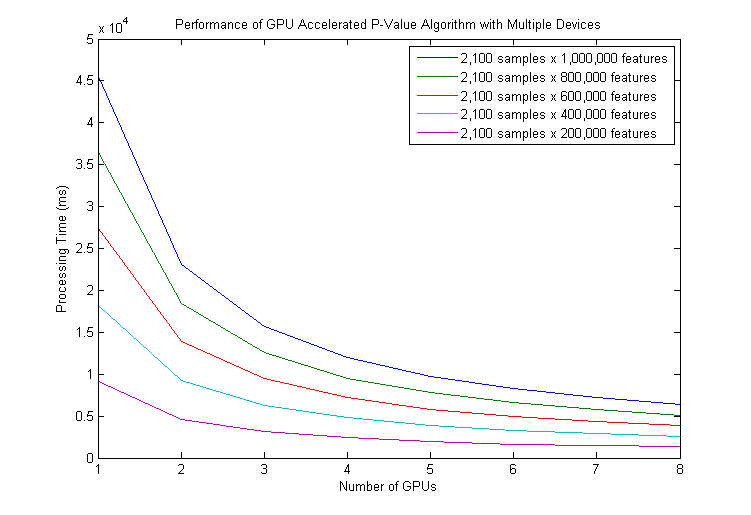


Figure : Processing Time VS Numbers of GPUs

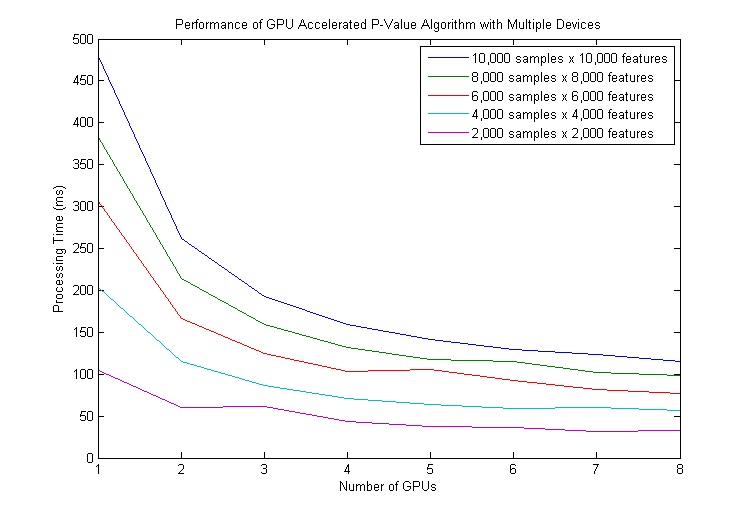


Figure : Processing Time VS Numbers of GPUs

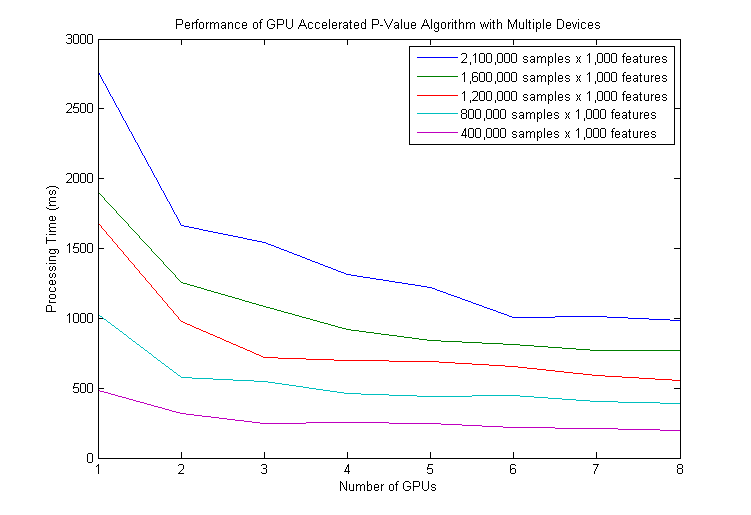


Figure : Processing Time VS Numbers of GPUs

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Data Size** | |  | **Number of GPUs** | | | | | | | |
| **Sample Size** | **Feature Size** | **x1** | **x2** | **x3** | **x4** | **x5** | **x6** | **x7** | **x8** |
| 2,100 | 1,000,000 | Processing Time (ms) | 45,612 | 23,072 | 15,692 | 11,940 | 9,674 | 8,262 | 7,232 | 6,431 |
| Accelerated (times) | N/A | 2.0 | 2.9 | 3.8 | 4.7 | 5.5 | 6.3 | 7.1 |
| 2,100 | 200,000 | Processing Time (ms) | 9,146 | 4,636 | 3,158 | 2,401 | 1,980 | 1,665 | 1,470 | 1,325 |
| Accelerated (times) | N/A | 2.0 | 2.9 | 3.8 | 4.6 | 5.5 | 6.2 | 6.9 |
| 10,000 | 10,000 | Processing Time (ms) | 481 | 262 | 193 | 159 | 142 | 130 | 123 | 115 |
| Accelerated (times) | N/A | 1.8 | 2.5 | 3.0 | 3.4 | 3.7 | 3.9 | 4.2 |
| 2,000 | 2,000 | Processing Time (ms) | 105 | 60 | 61 | 43 | 38 | 36 | 32 | 33 |
| Accelerated (times) | N/A | 1.8 | 1.7 | 2.4 | 2.8 | 2.9 | 3.3 | 3.2 |
| 2,100,000 | 1,000 | Processing Time (ms) | 2,770 | 1,662 | 1,543 | 1,316 | 1,220 | 1,003 | 1,011 | 988 |
| Accelerated (times) | N/A | 1.7 | 1.8 | 2.1 | 2.3 | 2.8 | 2.7 | 2.8 |
| 400,000 | 1,000 | Processing Time (ms) | 487 | 318 | 247 | 251 | 246 | 219 | 209 | 199 |
| Accelerated (times) | N/A | 1.5 | 2.0 | 1.9 | 2.0 | 2.2 | 2.3 | 2.4 |

The numbers show that using multiple GPUs can accelerate this data mining algorithm well for small sample sizes, which is around 7 times faster. But not so good for big sample sizes, which is around 3 times faster.

### GPU Accelerated T-Test Processor

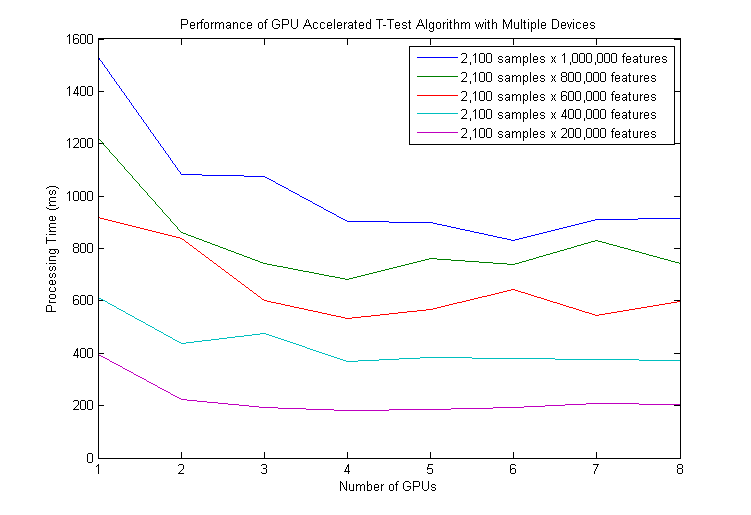


Figure : Processing Time VS Numbers of GPUs

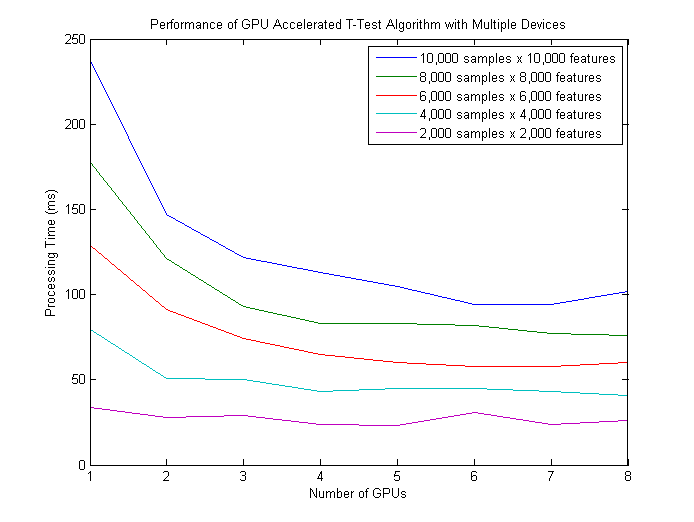


Figure : Processing Time VS Numbers of GPUs (regenerate)

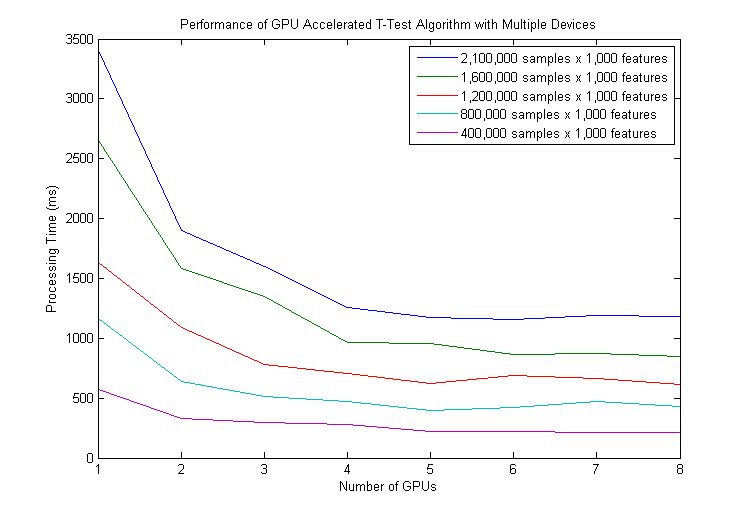


Figure : Processing Time VS Numbers of GPUs

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Data Size** | |  | **Number of GPUs** | | | | | | | |
| **Sample Size** | **Feature Size** | **x1** | **x2** | **x3** | **x4** | **x5** | **x6** | **x7** | **x8** |
| 2,100 | 1,000,000 | Processing Time (ms) | 1,533 | 1,082 | 1,076 | 904 | 898 | 831 | 911 | 916 |
| Accelerated (times) | N/A | 1.4 | 1.4 | 1.7 | 1.7 | 1.8 | 1.7 | 1.7 |
| 2,100 | 200,000 | Processing Time (ms) | 395 | 222 | 193 | 182 | 184 | 191 | 209 | 203 |
| Accelerated (times) | N/A | 1.8 | 2.0 | 2.2 | 2.1 | 2.1 | 1.9 | 1.9 |
| 10,000 | 10,000 | Processing Time (ms) | 238 | 147 | 122 | 113 | 105 | 94 | 94 | 102 |
| Accelerated (times) | N/A | 1.6 | 2.0 | 2.1 | 2.3 | 2.5 | 2.5 | 2.3 |
| 2,000 | 2,000 | Processing Time (ms) | 34 | 28 | 29 | 24 | 23 | 31 | 24 | 26 |
| Accelerated (times) | N/A | 1.2 | 1.7 | 1.4 | 1.5 | 1.1 | 1.4 | 1.3 |
| 2,100,000 | 1,000 | Processing Time (ms) | 3,417 | 1,903 | 1,602 | 1,255 | 1,176 | 1,155 | 1,191 | 1,182 |
| Accelerated (times) | N/A | 1.8 | 2.1 | 2.7 | 2.9 | 3.0 | 2.9 | 2.9 |
| 400,000 | 1,000 | Processing Time (ms) | 574 | 326 | 300 | 278 | 223 | 223 | 212 | 215 |
| Accelerated (times) | N/A | 1.8 | 1.9 | 2.1 | 2.6 | 2.6 | 2.7 | 2.7 |

The numbers show that using multiple GPUs can accelerate this data mining algorithm slightly. And GPU acceleration perform better with bigger data size with around 3 times acceleration. When sample size is small, the overhead of using multiple GPUs degrades the performance.

### GPU Accelerated Mutual Information Processor

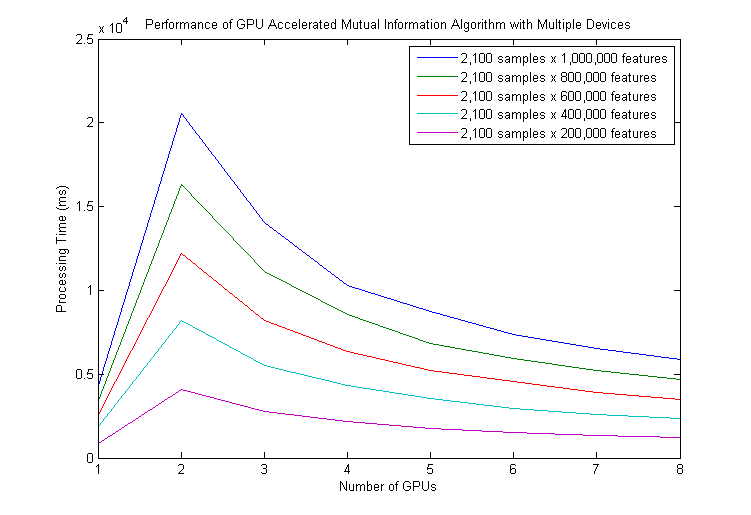


Figure : Processing Time VS Numbers of GPUs

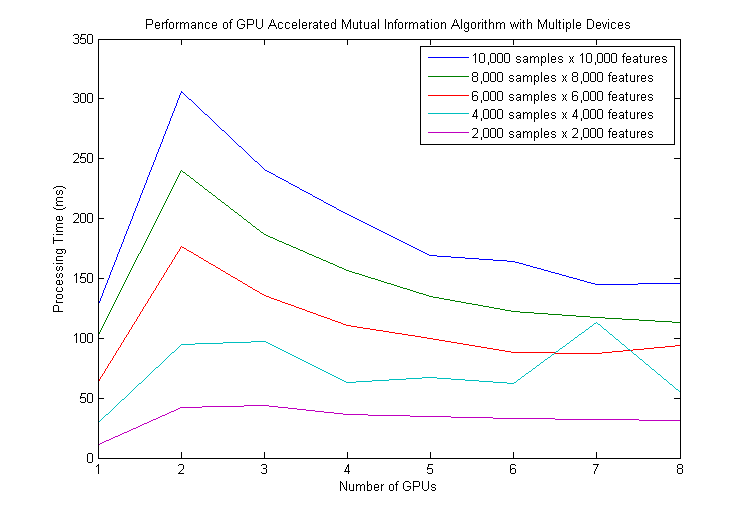


Figure : Processing Time VS Numbers of GPUs

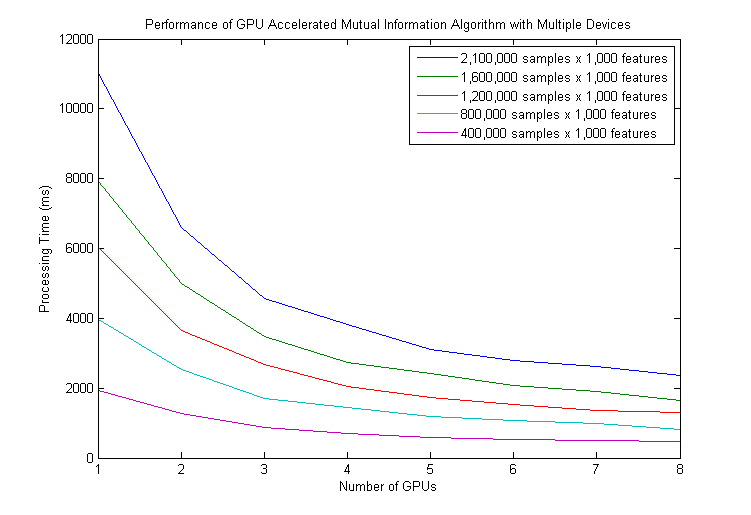


Figure : Processing Time VS Numbers of GPUs

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Data Size** | |  | **Number of GPUs** | | | | | | | |
| **Sample Size** | **Feature Size** | **x1** | **x2** | **x3** | **x4** | **x5** | **x6** | **x7** | **x8** |
| 2,100 | 1,000,000 | Processing Time (ms) | 4,237 | 20,551 | 14,042 | 10,298 | 8,723 | 7,362 | 6,526 | 5,854 |
| Accelerated (times) | N/A | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.6 | 0.7 |
| 2,100 | 200,000 | Processing Time (ms) | 841 | 4,097 | 2,767 | 2,156 | 1,757 | 1,507 | 1,329 | 1,199 |
| Accelerated (times) | N/A | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.6 | 0.7 |
| 10,000 | 10,000 | Processing Time (ms) | 127 | 306 | 241 | 203 | 169 | 164 | 145 | 146 |
| Accelerated (times) | N/A | 0.4 | 0.5 | 0.6 | 0.8 | 0.8 | 0.9 | 0.9 |
| 2,000 | 2,000 | Processing Time (ms) | 11 | 42 | 44 | 36 | 35 | 33 | 32 | 31 |
| Accelerated (times) | N/A | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.4 |
| 2,100,000 | 1,000 | Processing Time (ms) | 11,049 | 6,603 | 4,569 | 3,812 | 3,101 | 2,785 | 2,624 | 2,375 |
| Accelerated (times) | N/A | 1.7 | 2.4 | 2.9 | 3.6 | 4.0 | 4.2 | 4.7 |
| 400,000 | 1,000 | Processing Time (ms) | 1,944 | 1,268 | 873 | 698 | 588 | 525 | 490 | 462 |
| Accelerated (times) | N/A | 1.5 | 2.2 | 2.8 | 3.3 | 3.7 | 4.0 | 4.2 |

The numbers show that using multiple GPUs can accelerate this data mining algorithm slightly. And GPU acceleration perform better with bigger data size with around 4.5 times acceleration. When sample size is small, the overhead of using multiple GPUs degrades the performance.

### GPU Accelerated ReliefF Processor

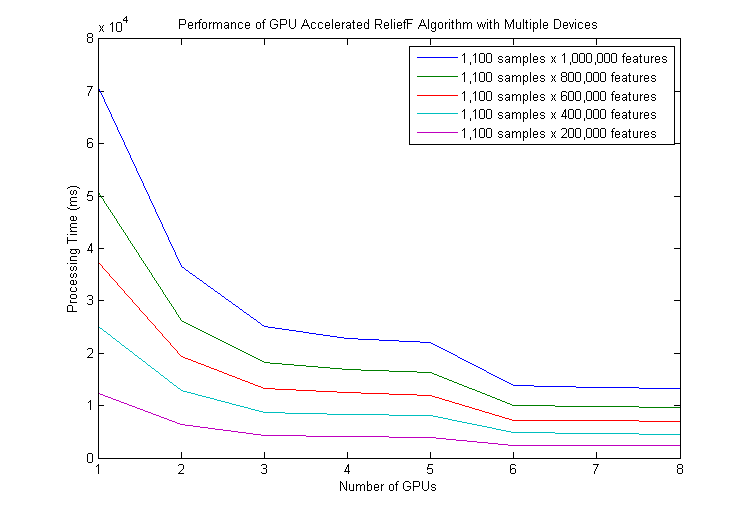


Figure : Processing Time VS Numbers of GPUs

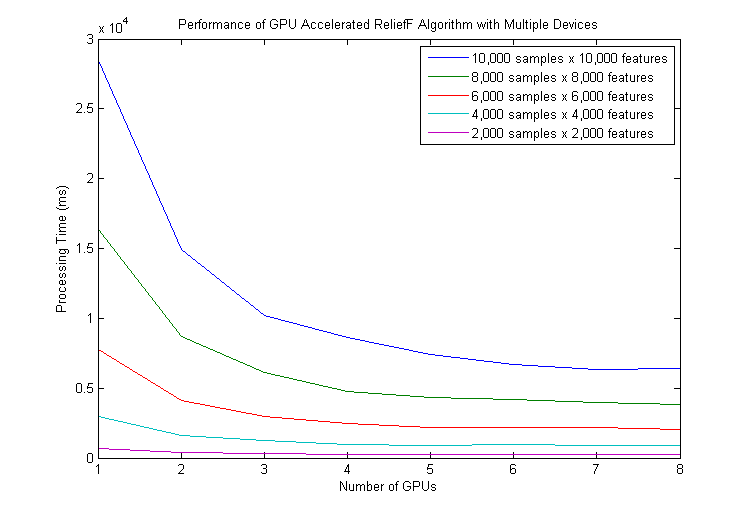


Figure : Processing Time VS Numbers of GPUs

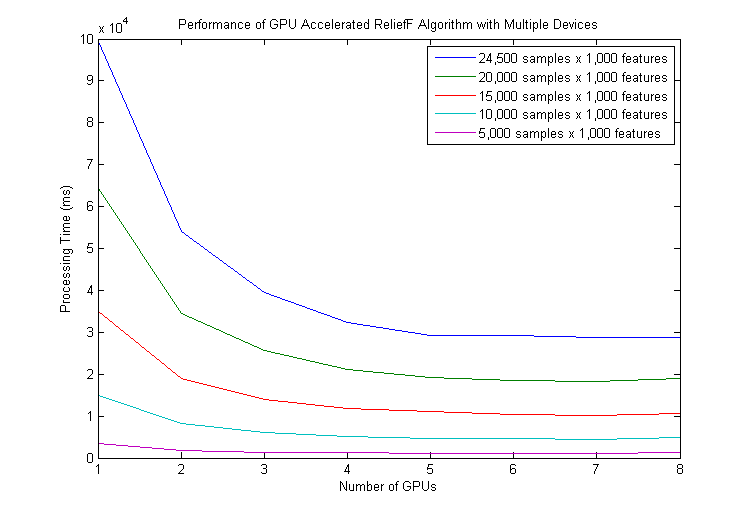


Figure : Processing Time VS Numbers of GPUs

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Data Size** | |  | **Number of GPUs** | | | | | | | |
| **Sample Size** | **Feature Size** | **x1** | **x2** | **x3** | **x4** | **x5** | **x6** | **x7** | **x8** |
| 1,100 | 1,000,000 | Processing Time (ms) | 70,982 | 36,593 | 25,175 | 22,811 | 22,023 | 13,915 | 13,446 | 13,248 |
| Accelerated (times) | N/A | 1.9 | 2.8 | 3.1 | 3.2 | 5.1 | 5.3 | 5.4 |
| 1,100 | 200,000 | Processing Time (ms) | 12,309 | 6,320 | 4,339 | 4,161 | 3,968 | 2,391 | 2,365 | 2,393 |
| Accelerated (times) | N/A | 1.9 | 2.8 | 3.0 | 3.1 | 5.1 | 5.2 | 5.1 |
| 10,000 | 10,000 | Processing Time (ms) | 28,546 | 14,949 | 10,168 | 8,663 | 7,399 | 6,713 | 6,355 | 6,406 |
| Accelerated (times) | N/A | 1.9 | 2.8 | 3.3 | 3.9 | 4.3 | 4.5 | 4.5 |
| 2,000 | 2,000 | Processing Time (ms) | 653 | 381 | 325 | 283 | 240 | 255 | 263 | 283 |
| Accelerated (times) | N/A | 1.7 | 2.0 | 2.3 | 2.7 | 2.6 | 2.5 | 2.3 |
| 24,500 | 1,000 | Processing Time (ms) | 99,773 | 54,164 | 39,525 | 32,315 | 29,167 | 29,128 | 28,782 | 28,856 |
| Accelerated (times) | N/A | 1.8 | 2.5 | 3.1 | 3.4 | 3.4 | 3.5 | 3.5 |
| 5,000 | 1,000 | Processing Time (ms) | 3,383 | 1,885 | 1,428 | 1,244 | 1,173 | 1,171 | 1,160 | 1,226 |
| Accelerated (times) | N/A | 1.8 | 2.4 | 2.7 | 2.9 | 2.9 | 2.9 | 2.8 |

The numbers show that using multiple GPUs can accelerate this data mining algorithm well. GPU acceleration perform better with bigger data size with around 4 to 5 times acceleration. When sample size is small, using multiple GPUs can still accelerate 2 to 3 times.

# Discussion

## The Zig Zag Curve Shape

In the experiment results of "Performance of GPU Accelerated Algorithms in Different Thread Numbers" for P-Value processor, a "Zig Zag" curve shape was found.

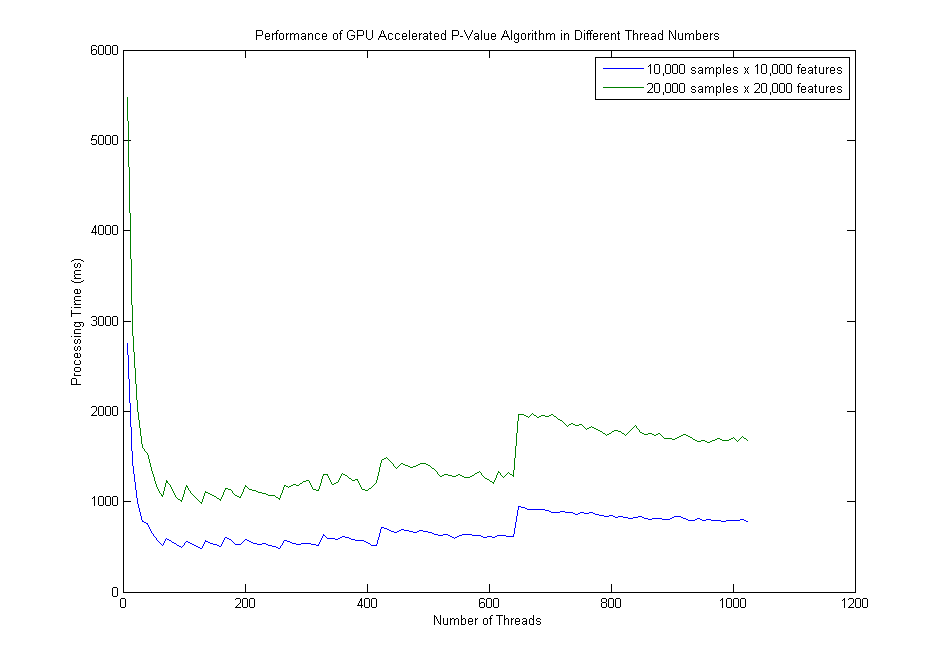


Figure : Zig Zag Curve Shape of The Different Numbers of Threads Performance Test

The minimum point of every "Zig Zag" happens in the multiplications of 32. Which is related to the 64 warp size of NVIDIA K20 GPU.

According to "CUDA Programming, A Developer's Guide to Parallel Computing with GPUs", "Warps are the basic unit of execution on the GPU." "Each group of threads, or warps, is executed together." "In the idea case, only one fetch from memory for the current instruction." (Cook, 2013). That means configuring the number of threads in a block to the power of warp size can minimize the number of fetches. In fact, the best performance happened at 128 threads in this data size.

## Performance of GPU Acceleration among Algorithms

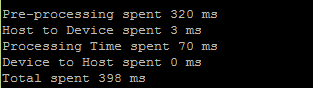
In this project, although we specially selected 4 algorithms that are good for data parallelism, their degree of acceleration varies a lot. When running big data set, GPU accelerated P-Value score calculation around 200 times, ReliefF score calculation around 60 times, but T-Test score calculation around 10 times and Mutual Information calculation around 8 times.

We found that unlike T-Test score calculation and Mutual Information calculation, both P-Value and ReliefF have to repeat some process over millions of times in different stages in some for loops. That's why they can gain much performance improvement in data parallelism.

## Limitation

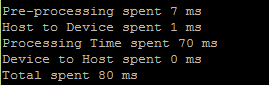
When a NVIDIA GPU is inactivated, the first API call to the GPU always takes around 300-400ms for activation. It is because the first run "might well be initialization overhead that only happens once." It is possible to "run nvidia-smi in daemon mode, that is a CUDA client that keeps the driver active and prevents it shutting down." (avidday, 2010). Retrieved from https://devtalk.nvidia.com/default/topic/464519/gpu-cards-warming-up-/

Following are the first and second run of P-Value GPU Processor with 256 threads for the 1,000 feature data set.



GPU WARM UP ~310ms

Figure : Before GPU warm up



Reduced after GPU WARM UP

Figure : After GPU warm up

## Future Work

Finding the best configuration for running a GPU accelerated algorithm by trying different combination (number of threads and number of GPUs) one by one is very time consuming. Hence one future work is to automate such process or predict the best configuration by a formula, it can save users lots of time.

Beside, because of the limited time frame, we couldn't repeat our experiments lots of time to average out noises, especially those for T-Test GPU acceleration. Also, we didn't have time to push not GPU accelerated implementations off their limit for comparisons. So more experiments have to be done in the future.

In addition, it is worth to analyze the complexity of GPU accelerated algorithms in different conditions. We may further improve the performance with the results of complexity analysis.

Lastly, we can GPU accelerate more data mining algorithms.

# Conclusion

According to our experiment results, the performance improvement of GPU acceleration is significant for our selected algorithms. Even though the performance of P-Value algorithm is not fully optimized, we found that with suitable setup, it can be accelerated for 175 (big data set) x 7 (8 GPUs) ~ 1,200 times if comparing with CPU. GPU acceleration can also improve some simple algorithms calculation like T-Test around 7(big data set) x 2 (8 GPUs) ~ 14 times. Which shows that GPU acceleration is promising for data mining, it is worth to apply GPU parallel computing to most of the data mining algorithms as they are good for data parallelism in nature. In fact, GPU is widely adopted to accelerate bioinformatics analysis.

Besides, as we completed an GPU acceleration architecture, and developed a full set of performance test tools for different configuration setups, we can integrate and accelerate new algorithms with this architecture efficiently.

# Schedule

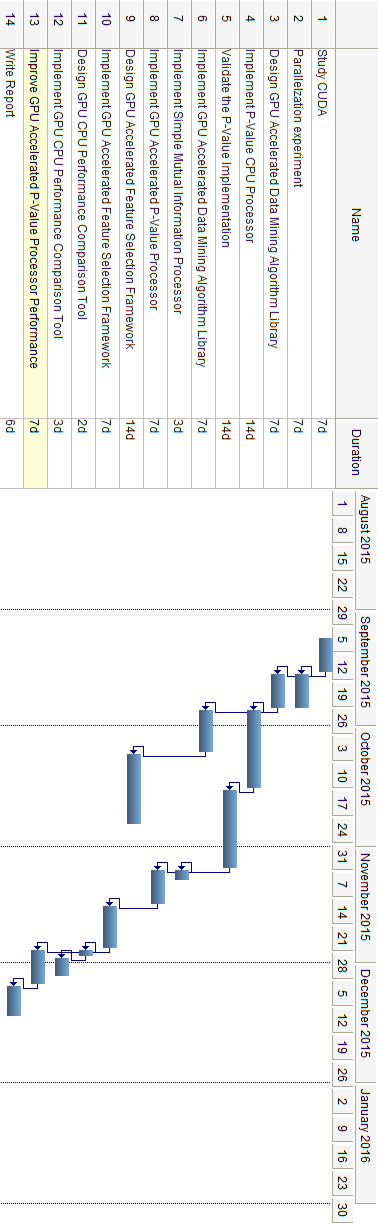


Figure : Semester 1 Schedule

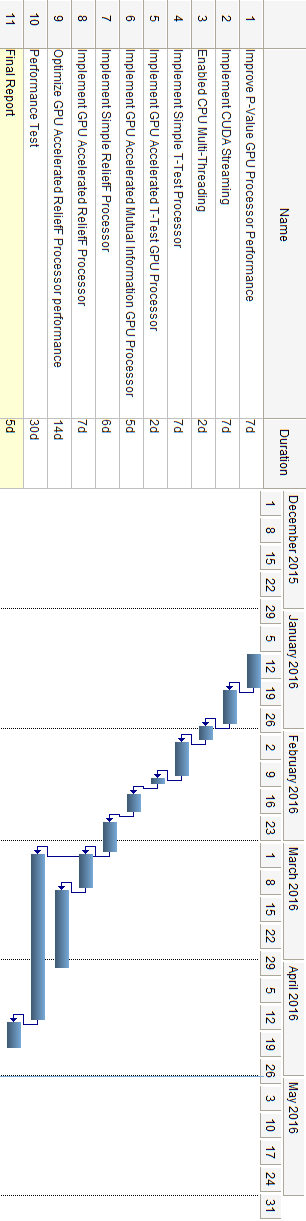


Figure : Semester 2 Schedule

# Reference

Abi-Chahla, F. (2008, June 18). *Nvidia's CUDA: The End of the CPU?* Retrieved from Tom's Hardware: http://www.tomshardware.com/reviews/nvidia-cuda-gpu,1954.html

*About CUDA*. (n.d.). Retrieved from NVIDIA Developer: https://developer.nvidia.com/about-cuda

avidday. (2010, April 14). *GPU cards warming up?* Retrieved from NVIDIA Developer Forums: https://devtalk.nvidia.com/default/topic/464519/gpu-cards-warming-up-/

Cook, S. (2013). WARPS. In S. Cook, *CUDA Programming A Developer's Guide to Parallel Computing with GPUs* (pp. 91-92). 225 Wyman Streetm Waltham, MA 02451, USA: Morgan Kaufmann.

*CUDA - Wikipedia, the free encyclopedia*. (n.d.). Retrieved from Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/CUDA#Language\_bindings

*Linux GPU Instances - Amazon Elastic Compute Cloud*. (n.d.). Retrieved from Amazon Web Services (AWS) - Cloud Computing Services: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using\_cluster\_computing.html

Pocock, A. (n.d.). *MIToobox for C and MATLAB*. Retrieved from School of Computer Science | The University of Manchester: http://www.cs.man.ac.uk/~pococka4/MIToolbox.html

*Rosetta Code Home Page*. (n.d.). Retrieved from Rosetta Code Home Page: http://rosettacode.org/wiki/Rosetta\_Code

*scikit-learn - Wikipedia, the free encyclopedia*. (n.d.). Retrieved from Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/Scikit-learn

*SciPy Home Page*. (n.d.). Retrieved from SciPy.org: http://www.scipy.org/

*TensorFlow Home Page*. (2015). Retrieved from TensorFlow: https://www.tensorflow.org/

Urbanowicz, R. J.-A. (2012). Urbanowicz, R. J., Kiralis, J., Sinnott-Armstrong, N. A., Heberling, T., Fisher, J. M., & Moore, J. H. *BioData mining*, 5(1), 1-14.

Waikato, D. o. (2008, November 1). *Attribute-Relation File Format (ARFF)*. Retrieved from Department of Computer Science : University of Waikato: http://www.cs.waikato.ac.nz/ml/weka/arff.html

*Weka 3 - Data Mining with Open Source Machine Learning Software in Java*. (n.d.). Retrieved from Department of Computer Science : University of Waikato: http://www.cs.waikato.ac.nz/ml/weka/