Liu Pengfei, *1155058548*, Li Shuai, *1155073198*, Yi Weiying *1155026053*

Group 13

[[1]](#footnote-1)

A Hybrid MapReduce Framework for SNP Selection

*Abstract*— with the development of next generation sequencing technology, researchers are facing data with higher and higher dimensions. Single nucleotide polymorphisms (SNP) selection become critical in bioinformatics since it is quite inaccuracy and inefficacy to do work in the high dimension space. In this paper, we implemented a CPU-GPU hybrid MapReduce Framework to accelerate the process of SNPs selection in GWAS. To evaluate our framework, we did experiments both on synthetic data and real data. The results show that our framework performed well both in term of acceleration and accuracy. In addition, a web based interface is built to make our framework friendly to use.

*Index Terms*—SNP selection, MapReduce, GPU

# INTRODUCTION

N

owadays, next-generation sequencing technologies allow scientists to sequence a large number of samples in a reasonable time and cost which encourages biologist to perform research in data-driven manner e.g. genome-wide association studies (GWASs). In GWAS, statistical or computational analyses are applied to compare the DNA sequences of healthy sample (control) and patient of a specific genetic disease (case) in order to identify single nucleotide polymorphisms (SNPs) that are associated to complex genetic diseases like cancers and hepatitis’s. A single nucleotide polymorphism (SNP) is a genetic variation of a single nucleotide on a specific location of the DNA sequences across the samples in a population. As human is a diploid and each chromosome has 2 different copies, the genotype of a SNP is composed by two alleles (nucleotides). A SNP can be associated to a genetic disease as a single SNP or as a SNP-SNP interaction. SNPs which are associated to a genetic disease individually could be found through statistical test like p-value testing. Although most of the individually disease associated SNPs were found in various research, those SNPs alone were insufficient in explaining the heritability of those genetic diseases they were associated to. One possible reason is some disease associated SNP-SNP interactions remain undiscovered. However, it is difficult to detect these interactions. It is well-known that independent marginal effect of each SNP in most SNP-SNP interactions is small. Therefore non-linear SNP-SNP interactions can model the cause of a genetic disease better than linear models. However, detecting non-linear disease-associated SNP-SNP interactions suffers from the curse of dimensionality. One possible approach to improve the efficiency is to reduce the search space through carefully selecting a subset of SNPs that have high potential to be disease associated.

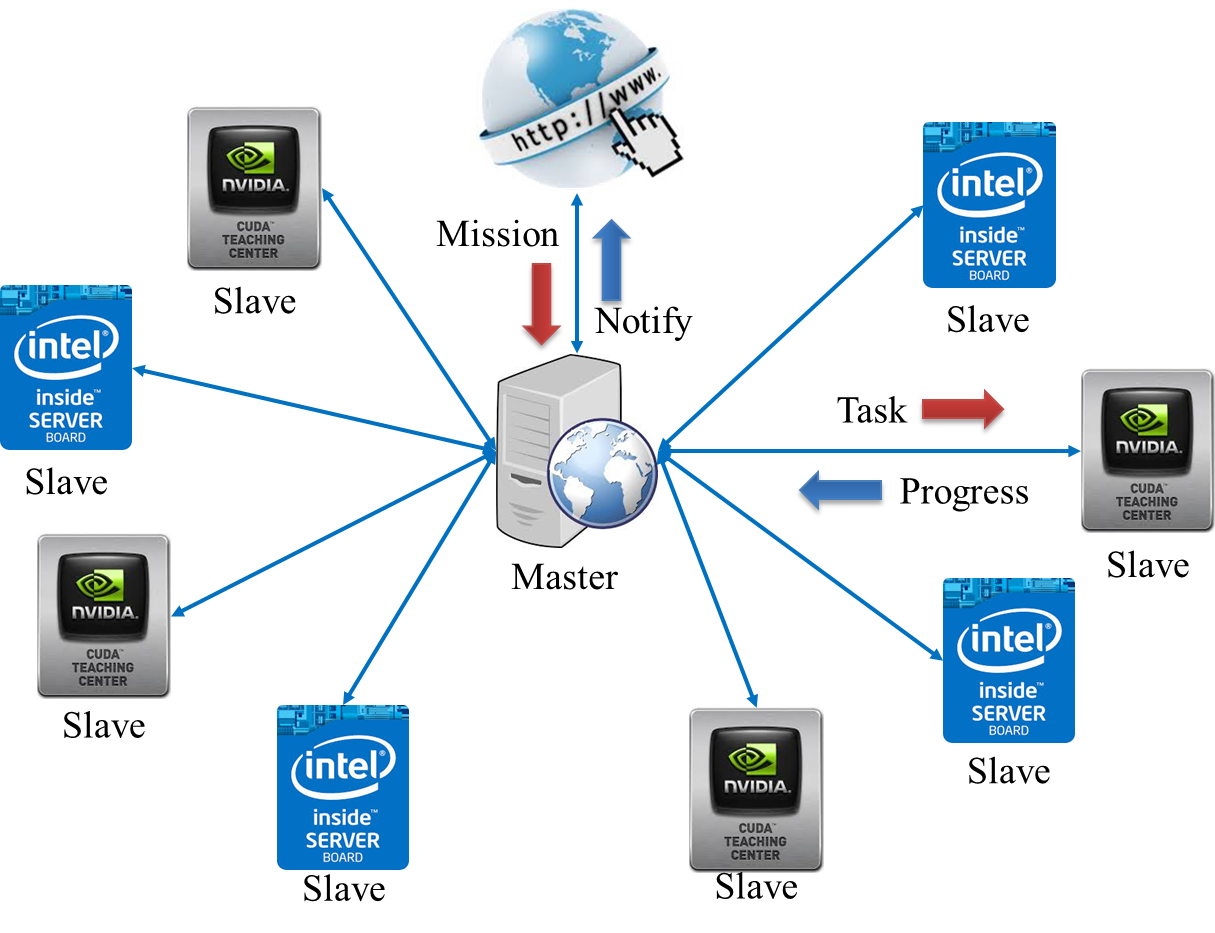


Fig 1 System Architecture

Parallel computing is becoming more and more important and popular in computer science or even our daily life because it has been proven to be critical when researching high performance solutions. Nowadays, we are facing enormous number of computing problems that involve mega large data sets and require tremendous computing power. Problems with such characteristics can be easily found in science and technology, natural sciences, information technology, and bioinformatics technology. Traditional sequential algorithms are no longer suitable for these problems because the computing resources and computing power of a single computer have reached their limits.

The evaluation of multi-core and many-core computing architecture (CPU based), and the parallel computing architecture (GPU based) confirm that parallelism is the best choice for speeding up an algorithm and solving the issue mentioned above. Today, many of these problems can be solved with much high efficiency by utilizing massive number of parallel processors.

In current data explosion era, parallel computing is essential to processing a massive volume of data when timing issues are critical. A parallel programming model called MapReduce, which is popularized by Google, is gaining significant attentions from both industry and academia. In the MapReduce model, there are two primitive functions. One is call Mapper(), and the other one is called Reducer(). The Mapper() takes a list of (key1, value1) pairs and outputs a list of intermediate values of the key, e.g. (key2, value2). The intermediate key-value pairs are then grouped together based on the keys, e.g. (key2, list(value2)) and forwarded to the Reducer() function. Then, the Reducer() function produces zero or more aggregated results, usually combining values of a specific key. The foundation of the Mapper() and Reducer() functions is that they are written in such a way that multiple Mapper() and Reducer() function can be executed concurrently. A MapReduce example is shown in the following figure

# ReLATE WORKS

With the increasing popularity of data mining, the application

of feature selection has been widen in various research area Generally speaking, there are 2 different kinds of feature selection algorithms. The first kind utilized one or more statistical properties of each attribute such as information entropy and t-test value, to search for potential useful features The second kind exploits the topology or the structure in the data space to search for potentially important features by measuring the inner relationships between samples and features.

As SNP-SNP interactions are most likely to be nonlinear, pure statistical algorithms which measured the statistical property of each SNP perform poorly in GWAS. Meanwhile structural based algorithms like Relief algorithm has been proven to be efficient and effective in prioritizing SNPs in GWAS.

Very Large Scale ReliefF (VLSRF) was developed to select important features from a dataset with enormous number of features through divide and conquer. First, it splits the original feature set into a full collection of feature subsets with a user specific size. The size of feature subset is significantly smaller than the original feature set to ensure that ReliefF can maintain a high detection power on those subsets. Second, the local score of every feature under each feature subsets is calculated independently through ReliefF algorithm. Finally the local feature scores calculated by each independent ReliefF execution are merged and the final global score of each feature is its maximum score across all its local score. Although VLSRF maintains a high detection accuracy under extremely large number of features, it has a much higher time complexity than ReliefF as a trade-off. To ensure at least one feature subset contains all important features and such that their association can be measured, an exponential number of feature subsets need to be evaluated.

# SYSTEM DESIGN

## Framework

To make the framework more stable and convenient to implement, we adopted the three layer architecture, as showed in figure 1.

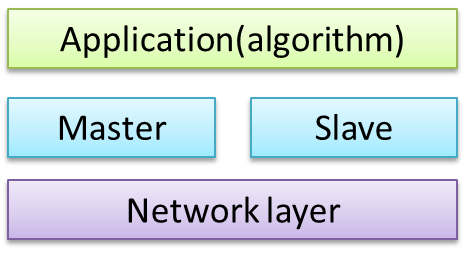


Fig.2. Framework architecture

Network layer is in the bottom of the architecture, which is responsible to provide stable communications between Master and Slaves. When sending the packages from upper layer, network layer will first serialize the packages into linear messages and then use socket to send the message. When receiving information, network layer will first de-serialize the linear message to construct the package and then report the package to upper layer.

There are two main functions for masters, managing the slaves and distributing task to slaves. To manage the slaves, the master keeps a list of connected slaves and the communication channel with each slave. There are two kinds of slaves, GPU slaves and CPU slaves. For GPU slaves, them are deployed in a GPU server and can take the advantages of hardware acceleration. For CPU servers, they use multiple threading methods to accelerating.

In the top of the architecture is the Application layer, which is responsible for splitting a big mission into many small tasks and monitoring the progress in executing. To split a big mission, we implement many different feature selection algorithms for SNPs. The application layers provide a default method for all algorithms, which will equally divide a big mission to sub mission based on the number of SNPs, and each algorithm can overwrite this default method to provide its owe way of splitting sub mission. To monitoring the progress, there is a scheduler that tracks the status for each task.

1. Network layer

Network layer can provide stable communication for upper layers by sending and receiving information among interface, master and slaves. The information transferring between interface and master includes

* **Mission information**, which describes the location of the input file, the name of the algorithm.
* **Notification information**, which contain the location of the result file.

The information transferring between master and slaves includes

* **Register information**, which contain ip address and hostname of the client
* **Task information**, which describes the location of the input file, the name of the algorithm, the task id, the the id of the mission that it belongs to, a hash table that contains a subset of SNPs in the input file, and the length of the hash table.
* **Progress information**, which contain the task id, the mission id that the task belongs to, an array of SNP scores.

The network layer defines method to serialize and de-serialize all these information.

1. Middle layer

According to our expectation, the number of slave for each master should be restricted below 100, which is not a very big number. So for each slave, master will create a thread to manage it. This solution is convenient to implement and can quickly response the request from slaves. Each threading should keep alive to provide a real time control over slaves

There are three kinds of information that the master may receive and the master should react to the information accordingly.

* **Registering from slave**

When receiving register information from the slave, the master will first check whether it is in the slave list based on the ip address of the slave. For a new slave, it will be in added in the slave list of the master, For an already exist slave, its information will be updated

* **Mission from website**

When receiving mission from website, master will first find out which algorithm is specified in the mission that then split the mission according to the algorithm. After that, the master will use the scheduler of the algorithm to track the progress .

* **Progress information**

When receiving a progress from slaves, master will first report it to the scheduler of the algorithm, and to ask the scheduler whether there is task to handle. If there is still unhandled task, the master will distribute it to slave, otherwise the master will report to the website that mission completed.

For slaves, there is only one kind of information that it can receive, which is the task information from master. when the master receive the task information, it will read the input file, and call the algorithm to handle the subset of SNPs contains in the hash table.

1. Application layer

In the application layer, we provide a split method to split the whole input file into many smaller tasks, and a scheduler method to control the executing process. For the splitting method, since the computing power of GPU is better than that of a CPU, so it create different size of Tasks to GPU slave and CPU slave to achieve performance balance. Roughly speaking, the computing power of a single Kepler GPU board is 50 time of the single CPU core, considering the multi-board acceleration within GPU slave and multi-core acceleration within CPU Slave, the ratio of task size for GPU and CPU should be . For the scheduler method, it keeps a list of all the task that were created by the splitting method. In the beginning, it distribute a task for each slave, after that, when receive a progress from a slave, it will give a new task to that slave again until all task has been processed. Finally, the scheduler will send a complete signal to the master to notify the website that mission complete.

Some algorithm may need a special method for splitting and scheduling. For example, VLSReliefF will compute the score for each SNP for more than one time to keep that highest one, so we provide an overwrite architecture to provide algorithm specific splitting and scheduling method.

## Master and Slave

Although Master is the center of the system which is responsible for control all the slave and working process of the system. The main function the master has been introduced in scheduler, so we will focus on introducing the slave node in this part.

1. Uniform interface

It is on the slave that many different SNP selection algorithms are performed, thus it is critical to define a uniform interface to all the algorithms.

Initially, slave receives a task from the master, which contains the following information:

* Input file location
* Algorithm id
* A hash table
* Hash table length

The slave will take care of the preprocessing such as file reading, and transfer the input file in to a matrix, where each row is a sample and each column is a SNP. Then slave will call the wanted algorithm based on the algorithm id. So the input interface for all the algorithms is

* Data matrix
* A hash table
* Hash table length

After finishing processing, algorithm should return a score list to slave to form progress report, so the output interface for algorithm is

* Score list

1. Acceleration

For the two kind’s slaves, GPU slave and CPU slave, they adopt different acceleration mechanism. For CPU slave, they use multiple thread method to process as many as possible SNPs

For GPU slave, they use CUDA, which is the parallel computing platform and application programming interface created by NVIDIA

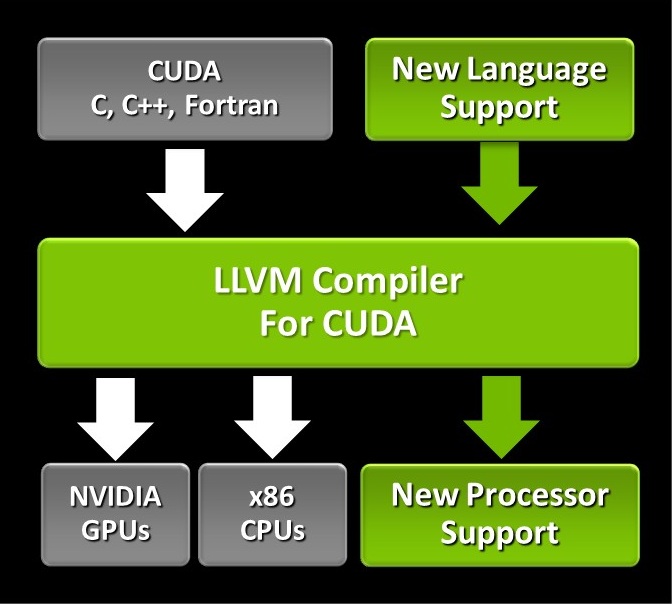


Fig.3. Framework architecture

## Web Interface

To make the system more convenient to use, we provide a web based interface to users. The web interface provides the following functions:

1. File management

.ARFF file is widely used in bioinformatics for storing data because of its conciseness. Our framework takes .arff file as the default file type. Users can upload his .arff file into the system through our web interface, and the file will be store in our server for future use.

1. Various feature selection algorithms

Based on the execution process, feature selection algorithms are divided into three categories: filter, wrapper, and embedded algorithms. All of these three types of feature selection algorithm are supported in our frame work. To start a mission, the user should choose one of his uploaded file and choose an algorithm in the algorithm list of our system.

1. Asynchronous notification

Since it may take minutes, even hours to process one .arff file, our framework takes an asynchronous method to inform the user of the result. After submit a process request, the user is free to browse other website, or even close the browser. When mission complete, the users will get a message from our system if he is still in our website, otherwise we will send an email to inform him, and the user can check the result next time he logins.

1. Privilege management

There are two types of users in our system: common user and administrator. For the administrator, except all the functions above, he has the privilege to control the background MapReduce framework, such as checking the online slave list and switch on/off a slave.

# IMPLEMENTATION

This system is implemented using C++ 11, a powerful version of C++. To achieve higher efficient, we do not use any third part library instead of the standard library in C++ 11.

## Framework

All information package that introduced in previous part are defined as *struct*, which will be serialized and de-serialized in the network layer to send using TCP protocol. To serialize a *struct*, we first store the components that with static length, such as int, float, and then store the components with dynamic length, such as char\*, int\*. The de-serialization method can work in a reverse order.

The fundamental communication function for master and slaves works like TCP server and client. The master creates a thread for each connected slave and keeps a mutex protected list to manage them. The slaves are also multiple threaded since that there may be many request from different users. All thread are created using standard C++ 11 library and detached from the main thread to free the main thread to monitoring the port instead of waiting sons to finish.

The MapReduce functions for master and slaves are integrated into a class named scheduler.

## ReliefF Slave

The system is design as a platform to run many feature selection algorithms, among which we choose the VLSReliefF algorithm to test the performance of the system.

Very Large Scale ReliefF (VLSRF) was developed to select important features from a dataset with enormous number of features through divide and conquer [9]. First, it splits the original feature set into a full collection of feature subsets with a user specific size. The size of feature subset is significantly smaller than the original feature set to ensure that ReliefF can maintain a high detection power on those subsets.

Second, the local score of every feature under each feature subsets is calculated independently through ReliefF algorithm. Finally, the local feature scores calculated by each independent ReliefF execution are merged and the final global score of each feature is its maximum score across all its local score. Although VLSRF maintains high detection accuracy under extremely large number of features, it has a much higher time complexity than ReliefF as a trade-off. To ensure at least one feature subset contains all important features and such that their association can be measured, an exponential number of feature subsets need to be evaluated. Its pseudocode is shown in figure 2.

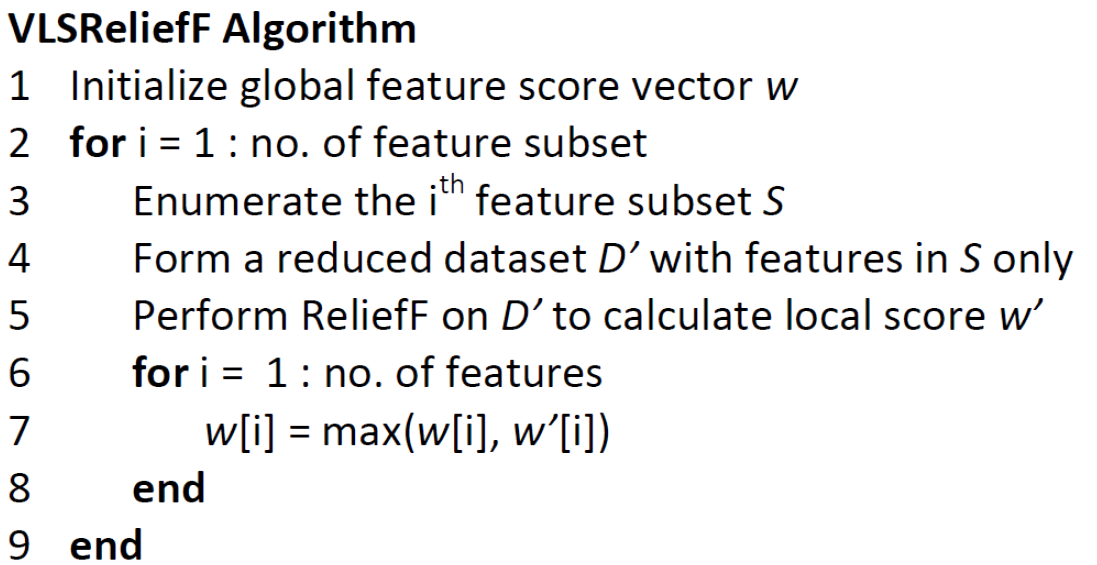


Fig.4. Pseudo code of VLSReliefF

## Web Interface

The Web Interface is implemented using PHP language, in which we keep a file table to manage user’s file and a list of supported algorithm in MapReduce framework. Asynchronous notification is achieved by AJAX, and information of different users are stored in the MySQL database.

# EXPERIMENTS AND RESULT

## Dataset

All synthetic datasets were generated by GAMETES which is a dataset generator specifically designed for generating highly accurate GWAS synthetic dataset. We generated 4 groups of datasets under different heritabilities using GAMETES and each dataset group contained 50 datasets.

The real dataset is found from a Parkinson's disease study called Mayo-Perlegen LEAPS (Linked Efforts to Accelerate Parkinson's Solutions) Collaboration which is originated from National Human Genome Research Institute to evaluate its performance on a real dataset. This study had two tiers and we picked the tier 2 for performing our experiment. Tier 2 had 660 samples (332 case and 332 control) and each sample had 3000 SNPs. Before we performed experiments on this dataset, we had performed data cleansing and converted it into ARFF file format with the encoding scheme described in section 3 such that it could be processed by our programs easily.

## Configuration

All the experiments on CPU implementations were performed on a workstation with one Intel I5-4670 3.4GHZ CPU and 8GB DDR3 main memory. On the other hand, all the experiments on the GPU implementations were performed on a high performance Linux server with two Intel Xeon E5-2670 2.6GHZ CPU, 128GB ECC DDR3 main memory and a NVIDIA GK110GL Kepler GPU

## Acceleration

We conduct experiments on both CPU and GPU slaves to test the performance of our frame work.

To check the acceleration caused by CPU, we test the relationship between execution time with the number of CPU slaves, which is showed in Fig. 5. In figure 5, the horizontal line is the number of CPU slaves, and the vertical line is the execution time on synthetic data. We can see that initially the processing time decreases almost exponentially with respect to the number of slaves. And it change little when the number of slaves exceed 10 because that communication become heavy over the whole net.

To check the acceleration caused by GPU, we made a comparison between the processing time of CPU and GPU in figure 6.

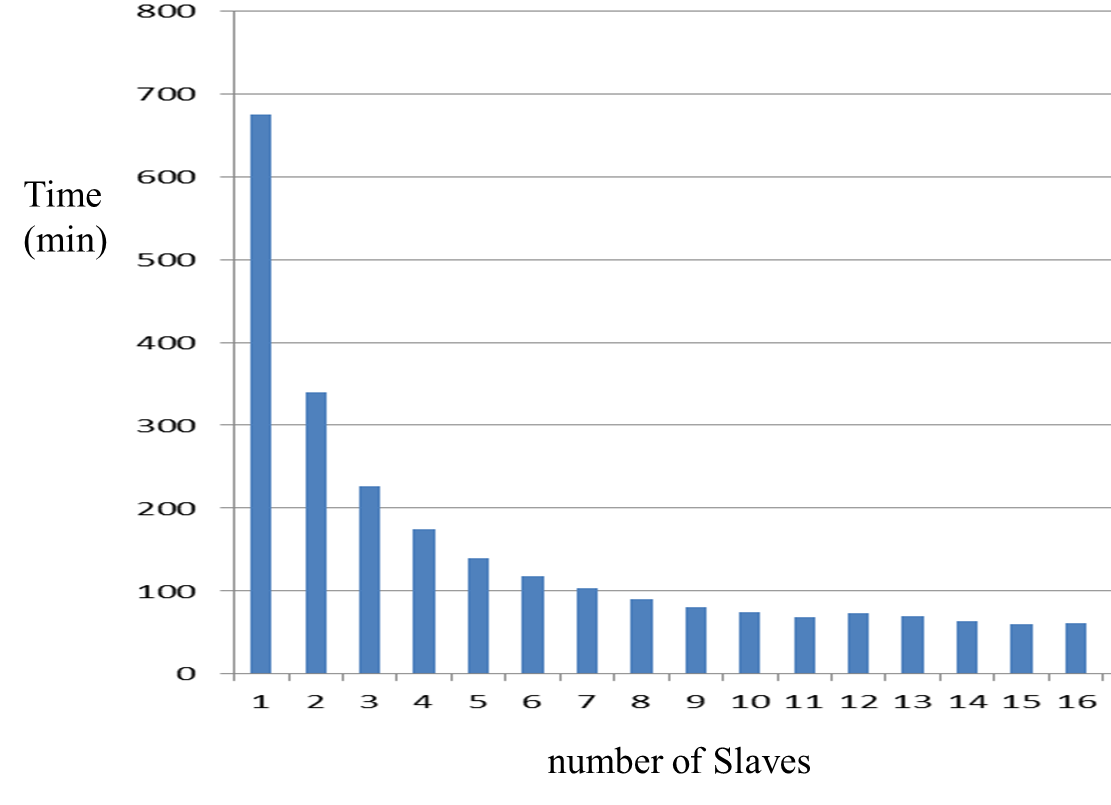


Fig.5 Acceleration with respect to the number of slaves

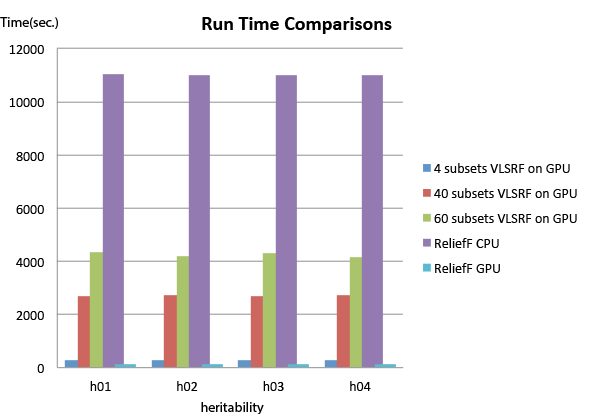


Fig. 6 processing time comparison between single core CPU and single board GPU

## Accuracy

After we computed the score of all SNPs using our GPU based VLSRF, we ranked those SNPs according to their score and extracted the top 10% SNPs for further analysis. Among those top 10% SNPs, there are 3 SNPs which are associated to Parkinson's disease related gene. These 3 SNPs are rs2287403, rs1979687 and rs2303703 which are associated to YLPM1, USP47 and MYO10 respectively and these 3 genes were all reported to be associated to Parkinson's disease by a Parkinson's disease database. Furthermore, gene USP47 and MYO10 were separately reported to be associated to Parkinson's disease under 2 more literatures. All these showed that our GPU based VLSRF algorithm are able to select disease associated SNPs effectively.

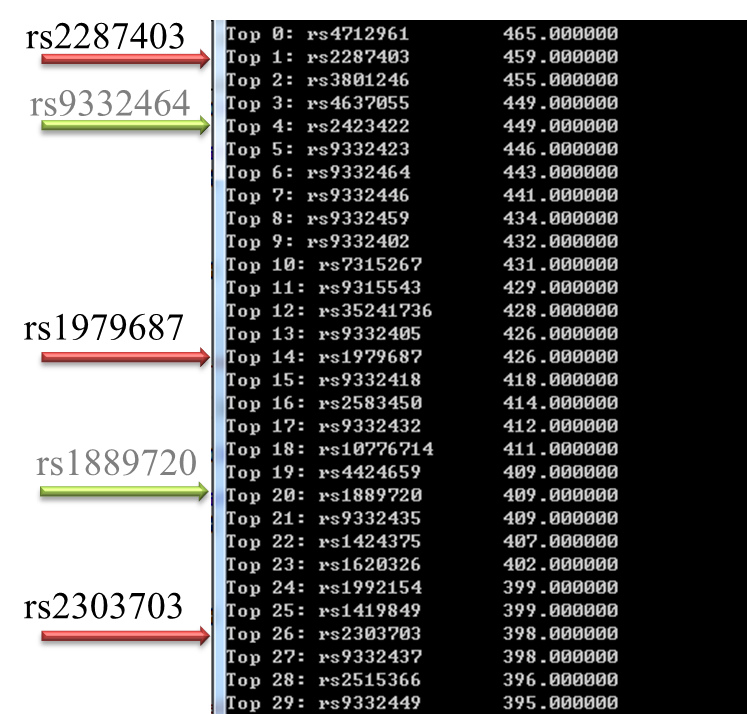


Fig.7 Meaningful result in Parkinson disease dataset

# Conclusion

In this paper, we have demonstrated MapReduce Framework for SNP selection and tested it thoroughly using multiple sets of synthetic datasets and a real GWAS dataset on Parkinson’s disease. Under various experiments, our Framework was shown to be an effective and efficient algorithm in identifying important SNPs. It outperformed single CPU based algorithms in term of speed while maintaining a comparable accuracy with them. Under the experiment of Parkinson’s disease dataset, it has identified SNPs consistent to existing literature.

References

1. J. K. Author, “Title of chapter in the book,” in *Title of His Published Book, x*th ed. City of Publisher, Country if not

1. \*This report is for the class of big data analytics CSCI5108 CSE CUHK. [↑](#footnote-ref-1)