# A Hybrid Distributed Framework for SNP Selections

**Pengfei Liu**[1], **Shuai Li**[1], **Weiying Yi** [1] **and Kwong Sak Leung**[1]
[1]Department of Computer Science and Engeering
The Chinese University of Hong Kong, Hong Kong

**Abstract**— *With the development of next generation sequencing technology, researchers are able to obtain extremely high-dimensional data. However, only a fraction of the data is related to diseases and the computational time on processing the whole sequences is tremendous. Moreover, using the high-dimensional data directly will greatly reduce the accuracy of the machine learning and data mining algorithms. Single nucleotide polymorphism (SNP) selections is critical for addressing these problems in genome wide association study (GWAS). Typically, it needs days to perform SNP selections even though the relationship between SNPs and diseases is assumed to be linear. More time is needed when the relationship is nonlinear. In order to speed up the SNP selection processes, a CPU-GPU hybrid distributed framework (HDF) specifically for SNP selection algorithms is introduced in this paper. The HDF fully utilizes the computational power of machines. And the interfaces are also provided, which help researchers to extend their SNP selection algorithms into distributed version. The results demonstrate that the acceleration by HDF is about hundreds times on SNP selections with synthetic and real data, compared to single machine.*

**Keywords:** SNP selection, CPU, GPU, distributed

## 1. Introduction

Genome-wide association study (GWAS) is an analysis to identify which part of the human genomes are associated to a certain trait. In GWAS, statistical and computational analyses are applied to compare the DNA sequences of the controls (healthy samples) and the cases (patients with the specific genetic disease) in order to identify the related single nucleotide polymorphisms (SNPs) [1][2][3]. With the technology of next generation sequencing, researchers are able to obtain millions of SNPs in DNA sequences. However, only a small fraction of the SNPs are related to diseases, and the rest are irrelevant and regarded as noises. These noises will severely reduce the accuracy and reliability of the GWAS algorithms [4]. Hence, identifying the useful SNPs before analyzing their relationship with diseases is a critical issue in GWAS.

There are about 4 million SNPs in human DNA sequences, and many SNPs work in coordination to manifest a disease [5]. Analyzing such a high dimensional combinatorial relationship increases the computational complexity a lot. To speed up the process of SNP selection, parallel computing is adopted in this work.

Nowadays, most of the computers are equipped with both CPUs and GPUs. In order to maximize the utilization of the computing resources, a CPU-GPU hybrid distributed framework (HDF) specifically for SNP selections is proposed and implemented. The HDF provides interfaces that help researchers extend their SNP selection algorithms into distributed versions. To the authors' best knowledge, the proposed HDF is the first CPU-GPU based distributed system specifically designed for speeding up the SNP selection processes.

The HDF consists of three components:

- **Controller**
  The Controller decomposes the original computational mission from a user into many small tasks, and distributes them to the CPU clients and GPU clients described below. After receiving the progress report from the clients, the Controller merges the progresses and find a new task to distribute, which can be manipulated using the provided interfaces.
- **CPU client**
  The CPU clients use multi-thread architecture to handle the tasks distributed by the Controller and return the results back to the Controller after finishing the task.
- **GPU client**
  The GPU clients use Nvidia CUDA API to process the tasks distributed by the Controller and return the results back to the Controller after finishing the task.

To test the performance of the HDF, we implement an SNP selection algorithm ReliefF on the HDF using the provided interfaces. The experimental results show that the distributed version is about hundreds times faster than the a single thread CPU version.

The rest of the paper is organized as follows. Literature reviews on SNP selection algorithms and distributed systems are introduced in Section 3. Section 4 and 5 are the architecture design and the implementation of the HDF. Experiments are performed in Section 6 and the results are analyzed in Section 7. Section 8 is the conclusion and discussion.

## 2. Definition of SNP Selections

Each SNP in human genome is either an A-T pair or a C-G pair. For each SNP, the pair with higher probability is called the major allele, otherwise it is called the minor allele.

For each pair of chromosomes, there are three different kinds of SNP combinations, major-major, major-minor, and minor-minor.

**Input:** A dataset that contains the cases and the controls. Each sample stores an array of human SNP genotypes, which are encoded in Table 1.

| Genotype | Value |
|---|---|
| missing | 0 |
| major allele–minor allele | 1 |
| major allele–minor allele | 2 |
| minor allele–minor allele | 3 |

Table 1: SNPs encoding scheme

**Output:** The association weight between each SNP, or a set of SNPs, and the target disease.

# 3. Related Work

With the increasing dimension of data, feature (that is SNP in this paper) selection becomes more and more important in various research areas [6][7][8]. Generally speaking, there are three types of feature selection algorithms. *Filters* utilizes one or more statistical properties of each attribute, such as information entropy and t-test value, to identify useful features [9]. *Wrappers* evaluate the association between labels and groups of features. *Embedded methods* treat features selection as part of main algorithms.

Bacause SNPs may work in coordination to manifest a disease and the interactions between SNPs probably are nonlinear, it takes days to perform SNP selections [10].

The ReliefF [11] and its variations [12] [13][9] are widely used in SNP selections. It evaluates every SNP in the following steps. First, it compute the distance matrix among all the samples. Second, for each sample, ReliefF finds its nearest neighbor with the same phenotype (called nearest hit) and the nearest neighbor with different phenotype (called nearest miss). Third, for each SNP, its value in the selected sample is compared with the nearest hit and nearest miss. The weight of each SNP will be updated based on the comparison result.

Meanwhile, many parallel and distributed studies have been reported to accelerate the SNP selection algorithms [14]. Some of them are multi-core CPU based architecture [15], and some of them are GPU based architecture [16][17]. Although there are many distributed machine learning packages, such as the TensorFlow [18] from Google and the DMTK from Microsoft. There are no packages specially designed for SNP selections.

# 4. System Design

The CPU-GPU hybrid distributed framework (HDF) adopts a central control structure. In the proposed HDF, there is a master, called the Controller, which controls the work flow of the whole system. The Controller decomposes the computing mission, uploaded by a user, into many small tasks and distributes the small tasks to the CPU and GPU clients, where lots of CPU clients and GPU clients can be activated in this HDF. Those clients receive tasks from the Controller and return the computed results back. The architecture of the HDF is illustrated in Fig. 1.

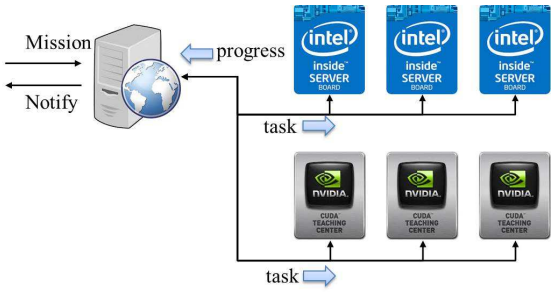


Fig. 1: The architecture of the CPU-GPU hybrid distributed framework.

The CPU-GPU hybrid distributed framework (HDF) adopts a central control structure. In the proposed HDF, there is a master, called the Controller, which controls the work flow of the whole system. The Controller decomposes the computing mission, uploaded by a user, into many small tasks and distributes the small tasks to the CPU and GPU clients, where lots of CPU and GPU clients can be activated in this HDF. Those clients receive tasks from the Controller and return the computed results back. The architecture of the HDF is illustrated in Fig. 1.

## 4.1 Two levels acceleration

Our HDF uses two levels of acceleration to speed up the computing process. The first level is in the Controller-clients cooperation. The clients are independent from each other and they can work concurrently. By decomposing the mission and distributing the tasks in the Controller, and executing the tasks concurrently in the clients, the HDF achieves the first level of acceleration. The second level of acceleration is in the clients. CPU clients can use the multi-thread scheme to further speedup the execution. And the advantage in GPU clients lies in vector operations, which can greatly accelerate the process of SNP selections.

### 4.1.1 Acceleration by distributing

For most SNP selection algorithms, the evaluation of an individual SNP, or a set of SNPs, is independent from other SNPs, or other sets of SNPs. This enables the clients to work concurrently. So the Controller maintains two lists. The first list is all clients that are in connection with the Controller, and the second list is all small tasks which are decomposed by the mission. Each task keeps a record of its destination client. When there is a progress report from a client, the Controller searches this task from the task list

and merges this progress with the progresses obtained so far. The Controller can distribute small tasks to different clients one by one or distribute randomly to make the clients have balanced load. The Controller keeps distributing and merging until all tasks are finished. In addition, if no progress report of a task after distributing is heard for too long time, the HDF will redistribute this task.

#### 4.1.2 Acceleration by multi-threads

After receiving the tasks from the Controller, CPU and GPU clients can further speedup the SNP selection processes. Although they both use the multi-thread scheme, their architectures are different from each other.

**CPU Clients** When receiving a task from the Controller, each CPU client divides it into smaller ones and assigns a thread from its thread pool to handle the smaller tasks. Previous studies show that the performance of CPU multi-thread parallelization is highly related to CPU thread scheduling, and there is little increase when there are more threads than there are more cores in CPU [19]. (This is validated by our experiments in Section 7). In order to achieve an optimal trade-off between the acceleration and CPU thread switch cost, the CPU clients keep a thread pool with the number of threads a little bit more than the number of physical cores in the CPU.

**GPU Client** The hardware structure of GPU is different from that of CPU. GPU has many pipelines which can process multiple data under one instruction (SIMD). When doing vector operations, these pipelines can process different elements in the vectors simultaneously. This makes GPU much faster than CPU in many scientific computations involving vectors and matrices. To utilize the advantages of GPU, researchers need to transform the task data into matrices with rows standing for samples and columns representing SNPs, and rewrite the SNP selection algorithms using the computing API of GPU, such as CUDA.

### 4.2 Scheduling

The main challenge for the distributed system is the communication and synchronization among all the machines. In the HDF, the Controller is responsible for data splitting, distributing and the synchronization of all the computing clients. When researchers use this HDF to develop the distributed version of their SNP selection algorithms, they can use the defaulted interfaces of the HDF to handle these scheduling problems, or that they can define their own scheduling methods.

#### 4.2.1 Splitting scheme

One important difference of bioinformatics data from data of other fields is that there are extreme high number of features ($10^6$) but relatively low number of samples($10^3$). To split this imbalanced data, we provide four defaulted interfaces, which are features oriented and samples oriented splitting, with or without overlapping. Users can use any one of them, or define their own splitting methods based on their algorithm.

There are some cases that the users want to define their own splitting methods. For example, the processing power of CPU and GPU clients are imbalanced, and users may want to distribute them with different kinds of tasks. Note that some algorithms may need a special method for splitting and scheduling. For instance, VLSReliefF [12] samples features with replacement.

#### 4.2.2 Distributing scheme

Since the acceleration comparisons of GPU and CPU vary on different algorithms, our HDF provides several distributing interfaces to optimize the performance. It can distribute tasks to CPU or GPU clients in priority, where it will not consider the other type of clients unless all the same types clients have been used. It can just distribute to GPU clients only or CPU clients only. It can also choose any of the two types of clients randomly. The interfaces are also provided for users to define their own methods of distributing.

#### 4.2.3 Synchronization

By defining three states for the tasks, *Unprocessed*, *Waiting* (sent but without progress reported) and *Finished*, the Controller synchronize the HDF in the following steps. First, when a task is created, its state is set to be *Unprocessed*, and it is assigned with a unique id to identify from other tasks. Second, when a task is sent to a client, the task will remember the ip address of the client and update its state to be *Waiting*. Third, when the Controller receives a progress report from a client, which contains the id of the corresponding task and the ip address of the client, the Controller will check whether it is a legal progress (by comparing the task id and the ip address). If legal, the Controller merges this progress result with others' (by task id), and updates the task state to be *Finished*, and then find an *Unprocessed* task to distribute. For tasks which we haven't received their progress reports for a long time, the Controller will assume there is something wrong and will redistribute the task. When all tasks in the list have be updated to *Finished*, the Controller will notify the result to the users.

## 5. Implementation

As illustrated in Fig. 2, the proposed HDF is divided into three layers, the network layer, the middle layer, and the algorithm layer. Each layer focuses on its own target and provides service to its upper layer.

### 5.1 Network layer

Network layer handles the packages transmitting and receiving between the Controller and clients, and focus

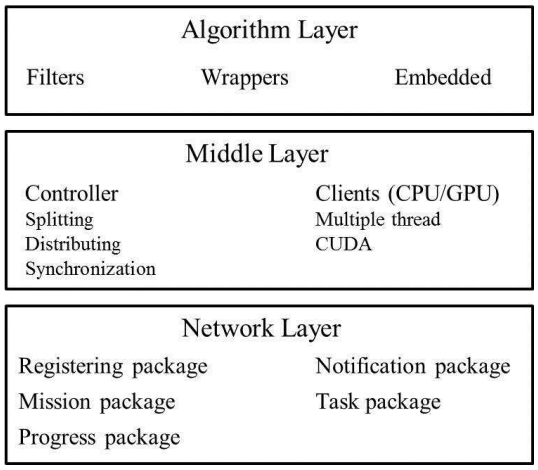


Fig. 2: Three-layer structure

on providing stable communication services to its upper layer, the middle layer. The following kinds of packages are defined in the network layer.

- **Register package**, which is sent from the client to the Controller, and contains the IP address and the hostname of the client. The Controller will add the socket ID to the package after receiving it from the client.
- **Mission package**, which is sent from the user to the Controller, and contains the location of the input file and the name of the algorithm.
- **Task package**, which is sent from the Controller to the client, and contains the location of the input file, the name of the algorithm, the task ID, the mission ID that the task belongs to, and a hash table that contains a subset of SNPs in the input file. The task package contains the file path instead of the content of the input file to reduce the communication overhead for the Controller.
- **Progress package**, which is sent from the client to the Controller, and contains the task ID, the mission ID that the task belongs to, and an array of SNP scores. This array of SNP scores is associated with the hash table in the task package, and the Controller will merge it with score arrays of other progress packages.
- **Notification package**, which is sent from the Controller to the user, and contains the location of the result file.

These packages are structured information. The network layer provides methods to serialize them into streams and to deserialize them from streams for transmitting and receiving.

## 5.2 Middle layer

The work flow of the proposed HDF is based on the event driven model. The Controller and clients start to work after some packages arrived. For the Controller, it may receive the following three packages:

**a) Registration from clients:** After receiving register package from the client, the Controller checks whether this client is on the clients list based on the ip address. If it is a new client, its IP address will be added into the clients list, which are maintained by the Controller. If not, the client's information will be updated.

**b) Mission from user:** After receiving mission package from the user, the Controller looks for the algorithm specified in the mission task, and then splits and distributes the data using the interfaces specified by the user. Different interfaces are provided to handle the splitting and distributing processes.

**c) Progress from clients:** The progress package's response from the Controller has been specified in Section 4.2.3.

For clients, it can only receive the task package from the Controller.

**d) Task from the Controller:** After receiving task package from the Controller, the client reads the data file from the file path provided by the task package. Then the indicated algorithm is called to handle the subset of data specified in the hash table stored in the task package. When the task is completed and a progress report to the Controller is generated, the client will copy the task ID of the task package to help the Controller synchronize.

## 5.3 Algorithm layer

Enabled by the network layer and middle layer, all kinds of SNP selection algorithms (described in Section 3) that evaluate a subset of SNPs independently from the rest of SNPs can be extended to distributed versions using the proposed HDF. An algorithm manager is also implemented in the proposed HDF to manage the algorithms when multiple SNP selection algorithms are adopted into the HDF.

# 6. Experimental Designs

## 6.1 ReleifF algorithm

According to the proposed system design in Section 4, the HDF has been implemented using C++ 11. Also, the ReliefF algorithm has been integrated into the HDF in order to compare its performance with that of a single PC. The pseudo-code of the ReliefF algorithm is illustrated in Fig. 3.

## 6.2 Hardware configurations

In the following experiments, 16 CPU clients (Debian 6.06 workstations with Intel Xeon E5 processors) are adopted. Among these 16 CPU clients, client 1 to client 4 are single-core machines; client 5 to client 8 are 4-core machines; client 9 to client 12 are 5-core machines; and client 13 to client 16 are 16-core machines. 6 GPU clients that equipped with

*Input:* for each training instance a vector of attribute values and the class value
*Output:* the vector W of estimations of the qualities of attributes

1. set all weights $W[A] := 0.0$;
2. **for** i := 1 **to** $m$ **do begin**
3. randomly select an instance $R_i$;
4. find nearest hit $H$ and nearest miss $M$;
5. **for** $A := 1$ **to** $a$ **do**
6. $W[A] := W[A] - \text{diff}(A, R_i, H)/m + \text{diff}(A, R_i, M)/m$;
7. **end**;

Fig. 3: Pseudocode of ReliefF algorithm.

| | Group 1 | Group 2 |
|---|---|---|
| Heritibiltiy (folders) | 0.1, 0.2, 0.3, 0.4 | 0.1, 0.2, 0.3, 0.4 |
| Dataset in each folder | 100 | 100 |
| Samples in each dataset | 1000 (500:500) | 1000 (500:500) |
| Features(SNPs) in each dataset | 1000 | 10000 |
| Total Size | 700MB | 9.8 GB |

Table 2: Details for the synthetic data.

NVIDIA Kepler (GK110GL) display board are also adopted. The memory capacity of each one of these 22 machines is larger than 100 GB.

### 6.3 Data sources

**a) Synthetic data:** The synthetic data used in the experiments are generated by GAMETES[20], which is a specified tool with high accuracy GWAS data generation ability. Two groups of synthetic data with different sizes are generated. The first group contains datasets with 1k features, and the second group contains datasets with 10k features. For each group, there are 4 folders representing different heritability (0.1, 0.2, 0.3, and 0.4). Each of these folders contains 100 files in it and each file is an independent dataset containing 1000 samples (500 positive samples and 500 negative samples). Detailed information of the synthetic data is listed in Table 2

**b) Real data:** A real dataset *phs000019v1* for psoriasis study from the Genotypes and Phenotypes datebase(dbGaP) is used in the experiments. This dataset contains 1659 samples, in which there are 950 cases, and 709 controls. For each sample, there are 448955 SNPs. The size of the *phs000019v1* is 2.1GB.

In the experiments, we design 3 tasks for the proposed HDF. task No.1 is to run the ReliefF algorithm on the synthetic dataset with 1k SNps. task No.2 is to run the ReliefF algorithm on the synthetic dataset with 10k SNPs. And task No.3 is to run the ReliefF algorithm on the real dataset. For each task, the speedup achieved by the proposed HDF, compared to one single-thread PC, is recorded.

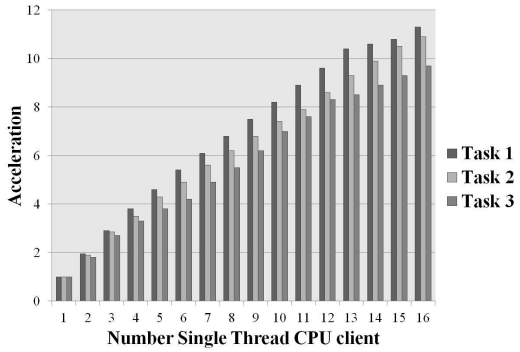


Fig. 4: Acceleration achieved by multiple CPU clients.

## 7. Results

Previous experiments have indicated that single-thread CPU client can accomplish tasks No.1-3 in 10825s (3.1h), 107982s (32h), and 29810s(8.3h) respectively. In the following experiments, these three results will be considered as benchmarks and all the accelerations are calculated with respect to them.

In the following experiments, the two levels of acceleration techniques of the HDF will be tested, and the results illustrate that significant acceleration rate can be achieved for both levels.

### 7.1 First level acceleration

**a) Acceleration of multiple CPU clients:** In this experiment, the number of CPU clients is increasing from 1 to 16, where each client has only one single-thread. The result is shown in Fig. 4.

In Fig. 4, the horizontal axis is the number of adopted CPU clients, and the vertical axis is the acceleration achieved compared to the benchmarks described in Section 7.1. A linear relationship with coefficient $< 1$ between the number of clients and the acceleration can be obtained in this figure. When the number of clients adopted is small ($< 5$), the linear coefficient is above $0.9$. It drops to around $0.7$ when the number of clients adopted increases. The reason that task No.1 achieves better acceleration than tasks No.2-3 is that the size of task No.1's dataset is much smaller than that of the other two. Such a small size dataset makes task No.1 suffers less from the communication overhead. The accelerations achieved by the task No.3 is the smallest because the dataset is stored in one single file, while the datasets of tasks No.1-2 are stored in many small files. Hence, when performing tasks No.1-2, different clients read different small size files simultaneously. However, when performing task No.3, different clients read the large size single file independently. Such a read operation dramatically increases the reading time of the hard disk, and further encumbers the acceleration.

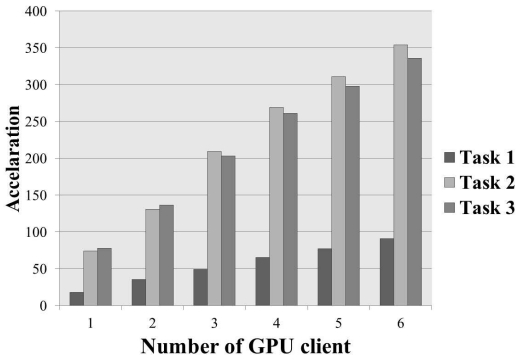


Fig. 5: Acceleration achieved by multiple GPU clients.

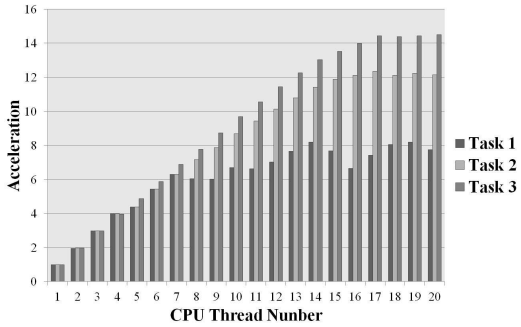


Fig. 6: Acceleration achieved by CPU multi-thread processing

**b) Acceleration of multiple GPU clients:** In this experiment, the number of GPU clients is increasing from 1 to 6. The result is shown in Fig. 5.

In Fig. 5, the horizontal axis is the number of GPU clients adopted, and the vertical axis is the achieved acceleration rate, compared to the benchmarks described in Section 7.1. The speedup by GPU is much better than that by CPU. The performance differences among tasks No.1-3 are mainly caused by the interchanging property of GPU computing states. There are two computing states inside one GPU computation process, the working state and the sleeping state. Whenever a GPU operation arrives, the computing state changes from the sleeping state to the working state, and changes back to the sleeping state after accomplishing the tasks. A specific period of time is needed for interchanging states. For task No.1, frequent interchanging of states occurred due to the small size of the dataset files. For tasks No.2-3, the overhead on interchanging states is much smaller than that of task No.1 because the size of the dataset files is larger. The difference between tasks No.2-3 is caused by how the dataset is stored (multiple small-size files or single large-size file).

## 7.2 Second level acceleration

The performance of the second level acceleration is tested by conducting experiments on a single CPU client, and a single GPU client respectively.

**a) Acceleration of CPU multi-thread processing:** In this experiment, there are 16 physical cores inside the CPU client. The number of threads is increasing from 1 to 20, and the result is shown in Fig. 6

In Fig 6, all the three tasks can be speeded up by the multi-thread technique. However, scheduling the threads takes time. When the size of the processed dataset by each thread decreases, the thread scheduling overhead becomes important. For task No.1, there are only 1000 features for each sample. When the number of threads increases, each thread will process about one hundred SNPs. The time spent on scheduling threads is then more significant than the time spent on processing the SNPs. For tasks No.2-3, the increasing tendency of acceleration stops when the number of threads exceeds the number of physical cores inside the CPU. Performance of task No.3 is typically better than that of task No.2 because the size of dataset processed by each thread is larger and the time spent on thread scheduling is relatively insignificant.

**b) Acceleration of GPU parallel processing:** GPU parallel processing is a little bit more complex than CPU multi-threading. In CUDA API, each GPU process contains several blocks, and each block contains many threads. Despite the accessible global memory for all threads, each block also has its own faster memory. Users can modify the memory distribution by adjusting the ratio between the number of blocks and the number of threads in each block. In the experiment, the number of threads in each block is increasing from 1 to 1000. The results illustrate constant acceleration for all the three tasks, which indicates that there may be some optimization inside CUDA.

## 7.3 Acceleration of the proposed CPU-GPU hybrid distributed framework

In this subsection, the CPU clients and GPU clients have been integrated into one hybrid system called CPU-GPU hybrid distributed framework (CPU-GPU HDF) and its performance on SNP selections has also been tested. For the CPU clients, 20 threads are created in each client. For the GPU clients, the number of blocks in GPU doesn't matter as discussed in Section 7.2. The results are shown in Fig. 7.

In Fig. 7, the acceleration increases when the number of CPU and GPU clients increase. The acceleration power of CPU clients is almost linear to the number of physical cores in it, and the acceleration power of GPU client is algorithm dependent. For ReliefF algorithm, GPU client can achieve 30 to 70 times speedup compared to benchmarks. When there are 16 multi-thread CPU clients and 6 GPU clients, the acceleration for task No.1-3 are 151, 423, 419 respectively.

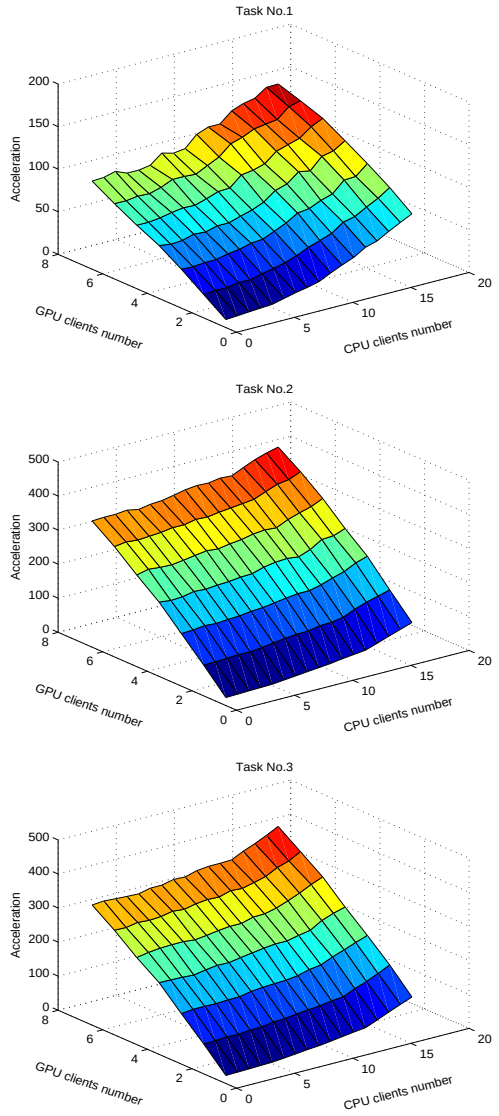


Fig. 7: Acceleration achieved by the HDF.

# 8. Conclusion

SNP selections is critical in genome wide association study (GWASs). However, SNP selections is really time consuming, so the acceleration on performance is essential. In this paper, a CPU-GPU hybrid distributed framework specifically for SNP selection algorithm is proposed, designed, implemented, and tested. The HDF provides interfaces for data splitting and communication synchronization. It can support many different SNP selection algorithms using the interfaces provided. In the experiments, the algorithm of ReliefF SNP selection has been integrated into the HDF to test its performance on different datasets. The experiments indicate that the proposed HDF is able to achieve significant acceleration on different datasets.

# References


[1] P. M. Visscher, M. A. Brown, M. I. McCarthy, and J. Yang, "Five years of gwas discovery," *The American Journal of Human Genetics*, vol. 90, no. 1, pp. 7–24, 2012.

[2] M. A. Rivas, M. Beaudoin, A. Gardet, C. Stevens, Y. Sharma, C. K. Zhang, G. Boucher, S. Ripke, D. Ellinghaus, N. Burtt *et al.*, "Deep resequencing of gwas loci identifies independent rare variants associated with inflammatory bowel disease," *Nature genetics*, vol. 43, no. 11, pp. 1066–1073, 2011.

[3] D. L. Nicolae, E. Gamazon, W. Zhang, S. Duan, M. E. Dolan, and N. J. Cox, "Trait-associated snps are more likely to be eqtls: annotation to enhance discovery from gwas," *PLoS Genet*, vol. 6, no. 4, p. e1000888, 2010.

[4] J. Bedő, D. Rawlinson, B. Goudey, and C. S. Ong, "Stability of bivariate gwas biomarker detection," *PloS one*, vol. 9, no. 4, p. e93319, 2014.

[5] A. Dahl, V. Iotchkova, A. Baud, Å. Johansson, U. Gyllensten, N. Soranzo, R. Mott, A. Kranis, and J. Marchini, "A multiple-phenotype imputation method for genetic studies," *Nature genetics*, 2016.

[6] M. Dash and H. Liu, "Feature selection for classification," *Intelligent data analysis*, vol. 1, no. 3, pp. 131–156, 1997.

[7] ——, "Feature selection for clustering," in *Knowledge Discovery and Data Mining. Current Issues and New Applications*. Springer, 2000, pp. 110–121.

[8] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014.

[9] J. H. Moore and B. C. White, "Tuning relieff for genome-wide genetic analysis," in *Evolutionary computation, machine learning and data mining in bioinformatics*. Springer, 2007, pp. 166–175.

[10] K.-Y. Lee, P. Liu, K.-S. Leung, and M.-H. Wong, "Very large scale relieff algorithm on gpu for genome-wide association study," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2015, p. 78.

[11] I. Kononenko, E. Šimec, and M. Robnik-Šikonja, "Overcoming the myopia of inductive learning algorithms with relieff," *Applied Intelligence*, vol. 7, no. 1, pp. 39–55, 1997.

[12] M. J. Eppstein and P. Haake, "Very large scale relieff for genome-wide association analysis," in *Computational Intelligence in Bioinformatics and Computational Biology, 2008. CIBCB'08. IEEE Symposium on*. IEEE, 2008, pp. 112–119.

[13] Y. Sun, "Iterative relief for feature weighting: algorithms, theories, and applications," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 6, pp. 1035–1051, 2007.

[14] M. Cadzow, J. Boocock, H. T. Nguyen, P. Wilcox, T. R. Merriman, and M. A. Black, "A bioinformatics workflow for detecting signatures of selection in genomic data," *Frontiers in genetics*, vol. 5, 2014.

[15] X. Zheng, D. Levine, J. Shen, S. M. Gogarten, C. Laurie, and B. S. Weir, "A high-performance computing toolset for relatedness and principal component analysis of snp data," *Bioinformatics*, vol. 28, no. 24, pp. 3326–3328, 2012.

[16] X. Hu, Q. Liu, Z. Zhang, Z. Li, S. Wang, L. He, and Y. Shi, "Shesisepi, a gpu-enhanced genome-wide snp-snp interaction scanning algorithm, efficiently reveals the risk genetic epistasis in bipolar disorder," *Cell research*, vol. 20, no. 7, pp. 854–857, 2010.

[17] L. S. Yung, C. Yang, X. Wan, and W. Yu, "Gboost: a gpu-based tool for detecting gene–gene interactions in genome–wide case control studies," *Bioinformatics*, vol. 27, no. 9, pp. 1309–1310, 2011.

[18] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous systems, 2015," *Software available from tensorflow. org*.

[19] C. Li, C. Ding, and K. Shen, "Quantifying the cost of context switch," in *Proceedings of the 2007 workshop on Experimental computer science*. ACM, 2007, p. 2.

[20] R. J. Urbanowicz, J. Kiralis, N. A. Sinnott-Armstrong, T. Heberling, J. M. Fisher, and J. H. Moore, "Gametes: a fast, direct algorithm for generating pure, strict, epistatic models with random architectures," *BioData mining*, vol. 5, no. 1, p. 1, 2012.