Supplemental Data

GBOOST: A GPU-based tool for detecting gene-gene interactions in genome-wide case-control studies

Ling Sing Yung, Can Yang, Xiang Wan, and Weichuan Yu

In this supplementary document, we first describe some details of the Graphical Processing Unit (GPU) implementation of Boolean operation based screening and testing (BOOST) [1]. Then, we illustrate the performance improvement of GBOOST and some visualization functions.

1 Recommended hardware, platform and software version for GBOOST

CPU : Intel CPU 2.13GHz or above

Main Memory : 2 GB or above

Display Card : Nvidia GTX 285

Operating System : Windows Vista x64

CUDA Driver : 2.3

Please refer to http://www.nvidia.com/object/cuda_gpus.html for a list of CUDA Enabled GPUs.

2 GPU Implementation of BOOST

GBOOST is a C++ parallel implementation of the BOOST method [1] using Compute Unified Device Architecture (CUDA) runtime application programming interface (API) [2]. Threads, blocks and grids are basic components of CUDA architecture. A thread is the smallest working unit in CUDA. A computation problem can be divided into sub-tasks and every GPU core is responsible for one or multiple sub-tasks of the original problem. A block is formed by multiple threads. Grids are a collection of thread blocks. They represent all tasks of the original problem. The GPU implementation of BOOST is divided into two main stages: problem partition and memory optimization.

2.1 Problem Partition

BOOST collects contingency tables of all single-nucleotide polymorphism (SNP) pairs. We use the contingency tables to compute the likelihood ratio statistic in the screening stage. A predefined threshold is employed to filter out non-significant interactions. The survivors are then scored by a classical likelihood ratio test in the testing stage. For any SNP pair(SNP_i, SNP_j), the contingency table is the same as that for the pair(SNP_j, SNP_i). Thus, half of the computation can be saved if we go through all SNP pairs following a triangle pattern, as shown in figure S1.

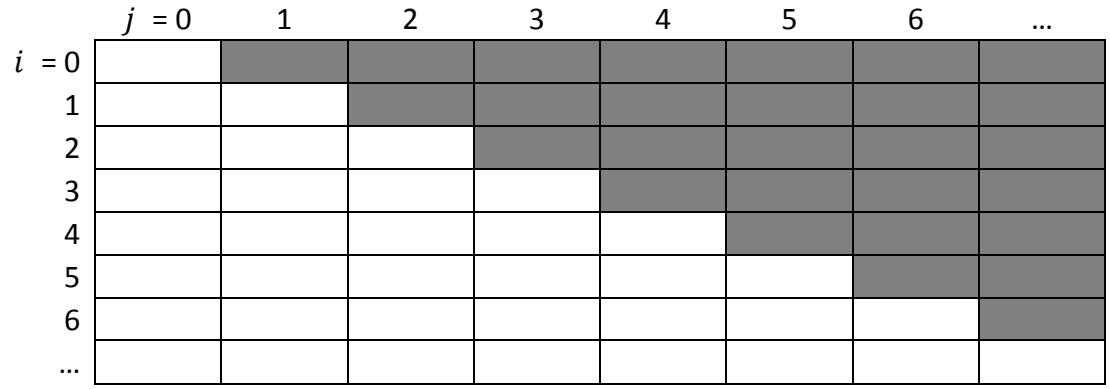


Figure S1: Problem domain of the pairwise gene-gene interaction analysis. Each block represents a SNP pair in the gene-gene interaction analysis and the shaded blocks are SNP pairs required to examine. Here i and j indicates the SNP indices of a specified SNP pair.

To achieve load balancing and effectively utilize the computation resources, GBOOST divides all SNP pairs (the shaded blocks in figure S1) into different subsets. Figure S2 gives an example of the partition scheme for a dataset with 8 SNPs. A larger number of threads running concurrently will lead to a better performance of the GPU program, but one should note that the maximum number of concurrent threads is limited by the registers used per thread. In GBOOST, most registers are used to temporally store contingency tables. Considering that the memory requirement for storing contingency tables of all SNP pairs from a human genome dataset far exceeds the capacity of the best GPU available in the market and that the number of thread blocks per grid is limited by the GPU hardware, we decided to compute partial results and output significant SNP pairs step by step.

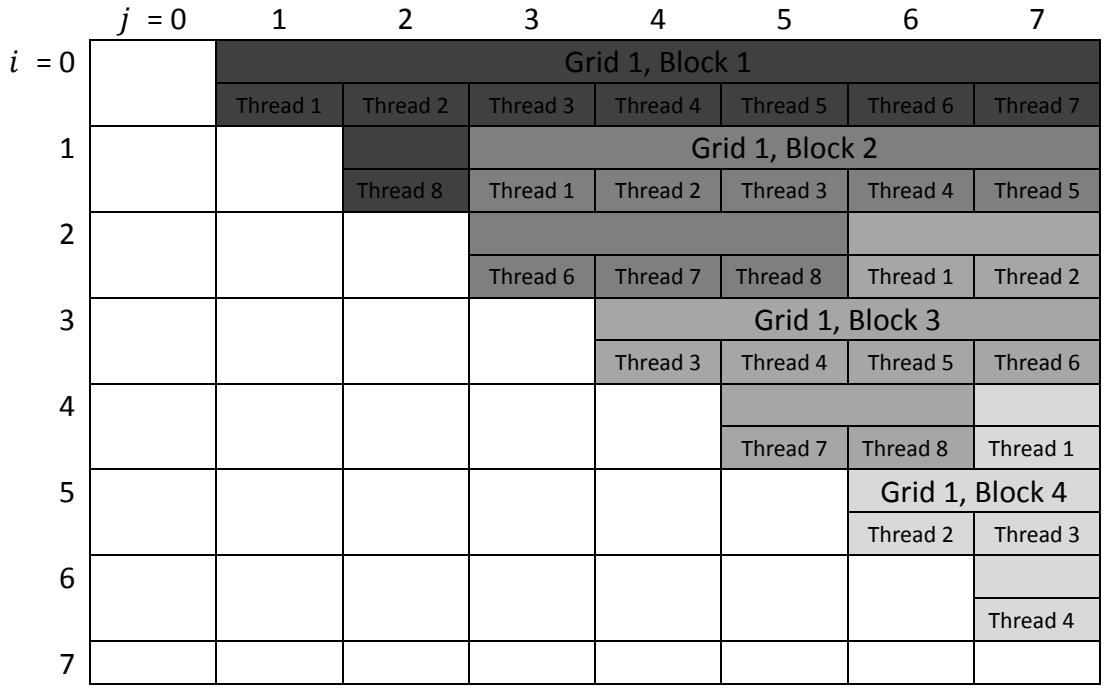


Figure S2: A toy example of the partition scheme in GBOOST. Here i and j indicates the SNP indices of SNP pairs. The grid size is 1, the block size is 8 and the thread size is 8. The thread size can be any number smaller than or equal to the block size. If the thread size exceeds the block size, some threads will be idle at all time and will waste valuable computer resources.

2.2 Memory optimization

High memory bandwidth is one of the key advantages of GPU. While CPUs only have cache and Random Access Memory (RAM), GPUs have more types of computer memory with different speed and size. For Nivida display card with CUDA support, there are four different types of computer memory: global memory, texture memory, shared memory and constant memory. Optimizing the use of memory becomes the key issue to achieve a high speed-up. Global memory is the largest memory available in GPU. It supports coalesced memory access. A coalesced memory access completes memory operations of multiple threads in a single transaction. Perfectly aligned memory blocks speed up memory operations effectively.

Normally, the genome-wide SNP data is represented as a $p \times n$ matrix with p denoting the number of SNPs and n the number of samples. Each element in this huge matrix will have a value of 0, 1 or 2, corresponding to the genotype AA, Aa, aa, respectively. In BOOST, we use Type index to indicate these three genotypes, i.e. {Type 0, Type 1, Type 2} correspond to {AA, Aa, aa}. BOOST also compactly stores the genotypes of every 64 samples in three bit strings (please refer to [1] for more details). Figure S3 uses a toy example to illustrate the conversion of the $p \times n$ matrix to the bit representation used in BOOST.

$$M = \begin{array}{c|cccccccccccccccc} & C_1 & C_2 & C_3 & C_4 & C_5 & C_6 & C_7 & C_8 & D_1 & D_2 & D_3 & D_4 & D_5 & D_6 & D_7 & D_8 \\ X_1 & 0 & 2 & 1 & 1 & \underline{\mathbf{2}} & \underline{\mathbf{2}} & \underline{\mathbf{0}} & \underline{\mathbf{0}} & 1 & 1 & 2 & 2 & 0 & 1 & 0 & 0 \\ X_2 & 2 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 2 & 2 & 0 & 0 & 0 & 1 & 0 & 1 \end{array}$$

(a) Input matrix M

$$M_{bit} = \begin{array}{c|cc} & \text{Control} & \text{Disease} \\ X_1 = 0 & 10000\underline{\mathbf{11}} & 00001011 \\ X_1 = 1 & 00110000 & 11000100 \\ X_1 = 2 & 0100\underline{\mathbf{11}}00 & 00110000 \\ X_2 = 0 & 00001110 & 00111010 \\ X_2 = 1 & 01110001 & 00000101 \\ X_2 = 2 & 10000000 & 11000000 \end{array}$$

(b) Bit representation of M

Figure S3: Matrix conversion in BOOST. The input matrix is a 2×16 matrix. Each sample has a genotype label {0,1,2} corresponding to {AA,Aa,aa}. C_x and D_x denote control samples and disease sample, respectively. After performing type conversion, the eight control samples of X_1 $[\mathbf{02112200}]$ have a new representation of three bit strings (i.e. $X_1 = \mathbf{0}: [\mathbf{10000011}]$, $X_1 = \mathbf{1}: [\mathbf{00110000}]$, and $X_1 = \mathbf{2}: [\mathbf{01001100}]$).

The data structure in BOOST follows the input order of SNP index, Type index and Sample index, as shown in figure S4. However, this structure is bad for GPU memory access and leads to a poor performance in collecting contingency tables in GPU kernels. To accelerate the collection of contingency tables, the bit representation of data has been restructured to maximize the number of coalesced memory access. As SNP pairs follow a triangle pattern, a consecutive thread will work on a consecutive SNP. The bit representation is therefore restructured to follow the computational order of the SNP pairs in GBOOST with the data structure having an order of Type index, Sample index, and SNP index, as shown in figure S5. This converts most memory operations into coalesced memory access and reduces the total number of memory transactions significantly.

The texture memory is a cached read-only memory. We use it to store the pre-computed bin count for counting the number of 1's in a bit string. Shared memory has similar performance as register and is shared between threads in the same block. In GBOOST, computations of different SNP pairs are independent. Shared memory is therefore used with constant memory and serves as temporal storage to reduce register usage. Thus, more threads can run concurrently in a block as fewer registers are used per thread.

Address 1	Address 2	Address 3	Address 4	Address 5	Address 6
SNP 1	SNP 1	SNP 1	SNP 1	SNP 1	SNP 1
Type 0	Type 0	Type 1	Type 1	Type 2	Type 2
Sample 0-63	Sample 64-127	Sample 0-63	Sample 64-127	Sample 0-63	Sample 64-127
Address 7	**Address 8**	**Address 9**	**Address 10**	**Address 11**	**Address 12**
SNP 2	SNP 2	SNP 2	SNP 2	SNP 2	SNP 2
Type 0	Type 0	Type 1	Type 1	Type 2	Type 2
Sample 0-63	Sample 64-127	Sample 0-63	Sample 64-127	Sample 0-63	Sample 64-127
Address 13	**Address 14**	**Address 15**	**Address 16**	**Address 17**	**Address 18**
SNP 3	SNP 3	SNP 3	SNP 3	SNP 3	SNP 3
Type 0	Type 0	Type 1	Type 1	Type 2	Type 2
Sample 0-63	Sample 64-127	Sample 0-63	Sample 64-127	Sample 0-63	Sample 64-127

Figure S4: Structure of input data in BOOST. There are 128 samples in the figure. BOOST forms a data group for every 64 samples and stores the group information in a 64-bit integer. The 64-bit integers are then ordered by SNP index, Type index and Sample index in the memory.

Address 1	Address 2	Address 3	Address 4	Address 5	…
SNP 1	SNP 2	SNP 3	SNP 4	SNP 5	
Type 0	Type 0	Type 0	Type 0	Type 0	…
Sample 0-63	Sample 0-63	Sample 0-63	Sample 0-63	Sample 0-63	
Address N+1	**Address N+2**	**Address N+3**	**Address N+4**	**Address N+5**	**…**
SNP 1	SNP 2	SNP 3	SNP 4	SNP 5	
Type 0	Type 0	Type 0	Type 0	Type 0	…
Sample 64-127	Sample 64-127	Sample 64-127	Sample 64-127	Sample 64-127	
Address 2N+1	**Address 2N+2**	**Address 2N+3**	**Address 2N+4**	**Address 2N+5**	**…**
SNP 1	SNP 2	SNP 3	SNP 4	SNP 5	
Type 1	Type 1	Type 1	Type 1	Type 1	…
Sample 0-63	Sample 0-63	Sample 0-63	Sample 0-63	Sample 0-63	

Figure S5: Different structure of input data in GBOOST. Here N is the number of SNPs in the dataset. There are 128 samples in the figure. GBOOST uses the same strategy as BOOST to group every 64 samples into a 64-bit integer, but GBOOST orders the 64-bit integers by Type index, Sample index and SNP index. Thus, consecutive threads have a better chance to group memory access into coalesced memory access to maximize the memory throughput.

3 Performance of GBOOST on different datasets with different thread sizes, block sizes and optimization techniques

In this section, we first present running time of GBOOST on different datasets with different thread sizes and block sizes. Then, we show the work distribution of the datasets. Finally, we compare the performance of GBOOST with different settings and explain the reasons of the performance variations. The reported performance time in this section includes the execution of “Screening”, “Testing” and overhead of the memory transactions between GPU and CPU. We exclude time spent on loading data from the disk to CPU main memory and time spent on saving the computation results back to the disk.

3.1 Performance on different datasets with different thread sizes, block sizes

Figures S6, S7 and S8 show consistence performance of GBOOST under different thread numbers and block sizes. This illustrates that GBOOST is scalable and achieves stable performance for small simulation datasets as well as large genome-wide datasets.

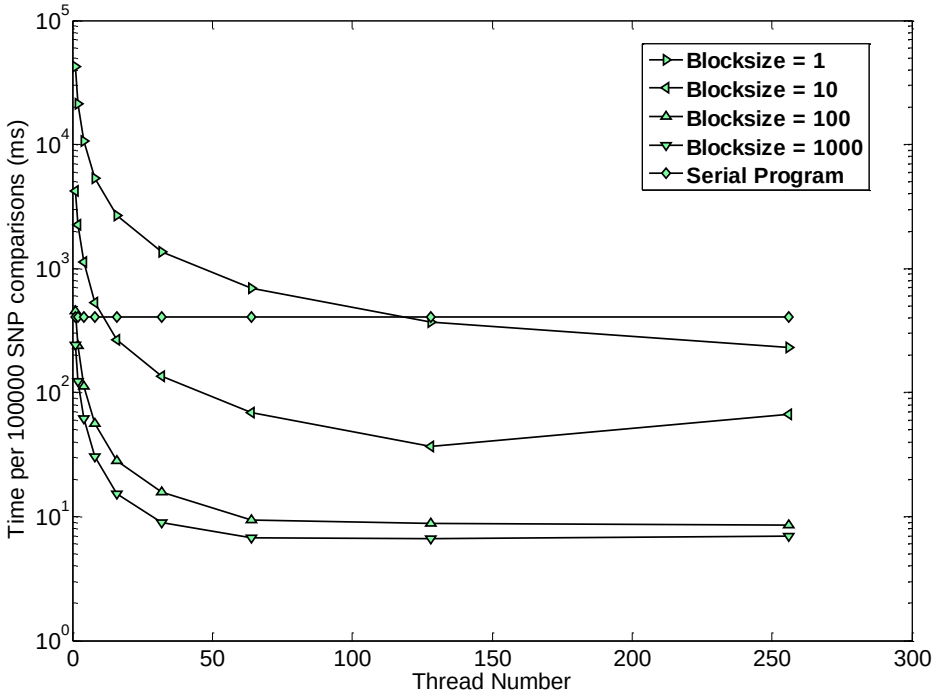


Figure S6: Performance of GBOOST on simulation data (p=1,000, n=5,000) with different thread numbers and block sizes. Here p denotes the number of SNPs and n denotes the sample size.

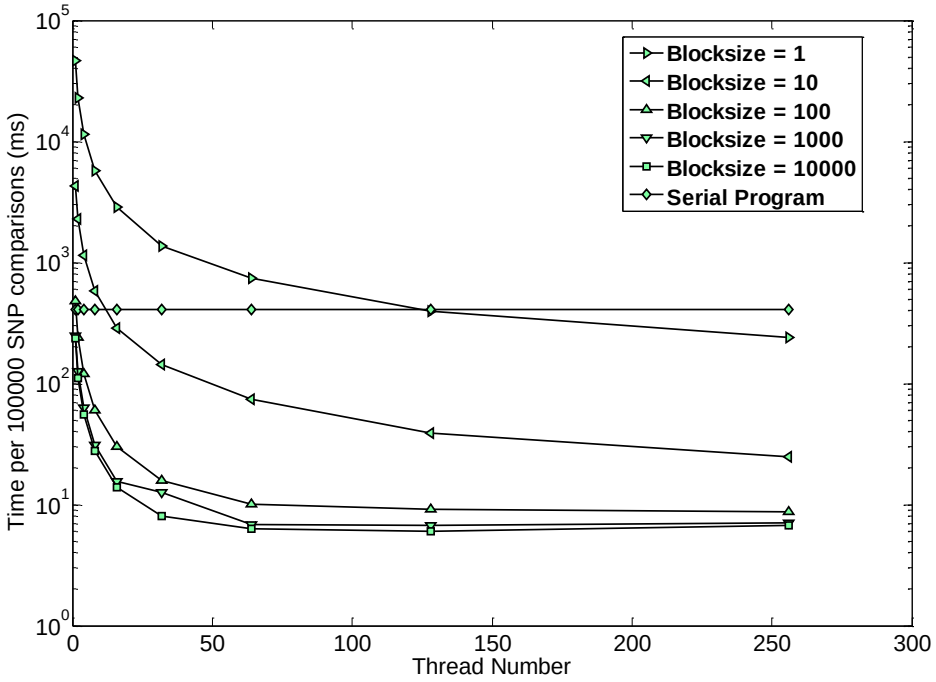


Figure S7: Performance of GBOOST on simulation data (p=10,000, n=5,000) with different thread numbers and block sizes. Here p denotes the number of SNPs and n denotes the sample size.

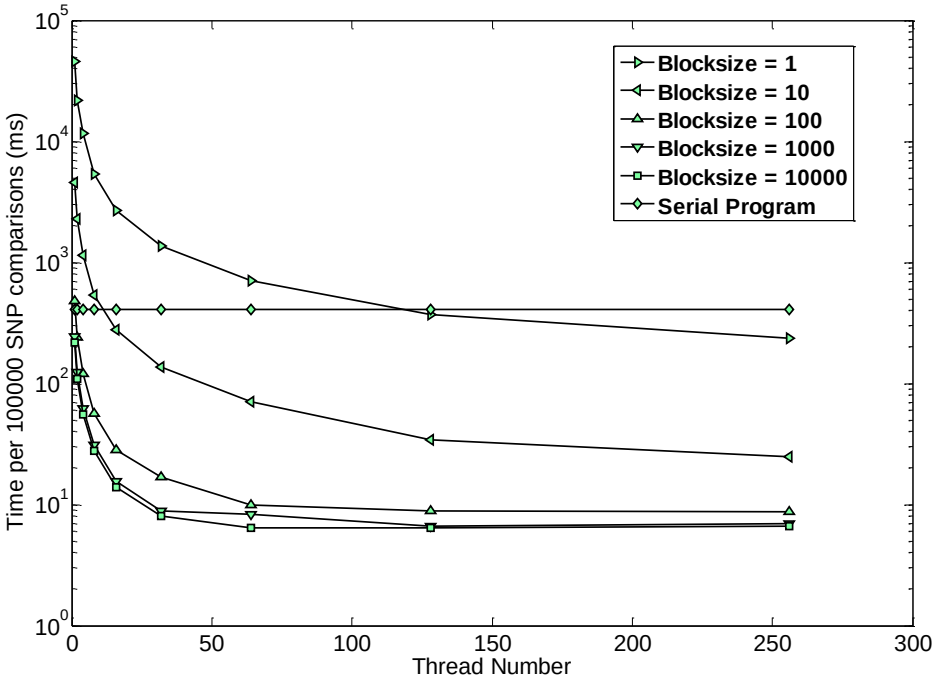
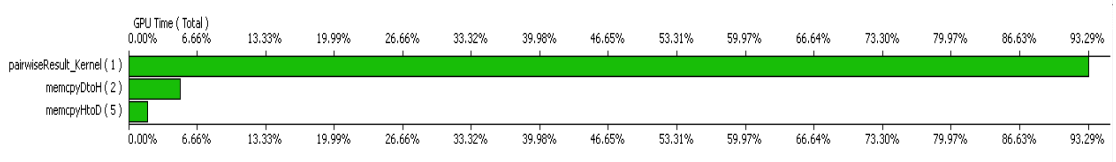
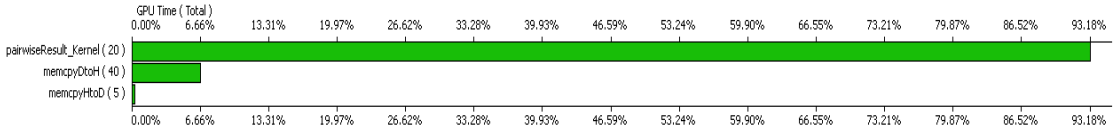


Figure S8: Performance of GBOOST on Wellcome Trust Case Control Consortium Type 1 Diabetes (WTCCC T1D) chromosome 1 data (p=28,121, n=5,004) with different thread numbers and block sizes. Here p denotes the number of SNPs and n denotes the sample size.

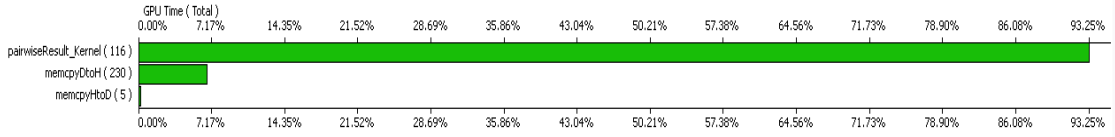
3.2 Work Distribution



(a) Simulation data (p=1,000, n=5,000)



(b) Simulation data (p=10,000, n=5,000)



(c) WTCCC T1D chromosome 1 data (p=28,121, n=5,004)

Figure S9: Work distribution of GBOOST on different datasets. Here p denotes the number of SNPs and n denotes the sample size.

From the experimental results presented in figure S9, we observe that additional overhead of the memory transactions between CPU and GPU is about 7 % of the total execution time in GBOOST (excluding time on loading data from the disk and saving the results to the disk).

3.3 Performance with different optimization techniques

Algorithm	Data Type		Data Structure		Bit string counting algorithms			Register limits		
	32-bit int	64-bit int	Figure S4	Figure S5	Look-up table		Hamming weight	8	16	No Limit
					Constant memory	Texture memory				
Algorithm I		✓	✓		✓					✓
Algorithm II		✓	✓			✓				✓
Algorithm III		✓		✓		✓		✓		
Algorithm IV		✓		✓		✓			✓	
Algorithm V	✓			✓		✓				✓
Algorithm VI		✓		✓		✓				✓
Algorithm VII		✓		✓			✓			✓

Table S1: Algorithms with different data types, data structures, bit string counting algorithms and register limits in GBOOST.

Figure S10 shows the performance of GBOOST with different settings on the WTCCC T1D chromosome 1 data. We can observe a significant gap between the first algorithm and other algorithms. This is because there is a limitation of concurrent threads accessing the constant memory in GPUs. This limitation forces multiple threads accessing the constant memory serially and leads to a significant degradation in performance. Also, the gap between the second algorithm and the last algorithm gives a conclusion that maximizing coalesced global memory operations is the key optimization technique to achieve a high speed-up.

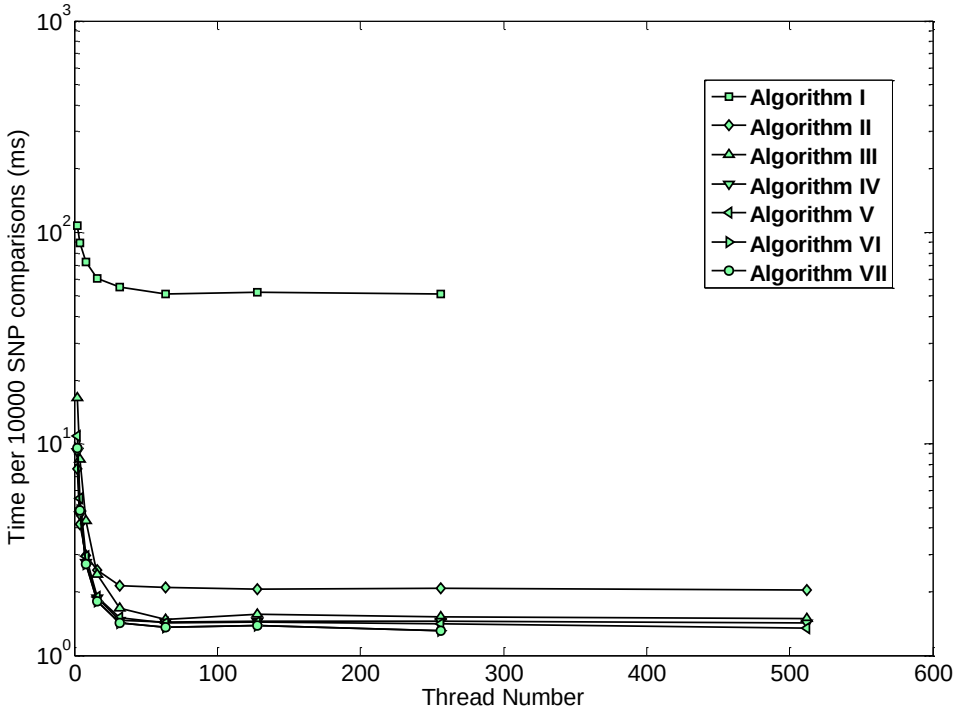


Figure S10: Performance of GBOOST with different settings on Wellcome Trust Case Control Consortium Type 1 Diabetes (WTCCC T1D) chromosome 1 data (p=28,121, n=5,004) with block size 10000. Here p denotes the number of SNPs and n denotes the sample size. For algorithm I, VI, and VII, there are not enough registers to run the setting of 512 threads. Table S1 summaries the differences between the algorithms.

Setting a hard limit on register usage does not have serious effects on the performance of GBOOST. GBOOST has a simple GPU kernel with a low register usage. Thus, registers are not the bottleneck of the current GPU implementation of GBOOST.

4 Visualization Examples

4.1 Pathway Graph

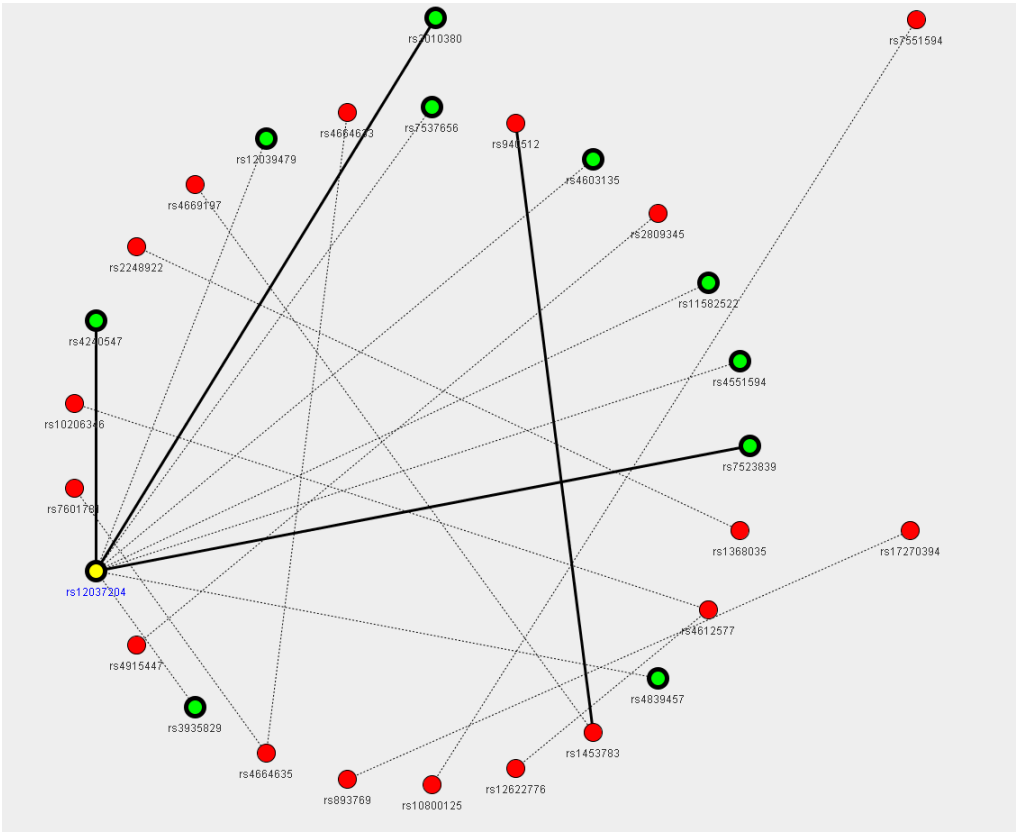


Figure S11: A pathway graph generated using GBOOST GUI output API.

Figure S11 shows the relationship of different SNPs with different pathways. The node in a pathway graph represents a SNP and its score denotes the marginal association. The edge connecting two nodes represents the Chi-Square statistic of the SNP pair. Nodes (SNPs) with the same pathway will be grouped and organized on a circle. In figure S11, there are 4 groups of SNPs. The largest group contains SNPs with unknown pathway, while the other three groups contain SNPs with known pathway and have only one entry. Interfaces are available to selectively show the score, pathway name or SNP name as the node label. The selected node is

highlighted in yellow and its neighbor nodes are highlighted in green. After inputting a threshold, we can highlight edges with scores larger than the threshold value.

4.2 Manhattan Plot

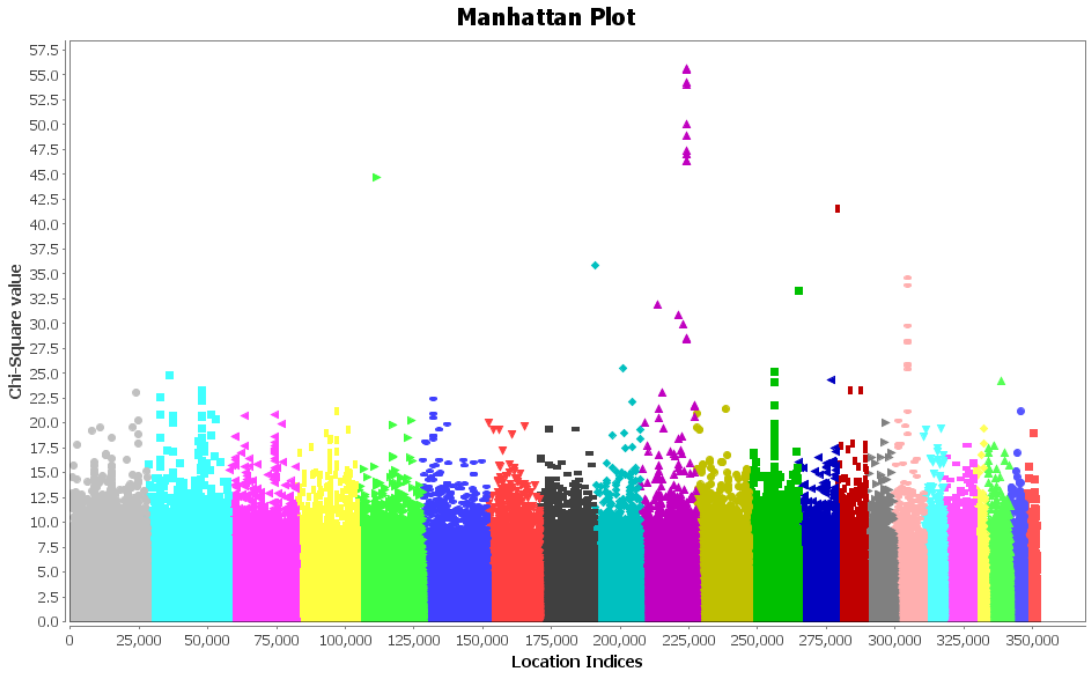


Figure S12: A Manhattan plot generated using GBOOST GUI output API.

A Manhattan plot shows the marginal associations of all SNPs in one dataset. Manhattan plot is a common visualization technique in Genome-Wide Association Studies (GWAS). We implement this feature as a basic visualization tool in GBOOST Graphical User Interface (GUI). SNPs at different chromosomes are indicated by different colors. The x-axis and y-axis represent the location indices of all SNPs and the Chi-Square values, respectively. Figure S12 shows the Manhattan plot of the association analysis result obtained from the Wellcome Trust Case Control Consortium Type 2 Diabetes (WTCCC T2D) dataset.

5 References

[1] X. Wan, C. Yang, Q. Yang, H. Xue, X. Fan, N. Tang and W. Yu, "BOOST: A Boolean Representation-based Method for Detecting SNP-SNP Interactions in Genome-wide Association Studies," *The American Journal of Human Genetics,* vol. 87, pp. 325-340, 2010.

[2] Nvidia Corporation, "NVIDIA CUDA programming guide 2.3," 2009.