Spring 5-31-2013

A GPU program to compute SNP-SNP interactions in genome-wide association studies

Srividya Ramakrishnan
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

A GPU PROGRAM TO COMPUTE SNP-SNP INTERACTIONS IN GENOME-WIDE ASSOCIATION STUDIES

by
Srividya Ramakrishnan

With the recent advances in the next generation sequencing technologies, short read sequences of human genome are made more accessible. Paired end sequencing of short reads is currently the most sensitive method for detecting somatic mutations that arise during tumor development. In this study, a novel approach to optimize the detection of structural variants using a new short read alignment program is presented.

Pairwise interaction effects of the Single Nucleotide Polymorphisms (SNPs) have proven to uncover the underlying complex disease traits. Computing the disease risk based on the interaction effects of SNPs on a case – control study is a difficult problem. As another part of the thesis, a fast GPU program that can calculate the chi-square statistics of SNP-SNP interactions and output the significant interacting SNPs is presented. The algorithm is applied to the datasets of seven common diseases obtained from Wellcome Trust Case Control Consortium (WTCCC). The algorithm computed the significant SNP pairs much faster than the existing algorithms and also identifies 3 significant pairs associated with genes IL23R and C11orf30 which are associated with pathogenesis in the Crohns disease dataset.

A GPU PROGRAM TO COMPUTE SNP-SNP INTERACTIONS IN GENOME-WIDE ASSOCIATION STUDIES

by
Srividya Ramakrishnan

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Bioinformatics

Department of Computer Science

May 2013

APPROVAL PAGE

A GPU PROGRAM TO COMPUTE SNP-SNP INTERACTIONS IN GENOME-WIDE ASSOCIATION STUDIES

Srividya Ramakrishnan

Dr. Usman Roshan, Thesis Advisor Associate Professor of Computer Science, NJIT	Date
Dr. Zhi Wei, Committee Member Assistant Professor of Computer Science, NJIT	Date
Dr. Alexandros V. Gerbessiotis, Committee Member Associate Professor of Computer Science, NJIT	Date

BIOGRAPHICAL SKETCH

Author: Srividya Ramakrishnan

Degree: Master of Science

Date: May 2013

Undergraduate and Graduate Education:

- Master of Science in Bioinformatics,
  New Jersey Institute of Technology, Newark, USA, 2013

- Bachelor of Science in Biotechnology,
  SASTRA University, Thanjavur, India, 2008

Major: Bioinformatics

With great love and honor I dedicate this thesis to

My father and Mother for encouraging and supporting me

To my sisters, who cared and believed in me

To my uncle and aunt, for the enormous affection, blessing and care

To my brother, who stands as an inspiration in my career.

To my grandfathers and grandmothers, who always loved and blessed me

To all my Friends, who supported me in every aspect of life.

I'd thank you all for loving and believing in me.

ACKNOWLEDGMENT

I extend a great Thanks to my thesis advisor Dr. Usman Roshan, for his guidance, support and encouragement through the course of my study at New Jersey Institute of Technology. It is been a great honor to work with him on a challenging research work. I believe my learning's from him would benefit me a growth in my career. I also thank Dr. Zhi Wei, my professor and also Master's thesis committee member for the support and guidance. A Special thanks to Dr. Alexandros V. Gerbessiotis for being part of my Master's thesis committee.

I would like a thank Dr. David Perel, University of Computing Systems. I have been extremely fortunate to work with him and his team Kevin Walsh, Richard Gaine. Working with them provided enormous opportunity to learn on Linux System Administration and resolving issues. They have also been instrumental in completing part of my research on the afs filesystems at NJIT. I would also like to thank University of North Carolina computing systems for providing a sophisticated server enabled for CUDA.

I also thank my friend Vijay Jana who supported me and cared for me all through ever since I have come to United States.

Once Again, I greatly thank you all for helping me in making this research a success.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

GWAS	Genome Wide Association Study
SNP	Single Nucleotide Polymorphism
GPU	Graphical Processing Unit
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
WTCCC	Wellcome Trust Case Control Consortium

CHAPTER 1

STRUCTURAL VARIANT DETECTION PROBLEM

1.1 Introduction

Structurally variants are 5% of the human genome and have more than 800 independent genes that are more likely to contribute to disease susceptibility. The expanded use of next-generation sequencing with "paired end" methods has enabled a whole-genome analysis with essentially unlimited resolution. The discovery of submicroscopic copy number variations (CNVs) present in our genomes has changed dramatically our perspective on DNA structural variation and disease (Stankiewicz et al 2010). It is now thought that CNVs encompass more total nucleotides and arise more frequently than SNPs. Detection of SVs in a human genome using NGS technologies was first presented by Korbel et al. (2007) and later Kidd et al. (2008) detected, experimentally validated and sequenced Structural variants (SVs) from eight different individuals. These validated sites are made publicly available through the Human Genome Structural Variation Project browser. In this thesis, a novel approach to identify structural variants is presented.

1.2 Existing Methods in Structural Variant Discovery

The current methods in structural variant detection use paired-end sequencing: Inserts from a genome are read at both the ends, which are later aligned to reference genome. If the mapping loci are identified correctly, an increase or decrease of the distance between the end reads indicates an insertion or deletion.

The tools which were developed for Structural variant detection like Variation Hunter, PEmer (Korbel et al 2009), Pindel (Ye et al. 2009), BreakDancer (Chen et al. 2009) focus mainly on the best mapping of each read provided by the mapping software in use. In a recent study (Hach et al. 2010), it is been demonstrated that ignoring possible mapping locations of a read may lead to loss of accuracy in Structural variant detection. VariationHunter implements a soft clustering method that aims to resolve repetitive regions of the human genome through a combinatorial optimization framework for detecting insertion and deletion polymorphisms. In this study, a short reads of Next generation Sequencing (NGS) genome of a Yoruba individual (NA18507) has been used.

1.3 Materials and Methods

True Structural variant discovery does not just depend upon the SV detection algorithm used but also largely depend upon the Short read alignment program. This research focuses on comparing the Structural variants identified on the short reads aligned using Stampy and a new short read aligner. Then compare the Structural variants with the known structural variants detected on NA18507 (Kidd et al).

Short reads for NA18507 were obtained from Next generation sequencing (NGS) reads from SOLiD™ for the Yoruban individual were obtained from the 1000 genomes project. A Perl script is used to map SOLiD™ color-space reads to base space. The shorts reads in the base space has to be mapped to the reference genome before being passed to the Structural variant detection program.

Whole genome short reads for NA18507 from SOLiD™ sequencing system has been obtained from 1000 Genomes project website. The dataset have a total number of reads is 1,504,002,272 (approx 1.5 billion) of 51 bp length.

1.4 Results and Discussion

Stampy took approximately 30 days of CPU time to complete the mapping of the whole genome short reads of NA18507 to the human genome completely on phi.njit.edu. But the new short read aligner was not available due to unavoidable circumstances and was still under development. So it was not possible to continue research further on this area.

CHAPTER 2

GWAS STUDY ON SNP INTERACTIONS

2.1 Background on GWAS

Genome-wide association study (GWAS) is an examination of many common genetic variants in different individuals to see if any variant is associated with a trait. GWAS typically focus on associations between single-nucleotide polymorphisms (SNPs) and traits like major diseases. These studies normally compare the DNA of two groups of participants: people with the disease (cases) and similar people without (controls). With the recent diverse range of SNP genotyping methods there are more than 4,000 SNP associations with a variety of diseases have been identified.

2.2 SNP Epitasis and Disease Risk

It may not be enough to conduct a GWAS study of SNPs considering one at a time to understand complex disease traits. Epistasis is the interaction between two genes that can suppress the effect of one gene over the other (Miko, I. 2008). Various studies suggest that the pairwise Epistatic effects on SNPs are causative for complex diseases such as sporadic breast cancer, late-onset Alzheimer's disease (LOAD) etc. In this research, an approach is presented to study the SNP - SNP interactions across the whole genome of GWAS data.

2.3 Methods

2.3.1 SNP -SNP Interaction Model

Most common method to identify SNPs responsible for higher disease risk is to rank them based on the probability of disease risk under an assumption or model. In this study, chi-square statistics has been employed for the purpose. It is considered that every SNP above the linkage disequilibrium (LD) width interacts with every other SNP in the whole genome. The chi-square values are obtained for every pairwise interacting SNPs. Then the significant SNPs are identified by reporting the SNP pairs less than the bonferroni corrected p-value of 0.05.

2.3.2 Chi-square Statistics

A Chi-square Statistic (*X2*) is categorical distribution statistic of two independent groups. The test statistic is given by the formula.

$$\chi^2 = \sum \frac{(\text{observed} - \text{expected})^2}{\text{expected}}$$

If given two random variables

D- Disease Status

G- Allele type

Null Hypothesis to predict SNP interactions is the two SNPs are independent of each other (unrelated).

Table 2.1: Contingency table for the Interacting SNP Data for Case and Controls

	Interacting SNP Alleles								
	0 0	0 1	0 2	1 0	1 1	1 2	2 0	2 1	2 2
Case	c1	c2	c3	c4	c5	c6	c7	c8	c9
Control	c10	c11	c12	c13	c14	c15	c16	c17	c18

Total number of samples (n)
=c1+c2+c3+c4+c5+c6+c7+c8+c9+c10+c11+c12+c13+c14+c15+c16+c17+c18

The following are the conditions for the Interacting SNPs to be independent of disease risk

1) $P(D,G) = P(D)P(G)$
2) Probability of the disease P(D) is given by the formula

   $P(D=case) = (c1+c2+c3+c4+c5+c6+c7+c8+c9) / n$
3) Probability of that interacting SNP pair will be given by the formula

   $P(G=0\ 0) = (c1+c10)/n$
4) Expected Values can be calculated using the formula

   $E(X1) = P(X=case)P(Y=00)n = (c1+c2+c3+c4+c5+c6+c7+c8+c9)(c1+c10) / n$

Once the chi-square values are computed the p-values corresponding to 8 degrees of freedom is computed. The p-values are corrected to a bonferroni p-value of 0.05 cutoffs and only the significant p-values and the corresponding SNP positions are reported.

CHAPTER 3

GPU PROGRAMMING USING CUDA

3.1 Introduction

Various studies suggest that the interactions between single nucleotide polymorphisms (SNPs) are causative for complex diseases such as sporadic breast cancer, late-onset Alzheimer's disease (LOAD) etc. Identifying the epistatic effect associated with the disease can be computationally intensive at a genomic scale. It takes up to many days to identify the interactions in SNPs, even by using the latest multiple core CPUs. In Modern Computers, Graphic Processing units have more memory and bandwidth compared to the CPUs. Hence to GPUs with hundreds of cores can be employed in identifying the SNP-SNP interactions in a genomic scale. In this research project, an implementation of Chi Square statistics capable of running on graphics processing units (GPUs) using the NVIDIA Compute Unified Device Architecture (CUDA) framework is presented to predict disease risk on a case-control population.

3.2 Background on GPUs

Increasing demand in the high resolution, 3D graphic cards have led to the production of GPUs with multiple cores at relatively lower rates. In the year 2006, to make use of the GPUs memory and bandwidth in resolving the complex computational challenges, NVIDIA released a proprietary development platform called Compute Unified Device Architecture (CUDA). CUDA™ is a parallel computing platform and programming

model that enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU).

3.3 GPU AND CUDA ARCHITECTURE

GPU uses much more of the hardware per core for data processing than CPUs, but possess far less hardware for data caching and flow control. This makes GPUs ideal to solve problems which require a small program being executed on a very large dataset in parallel.

CUDA extends GPU programmability using C functions called kernels, which when called is executed N times in parallel using the N different CUDA threads with unique thread ids in multiple blocks. There is a limit to the number of threads per block, since all threads of a block are expected to reside on the same processor core and must share the limited memory resources of that core. On current GPUs, a thread block may contain up to 1024 threads. However, a kernel can be executed by multiple equally-shaped thread blocks, so that the total number of threads is equal to the number of threads per block times the number of blocks. Blocks are organized into a one-dimensional, two-dimensional, or three-dimensional grid of thread blocks. The number of thread blocks in a grid is usually dictated by the size of the data being processed or the number of processors in the system, which it can greatly exceed.

CUDA C is an extension of C implementation which allows general purpose programming on GPUs. Currently CUDA C is largely used my many computational biologists and bioinformaticians to resolve many computationally intensive complex biological problems.

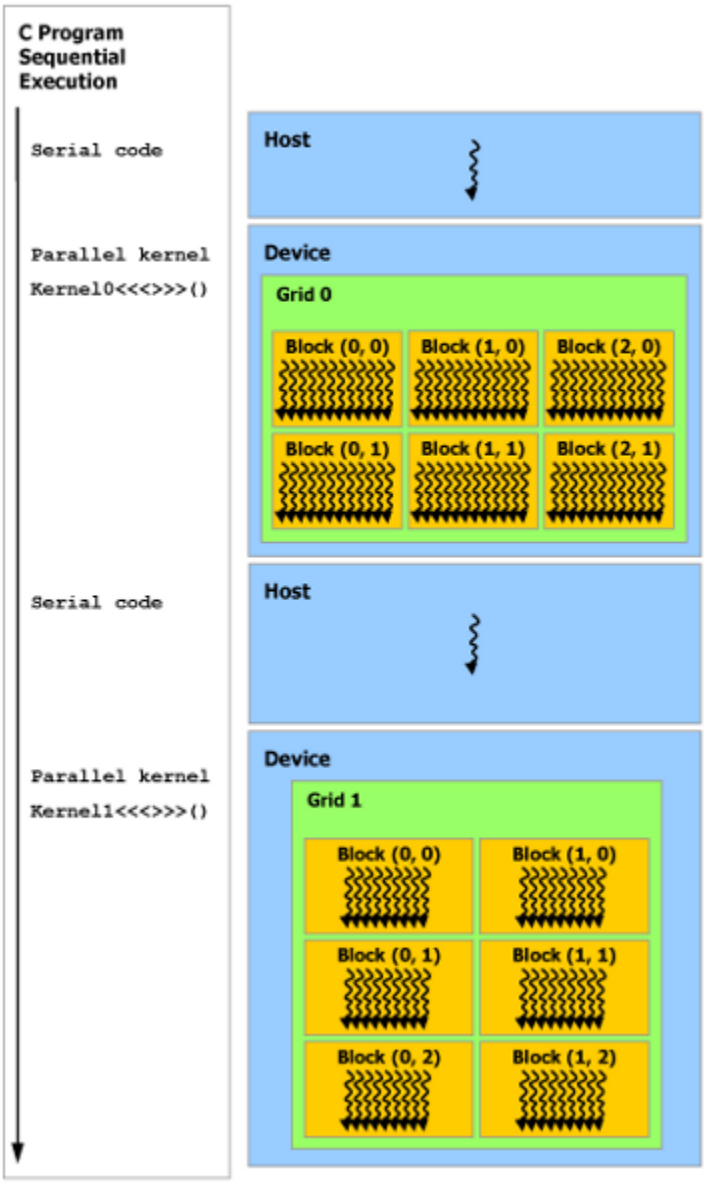


Figure 3.1 CUDA C - Heterogeneous Programming Model.
Note: Serial code executes on the host while parallel code executes on the device.

Source: NVIDIA Compute Unified Device Architecture Programming Guide Version 2.1 Tech. rep., NVIDIA Corporation; 2008

3.4 Heterogeneous Programming

In CUDA programming model, the CUDA threads execute on a physically separate device that operates as a coprocessor to the host running the C program. For example, when the kernels execute on a GPU and the rest of the C program executes on a CPU. The CUDA programs access the host and the device at their own separate memory spaces in DRAM, referred to as host memory and device memory. Hence it is necessary to do device memory allocation and de-allocation as well as data transfer between host and device memory.

3.5 Memory Coalescing

Performance can be greatly improved by some good programming practices when using the GPUs. It is evident that by increasing the global memory bandwidth by reducing the number of bus transactions and coalesce memory accesses can help CUDA program execute faster. Memory coalescing can be achieved by finding the memory segment that contains the address requested by the lowest active thread and find all the other active threads in the same segment then reducing the transaction size and by marking threads inactive when transaction is complete.

CHAPTER 4

PROBLEM STATEMENT

4.1 Background on Existing Programs

Very few algorithms have been developed to predict the SNP-SNP interactions associated to high risk of a disease. GBOOST (Dinu et al. (2012), a GPU implementation of logistic regression for analyzing the interacting pairs of the SNPs. GENIE, another GPU CUDA software package to identify epistatic interactions on SNPs was developed by Satish et al. (2011). EPIBLASTER (Kam-Thong et al. 2011) is a GPU program to detect the epistatic interaction between two locus using the logistic regression statistics.

The major limitations in using the existing SNP interaction programs are the running time and the system requirements. Computing pair wise SNP interactions on a genomic scale of a 5000 case control data costs so much CPU time and memory. The other limitations in using these programs are hardware requirement. In this thesis, a CUDA C program aimed to calculate chi-square statistics of interacting SNPs at comparatively less running time is been developed.

4.2 Implementation

4.2.2 Development Environment

This program was implemented using C language and CUDA C language extension. It was developed on a GPGPU cluster that has 12 GBs RAM of main memory, dual Intel Xeon 2.67 GHz 6-core processors – X5650r, 3 NVidia Fermi M2050 GPU cards. The

NVIDIA Fermi M2050 card has 32 nodes and 384 cores. The programs were compiled using the gnu compiler and requires GSL 1.15 library installed on the system.

4.2.3 Parallelization Algorithm

An algorithm to parallelize the interaction analysis of different SNP pairs is proposed. The input dataset into ordered in a 1 dimensional array by placing all the rows next to each other. This helps the algorithm to achieve memory coalescing. In order to minimize the effect of a significant SNP allele over the other, the single chi-square values for all the SNPs are calculated and the single significant SNPs are removed before computing the pairs. To avoid SNP interactions due to the linkage disequilibrium, a LD width variable is introduced. Hence every reference column is compared with every other column above the LD width and the respective chi-square values are calculated. The pairwise p-value is computed from the chi-square values using a c function from the GSL library, to identify the significant interacting pairs. The program reports all the SNP-SNP interactions that are significant at the bonferroni level calculated for each dataset. The bonferroni p-value = 0.05 / n, where n is the number of SNP-SNP interactions analyzed for the dataset.

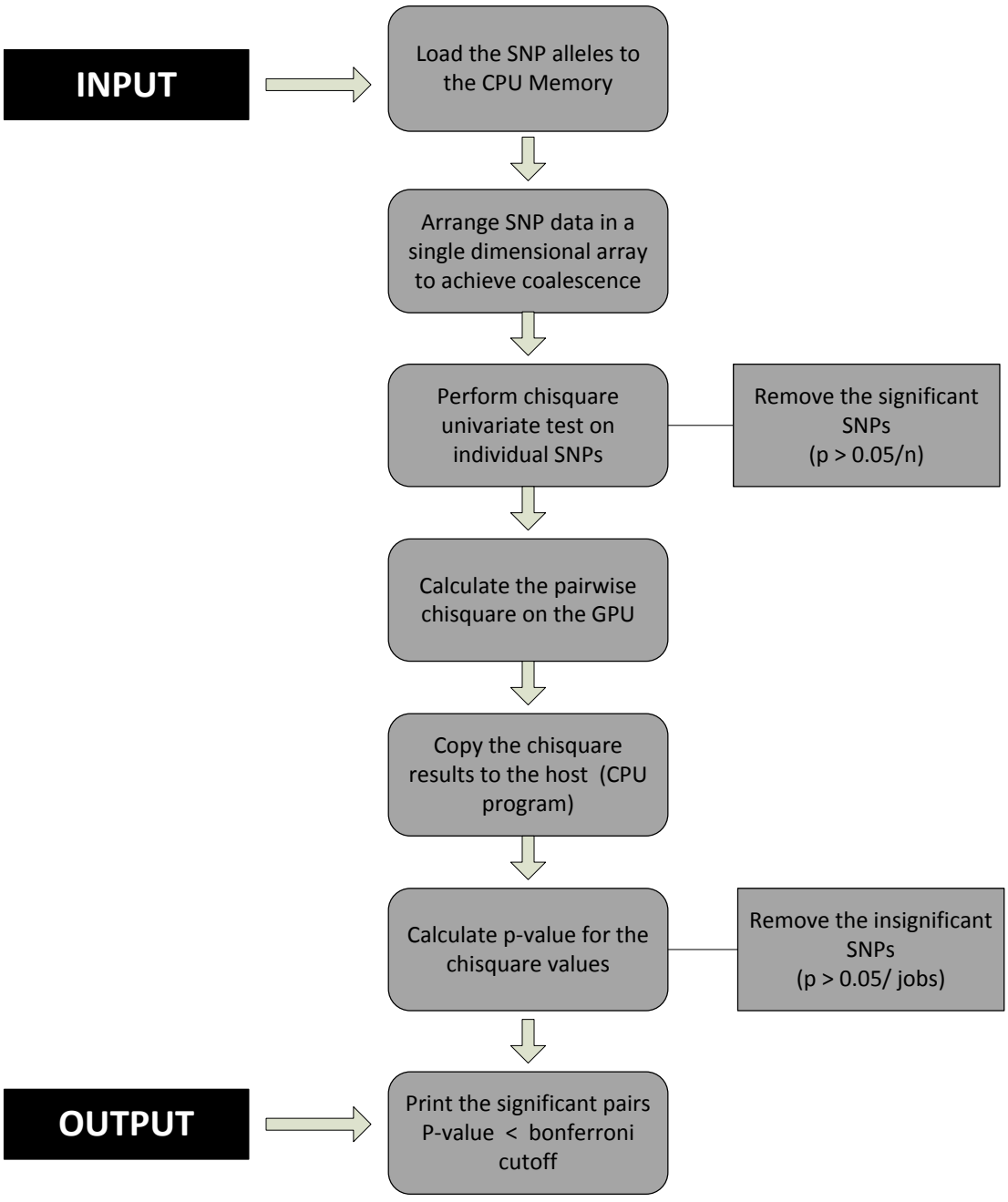


Figure : 4.1 Algorithm Design for the CUDA C Program

4.3 Dataset Used

The Wellcome Trust Case Control Consortium (WTCCC) is a collaboration of 24 leading human geneticists, who will analyse thousands of DNA samples from patients suffering with different diseases to identify common genetic variations for each condition. The WTCCC released their data in the genome-wide association study of 14,000 cases of seven common diseases and 3,000 shared controls (Burton PR et al 2007). They obtained the comparison of the control data with data obtained from the following disease samples: type 1 diabetes, type 2 diabetes, inflammatory bowel disease, breast cancer, coronary heart disease, hypertension, bipolar disorder, rheumatoid arthritis, multiple sclerosis, ankylosing spondylitis, autoimmune thyroid disease, malaria and tuberculosis. To test the performance of the program based on running time and for finding significant SNP pairs, the algorithm was tested on the following seven datasets. The size of the datasets used for the analysis is reported in the Table 4.1.

Table 4.1 Size of the WTCCC datasets used in the study

Dataset	**Total # of SNPs**	**No of Controls**	**No of Cases**
Crohns disease (CD)	405306	1748	2938
Bipolar disease (BD)	396320	1868	2938
Type 1 Diabetes (T1D)	402532	1924	2938
Type 2 Diabetes (T2D)	402532	1924	2938
Coronary heart disease (CAD)	404145	1926	2938
Rheumatoid arthritis (RA)	403301	1860	2938
Hypertension (HT)	402895	1952	2938

4.4 Results and Discussion

This study majorly contributes a new super fast algorithm to identify the pair wise significant SNPs. The algorithm was made efficient to identify the SNP pairs without testing all the pairs genome wide. To reduce the effect of false positives on the significant pair prediction, the single significant SNPs are skipped by comparison with the bonferroni corrected p-value of 0.05. The algorithm also restricts the analysis of pairs below a linkage disequilibrium width (LD width) to avoid noise in the significant pair results. Finally, the chi-square values of the computed SNP pairs are finally corrected to the bonferroni corrected p–value of 0.05.

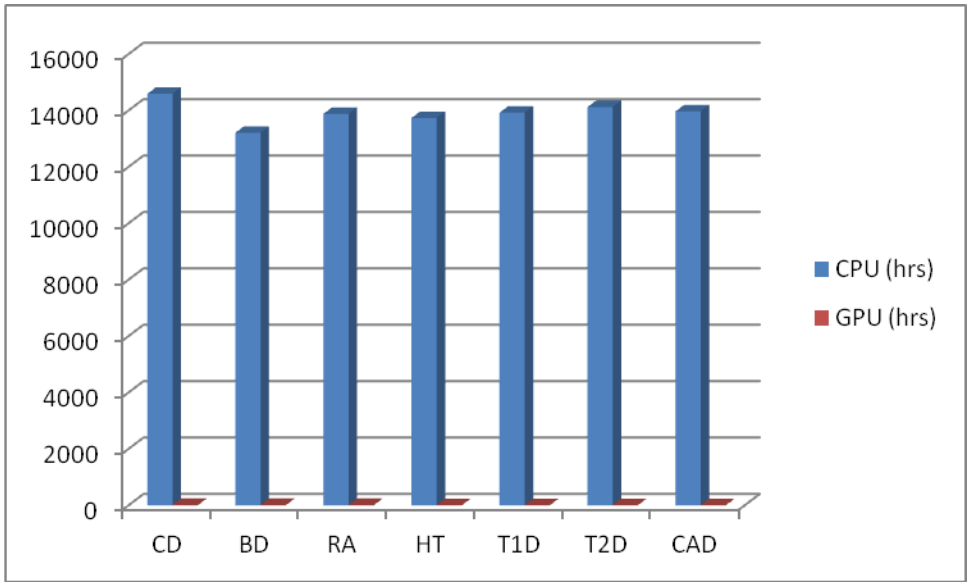
4.4.1 Algorithm Running Time

The program was used to analyse the effect of significant SNP pairs to disease association of seven WTCCC datasets. The GPU program took much less time to identify the significant pairs as opposed to other programs which runs on a CPU. The running times of each of these dataset is reported in a Table 4.2.

Table 4.2 Running Time of the WTCCC Datasets using the program

WTCCC Dataset	**Running Time (hrs)**
Crohns disease (CD)	14.15
Bipolar disease (BD)	12.1
Type 1 Diabetes (T1D)	10.3
Type 2 Diabetes (T2D)	10.45
Rheumatoid arthritis (RA)	12.2
Hypertension (HT)	10.3
Coronary heart disease (CAD)	10.1

Figure 4.1 Comparison of the GPU vs. CPU implementation



The graph above dictates the estimated running time of the algorithm is a normal CPU vs GPU implementations. The WTCCC datasets are placed against the x-axis and running time in hours across the y-axis. From the graph it is evident that there are about 130% increase in the running time in using GPUs than CPU computation for this algorithm.

Table 4.3 Significant pairs identified in the WTCCC datasets

Dataset	SNP1	SNP1 id	Single p-value	SNP2	SNP2 id	Single p-value	Pair wise Chi-square	Pair wise p-value
CD	9157	rs4655684	1.35E-07	273114	rs12789493	8.56E-04	7.92E+01	6.93E-14
CD	9157	rs4655684	1.35E-07	273113	rs2155226	1.16E-03	7.85E+01	9.63E-14
CD	9157	rs4655684	1.35E-07	273115	rs3862807	1.33E-03	7.69E+01	2.09E-13
CD	9157	rs4655684	1.35E-07	273112	rs1892953	3.65E-03	7.54E+01	4.19E-13
RA	12103	rs691531	1.98E-06	12134	rs3929937	1.79E-01	1.18E+02	1.03E-21
RA	12103	rs691531	1.98E-06	12130	rs10782591	1.42E-01	1.11E+02	2.40E-20
RA	12103	rs691531	1.98E-06	154313	rs9784858	2.69E-07	7.48E+01	5.32E-13
RA	154034	rs4394275	4.87E-04	154295	rs2857212	9.59E-06	7.79E+01	1.31E-13
RA	154040	rs2523534	3.99E-06	154236	rs3135392	1.69E-06	8.00E+01	4.81E-14
RA	154040	rs2523534	3.99E-06	154186	rs910050	8.74E-07	7.66E+01	2.41E-13
RA	154040	rs2523534	3.99E-06	154253	rs4530903	2.31E-07	7.60E+01	3.18E-13
RA	154045	rs5025315	2.68E-05	154236	rs3135392	1.69E-06	7.47E+01	5.67E-13
RA	154048	rs5022119	2.33E-05S	154236	rs3135392	1.69E-06	7.45E+01	6.14E-13
RA	154095	rs2516478	6.66E-03	154295	rs2857212	9.59E-06	9.16E+01	2.22E-16
RA	154107	rs760293	3.16E-06	154236	rs3135392	1.69E-06	8.74E+01	1.56E-15
RA	154107	rs760293	3.16E-06	154186	rs910050	8.74E-07	8.13E+01	2.67E-14
RA	154143	rs3130287	4.48E-07	154186	rs910050	8.74E-07	9.54E+01	3.65E-17
RA	154143	rs3130287	4.48E-07	154236	rs3135392	1.69E-06	7.86E+01	9.32E-14
RA	154186	rs910050	8.74E-07	154253	rs4530903	2.31E-07	9.58E+01	3.09E-17
RA	154186	rs910050	8.74E-07	154296	rs2857210	9.47E-05	8.44E+01	6.32E-15
RA	154186	rs910050	8.74E-07	154328	rs241403	1.50E-05	7.60E+01	3.14E-13
RA	154195	rs9391858	3.64E-04	154236	rs3135392	1.69E-06	9.69E+01	1.88E-17
RA	154195	rs9391858	3.64E-04	154230	rs5000563	1.43E-01	8.03E+01	4.28E-14
RA	154195	rs9391858	3.64E-04	154234	rs3129877	1.43E-01	8.00E+01	4.81E-14

Dataset	SNP1	SNP1 id	Single p-value	SNP2	SNP2 id	Single p-value	Pair wise Chi-square	Pair wise p-value
RA	154195	rs9391858	3.64E-04	154229	rs3135342	8.41E-02	7.86E+01	9.18E-14
RA	154195	rs9391858	3.64E-04	154232	rs3129872	1.21E-01	7.78E+01	1.34E-13
RA	154198	rs12528797	8.56E-06	154236	rs3135392	1.69E-06	9.83E+01	9.62E-18
RA	154201	rs6930777	4.54E-06	154236	rs3135392	1.69E-06	1.01E+02	2.97E-18
RA	154229	rs3135342	8.41E-02	154328	rs241403	1.50E-05	7.71E+01	1.89E-13
RA	154230	rs5000563	1.43E-01	154328	rs241403	1.50E-05	7.60E+01	3.08E-13
RA	154232	rs3129872	1.21E-01	154328	rs241403	1.50E-05	7.50E+01	4.92E-13
RA	154234	rs3129877	1.43E-01	154328	rs241403	1.50E-05	7.48E+01	5.51E-13
RA	154236	rs3135392	1.69E-06	154328	rs241403	1.50E-05	9.00E+01	4.75E-16
RA	154236	rs3135392	1.69E-06	154296	rs2857210	9.47E-05	7.74E+01	1.61E-13
RA	154272	rs9275765	3.39E-03	154304	rs2857154	4.66E-03	9.35E+01	9.16E-17
RA	154272	rs9275765	3.39E-03	154305	rs7382347	7.29E-03	8.89E+01	7.61E-16
RA	154272	rs9275765	3.39E-03	154306	rs2857129	1.10E-02	8.85E+01	9.18E-16
RA	154273	rs9275772	5.61E-03	154304	rs2857154	4.66E-03	9.22E+01	1.64E-16
RA	154273	rs9275772	5.61E-03	154305	rs7382347	7.29E-03	8.77E+01	1.33E-15
RA	154273	rs9275772	5.61E-03	154306	rs2857129	1.10E-02	8.72E+01	1.76E-15
RA	154275	rs9275793	5.40E-03	154306	rs2857129	1.10E-02	8.73E+01	1.62E-15
RA	154275	rs9275793	5.40E-03	154304	rs2857154	4.66E-03	9.24E+01	1.51E-16
RA	154275	rs9275793	5.40E-03	154305	rs7382347	7.29E-03	8.79E+01	1.23E-15
RA	154295	rs2857212	9.59E-06	154367	rs9296069	1.00E-02	8.49E+01	4.98E-15
RA	155430	rs9296318	3.26E-01	155460	rs10947857	1.80E-01	7.73E+01	1.72E-13
RA	155433	rs2894387	2.72E-01	155460	rs10947857	1.80E-01	7.96E+01	5.79E-14
CAD	12132	rs691531	1.58E-04	12162	rs3929937	7.73E-01	1.35E+02	2.12E-25
CAD	12132	rs691531	1.58E-04	12158	rs10782591	6.70E-01	1.32E+02	1.17E-24

The above table gives the SNP positions and SNP ids of the interacting which are identified to be significant by this algorithm. The program reported 4 significant pairs in the crohns disease dataset. The significant SNP pairs identified in crohns disease dataset are part of the genes IL23R and C11orf30 which are biologically evident and significant in correlations with crohns disease pathogenesis.

The program reported about 2 significant pairs in Coronary Heart disease dataset rs691531, rs3929937 and rs691531, rs10782591 that lie in the chromosome 1 as part of the genes RPL17P5 and HS2ST1.There are about 34 significant pairs reported in the rheumatoid arthritis dataset that come from different genes RPL17P5 , HS2ST1, HLA S, HCP 5,HCG 26 , HLA-DRA , BAG6, TNXB, C6orf10.

There is not much significance seen in the datasets bipolar disease, hypertension, type 1 diabetes and type 2 diabetes.

CHAPTER 5

CONCLUSION

In this study, an algorithm which can identify the significant SNP pairs of a huge GWAS dataset is presented. This algorithm is efficient and fast paced to identify the significant SNP pairs.

The significant SNP pairs identified in Crohns disease dataset are part of the genes IL23R and C11orf30. These genes are biologically evident and significant in correlation to pathogenesis.

The algorithm performs well in getting rid of the false positives by assigning a LD width, removing the single significant SNPs and also by correcting the significant SNP pairs to bonferroni p-value of 0.05.

APPENDIX A

GPU PROGRAM TO COMPUTE SNP-SNP INTERACTIONS USING GENOME-WIDE ASSOCIATION DATA

This appendix includes the source code of the GPU program to compute the chi-square statistics of SNP-SNP interactions.

A.1 CUDA C Program

Main.cu

Purpose: Main program which calls the kernel function

Output: Tab delimited text file containing the SNP interacting positions and p-values

```
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <cuda.h>
#include "book.h"
#include "kernel.cu"
#include <gsl/gsl_cdf.h>
#include <gsl/gsl_sf.h>

int main(int argc ,char* argv[]) {
  FILE *fp;
  int size;

/* Initialize rows, cols, ncases, ncontrols from the user */
  int rows=atoi(argv[2]);
  int cols=atoi(argv[3]);
  int ncases=atoi(argv[4]);
  int ncontrols=atoi(argv[5]);

  printf("%d,%d,%d,%d\n",rows,cols,ncases,ncontrols);
```

```
/*Kernel variable declaration */

   int THREADS = 256;
   int BLOCKS;
   int LD_width = 10;
   unsigned char *dev_dataT;
   float *results;
   float host_results[cols];
   float result; int colid;
   float p_temp;
   int host_colid[cols];
   float chisq[cols];
   int *dev_colid;
   int jobs, ref_col;
   int transfer = 0;
   float pair_cutoff = 0;
   float totaljobs ;

/* Variable initialized to perform univariate tests */

float c_zero, c_one, c_two, con_two,con_zero,con_one,c1_expected,c2_expected,
c3_expected,c_total;
float con1_expected, con2_expected, con3_expected,con_total ;
float chi,total,t_col1,t_col2,t_col3;
float p[cols];

/* set the cut off */
float cutoff = 0.05/cols;
printf("CUTOFF IS %e\n",cutoff);
/* Validation to check if the data file is readable */
fp = fopen(argv[1], "r");

if (fp == NULL) {
       printf("Cannot Open the File");
       return 0;
   }

size = rows * cols;
totaljobs = gsl_sf_choose(cols,2);
pair_cutoff = 0.05 / totaljobs ;

printf("Size of the data: %d    pair_cutoff %e\n",size,pair_cutoff);

unsigned char *dataT = (unsigned char*)malloc(size*sizeof(unsigned char));
printf("Transferring data to Memory\n");
```

```
/* Transfer the SNP Data from the file to CPU Memory */

for(int i=0 ; i < size; i++ ) {
       int tmp;
       fscanf(fp,"%d",&tmp);
       dataT[i] = (char)(((int)'0')+tmp);s
       if(i == size - 1){
             transfer = 1;
             printf("SNP Data Transferred to the Memory.... Processing\n");
       }
       fflush(stdout);
}
fclose(fp) ;

/* Univariate tests on all columns */

for (int m = 0 ; m < cols ; m++ ) {
    int n = m;
    c_one = 1.0f; c_zero = 1.0f; c_two = 1.0f; con_one = 1.0f ; con_zero = 1.0f;
    con_two = 1.0f;
     while(n < ncases * cols) {
         if(dataT[n] == '0') { c_zero ++; }
             else if(dataT[n] == '1') { c_one++; }
             else if(dataT[n] == '2') { c_two++ ; }
             n = n + cols;
         }
     c_total = c_zero + c_one + c_two;
     n = m + ncases * cols;
     while(n < size) {
         if(dataT[n] == '0') { con_zero++; }
             else if(dataT[n] == '1') { con_one++;  }
             else if(dataT[n] == '2') { con_two++ ; }
             n = n + cols;
          }
     con_total = con_zero + con_one + con_two;
     total = c_total + con_total;
     t_col1 = c_zero + con_zero;
     t_col2 = c_one + con_one;
     t_col3 = c_two + con_two;
     c1_expected = t_col1 * c_total / total;
     c2_expected = t_col2 * c_total / total ;
     c3_expected = t_col3 * c_total / total ;
     con1_expected = t_col1 * con_total / total ;
     con2_expected = t_col2 * con_total / total ;
     con3_expected = t_col3 * con_total / total ;
     chi = (c_zero - c1_expected) * (c_zero - c1_expected) / c1_expected;
```

```
    chi += (c_one - c2_expected) * (c_one - c2_expected) / c2_expected;
    chi += (c_two - c3_expected) * (c_two - c3_expected) / c3_expected;
    chi += (con_zero -con1_expected) * (con_zero -con1_expected) / con1_expected;
    chi += (con_one - con2_expected) * (con_one - con2_expected) / con2_expected;
    chi += (con_two - con3_expected) * (con_two - con3_expected) / con3_expected;
     chisq[m] = chi;
     p[m] = gsl_cdf_chisq_Q(chi,2);
    }
if(transfer == 1){
      /* Reading the dataT array for comparison and kernel function*/
      /* allocate the Memory in the GPU for SNP data */

      fflush(stdout);

      HANDLE_ERROR(cudaMalloc((unsigned char**) &dev_dataT, size *
sizeof(unsigned char) ));
      HANDLE_ERROR(cudaMalloc((float**) &results, cols * sizeof(float) ));
      HANDLE_ERROR(cudaMalloc((int**) &dev_colid,cols * sizeof(int) ));

      /*Copy the SNP data to GPU dev_dataT*/

      HANDLE_ERROR(cudaMemcpy(dev_dataT, dataT, size * sizeof(unsigned char),
cudaMemcpyHostToDevice));
      HANDLE_ERROR(cudaMemcpy(dev_colid,host_colid, cols * sizeof(int),
cudaMemcpyHostToDevice));

      /* as indexing start from 0 - 49 but cols are 50 so cols -1 */

      //fflush(stdout);
      printf("SNP 1   Uni p-value     SNP 2   Uni p-value     Chi-Square      P-value\n");
      printf("#####   ############    #####   ############    ########
########\n");

      for(int j=0;j < (cols - (2 * LD_width + 1));j++) {
         if( p[j] > cutoff ) {
            jobs = cols - j;
            ref_col = j;
            BLOCKS = (jobs + THREADS - 1)/THREADS;

            /*Calling the kernel function */
            cudaPrintfInit();

            fflush(stdout);


kernel<<<BLOCKS,THREADS>>>(rows,cols,ncases,ncontrols,jobs,ref_col,dev_dataT,
```

```
results,dev_colid);

 cudaPrintfDisplay(stdout, true);
 cudaPrintfEnd();

 fflush(stdout);

            //Copy the results back in host
            HANDLE_ERROR(cudaMemcpy(host_results,results,cols *
sizeof(float),cudaMemcpyDeviceToHost));
            HANDLE_ERROR(cudaMemcpy(host_colid,dev_colid,cols *
sizeof(int),cudaMemcpyDeviceToHost));

            fflush(stdout);
            /* check condition to find out the SNP values to be sorted */

            for(int k = 11 ; k < jobs; k++) {
            float max = 0.0f ;
                  for( int i=11; i < jobs; i++ ){
                              if((host_results[i] >= max) &&  (p[ref_col + host_colid[i]] >
                               cutoff)) {
                                    max = host_results[i];
                                    result = host_results[i];
                                    colid = host_colid[i];
                              }
                  }
                  host_results[colid] = 0;
                  p_temp = gsl_cdf_chisq_Q(result,8);
                  final_colid= ref_col + colid;
                  if( p_temp <  pair_cutoff ) {
                        printf("%d     %e     %d     %e     %e
                        %e\n",ref_col,p[ref_col],final_colid,p[final_colid],result,p_temp);
                  }
                  else {
                        break;
                   }
            }
           }
      else {
            continue;
      }
      }

/* free the Memory in the GPU */
fflush(stdout);
printf("\n###DONE###\n");
```

```
cudaFree( dev_dataT );
cudaFree( results );
cudaFree(dev_colid );
return 0;
}
else {

	printf("ERROR: ERROR loading the data.\n");
}
}
```

A.2 Kernel Function

Kernel.cu

Purpose : kernel function to calculate the chi-square values

Output : When called from the main function, computes the chi-square statistics of the

SNP pairs and returns the chi-square values to the main function

```
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <cuda.h>
#include <math.h>
#include "book.h"
#include "cuPrintf.cu"

__global__ void kernel( int rows, int cols , int cRows , int contRows ,int jobs,int ref, unsigned char *snpdata,float *results,int *dev_colid){
	unsigned char x, y;
	int m, n ;
	unsigned int p = 0 ;
	int cases[9];
	int controls[9];
	int tot_cases = 1;
	int tot_controls= 1;
	int total = 1;
	float chisquare = 0.0f;
	float exp[10];
	float Conexpected[9];
```

```
float Cexpected[9];
float numerator1;
float numerator2;

int tid  = threadIdx.x + blockIdx.x * blockDim.x;
cases[0]=1;cases[1]=1;cases[2]=1;cases[3]=1;cases[4]=1;cases[5]=1;cases[6]=1;
cases[7]=1;cases[8]=1;


controls[0]=1;controls[1]=1;controls[2]=1;controls[3]=1;controls[4]=1;controls[5]=1;con
trols[6]=1;controls[7]=1;controls[8]=1;
if ((tid < jobs) && (tid > 10 )) {
      for ( m = 0 ; m < cRows ; m++ ) {

            x = snpdata[m * cols + ref];
            y = snpdata[m * cols + (ref + tid)];

              if ( x == '0' && y == '0') { cases[0]++; }
              else if (x == '0' && y == '1') { cases[1]++; }
              else if (x == '0' && y == '2') { cases[2]++; }
              else if (x == '1' && y == '0') {  cases[3]++; }
              else if (x == '1' && y == '1') {  cases[4]++; }
              else if (x == '1' && y == '2' ) { cases[5]++; }
              else if (x == '2' && y == '0' ) { cases[6]++;}
else if (x == '2' && y == '1') { cases[7]++; }
              else if (x == '2' && y == '2') { cases[8]++; }
              else { //do nothing
             }

         }
      for ( n = cRows ; n < cRows + contRows ; n++ ) {
            x = snpdata[n * cols + ref];
            y = snpdata[n * cols + (ref + tid)];

              if ( x == '0' && y == '0' ) { controls[0]++; }
              else if (x == '0' && y == '1') { controls[1]++; }
              else if (x == '0' && y == '2') { controls[2]++; }
              else if (x == '1' && y == '0') { controls[3]++; }
              else if (x == '1' && y == '1') { controls[4]++; }
              else if (x == '1' && y == '2' ) { controls[5]++; }
              else if (x == '2' && y == '0' ) { controls[6]++;}
              else if (x == '2' && y == '1') { controls[7]++; }
              else if (x == '2' && y == '2') { controls[8]++; }
              else { //do nothing
             }
```

```
                }

                 tot_cases =
                cases[0]+cases[1]+cases[2]+cases[3]+cases[4]+cases[5]+cases[6]
                +cases[7]+cases[8];
                tot_controls =
                controls[0]+controls[1]+controls[2]+controls[3]+controls[4]
               +controls[5]+controls[6]+controls[7]+controls[8];
                total = tot_cases + tot_controls;

                   for( p = 0 ; p < 9; p++) {
                              exp[p] = (float)cases[p] + controls[p];
                              Cexpected[p] = tot_cases * exp[p] / total;
                              Conexpected[p] = tot_controls * exp[p] / total;
                              numerator1 = (float)cases[p] - Cexpected[p];
                              numerator2 = (float)controls[p] - Conexpected[p];
                              chisquare += numerator1 * numerator1 / Cexpected[p] +
                              numerator2 * numerator2 / Conexpected[p];

                   }

                   cuPrintf("tid is %d\n", tid);
                   dev_colid[tid] = tid;
                    results[tid] = chisquare;
                   cuPrintf("SNP1 is %d SNP2 is %d results[tid] is %f\n",ref
                  ,dev_colid[tid], results[tid]);
          }
}
```

APPENDIX B

GPU PROGRAM SPECIFICATIONS

B.1 Pre-requisites

1. A NVIDIA Graphics card with CUDA Support

   This includes any chipset from the Geforce 8, 9, 100, 200, 300, 400 and 500 series with at least 256MB on-board RAM.

2. CUDA LLVM Compiler (NVCC)

3. GNU Scientific Library (GSL)

B.2 Command to Compile the Program:

nvcc *-I/<Path to gsl library> /gsl1.15/include main.cu*
-L/<Path to gsl library> /gsl1.15/lib –lgsl -lgslcblas

B.3 Execute the Program:

B.3.1 Input File:

SNP Dataset should be encoded as per additive and dominance coding ie Count of the minor alleles per person.

Argument 1 - Number of rows in the dataset.

Argument 2 - Number of SNPs in the dataset.

Argument 3 - Number of Controls in the dataset.

Argument 5 - Number of Cases in the dataset.

B.3.2 Command Used:

<Executable> <Dataset Filename> <# Rows> <#SNPS> <# Controls> <# Cases> > <Output Filename>

APPENDIX C

PERL SCRIPT TO OBTAIN THE SNP IDS FROM SNP POSITIONS

Pos2snpid.pl

Purpose : Map the SNP positions to SNP ids.

Input : Output file from main.cu, SNP ids file

Output : Tab delimited text file with SNP positions mapped to SNP ids.

```
$SNPfile = shift;
$diseasefile = shift;

open(IN, $SNPfile);
@SNPfile = <IN>;
chomp @SNPfile;
close IN;

open(IN, $diseasefile);
@diseasesnps = <IN>;
chomp @diseasesnps;
close IN;

%SNPids = ();
@id1 = () ; @b = (); @id2 =();

for($i=0;$i < scalar(@SNPfile);$i++) {
	@a = split(/ /,$SNPfile[$i]);
	$SNPids{$a[0]} = $a[1];
}

@keys = keys(%SNPids);
$size = scalar(@keys);

for($j=0; $j < scalar(@diseasesnps); $j++ ) {
	@b = split("\t",$diseasesnps[$j]);
	if(defined($SNPids{$b[0]})) {
		$id1[$j] = $SNPids{$b[0]};
	}
	if(defined($SNPids{$b[2]})) {
```

```
                $id2[$j] = $SNPids{$b[2]};
        }
}

for($k=0;$k < scalar(@diseasesnps);$k++) {
        @m = split("\t",$diseasesnps[$k]);
        print "$m[0]\t$id1[$k]\t$m[1]\t$m[2]\t$id2[$k]\t$m[3]\t$m[4]\t$m[5]\n";
}
```

Usage:

perl pos2snpid.pl <FILE 1> <FILE 2>

FILE1 - File containing the list of SNP ids and SNP positions.
FILE2 - Output from the main.cu CUDA C program

REFERENCES

Stankiewicz P, Lupski JR (2010). Structural Variation in the Human Genome and its Role in Disease. *PubMed, PMID: 20059347*.

Jan O. Korbel et al. (2007). Paired-End Mapping Reveals Extensive Structural Variation in the Human Genome, *PubMed, PMID: 20059347*.

Kai Ye, Marcel H. Schulz1, Quan Long, Rolf Apweiler1 and Zemin Ning (2009). Pindel: a Pattern Growth Approach to Detect Breakpoints of Large Deletions and Medium Sized Insertions from Paired-end Short reads, *Bioinformatics, 25 (21):2865-2871.doi:10.1093/bioinformatics/btp394.*

Chen K, Wallis JW, McLellan MD, Larson DE, Kalicki JM, Pohl CS, McGrath SD, Wendl MC, Zhang Q, Locke DP, Shi X, Fulton RS, Ley TJ, Wilson RK, Ding L, Mardis ER (2009). BreakDancer: an algorithm for high-resolution mapping of genomic structural variation, *PubMed, PMID: 19668202*.

Faraz Hach, Fereydoun Hormozdiari, Can Alkan, Farhad Hormozdiari, Inanc Birol, Evan E Eichler, and S Cenk Sahinalp (2010).mrsFast: a Cache-Oblivious Algorithm for Short-read Mapping, *PMC, PMCID: PMC3115707*.

Kidd JM, Cooper GM, Donahue WF, Hayden HS, Sampas N, Graves T, Hansen N, Teague B, Alkan C, Antonacci F (2008). Mapping and Sequencing of Structural Variation from Eight Human Genomes. *Nature. 2008;453:56–64.*

Miko, I. (2008) Epistasis: Gene interaction and phenotype effects. *Nature Education 1(1).*

NVIDIA Compute Unified Device Architecture Programming Guide Version 2.1 Tech. rep (2008), *NVIDIA Corporation.*

Yung L.S, et al (2011). GBOOST: a GPU-based tool for detecting gene–gene interactions in genome-wide case control studies. *Bioinformatics. 27:1309–1310.*

Satish Chikkagoudar, Kai Wang and Mingyao Li (2011). GENIE: a Software Package for Gene-Gene Interaction Analysis in Genetic Association Studies using Multiple GPU or CPU Cores. *BMC Research Notes*, 4:158 doi:10.1186/1756-0500-4-158

Kam-Thong T, Czamara D, Tsuda K, Borgwardt K, Lewis CM, Erhardt-Lehmann A, Hemmer B, Rieckmann P, Daake M, Weber F, Wolf C, Ziegler A, Pütz B, Holsboer F, Schölkopf B, Müller-Myhsok B(2011).EPIBLASTER-fast exhaustive two-locus epistasis detection strategy using graphical processing units, *PMC, PMCID: PMC3060319*

Burton PR, Wellcome Trust Case Control Consortium.(2007) Genome-wide Association Study of 14,000 Cases of Seven Common Diseases and 3,000 Shared Controls. *Nature. 7;447(7145):661-78.*