ROYAL INSTITUTE
OF TECHNOLOGY

Design and Implementation of a Computational Platform and a Parallelized Interaction Analysis for Large Scale Genomics Data in Multiple Sclerosis

DANIEL UVEHAG <UVEHAG@KTH.SE>

Master's Thesis at ICT
Supervisor: Helga Westerlind <helga.westerlind@ki.se>, Ryan Ramanujam <ryan.ramanujam@ki.se>
Examiner: Jim Dowling <jdowling@kth.se>

TRITA-ICT-EX-2013:129

Abstract

The multiple sclerosis (MS) genetics research group led by professor Jan Hillert at Karolinska Institutet, focuses on investigating the aetiology of the disease. Samples have been collected routinely from patients visiting the clinic for decades. From these samples, large amounts of genetics data is being generated. The traditional methods of analyzing the data is becoming increasingly inefficient as data sets grow larger. New approaches are needed to perform the analyses.

This thesis gives an introduction to the relevant genetics and discusses possible approaches for enabling more efficient execution of legacy analysis tools, as well as improving a gene-environment and gene-gene interaction analysis. Different computational paradigms are presented followed by the implementation of a computational platform to support the researchers' existing, and possibly future, analysis needs. The improved interaction analysis application is then implemented and executed in a virtual instance of this platform. The performance of the analysis application is then evaluated with respect to the original reference application.

Referat

Design och implementation av beräkningsplattform och paralelliserad interaktionsanalys för storskaliga genetiska data inom multipel skleros

Professor Jan Hillert vid Karolinska Institutet leder en forskargrupp som fokuserar på etiologin bakom multipel skleros (MS). Under flera årtionden har patientprover samlats in från kliniken och från dessa prover har stora mängder genetiska data genererats. De traditionella analysmetoderna blir allt mer ineffektiva då datamängderna öker. Det finns ett stort behov av nya tillvägagångssätt och metoder för att analysera dessa data.

Denna uppsats ger en introduktion i relevant genetik och diskuterar olika tillvägagångssätt för att möjliggöra effektivare exekvering av befintliga analysverktyg, så väl som förbättring av en gen-miljö och gen-gen-interaktionsanalys. Olika etablerade beräkningsparadigmer presenteras, följt av en implementation av en beräkningsplattform som ett stöd i att tillgodose forskargruppens nuvarande och möjliga framtida behov. Den förbättrade interaktionsanalysen är sedan implementerad och exekverad i en virtuell instans av plattformen. Interaktionsanalysens prestanda utvärderas sedan och jämförs med ursprungsimplementationen.

Contents

Nomenclature

A	Adenine, a nucleotide base
Allele	Variant of a SNP
Allosome	Sex chromosome, determines an organism's sex
Antigen	A contraction of the words *antibody generator*, a foreign element causing a reaction in the immune system
AP	Attributable proportion due to interaction
API	Application Program Interface
Autosome	Chromosomes appearing in pairs where both chromosomes are near identical
Base pair	Two base nucleotide bases on opposing strands in a DNA molecule
C	Cytosine, a nucleotide base
Candidate Gene	A gene believed to be associated with disease based on prior knowledge
Daemon	A background process or service
DFS	Distributed File System
DNA	Deoxyribonucleic acid
G	Guanine, a nucleotide base
Genotype	The genetic traits in an individual or organism
GWAS	Genome Wide Association Studies
Monogenic disease	An inherited disease caused by a single genetic factor
MS	Multiple sclerosis
Myelin	A "sheath" surrounding and protecting the nerve fibers, isolating the nerve signals
Phenotype	An observable trait in an individual or organism
RERI	Relative risk due to interaction
RNA	Ribonucleic acid
SNP	Single-Nucleotide Polymorphism, a single nucleotide variation in the DNA sequence (pronounced *snip*)
Scratch space	Local disk space used for temporary storage
T	Thymine, a nucleotide base

Chapter 1

Introduction

1.1 Acknowledgements

I would like to express my deepest gratitude and appreciation to my dear supervisors, Helga Westerlind and Ryan Ramanujam, and to professor Jan Hillert for providing me with this wonderful opportunity. It has been an astonishing journey into uncharted territories giving me lots of experience and new perspective. Your support, through bright and dark times, have been invaluable.

I'd like to thank Olle Gartell and the IT department for providing me with information and access to some of the central infrastructure, as well as the entire research group of Jan Hillert for their support and good times. Last, but not least, I'd like to thank my examiner, Jim Dowling for the discussions and making this project a possibility.

1.2 Background

The multiple sclerosis (MS) genetics research group lead by professor Jan Hillert at Karolinska Institutet, focuses on investigating the aetiology of the disease. MS is a debilitating neurological disorder affecting primarily young adults which leads to declining neurological function with symptoms such as impaired motor activity, double vision, tremors and pain. It has a severe impact on the quality of life for the patients and also causes major socioeconomic consequences [44]. There currently is no cure for the disease and it requires lifelong treatment.

MS is a research area of large interest with focus on prevention and mitigation of symptoms. This is commonly done by investigating the inherited genetic factors that are thought to cause or contribute to the disease. One method for this is to query individuals' genomes using *genome wide association studies* (GWAS). This is computationally intensive and generates large amounts of data. Statistical methods are applied to examine differences between groups of cases and controls. However, methods for analyzing the data efficiently in more complex ways is lacking.

Given the increased volumes of data collected and generated on an annual basis,

analyses performed using traditional methods are becoming increasingly infeasible. Many of the existing tools available for performing analyses are developed in laboratories around the world to solve a particular problem at that particular time. Some gain traction, becoming popular and widely used in the scientific community, albeit a number of these tools have outgrown their usage as larger data sets are becoming available and their execution times grow too large.

There are multiple ways to address this problem depending on the nature of the data and analysis. As an example, it might be possible to split an analysis into multiple smaller discrete parts, possibly with overlapping data, and then run the tools on the individual parts in parallel, drastically reducing the overall execution time and possibly even lowering memory usage. Another approach, and the preferred one, is to develop new methods and algorithms which are faster and utilize today's multi-core computers and multi-node computational resources more efficiently.

1.3 Objectives

This thesis focuses on designing and implementing two different, but related problems:

1. a concurrent and distributed version of GEIRA, a gene-environment and gene-gene interaction analysis tool, written in Java.
2. a small versatile computational platform for various genetic analyses on which GEIRA can run as well as the existing legacy applications.

This project will contribute in the advancement of research within, and possible outside, the genetics research group as well as other diseases than multiple sclerosis.

1.4 Limitations

Due to the limited time constraints, this project will not focus on:

- implementing any other analysis than the aforementioned gene-environment and gene-gene interaction analysis (GEIRA).
- complete integration with the existing IT infrastructure.

1.5 Motivation

The main motivation for this project is to accelerate research and increase efficiency in computational analyses, initially in the fields of MS genetics and complex disease. Currently, much time is spent waiting for executions of analyses to complete. Under certain circumstances it is not even possible to perform an analysis the preferred way as the execution time would reach years. Enabling parallel execution of these

analyses, or parts of them, on a dedicated computational platform would lead to much faster results, which in turn could lead to more efficient and better research. Future steps would be to develop new methods and analysis tools which could natively harness the capabilities of such an environment more efficiently.

1.6 Outline

The following contains a brief description of each chapter:

Chapter 2 provides an introduction to the different organizations, a background in relevant genetics and multiple sclerosis, relevant statistics for the analysis, different computational paradigms, the original analysis and the existing resources available.

Chapter 3 describes the design and implementation of the computational platform.

Chapter 4 describes the design and implementation of the gene-environment and gene-gene interaction analysis tool.

Chapter 5 presents the results of combining the two implementations and the experimental environment.

Chapter 6 discusses the conclusions of the implementations and results.

Chapter 7 discusses possible future usages for the analysis and computational platform.

Chapter 2

Background

This chapter provides background information about the various organizations involved, i.e. the Karolinska Institutet, Center for Molecular Medicine and the MS research group led by Professor Jan Hillert. Then, a brief introduction to the relevant genetics and statistics for the analysis is given, followed by a description of the analysis and existing resources.

2.1 Organizations

2.1.1 Karolinska Institutet

The *Karolinska Institutet* (KI) [8] is one of the world's most prominent medical universities [1]. It's located in Sweden with its main campus just north of Stockholm in Solna. Karolinska Institutet consists of several boards, committees and departments focusing on different research areas. Each department consists of an administrative unit and several different research groups, focusing on various research areas. The overall organization more or less follows a traditional hierarchical structure [9].

2.1.2 Center for Molecular Medicine

The Center for Molecular Medicine (CMM) [2] is a foundation, currently gathering almost 30 research groups from four different departments at Karolinska Institutet, and clinical researchers from the Karolinska University Hospital. The research areas are mostly focused on common diseases and the foundation works to further promote inter-group collaborations since its establishment in 1997. To help pursue this goal, the foundation has several auxiliary facilities, including a local IT department tasked with serving the various needs of the many researchers and maintaining the local IT infrastructure.

2.1.3 Multiple Sclerosis Research Unit

Professor Jan Hillert leads the multiple sclerosis research group at the Department of Clinical Neuroscience (CNS). It consists of two subgroups, the genetics and the immunology subgroups. The genetics subgroup focuses on finding factors, both genetic and environmental, which could contribute to disease development. Samples have been accumulated from patients routinely visiting the clinic over the course of several decades, from which large amounts of data is being generated. The research group is primarily located at CMM.

2.2 Multiple Sclerosis and Genetics

This section will give a brief introduction on the genetics relevant for the analysis and explain the basics of multiple sclerosis.

2.2.1 Genetics

Genetics is a sub-field of biology focusing on the study of *genes*, how they affect traits in living organism as well as *heredity*, i.e. how those traits are inherited from their parent organisms. A gene is commonly defined as *"a locatable region of genomic sequence, corresponding to a unit of inheritance, which is associated with regulatory regions, transcribed regions, and or other functional sequence regions"* [52]. In simpler terms, they are segments on the *DNA* molecule. The DNA molecule is a double stranded helix composed of *phosphate*, *sugar* and four different nucleobases, namely *Adenine*, *Thymine*, *Cytosine* and *Guanine*.

Every cell has a near-identical set of chromosomes containing DNA organized in a compact structure. The number of chromosomes and their lengths vary between different species. The human set of chromosomes typically consists of 22 *autosome* pairs and one *allosome* pair (sex chromosome pair). Different chromosomes have a different number of genes with varying lengths. The autosomes are ordered by the total number of *base pairs*, in descending order (i.e. chromosome pair 1 has the largest number of base pairs, while the opposite is true for chromosome pair 22).

The genes dictate the behavior of a cell, e.g. what function it performs and which proteins are to be produced. The DNA is transcribed into RNA which then is translated into proteins. All the DNA is stored in the nucleus of the cell. Depending on current conditions and external stimuli, different genes are activated and different proteins are synthesized.

When new cells are created and the genome is copied, a single base pair is sometimes substituted for another. This is referred to as a *point mutation*. Normally, these are nonsense mutations without any major change of function, but occasionally they are beneficial and thus stay persistent in subsequent generations. This type of bi-allelic mutation is called a *single-nucleotide polymorphism* (SNP). Other mutations also occour, such as *insertions* and *deletions* where a single base pair gets added or left out. These variations sometimes express themselves in visible traits

amongst organisms, e.g. different appearances, monogenic diseases or treatment response. However, most of the time the variations result in no observable difference. SNPs are thought to contribute to a large proportion of the human variation. According to *dbSNP* build 128, there are around 12 million unique SNPs in the human genome [6]. The different variants of a SNP are known as the *alleles* [29].

2.2.1.1 Genetic Models

Gregor Johann Mendel is considered to be the founder of modern genetics. He conducted experiments in the middle of the 19th century which laid the foundation for today's modern genetics [50]. During his experiments, he discovered that when colored and white pea flowers were cross-bred, the colors would not blend and the offspring would express the same color as its colored parent. The same phenomena would be observed for a number of other traits, such as the shape of the pea pods and the seeds. He called these visible traits (or *phenotypes*) *factors*, later renamed to *genes*. Through this, he introduced the concept of dominant and recessive genes, where one gene would dominate the other and thus be the expressing phenotype. Since then, the meaning of genes has changed and in modern genetics they are thought to be protein coding regions [52].

2.2.1.2 Complex Diseases

Complex diseases are a subset of *genetic disorders* which are multi-factorial and cannot be derived from a single cause [42]. These disorders, such as MS, are thought to be caused by multiple genetic factors in combination with environmental factors and lifestyle habits (e.g. smoking). It is still not known how the factors interact with respect to disease contribution and there are many unknown factors yet to be discovered. Many factors are being found with low contributing effects, making it very difficult to determine the true cause of genetic predisposition. `HLA-DRB1*15:01` is the strongest known genetic risk factor for MS [55] with a presence of 61% in patients and 31% in controls in Sweden [48]. Effect sizes diminish very strongly for all other associations compared with `HLA-DRB1*15:01`.

As research progresses, some diseases previously thought to be monogenic have been demonstrated to show signs of complex disease. One disease previously thought to be monogenic, Rett's syndrome, manifested different impairments amongst patients where the associated gene mutation was present in all individuals. This suggests multiple contributing factors e.g. environmental factors, rather than a single genetic mutation [36].

One common analogy used within complex disease is *Rothman's "pie" model*, also known as the *sufficient cause* model [59]. In this model, a pie represents a cause for disease and a disease may have one or more causes (pies). The pieces of a pie represent *risk factors* and the same factor can be present in more than one pie. The risk factors alone do not cause disease and they could be genetic mutations, different alleles or environmental factors. Once a pie is complete (i.e. all risk factors

are present for a cause), a sufficient cause for the disease is present, even if several causes (pies) are known but their criteria are not satisfied. All risk factors in a cause are considered to be of equal importance, i.e. all the pieces in each pie are of equal size.

Given a hypothetical disease with the known causes $C = \{C_1, C_2\}$ with their factors $F_1 = \{a, b, c, d, e\}$ and $F_2 = \{a, b, f, g, h\}$, respectively, the equivalent pie model for this disease is given in figure 2.1. The subset of the factor sets $F_s = F_1 \cap F_2 = \{a, b\}$ is present in both pies and contribute to both causes. So by definition, if one individual has all factors for at least one of the pies, disease will be present. The reverse is also true, i.e. if disease is present, all factors for at least one of the pies exist.

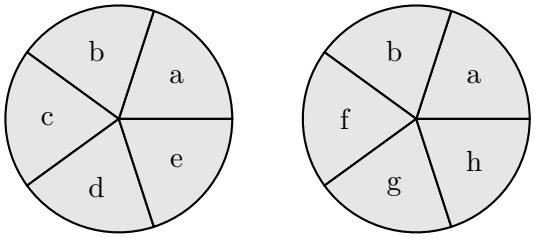

Figure 2.1. Example pie model for a hypothetical disease

2.2.2 Multiple Sclerosis

Multiple sclerosis (MS) is considered by most researchers to be autoimmune in nature (a *chronic demyelinating disease*). The immune system reacts to specific parts of the body as though it was foreign *antigen*, treating them as hostile. In MS, the immune system attacks the central nervous system by targeting the *myelin*, damaging the sheath. This causes nerve signals to deteriorate which negatively impacts motor abilities and causes pain. Today there are more than 50 known genes with an association to MS [61].

2.3 Statistics

This section will give a brief introduction to the relevant statistics used within the analysis and multiple sclerosis studies.

2.3.1 Association Studies

In association studies, frequencies of SNPs (or markers) are studied in order to try and find association to a disease. Although an association cannot be used to prove causality, it may further reveal more about disease mechanisms. Since these

associations are unlikely to be found by chance due to study design, it is very likely to have a disease related effect and is a prime candidate for future focused studies.

When performing association studies, there are several designs to choose from. One frequently used design is the *case-control study* where individuals from a population are separated into *cases* (individuals with disease) and *controls* (individuals without disease). These two groups are mutually exclusive and exhaustive. An analysis is performed in order to try to find any associations between *genotypes* and *phenotypes* by finding variations between the two groups. Using statistical tests, *allele frequencies* are measured to determine whether any statistically significant relations may exist or not.

One metric used when trying to measure the effect size between the cases and controls is by calculating the *odds ratios* (OR) between the two groups in terms of *minor allele frequencies* (MAF). Odds ratios in these studies are often defined as the ratio of the odds of developing a disease while having an allele, against the odds of developing the disease while not having the allele. It also has an important role in the *logistic regression* model discussed in section 2.3.2.

As an example, let X (the predictor variable) denote a dichotomized variable representing the presence of an allele (1 for present, 0 for absent) and Y (the response variable) denote the presence of a disease. The odds ratio can then be estimated by using equation 2.1.

$$OR = \frac{P(Y = 1|X = 1)/(1 - P(Y = 1|X = 1))}{P(Y = 0|X = 1)/(1 - P(Y = 0|X = 1))} \tag{2.1}$$

2.3.1.1 Genome Wide Association Studies

A genetic study can be performed at a variety of levels e.g. a single marker, *candidate genes* or genome wide association studies (GWAS). In a GWAS, SNPs from the entire genome is used from all the participating individuals in the study. The number of SNPs in the study depends on what chip is used for the genotyping of the individuals, and the genotyping chip can be disease specific, focusing on a prespecified set of SNPs. There currently are chips available supporting more than one million SNPs.

2.3.2 Logistic Regression

Logistic regression is a regression model commonly used when the outcome variable (Y) is dichotomized (i.e. it belongs to one of two possible groups). This is used instead of the linear regression model, since the latter cannot handle categorical outcome variables. This is often the case in genetics as disease is considered a categorical variable, i.e. it's either present or not. Observations where an affection status is unknown are normally excluded from an analysis.

As with linear regression, logistic regression is used for predicting the outcome variable, or, more specifically, predicting the probability (p) of the outcome variable being $Y = 1$. This probability is a continuous value between 0 and 1. It is then

normalized by using a logistic transformation of p, called the *logit of p* as shown in equation 2.2.

$$\text{logit}(p) = \ln\left(\frac{p}{1-p}\right) \tag{2.2}$$

Unlike the probability, this function assumes values from $-\infty$ to $+\infty$ and it is also the log of the *odds ratio* that the outcome variable is $Y = 1$, previously shown in equation 2.1. Given the transformation of p, it is possible to express it as a linear equation given in equation 2.3. Fitting the model is performed using the *maximum likelihood* method.

$$\text{logit}(p(x)) = \ln\left(\frac{p(x)}{1-p(x)}\right) = \beta_0 + \sum_{i=1}^{n} \beta_n x_n \tag{2.3}$$

Using this, the probability p can be calculated using equation 2.4.

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n)}} \tag{2.4}$$

2.3.3 Interaction Models

Interaction is traditionally defined as statistical interaction where two independent variables combined have a multiplicative effect on the dependent variable [28]. This does not necessarily reflect the mechanisms in biological interaction, which is of interest for epidemiological and genetic research. Therefore, the term *biological interaction* is introduced, also known as *additive interaction*, and defined as the departure from additivity of effects between risk factors [58, 30].

There are still no full explanations for complex disease such as MS, despite all associations found. Furthermore, associations found through genome wide association studies still cannot fully explain the heritability [38, 67]. Researchers are now investigating how different markers interact with each other, both on an additive scale and on a multiplicative scale, in hopes to better understand the underlying mechanisms.

2.4 Gene-Environment and Gene-Gene Interaction Research Application

Gene-Environment and Gene-Gene Interaction Research Application (GEIRA) [34] is a statistical analysis tool used for genome-wide gene-environment and gene-gene interaction analyses. It is available, both for the SAS software and R language [56], in multiple versions depending on which of the previously introduced interaction model is to be used. It is also capable of performing computations using either a dominant or recessive genetic model.

GEIRA can be used for analyzing smaller data sets, but quickly becomes impractical as the data sets become larger. Given an example data set of $8,000$ individuals and $500,000$ SNPs, the execution time on an Intel Xeon E5520 at 2.27 GHz reaches over 5 months (extrapolated from a partial benchmark execution, data not shown). The original analysis also needs to be extended to better detect possible *false positives*. This is achieved by implementing statistical *resampling*, giving better estimates of the sample statistics. Implementing this in the original application would immensely increase the execution time, possibly resulting in execution times reaching several years.

The application uses transposed data sets [16] from the PLINK program [66] for the genotype and phenotype data. Each data set contains two text files, one `TPED` file and one `TFAM` file. These are essentially two tables, containing a large number of rows and columns. Each row in the `TPED`-file represents a SNP with genotype data for all the individuals. It also contains the meta data relevant to the SNPs, such as identifier and genomic location. The `TFAM` file contains information regarding the individuals, e.g. identifier and affection status (affected, unaffected or unknown). Example files containing 10 SNPs and 10 individuals can be found in appendix A.1.2.

A second data set, a whitespace delimited text file, contains the interaction variable for all individuals in the PLINK data set. The interaction variable can be either a genetic marker (such as a SNP or a gene) or an environmental factor, e.g. smoking. Unlike the transposed PLINK data set, the first line in the file is a header containing column names. This file may also contain covariate data with one covariate per column, although the R version of the program currently cannot handle more than a single covariate (or the absence of one). An example file containing an interaction variable and one covariate can be found in appendix A.1.1.

Both data sets are initially loaded into main memory. Depending on the size of the data set, this can be a lengthy process. Each individual SNP in the PLINK data set (a row vector) is then compared to the interaction variable (column vector) from the second data set. A series of statistical computations are then performed in order to determine which allele is thought to be associated with risk. Odds ratios are calculated for the different combinations of exposures (presence and absence of genetic risk combined with and without environmental exposure), both on an additive scale and multiplicative scale. These operations will be explained further down in the implementation section.

An additional feature is recoding of the data when the additive effect of factors, combined or independent, gives an odds ratio lower than 1.0. This indicates that exposure of a factor is considered protective rather than a risk. When assessing *attributable proportion due to interaction* (AP, described further in section 4.2.5), all odds ratios must fulfill this criterion (i.e. all odds ratios must be greater than or equal to 1.0) [46]. Thus, the data is recoded such that the interpretation of risk for the factor related to the odds ratio changes to instead be associated with a protective effect. This only affects the calculations on the additive scale. As an example,

if the odds ratio related to the absence of a genetic risk and the simultaneous presence of an interaction variable is less than 1.0, the presence of the interaction variable is considered to have a protective effect rather than a risk. If multiple odds ratios simultaneously are lower than 1.0, the lowest is chosen as the reference and recoding is performed accordingly. All results are buffered in main memory until all calculations are complete, whereas the results are flushed to an output file.

2.5 Computing Paradigms

There are several different computing paradigms available, each with their own requirements and benefits. The common goal of all of these is that they try to solve a computationally intense problem by exploiting parallelism in various architectures, albeit using different approaches and for different purposes. They differ in properties such as heterogenity, inter-connects and the applicability of traditional methods and tools.

Michael J. Flynn devised a taxonomy containing four classifications for different computer architectures [37]. Today, the most commonly used is the *Multiple Instruction, Multiple Data* (MIMD) architecture. A computer using a multi-core CPU is considered as a MIMD architecture where different cores may execute multiple instructions independently on multiple data. This architecture can further be divided into two categories:

Single Program, Multiple Data (SPMD) which focuses on running the same application on different data over multiple processors, currently considered to be the dominant style of parallel programming [12]. Although the same application is running on multiple processors, each instance may execute at different points in the code.

Multiple Program, Multiple Data (MPMD) which focuses on running multiple applications on different data, sometimes referred to as the *manager/worker* paradigm [5].

2.5.1 High Performance Computing

High Performance Computing (HPC) [33] is the field focusing on supercomputers and related areas. One of its major application areas is creating concurrent software and algorithms, solving large computational problems in a short amount of time. This involves massive processing of data, requiring many CPUs and cores, lots of memory and storage space. Ideally, such an environment is under a single administrative domain.

A supercomputer is a high-end machine consisting of several inter-connected computers (nodes), usually characterized as being one of the fastest computers available. Thus, it's a volatile definition which is constantly changing over time.

What is considered to be a supercomputer today might not qualify as such in the future.

The supercomputer is closely related to the *mainframe* but differ in a few key architectural aspects. The mainframe focuses on executing many applications concurrently and must run reliably for a long period of time. These properties make the mainframe ideal for corporates and governments, and in transaction heavy environments, e.g. banks or trading companies. Unlike the mainframe, the supercomputer focuses on running a few number of applications as fast as possible. Reliability is not such a critical property as with the mainframe, failed executions would ideally be rescheduled once the system is operational again.

The hardware involved in a high performance computing environment is often tightly-coupled, high-end, dedicated and homogenous, using high-speed interconnects for inter-node communication (e.g. InfiniBand or Ethernet). Many of the applications make strong assumptions of homogenity and high synchronization using standard APIs, such as by using *Message Passing Interface* (MPI) [62].

The metrics used for measuring high performance computing environments is often how many *floating-point operations per second* (FLOPS) it can perform. This is a consequence of the heavy use of floating-point calculations in the various fields of scientific computing. Current supercomputers (e.g. Lindgren [20] at PDC or Titan [13] at Oak Ridge National Laboratory) theoretically reaches several hundreds or thousands of Tera FLOPS and consists of a large number of cores.

2.5.2 High Throughput Computing

High Throughput Computing is a paradigm described by M. Livny et al. [47] as an environment which focuses on solving computational problems over a longer period of time, as opposed to the HPC environment. For long-running executions, measuring how many floating-point operations per second such an environment can perform is often of little interest. Instead, these systems are measured by how many such operations it can perform per week, or month, focusing on overall throughput.

Given that todays commodity computers are sufficiently powerful to perform various computations, the HTC environment focuses on how to better utilize collections of these instead of focusing on time-efficiency. Applications typically decompose problems into a large number of smaller sub-problems with little to no expectations of synchronization. These sub-problems can then be executed in parallel over a network of nodes.

Since the executions run for a longer period of time, reliability becomes an important factor. The environment has to be robust even though the system commonly is composed of unreliable components. To solve this, different mechanisms could be employed, e.g. *checkpointing* with rollback functionality. This could be implemented transparently to an application using external mechanisms (e.g. CRIU [3]), by using an application specific mechanisms [64, 43] or by implementing the necessary logic in the application layer itself.

The HTC paradigm began with the *HTCondor* project at the University of

Wisconsin as a resource management system, enabling better utilization of their large number of UNIX workstations [47]. Their computer environment is heterogenous with a decentralized ownership. HTCondor solves the problem of coordinating these machines by providing class advertisements, or *ClassAds*, as a match making mechanism where resources advertise their capabilities through *Resource Offers*, and applications query available resources using *Resource Requests*.

2.5.3 Many-Task Computing

A third computing paradigm is *Many-Task Computing* [57] which aims to bridge the gap between the HPC and HTC paradigms. MTC applications are, just as HPC applications, focused on performance. Although an HPC application usually follows the SPMD model, the same is not necessarily true for MTC applications. MTC applications usually consist of many small, loosely coupled tasks where file sharing is the primary means of inter-process communication. This enables workflows of heterogenous tasks, with or without dependent steps, where scripting and macros become an important factor in order to automate the execution process.

Since the tasks are mostly short lived, file management and I/O becomes a non-negligible overhead for communication intensive applications. This puts further requirements on the environment in order to provide computational efficiency. Many existing HPC solutions could be a viable environment for MTC applications, with little to no modifications of the environment.

2.5.4 Peer-to-Peer Computing

Peer-to-Peer (P2P) [51] computing removes the traditional distinct separation of the client and server model. It is a sub-field of distributed computing where applications utilize distributed resources in a decentralized manner (e.g. computing power and storage) to achieve a specific goal. Nodes (computers) participating in a peer-to-peer network can act as both a server and client, depending on the application requirements, and contributes with local resources to the system. Eliminating central servers reduces potential bottlenecks and the number of single points of failures, ideally making the system more robust and resilient.

Nodes in a P2P system are loosely coupled and their participation is highly dynamic. The system organizes the participating nodes in abstract network overlays using structured [63, 49, 60] or unstructured [65] membership management mechanisms in the application layer. Given the dynamic nature of the node participation where nodes leave and join the system, causing *churn*, P2P systems are usually designed to encompass this with fault tolerance. Where data availability is a key concern, redundancy could be implemented e.g. through replication of data.

2.6 Existing Resources

The IT environment is heterogenous and distributedly owned by the different research groups and the local IT department at CMM. Both the client and server environment is highly heterogenous with respect to hardware, operating systems and software applications. The IT department manages the central infrastructure and provides centralized authorization, authentication and storage facilities.

There currently is no local computational infrastructure available for the researchers. Most of the analyses are currently being performed on the local workstations or portable computers (laptops). Some of the research groups have acquired their own dedicated servers for performing their computations, although these are very few in numbers. It is feasible to a certain extent but as the amount of data is increasing, this is becoming inefficient and time consuming and no mechanism for resource management or allocation is available. These resources are also very rarely shared between the different research groups.

Many of the tools used are legacy software written by various laboratories around the world (e.g. PLINK [66], EIGENSTRAT [54], Unphased [35], BEAGLE [32] and GERMLINE [39]), intended for smaller data sets usually analyzed in a sequential manner. The presence of concurrent software is scarce at best, despite the fact that many of the problems they solve are parallel by nature (e.g. independent pair-wise comparisons between different records in data sets).

Even though external software is predominantly used, some analysis tools are created by the researchers or their collaborators, mostly in higher level languages aimed at statistical analyses, e.g. *SAS* and R [56]. A negligible number of these tools have been identified to use more than a single core when executed. The need for more effective algorithms and concurrency is drastically increasing, as well as resource management and the possibility to batch analyses to dedicated compute nodes in order to offload the local workstations and execute analyses in parallel.

2.6.1 External Resources

There are a number of already existing resources on a national level available for academic researchers. Vetenskapsrådet (Swedish Research Council) [27] is an administrative authority in Sweden responsible for the organization *Swedish National Infrastructure for Computing* (SNIC) [25]. SNIC coordinates the six Swedish HPC centers and makes these resources available for academic researchers. This includes managing the infrastructure for computations as well as data storage.

Two of these HPC centers used at the Karolinska Institutet are the *Centre for Parallel Computers* (PDC) [15] at the Royal Institute of Technology (KTH), and Uppsala Multidisciplinary Center for Computational Science (UPPMAX) [26]. Using these compute and storage resources is not always possible for a number of reasons:

- **Ethical permissions.** It is not always permitted to export data to off-site

locations due to ethical permissions, making it more or less impossible to utilize the existing external resources without negotiating new permissions exempting specific centers, e.g. PDC or UPPMAX. In genetics, this is not an uncommon scenario due to the nature of the data involved.

- **Queuing times.** Other researchers are often utilizing the resources for long-running analyses, consuming all the resources for a long period of time. Queuing time then becomes a significant negative factor for shorter analyses that could get stuck in queue.
- **Environment mismatch.** For many small and short lived tasks, an HPC environment is not always ideal and might not fulfill some of an application's requirements, e.g. I/O requirements as they might target a different use case. As an example, an MTC application consisting of several job steps, where file sharing is used as inter-process communication, might not work ideally in an HPC cluster where message passing is used for inter-process communication and a shared file system might not even exist. The high I/O required for shared data, or local temporary files, could easily become a bottleneck.
- **Administrative overhead.** Applying for time and resource allocation is a process which is considered time consuming and costly compared to running a smaller analysis using locally available resources.

Some analyses are so computationally intensive that a single workstation would not suffice and one of the available HPC centers would be superfluous. New analysis tools also need to be tested and verified before being deployed in a larger HPC environment. There is an urgent need to fill the gap between these two extremes where the researches can verify their newly developed tools, as well as run smaller analyses not large enough for said HPC environment. Analyses where the ethical permissions can not be changed are also candidates for such a solution.

Chapter 3

Design and Implementation of the Computational Platform

This section will describe the computational platform, it's requirements and design decisions.

3.1 Requirements

The computational platform has to fulfill the following requirements:

Scalability: It should be possible to scale from a few seperate machines (nodes) with a few cores to several thousand nodes if necessary.

Portability: The solution has to be portable among different GNU/Linux distributions, as GNU/Linux is the predominantly used operating system. The distribution initially used within the platform may change in the future.

Fault Tolerance: The system has to be fault tolerant and self healing. Node failures, network failures, partitioning and other transient failures should be handled accordingly. The system should be able to resume at a later stage when any interruptions have been averted.

3.2 Design decisions

The computational work load within the research group consists mainly of executing legacy applications, and novel analysis tools. The latter category has recently emerged and is a large contributing factor for introducing a computational platform. The applications are mostly written in C, Java, R and Python for Linux, and the newly developed analysis tools are to some extent parallelized. The legacy tools are usually single threaded and sometimes uses excessive amounts of memory. The researchers have a need to batch large work loads, either as independent

jobs, or pipelines consisting of several dependent tasks. These pipelines almost exclusively use file sharing as means of inter process communication, demanding a shared file system and very little need for synchronization using message passing and fast inter-connects. It was decided that the most appropriate solution would be an initial local, small cluster designed for this workload, resembling more or less an MTC environment.

To harmonize with the local IT departments preferences, its stability and large support in the scientific community, *Scientific Linux* [19] was chosen as the GNU/Linux distribution for the cluster.

3.3 Authentication and Authorization

All researchers' user accounts in these systems are assigned unique ID numbers. Synchronizing these account IDs between the cluster nodes is of outmost importance to several auxiliary systems. The cluster needs to integrate with the central IT department's authentication, authorization and storage services for easier utilization and these services rely heavily on consistent use of these user ID numbers (UIDs). The researchers would use their already existing central credentials to log on to a *front-end node* and transparently reach their centrally stored datasets through mounted network file systems, present on all nodes. The central authentication system has no attribute for the UIDs, instead it would use its *record ID* from which an UID is derived and deterministically mapped to a configurable integer range.

Accessing many of the central services heavily depend on *Kerberos* [17] for authentication and an *LDAP* database backend for resolving security group memberships, used as *roles* for authorization. Due to local policy restrictions, the lifetime of a Kerberos ticket is limited to mere hours. However, it can be renewed for a substantially longer period. The short lifetime of a ticket becomes an obstacle for long running executions, which is often the case. To overcome this limitation, lifecycle management of Kerberos tickets becomes a necessity and is achieved through the *AUKS* subsystems [40]. It is a utility for managing Kerberos credentials for non-interactive applications, e.g. applications running on a cluster in batch mode. AUKS keeps a local cache of forwardable tickets that are automatically renewed shortly before expiration, allowing applications to continue accessing the Kerberized services, e.g. the central storage service.

3.4 Job and Resource Management

The cluster would require a versatile resource management and queuing system, supporting the execution and scheduling of a large number of jobs. For this, *Simple Linux Utility for Resource Management* (SLURM) [45] was chosen for a number of reasons. The most important contributing factors were its flexibility, feature completeness, licensing and maturity compared to some other existing solutions. Another factor further supporting this decision was the researchers' familiarity with

it as it is used in the UPPMAX HPC center in some of its environments [21, 22, 23]. It also provides seamless integration with AUKS through the use of SLURM's plugin architecture.

SLURM offers a vast number of features which could initially be considered superfluous, but could be employed at a later stage as the cluster grows. An initial cluster of a few nodes under a single administrative domain, used by a single research group, might not need priority queues, partitioning, accounting and reporting as it would be under said group's sole administrative authority. However, as the cluster grows and more researches vest in it and the ownership becomes distributed, some of these will be of higher importance, albeit implementing this is outside the scope of this thesis.

The central component of SLURM is the controller daemon. It monitors resources and delegates work to all the different compute nodes. The controller daemon is a potential single point of failure and a bottle neck as there may only be one active controller in any cluster. To address this, a passive backup controller is introduced, providing fail-over functionality. The backup controller monitors the primary controller's health through the use of heartbeat messages. In the event the primary controller fails, the backup controller would assume the role as the primary controller after a configured timeout is reached.

Each participating compute node in the cluster has a SLURM daemon accepting work from its controller. All incoming instructions are executed under the context of the submitting user. To achieve this, SLURM heavily depends on the highly scalable *MUNGE* (MUNGE Uid 'N' Gid Emporium, a recursive acronym) authentication service [11]. It uses symmetric key encryption with a pre-shared key amongst all cluster nodes (compute and controller nodes) and also requires a common resolution mechanism of the users' IDs and names between the nodes as these are being transmitted rather than the user names.

Compute nodes are organized into *partitions* which are logical sets of nodes. These partitions may be disjoint or overlapping (despite its name), depending on the configuration. Initially, a single partition is used consisting of all available nodes. As more nodes are added and the configuration becomes heterogenous and more complex, this configuration is subject to change.

When submitting a job to SLURM, an allocation is requested for the execution. The request contains specifications for the job allocation, e.g. required number of nodes, cores, memory and possibly scratch space. If these requirements can never be met by the cluster, the request is immediately canceled. If the requested resources are present, but currently unavailable, the request is queued until the resources become available. Otherwise, the request is immediately granted and the required resources are reserved for the job. When a job terminates, the acquired resources are automatically released.

3.4.1 Data Sharing

There are multiple ways for a running job to share data amongst its allocated nodes over an interconnect. They can either share data through message passing in the application layer, or through file sharing. Some resource management or batch systems (e.g. SLURM) have built in tools for distributing the data to nodes that an application could use. These tools usually only distribute the data to the nodes involved in the current job. One such tool is SLURM's `sbcast` which uses a fan-out approach to disseminate data to a local destination on each node. This relies heavily on the homogenity of the nodes' file system hierarchy as the target destination has to be specified as an absolute path, present on all nodes.

Another approach is to use a *distributed file system* (DFS) between the nodes for sharing data. This enables the nodes to transparently see and share data amongst each other through locally mounted network file systems. It is of high importance when applications use file sharing as inter-process communication between different jobs or job steps. Initial benchmarking revealed that the built-in file transmission tool in SLURM (`sbcast`) did not perform as well when compared to a distributed file system (data not shown). Using a DFS enables many different configuration options with regards to fault tolerance and performance, but also introduces increased complexity.

It was decided to include an inter-cluster DFS for data sharing. Several different distributed file systems were reviewed, and amongst them *GlusterFS* was chosen. It is officially supported by Red Hat which *Scientific Linux* is based upon and supports simultaneous replication and striping for fault tolerance and performance.

Each node participating in the distributed file system is a member of a *volume*. Each node exports a brick (a sub-volume) which is added to the volume and operates as specified by the volume's configuration. In the replicated-distributed configuration, some nodes would mirror each other (e.g. in groups of three) to achieve both redundancy and performance. Data can be read from multiple nodes simultaneously, increasing overall read performance, and a single failed node would not make any data unavailable due to the usage of replicas. These features and its scale-out capabilities makes it ideal for such an environment.

3.5 Environment

Since the physical hardware for a cluster is currently not available, the experiments were performed in a virtual environment. The physical environment consists of a single workstation with the following specifications:

- Dual Intel Xeon X5650 CPUs operating at 2.67 GHz with 6 cores.
- 96 GiB DDR3 memory.
- Dual OCZ Vertex 4 SSD drives with 256 GB capacity each.

This workstation was used as the virtualization host (hypervisor) for the cluster, using Oracle VirtualBox 4.2 [14]. All development, testing and simulation was performed on this workstation.

3.5.1 Virtual Environment

There was a total of nine virtual machines present, all running with the same configuration:

- One virtual CPU with a single core.
- Eight GiB of RAM.
- One 16 GiB disk drive used by the operating system.
- One 16 GiB disk drive used for the distributed file system.

One of the virtual machines was used as the *interactive login node*, the frontend, called *terminus*. The remaining machines were dedicated compute nodes (node[0-7]). A switched internal virtual gigabit ethernet network was used as the interconnect between the nodes and the hypervisor.

3.5.2 Distributed File System

A GlusterFS volume was configured in a distributed-replicated configuration with three replicas. This gave a good trade-off between fault tolerance and performance. All nine nodes were participating in the distributed file system as storage bricks (participating sub-volumes). This file system was used for sharing intermediary files between nodes and jobs. No mechanisms for epilogue initialization of temporary storage, or prologue cleanup, is in effect.

3.6 Authentication and Authorization

As previously mentioned, SLURM makes heavy use of MUNGE as an authentication service and relies on synchronized user IDs between the different nodes, e.g. a user with the ID $1,000$ needs to have this ID across all the different nodes. This is also the case for the distributed file system. To accommodate this need, *Winbind* from the Samba project [18] was used as a network account authentication daemon. The nodes were promoted to full members in the existing Microsoft Active Directory domain services, provisioning user and group information and thus complete authentication and authorization integration. User and group IDs were derived from the central *record ID*s (RIDs) as previously mentioned, using a deterministic algorithm. Using the *pluggable authentication modules* sub-system in Linux, users were allowed remote access through a *secure shell* (SSH) connection to nodes where an allocated job was executing through the `pam_slurm` module.

Chapter 4

Design and Implementation of the Analysis

The following chapter describes the design choices and the implementation of the analysis for this project. First, a framework for parallel execution using Java is introduced, followed by the implementation of the analysis application.

4.1 Java Parallel Processing Framework

The Java Parallel Processing Framework (JPPF) [7] is a freely available framework for executing concurrent tasks over a network of compute nodes. It is highly scalable and designed for processing large numbers of computationally intensive tasks. It is intended as a grid solution, offering features such as fault-tolerance and self-healing capabilities. The framework consists of three major components:

Client: The client is the main application which utilizes a network of nodes for its computations. It creates and submits *jobs*, sets of *tasks*, to the network through a connection to a server.

Server: The server (sometimes called driver) receives connected clients' jobs and disseminates bundles of tasks to its connected computational nodes. The results from the nodes are returned to the submitting client through the server.

Node: The node receives tasks from its connected server and executes them, eventually sending back the results the server.

This architecture follows the traditional MPMD (master/worker) paradigm. The server divides the jobs and orchestrates the compute nodes. However, both the client and the server can be configured to be participating nodes by enabling local execution of tasks. All configuration options can be set through external configuration files read at runtime, programmatically in the application or a combination of both.

The framework utilizes multiple methods of achieving parallel processing. During application development and design, it is possible to use the framework's native API and explicitly define jobs and tasks. A job is a set of tasks and a task is the smallest unit of code that can be executed on the networked nodes. Another approach is to use the framework's *ExecutorService* API which works as a facade to the JPPF client. This capability makes porting of existing applications using Java's concurrency API a trivial effort.

Submitted tasks are serialized and either sent as individual jobs, or batched together depending on how the framework is configured. When a task reaches a node, the class loader mechanism in JPPF remotely loads all required classes from the client through the server, unless they are previously cached or has already been deployed in the node's environment. The server also maintains its own cache for performance reasons. These caches are bounded and old entries are discarded as new classes are required.

The nodes establish a separate communication channel between each other, used for detecting crashes. At regular intervals, heartbeat messages are sent on this channel, which is acknowledged by the recipient. If a heartbeat message is not acknowledged within a configurable time window, or a specified number of consecutive messages are not acknowledged, the remote node is considered to be crashed (or failed). The submitting node (i.e. a client or server) would then requeue any submissions that have been lost, for execution on a live node. If no executing nodes are available (i.e. compute nodes are missing and local execution is disabled), execution stalls until a working node becomes available.

The JPPF framework will be used in the implementation for a number of reasons. Besides the already mentioned features it also offers excellent monitoring, deployment and configuration capabilities, a free license, and a well structured and documented API. The configuration settings will be read from external files for greater flexibility.

4.1.1 Node Discovery Process

The nodes and clients are responsible for connecting to a server, which can be achieved in different ways. They may either manually connect using pre-configured addresses or perform dynamic discovery by multicasting in the local network. In the virtual environment, multicasting is used. Both nodes and clients can be connected to multiple servers simultaneously, but only a single server will be used for job submission and retrieval. Other servers will be used as fail-over servers in case of failure of the primary server.

The servers are aware of how many logical processors there are available in the network through their connected nodes. They may also discover other servers and connect to them. These connected servers will be seen as aggregated nodes to each other, i.e. they will expose the aggregated capabilities of their connected nodes, forming a peer-to-peer network and delegating tasks amongst each other. The connections are directional, such that if a server A connects to a server B,

A will be seen as a node by B and A will see B as a client. By establishing bi-directional connections between said servers, all nodes effectively become available for all servers.

The traditional and most simple topology of a JPPF network is the star topology, having a single server with all nodes and clients connected to it as shown in figure 4.1. The client sends bundles of tasks to the server which in turn disseminates them to the available nodes. One drawback of having a star topology with one central server is that it introduces a single-point of failure and may potentially become a bottleneck. Introducing additional servers with bi-directional connections helps mitigate these problems, forming a peer-to-peer overlay between the servers.

Since the server is aware of each state of its connected nodes, different strategies and policies for load balancing and scheduling can be employed. Many of these adapt in real time, altering its parameters to workload changes. One of the most common algorithms is the *proportional* algorithm which is an adaptive heuristic algorithm based on the each nodes' contribution in the task executions. As execution progresses, the dissimation of tasks to the nodes is adjusted according to their performance. This is highly desired as maximum utilization of each node is pursued.

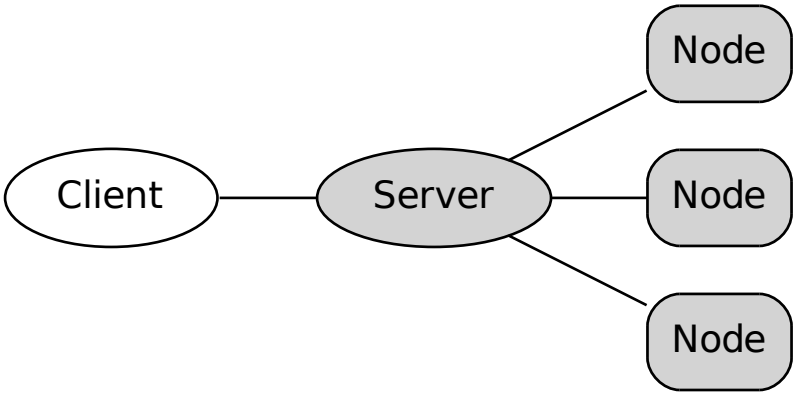

Figure 4.1. Client, server and nodes

Instead of separating the server and the node, it is possible to integrate the node into the server by enabling local execution in the server. This improves communication performance between the two components as the network overhead between them is removed. It also enables for a pure peer-to-peer overlay where all nodes are sharing the same role, with the exception of the client. An example configuration is shown in figure 4.2 where the nodes are fully connected with the client connected

to an arbitrary discovered server. If the client's connection to the server is lost, another discovered server could be selected and the execution would continue. Lost jobs would be resubmitted to the new server. This approach introduces some coordination overhead between the servers but removes the single-point of failure with having only a single server. The client would then remain as the only single-point of failure.

Given this versatile nature of the framework, it is possible to construct many fault-tolerant and creative topologies. They can be constructed in a classical master-worker topology as well as in a peer-to-peer topology, even across network boundaries.

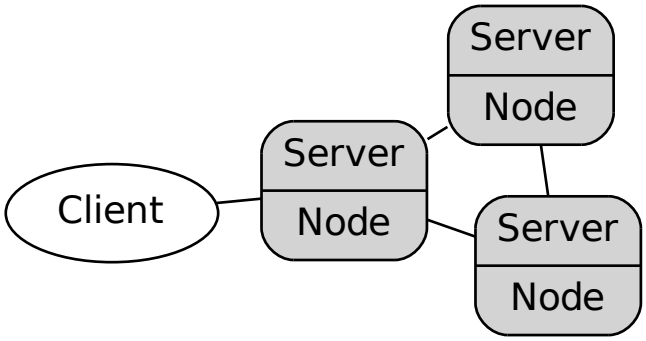

Figure 4.2. Fully connected server topology with one client

Both the server/client and peer-to-peer configurations will be evaluated.

4.2 Implementation

This section will discuss the implementation of the analysis application and the many improvements it offers over the original application. The statistical analysis is based on the GEIRA [34] paper, following the same pattern although with numerous implementation improvements compared to the R reference implementation.

4.2.1 Data Management

Unlike the original implementation which uses transposed PLINK data sets, JEIRA uses binary PLINK data sets. This dataset consist of three files: a `FAM`, `BIM` and `BED` file. The `FAM` file is identical to the `TFAM` file previously described in section 2.4. The `BIM` file shares the first four columns with the `TPED` file, containing the identifier

meta data for each SNP but without the genotype data. An additional two columns are appended, containing the nucleotide bases for the two alleles for each SNP in the data set. All genotype data has been moved and binary encoded into the `BED` file. This raw binary data is also used internally in a memory or file based data store, giving a much smaller footprint than textual or object representations. A more detailed description of the file format and an example data set containing 10 SNPs and 10 individuals can be found in appendix A.1.3. The PLINK program is used for converting the data between the different data set formats.

The data management of the application consists of three components: a library reading the various PLINK data sets, a data store holding all internal data structures, and a data set processor responsible for parsing the data set entries and populating the data store. The library is capable of reading all the different PLINK data sets, although only the processor for parsing binary data and the memory backed data store are fully implemented. Abstractions for implementing the other processors and data stores are available.

4.2.2 Introducing Concurrency

Since each pairwise comparison between the SNPs in the reference data set and the candidate SNP are independent of each other, they can naturally be executed in parallel. The statistical resampling process (discussed later in section 4.2.4) within each comparison could also be executed in parallel. However, initial experiments revealed that the synchronization overhead cost in terms of execution time for implementing concurrency at such a granular level was too expensive (data not shown), thus it was decided to implement concurrency at the SNP-comparison level. Each independent comparison will be referred to as a *task* in the analysis.

The application may use either local execution or remote execution using JPPF. This makes it possible to run the application locally on a workstation as a drop-in replacement for GEIRA, as well as in a computational network. Java's built-in concurrency APIs is used where each task is implemented as a *Callable* and executed using the *ExecutorService* API. A thread-safe queue is used as a mediator and the tasks are submitted to the executor, which in the case of local execution is a thread pool processing the queue of tasks. The size of the thread pool defaults to the number of logical processors available, but can be specified as an argument to the application. A separate consumer thread outputs the results in their natural ordering (i.e. order of submission) to an output file.

When remote execution is used, the local thread pool is replaced with a facade from the JPPF framework. All tasks submitted are grouped together into batches (default size of 20 tasks) and then submitted to a discovered JPPF server. If no servers have yet been discovered, local execution is performed to prevent the application from stalling. Servers are dynamically added as they are discovered and the workload is distributed according to the adaptive load-balancing strategy.

4.2.3 Logistic Regression

The most important part of the statistical analysis in the application is the logistic regression. None of the common, open source Java mathematical libraries provide such an implementation [10, 4]. Some libraries, e.g. *SuanShu* from Numerical Method [24], provides Java implementations of logistic regression, but due to licensing restrictions they cannot be used in this application. Therefore, it was decided to make a native implementation of the logistic regression using one of the many existing mathematical libraries supporting linear algebra. For this, Apache Commons Math [10] was chosen.

The algorithm uses the Newton-Raphson method [53] to iteratively find an approximation of the $\boldsymbol{\beta}$ vector, containing the intercept and coefficients of the predictor variables. The iterations stops when the sum of the absolute differences between the previous and new β values is less than a specified threshold (with the default being 10^{-3}), or when the maximum number of iterations has been reached without converging (default is 500). The implementation is shown as pseudo-code in algorithm 1.

Algorithm 1 Logistic Regression

Require: $\mathbf{X}_{m,n}, \mathbf{y}_n$

$\mathbf{A} \leftarrow \begin{bmatrix} \begin{bmatrix} 1 \\ \vdots \end{bmatrix}, \mathbf{X} \end{bmatrix}$
$\boldsymbol{\beta} \leftarrow [0; ...]$
$iter \leftarrow 0$
$\delta \leftarrow 1$
while $iter < max\_iter \wedge \delta > threshold$ **do**
  $\boldsymbol{\alpha} \leftarrow \boldsymbol{\beta}$
  $\mathbf{p} \leftarrow \frac{e^{\boldsymbol{\beta} \cdot \mathbf{A}}}{1 + e^{\boldsymbol{\beta} \cdot \mathbf{A}}}$
  $ll \leftarrow \sum (\mathbf{y} * \ln \mathbf{p} + (1 - \mathbf{y}) * \ln(1 - \mathbf{p}))$
  $\mathbf{s} \leftarrow \mathbf{A} \cdot (\mathbf{y} - \mathbf{p})$
  $\mathbf{J} \leftarrow (\mathbf{A}(\mathbf{p} * (1 - \mathbf{p}))) \cdot \mathbf{A}^{\mathbf{T}}$
  $\boldsymbol{\beta} \leftarrow \boldsymbol{\alpha} + \mathbf{J}^{-1} \cdot \mathbf{s}$
  $\delta \leftarrow \sum |\boldsymbol{\beta} - \boldsymbol{\alpha}|$
  $iter \leftarrow iter + 1$
end while
return $< \boldsymbol{\beta}, \mathbf{J}, ll >$

Note: the $*$ operation represents element-by-element multiplication.

4.2.4 Bootstrapping

In order to better detect possible false positive results, and get more accurate estimates of the confidence intervals, statistical resampling is performed. *Bootstrapping*

[31] is the chosen method but the default behavior is to not perform any bootstrapping at all. Instead, it is specified as an optional argument to the application. Bootstrapping is performed within each task as previously mentioned and thus operates at the SNP comparison level in terms of concurrency.

When bootstrapping is performed, a sample of the same size as the initial sample is randomly chosen with replacement amongst the individuals. This means that the same individual may appear multiple times (or none at all) in the new data set. The data set is used to perform the same computation again, giving a new set of results. This step is repeated for a specified number of times with $10,000$ being a typical number, giving an estimate of the sample distribution. Using the results from all the bootstraps, a *covariance matrix* is computed from where summary statistics, e.g. standard error, is obtained and used to provide more accurate estimates of the confidence intervals.

4.2.5 Statistical Analysis

Each SNP in the PLINK dataset is compared to the interaction variable in the second data set independently of each other. The first operation is to determine the major and minor alleles amongst the cases and controls along with their frequencies. The *minor allele frequencies* (MAFs) in the case and control groups are compared to estimate a *risk allele*. If the MAF amongst the cases is greater than or equal to the MAF amongst the controls, the minor allele is assigned as the risk allele. Otherwise, the major allele is assigned as the risk allele.

The data is then converted into a new data set of risk factors depending on which genetic model is used (i.e. dominant or recessive). The raw bi-allelic genotype data is mapped to a *genetic risk factor* for each individual as shown in table 4.1. The presence of the interaction variable is also mapped to a risk factor. Assuming a dominant genetic model for a SNP with the alleles A and G where A is assigned as the risk allele, risk would be present if any of the two alleles is an A. If a recessive genetic model is assumed, risk would only be assigned when the genotype is homozygous for AA.

Genotype	Dominant model	Recessive model
AA	Risk	Risk
AG	Risk	No risk
GG	No risk	No risk

Table 4.1. Genetic risk assignment for alleles A and G

All individuals are then divided into eight groups based on their combination of *genetic risk*, *interaction variable* and *phenotype* (affected or unaffected by disease). If any of these groups have a count below a certain threshold (default is five), the analysis for that SNP is aborted due to insufficient data and an empty result is returned. Otherwise, the risk factors are mapped into two matrices, one for the

additive model and another for the multiplicative model, where each row represents an individual. The risks are mapped to integers according to table 4.2.

Risk factor	Integer value
No risk	0
Risk	1

Table 4.2. Risk to Integer assignment

The matrix for the multiplicative model consists of three static columns. The first column denotes the presence of genetic risk, the second column denotes the presence of the interaction variable's risk factor and the third column denotes the simultaneous presence of the two factors (i.e. the product of their integer values). Using Boolean algebra where g represent the presence of genetic risk, and i represents the presence of the interaction variable, the columns in the matrix are g, i and $g \wedge i$. Any covariate variables are horizontally concatenated to the matrix. An example matrix is shown in table 4.3.

g	i	$g \wedge i$	...
1	0	0	...
0	1	0	...
1	1	1	...

Table 4.3. Example multiplicative matrix

The additive matrix consists of four static columns. These four columns are all possible combinations of the genetic risk factor and interaction variable, i.e. $\neg g \wedge \neg i$, $\neg g \wedge i$, $g \wedge \neg i$ and $g \wedge i$. As with the multiplicative matrix, all covariate variables are horizontally appended to matrix. An example matrix is shown in table 4.4.

$\neg g \wedge \neg i$	$\neg g \wedge i$	$g \wedge \neg i$	$g \wedge i$	...
1	0	0	0	...
0	0	1	0	...
0	1	0	0	...

Table 4.4. Example additive matrix

An initial logistic regression analysis is performed on both the matrices. The logistic regression model does not include the $\neg g \wedge \neg i$ column from the additive matrix as it serves as the reference group. If any of the coefficients in the additive matrix (β values) are negative, the presence of one (or both) of the risk factors indicate a protective effect rather than a risk. The previous assumptions of risk associations are then inverted for the affected factors. This is called *recoding* [46] as mentioned earlier, and the additive matrix is recalculated using the new assumptions. This does not affect the multiplicative matrix. As an example, if the coefficient for $\neg g \wedge i$ is negative, the presence of the interaction variable is instead considered to have

a protective effect. The previous risk assignment is negated and absence of the interaction variable is instead considered to be a risk.

Using the coefficients and their standard errors, odds ratios for the various categories (risk factor combinations) are calculated, both on the additive and multiplicative scale. The results from the logistic regression on the additive matrix are used to calculate two measures of biological interaction: *relative excess risk due to interaction* (RERI) and *attributable proportion due to interaction* (AP). RERI is calculated as shown in equation 4.1 and AP in equation 4.2 where RR_{11} is the relative risk of developing disease when being exposed of both the genetic risk factor and the interaction variable, RR_{01} is the relative risk when not having the genetic risk but the interaction variable, and RR_{10} is the relative risk when having the genetic risk but not the interaction variable.

The default is to calculate 95% confidence intervals (CIs) for all the ORs and the AP value using Hosmer and Lemeshow's delta method [41] as in the original analysis. However, Assman et al. showed that bootstrapping gives a better estimation of the CIs [31] than the delta method. This was omitted in the original analysis due to computational time concerns but added as a feature in this implementation. If bootstrapping is omitted, the analysis uses the delta method as in the reference application.

If bootstrapping is enabled, a new data set of the same size as the original is created by randomly sampling the original data set, with replacement. Logistic regression is then performed on the new data set and estimates are given for each bootstrap. All previously assigned parameters are kept constant, e.g. allele assessments and possible recoding. The aggregated results from the bootstrapping is then used to calculate a covariance matrix from which new confidence intervals are obtained.

$$RERI = RR_{11} - RR_{01} - RR_{10} + 1 \tag{4.1}$$

$$AP = \frac{RERI}{RR_{11}} \tag{4.2}$$

The different odds ratios, confidence intervals, allele and individual frequencies and other calculated statistics are then returned as a result set for the SNP comparison. The main application concurrently aggregates all the results into a result file as they become available through its consumer thread.

4.2.6 Execution and Node Isolation

The application can be executed as both a local stand-alone instance, or a cluster instance where JPPF nodes and servers are available. In both cases, a shell script is provided for the respective execution environments, acting as a simple launcher. The shell script for the cluster is a bit more advanced than the stand-alone script. It is used when submitting an analysis to SLURM, which bootstraps the execution by initiating both the JPPF server and nodes amongst the execution job submissions's

allocated nodes before executing the analysis application. This is performed using the tools available through SLURM. This provides fair utilization according to the configuration of the SLURM instance.

As the analysis is launched for remote execution using JPPF, all nodes connect to their discovered servers. This discovery is performed using multicasting as previously discussed. Multiple servers will also perform discovery and connect to each other, constructing an overlay topology in the logical network. In the case that the servers are operating in peer-to-peer mode with local execution enabled, a fully connected topology would eventually be constructed. If multiple simultaneous instances of the analysis program is running, the servers and nodes would connect to each other, across cluster job boundaries and would be sharing their respective resources. This would, in the case of an asymmetric resource allocation between two cluster jobs running the application (or any other JPPF enabled application), result in unfair scheduling of the tasks as the smaller allocation would utilize some of the larger allocation's resources according to the scheduling strategy used by the different components (JPPF server, client and node).

To overcome this limitation and provide isolation from other concurrently executing tasks, the wrapper script for running on the cluster generates session encryption keys for all network communication. All components generate a private/public key-pair and import the public keys into a keystore, effectively preventing external unauthorized nodes from communicating with the job allocation's nodes. This allows for fair execution of simultaneous jobs and isolates the resources from each other.

Chapter 5

Simulations and Results

In this section, the computational platform and analysis is evaluated with respect to the project's objectives and limitations.

5.1 Execution

A shell script is used in order to bootstrap the JPPF environment within the cluster environment. It can be submitted either as a batch job or run interactively. The script queries the environment variables set by SLURM to determine the number of nodes available in the job allocation. Initially, session keys are generated for all the nodes and inserted into each node's trusted key store. Then, the JPPF servers and nodes are started simultaneously, followed by the analysis application itself.

Executions were performed in various configurations with two different synthetic data sets. The first dataset contains $2,000$ individuals and 100 SNPs and is mostly used for validating the analyzis output. The second data set is a synthetic data set containing $10,000$ individuals and 100 SNPs. It was generated to simulate a medium sized study with respect to the number of individuals, giving a more realistic work load within each task. The number of SNPs remains low and does not reflect a normal sized study. Execution times for the various configurations are shown in the following sub sections.

GEIRA was executed on a single node as it cannot harness a multi-node environment. The execution time was used as reference compared to the improved implementation.

5.1.1 Server/client configuration

When running the analysis in a server/client configuration, one node is used as the *master* (JPPF server) while the remaining nodes assume the role of *workers* (JPPF nodes). Data sets are transmitted using the framework's internal communication channel through encapsulation in the tasks. Average execution times of 10 runs for the first and second data sets are shown in tables 5.1 and 5.2, respectively.

	No bootstraps	1,000 bootstraps	10,000 bootstraps	Speedup
GEIRA	0.82	-	-	-
1 node	0.13	7.95	86.98	6.31
2 nodes	0.12	5.29	52.46	6.83
4 nodes	0.11	3.69	39.17	7.45
8 nodes	0.13	1.67	14.29	6.31

Table 5.1. Master/worker execution times for 2,000 individuals and 100 SNPs.

	No bootstraps	1,000 bootstraps	10,000 bootstraps	Speedup
GEIRA	4.00	-	-	-
1 node	0.42	28.93	287.70	9.63
2 nodes	0.38	19.68	190.98	10.66
4 nodes	0.33	14.09	136.62	12.19
8 nodes	0.28	5.83	55.08	14.05

Table 5.2. Master/worker execution times for 10,000 individuals and 100 SNPs.

There is a negligible difference in execution time when performing no bootstrapping between the executions with varying number of nodes. However, there is a distinct difference in execution time compared with the reference application, especially in the latter case with the synthetic data set.

5.1.2 Peer-to-peer configuration

In a peer-to-peer configuration, both the JPPF server and node is integrated into a single component. This removes the network overhead between the two, as previously mentioned. However, it introduces further coordination overhead as each node disseminates tasks further, if possible. As with the previous executions, the data sets are encapsulated in the tasks. Average execution times of 10 runs for the first and second data sets are shown in tables 5.3 and 5.4, respectively.

	No bootstraps	1,000 bootstraps	10,000 bootstraps	Speedup
GEIRA	0.82	-	-	-
1 node	0.17	11.13	108.00	4.82
2 nodes	0.16	6.47	63.63	5.13
4 nodes	0.23	3.93	43.12	3.57
8 nodes	0.17	2.09	18.81	4.82

Table 5.3. Peer-2-peer execution times for 2,000 individuals and 100 SNPs.

As with the master/worker configuration, there is a negligible difference in execution time when performing no bootstrapping between the executions with varying number of nodes, albeit a bit higher for the peer-to-peer configuration. It also excels compared to the reference implementation.

	No bootstraps	1,000 bootstraps	10,000 bootstraps	Speedup
GEIRA	4.00	-	-	-
1 node	0.39	40.58	401.80	10.34
2 nodes	0.37	23.60	232.45	10.79
4 nodes	0.35	20.27	204.35	11.40
8 nodes	0.31	10.50	100.98	12.94

Table 5.4. Peer-2-peer execution times for 10,000 individuals and 100 SNPs.

5.2 Evaluation

Choosing Apache Commons Math as the mathematic library for the implementation of logistic regression turned out to be an expensive choice. Profiling an example execution using a modest data set (the 2,000 individuals and 100 SNPs data set) reveals that as much as 50% of the execution time is spent on matrix and vector boundary checks internally in the library (i.e. calls to `checkMatrixIndex()`, `checkRowIndex()` and `checkColumnIndex()`) as shown in table 5.5. This boundary check has already been performed by the logistic regression implementation and is thus wasted cycles. Using a different library where boundary checks are omitted could theoretically reduce the execution time of a single logistic regression by up to half compared to the current implementation.

Method	Time
`Array2DRowRealMatrix.getEntry()`	17.23%
`MatrixUtils.checkColumnIndex()`	14.50%
`Array2DRowRealMatrix.getColumnDimension()`	12.18%
`Array2DRowRealMatrix.getRowDimension()`	11.12%
`MatrixUtils.checkRowIndex()`	8.22%

Table 5.5. Profiling of a local execution

Chapter 6

Conclusions

This is an initial attempt to bridge the gap between running local analyses on the researchers' computers, and submitting them to a large scale cluster at an HPC center. A cluster consisting of a few nodes with the SLURM resource manager was deployed for enabling execution and scheduling of legacy applications, as well as future research applications. A foundation has been laid for a potentially larger cluster installation with many additional features.

There are currently very few researchers with a background in computer science within the research group, and no more than a couple of researchers are fluent in a programming language. This is a severely limiting factor as new methods and software will be required to perform more exhaustive analyses, e.g. parameter sweeping, with the increasing data sizes.

The JPPF framework used is flexible and highly configurable. It can operate in a classical master/worker configuration, a peer-to-peer configuration and a number of varying configurations depending on the topology. Using it, or a similar framework, provides noticeable performance improvements in the analysis implementation presented in this thesis.

Some of the features initially planned had to be postponed for future versions due to time constraints and other external factors. There is still room for improvements, although this project has enabled the researchers to accelerate their analyses.

Chapter 7

Future works

Due to time constraints and project limitations, some features of the analysis and cluster were not implemented. Some of those improvements are desirable and will be discussed in this chapter.

As the interest for computational resources within the organization grows, it will be necessary to better facilitate monitoring and access to the different compute nodes. By implementing priority queues and different partitions of compute nodes, inter-group resource sharing could be a viable alternative to better utilization of the available resources. This would require accounting information and possibly job preemption/cancellation to better accommodate the need for prioritized access to a research group's own resources. There are mechanisms in place for enabling quality of service (QoS) functionality but they have to be modeled and configured appropriately to meet the needs of the organization.

Cluster resource isolation between different jobs is a focus for improvement. Concurrent executions of the analysis is enabled through the use of communication encryption, although this does not prevent the local node processes from interfering with each other. Linux provides a feature enabling limitation and isolation of resource usage, called *control groups* (cgroups). SLURM has built-in compatibility for cgroups that can be used to isolate different local processes from each other. Memory, CPU and disk I/O usage can be limited to a set of processes on a local node. This, in combination with an epilogue and prologue mechanism for setting up an environment, e.g. allocating and reserving scratch space for jobs, would effectively provide a semi-jailed environment.

Instead of using encryption as means of isolating node communication, other methods could be employed. One possible approach is to negotiate non-standard port numbers used amongst an allocated set of nodes. This would have to be done in a pre-stage phase before the JPPF node instances are started. Another approach is to utilize session specific multicast groups for communication.

The analysis implemented in this thesis is a novel project and the specifications for it changed multiple times during the development phase. In retrospect, some of the design decisions made are based on premises that are no longer necessarily

true. Some possible improvements are:

- default to a standardized input data format, eliminating the need for an intermediate data store.
- implement full support for storing the data in a shared location, e.g. the distributed file system, and let each node load the necessary data for its tasks. This would decrease necessary and expensive serialization operations performed by the JPPF framework. Currently, the distributed file system is only used by other analysis tools and not this implemented analysis.
- replace the mathematical framework used by the logistic regression to eliminate the unnecessary bounds checks.

The client is currently the only single point of failure. By implementing checkpoint and rollback support in the client, fault tolerance could be further improved. At regular time intervals, or discrete events, the execution state of the client could be recorded on stable storage. This would enable the analysis to recover after a node failure where the client executed, by loading the last-known good state.

Bibliography

[1] Academic Ranking of World Universities 2012. http://www.shanghairanking.com/Institution.jsp?param=Karolinska%20Institute. Online; accessed 2012-09-15.

[2] Center for Molecular Medicine. http://www.cmm.ki.se/en/About/. Online; accessed 2012-09-16.

[3] Checkpoint/Restore In Userspace. http://criu.org/. Online; accessed 2012-11-03.

[4] Colt, Open Source Libraries for High Performance Scientific and Technical Computing in Java. http://acs.lbl.gov/software/colt/. Online; accessed 2012-10-28.

[5] Cornell University, SPMD or Manager/Worker. http://web0.tc.cornell.edu/Services/Education/Topics/Parallel/Design/SPMD.aspx. Online, accessed 2012-10-11.

[6] dbSNP FAQ. http://www.ncbi.nlm.nih.gov/books/NBK44423/#Content.according_to_dbsnp_is_it_safe_t. Online, accessed 2012-10-09.

[7] JPPF: Java Parallel Processing Framework. http://jppf.org/. Online; accessed 2012-10-28.

[8] Karolinska Institute, a medical university. http://ki.se/ki/jsp/polopoly.jsp?l=en&d=600. Online; accessed 2012-09-15.

[9] Karolinska Institutet's organization. http://ki.se/ki/jsp/polopoly.jsp?l=en&d=261. Online; accessed 2012-09-15.

[10] Math - Commons-Math: The Apache Commons Mathematics Library.

[11] MUNGE (MUNGE Uid 'N' Gid Emporium). https://code.google.com/p/munge/. Online; accessed 2012-11-04.

[12] NIST, Single Program Multiple Data. http://xlinux.nist.gov/dads/HTML/singleprogrm.html. Online, accessed 2012-10-11.

[13] Oak Ridge National Laboratory: Titan. http://www.olcf.ornl.gov/titan/. Online; accessed 2012-11-02.

[14] Oracle Virtual Box). http://virtualbox.org/. Online; accessed 2012-11-04.

[15] PDC Center for High Performance Computing. http://www.pdc.kth.se/. Online, accessed 2012-10-10.

[16] PLINK transposed filesets. http://pngu.mgh.harvard.edu/~purcell/plink/data.shtml#tr. Online; accessed 2012-08-12.

[17] RFC 4120: The Kerberos Network Authentication Service (V5). http://www.ietf.org/rfc/rfc4120.txt. Online; accessed 2012-11-03.

[18] Samba Winbind Man Page. http://www.samba.org/samba/docs/man/manpages-3/winbindd.8.html. Online; accessed 2012-10-28.

[19] Scientific Linux. https://www.scientificlinux.org/. Online; accessed 2012-11-03.

[20] SNIC-PDC: Lindgren. http://www.pdc.kth.se/resources/computers/lindgren. Online; accessed 2012-11-02.

[21] SNIC-UPPMAX: Halvan User Guide. http://www.uppmax.uu.se/halvan-user-guide. Online; accessed 2012-11-02.

[22] SNIC-UPPMAX: Kalkyl User Guide. http://www.uppmax.uu.se/kalkyl-user-guide. Online; accessed 2012-11-02.

[23] SNIC-UPPMAX: Tintin User Guide. http://www.uppmax.uu.se/tintin-user-guide. Online; accessed 2012-11-02.

[24] SuanShu, math library of numerical methods for numerical analysis. http://numericalmethod.com/blog/suanshu/. Online; accessed 2012-10-28.

[25] Swedish National Infrastructure for Computing, About. http://www.snic.vr.se/about-snic. Online, accessed 2012-10-10.

[26] Uppsala Multidisciplinary Center for Advanced Computational Science. http://www.uppmax.uu.se/. Online, accessed 2012-10-10.

[27] Vetenskapsrådet, About. http://vr.se/omvetenskapsradet.html. Online, accessed 2012-10-10.

[28] A. Ahlbom and L. Alfredsson. Interaction: A word with two meanings creates confusion. *Eur. J. Epidemiol.*, 20(7):563–564, 2005.

[29] P. Almgren, P.P. Bendahl, H. Bengtsson, O. Hössjer, and R. Perfekt. Statistics in genetics, lecture notes. http://www1.maths.lth.se/matstat/kurser/statgen/book/StatisticsInGenetics-20031125.pdf.

[30] T. Andersson, L. Alfredsson, H. Kallberg, S. Zdravkovic, and A. Ahlbom. Calculating measures of biological interaction. *Eur. J. Epidemiol.*, 20(7):575–579, 2005.

[31] S. F. Assmann, D. W. Hosmer, S. Lemeshow, and K. A. Mundt. Confidence intervals for measures of interaction. *Epidemiology*, 7(3):286–290, May 1996.

[32] Brian L. Browning and Sharon R. Browning. A fast, powerful method for detecting identity by descent. *American journal of human genetics*, 88(2):173–82, February 2011.

[33] K. Dowd C. Severance.

[34] B. Ding, H. Källberg, L. Klareskog, L. Padyukov, and L. Alfredsson. GEIRA: gene-environment and gene-gene interaction research application. *Eur. J. Epidemiol.*, 26(7):557–561, Jul 2011.

[35] F. Dudbridge. Likelihood-based association analysis for nuclear families and unrelated subjects with missing genotype data. *Human Heredity*, 66(2):87–98, 2008.

[36] F. Ariani et al. Revealing the complex nature of a monogenic disease: exome sequencing of rett syndrome. 2012.

[37] Michael J. Flynn and Kevin W. Rudd. Parallel architectures. *ACM Comput. Surv.*, 28(1):67–70, March 1996.

[38] G. Gibson. Hints of hidden heritability in GWAS. *Nat. Genet.*, 42(7):558–560, Jul 2010.

[39] A. Gusev, J. K. Lowe, M. Stoffel, M. J. Daly, D. Altshuler, J. L. Breslow, J. M. Friedman, and I. Pe'er. Whole population, genome-wide mapping of hidden relatedness. *Genome Res.*, 19(2):318–326, Feb 2009.

[40] M. Hautreux.

[41] D. W. Hosmer and S. Lemeshow. Confidence interval estimation of interaction. *Epidemiology*, 3(5):452–456, Sep 1992.

[42] David J. Hunter. Gene-environment interactions in human diseases. *Nature Reviews Genetics*, 6(4):287 – 298, 2005.

[43] Joshua Hursey, Jeffrey M. Squyres, and Andrew Lumsdaine. A checkpoint and restart service specification for open mpi. Technical Report TR635, Indiana University, Bloomington, Indiana, USA, July 2006.

[44] P. Jennum, B. Wanscher, J. Frederiksen, and J. Kjellberg. The socioeconomic consequences of multiple sclerosis: a controlled national study. *Eur Neuropsychopharmacol*, 22(1):36–43, Jan 2012.

[45] Morris A. Jette, Andy B. Yoo, and Mark Grondona. Slurm: Simple linux utility for resource management. In *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*, pages 44–60. Springer-Verlag, 2002.

[46] M. J. Knol and T. J. VanderWeele. Recoding preventive exposures to get valid measures of interaction on an additive scale. *Eur. J. Epidemiol.*, 26(10):825–826, Oct 2011.

[47] Miron Livny, Jim Basney, Rajesh Raman, and Todd Tannenbaum. Mechanisms for high throughput computing. *SPEEDUP Journal*, 11(1), June 1997.

[48] T. Masterman, A. Ligers, T. Olsson, M. Andersson, O. Olerup, and J. Hillert. HLA-DR15 is associated with lower age at onset in multiple sclerosis. *Ann. Neurol.*, 48(2):211–219, Aug 2000.

[49] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 53–65, London, UK, UK, 2002. Springer-Verlag.

[50] Gregor Mendel and William Bateson. *Experiments in plant-hybridisation /.* Cambridge, Mass. :Harvard University Press,, 1925. http://www.biodiversitylibrary.org/bibliography/4532.

[51] Andy Oram, editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001.

[52] Helen Pearson. Genetics: What is a gene? *Nature*, 441:398 – 401, 2006.

[53] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing.* Cambridge University Press, New York, NY, USA, 3 edition, 2007.

[54] Alkes L et al. Price. Principal components analysis corrects for stratification in genome-wide association studies. *Nature Genetics*, 38(8):904–909, Aug 2006.

[55] E. Quelvennec, O. Bera, P. Cabre, M. Alizadeh, D. Smadja, F. Jugde, G. Edan, and G. Semana. Genetic and functional studies in multiple sclerosis patients from Martinique attest for a specific and direct role of the HLA-DR locus in the syndrome. *Tissue Antigens*, 61(2):166–171, Feb 2003.

[56] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2012. ISBN 3-900051-07-0.

[57] Ioan Raicu, Ian T. Foster, and Yong Zhao. Many-task computing for grids and supercomputers. In *IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08) 2008.*

[58] K.J. Rothman. *Epidemiology:An Introduction*. Oxford University Press, USA, 2002.

[59] K.J. Rothman, S. Greenland, and T.L. Lash. *Modern Epidemiology*. Lippincott Williams & Wilkins, 2008.

[60] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *IN: MIDDLEWARE*, pages 329–350, 2001.

[61] S. et al. Sawcer. Genetic risk and a primary role for cell-mediated immune mechanisms in multiple sclerosis. *Nature*, 476(7359):214–219, Aug 2011.

[62] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI-The Complete Reference, Volume 1: The MPI Core*. MIT Press, Cambridge, MA, USA, 2nd. (revised) edition, 1998.

[63] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, August 2001.

[64] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[65] Spyros Voulgaris, Daniela Gavidia, and Maarten Van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13:2005, 2005.

[66] Jonathan P. Weeks. plink: An R package for linking mixed-format tests using irt-based methods. *Journal of Statistical Software*, 35(12):1–33, 2010.

[67] J. Yang, B. Benyamin, B. P. McEvoy, S. Gordon, A. K. Henders, D. R. Nyholt, P. A. Madden, A. C. Heath, N. G. Martin, G. W. Montgomery, M. E. Goddard, and P. M. Visscher. Common SNPs explain a large proportion of the heritability for human height. *Nat. Genet.*, 42(7):565–569, Jul 2010.

Appendix A

File Formats

A.1 Example Data Set

The example data sets contains 10 individuals from different families and 10 SNPs with the alleles `A` and `G`.

A.1.1 Interaction Variable File

The interaction variable file contains a minimum of two columns: the individual's ID and the interaction variable. Optional columns can be added for covariates. The first line in the file is a header with column labels. An example file with one covariate (labeled *cov*), an interaction variable (*env*) and individual ID (*indid*) is given in figure A.1.1.

```
cov env indid
1   1   ind0
2   0   ind1
0   1   ind2
0   1   ind3
1   1   ind4
1   0   ind5
0   0   ind6
1   1   ind7
1   0   ind8
0   0   ind9
```

Figure A.1. Example Interaction Variable File

A.1.2 Transposed PLINK Data Set

The transposed PLINK data sets consist of two whitespace delimited text files, a `TPED` and `TFAM` file. Each line in the `TPED` file describes a single SNP and contains four static columns followed by a variable number of pair columns for the genotype data. An example file is shown in figure A.1.2. The static columns are:

Chromosome pair number with valid values being integers ranging from $1-22$, X, Y or 0 if unplaced

Identifier being an *rs* number or SNP identifier

Genetic distance measured in *morgans*

Base-pair position

```
1 rs0 0  1 A A A G G G G A G G G A G G G A G A G
1 rs1 0  2 A A A A A A A A G A A A A A A A A G G
1 rs2 0  3 G G G G G G A G G G G G G G A G G G G
1 rs3 0  4 A A A G G G A G A G A A G G A A A G G G
1 rs4 0  5 G A A A A A A A G A A A A A A A A A A
1 rs5 0  6 G G G A A A A A A A G A A A G A G A G A
1 rs6 0  7 A A G A G A A A G A A A G A G A G A G A
1 rs7 0  8 A A A A A A A A A A G A A A G A A A G A
1 rs8 0  9 G A G G A A A A G A G A A A G A G A A A
1 rs9 0 10 G A G A G A G G A A G A A A G A A A A A
```

Figure A.2. Example PLINK TPED file

Each line in the `TFAM` file describes a single individual and always contains six columns: *family ID*, *individual ID*, *paternal ID*, *maternal ID*, *sex* (1 for male, 2 for female, other values for unknown) and *phenotype*. In the analysis, phenotype can is the same as affection status. An example file is given in figure A.1.2.

A.1.3 Binary PLINK Data Set

The binary PLINK data set consists of two whitespace separated text files, a `BIM` and `FAM` file , and one binary data file, a `BED` file. The `BIM` file contains the static four columns shown in the `TPED` file from the transposed data set, with an additional two columns: the names of the two alleles as shown in figure A.1.3. The `FAM` file is identical to the `TFAM` file (see figure A.1.2).

The `BED` file is a binary encoded file containing all the genotypes for all individuals and markers. The file contains a three byte header with the first two bytes being the file's magic number (0x6C 0x1B) and the third byte being the file's *mode*. The mode can either be SNP major (0x1, all individuals for each SNP) or individual

```
fam0 ind0 0 0 2 2
fam1 ind1 0 0 2 2
fam2 ind2 0 0 2 2
fam3 ind3 0 0 2 2
fam4 ind4 0 0 2 1
fam5 ind5 0 0 2 2
fam6 ind6 0 0 2 1
fam7 ind7 0 0 2 1
fam8 ind8 0 0 2 1
fam9 ind9 0 0 2 1
```

Figure A.3. Example PLINK TFAM file

```
1 rs0 0  1 A G
1 rs1 0  2 G A
1 rs2 0  3 A G
1 rs3 0  4 A G
1 rs4 0  5 G A
1 rs5 0  6 G A
1 rs6 0  7 G A
1 rs7 0  8 G A
1 rs8 0  9 G A
1 rs9 0 10 G A
```

Figure A.4. Example PLINK BIM file

major (0x0, all SNPs for each individual). Default is SNP major. All genotype data is coded as two bits, following the header and is *least significant bit* ordered. The conversion is shown in table A.1 and an example file is given in figure A.1.3. A new entry (e.g. SNP) always begin with a new byte, excess bits from a previous SNP are discarded.

Binary code	Genotype
00	Homozygote for Allele 1
10	Heterozygote
11	Homozygote for Allele 2
01	Missing

Table A.1. Binary genotype coding

```
0000000: 01101100 00011011 00000001 11111000 11101110 00001010  l.....
0000006: 11111111 11111110 00000011 10111111 10111111 00001111  ......
000000c: 10111000 00110010 00001110 11111110 11111110 00001111  .2....
0000012: 11111000 10111011 00001010 11101011 10101110 00001010  ......
0000018: 11111111 10111011 00001011 11110010 10111010 00001110  ......
000001e: 00101010 10111011 00001111                             *..
```

Figure A.5. Example PLINK BED file