Speeding up Genome-Wide Association Analyses Applying Parallel Computing

Unitsa Sangket

A Thesis Submitted in Fulfillment of the Requirements
for the Degree of Doctor of Philosophy in
Molecular Biology and Bioinformatics
Prince of Songkla University
2011

Thesis Title	Speeding up Genome-Wide Association Analyses Applying Parallel Computing
Author	Miss Unitsa Sangket
Major Program	Molecular Biology and Bioinformatics

Major Advisor

.....

(Asst. Prof. Dr. Pichaya Tandayya)

Co-advisors

.....

(Assoc. Prof. Dr. Wasun Chantratita)

.....

(Dr. Surakameth Mahasirimongkol)

Examining Committee:

.....Chairperson

(Prof. Dr. Amornrat Phongdara)

.....

(Prof. Dr. Yutaka Yasui)

.....

(Assoc. Prof. Dr. Wasun Chantratita)

.....

(Asst. Prof. Dr. Pichaya Tandayya)

.....

(Dr. Surakameth Mahasirimongkol)

The Graduate School, Prince of Songkla University, has approved this thesis as fulfillment of the requirements for the Doctor of Philosophy Degree in Molecular Biology and Bioinformatics.

.....

(Prof. Dr. Amornrat Phongdara)

Dean of Graduate School

ชื่อวิทยานิพนธ์	การเพิ่มความเร็วในการวิเคราะห์ความสัมพันธ์ทั่วทั้งจีโนมโดยการประยุกต์ใช้การประมวลผลแบบขนาน
ผู้เขียน	นางสาวอุนิตษา สังข์เกตุ
สาขาวิชา	ชีววิทยาโมเลกุลและชีวสารเทศ
ปีการศึกษา	2554

บทคัดย่อ

การศึกษาความสัมพันธ์ทั่วทั้งจีโนม (Genome-Wide Association Study: GWAS) เป็นวิธีการที่มีประสิทธิภาพสำหรับระบุตำแหน่งของยีนที่มีส่วนเกี่ยวข้องกับการเปลี่ยนแปลงของลักษณะทางพันธุกรรมที่ซับซ้อน ซึ่งอาจมีผลทำให้เกิดโรค การวิเคราะห์เพื่อหาค่าสถิติสำหรับการศึกษา GWA โดยใช้กลุ่มตัวอย่างและกลุ่มควบคุมสามารถทำได้โดยใช้ไลบรารี GenABEL และไลบรารี LogicReg ซึ่งเป็นไลบรารีที่ทำงานอยู่ภายใต้โปรแกรม R อย่างไรก็ตามในการวิเคราะห์เพื่อหาค่าสถิติในการศึกษา GWA จากข้อมูลที่มีขนาดใหญ่จะต้องใช้ระยะเวลาในการประมวลผลนาน ทั้งนี้อาจจะใช้เวลานานเป็นชั่วโมง สัปดาห์ หรือเดือน

การประมวลผลแบบขนานเป็นวิธีการที่ใช้ได้ผลดีในการเพิ่มประสิทธิภาพในการประมวลผล และสามารถนำมาประยุกต์ใช้กับการประมวลผลเพื่อศึกษา GWA ในแต่ละขั้นตอนได้ เนื่องจากขั้นตอนเหล่านั้นมีการประมวลผลข้อมูลที่อิสระจากกัน นอกจากนี้ไลบรารี Rmpi เป็นไลบรารีที่ทำงานภายใต้โปรแกรม R ได้เก็บรวบรวมฟังก์ชันต่างๆสำหรับการประมวลผลแบบขนาน โดยใช้ MPI (Message-Passing Interface) เป็นตัวกลางในการสื่อสารระหว่างเครื่องแม่ข่ายกับเครื่องลูกข่าย แต่ผู้ใช้อาจนำไลบรารี Rmpi มาประยุกต์ใช้ในการวิเคราะห์ GWAได้ยากลำบาก เนื่องจากผู้ใช้จำเป็นต้องมีความรู้ทางด้านการเขียนโปรแกรมแบบขนานขั้นสูง เพื่อแบ่งข้อมูล กระจายข้อมูล ควบคุมงาน และสังเกตการณ์งานระหว่างหน่วยประมวลผลกลางหรือคอมพิวเตอร์ และสุดท้ายรวบรวมผลลัพธ์

ในวิทยานิพนธ์นี้ได้นำเสนอไลบรารี ParallABEL และไลบรารี ParallLogicReg เพื่อเพิ่มประสิทธิภาพในการวิเคราะห์ GWA โดยการประยุกต์ใช้การประมวลผลแบบขนาน การวิเคราะห์ทางสถิติบนไลบรารี ParallABEL และไลบรารี ParallLogicReg ได้ถูกดัดแปลงมาจากไลบรารี GenABEL และไลบรารี LogicReg ตามลำดับ องค์ประกอบในการวิเคราะห์ GWA ของไลบรารี ParallABEL สามารถถูกแบ่งเป็นส่วนย่อยๆ เท่ากันได้ ทั้งนี้ขึ้นอยู่กับประเภทของข้อมูลเข้า

(ตาม SNP และตามรายบุคคล) และลักษณะผลลัพธ์ค่าสถิติที่ต้องการ ขณะที่ข้อมูลเข้าของ ParallLogicReg จะถูกแบ่งเป็นส่วนย่อยตามจำนวนยีนที่นำมาวิเคราะห์

ข้อมูลจากสมาคมโรคไขข้ออักเสบอเมริกาเหนือประกอบด้วยจำนวนตัวอย่าง 2,062 คน และแต่ละตัวอย่างมีจำนวน SNP 545,080 ตำแหน่ง ได้ถูกนำมาใช้เพื่อวัดประสิทธิภาพของไลบรารี ParallABEL ผลปรากฏว่าไลบรารี ParallABEL ประมวลผลข้อมูลได้อย่างรวดเร็วขึ้นมาก ยกตัวอย่างเช่นไลบรารี ParallABEL สามารถลดระยะเวลาในการประมวลผลข้อมูลเพื่อหา identity-by-state จากเดิมใช้เวลาประมาณ 8 ชั่วโมงเหลือเพียงประมาณ 1 ชั่วโมงเท่านั้นเมื่อใช้หน่วยประมวลผลกลาง 8 ตัว ชุดข้อมูลโรคลำไส้อักเสบจากสมาคม WTCCC ซึ่งประกอบด้วยจำนวนตัวอย่าง 4,680 ตัวอย่างและแต่ละตัวอย่างมีจำนวน SNP ประมาณ 2,000 ตัว ถูกนำมาทดลองเพื่อวัดประสิทธิภาพของไลบรารี ParallLogicReg ผลการทดสอบปรากฏว่าไลบรารี ParallLogicReg สามารถลดระยะเวลาในการประมวลผลยีน 200 ตัวด้วยการเรียงสับเปลี่ยน 20 รอบจากเดิมใช้เวลาประมาณ 7.3 วันเหลือเพียงประมาณ 0.9 วัน เมื่อใช้หน่วยประมวลผลกลาง 8 ตัว ทั้งนี้ผลลัพธ์จากไลบรารี ParallABEL และไลบรารี ParallLogicReg มีความถูกต้องเช่นเดียวกับค่าผลลัพธ์จากไลบรารี GenABEL และไลบรารี LogicReg เนื่องจากข้อมูลที่นำมาวิเคราะห์เพื่อศึกษา GWA มีความเป็นอิสระจากกัน

การประมวลผลข้อมูลทางด้าน GWA โดยใช้ไลบรารี ParallABEL และไลบรารี ParallLogicReg บนเครื่องคอมพิวเตอร์คลัสเตอร์หรือเครื่องคอมพิวเตอร์ที่มีหน่วยประมวลผลกลางหลายแกนหรือหลายตัวเป็นวิธีการที่มีประสิทธิภาพ เนื่องจากสามารถเพิ่มความเร็วในการประมวลผลและผู้ใช้สามารถใช้งานได้ง่าย โดยถือได้ว่าไลบรารี ParallABEL และไลบรารี ParallLogicReg เป็นไลบรารี GenABEL และไลบรารี LogicReg เวอร์ชันที่รองรับการประมวลผลแบบขนานที่เป็นมิตรกับผู้ใช้ ยิ่งไปกว่านั้นไลบรารีทั้งสองยังสามารถประมวลผลข้อมูลในโรคอื่นๆ ได้อีกด้วย เช่น ข้อมูลโรคมะเร็งที่ลำคอ เพื่อค้นหายีนที่มีส่วนเกี่ยวข้องกับการเกิดโรคนั้นๆ ซึ่งในที่นี้คือโรคมะเร็งที่ลำคอ เป็นต้น

Thesis Title	Speeding up Genome-Wide Association Analyses Applying Parallel Computing
Author	Miss Unitsa Sangket
Major Program	Molecular Biology and Bioinformatics
Academic Year	2011

ABSTRACT

Genome-Wide Association Study (GWAS) is a powerful method for identifying loci associated with variations of complex genetic traits such as common diseases. Statistical analyses for GWAS with both case and control participants can be processed by GenABEL and LogicReg libraries implemented in R. Nevertheless, statistical analysis of very large data sets is computationally challenging and may take hours, weeks or months to complete.

Parallel computing is an intuitive and effective method for increasing computational throughput. Most tasks solved in GWA analysis are suitable for parallelization, due to their computational independency and with parallelization achieved at the data level. In addition, Rmpi [14] is an R library which provides various functions to parallelize tasks in R using MPI. However, it is very difficult and complicated for users to apply a parallel computing library such as Rmpi to conduct statistical analyses of GWA studies because they need advanced programming skills to correctly partition and distribute data, control and monitor tasks across the computers and finally merge outputs.

In this thesis, ParallABEL and ParallLogicReg, the novel R libraries, were presented to boost performance of GWA analyses applying parallel computing based on Rmpi. Statistical analyses of the ParallABEL and ParallLogicReg are adapted based on GenABEL and LogicReg, respectively. In ParallABEL, most components of GWA analysis can be equally divided into subsets depend on the types of input data (SNPs and individuals) and statistical outputs, while the input data of ParallLogicReg is partitioned into *G* subsets (where *G* is the number of genes to be analyzed).

The data set from the North American Rheumatoid Arthritis Consortium (NARAC) includes 2,062 individuals with 545,080, SNPs' genotyping, was used to measure the ParallABEL performance. Almost perfect speed-up was achieved for many types of analyses. For example, the computing time for the identity-by-state matrix was linearly reduced from approximately eight hours to one hour when ParallABEL employed eight processors. The Crohn's disease GWA study dataset from the Wellcome Trust Case Control Consortium (WTCCC) that includes 4,680 individuals with 2,000 SNPs' genotypes was analyzed using logic regression on a computer cluster to evaluate the ParallLogicReg performance. The ParallLogicReg library also accelerated the logic regression analysis perfectly. For instance, with two hundred genes and twenty permutation rounds, the computing time was continuously decreased from 7.3 days to only 0.9 day when ParallLogicReg applied eight CPUs. The statistical outputs from ParallABEL and ParallLogicReg with any number of CPUs are as valid as those from GenABEL and LogicReg with one CPU because of their computational independency of GWA analyses at the data level.

Executing genome-wide association analysis using the ParallABEL and ParallLogicReg library on a computer cluster or a computer with multi-core CPUs is effective way to boost the performance and to simplify the parallelization of GWA studies. ParallABEL and ParallLogicReg are the user-friendly parallelization versions of GenABEL and LogicReg respectively. Moreover, ParallABEL and ParallLogicReg also can process other disease data sets such as a neck cancer data set to find genes associated the diseases such as the neck cancer.

ACKNOWLEDGEMENTS

This thesis would not complete without helps of many people, whom I would like to thank.

I would like to express my sincere gratitude and deep appreciation to my advisors Asst. Prof. Dr. Pichaya Tandayya, Assoc. Prof. Dr. Wasun Chantratita and Dr. Surakameth Mahasirimongkol for their guidance, unmeasurable advice, understanding and supports throughout this work.

I am very grateful to Prof. Dr. Yutaka Yasui for giving me a chance to learn and create a part of my research at the Department of Public Health Sciences, School of Public Health, University of Alberta, Canada.

I am thankful to Prof. Dr. Amornrat Phongdara and Assoc. Prof. Dr. Wilaiwan Chotigeat for establishing the PSU research group in Bioinformatics and supporting my research.

Many thanks go to Dr. Yurii S Aulchenko, Department of Epidemiology, Erasmus MC Rotterdam, The Netherlands and Qi Liu, Department of Public Health Sciences, School of Public Health, University of Alberta, Canada for their collaborating and helps in this research.

I am indebted to the program for Strategic Scholarships for Frontier Research Network for the Joint Ph.D. Program Thai Doctoral degree from the Office of the Higher Education Commission and the lecturer scholarship from Prince of Songkla University.

I would like to thank examination committee of this thesis for their invaluable advice and for taking time to review my thesis. I also thank the Thai National Grid Center and Prince of Songkla University Grid Center for providing the computer clusters used in this research.

Special thanks to all members of Thai Students Association at the University of Alberta (TSA) and all loving friends in Canada and Thailand for their kind helps, encouragement and friendship.

Finally, I dedicate this work to my beloved father and mother for giving me their love and help me overcome difficulties and pains. Without their encouragement and supports, it would not have been possible for me to come up to this stage.

Unitsa Sangket

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

ABBREVIATIONS AND SYMBOLS

GWA	=	Genome-wide association
GWAS	=	Genome-wide association studies
WGA	=	Whole genome association
WGAS	=	Whole genome association studies
SNPs	=	Single-nucleotide polymorphisms
DNA	=	Deoxyribonucleic acid
CPUs	=	Central Processing Units
MPI	=	Message-passing interface
WTCCC	=	Wellcome Trust Case Control Consortium
NARAC	=	The Rheumatoid dataset form the North American Rheumatoid Arthritis Consortium
SISD	=	Single instruction, single data
SIMD	=	Single instruction, multiple data
MISD	=	Multilple instruction, single data
MIMD	=	Multilple instruction, multiple data
PCs	=	Personal Computers
UMA	=	Uniform memory access
NUMA	=	Non-uniform memory access
CC-UMA	=	Cache coherent UMA
SMP	=	Symmetric multiprocessor
GPUs	=	Graphics processing units
SPMD	=	Single program multiple data
MPMD	=	Multiple program multiple data
API	=	Application programming interface
BLAST	=	Basic local alignment search tool
HWE	=	Hardy-weinberg equilibrium
LAM/MPI	=	Local area multicomputer/message passing interface
CART	=	Classification and regression trees
SVMs	=	Support Vector Machines

ABBREVIATIONS AND SYMBOLS (Continued)

L	=	A logic expression
G	=	Gene
500K	=	500,000

CHAPTER 1

General Introduction

This chapter presents an overview of this thesis. It starts with the background and rationale of genome-wide association (GWA) studies using parallel computing. It also introduces the purposes, the scopes, the benefits and the summaries of this project.

1.1 Background and rationale

In genetic epidemiology, genome-wide association studies (GWA studies, or GWAS), also called as whole genome association studies (WGA studies, or WGAS) are comparisons of the genomes of distinct individuals in a particular species to find variations of genes among individuals. Different variations can be associated with different traits, such as diseases. Researchers can use the information to develop better strategies to detect, treat and prevent the diseases. In addition, in the near future, if there are low cost and high efficiency genome-wide scans and other novel technologies, health experts can apply the tools to determine from individualized patients information whether there are possible hazards of causing certain diseases. Also, when a patient becomes sick, the information can be used to find the most effecient treatments with the least like probably to develop adverse reactions for that particular patient [1].

GWA analyses succeed to conduct the research discovery of associations of specific genes with diseases such as coronary heart disease, diabetes, rheumatoid, Chrohn's disease, biolara disorder and hypertension [2-3]. The genomic discoveries of complex and non-Mendelian diseases are growing, and more than one hundread loci for as many as fourty common diseases are powerfully determined and replicated by GWA studies. The hundreds of thousands of the common forms of genetic variants or single-nucleotide polymorphisms (SNPs) are assayed by high throughput genotyping technologies and refered to diseases or health-related traits [4].

In the National Center for Biotechnology Information's dbSNP database, closely twelve million unique human SNPs have been coded a reference SNP (rs) number [5] and marked as specific alleles (an alternate form of the SNPs). Also, summary allele frequencies and other genomic information can be calculated from the human SNPs [6]. In 118 articles, 56,411 significant SNPs related to diseases are found [7]. The GWA method allows inquiry of the entire human genome at levels of solving previously unachievable, in thousands of unrelated individuals, unconstrained by prior hypotheses regarding genetic associations with diseases [8].

The conventional GWA study has 4 processes: (1) selection of a huge number of individuals with the disorder or trait of interest and an eligible comparison category; (2) Deoxyribonucleic acid (DNA) isolation, genotyping, and data checking; (3) statistical analyses for associations between the SNPs passing suitable thresholds and the disorder or trait; and (4) replication of identified associations [9]. In the processes, case-control design has been often used to create GWA studies. In this method, allele frequencies in patients with the disorder of interest are compared to those in participants with disorder-free of interest. Case-control studies are frequently easier and less expensive to create than studies applying other designs such as cohort and trio designs [9]. Statistical analyses for GWA studies with both case and control participants can be processed by Bioinformatic tools including GenABEL and LogicReg. GenABEL is a specialized library package for GWA analysis of quantitative, binary and time-till-event traits to find associations between the SNPs [10]. GenABEL has been implemented in R, an open source statistics programming language and environment [11-12]. LogicReg is a famous R library for logic regression analyses [13] and can be applied to various regression/classification problems, one of which is the analysis of SNP interactions with each gene related to diseases. Nevertheless, statistical analysis of very large data sets is computationally challenging and may take hours, weeks or months to complete. Examples include the utilization of sophisticated adjustments for population stratification and relationship structures, the estimation of linkage disequilibriums and the calculation of genome-wide identity-by-state, haplotypic tests, permutation analyses and deviance of logic regression analyses.

Parallel computing is an intuitive and powerful method for increasing computational throughput. A task is separated into smaller tasks, and each is processed independently, in parallel, using multiple Central Processing Units (CPUs) or a cluster of computers. The outputs from each task must later be merged [14]. Most tasks solved in GWA analysis are suitable for parallelization, due to their computational independency, with parallelization achieved at the data level. For example, association tests can usually be done separately for each SNP and/or a small group of SNPs. Consequently, parallelization is a beneficial way to reduce the computing time, with few overheads incurved in large-scale GWA analyses. In addition, Rmpi [15] is an R library which provides various functions to parallelize tasks in R using the message-passing interface (MPI) [16]. Rmpi employs various functions to manage flow analysis in parallel environment, and is applicable for employing not only multi-core CPUs on a single computer but also multi-core CPUs distributed across many computers. However, it is very difficult and complicated for users to apply a parallel computing library such an Rmpi to statistical analyses of GWA studies because they need advanced programming skills to correctly partition and distribute data, control and monitor tasks across the computers, and merge outputs. For example, the analyses will be failed, if the users mistakenly partition the large data. Another example is that the outputs from the computers are usually messy and their order may be hard to follow.

1.2 Objectives

1.2.1 To propose the design of novel methods to speed up the computation of large-scale GWA analyses with valid statistical outputs.

1.2.2 To present development of novel R libraries, which are as easy-to-use as the more conventional GenABEL and LogicReg, based on the novel methods to accelerate the computation of large-scale GWA analyses with effective statistical outputs.

1.3 Scopes

1.3.1 Parallel computing is applied to the novel libraries to accelerate the computing time of large-scale GWA.

1.3.2 Statistical analyses of the novel libraries are adapted based on GenABEL and LogicReg

1.3.3 Rmpi is applied to parallelize statistical functions of novel libraries.

1.3.4 The novel libraries require Rmpi, GenABEL and LogicReg for data analyses.

1.3.5 The Crohn's disease GWAS dataset from the Wellcome Trust Case Control Consortium (WTCCC) [17] and The Rheumatoid dataset form the North American Rheumatoid Arthritis Consortium (NARAC) [18] are used to measure the performance of the novel libraries.

1.4 Benefits

1.4.1 The novel methods implemented in the novel libraries can speed up GWAS computing using parallel computing.

1.4.2 The novel libraries can be used to boost the performance of GWA analyses and are user-friendly libraries like the other famous R libraries.

1.4.3 The user can use statistical outputs from the novel libraries to quickly find genes associated to various diseases

1.5 Summary

This thesis presents ParallABEL and ParallLogicReg to boost performance of GWA analyses applying parallel computing. Both novel libraries can be executed on not only multi-core CPUs on a single computer but also multi-core CPUs or single-core CPU distributed across many computers (a computer cluster). ParallABEL is a user-friendly parallelization of GenABEL whereas ParallLogicReg is a user-friendly parallelization of LogicReg. With ParallABEL and ParallLogicReg libraries, the users can immensely accelerate the computing time of GWA analyses.

Nonetheless, they can easily execute ParallABEL and ParallLogicReg, since they do not need to be programming experts in parallel computing which concerns partitioning and distributing data, controling and monitoring tasks, and merging output files**.** Moreover, the statistical outputs from ParallABEL and ParallLogicReg with any number of CPUs are as valid as those from GenABEL and LogicReg due to their computational independency of GWA analyses at the data level.

CHAPTER 2

Parallel Computing

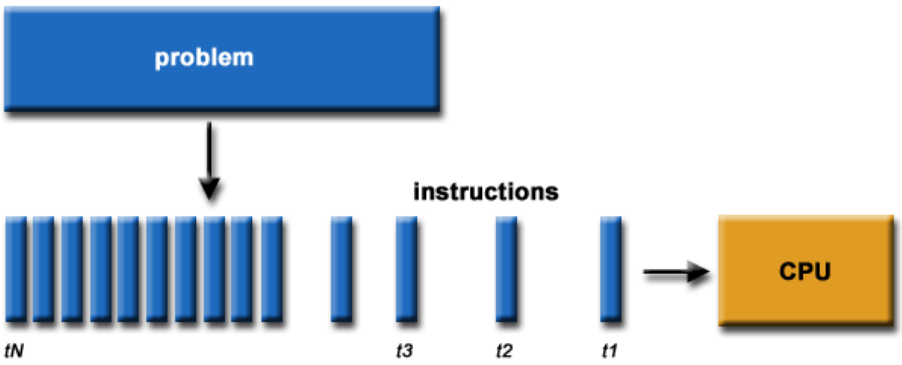
This chapter presents overview of parallel computing. There are five sections including overview, types of parallel computing, parallel computer memory architectures, parallel programming models and parallel program design [19].

2.1 Overview

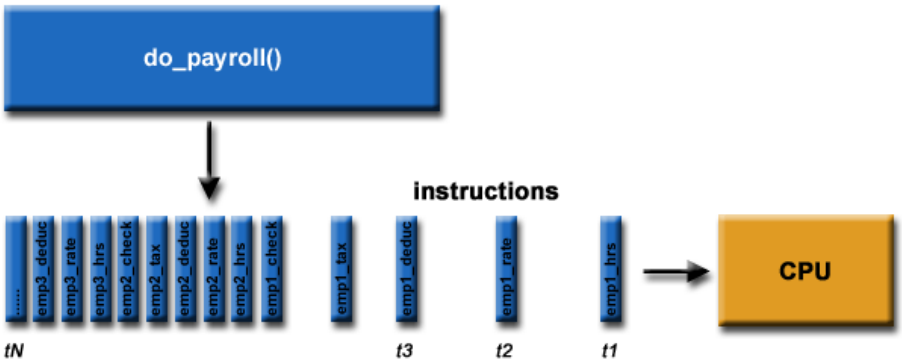
2.1.1 What is parallel computing?

Basically, a computer program is coded for sequential computing to be executed on a single computer having a single central processing unit (CPU). A problem or a job is divided into a discrete series of commands. Then, the commands will be running one after another. Figure 2.1 shows the concept and an example of sequential computing [19].

Nowadays, many programs need more computational power than conventional sequential computing can offer. Consequently, parallel computing has been developed to speed up the computational power by growing the number of CPUs in a computer or a computer cluster. Parallel computing is a useful methodology, enabling the concurrent handling of multiple computing resources to gain computational throughput. In parallel computing, a problem or a job will be divided into unassociated smaller tasks including series of commands. Each task will then be executed freely using multiple or multi-core CPUs on a computer or a computer cluster; after that, one of these CPUs will combine the outputs from all tasks. Figure 2.2 shows the concept and an example of parallel computing [19].

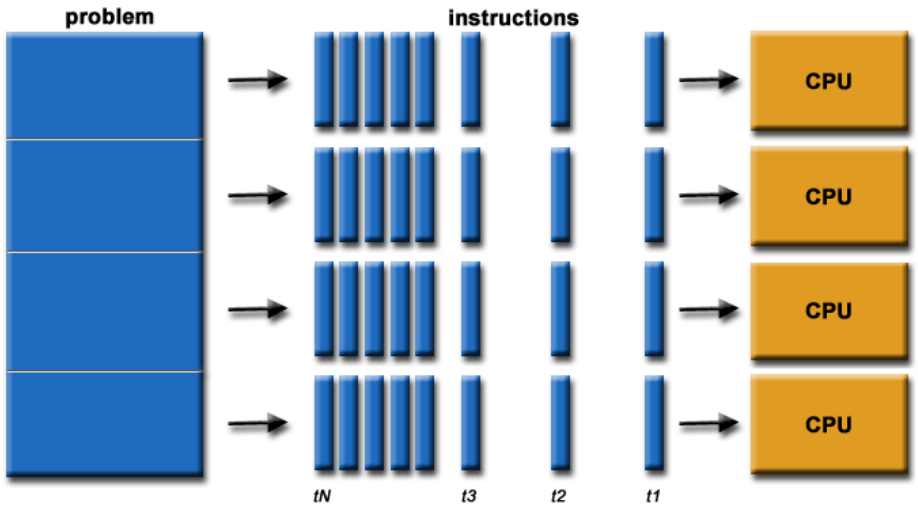


a) The concept of sequential computing [19]

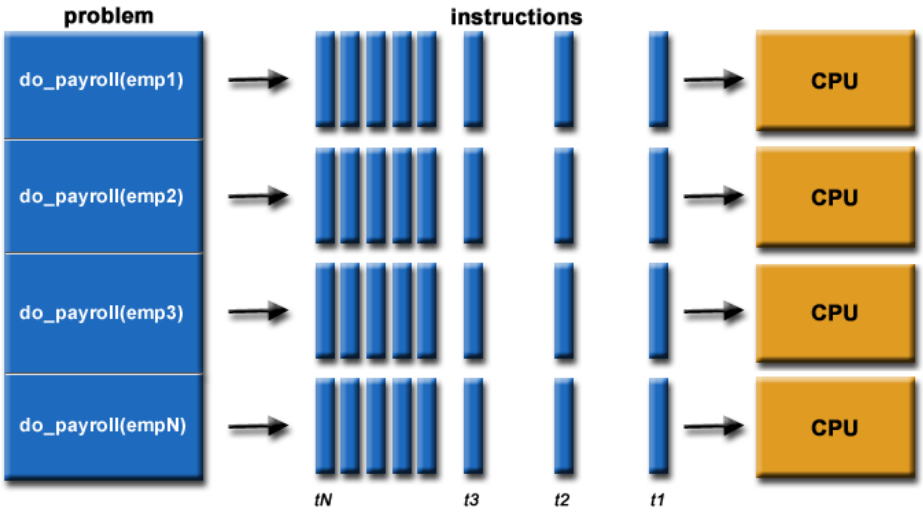


b) The example of sequential computing [19]

Figure 2.1: The concept and the example of sequential computing. The CPU sequentially executes N commands [19]



a) The concept of parallel computing [19]



b) The example of parallel computing [19]

Figure 2.2: The concept and the example of parallel computing [19]. The problem is divided into four unassociated tasks containing series of commands. Each task will be executed on each CPU [19]

Parallel computing can be used to solve arduous problems not only in Bioinformatics but also other fields of Science and Engineering such as Atmosphere, Physics, Chemistry, Biology, Geology, Mechanical Engineering and Computer Science [19].

2.1.2 Why apply parallel computing?

When parallel computing is applied to any computer program, there are many benefits, which are saving of computing time and/or cost, carrying out of bigger problems, supporting of concurrency, and applying of non-local resources [19].

First of all, parallel computing will save computing time and/or cost since it can exploit more resources such as computers or CPUs than sequential computing, with possible cost reduction. A cluster computer for executing parallel computing can be set up from cheap and profitable components; in contrast, it is expensive to build a single CPU providing the same or better performance [19].

The second benefit is that parallel computing can figure out larger or complicated problems, which can be solved by sequential computing slowly and arduously. For instance, web search engines or databases perform millions of transactions per second as parallel computing can help reducing their computing time [19].

In addition to save computing time or cost and carry out bigger problems, supporting concurrency is another advantage of parallel computing. Sequential computing can only execute one job at a time, while parallel computing can be run many tasks concurrently. For example, users from anywhere can see and do work “virtually” using the Access Grid (www.accessgrid.org) supporting a worldwide cooperation network [19].

Finally, parallel computing can access compute resources on a wide area network or the Internet while local computer resources are unavailable. For instance, SETI@home (setiathome.berkeley.edu) works with million computers in 253 countries, whereas Folding@home (folding.stanford.edu) consumes more than 450,000 CPUs universally [19].

2.2 Types of parallel computing

Parallel computing includes many parts of von Neumann architecture. The von Neumann architecture consists of four main components, which are memory, control unit, arithemetic logic unit, input and output as shown in Figure 2.3 [19].

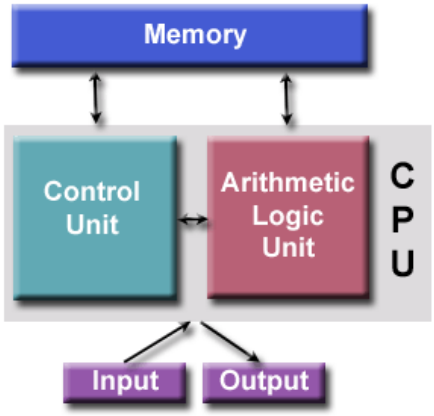


Figure 2.3: The von Neumann architecture [19]

The program commands and data are stored in read/write random access memory. The computers process data using the program commands. The control unit conveys commands and data from memory, translates and sequentially performs the commands to fulfil the programmed job. The aritmetic unit works on arithmetic operations. Input/output is the interface to the human operator [19].

Parallel computing can be grouped in various ways. The famous grouping is called Flynn's Taxonomy. It groupeds parallel computing using two independent dimensions of instructions and data. Only one of two possible states, single or multiple, can be included in each of dimensions. According to Flynn's Taxonomy, The four possible groupings can be shown in the matrix in Figure 2.4 [19].

SISD Single instruction, single data	SIMD Single instruction, multiple data
MISD Multiple instruction, single data	MIMD Multiple instruction, multiple data

Figure 2.4: Flynn's Taxonomy matrix [19]

First of all, SISD is the only sequential computing that only one instruction stream and one data stream are executed by the CPU at any clock cycle. It

is the most common type of computing performed on older generation mainframes, minicomputers and workstations, and most modern day Personal Computers (PCs). SIMD is next type of parallel computing that parallel CPUs process the same set of instructions but with different data sets on each at any supplied clock cycle. This type is famous for operatingto operate on most modern computers. Another type of parallel computing is MISD. Parallel CPUs perform different instruction streams with the same data set. The example of applications for MISD is that multiple cryptography algorithm tries to crack a single coded message. The last parallel computing type is MIMD. Different instruction streams with various data sets may be executed on each CPU. MIMD is commonly applied to supercomputers, networked parallel computer clusters and “grids”, multi-CPUs computers, and multi-core PCs. Basically, SIMD execution elements are contained in many MIMD [19].

2.3 Parallel computer memory architectures

There are three kinds of parallel computer memory architectures including shared memory, distributed memory and hybrid distributed-shared memory [19].

2.3.1 Shared memory architecture

Shared memory architecture allows all CPUs to fetch any memory as they appear in the global address space. Although each CPU shares the same memory resource, it processes a task simultaneously and independently. The main characteristic of shared memory architecture is cache coherent of which concept is that the CPU can modify any memory location also seen by other CPUs. Based on memory access times, shared memory architecture can be divided into two groups, which are uniform memory access (UMA) and non-uniform memory access (NUMA). UMA architecture called cache coherent UMA (CC-UMA) or symmetric multiprocessor (SMP) computer is shown in Figure 2.5. Each CPU can equally access the global memory and also take equal duration to do as well. Access and access time for each CPU to memory is equally. NUMA architecture or CC-NUMA is shown in

Figure 2.6. NUMA is frequently contains two or more physically linked SMPs. The memory of a SMP can directly be accessed by another SMP. However, access and access time for each CPU to all memories are not equal. Memory access in a SMP is faster than between SMPs. The benefit of this architecture is that global address space supports a user-friendly programming viewing to memory. Also, data sharing between tasks is speedy and uniform because of the adjacency of memory to CPUs. Nevertheless, there are three disadvantages of the architecture. First, the lack of scalability between memory and CPUs, increasing more CPUs can relatively gain traffic on the shared memory-CPU path and for cache coherent systems. Another disadvantage is that programmers need a special skill to synchronize constructs accessing to global memory correctly. The last disadvantage is that it is difficult and expensive to add new CPUs on a shared memory computer [19].

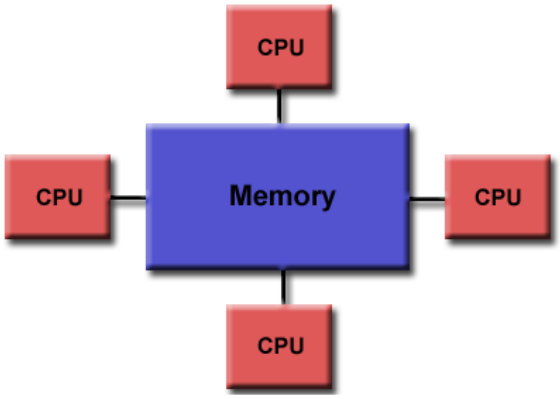


Figure 2.5: Uniform shared memory architecture (UMA) [19]

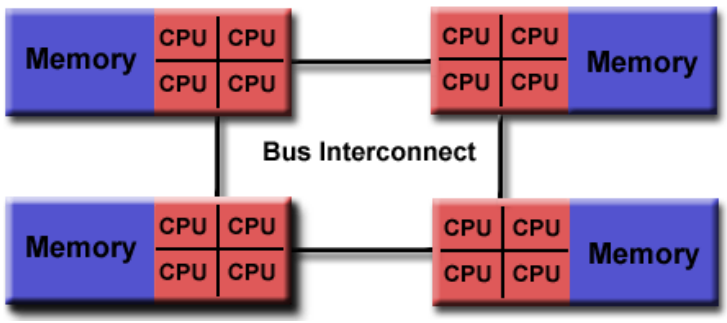


Figure 2.6: Non-uniform shared memory architecture (NUMA) [19]

2.3.2 Distributed memory architecture

Distributed memory architecture is another type of parallel computer memory architectures as shown in Figure 2.7. Each CPU has its own memory. Memory addresses in a CPU are not mapped for another CPU. Therefore, distributed memory architecture does not support global address space across all CPUs and cache coherence. The programmers must write a program to define how and when data is communicated via Ethernet when a CPU requires an access to data located in control of another CPU. The benefit of distributed memory architecture is that memory can be increased easily with the number of CPUs. Each CPU can quickly fetch its own memory without conflict or without overhead. Nonetheless, the programmers need a special skill to manage data communication between CPUs [19].

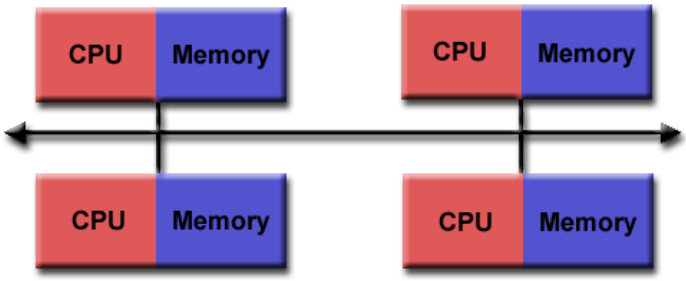


Figure 2.7: Distributed memory architecture [19]

2.3.3 Hybrid distributed-shared memory architecture

The last architecture is hybrid distributed-shared memory as shown in Figure 2.8. It is applied to the recent largest and fastest computers in the world. A cache coherent SMP machine and/or graphics processing units (GPUs) can be shared memory components, whereas the network of multiple SMP/GPU machines can be the distributed memory components. Each machine can only access its own memory. Hence, network communication is needed to transfer data from a SMP/GPU to another SMP/GPU. The hybrid distributed-shared memory is widespread today and tends to grow at the high end of computing in the future. Advantages and

disadvantages of hybrid distributed-shared memory can be inferred from both shared and distributed memory architectures [19].

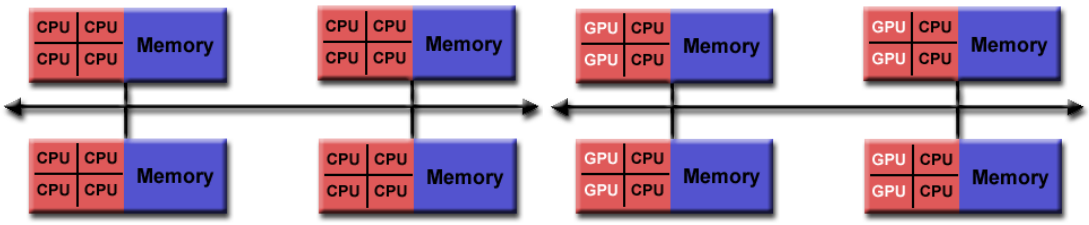


Figure 2.8: Hybrid distributed-shared memory architecture [19]

2.4 Parallel programming models

There are seven programming models in common use including share memory (without threads), threads, distributed memory/message passing, data parallel, hybrid, single program multiple data (SPMD) and multiple program multiple data (MPMD) [19].

2.4.1 Shared memory model (without threads)

In shared memory programming model, a common address space is shared by tasks, and it can be read and written asynchronously. The shared memory can be controlled by several mechanisms such as locks and semaphores. The advantage of this model is that programmers do not have to explicitly specify the communication of data between tasks. The downside of shared memory programming model is that it is difficult to understand and to manage a data locality when multiple CPUs use the same data [19].

2.4.2 Thread model

The thread model is a type of shared memory programming that a single process can have multiple and concurrent execution paths. A thread's work can be explained like a subroutine within the main program. Each thread can execute any

subroutine at the same time as other threads. Threads communicate with each other via global memory (updating address locations). Synchronization is needed to ensure that only one thread can update the global address at a time. POSIX threads and OpenMP are the implementations of thread programming model [19].

2.4.3 Distributed memory/message passing model

Message passing programming model is commonly applied in distributed memory computers. Several tasks consume their own local memory while processing. A set of tasks can be executed on the same physical machine or different machines. Communication methods including sending and receiving messages are used to exchange data between tasks as shown in Figure 2.9. Message passing interface (MPI), an implementation of this model, is an application programming interface (API) specification that allows processes to communicate with each other by sending and receiving messages [19].

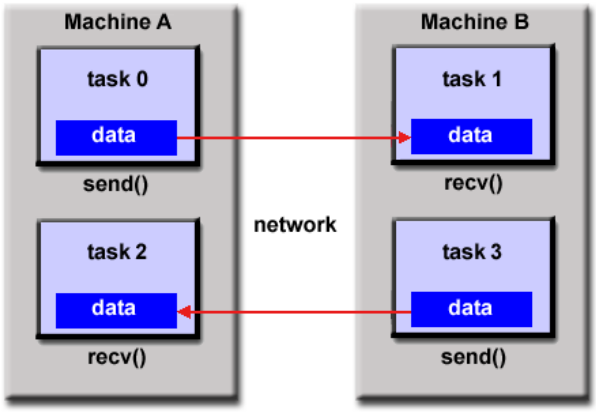


Figure 2.9: An example of message passing programming model [19]

2.4.4 Data parallel model

In data parallel model, each task performs the same operation on a different portion of the same data structure as shown in Figure 2.10. On shared memory architecture, all tasks may fetch the data structure via global memory. Whereas on distributed memory architecture the data structure is partitioned into "chunks" in the local memory of each task [19].

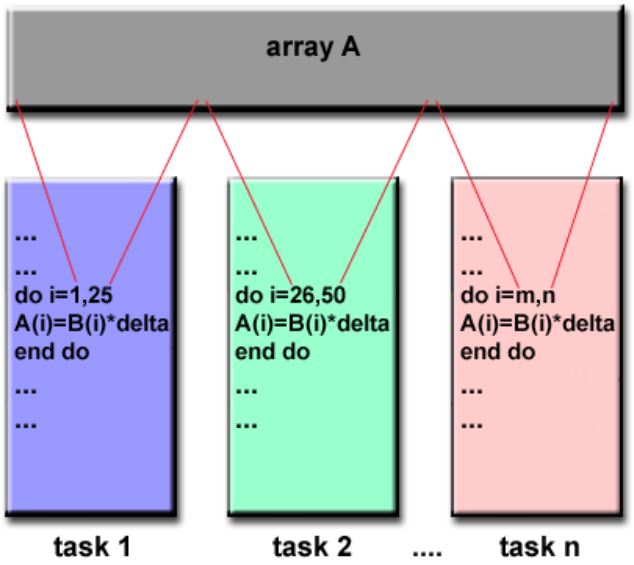


Figure 2.10: An example of data parallel model [19]

2.4.5 Hybrid model

A hybrid programming model contains more than one of the already explained programming models. A combination of the message passing model (MPI) with the thread model (OpenMP) is a general example of a hybrid programming model as shown in Figure 2.11. Computationally intensive kernels using local or on-node data are performed by thread, while communication between processes on other nodes over the network are operated by MPI [19].

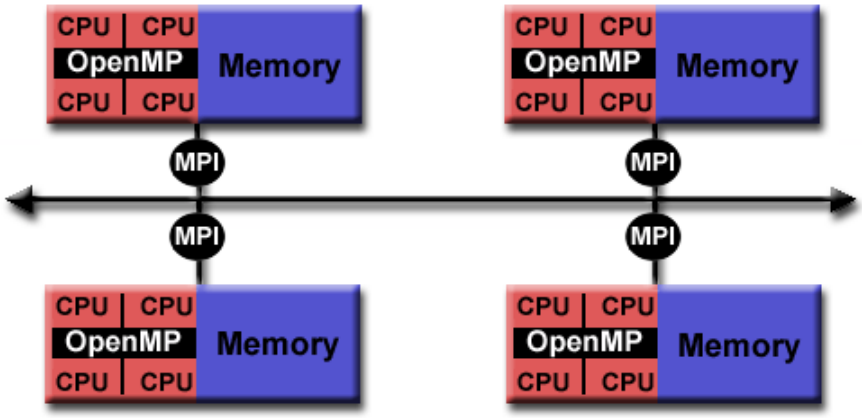


Figure 2.11: An example of hybrid programming model [19]

2.4.6 Single program multiple data (SPMD)

SPMD is an advanced level programming model combined the previously described programming models. All tasks perform their copy of the same program concurrently but may use different data as shown in Figure 2.12. The program can be threads, message passing, data parallel or hybrid. The SPMD programming model using message passing or hybrid programming is the most basically applied for multi-node clusters [19].

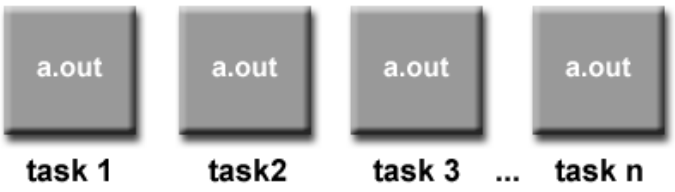


Figure 2.12: An example of SPMD [19]

2.4.7 Mutilple Program Multiple Data (MPMD)

MPMD programming model is also an advanced level programming model combined the previously described programming models. Each task performs a different program at the same time, and may process different data as shown in Figure 2.13. Like SPMD, The program can be threads, message passing, data parallel or hybrid [19].

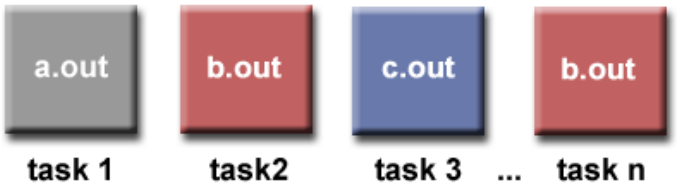


Figure 2.13: An example of MPMD [19]

However, these programming models are not specific to a certain type of computer or memory architecture. For instance, the shared memory programming model can be employed to a distributed memory computer. Physical memory of the computer is distributed but presented to the user as a single shared memory (global address space) called "virtual shared memory." Another instance, distributed memory programming model (MPI) can be applied to a shared memory computer. Tasks directly access to global address space of all computers. Nonetheless, MPI is used to send and receive massages over shared memory [19].

2.5 Parallel program design

2.5.1 Automatic vs. manual parallelization

Generally, manual parallelization is applied to design and develop parallelable programs more than automatic parallelization. However, manual parallelization is a time consuming, complex, error-prone and iterative process. For this reason, various tools for automatic parallelization have been released, for example, a compiler or pre-processor used to convert sequential programs into parallel programs. Nevertheless, automatic parallelization is limited to a subset (mostly loops) of code and may produce incorrect outputs and give poor performance [19].

2.5.2 Understand the problem and the program

Before develop the parallel program, we have to understand how the problem to be solved in parallel. If we begin with a sequential program, we must understand the existing code. Since not all problems can be solved by parallel computing, we should check whether parallel computing can be applied before starting to develop a parallel program. An example of a parallelizable problem is sequence similarity finding using the basic local alignment search tool (BLAST). This problem can be divided into a set of independent tasks. In contrast, an example of a non-parallelizable problem is the Fibonacci sequence, which can not be divided to independent tasks. Moreover, parallel computing should only be applied to the program's hotspots. Therefore, those sections of the program that consume little CPU usage can be ignored [19].

2.5.3 Decomposition methods

Two simple methods to create parallel tasks which are domain decomposition and functional decomposition [19].

In domain decomposition, the data related with the problem is to be decomposed. After that, each parallel task will process only a portion of data as shown in Figure 2.14. In addition, there are two data partitioning techniques which are block and cyclic partitioning. Figure 2.15 shows the examples of one-dimension and two-dimensions data decomposition using block and cyclic partitioning techniques [19].

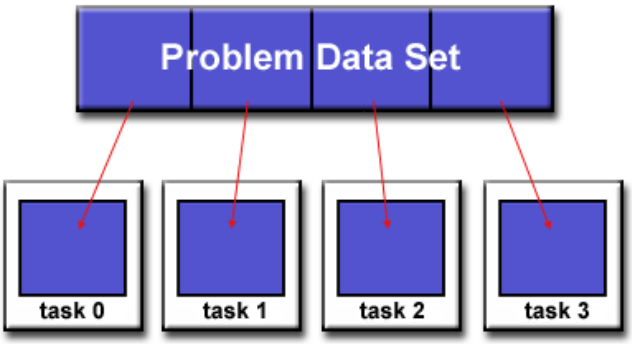


Figure 2.14: An example of domain decomposition

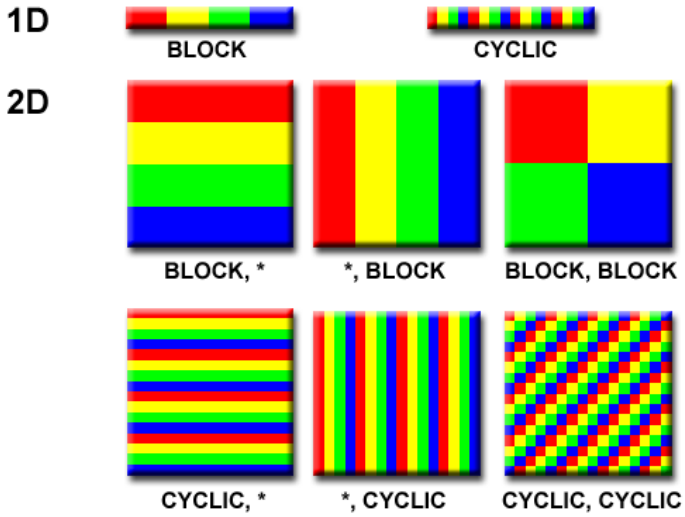


Figure 2.15: An example of domain decomposition using block and cyclic partitioning techniques [19]

Functional decomposition is another parallelization technique. In this technique, the instruction set is decomposed into a set of tasks as shown in figure 2.16. Then, each task is processed on parallel machines [19].

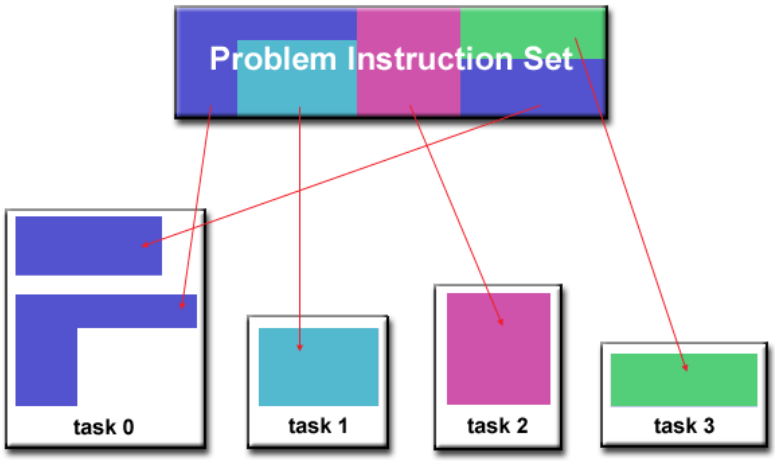


Figure 2.16: An example of functional decomposition [19]

2.5.4 Load Balancing

Load balancing is a method concerning task distribution to keep each CPU busy working. Better load balancing can produce higher performance of parallel computing. There are two approaches to achieve load balance. The first approach is equal work partitioning that each task processes within need the same period of computing time. Another approach is dynamic work assignment. An example of dynamic work assignment is the task pool method. If a task is finished, the next task in a queue will be executed [19].

2.6 Summary

Since ParallABEL and ParallLogic analyze genotype and phenotype data containing SNPs and individuals, the libraries are developed based on the SIMD parallel computing and distributed memory/message passing programming model. ParallABEL and ParallLogic also can be run on shared memory architecture and distributed memory architecture. Manual parallelization is applied to design and develop the libraries.

CHAPTER 3

Speeding up SNP Association Analyses Applying Parallel Computing for GWA Studies

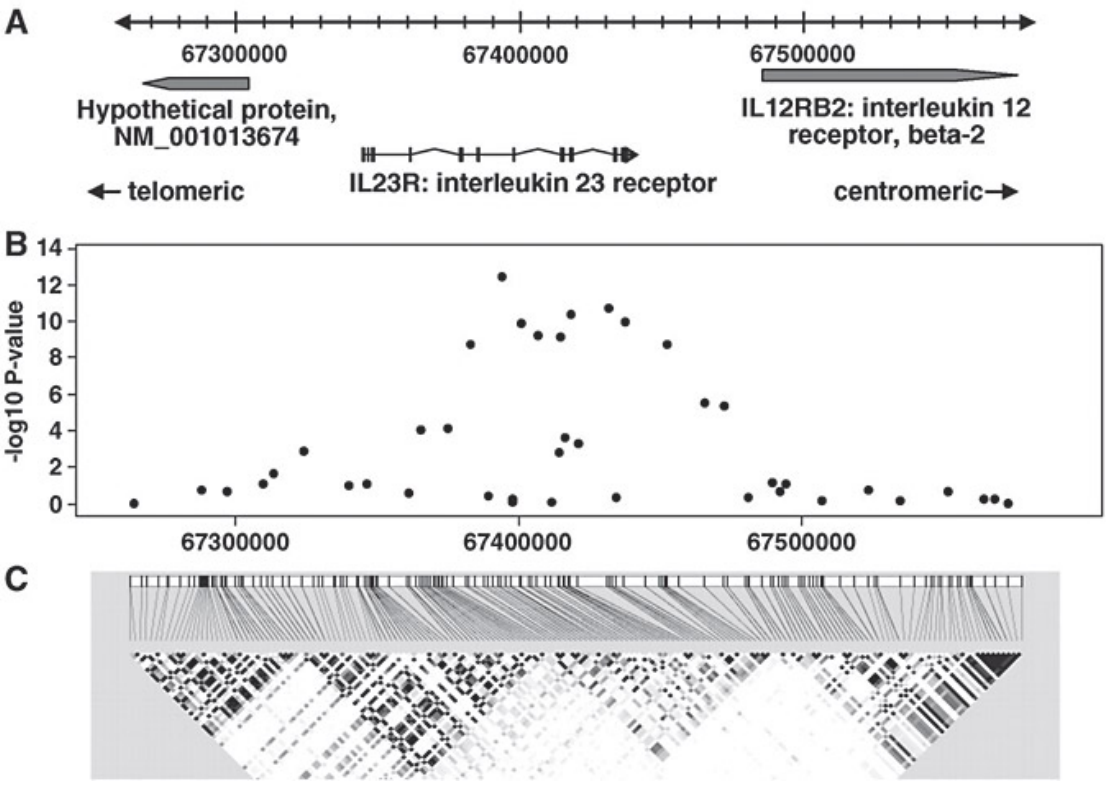
3.1 Introduction

GWA analysis [9] is a well established and powerful method for identifying loci associated with variations of complex genetic traits such as common diseases. For non-Mendlian consideration, GWA studies are more effective than family-based linkage studies, which have arduously assembled results related to several hundred markers throughout the genome. Eventhough family-based linkage studies can identify genes of large effect in Mendelian diseases such as neurofibromatosis, it limits to only common diseases like asthma [20]. The disadvantages of linkage studies are low proficiency for complex disorders influenced by multiple genes, and that it is hard to identify a causative gene due to the large size of the chromosomal regions shared among family members. GWA studies are developed based on the benefits of candidate genes, family linkage studies and the expanding knowledge of the relationships among SNP variants created by the International Hapmap Project [21-22]. GWA studies aim to acquire the important differences among individuals and associate them to health and illness. Hundreds of new genes have been implicated in human health and diseases during the last few years in various GWA studies [23]. GWA analyses succeed to lead discovery of associations of specific genes with diseases such as coronary heart disease, diabetes, rheumatoid, Chrohn's disease, biolara disorder and hypertension [2-3]. The case-control design has often been used to create GWA studies. In this method, allele frequencies in patients with the disorder of interest are compared to those in participants with disorder-free of interest. Case-control studies are frequently easier and less expensive to create than studies applying other designs [9]. In a typical study, hundreds of thousands of the common form of genetic variants or SNPs are assayed by high throughput genotyping technologies in order to detect genetic risk factors [4].

In the National Center for Biotechnology Information's dbSNP database, closely twelve million unique human SNPs have been coded a reference SNP (rs) number [5] and marked as to specific alleles (alternate form of the SNPs). Also, summary allele frequencies and other genomic information can be calculated from the human SNPs [6]. There are 56,411 significant SNPs from 118 articles related to diseases [7].

Basically, the GWA study has 4 processes: (1) selection from a huge number of individuals with the disorder or trait of interest and an eligible comparison category; (2) DNA isolation, genotyping, and data checking; (3) statistical analyses for associations between the SNPs passing suitable thresholds and the disorder or trait; and (4) replication of identified associations [9].

Figure 3.1 shows the statistical output for genome-wide association study of inflammatory bowel disease. The IL23R gene has two blocks of linkage disequilibrium. The association signals are strongest in the centromeric block containing exons 5 to 11, whereas markers in the block encompassing the IL12RB2 gene do not demonstrate significant association [24].



Genome-wide association studies frequently identify associations with many highly correlated single-nucleotide polymorphisms (SNPs) in a chromosomal region, due in part to linkage disequilibrium, among the SNPs. This can make it difficult to determine which SNP within a group is likely to be the causative or functional variant. A, Genomic locations of 2 genes, the interleukin 23 receptor (*IL23R*) and the interleukin 12 receptor, beta-2 (*IL12Rb2*), and a hypothetical protein, NM_001013674, between positions 62700000 and 67580000 of the short arm of chromosome 1 at region 1p31, are shown. B, The $-\log_{10} P$ values for association with inflammatory bowel disease are plotted for each SNP genotyped in the region; those reaching a prespecified value of $-\log_{10}$ of 7 or greater are presumed to show association with disease. Several strong associations, at $-\log_{10} P$ values or greater, are seen in the region just telomeric of position approximately 67400000 and extending just centromeric of position approximate 67450000. C, Pairwise linkage disequilibrium estimates between SNPs (measured as r^2) are plotted for the region. Higher r^2 values are indicated by darker shading. The region contains 4 "triangles" or "blocks" of linkage disequilibrium, 2 on either side of position 67400000 in the *IL23R* gene, another in the hypothetical protein telomeric of *IL23R*, and a fourth in the *IL12RB2* gene at the centromeric end of the region. The 2 *IL23R* linkage disequilibrium regions each contain SNPs associated with inflammatory bowel disease, while the *IL12RB2* region does not. Reproduced with permission from Duerr et al.[53]

Figure 3.1: Associations in the IL23R gene region identified by a GWAS of inflammatory bowel disease [9].

GenABEL is a specialized library package for GWA analysis [10] implemented in R, an open source statistics programming language and environment [11-12]. GenABEL enables GWA analysis to be done using a regular desktop computer due to its efficient data storage and memory management. Nevertheless, analysis of very large data sets is computationally challenging and may take hours or

weeks or months to complete. Examples include the utilization of sophisticated adjustments for population stratification and relationship structures, the estimation of linkage disequilibriums and the calculation of genome-wide identity-by-state, haplotypic tests, and permutation analyses.

To increase the computational throughput, a user can partition their data into sets, and perform the analysis of the sets across a network of computers; a concept known as parallel and/or distributed computing. It is arduous acquiring the necessary programming skills to correctly partition and distribute data, control and monitor tasks on clustered computers, and merge output files. Occasionally, a data set may fail to be processed, e.g. if the user did not partition the data into small enough subsets to be processed on a particular machine. Also, the outputs from the computers may be scattered and their order is hard to follow.

Parallel computing is an intuitive and powerful method for increasing computational throughput. A task is separated into smaller tasks which are processed simultaneously on multiple Central Processing Units (CPUs) or a cluster of computers. The outputs from each task must later be merged [19]. A general architecture for parallel computing is shown in Figure 3.2.

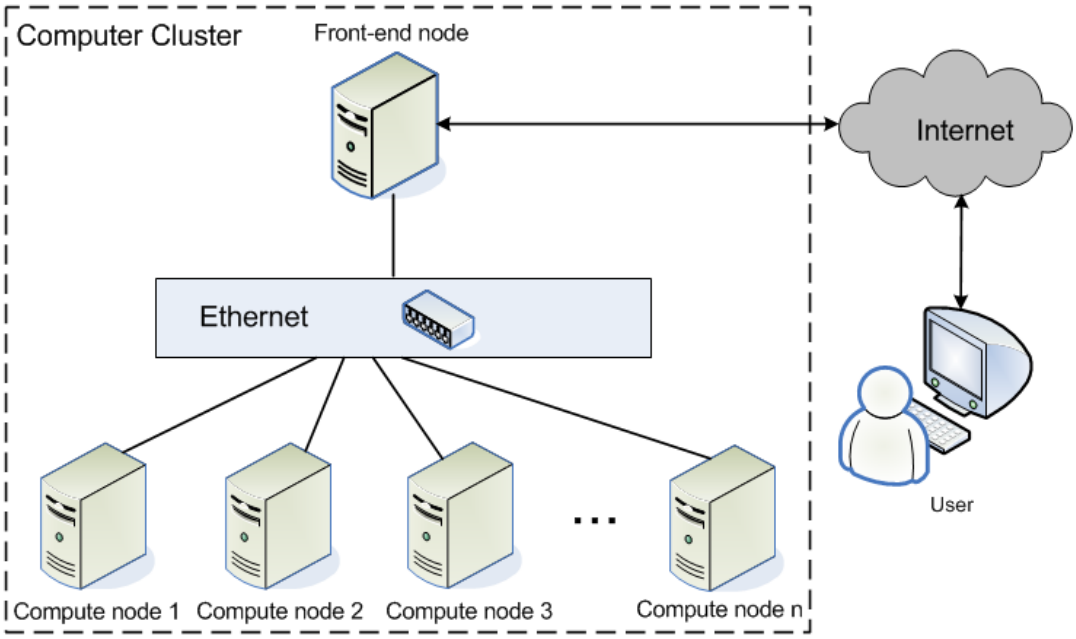


Figure 3.2: The user can submit tasks to the cluster of computers via the Internet. Once the user submits a job to the computer cluster, the front-end node schedules and distributes the smaller partitioned tasks to be processed on the compute nodes. The output from each compute node will then be merged by the front-end node.

Most tasks solved in GWA analysis are suitable for parallelization due to their computational independency so that parallelization can be achieved at the data level. For example, association tests can usually be done separately for each SNP and/or a small group of SNPs. Consequently, parallelization is a beneficial way to reduce the computing time, with few overheads incurved in large-scale GWA analyses.

Several attempts had been made to parallelize genetic association analyses. Grid Engine, a cutting-edge parallel tool, can schedule parallel tasks involving genetic association analysis programs [25] such as FBAT [26] and UNPHASED [27]. The approach, first proposed by Mishima et al., is based on non-parallel code combined through process-based parallelization. The downside is that the user still needs to monitor when each task is finished, and when the outputs from all the tasks can be merged. Moreover, each process may take a very long time to finish, and load balance can be problematic. A granularity problem (a high

computation to communication ratio) may occur. However, using higher power compute nodes or code parallelization are possible solutions. The R/parallel package has been used to automate loop parallel execution, but the application must run on a single computer with multi-core CPUs, and does not currently support cluster computing [28]. Its inclusion of cluster computing would eliminate the computing time limit of the package. Misawa and Kamatani [29] developed the ParaHaplo package for haplotype-based whole-genome association studies using parallel computing. It is aimed at correcting multiple comparisons in multiple SNP loci in linkage disequilibrium. Also, Ma et al. [30] developed EPISNPmpi, a parallel system for epistasis testing in large scale GWA analysis. However, there are other statistical analyses requirements in GWA studies, such as obtaining statistics for a particular SNP or a trait, association test, characterizing an individual in the study, and pair-wise statistics between individuals.

Rmpi [15] is an R library which provides various functions to parallelize tasks on R using the MPI (Message-Passing Interface) [16]. Rmpi employs various functions to manage flow analysis in parallel environment, and is applicable for employing multi-core CPUs distributed across many computers, not only multi-core CPUs on a single computer. However, it is difficult, if not impossible, for a non-programmer to write a parallel Rmpi program. Therefore, SPRINT [31] was developed to implement parallel R functions. Although users can use SPRINT easily, it does not specifically support GWA studies.

In this chapter, we present the development of our ParallABEL library, a new R library for parallelization of GWA studies based on Rmpi and GenABEL. ParallABEL aims to speed up the computation of GWA studies for various statistical analysis requirements and also simplify analysis parallelization. With ParallABEL, the users do not need to be experts in parallel programming, no need to know about partitioning and distributing data, controling and monitoring tasks, and merging output files**.**

3.2 Methods

3.2.1 GWA Function Grouping

Statistical analyses in GWA studies can be categorized into four groups based on the nature of the statistics computed and types of data used. These four groups can be parallelized in distinct ways. Table 3.1 shows the names and descriptions of the GenABEL functions in each group. The first group contains statistics computed for a particular SNP, or a trait, such as the SNP characterization statistics (e.g. call rates, hardy-weinberg equilibrium (HWE) testing [10]), produced by GenABEL's *summary.snp.data* or association test statistics (the *qtscore, mlreg* and *mmscore* GenABEL functions [10]). The second group holds statistics characterizing an individual in the study, such as, summary statistics of genotype quality for each sample (obtained with the GenABEL *perid.summary* and *hom* GenABEL functions [10]). The third group consists of pair-wise statistics derived from analyses between each pair of individuals in the study, including genome-wide identity-by-state and genomic kinship analyses. This is one of the most computationally intensive analyses, obtained through GenABEL's *ibs* function [10]. The final group concerns pair-wise statistics derived for pairs of SNPs, such as linkage disequilibrium characterisation (the *dprfast, rhofast* and *r2fast* functions [10]). While the number of SNP pairs is generally very large, analyses are usually limited by their pair-wise physical distance, making them less demanding than pair-wise individual analyses, such as IBS computations [10].

Table 3.1: The names and descriptions of GenABEL functions in each group

function name of GenABEL	Description	group
summary.snp.data	Provides summary of observed genotypes, allelic frequency, genotypic distribution, P-value of the exact test for HWE and chromosome	1
qtscore	Fast score test for association between a trait and genetic polymorphism	1
mlreg	Linear and logistic regression and Cox models for genome-wide SNP data	1
mmscore	Score test for association between a trait and genetic polymorphism, in samples of related individuals	1
perid.summary	Produces call rate and heterozygosity per person	2
hom	Computes average homozygosity (inbreeding) for a set of people, across multiple markers. Can be used for Quality Control (e.g. contamination checks)	2
ibs	Given a set of SNPs, computes a matrix of average IBS for a group of people	3
dprfast	Given a set of SNPs, computes a matrix of D'	4
rhofast	Given a set of SNPs, computes a matrix of rho	4
r2fast	Given a set of SNPs, computes a matrix of r2	4

We have developed the ParallABEL library to parallelize the serial functions of these groups using Rmpi library. The four implementation groups are named Type1_parall_by_SNPs for the first group, Type2_parall_by_individuals for the second group, Type3_parall_by_pairs_of_individuals for the third group and Type4_parall_by_pairs_of_SNPs for the fourth group.

3.2.2 Data Partitioning

An advantage of ParallABEL is usage simplicity, hiding otherwise tedious scripts for file management monitoring tools. These functions not only partition input data with automatic load balancing, but also gather output from each CPU automatically. Load balancing is critical because an unbalanced work load will result in higher loads for particular CPUs, which eventually undermines the overall performance.

The input data of Type1_parall_by_SNPs are SNPs equally partitioned into *P* subsets (where *P* is the number of available CPUs). If the number of SNPs is *M*, the number of SNPs in a subset is:

$$num\_SNPs = floor(M/P)$$

If there are *M* SNPs and 4 CPUs, the SNPs will be partitioned into 4 smaller subsets. Each contains *M*/4 SNPs as shown in Figure 3.3.

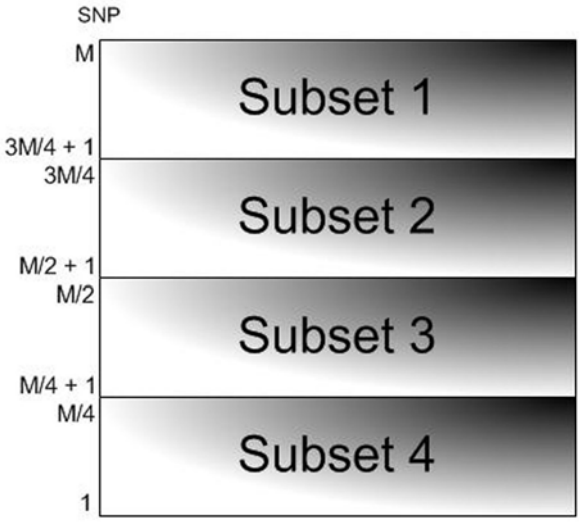


Figure 3.3: Data partitioning for Type1_parall_by_SNPs
Type2_parall_by_individuals when M = 800 and P = 4

However, the last subset to be generated may contain more SNPs than others, caused by integer division. For example, if there are 801 SNPs and 4 CPUs, Subset 1 to Subset 3 will contain 200 SNPs, but Subset 4 will have 201 SNPs. The SNPs in each subset will be executed on separate CPUs.

The input data for Type2_parall_by_individuals are individuals, partitioned like Type1_parall_by_SNPs.

The input data for Type3_parall_by_pairs_of_individuals is a pair of individuals, and performs a more complicated partitioning than Type1_parall_by_SNPs and Type2_parall_by_individuals. The data is divided until the number of CPUs is equal to, or less than, the number of subsets for load balancing on each CPU. If the number of CPUs is equal to the number of subsets, then each CPU executes an individual pair of each subset. If the number of CPUs is less than the number of subsets, then each CPU executes an equal number of individual pairs (where it is possible). Figure 3.4 shows Type3_parall_by_pairs_of_individuals with N individuals. The statistics is calculated from the cross operation of an individual in a row with an individual in a column. The input data is partitioned into 4 subsets using the data partitioning shown in Figure 3.4A. However, if the number of CPUs is more than 4, the subsets will be partitioned again. Subset 1 and Subset 4 are split into 8 subsets during the first stage of the data partitioning, while Subset 2 and Subset 3 are divided into 8 subsets by row, as shown in Figure 3.4B. There are 16 subsets altogether in the second stage of the data partitioning.

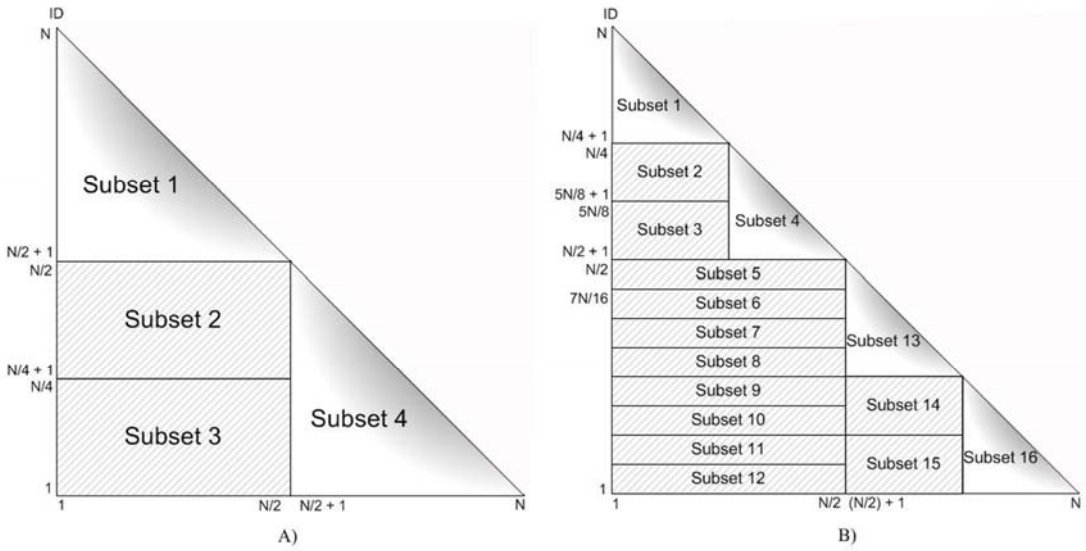


Figure 3.4: A) The first data partitioning for Type3_parall_by_pairs_of_individuals when the number of individuals = N. There are 4 equal subsets. B) The second data partitioning for Type3_parall_by_pairs_of_individuals when the number of individuals = N. There are 16 equal subsets.

The SNPs input of Type4_parall_by_pairs_of_SNPs will be partitioned in a similar way to Type3_parall_by_pairs_of_individuals.

3.2.3 Implementation

The workflow for GWA analysis on a single CPU or computer is presented in Figure 3.5A. This workflow runs properly. The genotype and phenotype data (as shown in Table 3.2 and Table 3.3 respectively) is processed by the GenABEL library that works under the R program. GenABEL sequentially processes the raw data, producing statistical data as its outputs.

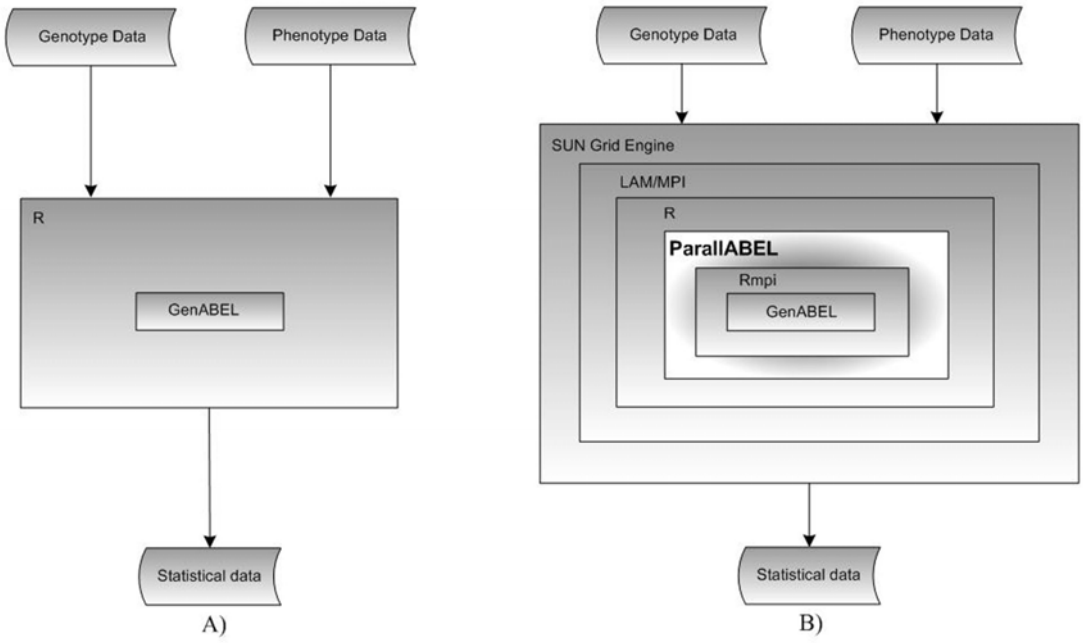


Figure 3.5: A) Sequential GWA Computing Workflow, which runs on a single CPU or computer. B) Parallel GWA Computing Workflow that runs on a multiprocessor or a set of computers.

Table 3.2: The example of genotype data executed in GenABEL and ParallABEL

snpid	chrom	chromEnd	strand	id199	id287	id300
rs7435137	1	4259040	-	CT	CT	CT
rs7725697	3	10806991	-	CC	CG	CC
rs664063	2	7288020	-	GG	GC	GG
rs4670072	X	13387482	+	AA	--	AA
rs546570	2	6120257	+	AA	AA	AA
rs7908680	1	2311762	-	CC	CA	CC
rs166732	1	4716343	-	TT	TG	TT
rs4257079	1	3455895	-	AA	AA	AA
rs5150804	2	7178160	+	AG	AG	GG

Table 3.3: The example of phenotype data executed in GenABEL and ParallABEL

id	sex	age	disease	height	weight
id199	1	59	1	164	80
id287	0	43	1	169	139
id300	1	42	1	177	81

This sequential workflow may take a very long time to produce some demanding statistical analyses. Our novel parallel workflow for producing statistical data in GWA studies shown in Figure 3.5B can save the computing time. The genotype and phenotype data (as shown in Table 3.2 and Table 3.3 respectively) is passed for distribution to the SUN Grid Engine [32], a job scheduler. It queues jobs and assigns them to CPUs in a cluster. LAM/MPI (Local Area Multicomputer/Message Passing Interface) [33] has various functions which can be called by Rmpi to parallelize R operations. ParallABEL parallelizes GenABEL using this Rmpi library. The statistical data from this workflow has been validated by comparing it with the outputs from the non-parallel approach. ParallABEL runs not only on Linux cluster, such as the Rocks Cluster Distribution, but also on any Operating System that supports R and LAM/MPI or Open MPI, such as the Unix and Solaris operating systems. It can also run on computer clusters not using the Sun Grid Engine but it will fully occupy the CPUs until the end of the program, so that other applications can not share the execution time of the occupied CPUs. However, normally the administrator will not allow a user to run a parallel program without utilizing a queuing process from the Sun Grid Engine or a scheduler.

ParallABEL is developed based on SIMD parallel computing and distributed memory/message passing programming model. ParallABEL can also be run on shared memory architecture and distributed memory architecture. Manual parallelization is applied to design and develop ParallABEL for more flexibility in programming.

To parallelize GWA studies, ParallABEL running on the front-end node partitions input data into smaller subsets so that tasks can be fairly distributed among the CPUs. It sends tasks to idle CPUs on compute nodes. When the

computation on a compute node has finished, the front-end node will send another task to the idle CPU – a cycle that continues until all the tasks are completed, which is known as the 'task pull' method [34]. When all the tasks are finished, the front-end node automatically merges all the outputs.

The task pull template [35] has been adapted for all types of ParallABEL. The example of Type1_parall_by_SNPs is shown in the source code of Appendix D. To parallelize Type1_parall_by_SNPs, there are five steps: (1) task separation (2) task distribution; (3) task computation in compute nodes; (4) result storing; and (5) result combination. The detail of each step can be seen in 3.2.2 Data Partitioning section, and the source code of Type2_parall_by_individuals, Type3_parall_by_pairs_of_individuals and Type4_parall_by_pairs_of_SNPs has been published at https://r-forge.r-project.org/R/?group_id=505.

Users can use ParallABEL to parallelize GenABEL GWA functions as easily as using GenABEL for sequential analyses. An example of the *mlreg.p* command sequentially run on a CPU is shown in Figures 3.6A and 3.7A. The executable command that parallelizes *mlreg.p* to run on multiple CPUs using Type1_parall_by_SNPs is shown in Figures 3.6B and 3.7B.

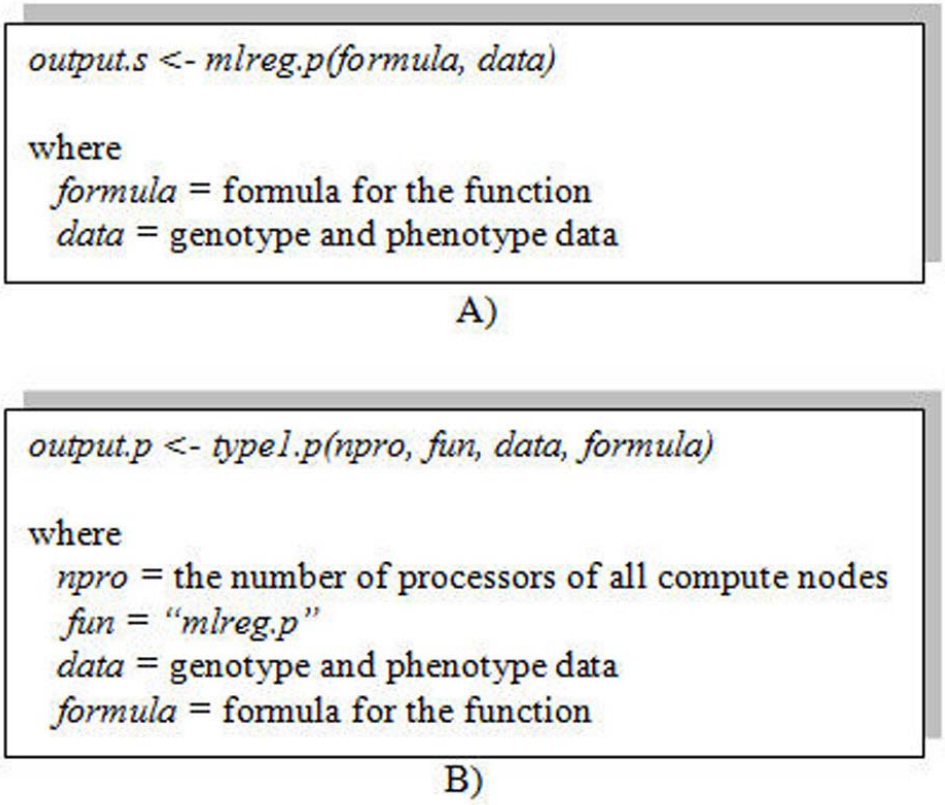


Figure 3.6: A) Executing the *mlreg.p* function sequentially on a CPU B) Parallelizing the *mlreg.p* function on more than one CPU. The user supplies the function name and number of available CPUs to the parallel function. However, if the user does not specify the number of CPUs, ParallABEL will automatically get it from Sun Grid Engine or from the default value (2).

```
library(GenABEL)

data <- ge03d2.clean[,]
formula <- dm2~sex+age

output.s <- mlreg.p(formula=formula, data=data)
```

A)

```
library(ParallABEL)
library(GenABEL)

data <- ge03d2.clean[,]
formula <- dm2~sex+age

output.p <- type1.p(npro=2, fun=mlreg.p, data=data, formula=formula)
```

B)

Figure 3.7: A) Execute the *mlreg.p* function sequentially on a CPU B) Parallelize the *mlreg.p* function on more than one CPU.

3.3 Results

Our computer cluster, Hanuman, runs Rocks Cluster Distribution version 4.3 which includes the SUN Grid Engine version 4.3 [36]. The cluster consists of 5 IBM servers xSeries 336s, comprising of a front-end node and four compute nodes. All servers have 2 SINGLE-CORE Intel Xeon (2.8 GHz) CPUs and 4 GB RAM. The front-end node and all the compute nodes are connected through an Ethernet switch, and the user can connect to the system via the Internet. The cluster provides LAM/MPI version 7.1.2, R program version 2.8.1, Rmpi library version 0.5-6, and GenABEL version 1.4-2, which are utilized as components by our ParallABEL library.

The North American Rheumatoid Arthritis Consortium (NARAC) data is part of a dataset employed to observe associations between disease and variants in the major-histocompatibility-complex locus [17]. The NARAC genotype data contains 545,080 SNPs from 2,062 individuals. The data was used to measure the performance of ParallABEL by employing 868 individuals for cases, and 1,194 individuals as controls.

Trace results from Type1_parall_by_SNPs, Type2_parall_by_individuals, Type3_parall_by_pairs_of_individuals, and Type4_parall_by_pairs_of_SNPs for the NARAC data are shown in Figure 3.8. Type1_parall_by_SNPs was executed by the GenABEL *mlreg* function, Type2_parall_by_individuals was executed by the GenABEL *hom* function, Type3_parall_by_pairs_of_individuals was executed by the GenABEL *ibs* function, and Type4_parall_by_pairs_of_SNPs was executed by the GenABEL *r2fast* function. ParallABEL reduced the computing time for Type3_parall_by_pairs_of_individuals, especially with 8 CPUs. The Type3_parall_by_pairs_of_individuals executing speed on eight CPUs was approximately seven times faster than on one CPU. On a single CPU, the complete analysis took 8.1 hours, but only 1.1 hours with 8 CPUs. The computing time for Type1_parall_by_SNPs also tends to be like that for Type3_parall_by_pairs_of_individuals.

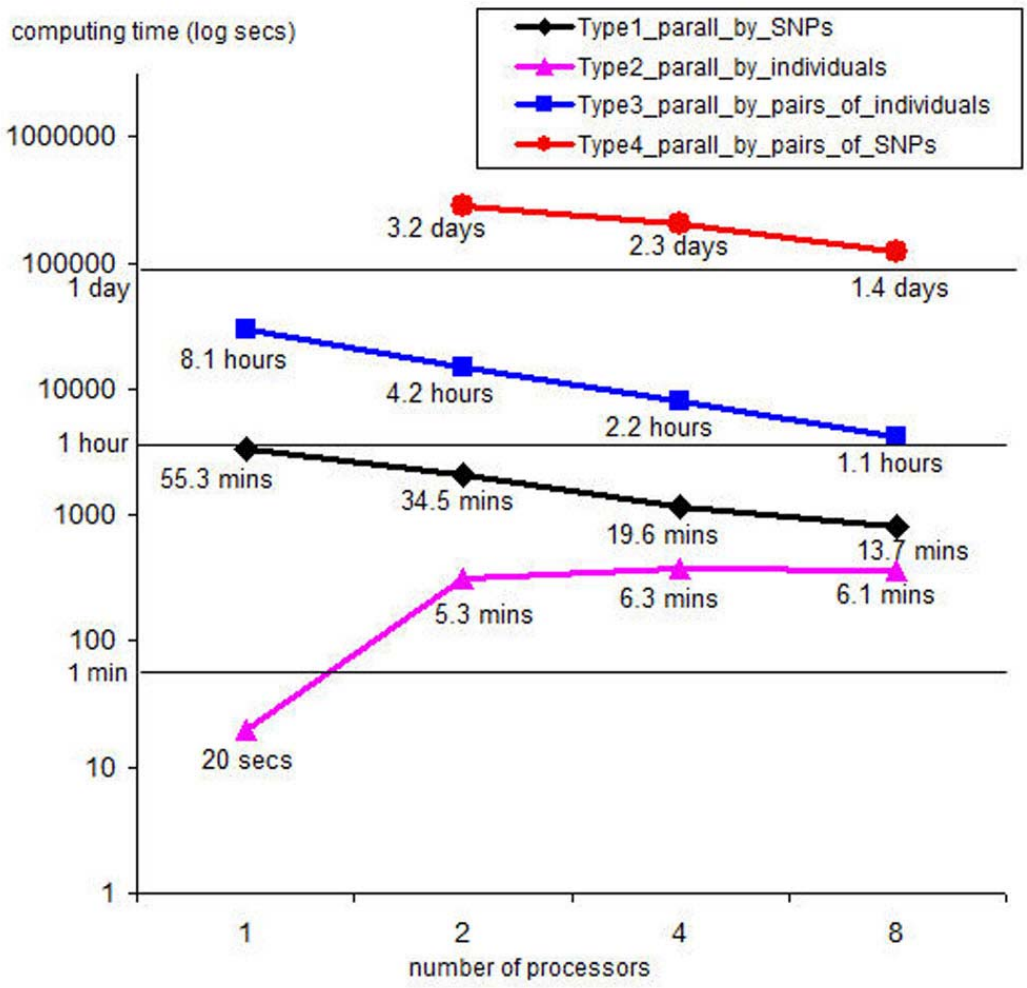


Figure 3.8: Trace results from Type1_parall_by_SNPs, Type2_parall_by_individuals, Type3_parall_by_pairs_of_individuals, and Type4_parall_by_pairs_of_SNPs for NARAC data. When Type1_parall_by_SNPs is executed by the GenABEL mlreg function, Type2_parall_by_individuals is executed by the GenABEL hom function, Type3_parall_by_pairs_of_individuals is executed by the GenABEL ibs function, and Type4_parall_by_pairs_of_SNPs is executed by the GenABEL r2fast function. If there is only one CPU, then the data will be analysed using GenABEL. If there are more than one CPU, the data will be analysed using ParallABEL package.

The computing time for the sequential version of Type2_parall_by_individuals can be very short (e.g. 20 seconds). While the parallel version took longer (5.3 minutes for 2 CPUs), due to the overhead of data partitioning, data distribution, and data merging. Data distribution can be time consuming because the data must be saved on the front-end node before the compute nodes can load it, and the front-end node must also spend time communicating with the compute nodes. In addition, ParallABEL is tailored to quickly retrieve subsets of SNPs, as this is a typical GWA scanning procedure, but is much less efficient in retrieving subsets of individuals, which is less typical. Thus, the overhead of data partitioning in subsets of individuals prevailed over the gain achieved by parallel processing. These results highlighted a place where ParallABEL data storage and processing is ineffective. It is a waste of time to speed up Type2_parall_by_individual because the computation of Type2_parall_by_individuals on a CPU is fast.

Type4_parall_by_pairs_of_SNPs was executed by the GenABEL *r2fast* function. A single CPU can not pass all the SNPs in the NARAC data to *r2fast* due to CPU memory limitations so the analysis was done separately for each chromosome. Even then, a single CPU can not call *r2fast* with a chromosome with more than 10,000 SNPs, which affects 20 chromosomes in the data. However, ParallABEL can run *r2fast* with a chromosome with more than 10,000 SNPs by employing a set of CPUs. The chromosome data is automatically partitioned based on the number of SNPs as shown in Table 3.4.

Table 3.4: The least number of subsets of each chromosome partitioned by the number of SNPs

Chromosome name	Number of SNPs	Number of subsets
19,20,21,22,X,Y	11-14,000	4
9,11,12,13,14,15,16,17,18	14,001-28,000	16
1,2,3,4,5,6,7,8,10	28,001-56,000	64

If the number of SNPs for a chromosome is between 11 and 14,000, then the data will be partitioned into at least 4 balanced subsets. If the number of the SNPs is between 14,001 and 28,000, then the data wil be divided into at least 16

balanced subsets. If the number of SNPs is between 28,001 and 65,000, then the data will be split into at least 64 balanced subsets. The data will be automatically partitioned until the number of CPUs is equal to, or less than, the number of subsets for load balancing on each CPU. The trace example results for Type4_parall_by_pairs_of_SNPs of NARAC data are shown in Figure 3.8.

Type4_parall_by_pairs_of_SNPs took only 1.4 days to execute on eight CPUs, indicating that time-saving with ParallABEL is linearly correlated to the number of nodes. This suggests that with more SNPs, more computing time will be saved by ParallABEL.

If the number of available CPUs is *P*, the parallel computing time for *P* CPUs is *time_P_cpus*, and the serial computing time for a CPU is *time_a_cpu*; the overhead for *P* CPUs will be:

$$overhead = time\_P\_cpus - time\_a\_cpu/P$$

Different numbers of CPUs produce different overheads depending on data partitioning, network communication, and data merging. However, the overheads can be predicted based on the overhead of eight CPUs shown in Figure 3.8. The computing time on a large cluster for Type1_parall_by_SNPs, Type3_parall_by_pairs_of_individuals and Type4_parall_by_pairs_of_SNPs extrapolated from Figure 3.8 applying the above overhead equation are shown in Figure 3.9. It is clear that ParallABEL also saves the computing time on a large cluster. In addition, the time-saving rates for these types will be much increased when the number of CPUs is in between 2 and 50. Nevertheless, the time-saving rates will be slowly increased when the number of CPUs is greater than 50. This applies to the particularly and relatively small data set analyzed here. With bigger data sets, the time-saving rates can be larger. However, the user should optimize the number of CPUs according to the gain in computational throughput.

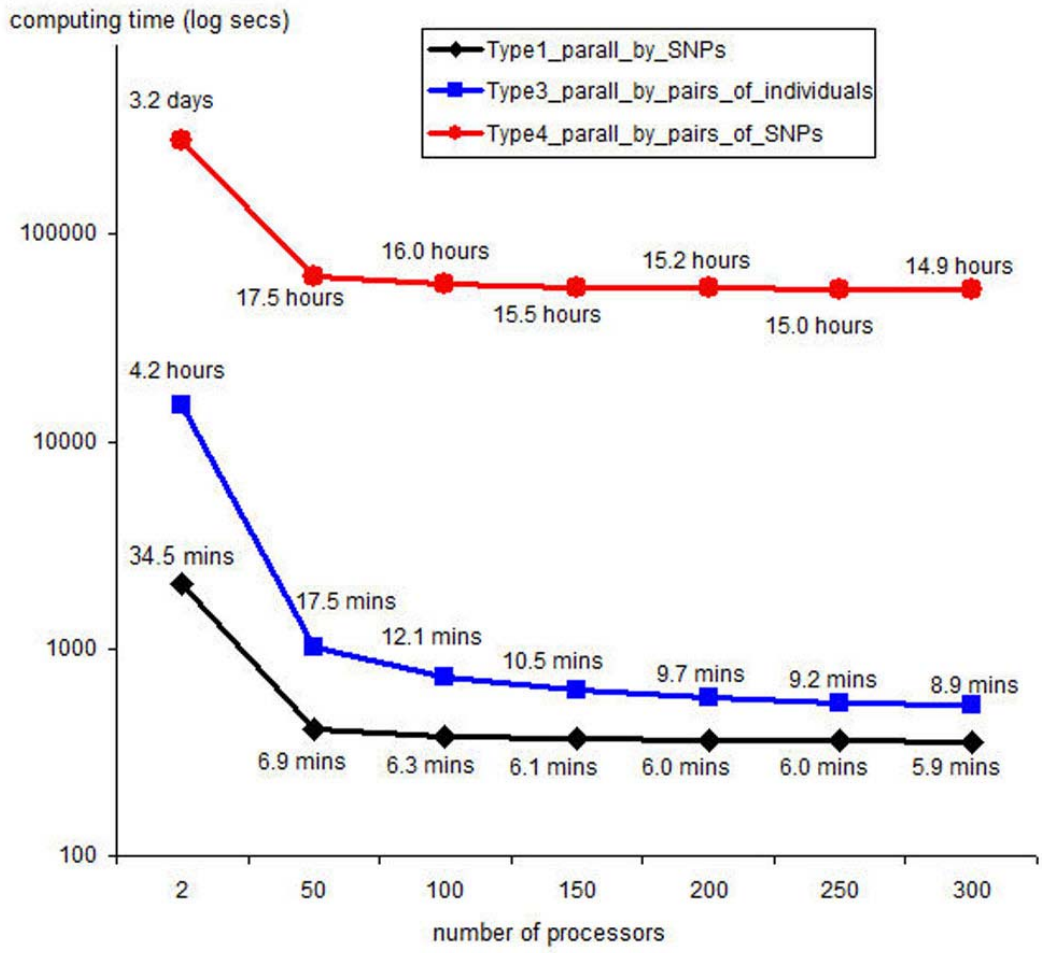


Figure 3.9: The computing time on a large cluster for Type1_parall_by_SNPs, Type3_parall_by_pairs_of_individuals and Type4_parall_by_pairs_of_SNPs extrapolated from Figure 3.8 applying the overhead equation.

3.4 Discussion and summary

We have presented the ParallABEL library which employs parallel computing to reduce computing time for data intensive tasks. ParallABEL can run on clustered computers that support LAM/MPI and R. With clustered computers, CPUs or even personal computers can be easily added as new compute nodes. ParallABEL runs on both distributed and shared memory architectures as it was developed with MPI. For a distributed memory architecture, MPI usually uses a computer network for task communications. For a shared memory architecture, MPI employs shared

variables instead of the network for task communications. This means that a distributed memory architecture may exhibit more overhead than a shared memory architecture (for example, eight single-core CPUs versus a single eight-core CPU). In our experiments, Type1_parall_by_SNPs took only 6 minutes to execute on a shared memory architecture but 14 minutes on a distributed memory architecture. The overhead of the shared memory architecture was tested on a server, which has 2 QUAD-CORE Intel Xeon(R) (2.8 GHz) CPUs and 8 GB. The server runs on CentOS version 5.4, and provides Open MPI version 1.4.1. Whereas, the overhead of the distributed memory architecture was measured on the computer cluster comprising of a front-end node and four compute nodes. Each node has 2 SINGLE-CORE Intel Xeon (2.8 GHz) CPUs and 4 GB RAM. Although the specification of the shared memory server is lower than the specification of the distributed memory cluster, its performance is still better than the performance of the distributed memory cluster.

ParallABEL allows the user to specify the number of CPUs employed for data execution. We expect the computational performance to increase linearly with the number of CPUs when using Type1_parall_by_SNPs, Type3_parall_by_pairs_of_individuals, and Type4_parall_by_pairs_of_SNPs. In addition, ParallABEL using multiple CPUs is faster than GenABEL using only one CPU. Computing times for Type3_parall_by_pairs_of_individuals and Type4_parall_by_pairs_of_SNPs are longer than those for Type1_parall_by_SNPs because the input data are pairs of individuals and SNPs respectively, which are much larger than the SNPs input for Type1_parall_by_SNPs. In addition, if the number of SNPs is n, then the number of inputs for Type1_parall_by_SNPs will be n but the number of inputs data for Type4_parall_by_pairs_of_SNPs will be n*n. ParallABEL can save much more computational time when utilizing Type3_parall_by_pairs_of_individuals and Type4_parall_by_pairs_of_SNPs than when using Type1_parall_by_SNPs. Therefore, as the amount of input data increases, the time saved by ParallABEL also increases. ParallABEL does not only reduce the computing time but also is as easy-to-use as the more conventional GenABEL.

ParallABEL can not reduce the computing time when the data size is too small, such as the result shown when employing the hom function of Type2_parall_by_individuals, because the computing time is too short. In that case,

the overheads of data partitioning and output merging overwhelm the computational performance.

CHAPTER 4

Speeding up SNP Interaction Analyses Based on Logic Regression Applying Parallel Computing for GWA Studies

4.1 Introduction

Logic regression, developed by Ruczinski et al., is a flexible method of regression with Boolean combinations of binary covariates as explanatory variables [37]. It has certain advantages over other analyses, such as Classification and Regression Trees (CART) [38] and random forests [39], which relate only the main effects and simple (two to three-way at most) interactions between predictors. The strength of logic regression is its capacity for finding complex interactions between predictors. Logic regression can be applied to various regression/classification problems, one of which is the analysis of Single Nucleotide Polymorphism (SNP) interactions.

SNPs refer to genetic variations at the single nucleotide level. There are more than one million SNPs in the human genome. From a large set of SNP measurements, finding SNPs whose variations are associated with a disorder is an important analytic goal of bioinformatics. Such analyses can help researchers discover genes that predispose individuals to a higher risk of the disorder. In addition, SNP analyses may assist researchers to explain possible heterogeneity in individuals' responses to a certain medicine [40].

Schwender and Ickstadt suggested that it is usually not an individual SNP that plays an imperative role in the risk of a complex disorder. Rather, it is SNP interactions that strongly influence the risk of a complex disorder [41]. This suggests that SNP interactions may identify high risk groups [42], to whom an intervention strategy for decreasing the risk or detecting the disorder early for treatment may be considered. Logic regression can be employed to search for multi-way SNP interactions, e.g., 4-way interactions: such an analysis is often difficult with other methods including random forests, CART, and Support Vector Machines (SVMs) [41,

43]. For this reasons, logic regression is a powerful methodology for identifying SNPs interactions associated with risks of complex disorders.

LogicReg is an R library for logic regression analyses [13] implemented in R [11], a well-known open source statistics programming language and environment. To allow a large number of permutation rounds for a large dataset such as ones from GWAS, it is advantageous to create an R-library that allows parallel computation of logic regression. For instance, for the gene-level SNP analysis of Crohn's disease dataset from the Wellcome Trust Case Control Consortium (WTCCC) including approximately 13,500 genes, we need more than 400,000 runs of logic regressions when SNP interactions within each gene must be analyzed with thirty permutations. Moreover, to the size of the dataset is large: for example, the WTCCC Crohn's disease dataset includes 4,680 individuals. Accordingly, without parallel computing, the logic regression analysis requires massive computing time, hours to months, depending on the size of the dataset being analyzed and the computer capacities.

Possible ways to speed up the computing time of any program include editing of the algorithm and using parallel computing. Since it is not simple to alter the logic regression algorithm, the best way to speed up the logic regression analysis is to employ parallel computing. Parallel computing is a useful methodology, enabling concurrent handling of multiple computing resources to gain computational throughput. In parallelization, a problem or a job is partitioned into unassociated smaller tasks including series of commands. Each task will then be executed freely using multiple Central Processing Units (CPUs) on compute nodes of a cluster; after that, one of these CPUs on the front-end node will combine the outputs from all tasks [44].

Rmpi [15], an R library, supports many functions to parallelize broken tasks on R using the MPI (Message Passing Interface) [16]. Besides, Rmpi can be applied to control the flow of computation in a parallel environment with both single-core CPUs and multi-core CPUs, and on a single computer or on a computer cluster. Nonetheless, it is arduous for general users to write a parallel Rmpi program including partitioning and distributing data, controlling and monitoring tasks, and merging output files.

In this chapter, we propose a development of ParallLogicReg, a new R library for parallelization of logic regression analyses using Rmpi. ParallLogicReg aims not only to accelerate the computation of logic regression analyses, but also simplify analysis parallelization. Moreover, using ParallLogicReg, users do not need to be proficient with parallel programming because it will automatically partition and distribute data, control and monitor tasks across the computers, and merge output files.

4.2 Methods

4.2.1 Logic regression analyses

Logic regression aims to find Boolean combinations of the predictors. We consider that all predictors are binary (0 or 1, yes or no), for identification of SNP associations. Specifically, the predictor $X_i = 1$ if the i^{th} SNP has a certain genotype, and $X_i = 0$ otherwise. Each Boolean combination of SNPs could use three operators, □ (AND), □ (OR), and c (NOT) to form a logic expression, L_j, $j = 1,\ldots, t$ such as:

$$L_j = (X_1 \square X_2) \square X_3^c$$

This example of Boolean logic expression means:

$$L_j = (SNP_1 \square SNP_2) \square SNP_3^c$$

Figure 4.1 shows the example of SNP interaction associations between SNP_1 and SNP_2 that both SNP_1 and SNP_2 high risk; or either SNP_1 or SNP_2 high risks.

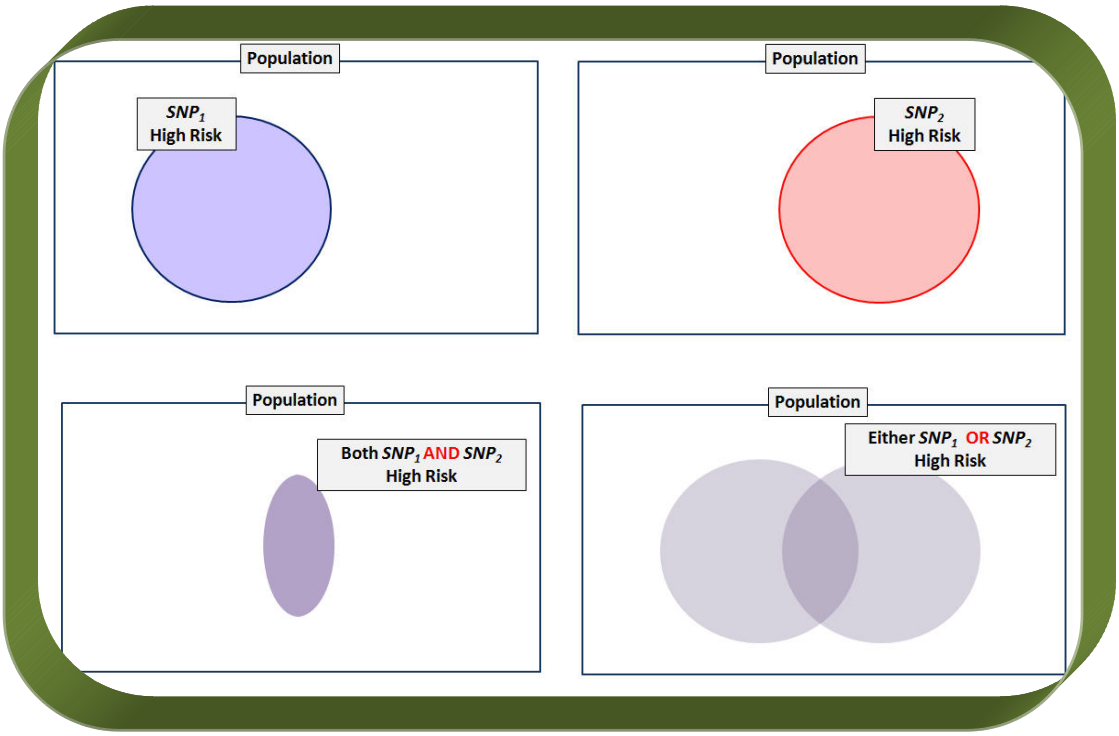


Figure 4.1: SNP interaction associations between SNP_1 and SNP_2 [45].

Logic regression uses L's instead of X's in its linear predictor and takes the form:

$$f(E[Y]) = \beta_0 + \sum_{j=1}^{t} \beta_j L_j$$

where Y is a response variable, f is a link function, and parameters $\beta_j, j = 0,\ldots, t$ are concurrently estimated with the search for the Boolean expressions L_j's in the above equation that minimizes the scoring function related with this model type [37].

4.2.2 Data partitioning and distribution

The computation of logic regression is demanding as it explores a large space for an optimal set of logics and needs a large number of permutation tests to assess signals in the data. Hence, parallel computing is very important as it decreases the computing time. To parallelize a logic regression analysis, ParallLogicReg

running on the front-end node automatically partitions the input dataset into G subsets, where G is the number of genes to be analyzed. The ‘task full’ approach [34] is used to keep load balancing when ParallLogic is being executed. Also, this approach is not sensitive to the number of CPUs or compute nodes .The front-end node sends these subsets to idle CPUs on compute nodes. The example of data partitioning and distribution are shown in Figure 4.2. If there are four compute nodes, and each compute node has only one CPU, SNPs of G_1 - G_4 ($Gene_1$ - $Gene_4$) will separately be executed on these compute nodes as shown in Figure 4.2A. When the execution of the second compute node has finished, the front-end node will send the SNPs of the next gene (G_5) to it – a cycle that proceeds until all the genes are sent as shown in Figure 4.2B. After all the compute nodes has finished their tasks, the front-end node will combine all the outputs automatically.

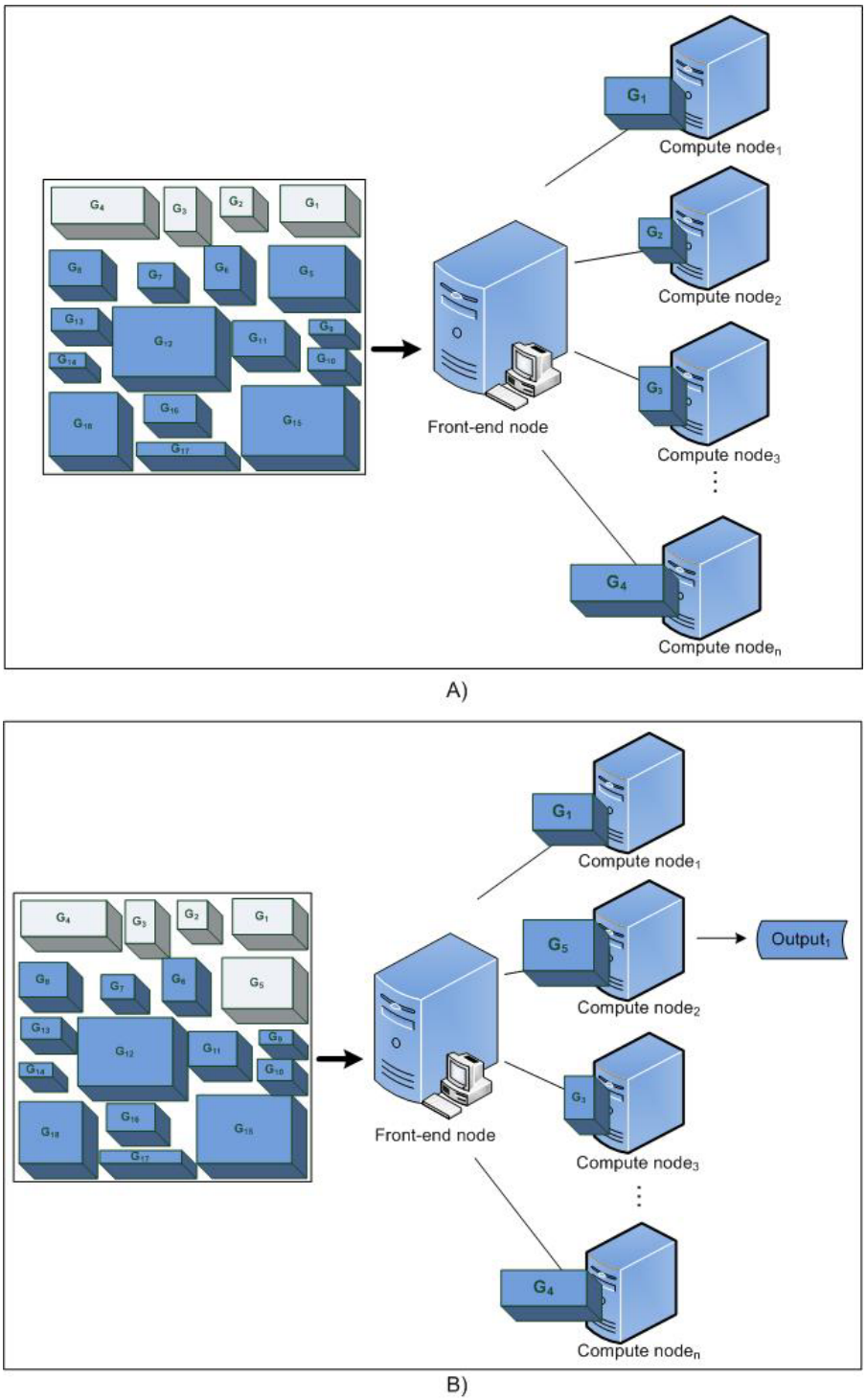


Figure 4.2: A) The data is partitioned into fifteen subsets, and each subset contains SNPs of a gene. G_1 - G_4 subsets will then be executed on different compute nodes. B) The G_5 is sent to be executed on the second compute node after the execution of G_1 has finished.

4.2.3 Implementation

The sequential workflow for a logic regression analysis on a single CPU/computer is shown in Figure 4.3A. The genotype and phenotype data are analyzed by the LogicReg library, working under the R program. LogicReg sequentially analyzes the raw data, and produces statistical data (e.g., deviance) as outputs.

Since this sequential workflow generally takes great computing time to conduct statistical analyses, we have developed a novel parallel workflow for ParallLogicReg to save the computing time. The novel parallel workflow in a logic regression analysis is shown in Figure 4.3B. A job scheduler such as the SUN Grid Engine [32] distributes the genotype and phenotype data to each compute node on a cluster to queue jobs and reserve a set of CPUs required by the employed MPI (Message Passing Interface) library such as LAM/MPI (Local Area Multicomputer/Message Passing Interface) [33] and Open MPI [46]. The MPI library has various functions called by Rmpi to parallelize R functions. ParallLogicReg uses this Rmpi library to parallelize LogicReg. In addition, ParallLogicReg partitions a job into several smaller tasks on a front-end node using basic R commands and distributes them with genotype and phenotype data to the reserved CPUs using Rmpi. These CPUs execute the tasks on compute nodes and call the LogicReg. Later, the outputs will return to Rmpi and be combined by ParallLogicReg on the front-end node. The statistical data from the parallel workflow can be approved by comparison with the statistical data from the sequential workflow. ParallLogicReg can run on any Operating System supporting components for the parallel workflow such as Linux and Solaris.

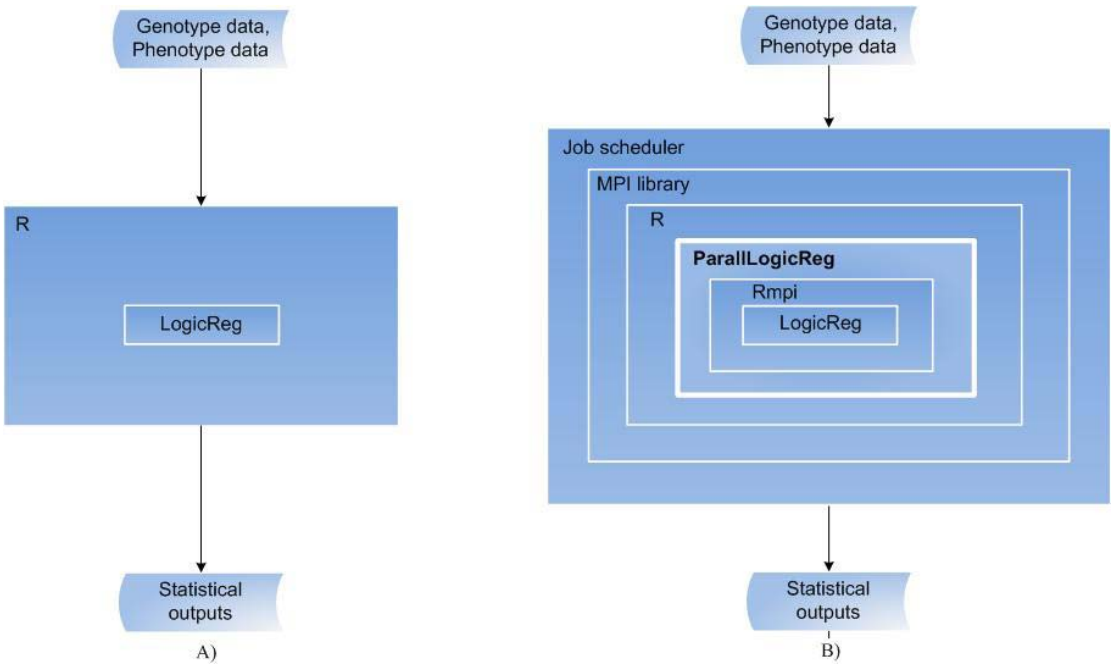


Figure 4.3: A) Sequential logic regression computing workflow runs on a single CPU or a computer. B) Parallel logic regression computing workflow runs on a multiple CPUs or a set of computers.

ParallLogic is designed based on SIMD parallel computing and distributed memory/message passing programming model and can be executed on shared memory architecture and distributed memory architecture. Manual parallelization is applied to design and develop ParallLogic.

Users can easily use ParallLogicReg to parallelize logic regression function. The executable command that parallelizes logic regression on multiple CPUs is shown in Figure 4.4. To run the function, the number of CPUs can be specified in the Sun Grid Engine.

```
library(ParallLogicReg)

resp=c(rep(0,2935),rep(1,1745))        # number of controls = 2,935; number of cases = 1,745

nperm=20                               # number of permutations
niter=20                               # number of iterations
begin=1                                # the first gene id that will be run
end=10                                 # the last gene id that will be run
infile=("A01Chro1.txt")                # data will be run

output = ParallLogicReg(infile=infile,resp=resp,begin=begin,end=end,nperm=nperm,niter=niter)
```

Figure 4.4: The example of parallel execution used to analyze Crohn's disease data when gene ids were between one and ten, and the numbers of permutations and iterations were twenty. The data contained 1,745 cases and 2,935 controls. Besides, the user could set the number of CPUs in a job scheduler such as the SUN Grid Engine.

4.3 Results

A computer cluster, Hanuman, has been used to evaluate the performance of ParallLogicReg. This cluster includes five IBM servers XSeries 3362, which are comprised by a front-end node and four compute nodes, with two SINGLE-CORE Intel Xeon (2.8 GHz) CPUs and four GB RAM, respectively. The front-end node of the cluster can be connected via the Internet, and can control the compute nodes of the cluster through an Ethernet switch. Also, this cluster provides Rocks Cluster Distribution version 4.3 [36] including the SUN Grid Engine version 4.3 [32] , LAM/MPI version 7.1.2, R program version 2.8.1, Rmpi library version 0.5-6, LogicReg version 1.4.9 and ParallLogicReg 1.0. Crohn's disease data set, a chronic inflammatory disease data set of the intestines [18] which contains 1,745 controls and 2,935 cases with approximately 2,000 SNPs, was used to measure the performance of ParallLogicReg.

Results from logic regression using ParallLogicReg function for the Crohn's disease data are shown in Figure 4.5 with twenty permutations and iterations. ParallLogicReg saved the computing time, especially with eight CPUs. For example,

on a single CPU, the two hundreds gene analyses on the first chromosome took 7.3 days, but took only 0.9 day with eight CPUs.

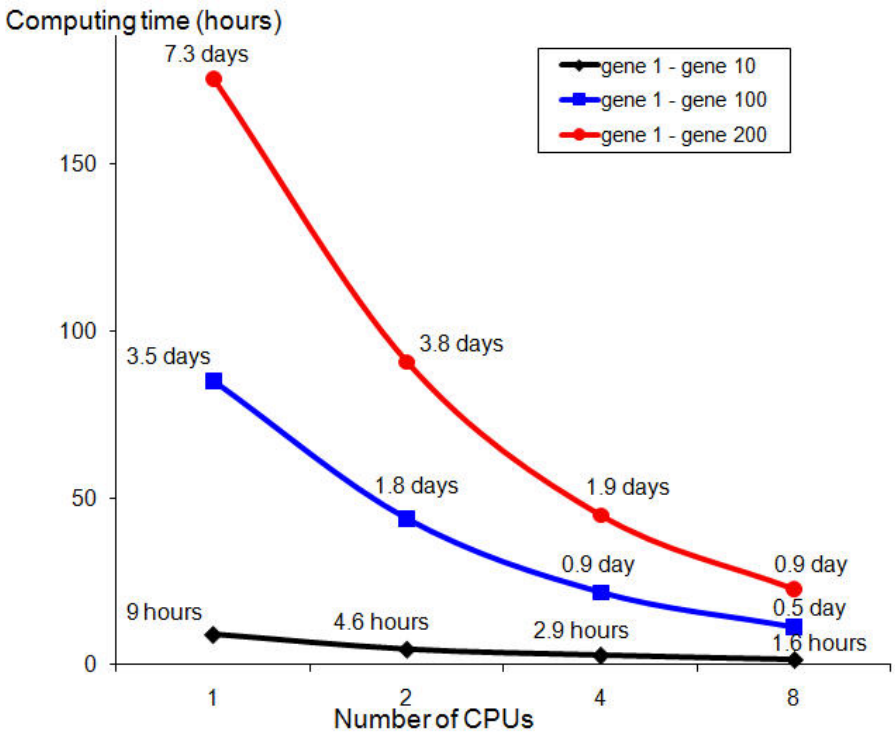


Figure 4.5: Ten, one hundred and two hundred genes of Crohn's disease data were executed with ParallLogicReg. Also, these genes were running on one, two, four and eight CPUs. The results showed that the more CPUs, the more computing times were saved by ParallLogicReg.

If the number of available CPUs is P, the computing time for P CPUs is $time_P$, and the sequential computing time for a CPU is $time_1$, thus, the speedup for P CPUs will be:

$$speedup_P = time_1 / time_P$$

The speedups of analyzing Crohn's disease data using ParallLogicReg function applying the above equation are shown in Figure 4.6. It shows that the saved time by ParallLogicReg is linearly correlated to the number of CPUs. For instance,

the executing speed of the two hundreds gene analyses on eight CPUs was approximately eight times faster than that on only one CPU.

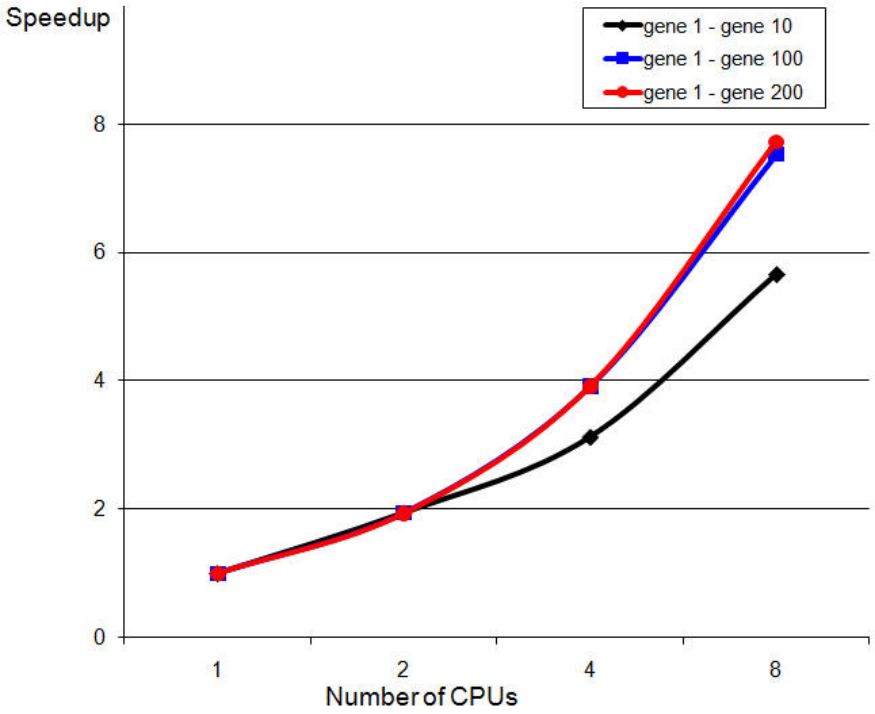


Figure 4.6: The speedups were extrapolated from Figure 4.5 applying the speedup equation. The speedups showed that the more CPUs, the more speedup were increased by ParallLogicReg.

4.4 Discussion and summary

We have developed a novel R-library called ParallLogicReg to speed up logic regression analyses using parallel computing components which consist of a job scheduler, a MPI library, an Rmpi library and a LogicReg library. ParallLogicReg has been designed to be a user-friendly library. Identification of SNPs associated with Crohn's disease is used to measure the performance of the ParallLogicReg function. The results showed that ParallLogicReg using parallel computing can save the computing time for analyzing massive data. The statistical data from ParallLogicReg with the number of CPUs is the same as the statistical data from non-parallel method

because ParallLogicReg partitions data into small subsets which have no effect for logic regression analyses.

According to speedup equation, the overhead for *P* CPUs is

$$overhead_P = time_p\text{-}(time_1 / P)$$

Since ParallLogicReg is not sensitive to the number of CPUs, it can be run on a large cluster. The overhead of analyses with various numbers of CPUs on a large computer cluster can be predicted based on the overhead of eight CPUs as shown in Figure 4.7.

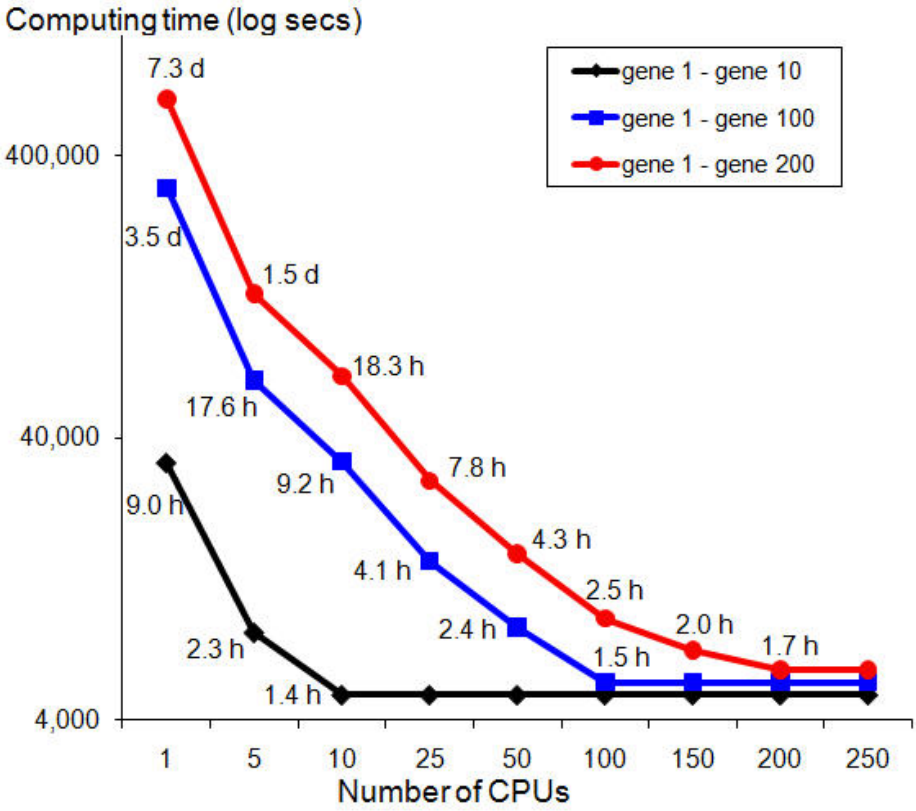


Figure 4.7: The computing time on a large cluster for ten, one hundred and two hundred genes analyses were extrapolated from Figure 4.5 applying the overhead equation.

The computing time on a large cluster for ten, one hundred and two hundred analyses extrapolated from Figure 4.5 applying the above overhead equation are shown in Figure 4.8.

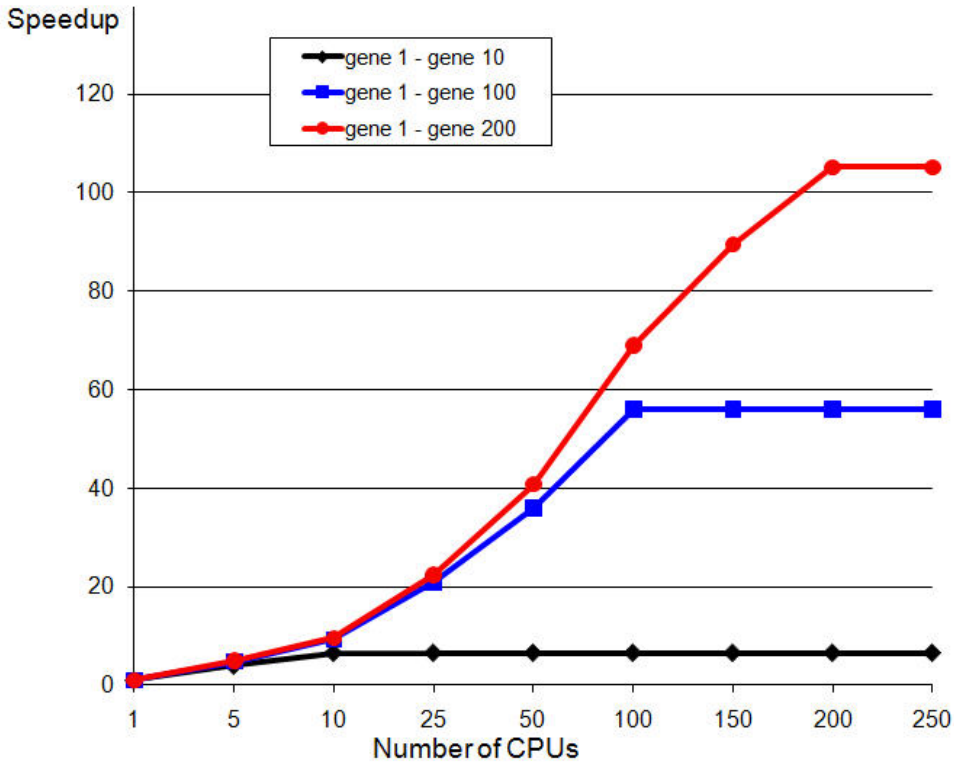


Figure 4.8: The speedups on a large cluster for ten, one hundred and two hundred genes analyses were extrapolated from Figure 4 applying the overhead equation and speedup equation.

The time-saving rates are grown when the numbers of CPUs are increased until the numbers of CPUs are greater than number of genes. Thus, with bigger data, the time-saving rates will be larger in a large computer cluster. Users can set the number of CPUs in ParallLogicReg to execute data, which will reduce the computing time that growingly correlates to the number of CPUs. Also, if the user applies more CPUs, more computing time will be saved by ParallLogicReg. Nonetheless, the user should optimize the number of CPUs suitable for the computational throughput. In particular, the number of CPUs assigned should be less than, or equal to, the number of genes to avoid idling CPUs. Due to the benefit of MPI, ParallLogicReg can be run not only on a distributed memory architecture like the architecture of Hanuman but also on a shared memory architecture. Nevertheless,

a distributed memory architecture produces more overhead than a shared memory architecture.

CHAPTER 5

Conclusions and Furture Work

5.1 Conclusions

Genome-Wide Association (GWA) analysis is a powerful method for identifying loci associated with complex genetic traits such as Crohn's disease, Type I Diabetes Mellitus (DM) and Type II DM. Parts of GWA analyses, especially those involving interactions or pair-wise analysis of thousands individuals or millions genetic markers consuming hours to months of computation time, will benefit from parallel computation. However, it is arduous acquiring the necessary programming skills to correctly partition and distribute data, control and monitor tasks on multiple CPUs, and merge output files.

ParallABEL and ParallLogicReg have been presented to improve the performance of GWA analyses by applying parallel computing. With ParallABEL and ParallLogicReg libraries, users can immensely accelerate the computing time of GWA analyses. For example, the computing time of the Rheumatoid Arthritis data set for the identity-by-state matrix was theoretical reduced from approximately eight hours to one hour when ParallABEL employed eight processors. Another instance, with two hundred genes and twenty permutation rounds, the computing time of the Crohn's disease data set was decreased from about seven days to only one day when ParallLogicReg applied eight CPUs.

The users can execute ParallABEL and ParallLogicReg to parallelize GWA analyses without having the advanced programming skills including partitioning and distributing data, controling and monitoring tasks, and merging output files**.** Moreover, expert users have no waste of time to develop the libraries. ParallABEL is a user-friendly parallelization of GenABEL whereas ParallLogicReg is a user-friendly parallelization of LogicReg for GWAS analyses. The statistical outputs from both libraries with any number of CPUs are valid as the statistical data from non-parallel approach (GenABEL and LogicReg). Both novel libraries can be

executed not only on multi-core CPUs on a single computer but also on multi-core CPUs or single-core CPU distributed across many computers (a computer cluster). Nevertheless, the computers must support Rmpi running under a MPI library such as LAM/MPI and Open MPI.

Since ParallABEL and ParallLogicReg can produce statistical outputs of GWA analyses faster than the conventional approach, users can save time to find genes referred to diseases. Besides, the more CPUs, the more finding times were saved by ParallABEL and ParallLogicReg. Researchers can use the information to develop better strategies to detect, treat and prevent the diseases more quickly than before.

The user can specify the number of processors employed for data execution in ParallABEL and ParallLogicReg. With ParallABEL, users could expect the computational performance of GWA analyses to linearly increase with the number of processors when using the functions of ParallABEL to compute the SNP characterization statistics, the pair-wise individuals statistics and the pair-wise SNPs statistics. In addition, ParallABEL using multiple CPUs is faster than GenABEL using only one processor. Computing times for the pair-wise individuals statistics and the pair-wise SNPs statistics are longer than those for the the SNP characterization statistics because the input data is pairs of individuals and SNPs respectively, which are much larger than the SNPs input for the SNP characterization statistics. Also, if the number of SNPs is n, then the number of inputs for computation of the SNP characterization statistics will be n but the number of input data for computation of the pair-wise SNPs statistics will be n*n. ParallABEL can save much more computational time when producing the pair-wise individuals statistics and the pair-wise SNPs statistics than when producing the SNP characterization statistics. Therefore, as the amount of input data increases, the time saved by ParallABEL also increases. However, ParallABEL can not reduce the computing time when the data size is too small, such as the result shown when employing the hom function (an individual characterization statistic) because the computing time is too short. In that case, the overheads of data partitioning and output merging overwhelm the computational performance. With ParallLogicReg, the time-saving rate grows when the number of CPUs increases until the number of CPUs is greater than the number of

genes. Therefore, the number of CPUs should be less than, or equal to, the number of genes in order to avoid idling CPUs. Nonetheless, the user should optimize the number of CPUs suitable for the gained computational throughput.

ParallABEL and ParallLogicReg can process not only the Rheumatoid Arthritis data set and Crohn's disease data set but also other disease data sets such as a neck cancer data set. In addition, the user can use statistical outputs from ParallABEL and ParallLogicReg to find genes associated the other diseases such as the neck cancer.

5.2 Furture work

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable computability in the cloud. It supports users with complete control of their requiring resources (a large computer cluster) and lets the users run on Amazon’s proven computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes. It also allows the users to quickly scale capacity both up and down when their computing requirements change. Amazon EC2 changes the economics of computing by allowing the users to pay only for capacity that the users actually use [47]. If ParallABEL and ParallLogicReg can run on Amazon EC2, the computing time of GWA analyses will be much saved. Nevertheless, a trouble of executing of ParallABEL and ParallLogicReg on Amazon EC2 may occur since we still do not exactly know about infrastructure of Amazon EC2. Therefore, we will intensively check the infrastructure before running of ParallABEL and ParallLogicReg on Amazon EC2.

REFERENCES

[1] *Genome-Wide Association Studies*. Available: http://www.genome.gov/20019523

[2] *Largest ever study of genetics of common diseases published today*. Available: http://www.wtccc.org.uk/info/070606.shtml

[3] "Genome-wide association study of 14,000 cases of seven common diseases and 3,000 shared controls," *Nature,* vol. 447, pp. 661-78, Jun 7, 2007.

[4] K. Christensen and J. Murray, "What genome-wide association studies can do for medicine," *N Engl J Med,* vol. 356, 2007.

[5] *Database of Single Nucleotide Polymorphisms*. Available: http://www.ncbi.nlm.nih.gov/snp

[6] D. L. Wheeler*, et al.*, "Database resources of the National Center for Biotechnology Information," *Nucleic Acids Res,* vol. 36, pp. D13-21, Jan 2008.

[7] A. D. Johnson and C. J. O'Donnell, "An open access database of genome-wide association results," *BMC Med Genet,* vol. 10, p. 6, 2009.

[8] J. N. Hirschhorn and M. J. Daly, "Genome-wide association studies for common diseases and complex traits," *Nat Rev Genet,* vol. 6, pp. 95-108, Feb 2005.

[9] T. A. Pearson and T. A. Manolio, "How to Interpret a Genome-wide Association Study," *The Journal of the American Medical Association,* vol. 299, pp. 1335-1344, 2008.

[10] Y. S. Aulchenko*, et al.*, "GenABEL: an R library for genome-wide association analysis," *Bioinformatics,* vol. 23, pp. 1294-6, May 15, 2007.

[11] *The Comprehensive R Archive Network (CRAN)*. Available: http://www.r-project.org/

[12] R. Ihaka and R. Gentleman, "R: A language for data analysis and graphics," *Journal of Computational and Graphical Statistics,* vol. 5, pp. 299-314, 1996.

[13] C. Kooperberg and I. Ruczinski. Mar 23, 2011). *LogicReg: Logic Regression*. Available: http://cran.r-project.org/web/packages/LogicReg/index.html

[14] *Introduction to Parallel Computing*. Available: https://computing.llnl.gov/tutorials/parallel_comp/

[15] *Rmpi: Interface (Wrapper) to MPI (Message-Passing Interface)*. Available: http://www.stats.uwo.ca/faculty/yu/Rmpi/

[16] *Message-Passing Interface Forum (MPI)*. Available: http://www.mpi-forum.org/

[17] R. M. Plenge, *et al.*, "TRAF1-C5 as a risk locus for rheumatoid arthritis--a genomewide study," *N Engl J Med,* vol. 357, pp. 1199-209, Sep 20, 2007.

[18] M. Parkes, *et al.*, "Sequence variants in the autophagy gene IRGM and multiple other replicating loci contribute to Crohn's disease susceptibility," *Nat Genet,* vol. 39, pp. 830-2, Jul 2007.

[19] B. Barney. (2009, Apr 21,2009). Introduction to Parallel Computing. Retrieved on Apr 21, 2009. Available: https://computing.llnl.gov/tutorials/parallel_comp/

[20] J. Altmüller, *et al.*, "Genomewide Scans of Complex Human Diseases: True Linkage Is Hard to Find," *Am J Hum Genet,* vol. 69, p. 1413, 2001.

[21] "A haplotype map of the human genome," *Nature,* vol. 437, pp. 1299-320, Oct 27, 2005.

[22] K. A. Frazer, *et al.*, "A second generation human haplotype map of over 3.1 million SNPs," *Nature,* vol. 449, pp. 851-61, Oct 18, 2007.

[23] L. A. Hindorff, *et al.*, "Potential etiologic and functional implications of genome-wide association loci for human diseases and traits," *Proc Natl Acad Sci U S A,* vol. 106, pp. 9362-7, Jun 9, 2009.

[24] R. H. Duerr, *et al.*, "A genome-wide association study identifies IL23R as an inflammatory bowel disease gene," *Science,* vol. 314, pp. 1461-3, Dec 1, 2006.

[25] H. Mishima, *et al.*, "Application of the Linux cluster for exhaustive window haplotype analysis using the FBAT and Unphased programs," *BMC Bioinformatics,* vol. 9 Suppl 6, p. S10, 2008.

[26] N. M. Laird, *et al.* (2000, Implementing a unified approach to family-based tests of association. *Genet Epidemiol 19 Suppl 1,* S36-42 [http://biosun1.harvard.edu/~fbat/fbat.htm]. Available: http://biosun1.harvard.edu/~fbat/fbat.htm

[27] F. Dudbridge, "Pedigree disequilibrium tests for multilocus haplotypes," *Genet Epidemiol,* vol. 25, pp. 115-21 [http://portal.litbio.org/Registered/Help/unphased/], Sep 2003.

[28] G. Vera*, et al.*, "R/parallel--speeding up bioinformatics analysis with R," *BMC Bioinformatics,* vol. 9, p. 390, 2008.

[29] K. Misawa and N. Kamatani, "ParaHaplo: A program package for haplotype-based whole-genome association study using parallel computing," *Source Code Biol Med,* vol. 4, p. 7, 2009.

[30] L. Ma*, et al.*, "Parallel and serial computing tools for testing single-locus and epistatic SNP effects of quantitative traits in genome-wide association studies," *BMC Bioinformatics,* vol. 9, p. 315, 2008.

[31] J. Hill*, et al.*, "SPRINT: a new parallel framework for R," *BMC Bioinformatics,* vol. 9, p. 558, 2008.

[32] *Sun Grid Engine*. Available: http://www.rocksclusters.org/roll-documentation/sge/5.4/using-sge.html

[33] *Local Area Multicomputer/Message Passing Interface*. Available: http://www.lam-mpi.org/

[34] *Rmpi Program Structure*. Available: http://math.acadiau.ca/ACMMaC/Rmpi/structure.html

[35] "Task pull method."

[36] *Rocks Cluster Distribution*. Available: http://www.rocksclusters.org/wordpress/

[37] I. Ruczinski*, et al.*, "Logic regression," *Journal of Computational and Graphical Statistics,* vol. 12, pp. 475-511, Sep 2003.

[38] L. Breiman, *Classification and regression trees*. Belmont, Calif.: Wadsworth International Group, 1984.

[39] L. Breiman, "Random forests," *Machine Learning,* vol. 45, pp. 5-32, Oct 2001.

[40] Mar 9,2011). *Genetics Home Reference*. Available: http://ghr.nlm.nih.gov/handbook/genomicresearch/snp

[41] H. Schwender and K. Ickstadt, "Identification of SNP interactions using logic regression," *Biostatistics,* vol. 9, pp. 187-98, Jan 2008.

[42] S. Garte, "Metabolic susceptibility genes as cancer risk factors: time for a reassessment?," *Cancer Epidemiol Biomarkers Prev,* vol. 10, pp. 1233-7, Dec 2001.

[43] I. Guyon*, et al.*, "Gene selection for cancer classification using support vector machines," *Machine Learning,* vol. 46, pp. 389-422, 2002.

[44] U. Sangket*, et al.*, "ParallABEL: an R library for generalized parallelization of genome-wide association studies," *BMC Bioinformatics,* vol. 11, p. 217, 2010.

[45] Y. Yutaka, "Crohn's Disease GWAS Gene-level logic regression analysis," 2011.

[46] *Open MPI*. Available: http://www.open-mpi.org/

[47] October 24 th, 2011). *Amazon Elastic Compute Cloud (Amazon EC2)*. Available: http://aws.amazon.com/ec2/

APPENDICES

Appendix A

Publication

Sangket et al. BMC Bioinformatics 2010, 11:217
http://www.biomedcentral.com/1471-2105/11/217

BMC Bioinformatics

SOFTWARE Open Access

ParallABEL: an R library for generalized parallelization of genome-wide association studies

Unitsa Sangket*[1], Surakameth Mahasirimongkol[2], Wasun Chantratita[3], Pichaya Tandayya[4] and Yurii S Aulchenko[5,6]

Abstract

Background: Genome-Wide Association (GWA) analysis is a powerful method for identifying loci associated with complex traits and drug response. Parts of GWA analyses, especially those involving thousands of individuals and consuming hours to months, will benefit from parallel computation. It is arduous acquiring the necessary programming skills to correctly partition and distribute data, control and monitor tasks on clustered computers, and merge output files.

Results: Most components of GWA analysis can be divided into four groups based on the types of input data and statistical outputs. The first group contains statistics computed for a particular Single Nucleotide Polymorphism (SNP), or trait, such as SNP characterization statistics or association test statistics. The input data of this group includes the SNPs/traits. The second group concerns statistics characterizing an individual in a study, for example, the summary statistics of genotype quality for each sample. The input data of this group includes individuals. The third group consists of pair-wise statistics derived from analyses between each pair of individuals in the study, for example genome-wide identity-by-state or genomic kinship analyses. The input data of this group includes pairs of SNPs/traits. The final group concerns pair-wise statistics derived for pairs of SNPs, such as the linkage disequilibrium characterisation. The input data of this group includes pairs of individuals. We developed the ParallABEL library, which utilizes the Rmpi library, to parallelize these four types of computations. ParallABEL library is not only aimed at GenABEL, but may also be employed to parallelize various GWA packages in R. The data set from the North American Rheumatoid Arthritis Consortium (NARAC) includes 2,062 individuals with 545,080, SNPs' genotyping, was used to measure ParallABEL performance. Almost perfect speed-up was achieved for many types of analyses. For example, the computing time for the identity-by-state matrix was linearly reduced from approximately eight hours to one hour when ParallABEL employed eight processors.

Conclusions: Executing genome-wide association analysis using the ParallABEL library on a computer cluster is an effective way to boost performance, and simplify the parallelization of GWA studies. ParallABEL is a user-friendly parallelization of GenABEL.

Background

GWA analysis [1] is a well established and powerful method for identifying loci associated with variations of complex genetic traits such as common diseases. Hundreds of new genes have been implicated in human health and disease during the last few years in various GWA studies[2]. In a typical study, hundreds of thousands, or millions, of single-nucleotide polymorphisms (SNPs) are typed in thousands of individuals in order to detect genetic risk factors.

GenABEL is a specialized library package for GWA analysis [3] implemented in R, an open source statistics programming language and environment [4,5]. GenABEL enables GWA analysis to be done using a regular desktop computer due to its efficient data storage and memory management. Nevertheless, analysis of very large data sets are computationally challenging and may take hours or weeks to complete. Examples include the utilization of sophisticated adjustments for population stratification and relationship structures, the estimation of linkage dis-

* Correspondence: usangket@yahoo.com
[1] Center for Genomics and Bioinformatics Research, Faculty of Science, Prince of Songkla University, Songkhla, 90112, Thailand
Full list of author information is available at the end of the article

equilibriums and the calculation of genome-wide identity-by-state, haplotypic tests, and permutation analyses.

To increase the computational throughput, a user can partition their data into sets, and perform the analysis of the sets across a network of computers; a concept known as parallel and/or distributed computing. However, performing such analysis requires high levels of computer expertise. The user needs sufficient programming skills to partition and distribute data, control and monitor tasks across the computers, and merge output files. Occasionally, a data set may fail to be processed, e.g. if the user did not partition the data into small enough subsets to be processed on a particular machine. Also, the outputs from the computers may be scattered and their order hard to follow.

Parallel computing is an intuitive, and powerful, method for increasing computational throughput. A task is separated into smaller tasks, and each is processed independently, in parallel, using multiple Central Processing Units (CPUs) or a cluster of computers. The outputs from each task must later be merged [6]. A general architecture for parallel computing is shown at Figure 1. Most tasks solved in GWA analysis are suitable for parallelization, due to their computational independency, with parallelization achieved at the data level. For example, association tests can usually be done separately for each SNP and/or a small group of SNPs. Consequently, parallelization is a beneficial way to reduce the computing time, with few overheads incurved in large-scale GWA analyses.

Several attempts had been made to parallelize genetic association analyses. Grid Engine, a cutting-edge parallel tool, can schedule parallel tasks involving genetic association analysis programs [7] such as FBAT [8] and UNPHASED [9]. The approach, first proposed by Mishima et al., is based on non-parallel code combined through process-based parallelization. The downside is that the user still needs to monitor when each task is finished, and when the outputs from all the tasks can be merged. Moreover, each process may take a very long time to finish, and load balance can be problematic. A granularity problem (a high computation to communication ratio) may occur, but higher power compute-nodes or code parallelization are possible solutions. The R/parallel package has been used to automate loop parallel execution, but the application must run on a single computer with multi-core processors, and does not currently support cluster computing [10]. Its inclusion would allow the computing time limit of the package to be eliminated. Misawa and Kamatani [11] developed the ParaHaplo

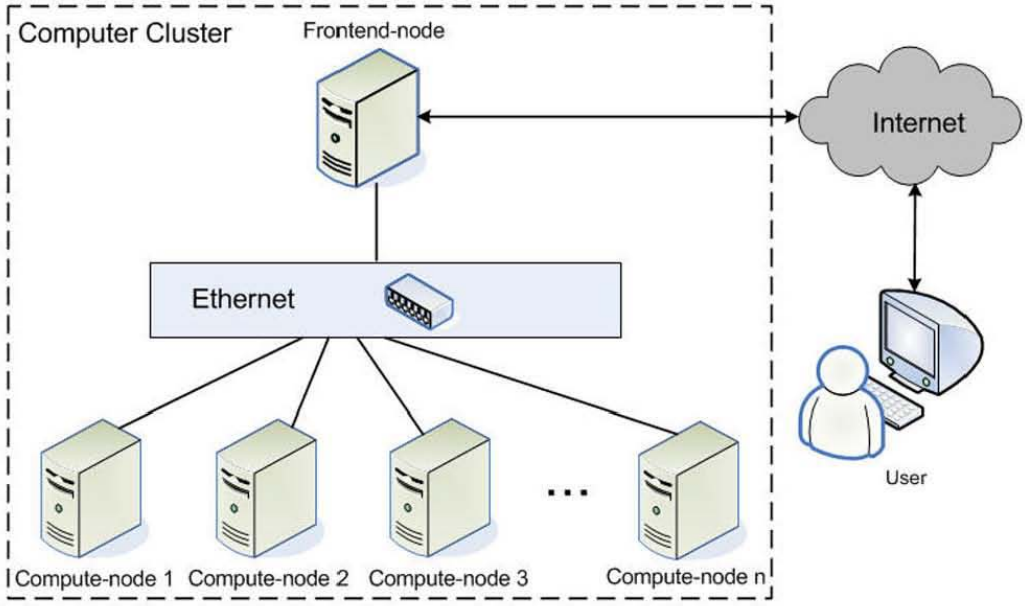


Figure 1 Computer Cluster Architecture. The user can submit tasks to the cluster of computers via the Internet. Once the user submits a job to the computer cluster, the frontend-node schedules and distributes the smaller partitioned tasks to the compute-nodes. The output from each compute-node is merged by the frontend-node.

package for haplotype-based whole-genome association studies using parallel computing. It is aimed at correcting multiple comparisons in multiple SNP loci in linkage disequilibrium. There are other statistical analyses requirements in GWA studies, such as obtaining statistics for a particular SNP or a trait, association test, characterizing an individual in the study, and pair-wise statistics between individuals. Furthermore, Ma et al. [12] developed EPISNPmpi, a parallel system for epistasis testing in large scale GWA analysis.

Rmpi [13] is an R library which provides various functions to parallelize tasks on R using the MPI (Message-Passing Interface) [14]. Rmpi employs various functions to manage flow analysis in parallel environment, and is applicable for employing multi-core CPUs distributed across many computers, not only multi-core CPUs on a single computer. However, it is difficult, if not impossible, for a non-programmer to write a parallel Rmpi program. Therefore, SPRINT [15] was developed to implement parallel R functions. Although users can use SPRINT easily, it does not specifically support GWA studies.

In this article, we present the development of our ParallABEL library, a new R library for parallelization of GWA studies based on Rmpi. ParallABEL aims to speed up the computation of GWA studies for various statistical analysis requirements and also simplify analysis parallelization. With ParallABEL, the users do not need to be experts programming on partitioning and distributing data, controling and monitoring tasks, and merging output files.

Implementation

GWA Function Grouping

Statistical analyses in GWA studies can be categorized into four groups based on the nature of the statistics computed and type of data used. These four groups can be parallelized in distinct ways. Table 1 shows the name and description of the GenABEL function in each group. The first group contains statistics computed for a particular SNP, or a trait, such as the SNP characterization statistics (e.g. call rates, HWE testing), produced by GenABEL's *summary.snp.data* or association test statistics (the *qtscore, mlreg* and *mmscore* GenABEL functions). The second group holds statistics characterizing an individual in the study, such as, summary statistics of genotype quality for each sample (obtained with the GenABEL *perid.summary* and *hom* GenABEL functions). The third group consists of pair-wise statistics derived from analyses between each pair of individuals in the study, including genome-wide identity-by-state and genomic kinship analyses. This is one of the most computationally intensive analyses, obtained through GenABEL's *ibs* function. The final group concerns pair-wise statistics derived for pairs of SNPs, such as linkage disequilibrium characterisation (the *dprfast, rhofast* and *r2fast* functions). While the number of SNP pairs is generally very large, analyses are usually limited by their pair-wise physical distance, making them less demanding than pair-wise individual analyses, such as IBS computations.

We have developed the ParallABEL library to parallelize the serial functions of these groups using Rmpi library. The four implementation groups are named Type1_parall_by_SNPs for the first group, Type2_parall_by_individuals for the second group, Type3_parall_by_pairs_of_individuals for the third group and Type4_parall_by_pairs_of_SNPs for the fourth group.

Data Partitioning

An advantage of ParallABEL is usage simplicity, hiding otherwise tedious scripts for file management monitoring tools. These functions not only partition input data with automatic load balancing, but also gather output from each processor automatically. Load balancing is critical because an unbalanced work load will result in higher loads for particular processors, which eventually undermines the overall performance.

The input data of Type1_parall_by_SNPs contains SNPs equally partitioned into P subsets (where P is the number of available processors). If the number of SNPs is M, the number of SNPs in a subset is:

$$num\_SNPs = floor\left(M / P\right)$$

If there are M SNPs and 4 processors, the SNPs will be partitioned into 4 smaller subsets. Each containing $M/4$ SNPs as shown in Figure 2. However, the last subset to be generated may contain more SNPs than others, caused by integer division. For example, if there are 801 SNPs and 4 processors, Subset 1 to Subset 3 will contain 200 SNPs, but Subset 4 will have 201. The SNPs in each subset will execute on separate processors.

The input data for Type2_parall_by_individuals consists of individuals, partitioned like Type1_parall_by_SNPs

The input data for Type3_parall_by_pairs_of_individuals is a pair of individuals, and performs a more complicated partitioning than Type1_parall_by_SNPs and Type2_parall_by_individuals. The data is divided until the number of processors is equal to, or less than, the number of subsets for load balancing on each processor. If the number of processors is equal to the number of subsets, then each processor executes an individual pair of each subset. If the number of processors is less than the number of subsets, then each processor executes an equal number of individual pairs (where possible). Figure 3 shows Type3_parall_by_pairs_of_individuals with N individuals. The statistics is calculated from the cross operation of an individual in a row with an individual in a column. The input data is partitioned into 4 subsets using

Table 1: GWA analyses grouping

function name of GenABEL	Description	group
summary.snp.data	Provides summary of observed genotypes, allelic frequency, genotypic distribution, P-value of the exact test for HWE and chromosome	1
qtscore	Fast score test for association between a trait and genetic polymorphism	1
mlreg	Linear and logistic regression and Cox models for genome-wide SNP data	1
mmscore	Score test for association between a trait and genetic polymorphism, in samples of related individuals	1
perid.summary	Produces call rate and heterozygosity per person	2
hom	Computes average homozygosity (inbreeding) for a set of people, across multiple markers. Can be used for Quality Control (e.g. contamination checks)	2
ibs	Given a set of SNPs, computes a matrix of average IBS for a group of people	3
dprfast	Given a set of SNPs, computes a matrix of D'	4
rhofast	Given a set of SNPs, computes a matrix of rho	4
r2fast	Given a set of SNPs, computes a matrix of r2	4

The name and description of function of GenABEL in each group

the data partitioning shown in Figure 3A. However, if the number of processors is more than 4, the subsets are partitioned again. Subset 1 and Subset 4 are split into 8 subsets during the first stage of the data partitioning, while Subset 2 and Subset 3 are divided into 8 subsets by row, as shown in Figure 3B. There are 16 subsets altogether in the second stage of the data partitioning.

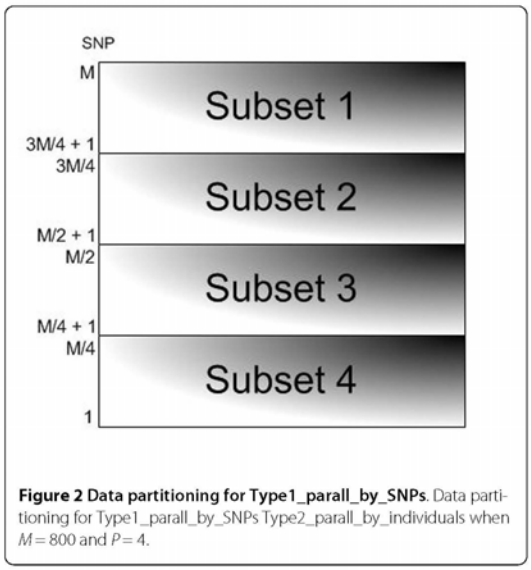


Figure 2 Data partitioning for Type1_parall_by_SNPs. Data partitioning for Type1_parall_by_SNPs Type2_parall_by_individuals when $M = 800$ and $P = 4$.

The SNPs input of Type4_parall_by_pairs_of_SNPs will be partitioned in a similar way to Type3_parall_by _pairs_of_individuals.

Implementation

The workflow for GWA analysis on a single processor or computer is presented in Figure 4A. This workflow runs. The genotype and phenotype data is processed by the GenABEL library, which works under the R program. GenABEL sequentially processes the raw data, producing statistical data as its outputs.

This sequential workflow may take a very long time to produce some demanding statistical analyses. Our novel parallel workflow for producing statistical data in GWA studies is shown in Figure 4B, and can save computing time. The genotype and phenotype data is passed for distribution to the SUN Grid Engine, a job scheduler. It queues jobs and assigns them to processors in a cluster. LAM/MPI (Local Area Multicomputer/Message Passing Interface) [16] has various functions which can be called by Rmpi to parallelize R. ParallABEL parallelizes GenABEL using this Rmpi library. The statistical data from this workflow has been validated by comparing it with the outputs from the non-parallel approach. ParallABEL runs not only on Linux cluster, such as the Rocks Cluster Distribution, but also on any Operating System that supports R and LAM/MPI or Open MPI, such as the Unix and Solaris Operating Systems. It can also run on computer clusters lacking the Sun Grid Engine by executing immediately. However, the administrator will normally not

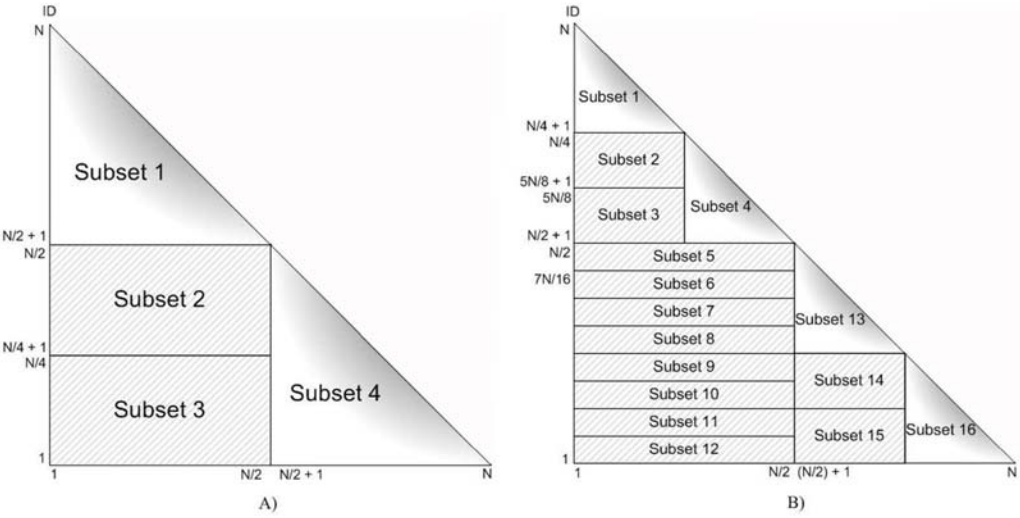


Figure 3 Data partitioning for Type3_parall_by_pairs_of_individuals. A) The first data partitioning for Type3_parall_by_pairs_of_individuals when the number of individuals = *N*. There are 4 equal subsets. B) The second data partitioning for Type3_parall_by_pairs_of_individuals when the number of individuals = *N*. There are 16 equal subsets.

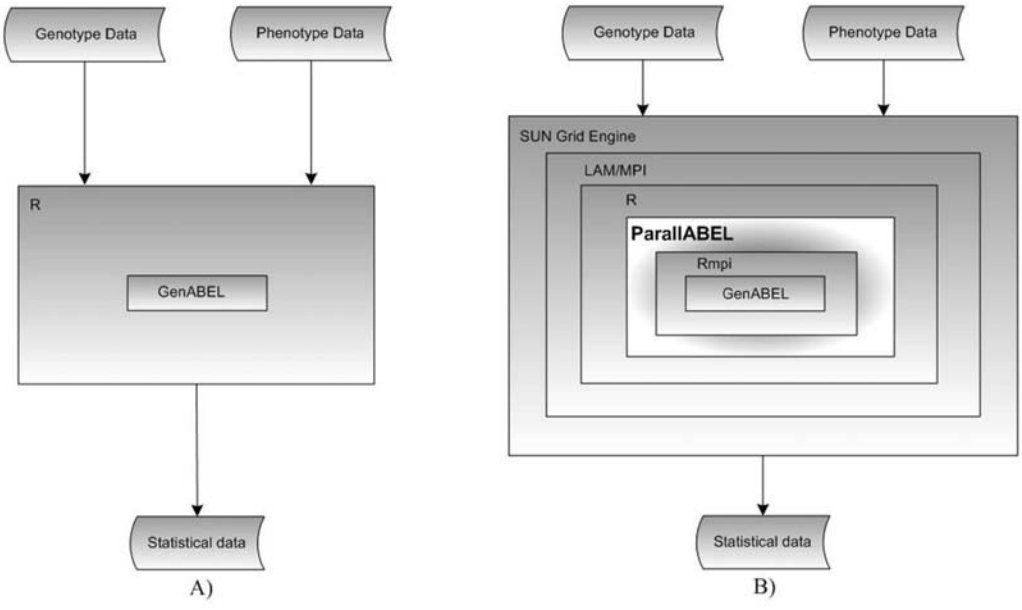


Figure 4 GWA Computing Workflow. A) Sequential GWA Computing Workflow, which runs on a single processor or computer. B) Parallel GWA Computing Workflow which runs on a multiprocessor or a set of computers.

Sangket et al. BMC Bioinformatics 2010, 11:217
http://www.biomedcentral.com/1471-2105/11/217

allow a user to run a parallel program without utilizing a queuing process from the Sun Grid Engine.

To parallelize GWA studies, ParallABEL running on the frontend-node partitions input data into smaller subsets so that tasks can be fairly distributed among the processors. It sends tasks to idle processors on compute-nodes. When the computation on a compute-node has finished, the frontend-node will send another task to the idle processor - a cycle that continues until all the tasks are completed, which is known as the 'task pull' method [17]. When all the tasks are finished, the frontend-node automatically merges all the outputs.

Users can use ParallABEL to parallelize GenABEL GWA functions as easily as using GenABEL for sequential analyses. An example of the *mlreg.p* command sequentially on a processor is shown in Figures 5A and 6A. The executing command that parallelizes *mlreg.p* to run on multiple processors using Type1_parall_by_SNPs is shown in Figures 5B and 6B.

Results

Our computer cluster, Hanuman, runs Rocks Cluster Distribution version 4.3, which includes the SUN Grid Engine version 4.3 [18]. The cluster consists of 5 IBM servers xSeries 336s, comprising of a frontend-node and four compute-nodes. All servers have 2 SINGLE-CORE Intel Xeon (2.8 GHz) processors and 4 GB RAM. The frontend-node and all the compute-nodes are connected through an Ethernet switch, and the user can connect via the Internet. The cluster provides LAM/MPI version 7.1.2, R program version 2.8.1, Rmpi library version 0.5-6, and GenABEL version 1.4-2, which are utilized as components by our ParallABEL library.

The North American Rheumatoid Arthritis Consortium (NARAC) data is part of a dataset employed to observe associations between disease and variants in the major-histocompatibility-complex locus [19]. The NARAC genotype data contains 545,080 SNPs from 2,062 individuals. The data was used to measure the performance of ParallABEL by employing 868 individuals for cases, and 1,194 individuals as controls.

Trace results from Type1_parall_by_SNPs, Type2_parall_by_individuals, Type3_parall_by_pairs_of_individuals, and Type4_parall_by_pairs_of_SNPs for the NARAC data are shown in Figure 7. Type1_parall_by_SNPs was executed by the GenABEL *mlreg* function, Type2_parall_by_individuals was executed by the GenABEL *hom* function, Type3_parall _by_pairs_of_ individuals was executed by the GenABEL *ibs* function, and Type4_parall_by_pairs _of_SNPs was executed by the GenABEL *r2fast* function.

ParallABEL reduced the computing time for Type3_parall_by_pairs_of_individuals, especially with 8 processors. The Type3_parall_by_pairs_of_individuals executing speed on eight processors was approximately seven times faster than on one processor. On a single processor, the complete analysis took 8.1 hours, but only 1.1 hours with 8 processors. The computing time for Type1_parall_by_SNPs also tends to be like that for Type3_parall_by_pairs_of_individuals.

The computing time for the sequential version of Type2_parall_by_individuals can be very short (e.g. 20 seconds). While the parallel version took longer (5.3 minutes for 2 processors), due to the overhead of data partitioning, data distribution, and data merging. Data distribution can be time consuming because the data must be saved on the frontend-node before the compute-nodes can load it, and the frontend-node must also speed time communicating with the compute-nodes. In addition, GenABEL is tailored to quickly retrieve subset of SNPs, as this is a typical GWA scan procedure, but is much less efficient in retrieving subsets of individuals, which is less typical. Thus the overhead of data partitioning in subsets of individuals prevailed over the gain achieved by parallel processing. These results highlighted a place where GenABEL data storage and processing is ineffective, and we are currently working on better algorithms to do by-individual analyses.

Type4_parall_by_pairs_of_SNPs was executed by the GenABEL *r2fast* function. A single processor can not pass all the SNPs in the NARAC data to *r2fast* due to CPU memory limitations so, the analysis was done separately for each chromosome. Even then, a single processor can not call *r2fast* with a chromosome with more than 10,000 SNPs, which affects 20 chromosomes in the data. However, ParallABEL can run *r2fast* with a chromosome with more than 10,000 SNPs by employing a set of processors. The chromosome data is automatically partitioned based on the number of SNPs, as shown in Table 2. If the number of SNPs for a chromosome is between 11 and 14,000, then the data is partitioned into at least 4 balanced subsets. If the number of the SNPs is between 14,001 and 28,000, then the data is divided into at least 16 balanced subsets. If the number of SNPs is between 28,001 and 65,000, then the data is split into at least 64 balanced subsets. The data is automatically partitioned until the number of processors is equal to, or less than, the number of subsets for load balancing on each processor. The trace example results for Type4_parall_by_pairs _of_SNPs of NARAC data are shown in Figure 7.

Type4_parall_by_pairs_of_SNPs took only 1.4 days to execute on eight processors, indicating that time-saving with ParallABEL is linearly correlated to the number of nodes. This suggests that with more SNPs, more computing time will be saved by ParallABEL.

If the number of available processors is P, the parallel computing time for P processors is *time_P_cpus*, and the

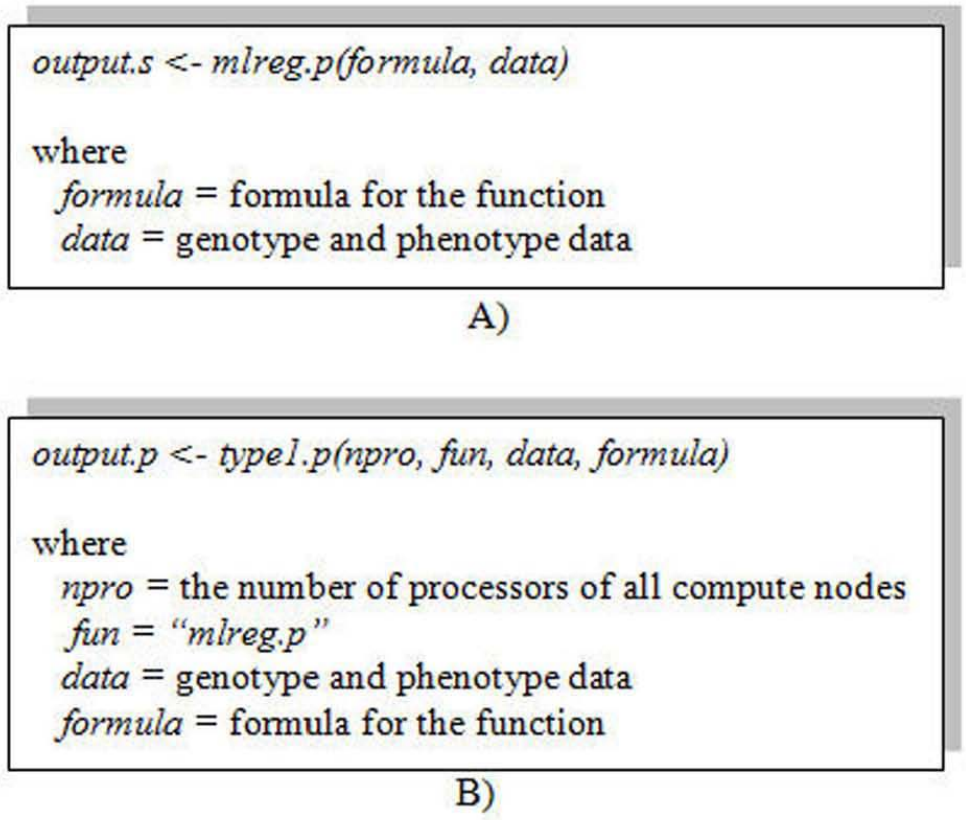


Figure 5 A comparison of using a sequential and parallel function. A) Executing the *mlreg.p* function sequentially on a processor B) Parallelizing the *mlreg.p* function on more than one processor. The user supplies the function name and number of processors to the parallel function.

serial computing time for a processor is *time_a_cpu*; the overhead for *P* processors is:

$$overhead = time\_P\_cpus - time\_a\_cpu / P$$

Different numbers of processors produce different overheads depending on data partitioning, network communicating, and data merging. However, the overheads can be predicted based on the overhead of eight processors shown in Figure 7. The computing time on a large cluster for Type1_parall_by_SNPs, Type3_parall_by_pairs_of_individuals and Type4_parall_by_pairs_of_SNPs extrapolated from Figure 7 applying the above overhead equation are shown in Figure 8. It is clear that ParallABEL also saves the computing time on a large cluster. In addition, the time-saving rates for these types are much increased when the number of processors is between 2 and 50. Nevertheless, the time-saving rates are slowly increased when the number of processors is greater than 100. This applies to the particularly and relatively small data set analyzed here. With bigger data sets, the time-saving rates can be larger. However, the user should optimize the number of processors according to the gain in computational throughput.

Discussion and conclusions

We have presented the ParallABEL library which employs parallel computing to reduce computing time for data intensive tasks. ParallABEL can run on clustered computers that support LAM/MPI and R. With clustered computers, processors or even personal computers can be easily added as new compute-nodes. ParallABEL runs on both distributed and shared memory architectures as it was developed with MPI. For a distributed memory architecture, MPI usually uses a computer network for

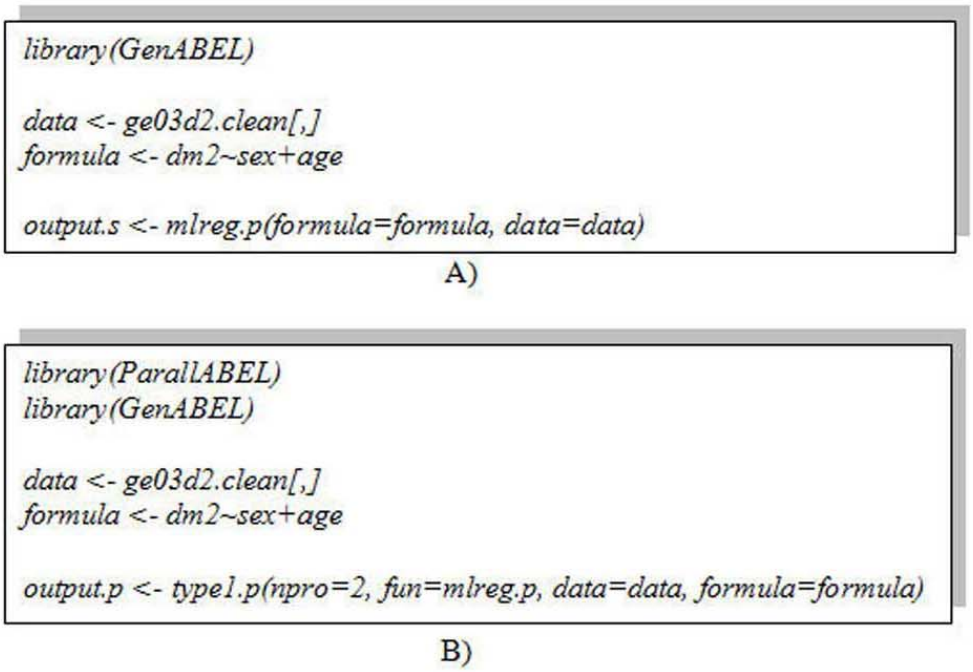


Figure 6 A comparison execute sequential and parallel function. A) Execute the *mlreg.p* function sequentially on a processor B) Parallelize the *mlreg.p* function on more than one processor.

task communications. For a shared memory architecture, MPI does not employ the network for task communications. This means that a distributed memory architecture may exhibit more overhead than a shared memory architecture (for example, eight single-core processors versus a single eight-core processor). In our experiments, Type1_parall_by_SNPs took only 6 minutes to execute on a shared memory architecture but 14 minutes on a distributed memory architecture. The overhead of a shared memory architecture was tested on a server, which has 2 QUAD-CORE Intel Xeon(R) (2.8 GHz) processors and 8 GB. The server runs on CentOS version 5.4, and provides Open MPI version 1.4.1.

ParallABEL allows the user to specify the number of processors employed for data execution. We expect computational performance to increase linearly with the number of processors when using Type1_parall_by_SNPs, Type3_parall_by_pairs_of_individuals, and Type4_parall_by_pairs_of_SNPs. In addition, ParallABEL is faster than GenABEL on one processor. Computing times for Type3_parall_by_pairs_of_individuals and Type4_parall_by_pairs_of_SNPs are longer than those for Type1_parall_by_SNPs because the input data consists of pairs of individuals and SNPs respectively, which are much larger than the SNPs input for Type1_parall_by_SNPs. In addition, if the number of SNPs is n, then the number of inputs for Type1_parall_by_SNPs is n but the number of inputs data for Type4_parall_by_pairs_of_SNPs is n*n. ParallABEL can save much more computational time when utilizing Type3_parall_by_pairs_of_individuals and Type4_parall_by_pairs_of_SNPs than when using Type1_parall_by_SNPs. Therefore, as the amount of input data increases, the time saved by ParallABEL also increases. ParallABEL does not only reduce the computing time but also is as easy-to-use as the more conventional GenABEL.

ParallABEL can not reduce the computing time when the data size is too small, such as the result shown when employing the hom function of Type2_parall_by_ individuals, because the computing time is too short. In that case, the overheads of data partitioning and output merging overwhelm the computational performance.

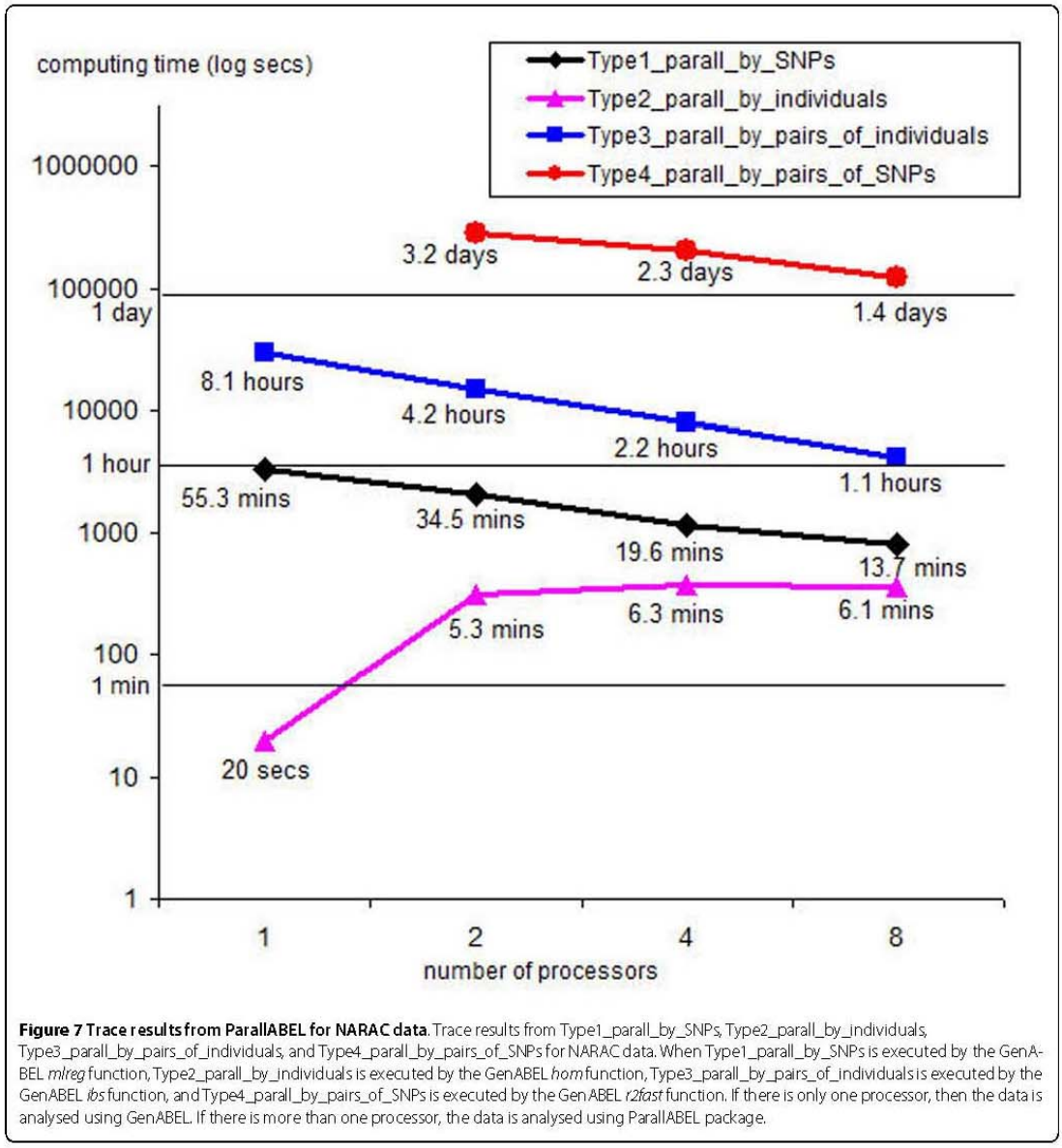


Figure 7 Trace results from ParallABEL for NARAC data. Trace results from Type1_parall_by_SNPs, Type2_parall_by_individuals, Type3_parall_by_pairs_of_individuals, and Type4_parall_by_pairs_of_SNPs for NARAC data. When Type1_parall_by_SNPs is executed by the GenABEL *mlreg* function, Type2_parall_by_individuals is executed by the GenABEL *hom* function, Type3_parall_by_pairs_of_individuals is executed by the GenABEL *ibs* function, and Type4_parall_by_pairs_of_SNPs is executed by the GenABEL *r2fast* function. If there is only one processor, then the data is analysed using GenABEL. If there is more than one processor, the data is analysed using ParallABEL package.

Availability and requirements

- **Project home page:**
  http://www.sci.psu.ac.th/units/genome/CGBR/ParallABEL/index.html
  http://parallabel.r-forge.r-project.org/
- **Operating system(s):** Platform independent
- **Programming language:** R
- **Other requirements:** LAM/MPI or Open MPI, Rmpi, GenABEL
- **License:** GPL for non-profit organizations
- **Any restrictions to use by non-academics:** license needed

Sangket *et al. BMC Bioinformatics* 2010, **11**:217
http://www.biomedcentral.com/1471-2105/11/217

Table 2: Data partitioning for each chromosome

Chromosome name	Number of SNPs	Number of subsets
19,20,21,22, X, Y	11-14,000	4
9,11,12,13,14,15,16,17,18	14,001-28,000	16
1,2,3,4,5,6,7,8,10	28,001-56,000	64

The least number of subsets of each chromosome partitioned by the number of SNPs

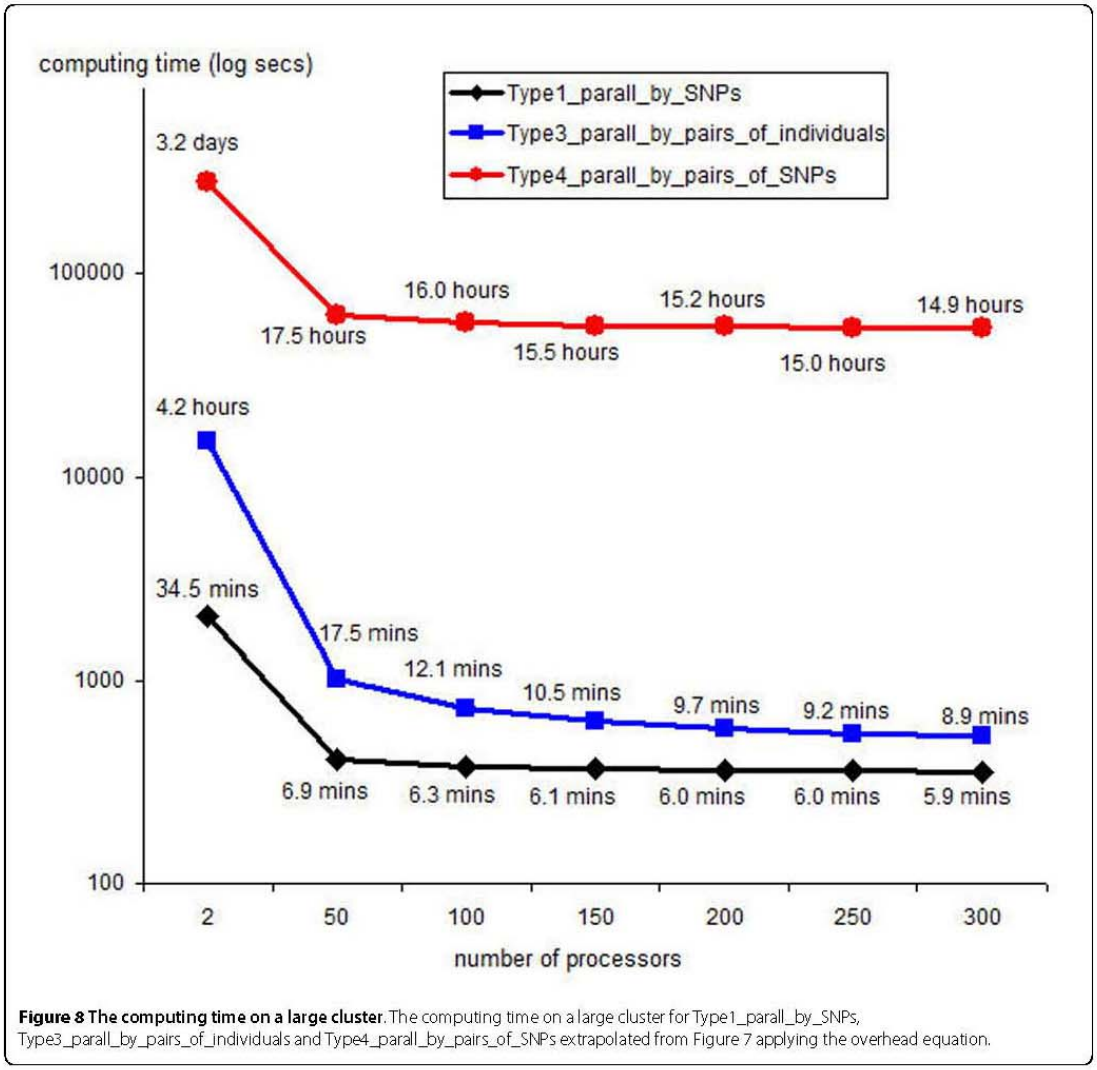


Figure 8 The computing time on a large cluster. The computing time on a large cluster for Type1_parall_by_SNPs, Type3_parall_by_pairs_of_individuals and Type4_parall_by_pairs_of_SNPs extrapolated from Figure 7 applying the overhead equation.

Authors' contributions
All authors conceived and designed the project. PT provided the Hanuman Cluster. US, SM and YSA implemented the software. US conducted computational performance evaluation. All authors drafted, read, revised, and approved the manuscript.

Acknowledgements
This research was supported by a grant from the program for Strategic Scholarships for Frontier Research Network for the Joint Ph.D. Program Thai Doctoral degree from the Office of the Higher Education Commission, Thailand; the Thailand Center of Excellence for Life Sciences (TCELS); and Prince of Songkla University, Thailand. The work of YSA was supported by grants from the Netherlands Scientific Foundation (NWO), the Russian Foundation for Basic Research (RFBR), Netherlands Genomics Initiative (NGI) and Centre for Medical Systems Biology (CMSB). We are grateful to Prof. Dr. Amornrat Phongdara and Assoc. Prof. Dr. Wilaiwan Chotigeat for establishing the PSU research group in Bioinformatics. The NARAC data was supported by the GAW grant (R01 GM031575) and the NIH grant that supports a collection of RA data (AR44422). We would like to thank Dr. Jean W. MacCluer and Vanessa Olmo for the permission to use the data, and Dr. Andrew Davison for polishing the written English of the manuscript. We also thank the Thai National Grid Center and Prince of Songkla University Grid Center for supporting the computer clusters used in this research.

Author Details
[1]Center for Genomics and Bioinformatics Research, Faculty of Science, Prince of Songkla University, Songkhla, 90112, Thailand, [2]Medical Genetic Section, National Institute of Health, Department of Medical Sciences, Ministry of Public Health, Nonthaburi, 11000, Thailand, [3]Department of Pathology, Faculty of Medicine, Ramathibodhi Hospital, Mahidol University, Bangkok, 10400, Thailand, [4]Department of Computer Engineering, Faculty of Engineering, Prince of Songkla University, Songkhla, 90112, Thailand, [5]Department of Epidemiology, Erasmus MC Rotterdam, Postbus 2040, 3000 CA Rotterdam, the Netherlands and [6]Quantitative Integrative Genomics Group, Institute of Cytology & Genetics SD RAS, Novosibirsk 630090, Russia

Received: 21 October 2009 Accepted: 29 April 2010
Published: 29 April 2010

References

1. Pearson TA, Manolio TA: **How to Interpret a Genome-wide Association Study.** *The Journal of the American Medical Association* 2008, **299(11):**1335-1344.
2. Hindorff LA, Sethupathy P, Junkins HA, Ramos EM, Mehta JP, Collins FS, Manolio TA: **Potential etiologic and functional implications of genome-wide association loci for human diseases and traits.** *Proceedings of the National Academy of Sciences of the United States of America* 2009, **106(23):**9362-9367.
3. Aulchenko YS, Ripke S, Isaacs A, van Duijn CM: **GenABEL: an R library for genome-wide association analysis.** *Bioinformatics (Oxford, England)* 2007, **23(10):**1294-1296.
4. **The Comprehensive R Archive Network (CRAN)** [http://www.r-project.org/]
5. Ihaka R, Gentleman R: **R: A language for data analysis and graphics.** *Journal of Computational and Graphical Statistics* 1996, **5(3):**299-314.
6. **Introduction to Parallel Computing** [https://computing.llnl.gov/tutorials/parallel_comp/]
7. Mishima H, Lidral AC, Ni J: **Application of the Linux cluster for exhaustive window haplotype analysis using the FBAT and Unphased programs.** *BMC bioinformatics* 2008, **9(Suppl 6):**S10.
8. Laird NM, Horvath S, Xu X: **Implementing a unified approach to family-based tests of association.** *Genetic epidemiology* 2000, **19(Suppl 1**S36-42 [http://biosun1.harvard.edu/~fbat/fbat.htm].
9. Dudbridge F: **Pedigree disequilibrium tests for multilocus haplotypes.** *Genetic epidemiology* 2003, **25(2**115-121 [http://portal.litbio.org/Registered/Help/unphased/].
10. Vera G, Jansen RC, Suppi RL: **R/parallel--speeding up bioinformatics analysis with R.** *BMC bioinformatics* 2008, **9**:390.
11. Misawa K, Kamatani N: **ParaHaplo: A program package for haplotype-based whole-genome association study using parallel computing.** *Source code for biology and medicine* 2009, **4**:7.
12. Ma L, Runesha HB, Dvorkin D, Garbe JR, Da Y: **Parallel and serial computing tools for testing single-locus and epistatic SNP effects of quantitative traits in genome-wide association studies.** *BMC bioinformatics* 2008, **9**:315.
13. **Rmpi: Interface (Wrapper) to MPI (Message-Passing Interface)** [http://www.stats.uwo.ca/faculty/yu/Rmpi/]
14. **Message-Passing Interface Forum (MPI)** [http://www.mpi-forum.org/]
15. Hill J, Hambley M, Forster T, Mewissen M, Sloan TM, Scharinger F, Trew A, Ghazal P: **SPRINT: a new parallel framework for R.** *BMC bioinformatics* 2008, **9**:558.
16. **Local Area Multicomputer/Message Passing Interface** [http://www.lam-mpi.org/]
17. **Rmpi Program Structure** [http://math.acadiau.ca/ACMMaC/Rmpi/structure.html]
18. **Rocks Cluster Distribution** [http://www.rocksclusters.org/wordpress/]
19. Plenge RM, Seielstad M, Padyukov L, Lee AT, Remmers EF, Ding B, Liew A, Khalili H, Chandrasekaran A, Davies LR, *et al.*: **TRAF1-C5 as a risk locus for rheumatoid arthritis--a genomewide study.** *The New England journal of medicine* 2007, **357(12):**1199-1209.

doi: 10.1186/1471-2105-11-217
Cite this article as: Sangket *et al.*, ParallABEL: an R library for generalized parallelization of genome-wide association studies *BMC Bioinformatics* 2010, **11**:217

Appendix B

ParallABEL manual

Package 'ParallABEL'

February 22, 2010

Title Parallel of Genome-Wide Association function

Version 0.1-0

Author Unitsa Sangket

Description Support for parallel GenABEL package in R.

Maintainer Unitsa Sangket <usangket@yahoo.com>

License GPL

Depends R (>= 2.8), utils

R topics documented:

`type1.p` *Parallel for the analyses of statistics of each SNP*

Description

Parallel for the analyses of statistics of each SNP, such as SNP characterization statistics or association test statistics

Usage

```
type1.p(npro,fun,data,data_f,...)
```

Arguments

`npro`	number of processors on compute nodes
`fun`	function name will be process such as mlreg.p
`data`	object of snp.data-class
`data_f`	file name that was saved the input object of snp.data-class, the object must be named "data"
`...`	further arguments passed to function of fun argument

Author(s)

Unitsa Sangket, Yurii S. Aulchenko and Surakameth Mahasirimongkol

Examples

```
Example 1 (summit job on R)
################################################
#clear working space
rm(list = ls())
library(GenABEL)
library(ParallABEL)
formula=dm2~sex+age
data(ge03d2.clean)
data <- ge03d2.clean[,]
npro=2 # npro = number of processors on the compute-nodes
fun=mlreg.p

output.p = type1.p(npro,fun,data,formula=formula)

output.p[1:5,]

mpi.quit(save="no")

################################################

Example 2 (summit job on Sun Grid Engine)
You have to create 2 files if you want to summit a job on Sun Grid Engine.
(http://math.acadiau.ca/ACMMaC/Rmpi/submitting.html)

File 1 (R_script.sh):
#############################################################################
#!/bin/bash

# Run in the current directory
#$ -cwd

#$ -j y

#$ -V

# Run using bash
#$ -S /bin/bash

# The number of processors required - 1 for frontend-node, plus whatever
# number for compute-nodes.
# This example runs with a processor on frontend-node, plus 2 processors on compute-nodes
#$ -pe lam 3

# Run the job.   Replace with whatever R script should run
lamrun -np 1 R  --slave CMD BATCH R_script.R R_script_sh.Rout
#############################################################################

File 2 (R_script.R):
#############################################################################
#clear working space
```

type2.p 3

```
rm(list = ls())
library(GenABEL)
library(ParallABEL)
formula=dm2~sex+age
data(ge03d2.clean)
data <- ge03d2.clean[,]
npro=2 # npro = number of processors on the compute-nodes
fun=mlreg.p

output.p = type1.p(npro,fun,data,formula=formula)

output_Rdata ="output_type1_mlreg_c3.Rdata"
save(output.p,file=output_Rdata)
# You can load the output from output_type1_mlreg_c3.Rdata

output.p[1:5,]

mpi.quit(save="no")

##############################################################################

For summit the job please use command:
qsub R_script.sh

You can see the progress on R_script_sh.Rout.
```

`type2.p` *Parallel for the analyses of statistics of each individual*

Description

Parallel for the analyses of statistics of each individual, for example, summary statistics of genotype quality for each sample

Usage

```
type2.p(npro,fun,data,data_f,...)
```

Arguments

`npro`	number of processors on compute nodes
`fun`	function name will be processed such as hom
`data`	object of snp.data-class
`data_f`	file name that was saved the input object of snp.data-class, the object must be named "data"
`...`	further arguments passed to function of fun argument

Author(s)

Unitsa Sangket, Yurii S. Aulchenko and Surakameth Mahasirimongkol

Examples

```
Example 1 (summit job on R)
################################################
#clear working space
#clear working space
rm(list = ls())
library(GenABEL)
library(ParallABEL)
data(ge03d2.clean)
data <- ge03d2.clean[,]

npro=2 # npro = number of processors
fun=hom

output.p = type2.p(npro,fun,data)

output.p[1:5,]

mpi.quit(save="no")

################################################

Example 2 (summit job on Sun Grid Engine)
You have to create 2 files if you want to summit a job on Sun Grid Engine.
(http://math.acadiau.ca/ACMMaC/Rmpi/submitting.html)

File 1 (R_script.sh):
#############################################################################
#!/bin/bash

# Run in the current directory
#$ -cwd

#$ -j y

#$ -V

# Run using bash
#$ -S /bin/bash

# The number of processors required - 1 for frontend-node, plus whatever
# number for compute-nodes.
# This example runs with a processor on frontend-node, plus 2 processors on compute-nodes
#$ -pe lam 3

# Run the job.   Replace with whatever R script should run
lamrun -np 1 R  --slave CMD BATCH R_script.R R_script_sh.Rout
#############################################################################

File 2 (R_script.R):
#############################################################################
#clear working space
rm(list = ls())
library(GenABEL)
library(ParallABEL)
```

type3.p 5

```
data(ge03d2.clean)
data <- ge03d2.clean[,]

npro=2 # npro = number of processors
fun=hom

output.p = type2.p(npro,fun,data)

output_Rdata = "output_type2_hom_c3.Rdata"
save(output.p,file=output_Rdata)
# You can load the output from output_type2_hom_c3.Rdata

output.p[1:5,]

mpi.quit(save="no")

##############################################################################

For summit the job please use command:
qsub R_script.sh

You can see the progress on R_script_sh.Rout.
```

`type3.p` *Parallel for the pairwise statistics derived from analyses between each individual*

Description

Parallel for pairwise statistics derived from analyses between each individual, for example genome-wide identity-by-state or genomic kinship analyses

Usage

```
type3.p(npro,fun,data,data_f,...)
```

Arguments

`npro`	number of processors on the compute-nodes
`fun`	function name will be processed such as ibs
`data`	object of snp.data-class
`data_f`	file name that was saved the input object of snp.data-class, the object must be named "data"
`...`	further arguments passed to function of fun argument

Author(s)

Unitsa Sangket, Yurii S. Aulchenko and Surakameth Mahasirimongkol

Examples

```
Example 1 (summit job on R)
################################################
#clear working space
rm(list = ls())

library(GenABEL)
library(ParallABEL)
data(srdta)
data <- srdta[,]
npro=2 # npro = number of processors on the compute-nodes
fun=ibs

output.p = type3.p(npro,fun,data)

output.p[1:5,1:5]

mpi.quit(save="no")

################################################

Example 2 (summit job on Sun Grid Engine)
You have to create 2 files if you want to summit a job on Sun Grid Engine.
(http://math.acadiau.ca/ACMMaC/Rmpi/submitting.html)

File 1 (R_script.sh):
#############################################################################
#!/bin/bash

# Run in the current directory
#$ -cwd

#$ -j y

#$ -V

# Run using bash
#$ -S /bin/bash

# The number of processors required - 1 for frontend-node, plus whatever
# number for compute-nodes.
# This example runs with a processor on frontend-node, plus 2 processors on compute-nodes
#$ -pe lam 3

# Run the job.   Replace with whatever R script should run
lamrun -np 1 R --slave CMD BATCH R_script.R R_script_sh.Rout
#############################################################################

File 2 (R_script.R):
#############################################################################
#clear working space
rm(list = ls())

library(GenABEL)
library(ParallABEL)
```

type4.p 7

```
data(srdta)
data <- srdta[,]
npro=2 # npro = number of processors on the compute-nodes
fun=ibs

output.p = type3.p(npro,fun,data)

output_Rdata ="output_type3_ibs_c3.Rdata"
save(output.p,file=output_Rdata)
# You can load the output from output_type3_ibs_c3.Rdata

output.p[1:5,1:5]

mpi.quit(save="no")

################################################################################

For summit the job please use command:
qsub R_script.sh

You can see the progress on R_script_sh.Rout.
```

`type4.p` *Parallel for the pairwise statistics derived from each pair of SNP*

Description

Parallel for pairwise statistics derived from each pair of SNP, e.g. linkage disequilibrium characterisation

Usage

```
type4.p(npro,fun,data,data_f,output_f,...)
```

Arguments

`npro`	number of processors on compute nodes
`fun`	function name will be processed such as r2fast
`data`	object of snp.data-class
`data_f`	file name that was saved the input object of snp.data-class, the object must be named "data"
`output_f`	a file will be saved the outputs
`...`	further arguments passed to function of fun argument

Author(s)

Unitsa Sangket, Yurii S. Aulchenko and Surakameth Mahasirimongkol

Examples

```
Example 1 (summit job on R)
################################################
#clear working space
rm(list = ls())

library(GenABEL)
library(ParallABEL)
data(srdta)
data <- srdta[,]
npro=2 # npro = number of processors
output_f="output_type4_r2fast_c3"
fun=r2fast

output.p = type4.p(npro,fun,data,output_f=output_f)

# You can load the output from output_type4_r2fast_c3.Rdata
mpi.quit(save="no")

################################################

Example 2 (summit job on Sun Grid Engine)
You have to create 2 files if you want to summit a job on Sun Grid Engine.
(http://math.acadiau.ca/ACMMaC/Rmpi/submitting.html)

File 1 (R_script.sh):
#############################################################################
#!/bin/bash

# Run in the current directory
#$ -cwd

#$ -j y

#$ -V

# Run using bash
#$ -S /bin/bash

# The number of processors required - 1 for frontend-node, plus whatever
# number for compute-nodes.
# This example runs with a processor on frontend-node, plus 2 processors on compute-nodes
#$ -pe lam 3

# Run the job.   Replace with whatever R script should run
lamrun -np 1 R  --slave CMD BATCH R_script.R R_script_sh.Rout
#############################################################################

File 2: R_script.R:
#############################################################################
#clear working space
rm(list = ls())

library(GenABEL)
library(ParallABEL)
```

type4.p 9

```
data(srdta)
data <- srdta[,]
npro=2 # npro = number of processors
output_f="output_type4_r2fast_c3"
fun=r2fast

output.p = type4.p(npro,fun,data,output_f=output_f)

# You can load the output from output_type4_r2fast_c3.Rdata
mpi.quit(save="no")

################################################################################

For summit the job please use command:
qsub R_script.sh

You can see the progress on R_script_sh.Rout.
```

Index

Appendix C

ParallLogicReg manual

Package 'ParallLogicReg'

May 7, 2011

Title Logic regression using parallel computing

Version 1.0-0

Author Unitsa Sangket

Description Support for parallel LogicReg package on R.

Maintainer Unitsa Sangket <usangket@yahoo.com>

License GPL

Depends R (>= 2.8), utils

R topics documented:

`ParallLogicReg` *Logic regression analyses using parallel computing*

Description

This function can be used in many situations, especially identification of SNP interactions

Usage

```
output = ParallLogicReg(infile,resp,begin,end,nperm,niter)
```

Arguments

`infile`	A file name that contains data, which will be analyzed.
`resp`	vector with the response variables. See more detail in LogicReg manual.
`begin`	The first gene id will be run.
`end`	The last gene id will be run.
`nperm`	number of permutations
`niter`	number of iterations

Details

An example of infile:

snp	chr	pos	gene-symbol	gene-id	V1	V2	V3	V4	V5
rs1	1	12691937	AADACL3	1	1	1	1	1	1
rs1	1	12691937	AADACL3	1	1	1	1	1	1
rs31	1	12717433	AADACL3	1	1	0	1	1	1
rs31	1	12717433	AADACL3	1	1	1	1	1	1
rs2	1	12668821	AADACL4	2	0	1	1	1	1
rs2	1	12668821	AADACL4	2	1	1	1	1	1
rs58	1	12633603	AADACL4	2	0	0	0	0	0
rs58	1	12633603	AADACL4	2	0	0	1	0	0
rs59	1	12651828	AADACL4	2	1	1	1	1	1

where,

snp = snp id
chr = chromosome number
pos = position of the snp on chromosome
gene-symbol = gene symbol or gene name
gene-id = gene id
V1,V2,V3,V4,V5,... = snp of control or case

Note: One snp has two data rows, and each field must be separated by tab.

Author(s)

Unitsa Sangket and Qi Liu

Examples

```
#------------------------------------------------------------------------
#An example run on Sun Grid Engine.
#You have to create two files, which are a ".sh" file, and a ".R" file.
#for more information please visit http://math.acadiau.ca/ACMMaC/Rmpi/submitting.html
#------------------------------------------------------------------------

#File 1 (R_script.sh):
##########################################################################
#!/bin/bash

# Run in the current directory
#$ -cwd

#$ -j y

#$ -V

# Run using bash
#$ -S /bin/bash

# Set the number of processors
# For example, 2 means one processor is master and slave1,
# and the rest of processors are slaves.
#$ -pe lam 2

# Run the job.
# lamrun -np 1 R  --slave CMD BATCH R_script.R R_script_sh.Rout
```

```
##############################################################################
#!!! remove "#" out from "lamrun" command when run the file

#File 2 (R_script.R):
##############################################################################
library(ParallLogicReg)

resp=c(rep(0,2935),rep(1,1745)) # number of controls = 2,935; number of cases = 1,745

nperm=20                        # number of permutations
niter=20                               # number of iterations
begin=1                         # the first gene id that will be run
end=10                          # the last gene id that will be run
in=("input.txt")        # data will be run

output = ParallLogicReg(infile=in,resp=resp,begin=begin,end=end,nperm=nperm,niter=niter)

mpi.quit(save="no")

##############################################################################

#-----------------------------------------------------------------------------
#You can summit the job using this command:
#qsub R_script.sh

#You can see the progress on R_script_sh.Rout and progress files.
#-----------------------------------------------------------------------------
```

Index

Appendix D

Type1_parall_by_SNPs source code

```
#####################################################################
#Function:      parallel type1 function
#Programer:     Unitsa Sangket
#Date:          2010
#Objective:     to parallel type1 functions of GenABEL
#Note:          an example of type1.p function is mlreg.p
#####################################################################

"type1.p" <- function(npro,fun,data,data_f="no",...){
#Initialize MPI
library("Rmpi")

# Notice we just say "give us all the slaves you've got."
mpi.spawn.Rslaves()

if (mpi.comm.size() < 2) {
   print("More slave processes are required.")
   mpi.quit()
   }

.Last <- function(){
   if (is.loaded("mpi_initialize")){
      if (mpi.comm.size(1) > 0){
         print("Please use mpi.close.Rslaves() to close slaves.")
         mpi.close.Rslaves()
      }
      print("Please use mpi.quit() to quit R")
      .Call("mpi_finalize")
   }
}
```

```
##########  the slaves will call to perform a validation on the
# fold equal to their slave number.
# Assumes: fold,foldNumber
foldslave <- function(){
   # Note the use of the tag for sent messages:
   #     1=ready_for_task, 2=done_task, 3=exiting
   # Note the use of the tag for received messages:
   #     1=task, 2=done_tasks
   junk <- 0
   done <- 0

   while (done != 1) {
      # Signal being ready to receive a new task
      mpi.send.Robj(junk,0,1)

      # Receive a task
      task <- mpi.recv.Robj(mpi.any.source(),mpi.any.tag())
      task_info <- mpi.get.sourcetag()
      tag <- task_info[2]

      if (tag == 1) {

                        #****** 3. task computation in compute nodes *****
                        # load GenABEL library
                        library(GenABEL)

                        snpsubset  = task$snpsubset
                        foldNumber = task$foldNumber
                        source(temp_fun_type1_f)
```

```
                    load(data_f)

                    ## edit fro test eigth_core
                    start = task$start
                    stop = task$stop
                    data <- data[,start:stop]
                    ###

                    args_oth$data = data
                    args_oth$snpsubset = snpsubset
                    args_oth$fun = fun
                    formals(temp_fun) = args_oth
                    output=temp_fun()

                    results <- list(foldNumber=foldNumber,output=output)
                    rm(data)
                    #********* end of task computation *********

        mpi.send.Robj(results,0,2)
        }
    else if (tag == 2) {
        done <- 1
        }
    # We'll just ignore any unknown messages
  }

  mpi.send.Robj(junk,0,3)
}
```

```
########## We're in the parent.

#************ 1. task separatation   *************
# load GenABEL library
library(GenABEL)

if (missing(npro))
   stop("Missing number of processors")
if (missing(fun))
   stop("Missing function name")

if (missing(data) && missing(data_f))
   stop("Missing data")

# generate subscript file
t_subscript <- 1:99999999
subscript = sample(t_subscript,1)

if(data_f == "no"){ # there are no data file
        data_f = paste("data_",subscript,".Rdata",sep = "")
        save(data,file=data_f)
        data_f_n = 1
}
else{
        data_f_n = 0
        load(data_f)
}
##### check number of snps
number_of_snps = data@gtdata@nsnps
```

```
if (number_of_snps < 11)
   stop("The data is too small.")

##### check arguments
snpsubset <- data@gtdata@snpnames[1:10]
a = fun(data=data,snpsubset=snpsubset,...)

##### separate data
nsnps = length(data@gtdata@snpnames)
nsnps_p = floor(nsnps/npro)
pointer = 0

#create data@gtdata@snpnames = data@gtdata@snpnames[start:stop]
#Create task list
tasks <- vector('list')
for (i in 1:(npro-1)) {
   tasks[[i]] <- list(foldNumber=i,snpsubset=data@gtdata@snpnames[(pointer
+ 1):(pointer + nsnps_p)], start = (pointer + 1), stop = (pointer + nsnps_p))
   pointer = pointer + nsnps_p
}
#last process
i = i + 1
tasks[[i]] <- list(foldNumber=i, snpsubset=data@gtdata@snpnames[(pointer +
1):(nsnps)], start = (pointer + 1), stop = nsnps)

# initial results
results <- vector('list')
for (i in 1:npro) {
   results[[i]] <- list(output=i)
}
#********************** end of task separation  ********************
```

```
# Now, send the data to the slaves

# Send the function to the slaves
mpi.bcast.Robj2slave(foldslave)

#******************** 2. task distribution   ********************
# Call the function in all the slaves to get them ready to
# undertake tasks

### prepairing args

"temp" <- function(data,...){
        # argument must add later
        rm(data)
        rm(snpsubset)

        # check this argument before remove
        if (missing(idsubset))
                rm(idsubset)

        args=ls()

        old_formals = formals(temp)
        n_old_formals = names(old_formals)

        match_args = match(args, n_old_formals)
        temp = old_formals
```

```
        #update arguments of new_formals
        for(i in 1:length(args)){
                if( !is.na(match_args[i])){
                        if (temp[[match_args[i]]] == get(args[i])) # default arg
value
                                temp[[match_args[i]]] = ""
                        else temp[[match_args[i]]] = get(args[i])

                }
        }

        # delete empty value arguments
        i = 1
        n_temp = names(temp)
        while ( i <= length(n_temp)){
                if (temp[i] == "")
                        temp[i] <- NULL
                else
                        i = i + 1
        }

        return(temp)  # return all arguments except data, snpsubset and the
argument which have default value
}

formals(temp) = formals(fun)
args_oth = temp(data=data,...)

### peparing call_fun
n_args_oth = names(args_oth)
```

```
call_fun = paste("temp_fun <- function(fun){", "\n", sep="")

call_fun = paste(call_fun, "output <- fun(data=data, snpsubset=snpsubset",sep
= "")

if (length(n_args_oth) > 0 ){
        for(i in 1:length(n_args_oth)){
                call_fun = paste(call_fun,", ",n_args_oth[i], "=args_oth$",
n_args_oth[i],sep="")
        }
}

# insert ) and }
call_fun = paste(call_fun,")","\n", "return(output)", "\n", "}",sep="")

temp_fun_type1_f = paste("temp_fun_type1_",subscript,".R",sep = "")

write(call_fun, temp_fun_type1_f)

### send argument
mpi.bcast.Robj2slave(temp_fun_type1_f)
mpi.bcast.Robj2slave(data_f)
mpi.bcast.Robj2slave(args_oth)
mpi.bcast.Robj2slave(fun)
mpi.bcast.cmd(foldslave())

rm(data)

#**********end of task distribution **********
```

```
junk <- 0
closed_slaves <- 0
n_slaves <- mpi.comm.size()-1

while (closed_slaves < n_slaves) {
   # Receive a message from a slave
   message <- mpi.recv.Robj(mpi.any.source(),mpi.any.tag())
   message_info <- mpi.get.sourcetag()
   slave_id <- message_info[1]
   tag <- message_info[2]

   if (tag == 1) {
      # slave is ready for a task. Give it the next task, or tell it tasks
      # are done if there are none.
      if (length(tasks) > 0) {
         # Send a task, and then remove it from the task list
         mpi.send.Robj(tasks[[1]], slave_id, 1);
         tasks[[1]] <- NULL
         }
      else{
         mpi.send.Robj(junk, slave_id, 2)
         }
   }
   else if (tag == 2) {

                #********** 4. result storing **********
                # The message contains results. Do something with the results.
                # Store them in the data structure
                results[[message$foldNumber]] = message$output
                #********** end of result storing **********
```

```
   }
   else if (tag == 3) {
      # A slave has closed down.
      closed_slaves <- closed_slaves + 1
   }

}

mpi.close.Rslaves()

#****************** 5. result combining ***************
# combine order by snpnames because may be slave2 finish before slave1

#return(results)

#stop("pause")

results_list = results
results = results_list[[1]]

##check structure of result
# if data.frame use rbind
if (is.data.frame(results)) {
        for (i in 2:npro) {
                results = rbind(results,results_list[[i]])
        }

}else if (is.list(results)){
        # create flag_do array
        # flag = 1 -> will combine, flag = 0 -> not combine
```

```
	n_results = names(results_list[[1]])
	flag_do = n_results
	for(i in 1:length(flag_do)){
		if ( (length(results_list[[1]][[n_results[i]]]) ==
length(results_list[[1]][["snpnames"]]) ) && (n_results[i] != "idnames") )
			flag_do[i] = 1
		else flag_do[i] = 0
	}

	# combine results
	results = results_list[[1]]
	for (i in 2:npro) {
		for (j in 1:length(n_results)){
			if (flag_do[j] == 1)
				results[[n_results[j]]] = c(results[[n_results[j]]],
results_list[[i]][[n_results[j]]])
		}
	}

}else {
	message_error = paste("Error: structure of result from ", fun, " doesn't a
list or a data.frame", sep = "")
	stop(message_error)
}

# remove data_f, temp_fun_type1_f

if (data_f_n==1){ # if data is loaded from a file, then
file.remove(data_f)
}
```

```
file.remove(temp_fun_type1_f)

#************ end of result combining ***********

return(results)

} # End of function
```

VITAE

Name Miss Unitsa Sangket

Student ID 5110230013

Education Attainment

Degree	Name of Institute	Year of Graduation
Bachelor of Science (Computer Science)	Prince of Songkla University	2002
Master of Science (Computer Science)	Prince of Songkla University	2006

Scholarship Awards during Enrolment

1. The program for Strategic Scholarships for Frontier Research Network for the Joint Ph.D. Program Thai Doctoral degree from the Office of the Higher Education Commission, Thailand.

2. The lecturer scholarship from Prince of Songkla University, Thailand.

Work – Position and Address

I am a lecturer at Center for Genomics and Bioinformatics, Faculty of Science, Prince of Songkla University, Kanchanawanish Rd., Hat-Yai, Songkhla, Thailand, 90112.

List of Publications and Proceedings

Publication

U. **Sangket**, S. Mahasirimongkol, W. Chantratita, P. Tandayya and Y.S. Aulchenko, "ParallABEL: an R library for generalized parallelization of genome-wide association studies," *BMC Bioinformatics,* vol. 11, p. 217, 2010.

Presentation

Poster presentation

U. **Sangket**, S. Mahasirimongkol, W. Chantratita, P. Tandayya and Y.S. Aulchenko, " ParallABEL: an R library for speedup of GWAS applying parallel computing," The International Conference on Bioinformatics/International Society for Computational Biology-Asia (InCoB/ISCB-Asia) Joint Conference 2011, Kuala Lumpur, Malaysia, November 30th, 2011.