Parallelisation of IBD computation for determining genetic disease map

Nouhad J. Rizk
Notre Dame University
Computer Science Department
P.o.Box 72, Zouk-Mosbeh, Lebanon
nrizk@ndu.edu.lb

Abstract

A number of software packages are available for the construction of comprehensive human genetic maps. In this paper we parallelize the widely used package Genehunter. We restrict our attention to only one function of the package, namely the computations of Identity By Descent (IBD) genes of a family. We use a master-slave model with the Message Passing Interface (MPI) parallel environment. Our tests are done on two different architectures: a network of workstations and a shared memory multiprocessor. A new and efficient strategy to classify the parallelization of genetic linkage analysis programs results from our experiments. The classification is based on values of parameters which affect the complexity of the computation.

1 INTRODUCTION

Many genetics projects focus on the development of a comprehensive human genetic map, which is used in the identification of genes associated with genetic diseases. Amongst the software packages resulting these projects, Genehunter is one of the most popular. The package provides a wide range of methods for performing linkage and disequilibrium analysis. It has been implemented by the Whitehead Institute for Biomedical Research and can be downloaded from the site [2]:

`www-genome.wi.mit.edu/ftp/distribution/software/genehunter`

The program is based on a study done by Lander and Green [5] that uses the Hidden Markov Models in calculating inheritance distribution. The package can rapidly analyze moderately sized families but fails to analyze an arbitrary sized instances, as calculations require large amounts of memory and central processing unit (CPU) time.

Genehunter is the main tool for the researchers who are particularly interested in using the genetic data of all members of a family to identify the identical locus[1] transmitted by a common ancestor. This problem is known as *Identity by Descent (IBD)* problem. To specify the genetic resemblance between two relatives[2] affected by the same genetic disease, researchers localize regions where this resemblance is frequent and then identify the disease genes in founded regions. Polymorph loci used to quantify the genetic resemblance are called *markers* and a family is called a *pedigree*. For a given pedigree, performing multipoint linkage analysis with many markers, and taking account of all pedigree information, is a fundamental task for studying a complex inherited diseases. Thus, the computation time scales linearly with the `number of markers` studied, but exponentially with the number of non-founders. An individual is called non-founder if his parents are members of the family—in other words, if his genes values on each marker(genotypes) are known.

The time required for Hidden Markov Model reconstruction using this algorithm is $O(m.2^{4.n})$ where m is the number of markers and n is the number of non-founders. Of course, this scaling behaviour limits the size of pedigrees that may be studied. The different approaches used to reduce this complexity have been implemented in special cases. For example, Krugklak et al. [4] focus on the possibility of using a different pedigree size and reduce the complexity to $O(m.n.2^{2.n})$.

Thus, the complexity is reduced to $O(m.n.2^{2.n-f})$. Time complexity is not the only consideration: the space complexity for even moderate pedigree sizes, say $2.n-f \sim 16$ will consume all the memory on a typical workstation. Finally, an additional improvement was done to give an algorithm which scales as $O(2^{2.n-f}log^2(2^{2.n-f}))$ [3].

Moreover, the computation time depends on three other parameters: the `number of families`, the `informativity of the markers` and the `familial`

[1] locus: small DNA sequence presenting several variations in a population.

[2] relatives: two individuals with a common ancestor(e.g. two brothers or an uncle and a nephew).

`structure`. Marker informativity denotes the average amount of variant diversity in a population. If parents present identical variants, genetic resemblance in sibs is less accurate. The familial structure contains the number of typed parents[3] (also known as founders), the sibship size (the number of children) and the number of generations. When no genetic information is available for one or both parents, this information has to be inferred from children. In this case, missing information is better inferred from large sibship. When parent missing information is poorly reconstructed, we have to compute and summarize probabilities from many possible alternatives.

Geneticists are always facing the limitations of the sequential Genehunter version which are memory allocation and CPU time consumption. The goal of this paper is to provide a new strategy for solving linkage packages analysis based on a parallel program capable of managing these two main handicaps. Conant et al. [1] parallelize the Genehunter by distributing markers among different processors—we call this algorithm the low-level parallel model. However, our high-level model is based on a master-slave paradigm. A more general strategy would be the possibility of selecting either one of these models, or a combination of both. In the high-level model, the granularity of a task is one family. Thus, computing the IBD of a family can be considered as independent task and it is possible to run all the tasks on the available processors using master-slave approach. In addition, the use of Message Passing Interface provides source-code portability and allows efficient implementation across a range of architecture.

We begin with a brief introduction to the problem's mathematical basis in Section 2. Section 3 describes the parallel implementation. Some experiments and computational results on parallel machines are presented in in Section 4. In Section 5, we propose a general parallelisation strategy for the linkage genetic analysis problem and finally we conclude in Section 6.

2 Problem description

The solution of the IBD problem is based on the Hidden Markov Model (HMM) and is presented in detail in [6, 8]. The purpose of this paper is not to alter the mathematical details of the problem but to parallelize the Genehunter package as it is. The interested reader may refer to the above two papers for more information. To facilitate the comparison between (HMM) and Genehunter characteristics and parameters we give here only few definitions: the known states of the genes represent the genotypes, the observed states of the genes represent the phenotypes, the different time status represents the marker locus, the forward variable represents the probabilities of ancestors, the backward variable represents the probabilities of descendants and siblings, and the different states are represented by the inheritance vectors. In fact, the reconstruction of the hidden states is the reconstruction of the genetic mapping. All the computation is based on the probability of the unobserved states that must be recalculated [8, 7].

We focus, in this paper, on the calculation of the IBD probability. The complexity of this computation is based on the length of the states of genes in a family. Suppose that n is the number of non-founders in a family. The conditional inheritance distribution at any point along a chromosome is obtained after the computation of the forward matrix and the backward matrix of HMM with states of length equal to $2 * n$ in a family, and with number of possible states equal to 2^{2*n}. If m is the number of markers, the algorithm calculates the complete probability over all markers forwards and backwards. From here, the IBD probability at any point is equal to the summation of the conditional inheritance distribution over the state corresponding to inheriting either zero, one or two common genes from the same ancestor ($IBD = 0, 1, 2$). See [10, 9] for more details.

3 Parallel implementation

Our algorithm is based on the master-slave paradigm. In order to guarantee the scalability of the algorithm, a dynamic allocation method assigns tasks to processes at execution time. The advantage of this over static allocation is that it tends to keep processes busier. Suppose we have p processors and we run k processes $P_0, \cdots, P_{k-1}$. P_0 is the master process and the others are slave processes.

The algorithm works as follows. The master process is responsible for the distribution of the work to slaves. At the beginning the master assigns a task to each slave P_i where $0 < i < k$. Then, it waits for a WORK-REQUEST message from P_i, which means that P_i is idle. When the WORK-REQUEST message arrives, the master identifies the slave and checks if there are still any tasks to be executed. If so, it will send a task to the slave asking for a work. Otherwise, the master sends an END-SIGNAL requesting the slave to stop working.

Each slave process works as follows. At the beginning it sends a WORK-REQUEST message to the master and waits for a new assignment or for an END-SIGNAL. If an END-SIGNAL is received, the slave stops. Otherwise, it stores the message data from the received message and starts the execution of the task.

The master processor uses a simple data structure to manage the load balancing: a vector of size k, where k is the number of processes. At the beginning, tasks are allocated sequentially based on processor identity. Then, when

[3] type parent: parent with available genetic information. If parent is dead, no genetic information is available.

a slave finishes one task the master sends the next task from the task vector. Thus, the load balancing is done relative to the number of families, and not to their size (and hence computational complexity). Evaluating the performance of our algorithm is shown in The section 4.

4 Experimental Results

We tested our program on three types of architecture, including both distributed and shared memory platforms(SMP): a network of workstations (NOW) Ultra Sparcs10 with 128 Megabytes of memory, with 750MHz and 100 Mbit Ethernet card and an MPI standard software; a Sun Enterprise HPC 3500, which is an SMP system with eight 400MHz UltraSPARC-II processors, 8 Gbyte of shared memory and 72 Gbyte of disc space; and on a SunFire 6800 SMP system, with twenty-four 750MHz UltraSPARC-III processors, 48 Gbyte of shared memory each, and 120 Gbyte of disc space. All three architectures use Solaris as operating system.

The experiments on the network of workstations (NOW) were launched on a maximum of 10 processors. On the HPC 3500 we use up to 8 processors and on the SunFire 6800 we use up to 24 processors. In addition, the times reported from our experiments are the average of ten executions.

In the case of the network of workstations we run in a way such that each process is assigned to one processor, except for the master whose work is only to assign processes to processors. On the HPC 3500 we run in interactive mode, sharing the resources with others users. The operating system scheduler takes care of the process to processor assignments. On the SunFire 6800, we run our jobs in batch mode, which guarantees one process per processor.

On all platforms, all our tests (Table 1) are based on four parameters. The first parameter is the number of families in the input file to be analyzed; the second parameter is the number of markers; the third parameter is the informativity of the marker, and the last parameter is the familial structure. It is worth explaining the meaning of third parameter and the fourth parameter. The human genome is composed of 23 pairs of chromosomes. Each individual inherits one chromosome of each pair from his mother and his father. So, each individual has two versions of each markers (2 alleles). When these two versions present different variants(alleles), an individual is heterozygote for this marker. Then 50% of homozygote means that each founder is homozygote for 50% of the markers. Therefore, it is half informative.

For the fourth parameter consider a family composed of two generations: the first generation consists of grandparents and parents and the second generations consist of parents and children. For our tests both generations in a family (noted as Fam.) are of the form two parents (noted as pa.) and two children (noted as ch.).

The following figures are different variations of all parameters according to table 1 and their relatives efficiency.

Table 1 shows the values of these parameters for the eight test cases (labelled A-H) used in our experiments.

Case	#Fam.	#Markers	Informative	Fam. structure
A	200	10	Yes	2 pa. & 5 ch.
B	400	10	Yes	2 pa. & 5 ch.
C	200	20	Yes	2 pa. & 5 ch.
D	200	10	Half	2 pa. & 5 ch.
E	200	10	Yes	0 pa. & 5 ch.
F	200	10	Yes	2 pa. & 3 ch.
G	200	10	Yes	0 pa. & 3 ch.
H	200	10	Yes	3 generations

Table 1. Parameter values for different test cases.

Case	NOW	HPC 3500	SunFire 6800
A	75.01	54.431	28.568
B	143.76	109.989	57.545
C	132.38	101.072	50.367
D	78.07	56.274	29.598
E	69.27	54.217	28.22
F	12.74	9.201	4.58
G	13.16	9.65	4.73
H	23.56	17.33	9.14

Table 2. Execution times on one processor

Table 2 shows the sequential (one processor) execution time for each test case on each of the three systems.

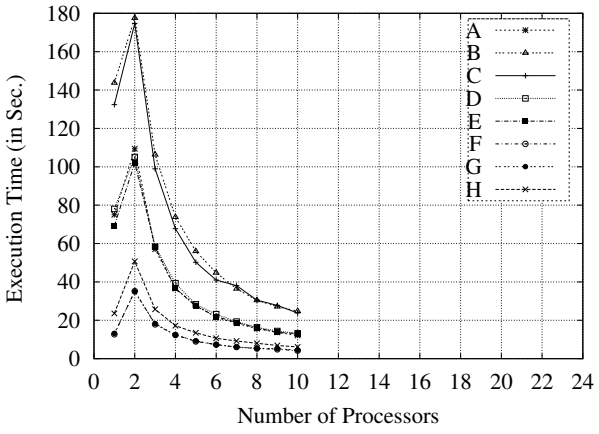

Figure 1. Execution time of all test cases on NOW.

Figure 1 shows the execution time of all the test cases with on the network of workstations. The running time on

two processors is greater than on one processor, because the master process is not doing any execution, but only assignment of jobs and the second processor is the only worker slave.

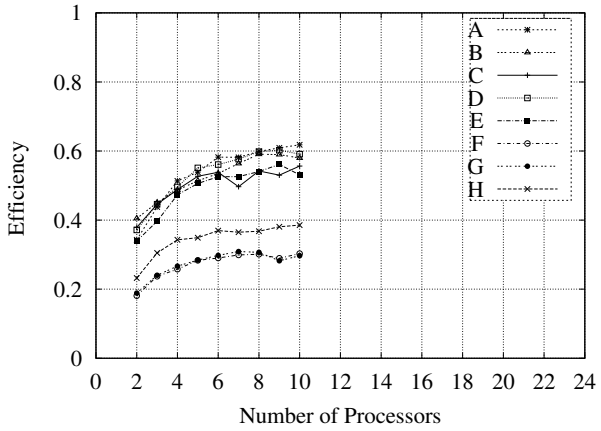

Figure 2. Efficiency of all test cases on NOW.

Figure 2 shows the efficiency of all the test cases on the network of workstations. The efficiency initially decreases from one to two processor, but then tends to increase with the number of processors. The small variations in the graph result from running our tests in interactive mode on this architecture.

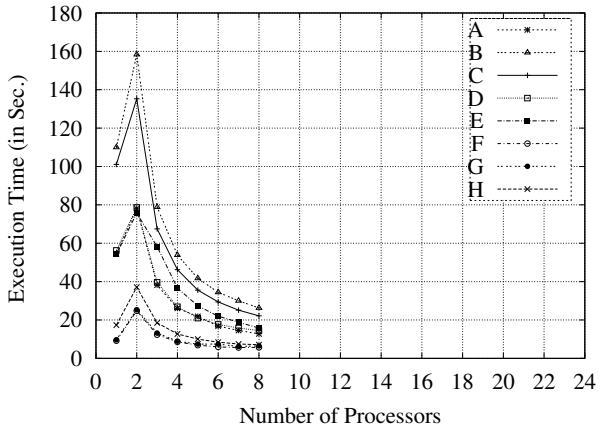

Figure 3. Execution time of all test cases on HPC 3500.

Figure 3 shows the execution of all the test cases on the HPC 3500. We notice that the overall shape of curves is the same as on the NOW, and the enhancement of the execution time is due to the faster clock speed. For example, the range of values for case B in Figure 1 is approximately 143 seconds on one processor to 30 seconds on 8 processors. However, the range of values for case B in Figure 3 is approximately 109 seconds on one processor to 23 seconds on 8 processors.

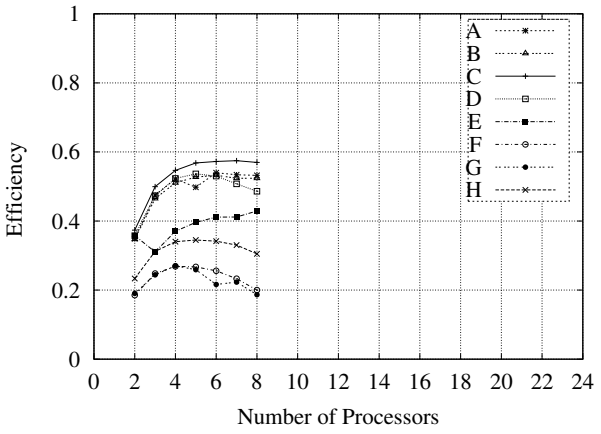

Figure 4. Efficiency of all test cases on the HPC 3500.

Figure 4 shows the efficiency of all test cases on the HPC 3500. Notice that when the number of markers increases (Case C) the efficiency is highest. Also, when the parents' genotype is unknown (Case E) the efficiency increases with the number of processors.

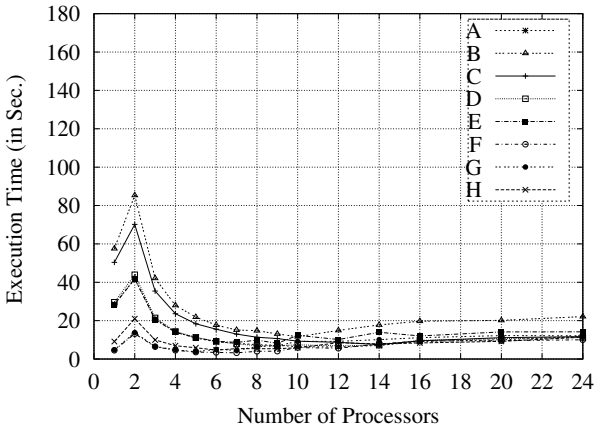

Figure 5. Execution time of all test cases on the SunFire 6800.

Figure 5 shows the execution time of all test cases on the SunFire 6800. The machine's CPU speed noticeably decreases the execution time of all test cases. The range of

execution times when the number of families is doubled is higher (Case B) than the range when the number of marker is doubled (Case C). In all figures, these two tasks are the most time consuming.

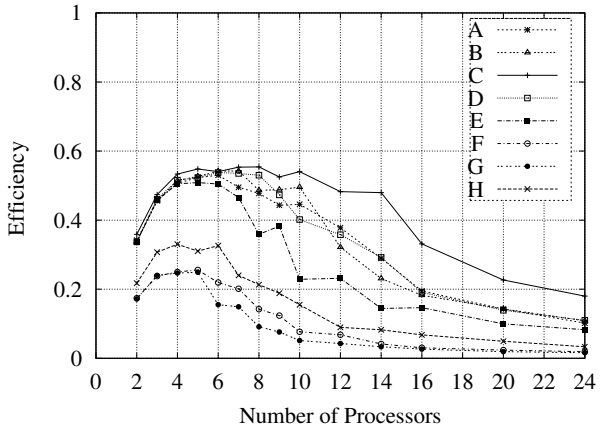

Figure 6. Efficiency of all test cases on the SunFire 6800.

Figure 6 shows the efficiency of all test cases on the SunFire 6800. The decrease of efficiency when the number of processor is greater than 8 is due to the high number of communications between processors, relative to the small execution time (approximately 0.6 ms) for each task.

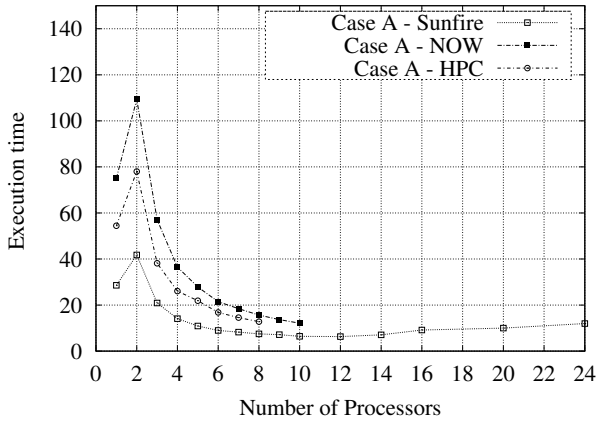

Figure 7. Execution time for Case A on different architectures

Figure 7 shows the execution time for Case A on the different architectures. The use of the SunFire 6800 is the fastest up to 15 processors—even though the running time increases when the number of processors is greater than 8, it is still less the execution time on the other two architecture.

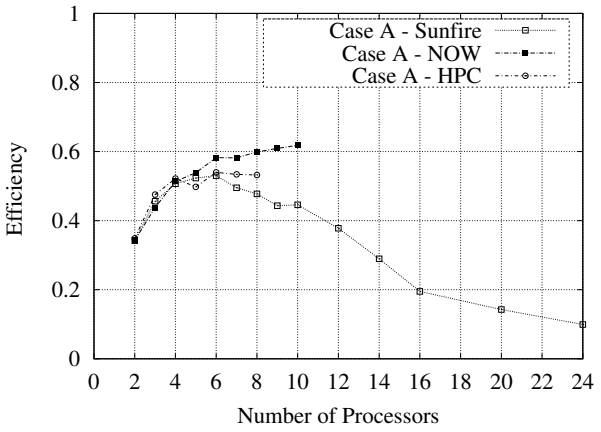

Figure 8. The efficiency of Case A on all architectures.

Figure 8 shows the efficiency of Case A on different architectures. The efficiency on the network of workstation is the best and may be better still if we were to increase the number of processors.

5 A general parallelisation strategy for Genehunter

Our approach is based on the contents of the file contain pedigrees, so that depending on certain variables we will select the most suitable model of parallelism. We distinguish three models of parallelism: the first model, high-level parallelism, is the use of master slave model described above which assigns families dynamically to processors. The second model, low-level parallelism, is that developed by Conant [1] which assigns different markers to different processors, and the third model is a combination of both models.

Suppose that $c1, c2, c3, c4$ are respectively the values of the four parameters: the first parameter is the number of families in the input file to be analyzed; the second parameter is the number of markers; the third parameter is the informativity of the marker, and the last parameter is the familial structure. Our approach is to select the appropriate model of parallelism based on the values of $c1, c2, c3, c4$.

Increasing the value of $c1$ while keeping $c2, c3, c4$ fixed (e.g. Case B) increases the complexity linearly. However, the execution time could be reduced by sorting the families in descending order of number of individuals before starting the dynamic allocation. This would minimise the risk of load imbalance resulting from assigning an expensive task

near the end of the computation. Thus, the first model is the most suitable in case of large values of the parameter **number of families**.

Increasing the value of $c2$ while keeping $c1, c3, c4$ fixed (e.g. Case B) also increases the complexity linearly. However in this case, the first model is not optimal. Since the second model assigns markers to different processors, this may be the most efficient in case of large values the parameter **number of markers**.

The change of value of the parameter **informativity of markers** $c3$, has no influence on the running time (compare Cases A and D). This parameter can be ignored when determining the best parallel strategy.

Finally, the last parameter **familial structure** $c4$ is more complicated. Execution time depends on the number of members in a family as well as the number of generations. As the same input file may contain different families of different sizes and generations, we propose that after sorting it in descending order all families, we apply the second model to sufficiently large families. Once family size decreases to a point where this model is no longer efficient, we can apply the first model to the remaining families. Thus a combined model may be the most efficient in case of changes in the parameter **familial structure**.

6 Conclusion and Future work

We have discussed in this paper the parallelization of the popular Genehunter package used to solve the Identity by Descent problem. We have been motivated for this study by the huge time and memory consumption of the original Genehunter program which is a real drawback for researchers in the domain. Based on the observation that a treatment of family can be considered as an independent task, we have showed that the master-slave paradigm can be successfully applied in order to reduce significantly the time and memory requirements in cases where low-level parallelism fails to be the optimal solution.

Future work includes the implementation of the combined model on different platforms, and to use a classification of parameters so that the appropriate model of parallelism can be selected. Finally, this result shows that the linkage analysis problem can be a classification problem to predict the optimized complexity based on the selection of the appropriate model.

7 Acknowledgements

I would like to acknowledge the support of the European Commission through grant number HPRI-CT-1999-00026 (the TRACS Programme at EPCC). Moreover, special thanks to Dr. Mark Bull for providing valuable support and input for accomplishing the aims of this research study.

References

[1] G. Conant, S. Plimpton, W. Old, A. Wagner, P. Fain, and G. Heffelfinger. Parallel genehunter: Implementation of a linkage analysis package for distributed-memory architectures. In *Proceedings of the First IEEE Workshop on High Performance Computational Biology, International Parallel and Distributed Computing Symposium*, 2002. 16 electronic.

[2] W. I. for Biomedical Research. http://www-genome.wi.mit.edu.

[3] L. Kruglayk, M. R.-D. M.J. Daly, and E. Lander. Rapid multipoint linkage analysis of recessive traits in nuclear families, including homozygosity mapping. In *Am. J. Hum. Genet.*, volume 56, pages 519–527, 1995.

[4] L. Kruglayk, M. Reeve-Daly, and E. Lander. Parametric and nonparametric linkage analysis: a unified multipoint approach. In *Am. J. Hum. Genet.*, volume 58, pages 1347–1363, 1996.

[5] E. Lander and P. Green. Construction of multilocus genetic linkage maps in humans. In *Proc. Natl. Acad. Sci. USA*, volume 84, pages 2363–2367, 1987.

[6] S. Lin. A scheme for constructing an irreductible markov chain for pedigree data. In *Biometrics*, volume 51, pages 318–322, 1995.

[7] J. Olson, J. Witte, and R. Elston. Tutorial in biostatistics genetic mapping of complex traits. In *Statist. Med.*, volume 18, pages 2961–2981, 1999.

[8] S. Ray and M. Craven. Representing sentence in hidden markov models for information extraction. In *Proceedings of the 17th International Joint Conference on aritifical Intelligence*, 2001.

[9] C. Smith and D. Stephens. Simple likelihood and probability calculations for linkage analysis. In *Genetic Mapping of Disease Genes eds I.H Pawlowitzki, J.H Edwards, E.A Thompson*, pages 73–96. Academic Press, London, 1997.

[10] A. Whittemore and J. Halpern. Probability of gene identity by descent: computation and applications. In *Biometrics*, volume 56, pages 109–117, 1994.