Integrating Parallelization Strategies for Linkage Analysis

Sandeep K. Gupta *
Department of Computer Science
Rice University
Houston

Alejandro A. Schäffer †
Department of Computer Science
Rice University
Houston

Alan L. Cox ‡
Department of Computer Science
Rice University
Houston

Sandhya Dwarkadas §
Department of Computer Science
Rice University
Houston

Willy Zwaenepoel ¶
Department of Computer Science
Rice University
Houston

Address for correspondence: Alejandro A. Schäffer Department of Computer Science, MS132, Rice University, 6100 Main, Houston, TX 77005-1892.
Phone: (713) 527-8101 x3813
FAX: (713) 285-5930

*Present address: Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139; skgupta@pdos.lcs.mit.edu

†schaffer@cs.rice.edu

‡alc@cs.rice.edu

§sandhya@cs.rice.edu

¶willy@cs.rice.edu

Abstract

We describe a second parallel implementation of the ILINK program from the LINKAGE package that improves on our previous implementation [*Human Heredity* 44(1994), pp. 127–141]. To improve running time we integrated the strategy of parallel estimation of the gradient at a candidate recombination fraction vector with a previously implemented strategy for evaluating in parallel the likelihood at one vector. We also integrated an adaptive loadbalancing strategy in conjunction with our previous static loadbalancing strategy. We implemented a strategy for partitioning input pedigrees, but this slowed down the program; we give some evidence for what the problems are. To best exploit parallelism at all levels of the program and to take advantage of both coarse-grain and fine-grain parallelism it is necessary to combine multiple algorithmic strategies.

1 Introduction

Genetic linkage analysis is a statistical technique used to map human genes and locate disease genes. As data collection methods have improved, the size and complexity of linkage analysis problems have grown much faster than the speed of readily available computers. In this paper we report on a multi-level attempt to parallelize the ILINK program from the LINKAGE package [15, 13, 16], the most widely-used software package for linkage analysis. This work represents a continuation of the FASTLINK project in which we have significantly speeded up the main sequential programs [3, 21] in LINKAGE and parallelized one of them (ILINK) [5].

There are several different levels at which linkage computations may be parallelized. In practice, linkage analysis computations usually consist of several different runs corresponding to different orders of possible genes and/or different candidate recombination fraction vectors. Also, any given run may involve likelihood computations for multiple pedigrees. Our work on parallelizing ILINK has focused on strategies that can achieve parallel speedup even when there is only one pedigree and one starting candidate recombination fraction vector. We feel that this is the limiting case for parallelizing the LINKAGE programs and that any comprehensive parallel implementation must try to do something about this hard case.

The principal theme of this paper is that it is possible to integrate different parallelization strategies in ILINK to improve the parallel speedup. All previous attempts at parallelizing linkage computations have tried essentially one strategy. Our experience suggests that multiple strategies are needed to handle different inputs and to work on different hardware platforms, but integrating parallelization strategies is a difficult programming task.

Our first implementation parallelized a single likelihood function evaluation at a fairly low level. Experiments with our first implementation showed that this was a good strategy for some data sets, but not so good for others. One of the problems was a low-level loadbalancing problem, which we have tried to address with an adaptive strategy. The introduction of an adaptive loadbalancing strategy is also an important step towards being able to run parallel likelihood function evaluations on heterogeneous parallel computers.

All the candidate recombination fraction vectors can be specified in advance, and multiple likelihood evaluations can be done in parallel in the LINKMAP and MLINK programs of the LINKAGE/FASTLINK package. The ILINK program does present a limited opportunity to do likelihood evaluations in parallel, during the estimation of the gradient. Doing likelihood evaluations in parallel can be advantageous because processors working on different evaluations do not need to share data with each other. The computation proceeds more quickly because sharing data causes significant communication overhead. Our new parallel implementation performs the likelihood function evaluations for gradient estimation in parallel, but the number of processors is still usually larger than the number of likelihood evaluations that can be done simultaneously. Therefore, we integrated the strategies of doing multiple likelihood evaluations simultaneously and having many processors work on one likelihood evaluation. This combined strategy is also likely to be useful for LINKMAP and MLINK and we plan to adapt our parallel code to those programs in the future.

We implemented a strategy of partitioning the input pedigrees and having different processors work on different nuclear families. Our implementation was similar to a theoretical proposal of Schork [22]. We found that partitioning the input pedigrees actually slowed the program down measurably; therefore, we do not devote any space in the body of the paper to explaining our partitioning implementation, but we briefly explain the problems we encountered in the Discussion at the end.

Both of our parallel implementations are written in a shared memory programming style using the TreadMarks distributed shared memory system [9], which is under development at Rice University. TreadMarks is a runtime library that enables a shared memory program to run on a network of workstations. Networks of workstations are quite common in research institutions and are more readily available than shared-memory multiprocessors. However, after simple syntactic changes, it is also possible to run our parallel implementation on a shared-memory multiprocessor computer. The shared-memory programming model supported by TreadMarks is easier to use than message passing.

We tried our parallel implementation on 5 data sets, 3 of which were used in [5]. We used an 8-processor network with either a 100Mb/s ATM (Asynchronous Transfer Mode) switch, or 10Mb/s Ethernet. Using the faster ATM network, speedup on the 3 common data sets improved from 5.38, 3.15, and 5.73 to 6.04, 3.91, and 6.33 respectively. On the 2 new data sets speedup is 7.02 and 6.88. Using the slower Ethernet, the speedup on the 3 shared data sets improved from 3.82, 1.86, and 5.09 to 4.86, 2.80, and 5.71 respectively. Speedup on the 2 new data sets using Ethernet is 6.30 and 6.34.

The rest of this paper is organized as follows. Section 2 describes background information on LINKAGE, FASTLINK, and ILINK. Section 3 summarizes our first parallel implementation and compares our work with other attempts to parallelize linkage analysis computations. Section 4 describes our new adaptive load balancing strategy. Section 5 describes our strategy for parallelizing the gradient estimation step. Section 6 describes the computer hardware and software we used for measurements and a few other low-level algorithmic improvements that we made to our parallel implementation. Section 7 evaluates our new parallel implementation. We conclude with a short Discussion.

2 Summary of LINKAGE

The basic computational goal in genetic linkage analysis is to compute the probability that a recombination occurs between two loci L_1 and L_2. A locus is an identifiable location on the chromosome whose inheritance can be traced; it need not be part of a gene. The probability of recombination is called the *recombination fraction* and denoted by θ. Most programs in common usage estimate θ using a maximum likelihood approach. A thorough treatment of maximum likelihood linkage analysis is given in Ott's book [20]. In this section we highlight a few aspects of the LINKAGE/FASTLINK software package that are relevant to our parallel implementation of ILINK.

The recombination fraction can be generalized to more than two loci. Suppose $L_1, L_2, \ldots, L_{k+1}$ are different loci, conjectured to occur in that order. Then we define θ to be a vector of recombination fractions, $(\theta_1, \ldots, \theta_k)$, where θ_i is the recombination fraction between locus L_i and locus L_{i+1}. The task of estimating the θ vector for more than two loci, is called *multilocus analysis*.

A major advance of the LINKAGE package over its predecessor LIPED [19] is that LINKAGE supports multilocus analysis [15]. The LINKAGE package contains four related principal programs, LODSCORE, ILINK, LINKMAP, and MLINK. The FASTLINK package [3, 21] contains significantly faster sequential versions of these four programs. FASTLINK does not include the many auxiliary programs for preprocessing the data and postprocessing the results that come with LINKAGE. For this paper, LODSCORE can be viewed as a special version of ILINK, tuned to handle 2-locus problems, and MLINK can be viewed as a variant of LINKMAP.

For our purposes there is one fundamental distinction between ILINK and LINKMAP. ILINK starts with one candidate θ vector and improves θ through a numerical optimization algorithm implemented in the GEMINI package [10]; all of the components in the vector can change from one estimate to the next. LINKMAP takes the relative positions of all but one of the loci to be fixed and the position of one locus can be varied, either at the end or in a gap between two loci. In either case, LINKMAP generates a fixed, and computable a priori, set of candidate θ vectors and computes the likelihood for each. The routines to compute the likelihood for any particular θ are essentially the same in ILINK and LINKMAP (as well as LODSCORE and MLINK).

As in [5], we focus on parallelizing ILINK because it has the longest typical running times and is the hardest of the four programs to parallelize. Most of the routines that we have modified are shared between all four programs in the sequential FASTLINK code.

As with virtually all numerical optimization procedures, GEMINI can only guarantee to produce a locally optimum solution. The GEMINI package as used in ILINK proceeds in stages, which the code calls *iterations*. In the first iteration, the first likelihood function evaluation computes the likelihood of the candidate θ supplied by the user and then does k likelihood function evaluations, one for each dimension of the θ vector, to estimate the gradient at the initial θ. Each subsequent iteration uses the best previous θ and the gradient estimate to search for a better θ; the number of candidate θ vectors tried varies substantially. Once GEMINI has found a better θ and cannot immediately improve it, GEMINI again estimates the gradient at the new θ. When no better θ can be found in the direction of the gradient, a local optimum has been reached and the program exits shortly thereafter.

Each evaluation of the likelihood function is done by traversing the pedigree, ending at one person p called the *proband*. The goal is to compute for each multilocus genotype g, the conditional probability that p has genotype g, conditioned on the observed phenotype data for all the pedigree members and the candidate θ. This is done by visiting the nuclear families one at a time, and for each nuclear family updating the conditional genotype probabilities for that family member that connects the current nuclear family to the as yet untraversed part of the pedigree. The current nuclear family is updated conditionally on the nuclear families previously visited, its observed data, and θ.

Details of the algorithm that LINKAGE uses to order the nuclear families in a pedigree traversal and to handle loops can be found in documentation that comes with the distribution of sequential FASTLINK (ftp to softlib.cs.rice.edu, login as anonymous, cd pub/fastlink, and retrieve the files traverse.ps and loops.ps).

3 Previous Work on Parallel Linkage Analysis

Linkage analysis seems to be a particularly hard problem to parallelize in a way that works on all varieties of typical data sets. In this section, we briefly review attempts by other research groups to parallelize linkage analysis and review a few aspects of our first parallel implementation needed to understand the subsequent sections.

Miller, et al. [18] did a parallel implementation of LINKMAP using the Linda package. Their parallelization strategy assumes that that there are many pedigrees and/or many candidate θ vectors. They treat the evaluation of each likelihood for one pedigree as a separate task that can be assigned to one processor. If there are enough tasks, the load can be balanced effectively using a work-queue approach. Goradia, et al [6] did a similar parallelization of the linkage analysis program MENDEL [11, 12].

Vaughan [25] did a parallel implementation of LINKMAP using the ISIS package. Her algorithm does parallelize the case of one pedigree and one likelihood, but she did not present any data on running times. She was primarily concerned with balancing the load on a heterogeneous network.

Schork [22] proposed parallelizing the computation of one likelihood by assigning the probability updates of different nuclear families to different sets of processors. His paper does not describe any implementation. We experimentally implemented an algorithm similar to Schork's, but found that it slowed down ILINK measurably. Some of the problems we encountered are mentioned briefly in the Discussion at the end of of the paper.

Chamberlain, Franklin, Peterson, and Province [2] parallelized the gradient computation in the GEMINI optimization package using PVM in the context of two other application programs in genetics. However, in both of those programs, the number of dimensions in the optimization problem is typically much higher than the number of processors available, hence it is possible to assign each function evaluation to one processor and get good speedup. ILINK optimization problems typically have dimension only 2 or 3. We note also that [2] did not parallelize those function evaluations that update the parameters, which are therefore done one at a time by one processor each. They mention that it is important to integrate parallelization strategies for individual function evaluations with a strategy to parallelize the gradient estimation. This is precisely the strategy integration problem that we address in our work.

Our first parallel implementation parallelized the update of one nuclear family at a time. The reason for this is that in some pedigrees with many generations and many untyped (except for a disease locus) individuals, updating the conditional probabilities of the nuclear families in the top two generations take the bulk of the computing time.

We observed that the code for an update of a nuclear family is wrapped in two outer loops that iterate over all the possible genotypes of one parent and then all the possible genotypes of the other parent. This describes a rectangular iteration matrix R. The probability updates corresponding to each possible pair of genotypes (i.e. entry in R) can be done almost entirely in parallel. However, the time to do the updates varies widely from entry to entry. We found that with rare exceptions the update times for consecutive rows of R are similar because they typically correspond to genotypes that have the same heterozygosity pattern (i.e., which loci are homozygous and heterozygous) and similar alleles. Therefore, a round-robin strategy that assigns consecutive rows of R to consecutive processors works well.

In our first implementation, all the processors participate in a function evaluation, but one of them (which we designate processor number 0) plays a special role of controlling which processors get which parts of R and of collecting all the results.

The next section explains one flaw with the static loadbalancing strategy for partitioning R that occurs in some data sets and our new attempt to make the loadbalancing strategy more general.

4 Adaptive Load Balancing

By *adaptive load balancing* we mean collecting performance statistics during a program run and using those statistics later in the same run to balance the load among different processors. The statistics we collect are running times for small pieces of each nuclear family update. A theoretical examination of our first parallel implementation suggested that there could be significant load imbalance when many nuclear families had the property that for many parental genotype pairs (i, j):

1. Genotype i for the first parent is consistent with spouse and children, and

2. Genotype j for the second parent is consistent with spouse and children, but

3. Genotypes i, j are not simultaneously consistent with the children.

We call this the bad-pair property. A pair of parental genotypes that is feasible and consistent with the children is called a *good pair*. A simple one-child, one-locus example of the bad-pair property occurs when a child has a heterozygous genotype AB. Each parent may have the homozygous genotype AA or BB, but it is not simultaneously possible for both parents to have genotype AA or for both parents to have genotype BB. In this case the ordered pairs (AA,AA) and (BB,BB) are bad pairs.

The static load balancing strategy we used previously assumed that the vast majority of genotype pairs are not bad, which is true for many data sets and choices of loci. It can also work well if the bad pairs are distributed close to uniformly among the processors. Since the order of visiting nuclear families in each likelihood function evaluation is fixed, the determination of bad pairs depends explicitly only on the possible genotypes of parents and children (and implicitly on the previously visited nuclear families, which are always the same). Thus for a fixed nuclear family, the set of bad pairs is always the same. The set of bad pairs grows as the square of the number of joint genotypes and is therefore, too large to store.

Since the bad pairs are not consistent with Mendelian inheritance, they require no conditional probability updates. Our static load balancing strategy did not distinguish between good pairs and bad pairs in distributing genotype pairs to different processors. Performance is hampered when a disproportionate number of good pairs is distributed to one of the processors.

To address this problem, we compute timing statistics for each nuclear family on one of the early function evaluations. The rows are initially permuted into round-robin order as in our static loadbalancing strategy. For each row in the (permuted version of) genotype matrix R defined in Section 3, we compute how much time it takes to do all the computation for all the pairs in that row. This includes both the time to distinguish good pairs and bad pairs and the time to do the conditional probability updates for the good pairs.

In subsequent iterations we use a greedy strategy to partition the rows into pieces that will take roughly equal time. Since the number of rows is generally much larger than the number of processors, we can generally find consecutive sets of rows that take roughly 1/p of the total time each. We found that the time needed to do the first row on each processor, except the master processor, can be large due to the time for initial transfer of shared data over the network. Therefore, we artificially set the time for the first row to be exactly that for the second row when doing the adaptive row partitioning to be used in later function evaluations. The amount of space required to store the row times is linear in the number of genotypes; if space is tight, we can get rid of the information once the partitions are computed.

When both the adaptive loadbalancing and the parallel gradient estimation strategies are used, one cannot collect accurate timing measurements until the first function evaluation after the first gradient estimation. If there are k loci, this will be the $(k+1)$ st likelihood function evaluation. The reasons are that we want to do measurements when all processors work together on the function evaluation, but a few special things happen on the first function evaluation which distort the timing; also, the second through $(k+1)$st evaluations estimate the first gradient and may have only a subset of the processors working on each likelihood function evaluation. If k is 3 or 4, the typical number of function evaluations is about 30 or 40 respectively. Roughly 1/10th of the function evaluations must be done with a static loadbalancing strategy. Therefore, it is still necessary to have a good static loadbalancing strategy and integrate it with the adaptive strategy.

Due to the need to transfer data over the network, the extra time needed for the first row may be large enough that it is better to have only the master work on a particular nuclear family. We use a user-specified threshold of $((20+10p)p/(p-1))$msec on the "nuclear family overhead" to determine whether a nuclear family requires enough computation to justify splitting up the work. The threshold function attempts to balance two considerations. First, the communication time to share data goes up with the number of processors — this accounts for the $20+10p$ term, which increases with p. Second, the overhead can be amortized better as the number of processors increases because there are more processors to share the computational work — this accounts for the $p/(p-1)$ term, which decreases with p.

5 Parallel Gradient Estimation

Our second improvement in parallel ILINK is to perform likelihood function evaluations in parallel during estimation of the gradient. The optimization package GEMINI uses two different methods to estimate the gradient, forward differences and central differences. In each case, it perturbs the components of the current candidate θ one dimension at a time by a small amount, h_i for dimension i, and estimates that component, g_i, of the gradient by the observed rate of change in the likelihood function. In forward differences, g_i is computed as $(f(\theta+h_i)-f(\theta))/h_i$, where $\theta+h_i$ means add h_i to only the ith component of θ. In central differences g_i is computed as $(f(\theta+h_i)-f(\theta-h_i))/2h_i$. Central differences are more time-consuming than forward differences because they require twice as many likelihood function evaluations. The merits of both methods in LINKAGE are discussed and analyzed experimentally in [14]. The currently distributed versions of ILINK in both LINKAGE and FASTLINK start using forward differences and switch to central differences if successive values of θ appear sufficiently similar.

As suggested by Miller et al. [18] in the context of LINKMAP, performing likelihood function evaluations in parallel is a good strategy. However, in their tests they had enough function evaluations to assign each evaluation to only one processor. Different processors working on different function evaluations do not have to share data, eliminating significant amounts of communication. For two different values of the θ vector, θ_1 and θ_2, the running time to compute $f(\theta_1)$ and $f(\theta_2)$ on the same pedigree should be roughly the same, provided that neither θ_1 nor θ_2 has any zero components. The structure of GEMINI prohibits θ components from going to zero, so we can get reasonably good loadbalance provided the same number of processors work on each likelihood evaluation going on in parallel.

As a side remark, it is worth contrasting the opportunity for parallel likelihood function evaluations in ILINK to that in LINKMAP. In ILINK only those function evaluations used for gradient estimation can be done in parallel, but it appears safe to assume that those that can be done in parallel will take approximately the same amount of time. In LINKMAP, all the likelihood function evaluations can be done in parallel, but virtually every run of LINKMAP done in practice contains 1 or 2 function evaluations with at least one zero component. Thus, as we look to extend our parallel ILINK implementation to LINKMAP, the ability to integrate doing multiple evaluations on separate processors and doing a single evaluation on multiple processors will be crucial.

Typical ILINK problems have 3 or 4 loci and hence 2 or 3 dimensions in the θ vector. Larger problems are generally prohibitive in time (even with FASTLINK) and space. It is not unusual for sequential ILINK runs to take weeks; because of the need to run many different tests, we picked ILINK instances of moderate size to use in our timing experiments. Because ILINK problems are low-dimensional (unlike the applications of GEMINI considered in [2]), we often have more processors available than likelihood evaluations to do. To balance the load, we want the same number of processors to work on each likelihood function evaluation.

To assign the function evaluations to processor sets, we use the following greedy algorithm. Let n be the number of evaluations needed to estimate the gradient. Let p be the number of processors. Find the largest integer, $l <= n$ that divides p. Assign the first l evaluations to disjoint sets of p/l processors. If $n > l$, the remaining $n - l$ function evaluations are done one at a time, with all processors participating. This algorithm has the advantage that all the processors are kept busy, but it has the disadvantage that if the number of processors is not divisible by 2 or 3, we do not do any function evaluations in parallel.

As a simple example, suppose we want to perform 3 function evaluations and we have 8 processors. We do the first 2 evaluations in parallel on 4 processors each and do the third evaluation on all 8 processors.

To coordinate the function evaluations, we designate a *master* processor for each evaluation that plays the same organizing role that processor 0 plays when all processors work together. A few pieces of global data, such as the candidate θ must be different for the different evaluations going on simultaneously. We made these data into arrays (or if they were already arrays, increased the dimension by 1) that can be indexed by the evaluation number. Each processor knows which evaluation it is working on and which processor is its master, and uses these two indices to know from where to get its data and with which processor it must synchronize to share data.

When the gradient phase is over, we start the next iteration where all processors work on one function evaluation at a time.

6 Methods and Other Improvements

We evaluated the performance of our new implementation of parallel ILINK on a network multicomputer. The code is also easily portable to a shared-memory multiprocessor; we verified this claim on an SGI machine. A network multicomputer is simply a cluster of ordinary workstations connected by a general-purpose local area network, such as ATM (which stands for Asynchronous Transfer Mode), Ethernet, or FDDI (which stands for Fiber Distributed Data Interface and is a 100 Megabit/s local area network in which the stations are connected in the form of a ring). In contrast, a shared-memory multiprocessor is a single machine containing several processors that are connected by a specially-designed bus or dedicated network. Some tradeoffs between these two types of parallel computers are discussed in the Methods section of [5].

In a network multicomputer, processors communicate by passing messages with `send` and `receive` operations, while a shared-memory multiprocessor supports communication by reading and writing globally accessible memory. Most sequential programs, including ILINK, are more easily parallelized by writing code in terms of shared memory. To use message passing, the programmer must write additional code to copy data into and out of message buffers and perform `send` and `receive` operations.

To enable programmers to use networks of computers without writing message-passing programs we have developed a software *distributed shared memory* (DSM) system for network multicomputers called TreadMarks [9]. In essence, TreadMarks provides a shared memory abstraction to the programmer, and implements this abstraction efficiently using the underlying message passing system [8, 4]. The programmer writes the parallel program as if it were intended for a shared-memory multiprocessor, but the TreadMarks system enables the program to run on a network multicomputer.

Since we did the performance measurements on our first parallel ILINK, TreadMarks has been improved in various ways. Some of the improvements are improvements in functionality which were necessitated by the algorithms used in the new ILINK as well as another application program. Other changes were improvements in the basic communication protocols used in TreadMarks. To make the comparisons fair we also continued to use sequential code based on version 1.0 of FASTLINK as in our previous parallel implementation.

The current version of TreadMarks can provide accurate statistics on how many messages are sent and how many bytes they contain. It also comes with a profiler tool that provides a transcript of where communication occurs to simulate shared memory over the network.

We used the output of the TreadMarks profiler to guide five minor changes in the ILINK implementation to reduce the amount of communication. We would expect that on any system such as TreadMarks some minor tuning is helpful to adjust a parallel implementation to the parameters and implementation features of that parallel system. The changes we made to tune for TreadMarks come in two forms. Three changes essentially rearranged the initialization of some data structures, so that they are computed on the processor(s) that need(s) the data. Two other changes rearranged the declarations of certain small arrays, so that they are forced onto the same page of memory. The memory updates in TreadMarks are done on a page-by-page basis, so there are circumstances where having two data structures on the same page of memory enables TreadMarks to communicate the update with a single message, instead of two messages. These changes reduced the communication on all data sets; the running time improvement was much more significant on the BAD data set (described in next section) than on any of the others. We used the new code, instrumented so that it does static loadbalancing and no parallel gradient evaluations, for our baseline speedup measurements in the next section. This is because our primary purpose is to measure the effectiveness of our two major algorithmic improvements.

The network multicomputer used to perform our evaluation of parallel ILINK consists of 8 DECStation-5000/240 workstations, each with 24 Mbytes of memory, running the Ultrix version 4.3 operating system. All of the workstations are connected to an Ethernet and a high-speed ATM network. TreadMarks can utilize either the Ethernet or the ATM network. The interface for Ethernet is a standard component of the workstation. An important difference between Ethernet and ATM is that on an Ethernet the packets in the communication medium are available to all the processors, while on ATM the communication is point-to-point. The point-to-point communication can be exploited by algorithms, such as our parallel gradient computation, that partition the network into pieces in which no communication goes across the pieces. The interface for ATM is a Fore Systems TCA-100 network adapter card supporting communication at 100 Megabit/s. In addition, we used the same compiler, gcc 2.4.5 with -g3 -O2 flag for optimization, with both networks.

The shared-memory multiprocessor used to check the correctness of our parallel ILINK is a Silicon Graphics Iris 4D/380 memory running the IRIX Release 4.0.5 System V operating system. This machine has 8 processors that communicate via a dedicated bus. Unfortunately, the processors on this SGI machine are somewhat slower than our DECStation processors and we were not able to get single-user access to the machine for long enough periods of time to do meaningful timing measurements.

TreadMarks is still under development, but an early version is available for license. We have tried to keep the cost of a license low for universities and other nonprofit institutions. Contact the third author by electronic mail for information about TreadMarks.

We have begun the tasks of modifying our parallel ILINK to be more consistent with ILINK from the latest version of FASTLINK and of adapting the code to parallelize LINKMAP and MLINK as well. We hope to be ready to distribute parallel versions of all three programs in 1995. Contact the second author by electronic mail for a status report and also about obtaining the sequential FASTLINK programs.

7 Results

We present speedups for parallel LINKAGE with several input data sets. Uniprocessor execution times are given as well so that execution time differences may be inferred. We use two different network types - the commonly available Ethernet networks and the emerging ATM networks.

For consistency we use exactly the same three disease data sets and one sample run each from [5]. We also use one sample run each from two large data sets that we obtained subsequent to the publication of [5]. The new data sets appear to be more representative of the kinds of data sets that cause users of sequential ILINK to want faster alternatives.

- RP01: data on a large family, UCLA-RP01, with autosomal dominant retinitis pigmentosa (RP1) from the laboratory of Dr. Stephen P. Daiger at the University of Texas Health Science Center at Houston. As shown in [1], this pedigree had to be split into three pieces because computation on the whole family together was prohibitively long. RP01-3 denotes the analysis with the family split in three pieces.

- BAD: data on a portion of the Old Order Amish pedigree 110 (OOA 110), with bipolar affective disorder (BAD) from the laboratory of Drs. David R. Cox and Richard M. Myers at the University of California at San Francisco [17].

- CLP: Data on 12 families with autosomal dominant nonsyndromic cleft lip and palate (CLP) from the laboratory of Dr. Jacqueline T. Hecht at the University of Texas Health Science Center at Houston [7].

- ADNIV: Data on 1 large family with autosomal dominant neovascular inflammatory vitroretinopathy (ADNIV) provided by Drs. Ed Stone and Brian Nichols from the University of Iowa [24]. The family has 93 individuals, of whom 37 have unknown genotypes at all non-disease loci, and 9 have some unknown genotypes. The family has no loops.

- LGMD: Data on 4 families with one form of limb-girdle muscular dystrophy (LGMD) provided by Drs. Marcy Speer and Margaret Pericak-Vance from Duke University [23]. Altogether the families have 416 individuals, 269 of which are in the huge family number 39 (discussed at length in [23]). The families have no loops.

RP01-3	BAD	CLP	ADNIV	LGMD
4682	833	4085	9570	13011

Table 1: Uniprocessor execution times in seconds on a DECStation-5000/240

More detailed descriptions of the first three sets of pedigrees are given in [5] and diagrams can be found in the papers cited for each data set. The loci chosen for the RP01-3 data set have an allele product of $2 \times 6 \times 9 = 108$; this implies that the number of genotypes is $108 \times (108 + 1)/2 = 5886$. The loci chosen for the BAD, CLP, ADNIV, and LGMD data sets have allele products of $2 \times 4 \times 4$, $2 \times 4 \times 4 \times 4$, $2 \times 3 \times 4 \times 4$, and $2 \times 10 \times 7$ respectively. In all cases, the 2-allele locus is the disease locus. Of all these data sets, the ADNIV data set is most ideally suited to our parallelization strategy since it has only one family and the complexity of the analysis is not caused by loops.

Figures 1 through 10 show speedup performance for the five data sets on each of the ATM network and the Ethernet network. Each graph compares four parallel versions: our first parallel implementation with the minor changes described in Section 6 (base), a version with only adaptive loadbalancing (loadbalance), a version with only parallel gradient estimation (parallel thetas), and a version with both adaptive load balancing and parallel gradient estimation (integrated).

Comparing the version with no changes and the version with both changes on 8 processors, the ATM network speedups improved from (5.52,3.43,5.96,6.69,6.45) to (6.04,3.91,6.33,7.02,6.88) on the five data sets respectively, The speedups on the first three data sets for our first implementation [5] were (5.38,3.15,5.73) respectively.

Comparing the version with no changes and the version with both changes on 8 processors the Ethernet network speedups improved from (3.97,2.28,5.32,5.09,5.40) to (4.86,2.80,5.71,6.30,6.34). The speedups on the first three data sets for our first implementation [5] were (3.82, 1.86, and 5.09) respectively. The amount of improvement caused by adaptive loadbalancing and parallel gradient evaluation vary widely from data set to data set. The improvements become more apparent as the number of processors grows, which is not surprising because having more precise loadbalancing and reducing communication become more significant objectives with more processors. In fact with 2 and 4 processors there are a few anomalies (e.g. RP01-3 with 4 processors) where the integrated version is alightly slower than at least one other version, but these disappear with 6 and 8 processors.

The speedups on RP01-3, CLP, ADNIV, and LGMD are quite pleasing, but the BAD data set remains a problem. Table 2 tries to assess the reasons for imperfect speedup in BAD and the other data sets. We identified four distinct reasons for imperfect speedup:

1. Computations done outside the parallel code, which we call "overhead". Almost all of this computation is to initialize the genotype probabilities of individuals in the first nuclear family where they are encountered in a pedigree

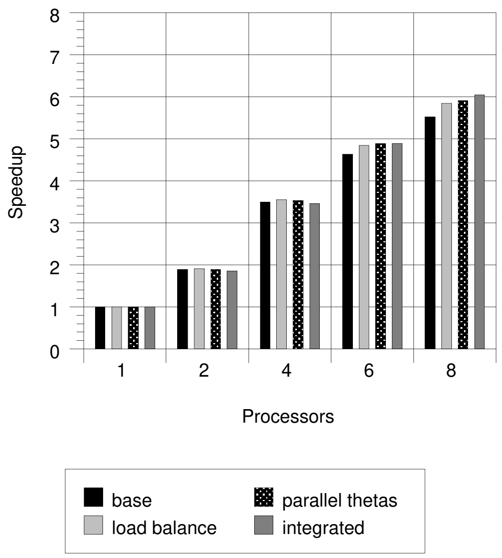

Figure 1: Speedup on an ATM Network – RP01-3

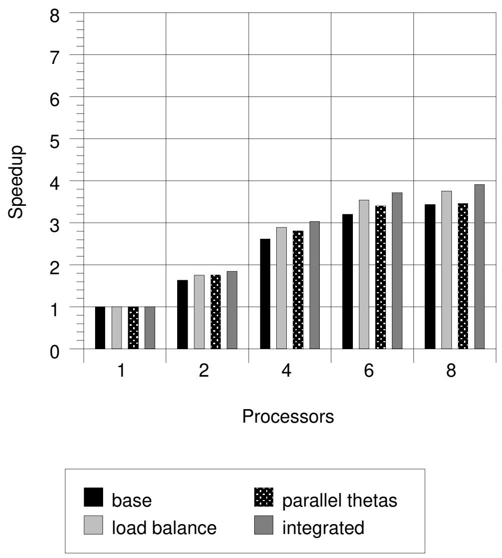

Figure 2: Speedup on an ATM Network – BAD

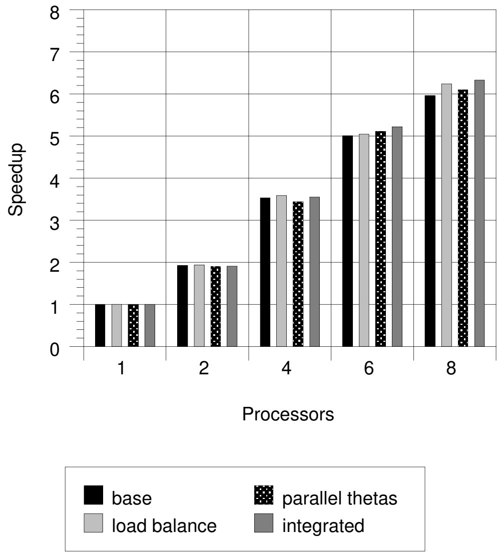

Figure 3: Speedup on an ATM Network – CLP

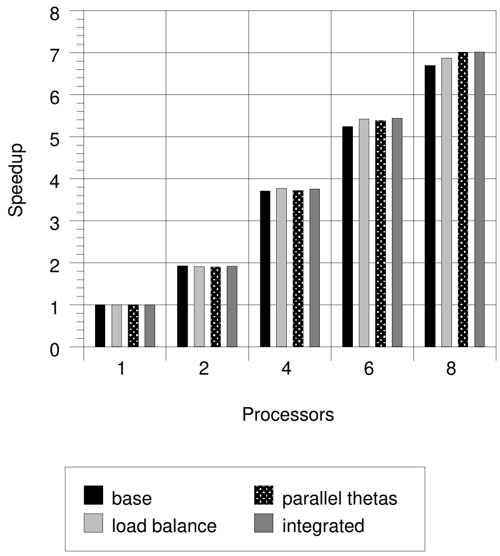

Figure 4: Speedup on an ATM Network – ADNIV

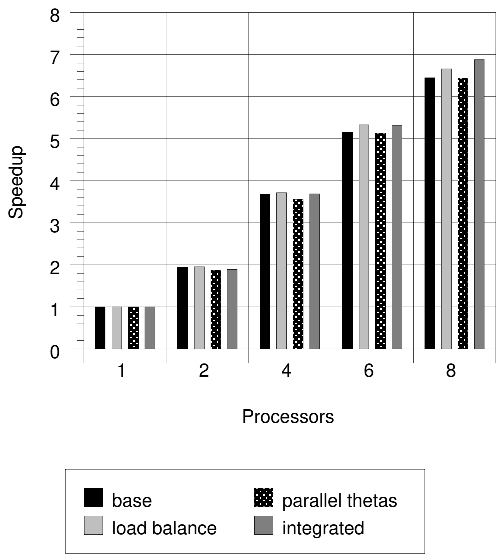

Figure 5: Speedup on an ATM Network – LGMD

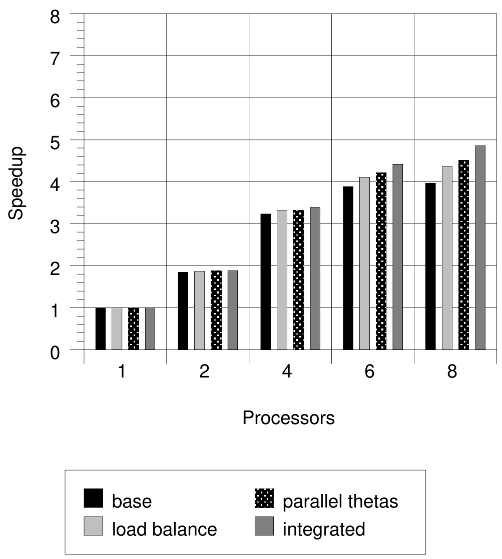

Figure 6: Speedup on an Ethernet Network – RP01-3

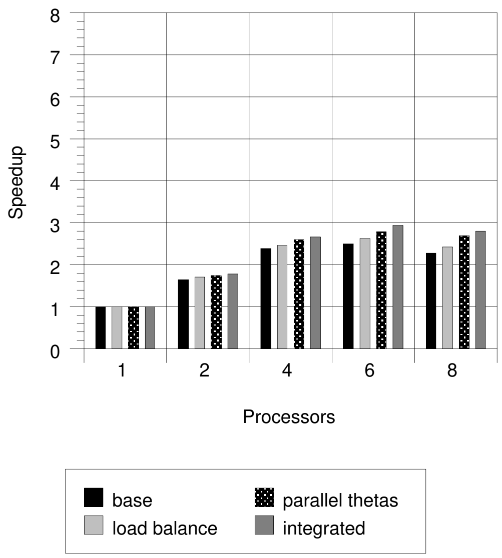

Figure 7: Speedup on an Ethernet Network – BAD

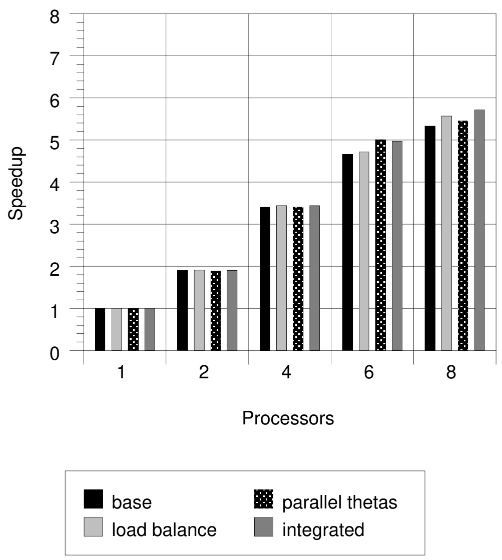

Figure 8: Speedup on an Ethernet Network – CLP

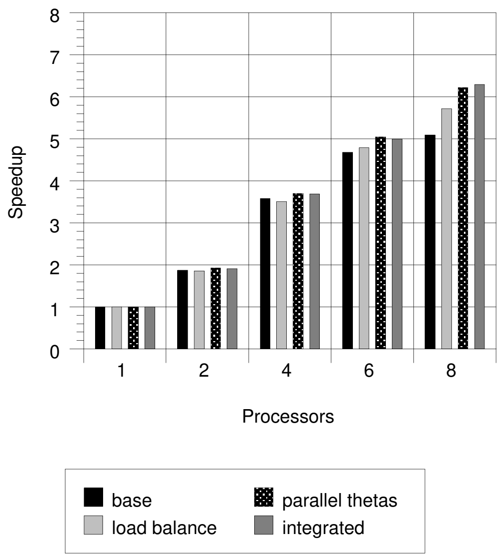

Figure 9: Speedup on an Ethernet Network – ADNIV

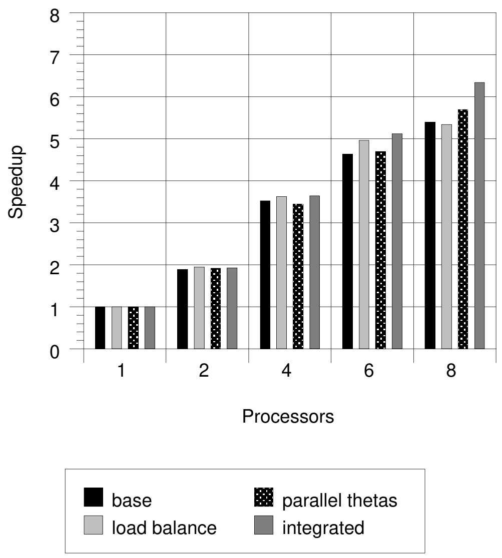

Figure 10: Speedup on an Ethernet Network – LGMD

	RP01-3	BAD	CLP	ADNIV	LGMD
1-proc. time	4682	833	4085	9570	13011
overhead	52	22	47	37	122
seq. families	29	18	40	20	40
seq. time	81	40	87	57	162
8-proc. time	775	213	645	1363	1890
ideal time	657	139	587	1246	1768
ratio	1.18	1.53	1.10	1.09	1.07

Table 2: Assessment of Reasons for Imperfect Speedup on 8 processors with ATM

traversal. This computation is done only by the master processor for the likelihood function evaluation.

2. Nuclear families whose updates are done sequentially by one processor. This is called "sequential families" (abbreviated as seq. families in Table 2). The sum of overhead and sequential families is called "sequential time" (abbreviated as seq. time in Table 2).

3. Communication of shared data.

4. Load Imbalance.

To assess the overhead and sequential time, we inserted timers into the code to measure the time spent outside the parallel code and the time spent updating nuclear families sequentially. It should be noted that there may be a slight "probe effect" in the figures.

To assess the combined cost of communication and imperfect loadbalance, we estimated what the 8-processor running time would be if there were no communication and if loadbalance were perfect for the nuclear families updated in parallel. This is called the "ideal running time" and is computed as:

$$(1 \text{ processor time} - \text{sequential time})/8 + \text{sequential time}.$$

The difference between the actual 8 processor running time and the ideal 8-processor running time is our estimate of the cost of communication and load imbalance. In Table 2, we instead show the ratio of actual 8-processor running time to ideal running time. The lowest ratio we could hope for would be 1.0; anything above 1.0 is an indication of inefficiency due to communication or load imbalance.

We tried to evaluate the performance of the adaptive loadbalancing strategy by itself, using one of the runs where speedup is not as good as on some others. We sought to measure how much variation there is in the work assigned to different processors, when a nuclear family is updated in parallel by all the processors. To measure the fluctuation we did the RP01-3 run reported later, measuring for each of 8 processors, each function evaluation, and each nuclear family, how much time

# Family 52 Max.	Family 52 Min.	Family 37 Max	Family 37 Min
14.27	13.89	1.13	1.01
14.08	13.84	1.08	0.98
14.21	13.98	1.07	0.98
14.23	14.00	1.09	1.00
14.21	13.99	1.13	1.03
14.29	13.99	1.07	0.98
14.16	13.91	1.07	0.98
14.21	13.98	1.09	0.96
14.22	13.99	1.08	1.00
14.21	13.98	1.08	0.99
14.23	14.00	1.07	0.98
14.16	13.92	1.07	0.98
14.21	13.98	1.08	0.98
14.20	13.97	1.08	0.98
14.21	13.97	1.09	1.00
14.14	13.90	1.10	0.98
14.16	13.93	1.07	0.98
14.23	13.98	1.06	0.98
14.21	13.98	1.08	1.01
14.22	13.97	1.09	1.00
14.23	14.00	1.07	0.98
14.14	13.92	1.07	0.98
14.22	13.99	1.07	0.98
14.22	13.97	1.10	1.00
14.22	13.99	1.10	0.98
14.23	13.98	1.07	0.98
14.14	13.91	1.06	0.98
14.15	13.93	1.07	0.98
14.25	14.00	1.09	1.00
14.23	13.98	1.09	1.00

Table 3: Maximum and minimum processor times (in seconds) for Two Nuclear Families; each row is a different likelihood function evaluation

that processor spent on that nuclear family in that function evaluation. We selected two of the families, including the one that takes the most time by far, and noted for each function evaluation the largest and smallest processor times. The data are show in Table 3.

As the reader can see, there is a pretty consistent difference of about .24s for family 52 and about .09s for family 37. There are two or three processors which consistently get the lightest loads and two or three processors which consistently get the heaviest loads. This is largely due to the effects of discretization on the partitioning problem. The effects are relatively larger for those families that have fewer possible genotypes.

To assess the relative importance of communication and load imbalance, we present four pieces of evidence that all suggest that the cost of communication is far more significant than the cost of load imbalance.

First, we can look at the data from Table 3. We assume (in all cases trying to increase the effect of load imbalance) that family 37 is roughly representative of all the other nuclear families updated in parallel, that load imbalance is roughly half the difference between maximum and minimum times, and that there is no probe effect. With these assumptions, the data in Table 3 would indicate that the overall load imbalance for our RP01-3 run is very roughly 20s. Even if this were too low by a factor of 2, it would still be less than half of the gap between the actual running time of 775s and the ideal running time of 657s. Because the load imbalance that remains occurs in very small quanta, it seems to be a very hard problem to fix.

Second, if we compare the baseline speedup against the speedup with just parallel gradient estimation and Ethernet vs. ATM, we see that parallel gradient estimation has a much larger effect on Ethernet than on ATM. An important difference between Ethernet and ATM is that communication is much faster with the ATM switch. The main virtue of parallel gradient estimation is that it significantly reduces the amount of communication for those function evaluations where it is applied because sets of processor working on different function evaluations do not need to communicate. For example, on the RP01-3 run using ATM and 8 processors, the parallel function evaluation reduced the amount of communication from approximately 359Mbytes of data shipped and 851K messages to 286Mbytes and 620K messages. However, the function evaluations where θ is being updated cannot be done in parallel. Therefore, it stands to reason that communication is still a major bottleneck on those function evaluations. We can see this in more detail with data for the RP01-3 run. That run has a total of 33 function evaluations, of which 14 involve estimating gradients and 19 involve updating θ. The running time using 8 processors and ATM without either strategy is 848s. The running time with just parallel gradient estimation is 793s. Therefore, we saved 55s essentially by reducing communication on those 14 function evaluations where the gradient is being estimated. If we extrapolate this to the other 19 function evaluations, the extra communication accounts for $(19/14) \times 55 = 75$s, which is the majority of the gap between the 8-processor running time with both strategies of 775s and the ideal running time of 657s in Table 2.

Third, several of the low-level changes described in the last section had the primary effect of reducing the number of messages sent. On the 8-processor ATM run of the BAD data set, they reduced running time by about 15s while reducing the total number of messages from about 197K to about 164K. If we extrapolate these figures, we see that the remaining 164K messages probably account for the bulk of the gap between the actual 8-processor running time of 213s and the ideal 8-processor running time of 139s.

Fourth, we tried to improve the running time on BAD by reparameterizing the loadbalancing strategy to reduce the load on the master processor which sets up each nuclear family update. We assigned the work so that this processor would get substantially less than $1/p$ of the work on those nuclear families that were split among all the processors. Reducing the amount of work for the first processor did not improve the running time measurably.

In general, the speedups for all data sets except BAD are quite good. From the data in Table 2 and the above discussion, we see that three reasons contribute to the inferior speedup obtained for BAD, but communication is the biggest contributor. The cost of communication is roughly 74s (213–139), as opposed to only 40s of sequential time. It seems very difficult and somewhat pointless to do much about the sequential time. Table 2 shows clearly that the sequential time tends to decrease in significance as the runs get longer; for example it accounts for 5% of the BAD 1-processor time, but only 1% of the LGMD 1-processor time.

The larger amount of communication per unit time for BAD is due primarily to the loop in the data set. Unfortunately the BAD data set has data specified for only 3 loci, so it is not possible to do a longer 4-locus run to understand the asymptotic behavior better. When pedigrees have loops, each function evaluation does multiple traversals of the pedigree. It would be natural to do the different loop traversals in parallel, but this is tricky because different traversals will take different amounts of time. Furthermore, Schäffer, Gupta, Shriram, and Cottingham [21] made a substantial improvement in the loop handling strategy in version 2.0 of FASTLINK. We are currently working on other fundamental improvements to the sequential algorithms for handling loops. To parallelize the loop traversals it would make sense to start with the improved sequential algorithm, but this would make it impossible to properly compare running times with our first parallel implementation of ILINK.

8 Discussion

The sequential algorithmic improvements in FASTLINK have helped many users, but they do not provide enough speedup to solve many problems that users would like to solve. In response to a questionnaire we sent out to several dozen sequential FASTLINK users, only one respondent reported that she did not need further speedup beyond that in our sequential code. An effective and comprehensive parallel implementation of FASTLINK is clearly called for. Previous attempts to parallelize LINKAGE have handled only some types of inputs and have not been used by the LINKAGE user community.

This paper describes the second major scientific step towards a truly usable parallel FASTLINK. We have completely integrated a strategy for parallel likelihood evaluations in a setting where it is applicable only during parts of the run, which is harder than if it were applicable throughout a run. These modifications allow us to exploit parallelism at different levels of the computation, where possible. They take advantage of the fact that newer networks, such as ATM networks, allow point-to-point communication.

We introduced a low-level adaptive loadbalancing strategy to protect against certain potential imbalance problems inherent in the data. Our adaptive loadbalancing strategy is implemented for a homogeneous network because that is what we have available. However, we think it would not be hard to modify the algorithm for a heterogeneous network with machines of different speeds. Since we compute running times for small pieces of each nuclear family update, we could balance the load on a heterogeneous network by weighting each row time by the relative speed of the processor that computes that row.

Each of the two parallelization strategies improves the speedup by a fraction of a processor on most runs with 6 or 8 processors using either Ethernet or ATM networks. The improvements close much of the gap between the speedup of our first parallel implementation and perfect speedup. The new parallelization strategies are particularly noticeable as the number of processors increases; this is not surprising because the speedups for our first implementation were very good for up to 4 processors, leaving little room for improvement there. The speedups on runs that take over 1 hour sequentially are quite satisfactory on the ATM, but the speedup on the one shorter run (less than 15 minutes sequentially) that we tried is still not very good. Users of sequential LINKAGE and FASTLINK often wait for 1–3 weeks for a sequential run, so it is the speedup on the long runs that matters, in practice.

We investigated the possibility of splitting up the likelihood function evaluation by assigning different nuclear families to different processors. This had been proposed by Schork [22] in a theoretical paper. We have not reported in detail on our trial implementation because we found that it consistently increased the running time measurably.

Qualitative observations of our implementation of Schork's idea in ILINK suggest the following difficulties:

1. Most of the nuclear families that take a lot of time to update are at the top of the pedigree, where the number of nuclear families whose probabilities can logically be updated in parallel is almost certain to be fewer than the number of processors.

2. Because of problem 1, the only way to get good speedup is to assign nuclear families to subsets of processors that have more than 1, but fewer than p processors. This is extremely hard to implement correctly let alone implement with good loadbalance.

3. At the lower levels of the pedigree, the nuclear families are updated so quickly that sending the data over the network (so that different processors work on different families in parallel) slows down the computation significantly.

Nevertheless, Schork's suggestion is theoretically appealing and we would be most interested to see if other researchers can get it to work effectively in FASTLINK.

Our integration fo multiple parallelization strategies in ILINK exploits different opportunities for parallelism. It is suited to networks with point-to-point communication, but also works reasonably well on Ethernet. By using a distributed shared memory system such as TreadMarks we have tried to anticipate what hardware potential users will have available. Our implementation can run either on a network of workstations or on a shared-memory multiprocessor. However, many FASTLINK users do not have access to more than one workstation. With this constraint in mind, several FASTLINK users have started to build servers, where linkage analysis problems can be submitted by electronic mail and the server returns the answer by mail. When our parallel code is ready for general usage and distribution we hope it will be used in such linkage servers.

Acknowledgments

We thank Dr. Stephen P. Daiger, Dr. Lori A. Sadler, Mr. Robert W. Cottingham Jr., Dr. David R. Cox, Dr. Richard M. Myers, Dr. Susan H. Blanton, Dr. Jacqueline T. Hecht, Dr. Ed Stone, Dr. Brian Nichols, Dr. Marcy Speer, and Dr. Margaret Pericak-Vance for contributing the disease family data for our experiments. Development of the RP data set was supported by grants from the National Retinitis Pigmentosa Foundation and the George Gund Foundation. The Amish family data was developed with the support of a grant from the National Institutes of Health. Development of the CLP data set was supported by grants from the National Institutes of Health and Shriners Hospital. Development of the LGMD data set was supported by grants from the National Institutes of Health and from the Muscular Dystrophy Association. This work was supported by grants from the National Science Foundation and the Texas Advanced Technology Program and a contract from IBM.

References

[1] S. H. Blanton, J. R. Heckenlively, A. W. Cottingham, J. Friedman, L. A. Sadler, M. Wagner, L. H. Friedman, and S. P. Daiger. Linkage mapping of autosomal

dominant retinitis pigmentosa (RP1) to the pericentric region of human chromosome 8. *Genomics*, 11:857–869, 1991.

[2] Roger D. Chamberlain, Mark A. Franklin, Gregory D. Peterson, and Michael A. Province. Genetic epidemiology, parallel algorithms, and workstation networks. Technical Report WUCCRC-94-14, Washington University, 1994.

[3] R. W. Cottingham Jr., R. M. Idury, and A. A. Schäffer. Faster sequential genetic linkage computations. *American Journal of Human Genetics*, 53:252–263, 1993.

[4] S. Dwarkadas, P. Keleher, A.L. Cox, and W. Zwaenepoel. Evaluation of release consistent software distributed shared memory on emerging network technology. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 144–155, 1993.

[5] S. Dwarkadas, A. A. Schäffer, R. W. Cottingham Jr., A. L. Cox, P. Keleher, and W. Zwaenepoel. Parallelization of general linkage analysis problems. *Human Heredity*, 44:127–141, 1994.

[6] T. M. Goradia, K. Lange, P. L. Miller, and P. M. Nadkarni. Fast computation of genetic likelihoods on human pedigree data. *Human Heredity*, 42:42–62, 1992.

[7] J. T. Hecht, Y. Wang, B. Connor, S. H. Blanton, and S. P. Daiger. Nonsyndromic cleft lip and palate: No evidence of linkage to hla or factor 13a. *American Journal of Human Genetics*, 52:1230–1233, 1993.

[8] P. Keleher, A. L. Cox, and W. Zwaenepoel. Lazy release consistency for software distributed shared memory. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pages 13–21, May 1992.

[9] P. Keleher, S. Dwarkadas, A. Cox, and W. Zwaenepoel. Treadmarks: Distributed shared memory on standard workstations and operating systems. In *Proc. 1994 Winter USENIX Conference*, pages 115–131, 1994.

[10] J. M. Lalouel. GEMINI - a computer program for optimization of general nonlinear functions. Technical Report 14, University of Utah, Department of Medical Biophysics and Computing, Salt Lake City, Utah, 1979.

[11] K. Lange, D. Weeks, and M. Boehnke. Programs for pedigree analysis: MENDEL, FISHER, and dGene. *Genetic Epidemiology*, 5:471–473, 1988.

[12] K. Lange and D. E. Weeks. Efficient computation of lod scores – genotype elimination, genotype redefinition, and hybrid maximum likelihood algorithms. *Annals of Human Genetics*, 53:67–83, 1989.

[13] G. M. Lathrop and J. M. Lalouel. Easy calculations of lod scores and genetic risks on small computers. *American Journal of Human Genetics*, 36:460–465, 1984.

[14] G. M. Lathrop and J. M. Lalouel. Efficient computations in multilocus linkage analysis. *American Journal of Human Genetics*, 42:498–505, 1988.

[15] G. M. Lathrop, J. M. Lalouel, C. Julier, and J. Ott. Strategies for multilocus linkage analysis in humans. *Proc. Natl. Acad. Sci. USA*, 81:3443–3446, 1984.

[16] G. M. Lathrop, J. M. Lalouel, C. Julier, and J. Ott. Multilocus linkage analysis in humans: detection of linkage and estimation of recombination. *American Journal of Human Genetics*, 37:482–498, 1985.

[17] A. Law, C. W. Richard III, R. W. Cottingham Jr ., G. M. Lathrop, D. R. Cox, and R. M. Myers. Genetic linkage analysis of bipolar affective disorder in an old order amish pedigree. *Human Genetics*, 88:562–568, 1992.

[18] P. L. Miller, P. Nadkarni, J. E. Gelernter, N. Carriero, A. J. Pakstis, and K. K. Kidd. Parallelizing genetic linkage analysis: A case study for applying parallel computation in molecular biology. *Computers and Biomedical Research*, 24:234–248, 1991.

[19] J. Ott. Estimation of the recombination fraction in human pedigrees– efficient computation of the likelihood for human linkage studies. *American Journal of Human Genetics*, 26:588–597, 1974.

[20] J. Ott. *Analysis of Human Genetic Linkage*. The Johns Hopkins University Press, Baltimore and London, 1991. Revised edition.

[21] A. A. Schäffer, S. K. Gupta, K. Shriram, and R. W. Cottingham Jr. Avoiding recomputation in linkage analysis. *Human Heredity*, 44:225–237, 1994.

[22] N. Schork. The Parallel Computation of Pedigree Likelihoods. In *Proc. First International Conference on Intelligent Systems for Molecular Biology*, pages 371–379, 1993.

[23] M. C. Speer, L. H. Yamaoka, J. H. Gilchrist, C. P. Gaskell, J. M. Stajich, J. M. Vance, Z. Kazantsev, A. Lastra, C. S. Haynes, J. S. Beckmann, D. Cohen, J. L. Weber, A. D. Roses, and M. A. Pericak-Vance. Confirmation of genetic heterogeneity in limb-girdle muscular dystrophy: Linkage of an autosomal dominant form to chromosome 5q. *Am. J. Hum. Genet.*, 50:1211–1217, 1992.

[24] E. M. Stone, A. E. Kimura, J. C. Folk, S. R. Bennett, B. E. Nichols, L. M. Streb, and V. C. Sheffield. Genetic linkage of autosomal dominant neovascular inflammatory vitreoretinopathy to chromosome 11q13. *Human Molecular Genetics*, 1:685–689, 1992.

[25] M. S. Vaughan. A distributed approach to human genetic linkage analysis. M. S. Thesis, Department of Computer Science, Duke University, 1991.