A Platform for Biological Sequence Comparison on Parallel Computers

A. S. Deshpande
D. S. Richards
W. R. Pearson

Computer Science Report No. TR-90-25
Sptember 3, 1990

submitted to CABIOS
Aug 29, 1990

A Platform for Biological Sequence Comparison on Parallel Computers

A. S. Deshpande, D. S. Richards,

Department of Computer Science
University of Virginia
Charlottesville, VA 22903

and W. R. Pearson*

Department of Biochemistry
University of Virginia
Charlottesville, VA 22908

*To whom correspondence should be addressed. Dept. of Biochemistry, Box 440 Jordan Hall, Charlottesville, VA, 22908 (804) 924-2818

ABSTRACT

We have written two programs for searching biological sequence databases that run on Intel hypercube computers. PSCANLIB compares a single sequence against a sequence library, and PCOMPLIB compares all the entries in one sequence library against a second library. The programs provide a general framework for similarity searching; they include functions for reading in query sequences, search parameters, and library entries, and reporting the results of a search. We have isolated the code for the specific function that calculates the similarity score between the query and library sequence; alternative searching algorithms can be implemented by editing two files. We have implemented the rapid FASTA sequence comparison algorithm and the more rigorous Smith-Waterman algorithm within this framework. The PSCANLIB program on a 16-node iPSC/2 80386-based hypercube can compare a 229 amino acid protein sequence with a 3.4 million residue sequence library in about 16 seconds with the FASTA algorithm. Using the Smith-Waterman algorithm, the same search takes 35 minutes. The PCOMPLIB program can compare a 0.8 million amino acid protein sequence library with itself in 5.3 minutes with FASTA on a third-generation 32-node Intel iPSC/860 hypercube.

INTRODUCTION

Improvements in molecular cloning and DNA sequencing techniques have dramatically increased the rate with which protein and DNA sequences are determined. These breakthroughs in molecular biology have stimulated interest in determining the complete sequence of the entire human genome over the next 15 years. As a result of the human genome initiative, the DNA sequence databases will increase by more than 75-fold from their current 40 million residues, and the protein sequence databases will increase about 10-fold from their current 5 million residues. The biochemical techniques required to characterize a newly sequenced protein have not kept up with improvements in sequencing and computer technology. As a result, the first characterization of a protein sequence frequently involves the comparison of the sequence against the entries in a protein or DNA sequence database. In anticipation of rapid growth in these databases, research efforts have been devoted to improving the speed and sensitivity of biological sequence comparison algorithms.

Much research in biological sequence comparison seeks to identify those features of protein and DNA molecules that can be extracted from the sequence alone. The feature that is most easily extracted from sequence data is common evolutionary ancestry. Excellent algorithms for calculating optimal similarity scores and sequence alignments have been available for twenty years (1-3); these methods are capable of demonstrating common evolutionary ancestry for proteins that diverged more than two billion years in the past (4). Since proteins that have evolved from a common ancestor always have similar structures, and frequently have related functions, demonstration of common ancestry can suggest a variety of hypotheses that can be tested experimentally.

Traditional methods for comparing biological sequences use a dynamic programming strategy that requires $O(n^2)$ time to compare two sequences of length n. While there are considerably more efficient algorithms for finding exact matches in strings, common evolutionary ancestry can be convincingly demonstrated in sequences that share less than 20% sequence identity if weighted replacement penalties are used (4). Methods designed for exact matches or a small number of mismatches are poorly suited for

protein sequence comparison. More recently, techniques have been developed that focus on the subset of alignments between the two sequences that are most likely to yield the best similarity score (5-7). These methods have reduced the time required to compare DNA and protein sequences more than 50-fold, so that comparison of a single sequence against a library of all known protein sequences (4 million amino acids) requires only a few minutes on a Sun workstation.

As new methods for biological sequence comparison are developed, they must be evaluated on the basis of their sensitivity – the ability to identify very distantly related sequences – and their selectivity – the ability to avoid high scores for unrelated sequences. The statistical properties of biological sequences and sequence comparison algorithms are poorly understood, so that evaluation of a new comparison method frequently involves many sequence comparisons, and, sometimes, complete comparison of all the members of a protein sequence library (6000^2 sequence comparisons). Lander et al. have used a large Connection Machine to evaluate scoring parameters for the dynamic programming algorithm (8).

To provide a flexible platform for large scale sequence comparisons, we have developed and implemented two general sequence comparison programs on an Intel hypercube parallel computer. These programs are designed to provide the framework that any biological sequence comparison program might need: functions to read a query sequence and search parameters, and to apply a comparison function to all the entries in a sequence library. The PSCANLIB program can be used to compare a single query sequence to a sequence database. We have tested versions of PSCANLIB that use either the FASTA algorithm (7, 9) or the more rigorous Smith-Waterman algorithm (2). The PCOMPLIB program is designed to compare all the sequences in one library to all the sequences in another. Although rapid methods for comparing a single sequence to a sequence library are widely available (7), library-library comparison is a task that requires the performance of parallel computers.

Comparison of a sequence or set of sequences to a sequence database is a problem that would appear to be easily mapped onto a parallel computer with a hypercube architecture. The problem can be broken up into many independent sequence comparison problems, one for each sequence in the library,

and performed in parallel. Surprisingly, our first implementation of FASTA on the hypercube performed at 1/10 the expected speed, due to the high communications costs on the iPSC/2 hypercube. The optimizations required to obtain reasonable performance with a rapid sequence comparison algorithm are described below.

SYSTEM and METHODS

Hardware

The programs were developed on an Intel iPSC/2 hypercube (10, 11). The Intel iPSC/2 System with I/O consists of a set of compute-nodes connected in a hypercube topology with an attached set of I/O nodes and a host computer known as the System Resource Manager (Figure 1).

Each iPSC/2 node contains an Intel 80386 32 bit microprocessor with 4 Mbytes of memory. The 16 MHz version that we used is rated at 4 MIPS. Each node is equipped with a Direct Connect Module (DCM) for high speed routing of messages between nodes. For a node to communicate with another node, a series of switches are closed, a path is established, and messages proceed at the full hardware speed of 2.8 Mbytes/second. Only the sending and receiving processors are involved in the communication. An entire message is transmitted when sufficient buffer space is available on the receiving node. Since the entire message is sent at once, large messages take about as much time to transmit as small messages. Because of the fixed time for building a communications path, buffering messages between the nodes to generate a smaller number of larger messages can result in a dramatic reduction in communication overhead.

Storage for database files is provided by an assembly of disk I/O notes and interleaved disks called the Concurrent File System (CFS). Compute nodes on the hypercube communicate with the CFS through a separate set of disk I/O nodes that do not run application processes directly. The I/O subsystem consists of several disks served by parallel I/O channels. Each I/O node also has full access to the hypercube interconnect of the compute nodes. The CFS views all the disks in the system as a single logical disk with a

single file system. A file on the CFS may be distributed to all the disks using a technique known as declustering. As a result, the data required by multiple compute nodes, whether from a single or multiple files, is likely to be on separate disks and can be transferred simultaneously. Hence, large parallel applications can obtain access to large data files concurrently from multiple nodes at a high aggregate data rate.

The hypercube nodes have little input/output capability of their own. All keyboard input goes through the host processor, although the node programs are capable of producing output to the screen. Node programs can read from and write to files on either the host file system or the CFS.

The iPSC/2 system that we used for our experiments consisted of 16 compute nodes and 4 I/O nodes with 4 disk drives. Each drive had a capacity of about 380 megabytes, formatted, for a total of about 1.5 Gigabytes of mass storage.

We have also conducted some experiments on Intel's third-generation iPSC/860 hypercube. This system had 32 compute nodes and 8 I/O nodes. Each compute node on the iPSC/860 uses an Intel i860 64 bit microprocessor. The 40 MHz version that we used has a rating of 27 MIPS. The host uses an Intel 80386 processor. This computer was running a preliminary release of the operating system and compilers; more mature versions are expected to perform better.

Accurately timing program execution on the Intel hypercubes is difficult. Each of the nodes in the hypercube has its own clock, which is not synchronized with other clocks in the cube. Hence, we must keep track of execution patterns on each node to generate accurate timing information. Since the clocks on the nodes are independent, we cannot know precise start and finish times. However, we can deduce node timings by distributing the start times of each node uniformly across a known interval and then using each node clock independently. Only elapsed wall-clock times can be measured on the hypercube. Since the host processor is shared with other users, timings reported by the host vary considerably for the same program. Hence, the timings reported in this paper were obtained from the nodes. The execution times that we report were returned from the manager-node.

Other searches were performed using a MIPS M/120 workstation running Unix and an IBM-PC/AT running Xenix. The MIPS M/120 uses a MIPS R2000 CPU running at 16 Mhz. The IBM-PC/AT contained an Intel 80386DX CPU running at 25 Mhz with 64K cache memory. Runs on these machines were timed using the unix times() system call; only user times are reported. These computers ran version 1.4 of the FASTA program package (9). The Smith-Waterman algorithm (2) was implemented in a program called SSEARCH (Pearson, manuscript submitted). FASTA timings for the M/120 and PC/AT386 are for the initial library scan only; scan times were about 90% of the total execution time.

Software

PSCANLIB and PCOMPLIB execute under the UNIX System V/386 Release 3.2 operating system on the Intel iPSC/2 and iPSC/860 hypercubes. The nodes run the NX/2 operating system, which provides message passing services and process management, but no I/O services. Programs are loaded onto the node processors by executing the load() function on the host, and immediately begin execution. Messages can be passed between the host and the nodes or from nodes to node with the csend() and crecv() primitives. Csend() allows a node to send a message to a single node or simultaneously to several nodes, and causes the sending process to wait until the message has been transferred to the DCM. Crecv() allows a node to receive a particular message type from one or more nodes, causing the receiving process to wait until the message has been received.

The PSCANLIB and PCOMPLIB programs are written in the C programming language. PSCANLIB compares a protein or DNA sequence to a data bank or list of data banks. The similarity calculation may be split into two phases so that a faster, less sensitive, first phase can be applied to the entire data bank and a slower, more sensitive, second phase can be applied to the highest scoring sequences from the first pass. FASTA normally uses this strategy for displaying "optimized" scores. PCOMPLIB compares all the sequences in one sequence library with all the sequences in another library. It requires that all the sequences in the shorter library fit into memory on the hypercube. For example, a 4 Mbyte 16-node hypercube would have 15 worker-nodes and more than 45 Mbytes of memory available for the library.

We have implemented two comparison functions within this parallel framework: the FASTA algorithm (7, 9) and the Smith-Waterman function (2). FASTA is a rapid similarity search algorithm that uses multiple phases. The first phase rapidly scans the entire library of protein or DNA sequences and calculates an initial similarity score based on regions with higher densities of sequence identity. The second phase then computes an optimized score for a few of the best sequences discovered in the first phase. This score is calculated by constructing an approximate optimal alignment between the two sequences using a 32-residue band centered on the alignment found in the first phase. FASTA uses the *ktup* parameter to vary the speed and sensitivity of the search. *Ktup=2* and *ktup=1* are used for protein sequences. Searches done with *ktup=1* are more sensitive that those done with *ktup=2*, and take about 5-times as long on a conventional (serial architecture) computer.

The Smith-Waterman algorithm (2) uses a single phase to calculate a single, optimal, similarity score between the query sequence and the library sequence. The method is sensitive, but is also slow. The FASTA algorithm with optimization gives results that are similar to those obtained with the Smith-Waterman algorithm, but runs 20-50 times faster.

Databases

All searches were performed using release 21 of the National Biomedical Foundation Protein Identification Resource (PIR) protein sequence database (3,406,128 residues in 12,476 entries, ref. 12) unless otherwise noted. The order of entries in the library was randomized for library to library comparisons.

IMPLEMENTATION

Similarity searches of DNA and protein sequence databases is a problem that is well-suited to parallel computation. Programs that compare protein or DNA sequences to sequence databases apply a comparison function that calculates a similarity score between the query sequence and each of the sequences in the library. Since the similarity score for each sequence in the library can be calculated independently,

and since the sequences are relatively short (less than 4000 amino acids, and less than 300,000 nucleotides), parallel implementation is straightforward. One simply apportions the library to individual nodes, sends each node the query sequence, starts them running, and collects the results.

This is the strategy that we used in our initial implementation of the FASTA algorithm on the hypercube. A program on the host processor read in the query sequence, the search parameters, and the library file name. The program then divided the sequence library into a number of partitions, distributed a partition to each node, and collected similarity scores. When a node returned a special similarity score that indicated it had exhausted its portion of the library, the host handed out a new partition, until all the partitions had been searched. Identical worker-programs running in parallel on the nodes read in portions of the library, computed similarity scores for each library sequence, and transmitted them back to the host. In this implementation, the worker-nodes generated a substantial amount of message traffic (about 100 bytes for each of the 12,000 sequences in the library). We learned, in retrospect, that communication between nodes and the host is much less efficient that node-node communication, and our initial implementation was very slow. A search of a 3.5 million amino acid library with 229 amino acid query sequence using the FASTA function with *ktup=2* took more than 15 minutes to run on 16 nodes of the hypercube. An IBM-PC with a single 25-Mhz 80386 can perform the same search in 1.7 minutes; a MIPS M/120 or Sun Sparcstation requires about 20 seconds.

To reduce the host-hypercube communications overhead, we introduced a manager-node into the program. By shifting the task of distributing the library partitions and receiving similarity scores from the host to the manager-node, communication time was drastically reduced and the search time was reduced to about 2 minutes. While this was a dramatic improvement, it was still substantially slower than the expected 10-fold speed-up over a single 25 Mhz 80386. Although we had decreased host-node communication inefficiencies by including a manager-node, we were still reading the sequence library off the host file system. As a result, many worker-nodes were simultaneously trying to read library sequences from a single file on the host file system. This created a bottleneck at the host that resulted in all the nodes waiting for the data.

Next we moved the libraries to the Concurrent File System (CFS). Now the nodes were able to access the sequence library concurrently. As a result, execution time decreased to about 25 seconds – a second eight-fold improvement. Buffering messages to reduce message traffic (see below) resulted in an additional reduction in execution time by around 30%. The program now takes 16 seconds to compare a single 229 amino acid sequence against a 3.4 million amino acid sequence library with the FASTA function on 16 nodes of an iPSC/2 hypercube.

The PSCANLIB and PCOMPLIB programs

Both the PSCANLIB and PCOMPLIB programs consist of three parts: a host program running on the host, a manager program running on node 0 of the hypercube, and worker programs running on the remaining nodes in the hypercube (Figure 1). When the program starts, the host acquires a subcube of a particular size and loads the manager program onto node 0 (Figure 2). In PSCANLIB, the host process does little more than load the node programs and read keyboard input. The manager program that is executing on node 0 partitions the library among the worker-nodes and collects the results they return. The host is more active in PCOMPLIB; it reads the first library and sends each of the sequences from that library to the manager-node. In both PSCANLIB and PCOMPLIB, the worker-nodes read the library sequences and perform the similarity calculation.

PSCANLIB differs from PCOMPLIB in the way the query sequences and library sequences are balanced among the parallel nodes and in the role of the host. PSCANLIB compares a single query sequence to a sequence library once, so each worker-node is given the query sequence and a portion of the sequence library to scan. When a worker-node has finished its portion of the library, the manager-node provides another portion, until all the library has been scanned. Since PSCANLIB performs only one complete search of the library, we have attempted to minimize the time between the start of the search and the presentation of the results. Depending on the query sequence and the time required to search different portions of the library, different nodes may scan different portions of the library.

PCOMPLIB compares all the sequences in one library to all the sequences in a second library. PCOMPLIB may compare a thousand sequences to the same library, so the program preprocesses the library and apportions it to the worker-nodes in a preliminary pass, after which the sequences from the second library remain in a worker-node's memory. In PSCANLIB, the manager-node starts the worker-nodes by giving them the query sequence and then distributes partitions of the library as worker-nodes become available to process them. In PCOMPLIB, the manager-node divides the search library into as many portions as there are worker-nodes, and then sends each worker a portion of the search library. This portion remains unchanging for the lifetime of the worker. The manager-node then receives a query sequence from the host, sends the query sequence to each worker-node, waits for each worker to finish, sorts the results of the search, and waits for the next query from the host. Thus, with PSCANLIB, the query sequence remains constant in each worker-node and the library sequences vary, while in PCOMPLIB, the (second) library sequences remain constant and the query sequences (from the query library) vary. PCOMPLIB also has an option (`-i`) for comparing a sequence library to itself. In this case, only $\frac{n \times (n-1)}{2}$ similarity scores are calculated, and the comparison takes half as long* (Table III).

In writing PSCANLIB and PCOMPLIB, we have attempted to separate the general library-searching-program code from the code that is specific to a particular similarity function. The four subroutines that must be tailored to a similarity function – two in the host program and two in the worker program – are highlighted in Figure 2. The host program collects the name of the query sequence and library file. Then, using a similarity-function-specific part of the program, the host program prompts for any

*A straightforward implementation of the "-i" option would calculate only the upper triangular entries of the matrix of comparison scores of ordered pairs of sequences; that is, compare sequence i against sequence j **iff** $i \geq j$. The entries below the diagonal can be inferred from the symmetric entries above the diagonal. This simple method does not work well because there are severe load imbalances in the latter stages of the algorithm. (If there are N worker-nodes, the j-th worker-node will be idle after j/N of the library has been processed.) Instead, we calculate half of the entries above the diagonal such that they form a checkerboard pattern; below the diagonal is the complementary pattern. Specifically, compare sequence i against sequence j when either $i+j$ is even and $i \geq j$ or when $i+j$ is odd and $i < j$. With this approach, each worker-node uses only half of its portion of the library.

additional search parameters that may be required, such as the *ktup* parameter used by FASTA. (The Smith-Waterman algorithm does not require any additional parameters.) The parameters are passed down through the manager-node directly to the worker-routine conducting the similarity search. The definition of the parameter structure can be changed to suit the needs of the analysis method. One defines a parameter structure for a particular analysis method, reads in the parameters with the host program, and the parameters are passed automatically to the analysis routine that uses them. The worker-nodes may also perform some method-specific initialization. For FASTA, this includes allocating several data structures and calculating the lookup table from the query sequence; for the Smith-Waterman algorithm, deletion penalties are pre-calculated. The worker-nodes then perform the specific similarity calculation.

As shown in Figure 2, the PSCANLIB manager program can accommodate analysis functions that calculate similarity scores in several stages. For example, FASTA calculates two scores, *init1* and *initn*, for every sequence in the library in the first stage. FASTA calculates a third score (*opt*), in the second stage, for those library sequences with the highest *initn* scores. In the first stage, the manager-node partitions all the sequence library. After the first stage is complete, the highest scoring library sequences are identified and individual sequences are partitioned to the nodes for the second stage calculation. The second phase is usually slower, but more sensitive, than the first. Since the second phase examines only a small percentage of the sequences examined by the first phase, the additional cost is small. We have implemented the two stages that are used by FASTA: a first stage that identifies regions high densities of identities without gaps and a second stage that performs a limited Smith-Waterman alignment around the diagonal that contains the best initial score. It would be straightforward to use a complete Smith-Waterman calculation for the second similarity calculation. The Smith-Waterman implementation calculates a single similarity score in a single stage. Adding additional phases is simple: additional initialization and evaluation functions must be provided. The program only calculates pairwise similarity scores, it does not display the implied alignment of the two sequences.

Copies of PSCANLIB and PCOMPLIB are available from William R. Pearson (electronic mail: wrp@virginia.EDU).

RESULTS

PSCANLIB

We have implemented two general programs that can be used for searching libraries of DNA and protein sequences on an Intel hypercube computer – PSCANLIB and PCOMPLIB. Execution times for the PSCANLIB program on both an iPSC/2 386-based machine and an iPSC/860 hypercube are shown in Table I. Searches were done against the PIR protein sequence database using 16-nodes of the iPSC/2 and iPSC/860. Timings are also shown for a MIPS M/120 computer (a workstation-class computer that is about as fast as a Sun Sparcstation) and an IBM-PC/AT with a 25 Mhz Intel 386. A 16-node iPSC/2 (15 worker Intel 386's running at 16 Mhz) performs a little better than a MIPS M/120 (a single R2000 processor running at 16 MHz) on short sequences for FASTA with *ktup=2*. However, as the computation becomes more time-consuming (either by increasing the length of the query sequence or by using *ktup=1*), the performance of the iPSC/2 increases relative to the MIPS/120. This suggests that the fixed overhead for scanning the sequence library is somewhat larger on the iPSC/2 than on the MIPS. The difference in performance between the 16-node iPSC/2 and a single 25 MHz Intel 386 ranges from a factor of 5 for the fastest scan to a factor of 10 for longer sequences or slower comparison functions. A factor of 10 is exactly the speed-up expected based on the relative clock speeds of the two machines – 15 worker-nodes running at 16 Mhz would appear to be a 240 Mhz machine in the ideal case. Thus, our final implementation of PSCANLIB makes very efficient use of the hypercube nodes with long query sequences or slow comparison functions.

The most dramatic performance was obtained on the 32-node iPSC/860. This machine is so fast that only about 0.9 of the 3.3 seconds required to compare a 229 residue sequence to a 12,476 entry library (*ktup=2*) was used to perform the sequence comparison, the rest of the time was spent reading the sequence library. With FASTA and *ktup=1*, a 16-node iPSC/860 performs about 9-times faster than the MIPS M/120 and 5-times faster than the iPSC/2. However, we note that the iPSC/860 performs about 12-fold faster than the iPSC/2 with the Smith-Waterman algorithm. The reason for this difference is

unclear, the iPSC/2 also performs slower than expected on the Smith-Waterman algorithm when compared with the MIPS M/120. This anomaly may be due to the architecture of the iPSC/2.

An important consideration in mapping an algorithm to a parallel machine is to allocate data and control in such a way that one takes maximum advantage of the underlying machine architecture. In a message-passing system, interaction between processors should be maintained at a level in keeping with the communication characteristics of the machine. However, we note that the optimizations described below are required only for a rapid search algorithm such as FASTA. With a slow, computationally intensive algorithm like Smith-Waterman, the communication time is usually small compared to the time spent in computation.

We have explored the relationship between computational complexity and parallel performance to evaluate our programs. The computational complexities of FASTA and the Smith-Waterman are both $O(n^2)$ – doubling the query sequence length or the library size doubles the amount of time required to perform the search. In addition, FASTA has a search parameter, the *ktup*, parameter, that can be used to vary the speed and sensitivity of the search. As noted above, protein library searches with *ktup=1* are more sensitive but take about 5-times as long as searches with *ktup=2*. The results presented in Tables I and II and in Figs. 3 and 4 show the interaction between computational complexity, either due to query sequence length or comparison function, and parallel performance.

Relative performance on parallel processors can be viewed several ways. A simple criterion is execution speed – how long does a library search take. Table II shows the effect of running the PSCANLIB program on additional iPSC/2 or iPSC/860 hypercube-nodes; the program runs faster when more worker-nodes are used. While Table II shows that additional worker-nodes help, it is not clear how efficiently the additional nodes are used. This efficiency can be estimated by measuring the relative speed-up of a search as additional processors are used. Ideally, a search would be finished 4-times faster with 16-processors than with 4-processors. Parallel programs rarely behave this well. Usually there are overhead costs, such as contention for communications links between the parallel processors, that prevent the sixteenth proces-

sor from being used as efficiently as the fourth one.

Since PSCANLIB uses one hypercube-node to manage the others, a sixteen-node hypercube (fifteen worker-nodes) should search a sequence database 15-times faster than a two-node cube, and 5-times faster than a four-node cube. Figure 3 plots the data from Table II based on this assumption. Figure 3 shows that PSCANLIB makes the most efficient use of additional hypercube-nodes only when the comparison function is not too fast. When a 229 residue sequence is compared to a 3.4 million residue library on either the iPSC/2 or iPSC/860 with the FASTA function (*ktup=2*), the sixteenth node is used about 60 - 70% as efficiently as the second node. Indeed, a 32-node iPSC/860 takes only 0.5 second less time to search the library than a 16-node machine. These inefficiencies gradually disappear as the comparison function becomes slower (FASTA with *ktup=1* or Smith-Waterman) or the query sequence becomes longer. Thus, on the iPSC/2 the sixteenth node works at 92% of the second node's efficiency with FASTA, *ktup=1*, and at 99% efficiency with Smith-Waterman. On the iPSC/860, high efficiency is not seen until a long query sequence (RNBY3L, 1460 aa) and a slower comparison function (FASTA, *ktup=1*) are used, but then the efficiency on the 32nd node is about 95% that of the second node.

The two hypercubes are not perfectly efficient for rapid comparison functions because of delays in reading the sequence from the library. The time required to read the library is fixed by the speed of the CFS, additional worker-nodes cannot decrease it. To examine the overhead required to read the sequence library and to send the messages that report the similarity scores to the manager-node, we determined the time required to read the protein sequence library on the iPSC/2 hypercube by running the PSCANLIB program without a similarity function and without sending messages to the manager-node. Eight seconds were required to read the sequence library without any messages to the manager-node on a 16-node iPSC/2; an additional second was required to send the manager-node a message reporting a score of 0 for each sequence in the library. A similar estimate for overhead (13 seconds) can be calculated by extrapolating the data Table I to a zero length query sequence. Thus, the hypercube is not compute-bound when using a rapid comparison algorithm like FASTA. The speed of the concurrent file system is the major limiting factor; communications from the worker-nodes to the manager-node contributes about 10% of

the overhead. When the comparison becomes more time-consuming, either because of a slower function or a longer query sequence, this overhead becomes less significant. Thus, additional worker-nodes on a larger hypercube would be efficiently used to decrease the time required for the Smith-Waterman function.

Parallel performance on a hypercube architecture can also be measured by collecting timing statistics in each node of the hypercube. It is possible that some inefficiencies would be distributed among the nodes in such a way that additional nodes would decrease the execution time, while each of the nodes was idle for a substantial portion of the time. For example, a results message that consists of one or more similarity scores is returned to the manager-node from the worker for each sequence in the sequence library. The volume of messages generated could cause the manager-node to become a bottleneck because everyone is trying to send to and receive from it. While the DCM on each node reduces contention by handling message passing on its own and thus freeing the main processor for computation, the number of messages generated may become so large that there is blocking and wasted time. We addressed this problem by buffering the results messages sent from the worker-nodes to the manager-node. There are two classes of inter-node messages on the iPSC/2; there is a substantial overhead cost for sending a message that is longer than 100 bytes. As a result, the time required to send messages between 100 and 2500 bytes is constant. Thus, instead of sending a message for every library sequence, the worker stores the scores and transmits them in batches of 25. Reduced message traffic implies less contention, blocking, and wasted time, and usually we have been able to maintain idle time below 2% of the elapsed time of the worker process. Buffering the results messages in groups of 25 reduced execution time by about 30%. (All the times reported were taken from programs that used buffered-message-passing.)

PCOMPLIB

Comparison of all the sequences in one library to all the sequences in a second library is a more suitable problem for parallel processors. While FASTA can compare a single query sequence to a

sequence database in less than a minute on a high performance workstation, comparison of two 6,000 - 14,000 entry libraries takes several days. With PSCANLIB, more than 50% of the execution time is spent reading the sequence library when the query sequence is shorter than 300 residues and the FASTA comparison function is used. With PCOMPLIB, the library is only read from the disk once, after which it is stored in the memory of the hypercube-nodes and searched hundreds of times. Thus, once a library is read into memory on a node, it can be used for the similarity calculation without any additional computation, and the overhead of reading in a large sequence library is spread over a large number of library scans.

The performance of the PCOMPLIB program on the iPSC/860 is summarized in Table III; additional timings on the iPSC/2 are shown in Figure 4 and Table IV. A measure of the parallel efficiency of the PCOMPLIB program is shown in Figure 4. The expected 2-fold speed-up can be seen when the -i option is used for library self-comparison.

PCOMPLIB (Figure 4) is much more efficient with a rapid comparison function (FASTA, *ktup=2*) than PSCANLIB is. While PSCANLIB is able to use only about 25% of the 31st worker-node (*ktup=2*, Figure 3), PCOMPLIB uses 75% this node when searching a 3,000 entry library, and more than 85% of the node when searching 12,476 sequences (Figure 4). This improved performance is expected; PCOMPLIB amortizes the cost of reading the sequence library over hundreds or thousands of sequence searches. The improved efficiency of PCOMPLIB confirms that the relatively poor performance of PSCANLIB on a 32-node iPSC/860 reflects the overhead required to read the sequence library. Calculations from the timings Table III suggest that if all of the sequences were in memory, PSCANLIB would only require about 0.6 seconds to compare a 229 amino acid sequence to the PIR protein sequence library on a 32-node iPSC/860 hypercube.

The difference in the efficiency for scanning a 3,000 entry library versus a 12,476 entry library (74% vs 88%) suggests that the time required to read the next sequence from the host can be limiting when the library on the worker-nodes is short and the comparison function is fast. Normally PCOMPLIB

uses the host processor to read the query sequence library. In our early implementation of FASTA on the iPSC/2 hypercube, we found that host-node communications can be slow, so we also modified PCOMPLIB to use the manager-node to read the query library. This modification did not produce any measurable improvement in performance or parallel efficiency, perhaps because the increased efficiency of having the manager-node read the library was offset by additional delays in communicating the results from the worker-nodes to the manager-node.

More sophisticated logic may improve the performance of PCOMPLIB. The PCOMPLIB manager-node waits for all the worker-nodes to finish a search before moving on to the next query library sequence. Since the all the library has been distributed to the worker-nodes before the first PCOMPLIB search starts, a slow search on single worker-node will cause the other nodes to be idle. This is more likely to be a problem with the FASTA function, since its speed is a function of the sequence context (FASTA takes longer to compare related sequences than to compare unrelated ones). The delay can be reduced by randomizing the second library among the worker-nodes (the PIR database groups related sequences), and the problem is less likely to occur with a 12,000 entry library than with a 3,000 entry library. The timings that we report were determined with randomized libraries. Buffering the results from the worker may also help; worker-nodes would then start searching with the next query sequence as soon as they are finished with the previous one.* Finally, the manager-node is responsible for sorting all the results after they are sent back from the worker-nodes. Currently, the manager-node waits until all the results are in before starting the sorting procedure. This minimizes the competition between the worker-nodes for communications paths to the manager-node, but as a result, there is a delay before the next search starts. It is possible that parallel sorting on the worker-nodes, followed by a merge of the sorted list on the manager-node, would be more efficient. Nevertheless, only modest improvements can be

*We note that simply buffering the results from the worker-nodes, without randomizing the library, would simply delay the problem when databases with contiguous related entries were compared. If all the cytochrome c entries were on one node, the first cytochrome c sequence would wait on that node, and then the second cytochrome c sequence would wait, and so on, until all the buffers were used.

expected. PCOMPLIB is very fast; on the 32-node iPSC/860 it can perform 112 × 12,476 sequence comparisons in 1.27 min.

DISCUSSION

Biological sequence comparison appears to be a problem that is tailor-made for parallel computers. A comparison function is independently applied to each of 12,000 to 40,000 sequences in a database, the results are returned, and the highest scoring sequences are displayed. Thus, we were surprised when our initial implementation of FASTA on the Intel iPSC/2 hypercube was slower than running the same program on a 80386 IBM/PC. The poor performance of the hypercube was in large part due to the high performance of the FASTA algorithm. Had we started with the Smith-Waterman algorithm, which is about 50-100 times slower than FASTA, the benefits of a parallel implementation would have been apparent immediately. Past parallel implementations of the Smith-Waterman algorithm have used fine-grained parallelism on single-instruction multiple-data (SIMD) computers like the Connection Machine (8) and the ICL Distributed Array Processor (DAP) (13, 14), although an implementation of the dynamic programming algorithm on hypercubes and shared memory multiple-instruction multiple-data architectures has been described (15). While the dynamic programming algorithm maps well onto massively parallel SIMD machines, the implementation of less regular algorithms, such as FASTA, is considerably more difficult. FASTA is so fast that it is impractical to use a fine-grained approach on a message-passing system such as the hypercube. The large number of individual sequences in a DNA or protein database can be accommodated naturally by the coarse-grained approach that we describe.

The performance of several serial and parallel architectures is summarized in Table IV. While the fastest comparison times are found with parallel processors, the performance of today's RISC architectures is comparable to that obtained on the DAP and Cray-1 several years ago. The serial architectures have the advantage that they are readily available in inexpensive, high performance, machines. In addition, on some parallel machines, the systems software, development tools, and hardware reliability remind one more of computing in the 1960s than the 1990s. Because of their accessibility, serial processors

are likely to remain the machines of choice for single database searches. Nevertheless, parallel machines are necessary for large inter-database comparisons.

The performance of the FASTA algorithm on the hypercube has improved 16-fold in the course of our implementation. By removing almost all communication with the host processor, placing the sequence libraries on the concurrent file system, and buffering communication between the worker-nodes and the manager-node, node idle time has been reduced to less than 10% of elapsed time for longer sequences or slower comparison functions. Despite this high efficiency, the 16-node iPSC/2 hypercube performs only slightly better than a MIPS M/120 workstation. The benefits of parallel computation are more dramatic when the same programs are run on the third generation Intel iPSC/860. The data in Tables I and II suggest that a single node on the /860 is almost as fast as the MIPS M/120 – a Smith-Waterman search that required 31 min on the M/120 required 37 min on a one-worker-node iPSC/860 and 2.5 min on a 16-node iPSC/860. Improvements in the iPSC/860 compiler may be able to increase this machines performance by 30 - 50%. Because of the dramatically greater speed on the iPSC/860, communications overhead becomes a serious cost, even on compute-intensive functions, and our fine-tuning has a more substantial payoff.

We have designed a general, efficient, and flexible platform that can easily be adapted to evaluate a variety of biological sequence comparison algorithms. These programs display close to optimal performance characteristics. Since the platform has been implemented with coarse-grained parallelism on a hypercube architecture machine, little effort should be required to implement new comparison algorithms. The general framework our programs may also be useful in implementing versions for new machines, as well as for machines with similar architectural characteristics. For example, a variant of the PCOMPLIB program might be easily implemented on a large network of high performance workstations. For library to library comparisons, a Sun/3 network would be expected to perform somewhat better than the iPSC/2 hypercube of equal size, and a network of RISC-based processors (Sun Sparcstations or Digital Equipment Decstations) should be about as fast as the iPSC/860.

ACKNOWLEDGEMENTS

This research was supported by a grant from the National Library of Medicine (LM04969). Time on the Intel iPSC/2 was provided by the U. of Virginia Institute for Parallel Computation (JPL957721). Access to the Intel iPSC/860 was provided through a cooperative arrangement between the U. of Virginia Institute for Parallel Computation and the Institute for Computer Applications in Science and Engineering (ICASE).

References

1. Needleman, S. and Wunsch, C. (1970) A general method applicable to the search for similarities in the amino acid sequences of two proteins. *J. Mol. Biol.*, **48**,444-453.

2. Smith, T. F. and Waterman, M. S. (1981) Identification of common molecular subsequences. *J. Mol. Biol.*, **147**,195-197.

3. Sankoff, D. and Kruskal, J. B. (1983) *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*,Addison-Wesley

4. Doolittle, R. F., Feng, D. F., Johnson, M. S., and McClure, M. A. (1986) Relationships of human protein sequences to those of other organisms. *Cold Spring Harb. Symp. Quant. Biol.*, **51**,447-455.

5. Wilbur, W. J. and Lipman, D. J. (1983) *Proc. Natl. Acad. Sci. USA*, **80**,726-730.

6. Lipman, D. J. and Pearson, W. R. (1985) Rapid and Sensitive Protein Similarity Searches. *Science*, **227**,1435-1441.

7. Pearson, W. R. and Lipman, D. J. (1988) Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci. USA*, **85**,2444-2448.

8. Lander, E., Mesirov, J. P., and Taylor, W. (1989) Study of protein sequence comparison metrics on the Connection Machine CM-2. *J. of Supercomputing*, **3**,255-269.

9. Pearson, W. R. (1990) Rapid and sensitive sequence comparison with FASTP and FASTA. In Doolittle, R. F., ed. *Meth. Enz.*, vol. 183. (New York: Academic Press), pp. 63-98.

10. Arlauskas, R. (1988) iPSC/2 system: a second generation hypercube. *Comm. ACM*, **31**,38-42.

11. Close, P. (1988) The iPSC/2 node architecture. *Proc. 3rd Conf. on Hypercube Concurrent Computers and Applications*, , pp. 43-50.

12. Barker, W. C., George, D. G., and Hunt, L. T. (1990) Protein Sequence Database. In Doolittle, R. F., ed. *Meth. Enz.*, vol. 183. (New York: Academic Press), pp. 31-49.

13. Coulson, A. F. W., Collins, J. F., and Lyall, A. (1987) Protein and nucleic acid sequence database searching: a suitable case for parallel processing. *Computer J.*, **30**,420-424.

14. Collins, J. F., Coulson, A. F. W., and Lyall, A. (1988) The significance of protein sequence similarities. *Comp. Appl. Biosci.*, 4,67-71.

15. Carriero, N. and Gelernter, D. (1989) Linda in context. *Comm. ACM*, **32**,444-458.

Figure Legends

Fig. 1. The Intel iPSC/2 hypercube configuration used by PSCANLIB and PCOMPLIB – Each box (host, manager, worker) denotes a 16 MHz Intel 386 with 4 Mbytes of memory. The Concurrent File System box denotes a set of four 16 MHz Intel 386 nodes connected to a set of interleaved disk drives. The dashed line indicates the relatively slow communications boundary between the host and the hypercube nodes. Lines with arrows show the communications pathways used by the programs. Any of the nodes can communicate with each other on the hypercube; PSCANLIB and PCOMPLIB use a hierarchical structure.

Fig. 2. The structure of the PSCANLIB program – The three programs used by PSCANLIB – host, manager, and worker – are outlined. Those portions of the program that can be changed for a new similarity function are underlined.

Fig. 3. Performance of PSCANLIB on different size hypercubes – A 3.4 million amino acid library was scanned using bovine prolactin (LCBO, 229 amino acids) or yeast RNA polymerase (RNBY3L, 1460 amino acids) using an iPSC/2 (A) or an iPSC/860 hypercube (B). Let T_n be the total execution time when there are n worker-nodes. We define the *relative efficiency* of a program to be $E_{a,b}=\frac{(a\times T_a)}{(b\times T_b)}$ where $a<b$. We expect $E_{a,b}$ to be close to 1.0 if the program scales well from a worker-nodes up to b worker-nodes. The relative efficiencies are plotted; in all cases the value shown is $E_{1,b}$, for various b, except for the Smith-Waterman comparison of the LCBO sequence on the iPSC/2, where $E_{3,b}$ is shown. The execution times are shown in Table II. (Δ) LCBO using FASTA *(ktup=2)*; (O) LCBO, FASTA, *ktup=1*; (□) LCBO, Smith-Waterman. (Δ) RNBY3L, FASTA, *(ktup=2)*; (O) RNBY3L, FASTA, *ktup=1*; (□) RNBY3L, Smith-Waterman.

Fig. 4. Performance of PCOMPLIB on different size hypercubes – Relative efficiency was calculated as in Fig. 3. PCOMPLIB timings were determined by comparing a library of 112 entries with a library of 3,000 entries (solid lines), or a 112 entry library with a 12,476 entry library (dashed line). Comparisons with FASTA, *ktup=2* (Δ, iPSC/2, Δ, iPSC/860), FASTA, *ktup=1*, (O, iPSC/2, O, iPSC/860), and the

Smith-Waterman algorithm (□, iPSC/860) are shown. $E_{7,b}$ is plotted for the 12,476 entry library; the other values are $E_{3,b}$.

Table I. PSCANLIB execution times

PIR Entry	Description	Length	iPSC/2	iPSC/860	M/120	AT/386
		FASTA *ktup=2*				
CCOS	Ostrich Cytochrome c	104	0:14	0:03.1	0:16	1:15
LCBO	Bovine Prolactin	229	0:16	0:03.3	0:19	1:37
A27366	Rat AMP deaminase	747	0:22	0:04.4	0:30	2:58
RNBY3L	Yeast RNA pol III	1460	0:32	0:06.0	0:48	5:00
		FASTA, *ktup=1*				
CCOS	Ostrich Cytochrome c	104	0:29	0:05.1	0:44	4:18
LCBO	Bovine Prolactin	229	0:53	0:08.1	1:21	8:02
A27366	Rat AMP deaminase	747	2:07	0:22.0	3:44	23:05
RNBY3L	Yeast RNA pol III	1460	4:07	0:45.5	7:44	46:21
		Smith-Waterman				
CCOS	Ostrich Cytochrome c	104	15:19	1:10.5	14:15	–
LCBO	Bovine Prolactin	229	33:36	2:32.9	31:11	–
A27366	Rat AMP deaminase	747	109:11	8:21.5	101:43	–
RNBY3L	Yeast RNA pol III	1460	213:32	16:21.8	197:53	–

Timings (in min:sec) are reported for searches against a 12,476 sequence, 3,406,128 amino acid library (PIR release 21). Sixteen nodes were used on the iPSC/2 and iPSC/860. – : not determined.

Table II. PSCANLIB execution time vs hypercube size

PIR Entry	Function	Number of worker nodes				
		1	3	7	15	31
iPSC/2						
LCBO (229 aa)	FASTA (*ktup=2*)	2:42	0:57	0:26	0:16	–
	(*ktup=1*)	11:06	3:44	1:39	0:48	–
	Smith-Waterman	–	166:31	71:38	33:36	–
RNBY3L (1490 aa)	FASTA (*ktup=2*)	6:56	2:23	1:04	0:32	–
	(*ktup=1*)	60:31	20:14	8:45	4:07	–
iPSC/860						
LCBO (229 aa)	FASTA (*ktup=2*)	0:26.3	0:09.0	0:04.8	0:03.3	0:02.6
	(*ktup=1*)	1:40.4	0:33.5	0:15.4	0:08.1	0:05.0
	Smith-Waterman	37:21.1	12:32.2	5:24.0	2:32.9	1:15.4
RNBY3L (1490 aa)	FASTA (*ktup=2*)	1:06.8	0:23.0	0:10.8	0:06.0	0:03.9
	(*ktup=1*)	10:53.0	3:38.5	1:34.7	0:45.5	0:23.2
	Smith-Waterman	241:47.4	81:03.2	34:52.9	16:21.8	7:59.1

Timings (in min:sec) are reported for searches against a 12,476 entry, 3,406,128 amino acid, library (PIR release 21). – : not determined.

* Timings on the iPSC/860 should be considered preliminary.

Table III. PCOMPLIB execution times

Library size First	Library size Second	Function	Time (min:sec)
112 (32476 aa)	3,000 (804049 aa)	FASTA (*ktup=2*)	0:22
		FASTA (*ktup=1*)	2:00
		Smith-Waterman	35:04
112	12,476	FASTA (*ktup=2*)	1:16
3,000	3,000 (-i)	FASTA (*ktup=2*)	9:21 5:20
		Smith-Waterman	*15:38:00*[*]

Times are presented for a 32 node iPSC/860 hypercube, and do not include the time required to write the results (about 2 min for a 3,000 vs 3,000 comparison).

[*] Value was calculated from the 112 X 3,000 comparison time.

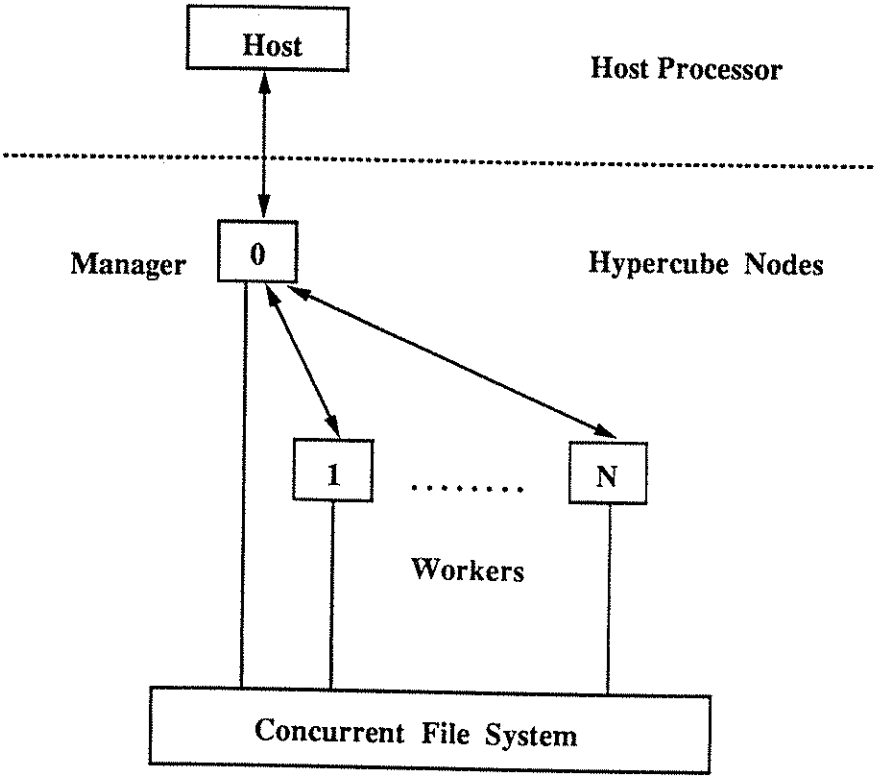
Table IV. Library comparisons on parallel computers

Machine	Size	Library sizes	Function	Time (hr:min:sec)	Matrix Entries per Second (millions)
iPSC/2	16 nodes	112 X 3,000	FASTA (*ktup=2*)	0:04:09	105*
iPSC/2	16 nodes	112 X 3,000	FASTA (*ktup=1*)	0:23:06	19*
iPSC/860	32 nodes	112 X 12,476	FASTA (*ktup=2*)	0:01:16	1,475*
iPSC/860	32 nodes	3,000[2]	FASTA (*ktup=2*)	0:09:00	1,206*
MIPS M/120	16 Mhz	1 X 12,500	FASTA (*ktup=2*)	0:00:16	49*
iPSC/860	32 nodes	112 X 3,000	Smith-Waterman	0:35:00	12
CM-2	32,000 proc	2,350[2]	Smith-Waterman	2:15:00	65†
Cray X-MP			Smith-Waterman		1.1†
DAP	64 X 64		Smith-Waterman		0.6§
Cray-1			Smith-Waterman		0.4†
MIPS M/120	16 Mhz	1 X 12,500	Smith-Waterman	0:31:11	0.4

*This values are only approximately comparable to those determined with the Smith-Waterman algorithm, as the matrix entries are compared in a very different fashion.

†Ref. 8.

§Coulson et al. (13) report that 1.8 DAP second per residue in the query sequence is required to scan a one million amino acid library.



Deshpande et al. Fig 1.

HOST PROGRAM

```
load manager;
get options;
get query sequence;
get library name;
get other parameters;
send query sequence to manager;
send library name and parameters;
wait for manager to complete;
exit;
```

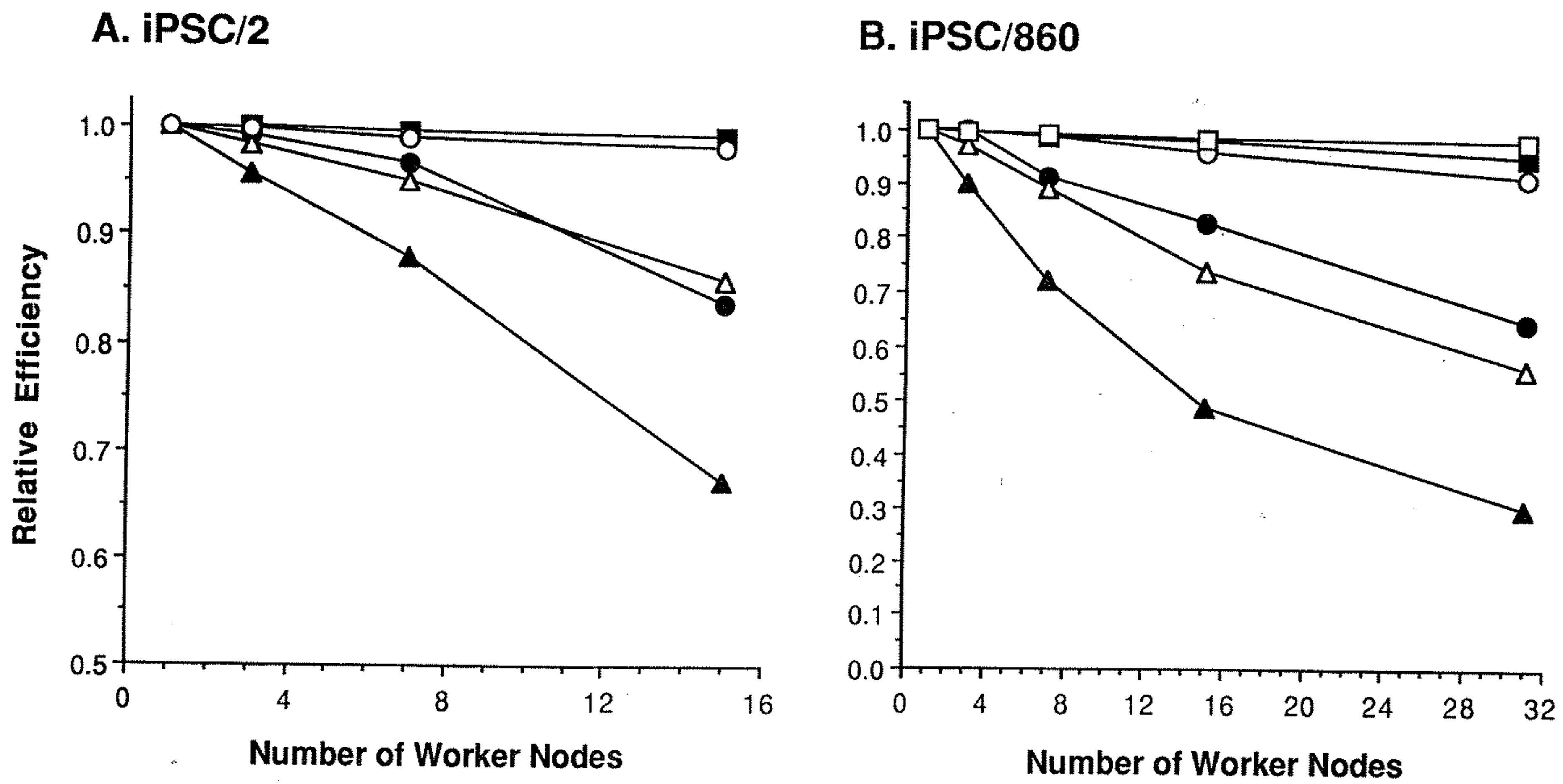
MANAGER PROGRAM

```
receive query sequence;
receive library name and parameters;
send query sequence to workers;
send parameters to workers;
for stages 1 ... N
    while (there is library to search) {
        send partition bounds to workers;
        collect similarity scores;
        }
send stop signal to workers;
sort similarity scores;
report similarity scores;
exit;
```

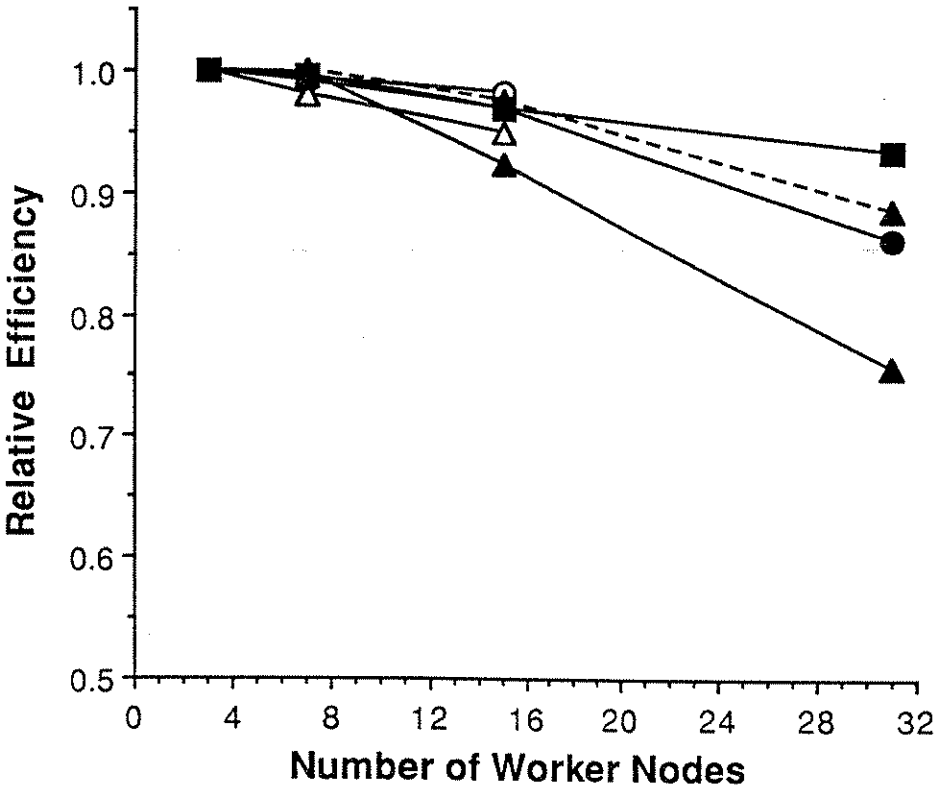
WORKER PROGRAM

```
receive query sequence from manager;
receive parameters from manager;
perform required initializations;
while (manager does not send stop signal) {
    get partition bounds from manager;
    for each sequence in the library partition {
        fetch a library sequence;
        compute similarity scores;
        }
    }
exit;
```

Deshpande, et al. Fig 2



Deshpande et al Fig 3



Deshpande et al