SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 25(3), 291–304 (MARCH 1995)

Coupling Hundreds of Workstations for Parallel Molecular Sequence Analysis

volker strumpen*
Institut für Informatik, Universität Zürich, CH-8057 Zürich, Switzerland

SUMMARY

We present a highly scalable approach to distributed parallel computing on workstations in the Internet which provides significant speed-up to molecular biology sequence analysis. Recent developments show that smaller numbers of workstations connected via a local area network can be used efficiently for parallel computing. This work emphasizes scalability with respect to the number of workstations employed. We show that a massively parallel approach using several hundred workstations, dispersed over all continents, can successfully be applied for solving problems with low requirements on communication bandwidth. We calculated the optimal local alignment scores between a single genetic sequence and all sequences of a genetic sequence database using the ssearch **code that is well known among molecular biologists. In a heterogeneous network with more than 800 workstations this job terminated after several minutes, in contrast to several days it would have taken on a single machine.**

key words: parallel computing; distributed systems; molecular biology sequence analysis; UNIX; workstations

INTRODUCTION

Recent developments in distributed parallel computing are aiming at the use of inexpensive computing resources for parallel computations as are available in heterogeneous networks of workstations. Systems such as *Linda*[1] or *PVM*[2] are widely used, especially for problems of scientific computing. Typically, up to ten machines, and in more computing and less communication intensive applications some tens of machines, are employed within local area networks. Experiments in heterogeneous environments, spanning several LANs have also been reported.[3] We developed a prototypical software platform for distributed massively parallel computing to support the use of a virtually unlimited number of machines connected via the Internet. Such a system is suited for parallel applications, where the computational expense within a LAN clearly outweighs the interprocessor communication across LAN borders, because the latency of Internet connections is considerable.

Compared to conventional supercomputers, powerful workstations offer less floating point performance but almost comparable integer performance. Recent performance

* Current address: Department Informatik, Institut für Wissenschaftliches Rechnen, ETH Zentrum, CH-8092 Zürich, Switzerland, email: strumpen@inf.ethz.ch.

CCC 0038–0644/95/030291–14
© 1995 by John Wiley & Sons, Ltd.

Received 19 July 1993
Revised 20 October 1994

evaluations of molecular biology sequence analysis which do not require floating point but integer computations showed that the blastpm code for sequence analysis runs just 20 per cent slower on a Sun4 (SPARCstation 330) compared to an unoptimized single processor run on a Cray Y-MP 8/864.[4] The Y-MP processor delivered only approximately 40 MIPS, which coincides with the peak performance of a Sun SPARCstation2.

In this paper, we show the possibility and describe how we employed a distributed workstation network of more than 800 machines, geographically scattered over all continents, to accelerate a computational problem that fits the capabilities of this system's architecture. The computing challenge of molecular biology sequence analysis, one of the key techniques of the Human Genome projects,[4,5] is accelerated from run-times of a couple of days on a single workstation or Cray processor to some minutes with our platform.

Today's DNA and protein databases are growing extremely fast and seem to exceed any computational power for similarity searches already in near future. A distributed system may be a solution to this problem, where huge distributed databases are geographically dispersed and accessible via electronic networks. Since the rapidly growing size of these databases prohibits frequent transfers via networks, the computing power has to be provided at the geographical site of the database. The structure of our system is therefore a network of globally distributed local area networks, each with access to a local database.

MOLECULAR SEQUENCE ANALYSIS

The biological challenge rises from the fact that DNA and protein molecules are the primary genetic materials that encode information necessary to understand life. The DNA consists of sequences of bases, and protein molecules are built from amino acids. The amino acids are encoded in sequences of three bases in the DNA. The function of a protein is determined by the structure in which the amino acids fold into a three-dimensional macromolecule. In order to understand this complex folding structure, the folding process would have to be simulated. However, Sander *et al.*[5] estimate that such simulations need 12 hours of CPU time on a computer with one million teraflops. Therefore, molecular biologists use less complex sequence analysis methods to understand the functionality of a new genetic sequence of a DNA or protein polymer by finding a DNA or protein with similar sequence structure and known functionality. At present, this approach is the only well recognized way to understand the genetic code.

Modern laboratory techniques provide fast and cheap means of analysing the sequence of bases in DNA and the sequence of amino acids in protein molecules. There exist databases of extracted DNA and amino acid sequences containing about 100,000 sequences that can be searched for similarity. In our experiments we used the NCBI-GenBank (version 75.0)[6] that contains a total of 106,684 genetic sequences comprising 126,212,259 bases. Owing to the growing size of the databases even the sequence analysis algorithms pose serious computational problems. In the case of the search for a human chromosome, containing 5000 sequences, against a database of 3,000,000 sequences, Bork[7] estimates that a powerful workstation would take 5000 days even with the tfasta approximation algorithm.

The similarity search can be conducted using various techniques that are still subject to active research. The computational aspects can be captured by regarding the sequences of bases of DNA or amino acids of a protein as strings over an alphabet. Wagner and Fischer[8] developed an algorithm for string correction in 1973 that has been adapted to the needs of biologists by Smith and Waterman[9] as well as Needleman and Wunsch[10]. These algorithms compute similarity scores for measuring the similarity between two sequences with respect to an optimal local alignment metric.[11,12] Owing to the computational complexity of this approach, faster but heuristic algorithms, implemented for example in the blast, fasta, or tfasta programs, can be used. The algorithm we used for obtaining the presented results on parallel alignment score computations is the rigorous optimal local alignment algorithm of Smith and Waterman. This algorithm is implemented in the ssearch program which is included in the popular fasta program package.[12] We determined those 50 genetic sequences of the NCBI-GenBank which match a given query sequence best by computing and sorting the scores between the query sequence and all sequences of the gene bank. We chose two query sequences consisting of 105 and 773 bases.

PARALLEL INTERNET COMPUTING

Based on our experience with The Parform,[13] we extended our work on parallel computing from local area networks to world-wide area networks. Features such as configuring a large process topology across LAN borders and fault tolerance are not available with other systems. Therefore, we designed a new networking platform that incorporates these features. A system for Internet-wide parallel computations has to fulfil the following requirements: (1) portability, which is indispensable in order to collect an appropriate number of machines, (2) non-intrusion with respect to other users in the network by using only idling CPU shares and not congesting the network, and (3) efficiency, which can only be the secondary goal of our efforts with respect to the interactive users of the workstations employed.

The existing infrastructure of the Internet and the increasing availability of powerful workstations running a UNIX operating system are the basis for this work. The *de facto* standard for open system interconnection, the TCP/IP Internet Protocol Suite and the Berkeley Socket Interface[14,15] provide the tools to implement a parallel computing platform spanning LANs in the Internet. Another property of the Internet is the commonly installed client–server structure of individual LANs. Although there exist many administrative differences, features such as network file systems or access rights are in principle uniformly available. In contrast to the variety of computer architectures of earlier days, modern workstations also simplify the implementation of low level communication, because they use identical data representations.

The guiding principle for the design of our system architecture and its implementation has been simplicity. The system therefore provides only fundamental features: (1) remote process start-up and termination, (2) asynchronous interprocess communication, (3) extensive logging and error handling, (4) fault tolerance and (5) load balancing. To achieve portability, all mechanisms are implemented in C and on top of the Berkeley Socket Interface.[16] The first two points are based on UNIX network programming.[14,15] Error handling at the system and application levels and logging proved to be invaluable aids during the development of the software. Fault tolerance and load balancing have been adapted to the needs of the application and are described below. Fault tolerance is indispensable when combining many machines that are scattered all over the world. First, the higher the number of machines used the higher the chance that one or more will crash or simply get switched off during a computation. Secondly, if these machines are installed in different geographical time zones, machines appear to be switched on and off at any time for the central user. Load balancing is an essential feature to achieve efficiency because of the heterogeneous performance profile of the participating machines and the permanently changing load situation caused by the owners of the workstations.

Technical problems such as distributing the actual object code versions among the different heterogeneous LANs have been managed by a collection of *shell scripts*. To avoid any disturbance of local users, our processes use the C library call ‘nice’ to keep the impact on the owners’ processes as low as possible. To prevent processes from wasting resources after the program has been aborted at one or more hosts, which is likely to happen, especially while developing the application program, we set all sockets to non-blocking so that system calls such as ‘accept’ and ‘connect’ cannot block. All non-blocking system calls, and ‘read’ and ‘write’ calls are guarded by a ‘select’ call with predefined time-outs and retry counts. This ensures that in case of failures at process start-up and during the parallel computation no processes remain in a blocking state, probably exhausting process tables. Furthermore, to prevent otherwise lost processes from wasting CPU power, we implemented an alarm mechanism to terminate those processes after a predefined period of time. This technique proved to be valuable in cases where mysterious effects such as overstressed network file systems led to uncontrollable situations. Security precautions of various local system administrations forced us to implement two different start-up mechanisms for remote processes via the UNIX daemons rexecd and rshd, respectively. Resource constraints, such as the limited number of file descriptors per process, contributed to the decision to implement the process topology in a tree structure as explained below. Despite these resource limits, the number of leaf nodes of the tree can be increased arbitrarily by adding non-leaf nodes at the expense of higher communication costs because of the increased depth of the tree.

Besides the technical problems, the organization and logistics of such an approach are quite complex, especially obtaining accounts and co-ordinating some 30 LANs with several hundred machines and users. Many organizations are not fond of giving the permission to foreign users to use their CPU power and providing disk space. Therefore, we are especially grateful to all people who generously supported us and did not delimit our disk space to a smaller capacity than necessary for the storage of gene data at the site where they have to be accessed during the match.

PARALLELIZATION AND IMPLEMENTATION

The goal for our application is to accelerate the comparison of a molecular sequence of unknown functionality against all sequences stored in a gene database. The query sequence is sent from a central interactive working site to all sites holding parts of the distributed gene database. After the local area networks have finished the sequence analysis, the 50 best results are collected, sorted and delivered to the central administration process of the user.

Process topology

The process structure of our system is based on a tree topology, as shown in Figure 1. The root node accommodates the *central administrative process*. The leaf nodes perform the sequence analysis. We distinguish a *global layer* and a *local layer*. The global layer comprises the root node and all descendants down to, but excluding, the father nodes of the leaves. The local layer consists of the leaf nodes and their father nodes. Thus, all participating LANs are part of the local layer, whereas the global layer is responsible for connecting the LANs with the central administrative process.

The father nodes of the leaves accommodate an *administrative process*. These are either directly connected to the central administrative process, or via *router* processes that are responsible for bidirectional message routeing. The router processes, as a part of the global layer, on the one hand are used to construct an arbitrarily sized and shaped tree topology by means of configuration files. On the other hand, they are necessary to connect subnets that are not directly accessible from the Internet for security reasons.

The time needed to construct this topology depends on the depth of the tree, and the time-out values and retry counts after which those machines that are shut down or too heavily loaded are given up. With our networking software it took between one and two minutes to start up the program on about 800 out of 1000 machines.

Parallelization strategy

The gene database is distributed among all LANs. Thus, parallelization on the global layer is achieved by running the sequence analysis in each LAN on a different

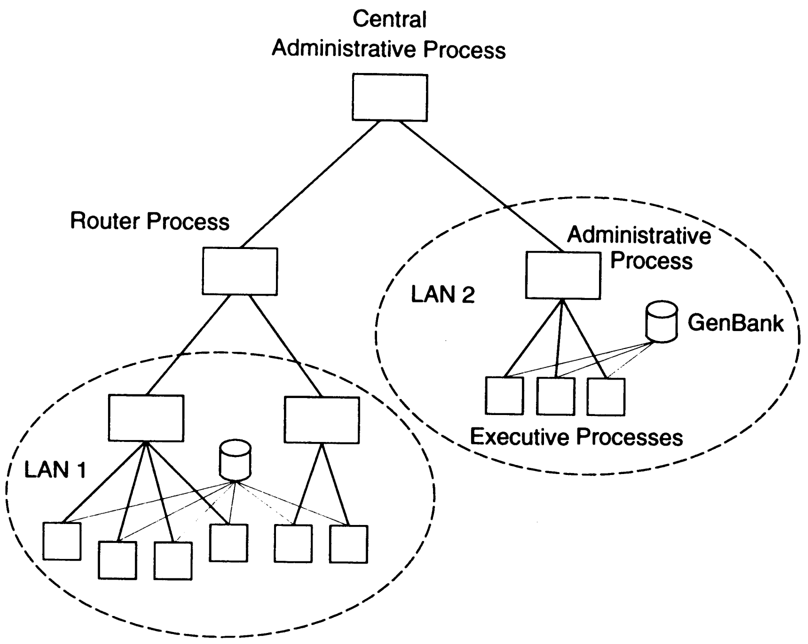

Figure 1. Process topology

part of the database. On the local layer, that is within each LAN, the search is parallelized by assigning parts of the local database to the leaf nodes which run a conventional sequential search algorithm, ssearch in this case.

Parallelization within a LAN is organized by means of a *workpool* that is provided by the administrative process. The workpool schedules the sequence analysis by assigning genes of the local database to the *executive processes* that are accommodated by the leaf nodes. In practice, the father node of the leaves is mapped on a server machine of the LAN and the leaf nodes itself on the client machines, the workstations. The workpool provides two essential features for large globally distributed heterogeneous environments: *fault tolerance* and *dynamic load balancing*.

Fault tolerance

Fault tolerance is provided by keeping a task in the workpool until it terminates successfully. If its executing process aborts because of a malfunctioning processor for example, the workpool reschedules this task to another available processor. The administrative processes are placed on server machines. In general, these machines provide access to the mass storage devices and the local gene database, and are thus indispensable within a LAN. Therefore, we did not implement further fault tolerance mechanisms for our experiments. Readers familiar with UNIX network programming may study the mechanism we developed to detect remote process abortion or network failures from the C code fragment in Figure 2. For details on network programming, we refer to the book of Richard Stevens.[15] The process executing this code is connected to another process via the non-blocking socket sock. If select returns with socket sock ready for reading, the second else part is executed. If recv returns zero, the receive buffer contains a null byte character. This indicates a broken connection. Assuming stable network connections, we conclude that the remote process aborted, for example because its host has been shut down.

Especially when invoking select with a set of sockets, the effect occurs that a socket is selected for reading although no message can be received with the following recv. We therefore implemented the busy wait loop of Figure 2. Only after executing several busy wait iterations is the message readable in such cases. This technique worked reliably, although we could not figure out why this effect occurs.

After program start-up, all processes wait for receiving messages in a select system call. This ensures that in particular the router processes and administrative processes do not consume system resources if this is not necessary. If a router process or an administrative process in the tree topology aborts or times out, such a mechanism guarantees that all descending processes of this process will terminate, and an appropriate error code can be sent to the central administrative process. In case an executive process aborts, this mechanism is used to detect that a task has to be restarted.

Dynamic load balancing

Dynamic load balancing is provided to ensure efficiency by scheduling parts of the local gene database dynamically to the executive processes. The technique is based on a *first come, first served* strategy that, in a simple manner, takes the different performance capabilities of the machines in the heterogeneous environment

```
... Initialize timeout
FD_SET(sock, &readfds);
maxfdpl = sock+1;

if ( (ready = select(maxfdpl, &readfds, (fd_set *) 0, (fd_set *) 0, &timeout)) < 0 ) {
    ... Error Handling
}
else if (ready == 0) { /* select timed out */
    ... Error Handling
}
else {
    if ( FD_ISSET(sock, &readfds) ) {
        while ((rc = recv(sock, msgbuf, 1, MSG_PEEK)) == -1)
            ;   /* busy wait for just incomming message */

        if (rc == 0) { /* Received null byte character */
            /* Partner process closed this connection unexpectedly */
            ... Error Handling
        }
    }
}
```

Figure 2. Detecting remote process abortion

into account. Distinct subsets of sequences in the database are assigned to tasks. The number of tasks is a multiple of the number of processors. The workpool manages task identifiers. Operationally, the workpool is an array with each component corresponding to a task. When a task is assigned to an executive process, the process identifier is stored in the array element with the corresponding index. When an executive process finishes its task, it reports its state to the workpool that in turn assigns a new task to this process. If an executive process is detected to be aborted the task that this process has been executing can be traced in the workpool array and is restarted on the next available executive process. The granularity of the tasks determines the efficiency of the computation. We obtained reasonably high efficiencies within a LAN with the number of tasks being proportional to the number of sequences in the local database, their lengths and the number of participating workstations as determined at program start-up.

EXPERIMENTAL RESULTS

For our experiments we implemented a simplified but optimized rigorous dynamic programming algorithm[11] of our own and also hooked up the fasta and ssearch codes[12] to our system. Here, we present the results of our experiments with the ssearch code to calculate the similarity scores of two genetic sequences against the NCBI-GenBank.[6] The ssearch program has been used with the default scoring matrix and gap penalties. We performed a number of single-query measurements comparing a *bacteriophage* sequence and a *human germline (AIDS)* sequence consisting of 105 and 773 bases, respectively, with the complete database.

Performance results

Table I reports elapsed run-times of two experiments, A and B, with our parallel implementation in comparison to extrapolated sequential run-times on a Sun SPARCstation1 and a Sun SPARCstation2. In this table, the *sequence length* denotes the number of bases of the DNA sequences that have been matched against all sequences of the gene bank. All run-times are wall-clock times. To determine the extrapolated sequential run-times, we compared our two genes with a representative portion of the database consisting of 845 sequences with an average length equal to the average sequence length of the entire gene bank. From the measured run-times we extrapolated the sequential run-times using the fact that the complexity of the sequencing algorithm is linear with the number of genes in the database and the number of bases of the genetic sequences to be matched.

We want to emphasize that these measurements have been conducted under normal daily loads. The computation used CPU power that would be idling otherwise. Therefore, the price/performance ratio of such configurations can be regarded as being close to zero.

Quantitatively, the two configurations that yielded the results presented in Table I are characterized in Table II. The entries in the bottom two lines of Table II have been determined with a small loop consisting of an integer test load with an average run-time of some 10 seconds on a SPARCstation2. Elapsed (wall clock) time and CPU time of this test load were measured and scaled so that a dedicated SPARCstation2 would deliver 40 MIPS. The MIPS values shown are the sums of these MIPS values, measured on all machines used just before the sequencing. These values give a rough idea of the *total performance* we would get from a dedicated configuration. They are considerably higher than those of today's fastest supercomputers.

Table III shows the different machine models that have been incorporated in our

Table I. Performance results

Experiment	Number of workstations	Sequence length	Parallel run-time	Sequential (SPARC1)	Sequential (SPARC2)
A	803	105	5·0 min	21·5 h	10·4 h
B	803	773	18·4 min	6·6 days	3·2 days

Table II. System configuration

	Experiment A	Experiment B
Machines used	803	803
Local area networks	31	31
Administrative processes	46	46
Router processes	7	7
Machine types	32	32
MIPS (SPARCstation2 ≡ 40 MIPS)	26,800	26,790
Average idle percentage	93·0	92·7

Table III. Employed machine models

Manufacturer	Machine models
DEC	DECstation, DECmicrovax
HP	HP9000, HP Apollo 9000
IBM	RS6000
NEC	EWS-4800
SGI	Crimson, Indigo, Iris 4D, Personal Iris
Sony	NEWS
Sun	Sun3, Sun4, SPARC10, SPARCserver

configuration. There have been various different types of most machine models and even more different UNIX operating system versions.

In Tables I and II we presented those two measurements of our experiments with the maximum number of machines. We had hands on about 1000 machines but typically about 20 per cent of them were shut down. Since a network of several hundred machines permanently changes its configuration, it is impossible to get a fixed number of machines with reproducible load situation for reference measurements. Actually, during experiment B, two machines left the configuration for unknown reasons, and the run terminated with 801 of the initial 803 machines. Most of the LANs we used had processor idle percentages of at least 90. Such values are very common as we also confirmed by logging CPU idle percentage data in our LAN for a period of several months. This average use remains almost constant from our experience, although the use of individual workstations varies permanently. However, some LANs with high-speed workstations were heavily loaded with parallel cluster applications.

Table IV gives additional information about our measurements. The minimum and

Table IV. Performance data

Experiment	Sequence length	Minimum run-time	Maximum run-time	Average run-time	Speed-up vs. SPARC1	Speed-up vs. SPARC2
A	105	32 s	302 s	1·8 min	258	125
B	773	172 s	1103 s	8·0 min	514	249

maximum run-times denote the minimum and maximum of the elapsed run-times in the individual LANs. The maximum value determines the total elapsed run-time given in Table I. The range of these values is relatively wide, because the gene database has been partitioned statically. This partitioning does not adapt to the permanently changing load situation of the individual workstations and thus the average computing power of entire LANs. The run-time distribution across LANs and hence the total performance can easily be improved by proper organization of the distributed database. However, in our experiments this was not possible since on some of the LANs the disk space available was too small to store a portion of the database with its size proportional to the computational power.

The *speed-up* values are calculated with respect to the elapsed parallel run-times and the sequential run-times given in Table I. The *average run-time* denotes the mean run-time with respect to the run-times of all participating LANs. Using these average values, we obtain speed-ups of 717 and 1188 corresponding to the 105 base and 773 base query sequences and the sequential run-times on a SPARCstation1. The value of 1188 can also be interpreted as the number of processors of a virtual machine consisting of SPARCstation1 processing elements and delivering almost optimal performance.

We presented the MIPS performance of our dedicated configuration in Table II. To obtain an idea of the efficiency of our experiments, we estimate the MIPS performance of our application by assuming the sequential run to perform 24 MIPS on a SPARCstation2, which is about 60 per cent of its peak performance, and scaling this value with the achieved speed-up given in Table IV. These values are given as *estimated MIPS* in Table V. The *average MIPS* values are calculated with respect to the average run-times given in Table IV and indicate the possible performance of our configuration with an optimally balanced distributed database.

Efficiency analysis

Efficiency, which is traditionally one of the primary goals of parallelization, in our case has to obey the boundary condition to not disturb other users in the network. Nevertheless, performance, and thus efficiency, remains our goal. It is well known[13] that interprocessor communication causes performance degradation because of the low bandwidth of today's networks such as the Ethernet. In our application, only communication within the LANs deserves closer examination, because communication across the LANs comprises just the initial distribution of the query sequence and the final collection of the best alignment scores. We investigate the behaviour and efficiency of our implementation within the LAN of our institute by means of Figure 3.

Figure 3 shows *normalized speed-ups* of the two sequence analyses with query

Table V. MIPS performance data

Experiment	Sequence length	Estimated MIPS	Average MIPS
A	105	3000	8350
B	773	6000	13,650

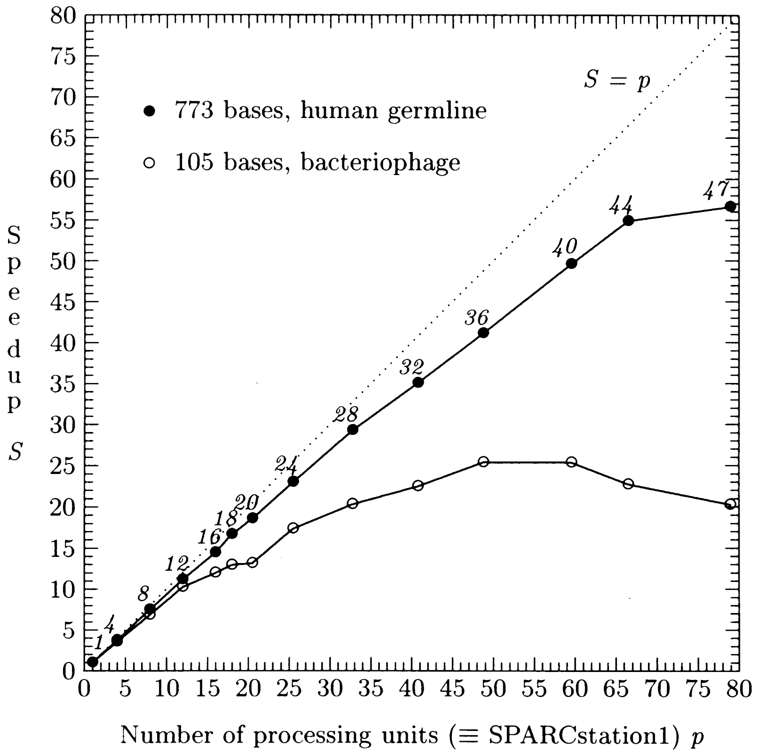

Figure 3. Normalized speed-up in a heterogeneous environment (cf. Table VI)

sequences consisting of 105 bases and 773 bases with respect to a unit processing element delivering a performance equivalent to a SPARCstation1, which is the slowest machine of the configuration. All participating machines have been benchmarked with the sequence analysis application that is the subject of our investigations. These values differ from the MIPS values on each particular machine model depending on the machine architecture. The normalized performances are given in Table VI. The

Table VI. Machine configuration and normalized peformance (cf. Figure 3)

Machine	Number	Normalized performance
Sun SPARCstation1	18	1·0
Sun SPARCstation1+	7	1·25
Sun SPARCstation2	11	2·0
Sun SPARCserver490	2	1·7
Sun SPARCserver690	2	3·7
SGI Iris 4D/70	1	1·1
SGI Iris 4D/20	1	1·7
SGI Iris 4D/320 (2 proc.)	2	2·1
SGI Personal Iris	1	2·1
SGI Iris Crimson	1	5·4
IBM RS6000	1	5·0

speed-up measurements up to 18 machines have been conducted on SPARCstation1 only. Then, other machines have been added in the order as listed in Table VI. We used both processors of the Iris 4D/320, each with a normalized performance of 2·1.

The numbers above the filled circles in Figure 3 denote the number of machines actually used. Both speed-up curves in Figure 3 show a typical behaviour. Up to 18 processors, we obtain slightly sublinear speed-up as expected with respect to the increasing communication rate. Then, the additional and more powerful machines raise the overall performance. With higher numbers of machines the increase of speed-up is reduced. The two curves are distinguished by the ratio of computation and communication, which increases proportional to the number of bases of the query sequence. Communication during the match of the 105 base query sequence even leads to Ethernet congestion and decreasing speed-up with more than 36 machines. Matching the 773 base gene, this effect does not occur with the number of machines used, because the ratio of computation and communication is still high enough.

The communication within a LAN comprises the exchange of identifiers and partitioning information between the workpool and all executive processes, and the access to the local portion of the gene database. The latter one is the reason for the high traffic and Ethernet congestion with larger numbers of processors. This implementation can still be improved because each executive process reads the whole local database before the sequence analysis is performed. The amount of data transferred via the Ethernet is therefore the size of the local database multiplied by the number of executive processes. If each executive process would only read that part of the database that is used for matching, the amount of transferred data would be reduced to the size of the database, independent of the number of particpating processes. This would improve speed-up and efficiency especially in those LANs with high numbers of workstations. The benefit for sequence analyses with shorter query sequences will be more significant because the improvement of the ratio of communication and computation will be relatively higher.

CONCLUSION AND FUTURE WORK

This work shows that a massively parallel approach using several hundred workstations, dispersed over all continents and connected via the Internet, can be successfully employed for solving problems with low communication requirements across LAN borders. Technically, we have confirmed the following points:

1. Managing a parallel computation on about 800 workstations located in 31 local area networks and five continents.
2. Co-ordinating a complex distributed and heterogeneous architecture, including different file system structures, and permission, authorization and security mechanisms.
3. Automatized distribution and installation of a portable program to about 1000 workstations of seven different manufacturers and numerous different UNIX operating system versions.
4. The proof that massively parallel computing in the Internet is feasible for appropriate applications.
5. Running a molecular sequence analysis program in several minutes, whereas the sequential implementation would take several days.

Although many technical problems have been solved in principle this approach needs further refinement, technically as well as organizationally. It would be very interesting to extend our prototype and design a general purpose platform for parallel computing in the Internet. Such a platform might serve as a basis for investigations on the feasibility of other compute-intensive problems to be solved with large numbers of machines dispersed in the Internet. We hope that our work encourages the use of network programming as a valuable tool for parallel programming in heterogeneous environments and inexpensive workstation networks in particular. Our experience has already benefitted several people parallelizing specific applications on workstation networks.

Combining our approach with modern database technology would be a next step towards a parallel, distributed system for molecular sequence analysis. To establish a system that uses the techniques developed in this paper, some technical as well as organizational and logistical efforts have to be undertaken. A straightforward design would include a *distributed database*, where replication could be employed to guarantee reliability and load balancing at the same time. Since the genetic sequences in such databases are *read-only* data, consistency mechanisms can easily be implemented. Currently, researchers are investigating new concepts for managing biological DNA and protein databases.[17] The main problem of achieving efficient load balancing with a particular distribution might be solved with a *balance-by-growth* concept, where those new sequences that are input to the distributed database would be stored at a site that computationally has been least used during previous computations. Assuming a sufficiently high rate of new sequences per day, proper balancing could even be guaranteed if new machines join the system. Furthermore, co-ordinated shutdown hours of the individual LANs for administrative purposes would increase the system reliability. Most of these aspects are technically straightforward to implement but a challenge to be done.

acknowledgements

Many people from all over the world have contributed to this work. Thanks to everyone who supported our requests for accounts and for the confidence and patience with the experiments. This work would not have been possible without Clemens H. Cap who came up with the idea to parallelize the sequence analysis application and co-operated in the organization of the experiment, as well as design and implementation of the networking part of the software. Kurt Bauknecht and Lutz Richter provided a stimulating working environment.

This work was supported by Siemens AG (ZFE), Germany and Schweizer Bundesamt für Konjunkturfragen, Grant No. 2254.1.

REFERENCES

1. S. Ahuja, N. Carriero and D. Gelernter, 'Linda and friends', *IEEE Computer*, **19**, (8), 26–34 (1986).
2. V. S. Sunderam, 'PVM: a framework for parallel distributed computing', *Concurrency: Practice and Experience*, **2**, (4), 315–339 (1990).
3. H. Nakanishi, V. Rego and V. Sunderam, 'Superconcurrent simulation of polymer chains on heterogeneous networks', *Supercomputing '92*, Minneapolis, 1992, pp. 561–569.
4. D. W. Smith, J. Jorgensen, J. P. Greenberg, J. Keller, J. Rogers, H. Garner and L. Ten Eyck, 'Supercomputers, parallel processing, and genome projects', in D. Smith (ed.), *Biocomputing: Informatics and Genome Projects*, Academic Press, in press.
5. C. Sander, R. Schneider and P. Stouten, 'The human genome and high performance computing in

molecular biology', in H. W. Meuer (ed.), *Supercomputer '92, Anwendungen, Architekturen, Trends,* Springer-Verlag, 1992, pp. 32–48.
6. *NCBI-GenBank*, National Center for Biotechnology Information, National Library of Medicine, 38A, 8N805, 8600 Rockville Pike, Bethesda, MD 20894, U.S.A.
7. P. Bork, 'Entschlüsselung von Proteinfunktionen mit Hilfe des Computers: Erkennen und Interpretation entfernter Sequenzähnlichkeiten', in R. Hofestädt, F. Krückeberg and T. Lengauer (eds), *Informatik in den Biowissenschaften*, Springer-Verlag, 1992, pp. 67–78.
8. R. A. Wagner and M. J. Fischer, 'The string to string correction problem', *Journal of the ACM,* **21**, (2), 168–173 (1974).
9. T. F. Smith and M. S. Waterman, 'Identification of common molecular subsequences', *Journal of Molecular Biology,* **147**, (1), 195–197 (1981).
10. S. B. Needleman and C. D. Wunsch, 'A general method applicable to the search for similarities in the amino acid sequence of two proteins', *Journal of Molecular Biology,* **48**, (3), 443–453 (1970).
11. R. J. Lipton, T. G. Marr and J. D. Welsh, 'Computational approaches to discovering semantics in molecular biology', *Proc. IEEE,* **77**, (7), 1056–1060 (1989).
12. W. R. Pearson, 'Searching protein sequence libraries: comparison of the sensitivity and selectivity of the Smith–Waterman and FASTA algorithms', *Genomics,* **11**, (3), 635–650 (1991).
13. C. H. Cap and V. Strumpen, 'Efficient parallel computing in distributed workstation environments', *Parallel Computing,* **19**, (11), 1221–1234 (1993).
14. D. E. Comer and D. L. Stevens, *Internetworking With TCP/IP, Vol. II: Design, Implementation, and Internals*, Prentice-Hall, 1991.
15. W. R. Stevens, *UNIX Network Programming*, Prentice-Hall, 1990.
16. M. R. Horton, *Portable C Software*, Prentice-Hall, 1990.
17. B. Rieche and K. R. Dittrich, 'A federated DBMS-based integrated environment for molecular biology', *Proceedings of the 7th International Working Conference on Scientific and Statistical Database Management*, Charlottesville, VA, IEEE, 1994, pp. 118–127.