CABIOS

Vol.7. no 1 1991
Pages 85 – 88

Scanning protein sequence databanks using a distributed processing workstation network

Geoffrey J.Barton

Abstract

The programme pscan has been developed to distribute protein databank scans over a network of computers that share a common filesystem. pscan may be used in conjunction with most conventional sequence comparison programmes with few modifications. In test runs using the Smith–Waterman dynamic programming algorithm, the time required to scan a 6858 sequence databank using a query sequence 740 residues long was reduced from ~50 min for a single processor, to ~11 minutes for five processors. Accordingly, pscan provides a low-cost, portable alternative to dedicated parallel processing computers.

Introduction

The identification of homology between a newly determined sequence and well-characterized proteins in the databank is one of the most powerful tools for the prediction of protein structure and function. Any algorithm developed for the comparison of two sequences, (e.g. see Needleman and Wunsch, 1970; Sellers, 1974; Smith and Waterman, 1981) may be applied to a databank scan; however, the most rigorous methods have traditionally not been used due to their heavy CPU requirements. Instead, ingenious techniques have been devised to allow the high-speed scanning of large databanks on small computers (e.g. see Lipman and Pearson, 1985; Pearson and Lipman, 1988), but these methods make approximations, and as a result can miss important relationships that are detected by the rigorous dynamic-programming algorithms (Barton and Sternberg, 1990, Barton, 1990). The recent development of high-performance, low-cost Unix workstations (e.g. the Sun Microsystems SPARCstation 1) make the use of rigorous methods feasible with current protein databanks and typical length queries. For example, a scan of the 16 524 sequence NBRF-PIR (National Biomedical Research Foundation—Protein Identification Resource) databank (v. 24, main and new entries), with a 141 residue query, using the Smith–Waterman algorithm (Smith and Waterman, 1981), requires <30 min CPU time, while a 750 residue query needs <100 min.

With the stimulus of projects to map and sequence complete genomes, the size of sequence databanks will grow at an increasing rate. As a consequence, still greater speed will be necessary to maintain current scan times, and permit the application of yet more sophisticated analytical techniques. One route to this goal exploits dedicated parallel processing hardware. For example, Coulson *et al.* (1987) describe the use of the AMT-DAP (active memory technology—Distributed array processor), a SIMD (single instruction multiple data) parallel processing machine, to provide significant speed improvements over conventional serial processors for databank scanning. SIMD machines perform the same operation simultaneously on data distributed over a large number of simple processors (e.g. add 1 to each element of an array). In contrast, a MIMD (multiple instruction multiple data) architecture parallel processing machine (e.g. an array of INMOS transputers) allows sophisticated operations to be run asynchronously on a number of processors.

While dedicated parallel processing hardware can give significant increases in speed for suitable algorithms, considerable effort must be expended to adapt existing software to take full advantage of the specific machine architecture. This often leads to fast, but non-portable code. A network of workstations that share common file storage can be used as a simple MIMD machine. A system of this type has the advantage that code written to run on a single workstation need not be modified significantly to run on multiple processors.

In this paper the programme *pscan*, which utilizes a network of workstations as a MIMD computer for protein databank scanning, is described. *pscan* can be used to run most conventional sequence comparison programmes on a network with minimal modifications to the scanning programme, and gives speedup factors approximately linear with processor number.

Algorithm

A simple strategy for sequence scanning on a network of processors is to predivide the databank into sections, one for each processor. A controlling programme resides on one processor, which when asked to execute the scan, sends a command to each processor on the network, instructing it to compare the query sequence to its local fragment of databank. The control programme monitors the remote processors, and when all are finished, merges the results into a single file. Clearly, if one processor takes appreciably longer to complete its job than the rest, the speedup is reduced. In order to minimize this problem, the databank file must be split into CPU-equivalent

Laboratory of Molecular Biophysics, The Rex Richards Building, South Parks Road, Oxford OX1 3QU, UK

chunks. However, this split must be repeated every time a new databank release is obtained, or the number of processors is changed. Furthermore, on a system in which the load on each host may vary due to other users, it is impossible to predict the subdivision of the databank that would ensure all processors are occupied throughout the scan. In order partially to circumvent these problems, this basic strategy was modified so that the databank fragments (chunks) are created at run-time.

In order to make best use of the available machines, the controlling programme should be tightly coupled to the scanning programmes on each processor. Small numbers of sequences would be sent to each remote node for scanning, and the CPU status and number of active processes continuously monitored. In this way, the programme could rapidly accommodate changing loads on individual processors and optimize the CPU power available. While such a system may be possible to implement on a workstation network, it would require machine-specific, system-level programming. In order to develop a system that is generally applicable to networked computers running NFS (network file system), the programme *pscan* was written to require only the basic remote-shell command 'rsh' that is common to systems running NFS protocols. This takes the form 'rsh host command' where 'host' is the name of the processor on which the 'command' is to be executed.

In outline, *pscan* creates a mini-databank of *C* sequences for each processor. The programme starts the scans on each remote machine, then monitors the status of the remote jobs at pre-determined intervals. When a remote process has finished, *pscan* creates a new mini-databank of *C* sequences for that machine. This operation is repeated until the databank is exhausted. Finally, once all sequences have been scanned, the results obtained by each processor are merged. By feeding each processor *C* sequences at a time, the problem of a heavy load on one processor excessively slowing the complete scan is reduced. However, the value of *C* (chunk size) must be adjusted to be small enough to cope with uneven processor loading, yet large enough to avoid excessive overheads in starting processes.

Programme details

pscan requires at least two computers connected on a network, a protein sequence databank file and a query sequence file. In addition, a hosts file is provided that identifies the working directory, and names of the processors that are to be utilized in the scan. Each hostname is accompanied by a factor by which the basic number of sequences, *C*, sent to that processor is multiplied. This allows the scan to be shared more efficiently where the available processors are of different speeds. The same mechanism may also be used to balance the scan where there are known to be heavily and lightly loaded processors on the same network.

pscan is supplied with two additional parameters. The chunk size (*C*) is the number of sequences to be sent to a host in each job, while the sleep time is the time in seconds that *pscan* waits between checks of which hosts have completed their chunk.

pscan first initializes output files and status files for each processor. The status files are the mechanism through which *pscan* and the remote hosts communicate. On initialization, each status file contains the string 'Finished', the alternative state being 'Running'.

Once initialization is complete, the main programme loop begins:

(i) The status files are read in turn, and a list of those that signal 'Finished' (i.e. all on the first pass) is created.

(ii) The list created in (i) is examined, and for each processor ('hostX') that is 'Finished', the next chunk of sequences is read from the databank, then written to the host specific sequence file. The scanning programme is then executed for each processor that has Finished, by using the Unix remote-shell command ('rsh'). Once the remote host process has started, the hosts status file is updated to 'Running'.

(iii) The programme that executes on the remote processor updates its status file to Finished when its scan is complete, thus signalling to *pscan* that it is ready for another chunk of sequences.

(iv) Items (i)–(iii) are repeated until the databank file is exhausted. Once all processors have Finished, the individual hostX.out files are sorted and merged.

The scanning programme may be any conventional databank-scanning programme that takes a query sequence, a databank file, and produces output for each query versus databank-entry comparison. However, the scanning programme must be modified to accept the host name (e.g. 'host3') as a parameter, and on completion of its scan, to update the hostX.status file to 'Finished'. If the source code for the scanning programme is not available, then an intermediary programme can be written which when executed by *pscan*, constructs the necessary commands for the scanning programme, then executes the scanning programme, before finally updating the hostX.status file. In this way, any suitable sequence analysis software may be adapted to run distributed over a network via *pscan*.

Implementation and availability

The programme *pscan* is written in C for the Sun-4 architecture running SunOS 4.0.3 and NFS. However, the programme should run with little modification on any Unix system that supports NFS, and provides a C library function to execute a Unix command from within a C programme.

The source code of and details of the implementation of *pscan* are available to academic users on a variety of media, or via electronic mail.

Evaluation and discussion

As a test, two proteins, human α-haemoglobin (HAHU: 141 residues) and human annexin 6 (A6H: 740 residues) were

scanned against the NBRF-PIR main databank (v. 24.0, 6858 sequences) using the programme *sw* (G.J.Barton, unpublished) that implements the Smith–Waterman (1981) local similarity algorithm. For each protein, scans were performed with from one to five processors, while the chunk size for each scan was varied from 25 to 2500 sequences for A6H and from 100 to 2500 for HAHU. The workstation network consisted of a 16 M.I.P.S. (millions of instructions per second) file-server, three 12.5 M.I.P.S. workstations, and two 16 M.I.P.S. workstations. Normally, the file-server on a network of this type is not used for compute-intensive tasks; accordingly, in order to establish the maximum gains possible by this method, *pscan* was executed alone on the file-server, while *sw* was run on the 12.5 M.I.P.S. machines (1–3 processors), leaving only scans with four and five processors making use of the two 16 M.I.P.S. workstations. In practice, *pscan* requires very little CPU time and system overheads, and may be run on a processor that also executes the scanning programme (see below). As far as possible, the test runs were performed when all processors were lightly loaded.

For each test run, a speedup ratio was calculated by dividing the elapsed time for a scan using *sw* on a single 12.5 M.I.P.S. processor (HAHU, 626 s; A6H, 2912 s) by the elapsed time required by the multiprocessor scan. Elapsed times were used since these give a fairer estimate of the advantage in using the distributed processing system than would summing the CPU times for individual processors.

Figure 1 illustrates the relationship between the speedup ratio and chunk size for each combination of processors and sequence. The uniprocessor scans (A and F) illustrate the additional overhead introduced by running *pscan* when compared to *sw* on its own. At a chunk size of 100, and a sequence length of 141 (scan A), the speedup ratio is reduced from 1.0 to 0.6. This reflects the cost incurred by *pscan* in creating the local chunk-sized databank files, starting the *sw* process and monitoring the hostX.status file. As the chunk size is increased, the number of processes that must be started during the course of a scan is reduced. This is seen in an improvement in the speedup ratio to 0.9 at a chunk size of 2500. With the longer query sequence (scan F), the ratio of *sw* CPU time to *pscan* CPU time is increased, leading to the higher observed speedup ratios (0.8 at 100 residues; 0.96 at 2500 residues). This effect is also observed for scans with more than one processor.

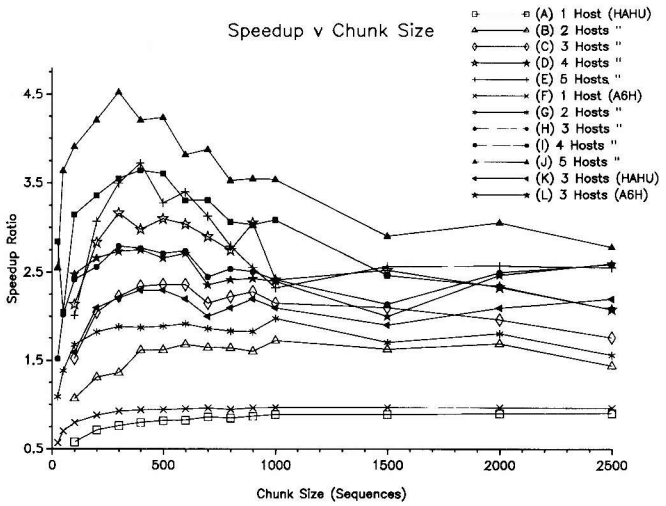

Fig. 1. The relationship between the speedup ratio obtained using *pscan* with 1–5 processors running the Smith–Waterman algorithm (programme *sw*) (Smith and Waterman, 1981), and the number of sequences sent to each processor in each job (chunk size, *C*). Runs A–E were performed with human α-haemoglobin (141 residues), while F–J used human annexin 6 (740) residues. The sleep time was set to 5 s for runs A–E and 10 s for runs F–J. Chunk sizes sent to the 16 M.I.P.S. machines were increased by a factor of 1.3. Runs K and L show speedup factors obtained for runs using three processors, where *pscan* shared one processor with *sw* jobs (see text).

The scans performed with more than one processor (B–D, F–L) show similar overall behaviour with respect to chunk size. A low speedup ratio is observed below a chunk size of ~200, this rises to a maximum between ~200 and 500, before falling again for chunk sizes >500. The reduction in performance for larger chunk sizes is due to poor phasing of the remote processor jobs. Since each individual chunk requires a significant proportion of the total scan time to complete, if the last databank chunk is sent to a processor shortly before all other processors complete their tasks, there is a period at the end of the scan when only one processor is active.

Unfortunately, it was not possible to dedicate all workstations on the network to this study. As a consequence, the speedup values for four and five-processor scans show major fluctuations due to transient high loading on one or more of the processors. This phenomenon is particular marked for chunk sizes >500 sequences due to the phasing problem outlined above.

Figure 1 shows that using two or more processors provides a useful improvement in execution time for the scan. For example, with the 740 residue query (A6H), a chunk size of

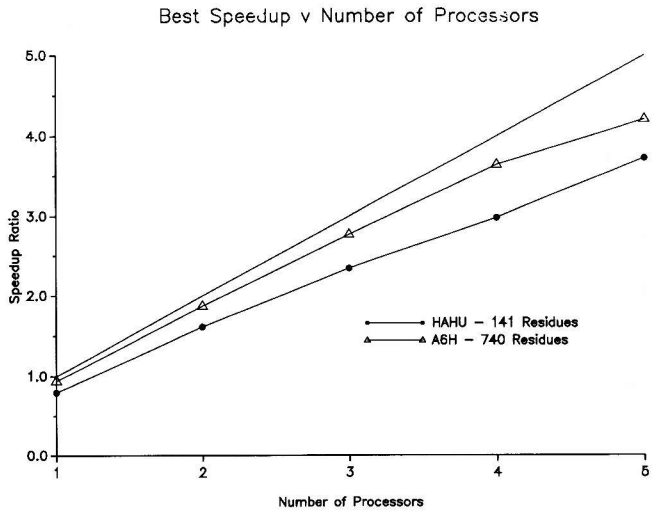

Fig. 2. Data from Figure 1 for the best speedup factor obtained for each sequence scan and number of processors. The upper line shows the values expected for a perfect parallel processing machine.

300 and three processors, the speedup factor is 2.8. This reflects a reduction in overall elapsed time from 48.5 to 17.2 min, while with five processors (speedup of 4.5), the time is reduced to 10.7 min. It may be argued that the test runs were actually performed using $N + 1$ processors, since *pscan* was executed on a machine that did not run *sw*. However, runs K and L were performed with *pscan* resident on one 12.5 M.I.P.S. workstation, while this workstation and two additional 12.5 M.I.P.S. hosts ran *sw*. Clearly, there is little advantage in running *pscan* on a separate processor.

Figure 2 illustrates the relationship between the best speedup obtained (regardless of chunk size) and the number of processors. Adding more processors is beneficial, however: as the number of processors increases, the speedup gained per processor diminishes. This effect is due to the greater likelihood as processors are added that more than one processor will simultaneously finish its chunk and be waiting for a new job. The problem could be countered by increasing the sophistication of the strategy for checking the hostX.status files, e.g. by using previous cycles to predict which hosts are likely to finish in the next cycle, and only checking those. It should be noted that this effect is actually larger for four and five processors than is suggested by Figure 2 since runs D, F, I and J include the two faster (16 M.I.P.S.) workstations.

The current version of *pscan* has been used by the author for large numbers of database scanning problems with few difficulties. However, in its current form, the programme will loop indefinitely if one processor node fails. This is a weakness of the strategy that would preclude the use of the programme to provide a general database-scanning service to inexperienced users. The simplest method of overcoming this problem would be to monitor the time required by each processor to complete a chunk as described above. If a processor is detected as taking a predetermined proportion longer to complete than anticipated, then its job could be resubmitted to another processor on the network. This would enable the scan to complete, albeit with the sacrifice of potential duplication in the sequences scanned. A more complete solution to failure recovery would require *pscan* to check which sequences sent to a node have not been processed. Once the database is exhausted, any unprocessed sequences could then be redistributed over the functional processors in order to complete the scan. Although more elegant, this solution would necessitate a more sophisticated mode of reading the database to allow direct access to individual sequence entries.

Conclusions

In this paper, a programme has been described that allows conventional sequence databank-scanning software to be distributed over a network of workstations. The best speedup factors obtained using the Smith–Waterman algorithm (Smith and Waterman, 1981) with a query length of 141 residues, were: two processors, 1.72; three processors, 2.36; four processors, 3.1; five processors, 3.73. With the more compute-intensive query of 740 residues, these factors improved to 1.98, 2.79, 3.64 and 4.52 respectively. The programme therefore gives greatest benefit for large problems.

While only one conventional single-pair protein sequence comparison algorithm has been discussed in this paper, the method could be readily applied to the more sensitive (and equally compute intensive) multiple alignment and flexible pattern-matching techniques (Barton and Sternberg, 1987; Barton and Sternberg, 1990; Barton, 1990), DNA and protein structure-scanning problems.

The distributed processing strategy described in this paper is unlikely to be as fast as machine-specific code written for a dedicated parallel processing computer. However, the approach requires minimal reprogramming to distribute existing algorithms, and has the advantage of giving a significant speedup on the type of low-cost computers that are becoming the mainstay of many departmental computing facilities.

Acknowledgements

The author thanks the Royal Society for the support of a University Research Fellowship, the Oxford Centre for Molecular Sciences for the use of computing facilities, Professor L.N.Johnson for her encouragement, and Dr D.Barford for his comments on the manuscript.

References

Barton,G.J. (1990) Protein multiple sequence alignment and flexible pattern matching. *Methods Enzymol.*, **183**, 403–428.

Barton,G.J. and Sternberg,M.J.E. (1987) A strategy for the rapid multiple alignment of protein sequences: confidence levels from tertiary structure comparisons. *J. Mol. Biol.*, **198**, 327–337.

Barton,G.J. and Sternberg,M.J.E. (1990) Flexible protein sequence patterns—a sensitive method to detect weak structural similarities. *J. Mol. Biol.*, **212**, 389–402.

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

Lipman,D.J. and Pearson,W.R. (1985) Rapid and sensitive protein similarity searches. *Science*, **227**, 1435–1441.

Needleman,S.B. and Wunsch,C.D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, **48**, 443–453.

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

Sellers,P.H. (1974) On the theory and computation of evolutionary distances. *SIAM J. Appl. Math.*, **26**, 787–793.

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

Received on August 27, 1990; accepted on September 25, 1990