Whole Genome Comparison on a Network of Workstations

Arpith Jacob
Vellore Institute of Technology, Vellore, India
arpith@arpith.com

Sugata Sanyal
Tata Institute of Fundamental Research, Mumbai, India
sanyal@tifr.res.in

Marcin Paprzycki
SWPS and IBS PAN, Warsaw, Poland
marcin.paprzycki@ibspan.waw.pl

Rajan Arora
Rajan Arora Indian Institute of Information Technology, Allahabad, India
arorarajan@gmail.com

Maria Ganzha
EUH-E, Elblqg and IBS PAN, Warsaw, Poland
ganzha@euh-e.edu.pl

Abstract

Whole genome comparison consists of comparing or aligning genome sequences with a goal of finding similarities between them. Previously we have shown how SIMD Extensions used in Intel processors can be used to efficiently implement the, genome comparing, Smith-Waterman algorithm. Here we present distributed version of that algorithm. We show that on somewhat outdated hardware we can achieve speeds upwards of 8000 MCUPS; one of the fastest implementations of the Smith-Waterman algorithm.

1. Introduction

Genome sequence similarity searches are often utilized in Computational Biology. Their goal is to identify closely related genomic sequences assuming that high degree of similarity in genome sequences may imply similarity of functional or structural characteristics. In computational practice, an alignment score between the query sequence and sequences in a database are calculated to assess similarity. One of popular algorithms to compare genomic sequences is the Smith-Waterman algorithm and our goal is to show how it can be efficiently implemented on a network of workstations.

In our earlier work ([1]) we have described how utilization of Intel's MMX/SSE2 instruction set speeds-up the Smith-Waterman algorithm on a single processor. Results of our experiments, preformed on a Pentium III running at 500 MHz and a Pentium 4 running at 1.4 GHz showed speedup of the order 10-62 over a sequential algorithm (see [1] for a complete description of the proposed approach and summary of obtained results).

In this paper we describe a coarse-grained approach to parallelization of the Smith-Waterman algorithm (which utilizes also our SIMD-based algorithm). We follow earlier research of Strumpen ([3]) who used a massively parallel approach to distribute the database search process in a heterogeneous environment on more than 800 workstations on the the Internet. Separately, Martins and associates ([4]) described an event-driven multithreaded implementation of the sequence alignment algorithm for the EARTH architecture, on a Beowulf cluster of 128 Pentium Pro microprocessors.

We proceed as follows. In the next section we

summarize the Smith-Waterman algorithm. We follow with the description of our coarse-grained approach and present results of experiments on 64 workstations.

2. The Smith-Waterman algorithm

Initially, Needleman and Wunsch [5] and Sellers [6] introduced the global alignment algorithm based on the dynamic programming approach. Smith and Waterman [7] proposed an $O(M^2N)$ algorithm to identify common molecular subsequences, which took into account evolutionary insertions and deletions. Later, Gotoh [8] modified this algorithm to run at $O(MN)$ by considering affine gap penalties. Each of these algorithms depended on saving the entire $M \times N$ matrix in order to recover the alignment. The large space requirement problem was solved by Myers and Miller [9] who presented a quadratic time and linear space algorithm, based on a divide and conquer approach. Finally, Aho, Hirschberg and Ullman [10] proved that comparison algorithms that compare symbols to see if they are equal or unequal, have to take time proportional to the product of their string lengths. Let us now describe

		T	C	G	A	C	A	T	A
	0	0	0	0	0	0	0	0	0
A	0	0	0	0	5	0	5	0	5
C	0	0	5	0	0	10	3	1	0
T	0	5	0	1	0	3	6	8	1
A	0	0	1	0	6	0	8	2	13
G	0	0	0	6	0	2	1	4	6
G	0	0	0	5	2	0	0	0	0
C	0	0	5	0	1	7	0	0	0
A	0	0	0	1	5	0	12	5	5

Figure 1. Comparison Matrix: Optimal score: 13, Match: 5, Mismatch: -4, Penalty: 0 + 7k. Optimal Alignment: A C A T A, A C - T A

the Smith-Waterman algorithm (an example of its operation was depicted in Figure 1; more details can be found in [1]). Let us consider two genomic sequences A and B of length M and N respectively, that are to be compared using a substitution matrix ∂. Here, we utilize the affine gap weight model. The gap penalty is given by: $W_i + kW_e$ where $W_i > 0$ and $W_e > 0$. W_i is the penalty for initiating the gap and W_e is the penalty for extension of the gap, which varies linearly on the length of the gap. The substitution matrix ∂ lists the probabilities of change from one nucleotide or amino acid into another in the sequence. There are two families of matrices used in the algorithm: the Percent Accepted Mutation (PAM) and the Block Substitution Matrices (BLOSUM). A maximization relation is used in order to calculate the optimum local alignment score according to the following recurrence relations (the highest value in the H matrix gives the optimal score):

$$E(i,j) = H(i,j) = F(i,j) = 0, \quad \text{for } i = 0 \text{ or } j = 0$$

$$E(i,j) = \max \begin{Bmatrix} E(i-1,j) - W_e \\ H(i-1,j) - W_i - W_e \end{Bmatrix}$$

$$F(i,j) = \max \begin{Bmatrix} F(i,j-1) - W_e \\ H(i,j-1) - W_i - W_e \end{Bmatrix}$$

$$H(i,j) = \max \begin{Bmatrix} 0 \\ E(i,j) \\ F(i,j) \\ H(i-1,j-1) + \partial(A_i, B_j) \end{Bmatrix}$$

Recurrences can be understood as follows: the E (F) matrix holds the score of an alignment that ends with a gap in sequence A (B). When calculating the $E(i,j)$th ($F(i,j)$th) value, both extending an existing gap by one space, or initiating a completely new gap is considered. The $H(i,j)$th cell value holds the best score of a local alignment that ends at position A_i, B_j. Hence, alignments with gaps in either sequence, or the possibility of increasing the alignment with a matched or mismatched pair are considered. A zero term is added in order to discard negatively scoring alignments and restart the local alignment. One of possible many optimal alignments can be retrieved by retracing steps taken during the H matrix computation, from the optimal score back to the zero term.

3. Coarse-grained parallelization

Let us now assume that we apply the algorithm proposed in [1] and estimate that on todays PC's it would be possible to obtain speed of order of 1000 MCUPS. Such speed is still insufficient for searching large databases and thus to obtain higher MCUPS rate we may harness power of a network of workstations. Specifically, in this work, we are interested in parallelizing database search using the Smith-Waterman algorithm on a LAN, containing dedicated workstations.

3.1. Distributed Smith-Waterman algorithm

The proposed distributed approach is a variant of a master-slave approach and consists of a master, an FTP server and a number of workers, connected by a network (Figure 2). The master divides the database sequences into as many buckets as the number of workers and uploads them onto the FTP server, which facilitates

data exchange between the master and the workers. Let us now describe elements found in Figure 2.

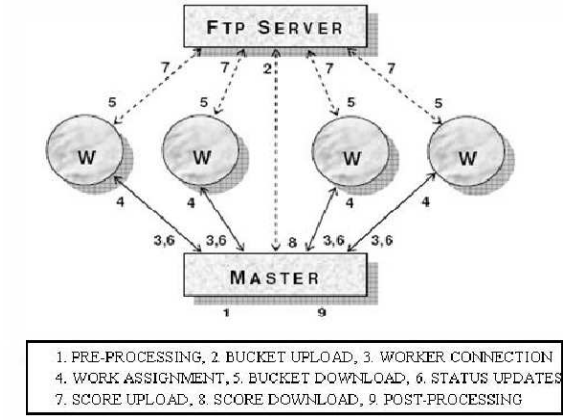

Figure 2. Distributed Smith-Waterman algorithm

1. *Master Process*: is started by furnishing it with the information about genomes (the *query sequence* and the *database sequence* sets) to be compared, the substitution scores and the gap penalty to be used, the URL and path to the FTP server, and the number of workers. It divides the database into equal sized buckets using the bucket workload balancing technique, and uploads them to the FTP server. The query sequence set is uploaded without splitting.

   The *master process* contains three threads: *control*, *reception* and *communication*. The *control thread* is the main interface between the user and the distributed database search system. It responds to user commands, updates the display and keeps track of the amount of work left for the individual workers. The *reception thread* acts as the rendezvous point between the master and the workers. After establishing connection and the initial handshake, a new worker is assigned a particular bucket to search. The *communication thread* is the channel for communication between the master and a worker. Status information is streamed at regular intervals (set by the user) from the worker to the master. The master uses this information to update a running display of work completed, and the speed of the distributed system. Upon completion of work, the control thread downloads the comparison scores from the FTP server and collates the data into a single result file. The collation process is a time consuming task as the scores from the different workers have to be assembled in the original order of the database.

2. *Data Exchange Server*: is used to handle data exchange between the master and workers. Since the master can be expected to be lightly loaded, no matter the number of workers, and the FTP server is loaded only at the start and completion of the search process, both can be run on the same workstation.

3. *Worker Processes*: run on separate workstations. *Worker process* contains two threads: *control* and *computation*. The *control thread* communicates with the master. It gets the search parameters, downloads its bucket file and query sequence, updates the master on the progress of work and uploads results. The *computation thread* executes the Smith-Waterman algorithm.

3.2. Load Balancing

Success of the coarse-grained approach depends on the load balancing strategy. A slightly modified version of the *bucket method* suggested by Yap, Frieder and Martino ([14]) which is a combination of the static allocation *portion* method and dynamic allocation *master-worker* method is utilized here (though in the future we plan to experiment with dynamic allocation methods). Yap defines the percentage of load imbalance (*PLIB*) as the time difference between the fastest finishing and slowest finishing workstations. A good workload balancing technique must have a *PLIB* close to zero.

The *bucket method* proceeds as follows. Sequences in the database are sorted in descending length order. Starting from the longest sequence, each one is placed into a bucket that has the smallest value for the function: $t_b = (\sum n_b)/MCUPS$ where $\sum n_b$ is the current sum of sequence lengths in the bucket, and *MCUPS* is the speed of the workstation to which the bucket is assigned (thus taking into account the heterogeneity of workstations). Because work is divided before computation begins, the number of workstations and their speeds of comparison must be known in advance.

Note that in practice the performance of the load balancing technique depends also on the composition of the sequences in the genome. The number of high scoring sequences in a bucket affects the number of reevaluations performed with the word implementation of the comparison algorithm. It is difficult to determine in advance the sequences that will have a comparison score above the saturation level of the byte implementation, but in general longer sequences have a higher probability of generating larger scores. The bucket method takes this into consideration by first sorting the database sequences in descending order before allocation.

Another potential problem is the degradation of this

scheme due to faulty workstations. If even a single workstation process fails, time for database comparison doubles as the corresponding bucket can only be allocated to another workstation once it completes its work (*PLIB* becomes 50%).

3.3. Implementation Details

Mithral client-server software development kit [15] (providing TCP/IP connections, thread control and file system functions) was used to build the distributed system. The *libcurl* [16] library compiled as a DLL was used to provide file transfer support for the FTP protocol. The code was written in C and assembler, and compiled in Visual C/C++ 6.0 and the Netwide ASeMbler 0.98.08 (NASM) on Windows NT. The system was used for whole genome comparison, where one set of annotated sequences of an organism was compared against another. A metadata file was created, containing the bucket number in which each sequence of that genome was placed. It was used to collate the generated scores into a single file (representing the original order within the genome). Proposed system was equipped with basic facilities to overcome worker failures. Intermittent faults caused by network connectivity problems or heavy load of a worker are detected upon its failure to report status information to the master. Permanent faults caused by the termination of a worker process, result in loss of work done by that worker. The system then reassigns the bucket to a new worker when one becomes available. Based on the results reported in [1], the diagonal method was used as the most efficient and accurate. Since our implementation was run on a network of homogeneous workstations, the pure homogeneous metric was used for the load balancing algorithm.

To quantify the performance of dynamic programming algorithms, the measure: *millions of dynamic programming cell updates per second* (MCUPS) was defined. It represents the number of cells in the H matrix that can be computed per second, and includes all memory operations and corresponding E and F matrix cell evaluations. It is calculated as $(MN)/t_{dp}/10^6$ where t_{dp} is the time (in seconds), to evaluate the entire H matrix and return the optimal score.

3.4. Experimental Results

The experiments were carried out on 64 Windows NT workstations. A separate Windows workstation was used to run the master process and a workstation running Redhat Linux was used to run the FTP server. Each worker node had a single Pentium 4 processor running at 1.4Ghz and 128 Mbytes of RAM. The interconnection network was a switched 100Mbps Ethernet.

The first experiment was the whole genome comparison of the annotated Bacillus Subtilis 168 genome (4100 DNA sequences; $3,650,998$ nucleotides) with the Mycoplasma Genitalium G-37 genome (484 DNA sequences; 528,750 nucleotides). A total of $1,370,339$ (69%) pairwise comparisons produced scores below the byte saturation level, while $614,061$ (31%) pairwise comparisons had to be reevaluated with the word implementation. The second experiment was the whole genome comparison of the annotated Escherichia coli k-12 genome (4405 DNA sequences; $4,130,746$ nucleotides) with the Haemophilus influenzae genome (1739 DNA sequences; 1,610,500 nucleotides). A total of $1,529,612$ (77%) pairwise comparisons produced scores below the byte saturation level, while $454,788$ (23%) pairwise comparisons had to be reevaluated with the word implementation.

Table 1. Experiment 1: Performance of *bucket* load balancing algorithm.

Work-stations	Largest Load (nucleotides)	Smallest Load (nucleotides)	PLIB
2	1825501	1825497	0.000219
4	912759	912744	0.001643
8	456402	456348	0.011832
16	228231	228132	0.043377
32	114120	114003	0.102524

Table 2. Experiment 2: Performance of *bucket* load balancing algorithm.

Work-stations	Largest Load (nucleotides)	Smallest Load (nucleotides)	PLIB
16	100691	100593	0.097327
32	50388	50286	0.202429
48	33630	33516	0.338983
64	25233	25116	0.463679

In Tables 1, 2 we quantify the performance of the *bucket* load balancing technique by displaying the size of the largest and the smallest buckets and the *PLIB* value. As we can see the *bucket* technique turns out to be quite efficient as the difference between the largest and the smallest load is at most about hundred nucleotides (PLIB is close to zero) for all experiments.

Figure 3 shows the speedup of the distributed algorithm. Overall, an almost linear speedup was observed and it can be attributed to the independence of the workers, homogeneity of worker computers and appropriateness of the load balancing technique. In Tables 3 and 4 we show performance of the two exper-

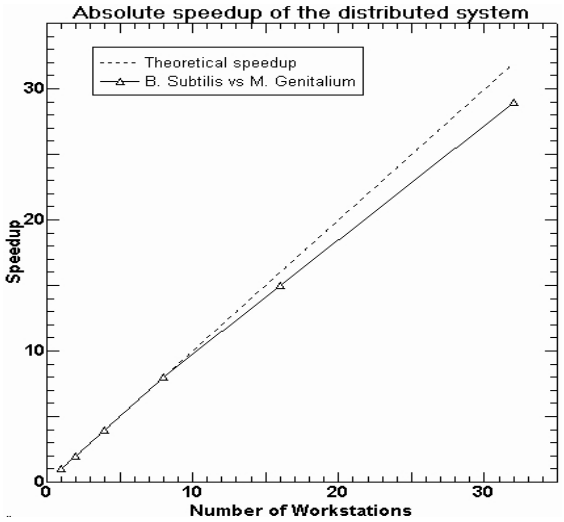

Figure 3. Speedup for the comparison of B. Subtilis versus M. Genitalium.

Work stations	Runtime (sec.)	Max MCUPS	Average MCUPS	MCUPS/ Work station
1	11337	170	170	170
2	5569	350	347	173
4	2842	708	679	170
8	1444	1400	1337	167
16	745	2730	2591	162
32	396	5343	4875	152

Table 3. Experiment 1: performance of comparing Bacilus Subtilis with Mycoplasma Genitalium

iments. Maximum MCUPS represents the maximum instantaneous speed of the system, while the Average MCUPS indicates the average speed of computation. The value of MCUPS / WORKSTATION measures worker efficiency. Experiment 1 produces excellent results for tests performed on up to 32 workstations. Experiment 2 compares genomes that are much larger and is substantially more time consuming. It produces good speeds for up to 32 machines after which there is a significant drop in performance growth. There are many reasons for this effect. First, a large number of workstations connecting simultaneously to the FTP server cause network congestion and overload the FTP server. A simple workaround could be to start groups of workers at different intervals of time. Second, there is a small overhead for splitting the genome into buckets at the beginning of the computation. This is typically in the order of 60 seconds and increases with the number of workstations used. However, the most important

Work stations	Runtime (seconds)	Max MCUPS	Average MCUPS	MCUPS/ Work station
16	2472	2886	2691	168
32	1311	5639	5074	159
48	1358	7090	4899	102
64	1233	8076	5395	84

Table 4. Experiment 2: performance of comparing Haemophilus Influenzae with Escherichia coli

reason for the performance drop is the time taken in collating the output produced by the different workers into a single result file. This time is in the order of up to 300 seconds for 64 workers, and significantly decreases the performance of the distributed system. Overall, maximum speed of 8076 MCUPS was achieved on 64 workstations, a speedup of approximately 2.8 when compared to 16 workstations. Finally, Table 5

Systems	Type	PEs	MCUPS	Cost (k$)
BioSCAN	SP	12992	25000	20
64 P4's	WC	64	8076	68
BISP	SP	256	3200	20
Kestrel	PC	1024	1600	30
DeCypher II-15	RH	1920	1400	173
SAMBA	SP	128	730	60

PC—programmable co-processor,
RH—reconfigurable hardware,
WC—network of workstations,
SP—Special Purpose VLSI.

Table 5. Performance of various hardware platforms according to [2] and our system

compares the performance of genomic comparisons on various hardware. System described in this paper, stands second only to the special purpose hardware solution, BioSCAN. The price-tag of our approach is a misnomer since in most institutions a large number of workstations are already available and can be used with little additional cost. Note also, that our experiments have been performed on a somewhat outdated hardware (which was the only hardware we had easy access to at the time). The performance of the more modern Pentium 4 processors (even these without hyper-threading technology) should allow running the Smith-Waterman algorithm at double the speed reported here.

3.5. Limitations

It is worthwhile discussing some of the limitations and problems that we have encountered. As mentioned before, the FTP server and the interconnecting network must be capable of simultaneously servicing a large number of clients. The FTP server is a bottleneck during the beginning and end of the computation. One of the ways this problem can be alleviated is by starting the clients in different batches, so that the load on the FTP server is reduced. A potential problem is in the availability of a large number of machines as PCs in a general laboratory are used by numerous people at varying times. Strategic time intervals when there is less demand must be targeted. Another practical problem encountered was the failure of the power supply source. Because sequence comparison is a CPU intensive task, distributed computing on a workstation cluster increases the power consumption of the CPUs and can heavily load the power supply.

4. Concluding remarks

The aim of this work was to test the feasibility of using a network of workstations for comparing whole genome sequences. The proposed coarse-parallelization approach directly utilized an earlier developed SIMD approach based on multimedia extensions of modern Intel processors. As a result of combining fine and coarse grain parallelization we observed performance comparable to the fastest implementations of whole genome comparisons on special purpose hardware. Since general-purpose processors with improved performance are constantly being developed, further performance boost can be expected. For instance, the newly introduced Simultaneous Multi-threading (hyper-threading) technology available on Pentium 4 microprocessors, or the multi-core approach to processor design offered by both Intel and AMD offers further potential speed increases. In the near future we plan to proceed in two directions. First, investigate effectiveness of dynamic load balancing strategies when applied to our problem. Second, to port the distributed Smith-Waterman algorithm to the grid. We will report on the results in subsequent reports.

Acknowledgments

We thank the manager of Centre for Technical Support at T. S. Santhanam Computing Centre, Muthu'G., and the laboratory personnel Mr. Ravikumar V. (GA), Mr. Elson Jeeva T. (MIS), Mr. Srinivasan V. and Ms Dharani (ME) for providing equipment required to conduct experiments.

References

[1] A. Jacob, M. Paprzycki, S. Sanyal, Whole Genome Comparison using Commodity Hardware, Technical Report, `http://www.tifr.res.in/~sanyal/papers/techreport_2007_1.pdf`, submitted

[2] R. Hughey, "Parallel hardware for sequence comparison and alignment," Computer Applications in the Biosciences, 12(6): 473–479, 1996

[3] V. Strumpen, "Parallel molecular sequence analysis on workstations in the Internet," Technical report, Department of computer science, University of Zurich, 1993

[4] W. S. Martins, J. B. del Cuvillo, F. J. Useche, K. B. Theobald, G. R. Gao, "A multithreaded parallel implementation of a dynamic programming algorithm for sequence comparison," In Proc. of the Pacific Symposium on Biocomputing, 311–322, 2001

[5] S. B. Needleman, C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two sequences," J. of Molecular Biology, 48(3): 443–453, 1970

[6] P. H. Sellers, "On the theory and computation of evolutionary distances," SIAM J. of Applied Mathematics, 26: 787–793, 1974

[7] T. F. Smith, M. S. Waterman, "Identification of common molecular subsequences," J. of Molecular Biology, 147(1): 195–197, 1981

[8] O. Gotoh, "An improved algorithm for matching biological sequences," J. of Molecular Biology, 162(3): 705–708, 1982

[9] E. W. Myers, W. Miller, "Optimal alignments in linear space," Comp. App. in the Biosciences, 4(1): 11–17, 1988

[10] A. V. Aho, D.S. Hirschberg, J. D. Ullman, "Bounds on the complexity of the longest common subsequence problem," J. of the ACM, 23(1): 1–12, 1976

[11] R. B. Lee, "Multimedia extensions for general-purpose processors," Proc. IEEE Workshop on Signal Processing Systems, 9–23, 1997

[12] A. Peleg, S. Wilkie, U. Weiser, "Intel MMX for multimedia PCs," CACM, 40(1):25–38, 1997

[13] Berkeley Drosophila Genome Project, 2003. Available: `http://www.fruitfly.org/`

[14] T. K. Yap, O. Frieder, R. L. Martino, "Parallel computation in biological sequence analysis," IEEE Trans. on Para. and Dist. Syst., 9(3):283–293, 1998

[15] Mithral cssdk, 2003. Available: `http://www.mithral.com/projects/cosm/`

[16] Libcurl, 2003. Available:`http://curl.sf.net/`

[17] A. Di Blas, D. M. Dahle, M. Diekhans, L. Grate, J. D. Hirschberg, K. Karplus, H. Keller, M. Kendrick, F. J. Mesa-Martinez, D. Pease, E. Rice, A. Schultz, D. Speck, R. Hughey, "The UCSC Kestrel Parallel Processor," IEEE Trans. on Para. and Dist. Syst., 16(1): 80–92, 2005