Hyper-BLAST: A Parallelized BLAST on Cluster System

Hong-Soog Kim, Hae-Jin Kim, and Dong-Soo Han

School of Engineering
Information and Communications University
P.O. Box 77, Yusong, Daejeon 305-600, Korea
{kimkk,hjkim,dshan}@icu.ac.kr

Abstract. BLAST is an important tool in bioinformatics. It has been used to find biologically similar sequences to the given query sequence from the database of the annotated sequences. For high throughput processing of huge number of query sequences, there have been many studies on parallel batch processing of sequence similarity search using BLAST. As the number of sequences in the database increases at exponential rate, the search speed of BLAST itself becomes important. Although NCBI has developed a parallel BLAST using the thread on SMP machines for the speedup of BLAST, the speedup is still limited because the SMP machine has restricted the number of processors due to its architecture. In this paper, we present our parallelized BLAST on cluster systems for further speedup. The main strategy used is the exploitation of the inter-node parallelism, which can be extracted by logical partitioning of the database. For the inter-node parallelism, we have designed and implemented a logical database partitioning method, initiation and coordination of the BLAST on remote node and communication protocol for collecting remote node's result. According to our performance test with 2-way 8 node cluster system, roughly 12 times speedup has been achieved in terms of response time of similarity search for individual query sequence.

1 Introduction

For biologist, finding and annotating characteristics of novel genes and protein sequences is an extremely important mission. The most reliable way to determine a biological molecule's structure and function is direct experimentation. It is, however, much easier to obtain the DNA sequence of the gene corresponding to a messenger RNA or protein than to experimentally determine gene's function of structure. This fact provides a strong motivation for developing computational methods that can infer biological information from sequence alone [1].

BLAST (Basic Local Alignment Search Tool) [2,3,4] is one of the most widely used similarity search tools available to computational biologist. It rapidly identifies statistically significant matches by comparing newly sequenced segments of genetic material or proteins with already annotated nucleotide or amino acid

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2659, pp. 213–222, 2003.

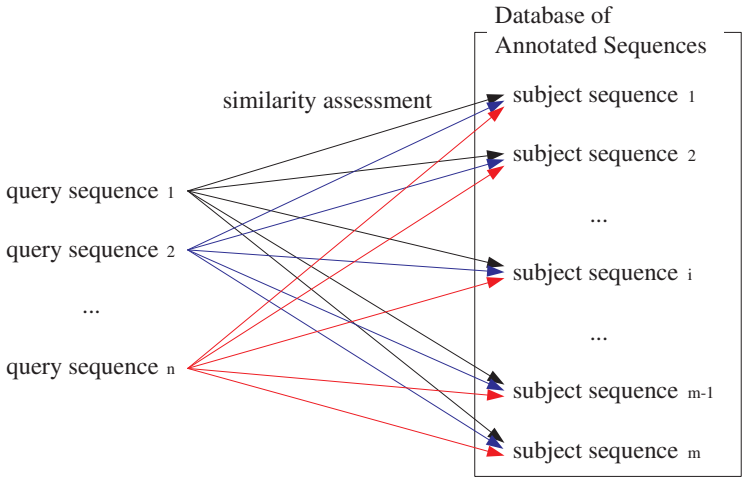

Fig. 1. Similarity search in BLAST

sequences in the database as shown in Fig. 1. This kind of search allows biologists to make inferences on the structure and function of the unknown gene or to screen new sequences for further investigation using more sensitive and computationally expensive methods. The information that BLAST provides in a few hours would otherwise takes months of laboratory work [5].

Similarity search with BLAST would be more useful, if there is more annotated sequences in the database. Because the time complexity of the similarity search algorithm used in BLAST is proportional to the size of the database [2], the response time of the BLAST becomes slow as the number of sequences in the database grows. GenBank, the primary repository for DNA sequence data, continues to grow at an exponential rate. It contains roughly 15,849,921,438 nucleotides in 14,976,310 sequences as of year 2001 [6]. Historically, GenBank has been doubling its size every 18 month, but that rate has accelerated to doubling every 15 months due primarily to the enormous growth in data from expressed sequence tags (ESTs). Keeping pace with the analysis of this data is difficult task for biologists.

As a result, sequence similarity analysis using BLAST is becoming a bottleneck [7]. A huge sequence database can also bring more serious problem. If the size of database exceeds the size of physical memory of the system, it may incur frequent paging while BLAST program scans the database and it could result in serious performance degradation. Therefore, as the size of the database grows, the speedup of BLAST becomes quite important.

Although a parallel version of BLAST, which exploits intra-search level parallelism explained in 3.1, has been developed by NCBI (National Center for Biotechnology Information) at the NIH (National Institute of Health) and Washington University, it is targeted only on SMP (Symmetric Multi-Processor) machines. Because SMP machine has certain limitation on the number of processors

due to its architecture, the speedup of BLAST that can be achieved solely on one SMP machine has some limitation. Hence, the speedup improvement of BLAST on the SMP machine is not sufficient to cope with the current situation where enormous sequences are newly added in the database at exponential rate. In order to use more processors for more speedup of BLAST, we consider PC cluster system as an alternative to SMP machine.

In this paper, we present *Hyper-BLAST*, a parallelized BLAST on cluster system, which can provides scalable speedup in terms of response time. Hyper-BLAST adds inter-node parallel execution techniques to the intra-search level parallelism that is used by NCBI BLAST. Logical partitioning of database is used to prepare and enable the inter-node parallel execution of intra-search level parallelism. In our parallelized BLAST on cluster system, the master node drives remote nodes to search similar sequence from logically partitioned database and collects minimal but complete data for reporting the search results.

2 Related Work

There have been several researches on parallel processing of sequence similarity analysis on multiprocessor systems. Braun et al. studied parallel processing of sequence analysis using BLAST on workstation of cluster [8]. They provided a good classification scheme for parallel processing of BLAST and reported implementation of coarse grained level of parallelism (parallel batch processing of similarity search). They pointed out that the parallel batch processing approach has overhead for maintaining consistency of partitioned or distributed database.

The coarse grained level of parallelism is extracted from the fact that respective similarity searches for different query sequences can be done independently. We call this kind of parallelism *inter-search level parallelism* in order to distinguish the intra-search level parallelism used in NCBI BLAST.

Efforts to parallelize FASTA and BLAST using the coarse grained level of parallelism have been published since the early 1990s [9,10,11]. Recently, Disperse system and BeoBLAST system have been reported. Disperse system parallelized FASTA and SSEARCH (Smith-Waterman) program using Perl and UNIX utilities such as rcp (or scp) and rsh (or ssh) [12]. While Disperse system used UNIX utilities for initiating parallel execution of program on workers and tallying the results from workers, BeoBLAST used queuing system (GNU Queue) for executing BLAST and PSI-BLAST on a Beowulf cluster [13].

There also have been a few commercial products, such as High-Throughput BLAST (HT-BLAST) of SGI [14] and Turbo BLAST of TurboGenomics [15], based on coarse grained level of parallelism.

Albeit parallel batch processing of similarity analysis based on inter-search level parallelism can achieve higher throughput over a set of query sequences, it cannot improve the response time of the similarity search for individual query sequence because the response time of a similarity search itself is dependent on the computational power of each node or processor.

3 Parallelization of BLAST on Cluster System

In this section, we explain intra-search level parallelism that is used in NCBI BLAST, our extension of intra-search level parallelism for cluster system and the other implementation details.

3.1 Intra-search Parallelism in NCBI BLAST

The NCBI BLAST is parallelized using the thread facility for SMP machine. The parallelization of NCBI BLAST is based on task pool. The task is a similarity assessment of the subject sequence in database for given query sequence. In NCBI BLAST, each thread gets its task chunk (i.e., range of subject sequences in database), aligns the query sequence with subject sequences in its chunk, merges its search result into global search result and receives next chunk until all the sequences in database are searched. All threads repeat these steps. As SMP machine has one shared memory, access to subject sequences, assigning the task chunk to thread and collection of slave thread's search result are straightforward using mutex for exclusive access to global data structure.

The parallelism in NCBI BLAST originates from the fact that similarity assessments for every subject sequence in sequence database are mutually independent. We call it *intra-search level parallelism* in order to distinguish it from the inter-search level parallelism that was used in parallel batching processing approach. In terms of the granularity of parallelism, intra-search level parallelism is much finer than inter-search level parallelism.

Even though NCBI BLAST successfully extracted parallelism and achieved some speedup in the similarity search of individual query sequence using intra-search level parallelism, its speedup ratio is still limited by its underling parallel processor system.

3.2 Extension of Intra-search Parallelism in Hyper-BLAST

In Hyper-BLAST, we extend application of intra-search level parallelism from one SMP node to multiple nodes in cluster system by logically partitioning of the sequence database, in which every node calculates its database partition from the node configuration and confines its search space within its own database partition. The intra-search level parallelism is realized in two levels: inter-node and intra-node level.

In intra-node level, master thread of Hyper-BLAST on every node has its logical database partition that is assigned by inter-node level realization of intra-search parallelism. The master thread of Hyper-BLAST on each node allocates a chunk of subject sequence to its slave threads on demand. Because the search result from each node is only partial result, one designated node (master node) collects the search result from the rest of nodes and then makes similarity search report for the given query sequence from the collected results.

For inter-node level realization of intra-search parallelism, we implement mechanisms for specifying computation node set and logical database partition

for each computation node, executing Hyper-BLAST on each computation node and controlling Hyper-BLAST program instances on the nodes. Since a cluster system is distributed memory multiprocessor system, we devise message communication protocol that can replace the read and write operation on shared memory of the SMP machine.

For logical database partitioning, we assume that all computation nodes access sequence database, query sequence and other internal data files for BLAST execution through NFS (Network File System).

Specification of Computation Node Set. For the specification of computation node set and logical database partition, we use a node configuration file that is specified in Hyper-BLAST command line option.

The node configuration file contains information on computation node set, role of each computation node (master/slave node), degree of parallelism (DOP) and the load weight for individual computation node. From the DOP and the load weight for individual node, the logical database partition for a node is calculated. In run-time, the master thread of Hyper-BLAST on computation node confines its search space to the calculated database partition instead of original database. Because NCBI BLAST uses memory mapped file I/O for database for fast access, logical database partitioning can reduce the possibility of paging if the whole sequence database is too large to fit into physical memory of a computation node. Paging is known to cause serious overhead in running application that accesses a large set of data.

Initiation of Hyper-BLAST on Slave Nodes. As multiple Hyper-BLAST process instances should be executed in different nodes of cluster system, it is necessary to explicitly execute Hyper-BLAST program on remote nodes. For the purpose of initiating Hyper-BLAST on remote nodes, we use rsh (remote shell) based approach. Except for the master node, initiation of the Hyper-BLAST program is performed using the rsh command in the below:

```
rsh node01 'hyperblastall -i sample.seq -o sample.html -p blastx
-d nr -e 10.0 -m 0 -b 15 -v 15 -I T -T T -c cluster_nodes.cfg' &
```

In the above rsh command for executing Hyper-BLAST on a remote node, all command line options are the same as those of NCBI BLAST except -c option added for specifying the node configuration file.

Since it could be a chore to make this kind of rsh command and node configuration file, we also have developed the GUI-based Hyper-BLAST execution environment (HBEE) program. In HBEE, users can check the node status and resources such as number of processors, processor types and physical memory size. While examining a node, users can add it to computation node set of Hyper-BLAST, designate it as master node or slave node, and specify the number of parallel processors used for Hyper-BLAST and its load weight. From the user's definition of computation node set, the HBEE generates node configuration file

and series of rsh command for executing Hyper-BLAST on remote nodes that are designated as slave nodes in the node configuration file.

Control of Hyper-BLAST Program on Slave Nodes. Once Hyper-BLAST programs start on a master node and remote slave nodes, the master thread of the Hyper-BLAST program at every computation node first identifies its role as master or slave node and computes its database partition using the information in the node configuration file. Then, every Hyper-BLAST program carries out similarity search for the first query sequence in query sequence file that is specified by the command line option -i.

For nodes that have the DOP value of two or more, the master thread of the Hyper-BLAST program creates or reactivates as many slave threads as the DOP value. The master thread assigns the chunk of sequences to its slave threads on demand while the slave threads initiate the similarity search for the query sequence from its assigned chunk of sequences.

The final similarity search result at each node is kept within in-memory data structure. The in-memory data structure is a linked list of the sequence alignment information between the query sequence and the similar sequence found in the sequence database. Within the linked list of the sequence alignment information, the sequence alignment information is sorted by its rank order.

When the Hyper-BLAST programs at the slave nodes finish their respective similarity search and make a linked list of sequence alignment information, the respective master threads of Hyper-BLAST at the slave nodes make a message that corresponds to the sequence alignment information in the linked list, send the message to the master node and wait for the acknowledgment of the message. The master thread of Hyper-BLAST at the master node makes sequence alignment information from the message that is received from the slave node, insert it into the linked list of sequence alignment information and send acknowledgment to the slave node. If the sequence alignment information, which is made from the message from the slave node, is less than the least sequence alignment information in the linked list of the master node in terms of rank order, the sequence alignment information is discarded. The acknowledgment for discarded sequence alignment information makes the slave stop to send next data message.

When the master node receives the search results from all the slave nodes, the linked list of sequence alignment information in the master node becomes a complete similarity search result for the first query sequence in the query sequence file. Then, the master thread of Hyper-BLAST makes the report for the similarity search result and prints it. After printing the report, the master thread of Hyper-BLAST program at the master node sends the control message that orders slave nodes to start similarity search for the next query sequence in the query sequence file.

Figure 2 depicts the run time behavior of Hyper-BLAST program instances on the master node and the slave node. The master thread of Hyper-BLAST determines each node's logical database partition, controls the parallel search activity and communication for search results, and makes similarity search re-

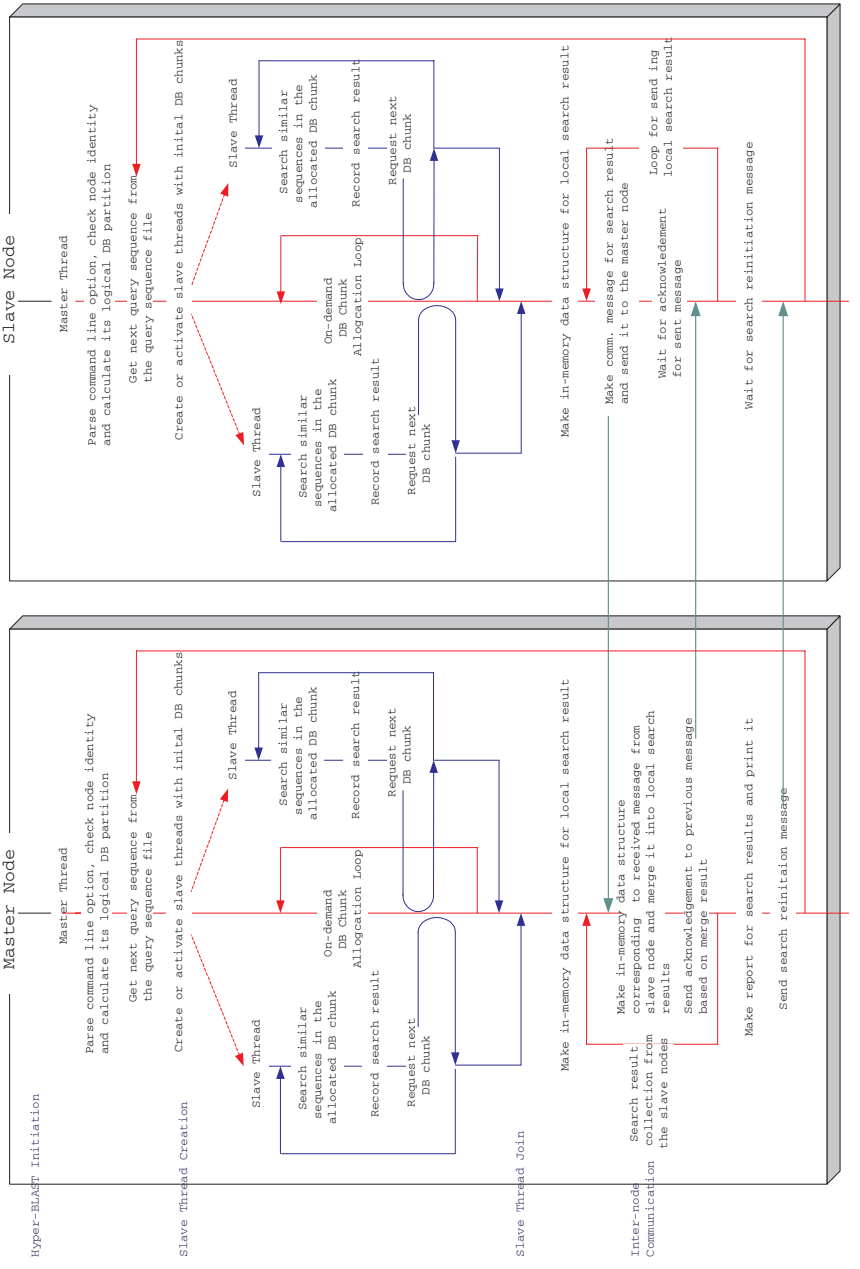

Fig. 2. Control flow of Hyper-BLAST program instances at computation nodes

port. The slave threads of Hyper-BLAST, which actually do the similarity search for given chunk of subject sequences within the logical database partition of the computation node.

4 Performance Evaluation

For the performance evaluation of Hyper-BLAST, we measured the execution time and calculated speedup on actual cluster system. The detailed specifications of cluster system and other experiment conditions are summarized in Table 1.

In our test cluster system, the local disk of the computation node is used for operating system and Hyper-BLAST program and data files are placed in the remote RAID server. The query sequences for testing are obtained from the GI 2054475 sequence by chopping the first n letters. The size of the query sequence is varied from 1000 bp to 5000 bp by 500 bp increment. Because Hyper-BLAST program, data file and sequence database are placed in the remote RAID server and all nodes access them through the NFS, the execution time can be affected by NFS cache, disk cache and memory cache. However disabling these effects for measurement is not simple. Hence, for the fair comparison of the response time, we have measured the response time twice and selected the second result for the measurements. The first execution of the Hyper-BLAST is for indirect enabling of the possible cache benefits.

Figure 3 compares the speedup ratio for query sequences of which length varies from 1000bp to 5000bp by 500bp increment.

As shown in Fig. 3, the speedup is proportional to the size of query sequence but it continuously increases as more processors are used. The maximum speedup using 16 processors in the cluster system ranges between 10.96 and 12.42. From the experimental results, we can conclude that our implementation of parallelized BLAST on cluster system gives scalable speedup as more processors are used for the similarity search.

Table 1. Specification of cluster system

	8 Node Cluster
Processors	Dual Intel Pentium III 1GHz per Node
Memory	1024KB per Node
Network	100Mbps Switch
O.S	Linux (Kernel version 2.4.18)
BLAST	Hyper-BLAST based on NCBI BLAST Version 2.2.1 blastx
BLAST DB	nr (929,420 sequences; 291,584,220 total letters)
Query sequence	GI 20544475 Homo sapiens adenylate cyclase 2 (brain) (ADCY2) (total length: 6551 letters)

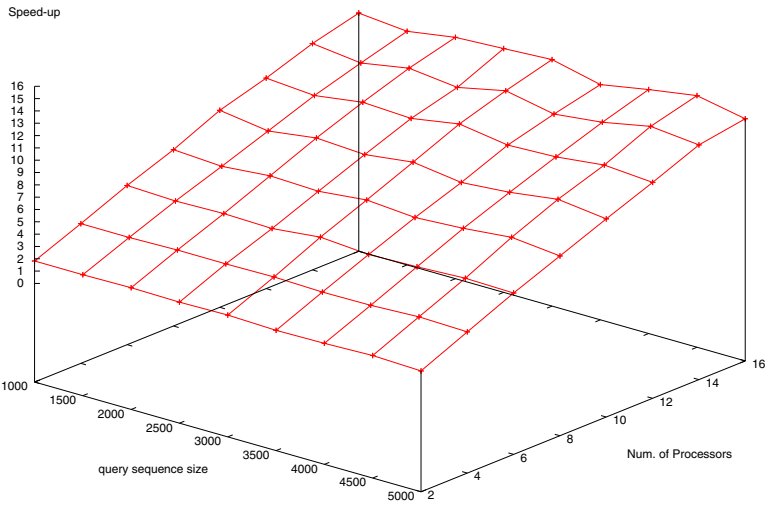

Fig. 3. Speedup comparison on 2-way 8 node cluster system

5 Conclusion and Future Work

BLAST is an important tool in bioinformatics. The exponential growth rate of the sequence database requires faster BLAST. Hence, the speedup provided by parallel NCBI BLAST on SMP machine becomes insufficient. To get more speedup of NCBI BLAST, we have designed and implemented Hyper-BLAST on cluster systems. Using logical partitioning of sequence database, Hyper-BLAST extends the intra-search level parallelism from one SMP machine to multiple computation nodes that are connected by network. A communication protocol and a message format is devised to control of Hyper-BLAST programs on slave nodes and collect final search results.

This extension of intra-search parallelism enables us to use more processors for similarity search using BLAST and gives more speedup of individual query sequence search. The performance evaluation result shows that Hyper-BLAST gives scalable speedup in terms of response time as more processors are used.

Currently, we are studying the performance of Hyper-BLAST on large-scale cluster systems. We anticipate that speedup might be saturated at some number of multiple processors. Hence, future work includes identification of speedup saturation point, modeling of speedup function and devising of combined approach that uses parallel batch processing technique on the top of Hyper-BLAST for more speedup and throughput.

References

1. Durbin, R., Eddy, S., Krogh, A., Mitchison, G., eds.: Biological sequence analysis: Probabilistic models of proteins and nucleic acids. Cambridge University Press (1998)
2. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic Local Alignment Search Tool. Journal of Molecular Biology **215** (1990) 403–410
3. Altschul, S., Gish, W.: Local alignment statistics. Methods in Enzymology **266** (1996) 460–480
4. Altschul, S., Madden, T., Schaffer, A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.: Gapped BLAST and PSI–BLAST: A new generation of protein katabase search programs. Nucleic Acids Research **25** (1997) 3389–3402
5. Gish, W., States, D.: Identification of protein coding regions by database similarity search. Nature Genetics **3** (1993) 266–272
6. NCBI: Growth of GenBank. Technical report, National Center for Biotechnology Information (March 12, 2002)
7. Chi, E.H., Shoop, E., Carlis, J., Retzel, E., Ried, J.: Efficiency of shared-memory multiporceossors for a genetic sequence similiarity search algorithm. Technical report, Computer Science Dept., University of Minnesota (1997)
8. Braun, R.C., Pedretti, K.T., Casavant, T.L., Scheetz, T.E., Birkett, C.L., Roberts, C.A.: Parallelization of local BLAST service on workstation clusters. Future Generation Computer Systems **17** (2001) 745–754
9. Miller, P.L., Nadkarni, P.M., Carriero, N.M.: Parallel computation and FASTA: Confronting the problem of parallel database search for a fast sequence comparison algorithm. Bioinformatics (formerly CABIOS) **7** (1991) 71–78
10. Barton, G.J.: Scanning protein sequence databanks using a distributed processing workstation network. Bioinformatics (formerly CABIOS) **7** (1991) 85–88
11. Julich, A.: Implementations of BLAST for parallel computers. Bioinformatics (formerly CABIOS) **11** (1995) 3–6
12. Clifford, R., Mackey, A.J.: Disperse: A simple and efficient approach to parallel database searching. Bioinformatics **16** (2000) 564–565
13. Grant, J.D., Dunbrack, R.L., Manion, F.J., Ochs, M.F.: BeoBLAST: Distributed BLAST and PSI-BLAST on a Beowulf cluster. Bioinformatics **18** (2002) 765–766
14. Camp, N., Cofer, H., Gomperts, R.: High-Throughput BLAST. Technical report, Silicon Graphics, Inc. (1998)
15. Bjorson, R.D., Sherman, A.H., Weston, S.B., Willard, N., Wing, J.: TurboBLAST: A parallel implementation of BLAST based on the TurboHub process integration architecture. Technical report, TruboGenomics, Inc. (2002)