Communication Protocols and Message Formats for BLAST Parallelization on Cluster Systems

Hong-Soog Kim
kimkk@icu.ac.kr

Woo-Hyuk Jang
torajim@icu.ac.kr

Dong-Soo Han
dshan@icu.ac.kr

Information and Communications University,
103-6 Munji-Dong, Yusong-Gu, Daejeon 305-714, Korea

December 28, 2007

Abstract

With the widespread use of BLAST, many parallel versions of BLAST on cluster systems are announced, but little work has been done for the parallel execution in the search for individual query sequence on BLAST on cluster systems. Since we can improve not only throughput but also response time, the techniques for parallel execution of BLAST on cluster systems in the search for individual query sequence deserve to be developed. This paper develops communication protocols and message formats to reduce the communication overheads for the parallel execution of BLAST in the search for individual query sequence on cluster systems. The developed communication protocols and message formats are implemented on a new version of BLAST on cluster systems. The new version of BLAST is named *Hyper-BLAST* in this paper. In this paper, we also measured the throughput and response time of *Hyper-BLAST* on various cluster systems. It turned out that considerable performance improvement of BLAST on cluster systems can be achieved through parallel execution in the search for individual query sequence on small or middle-sized cluster systems. On 1-way 64-node system, *Hyper-BLAST* achieved scalable speedup up to 63 processors for 1000-5000 length query size.

1 Introduction

BLAST (Basic Local Alignment Search Tool) [1, 2, 3] is one of the most widely used similarity search tools for computational biologist. It identifies statistically significant matches between newly sequenced segments of genetic material or proteins and databases of known nucleotide or amino acid sequences.

For the parallel execution of BLAST for a query sequence on a cluster system, there are two issues we need to address in large. The first one is how to divide the task of searching a query sequence into subtasks and assign the subtasks to the nodes of the cluster system. The second one is how to reduce the communication overhead incurred when collecting subtask's search results from each node into a master node. We address the first issue by partitioning BLAST DB and assigning the partitioned BLAST DB into the nodes of a cluster system. The subtask for each node only searches for the assigned BLAST DB and returns its search results to the master node.

Dividing a task into subtasks and assigning the divided subtasks into the nodes of a cluster system for parallel execution certainly enhance the possibility of improving the entire search performance. However this inevitably incurs communication overheads between client nodes and a master node. If the performance gains obtained

978-0-7695-3096-3/08 $25.00 © 2008 IEEE
DOI 10.1109/WAINA.2008.238

IEEE computer society

by parallel execution cannot outweigh the inevitably incurred communication overheads we cannot expect drastic performance improvement from the parallel execution of BLAST. Thus minimizing the communication overheads is one of key issues in the parallel execution of BLAST on cluster systems.

In this paper, we briefly explain a BLAST task assignment technique for cluster systems and then we focus on the second communication overhead reduction issue. That is, the primary purpose of this paper is to develop communication protocols and message formats to reduce the communication overheads for the parallel execution of BLAST in the search for individual query sequence on cluster systems. Communication protocols for low level socket communications are developed and in-memory structure of BLAST is analyzed for the design of message formats.

The developed communication protocols and message formats are implemented on a new version of BLAST on cluster systems. The new version of BLAST is named *Hyper-BLAST* in this paper. In this paper, we also measured throughput and response time of Hyper-BLAST. It turned out that considerable performance improvement of BLAST on cluster system can be achieved through parallel execution of the search for individual query sequence on small or middle-sized cluster systems.

The performance evaluations of *Hyper-BLAST* on 1-way 8-node cluster system, 2-way 8-node cluster system and 1-way 64-node system show that *Hyper-BLAST* achieves scalable speedup up to 30-40 processors. On 1-way 8-node cluster system, maximum speedup 7.15 is achieved. On 2-way 8-node system, 12.42 times maximum speedup is achieved. Finally, 30.64 maximum speedup is achieved on 1-way 64-node cluster system.

The rest of the paper is organized as follows: In Section 2, the highlight of the paper, presents communication protocols and message formats to reduce the communication overheads between a master node and computation nodes. In section 3, the performance evaluation of the proposed parallelized BLAST (*Hyper-BLAST*) on cluster systems are performed in terms of execution time and speedup. We draw conclusions in section 4.

2 Communication Protocol and Message Format

Hyper-BLAST invokes processes on remote node that search similar sequence(s) for the given query sequence within own partitioned sub-database. The local search results are delivered to the master node in the form of messages. This inevitably incur extra communication overheads. If the performance gain through parallel execution cannot outweigh the extra communication overheads, we cannot expect drastic performance improvement from the parallel execution of BLAST on cluster systems. Thus minimizing the communication overheads is one of key issues in the parallel execution of BLAST on cluster systems.

For the building of communication protocols and message formats among the master and computing nodes, we need to understand in-memory data structure of BLAST because more efficient form of the data and control messages are created based on in-memory data structure. In this section, in-memory data structures for sequence similarity search results and their message formats for communications are discussed.

2.1 Communication Message Format for Hyper-BLAST

The communication message used in *Hyper-BLAST* is designed to minimize the communication between the master node and slave nodes. For this purpose, the replicated information in the in-memory data structure is removed in the message and the whole information of *SeqAlign*s of the slave node is divided into related *SeqAlign*s messages in order to prevent the slave from sending useless *SeqAlign* information.

There are two types of messages for communication. One is used for sending *SeqAlign* data from the salve node to the master node and receiving the acknowledgment for the previously sent *SeqAlign* data from the master to the slave node. Another is used for sending the statistical data and its acknowledgment of the previous request.

2.1.1 Message Format for SeqAlign Data

The overall structure of message format for sending one or more *SeqAlign* nodes consists of header section and one or more HSP_SET records section. The header section has three fields; LEN, $CTRL$ and N field. The LEN field indicate the length of the current message and this field value is used for checking the length of message at master node. Because the message can be fragmented into one or more message segments due to network hardware MTU (maximum transmission unit), the receiver (master node) cannot read the whole message at once. The LEN field is used for assembling the whole message from the fragmented message segments.

The $CTRL$ field is used for controlling of the message transmission. For the message from the slave node to the master node, the $CTRL$ field indicates the type of message (*SeqAlign* data or statistical data) as well as various information such as no search result indication and last/non-last message indication. For the message from the master node to the slave node, the $CTRL$ field is used for the acknowledgment. The message from the master node to the slave node has header section only. For example, when the *SeqAlign* data from the slave has larger *E-value* than the least of the *SeqAlign* collected in the master node, the next *SeqAlign* data from the same slave node is useless. In that case, the CTRL filed is set by the value indicating that no more data is needed from that slave and the slave quit sending the *SeqAlign* data.

The N filed indicates the number of HSP_SET records in the current message. One HSP_SET record keep a *SeqAlign* data for an alignment between the given query sequence and one specific subject sequence. The number of HSP_SET record per message can be varied because the size of the information for one alignment is varied with the alignment results. In addition to, the group of *HSP_MULTI type SeqAling* nodes should be sent within single communication message because they should be treated as one unit. The current maximum size for one message is 4096 bytes but this value can be increased if necessary.

A HSP_SET record section presents an alignment for the query sequence and the specific subject sequence. There are two types (*HSP_UNI* and *HSP_MULTI*) of HSP_SET record for different in-memory data structures.

In *HSP_UNI* and *HSP_MULTI* HSP_SET records, the TAG field indicates the type of the current record. Since all score related information and strand information of one alignment is the same over its constituent segments, the message format does not replicate those information and reduces the size of the message. The N_SEG field at the score section indicates the number of segments in the alignment and location section contains the location information for the segments in the alignment. In location section of the given alignment type, Q_FROM, Q_TO, S_FROM and S_TO field indicate the start position of the segment in the query sequence, the end position of the segment in the query sequence, the start position of the segment in the subject sequence and the end position of the segment in the subject sequence.

2.1.2 Message Format for Statistical Data

The statistical message record holds information to use in the calculation of the statistical data for the final search report. The message for the statistical data has fixed format. The message is composed of a header section and a statistical record section. The header section has LEN and $CTRL$ field and the usage of these field is the same as that of the message for *SeqAlign* data. The statistical record section has twelve fields for various statistical data during BLAST search in the slave node.

For statistical data, the master node simply adds value received from the slave nodes into the current value of the field. The message for statistical data is the last message from the slave to master node. When the master node receives the statistical message from all slave nodes, it sends the synchronization message to all salve nodes and then master node and slave nodes restart the similarity search for the next query sequence.

In this section, we explained how to implement the scheme for parallelized BLAST on cluster systems, the requirements for parallelization, communication protocol for controlling the search activity, communication message for collecting search results, etc. We tried to minimize the communication overhead by eliminating the redundant information in the in-memory data structure, keeping the auxiliary data structure for fast access to the group of *SeqAlign*s and eliminating unnecessary message transmission from the slave nodes. Next section presents the performance evaluation of *Hyper-BLAST* on cluster systems.

3 Performance Evaluation

This section presents performance evaluation results of *Hyper-BLAST* on 2-way 8-node and 1-way 64-node cluster systems. Execution time and speedup of *Hyper-BLAST* are measured for the performance evaluation.

3.1 Performance on 1-way 64-node Cluster System

We performed performance evaluation for *Hyper-BLAST* on 1-way 64-node system to manifest the effectiveness of our novel task assignment scheme to computation nodes of cluster system. The maximum speedup, 30.64 is achieved on 1-way 64-node cluster system with 63 processors.

3.1.1 Execution Time Comparison

Figure 1 shows the execution time for query sequences with length from 1000bp to 5000bp by 1000bp increment.

The execution time constantly decreases as the more nodes are engaged in the search. But the performance is dependent on the size of query sequence and the performance gain is marginal when the number of nodes reaches around 40. Therefore, the number of nodes used for *Hyper-BLAST* must be determined considering query sequence size.

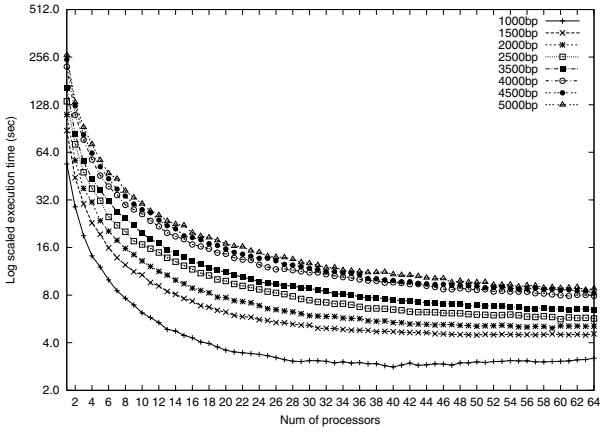

Figure 1: Execution time comparison for large size query sequences on 1-way 64-node cluster system.

3.1.2 Speedup Comparison

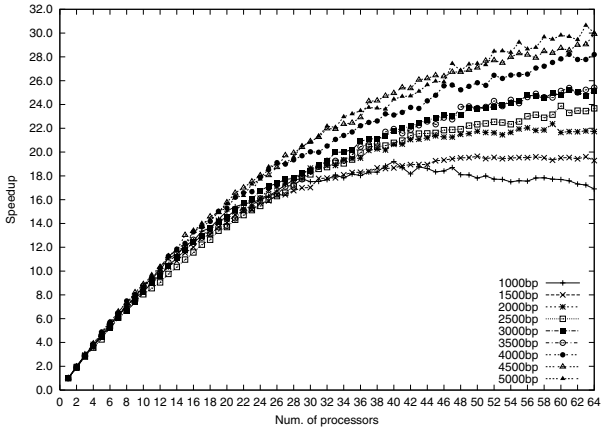

Figure 2: Speedup comparison for large size query sequences on 1-way 64-node cluster system.

Figure 2 is speedup comparisons on the 1-way 64-node cluster system. Figure 2 compares speedups for query sequences with length from 1000bp to 5000bp by 1000bp increment. The maximum speedup increases as the query sequence size increases with higher DOP. This fact indicates that speedup of *Hyper-BLAST* is scalable in terms of query sequence size.

As expected, the speedup of *Hyper-BLAST* saturates at some number of multiprocessors on cluster systems. The cluster system with large number of nodes can be efficiently used for BLAST if our approach for reducing response time and PBS based batch processing for high throughput are combined. In such a system, it is desirable that the computation nodes are grouped into computation node groups. The number of computation nodes in a group is the number of nodes at which *Hyper-BLAST* can achieve maximum speedup. The query sequences are distributed across the computation node groups using PBS based batch processing facility and then the similarity search for a given query sequence on each computation node group can be done with *Hyper-BLAST*.

3.2 Performance on 2-way 8-Node Cluster System

3.2.1 Execution Time Comparison

Figure 3 is execution time comparison on 2-way 8-node configuration. In all execution time comparison, the execution time is plotted in log scale of base 2 in order to amplify the minute changes at higher DOP.

Figure 3 is execution time comparison for query sequences of which length varies from 1000bp to 5000bp by 1000bp increment.

From all experiments, we can observe that the larger length of query sequence is, the more execution time the similarity search requires. Also we can identify that execution time constantly decreases as DOP increases. The execution is proportion to length of query sequence and reverse proportion to DOP. One noticeable thing is constant decrease in execution time in any query sequence length.

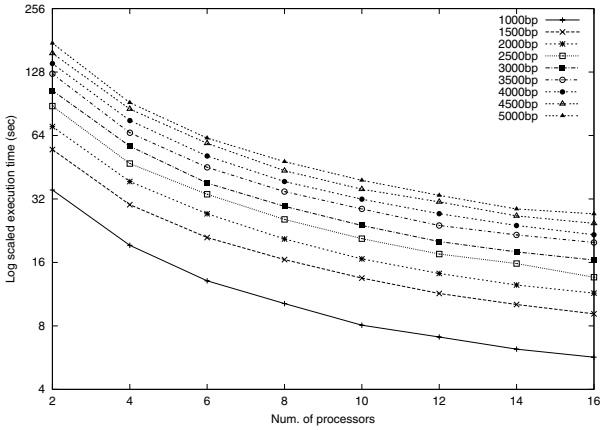

Figure 3: Execution time comparison for large size query sequences on 2-way 8-node cluster system.

3.2.2 Speedup Comparison

The 2-way 8-node cluster system can be configured in various ways with same number of processors. For example, 2-way 4-node configuration and 1-way 8-node configuration that uses 8 processors in the system. The effects of different node configuration with same number of processors are examined. In most cases, the 1-way n-node configurations give similar or higher speedup than the 2-way $n/2$-node cluster configurations, where n is 2, 4, 6 or 8. This means that parallelization used in *Hyper-BLAST* is superior than that used in NCBI BLAST and *Hyper-BLAST* gives cost-effective parallelization.

In terms of cost, 2-way node configuration is cost-effective than 1-way node configuration since dual processor node can be built with an additional processors while the cost of two 1-way nodes is two times that of the single 1-way node. Hence the 2-way node cluster system can be most cost-effective configuration for *Hyper-BLAST*.

4 Conclusion

In this paper, we developed communication protocols and message formats to reduce the extra communication overheads of the parallel execution for single query sequence on BLAST on cluster systems. The developed communication protocols and message formats were implemented in *Hyper-BLAST* and the performance of *Hyper-BLAST* was measured. It turned out that considerable performance improvement can be achieved by parallel execution in the search for single query sequence on BLAST on cluster systems.

The approach of *Hyper-BLAST*, in which the search for single query sequence is performed in parallel, is attractive because it can improve not only the throughput of BLAST but also the response time of individual query sequence. However when we consider the performance improvement of *Hyper-BLAST* is saturated at around 30-40 nodes of cluster systems, the approach of *Hyper-BLAST* should be adapted for the application of large-scale cluster systems.

Currently, we are studying incorporating Hyper-BLAST with PBS based batch processing facilities for large-scale cluster systems. The study includes modeling of speedup function of BLAST for the identification of speedup saturation point and devising parallel batch processing technique on the top of Hyper-BLAST for more speedup and throughput.

5 Acknowledgement

This research was financially supported by the Minstry of Commerce, Industry and Energy (MOCIE) and Korea Industrial Technology Foundation (KOTEF) through the Human Resource Training Project for Regional Innovation.

References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[2] S.F. Altschul and W. Gish. Local alignment statistics. *Methods in Enzymology*, 266:460–480, 1996.

[3] S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

[4] G. J. Barton. Scanning protein sequence databanks using a distributed processing workstation network. *Bioinformatics (formerly CABIOS)*, 7:85–88, 1991.

[5] R. D. Bjorson, A. H. Sherman, S. B. Weston, N. Willard, and J. Wing. TurboBLAST: A parallel implementation of BLAST based on the TurboHub process integration architecture. Technical report, TruboGenomics, Inc., 2002.

[6] R. C. Braun, K. T. Pedretti, T. L. Casavant, T. E. Scheetz, C. L. Birkett, and C. A. Roberts. Parallelization of local BLAST service on workstation clusters. *Future Generation Computer Systems*, 17(6):745–754, April 2001.

[7] N. Camp, H. Cofer, and R. Gomperts. High-Throughput BLAST. Technical report, Silicon Graphics, Inc., 1998.

[8] E. H. Chi, E. Shoop, J. Carlis, E. Retzel, and J. Ried. Efficiency of shared-memory multiprocessors for a genetic sequence similarity search algorithm. Technical report, Computer Science Dept., University of Minnesota, 1997.

[9] R. Clifford and A. J. Mackey. Disperse: A simple and efficient approach to parallel database searching. *Bioinformatics*, 16(6):564–565, 2000.

[10] Compugen. The BioXL/HTM system. Technical report, Compugen, 2002.

[11] NCBI. Growth of GenBank. Technical report, National Center for Biotechnology Information, March 12, 2002.

[12] J. D. Grant, R. L. Dunbrack, F. J. Manion, and M. F. Ochs. BeoBLAST: Distributed BLAST and PSI-BLAST on a Beowulf cluster. *Bioinformatics*, 18(5):765–766, 2002.

[13] Kai Hwang and Zhiwei Xu. *Scalable Parallel Computing*, chapter 12 Parallel Paradigms and Programming Model. McGraw-Hill Companies, Inc., 1998.

[14] Richard Hughey. Parallel hardware for sequence comparison and alignment. *Bioinformatics (formerly CABIOS)*, 12(6):473–479, 1996.

[15] A. Julich. Implementations of BLAST for parallel computers. *Bioinformatics (formerly CABIOS)*, 11(1):3–6, 1995.

[16] P. L. Miller, P. M. Nadkarni, and N. M. Carriero. Parallel computation and FASTA: Confronting the problem of parallel database search for a fast sequence comparison algorithm. *Bioinformatics (formerly CABIOS)*, 7:71–78, 1991.

[17] Paracel. GeneMatcher2 system. Technical report, Paracel, 2002. Available at http://www.paracel.com/products/gm2.html.

[18] TimeLogic. Adaptable hardware accelerated systems for bioinformatics. Technical report, TimeLogic, 2002.