paraBLAST: A Highly Scalable Parallelized BLAST Solution

YUTAO QI and KIN KEONG WONG
Bioinformatics Research Centre, School of Computer Engineering
Nanyang Technological University,
Nanyang Avenue,
SINGAPORE

Abstract: - Programs of the NCBI BLAST family have been widely used for retrieving homologous sequences from existing databases. This article briefly introduces and evaluates a parallelized version of the BLAST algorithm, paraBLAST, using Message Passing Interface (MPI) on a multi-node compute cluster. A dynamical database fragmentation scheme based on the availability of a compute cluster is proposed. Its application in querying nucleotide sequences against large-scale sequence databases is evaluated with different numbers of database fragments. As the tasks are made independent of each other, a highly scalable solution is achieved.

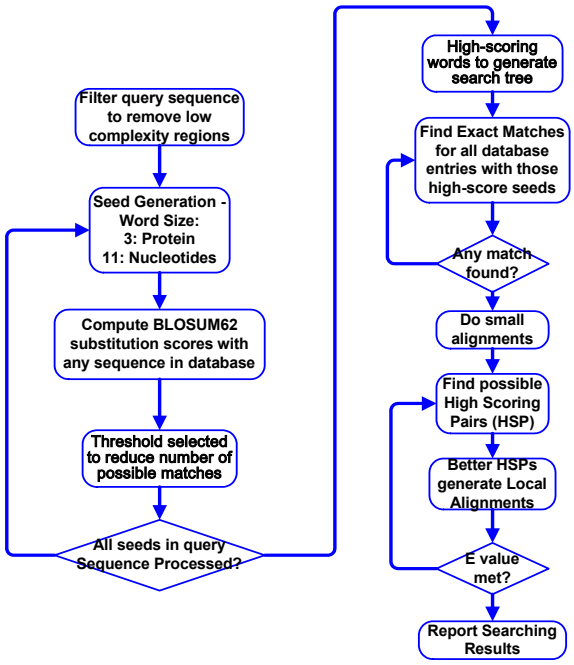
Key-Words: Computational biology, BLAST, Sequence searching, Parallel computing, High performance computing, MPI

1 Introduction

The explosive growth in the sizes of sequence databases over the past several years has overwhelmed the standard version of NCBI BLAST [1, 2, 3], which yields the urgent demand of the development of a more efficient version of BLAST. This naturally leads the researchers to the world of highly scalable parallel and distributed computing [6, 7].

At Bioinformatics Research Centre (BIRC), Nanyang Technology University, we have set up a multi-node compute cluster for high-performance computing. A variety of software of industry standard, such as MPI for parallel applications, has been installed. MPI allows processes in a parallel program to exchange information with each other [6].

BLAST is a set of similarity search programs designed to explore all of the available sequence databases regardless of whether the query is protein or DNA [1]. The BLAST programs have been designed for speed, with a minimal sacrifice of sensitivity to distant sequence relationships. The scores assigned in a BLAST search have a well-defined statistical interpretation, making real matches easier to distinguish from random background hits [1, 2, 3]. BLAST uses a heuristic algorithm that seeks local as opposed to global alignments and is therefore able to detect relationships among sequences that share only isolated regions of similarity. The basic BLAST [1] procedure is illustrated in Flow Chart 1.



Flow Chart 1. Basic BLAST Algorithm Description

A typical BLAST job assesses the similarities between each of input query sequences and all of the sequences in one or more large sequence databases.

2 Methods and Implementation

Our analyses indicate that some BLAST processes are highly scalable, for example, from

seed generation to finding better HSP. To speed up computation, the proposed paraBLAST processes all the tasks in parallel and integrates the results in a unified output. paraBLAST includes one Master process and several Slave process. The Master oversees the entire program execution including the File Provider, an application that manages the storage, versioning, and distribution of the sequence databases. The Slaves perform all of the computations for paraBLAST. All of the components of paraBLAST make use of MPI that manages processor scheduling and inter-processor communication.

Based on the above, we implemented a parallel BLAST called paraBLAST, using MPI. paraBLAST segments the BLAST database and distributes it into cluster nodes, and accordingly, it executes BLAST queries on multiple nodes simultaneously, resulting in a speedup for large-scale database queries. Speedup is achieved mainly through the development of the following techniques:

- **Task Preparation**: Based on the current availability of processors and memory spaces in each compute node, we dynamically convert the original sequence database into virtually segmented databases.
- **Task Creation**: We divide the original query task into a number of smaller tasks in which the similarity of one or several query sequences is assessed against a portion of the sequence databases. The modest-sized segment of database for an individual task fits comfortably into the physical memory available to a single processor.
- **Task Execution**: The tasks are made independent of each other, each available processor can run its own copy of BLAST queries, reducing interference among processors and leading to a highly scalable solution.
- **Result Integration**: Finally, we combine the individual task results into a unified output that matches the search result of the original BLAST. This is done by the paraBLAST Master process upon the accomplishments of all the segmented BLAST searches.

paraBLAST includes one Master process and several Slave processes. The Master oversees the entire program execution including a File Provider, an application that manages the fragmentation, storage and distribution of the sequence databases. The Slaves perform all of the computation for paraBLAST. All of the components of paraBLAST make use of MPI routines that manage processor scheduling and inter-processor communication.

Two different database fragmentation schemes are allowed. A simple way follows the requirement of the user to segment the database to the desired number accordingly. A more sophisticated one is based on the available resources of the used compute cluster. In particular, if the database to be queried can be fitted into the main memory, the searching time [8] will be greatly reduced. Based on a series of experiments (refer to Fig.1 and 2), we determine the size of database fragment using the following algorithm:

```
if (User_Defined_No_of_Fragments) {
        Size_Of_Fragments = Database_Size /
        Number_Of_Fragments + 1;

        /* Here add 1 to make sure that the
        database can be divided into
        Number_Of_Fragments partitions. */

        Number_Of_Fragments =ceil
        (Database_Size / Size_Of_Fragments);
} else if (Nucleotide_Sequence_Database) {
        if (Database_Size<=700.0Mb) {
                Number_Of_Fragments = 2;
                /* In order to use paraBLAST
        */
                Size_Of_Fragments =
        Database_Size / Number_Of_Fragments
        + 1;
        } else {
                Size_Of_Fragments = 700;
                Number_Of_Fragments =
        ceil (Database_Size /
Size_Of_Fragments);
        }
} else {
        …
        /* Only Fragment_Size is different from
        the NS one, identical implementation. */
}
```

3 Results

We executed the programs to evaluate the performance improvement of paraBLAST by querying an arbitrarily selected nucleotide sequence on a large nucleotide sequence database [3, 9] as shown in Table 1.

Table 1: Experimental nucleotide sequence data

	Database for Testing
Name	Nt
Description	All Non-redundant GenBank [4] + EMBL + DDBJ + PDB [5] sequences (but no EST, STS, GSS, or HTGS sequences)
Physical Size	8,187,091,720 bytes (unformatted)
Biological Size	1,682,174 sequences; 7,887,316,358 total letters (formatted)

Upon completion of the execution, paraBLAST outputs the same search result of all significantly matched sequences from the database as those of the NCBI BLAST. The top 10 matches from the output files are listed in the followings (Note that the most significantly matched sequence is one of *E. coli* that is a superset of the query sequence.):

Sequences producing significant alignments:	Score (bits)	E Value
gb|AF487900.1| Escherichia coli aspartokinase...	589	e-165
gb|U14003.1|ECOUW93 Escherichia coli K-12...	589	e-165
gb|AE000111.1|AE000111 Escherichia coli K12...	589	e-165
gb|AE015038.1| Shigella flexneri 2a str. 301...	579	e-162
gb|AE005177.1|AE005177 Escherichia coli...	567	e-159
dbj|AP002550.1| Escherichia coli O157:H7...	567	e-159
gb|AE005671.1|AE005671 Escherichia coli O157...	472	e-130
gb|AE016755.1| Escherichia coli CFT073 s...	468	e-129
emb|V00361.1|ECTHRA First structural...	428	e-117
dbj|D10483.2|ECO110K Escherichia coli gen…	428	e-117

The evaluation of execution time was done with different number of fragments of the database; hence different number of CPUs were utilized each time. For comparison, the original BLAST program running on a single CPU was also executed. (paraBLAST produces the same query result as that of the original BLAST.) In order to find the fragment number with the best performance, the database was first split into fragments in the number of 2's powers, and then the number was adjusted. The experimental result is shown in Fig.1.

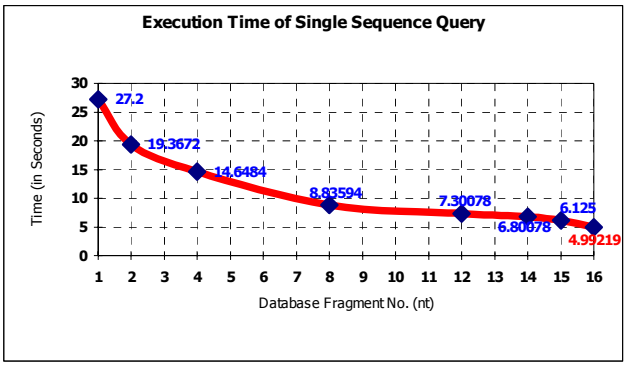
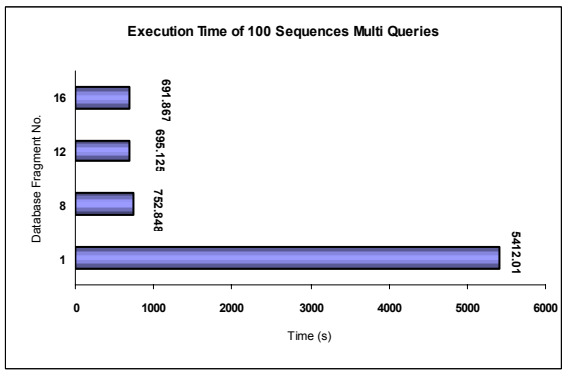


Fig.1. Analysis on Execution Time

As shown in Fig.1, the serial BLAST program takes 27.2 seconds to complete the search and paraBLAST takes much less time. The execution time is consistently reduced when the database fragment number is increased from 2 to 16, with up to 444.9% speedup.

To evaluate the proposed fragmentation scheme, we further used 100 and 500 DNA sequences, each with an average length 2021bps, to query the same database. The performance is shown in Fig.2.

a.



b.

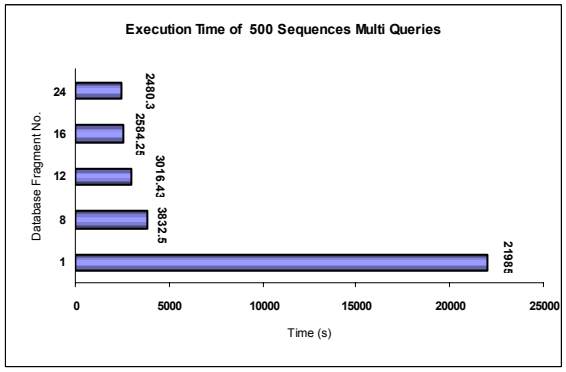


Fig.2. Batch Queries of DNA Sequences by ParaBLAST

To our expectation, as shown in Fig.2, the scalability of paraBLAST on multiple sequence queries is even better, up to 682.23% for 100 sequences (a) and 786.4% for 500 sequences (b) respectively, compared with 444.7% in the single sequence query.

The focus here is on the sensitivity of the database fragmentation. In Fig.2a, when the database was segmented into 8 fragments the execution time was sharply reduced. With 16 processors available in the cluster, paraBLAST system worked out a fragmentation with 12 (system desired number calculated through the proposed algorithm) fragments and each with 700MB. The execution time (695.125s) is very close to that of 8 fragments (752.848s) and 16 fragments (691.867s), which were manually set for comparison. This experimental series also prove that our proposed dynamic database-fragmentation scheme is practical.

Similarly, the sharp reduction of execution time is achieved in Fig.2b. The difference is that, in this case, with 24 processors available in the cluster, paraBLAST system worked out a full fragmentation with 24 fragments for more query sequences. We do observed the consistent reduction of execution time when the number of fragments increases from 8 to 24. In other

words, the fragmentation scheme is proved to be robust for a variety of sequence queries.

4 Conclusions

In conclusion, paraBLAST is developed to speedup the database searching through parallelization. It is by adapting the serial BLAST algorithm and applying MPI communication tools on a multi-node compute cluster. paraBLAST faithfully implements the BLAST searching algorithm while works at a much faster speed. Furthermore, paraBLAST is especially efficient for mess queries and large-scale databases. Better scalability is achieved. It helps bioinformatics researchers save much time in their routine work on the sequence similarity search.

References:

[1] Altschul,S.F, Gish,W., Miller,W., Myers,E.W. and Lipman,D.J. 1990. Basic local alignment search tool, *J. Mol. Biol*., 215:403-410.

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

[3] Ostell,J., The NCBI Software Development Toolkit, ftp://ftp.ncbi.nlm.nih.gov

[4] Bairoch,A. and Apweiler,R. 2000. The SWISS-PROT protein sequence data bank and its supplement TrEMBL in 2000, *Nucleic Acids Res.*, 28:45-48.

[5] Berman, H.M., Westbrook, J., Feng, Z., lliland, G., Bhat, T.N., Weissig, H., Shindyalov, I.N., and Bourne, P.E. 2000. The Protein Data Bank, *Nucleic Acids Res.*, 28:235–242.

[6] Gropp,W., Lusk,E. Doss,N.; Skjellum, A. 1966. A high performance, portable implementation of the MPI Message-Passing Interface standard. *Parallel computing,* 22:789-828.

[7] Pacheco,P.S. 1997. Parallel programming with MPI, Morgan Kaufmann, San Francisco

[8] Altschul,S.F., Boguski,M.S., Gish,W. and Wootton,J.C. 1994. Issues in searching molecular sequence databases, *Nat. Genet*, 6:119-129.

[9] Barker,W., George,D., and Hunt,L. 1990. Protein sequence database, *Methods Enzymol*., 183:31-49.