Improving the Performance of BLAST in a Memory Limited Environment

Warin Wattanapornprom[1], Natawut Nupairoj[2] and Prabhas Chongstitvatana[3]
Department of Computer Engineering, Chulalongkorn University
Phayathai Rd., Phathumwan Bangkok 10330, THAILAND
E-Mail: warin@chula.com[1], natawut@cp.eng.chula.ac.th[2] and prabhas@chula.ac.th[3]

Abstract: BLAST has become an important tool for the research in bioinformatics areas, since it can help scientists to make inferences about the functions of proteins. The BLAST's database is enormous and has been growing every day, and this causes the lower performance of the program. This paper presents an alternative way to improve the performance of BLAST in a single machine by reducing the overhead of disk swapping.

Key words: BLAST, Distributed database, Parallel search, Clustering, Memory Limited

1. Introduction

BLAST (Basic Local Alignment Search Tool) [1] is one of the most widely used search tools, which identifies statistically significant matches between newly sequenced segments of genetic material or proteins and databases of known nucleotide or amino acid sequences. Such searches allow scientists to make inferences about the structures and functions of their discoveries or to screen new sequences for further investigation.

Although BLAST have been designed and optimized for speed, the major drawback of BLAST is that it consumes large amount of CPU-time, memory, and I/O. It takes a very long time to search a large number of queries in a large database. There were attempts to reduce the searching time using many approaches such as upgrading computer hardware, using some parallel approach such as parallel queries search or using multiple processor machines. Our preliminary study of BLAST indicates that BLAST's running time is proportional to the size of the database. BLAST shows the highest efficiency if the whole database can be fitted in the memory. As the genome databases are enormous and doubling in size every 1.3 years [2], it is important to recognize the performance limitation due to the limited main memory.

To overcome this problem, we propose to separate the database into smaller parts, which each part fits the available memory and then search each part separately. We find that the time used for searching all separated parts is almost equal to the time used to search the whole database with the memory

large enough to hold it. This provides us the maximum efficiency for a given memory size.

This paper reports on our progress to design and develop the parallel system environment for BLAST. The paper is organized as follows. In Section 2, we begin with a preliminary experiment to the better understanding of the behavior of BLAST and the essential knowledge to improve the performance of the program. Then we describe the process to improve the performance and the result of the improved system in Section 3. Section 4 concludes the paper with status of the current prototype implementation and discussions of some future work.

2. Background

From our preliminary study of BLAST, we find that BLAST needs high bandwidth of the main memory. BLAST will show the highest efficiency if the whole database can be fitted in the main memory while searching. In the experiment, we vary the memory size while maintaining the other system environment such as CPU or hard disk; we find that the system with memory less than the size of the database will take longer time to process large amount of queries while the system with excess memory will not increase the performance of the search. This problem is called "memory-bounded problem", a problem in parallel processing [3].

Memory Size (Megabyte)	Number of Queries				
	10	50	100	500	1000
128	5	33	70	351	706
256	2	15	37	180	364
512	2	14	37	180	363

Table 1 shows the performance of BLAST at the different system environments

From the sample experiment, we use database named NR size 234 megabytes to represent the performance of the program because this size of database can fit into 256 megabytes of memory or higher. We can see that the performance is improved when we use larger size of memory. But if the memory size is too large, the system can no longer be improved. As we can see in the table 1, the performance of BLAST at 512 megabytes of memory cannot improve the performance from the system with 256 megabytes of memory at all.

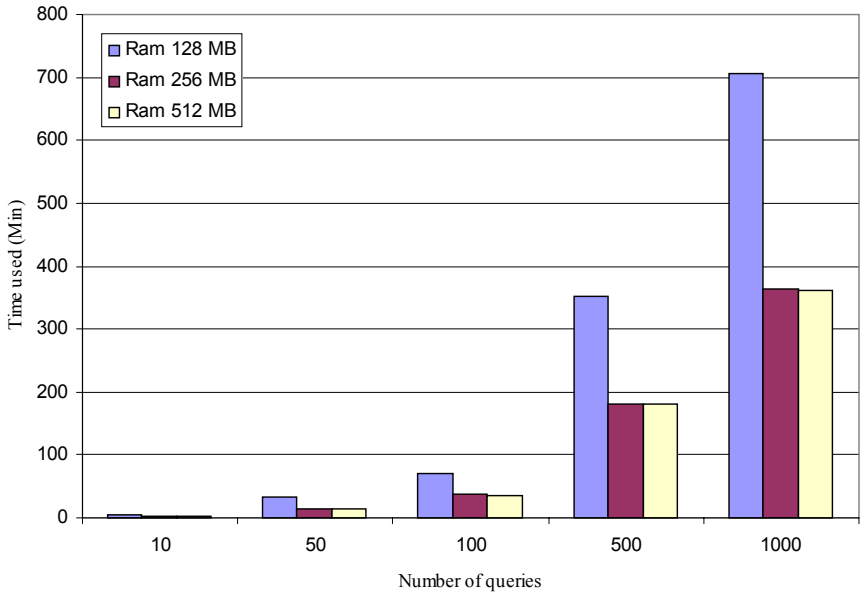

Figure 1 shows the performance of BLAST at the different system environments

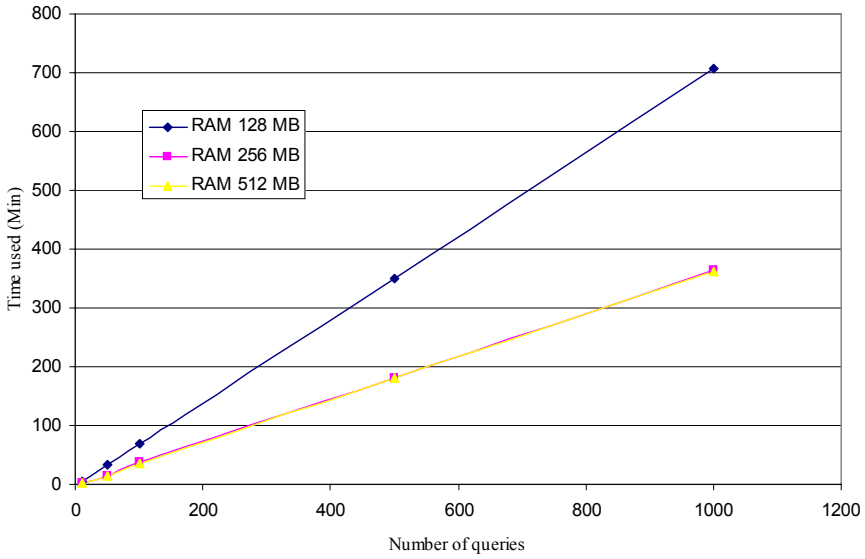

Figure 2 shows the linear performance of BLAST at the different system environments

The behavior of the searching process can be described. BLAST tries to copy the whole database into the main memory while searching every single query in the database. If the database is too large to fit into the main memory, the operating system will automatically allocate the memory space by swapping

the memory contents. The disk throughput is normally much less than the memory throughput, so the time is wasted by swapping between disk and memory. Since the memory which can hold the whole database at running time has to be enormous and expensive, moreover the systems that can install large size of database are scarce, the system should be improved to solve this problem.

We find that the database can be divided into smaller pieces. BLAST's database structure is simply a text file. Each database record is started with character ">" followed by necessary fields separated by "|" and ended up with protein or DNA sequence. The database file can be cut at the end of any record which is just before the ">".Output from the separated database can also be merged by just concatenate the result of each query together.

CKLSIKRATVIYEGERVAIQNKFKNGMLHGQKVSFFCKHKEKKCSYTEDAQ
CIDGTIEIPKCFKEHSSLAFWKTDASDVKPC
> gi|129249|sp|P02820|OSTC_BOVIN OSTEOCALCIN PRECURSOR (GAMMA-CARBOXYGLUTAMIC ACID-CONTAINING PROTEIN) (BONE GLA-PROTEIN) (BGP)_gi|538590|pir||GEBO osteocalcin precursor - bovine_gi|8|emb|CAA35997.1| (X51700) bone Gla precursor (100 AA) [Bos taurus]_gi|720|emb|CAA37737.1| (X53699) Gla protein precusor [Bos taurus]

Figure 2 shows an example of the BLAST's database records

3. Performance Improvement

The system performance is hypothetically better if the database size is fitted the memory size while searching. To avoid the overhead caused by swapping, we propose steps to reduce the time to search a large number of sequence queries as follows.

1. Preprocessing – separate the database into smaller parts to be able to fit in the memory (the size should be almost similar to the available memory) then create index files for each of the.
2. Processing – Search all queries in each of the splited database parts.
3. Postprocessing – Concatenate the output.

We proof this hypothesis by separate the database into different sizes which are small enough to fit in the main memory then process all queries and concatenate the output together. The experiment is carried out in one machine to eliminate the overhead of communication between machines. The total time is the sum of all processing time in each separated databases and postprocessing time. The preprocessing time is not counted because the preprocessing is done only once before processing any query.

$$t_{total} = \sum_{i=1}^{n} t_i + t_{post}$$

Where

t_{total} = Total time used.

t_{post} = Time to concatenate all output.

t_i = Time to process database part *i*.

n = Number of separated databases.

No. of separated Database	Database Part No.	Database size (Megabyte)	Time used (min)	Time used to concatenate (min)	Total time used (min)
1	1	234	70	0	70
2	1	109	17	1	38
	2	125	20		
3	1	100	16	1.5	38.5
	2	100	15		
	3	34	3		

Table 2 shows the performance of BLAST at various numbers of separated databases.

The experiment is done with the same database in the section 2. We separate the database into different sizes and process the same 100 of queries at the fixed memory size at 128 megabytes. We can see that no matter how many parts the database is separated, if each part of the database fits the memory, the time used to process all of them will equal to the time used to process the original database in the environment which memory can contain the whole database at runtime. The time used to concatenate the output depends on number of separated databases because the more output file will need more time to concatenate. The performance of the proposed method can be measured using “speedup”. Speedup is the ratio of the normal execution time to the improved execution time. In our experiment, the speedup of the improved system is the total time used in an improved system divided by the time used in a single database search which is 70/38 = 1.84 times faster.

The results show that the time used to search through the separated databases at the limited memory size is equal to the time used to search in the environment that memory size is large enough to hold the whole database. In fact, there are some overhead in merging the outputs which depend on the search result and the number of separated databases. The overhead is less than a minute in this case and will take longer time depends on the pieces of the separated database.

4. Conclusions and Future work

In this paper, we have studied the main factor which slowdowns the searching process of BLAST and have found the way to solve this problem. Since BLAST's databases are enormous and have been growing larger every year and main memory cannot hold the whole database at running time, so the overheads are generated by swapping of the memory contents. We propose to separate the database to be able to fit the available memory and search through each database part separately. This provides us the maximum efficiency for a given memory size. Form the results of our experiments, BLAST shows that the performance can be doubled when we apply our technique. However, the improvement can be varied depended on the database size, the disk speed and the size of memory. Our proposed steps are not only enabling BLAST to search through the whole database at the peak performance, but also reducing the access time. The access time of the disk is reduced due to the smaller database fits in the given memory and requires no disk swapping.

We plan to expand our studies to apply our techniques to improve the performance of the searching on the clustered computers by distributing each database part to each of the clustering node and performing the search in parallel, and we expect even greater performance improvement. We are in the process of designing an optimized system at different number of processors to maximize the performance of the system.

5. References

[1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W. & Lipman, D.J. "Basic local alignment search tool" *Journal of Molecular Biolog.* 215:403-410. (1990)

[2] Chi E.H., Shoop E., Carlis J., Retzel E., Riedl J,"Efficiency of Shared-Memory Multiprocessors for a Genetic Sequence Similarity Search Algorithm" (1997)

[3] Xian-He S, "Scalable Problems and Memory-Bounded Speedup" *Journal of Parallel and Distributed Computing* (1993)