Implementation Issues of Optimized Buffer Management for BLAST

Melissa Lemos[1], José Antônio Fernandes de Macedo[2], Luiz Seibel[1], Fabio Porto[3], Rogério Costa[1], Roberto Cavalcante[1], Vitor Medina Cruz[1]

[1]Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rua Marquês de S. Vicente 225, Rio de Janeiro, RJ 22453-900, Brazil

[2]Departamento de Computação, Universidade Federal do Ceara, Campus do PICI, Bloco 910, CEP 60451-970 Fortaleza - CE, Brazil

[3]Extreme Lab, Laboratório Nacional de Computação Cientifica, Av. Getúlio Vargas 333, Quitandinha - CEP 25651-075, Petrópolis - Rio de Janeiro, Brazil

Abstract. *Comparing sequences is one of the basic operations every Life Science scientist must execute. The most popular sequence comparison algorithm is BLAST. This paper presents experimental results of a buffer management strategy that optimizes the simultaneous execution of a set P of BLAST processes. The essence of the strategy is to cycle all the sequences in the database through a buffer so that all BLAST processes will perform their comparison synchronously.*

1. Introduction

Genome projects usually start with a sequencing phase, where experimental data (usually DNA sequences) is generated, without any biological interpretation. The fundamental challenge for Life Science scientists is to analyze sequences to extract biological relevant information, with the potential to unveil many aspects of the genetics, biochemistry and physiology of the organisms under study. One of the first peculiarities one encounters when studying genome databases is that sequence comparison is not exact pattern matching. In the context of molecular biology databases, optimal sequence comparison algorithms are unfeasible. For this reason, many alternative and faster methods have appeared. Among them, the more popular is BLAST [1].

BLAST performs an exhaustive search of a database to try to find the best match. The search is done sequentially. In other words, the first database sequence is the first sequence compared to the query sequence, the second database sequence is the second sequence compared to the query sequence, and so on. This order does not influence the BLAST results, since the query sequence is compared to all database sequences. The growth in size and complexity of public databases, associated to a large number of simultaneous BLAST processes running against these databases, has contributed to a poor performance of the BLAST program response time.

This paper shows experimental results of an ad-hoc buffer management strategy that optimizes the simultaneous execution of a set of BLAST processes. The essence of this strategy, which has the idea originally published in [9], is to synchronize the comparison operation of all BLAST processes by cycling all database sequences through a buffer. This approach differs from the traditional operating system buffer management strategy by the fact it takes into account specific features of the BLAST algorithm to implement its buffer management strategy.

2. Optimized buffer management

Consider first a single BLAST process p_1 executing on a single processor and accessing a database D. Assuming that it is not feasible to retrieve all sequences stored in D into the main memory, we may allocate a set of buffers B to p_1, organized as a ring (circular list) and managed in the usual way. We call this structure a *ring*. However, as we have already pointed out, there will be usually several BLAST processes simultaneously accessing D and starting at different times. Let $p_1, p_2,\ldots, p_n$ be these processes.

A naïve buffer management strategy, which we call *private-ring*, would allocate a private (or separate) buffer ring to each process p_i. This strategy is not very effective, because it may easily exhaust buffer space or I/O capacity. Another strategy, which we call *public-ring*, would be: (1) Allocate a public buffer ring B to all processes; (2) Regulate buffer consumption by the slowest process; (3) Continuously cycle all sequences in D through the buffers in B, creating reading cycles; (4) Signal to a process when it completes reading all sequences in D (with the help of auxiliary structures).

The fact that processes start at different times implies that they will start reading the database at different points.

3. Experimental results

This section presents experimental results which compares the original BLAST version with the proposed public-ring management (PRO-BLAST). The tests were performed using the BLASTN program from the WU-BLAST version 1.4 package[1] and a machine with 512 MB of RAM, disconnected from the network and running under the operational system Linux Fedora 2.6. To analyze the influence of concurrent processes in original BLAST and PRO-BLAST performance, we have executed N original BLAST processes concurrently, followed by N PRO-BLAST processes executed concurrently. Each group of processes compares a query sequence with S nucleotide bases with all the sequences stored in a nucleotide sequence database.

Figures 1 and 2 show results for query sequences with S = 100 and S = 300 nucleotide bases, respectively, using the database *patnt*[2] [4] and groups of different numbers (N = 1, 2, 3, 4, 5, 10, 15 and 20) of processes borrowed (randomly) sequences from *patnt*. These tests were performed using 256 MB of RAM (128 MB available for data). The public-ring was configured to avoid the database *patnt* fitting into the

[1] http://blast.wustl.edu/.

[2] NCBI

memory, leading the operating system to perform memory swapping. During our experiments, we notice that the more concurrent processes are in execution, the more efficient is PRO-BLAST when compared to BLAST (Fig. 1). Indeed, PRO-BLAST was faster than BLAST even when a single process was running. This occurs because PRO-BLAST executes sequence prefetching during the startup and the original BLAST uses the operating system memory manager to perform swap. Figures 3 and 4 show the results obtained with groups of N = 15 and 20 concurrent processes, respectively, created using a query sequence with S = 300 nucleotide bases and the database *patnt*.

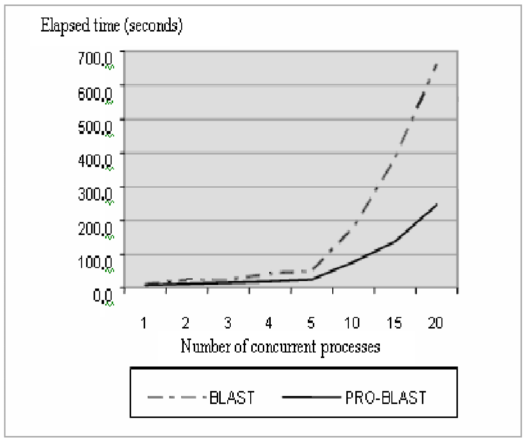

Fig 1. Query seqs. with 100 bases

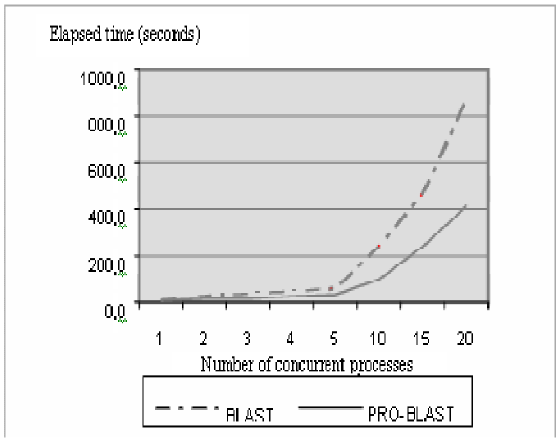

Fig 2 - Query seqs. with 300 bases

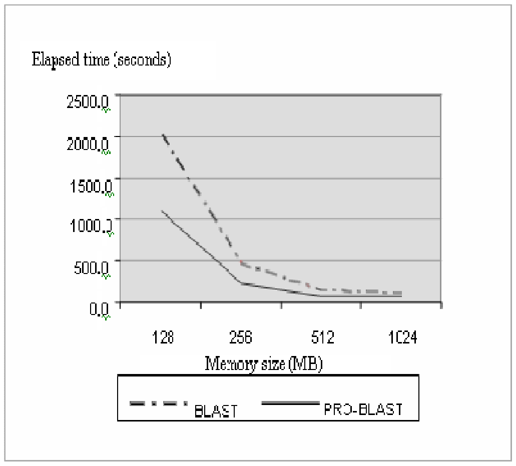

Fig 3. Query seqs with 300 bases (15 concurrent processes)

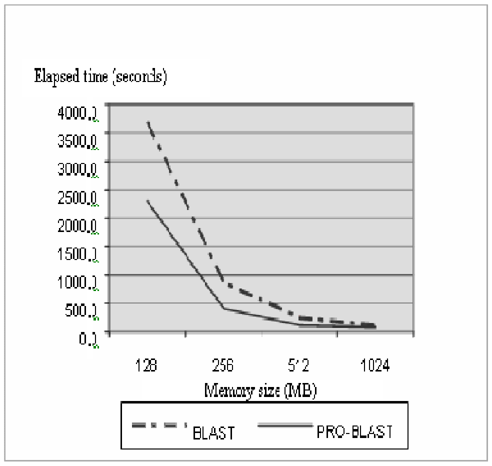

Fig 4. Query seqs. with 300 bases (20 concurrent processes)

6. Conclusions

We observed that PRO-BLAST is always more efficient than the original BLAST when there are processes executing concurrently. In addition, the efficiency is higher when the memory size is reduced. This fact confirms the usefulness of the public-ring strategy in reducing page swapping.

References

1. Altschul, S.F.,et al (1990), A Basic Local Alignment Search Tool. Journal of Molecular Biology, 215, 403-410.

9. Lemos, M., Lifschitz, S. (2003), Memory Management for BLAST Processing, 1st International Workshop on Biological Data Management in Database and Expert Systems Applications (DEXA), Prague, Czech Republic, (2003) 5-9.