

Performance evaluation of parallel BLAST

Nazif Cihan Tas, Indrajit Bhattacharya, Vinay Shet

1. Introduction

Proteins are the building blocks of life. Proteins and interactions between them define the biology of life as we know it. Information for creation of proteins lie encoded within the genes. Sequencing and annotating genes is therefore the basis of biological research, and it is the job of biologists to gather genetic information embedded within cells to study various bio-chemical processes.

Bioinformatics is the study of the information content in genomic data which allows scientists to process and analyze thousands of DNA and protein sequences each day. For each DNA or protein sequences that researchers encounter, they need to know whether that has previously been annotated. If not, they need to infer its properties, functions and structure by considering its similarity to annotated genes or proteins. This means comparing the sequence at hand against huge DNA or protein databases that are now available on public domain. The primary tool for this process is Pairwise Sequence Alignment.

Pairwise Sequence Alignment is computationally expensive and its time complexity is quadratic in the sequence length. This is infeasible for real life proteins or DNA fragments, when they have to be searched against entire databases. Thus optimal solutions to this problem are largely intractable and linear time heuristics like **BLAST** are usually resorted to.

Basic Local Alignment Search Tool (BLAST) is a powerful and by far the most popular sequence alignment tool that uses a scoring matrix and gap penalties based on empirical data to compute a 'good' alignment for two sequences. Researchers rely heavily on tools like BLAST to compare experimental sequences against huge databases put together by a global community. As the database sizes continue to grow and load on the query servers multiply, sequential query processing strategies are proving insufficient. With the goal of performance enhancement, various versions of BLAST have emerged. These versions make use of that fact that parallel computing is now becoming more affordable and can be used to speed up computationally intensive applications like BLAST. They make use of the fact that problems like sequence alignment against a database are embarrassingly parallel, making it possible to compare against each database entry independently and parallelly.

As a result of such high-performance developments, researchers today can choose between various versions of BLAST like MPI BLAST, threaded BLAST, sequential BLAST among others, which can be configured to run on expensive multiprocessor machines or relatively cheaper clusters of off-the-shelf workstations or PCs. Given the multitude of hardware and software choices one can make, it is not immediately clear what configuration should be chosen given one's requirements.

The aim of our project is to study performance of these different versions of BLAST with the eventual goal of recommending the scheme that runs that would yield results in the shortest time, given a query and database to search against.

2. Background and Motivation

Given a batch of queries and the option of using either a multi-processor system or a cluster of workstations, there are different ways of improving turnaround time. The strategy adopted by MPI BLAST when running on a cluster of workstations is to split the database physically across the different nodes, split the comparison tasks and memory requirements across the nodes and use explicit message passing to coordinate between the processors and combine the results. The extra overhead incurred by the coordination and message passing may not pay off for small databases and small-to-medium length queries, but for databases that are too big to fit in the physical memory of a single node, it clearly offers an advantage.

Threaded blast on the other hand, is designed to run on a multi-processor system where all processors use the same physical memory and communication overheads are largely reduced. So one would expect threaded versions to perform better for smaller databases, while distributed versions are likely to outperform them for larger database sizes.

When considering a batch of queries, there is another option for improving the throughput for the entire batch, without caring about the turnaround time for any particular job. We can forget the parallel versions of blast, assign each job to a different processor (assuming we have more processors than jobs) and run sequential blast. That suggests another option to evaluate in our experiments.

As observed in [1] and [2], in fast-turnaround mode, a user can reduce the solution time of a single-processor job by running the job in parallel over multiple processors. The performance measure by which a system is able to run a parallelized application is its parallel efficiency. Parallel efficiency is measured using an application's parallel speedup, which is the ratio of the solution time on a single processor to the solution time on multiple processors:

$$s_N = \frac{t_1}{t_N}$$

s_N = parallel speedup on N processors
 t_1 = time for serial code on 1 processor
 t_N = time for parallel code on N processors

One would ideally like to have linear speedup, where the speedup on N processors equals N (e.g., 4x speedup using four CPUs). In fact, it is even possible to achieve superlinear speedups, where the speedup on N processors is actually greater than N. But this is a rare occurrence and is often due to differences between the serial and

parallel implementations of the code (e.g., the parallel version may be more computationally efficient by eliminating certain data initializations). Both linear and superlinear speedup, however, are generally not achieved in practice because parallel speedup is limited by the portion of code that is run serially. This limitation is known as Amdahl's Law and is expressed by the following formula:

$$S_{\max} = \frac{1}{s + p/N}$$

S_{\max} = maximum possible speedup
 s = serial portion of code
 p = parallel portion of code
 N = number of processors

According to Amdahl's Law, the maximum speedup that one can achieve on a given number of processors is bounded by the serial or nonparallel portion of the code (i.e., as N approaches infinity, S_{\max} approaches $1/s$). So a program that is 100% parallelized can exhibit linear speedups such as 4.0x on four processors, but a program that is 90% parallelized can only achieve a maximum speedup of 3.1x when using 4 processors. Parallel efficiency, therefore, is the ratio of the parallel speedup to the number of processors:

$$E_N = \frac{S_N}{N}$$

E_N = parallel efficiency on N processors
 S_N = parallel speedup on N processors
 N = number of processors

3. Experimental Setup

Our experimental setup consists of two different parallel architectures. We installed the LAM (Local Area Multicomputer) MPI programming environment on a cluster of workstations (RI nodes). We executed MPI BLAST on this configuration for our distributed evaluation. We also set up threaded BLAST on a multiprocessor Sun SMP with 8 processors. The sequential version of BLAST was executed on the RI nodes.

Keeping in mind our ultimate goal to recommend a version of BLAST given the query and a database, we identified a number of variables for this project. These were the number of processors, length of query sequences, number of sequences in a batch query and size of the database being searched against. We then designed a number of experiments to evaluate the performance of the different versions of BLAST over the domain of these variables.

We divided our experimentation into two categories. In the first, we fixed the number of queries to 1 but varied the sequence lengths to study its effect on BLAST performance. For each query we varied the number of processors and the database size. We consulted with people involved with research in Genomics and were advised that statistically speaking biologically significant query lengths could be grouped into

three categories. We will refer to these categories as Low (mean of 40 bases), Medium (mean of 4000 bases) and High length (mean of 15000 bases).

The second category of experiments was done to try and understand if these versions of BLAST can optimize performance when multiple query sequences are submitted as a batch. We fixed the size of each of our queries to 15000 bases and varied the number of sequences from 3 to 9 in increments of 3.

For both these categories, we used 5 different nucleotide databases obtained from NCBI Genbank. The nucleotide databases at our disposal were Mitochondria (**size ~3 MB**), Ecoli (**~4.7 MB**), Yeast (**~12.3 MB**), Drosophila (**~124 MB**) and EST Human (**~2.1 GB**). We also varied the number of processors from 1 to 8. The queries that we used were generated randomly.

4. Experimental Results and Analysis

We found the results of our experiments to be quite interesting. The results are summarized in three figures which can be found in the appendix. In terms of our expectations, they are validated in the sense that threaded BLAST performs better than MPI BLAST for small databases. MPI BLAST slowly catches up with it as the database size grows and clearly outperforms it for our largest database. This is clearly visible from all the three figures.

But what we hadn't foreseen was that the execution time vs. number of processors plot more or less maintains its shape across differing sequence lengths for the same database as can be seen by looking across each row in Fig 1 and Fig 2.

Another observation was that for large databases, as the number of processors increased, the performance for threaded BLAST degrades as seen from the graphs. This is probably due to the database size being too large to fit in physical memory, which probably gets split for more number of processors and thrashing comes into the picture.

Yet another observation was that for small database sizes, sequential BLAST (MPI BLAST for one processor) was always better than both Threaded and MPI BLAST. However as the database size increases, Threaded and MPI BLAST catch up with sequential BLAST and outperform it beyond 2 processors. This can be seen in Figure 1 from the plot for Drosophila. For the EST database sequential blast is not even comparable to the MPI.

In our experiments with varying number of sequences submitted as a batch, the interesting thing to observe is that for large database sizes, the number of queries does not matter for the relative times taken by MPI and threaded BLAST. However, for the smaller database (Mitochondria, Ecoli and Yeast), MPI BLAST outperforms its threaded counterpart as the number of queries and processors go up.

The shape of the plots and the relative behavior of MPI and threaded BLAST can be better appreciated from the 3D plots rendered by MATLAB (Figure 3). The plots for MPI and threaded BLAST show a clear intersection after the yeast database (Figure 3a) and the time for threaded BLAST shoots up for more processors as the database size grows (Figure 3b). Figures 3c and 3d highlight the intriguing fact that the two plots have a second intersection as the number and processors and the batch size go up.

5. Recommendations

We summarize our recommendations to researchers regarding which version of BLAST to run given the query as follows:

- i) For database size greater than 15-20 MB it is better to use MPI BLAST irrespective of the number of sequences.
- ii) In such cases, using MPI BLAST over more number of processors will yield better results
- iii) For small databases, (smaller than 15-20 MB) Threaded BLAST performs better than MPI BLAST and there is no significant improvement by increasing the number of processors. However for such cases, it is preferable to use sequential BLAST rather than its parallel counterparts.
- iv) When there are many sequences (greater than 3), it is better to use MPI BLAST as it seems to have some inbuilt optimizations.
- v) For such cases, performance generally gets better with increasing number of processors
- vi) For small number of sequences and small databases, Threaded BLAST should be chosen and there is no significant difference with the number of processors
- vii) Length of sequence results in no significant changes to the guidelines stated above.

6. Conclusions

Sequence alignment is a crucial method in Bioinformatics technology. In this project, we experimented with several high performance versions of the most popular sequence alignment tool BLAST. It is usually not clear given the myriad options which researchers have available before them, which version of BLAST to run for a given query and database. Based on our experimentation, we suggest recommendations to address this dilemma.

7. References

[1] SGI Bioinformatics Performance Report,-Fall2001
www.sgi.com/solutions/sciences/chembio/resources

[2] Turnaround vs. Throughput: Optimal Utilization of a Multiprocessor , White Paper, System Haruna Cofer, Nick Camp, and Roberto Gomperts September 1998

Appendix

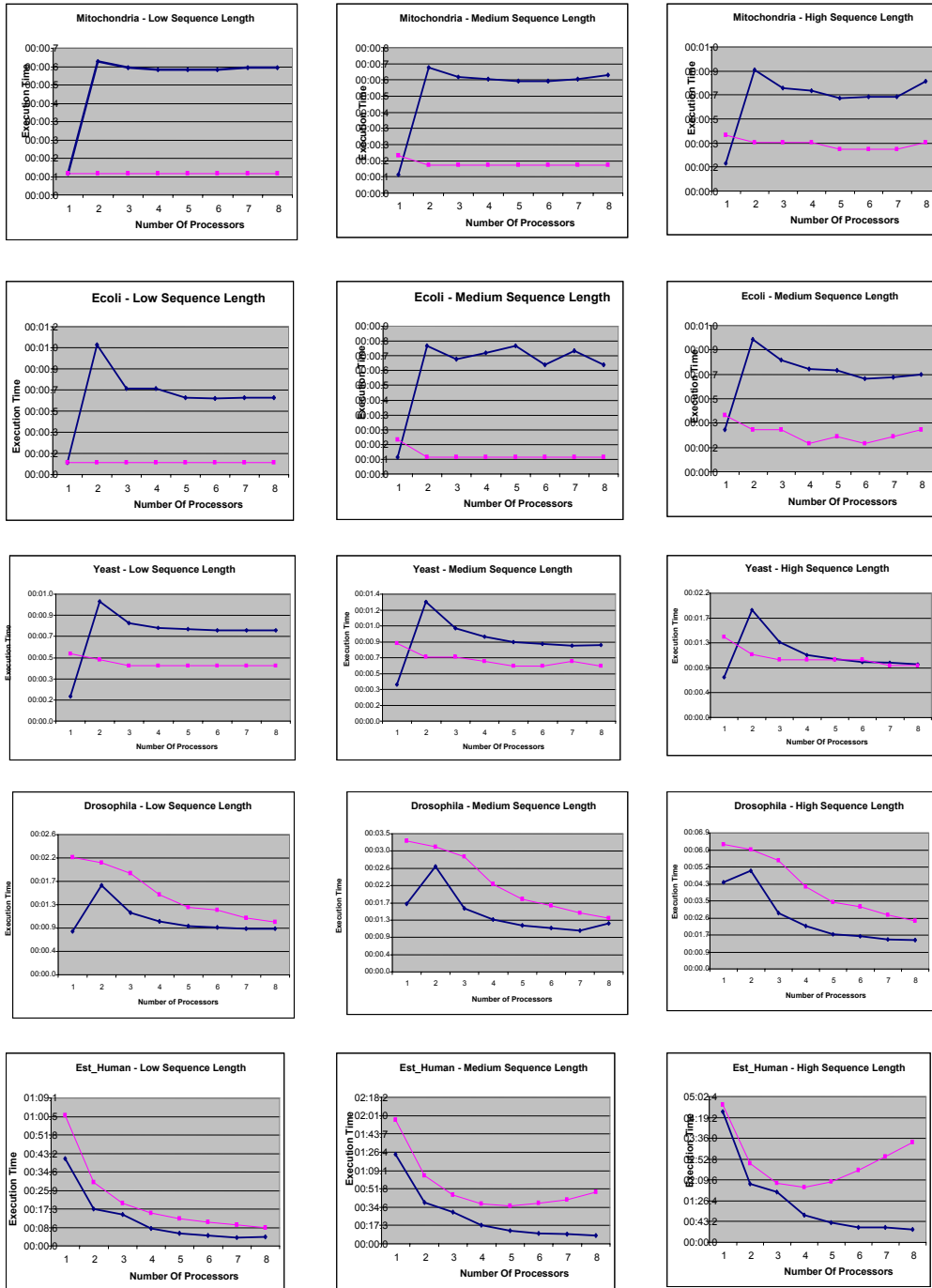


Figure 1. Execution times for varying Query sizes (The darker plot is MPI BLAST and the lighter one is Threaded BLAST)

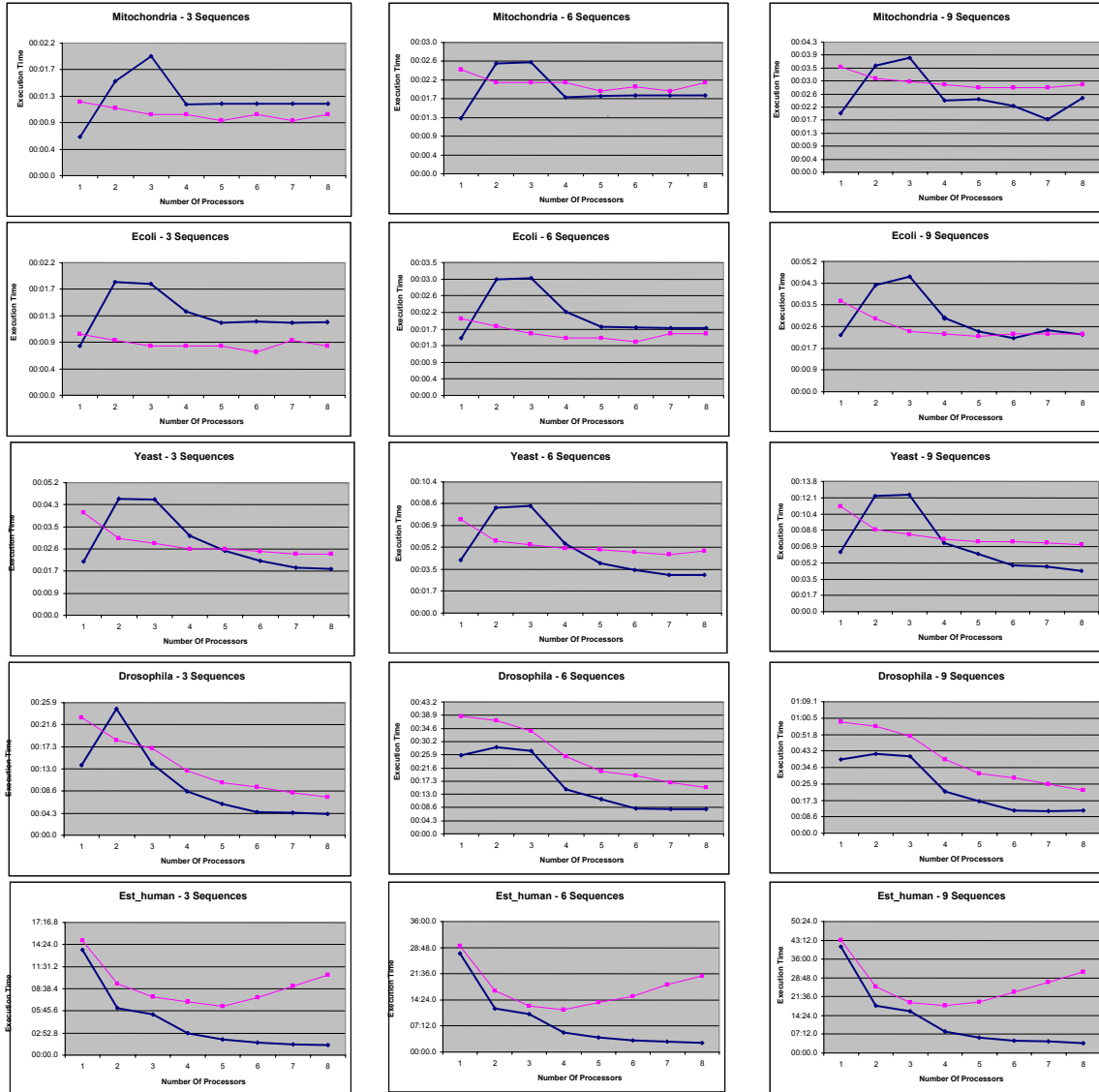


Figure 2. Execution times for varying number of Queries (The darker plot is MPI BLAST and the lighter one is Threaded BLAST)

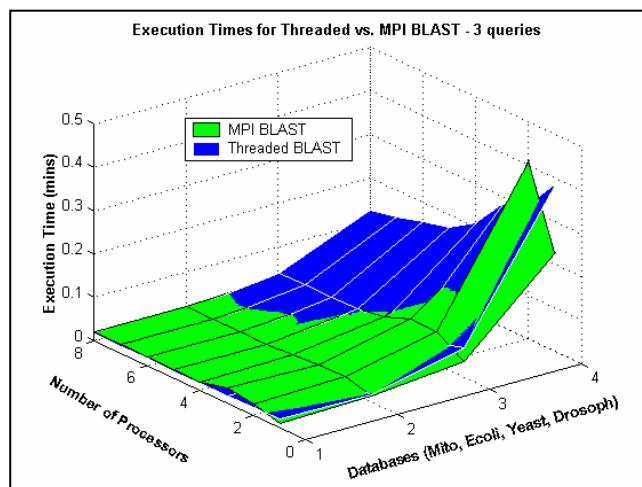
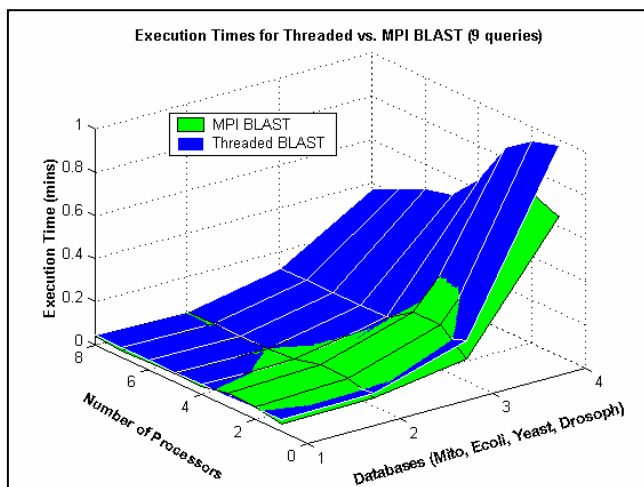
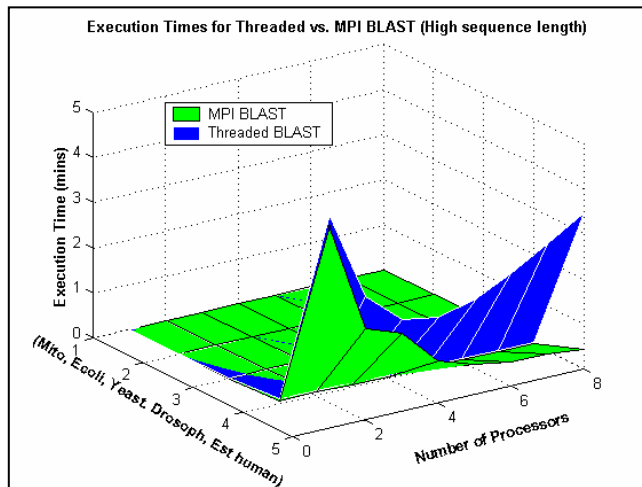
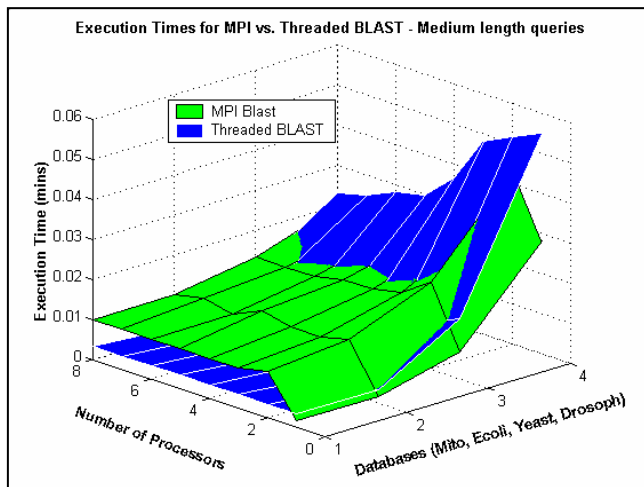


Figure 3: Execution times for different databases and processor numbers
 a. For medium length queries without EST, b. For high sequence queries in rotated view
 c. For 9 queries and d. For 3 queries.