Parallelisation of BLAST - DNA Search

H.H. Durrant
Institute of Information and Mathematical Sciences,
Massey University - Albany,
North Shore 102-904, Auckland, New Zealand,
Email: hh.durrant@gmail.com,
Tel: +64 9 413 8486 Fax: +64 9 413 8484

May 24, 2009

Abstract

DNA sequence comparisons, such as those executed in the Basic Logical Alignment Tool (BLAST) algorithm, can be shown to be both computationally expensive and embarrassingly parallel, making them excellent candidates for parallelisation. This report analyses various techniques used to parallelise the BLAST algorithm, with emphasis on one particular open source implementation; mpiBLAST. mpiBLAST has achieved super linear speedup of the BLAST sequence comparison algorithm, and has been in constant development since its inception. The report concludes with a discussion of several improvements that could be made to mpiBLAST, along with the acknowledgement that parallelisation in this area is essential in enabling biologists to search through the mass of sequence data available to them and to continue making major biological advancements.

Contents

List of Figures

Chapter 1

Introduction

The explosion of growth of biological information generated over the past few decades has led, in turn, to an "absolute requirement" for computerised databases and algorithms to search, organise and analyse biological data [1]. The parallelisation of computing techniques is essential in the bioinformatics field, for the mass of data dealt with often renders sequential algorithms unable to provide results in a reasonable time period. This report considers the parallelisation of the Basic Logical Alignment Search Tool (BLAST), a popular algorithm used in biological sequence comparison. Focus will be placed on one particular parallel implementation, mpiBLAST.

1.1 Sequencing

The two types of sequences recognised by the BLAST algorithm are nucleotide and peptide [2, 3]. DNA sequencing involves determining the order of the four bases that make up DNA, as it is this order that encodes the genetic information biologists are interested in. As the order is determined, it can be transcribed into an alphabetic string, using letters to represent the bases. Such strings are called DNA or nucleotide sequences. Peptide sequences are also made up of a string of letters, each representing a component amino acid of the protein [2, 3]. Proteins are made of a range of about twenty amino acids in total.

Biologists require the ability to compare sequences. A match can provide the first clue about the function of a newly sequenced gene [3, 1], and the results of comparisons can be used in the creation of drugs, to assist in treating diseases, or to help combat newly discovered viruses [4, 2].

A query sequence often needs to be compared against a large number of known sequences, stored in a database. GenBank is an annotated collection of all publicly available nucleotide sequences [5]. GenBank is run from the National Center for Biotechnology Information (NCBI), in Maryland, USA. The NCBI is part of the International Nucleotide Sequence Database Collaboration (INSDC). A daily

exchange of new sequences between the NCBI and its European and Japanese counterparts in the INSDC ensure that GenBank is a comprehensive, worldwide collection [5]. As can be seen in 1.1, the growth of the database is exponential. It is obvious that the speed of sequence comparison methods must reflect the mass of data against which a query sequence may be being searched.

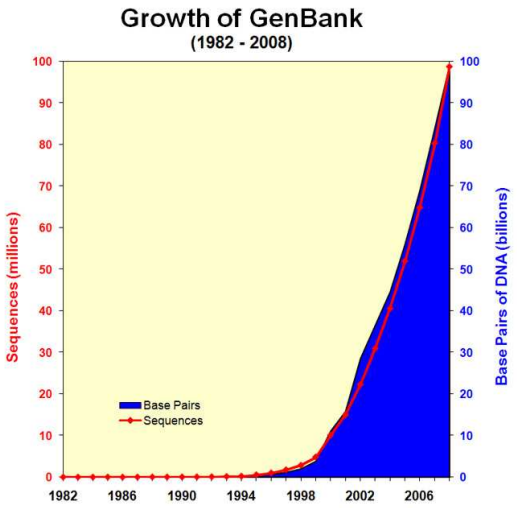

Fig. 1.1: The growth statistics of GenBank, an annotated collection of all publicly available DNA sequences.

1.2 The BLAST Algorithm

BLAST, detailed in [3], is a simple and robust algorithm for rapid sequence comparison, first published in the Journal of Molecular Biology in 1990. BLAST is designed to compare a query sequence against a sequence database, and identify sequences that resemble the query sequence above a certain threshold. It provides statistical information on the similarity of the sequence, which the biologists can then use to infer relationships between sequences.

1.2.1 Sequence Comparison

Sequence comparisons are carried out by recursively comparing small subsequences of one sequence against another sequence, until the whole first sequence has been compared against the length of the second. Each comparison generates a score, to represent how close a match they are. If the letters are identical, this will generate a very high score. This is not a binary exercise, however, and if letters

are not identical, there is still a chance the two sequences are related, and thus should have a score that reflects this.

The sequences of an organism's proteins are gradually altered, from one generation to the next. Each amino acid in the protein is more or less likely to transform into other amino acids. With this knowledge, matrices can be constructed to reflect the probability of one amino acid mutating into any other amino acid [6]. DNA Matrices can also be constructed, although here substitution is all but barred by the large penalties for mismatches in the matrices [6]. BLAST, and other sequence comparison algorithms, employ matrices such as these to score sequence comparisons in such a manner that acknowledges that nonidentical sequences may still have a shared ancestor [4, 14]. Gaps may also be allowed for in sequences, to decrease sensitivity of the comparison algorithms.

1.2.2 Heuristics

BLAST takes a heuristical approach to speeding up its underlying sequence comparison algorithm. This involves calculating partial similarity scores, and spending less time in regions of sequences that are unlikely to exceed a certain threshold score [4, 17]. Although this resulted in an algorithm that vastly outperformed existing methods, it is in no way capable of searching through databases the size of GenBank in a timely manner.

While it is possible to download the contents of GenBank, most people query the database through the NCBI website. This is becoming problematic, as the ability of the cluster of servers dedicated to processing such queries continues to decline, and the network traffic generated by the increasing number of requests is becoming intolerable [2, 2]. In peak times, a single BLAST search can take up to several hours on the NCBI site, and some research requires the ability to perform thousands of searches a day [7, 1]. The need for a dramatic speed up of the BLAST algorithm is obvious.

Chapter 2

BLAST Parallelisation

Fortunately, the problem of sequence comparison is well suited to parallelisation. As each query sequence needs to be searched against almost all sequences in the database (heuristics may prevent the search from extending to all sequences), and a sequence comparison only needs two sequences as inputs, either the queries, the database, or both, can be partitioned to run on separate processors with minimal necessary shared information. As noted in [8, 1], BLAST is both "computationally intensive and embarrassingly parallel," resulting in many attempts to parallelise the algorithm.

2.1 Hardware Parallelisation

Hardware parallelisation focuses on achieving parallelism at the lowest level, by allowing many sequence comparisons to be performed concurrently [9, 6]. To this end, custom built hardware may contain chips with hundreds or thousands of logic elements, along with optimised character processing, to bring massive speed up to the sequence comparison process. The 'DeCypher BLAST Hardware Accelerator' noted in [9] is reportedly able to provide 200 times faster execution time than an 8 CPU server, for 4300 protein sequences searched against 192 bacterial genomes. Despite this impressive speedup, however, hardware implementations suffer from very high system costs, and they are inflexible, as they only handle a very narrow set of algorithms and applications [9, 6].

2.2 Database Segmentation

The two most common forms of parallelisation of the BLAST algorithm are database and query segmentation. Database segmentation involves partitioning the database into subsets and distributing them to storage devices on a cluster of workstations - possibly replicating each subset some number of times on several

nodes of the cluster 2.1. Single queries are then replicated to the multiple nodes, and comparisons against each part of the partition can proceed in parallel [4, 20].

Database segmentation lessens memory requirements by allowing each node to store only a fragment of the entire database in persistent storage. The most common sequence databases are much larger than the core memory on most computers, and database segmentation is crucial in reducing the paging and extraneous disk I/O that occurs in this situation, causing a massive slow down in execution time [8, 2]. Braun et al. also note that by leveraging the power of multiple nodes, individual jobs are completed in a much shorter time, when not at peak load across nodes [2, 2].

The scalability of algorithms that use database segmentation can, however, be adversely effected by the higher parallel search overhead, and the need for local results to be merged globally [10, 9].

2.3 Query Segmentation

In contrast to database segmentation, query segmentation requires that each cluster or computing node have a full copy of the database. Once this has been actualised, batch processing and scheduling of sets of queries to each node can begin 2.1. This is of obvious use where large sets of queries need to be processed at once, but the memory requirements are often debilitating [11, 2]. The amount of memory required to store the full database at each cluster or node is costly, and if memory is not sufficient, paging will occur and slow down execution. If a database can fit into single processor memory, however, linear speedup can be achieved [4, 20].

Query segmentation does have several advantages. Unlike database segmentation, it has a low parallelisation overhead, and is thus more easily scaled to more processers [2, 5]. Several methods of optimisation can be used when scheduling, including prioritisation of interactive jobs if necessary [2, 2]. This is useful for situations such as that at the NCBI, where both batch jobs and queries from the internet are being put to GenBank using BLAST.

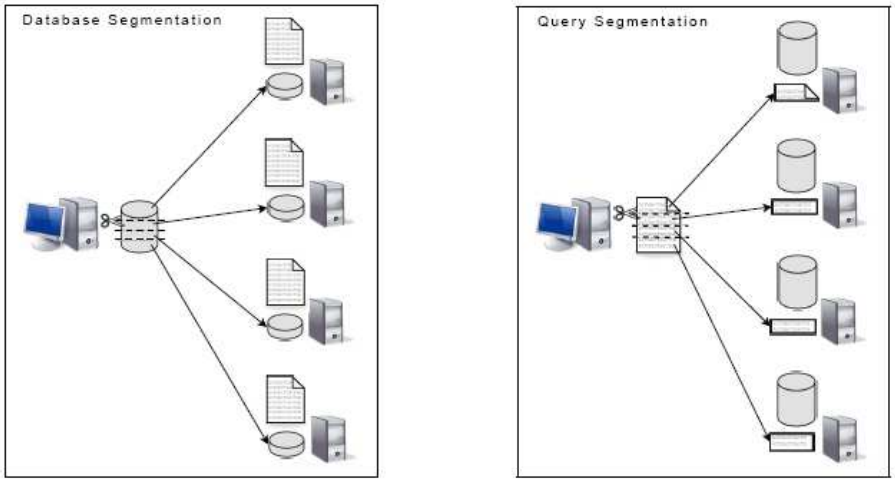

Fig. 2.1: Database segmentation partitions and sends fragments of the database to each cluster or node. Query segmentation requires that each cluster have a full copy of the database, against which some of a set of queries can then be searched.

Chapter 3

mpiBLAST

mpiBLAST is an open source, parallel implementation of the BLAST algorithm, that is based on database segmentation and uses the Message Passing Interface (MPI) standard for communication between computers. It is very popular, and was downloaded over 40,000 times across five major releases, between late 2002 (the year of its release) and 2007 [12, 1]. mpiBLAST has been developed to run on clusters with job-scheduling software such as PBS (Portable Batch System) [8, 2].

3.1 System Design

mpiBLAST uses a master-slave approach, with one master process and p - 1 worker processes. Firstly, the worker processes inform the master which database fragments they have stored locally. The master then reads the query sequences from disk, and broadcasts them to all other processes. When the worker nodes receive their allotted sequences, they return an idle message to the master. The master process can then assign each worker a fragment of the database to search or copy into local storage. As each worker completes its task, it will send an idle message to the master, and the process repeats until the search is completed [8, 4].

The original system design, as presented in mpiBLAST-0.9, requires the master to collect and sort partial results from the worker processes, when they have completed searching [4, 12]. The master must then read data required for result analysis from the database on shared storage. Once the results have been analysed and formatted, they are output and execution is finished.

3.2 mpiBLAST Development

Many improvements and developments have been made to the original system design. One performance limiting issue recognised in earlier releases is the static partitioning of the database. Firstly, this is inflexible, and re-partitioning is required to run on a different number of processors [13, 3]. Secondly, it can be inefficient if the number of database fragments is more or less than the number of processing nodes [4, 14].

In the original release, inefficient data handling meant a large and rapidly increasing non-search overhead as the number of processes / number of database fragments increased, and also as the output size increased [13, 3]. Once the search was complete, worker processes returned all results to the master for processing, which was inefficient, as the master node was left to process results while none of the other processes are working.

Despite these drawbacks, however, it still managed super linear speedup when using a small number of processes [8, 13].

3.3 pioBLAST

pioBLAST is a research prototype based on mpiBLAST, described in [13], and first published in 2005. It aimed to address some of mpiBLAST's performance and scalability problems by further parallelising the algorithm. pioBLAST employs dynamic database partitioning, to address the above mentioned issues with static partitioning. In the pioBLAST model, the master process sends workers two offsets to specify a range for its allocated database fragment, thus avoiding actually generating any physical database fragments. The worker processes can then use the file ranges to read in their allotted segment into local memory using MPI-IO [13, 4] .

Another added feature to help improve efficiency is was the introduction of results caching. When results are discovered, workers cache potentially useful sequence information so it need not be retrieved from disk again later, reducing communication between processes [4, 14].

pioBLAST also saw the parallelisation of results handling. When the search is complete, the master notifies the workers of which results should be included in the output, and the workers do the actual writing [4, 16]. It also implements collective writing I/O in parallel, where the underlying filesystem will support it.

These improvements to mpiBLAST dramatically improve its scalability. In [13], mpiBLAST-1.2 and pioBLAST are tested on the IBM Blade Cluster at the High Performance Computing Center, North Carolina State University. Each took a

150KB set of randomly sampled queries, and set them against the GenBank nr database. As can be seen in 3.1, the non-search overhead is significantly reduced in pioBLAST, and stays low, when scaled to more processes.

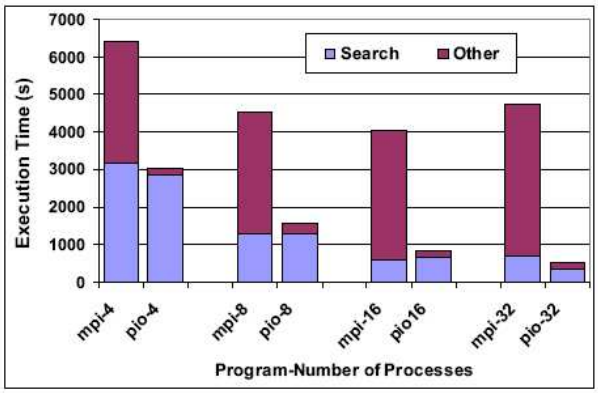

Fig. 3.1: A comparison of mpiBLAST-1.2 and pioBLAST search and overhead time. The time taken by the BLAST algorithm (shown in blue) does not vary between the two, as the underlying BLAST implementation is the same. However, note the overhead in time taken for data handling and I/O (purple) is significantly reduced in pioBLAST.

3.4 mpiBLAST-1.4

While pioBLAST was being developed, mpiBLAST-1.4 was released, incorporating improvements of its own. Most significantly, it implemented query segmentation as well as database segmentation. This was a significant advancement, and the combination of both database and query segmentation has several advantages.

Firstly, by duplicating certain fragments across nodes, and dividing queries between them, the overhead associated with database fragmentation can be reduced [4, 17].

Query segmentation can also assist in balancing load if certain database fragments require long times to search. If workers finish with their database fragment, they can take on the fragment of another worker and search different queries against it.

3.5 mpiBLAST-PIO

The latest release of mpiBLAST is mpiBLAST-1.5-pio. It builds on mpiBLAST-1.4, incorporating query segmentation, as well as some of the strategies used in

pioBLAST, such as using workers to process output and to write in parallel if the underlying filesystem supports it 3.2.

The scalability of mpiBLAST is much improved in this latest version - this was proved by researchers from North Carolina State University, who last year enabled mpiBLAST to scale to almost 33,000 cores of the IBM Blue Gene/P system, with 93% efficiency [14, 1]. In fact with a few optimisations, they used it to search an entire genome against itself (done in part to discover missing genes) in twelve hours. This problem was previous considered computationally intractable [14, 1].

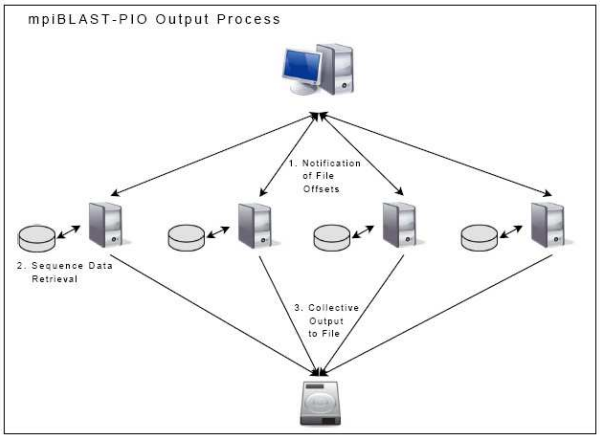

Fig. 3.2: The output model of mpiBLAST-PIO, which uses collective, parallel writing of results.

3.6 Future Directions

Despite the significant improvements made to mpiBLAST over the past few years, there is still room for development in certain areas. One issue that could be fixed with minimal effort is the lack of fault tolerance when a node goes down. Darling et al. [8] offer a solution whereby a node periodically sends messages to the master process, to let it know the node is still alive and searching. Should the master fail to hear from a certain node after a specified time, the master could dynamically redistribute that nodeŠs workload to another node. Also suggested are database updates, whereby nodes check a central repository of version information to ensure their database is up to date [8, 12].

Goddard [4] suggests removing the serialisation of worker requests at the master, and implementing task threading. This could reduce the bottleneck that occurs at the master process, which may leave processes waiting for their next task while the master processes sends another process a new fragment or query.

Chapter 4

Conclusion

DNA sequence comparisons are perfectly suited to parallelisation, as both the input queries and the database of sequences to be searched through can be segmented and split to run on separate processers. mpiBLAST has achieved super linear speedup of the sequential sequence comparison algorithm BLAST, and continues to incorporate improvements as research into better parallelisation techniques continues.

There are many other parallel BLAST implementations not mentioned in this report. mpiBLAST-2.0 is in development, and is discussed in detail in [11] and [12]. Based on his analysis of the strengths and weaknesses of mpiBLAST, Goddard [4] has implemented a generic grid framework for sequence searching, called gridRuby, which allows the automatic parallelisation of existing sequential applications.

The staggering amount of research that continues to be published in this field reflects the importance of the subject. Important biological advancements in analysing diseases and developing new drugs and vaccines would not be possible, were it not for the computational tools that allow biologists to sift through the overwhelming amount of data they are presented with today. This is certainly a field that highlights the importance of the continued development of parallelisation and parallel computing techniques.

Bibliography

[1] Just the facts: A basic introduction to the science underlying ncbi resources. http://www.ncbi.nlm.nih.gov/About/primer/bioinformatics.html (2004)

[2] Braun, R.C., Pedretti, K.T., Casavant, T.L., Scheetz, T.E., Birkett, C.L., Roberts, C.A.: Parallelization of local blast service on workstation clusters. Future Generation Comp. Syst. **17** (2001) 745–754

[3] Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipmanl, D.J.: Basic local alignment search tool. Journal of Molecular Biology **215** (1990) 403–410

[4] Goddard, C.J.: Analysis and Abstraction of Parallel Sequence Search. PhD thesis, Faculty of the Virginia Polytechnic Institute and State University, Blacksburg, Virginia (2007)

[5] Benson, D.A., Karsch-Mizrachi, I., Lipman, D.J., Ostell, J., Wheeler, D.L.: Genbank. Nucleic Acids Research **36** (2008) 25–30

[6] Help with matrices used in sequence comparison tools. http://www.ebi.ac.uk/help/matrix.html (2009)

[7] Grant, J.D., Jr., R.L.D., Manion, F.J., Ochs, M.F.: Beoblast: distributed blast and psi-blast on a beowulf cluster. Bioinformatics **18** (2002) 765–766

[8] Darling, A.E., Carey, L., Feng, W.C.: The design, implementation, and evaluation of mpiblast. In: In Proceedings of ClusterWorld 2003. (2003)

[9] Chang, C.: Blast implementation on bee2. Technical report, University of California, Berkeley (2004)

[10] de Carvalho Costa, R.L., Lifschitz, S.: Database allocation strategies for parallel blast evaluation on clusters. Distributed and Parallel Databases **13** (2003) 99–127

[11] Archuleta J.S., Feng W., T.E.: A pluggable framework for parallel pairwise sequence search. In: International Conference of the IEEE Engineering in Medicine and Biology Society. (2007)

[12] Archuleta J.S., Tilevich E., F.W.: A maintainable software architecture for fast and modular bioinformatics sequence search. In: 23rd IEEE International Conference on Software Maintenance. (2007)

[13] Lin, H., Ma, X., Chandramohan, P., Geist, A., Samatova, N.F.: Efficient data access for parallel blast. In: IPDPS, IEEE Computer Society (2005)

[14] Lin, H., Balaji, P., Poole, R., Sosa, C., Ma, X., chun Feng, W.: Massively parallel genomic sequence search on the blue gene/p architecture. In: SC, IEEE/ACM (2008) 33