Bioinformatics Clusters in Action

Marc A. Rieffel, Tristan G. Gill, and William R. White

Paracel, Inc.
1055 East Colorado Blvd.
Suite 410
Pasadena, CA 91106
USA

PARACEL
Applied High-Performance Computing

Portions of this article appeared in ClusterWorld, March 2004

Bioinformatics Clusters in Action

Marc A. Rieffel, Tristan G. Gill, and William R. White†

Due to the growing volume of DNA and protein sequence data, researchers are turning to Linux clusters for their bioinformatics research. A few years ago researchers were analyzing sequences in the kilobase range. (A 'kilobase range' refers to thousands of base pairs.) Today it is routine to analyze sequences of multiple megabase size and to perform these analyses across the genomes of many related organisms simultaneously. The amount of sequence data in public databases is doubling approximately every nine months. The work required to compare each sequence against all others is growing as the square of their size. Even the optimistic 18-month doubling of compute power predicted by Moore's law isn't enough to keep pace.

1. Introduction

High-performance parallel computing is one approach to handling the large database and high throughput requirements by splitting up large, complex tasks across multiple processors. Linux clusters, which are assembled from commercial off-the-shelf hardware, provide a cost-effective and scalable solution to these problems.

While Linux clusters can provide the computing power to handle large-scale sequence comparisons, a number of important factors must be considered when designing and implementing a bioinformatics cluster. Here we will look at some of the challenges encountered when using BLAST, the popular used bioinformatics search algorithm, and we'll see how to address these challenges so as to make full use of available cluster resources. The methods are often applicable to other high-performance software applications installed on Linux clusters. Principal challenges are:

1. Single large problems that must be divided and then reassembled,
2. Problems larger than node memory,
3. Load balancing, and
4. Large numbers of small tasks.

2. BLAST Overview

BLAST (Basic Local Alignment Search Tool) is a sequence comparison algorithm that makes use of a hash table, or index, to identify initial matches, and then applies a more rigorous algorithm to construct the final sequence comparison. BLAST was originally developed to run on desktop machines and small SMP systems when query sequences and sequence databases were small. BLAST has several algorithmic limitations that limit its applicability for large query sequences and databases. In multi-user environments, its computational demands easily exceed those available on desktop computers and SMP systems. NCBI BLAST, the most commonly used BLAST implementation, is not a parallel application, so it cannot make use of cost-effective cluster computing technology. In this article we discuss a BLAST implementation that exploits the capabilities of Linux cluster technology.

BLAST executes comparisons of query sequences against sets of database sequences. (These may be either nucleotides or amino acid sequences.) The key to the BLAST heuristic is that a statistically significant alignment is likely to contain a high scoring pair (HSP) of aligned 'words'. Using the index as a lookup table, BLAST first searches for short exact matches between the query and database sequences. (The typical default word length is 11 for nucleotides and 3 for amino acids.)

These hits are then extended, first without allowing for gaps, then allowing for them, and HSPs that meet a specified threshold are returned as statistically significant.

The extension part of process step is computationally very intensive, so BLAST avoids excessive computation by doing full alignments only on those pairs that, based on lookup table, are statistically likely to be HSPs.

3. Parallelism

Parallelism of an existing application is usually somewhat challenging. Fortunately, there are a number of ways to parallelize the BLAST algorithm, including job parallelism, query parallelism, database parallelism and query chopping. Each of these is considered in more detail.

3.1. Job Parallelism

The easiest way to achieve parallelism in a multi-user environment is by running each user's job on a different processor.

† Corresponding author: *rieffel@paracel.com*

This is sometimes referred to as 'trivially parallel', though in practice it can be difficult to achieve. The complexity of this approach lies in scheduling jobs so that each user receives a fair share of the cluster and no processors are overloaded.

One approach is to use a job or batch scheduler, such as Sun Grid Engine (SGE), Platform Load Sharing Facility (LSF), or Portable Batch System (PBS). In fact, many bioinformatics installations have chosen to use NCBI BLAST with a scheduler. This approach is reasonable on small clusters running small jobs, but it has inherent limitations of scalability and it does not address the inability of NCBI BLAST to run large jobs. Because there is little interaction between the BLAST application and the scheduler, it is difficult for the scheduler to allocate resources optimally.

Another approach is to develop a custom job scheduler. This is the approach we undertook for Paracel BLAST. The benefit of a custom scheduler is that it can be tightly coupled with the application, and know the detailed requirements of each job. Thus, it is able to make finely tuned resource allocations, which improves performance, efficiency, and scalability.

Job parallelism is by itself not adequate because it does not allow for any single job to benefit from the availability of multiple processors. A single large job may take hours or days to run on a single processor while the other processors in the system remain idle. For this reason it is necessary to find ways to parallelize individual jobs.

3.2. Query Parallelism

In query-parallelized BLAST searches, multiple query sequences are distributed among different processors. This is an easy implementation for BLAST because there is no dependency between searches of different query sequences. The only interaction or communication between processors that is required is to ensure that the search results for the different queries are returned to the user in the proper order.

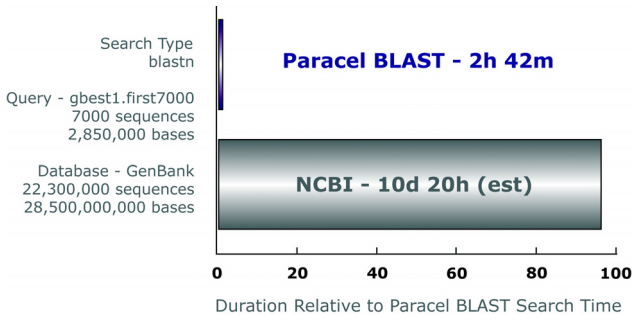
A query-parallel implementation is adequate for many BLAST applications. High performance and scalability can be achieved for searches with numerous query sequences against databases of moderate sizes. Several implementations of this approach are available for use with different batch schedulers. Query parallelism can even be achieved with simple 'wrapper' scripts and NCBI BLAST. This approach alone, however, still does not address the limitations of NCBI BLAST on large jobs. Also, the choices of which queries to assign to which processor must be made carefully.

It is important that the scheduler make these decisions dynamically, based on the attributes of each job and the current state of the cluster. For example, in order to reduce overhead and increase throughput, a job that would be highly partitioned to make use of an idle cluster might not be partitioned at all on a heavily loaded cluster. Of course, query parallelism is not applicable to single-query searches.

3.3. Database Parallelism

Another way to achieve parallelism with a BLAST search is by having different processors search separate pieces of the database simultaneously. For single-query searches, this may be the only way to achieve parallelism. For searches of large databases, database parallelism offers some additional performance benefits that are described below.

FIGURE 1: Database Parallelization



Database parallelization on a cluster enables searches to be completed in hours that would take days on a conventional computer.

It is more difficult to implement database-parallelism than query-parallelism or job-parallelism because there are dependencies between the results of the different sub-searches. Specifically, each sub-search will compute the top N matches in its section of the database, but these matches must be merged, re-sorted, and truncated to produce a global top N. Some simple wrapper scripts divide databases into multiple pieces and perform a separate search of each piece. This results in a separate search result for each database piece and users are sometimes required to merge these results manually.

Another approach uses simple scripts that parse, sort and cull the partial outputs and generate merged output reports. This approach has several limitations. First, the scores in a text report are only determined with a few digits of precision, making it difficult to reproduce the same ordering of similarly-scoring hits as in an unpartitioned search. Second, the variety of BLAST output report formats forces these merge scripts to become unwieldy or to support only a small subset of the available output formats. Lastly, the time to parse, process, and regenerate large BLAST output reports can consume a

large fraction of the total search time, limiting performance and scalability.

Paracel BLAST avoids these limitations by performing the merge 'internally'. That is, sub-jobs produce intermediate results in data structures with full machine precision. These intermediate results are sorted and merged with the same NCBI routines that are used in sequential runs. While the different order of operations can yield different sorting of hits with the same scores when compared to sequential runs, in practice, this approach produces output that is very nearly identical to that of sequential runs. Performance is better than that of 'external' or script-based approaches since no parsing is required and the output report is only generated once.

A further enhancement in Paracel BLAST is 'dynamic' database splitting. Instead of making database splitting decisions at the time of database loading (or formatting) and creating separate files for each piece of the database, Paracel BLAST creates a single database. Each process simply reads the piece that it needs out of the big file. This facilitates database management and allows the integrated scheduler to make database splitting decisions on the fly based on its knowledge of the job and the state of the system.

Another approach to database parallelism is more hardware related. In this case, the database is split nto as many pieces as there are nodes in the cluster, and each node is assigned to search all query pieces against one section of the database. While this approach can demonstrate a significant increase in performance, it may not always be the most efficient one possible. With large databases and a small number of nodes, the size of each database piece may still be too large to fit in memory reliably. Similarly, small databases on large clusters may be partitioned so finely that the additional cost of starting jobs and merging results exceeds the benefit of the parallelism. Finally, non-uniformity of the database pieces may result in load imbalance and poor utilization. For example, GenBank splits the database by organism. When running GenBank searches one node may receive a large part of the human database while other nodes would receive substantially smaller microbial organisms. The node searching the human database would take a much longer time to complete its task, leaving the other nodes idle.

Alternatively, you can adjust database partitioning so that there are enough pieces to benefit from parallelism: each piece is small enough to fit into RAM easily, but the database is not split into more pieces than is beneficial. By employing database parallelism, you can ensure that each process search only as much of the database as it can comfortably hold in RAM. This guarantees that memory mapping will succeed and that the search will have rapid access to all of the data that it needs. Thanks to the operating system's disk caching system, a node that searches a part of the database is likely to still have that part of the file in RAM so that a subsequent search of the same file will not have to load it from disk. The Paracel BLAST scheduler makes use of this fact by scheduling jobs on nodes that have recently searched the same database pieces.

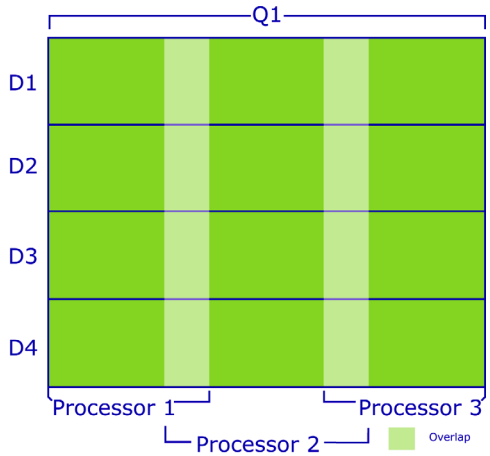
It is interesting to note that the combination of database partitioning and careful job scheduling can result in a super-linear speedup. This can happen when a job has to read a database from disk because it is too large to fit into memory. The database can be broken into pieces small enough to fit into memory. The speedup is particularly striking on large jobs that have been broken into many pieces – the first pieces run at the same speed as would be expected based on simply dividing by the number of processors. Subsequent pieces can use the cached database segments and are much faster. The overall job time (for longer jobs) therefore tends toward the RAM access time, not disk access time. Again, this occurs because the Paracel BLAST scheduler has been programmed to assign different sub-jobs that use the same database piece to the same processor whenever possible.

4. Query Chopping

For searches of large query sequences, another form of parallelism, 'query chopping', can be employed. In query chopping, an individual query sequence is segmented, and a separate processor searches each segment.

Query chopping is the most complicated parallelism approach for BLAST and is most beneficial when applied to searches that NCBI BLAST is unable to complete.

FIGURE 2: Query Chopping



The Paracel BLAST query chopping algorithm accurately deals with edge effects of alignments that span overlapping query segments.

NBCI BLAST typically does not perform well on query sequences larger than about 100 kilobases, and often runs out of memory with query sequences larger than 1 megabase.

Several script-based approaches have been used to implement query chopping. Here a complete BLAST search is run for each overlapping piece of the query, the text results are parsed and merged, and a final output is generated. These approaches have the same limitations as with database-parallelism – specifically problems with parsing, loss of accuracy, and decreased performance. In addition, they are limited in their ability to handle the "edge effects" of alignments in or near the overlapping regions. Edge effects can cause some alignments to be reported multiple times in slightly different ways and others to be lost completely.

Paracel has taken an alternative code-based approach that directly handles the edge effects and reprocesses them as appropriate. After all HSPs are generated, they are broken down into three categories: HSPs that are entirely contained in the non-overlapping regions, HSPs entirely contained in the overlap, and HSPs partially in the overlap and partially in the non-overlapping part.

The first class of HSPs presents no problems. The second class will be found twice – once for the left piece, once for the right. Other than making sure that they are reported only once, they present no special problem. The third class is the most difficult. Any alignment which is partly in an overlap, but which stops well short of a neighboring piece, would have stopped short of that piece if no chopping had occurred. The troublesome HSPs are those that encompass an entire overlap. Such (rare) HSPs will be cut off by chopping, but can be detected as HSPs in the chopped sequences that hit both ends of the overlap.

Query chopping detects such HSPs and goes back to the original hit that led to that HSP. The original hit is extended into an HSP in the context of the unchopped query. Note that although this operation is computationally expensive, it is exceedingly rare – it only occurs when there is an HSP that fully spans the overlap. We use a default overlap of 10 kilobases, so the time to recalculate these is typically very small compared to the savings gained by being able to split the query more finely and farm it out to multiple processors.

5. Handling Large Numbers of Short Queries

For searches with many small query sequences, such as primer or probe searches, a more efficient use of cluster resources is to search many queries simultaneously in one pass of the database. The efficiency comes from the fact that with short query sequences, the query hash table is very small and little computation is required for each section of the database that is being scanned. Several approaches have been used to address the 'short query bottleneck'. MegaBlast was developed to provide faster nucleotide vs. nucleotide searches for many small queries. There are also several shareware scripts that 'pack' queries together. For example, a script can insert 'don't care' characters between short queries, run a single search, then post-process the results to correct for the inserted characters. (The 'don't care' characters are used to make it unlikely that a hit would occur across queries.) The problem with MegaBlast and these scripts is that accuracy is lost – the statistics mean different things for a single long query and many short queries.

Paracel has taken a different 'code level' approach by developing a Query Packing Algorithm to maintain statistical validity with improved speed. Instead of building a hash table for a single query, scanning the whole database, and then starting again for the next query, Paracel BLAST creates a hash table with information about several query sequences, and handles all of these sequences with a single pass of the database.

This optimization involves modifications to the way the hash table is populated and the way hits are read out of the hash table – all subsequent processing on hits remains the same. The key is to populate the hash table with data from a packed pipeline of several queries, enough so that the hash table will be relatively full without being overcrowded. This improves performance of sets of small queries significantly.

6. Partitioning

Job partitioning on a multi-processor system must provide adequate parallelism and ensure load balance without creating unnecessary overhead. Load balancing is a key to performance enhancement. If a job that takes 100 minutes on a single processor is equally distributed among 50 processors, the job will take two minutes. However, if 49 processors receive pieces requiring 1 minute, and the fiftieth receives a piece requiring 51 minutes, the total problem will take roughly half the time required on a single CPU. Because of poor load balancing, the job does not approach the 50X speedup expected based on processor number alone.

To avoid this kind of pathological case, it is desirable to divide a job into as many pieces as possible. Since it is not known *a priori* which pieces will take the most time (a significant fraction of processing time depends on the number of hits, which is not knowable before running the search), dividing the job into more pieces will allow the faster pieces to finish sooner. Those processors may then be used for the remaining pieces. One might think that by breaking a job into extremely small pieces it would be possible to achieve the actual speedup equal to the number of processors. In practice, however, there is some amount of overhead associated with each piece, so there is a limit to how fine-grained the division should be.

Partitioning must take into account the number of machines: partitioning into too few pieces will leave machines idle, whereas overpartitioning – portioning into more pieces than machines – might be desirable to improve load balancing.

7. Coexisting with an IT Infrastructure

While some laboratories can afford dedicated resources for BLAST searches, many must make their computing resources available for other bioinformatics or general purpose computing tasks. One of the benefits of the Linux cluster approach to bioinformatics research is that the clusters can be easily configured to carry out BLAST searches and, on short notice, other computational tasks as well.

8. Case Study: *Schistosoma mansoni*

The investigation of the *Schistosoma mansoni* transcriptome at the University of São Paulo, Brazil exemplifies how Paracel's BlastMachine2 (a Linux cluster pre-configured with parallelized Paracel BLAST software) can accelerate bioinformatics research.

The University of São Paulo, in collaboration with a team of researchers from the University of Iowa and the University of York in the United Kingdom, has set out to study the evolution and biology of the parasite *Schistosoma mansoni*, the leading cause of schistosomiasis. Schistosomiasis is one of the world's biggest public health problems – affecting 200 million individuals in developing countries – with no vaccines available. One of the research goals is to identify novel proteins to be investigated as vaccine candidates and potential drug targets.

To better understand the cellular and physiological processes in *S. mansoni*, the multi-center effort has obtained and annotated extensive transcriptome data for the organism utilizing normalized cDNA libraries from adult parasites and ORESTES mini-libraries from six life-cycle stages. The work has been published in the October, 2003 issue of *Nature Genetics*.

The University of São Paulo chose Paracel's BlastMachine2 for the analysis of *S. mansoni* expressed-sequence tags (ESTs). In total, 163,000 EST reads were generated from the research. Using the BlastMachine2, the São Paulo researchers were able to run BLASTN and BLASTX algorithm searches against a local copy of the GenBank database quickly and accurately.

"By developing this first large-scale database of S. mansoni, we can enhance our understanding of the evolution, biology and adaptation to parasitism which ultimately will aid in treating and curing infections," said Sergio Verjovski-Almeida, Professor of Biochemistry at the University of São Paulo. *"The BlastMachine2 enabled our researchers to quickly scan for contaminants, mitochondrial, ribosomal, or transposon sequences to eliminate these from further studies. It further allowed us to assign putative protein products to our EST assemblies, as well as Gene Ontology classifications. By analyzing similarities between EST contigs and known proteins, a list of potential drug targets was derived for further analysis in the laboratory."*

9. Case Study: The Jackson Laboratory

The Roopenian Lab at The Jackson Laboratory is pursuing research in several areas involving gene expression. In addition to large-scale quantitative PCR, the Lab is deriving global expression patterns from several preexisting expression databases. These databases, which approach the capacity to provide expression for nearly all genes within a genome, can be organized in a way that allows a visual interpretation of gene expression within the context of entire chromosomes. It is then possible to look at chromosomes and recognize clustered regions that exhibit tissue-specific expression patterns.

In June, 2003 the Jackson Laboratory purchased a BlastMachine2 and also installed Paracel BLAST software on an existing 64-CPU Linux cluster to support its bioinformatics

research including the analysis of RNA transcripts and their alternative splice forms. The sequence databases and transcripts that are aligned against them are enormously large and require supercomputing speed, as well as efficient algorithms to accurately align sequences and parse out large tasks to multiple nodes. One application in which Paracel BLAST is used is to cluster mouse and human ESTs against their respective genomes. When ESTs have been localized to their correct positions, the Lab can extrapolate gene expression information with knowledge of the tissue source of the EST, and thus create chromosomal expression maps. This involves blasting approximately 3 million mouse or 5 million human ESTs against heavily masked genomes.

"When comparing Paracel BLAST against MegaBlast for this task, we found that MegaBlast took 2.5 days to accomplish what Paracel could do in hours. This is a significant time savings, especially when this procedure must be re-run for each BLAST parameter adjustment and periodic new releases of the EST and genomic databases. The output generated from Paracel BLAST is also in a format that allows easy integration into our downstream programming modules for analysis of the data," said Aaron Brown, computational biology researcher at The Jackson Laboratory.

In addition to clustering ESTs, the Lab uses Paracel BLAST to cluster Lynx MPSS™ sequence tags of 17 to 20 bases against masked genomic and mRNA sequences. By means of this clustering, the Lab can create chromosomal expression maps in much the same way as with much longer ESTs. The difference is that here the expression map involves a depth of millions of tags for a single tissue.

"Because of their short length, optimization of BLAST parameters is needed to filter out poor matches and localize somewhat redundant tags to their correct position in the genome. Paracel is quite accurate when splitting and aligning tags that are not a perfect match (such as an exon split), when compared to NCBI BLAST programs we have tested. In addition, Paracel lets us fine tune, constantly changing BLAST parameters, by performing blasts of hundreds of thousands of tags often within minutes," Brown added.

10. Conclusion

This article has described the process of turning a public domain sequential application into an optimized parallel application. A series of techniques for exploiting parallelism was presented. System-wide considerations, including partitioning, scheduling, resource allocation, and communications have been explored. The resulting hardware/software solution has performed well on benchmarks and in customer applications such as those cited in the case studies.

At each development stage we presented multiple approaches, ranging from wrappers and scripts to complex algorithmic code modifications. Developers in other application areas are likely to face similar choices. Our experience in developing Paracel BLAST argues that it is worth spending the time to develop code that exploits the full capabilities available on Linux clusters.

Resources:

Intel C++ Compiler for Linux:
http://www.intel.com/software/products/compilers/clin/

The Jackson Laboratory:
http://www.jax.org/staff/roopenian/labsite/

Lynx Therapeutics, Inc.
http://www.lynxgen.com

NCBI BLAST:
http://www.ncbi.nlm.nih.gov/BLAST/

Paracel BlastMachine:
http://www.paracelblast.com

University of São Paulo:
http://rbi.fmrp.usp.br/