33rd Annual International Conference of the IEEE EMBS
Boston, Massachusetts USA, August 30 - September 3, 2011

Design of Multiple Sequence Alignment Algorithms on Parallel, Distributed Memory Supercomputers

Philip C. Church, *Student Member, IEEE*, Andrzej Goscinski, Kathryn Holt, Michael Inouye, Amol Ghoting, Konstantin Makarychev, and Matthias Reumann, *Member, IEEE*

***Abstract*—The challenge of comparing two or more genomes that have undergone recombination and substantial amounts of segmental loss and gain has recently been addressed for small numbers of genomes. However, datasets of hundreds of genomes are now common and their sizes will only increase in the future. Multiple sequence alignment of hundreds of genomes remains an intractable problem due to quadratic increases in compute time and memory footprint. To date, most alignment algorithms are designed for commodity clusters without parallelism. Hence, we propose the design of a multiple sequence alignment algorithm on massively parallel, distributed memory supercomputers to enable research into comparative genomics on large data sets. Following the methodology of the sequential *progressiveMauve* algorithm, we design data structures including sequences and sorted k-mer lists on the IBM Blue Gene/P supercomputer (BG/P). Preliminary results show that we can reduce the memory footprint so that we can potentially align over 250 bacterial genomes on a single BG/P compute node. We verify our results on a dataset of E.coli, Shigella and S.pneumoniae genomes. Our implementation returns results matching those of the original algorithm but in 1/2 the time and with 1/4 the memory footprint for scaffold building. In this study, we have laid the basis for multiple sequence alignment of large-scale datasets on a massively parallel, distributed memory supercomputer, thus enabling comparison of hundreds instead of a few genome sequences within reasonable time.**

I. Introduction

COMPARATIVE genomics relies heavily on the alignment of multiple genomes. With recent advances in microbial diagnostics, typing and surveillance, comparative genomics is playing an increasingly important role in epidemiology, pathogen evolution and the fight against drug resistance. One way to characterize fine-scale genomic variation and confidently infer drug resistance mutations is through aligning and comparing many genomes. Therefore, there is a pressing need for computationally efficient algorithms and tools which are able to scale with current and future genomic datasets.

There are many methods of sequence alignment, common algorithms include: *ClustalW2* [2]; *MUSCLE* [3]; *T-Coffee* [4] and p*rogressiveMauve* [5]. The latter is a sequence aligner used predominantly for bacteria. It can be used to find genome re-arrangements and aligns to a scaffold of conserved regions. Alignment makes use of a tree method similar to *ClustalW2* and *MUSCLE*. The *progressiveMauve* algorithm is ideal for analyzing population diversity as genome rearrangements are taken into account during alignment. However, the current implementation is both computation and memory intensive.

It is common for alignment methods to trade between speed and accuracy. Exponential amounts of random access memory (RAM) with respect to input data size are often a key requirement. When aligning large genomic populations, RAM can become a problem. While virtual memory can be used, it often results in a slowdown of computation. Thus, considering that genomic data will only increase in size and volume in the future, the problem of multiple sequence alignment will need high performance computing (HPC) systems.

To our knowledge, there is no scalable alignment method available that accurately measures genetic diversity. Further, most alignment algorithms are designed for shared memory computers. These will require terabytes of RAM to carry out multiple sequence alignment of hundreds of genomes. Designing an algorithm for massively parallel, distributed memory machines such as the IBM Blue Gene/P supercomputer (BG/P) will create opportunities for fast, high throughput multiple sequence alignment in the future.

In this study we present our design and concept as well as preliminary results of a multiple sequence alignment algorithm for massively parallel, distributed memory HPC systems.

II. Materials and Methods

We choose to parallelize the *progressiveMauve* algorithm due to its advantages regarding analyzing population diversity. In addition, comparing our results with the original output serves to validate our method. First, we analyze *progressiveMauve* with respect to regions that have to be carried out sequentially

Manuscript received March 24, 2011. This research was supported by Victorian Life Sciences Computation Initiative (VLSCI) grant numbers VR0126 and VR0082 on its Peak Computing Facility at the University of Melbourne, an initiative of the Victorian Government.

P. C. Church is with Deakin University, Science and Technology (corresponding author; phone: +61 3 52271399, e-mail: pcc@deakin.edu.au).

A. Goscinski is with Deakin University, Science and Technology (e-mail: andrzej.goscinski@deakin.edu.au).

K. Holt is with the Dept. of Microbiology and Immunology at the University of Melbourne, Carlton, VIC, Australia (e-mail: kholt@unimelb.edu.au)

M. Inouye is with the Walter and Eliza Hall Institute of Medical Research, Parkville, VIC, Australia and with the Dept. of Medical Biology at the University of Melbourne, Parkville, VIC, Australia (e-mail: inouye@wehi.edu.au)

A. Ghoting and K. Makarychev are with the IBM T. J. Watson Research Center, Yorktown Heights, NY, USA (e-mail: {aghoting; konstantin}@us.ibm.com).

M. Reumann is with the IBM Research Collaboratory for Life Sciences-Melbourne, Carlton, VIC, Australia and the Dept. of Computer Science and Software Engineering, University of Melbourne (e-mail: mreumann@ieee.org).

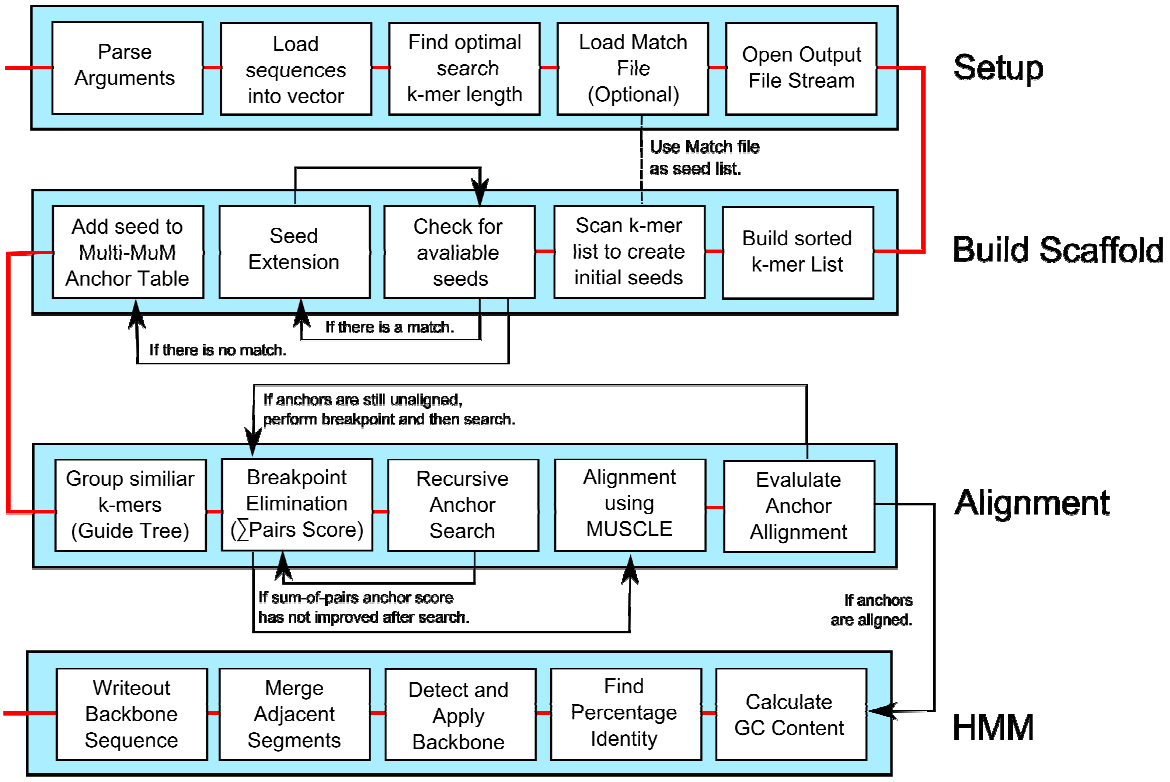

Fig. 1: Flowchart detailing the operation of the *progressiveMauve* algorithm.

and that can be carried out in parallel. Then, we carry out a performance analysis to investigate computation intensive regions in the algorithm. Finally, we present preliminary results for setup and scaffold formation of a distributed, low memory consuming version of the *progressiveMauve* algorithm designed for the IBM BlueGene/P supercomputer.

A. Overview of progressiveMauve

progressiveMauve was developed at the University of Wisconsin. This algorithm aligns conserved regions of a genome allowing for accurate alignments between closely related genomes and understanding of rearrangement history. Operation of the *progressiveMauve* algorithm (Fig 1.) is as follows:

progressiveMauve begins by loading sequence data, stored in FASTA format, into memory. The average sequence length is used to find the optimal k-mer length. *progressiveMauve* makes use of this k-mer size to create a spaced seed mask. Spaced seed masks are patterns which indicate locations where genetic mutations are allowed. Spaced seeds are used to locate k-mers which are not continuous, this is necessary when looking for genome re-arrangements [6]. A family of spaced seeds can be used to improve the accuracy of the k-mer list. A k-mer list is then sorted so that matching sequences are next to each other.

Once the sorted k-mer list has been generated, a reference genome called a scaffold is created. Scaffolds are built from k-mers that exist in two or more sequences. These k-mers are then called seeds. Each seed is extended in both directions until no further seeds are found.

Alignment utilizes a *MUSCLE* derived algorithm where distance values are generated for each k-mer. Un-aligned sequences with similar distance values are anchored to the previously derived scaffold. To speed up alignment, sum-of-pair greedy breakpoint elimination values are used to find related anchors.

Finally, a Hidden Markov Model is used to remove bias in areas containing differential gene content. The model makes predictions of pairwise homology. Regions found to be unrelated are removed from the final alignment.

B. Design Considerations

When parallelizing the *progressiveMauve* algorithm, hardware considerations must be taken into account. We choose to design the algorithm for the IBM BlueGene/P supercomputer (BG/P) [7] because it is a massively parallel, distributed memory system. BG/P consists of 1,024 compute nodes in a single rack connected through a high speed communication network. Multiple racks of BG/P can be connected together to form a larger computer. We carry out our simulations on the Victorian Life Sciences Computation Initiative (VLSCI) that comprises two racks of BG/P with 2,048 compute nodes, i.e. 8,192 cores. For our development we only used a partition of 64 compute nodes (256 cores), the smallest partition on BG/P at VLSCI. Each compute node has four 850 MHz PowerPC cores connected to a local memory unit of 4GB. BG/P can be configured to run in one of three modes. *SMP* mode has one process per node, up to four threads per process and 4GB RAM per process. *Dual* mode

gives two processes per node, up to two threads per process and 2GB RAM per process. *Virtual Node (VN)* mode uses four separate single-thread processes with 1GB RAM each.

Persistent storage is accessible through a few dedicated I/O nodes. As there is no internal storage, virtual memory is not available. Due to these hardware limitations it is necessary to devise methods that store data efficiently and limit the number of I/O operations.

Our algorithm currently runs in VN mode that maximizes compute power but limits the size of each FASTA file to 450 million base-pairs. It is possible to increase the available RAM by using SMP mode; however adoption of hybrid programming models that increase design complexity will be required to obtain computational efficiency.

C. Basic Design

When distributing the *progressiveMauve* algorithm onto a parallel, distributed memory computer we adopt a master-worker approach for parallelization. Its advantage is that it allows for more control over memory and computational allocation compared to other strategies. As this implementation is to be optimized for the BG/P, the master-worker approach was chosen. Further, its implementation is straight forward which is desirable for a first design.

In a master-worker approach, a single master node distributes data and jobs to free worker nodes. The framework used in the distributed *progressiveMauve* algorithm (Fig. 2) is based around sending commands that are interpreted by a worker node. This communication framework also supports message sending between worker nodes, which is used to transfer sequence data. In this case, the master node sends two messages: one for a worker to listen to a specified rank and one for the other worker to send to the listening node.

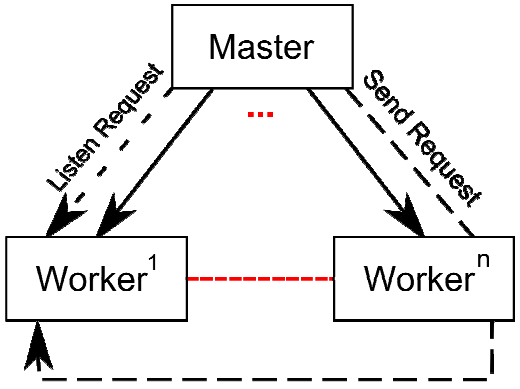

Fig. 2. Comunication framework utilized by the distribued *progressiveMauve* algorithm.

D. Implementation

The distributed *progressiveMauve* algorithm starts by loading sequence data into RAM. Due to memory limitations, each FASTA file is loaded onto an individual compute node corresponding to a single MPI task. Once created, these nodes have the ability to send specified portions of a sequence to other MPI tasks.

The generation of the sorted k-mer list is parallelized as follows. Each sequence is broken up into smaller fragments that are distributed to free worker nodes. To ensure accuracy of returned k-mers it is important for each sequence fragment to include the maximum k-mer length. After receiving data, each node constructs a seed mask, used to find and separate k-mers. K-mer lists are sorted and serialized before being sent to the master node. While the merging of the lists can be carried out in parallel as well, we chose to have the master node do the merging of the k-mer lists, keeping the basic implementation as simple as possible while running on a parallel, distributed supercomputer. Memory overhead of the sorted k-mer list has been reduced by dynamic condensing of like mers. Links are then made between the unique mers, sequence positions and FASTA files. This methodology removes repetitive data that is an advantage when aligning similar sequences. The final sorted mer list consists of unique mers and the number of sequences they appear in.

The current implementation has also been simplified in order to improve code readability and installation. The distributed algorithm consists of 7 files (reduced from 388 files) and removed all non-standard dependencies (such as Boost and pkg-config). The modifications made to the software have also removed a number of redundant variables in order to trim memory usage.

E. Data Sets and Benchmarking

We first carry out performance testing of the *progressiveMauve* algorithm with respect to setup, scaffold building, alignment and HMM run times. We further profile the memory footprint used by the algorithm. We tested our current design using 31 complete E. Coli, Shigella and S.pneumoniae genomes. The E. coli and Shigella data was downloaded from http://www.biotorrents.net/ (accessed 21/3/11) [5]. The S. pneumonia data was acquired from the NCBI Entrez website (http://www.ncbi.nlm.nih.gov/Entrez/ accessed 21/3/11) [8]. The number of genomes, sequences and bases per genome are provided in Table I. We carry out weak scaling simulations to test performance on BG/P using 64 computational nodes (256 cores). We measure run times for building the scaffold using 2, 4, 6, 8, 16 and 23 genomes combining E.coli and Shigella sequences since they have similar number of bases per genome. To test performance for genomes with smaller number of base pairs per genome we use 2, 4, 6 and 8 genomes of S.pneumoniae seperately.

III. Preliminary Results

Investigation of memory usage revealed that generating the sorted mer list is the most memory intensive operation. Our improvements in data storage allow the distributed algorithm to theoretically align a maximum of 200 bacteria sequences on a 64 node BG/P partition which would not be possible with the original *progressiveMauve* implementation. Performance

TABLE I
SUMMARY OF GENOMES USED FOR WHOLE GENOME COMPARISON

Organism	Number of genomes	Number of sequences	Base pairs per genome [Mbp]	Total number of base pairs [Mbp]
E.coli	18	18	5.07	91.3
Shigella	5	5	4.58	22.9
S.pneumoniae	8	8	2.13	17.1

Mbp – Mega base pairs

TABLE II
RUN TIME AND NUMBER OF K-MERS IN SCAFFOLD

Number of genomes	E. coli/Shigella		S.pneumoniae	
	run time [s]	k-mers	run time [s]	k-mers
2	84.06	10,020,773	14.32	4,084,686
4	158.48	19,921,809	34.65	8,539,454
6	229.62	30,118,466	48.58	12,838,812
8	313.78	40,307,489	62.52	17,070,882
16	646.91	81,663,419	-	-
23	929.32	114,205,647	-	-

analysis of the *progressiveMauve* shows that the alignment step (Fig. 1) is the most compute intensive phase. On average the alignment consumed 90% of total run time.

The results for building the scaffold on BG/P are shown in Table II. Our algorithm produces a scaffold comparing two E.coli genomes in less than 1.5 minutes resulting in a k-mer list with over 10 million entries. The scaffold for the total data set of 18 E. coli and five Shigella genomes took about 15 minutes with a k-mer list of over 114 million entries. Thus, while increasing the data volume by about a factor of 11 we find that the time to build the scaffold also increases by a factor of 11. This also corresponds to the number of entries in the k-mer list that has 11.4 times more entries for 23 genomes compared with the analysis of two genomes. The analysis for the S.pneumoniae genomes confirms this result with respect to run time although the number of k-mers found is smaller.

Overall, using the 23 E. coli and Shigella genomes, our implementation creates 23 FASTA nodes and 232 worker nodes. The creation of the sorted k-mer list took 16 minutes including setup, generating 3.7MB of compressed data per node. Comparing the run time between the original *progressiveMauve* algorithm and our implementation we achieve a speedup of 2.8. This is notable as the sequential version was run on a server with 1.6 GHz clock speed which is ~2 times faster than a BG/P node.

IV. DISCUSSION

While parallelization will have the greatest impact in the alignment step, one has to sequentially parallelize the algorithms for a complete port to massively parallel, distributed memory systems. Also, for massively parallel, distributed memory supercomputers with limited memory per node one has to optimize data storage first which we have achieved by implementing the setup and building the scaffold first that then allows to carry out the alignment to make full use of the computational resources of a supercomputer. Our method of distributing the sorted k-mer list can utilize large amounts of nodes to improve performance. Testing of the distributed aligner using E-coli, Shigella and S.pneumoniae genomes has confirmed the accuracy of this implementation; having returned back results an exact match to the original algorithm. While future focus has been put on aligning bacteria genomes our implementation using data compression means that it is possible to fit a human genome on each BG/P node running in SMP mode. Size of the sorted k-mer list depends on the amount of variation between sequences. Similar sequences will contain matching seeds which are condensed to positions.

V. CONCLUSION AND FUTURE WORK

Memory efficient data structures and the creation of the scaffold in a distributed manner is the first step in devising a scalable alignment algorithm. Further work is required to parallize the alignment and HMM phase of the *progressiveMauve* algorithm where we expect to have large impact on run time performance. Validation and verification as well as benchmark comparison of original and distributed *progressiveMauve* algorithms will then be carried out.

Sequence alignment is a genomic comparison technique commonly utilized in modern genomics. While alignment is achievable on small sets of genomes, when applied to large datasets, it becomes memory and computationally intensive, even intractable on common computer clusters. The design of multiple sequence alignment algorithms for massively parallel, distributed memory systems enables comparison of hundreds of genomes in the same time as a few genome sequences. In this study we have laid down the basis for multiple sequence alignment on massively parallel, distributed memory supercomputers for large scale genomic sequence alignment, thus enabling comparison of hundreds instead of a few genome sequences within reasonable time.

REFERENCES

[1] A. Ghoting and K. Makarychev, "Indexing genomic sequences on the IBM Blue Gene," Supercomputing 2009

[2] M. A. Larkin, *et al.*, "Clustal W and Clustal X version 2.0," *Bioinformatics,* vol. 23, pp. 2947-2948, November 1, 2007 2007.

[3] R. C. Edgar, "MUSCLE: multiple sequence alignment with high accuracy and high throughput," *Nucleic Acids Research,* vol. 32, pp. 1792-1797, March 1, 2004 2004.

[4] C. Notredame, *et al.*, "T-coffee: a novel method for fast and accurate multiple sequence alignment," *Journal of Molecular Biology,* vol. 302, pp. 205-217, 2000.

[5] A. E. Darling, *et al.*, "progressiveMauve: Multiple Genome Alignment with Gene Gain, Loss and Rearrangement," *PLoS One,* vol. 5, p. e11147, 2010.

[6] B. Ma, *et al.*, "PatternHunter: faster and more sensitive homology search," *Bioinformatics,* vol. 18, pp. 440-445, 2002.

[7] C. Sosa, B. Knudson. "IBM System Blue Genen Solution: Blue Gene/P Applicatino Development." 4th edition. IBM Redbooks. Sept 2009.

[8] S. V. Anguoli and S. L. Salzberg. "Mugsy: fast multiple alignment of closely related whole genomes." *Bioinformatics* vol 27(3) pp. 334-342, 2011