Analysis on Parallelism of Multiple Sequence Alignment Using MAGUS

Vikram Ramavarapu*, Tandy Warnow*
*Department of Computer Science, University of Illinois at Urbana-Champaign

Abstract

Motivation: Multiple sequence alignment is an essential part of many bioinformatics pipelines. Though alignment is difficult to compute, there are heuristics such as divide and conquer that improve scalability and preserve accuracy. This heuristic is used by MAGUS (Smirnov & Warnow, 2020). However, algorithmic approaches alone are not necessarily sufficient to bring scalable solutions and there exists a need for high performance computing methods. More specifically, there is a need for parallelism and portability for larger scale hardware.
Result: We develop an addition to MAGUS that allows for parallelism by interfacing with a multiple node cluster. Through studying the effects of using multiple nodes to compute multiple sequence alignment using MAGUS, we found that up to a 12 times speedup was induced with relation to a single core run of MAGUS through our cluster extension.
Availability: The source code for MAGUS-Cluster can be found at https://github.com/vikramr2/MAGUS-Cluster.

Keywords: Bioinformatics; Computational Biology; Multiple Sequence Alignment; Distributed System; High Performance Computing

1 Introduction

Multiple Sequence Alignment (MSA) is an essential first step in many different computational biology problems, from structural analysis and classification of sequences of amino acids in proteins, to phylogeny estimation from nucleotide sequences. However, MSAs are very computationally intensive as the MSA problem is classified an NP-Complete. To mitigate this issue, there are algorithmic approaches such as divide and conquer, which is used by tools such as PASTA (Mirarab et al., 2015), and Fast Fourier Transform, used by MAFFT (Katoh & Standley, 2013). These heuristics have proven to greatly accelerate alignment while preserving accuracy. A primary example of an algorithm that uses this approach is Multiple sequence Alignment using Graph clUStering (MAGUS).

1.1 Overview of MAGUS

MAGUS is a three stage pipeline. First the unaligned set is decomposed into subsets of equal number of sequences. MAGUS, in its original implementation, allows for the use of two different decomposition strategies:

1. Random decomposition: sequences are shuffled and evenly sectioned into subsets. This option is not recommended as it may lead to very inaccurate or long running results depending on the decomposition.
2. Guide tree decomposition: sequences are decomposed according to a computed tree. MAGUS gives the option of either using FastTree (Price, Dehal, & Arkin, 2010) or Clustal Omega (Sievers & Higgins, 2018) to construct the guide tree.

After the unaligned set is decomposed into subsets, each subset is aligned independently. In its original publication, MAGUS used MAFFT to align each subset.

Finally, all subalignments are merged into a single alignment using a graph cluster merger (GCM). Each subalignment is split evenly by their number of sequences and a set of backbone alignments from each set of sequences is then computed. Each backbone alignment has an equal number of sequences from each subalignment. From that, an alignment graph is computed where each node is a column from each original subalignment, and each edge is a homology between subalignment columns. The Markov Clustering Algorithm (MCL) (Enright, 2002) is run on the alignment graph. Finally violations where two nodes from the same column share an edge are resolved and an alignment is traced from the graph.

MAGUS allows the user to align a set of sequences from scratch, or start from a directory of subalignments or backbone alignments, and perform the GCM to output a merged alignment.

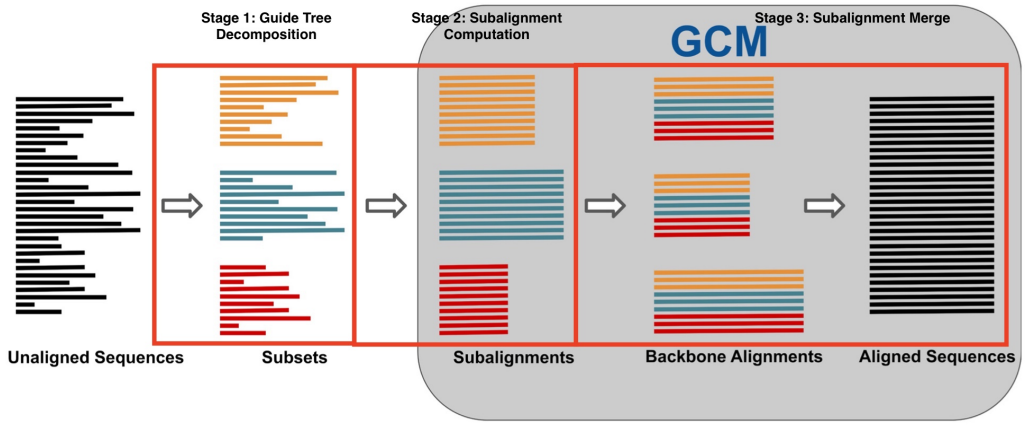

Figure 1: The Three Stages of MAGUS

1.2 Existing Parallel Implementations

In this study, we focus on the second stage of MAGUS: the independent subalignment computations using MAFFT. MAGUS's original implementation already has parallelism for this stage. However this is done within a single node using Python's subprocess module, and this does not transfer well to a distributed system for high performance computing. As opposed to running background processes from a shell script, Python's subprocess module induces overhead which can noticeably slow down running time. Moreover, in Smirnov's recursive implementation (Smirnov, 2021), MAGUS is run on a distributed system. However, in that study, MAGUS is called recursively and each node runs MAGUS independently, allowing MAFFT to be called as CPU processes by each node running MAGUS. This approach led to a slightly lower accuracy. In this study, we run MAGUS in it's original algorithmic form, but on a distributed system in efforts to preserve accuracy as algorithmically there are no changes, and improve runtime through parallel computing.

1.3 Related Work

PASTASpark (Abuín, Pena, & Pichel, 2017) uses a similar approach where independent instances of MAFFT are evenly distributed over multiple nodes during the second stage of PASTA. In PASTA, a guide tree is used to decompose the original set of unaligned sequences into subsets for them to be independently aligned. The aligned subsets are then merged pairwise and then through transitivity. Therefore, MAGUS and PASTA have the first two stages in common in their respective algorithmic pipelines, with their main difference being the method of subalignment merge in their third stage. In PASTASpark, the Apache Spark framework was used to interface with a compute cluster to evenly load balance MAFFT instances into cluster nodes so that each node would handle roughly the same number of subalignments. Spark collected the subalignments at the end of the node distribution, and the merger was run on the single head node of the cluster. This showed a great improvement in runtime while preserving the accuracy of PASTA. This was mostly due to the fact that as stated in Mirarab et al., 2015, most of PASTA's runtime was from the subalignment computation phase of its algorithmic pipeline.

In this study, we will extend MAGUS using SLURM (Yoo, Jette, & Grondona, 2003), a Unix framework for automated resource and task management in a cluster computing system, to distribute MAFFT alignment tasks evenly across nodes. Then, we will examine the speedup induced from using more nodes and cores. Moreover, this extension to MAGUS will be verified for accuracy using simulated RNA data sets. Finally, we will examine the relation between different biological protein data sets and speedup.

2 Method

In our extension of MAGUS, we will use a similar procedure as PASTASpark, where the independent subalignment computation instances using MAFFT will be parallelized in a multiple node cluster. MAGUS already has a decomposition script to divide the original unaligned set of sequences into subsets. Therefore, the decomposition stage of MAGUS is extracted from its original implementation. This decomposition program is run on the head node of the distributed system. Then, a SLURM job script runs MAFFT as background processes. Fortunately, SLURM automatically distributes tasks evenly to worker nodes when given the number of nodes and cores per node to use when executing a job. Moreover, calling MAFFT directly from a Unix shell script eliminates overhead induced by calling MAFFT from a Python subprocess. These subalignments are outputted within a temporary

directory. The head node waits for the worker nodes to finish and for the temporary directory to have all subalignments, leading to the last stage of this process where MAGUS in its original form is called on this directory of subalignments to run the GCM and output a final alignment.

The University of Illinois Campus Cluster with 12 compute nodes providing 296 CPU cores, 1152GB RAM + 8 dual-socket Intel Xeon E5-E2680v3 nodes w/ 24 CPU cores, 64GB of RAM, Xeon PHI 5100, FDR InfiniBand Interconnect + 2 dual-socket Intel Xeon E5-E2670v2 nodes w/ 20 CPU cores, 64GB of RAM, Xeon PHI 7120, FDR InfiniBand Interconnect + 2 dual-socket AMD EPYC 7302 nodes w/ 32 CPU cores, 256GB of RAM, HDR InfiniBand Interconnect was used for this study. However, only the original 12 nodes were used.

The software used in this study includes: slurm 22.05.2, MAGUS v0.1.0b2. The number of cores and nodes used were specified in the header of the SLURM job script. Then to run the job script, the following command was used:

```
sbatch jobscript.sh
```

To examine MAGUS in it's single core runtime, we made a SLURM script run on the compute cluster specifying the use of only one node and one core per that node. The following command was inserted in the script:

```
python3 MAGUS/magus.py -i <input file>
    -o result_baseline.txt --maxnumsubsets 24
```

And likewise 'sbatch' was used to run the SLURM script. Across all runs of MAGUS, default parameters were used with the exception of the maximum subsets being 24. This was due to divisibility so that tasks could remain evenly distributed across nodes. Therefore, along with the maximum number of subsets being 24, the maximum subset size is 50, 10 backbone alignments, and FastTree is being used as the guide tree.

3 Evaluation

3.1 Overview

We performed two experiments. In the first experiment, we examine a single data set and examined speedup with respect to the single core run time on the data set against the number of cores. Both full run times as well as only the subalignment stage run times will be examined. In the second experiment, we examine the effect of different biological data sets on speedup. In each protein data set, speedup of the 20-core runtime is calculated with respect to the 4-core run time on that respective data set.

3.2 Datasets

This evaluation uses biological datasets from previous studies. All datasets used in this study were also used in Smirnov & Warnow, 2020. Three of the eight protein datasets from BAliBASE (Thompson, Plewniak, & Poch, 1999) used in Smirnov & Warnow, 2020 were used in this study. Reference alignments based on structural features are available with these datasets.

Dataset	No. of Seqs	Avg. P-dist.	Max P-dist.	% gaps	Seq. length (avg)	Type
BBA0039	732	0.364	0.918	69	382	bio AA
BBA0081	195	0.860	0.967	63	578	bio AA
BBA0101	322	0.778	0.899	51	465	bio AA

4 Results

4.1 Experiment 1: BBA0039 Speedup vs. Number of Cores

In MAGUS's second stage, many instances of MAFFT are run in parallel to create aligned subsets. In this study, we focus on controlling parallelism on this stage, thus running these MAFFT instances concurrently. In this experiment, we increase the number of cores and examine the change in runtime on the BBA0039 dataset from BAliBASE. MAGUS was run on this dataset six times with one, four, eight, twelve, sixteen, and twenty cores respectively.

Experiment 1(a): Whole MAGUS Speedup vs. Number of Cores. The entire running time of MAGUS is considered in this sub-experiment to calculate overall speedup after parallelization. According to Amdahl's

Law (Amdahl, 1967), the speedup of a parallel program with respect to its single core counterpart is given by the following equation:

$$speedup = \frac{1}{f + \frac{1-f}{p}}$$

Where f is the serial portion of the parallel program and $1 - f$ is the portion that is parallelized. p is the number of cores that the parallel program is running on. This means that as p gets very large, it is shown that speedup cannot exceed $1/f$. Since speedup is approaching roughly 12, this means that roughly $1/12$ of the program is serial. This is very small compared to expectations. This is most likely explained by existing parallelism in MAGUS during the backbone computation phase.

Experiment 1(b): Stage Two Speedup vs. Number of Cores. Recall that stage two of MAGUS is the subalignment computation given a disjoint decomposed set of sequences. In this second sub-experiment, we focus solely on the runtime of this stage in the parallelized extension of MAGUS as the number of cores increases. As opposed to the entire runtime, there is a much more visible convergence as cores increase. The increase towards the end is likely a combination of slight randomization during guide tree decomposition, as well as convergence in speedup as stated by Amdahl's law.

Experiment 1(c): Percent Runtime of Stage Two In this sub-experiment, we analyze the proportion of the entire MAGUS runtime that is being taken by the second stage of the pipeline as cores increase. In contrast to what was brought forth by Abuín et al., 2017, as opposed to PASTA where subalignment computation takes most of its runtime, the subalignment computation in stage two is a very small portion of MAGUS's entire runtime. Moreover, there is a very loose correlation between the number of cores used and the percentage of total runtime that subalignment computation takes. This means that the percentage of runtime by which subalignment computation takes in MAGUS, especially when after parallelization, subalignment computation only takes a few seconds, this percentage is easily perturbed by slight randomization in clock speed or guide tree decomposition. Moreover, with subalignment computation now taking a low percentage of runtime, MAGUS is now bottle necked in running time by backbone alignment computation in the graph cluster merger stage.

4.2 Experiment 2: 20-core Speedups on Protein Datasets

In Smirnov & Warnow, 2020, Smirnov states that he had removed outliers, that is, sequences that were much longer than the standard deviation of length in the unaligned sequence set. This was due to many multiple sequence alignment methods, including PASTA, MAFFT, and in this case, MAGUS, experiencing a much longer runtime due to outliers. In this study, we did not remove outliers from the sequence datasets. This was in effort to study the effect of parallelism on different types of biological sequence data. In this experiment, we analyze the effect of parallelism on three different types of protein sequence data from BAliBASE.

In our findings, it is apparent that the proportion of overall runtime that is being taken by the second stage of MAGUS is very heavily dependent on the type and nature of the dataset in which the alignment is being computed on. In both BBA0081 and BBA0101, we see that there is significantly more speedup in the subalignment stage than speedup in the overall runtime. However, in BBA0101, there is much more of a difference between second stage speedup and overall speedup than in BBA0081. This is likely due to other stages such as merging, decomposition, and backbone alignment taking less of BBA0101's runtime, but a larger proportion of BBA0081's runtime. Interestingly, there is a higher total speedup in BBA0039 than second stage speedup on the same dataset. This is likely due to existing parallelism in other stages such as backbone alignment which is much more of a bottleneck due to the sequence length variance in BBA0039's unaligned set of sequences. Moreover, variance between subalignment lengths being much higher than sequence length variance within subalignments can make runtimes of backbone alignment computations far exceed runtimes for subalignment computations, thus being a potential explanation for the speedup difference between BBA0101 and BBA0039 for example.

Therefore, we find that the proportion of runtime being used by subalignment computation may not be a majority depending on the nature of the unaligned sequences.

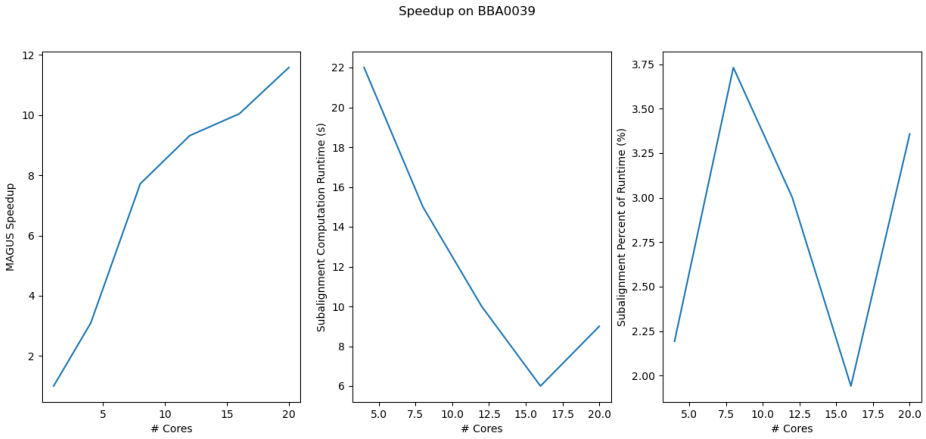

Figure 2: Experiment 1: (left) Entire MAGUS speedup vs. Number of Cores, (middle) Second Stage MAGUS Runtime vs. Number of Cores, (right) Second Stage percent of whole runtime vs. Number of Cores

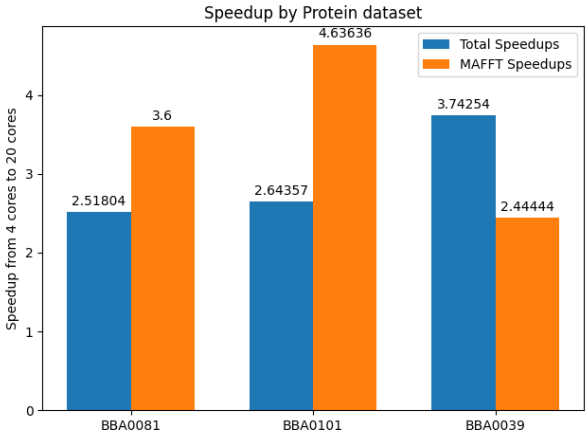

Figure 3: Experiment 2: Full MAGUS runtime speedups (blue) and Stage two only speedups (orange) for (left) BBA0081, (middle) BBA0101, (right) BBA0039

5 Conclusion

MAGUS-Cluster is an extension of MAGUS that is executable on a compute cluster. It allows for the extension to multiple nodes to allow for more core usage, which becomes useful for larger sets of data. Our study evaluated this extension by measuring the speedup curve with respect to a single core and single node execution. We evaluated the speedup of the entire program, the section being parallelized, and its proportion to the entire runtime as cores increased. Moreover we analyzed speedup and its correlation to the type of biological dataset being used. We have found that speedup can be achieved with this extension by up to a factor of twelve.

There are limitations to this study however. Further experimentation needs to be done with our MAGUS extension on larger sets of data such as the set of sequences from the Comparative RNA Web (Cannone et al., 2002) that is used in MAGUS's original implementation. Moreover there is a limitation to roughly 20 cores when solely parallelizing subalignment computation. This is due to the fact that after enough parallelization, MAGUS is bottlenecked by other stages in the pipeline. The second experiment had highlighted that speedup is heavily dependent on the type of dataset being used. Datasets that have more variance and outliers in sequence length will take much longer to be aligned. This still holds true as a limitation for our parellization of MAGUS. Speedup is much lower on datasets that have higher sequence length variance and outliers. We have found that the proportion of runtime taken by the different stages of MAGUS, and so the sensibility of parallelizing just the second stage of MAGUS, is heavily dependent on the sequences being operated on.

Our study suggests quite a range of directions for future work. It was shown that a relatively small proportion of MAGUS's runtime was comprised from subalignment computation. This is especially shown with the low

correlation between the percentage of full runtime that is comprised by subalignment computation and cores used. A possible future direction that is highlighted by this is the multiple core parallelization of the backbone alignment computation. This can be very portable to the architecture of a compute cluster as each node can handle a subalignment in the second stage of MAGUS, and each core in a node can decompose a subalignment to construct a a backbone alignment. Moreover, there is a setting of MAGUS that takes backbone alignments as inputs so that MAGUS solely finishes the GCM phase as its execution on the input. This makes this potential future direction much easier to implement. There is also potential in Graphics Processing Units (GPUs) in multiple sequence alignment. Parallelization of MAFFT instances can be designated to GPU threads. Parallelization of MAGUS on a GPU could lead to improved scalability with larger sets of data, especially when using a GPU-based compute cluster. Lastly, there already exist many massively parallel extensions of the Markov Cluster Algorithm, namely HipMCL (Azad, Pavlopoulos, Ouzounis, Kyrpides, & Buluç, 2018), which is a GPU parallelization of the Markov Cluster Algorithm. Future work should examine the impact of these methods on the parallelization of MAGUS and more generally, on tools for multiple sequence alignment.

Acknowledgements

I would like to thank the National Center for Supercomputing Applications (NCSA) for providing access to the University of Illinois Campus Cluster.

References

Abuín, J. M., Pena, T. F., & Pichel, J. C. (2017, 06). PASTASpark: multiple sequence alignment meets Big Data. *Bioinformatics*, *33*(18), 2948-2950. Retrieved from https://doi.org/10.1093/bioinformatics/btx354 DOI: 10.1093/bioinformatics/btx354

Amdahl. (1967). Validity of the single-processor approach to achieve large scale computing capabilities. *AFIPS Joint Spring Conference Proceedings 30 (Atlantic City, NJ, Apr. 18–20), AFIPS Press, Reston VA.*

Azad, A., Pavlopoulos, G. A., Ouzounis, C. A., Kyrpides, N. C., & Buluç, A. (2018, 01). HipMCL: a high-performance parallel implementation of the Markov clustering algorithm for large-scale networks. *Nucleic Acids Research*, *46*(6), e33-e33. Retrieved from https://doi.org/10.1093/nar/gkx1313 DOI: 10.1093/nar/gkx1313

Cannone, J. J., Subramanian, S., Schnare, M. N., Collett, J. R., D'Souza, L. M., Du, Y., ... Gutell, R. R. (2002, Jan 17). The comparative rna web (crw) site: an online database of comparative sequence and structure information for ribosomal, intron, and other rnas. *BMC Bioinformatics*, *3*(1), 2. Retrieved from https://doi.org/10.1186/1471-2105-3-2 DOI: 10.1186/1471-2105-3-2

Enright, A. J. (2002, April). An efficient algorithm for large-scale detection of protein families. *Nucleic Acids Res.*, *30*(7), 1575–1584.

Katoh, K., & Standley, D. M. (2013, April). MAFFT multiple sequence alignment software version 7: improvements in performance and usability. *Mol. Biol. Evol.*, *30*(4), 772–780.

Mirarab, S., Nguyen, N., Guo, S., Wang, L.-S., Kim, J., & Warnow, T. (2015, May). PASTA: Ultra-large multiple sequence alignment for nucleotide and amino-acid sequences. *J. Comput. Biol.*, *22*(5), 377–386.

Price, M. N., Dehal, P. S., & Arkin, A. P. (2010, March). FastTree 2 – approximately maximum-likelihood trees for large alignments. *PLoS One*, *5*(3), e9490.

Sievers, F., & Higgins, D. G. (2018, January). Clustal omega for making accurate alignments of many protein sequences. *Protein Sci.*, *27*(1), 135–145.

Smirnov, V. (2021, 10). Recursive magus: Scalable and accurate multiple sequence alignment. *PLOS Computational Biology*, *17*(10), 1-17. Retrieved from https://doi.org/10.1371/journal.pcbi.1008950 DOI: 10.1371/journal.pcbi.1008950

Smirnov, V., & Warnow, T. (2020, 11). Magus: Multiple sequence alignment using graph clustering. *Bioinformatics*, *37*(12), 1666-1672. Retrieved from https://doi.org/10.1093/bioinformatics/btaa992 DOI: 10.1093/bioinformatics/btaa992

Thompson, J. D., Plewniak, F., & Poch, O. (1999, 01). BAliBASE: a benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics*, *15*(1), 87-88. Retrieved from https://doi.org/10.1093/bioinformatics/15.1.87 DOI: 10.1093/bioinformatics/15.1.87

Yoo, A. B., Jette, M. A., & Grondona, M. (2003). Slurm: Simple linux utility for resource management. In D. Feitelson, L. Rudolph, & U. Schwiegelshohn (Eds.), *Job scheduling strategies for parallel processing* (pp. 44–60). Berlin, Heidelberg: Springer Berlin Heidelberg.