Parallelization of DNADIST

Shankar Lakshmanan, Cody Leach, and Keith Luchtel
Computer Science and Engineering
University of Nebraska - Lincoln
Lincoln, NE 66588-0115
(slakshma | cleach | kluchtel)@cse.unl.edu

Abstract

In the project, the task of parallelizing the DNADIST program in the Phylip package was given. DNADIST calculates a distance matrix, where each cell in the matrix represents an estimation of the branch length between two DNA sequences. MPI was used to implement the final solution, and the team was able to equally divide the work amongst a set of processes. The implemented solution sufficiently reduced the overall run time compared to serial version of the code, with a speed-up proportional to the size of the input.

1. Introduction

DNADIST is a program that calculates the branch length between two DNA sequences. The DNA sequences are read in as a string of characters, which can contain a combination of A, T, G, and C. Each of these characters represents the four nucleotides found in DNA. The program uses the sequences given in the input file and calculates the branch differences between each of them. At the end of the calculations, the program outputs a distance matrix with one row and one column for each sequence. The intersecting cell in the matrix for a row and a column is the branch length between the two respective sequences. This matrix by nature is symmetrical because each combination of sequences will cross twice. This means that only half of the matrix needs to be calculated and then mirrored across the diagonal. For a large number of sequences with a large number of characters per sequence, the calculations can become very time consuming. This is where the need of parallelizing the algorithm becomes necessary.

The rest of this paper discusses related work in Section 2, and then describes the implementation in Section 3. Section 4 describes how the team evaluated the program and presents the results. Section 5 explains the distribution of work completed by each team member. Finally, Section 6 presents conclusions and describes future work.

2. Related Work

The task of implementing DNADIST in a parallel fashion is something that has not been done before, at least not well-published. All of the implementations that were found during the research phase were adaptations of the open source serial program. The main website for DNADIST said it would be fairly easy to implement a parallel version of the code, which perplexed the team as to why no one had done it before.

3. Implementation

The task of parallelizing a serial program begins with research, especially to better understand how the serial program works. The implementation of DNADIST, as described below, was given to the team without any useful documentation, so research was a must. Before any coding or designing of a parallel algorithm could begin, the serial code had to be analyzed and thought through to understand what was being accomplished with it. Once the research and understanding of the serial program was complete, the team was able to devise and implement a parallel algorithm that did the same thing, as described in the following paragraphs.

The first step in parallelizing the algorithm was to determine which part(s) could be parallelized. The bulk of the work done by the program is when it calculates the actual distances between the sequences themselves. In order to accomplish this parallelization, the distance matrix that stores the branch lengths between the sequences was evenly divided. This allowed for an equal distribution of work amongst all the processes. Since only half of the matrix was being calculated (and then duplicated across the diagonal), dividing it up was not as easy as assigning rows or columns to the processes. Instead, the parallel implementation determines the number of cells that need to be calculated and divides them up by number, not by row or column. Each process determines its starting cell in the matrix based on its process number, and then it sequentially goes down the columns until it has calculated its specified number of cells.

Because each cell in the distance matrix is independent of one another, this is no need for communication between the processes. This helps increase the effectiveness of the parallelization since the overhead of communication has been reduced to a minimum. As it stands in the implementation, each process reads from the input file. This means that each process reads and processes the data from the file. The efficiency and improvements upon this implementation are described in the *Evaluation* and *Conclusions and Future Work* sections. Aside from the input, the only communication that is required by the program is sending the matrix data from each process back to the root process. The root process collects this data and puts it together in one complete distance matrix and outputs it to a file called ‘outfile’.

4. Evaluation

The implementation described above has been tested, and the team saw a speed up during testing. The speed up was seen in a comparison test where both the serial and parallel versions of the program were timed. This was accomplished by using the `MPI_Wtime()` function in the MPI library for the parallel and serial versions (this was the only function of the MPI library used within the serial version). Both of these methods gave us the elapsed seconds, which were used to time the programs as they entered and exited their `main` methods.

The inputs used for testing were those given by Dr. Moriyama in the Bioinformatics department. She gave the team two different files to test: one with 4285 sequences and the other with 33286 sequences. In addition to the input files, Dr. Moriyama included the output file from when she ran the 4285 sequence input file serially. She also indicated that she started to run the larger input file, but it was taking over 30 hours to complete so she terminated its execution.

With the 4285 sequences file, the team was able to reduce it down to 1000 and 2000 sequences for testing. This allowed for quicker testing of correctness and usability, and it provided useful insight into the speedup achieved. **Figure 1** below shows four test runs of the parallel program along with a run of the serial version (one processor). As one can see, a speed up was achieved in the parallel version over the serial.

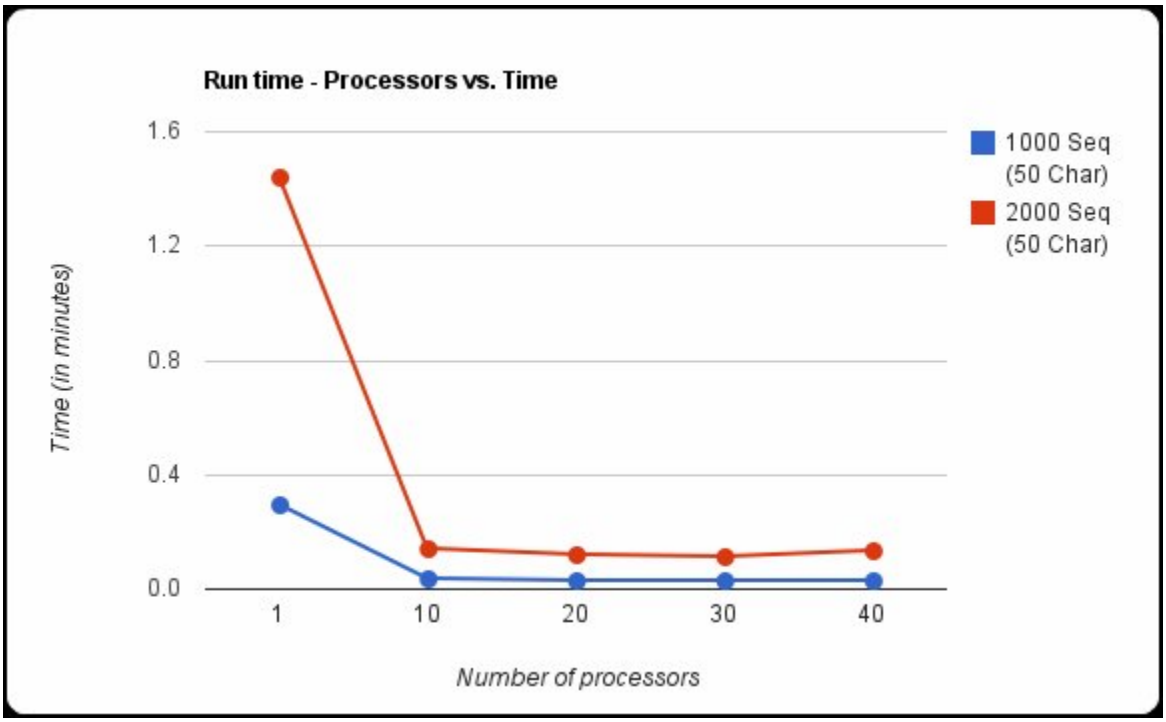

Figure 1. Test runs of the parallel program and one run of the serial version using 1000 and 2000 DNA sequences. The blue line shows a reduced version of the 4285 sequence input to 1000 sequences. The red line shows a reduced version of the 4285 sequence input to 2000 sequences. Each of the input files had sequences with 50 characters per sequence.

The real measure of speedup, however, comes from running the full 4285 sequences in the file. Since it is a realistic set of data (and not modified by the team), it is considered to be the standard for testing correctness and speedup. Through the tests that were ran, the output was identical to that provided by Dr. Moriyama. In addition, the team saw a speedup of the parallel program that is approximately seven times that of the serial version. **Figure 2** shows the test runs of this file:

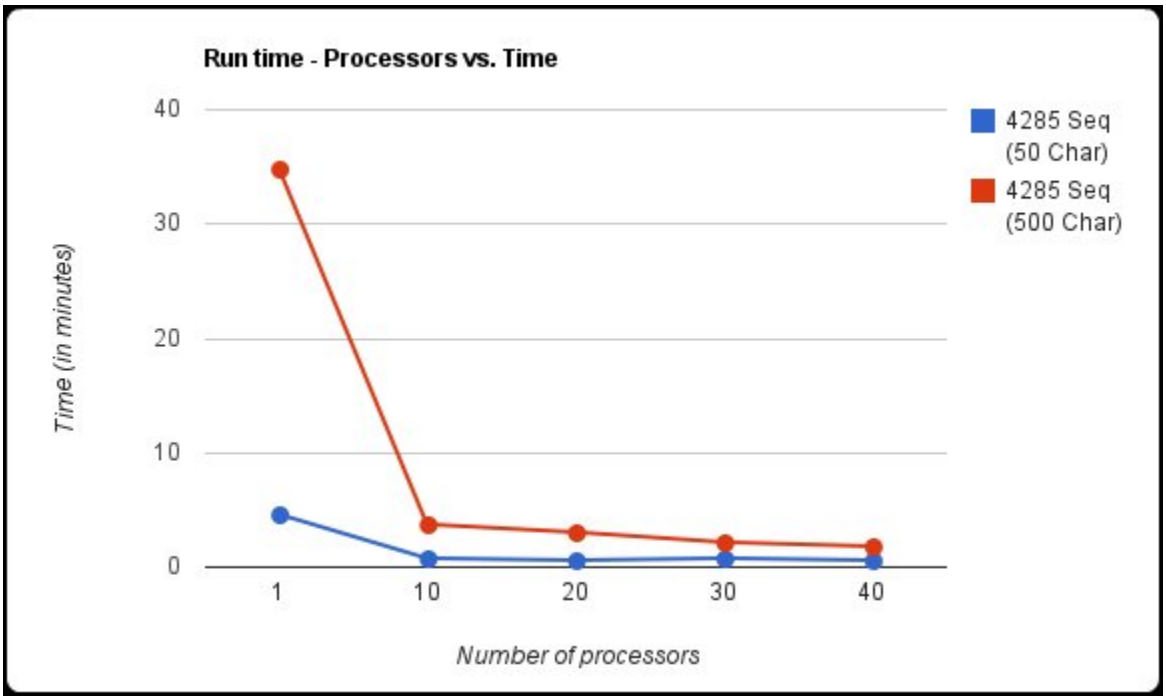

Figure 2. Test runs of the parallel program and one run of the serial version using 4285 DNA sequences. The blue line shows the set of tests with each sequence have 50 characters. The red line, however, shows tests with 500 characters per sequence.

Even though speed is apparent in both tests, it is important to note the time difference between the two test sets (the red and blue lines) in Figure 2. The blue line in the figure was ran with only 50 characters per sequence. The total run time for the 50 character test with the serial program was 4.61 minutes. This was sped up to 0.78 minutes with the use of ten processors. The total run time for the 500 character test with the serial program was 34.8 minutes. This was sped up to 3.6 minutes when parallelized with ten processors. In both test cases, the speed up achieved was 5.9 and 9.7, respectively. The data points used in the graphs and in the speed up calculations are tabulated below:

Number of Processors	1000 Sequences 50 characters (in minutes)	2000 Sequences 50 characters (in minutes)	4285 Sequences 50 characters (in minutes)	1285 Sequences 500 characters (in minutes)
1	0.29	1.44	4.61	34.8
10	0.035	0.14	0.78	3.6
20	0.031	0.12	0.59	2.9
30	0.03	0.11	0.64	2.1
40	0.028	0.13	0.55	1.75

Table 1. Run times obtained by varying the number of processors used and the sizes of the inputs.

In addition to the overall speed up achieved, as indicated in both figures, it is important to note what occurs when more processors are added. In both figures, as the number of processors increases the run time doesn't vary as greatly. This is especially apparent when comparing the runs with ten and 20 processors. Beyond ten processors the

speed up is almost zero. This occurs because of the communication that takes place when the root process gathers the results from the worker processes. The increase in the number of processes adds to the communication time, which nearly counteracts the efficiency of the increased number of processes.

As described above, the tests indicate a significant speed up by simply splitting up the calculations. This indicates that the effort put forth by the team was successful to some degree, and it is believe that Dr. Moriyama and the Bioinformatics department will be pleased with the results.

5. Distribution of Work

For the problem at hand, the team felt it would be difficult to equally distribute the work amongst each member. Therefore, the team consistently met on a weekly basis and did the majority of the work together. Some outside work was done by each of the individual members on their own time, for example some debugging or pieces of the report. In the end however, each member put forth an equivalent amount of effort.

In the middle of semester, a member of the team had a family emergency and had to leave unexpectedly. This inhibited work for part of the semester, but effort was made to continue the whole time, and special arrangements were made to keep communication flowing between all members.

6. Conclusions and Future Work

DNADIST is a program used to calculate the branch distances between two DNA sequences. To parallelize the program, the team evenly divided the calculations of the branch distances amongst separate processes. This parallelization yields significant speed results when compared with the serial implementation of DNADIST. This will provide faster results to the Bioinformatics department, and it will ultimately assist them in their research efforts.

To help improve the solution in the future, the team has identified a few ideas that could be incorporated. In addition some different implementation approaches could be investigated to see if they would yield a better speedup compared to the MPI solution. These ideas and approaches are described below:

Features and Improvements

- The ability to specify the names of the input and output files as command line arguments instead of using the default names ‘infile’ and ‘outfile’. This would allow the user to run multiple instances of the program without having to worry about the outputs getting overwriting one another.
- The manner in which files are read by the program. At the moment, each process reads the input file separately from the other processes. Ideally, the head process would read the input file and broadcast only the required data to the other processes. This would reduce the amount of memory used in each process, although it may not improve the run-time of the solution.
- Clean up the code to help improve readability and efficiency. This would include properly commenting functions and variables, as well as deleting sections of code that are not specifically used by the parallelize implementation of DNADIST.

Approaches to Investigate

- A CUDA version of DNADIST. CUDA could work well because each of the calculations in the distance matrix are completely independent of the others. CUDA would be able to fork a large number of threads and do more parallel calculations of the distance matrix at one time. One limitation could be the amount of memory available on a CUDA board. DNADIST requires a lot of memory as the input size increases, which would definitely be larger than the limitations of memory on a CUDA board.
- An implementation using OpenMP. OpenMP would allow for easier communication between the root process and the worker processes. This would be beneficial when gathering the data from the worker processes. In addition, this could be combined with another suggested implementation where the input data is read by the root process and dispersed to the worker nodes. Doing this in OpenMP would make this task much easier to implement.

References

[1] Felsenstein, J. “DNADIST -- Program to compute distance matrix from nucleotide sequences”. 2002. University of Washington. Accessed 2010. <http://www.bioinformatics.uthscsa.edu/www/phylip/doc/dnadist.html>

[2] Felsenstein, J. “PHYLIP”. 2002. University of Washington. Accessed 2010. <http://evolution.genetics.washington.edu/phylip.html>