A PARALLEL IMPLEMENTATION
OF A PARSIMONY-BASED METHOD
FOR PHYLOGENETIC INFERENCE

by

Ramarao V. Desaraju

A project submitted in partial fulfillment
of the requirements for the degree of

Master of Science in Computer Science

University of Colorado at Colorado Springs

May 2005
TABLE OF CONTENTS

1. Introduction ..................................................................................................................1
   1.1. Phylogenetic Tree Algorithms ..............................................................................1
   1.2. Parsimony-based Methods ..................................................................................3
2. Parallel Programming Methods ..................................................................................5
3. Implementation .............................................................................................................10
   3.1. Existing Algorithm ............................................................................................10
   3.2. Parallel Implementation .....................................................................................13
4. Results and Analysis ..................................................................................................17
   4.1. Test Conditions and Assumptions .......................................................................17
   4.2. Results ................................................................................................................19
   4.3. Analysis ...............................................................................................................21
   4.4. Scalability Testing ..............................................................................................22
   4.5. Tests with Larger Datasets ................................................................................24
   4.6. Validation Testing ..............................................................................................26
   4.7. Suggestions for Improvement ............................................................................29
5. Conclusions ................................................................................................................31
<table>
<thead>
<tr>
<th>Number</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1 - Possible Set of Unrooted Trees</td>
<td>2</td>
</tr>
<tr>
<td>Figure 2 – Shared Memory Multi-Processor System</td>
<td>5</td>
</tr>
<tr>
<td>Figure 3 – Message-Passing Multi-Processor</td>
<td>6</td>
</tr>
<tr>
<td>Figure 4 – Distributed Shared Memory System</td>
<td>7</td>
</tr>
<tr>
<td>Figure 5 – Tree construction in DNApenny (Newick format)</td>
<td>11</td>
</tr>
<tr>
<td>Figure 6 - Calculating Tree Cost</td>
<td>12</td>
</tr>
<tr>
<td>Figure 7 - Parallel program flowchart. Worker processes vary in number based on the number of processors available. This example assumes that the 6th taxon is added to the tree according to the process’s rank</td>
<td>16</td>
</tr>
<tr>
<td>Figure 8 – Test Results on SBC Cluster</td>
<td>20</td>
</tr>
<tr>
<td>Figure 9 - Speedup</td>
<td>21</td>
</tr>
<tr>
<td>Figure 10 – Results on the SGI Cluster</td>
<td>24</td>
</tr>
<tr>
<td>Figure 11 - Testing with Larger Datasets</td>
<td>26</td>
</tr>
<tr>
<td>Number</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>Table 1 - Sequence Data for 4 Taxa</td>
<td>2</td>
</tr>
<tr>
<td>Table 2 - Comparison of Parallel Programming Systems</td>
<td>8</td>
</tr>
<tr>
<td>Table 3 – Test Results on SBC Cluster</td>
<td>20</td>
</tr>
<tr>
<td>Table 4 - Test Results on SGI Cluster</td>
<td>23</td>
</tr>
<tr>
<td>Table 5 – Testing with larger datasets</td>
<td>26</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENTS

I wish to thank Dr. Jugal Kalita for his guidance throughout the course of the project. I would also like to thank Dr. Charles Shub and Dr. Robert Melamede for their valuable input during the presentation of my proposal for the project.

Several people helped me in making parallel programming hardware and software accessible for running my tests. I would like to thank Dave Lohmann of the Computer Science Department for help with my initial testing and debugging sessions on the June cluster, and later on SBC. I would like to thank Dr. James Mattoon of the Biology Department for giving me access to the SBC cluster in the Biology Department.

I would like to thank Steve Senator of the Air Force Academy for allowing me to access to the MPI cluster at the Academy. Thanks to Dr. Larry Pyeatt of Texas Tech University for granting me access to the HPCC cluster at TTU. Thanks to Dr. Kalita for facilitating both of these arrangements.
1. INTRODUCTION

Phylogenetic analysis of DNA and protein sequences is an important area of study in the field of bioinformatics. Inferring the evolutionary relationship between a set of related species based on their nucleic acid or protein sequence data is the goal of phylogenetics. The use of evolutionary trees is a fundamental step in many biological problems, such as multiple sequence alignments, protein structure, and function prediction, and drug design.

In recent times, a large amount of sequence data has become available, as well as a number of sequence alignment programs like ClustalW [11]. Sequence alignment is the essential primary step in phylogenetic tree analysis. The aligned sequences are the input for the phylogenetic analysis problem. This has lead to a requirement for more efficient algorithms for phylogenetic analysis.

1.1. Phylogenetic Tree Algorithms

There are three main methods for phylogenetic analysis - parsimony, distance, and maximum likelihood [3]. All three of these methods work by finding the tree that results in minimizing (or maximizing) a given optimality criterion. There is considerable debate over which method is best to use in a given situation. Sometimes, several of these methods are applied to validate the results obtained from one method. The characteristics of the sequence data is an important factor in deciding which method is appropriate in a given situation.

Input to a phylogenetic algorithm consists of sequence data for the taxa under consideration. Although DNA sequences are predominantly used for phylogenetic inference, protein sequences are also sometimes used. Table 1 shows a DNA sequence for 4 taxa. The sequence data is aligned using some alignment software such as ClustalW to account for various insertions, deletions, and mutations in the sequences. Given this pre-aligned set of sequence data for a number of potentially related species, the objective of phylogenetic analysis is to
reconstruct the tree-like pattern that describes the evolutionary relationships between the species.

<table>
<thead>
<tr>
<th>Taxa</th>
<th>Sequence (Length = 9 characters)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A A G A G T G C A</td>
</tr>
<tr>
<td>2</td>
<td>A G C C G T G C A</td>
</tr>
<tr>
<td>3</td>
<td>A G A T A T C C A</td>
</tr>
<tr>
<td>4</td>
<td>A G A G A T C C G</td>
</tr>
</tbody>
</table>

Table 1 - Sequence Data for 4 Taxa

Figure 1 - Possible Set of Unrooted Trees

There are 3 unrooted\(^1\) trees possible when comparing 4 taxa, and these are shown in Figure 1. To arrive at the final evolutionary tree, an outgroup is used to root the tree, thus establishing a direction of evolution, specifying which species is ancestral and which is derived. Thus for each unrooted tree there are several combinations of rooted trees. The goal of phylogenetic analysis is to infer the best tree from these possible set of arrangements, the one that is closest to evolution.

\(^1\) Unrooted trees show the evolutionary relationship between taxa, but not the direction of evolution. Unrooted trees are “rooted” using an outgroup, a taxon that one knows to have branched off earlier than the other taxa under study.
1.2. Parsimony-based Methods

Parsimony-based methods are generally used when there is a strong sequence similarity between the taxa under investigation [2]. Distance methods or Maximum Likelihood algorithms are used in other cases where the sequence data in question is less similar. Parsimony-based methods are named as such because they try to find the most "parsimonious" tree among the set of possible trees. This is a computationally intensive process, as the search space of possible trees increases exponentially with the number of taxa.

Phylogenetic analysis using parsimony is an optimization problem. It is based on the principle of parsimony - the tree that is closest to the actual relationship between species is the one that minimizes the number of steps required to generate the observed variation in the species [2].

Parsimony-based methods operate as follows. They produce all possible combinations of phylogenetic trees with the taxa under consideration. The evolutionary "cost" for each tree is calculated, and the tree with the least such cost is deemed to depict the relationship between the species. The number of trees that need to be examined increases rapidly as more sequences are added to the dataset. With five sequences, there are only 15 possible unrooted trees, but for ten sequences, there are 2,027,025 unrooted trees [3].

The number of unrooted binary trees that need to be examined for a data set with $n$ taxa is given by the expression: $(2n-5)(2(n-1)-5)(2(n-2)-5)\ldots 5.3.1$

For example, for a dataset with 15 taxa, close to $8 \times 10^{12}$ trees need to be examined. A large part of the running time is taken up in calculating the cost, which is a recursive function that operates on all positions in the sequence, on all the nodes. Heuristic methods exist, but these are not guaranteed to find the best tree. Better results for parsimony can be obtained by thorough searches of the search space [15].

DNAPenny, a program in the PHYLIP suite of programs, is one of the programs that is currently used for phylogenetic analysis using the principle of parsimony. It does not examine
all possible trees, but operates using a branch-and-bound approach. Even so, because of the large search space, the program runs very slow.

This project is focused at developing a parallel implementation of the sequential algorithm used by DNAPenny, with a view to reducing the running time.
2. PARALLEL PROGRAMMING METHODS

Parallel computers can be classified into shared-memory, message-passing, or distributed-shared memory systems [13]. This classification is based on how information is exchanged between the processors.

The shared-memory architecture refers to a system where there is one primary memory that is shared by multiple processors. A shared memory system employs a single address space, and each processor in the system has access to the entire address space. Communication between the processors is achieved by directly passing the data via shared memory. This arrangement is shown in Figure 2. It is difficult and expensive to implement the hardware required to implement shared memory systems.

![Figure 2 – Shared Memory Multi-Processor System](image)

Message-passing or distributed memory systems are basically multiple independent computers that are interconnected to form a cluster. Each of the nodes in a distributed-memory system is
an individual computer with its own memory with no specific knowledge about the other processors or their memory. Communication between the processors is accomplished by each processor explicitly sending and receiving messages from other processors. The messages can include data or results that other processors may require for computation. This is shown in Figure 3. The message-passing system will typically scale easier than a shared-memory system. Often, message passing computers can be implemented using off-the-shelf hardware. Parallelization is explicitly coded by the programmer.

Figure 3 – Message-Passing Multi-Processor

Distributed-shared memory systems are a hybrid of the two – local clusters of processors share memory, however, they must communicate with other clusters over some network. Sometimes, shared virtual memory is implemented, so that even when message passing is involved, each process can access the entire address space as if it were completely shared memory. This is illustrated in Figure 4.
A comparison of these three types of parallel programming techniques is given in Table 2.

<table>
<thead>
<tr>
<th></th>
<th>Shared-Memory</th>
<th>Message-Passing</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Architecture</strong></td>
<td>All processors share one primary memory, communication is via shared memory</td>
<td>Each processor has its own memory, communication is via interconnection network</td>
<td>Clusters of processors share memory, communicate with other clusters over network</td>
</tr>
<tr>
<td><strong>Cost</strong></td>
<td>Generally more expensive, requires special-purpose hardware</td>
<td>Can be built of off-the-shelf hardware, making them less expensive</td>
<td>Generally expensive, requiring special-hardware</td>
</tr>
</tbody>
</table>

Figure 4 – Distributed Shared Memory System
<table>
<thead>
<tr>
<th>Ease-of-use</th>
<th>Relatively easy to use with little deviation from serial programming paradigms. Parallelization is usually handled by compiler.</th>
<th>Parallelization needs to be explicitly coded by the user using libraries such as MPI. Debugging is difficult.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>Parallelization benefits hard to achieve after 8 to 10 processors Scale well, and can easily take advantage of faster hardware with no code changes</td>
<td>Problem-specific, and depends on the model of parallel programming chosen</td>
</tr>
<tr>
<td>Portability</td>
<td>Limited portability Easily portable to other platforms, including shared-memory</td>
<td>Limited portability</td>
</tr>
</tbody>
</table>

Table 2 - Comparison of Parallel Programming Systems

The message-passing model is widely used because of its low cost and performance advantages that can be achieved as the general-purpose hardware that it is based on becomes faster. Benefits of faster processors can be realized by simply incorporating them into the cluster. Because of these reasons, this is the model that was used for this project.

Several message-passing libraries are available, the most popular among them being PVM (Parallel Virtual Machine) and MPI (Message-Passing Interface).
For the purposes of this implementation, MPI was used. This model was chosen because it is a widely used standard. MPI programs can be run on any MPI-compliant implementation. There are several freely available implementations such as MPICH that can be set up to run on low cost general-purpose computers in a Beowulf configuration. A Beowulf is a dedicated cluster of interconnected computers with usually only the master node connected to the outside world [10]. Beowulf clusters are built of commercial-off-the-shelf (COTS) components, and hence are relatively inexpensive to construct. This was also easy to learn as a parallel programming methodology. A Beowulf cluster with MPI was also readily available for use at the university. Tests were also run a shared-memory parallel system during the later stages of the project, but using the same message-passing paradigm.
3. IMPLEMENTATION

DNAPenny is implemented in C. The source is available from the PHYLIP website[4]. This chapter describes the existing implementation, and the parallel version.

3.1. Existing Algorithm

The search strategy used by DNAPenny starts by making a tree consisting of the first three species (unrooted) tree. Then it tries to add the next species in all possible places (there are three of these). For each of the resulting trees it evaluates the number of base substitutions. It adds the next species to each of these, again in all possible spaces. The order in which trees are generated follows a depth-first search. The fourth species in the first possible place, then the fifth species in its first possible place, then the sixth and so on until the first possible tree has been produced. For each tree the number of steps is evaluated. Then the algorithm backtracks by trying the alternative placements of the last species. When these are exhausted it tries the next placement of the next-to-last species. The order of placement in a depth-first search is as shown in Figure 5 for a five-species case (parentheses enclose monophyletic groups). The algorithms in the PHYLIP suite use the Newick format to represent phylogenetic trees.

Make tree of first three species: \(((A,B),C)\)
Add D in first place: \(((A,B),(C,D))\)
  Add E in first place: \(((A,B),(E,C))\)
  Add E in second place: \(((A,B),(E,C),D)\)
  Add E in third place: \(((A,B),(E,C),D)\)
  Add E in fourth place: \(((A,B),(E,C),D)\)
  Add E in fifth place: \(((A,B),(E,C),D)\)
Add D in second place: (((A,D),B),C)
Add E in first place:   ((A,(E,D)),(B,C))
Add E in second place:  (((A,E),D),(B,C))
Add E in third place:   (((A,D),E),(B,C))
Add E in fourth place:  (((A,D),((B,E),C))
Add E in fifth place:   (((A,D),((C,E),B))
Add D in third place:  ((A,(B,D)),C)
Add E in first place:   (((A,E),B),(D,C))
Add E in second place:  ((A,(E,B)),(D,C))
Add E in third place:   ((A,B),(E,(D,C)))
Add E in fourth place:  ((A,B),(C,(D,E)))
Add E in fifth place:   ((A,B),(D,(C,E)))

Figure 5 – Tree construction in DNAPenny (Newick format)

As each tree is constructed, including the partial trees such as  ((A,B),(C,D)), its number of steps is evaluated. In addition a prediction is made as to how many steps will be added, at a minimum, as further species are added.

This is done by counting how many sites which are invariant in the data up the most recent species added will ultimately show variation when further species are added. Thus if 20 sites vary among species A, B, and C and their root, and if tree ((A,C),B) requires 24 steps, then if there are 8 more sites which will be seen to vary when species D is added, we can immediately say that no matter how we add D, the resulting tree can have no less than 24 + 8 = 32 steps. If a previously-found tree such as ((A,B),(C,D)) required only 30 steps, then we know that there is no point in even trying to add D to ((A,C),B). This gives the bound that enables us to cut off a whole line of inquiry (in this case five trees) and avoid going down that particular branch any farther. This branch-and-bound algorithm thus allows us to find all most parsimonious trees without generating all possible trees [4].
After each tree is generated, the recursive algorithm for calculating the cost at each site proceeds as follows:

Set Cost C = 0; k = 2n -1, the number of nodes in the tree

To obtain the set Rk

If (k is a leaf node) {
    Rk = sequence data at position k
} else {
    compute Ri and Rj for the two child nodes of k
    Set Rk = Ri \cup Rj
    if (Rk == Ø) {
        compute Rk = Ri\cup Rj
        increment C
    }
}

Figure 6 shows how the cost is calculated for a single site for a 4-taxon tree. The same process is repeated for all sites in the sequence and the total cost added to arrive at the total cost for the tree.

Figure 6 - Calculating Tree Cost
The goal is to find the tree that minimizes the “cost” – the number of steps required to generate the observed variation in the trees. The set of minimum cost trees found is printed out upon termination of the program.

In addition to the using the branch-and-bound approach, the algorithm also includes optimizations that remove input character sites that are not informative\(^2\), and that combine sites that show the same sequence across all the taxa being considered. The modified input data is used for the rest of the algorithm.

### 3.2. Parallel Implementation

The parallelizable part of the algorithm lies in the ability of each process to calculate the cost of each tree independently of the other processes. This is also the most expensive part of the computation. Thus, the nature of the problem lends itself to parallelization by data partitioning [13].

To parallelize the algorithm, the initial approach that was tried was a simple master-slave model. The master process is responsible for generating work (trees to evaluate) for the slave processes. Slave processes calculate the cost and return it to the master. The master process correlates the results from all the trees.

However, with this model, communication requirements overwhelmed the parallelization benefits achieved. Also, since the original algorithm depends on pruning the search space based on intermediate results, and since the intermediate results are not available to the master, this algorithm ends up examining a far larger number of trees than the serial version. Overall, this version of the parallel algorithm ran much slower than the original algorithm. There is also

---

\(^2\) To be informative, a site must have the same sequence character in at least two taxa, i.e., it favors one tree over another.
a high communication cost because the trees are transmitted back and forth from the master to the slaves as the program progresses.

At this point, another approach was tried. Instead of making the master responsible for generating all the trees, all the processes run as peers of each other, except for an initialization and termination phase. The generation of trees happens in parallel, so that there is very little communication between the processes after the computation begins. Each of the nodes involved traverses a certain portion of the tree independently of the others. However, the current lower bound at any instant is only known by a few processors at any instant in time. This lower bound needs to be communicated to all the other processes so they can take advantage of this to prune their search space.

The implementation consists of a master node and several slave nodes. The master node reads the input, calculates the weight vector from the input data, and dispatches this information to the slaves. At this point, the master functions as another slave node, and all the nodes run as peers of each other. The generation of the trees, and the calculation of cost are done independently by all the slave nodes in parallel. Each slave node works on a specific area of the search space as follows, without any further involvement by the master node, which basically controls initialization and termination of the processes. Trees are not sent over the network between the processes, only the best cost found so far.

This approach proved much more effective and lead to much better results. Details of the algorithm follow.

The generation of all possible trees is accomplished by the method addit, that is called with a value of 3 (we start with a three-taxon unrooted tree).

```
addit(m) {
    for (i = 0; i < n; i++)
        addit(m+1);
}
```
This algorithm generates trees in the following order:

Make tree of first three species: \(((A, B), C)\)
Add D in first place: \(((A, D), B), C)\)
Add D in second place: \(((A, (B, D)), C)\)
Add D in third place: \(((A, B), D), C)\)

The number of places that one can add the fourth species to a 3-taxon tree is 3, 5th species to a 4-taxon tree is 5, 6th species to a 5-taxon tree is 7, and so on. So, the fifth species can be added in a total of 15 places considering all the three 4-taxon trees, and the 6th species can be added in a total of 105 places considering all the fifteen 5-taxon trees.

If we have two processors, we can divide the search space roughly in half by adding the 6th species in only 52 of the possible places, and another process adds this species in the 53 possible places. This can be extended to any number of processors less than 105. Each processor bases this decision on its rank in the cluster. Process rank is set when the MPI process is initialized. For example, if there were 15 processors in the cluster, MPI rank would go from 0 (Master) to 9 (Slave processes get ranks 1 through 9). In this case, each process will only add the 6th node in 7 possible places out of the 105 available. Between all the processors, however, the entire tree search space is guaranteed to be covered.

However, this brings about the problem that a certain process may have found the best tree, and the other processes do not benefit from the low bound. To avoid this problem, all the processes synchronize after examining a set number of trees. At the end of the computation, all the processes synchronize, and the cost of the minimum tree found by each process is then compared. The processes that have found the overall most parsimonious trees then print out the tree files.

A schematic flowchart showing the communication of the various parallel processes as they execute is shown in Figure 7.
Figure 7 - Parallel program flowchart. Worker processes vary in number based on the number of processors available. This example assumes that the 6th taxon is added to the tree according to the process's rank.
4. RESULTS AND ANALYSIS

Details of the various tests performed and the results obtained are presented in this chapter. Tests were performed on two clusters. Most of the initial testing/debugging of the algorithm was done on June, a small 8 node Beowulf cluster available in the computer science department. Tests were run on SBC, a 16 node cluster in the Biology department at UCCS. These results are presented here. Later tests were also run on Pleione, a 30 node shared-memory cluster at the University of Texas at Lubbock. These results are also shown here.

4.1. Test Conditions and Assumptions

The SBC cluster in the Biology department is a Beowulf cluster with 16 nodes, including the master node. It has the following configuration:

Master: Dell Dimension PIII 800 MHz with 256MB of memory

Slaves: HP Kayak XA PII 233MHz with 96MB of memory.

OS: Red Hat Linux Version 6.2 (Kernel 2.2.19-12.beo)

MPI Implementation: Scyld Version 27BZ-8

Tests were run with varying number of nodes from 2 to 16. An arbitrary synchronization frequency of 1000 trees was set for the initial round of tests.

Although the master node has a much faster processor than the slave nodes, for comparison with the serial version, it is assumed that the entire system is running at a processor speed of 233MHz. This is a reasonable assumption, since the computation does not terminate until the last processor has finished its task. The communication mechanism used for synchronization of the lowest cost between the processes is implemented using a MPI "barrier" call,
MPI_Allreduce. The barrier ensures that all the processes have reached the same point in code and are ready to proceed.

Baseline tests with the serial version of the program were run on hardware with the following configuration:

Dell PII 350MHz with 392MB of memory (Single Processor)

Since the parallel version of the program was run on a cluster with 233 MHz processors, results obtained for the serial version of the program were not directly comparable with the parallel version. Since the original algorithm is computationally intensive, the processor speed has a high impact on the times obtained. To be able to compare the results, a multiplying factor proportionate the difference in speeds was applied to the results. The factor was calculated as follows:

\[
\text{Clock speed of processor for serial version} \quad 350
\]

\[
\text{-----------------------------------------------} = \quad ---
\]

\[
\text{Clock speed of each processor for parallel version} \quad 233
\]

For instance, if the recorded time for examining 14 taxa when running the serial version of the algorithm is 1000 seconds, the adjusted time is \(1.5 \times 1000 = 1500\) seconds.

The "Zilla" Dataset with varying number of species was used as the test dataset. This is a standard dataset used for comparing phylogenetic algorithms[6].

The best trees as found by both versions of the algorithm were compared to make sure that the parallel version was producing the same result as the baseline serial version. The trees obtained in each case are shown in Appendix A. In each case, the first species is chosen as the outgroup.

Data points were obtained by adding two processors at a time, with the first test run with two processors.
4.2. **Results**

The results obtained are summarized in Table 3. Times are in seconds, and show how the time varies with number of processors or number of taxa.

Times are noted using the MPI timer, MPI_Wtime(). Processes are synchronized with an MPI_Barrier call before noting the end time to ensure that all processes have completed execution.

The single processor times are multiplied with the factor 1.5 to account for the slower processors for the parallel version. Speedup is calculated using the formula[13]:

\[
S(n) = \frac{\text{Execution time using one processor}}{\text{Execution time using n processors}} = \frac{t_s}{t_p}
\]

where \( t_s \) is the execution time on a single processor and \( t_p \) is the execution time on a multiprocessor. \( S(n) \) gives the increase in speed in using a multiprocessor.

These results are displayed as a graph in Figure 6.

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10 Taxa</td>
</tr>
<tr>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>

19
<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>35</td>
</tr>
<tr>
<td>10</td>
<td>37</td>
</tr>
<tr>
<td>12</td>
<td>36</td>
</tr>
<tr>
<td>14</td>
<td>36</td>
</tr>
<tr>
<td>16</td>
<td>22</td>
</tr>
</tbody>
</table>

Table 3 – Test Results on SBC Cluster

![Test Results on SBC Cluster](image)

Figure 8 – Test Results on SBC Cluster
Figure 7 shows the speedup obtained in comparison with ideal speedup.

![Speedup Graph]

Figure 9 - Speedup

4.3. Analysis

The speedup obtained for 16 processors lies in the range 12 to 15. The running time of the 10-taxa case is too small to benefit from parallelization, and is excluded. Given that the ideal speedup is 16, the speedup that is achieved is good. Several factors that effect the performance of the parallel version. Some of these factors add overhead that has to do with parallelization, while there are others that directly improve the running time of the algorithm. These are:

- Parallelization Overhead: There are several periods when some processes are waiting on other processors for work. These periods include:
• Startup and Initialization: The master process reads the input and processes it before dispatching it to all the other processes.

• Synchronization: Depending on the synchronization frequency, processes block several times during execution to get the current lowest bound.

• Communication: This is a necessary part of the algorithm, to ensure that the processors have access to the lowest bound discovered so far. In addition, there is a small amount of communication overhead at startup.

• Performance gains due to parallelization: Looking at the speedup, it would appear that the overhead contributed by all the above factors is small, but there is one other factor that influences the time in the parallel version. Because each of the individual processes follows a separate path down in the search space, it is likely that a narrower bound is found by one of the processes in the parallel version faster than it can be found with a single process. Since this is broadcast to all the other processes at periodic intervals, all the processes benefit from being able to prune the search space more quickly than when running in a serial fashion.

• Load Balancing: Because the search space is based on a bound that is not known until run-time, work is not divided among the processes evenly. The amount of disparity depends to a large extent on the input data. Because of this, it was noticed that some processes finish their computations ahead of some of the others, and then are idle waiting for the other processes to finish.

4.4. Scalability Testing

Scalability: The term scalability is used to describe the ability of the parallel algorithm to take advantage of increased processing power, as more nodes become available. To test the scalability of the parallel version of the algorithm, tests were run on a 56 processor cluster at Texas Tech University at Lubbock. This cluster was based on a shared-memory architecture,
but the algorithm was run unchanged. As previously mentioned, this is one of the advantages of the message-passing model of parallel programming; the program can be run without code changes across multiple platforms, with only a recompile for the target platform.

This cluster, Pleione, is built using SGI hardware with the following configuration:

SGI Onyx2 with 56-300 MHz processors, 56 GB of shared RAM, and an HPC- Linpack rating of 24 Gflops, 530 GB of fiber channel RAID storage, 620 GB of SCSI storage, and a 2-pipe Infinite Reality 2 graphics system

Pleione ran an SGI implementation of MPI. It is part of the High Performance Computing Center at Texas Tech University [16]. Jobs are submitted and managed using the LSF [17] queueing system. This is a batch workload management software that manages parallel jobs submitted by multiple users.

Results obtained during these tests are presented in Table 4 and Figure 8.

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Time(seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>13 Taxa</td>
</tr>
<tr>
<td>8</td>
<td>711</td>
</tr>
<tr>
<td>16</td>
<td>442</td>
</tr>
<tr>
<td>20</td>
<td>379</td>
</tr>
<tr>
<td>25</td>
<td>394</td>
</tr>
<tr>
<td>30</td>
<td>295</td>
</tr>
</tbody>
</table>

Table 4 - Test Results on SGI Cluster
This shows that speedup obtained on this cluster is not as impressive as on the SBC cluster. The speedup obtained for 30 processors is close to that obtained for 16 processors on the SBC cluster. The times rapidly fall up to about 16 processors, then performance gain obtained from parallelization decreases considerably.

4.5. Tests with Larger Datasets

The various limitations to methods used in phylogenetic analysis lead to questions about the veracity of the resulting trees [3]. Several methods are used to test the accuracy of the resulting
phylogenetic tree. One of the methods recommended is to compare the output of several methods such as maximum parsimony, maximum likelihood, etc [2]. There are also several heuristic methods that predict the phylogenetic relationships between species.

The parallel version of the algorithm also cannot handle exhaustive searches of larger datasets. However, it can examine a bigger search space in a much smaller amount of time, allowing one to find trees that may have a lower total cost. A higher number of trees examined also give us a greater degree of confidence in the resulting tree. These trees can then be compared with trees produced from other algorithms to help in predicting the actual phylogenetic relationship.

Tests were run with larger datasets on both the serial version and parallel version of the algorithm for fixed run times, to compare the number of trees examined, as well as the cost of the minimum tree found.

The parallel version of the tests was run on the same SGI Cluster that was used to run the scalability tests.

The test was set to exit after 45 minutes. This number was chosen because of the cpulimit for user jobs on Pleione. The cost of the best tree produced, as well the total number of trees examined (in ‘000s), is presented. These results are presented in Table 5 and Figure 9.

The results indicate that the parallel version of the algorithm finds a better tree in some cases.

<table>
<thead>
<tr>
<th>Processors</th>
<th>20 Taxa</th>
<th>30 Taxa</th>
<th>40 Taxa</th>
<th>50 Taxa</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Trees</td>
<td>Cost</td>
<td>Trees</td>
<td>Cost</td>
</tr>
<tr>
<td>1</td>
<td>184</td>
<td>849</td>
<td>165</td>
<td>1144</td>
</tr>
<tr>
<td>30</td>
<td>4930</td>
<td>846</td>
<td>2700</td>
<td>1143</td>
</tr>
</tbody>
</table>
For smaller datasets (up to 16 taxa), results obtained from the parallel version of the algorithm were compared with the serial version to make sure that the results obtained were consistent, i.e., the parallel version should produce the same output trees as the serial version.

### 4.6. Validation Testing

In order to verify the trees generated by the algorithm, tests were run with several datasets containing taxa with phylogenetic relationships determined by other means. For this purpose, data was obtained from TreeBASE, an online phylogenetic database of phylogenetic information [18]. This database contains several phylogenetic trees from submitted from
research papers. For the purposes of testing, datasets that has DNA sequence data available were chosen. DNAPenny cannot handle protein data. The outgroup used in the resulting tree was used as the outgroup for the test, to make tree comparison easier. In each case, the documented tree, as well as the tree produced by DNAPenny are depicted.

The following subsections show three different datasets obtained from the TreeBase catalogue and compared with the trees produced by the DNAPenny algorithm. All tests were run on Pleione, the MPI cluster at TTU with 30 processors, and timed to terminate after 45 minutes.

4.6.1.


This is a 21 taxa tree with a DNA sequence length of 1635 characters. The tree was obtained using maximum likelihood analysis.

(((Dasybranchus_caducus,(((Praxilella_pacifica,((Nicomache_sp,Clymenura_clypeata),Johnstonia_clymenoides)),Unclassified_maldanid),(Metasychis_disparidentata,Maldane_sarsi)),(((Arenicola_cristata,Arenicola_loveni),Arenicolides_ecaudata),Arenicola_marina),((Branchiomaldane_sp,Branchiomaldane_vincenti)),((Abarenicola_claperedi,Abarenicola_pacifica),(Abarenicola_affinis,Abarenicola_gilchristi)))
,Scalibregma_inflatum),((Sternaspis_scutata,Poecilochaetus_serpen)s)
)

The tree obtained using DNAPenny is presented below:

((((((((Branchiomaldane_sp,Branchiomaldane_vincenti),Arenicolides_ecaudata),Arenicola_marina),(Arenicola_loveni,Arenicola_cristata)),(Abarenicola_gilchristi,Abarenicola_affinis),(Abarenicola_pacifica,Abarenicola_claperedi)),(((Clymenura_clypeata,Nicomache_sp),Johnstonia_clymenoides),Praxilella_pacifica),Unclassified_maldanid),Metasychis_disparidentata,Maldane_sarsi),((Sternaspis_scutata,Poecilochaetus_serpen)s),Scalibregma_inflatum),Dasybranchus_caducus))
The trees produced by both algorithms are different. However, there are several sub branches that are found to share the same phylogenetic relationships in both cases. These sub trees are shown here.

(((Abarenicola_gilchristi, Abarenicola_affinis), (Abarenicola_pacifica, Abarenicola_claperedi))
(((Sternaspis_scutata, Poecilochaetus_serpen)s), Scalibregma_inflatum)
(Branchiomaldane_sp, Branchiomaldane_vincenti)
(((((Clymenura_clypeata, Nicomache_sp), Johnstonia_clymenoides), Praxilella_pacifica), Unclassified_maldanid))
(Arenicola_loveni, Arenicola_cristata)

4.6.2.

This is a 15-taxa tree with a DNA sequence length of 1972 characters. The tree was obtained using parsimony analysis.

(((Hibiscus_grandiflorus_FL1_allele1, Hibiscus_grandiflorus_FL2), (Hibiscus_grandiflorus_FL1_allele2, Hibiscus_incanus_FL), (Hibiscus_moscheutos_TN, Hibiscus_lasiocarpos_IL), Hibiscus_palustris_NY, Hibiscus_dasycalyx_TX1_allele2)), (Hibiscus_laevis_TN, Hibiscus_laevis_NE), (Hibiscus_dasycalyx_TX1_allele1, Hibiscus_dasycalyx_TX2)), (Hibiscus_coccineus_MOBOT, Hibiscus_coccineus_FL)), Hibiscus_trionum);

DNAPenny found 116 trees with the same cost. Since this can happen when one is solely comparing cost ignoring other factors for possible phylogenetic relationships, inferences obtained in this manner need to be narrowed down using other techniques. The list of trees produced includes the tree as identified by the study above.

4.6.3.
This is a 22 taxon tree with a DNA sequence length of 1573 characters.

```
```

DNAPenny found 12 trees with the same cost. These trees do not match exactly the tree as found above, but several show consistent patterns that show the same relationships.

```
(((Cortinarius_marylandensis,Cortinarius_caperata),Cortinarius_violaceus),Descolea_gunnii),(((Hebelomina_neerlandica,Gymnopilus_penetrans),Gymnopilus_aeruginosus),Galerina_paludosa),Cortinarius_vibratilis),(((Agrocybe_dura,Agrocybe_praecox),Anamika_indica),((Hebeloma_longicaudum,Hebeloma_crustuliniforme),Naucoria_escharoides),Naucoria_bohemica),((Hypholoma_capnoides,Hypholoma_fasciculare),Pholiotina_subnuda),Phaeomarasmius_erinaceus),Tubaria_hiemalis),Tubaria_furfuracea)
```

### 4.7. Suggestions for Improvement

This section lists some of the problems that were encountered during the course of the project. Some of these problems have been resolved in the current implementation, while others have not been resolved. Potential ways to address some of the unresolved issues are also presented here.

The parallel implementation is hard-coded to work with MPI. This limits it to the number of platforms where it can be run. It would be useful to modify this to allow for different parallel implementations such as PVM (Parallel Virtual Machine) to be plugged in.
The current algorithm, even with the parallelization, is slow. It is not possible to search the entire tree space exhaustively. However, because of the fact that we search a greater search space, it is possible to find a better tree than would be possible otherwise. However, this can be improved further. The algorithm is a branch and bound algorithm. This means that if we could establish a lower bound at an early stage in the algorithm, it could result in a saving in time, because of the reduced search space, even though there is no effect on the parallelization of the algorithm itself. The amount of saving that would be obtained would depend on the actual bound that can be specified. To arrive at a “reasonable” lower bound to begin with, it is possible to use some other heuristic algorithm (such as DNAPenny) and use its result to mark the lower bound for the branch and bound algorithm. Since the branch and bound algorithm is guaranteed to find the most parsimonious tree, any tree found by an approximate algorithm is bound to have a higher or equal cost than the most parsimonious tree.

All the nodes in the cluster currently synchronize results on a set frequency of 10000 trees. This has been found to be the best value for the dataset under consideration based on experimentation with different numbers. It is hard to find out what the ideal value should be, as it varies widely depending on the dataset. On the one hand, one does not want to be synchronizing too frequently, as it increases communication costs. On the other hand, if a lower bound has been found by any one of the nodes, it is advantageous to communicate it to the other nodes, so they do not waste time searching trees that are not likely to produce a lower cost.
5. CONCLUSIONS

This parallel implementation of the parsimony-based DNAPenny algorithm results in better times for inferring phylogenetic trees. Parallelization for up to 16 processors is effective, achieving numbers close to ideal speedup. However, performance appears to degrade with greater number of processors, possibly due to the effect of inequitable load distribution.

Because of the nature of the algorithm it is not possible to run exhaustive searches of the search space even when running in parallel. However, it is possible to use the parallel version of the algorithm to run tests that cover a larger portion of the search space than would be possible with a single processor, which may lead to the inference of trees with lesser cost.
APPENDIX A

Phylogenetic trees inferred from different runs of DNAPenny with various numbers of taxa. The first few trees up to 14 taxa are illustrated in tree form, others up to 50 taxa are presented in the Newick format so that entire tree can be shown on a single page.

10-Taxa Tree, Cost 447.000

+--------------------------Nicotiana
  !
  !  +--Galphimia
  !
  !   +----2
  !
--1
  !  +----5
  
  !  +--Victoria
  !
  !  +--4  +--7
  !
  !  +--Barclaya
  !  !
  +--9
  !
  !  +--Ipomoea
  !
  !  +---------8
  !
  !  +--Convolvul
++6
  
++6
  !  +-------------Petunia
  !
  +-------------Lycopersi

11-Taxa Tree, Cost 527.000

+--------------------------Nicotiana
  !
  !  +--Galphimia
12-Taxa Tree, Cost 573.000
!     ! ! +---Borago
!     ! !
! +---9 ! +---Ipomoea
! ! ! +-----------10
! ! ! +---Convolvul
+11 !
  ! +-----------Petunia
  ! +-----------Lycopersi

13-Taxa Tree, Cost 599.00

+-------------------------Nicotiana
!
! +--------Galphimia
! +---2
! ! ! +-----Cypirapea
! ! +---5
! +--------4 ! +---Victoria
! ! ! +---8
! ! ! +---Barclaya
! ! !
--1 +----3 +---------Oenothera
! !
! ! ! +---Hydrophyl
! ! ! +--10
! +---7 +---------6 +---Heliotrop
! ! !
! ! ! +-----Borago
! ! !
! +---9 ! +---Ipomoea
! ! ! +-----------11
! ! ! +---Convolvul
+12 !
  ! +---------Petunia
14-Taxa Tree, Cost 617.000

+------------------------------------------Nicotiana
!
!
+--------Galphimia
!
+--2
!
+-----Cypirapea
!
+--5
!
+--------4  +--Victoria
!
+--8
!
+Barclaya
!

--1  +-----3
!
+-----Eriodicty
!
+--13
!
+---Hydrophyl
!

+--7  +----------6  +---Heliotrop
!

+---Borago
!

+--9  +---Ipomoea
!

+--11
!

+-12  +
!

+---Petunia
!

+--------------------------------------Lycopersi
20 Taxa Tree, Cost 849.0

((((((((((Callitric, Digitalis), Buddleja), Lamiumpur), Jasminum), ((((Barclaya, Victoria), Cypirapea), Galphimia), Oenothera), Adoxa)), (((Hyd rophyl, Eridocity), Heliotrop), Borago)), (Ipomea, Convolul)), Petunia), Lycopersi)

30 Taxa Tree, Cost 1143.0

((((((((((((((Callitric, (Digitalis, Antirrhin)), Verbena), (Lamiumpur, Biblis)), ((Piguicul, Utricular), Sesamum), Streptoca), Buddleja), Cat talpa), Jasminum), Vahlia), (((Eriodicty, Hydrophyl), Heliotrop), Borago)), ((Barclaya, Victoria), Cypirapea), ((Galphimia, Oenothera), Pterost)), (Ipomea, Convolvul)), Petunia), Lycopersi), Nocotiana)

40 Taxa Tree, Cost 1413.0

((((((((((((((Lamiumpur, Physosteg), Pogostemo), ((Teucrium, Clerod end), (Callicarp, Salvia))), (Scutellar, Prostanth)), Sesamum), (Catalpa, Buddleja)), (((Digitalis, Antirrhin), Callitric), Verbena)), ((Utricular, Pinguicul), Byblis), Streptoca), (Jasminum, Lingustrum)), Vahlia), (((Barclaya, Victoria), Cypirapea), Galphimia), Pterost), Oenothera), (Symphoric, Adoxa)), (((Eriodicty, Hydrophyl), Heliotrop), Borago)), (Ipomoea, Colvolvul)), Petunia), Lycopersi), Nocotiana)

50 Taxa Tree, Cost 1835.0

((((((((((Buddleja, Catalpa), (Sesamum, Streptoca)), ((Utricular, Piguicul), Biblis)), (((Antirrhin, Digitalis), Callitric), Verbena)), (((Teucrium, Clerodend), Prostanth), Scutellar)), (((Lamiumpur, Phososteg), Pogostemo), (Callicarp, Salvia))), (Ligustrum, Jasminum)), (Vahlia, Gent iana)), (((Galphimia, Oenothera), (Buddleja, Victoria), Cypirapea)), Pterost), (Coriandru, Hedera), (Cornuskou)), (((Valeriana, Dipsacus), Symphoric), Adoxa), (Menyanthe, Villarsia)), (Boopis, Lobelia))), (((Eriodicty, Hydrophyl), Heliotrop), Borago)), (Ipomoea, Convolvul)), Petunia), Lycopersi), Nocotiana)
DNA sequences from the Zilla dataset were used as test data. Sequence length for each taxa is 759 characters. Total number of taxa in the dataset is 500. The following shows the 50 taxa used for testing, with the first 50 characters of each sequence.

Nicotiana  GTTGACAATTAGCAATGCTTTTTTGAGCCAAACCAAGGATTTTATGAAC
Galphimia  .......G....TT......C.....CTGG......A.....C............
Oenothera   ........CC..TT................A.G........A.....C.......
Victoria   ..............T..G......C....TTG......CTT.C....C......
Cyprapea   ????????????.T..................CTG........A.....C.....
Barclaya   ..............T..G......C....TTG......CTT.C....C......
Petunia    .................................................G.
Lycopersi  ......................................................A....
Convolvul  ???????????????????????????????????????????????????..C....
Ipomoea    ???????????????????????????????????????????????????..C....
Borago     ???????????????????????????????????????????????????..C....
Heliotrop  ???????????????????????????????????????????????????..C....
Hydrophyl  ..........................................................A.G....A.....C.....
Eriodicty  ..........................................................A.G........A.....C.....
Digitalis  ........G.............................................A.S........A.....C.....
Buddleja  ???????????????????????????????????????????????????..C....
Callitric  ???????????????????????????????????????????????????..C....
Jasminum   ................GG....................................A.A........A.....C.....
Adoxa      ..C..........T...............................CTG........A.....C.....
Lamiumpur  ........G....T.........C..A.G........A.....C.....
Streptoca  ???????????????????????????????????????????????????..C..G....
Catalpa    ???????????????????????????????????????????????????..C....
Sesamum    ???????????????????????????????????????????????????..C....
Utricular  .........................................................GC........A.G........A.....C.....
Byblis     ........G....T...............................A.G........A.....C....T.
Pterost    ........A..T.............................CT........A.....C.....
Vahlia     .............................................ATG........A...G..C.....
Antirrhin  ???????????????????????????????????????????????????..C....
Verbena    ???????????????????????????????????????????????????..C......
<table>
<thead>
<tr>
<th>Species</th>
<th>Genetic Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teucrium</td>
<td>?????????????????????????????????????????????????????????????...C......</td>
</tr>
<tr>
<td>Prostanth</td>
<td>?????????????????????????????????????????????????????????????????????...C......</td>
</tr>
<tr>
<td>Clerodend</td>
<td>?????????????????????????????????????????????????????????????????????...C......</td>
</tr>
<tr>
<td>Salvia</td>
<td>........G.................A.G......A........C......</td>
</tr>
<tr>
<td>Ligustrum</td>
<td>?????????????????????????????????????????????????????????????????????...C......</td>
</tr>
<tr>
<td>Scutellar</td>
<td>........G.................C.C.C.????..G...........C......?</td>
</tr>
<tr>
<td>Physosteg</td>
<td>?????????????????????????????????????????????????????????????????????...C......</td>
</tr>
<tr>
<td>Callicarp</td>
<td>?????????????????????????????????????????????????????????????????????...C......</td>
</tr>
<tr>
<td>Pogostemo</td>
<td>?????????????????????????????????????????????????????????????????????...C......</td>
</tr>
<tr>
<td>Symphoric</td>
<td>?????????????????????????????????????????????????????????????????????...C......</td>
</tr>
<tr>
<td>Valeriana</td>
<td>.C......GC..T.............CT........A.....C......</td>
</tr>
<tr>
<td>Dipsacus</td>
<td>.C......GC..T.............CTG........A.....C......</td>
</tr>
<tr>
<td>Boopis</td>
<td>....T.G....TT............CTG...........C...C...</td>
</tr>
<tr>
<td>Menyanthe</td>
<td>?????????????????????????????????????????????????????????????????????...C......</td>
</tr>
<tr>
<td>Villarsia</td>
<td>?????????????????????????????????????????????????????????????????????...C.A.....</td>
</tr>
<tr>
<td>Lobelia</td>
<td>....A.G...TT...A........CTG...........T....T...?.T..</td>
</tr>
<tr>
<td>Gentiana</td>
<td>?????????????????????????????????????????????????????????????????????...C......</td>
</tr>
<tr>
<td>Cornuskou</td>
<td>.....................ACCTG........A.....C......G</td>
</tr>
<tr>
<td>Hedera</td>
<td>............T..G..........CT........A.....C.C....</td>
</tr>
<tr>
<td>Coriandru</td>
<td>?????????????????????????????????????????????????????????????????????...C......</td>
</tr>
</tbody>
</table>
BIBLIOGRAPHY


17. Platform LSF. http://www.platform.com/products/LSF/