Finding Repeats and Signatures in DNA Sequences Using MPI Clusters

Technical Report FIU-SCIS-2009-04-02
April 14, 2009

Michael Robinson¹, Guangyuan Liu², Camilo Silva¹, S. Masoud Sadjadi¹, Giri Narasimhan¹

¹{mrobi002, csilv003, sadjadi, giri} @cs.fiu.edu;
School of Computing and Information Sciences
Florida International University
11200 SW 8th Street
Miami, FL 33199

²lgy@cnic.cn
Computer Network Information Center
Chinese Academy of Sciences
4, 4th South Street, Zhongguancun, Haidian District
Beijing, China P.O. Box 349 (100190)

2009
Abstract
The problem of identifying subsequences that are either repeated or unique has many applications in Bioinformatics. A number of algorithms and implementations exist for these problems, but are compute-intensive. We use an algorithm based on suffix arrays for these problems and implement it to run on MPI clusters and demonstrate acceptable speedup.

**Keywords:** Genome, repeats, inverted repeats, complemented inverted repeats, sequences, probes, nucleotides.

1. Introduction
The main characteristic of genomic data is its large size. For example, Genbank [1], which is the NIH sequence database, has a total of over 65 billion nucleotides.

One of the most striking features of genomic DNA sequences is the extent to which repeated substrings occur in the genome. In *C. elegans* with a genome sequence of 100.2 million nucleotides over 7,000 families of repeat sequences have been identified. Families of repeat sequences account for about one third of the human genome. Repeat sequences come in many different flavors and are responsible for different functions and diseases. Finding repeats has applications in finding defective genes, and in forensic DNA fingerprinting. It also allows us to find differences between genomes.

There are a number of programs available for finding repeats, such as RepeatMasker [2] a popular software for finding repeats. However, the problem is clearly compute-intensive and, creates costly bottlenecks in large-scale analyses.

We have implemented an algorithm based on suffix arrays for finding repeats and unique signatures, which was then ported to run on MPI [3] clusters with good results. This algorithm also searches for direct repeats and other variants such as inverted repeats and complemented inverted repeats.

The implementation on a single computer appears to be very fast. It searches for single patterns in a sequence of roughly a few million bases in less than a second.

We tested our data on five strains of the bacteria *Pseudomonas aeruginosa*, each containing over six million nucleotides. The serial implementation completed its task averaging 30 hours for every pair of genomes. The performance of the MPI implementation was compared to that of the serial one. All nucleotide sequences were acquired from NCBI [4].
A similar approach by Afgan and Bangalore [5] utilizes grid computing to implement BLAST. Our results for scalability and balancing seem to be better, probably because of the nature of the problems being tackled.

2. Algorithms

2.1 Serial Algorithm
Our serial algorithm consists of three modules. The first one creates an index from the genomic sequences provided. The second one (Search) helps in searching for the presence of a probe in the genomic sequence by searching in the created index. The third one (Signature Search) searches for “signatures” or probes present in one genome but absent from the others. This serial implementation was used as an “off-the-shelf” program in our cluster implementation.

2.2 Cluster Algorithm
The parallelized cluster algorithm provides a communication paradigm of control that manages tasks sent from an administrator node (AN) to worker nodes (WN) in a cluster [Fig. 1]. This communication paradigm allows the AN to keep track of each task completed by each WN through the usage of MPI point-to-point communication—such functionality shall be referred as the Task Completion Control. At the same extent, the AN has the functionality of supervising the efficiency of each WN’s task’s time completion. Such functionality establishes a fault-handling control in cases where the cluster resources in a specific node or several ones are scarce due to non-exclusiveness of the current CPU usage—this functionality is known as the Time Efficiency Fault Handling Control.

Fig. 1: Communication Paradigm – The AN is responsible of sending the tasks to all WNs. Each WN will execute the sequential algorithm and later send a completion message to the AN. The AN will then be able to determine when a task is completed by disseminating the completion message sent from each WN.
2.3 Task Completion Control
This MPI implementation provides improved scheduling and management strategies. This implementation has a task queue known as the Task-Tag (TT) Registry that identifies the execution states of every task. It also provides the ability to read task information from a configuration file created by the user, which acts as the input file of the tasks to be processed. This implementation has worked properly on the GCB [6] cluster and other clusters such as Starscream. Below are the details of these features.

2.3.1 Task-Tag (TT) Registry
The master node maintains a TT registry which contains information on all tasks and their execution states. Every task contains a state tag that identifies its current execution state. There are a total of four states: completion, progression, timed-out, and timed-out in progress [Table 1].

<table>
<thead>
<tr>
<th>TAG</th>
<th>STATE DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>completion</td>
<td>Indicates when a task is completed successfully.</td>
</tr>
<tr>
<td>progression</td>
<td>It is the state defining a task that is being processed by a WN.</td>
</tr>
<tr>
<td>timed-out</td>
<td>This state is assigned to a task whenever its WN has timed-out to complete such task.</td>
</tr>
<tr>
<td>timed-out in progress</td>
<td>This state identifies a task that is timed-out but reassigned to another particular WN.</td>
</tr>
</tbody>
</table>

Table 1: State Definitions – This table provides the definitions for the TT registry. The AN maintains an active supervision of the states of each task and it continuously modifies them according to each task’s progression and completion.

Whenever a task is sent to a WN by the AN, its tag value will be initialized to progression. When a task is successfully processed, the WN will send a completion message, and the AN will change the state of the task to completion. With the TT registry, the WNs do not need to pay attention to the task they are currently executing; the AN can easily manage and schedule the tasks by simply modifying the tag states. This method aids in reducing the complexity of the distribution of tasks and improves the flexibility and performance of task management by keeping track of the progress of each task, especially when there are more tasks than worker nodes.

In the following figure [Fig. 2], it is depicted how four tasks are scheduled on a cluster through the use of the TT registry.
Fig. 2: Task-Tag (TT) Registry – In this figure, one can appreciate how each task is assigned with a different state value. It can also be seen how the performance of each WN affects the state of a task.

2.4 Time Efficiency Fault Handling Control

Part of the success of maintaining and reaching a maximum efficiency of the processing of each task in a cluster depends on how well each WN is able to compute each given task at an acceptable processing time. In some cases, the efficiency of a WN is affected due to lack of resources whenever a cluster does not provide exclusiveness for the tasks to be completed. Thus, one of the core functionalities of this parallel algorithm is to be able to handle such lacks of resources on any specific node.

2.4.1 Threshold Default Time

In order for the AN to supervise the efficiency of each WN, it needs to have a time assessment of the time it takes for a task to be completed. For that reason, a threshold default completion time is computed after the first task is successfully completed—it is computed from the timing of this task plus three times more of its completion duration.

2.4.2 Nodes-Task-Information (NTI) Registry

The AN also needs to keep track of the states of each WN in the cluster, consequently, a nodes-task-information (NTI) registry is created to hold the current states of all WNs. There are a total of five different states that represent a WN’s current processing state: inactive, timed-out activity, wait, awaken-inactive, and normal processing. The following table explains in detail each state description [Table 2].
<table>
<thead>
<tr>
<th>WN STATE</th>
<th>STATE DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>inactive</td>
<td>This state indicates any WN that is inactive and it is ready to process a task.</td>
</tr>
<tr>
<td>timed-out</td>
<td>Represents a WN that has exceeded the threshold completion time to complete a task.</td>
</tr>
<tr>
<td>wait</td>
<td>Indicates the state of a WN that is waiting for the resources to be available once again.</td>
</tr>
<tr>
<td>awaken-inactive</td>
<td>This state represents a WN that was previously on the wait state but is now ready to process a new task.</td>
</tr>
<tr>
<td>normal processing</td>
<td>This state indicates that the WN is normally processing the task assigned.</td>
</tr>
</tbody>
</table>

Table 2: WN States – This table defines the different states that all the WN have at any specific time throughout the processing of all tasks.

2.5 Cluster Algorithm Implementation

The MPI implementation is self-managing since many tasks are sent to be completed to various WNs. Therefore, the AN knows when a task finishes, when a node is available, and also whether a node is taking a long time to complete a task.

In the figure below [Fig. 3], assuming only two nodes are working in a cluster and three tasks are to be completed, the AN node initializes each WN by assigning tasks from a task list, which contains all tasks to complete. At the same time, the NTI registry is initialized accordingly. Such registries contain integer values that characterize and provide current state values to both the tasks and WNs at any moment in time.
At first, the 1\textsuperscript{st} WN completes a task—notice that the \textit{threshold default time} is computed after the first task is completed. The states in the TT and NTI registries are changed accordingly as time progresses. It can be seen that the task in the 2\textsuperscript{nd} WN is \textit{timed-out}—probably because there is lack of resources at that moment.

Consequently, the \textit{timed-out} task will be reassigned to the 1\textsuperscript{st} WN since it is available. The states change accordingly in the registries and the 1\textsuperscript{st} WN now has the \textit{normal processing} state and the task is assigned to \textit{timed-out in progress}.

The 1\textsuperscript{st} WN then completes the task assigned and an interrupt signal is sent from the AN to the 2\textsuperscript{nd} WN to stop its processing. Such node’s state is changed to \textit{wait} (a random waiting time is computed). The next task assigned to the 1\textsuperscript{st} WN is the 3\textsuperscript{rd} task.

After the random time computed has elapsed, the 2\textsuperscript{nd} WN will awaken and it will be ready for any available task assignment. In the end, all WNs will be assigned with the next available tasks. Once a node completes a task, the AN will assign the next task available. That
iterative process is repeated until the entire tasks are completed.

3. Results

The task was broken down into many smaller subtasks each of which corresponds to a process.

The figure below [Fig. 4] represents the results of the process run on the GCB cluster, showing scalability and balance.

![GCB Cluster at FIU](image)

**Fig. 3 GCB Runtime** — Running times of the tasks for each node of the cluster.

![Speedup](image)

**Fig. 5 Speedup** — Speedup results as a function of number of slaves nodes.
The above figure [Fig. 5] represents the running time for all data to be processed by one, two, three, four, and five slave nodes.

![Graph showing running time for different nodes](image)

**Fig. 6:** *Running Time* — Running time as a function of number of slave nodes.

### 3.1 Data validation

Two data validation processes were conducted in this project, validation of the searching process and validation of the Parallel process. The data validation process proved that the implementation of the parallel program worked correctly.

### 4. Conclusion

This project had a dual purpose, one from the Bioinformatics area and the other from the Parallel Computing area. We have been able to achieve both goals successfully. We were able to find the differences between five strains of *Pseudomonas aeruginosa* for probes of length 10; this process will allow us to find other differences using any probe length, utilizing the methods that we developed to validate the results.

As expected, the MPI cluster implementation allowed us to reduce the amount of time needed to process this project. We used two different MPI clusters, at Florida International University in Miami, one called GCB with 6 nodes, and the other startscream with 16 nodes. We were able to get better time results on the GCB cluster because we obtained exclusive access to run our experiments.

Our algorithms use current published genomic DNA. Users can search for repeats and signatures using DNA probes of any size. We
can do cross species searches (i.e., bacteria, virus, animal, plant).

We have successfully implemented our “off-the-self” application showing almost perfect scalability and balancing. We believe that following the lessons learned here, we will be able to implement more challenging Bioinformatics applications.

We believe that our techniques can be extended to other bioinformatics applications.

### 6. Future Work

An interesting topic of a broader implementation and usage of this application would be to create a dynamic web interface of the application. Such web interface could be a friendly interface that would allow the user to create a job list for the cluster to process. After the cluster finishes with the processing of the information, some output data would be created for the user to analyze.

### 7. Acknowledgments

We would like to take the opportunity to thank the people that helped us through out the planning and development of this project. Dr. Hector Duran from University of Guadalajara for his advice and guidance given to Camilo Silva while he was in Guadalajara, Mexico. A good number of FIU students had the generosity to help all of us at a challenging moment of our research with the MPI implementation and we are thankful to all of them: Javier Delgado, Javier Figueroa, Sean Leslie, Allison Lanager, Juan C. Martinez, and David Villegas. More importantly, many thanks to all the academic institutions and different programs that funded our research such as the NSF, GCB Cyberbridges Fellowship Program, PIRE Program, and REU.

Global CyberBridges Fellowship Award Announcement for 2008 National Science Foundation Office of Cyberinfrastructure CI-TEAM Program Award Id: OCI-0636031 October 1, 2006-December 31, 2009. Most importantly, many thanks to FIU and UdeG academic institutions and the programs that funded our research, namely, NSF PIRE Grant No. OISE-0730065, and REU Grant No. IIS-0552555. GN was partly supported by NIH/NIGMS Grant No. S06 GM008205.

### 8. References

[1] National Institutes of Health (NIH) created in 1887 within the Marine Hospital Service (MHS established in 1798), predecessor agency to the U.S. Public Health Service (PHS).

A tool for comprehensive analysis of BLAST and RepeatMasker results. In Silico Biology 7, 0027 (2007); ©2007, Bioinformation Systems e.V.


