Computer Systems Research

# An Investigation into Implementations of DNA Sequence Pattern Matching Algorithms

Peden Nichols

April, 2004-2005

## Abstract

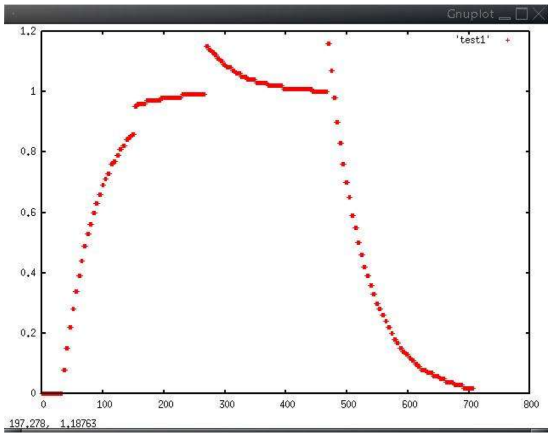
The BLAST (Basic Local Alignment Search Tool) algorithm of genetic comparison is the main tool used in the Bioinformatics community for interpreting genetic data. Existing implementations of this algorithm (in the form of programs or web interfaces) are widely available and free. Therefore, the most significant limiting factor in BLAST implementations is not accessibility but computing power. My project deals with possible methods of alleviating this limiting factor by harnessing computer resources which go unused in long periods of idle time. The main methods used are grid computing, dynamic load balancing, and backgrounding.

## Background

There is an immense amount of genetic data generated by government efforts such as the human genome project and by organization efforts such as The Institute for Genomic Research (TIGR). The task of extracting useful information from this data requires such processing power that it overwhelms current computational resources. However, there exist large amounts of unused processing power in schools and labs across the country; most computers are never being used all of the time, and most of the time that computers are being used their processors are nowhere near 100% load. Harnessing some of this unused power is a useful problem not just for the specific application in Bioinformatics of DNA sequence pattern matching, but for many computationally intensive problems which could be solved more accurately and faster with increased resources.

## Procedure

The first step in harnessing unused processor power is to clearly establish and document the existence and magnitude of that unused power. Accomplishing this task requires that we establish some metrics for describing computer load and develop a way to keep a record of those metrics over time. Perl is an ideal language with which to write a program which could perform this task because of its text manipulation capabilities and high speed. The program "cpuload" uses the Linux "uptime" command every second, parses the output, and writes the results to a file which is then plotted using gnuplot. The graph shows the results over one execution of the BLAST algorithm comparing two strains of e-coli bacteria.



Remote machine tests have the following procedure:

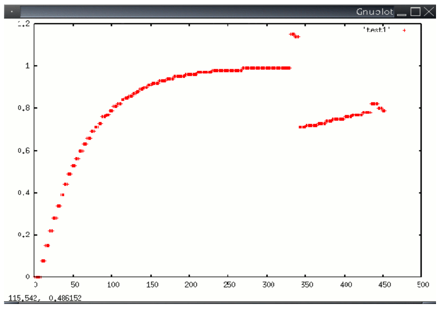
ssh to target processor
Record test number, processor name, and any users
Ask any users to notice performance changes
Run ~/web-docs/techlab/BLAST/formatdb -iEcK12.FA -pT -oT -nK12-Prot
Run ~/techdocs/cpuload for 5 data points
Record start time
Run ~/web-docs/techlab/BLAST/blastall -pblastp -dK12-Prot -iEcSak.FA -ok12vssak -e.001
Record end time
Allow cpuload run for approximately 5 more data points
vim runstats
:w tests/testX
Record any user-reported performance changes

## Results

The use of grid computing to optimize BLAST implementations is not an original idea; a program called mpiblast has already been written and made available to the public. However, implementing mpiblast in any given environment is not a trivial task. For example, our systems lab, although it has mpi installed on several computers, has not maintained a list of which computers are available to run parallel programs. My next task was to compile this list using essentially trial and error and running a test mpi program, mpihello.c. See poster for pictures of the old, obsolete lamhosts list and the updated working version.

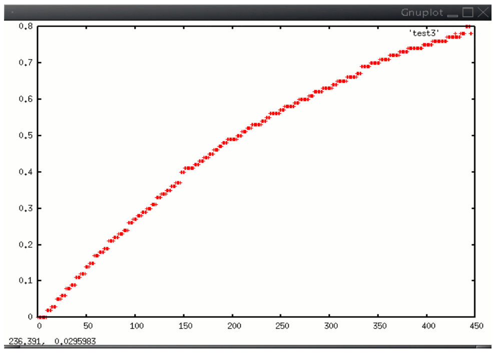
Here are the results for single remote machine tests, including selected graphs of cpuload output:

**Test 1:**
tess
No users
Start: 9:09
End: 9:16



**Test 2:**
beowulf
Jack McKay

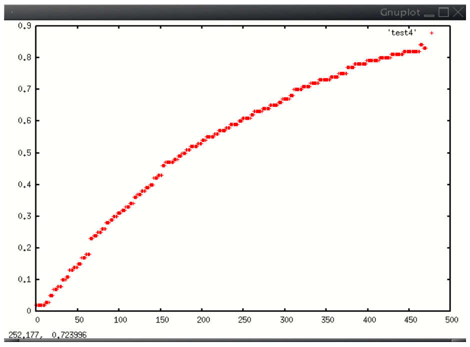
Start: 8:57
End: 9:04
User report: "I experienced no slow down or loss of performance. But if I had a loss of performance that persisted for over thirty six hours, rest assured, I would have contacted my doctor."



oedipus: no route to host

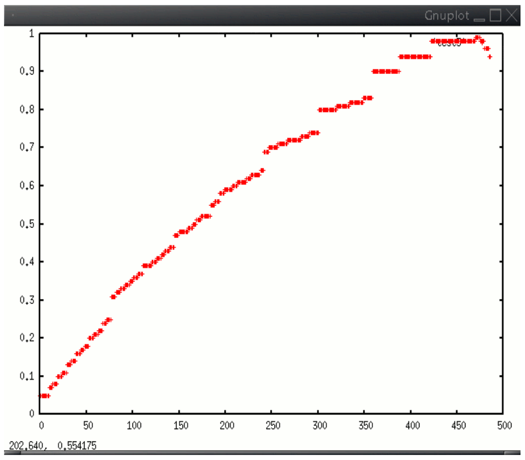
**Test 3:**
antigone
No users
Start: 8:43
End: 8:51

**Test 4:**
agammemnon
Jason Ji
Start: 9:53
End: 10:01
User report: Did you experience any slow down at all? "No".



**Test 5:**
loman
Michael Drukker
Start: 8:44/src/redirect.php

End: 8:51
User report: “I'm not noticing anything, but I'm not doing anything computationally intensive, so...”



**Test 6:**
lordjim
Robert Staubs
Start: 8:57
End: 9:04
User report: “I wasn't really using the computer during that time.”

**Test 7:**
faustus
Caroline Bauer
Start: 9:25
End: 9:34
User report: “I haven't noticed anything, so...”

**Test 8:**
okonokwo
Alex Volkovitski
Start: 10:10
End: 10:19
User report:

**Test 9:**
joad
No users
Start: 9:15
End: 9:23

**Analysis**

Tests I run on single remote machines generate two dependent variables: running time and CPU load over the test's duration. So far nine tests have been run, six with users on the target machine and three without users on the target machine. As is visible from the graphs above, the tests have similar results with similar durations, indicating that performance for grid computing in the systems lab is indeed predictable and repeatable.

Furthermore, the user testimonials so far unanimously agree that no change in performance was noticed.

**Further Testing Plans**

In future tests of multiple machines running simultaneously, I could look at how effectively each test used its resources by creating an "efficiency" metric.  A formula for this metric could perhaps be

$$E = 1/(t*n)$$
$$\text{efficiency} = 1/((\text{running time}) * (\text{\# of machines}))$$

Because of the transfer time involved in MPI programming, one machine will probably be the most efficient.  The interesting question I will address, though, is *how much* more efficient is one machine than two?  Three?  How many machines can you utilize before realizing a huge drop in efficiency?

In general, there is also an optimum balance between transfer time and processing power for any given algorithm to run in the shortest time.  At this point, adding more processors actually *slows down* the program because the increase in transfer time outweighs the added processing power.  The ideal number of processors is generally higher for more complex algorithms; adding two numbers together is clearly fastest when run on only one computer, while BLAST algorithms can benefit from more processors.  It will interesting to see whether or not I can surpass this "optimal number" for BLAST algorithms with the number of processors available in the Systems Lab.

A third dependent variable my tests could possibly generate would be accuracy of output.  If I could develop a method of measuring this variable, it would probably be the most interesting of all to investigate.  For now, however, I will leave it as a possibility while I focus on the other tests.

## GENOME@HOME

GENOME@HOME is a potential application of Grid Computing to the implementation of BLAST algorithms. The idea is to distribute implementations of BLAST on personal or institutional computers and run those implementations during down time or even in the background, while computers are being used. To justify such a program to users, it is necessary to demonstrate that such a program will not interfere with use of the computer or slow down the computer's performance in any noticeable way.

## References

www.ncbi.nlm.nih.gov - The National Center for Biotechnology Information's website, where I obtained several implementations of BLAST.

www.tigr.org - The Institute for Genomic Research's website, which contains helpful background information on genetic algorithms.

www.stanford.edu/group/pandegroup/genome/ - The primary site for GENOME@HOME