Statistical Analysis Of Parallel Code

New Mexico Adventures in
Supercomputing Challenge
Final Report
April 6, 2005

Team 033
Manzano High School

Team Members:
Stephanie McAllister
Matthew Bailey

Teacher:
Mr. Stephen Schum

Project Mentor:
Sue Goudy

Table of Contents

Executive Summary

Our problem is to do statistical analysis of parallel code running on two different supercomputers at Sandia National Laboratories. We will accomplish our objective by writing a C++ program that will analyze the out files and a result that can be used to find out which systems are running better for the equations that we are running. This will be run on a range of file sizes and on two different systems that have two different hardware systems and two different operating systems. Both of these systems run on a Myrinet communication system that will help to keep the experiments even.

The hypothesis hat we have developed is that EP (embarrassingly parallel) will run faster than CG (Conjugant Gradient) or LU decomposition. EP is a system for solving a system of matrices that involves very little communication between nodes. EP can do this because LU is a way that uses random pairs of processing nodes to decompose an N*N matrix into an upper and lower triangle matrix in an attempt to solve a system of liner equations. CG was designed as an iterative method for solving large systems of linear equations and sparse matrices. It was originally designed as a quicker and easier way to do steepest descents. Through every iteration, the equation gets much bigger, so CG was designed to find the constants quickly in each iteration of the problem.

The objective of our statistical analysis is to determine which of the two types of parallel systems will run faster Feynman or C-Plant. The major difference from a hardware perspective is that Feynman has two Alpha processors while C-plant has only one Xeon. When we run our comparisons they will by run on Feynman using both processors while C-plant uses its single processor. The object of this is that since C-plant was designed to use only one processor and Feynman two the only fair comparison would be with both systems running at maximum efficiency. We are comparing the entire system instead of just the Operating System or the processors.

Another difference between C-plant and Feynman is their operating systems. Feynman runs on a full Linux system that has all of the demons that can kick an executable file off of the computing nodes for a short time. C-plant, however, does not have all the demons of the full Linux system it can stop and then restart the executable. There is another demon that is associated with the batch system that could cause a significant impact on our runtimes. The batch system is a scheduling system PBS which stands for portable batch system that find available node after a job is submitted and schedules the executable to be carried out. This makes the start and stops of Feynman more variable than C-Plant. This stop and start can have an effect on computing times and the more variability can cause a longer or shorter break. Since Feynman can bring dynamic libraries and those dynamic libraries are subject to change this is another possibility for variation that we will have to check. C-plant is designed to not use dynamic libraries and is another way that variations could make the single-processor system faster.

Both systems use Myrinet to communicate so they operate on a level playing field as far as the capacity for communication. Myrinet runs at 200 Mb/s and another piece of data to look at is whether or not the data is stressing the communication limitations Of the Myrinet connection. The communication between nodes works on a nearest neighbor system the

nearest neighbor is the next node that has the shortest communication path between them. Since C-Plant is a system that is used very heavily used there are fewer available nodes. These factors need to be taken into account that our results are not coming to us from a system in a vacuum and as a result have variations that could cause unexpected results.

C++ program is used to separate between the four classes and compute mean and standard deviation. The classes refer to the size of the out file the four classes that we are using are S, A, B, C. There is a fifth class D but will not be using it in our project. The largest class that we will be running is C with the smallest being S. The mean that is being referred to is the mean time it takes to compute one out file or all of the individual operations in an entire file. The standard deviation is the difference between the times it takes one file to run on one system as compared to how long it takes on another system.

In order to insure that our results are authentic is to run the experiment a second time under a different set of circumstances and see how a change in the variables influences the end results. Hopefully we will be able to determine which system runs faster under different sets of systems and circumstances.

At this point we have not been able to run our programs on the out files to get any results to either verify or negate our hypothesis. As a result we have been unable to garner any results from analyzing our results.

Acknowledgements:

We would like to acknowledge and thank our teacher, Mr. Schum, our project mentor, Sue Goudy, for all of the help and time that they have put into helping us develop and work on the project. We would also like to thank Sandia National Labs for letting us conduct this project on the clusters C-Plant and Feynman and to the administrators of the machines who gave us permission to perform system analysis and put new code onto them. The specific administrator who helped us the most with support issues who we would like to recognize is Sophia Corwell. Thanks so much to all of you and for all of your support!

Appendix 1: Perl Program

```
#!/usr/bin/perl -w

#Stephanie McAllister "PerlGrabPB.pl"
#last mod. 15feb05

#iterating through each of the input files specified
#on the command line
foreach $input (@ARGV)
{
	print "reading input file ...$input\n";

	#print STDOUT "Please enter the filename to be searched.\n";
	#$file=<STDIN>;

	#print STDOUT "Please enter the name of the file to be written to.\n";
	#$fn=">".<STDIN>;

	$output = $input . ".out";
	print "output file = $output\n";

	open(WRITE, "> $output");
	open(READ, "< $input");

	$verify = 0;

	#read 1 line at a time
	while (<READ>){
		#if pattern is found,
		#write to file and screen
		#store as an array
		if (/CG Benchmark Completed./) {
			while (<READ>){
				last if /Compile options:/ ;
				 if (/Class/) {
					@class = $_;
					@class = split ' ', $_;
				 }
```

```
		elsif (/Size/){
			@size = $_;
			@size = split ' ', $_;

		}

		elsif (/Time in seconds/){
			@time = $_;
			@time = split ' ', $_;

		}
		elsif (/Total processes/){
			@Tprocs = $_;
			@Tprocs = split ' ', $_;

		}
		elsif (/Mop\/s total/){
			@Mops = $_;
			@Mops = split ' ', $_;

		}
		elsif (/Verification/){
			last if /SUCCESSFUL/;
			$verify = 1;
		}#end if verification


		@block = $_;
@block = split ' ', $_;
print "\n\n@block\n";#print whole @ to keep us sane


$verify = 0;

}#end while loop

#when vari. $verify is negated
#the false statement(0) is made true(1)
#and so w/ line, will call up func below
```

```
                        if (!$verify)
                        {
                                #input file "SUCCESSFUL" Verification
                                #print all needed info-class,size,time,Tprocs,Mops
                                print (WRITE " $class[2] $size[2] $time[4]
$Tprocs[3] $Mops[3]\n");
                        }#end print if statement
                        else{
                                 print "UNSUCCESSFUL RUN";#print to screen
only
                        }#end print else statement

                }#end Benchmark Completed if statement

        }#end 1st while (READ)




        close(READ);
        close(WRITE);
}
```