Statistical Analysis Of Parallel Code, Year 2

New Mexico Adventures in

Supercomputing Challenge

Final Report

April 25, 2006

Team #28

Manzano And Eldorado High School

Team Members

Stephanie McAllister

Vincent Moore

Teacher

Stephen Schum

Project Mentor

Sue Goudy

Table Of Contents

Executive summary

This year we have been able to run many of the standard benchmarks used to determine the speed and overall usefulness of a computing platform. In the constantly changing and improving computer world that we live in, finding where the slowdowns are within the platform can improve the performance significantly. In running the benchmarks and post-processing the statistical numbers that we generated, we can come up with where the bottlenecks of the platforms lie. We are mostly looking for communication slowdowns. This was tested by the standard benchmarks. The control benchmark that we used was the Numerical Aerospace Simulation Benchmark code of Embarrassingly Parallel, which has the least communication. We hope that our findings will be able to help improve future platforms and also future benchmarking codes and general programs that need to be run on parallel machines.

This project is a continuation from last year, yet we have made many improvements and have essentially redone everything that was done last year. The major portions of the project that were done previously were research into what the Numerical Aerospace Simulation (NAS) codes did in general computation. We found that Embarrassingly Parallel (EP) has very little communication and each of the processors are trying to come up with the same random number as the other nodes. Conjugate Gradient (CG) does a steepest descent problem using an easier method called conjugate gradient. Lower and Upper Triangle Matrices (LU) decomposes an N*N matrix into a lower and upper triangle matrix. This is an easier way to solve many variables [13].

The Perl script to post-process the outfiles generated from the NAS benchmarks was also written last year. This script goes through the NAS outfile generated from the run and pulls out the values that we are interested in by a brute force method.

Additionally, we completed several runs on Feynman and C-Plant. The runs that were previously made on Feynman are no longer valid because the operating system was upgraded, which also upgraded the libraries and services on the machine. If these results were combined with the results that we have obtained this year, there would be major discrepancies that could not be explained accurately and could not be retested. These results were so small that it did not matter that we could not use them as it takes a larger sample to get accurate numbers. C-plant is no longer a working system, as it was decommissioned over the past summer, so the results do not have any value to the project [13].

Though we did not get much done on the project last year, the skills that were gained made up for the lack of results and data. One such skill was the ability to write Perl scripts. This has greatly benefited the team because the Perl scripts are relatively quick and easy to write now. This has prepared the team for future projects, work in college, and improved the ability of all to dissect code and understand what it is doing. The other team member took a Java class and was able to benefit the team by writing the computational code and writing the outfile for graphing with Microsoft Excel.

This year:

Hypothesis

At the start of this year, the team sat down with our mentor and discussed the background information that had been figured out last year. From this information and with a little clarification from Sue, we came up with a hypothesis: We believe that EP will run the most efficiently because it has the least amount of communication and seems to be the most stable code on any given platform because of the communication patterns that it would display [15].

Scope

The scope of the project originally included many things that we have not been able to accomplish because they were too broad for this program. There are groups that have spent years doing the same studies and calculations on parallel platforms that we are doing, but they have more funding to do so and have been able to broaden the scope of their work [14]. The main points that we have been looking at, and continue to analyze, are the interconnect times and how the architecture and type or brand effects the overall performance of the parallel system and the codes that are being run on it. Part of the established way to analyze the overall efficiency of the interconnects on a particular system is to look at the latency [5]. Latency is defined as the total time from the initial sending to the finish of receiving a message across the interconnect, or the amount of time it takes to send a zero length message. This means that there is overhead for every interconnect and latency is the way to measure how much overhead there is. This depends on the switches that are used and time that it takes to construct the header on the packet (the message), transfer the packet down through the operating system layers to the network chip, onto the network, and over to the node that it is being sent to. The node then has to unpack the packet that was sent. This series is independent of the code running. So, latency is basically the time between when a message starts and when another message can start.

The scope of the project also included statistical analysis, through computing the mean and standard deviation, of the message passing interface. This was done to insure that the data sets that were collected were accurate samplings. The mean is the average of how long the run takes or the average of how many Mega-Operations per second (MOps). The value for mean is a true but unknown value for performance. Standard Deviation for a sample shows how much variation is in the sample compared to the mean. The reason that we compute this is because it is useful in other calculations to tell if the sample mean is really representative of the true mean. This shows us if the sample mean is accurate enough or not by the size of the standard deviation. If the standard deviation is large, the sample mean is not a good representation of the true mean; if the standard deviation is small, the representation is a much more accurate model of the true mean.

The original scope of the project included possible slowdowns and their causes. Though we acknowledge that these are inevitably part of the project, we did not allot for them in the calculations because they tend to be very irregular when they show up. This has others very puzzled as well. To avoid looking up if there were specific hardware problems or software interrupts during a run, the standard deviation is calculated to ensure that the possible error from run to run is within a certain range. If the runs do not vary by much, then a much smaller sample is needed for accurate calculations, giving the best results and being more predictable. Additionally, there tends to be a difference between 32- and 64-bit processors and the performance that is gained with either. Part of the investigation that was planned was to look at how much of a difference this caused and if 64-bit processors are more efficient for high performance computing codes, even if they are written for a 32-bit operating system and chipset. We have not been able to investigate the differences between 32-bit and 64-bit chipsets.

HPC

The High Performance Computing Challenge publishes a set of benchmarks (HPC), used for determining a list of the top 500 supercomputers in the world [12]. These benchmarks are a package of many tests wrapped together. The package that we used also had multiple configurations to use. These included the MPI package that we used, Version 1.0.0. Out of the many tests that HPC runs, we picked High Performance Linpack (HPL), Parallel Matrix Transpose (PTRANS), Giga Updates per second (GUPs), Fast Fourier Transform (FFT) [1]. HPL measures the floating point rate of execution for solving a linear system of equations. PTRANS is a test of the total communications capacity of the network because it has communication of pairs of processors at the same time. GUPs is just an output of how many updates there are per second. FFT computes a complex Discrete Fourier Transform and measures the floating point rate of execution of double precision [1].

NAS

The NAS benchmarks do not come packaged as one like the HPC codes do. Instead, each one must be built on the platform separately. The configuration that we used was MPI only, instead of the serial communication. The version that we used was 3.1. The tests we ran are EP, CG, and LU. EP distributes the program to each node [3]. They are all trying to come up with the same random number, which is generated with a “seed” a character that starts the random number generator. Each node receives a different seed, and the communication between nodes only happens when they have all come up with a number. These are collective MPI communication calls. This means that all of the nodes report through MPI to a single node. This differs from CG because though the messages are all passed to the same node, the root node does not distribute a message to the other nodes, they just report to the root node. CG is an iterative method to do a steepest descent problem. We originally thought that it was a random communication pattern, but upon dissection of the code discovered that it does collective communication. We decided to pick through the code because after we graphed it, we did not understand why the data and results did not look like LU, so we went back and found out. Though this test did not do the communication pattern that we had thought, it is still useful in our project because we are studying the communication times and types used by parallel computing platforms. CG is still useful because the communication that it does is known as “all-reduce” which is a very different communication pattern from those used in LU.

Supercomputers

The majority of the high performance computers that were used in our project reside at Sandia National Laboratory. All of the computer systems are parallel platforms, meaning that there are multiple processors and RAM (Random Access Memory) that are linked together to do a mathematical computation.

These computations can be a simulation, visual or purely computational, though visualizations generally need a separate area and program for viewing. The most basic level at which parallel computing can be explained is that there are many nodes (nodes consist of 1-2 processors, a motherboard, RAM, and sometimes a hard drive) which are all connected with an interconnect, and they are organized in such a way that they can all communicate with one another and solve part of a problem or simulation. The "mother nodes" write to the outfile and compile all of the data from the nodes collectively as well as distribute the code to all the nodes prior to processing. The main part of the supercomputers that we are looking at is the interconnect and the speed at which they can do computations. Five of the six machines that we are using come from Sandia, the other comes from Los Alamos National Laboratory. At Sandia, we are running on Feynman, Red Squall, Spirit, Red Storm and Thunderbird.

Feynman has Intel Xeon Processors, which run at 2GHz (Giga-Hertz); Myrinet interconnect, which is in a star topography and has a band width of 350Mbs (Mega-byte per second) [6]. A star topography is where every node is connected to a router, switch or hub. This means that all network traffic must go through one of the routers/switches/hubs in order to get passed to another node. The weakest spot of this network is the router/switch/hub because it gets bogged down with communications before the bandwidth of the cable can be reached from message passing. There has also been an Operating System (OS) upgrade since last year. This OS upgrade included a new set of compilers which can also have an impact on the efficiency of the code running on the platform.

The new OS is Red Hat Enterprise Linux 3, and the full version is on the I/O nodes, which control the computational nodes and are also known as “mother nodes”. The runs (iterations through running the benchmarking codes) that we have done so far were to re-run the tests from last year, seeing as the machine is different and it would not be a good idea to combine data sets from different machines. This also helps us to get an approximate figure of how many runs we would need to do to get good results. According to statisticians [5], we need at least 30 runs. As the runs increase, our odds of getting an accurate number increase.

Red Squall is another platform from Sandia that we are running on. It has 258 nodes, two-processors per node, 12 Terabytes (TB) of storage, and 792 TB of RAM. The processors range from 2.0 to 2.2 GHz and range from 2 GB to 4GB of RAM on the compute nodes [7]. It runs on a 256 port Quadrix Elan4 high speed interconnect [7]. The OS is Linux based, though we have not been able to find out which version it runs.

Spirit runs Intel Xeon 3.4 GHz dual processors on the compute nodes. The OS is Red Hat Enterprise Linux WS, release 3 [8].

Red Storm is a Cray XT3 system [9]. It is the 6th fastest supercomputer in the world, according to the HPC Challenge's 26th Annual Top 500 list at the Supercomputing Conference (SC|05) in November of 2005 [12]. It runs on 64-bit AMD Opteron Processors, clocked at 2.0GHz, and has 140 TB of disk space. The OS that the compute nodes run on is called Catamount [9]. It is a lightweight kernel based on ASCI Red, instead of being a full version of Linux. The lightweight kernel eliminates some of the interrupts that tend to cause slowdowns with the computations. The I/O nodes run with a full version of SuSE Linux (8.2.99, a server version). The I/O nodes need to have full Linux installed so that programs can be compiled and run. The interconnect is a Cray proprietary interconnect called SeaStar [9]. It has a very low latency.

Thunderbird is ranked 5th on the HPC Challenge's Top 500 list of 2005 [12]. Many of the student interns at Sandia were able to help and contribute to the effort to set it up over the past summer. One member of our team was also able to be a part of the team and did a little bit more than other student interns could. The interns helped with the cabling, Break/Fix, hardware replacement/troubleshooting, BIOS configuration, network configuration/troubleshooting, and system configuration/troubleshooting. Having been a part of the team who built Thunderbird, our team has been able to use the platform for this project. The hardware configuration includes Infiniband (IB) interconnects, dual processors on every node, and over 5,120 compute nodes [10].

Mauve is located at Los Alamos National Laboratory [11]. It is comprised of Intel Itanium 2, 1.3 GHz processors. It has 1TB of globally addressable RAM, and runs the Altix OS, which is based on Red Hat Linux. It also has 50 TB of scratch space and uses the Load Sharing Facility (LSF) batch system [11]. This machine differs from other because of having to load the modules by hand. We have built the HPC benchmarks on this platform. Part of doing so was loading modules by hand, as they are not previously loaded. We had to do some research into what modules we needed to be able to run the HPC code.

NAS Perl

The NAS Perl script picks through the outfiles from the NAS benchmarking codes and pulls out the values that we are interested in, creating a new file with the values inside it. This was done by finding certain character strings, finding the lines with the values that we are interested in, and putting the values into a variable [2]. The outfiles all built upon each other, so there had to be multiple iterations through the script, done by a *while* loop. The starting and stopping values were found with an *if* loop followed by a *while* loop and a last *if* loop. The *while* loop just tells the program to continue with the inside of the *while* loop. The insides consist of *if* and *elsif* loops. The loops basically say that if it finds a certain character string, put it into a variable. The variables are stored and put into a file outside of the while loop, but still inside the first if loop (See Appendix A). The file written to was named by the script and the convention was appending an ".out" to the original file name, which was specified on the command line before running the script. This is known as a "brute force" way to pull out values because it is specific to the NAS benchmark's outfile. [2]

HPC Perl

The HPC Perl script does much of the same thing that the NAS script does, but in a different and more efficient way. The file being put in to read through is still specified on the command line and the naming convention is the same. The main difference is that instead of pulling out certain values, the keywords are linked to the values by a hash table [2]. The values are put into the table by a regular expression within the last if loop under the while and if loops (See Appendix B). The main benefit to this method is that if we chose to modify what values we wanted, it was easy to just tell it to output them to the file by the keyword. In this case, the keywords are the values in the outfile that describe what we were looking at. For instance, the Fast-Fourier Transform (FFT) data that we were interested in is MPIFFT_time, which is the amount of time that the computation takes. In the hash table, all that we would had to do to specify printing this value out was to tell it to print MPIFFT_time. This method made it very simple to add and change what values go into the outfile. The way that we chose to output the data into the outfile was to put it first into arrays specifying where all of the data goes, and then go outside the first if loop, as above, to print out the values of each test to the outfile. This was the most efficient way that we could think of to do this.

Runs

Our mentor has helped us greatly by building the benchmarks on most of the systems, since we do not really have the knowledge to build them yet. We have been able to build the NAS parallel benchmarks on many platforms. These include Red Squall, Feynman, Spirit, and Thunderbird. From all four we have gotten runs, though they vary in number. HPC has also been built on many platforms. Feynman and Spirit were built early, as was Red Squall. The other systems came slightly later, including Thunderbird and Red Storm. We built the HPC code by ourselves on Mauve, and have been able to get several runs and results. We have gotten several runs on each platform and intend to get even more as we continue this project (See Sample outfile, Appendix C and D).

Java

The Java program is written and running. One of the main reasons that we chose to do the computations in Java was to utilize the entire team's abilities, as one member had previous knowledge from a programming class. Java is also more efficient for calculations. We found out after the Java program was written that Perl has the same abilities that Java does for calculations.

The java code is designed to take the reduced values from the out files produced by the Perl scripts and puts the values for each run and inserts them into a custom designed object. This object holds the size of the program run, the number of processors used, the time it took, and the number of operations done (MOps). This object has two more stored values only used when the data used a HPC benchmark run. The main java program also keeps track of whether the run is HPC or NAS and from that decides the maximum number of processors it will provide room for in the array it will output. After taking each line of the output file and inserting relevant data into the ArrayList of benchmarking objects, the main program searches the ArrayList for benchmarks fitting each criteria based on processors used and class of program.

(class/processors)	(class/processors)
NAS	HPC
s/1,s/2,s/4,s/8,s/16 etc.	ptrans/1,ptrans/4,ptrans/9 etc.
a/1,a/2,a/4.a/8 etc.	HPL/1,HPL/4,HPL/9 etc.
b/1,b/2,b/4 etc.	GUPS/1,GUPS/4 etc.
c/1,c/2 etc.	FFT/1 etc.
d/1 etc.	

Then the java program calculates the mean or average value and provides an output value for each class/processor set's time and MOps. It also calculates and outputs the standard deviation for each class/processor set's time and MOps (For full Java code, see Appendix F - G). The files that are imported into excel are “.csv”’s and are just comma separated values (See Appendix H).

Calculations

The calculations that we are doing is to compute the mean:

$$\overline{X}=\frac{1}{n}\sum_{i=1}^{n} X_i$$

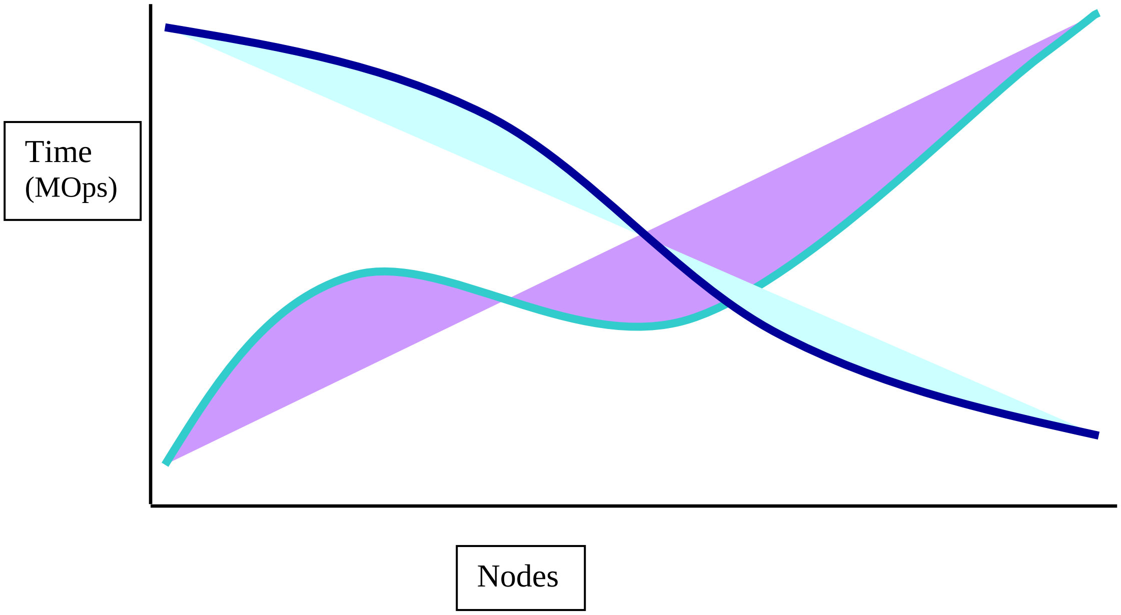
This equation basically calculates the mean through a summation[5]. The standard deviation is computed by taking the square root of the variance:

$$s_{N-1}=\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i-\overline{x})^2}\,.$$ [5]

We know that these are the correct formulas through our communication with a statistician[4] who does this for a living.

Graphs

There were many ways that we thought of to graph the results of this project. Our original conceptual graph, did not quite work as it was intended to. We were not able to figure out how to tell Excel to shade in parts of the lines to indicate the standard deviation. Instead, we decided to put the data into bar graph format with the standard deviation plotted as a line through the bar graphs. This seemed to work well to demonstrate the point (see Appendix I).



Results

We are still working on analyzing the graphs and data that we received from the runs that we have done statistical analysis on. Most of the professionals that do this for a living end up looking at graphs and data, wondering what is really going on. This is not an exact science, so the results are always shaky at best. For preliminary results, please see Appendix I.

Future Work

We intend to continue our work for as long as we can. We will be working on getting better results and conclusions for the Adventures in Supercomputing Challenge's Expo at Los Alamos National Laboratory, though if it is not totally satisfactory, we will still continue to work. We also intend to improve our graphing method. There are small issues that we need to fix in Excel. We also want to be able to graph multiple machines on the same axis to be able to compare certain tests and other information through Kiviat diagrams. Kiviat diagrams are more commonly known as star plots. We are also actively working on collecting more runs of HPC. The Java program works for HPC numbers, but we are in the process of writing a class that can handle HPC data better. This work will continue at least through summer, if not longer.

Thanks

We would like to thank our mentor, Sue Goudy, for all of the help that she has given us in our project and for keeping us accountable to deadlines. Also deserving of thanks is Mr. Schum for sponsoring a class during the day for credit to participate in the Supercomputing Challenge and to Mr. Whalen for supporting all of the students from Manzano High School to be in this program. Many thanks go to the Thunderbird team for allowing students to participate in the building of the machine, allowing it to seem more personal and allowing students to use their resource. Also the help that they have given us in compiling the benchmarking code and to Sophia Corwell for Perl scripting help. Thanks to Sandia National Laboratory for the use of the computing resources, Los Alamos National Laboratory for sponsoring the program and making computing resources available to the students. As always, we were very appreciative of the entire team of consult@challenge.nm.org. Thanks to all of you! Special thanks on the consult team go to David Kratzer for going out of his way to help us get an account on Mauve and for the guidance that he gave us throughout the year. Tom Laub for organizing the Sandia Tour and helping it get started, as well as his help to many teams in the challenge. A special thanks to Celia Einhorn for all of the support that she continually gives to the students in the challenge.

References

1. HPC Website: http://icl.cs.utk.edu/hpcc/index.html

2. O'Reilly Perl CD Bookshelf

3. NAS Website: http://www.nas.nasa.gov/Software/NPB/Specs/npb2_0/npb2_0.html

4. Rob Easterling

5. Lilja, David J., Measuring Computer Performance: A Practitioner's Guide, Cambridge University Press, 2000.

6. Feynman

7. Red Squall: Frequently Asked Questions list maintained by the system administrator.

8. Spirit: Information from the /proc/cpuinfo available on all linux machines.

9. Red storm: John Noe, Sandia National Laboratories, Tour of Red Storm 20March2006.

10. Thunderbird: https://computing.sandia.gov/platforms/thunderbird/

11. Mauve: http://computing.lanl.gov

12. HPC Challenge 26th Top500 list: http://top500.org/lists/2005/11/basic

13. Last Year's Final Report: http://challenge.nm.org/archive/04-05/finalreports/33.pdf

14. Performance Modeling: http://www.sandia.gov/perfmod/

15. Weekly Meeting Log

Appendices

Appendix A

```
#!/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);

}
```

Appendix B

```
#!/usr/local/bin/perl

#Monty - diff...


# !/usr/bin/env perl

#manzano specific

# !/usr/bin/perl -w

#Sandia specific (undo space to select)


# Perl script to get HPC data out through hash...


# Stephanie McAllister "PerlHPChash"

# last mod 23Jan06

# ch. 5-1, 5-2 of Learning Perl


# read in file...  HPC specific...

foreach $input (@ARGV)

{

        print "reading input file ... $input\n";


        $output = $input . ".out";

        print "output file = $output\n";
```

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

    #init hash table - outside loop!
    %H=();

    while (<READ>){
        if (/Begin of Summary section./){
            while(<READ>){
                last if /End of Summary section./ ;
            #put everything into a hash table here...

            #read in values/keywords
            #split by line

            #use regexpr to get values fr each side of =
            m/(\w+)\s*=\s*(.*)\s*$/;#find out exactly what I say
#for future reference: to avoid picking up whitespace with above line syntax,
#just have to remove the \s... The (.*)means anything on either side of =.

            $H{$1}=$2;
            #put into hash table
```

```
                }#end while read 2
        }#end start/end param. if statement

        #new hash value MPIFFT_time (++ all time records for MPIFFT)
        #declare arrays to print,
        @hplData =
                ("HPL", "$H{HPL_N}", "$H{HPL_time}",
"$H{CommWorldProcs}",

"$H{HPL_Tflops}");
        @ptransData =
                ("PTRANS", "$H{PTRANS_n}", "$H{PTRANS_time}",

"$H{CommWorldProcs}", "$H{PTRANS_GBs}");
        @gupsData =
                ("GUPs", "$H{MPIRandomAccess_N}",

"$H{MPIRandomAccess_time}","$H{CommWorldProcs}",
"$H{MPIRandomAccess_GUPs}");
        @fftData =
            ("FFT", "$H{MPIFFT_N}", "$H{MPIFFT_time}",
"$H{CommWorldProcs}",

"$H{MPIFFT_Gflops}");

        if ($H{Success}==1)
        {
            #do calc for MPIFFT_time
            $H{MPIFFT_time}=$H{MPIFFT_time0}+$H{MPIFFT_time1}+
$H{MPIFFT_time2}+$H{MPIFFT_time3}+$H{MPIFFT_time4}+$H{MPIFFT_time5}
+$H{MPIFFT_time6};
            print "@hplData\n@ptransData\n@gupsData\n@fftData\n";
        print (WRITE
"@hplData\n@ptransData\n@gupsData\n@fftData\n");
        }#end if statement
        else{
            print "UNSUCCESSFUL RUN";#print only to screen!!
        }#end else statement

    }#end while statement 1

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

```
}#end of foreach beginning statement...
```

Appendix C

```
NAS Parallel Benchmarks 3.1 -- CG Benchmark


 Size:      14000
 Iterations:    15
 Number of active processes:    64
 Number of nonzeroes per row:       11
 Eigenvalue shift: .200E+02
Contacting node allocation daemon...
64 nodes are allocated to your job ID 3424.

Awaiting synchronization of compute nodes before beginning user code.
Application processes begin user code.

>>> Exit code 1 on node 253 <<<
>>> further messages suppressed

We're sending a SIGTERM to your application.
Awaiting compute node completion messages.
If no response within 60 seconds, try interrupting yod with control-C.

 Name              Rank  Node   SPID   Elapsed  Exit  Signal
```

```
------------------- ----  ----  -----  --------  ----  ------
 node.n-21.t-32        0  1036   1866  04:01:05  killed by SIGTERM request
 node.n-22.t-32        1  1037   1507  04:01:05  killed by SIGTERM request
 node.n-23.t-32        2  1038   1507  04:01:05  killed by SIGTERM request
 node.n-24.t-32        3  1039   1382  04:01:05  killed by SIGTERM request
 node.n-17.t-32        4  1032   1528  04:01:05  killed by SIGTERM request
 node.n-10.t-36        5   929   1954  04:01:05  killed by SIGTERM request
 node.n-11.t-36        6   930   1924  04:01:05  killed by SIGTERM request
 node.n-12.t-36        7   931   1954  04:01:05  killed by SIGTERM request
 node.n-9.t-16         8   672   3129  04:01:05  killed by SIGTERM request
 node.n-10.t-16        9   673   2890  04:01:04  killed by SIGTERM request
 node.n-29.y-10       10   244    619  04:01:04  killed by SIGTERM request
 node.n-30.y-10       11   245    619  04:01:05  killed by SIGTERM request
 node.n-31.y-10       12   246    619  04:01:05  killed by SIGTERM request
 node.n-32.y-10       13   247    619  04:01:05  killed by SIGTERM request
 node.n-29.t-45       14   500    623  04:01:05  killed by SIGTERM request
 node.n-30.t-45       15   501    623  04:01:05  killed by SIGTERM request
 node.n-31.t-45       16   502    623  04:01:05  killed by SIGTERM request
 node.n-32.t-45       17   503    623  04:01:05  killed by SIGTERM request
 node.n-1.t-47        18   504    621  04:01:05  killed by SIGTERM request
 node.n-2.t-47        19   505    621  04:01:05  killed by SIGTERM request
 node.n-3.t-47        20   506    621  04:01:05  killed by SIGTERM request
 node.n-4.t-47        21   507    621  04:01:05  killed by SIGTERM request
 node.n-1.y-11       22   248    619  04:01:04  killed by SIGTERM request
 node.n-2.y-11       23   249    619  04:01:04  killed by SIGTERM request
 node.n-3.y-11       24   250    619  04:01:05  killed by SIGTERM request
 node.n-4.y-11       25   251    619  04:01:05  killed by SIGTERM request
 node.n-5.y-11       26   252    619  04:01:05  killed by SIGTERM request
 node.n-6.y-11       27   253    619  04:01:04  killed by SIGTERM request
 node.n-7.y-11       28   254    619  04:01:05  killed by SIGTERM request
 node.n-8.y-11       29   255    619  04:01:04  killed by SIGTERM request
 node.n-31.y-8       30   182   1109  04:01:04  killed by SIGTERM request
 node.n-25.t-25      31   880    679  04:01:05  killed by SIGTERM request
 node.n-26.t-25      32   881    679  04:01:05  killed by SIGTERM request
 node.n-27.t-25      33   882    679  04:01:05  killed by SIGTERM request
 node.n-28.t-25      34   883    679  04:01:05  killed by SIGTERM request
 node.n-25.t-35      35   624    677  04:01:05  killed by SIGTERM request
 node.n-26.t-35      36   625    677  04:01:05  killed by SIGTERM request
 node.n-27.t-35      37   626    677  04:01:05  killed by SIGTERM request
 node.n-28.t-35      38   627    677  04:01:04  killed by SIGTERM request
 node.n-2.t-33      39   633   1547  04:01:05  killed by SIGTERM request
 node.n-3.t-33      40   634   1538  04:01:05  killed by SIGTERM request
 node.n-4.t-33      41   635   1538  04:01:04  killed by SIGTERM request
 node.n-5.t-33      42   636   1524  04:01:04  killed by SIGTERM request
 node.n-6.t-33      43   637   1963  04:01:04  killed by SIGTERM request
 node.n-7.t-35      44   606   1502  04:01:05  killed by SIGTERM request
 node.n-8.t-35       45   607   1502  04:01:04  killed by SIGTERM request

 node.n-4.t-40       46   283    784  04:01:04  killed by SIGTERM request

 node.n-30.y-11       47   277    780  04:01:04  killed by SIGTERM request

 node.n-29.t-40      48   308   1235  04:01:05  killed by SIGTERM request

 node.n-30.t-40      49   309   1235  04:01:04  killed by SIGTERM request

 node.n-31.t-40      50   310   1235  04:01:05  killed by SIGTERM request

 node.n-32.t-40      51   311   1235  04:01:05  killed by SIGTERM request

 node.n-22.t-37      52   557   1092  04:01:05  killed by SIGTERM request

 node.n-23.t-37      53   558   1092  04:01:05  killed by SIGTERM request

 node.n-24.t-37      54   559   1092  04:01:05  killed by SIGTERM request

 node.n-21.t-17      55   812   1094  04:01:04  killed by SIGTERM request

 node.n-22.t-17      56   813   1094  04:01:05  killed by SIGTERM request

 node.n-23.t-17      57   814   1094  04:01:04  killed by SIGTERM request

 node.n-24.t-17      58   815   1094  04:01:04  killed by SIGTERM request

 node.n-21.t-30      59   780   1094  04:01:05  killed by SIGTERM request

 node.n-20.t-47      60   523    944  04:01:04  killed by SIGTERM request

 node.n-17.t-30      61   776   1086  04:01:04  killed by SIGTERM request

 node.n-18.t-30      62   777   1086  04:01:05  killed by SIGTERM request

 node.n-18.t-17      63   809   1822  04:01:04  killed by SIGTERM request
```

```
NAS Parallel Benchmarks 3.1 -- CG Benchmark
```

```
Size:     75000

Iterations:    75

Number of active processes:    64

Number of nonzeroes per row:       13

Eigenvalue shift: .600E+02


 iteration          ||r||                 zeta

      1       0.13257071746643E-12    59.9994751578754

      2       0.54021441387552E-15    21.7627846142538

      3       0.57508155930725E-15    22.2876617043225

      4       0.58907101679580E-15    22.5230738188352

      5       0.59342235842271E-15    22.6275390653890

      6       0.59736634325665E-15    22.6740259189537

      7       0.60192883908490E-15    22.6949056826254

      8       0.59984965235397E-15    22.7044023166871

      9       0.60134110898017E-15    22.7087834345616

     10        0.59805179779153E-15    22.7108351397172

     11        0.60025777990273E-15    22.7118107121337

     12        0.59913684339943E-15    22.7122816240972

     13        0.59723882884624E-15    22.7125122663245

     14        0.59499172777344E-15    22.7126268007597

     15        0.59980600694896E-15    22.7126844161817

     16        0.59727336571335E-15    22.7127137461758
```

17	0.59706054162841E-15	22.7127288401997
18	0.59345450712132E-15	22.7127366848298
19	0.59923959756901E-15	22.7127407981219
20	0.59496553997991E-15	22.7127429721364
21	0.59446318554205E-15	22.7127441294028
22	0.58988939248615E-15	22.7127447493899
23	0.59217087079768E-15	22.7127450834528
24	0.58787300236184E-15	22.7127452643879
25	0.58903467315753E-15	22.7127453628459
26	0.59200179652152E-15	22.7127454166513
27	0.59215841938171E-15	22.7127454461693
28	0.58711115670319E-15	22.7127454624205
29	0.59053141180686E-15	22.7127454713970
30	0.58794595971476E-15	22.7127454763705
31	0.58532270282392E-15	22.7127454791336
32	0.58837215167381E-15	22.7127454806729
33	0.58948683125210E-15	22.7127454815324
34	0.58760605201307E-15	22.7127454820135
35	0.58386041680778E-15	22.7127454822834
36	0.58482665918172E-15	22.7127454824351
37	0.58581294813970E-15	22.7127454825206
38	0.58237872387978E-15	22.7127454825687
39	0.58486977936419E-15	22.7127454825960

40	0.58080883423072E-15	22.7127454826114
41	0.58250016557550E-15	22.7127454826202
42	0.58195491440239E-15	22.7127454826250
43	0.57864286207501E-15	22.7127454826279
44	0.57910875164293E-15	22.7127454826295
45	0.58505851554448E-15	22.7127454826304
46	0.57918994944012E-15	22.7127454826310
47	0.58296854721644E-15	22.7127454826312
48	0.57528473820291E-15	22.7127454826315
49	0.57588570941779E-15	22.7127454826316
50	0.57693091994436E-15	22.7127454826317
51	0.57276400822640E-15	22.7127454826316
52	0.57703284240136E-15	22.7127454826318
53	0.57503199475455E-15	22.7127454826316
54	0.57605692609382E-15	22.7127454826317
55	0.57539360462044E-15	22.7127454826317
56	0.57685337146103E-15	22.7127454826317
57	0.57678312016558E-15	22.7127454826318
58	0.57295403745889E-15	22.7127454826317
59	0.56607975123872E-15	22.7127454826317
60	0.56355991533483E-15	22.7127454826318
61	0.56893759583559E-15	22.7127454826318
62	0.57020409279203E-15	22.7127454826318

```
    63      0.56325761140624E-15    22.7127454826317

    64      0.56820850502029E-15    22.7127454826318

    65      0.56776003788522E-15    22.7127454826317

    66      0.56899540556826E-15    22.7127454826317

    67      0.56801610048611E-15    22.7127454826317

    68      0.56636836583971E-15    22.7127454826317

    69      0.56592767005805E-15    22.7127454826317

    70      0.55954678746835E-15    22.7127454826317

    71      0.56651782063012E-15    22.7127454826317

    72      0.56570323464977E-15    22.7127454826316

    73      0.56447181225459E-15    22.7127454826317

    74      0.56170922419328E-15    22.7127454826317

    75      0.56106180093612E-15    22.7127454826318

Benchmark completed

VERIFICATION SUCCESSFUL

Zeta is      0.227127454826E+02

Error is     0.347251034577E-13



CG Benchmark Completed.

Class         =                   B

Size          =               75000

Iterations    =                  75
```

```
Time in seconds =                 145.25

Total processes =                    64

Compiled procs  =                    64

Mop/s total     =                376.64

Mop/s/process   =                  5.89

Operation type  =         floating point

Verification    =             SUCCESSFUL

Version         =                   3.1

Compile date    =           10 Nov 2004


Compile options:

   MPIF77       = $(CPtop)/bin/f77

   FLINK        = $(CPtop)/bin/f77

   FMPI_LIB     = -lfmpi -lmpi

   FMPI_INC     = (none)

   FFLAGS       = (none)

   FLINKFLAGS   = (none)

   RAND         = randi8



Please send the results of this run to:


NPB Development Team
```

Internet: npb@nas.nasa.gov

If email is not available, send this to:

MS T27A-1
NASA Ames Research Center
Moffett Field, CA 94035-1000

Fax: 650-604-3957

Appendix D

```
Begin of Summary section.
VersionMajor=1
VersionMinor=0
VersionMicro=0
VersionRelease=b
LANG=C
Success=1
sizeof_char=1
sizeof_short=2
sizeof_int=4
sizeof_long=8
sizeof_void_ptr=8
sizeof_size_t=8
sizeof_float=4
sizeof_double=8
sizeof_s64Int=8
sizeof_u64Int=8
CommWorldProcs=1
MPI_Wtick=1.000000e-07
HPL_Tflops=0.00256814
HPL_time=16.6232
```

```
HPL_eps=1.11022e-16
HPL_RnormI=1.18972e-11
HPL_Anorm1=1036.81
HPL_AnormI=1030.27
HPL_Xnorm1=4331.44
HPL_XnormI=4.79451
HPL_N=4000
HPL_NB=80
HPL_nprow=1
HPL_npcol=1
HPL_depth=1
HPL_nbdiv=2
HPL_nbmin=4
HPL_cpfact=R
HPL_crfact=C
HPL_ctop=1
HPL_order=R
HPL_dMACH_EPS=1.110223e-16
HPL_dMACH_SFMIN=2.225074e-308
HPL_dMACH_BASE=2.000000e+00
HPL_dMACH_PREC=2.220446e-16
HPL_dMACH_MLEN=5.300000e+01
HPL_dMACH_RND=1.000000e+00
```

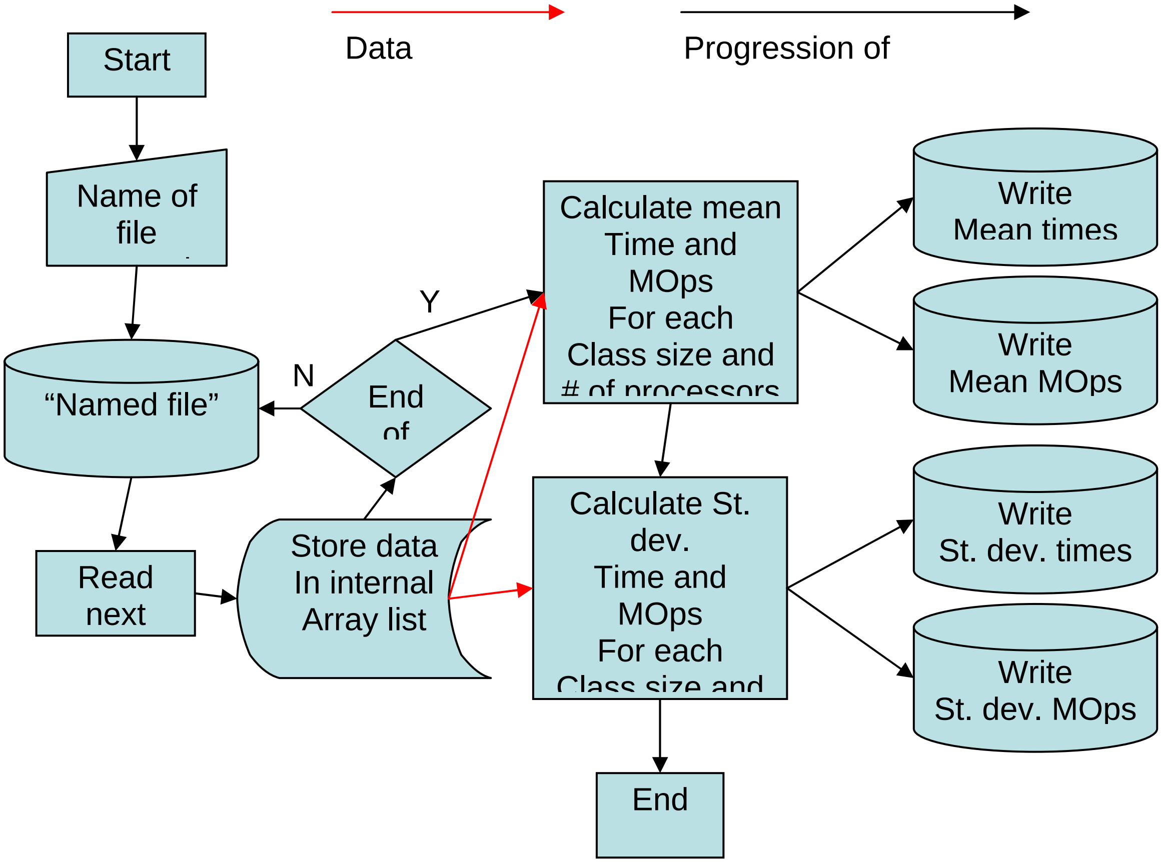
```
HPL_dMACH_EMIN=-1.021000e+03
HPL_dMACH_RMIN=2.225074e-308
HPL_dMACH_EMAX=1.024000e+03
HPL_dMACH_RMAX=1.797693e+308
HPL_sMACH_EPS=5.960464e-08
HPL_sMACH_SFMIN=1.175494e-38
HPL_sMACH_BASE=2.000000e+00
HPL_sMACH_PREC=1.192093e-07
HPL_sMACH_MLEN=2.400000e+01
HPL_sMACH_RND=1.000000e+00
HPL_sMACH_EMIN=-1.250000e+02
HPL_sMACH_RMIN=1.175494e-38
HPL_sMACH_EMAX=1.280000e+02
HPL_sMACH_RMAX=3.402823e+38
dweps=1.110223e-16
sweps=5.960464e-08
HPLMaxProcs=1
HPLMinProcs=1
DGEMM_N=2000
StarDGEMM_Gflops=3.8561
SingleDGEMM_Gflops=3.88734
PTRANS_GBs=0.353082
PTRANS_time=0.0906304
```

```
PTRANS_residual=0
PTRANS_n=2000
PTRANS_nb=80
PTRANS_nprow=1
PTRANS_npcol=1
MPIRandomAccess_N=8388608
MPIRandomAccess_time=10.3096
MPIRandomAccess_CheckTime=2.16439
MPIRandomAccess_Errors=0
MPIRandomAccess_ErrorsFraction=0
MPIRandomAccess_ExeUpdates=33554432
MPIRandomAccess_GUPs=0.00325468
MPIRandomAccess_TimeBound=60
RandomAccess_N=8388608
StarRandomAccess_GUPs=0.0174565
SingleRandomAccess_GUPs=0.0174559
STREAM_VectorSize=5333333
STREAM_Threads=1
StarSTREAM_Copy=4.14207
StarSTREAM_Scale=2.44942
StarSTREAM_Add=2.67901
StarSTREAM_Triad=2.28045
SingleSTREAM_Copy=4.14207
```

```
SingleSTREAM_Scale=2.44942
SingleSTREAM_Add=2.67918
SingleSTREAM_Triad=2.28045
FFT_N=2097152
StarFFT_Gflops=0.541913
SingleFFT_Gflops=0.542553
MPIFFT_N=1048576
MPIFFT_Gflops=0.348073
MPIFFT_maxErr=1.47582e-15
MaxPingPongLatency_usec=0
RandomlyOrderedRingLatency_usec=1.33732
MinPingPongBandwidth_GBytes=1e+90
NaturallyOrderedRingBandwidth_GBytes=1.09809
RandomlyOrderedRingBandwidth_GBytes=1.09885
MinPingPongLatency_usec=1e+105
AvgPingPongLatency_usec=0
MaxPingPongBandwidth_GBytes=0
AvgPingPongBandwidth_GBytes=0
NaturallyOrderedRingLatency_usec=1.32
FFTEnblk=16
FFTEnp=8
FFTEl2size=1048576
M_OPENMP=-1
```

```
omp_get_num_threads=0

omp_get_max_threads=0

omp_get_num_procs=0

MemProc=-1

MemSpec=-1

MemVal=-1

MPIFFT_time0=1.2e-06

MPIFFT_time1=0.0359416

MPIFFT_time2=0.0617188

MPIFFT_time3=0.0182264

MPIFFT_time4=0.129112

MPIFFT_time5=0.0388648

MPIFFT_time6=1.4e-06

End of Summary section.
```

Appendix E


Data
Progression of
Start
Name of file
"Named file"
Read next
Store data In internal Array list
N
End of
Y
Calculate mean Time and MOps For each Class size and # of processors
Calculate St. dev. Time and MOps For each Class size and
Write Mean times
Write Mean MOps
Write St. dev. times
Write St. dev. MOps
End

Appendix F

```
import java.util.*;                      // allows use of arraylists and Math functions
import java.lang.*;                      // allows use of wrapper classes
public class AiSC2{
 private static ArrayList all;
 public static void main(String[] args){
  int maxProcs=9;
  boolean hpc=false;
  Input in=new Input();
  System.out.println("what file should I analyse?");
  String f=in.readWord();
  if(f.indexOf("HPC")>=0){
   maxProcs=16;
   hpc=true;
   readFromHPC(f);
   }
   else{
   readFromNAS(f);
   }              // holds all NAS objects
  double[][][] mean =new double[5][maxProcs][2];
```

```
/*[order(0=s,1=a,etc...)]  [#of nodes (2^thisValue)]  [0=time,1=mop/s] */
  for(int o=0;o<5;o++){                             // order
   for(int n=0;n<maxProcs;n++){                         //nodes used = 2^n
    double totalTime=0;
    double totalMops=0;
          int numberFound=1;
    for(int c=0;c<all.size();c++){                 // search the  array for qualities
     NAS now =new NAS((NAS)all.get(c));
           if(now.getOrderInt()==o && now.getProcs()==Math.pow(2,n) && !hpc){
            totalTime+=now.getTime();
            totalMops+=now.getMops();
            numberFound++;
            }
           else{
            if(now.getOrderInt()==o && now.getProcs()==Math.pow(n,2)){
             totalTime+=now.getTime();
             totalMops+=now.getMops();
             numberFound++;
             }
            }
           }
          if(numberFound!=0){
           mean[o][n][0]=totalTime/numberFound;
           mean[o][n][1]=totalMops/numberFound;
           }
          }
   }
  double[][][] varience =new double[5][maxProcs][2];

/*[order(0=s,1=a,etc...)]  [#of nodes (2^thisValue)]  [0=time,1=mop/s] */
  for(int o=0;o<5;o++){                              // order
   for(int n=0;n<maxProcs;n++){                         //nodes used = 2^n
    double varTime=0;
    double varMops=0;
          int numberFound=1;
    for(int c=0;c<all.size();c++){                // search the  array for qualities
     NAS now =new NAS((NAS)all.get(c));
           if(now.getOrderInt()==o && now.getProcs()==Math.pow(2,n) && !hpc){
            varTime+=(Math.pow(now.getTime()-mean[o][n][0],2));
            varMops+=(Math.pow(now.getTime()-mean[o][n][1],2));
            numberFound++;
            }
           else{
            if(now.getOrderInt()==o && now.getProcs()==Math.pow(n,2)){
             varTime+=(Math.pow(now.getTime()-mean[o][n][0],2));
             varMops+=(Math.pow(now.getTime()-mean[o][n][1],2));
```

```
              numberFound++;
              }
             }
            }
           if(numberFound!=0){
            varience[o][n][0]=Math.pow(varTime/numberFound,0.5);
            varience[o][n][1]=Math.pow(varMops/numberFound,0.5);
            }
           }
   }
  double[][][] deviation =new double[5][maxProcs][2];

/*[order(0=s,1=a,etc...)]  [#of nodes (2^thisValue)]  [0=time,1=mop/s] */
  for(int o=0;o<5;o++){                              // order
   for(int n=0;n<maxProcs;n++){                          //nodes used = 2^n
    for(int ToM=0;ToM<2;ToM++){                                  // Time or Mop/s
            deviation[o][n][ToM]=varience[o][n][ToM];
            }
    }
   }
  CSPrintWriter outMeanTime =new CSPrintWriter(f+"-MeanTime.csv");
  CSPrintWriter outVarTime =new CSPrintWriter(f+"-VarTime.csv");
  CSPrintWriter outDevTime =new CSPrintWriter(f+"-StDevTime.csv");
```

```
 CSPrintWriter outMeanMops =new CSPrintWriter(f+"-MeanMops.csv");
 CSPrintWriter outVarMops =new CSPrintWriter(f+"-VarMops.csv");
 CSPrintWriter outDevMops =new CSPrintWriter(f+"-StDevMops.csv");
 for(int chooser=0;chooser<3;chooser++){                    // chooser decides
which

file to print to
  for(int o=0;o<5;o++){                         // order
   for(int n=0;n<maxProcs;n++){                      //nodes used = 2^n
       switch(chooser){
        case 0:
        outMeanTime.print(mean[o][n][0]+",");
        outMeanMops.print(mean[o][n][1]+",");
        break;
        case 1:
        outVarTime.print(varience[o][n][0]+",");
        outVarMops.print(varience[o][n][1]+",");
        break;
        case 2:
        outDevTime.print(deviation[o][n][0]+",");
        outDevMops.print(deviation[o][n][1]+",");
        }
       }
```

```
           outMeanTime.println(",");                          // starts new lines
           outVarTime.println(",");
           outDevTime.println(",");
           outMeanMops.println(",");                          // starts new lines
           outVarMops.println(",");
           outDevMops.println(",");
    }
   }
           outMeanTime.close();
           outVarTime.close();
           outDevTime.close();
           outMeanMops.close();
           outVarMops.close();
           outDevMops.close();
  }
private static void readFromNAS(String file){
 Input in =new Input(file);
 all=new ArrayList();
 while(!in.eof()){                              // eof = end of file
  String order=in.readWord();
  all.add(new NAS(in.readInt(),in.readDouble(),in.readInt(),in.readDouble()));
 }
}
```

```
 private static void readFromHPC(String file){
  Input in =new Input(file);
  int type= -1;
  all=new ArrayList();
  while(!in.eof()){                         // eof = end of file
   String order=in.readWord();
   if(order.equals("HPL")){type=0;}
   if(order.equals("PTRANS")){type=1;}
   if(order.equals("GUPS")){type=2;}
   if(order.equals("FFT")){type=3;}
   all.add(new

NAS(true,type,in.readInt(),in.readDouble(),in.readInt(),in.readDouble()));
  }
 }
}
```

Appendix G

```
import java.lang.*;
public class NAS{
 private double mops=0, time=0;
 private int procs=0, size=0, hType=0;     // 0=hpl 1=ptrans 2=gups 3=fft
 private boolean hpc=false;
 NAS(){
  }
 NAS(NAS old){
  mops=old.getMops();
  time=old.getTime();
  procs=old.getProcs();
  size=old.getSize();
  hpc=old.gethpc();
  }
 NAS(boolean h,int T,int s,double t,int p,double m){
  size=s;
  time=t;
  procs=p;
  mops=m;
  hpc=h;
  hType=T;
 }
NAS(int s,double t,int p,double m){
 size=s;
 time=t;
 procs=p;
 mops=m;
 }
public boolean gethpc(){
 return hpc;
}
public void sethpc(boolean h){
 hpc=h;
}
public int getOrderInt(){
 if(!hpc){
  switch (size){
   case 1400:                    //cg s
   return 0;
   case 4096:                    //lu s
   return 0;
   case 25:              //ep s=2^n
   return 0;
   case 14000:                   //cg a
```

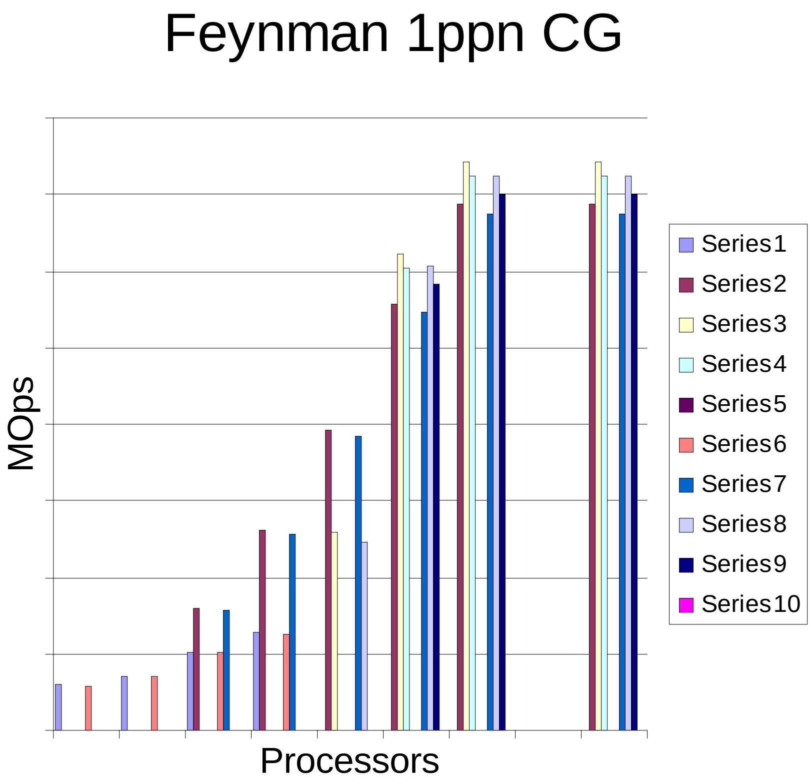
```
return 1;

case 262144:                    //lu a

return 1;

case 29:              //ep a=2^n

return 1;

case 75000:                     //cg b

return 2;

case 1061208:                   //lu b

return 2;

case 31:              //ep b=2^n

return 2;

case 150000:                    //cg c

return 3;

case 4251528:                   //lu c

return 3;

case 33:              //ep c=2^n

return 3;

case 1500000:                   //cg d

return 4;

case 67917312:                  //lu d

return 4;

case 37:              //ep d=2^n

return 4;
```

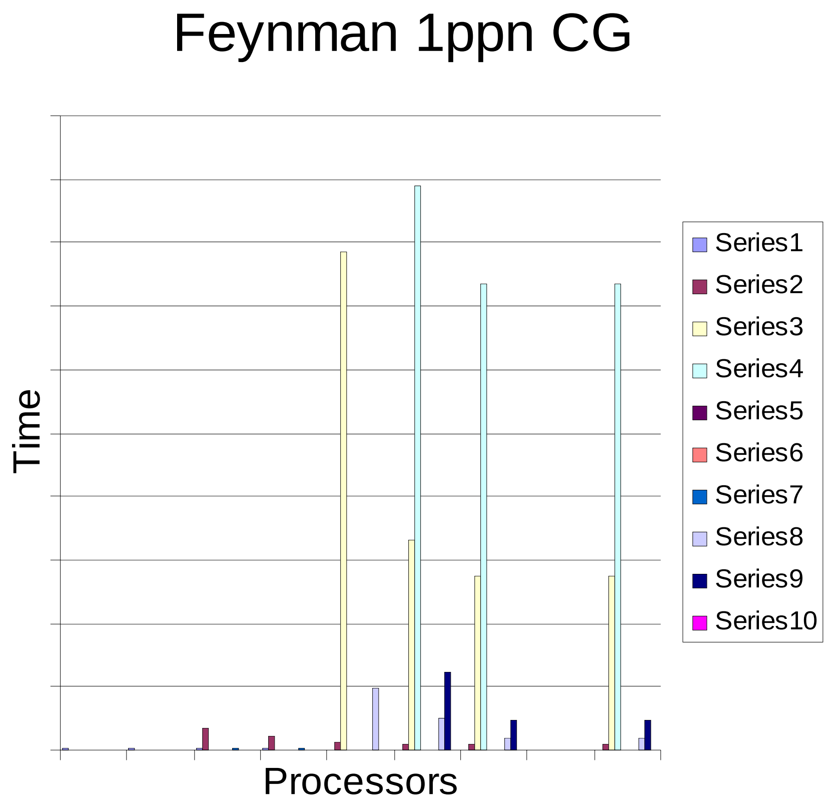
```
   }
  }
 else{
  return hType;
  }
 return -1;
 }
public double getMops(){
 return mops;
 }
public void setMops(double m){
 mops=m;
 }
public double getTime(){
 return time;
 }
public void setTime(double t){
 time=t;
 }
 public int getProcs(){
 return procs;
 }
public void setProcs(int p){
 procs=p;
 }
public int getSize(){
 return size;
 }
public void setSize(int s){
 size=s;
 }
}
```

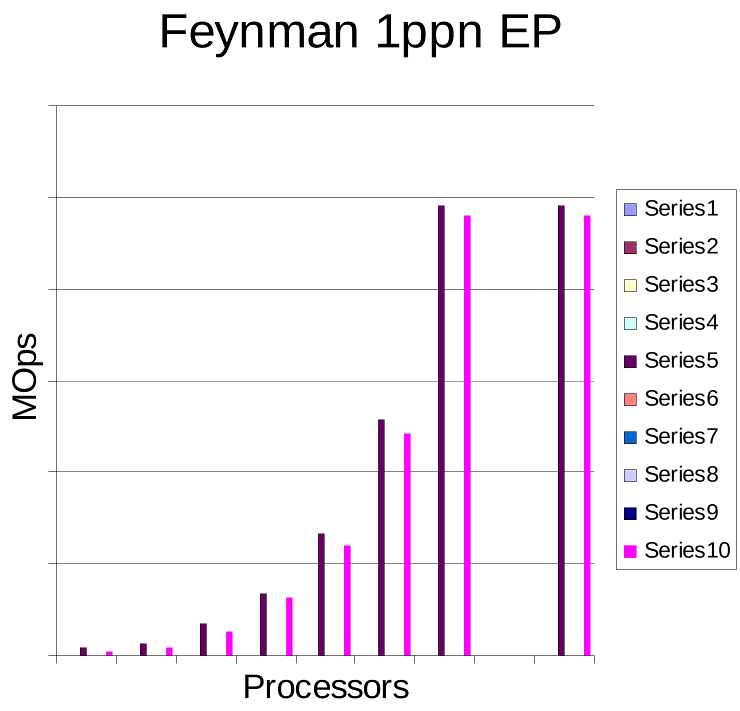
Appendix H

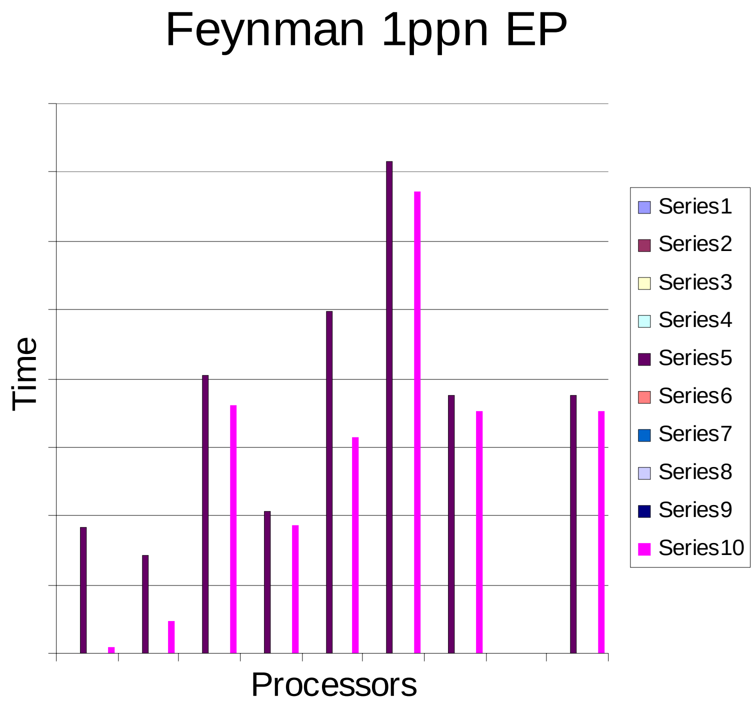
```
293.1176190476191,302.6634146341464,493.7914285714285,537.3166666666666,0.0,0.0,0.0,0.0,0.0,,
0.0,0.0,550.1661904761904,889.0495238095239,1310.5528571428572,1985.2466666666667,2303.7276923076925,0.0,2303.7276923076925,,
0.0,0.0,0.0,0.0,1145.9580952380954,2011.4876190476193,2646.4469230769237,2254.25,2646.4469230769237,,
0.0,0.0,0.0,0.0,0.0,1814.2014285714288,2780.599230769231,2814.935,2780.599230769231,,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,1125.15,0.0,,
```

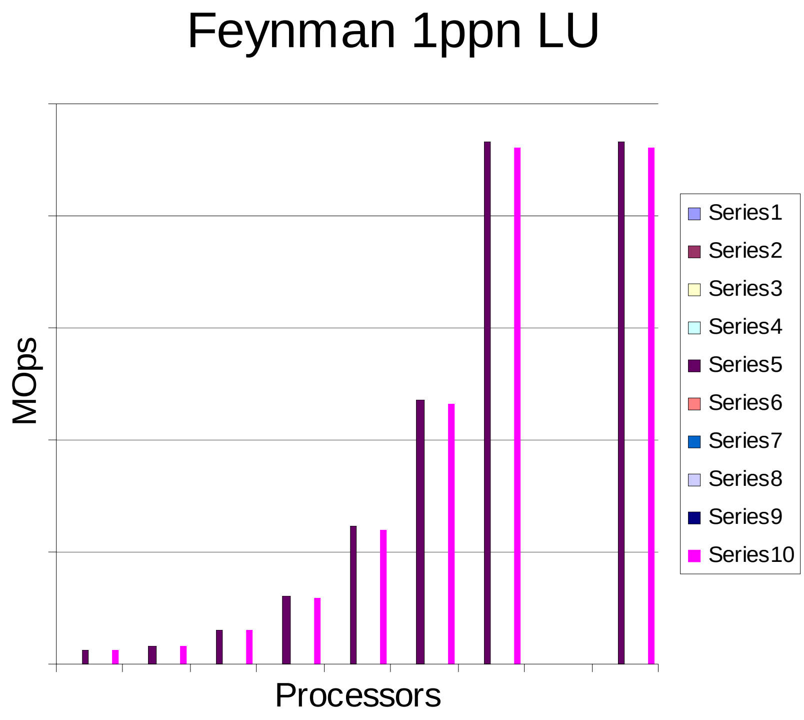
Appendix I

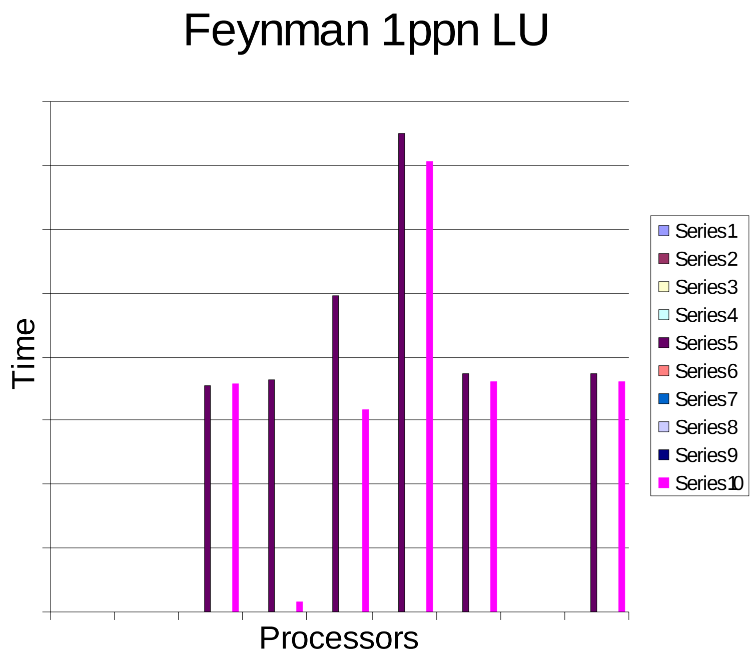

Feynman 1ppn CG
MOps
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

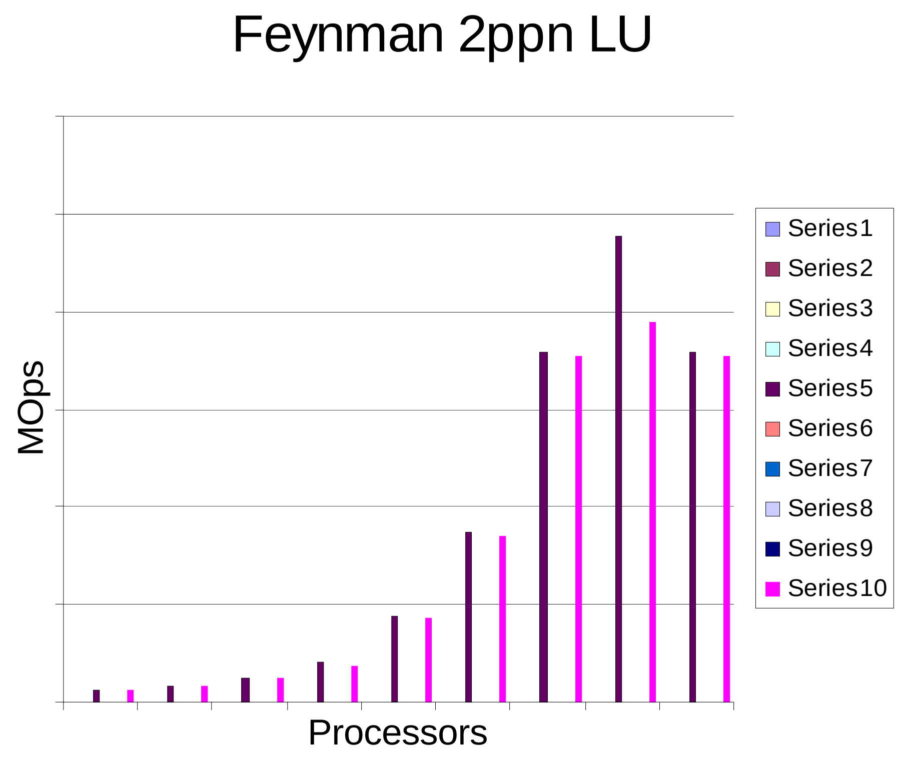

Feynman 1ppn CG
Time
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

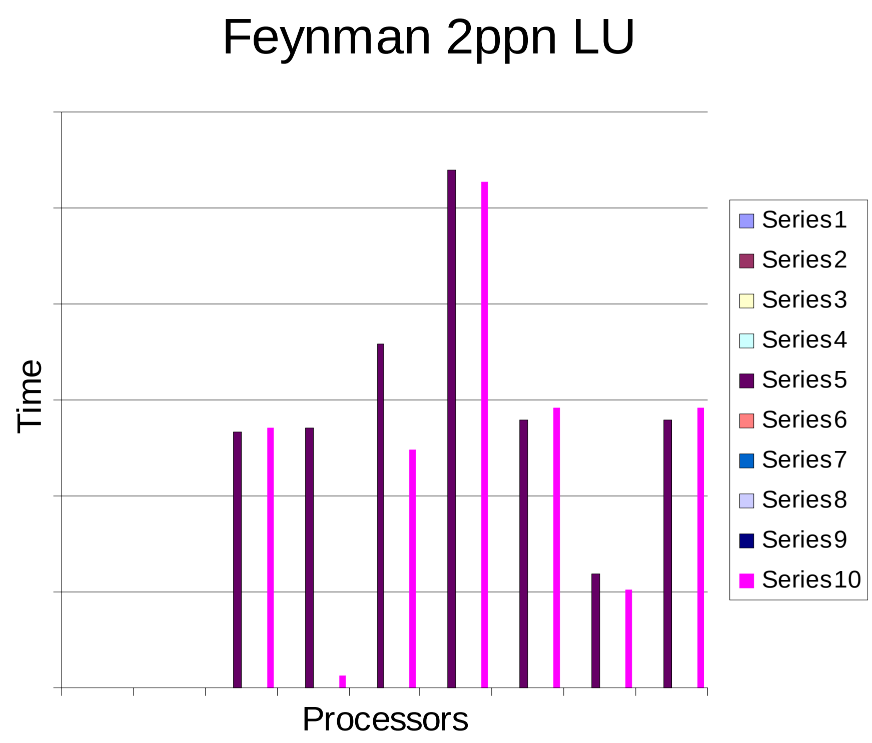

Feynman 1ppn EP

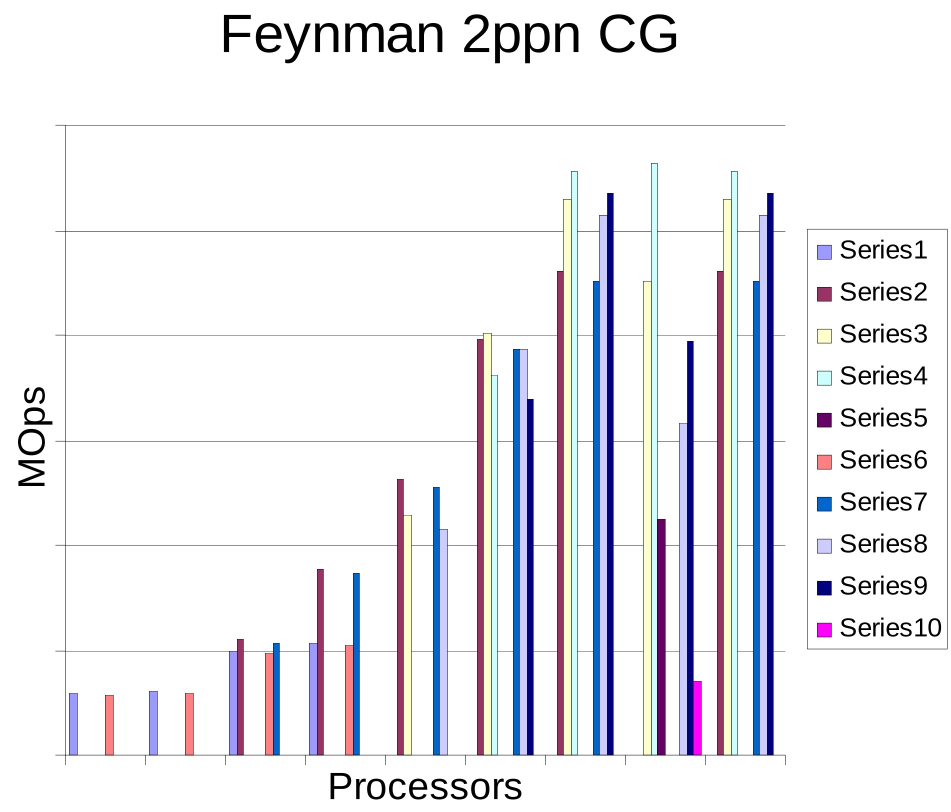

Feynman 1ppn EP

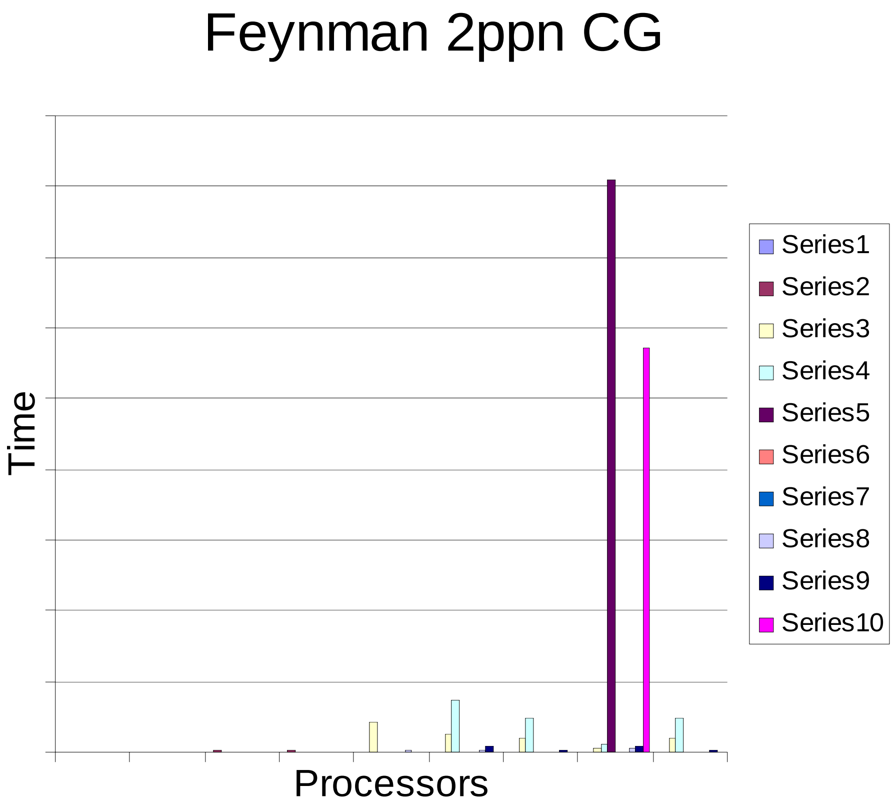

Feynman 1ppn LU
MOps
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

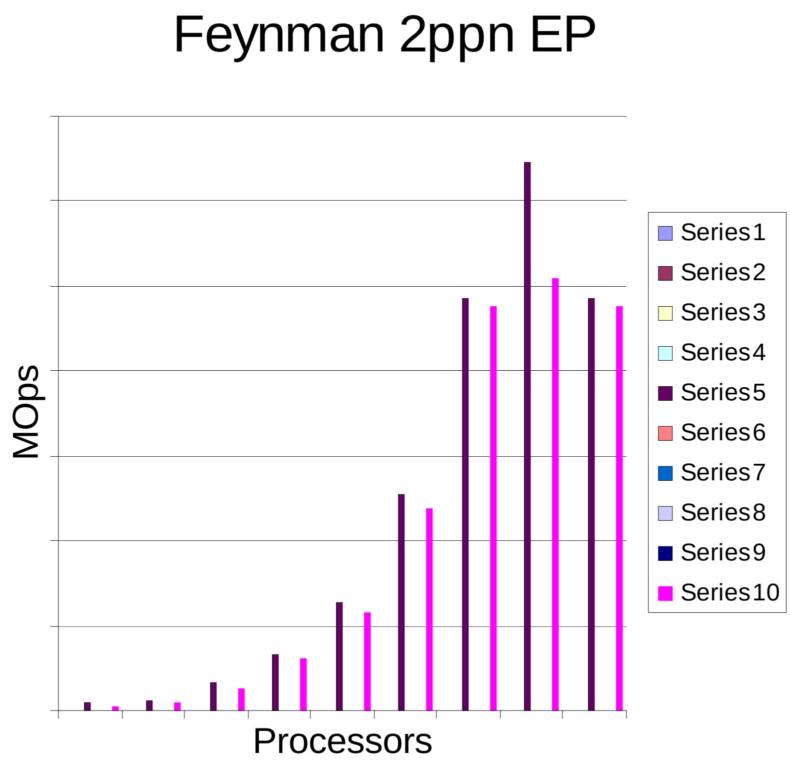

Feynman 1ppn LU
Time
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

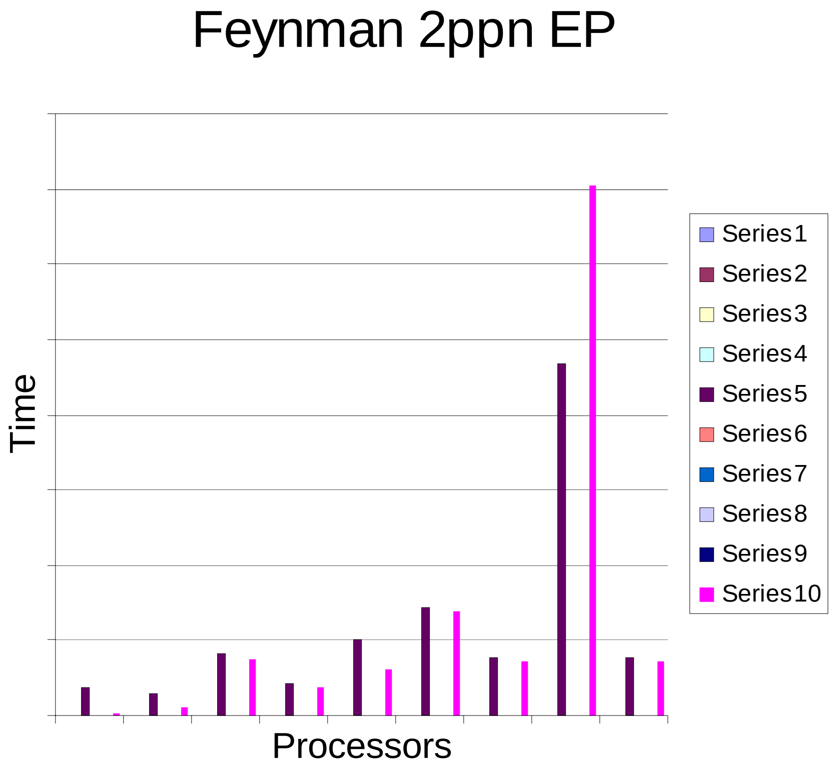

Feynman 2ppn LU
MOps
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

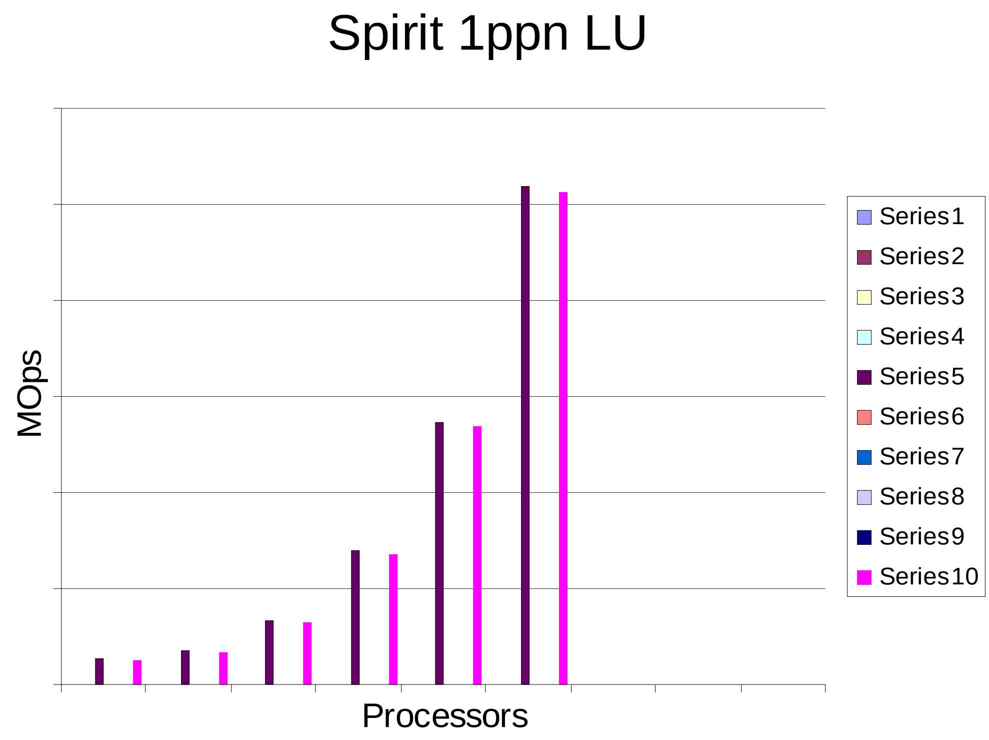

Feynman 2ppn LU
Time
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

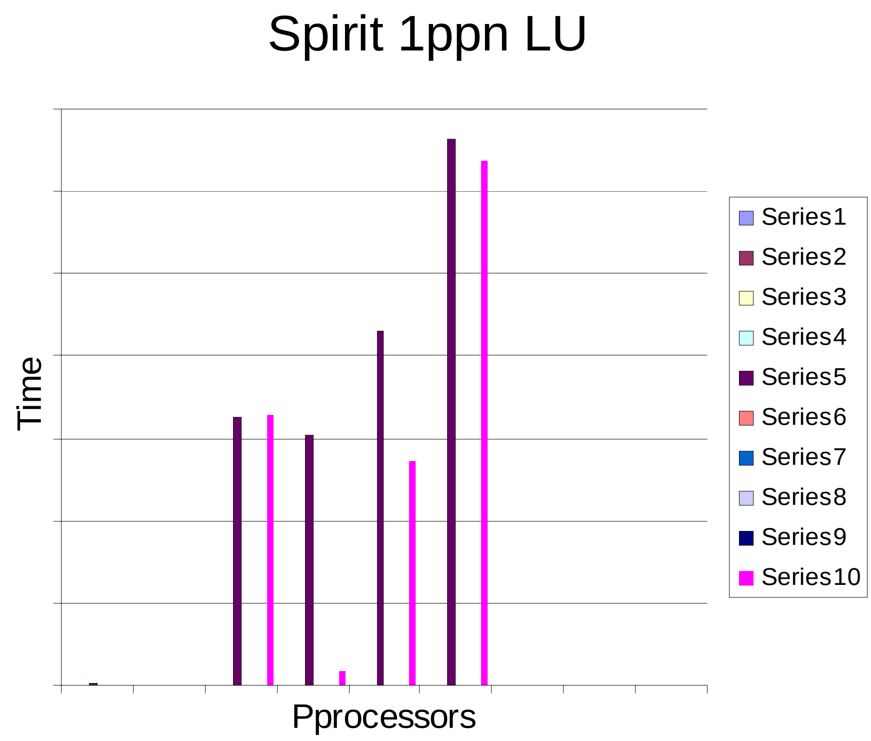

Feynman 2ppn CG
MOps
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

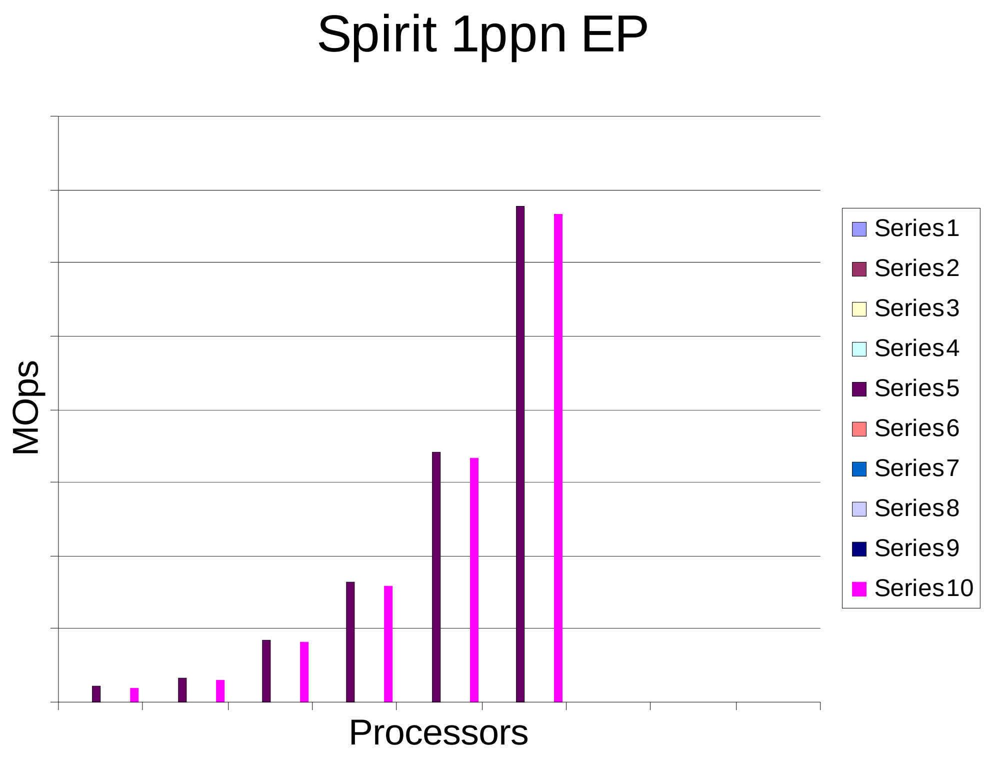

Feynman 2ppn CG
Time
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

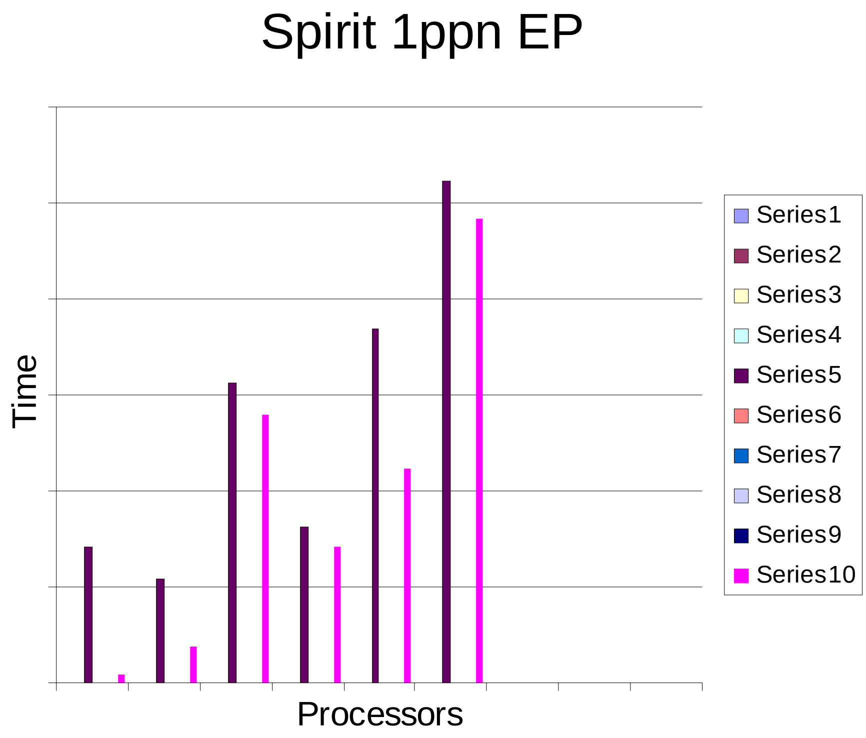

Feynman 2ppn EP
MOps
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

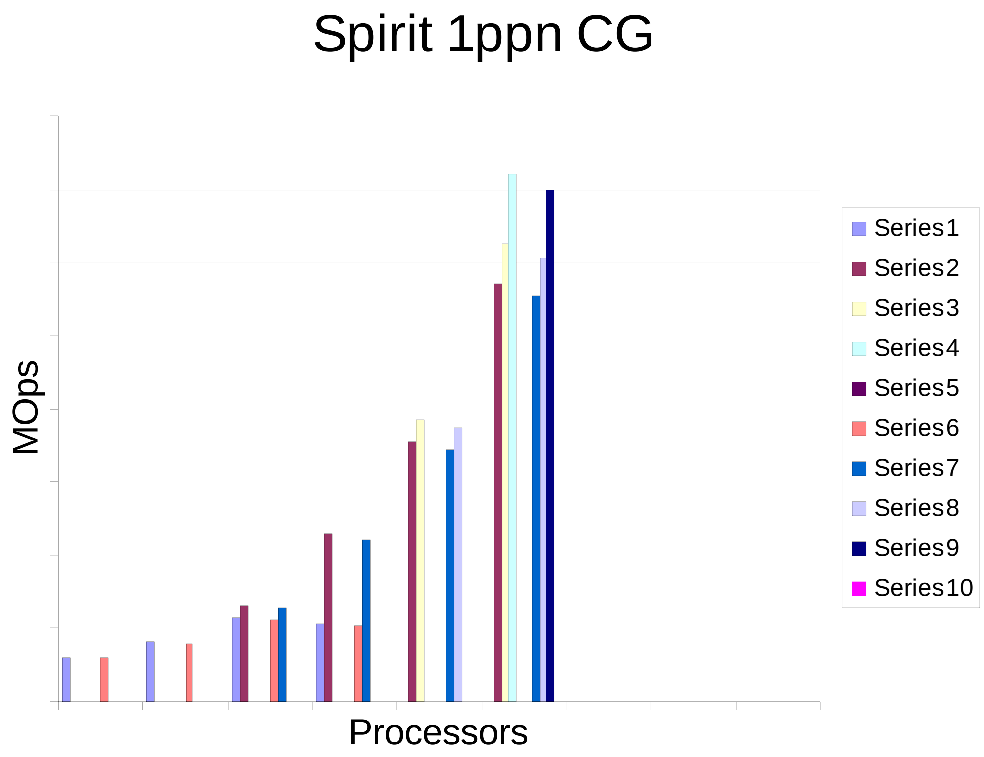

Feynman 2ppn EP
Time
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

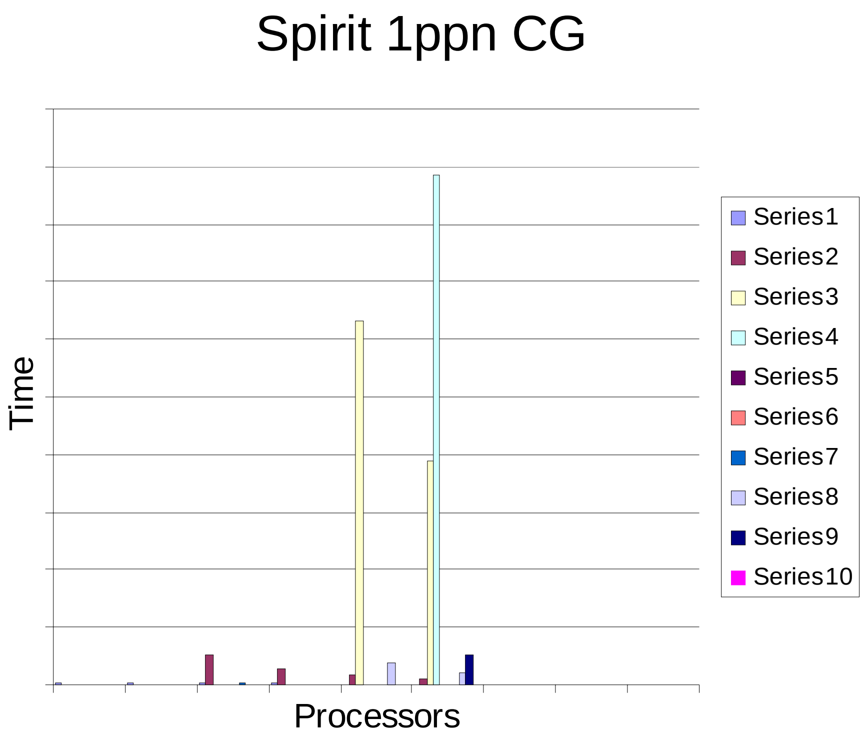

Spirit 1ppn LU
MOps
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

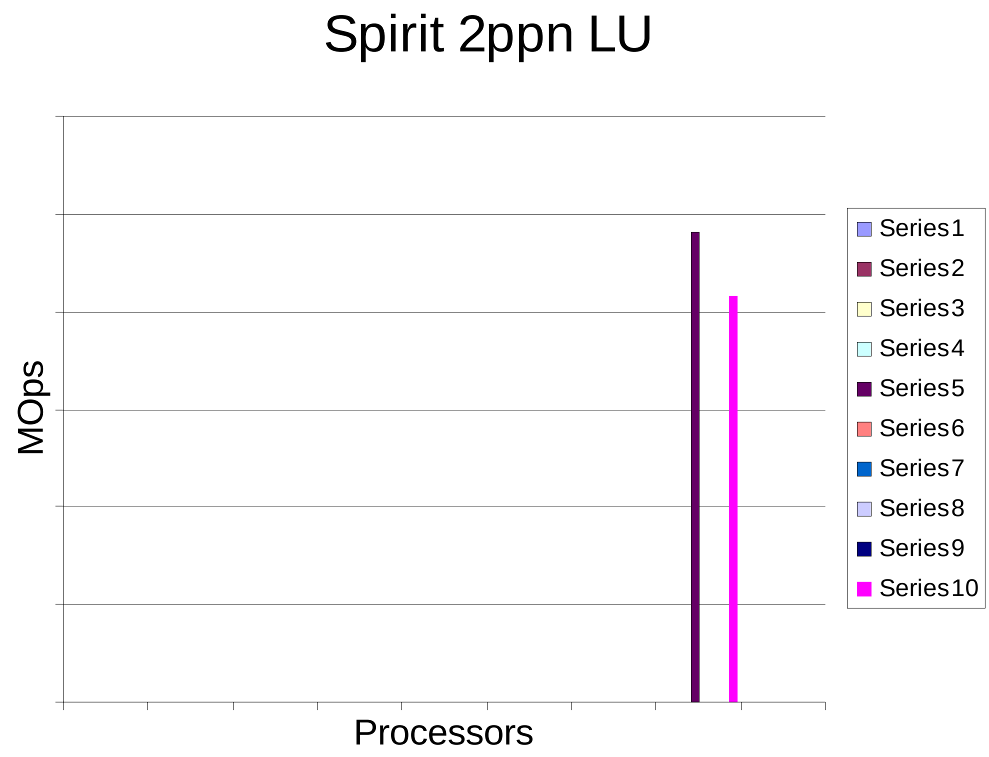

Spirit 1ppn LU
Time
Pprocessors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

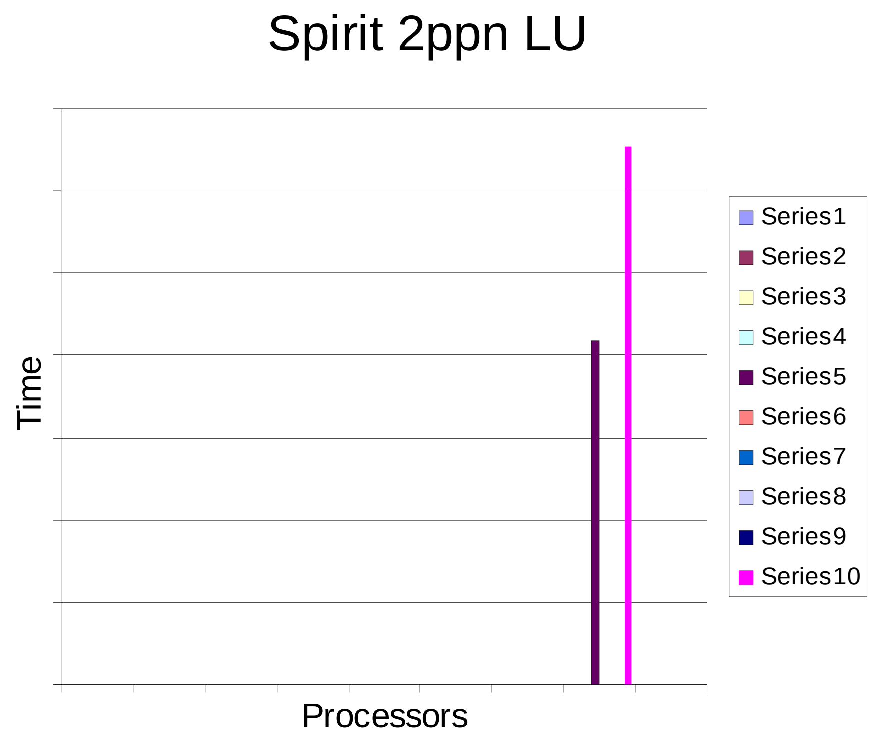

Spirit 1ppn EP
MOps
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

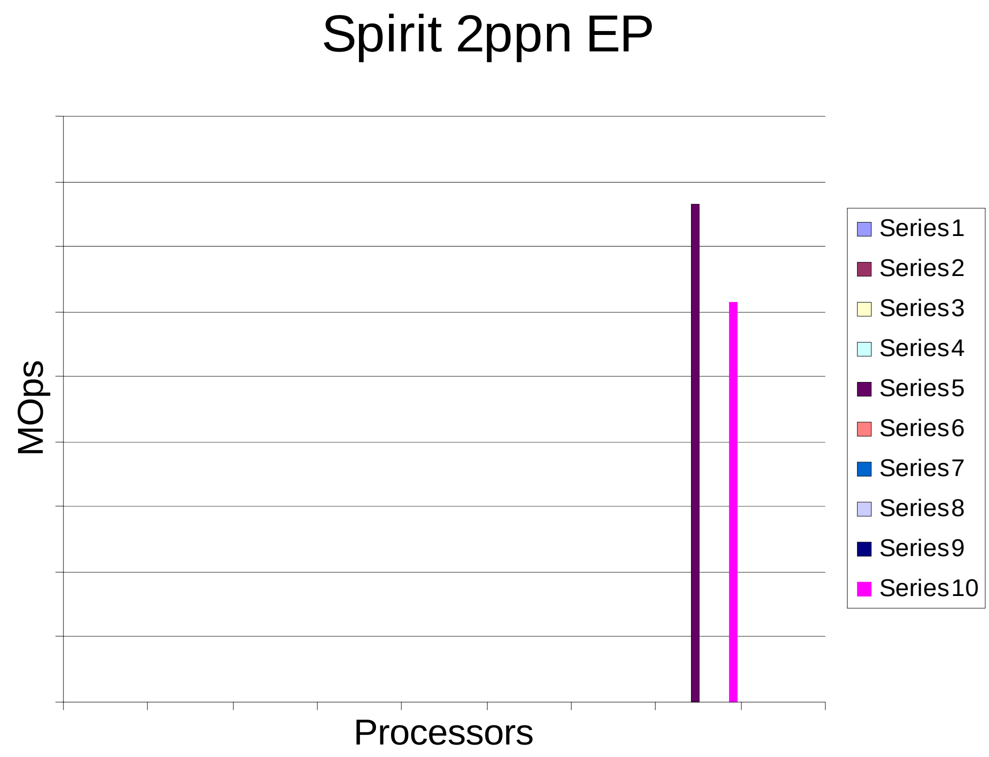

Spirit 1ppn EP
Time
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

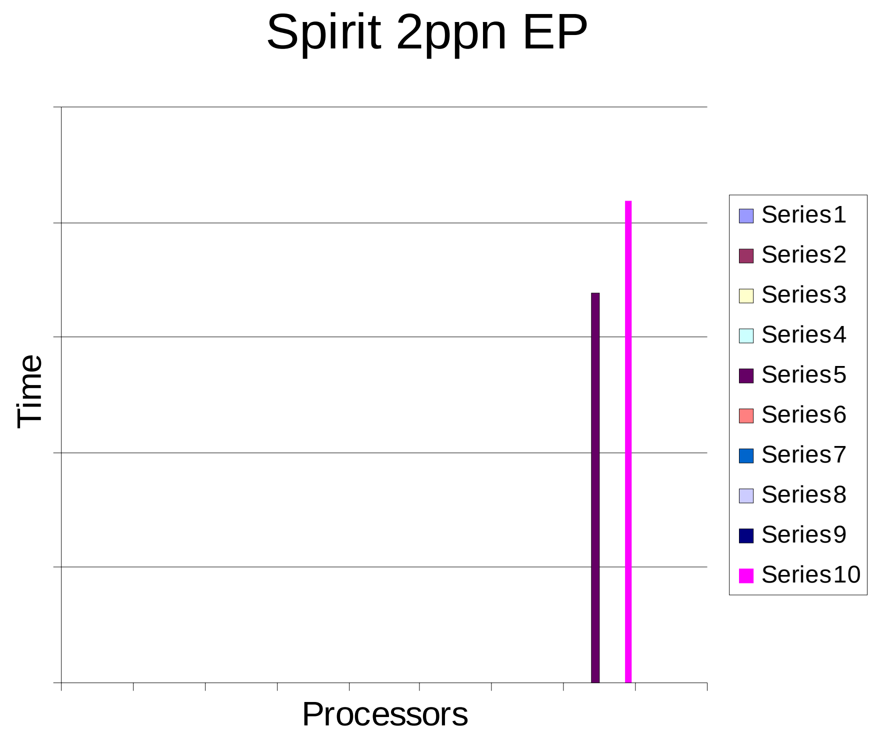

Spirit 1ppn CG
MOps
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

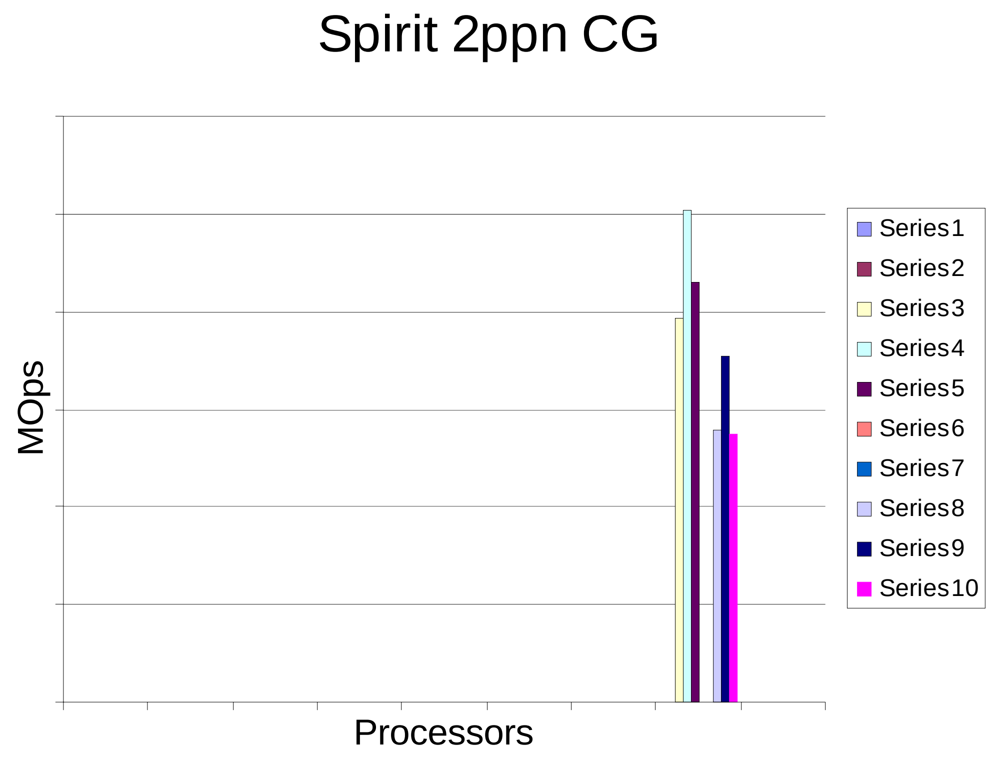

Spirit 1ppn CG
Time
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

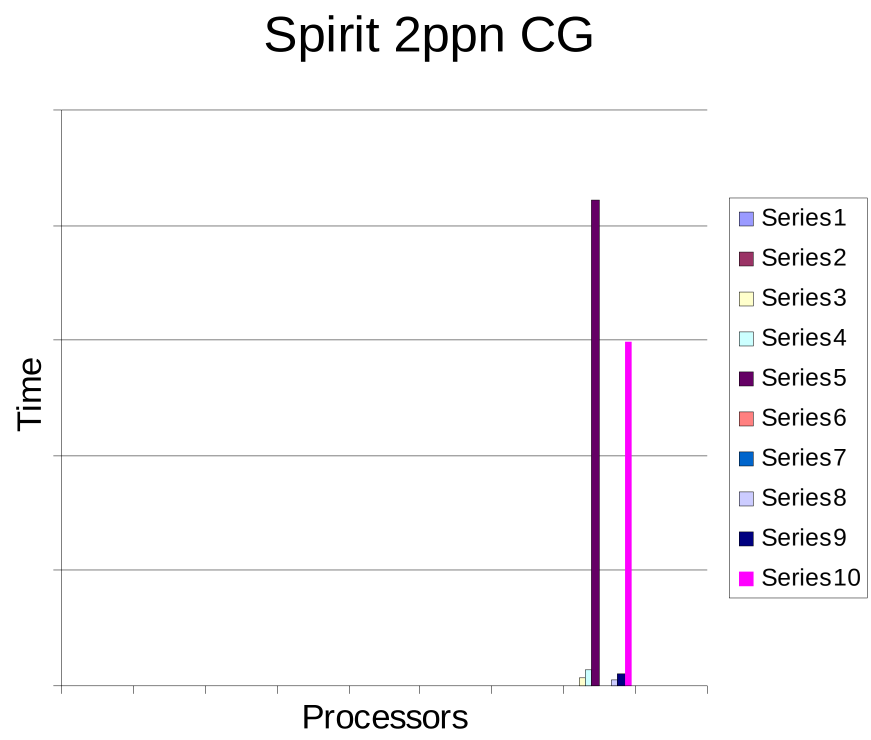

Spirit 2ppn LU
MOps
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

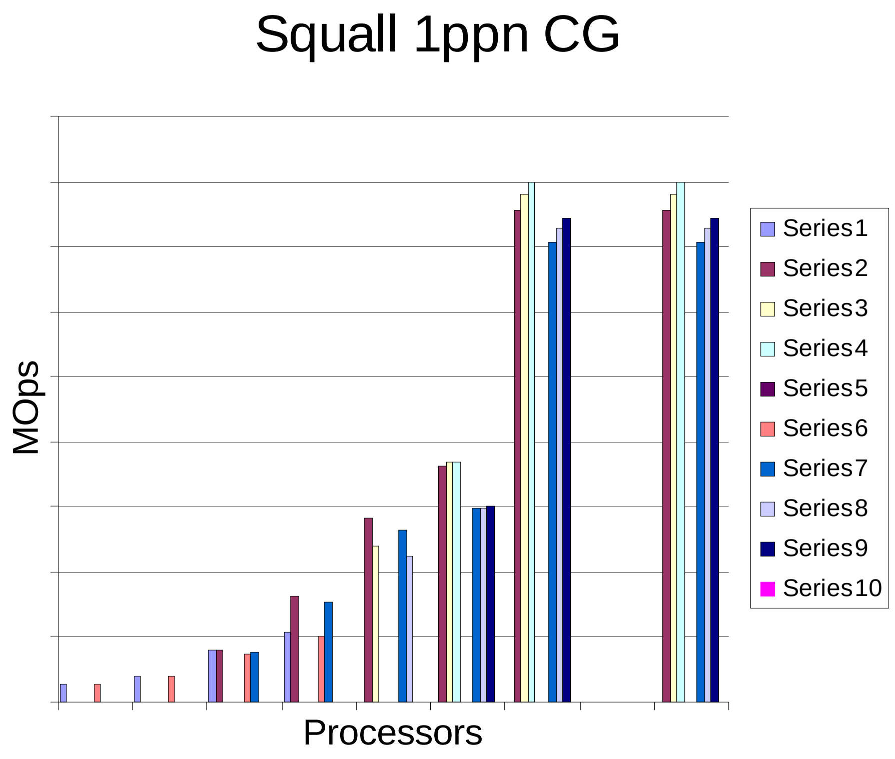

Spirit 2ppn LU
Time
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

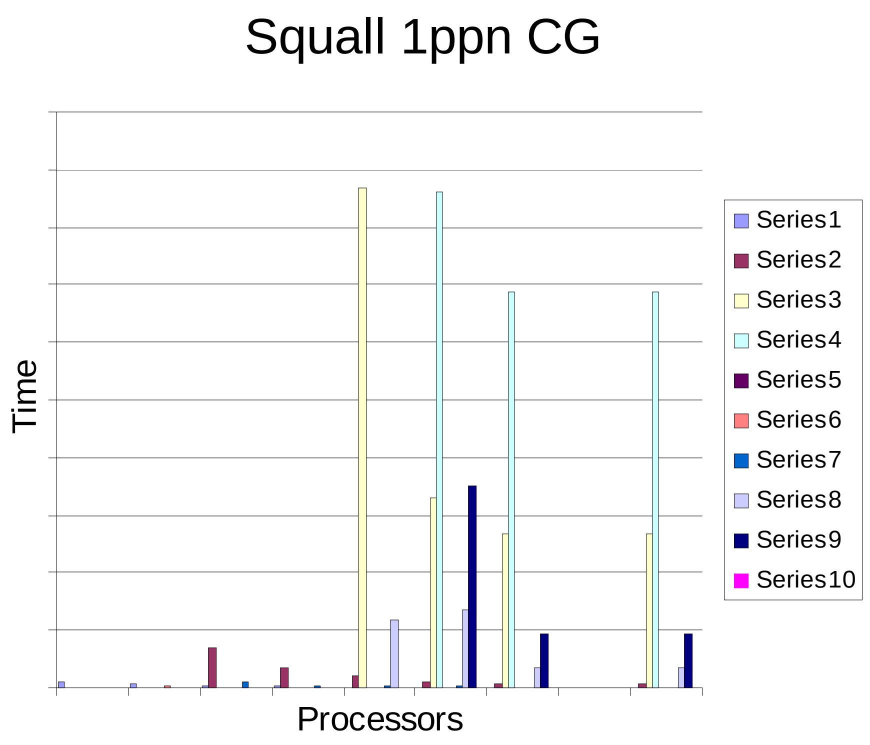

Spirit 2ppn EP
MOps
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

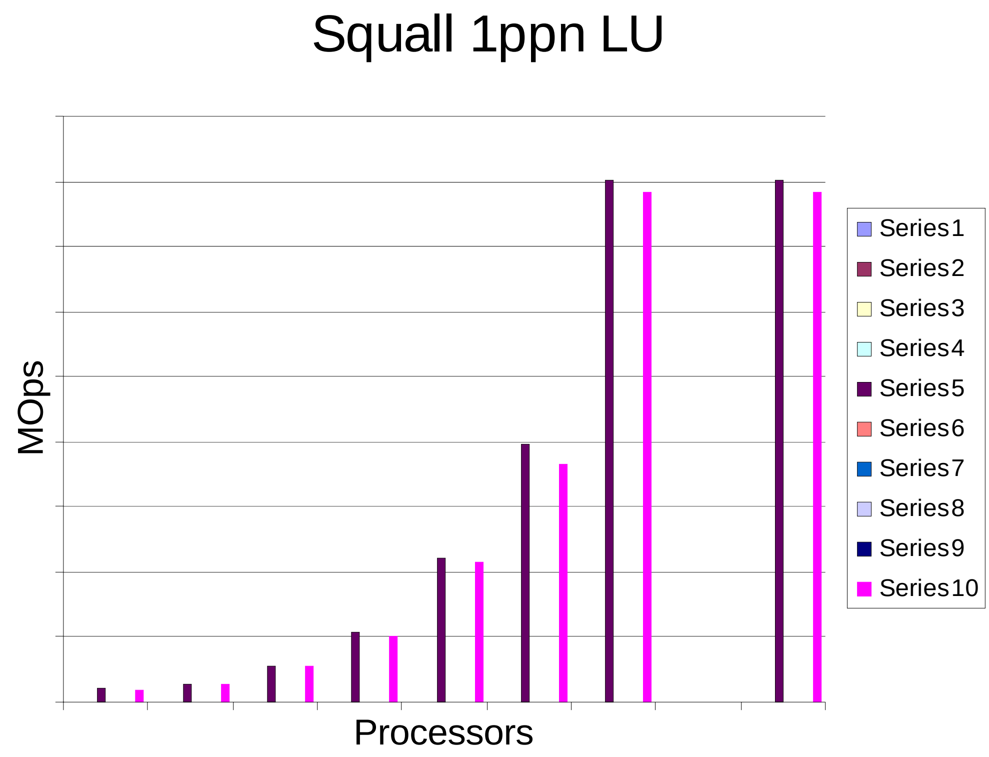

Spirit 2ppn EP
Time
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

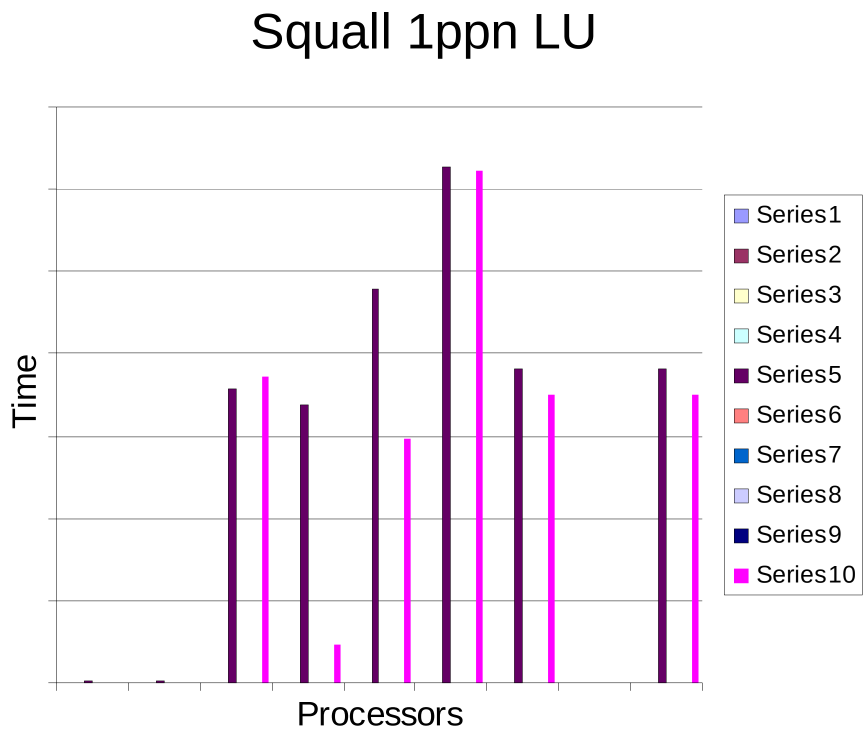

Spirit 2ppn CG
MOps
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10


Spirit 2ppn CG
Time
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

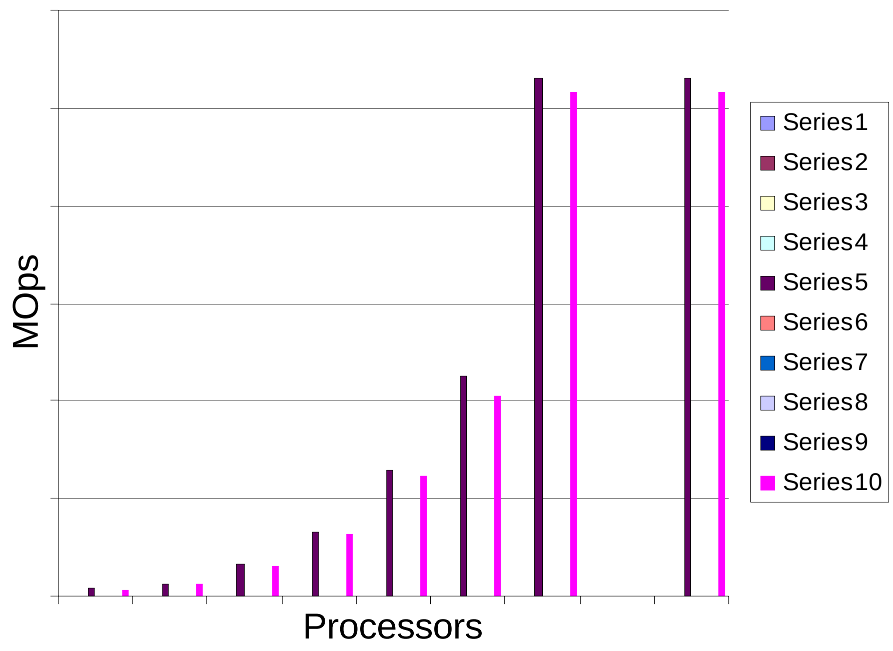

Squall 1ppn CG
MOps
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10


Squall 1ppn CG
Time
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

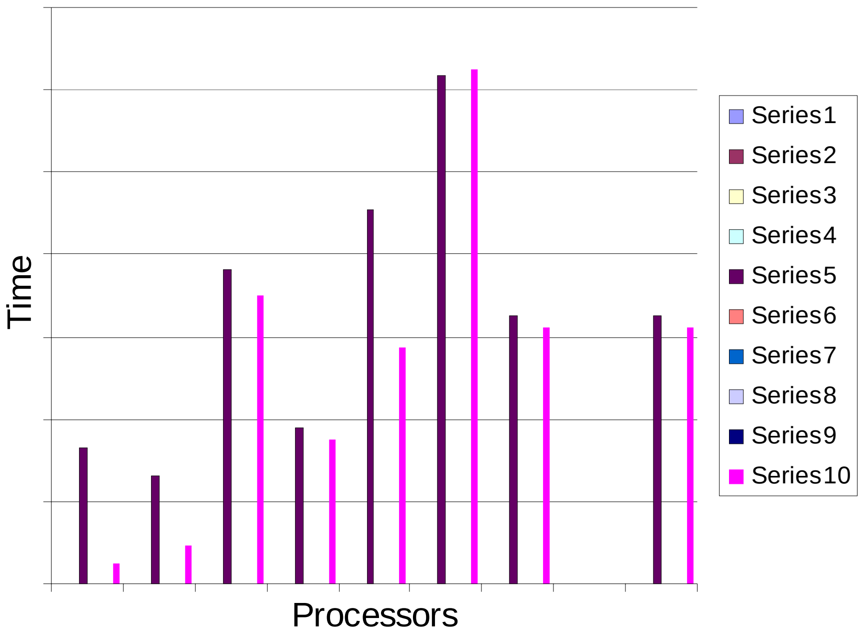

Squall 1ppn LU
MOps
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10


Squall 1ppn LU
Time
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10

Squall 1ppn EP



Squall 1ppn EP


Time
Processors
Series1
Series2
Series3
Series4
Series5
Series6
Series7
Series8
Series9
Series10