Benchmarking the Amazon Elastic Compute Cloud (EC2)

A Major Qualifying Project Report submitted to the Faculty of the WORCESTER POLYTECHNIC INSTITUTE in partial fulfillment of the requirements for the Degree of Bachelor of Science by

Patrick Coffey

Niyanta Mogre

Jason Beliveau

Andrew Harner

Approved By:

WPI Project Advisor
Professor Donald R. Brown

March 9, 2011

Abstract

This project sought to determine whether Amazon EC2 is an economically viable environment for scientific research-oriented activities within a university setting. The methodology involved benchmarking the performance of the cloud servers against the best systems available at the WPI ECE department. Results indicate that the newest ECE server outperformed the best EC2 instance by approximately 25% in most cases. A comprehensive cost analysis suggests that EC2 instances can achieve up to 60% better cost to performance ratios in the short-term when compared against the ECE servers. However, a long-term projected cost analysis determined that the overall cost of owning a large set of reserved instances comes out to almost 60% more than the cost of comparable in-house servers.

Acknowledgments

We would like to thank our advisor, Professor Donald R. Brown for his guidance throughout the course of this project. We would also like to thank Professor Sergey Makarov for providing us access to his research for our case study, as well as Professor Berk Sunar for advising us on the aspects of computer architecture that were needed to analyze our results. We would also like to thank Professor Hugh Lauer for allowing us the opportunity to run the SPEC benchmarks, as well as his advice in understanding benchmarking concepts. Finally, we would like to thank Robert Brown for allowing us the use of the ECE servers, as well as his generosity in providing us sufficient resources to enable our benchmarking.

Table of Contents

Contents

1 Introduction

Most private institutions run lengthy simulations and calculations in the process of conducting scientific research. To meet their computing needs, they typically invest in state of the art supercomputers and servers designated for general use by faculty and students. Owning a server generates a cumbersome amount of overhead costs, including the initial cost of purchasing the machine, the procurement of physical space, annual power and cooling costs, hard disk and power redundancy, and various other maintenance costs. In addition to the monetary costs, servers also carry certain usage and time constraints. During peak usage, public servers are exposed to high traffic situations which hamper the performance of all users. Utilizing task prioritization can help to mitigate this effect, but it is often either used incorrectly or is not an option. In addition to periods of heavy traffic, servers also experience extended stretches of time when they have nothing else to do but idle, which results in a waste of resources.

Rather than purchasing in-house servers, some institutions opt to either lease remote servers or purchase virtual computing resources; the latter, and the focus of our research, is known as cloud computing. Modern cloud computing services allow anyone with a credit card to rent out whatever scale of computational resources they require for almost any period of time. Amazon, Google, and Microsoft are just a few of the corporations that currently provide this form of computing. Their cloud computing services include the Amazon Elastic Compute Cloud (EC2), Google Apps, Microsoft Azure respectively.

Amazon's cloud resources are elastic, meaning that it is possible to modify the amount of resources being allocated at any given time. Rather than being charged for wasted use of processing power—or idle CPU time—the user is only charged for the time that the cloud is in use. Clouds also transfer the burden of providing physical space, power, cooling and disk backups from the institution onto the service provider. There are no wait periods or maintenance costs, and the cloud services are equipped with a large variety of instance types that provide a wide array of computing power. Instances range from small and trivial instances—which could be used for simple web servers—to larger instances with more resources, used for heavier scientific computation.

A considerable amount of prior research has focused on utilizing Amazon EC2 for cluster computing, while relatively few studies have investigated using the service for single-node applications. The few analyses that have been conducted on single-node computing state that EC2 shows a small yet acceptable virtualization overhead—the amount of resources that are wasted in providing a virtual environment—when compared to physical servers [1, 2]. There has been little notable examination into the use of EC2 as a replacement to permanent servers, with one of the few exceptions demonstrating that EC2 is capable of providing a cost benefit of approximately 37% for an enterprise-level IT system. [3]. Another relevant analysis involved comparisons between desktop grid computing and cloud computing, which proposed that the latter could provide a cost benefit of almost 40% or more [4]. However, this sort of analysis remains relatively uncommon in past work.

The focus of this project was to run a selection of benchmarks on EC2 that would emulate single-node applications, due to the inadequate documentation in prior work on EC2's capabilities in this respect—particularly in university-based environments. The benchmarks that were employed included single and multi-threaded CPU and memory-intensive applications, as well as a real-world research program. Each was run across every relevant type of on-demand instance and physical computer in order to develop a complete set of data. We also worked to create a comprehensive cost analysis which examines the different costs associated with owning an in-house server versus purchasing virtual resources. We also supplement this analysis by examining the cost versus the time to solution for several different scenarios on EC2.

We found that the physical server used for our testing outperformed the most comparable

EC2 instance by approximately 25% across all of the benchmarks that were run. We then compared the costs associated with maintaining a full deployment of in-house servers versus virtual resources from EC2 at an the institutional level, and found that that the cost difference between the two for a 9 year lifespan was sizable. The analysis showed that EC2 is about 57% more expensive, given almost identical computing resources. However, a noteworthy aspect about EC2 that we investigated is that the service can be used to minimize the time to solution—to a certain extent—with no increase in cost in the case of massively parallel applications. We discuss other aspects of cloud computing that must be taken into consideration prior to making a longterm investment in cloud services like EC2 throughout the entirety of our report.

2 Background

The purpose of this section is to provide the reader with a general overview of cloud computing as well as an introduction to Amazon EC2 and the services it provides. We then explore prior studies that researchers have conducted on the EC2 service and discuss the gaps that were found in their work. Finally, we discuss in detail the benchmarks that were used in this study, and their relevance to our project goals.

2.1 Cloud Computing

The idea of cloud computing, a virtual form of network-based processing, was first introduced in the 1960s in the hopes that computing could someday become a public utility. Rather than investing heavily in a physical computing infrastructure, users could instead purchase virtual resources in the same way that one would pay for any other public utility, such as gas or electricity. Cloud computing is essentially made up of a set of data centers located at different geographic locations that provide users with scalable remote computational resources. These data centers can be either private or public—the former is commonly reserved for organizations or corporations for private use, and the latter is widely available to the general public for personal use [5]. The on-demand nature of these services allows immediate acquisition of any amount of computing power, without requiring any commitment to the resources by the customer.

2.2 The Amazon Elastic Compute Cloud

Amazon was one of the first companies to become interested in the development of cloud computing, and officially launched their flagship service in 2008. EC2 currently offers a wide array of instances, each of which is priced differently depending on its grade of computing power. The computing power is expressed in terms of EC2 Compute Units (ECUs), which are each equivalent to 1.0-1.2 GHz of a 2007 Xeon or Opteron processor [6]. These instances (excluding the cluster computing resources) are shown in Table 1.

Type	**Grade**	**Architecture**	**CPU**	**Cores**	**Memory**	**Storage**	**Price [Linux] (USD/hour)**
Micro	Micro	32/64-bit	< 2 ECU	< 2	613MB	EBS only	$0.020
Standard	Small	32-bit	1 ECU	1	1.7GB	160GB	$0.085
	Large	64-bit	4 ECU	2	7.5GB	850GB	$0.340
	XLarge	64-bit	8 ECU	4	15GB	1690GB	$0.680
High Memory	XLarge	64-bit	6.5 ECU	2	17.1GB	420GB	$0.500
	Double XLarge	64-bit	13 ECU	4	34.2GB	850GB	$1.000
	Quadruple XLarge	64-bit	26 ECU	8	68.4GB	1690GB	$2.000
High CPU	Medium	32-bit	5 ECU	2	1.7GB	350GB	$0.170
	XLarge	64-bit	20 ECU	8	7GB	1690GB	$0.680

Table 1: Instances offered by Amazon EC2 (As noted on 2/15/2011)

EC2 provides customers with many different customizations, such as user-adapted operating systems, multiple storage options and real-time Graphical User Interface (GUI) based resource monitoring consoles. This Amazon Web Services (AWS) interface grants the user the ability to fully control each aspect of their systems' resources [6]. Investing in EC2 frees the user from the maintenance, cooling, powering and location costs associated with private servers while simultaneously granting them access to almost any amount of easily scalable computing power with the AWS console's user-friendly GUI.

Guilbault and Guha [7] praised the ease of use and resource acquisition via Amazon EC2 during their experiment, for which they required hundreds to thousands of computers at a time. The virtual resources offered by Amazon EC2 were ideal for the experimenters to acquire a considerable amount of computational power without resorting to a large number of physical machines to accomplish the same goal. There are, however, certain disadvantages that must be taken into consideration when determining the viability of using the EC2 systems.

It was determined in the early stages of the project that the newly introduced Micro instance was not suitable for scientific applications due to its highly reduced capabilities. Because of this, we chose not to include the instance in our benchmarking process. In addition to the Micro instance, we were also unable to employ the Standard Small and High-Compute Medium instances, due to their incompatibility with 64-bit operating systems.

2.3 In-House Servers

The Electrical and Computer Engineering (ECE) Department at Worcester Polytechnic Institute (WPI) granted us access to a Dell PowerEdge R610 server (Tesla) and a Sun Microsystems Sunfire X2200 M2 server (Henry). These servers are scientific computing devices that are publicly available to students and faculty members for research purposes, which is why they were chosen as a baseline for benchmarking against the EC2 instances. The specifications of the two servers are shown in Table 2.

Server	Operating System	Architecture	CPU	Cores	Memory
Tesla	Red Hat Enterprise Linux Server 5.5	64-bit	2.93 GHz Intel Xeon X5570	8	48GB
Henry	Red Hat Enterprise Linux Server 5.5	64-bit	2.81GHz AMD Opteron 2220	4	20GB

Table 2: WPI ECE Department Server Specifications (As noted in March 2011)

2.4 Prior Work

This section discusses relevant prior work that has been published in regards to EC2, presents the gaps observed therein and then explains how our research has filled these gaps.

2.4.1 Cluster vs. Single-Node Computing

A considerable amount of previous research has suggested that EC2 is not advisable for use in I/O-based or tightly-coupled applications [8, 9]. I/O-based applications are generally applicable to cluster computing, and due to EC2's reduced performance in this area, it has been observed that its cluster computing capabilities have also been compromised [2, 8, 9, 10].The

research conducted by Akioka and Muraoka [8] employed a high-performance linpack (HPL) benchmark, which solves linear equations by using a specific form of Gaussian elimination. The researchers took the specifications of the instances, calculated their theoretical performance and compared it to the actual performance. For the HPL benchmark, they found that the instances vastly underperformed compared to their baseline machine, with results showing performance that was up to 50% below what was expected. Additional studies that evaluated the performance of Amazon EC2 as a cluster computing resource have shown comparable results [9, 10, 11, 12].

Relatively little research has examined single-node computing using Amazon EC2. The study conducted by Berman et al [1] examined the performance of CPU, memory and I/O-intensive aspects of workflow applications. Their results show that EC2 performed between 1% and 10% worse than the capabilities of a supercomputing physical server. They also conducted a brief cost analysis that detailed the expenses incurred while running their benchmarks. The study did not specifically mention any cost-benefit analysis for the specific instances used or provide any type of cost comparison with a physical server.

Research conducted by Rehr et al [2] tested both the serial and parallel performance of three EC2 instances versus single and multiple nodes of a supercomputing cluster. The Athena cluster, a networked computing environment developed and used by MIT, was selected for this purpose [13]. The benchmark they used is known as Gasoline, which simulates various astronomical phenomena including the interaction of gravitational forces and galaxy/planet formation [2]. This benchmark was run on the Standard Small, High-Compute Medium and High-Compute Extra Large instances on EC2, as well as a single node of the Athena supercomputing cluster. Their results indicated that the performance of the Small instance was slightly hindered by virtualization overhead, while the performance of the High-Compute instances was more or less identical to that of the Athena node. There was no cost analysis of any kind described in this study.

2.4.2 Physical vs. Virtual Resources

As mentioned earlier, there has been limited research examining the migration from in-house servers to virtual computing. This type of comparison would look at not only the differences in the performance of a physical machine and the cloud, but also the overall cost involved with either route. This would lead to an evaluation determining the most viable option in terms of both cost and performance. The following studies, while relevant to this kind of analysis, are limited in scope and fail to completely address both sides of the equation.

A case study conducted by Greenwood et al [3] looked into the migration of an enterprise-level IT system to the cloud—more specifically, EC2. Their goal was to determine the various cost and maintenance benefits that could be readily observed by corporations and businesses due to such a migration. Three sets of EC2 instances were used—two Small, one Small and one Large, and two Large—to simulate the infrastructure for a particular oil and gas company. The overall cost benefit of migrating to each of these sets of instances, as well as the types of support issues that resulted thereof, were then examined quantitatively. They found that migrating the IT system to EC2 provided a 37% cost benefit over a span of 5 years (assuming the Small and Large instance set was used) as well as reducing the number of support issues by 21%. Despite these advantages, they also stated that there could be issues—including a lower quality of service and customer dissatisfaction, overall downsizing of the technology staff and reduced support resources—that would make migrating an enterprise-level infrastructure to EC2 an undesirable option.

Anderson et al [4] published research that illustrates, to a certain extent, a performance and cost comparison between physical computing resources and cloud computing. The study compares volunteer desktop grid computing to cloud computing, and tests whether the EC2 service is more

cost-efficient. Volunteer desktop grids are essentially physical computing resources that are donated by computer owners over a network for a group of individuals to use for scientific problems or research [14]. The study found that cloud computing, specifically EC2, was actually more cost-effective than developing a network of computer donors for the same computational purpose, with a cost-benefit in the range of 40-95%. Additional studies that compared cloud computing to desktop grids had similar findings, showing that the benefits of the former typically outweigh those of the latter [15, 16]. These studies suggest that despite some usability constraints, the scalability and on-demand nature of the EC2 resources are necessary for dealing with the rising need for flexible computing, which cannot always be provided by physical resources.

2.4.3 Cost Evaluation

While there exist some cost analyses in previous work that break down the cost of using EC2 for the purposes of the experimenters' testing [1, 5], there has been little research that quantitatively evaluates the cost of physical servers versus cloud computing. This lack of research was also noted by Greenwood et al [3], who stated that there has not been much information published on the risks involved in migrating enterprise-level IT systems to the cloud. Klems et al [17] provide some basic conceptual suggestions for steps to follow when conducting cost comparisons between IT infrastructures in cloud and non-cloud environments. These include identifying business needs, comparison metrics and analyzing costs based on performance comparisons. However this report did not publish any numerical results as the study was still in its early stages of planning [3, 17]. Similarly, Bibi et al [18] also provided a framework involving the hardware and software aspects of IT infrastructure—such as business premises, driver, and software costs—which must be considered when performing a cost analysis between on-site services and cloud computing. This study was also in its planning stages, and therefore did not provide any quantitative results.

Amazon EC2 has documentation on performing cost comparisons between the AWS cloud and privately owned IT infrastructure [19]. The document describes the costs that should be considered when comparing the two types of systems, including resource usage, hardware cost, power, redundancy, security, and staffing. The aspects of this documentation that directly apply to our comparison between university servers and EC2 have been taken into consideration in the analysis detailed in this report.

There is a numerical cost comparison provided by the EC2 website itself on the differences between investing in its various instance types and dealing with a privately owned machine [20]. This analysis takes into account the costs of owning a "do-it-yourself" computer versus investing in 1-year or 3-year reserved instances at 100% capacity. The on-demand instances used for this analysis were 35 Standard Small and 10 Standard Large instances. According to Amazon, the cost of using any of these EC2 instance types is more cost-effective than purchasing and maintaining a privately owned server, with a worst-case cost difference of about 62%. It is important to note that the comparison done by Amazon in this case emulates the expenses experienced by larger parties such as corporations or businesses that purchase a large number of servers and either lease server space or collocate their servers, thereby increasing facility costs. Large server labs in turn can incur more powering and cooling expenses. This sort of comparison does not directly apply to the servers in the ECE Department at WPI, since the university owns all buildings on campus and handles server space and maintenance in-house, rather than leasing space or collocating.

Finally, Amazon also provides a Cost Comparison Calculator that compares the cost of physical IT infrastructure with EC2 [21]. Due to inaccuracies associated with the costs of using reserved instances, this calculator has not been used.

From the background information that has been covered to this point, it can be seen that there has been limited cost comparison in prior work between utilizing EC2 instances as opposed

to maintaining in-house servers for single-node computing applications. In addition, there has been no in-depth analysis between the money spent versus the time that EC2 takes to perform a benchmark (cost-to-solution vs. time-to-solution). This analysis would aid in determining the instance best suited for a certain computing need, while supplying the expected cost and time required to solve the problem. These are the gaps observed in the past literature that will be examined in further detail in this report.

2.5 The Science of Benchmarking

Benchmarks are scalable computer programs that encompass a wide range of applications, and are generally used as a means of comparing the performance of computing systems by mimicking a user's average workload [22, 23]. Benchmarks can be designed to test various components of a machine, such as CPU, memory or I/O devices. In order to compare two specific aspects, it is necessary to control all other variables to ensure the accuracy of the results. This can be accomplished by keeping software constant across systems, making it possible to measure differences in hardware performance between platforms. In the case of inconsistent software, the results obtained from a benchmark on one machine can differ greatly from another machine, simply because the version of a compiler used by that benchmark was different on the two systems. The software that must be kept constant consists of the operating systems, compilers and applications, thereby allowing a fair comparison between the different machines being benchmarked.

2.6 Benchmarks and Case Study

Given that our research focused on single-node applications, we chose CPU and memory intensive benchmarks that were not I/O or network dependent. The CPU aspect was tested by the SPEC2006 and Pystone benchmarks, while the memory was tested using the RAMSMP benchmark. A third benchmark that involved matrix manipulation in MATLAB was selected in order to evaluate both of these performance factors simultaneously. Finally, we performed a case study in MATLAB, obtained from a researcher within the WPI ECE Department, in order to simulate "a real world application" that essentially mimics the load experienced by the servers found in the ECE department.

The Pystone and MATLAB benchmarks were particularly useful for the theoretical applications of our research, as they were also be used to simulate massively parallel applications (MPAs). MPAs are essentially large programs that can be split into smaller parts, where each section can be run on a separate machine, instance or thread. Once each individual section has completed execution, the final results can be queried and compiled via network connections between the machines, providing a quicker and possibly more economical means of solving a large problem. Due to difficulties in obtaining such a program, as well as the fact that MPAs stray into the realm of networking and I/O—which is not one of the focuses of this research—it was determined that a suitable simulation could be created with the Pystone and MATLAB benchmarks. This will be explained in further detail in Section 5.

The purpose of this project was to conduct research on EC2's single-node computing capabilities without replicating the few prior studies that have done the same. Our intention was to find new information that supported the research that has already been completed on this topic, while filling some of the gaps that previous work has not covered. Due to time and monetary constraints, it was not possible to obtain some of the benchmarks used in prior work, and so most of the benchmarks used for this project (other than the SPEC2006 suite) were not taken from previous studies. Instead, they were researched by the group in order to ensure they fit the scope of our research.

2.6.1 Pystone

The Pystone benchmark, written in Python, determines how many "pystones" or iterations of the main loop of the benchmark can be completed in one second. The main code—which executes a series of functions that each perform string, integer or array manipulations—loops through ten million times during its execution. The code for this benchmark is available on the Python Core Development website [24], and is also shown in Appendix A as there were some minor manipulations made to it by the group—mainly to have the program supply us with a more consistent execution time. The importance behind selecting Pystone was its ability to provide stable and consistent CPU-intensive results.

Ideally, faster machines are able to complete this benchmark in less time. A key limitation of the Pystone benchmark is that it is not multi-threaded and thus a single instance of the process can only run on one core at a time. However, it can be executed on multiple cores simultaneously, by simply starting multiple instances of the process, which allowed it to be used to simulate a massively parallel benchmark (as was mentioned in Section 2.6). Pystone's runtime was also long enough that a useful cost estimate could be derived from the results.

2.6.2 MATLAB benchmark

The MATLAB benchmark was originally meant to be run in GNU Octave, an open-source software similar to MATLAB. However, due to issues standardizing the versions of Octave across all machines being used, the benchmark (a *.m* file) was run in MATLAB instead. This benchmark is a CPU and memory intensive benchmark that goes through three different sets of matrix manipulations. The three sets are called Matrix Calculations, Functions and Programmations respectively. The source code for this benchmark originally ran for a few seconds, which returned results that were too small in scope to compare between machines. To counter this, a variable was added to the code which acts as a multiplier for the size of the matrices. At the end of every computation, the clock time is recorded by the program in order to determine the time taken to evaluate the each step, as well as the overall time taken to run the program.

The first set, Matrix Calculations, consists of five different manipulations that the program performs on matrices of varying size. The first matrix computation involves the creation, transpose and deformation of a matrix whose size can be obtained by multiplying the variable mentioned above by the given matrix size (calculated size: 6000 x 6000). The second takes a normal-distributed random matrix (size: 3200 x 3200) and raises each element to the 1000th power. The third sorts through a matrix of random values generated by the program (size: 8,000,000). The fourth evaluates the cross-product of two matrices (size: 2800 x 2800), and the fifth evaluates the linear regression over a matrix (size: 2400 x 2400).

Once these steps have been completed, the program moves onto the second set of matrix calculations, called Matrix Functions. The first function in this set calculates the Fast Fourier Transform (FFT) over a set of random values (size: 800,000). In the second, the program calculates the eigenvalues of a matrix (size: 320 x 320). The third calculates the determinant of a matrix (size: 2600 x 2600). The fourth performs a Cholesky decomposition of a matrix (size: 3600 x 3600). Lastly, the fifth calculates the inverse of a matrix (size: 1600 x 1600).

The last set of problems also consists of five computations called as Matrix Programmations. The first goes through and calculates the Fibonacci sequence (size: 3,000,000). The second creates a Hilbert matrix (size: 9000 x 9000). The third problem calculates the grand common divisors of 70,000 pairs of numbers. The fourth creates a Toeplitz matrix (size: 880 x 880), and lastly, the fifth creates calculates Escoufier's equivalent vectors.

At the end of all of these calculations, the program provides the total amount of time taken (in seconds) for all calculations to complete. Ideally, the system will finish the benchmark in the

least amount of time possible, and the CPU performance of different machines can then be analyzed by using the total execution time.

The MATLAB benchmark was particularly important not only because it combines both main aspects of system performance (CPU and memory), but also because it effectively emulates the software that is most widely used within the WPI ECE Department. It also represents a large set of scientific calculations that deal primarily with floating-point and matrix manipulations. A detailed version of the code for the MATLAB benchmark can be found in Appendix B.

2.6.3 RAMSpeed (RAMSMP)

RAMSMP is a cache and memory intensive benchmark written in C, that determines the bandwidth of a system's various memory components. Research conducted by Stefan Appell et al [25] at the Darmstadt University of Technology utilized the RAMSMP benchmark, stating that its results were comparable to the STREAM benchmark (another memory intensive benchmark), as well as commending its capability to automate testing and collect results. This research, along with its constituent RAMSMP results, was also presented at the International SPEC Benchmark Workshop 2010 held in Germany [26].

The RAMSMP benchmark allows the user to run one of 18 memory-intensive tests at a time, each of which determines a different aspect of the system's memory performance. Although each test executes differently, they all focus on testing either the reading, writing or data manipulation bandwidth of memory operations on the system. The tests that were run using this benchmark focused on floating-point and integer operations, since the majority of scientific applications use these data types.

To test memory bandwidth with respect to calculations using large blocks of data, the FLOATmem and INTmem tests first copy either floating-point or integer data from one memory location to another. They then perform numerical operations on both sets of data including scaling, adding and a combination of these two operations, while storing results in the memory. The bandwidth obtained at the end of all of these operations shows the rate at which the memory is able to manipulate different blocks of numerical data.

When testing the reading and writing bandwidths of the system's memory, the FLOATmark and INTmark tests read and write blocks of increasing data sizes to the memory. The blocks begin at 2 kilobytes and increase exponentially by a factor of two until the maximum size of the data has been reached, which is provided by the user. The bandwidth at which each block of data was written or read from memory allows distinction between each cache level and RAM present on the system being benchmarked, as well as the bandwidth at which integer and floating-point data is typically written to memory. The fastest bandwidths are typically observed in the L1 cache, while the slowest bandwidths are prevalent when dealing with reading or writing data to and from the RAM.

One of the key limitations of the RAMSMP benchmark was that we had to be careful when specifying the size of data to be either manipulated or written to the different levels of memory. This was because it is possible to provide a block of data that exceeds the size of the physical memory present on the machine, causing the system to overload its resources and force a shut down. This could pose as a serious problem for shared servers such as Tesla and Henry that are actively used by students and faculty. Not only could the other students' usage of the servers affect the amount of free memory available for testing, but a shut down of the entire server could cause significant inconvenience or loss of data to the individuals involved. Therefore, only a small block of memory was provided for each test—small enough such that it would not hamper other processes, but large enough to write the data to RAM.

2.6.4 SPEC2006 Benchmarks

The entire suite of SPEC2006 benchmarks was obtained by the group from a faculty member at WPI. The SPEC2006 benchmark suite is designed to be a set of computationally intensive benchmarks, primarily focusing on the processor, memory system and compilers [27]. SPEC2006 consists of the CINT2006 and CFP2006 suites, which are the integer and floating-point benchmark suites respectively, altogether comprising 29 total benchmarks.

Due to compiler constraints, only the integer suite (SPEC CINT2006) and a few additional floating-point benchmarks from the floating-point suite (CFP2006) could be run. This was due to the fact that many of the floating-point benchmarks were written in the archaic programming language Fortran, which the group was not able to implement. The suite was still beneficial as the instances and machines were still tested using a widely accepted standard set of benchmarks.

It was our goal to run the SPEC suites on both the ECE servers as well as EC2, but due to lack of administrative privileges on the former, we were only able to run the benchmarks on our set of EC2 instances. Fortunately, the SPEC organization had in its repository the results obtained from benchmarking a Dell PowerEdge R610 server, which could be used in place of running the benchmarks on Tesla. However, the repository did not contain the results for the model of the Sunfire X2200 server that the ECE department uses as Henry, and so we could not include this server in our overall comparison of the SPEC results.

The SPEC CINT2006 suite consists of computationally and memory intensive benchmarks using integer data types. The types of benchmarks include an email indexer, C compiler, a gene sequence searching benchmark, and several others, all written in either C or C++. Brief descriptions for each of the benchmarks are as follows.

The 400.perlbench benchmark, written in C, consists of scripts that run spam-checking software, HTML email generation, as well as a program part of the CPU2006 tool set that uses the email generator. Next, the 401.bzip2 benchmark compresses and decompresses various files (including JPEG images, source codes, HTML files, and so on) to different compression levels, and compares the results to the original files. This benchmark is also written in C. The 403.gcc benchmark emulates a C compiler for an AMD Opteron device and by using various optimization levels, runs though a series of preprocessed C code files. The 429.mcf benchmark simulates a single depot public transportation service, including costs, scheduling, stop locations, and other such aspects. Like most of the integer benchmarks, this program is also written in C. The 445.gobmk benchmark, also written in C, plays a game of Go by executing a series of commands and then examining the results. The 456.hmmer benchmark sorts through and statistically analyzes a series of pre-provided gene sequences. Similar to the 445.gobmk benchmark, the 458.sjeng program attempts to solve a game of Chess by finding the best move given a set of conditions in a game. The 462.libquantum simulates a quantum computer by creating a basic register and gate structure. The 464.h264ref benchmark evaluates video compression for purposes such as DVDs, based on two different types of encoding techniques provided and computed by the program. The next benchmark (written in C++) is the 471.omnetpp, which simulates the functioning of a vast Ethernet network while encompassing various networking protocols as well. The 473.astar benchmark essentially acts as a path-finding algorithm, used to evaluate artificial intelligence for gaming purposes. These include various types of terrains, moving speeds, graphs, and so on, all written in C++. Finally, the 483.xalancbmk (written in C++) program converts XSL sheets into HTML or other XML forms [27].

The floating-point benchmarks in the SPEC2006 set consists of programs that are written in C, C++, Fortran, or a combination of these languages. Since we were unable to implement Fortran, the seven benchmarks that were written in C and C++ were run by the group in order to account for the use of floating-point numbers in scientific calculations. These benchmarks encompass the fields of quantum chromodynamics, linear programming optimization, fluid

dynamics, and several other computationally-intensive floating-point applications.

The first CFP2006 benchmark that was used was the 433.milc, which simulates complex four-dimensional lattice designs. The 444.namd, written in C++, is a parallelized biomolecular system generator. The 447.dealII uses a C++ program library aimed at developing algorithms, while solving through complex mathematical problems including partial differential equations. The 450.soplex simulates a linear programming problem solver, written in C++. The final C++ benchmark is the 453.povray, which simulates a camera used to observe the behavior of light in various environments and surroundings, which can be specified by the user. The 470.lbm, a C program, deals with fluid dynamics and examines the behavior of incompressible liquids and gases by using the Lattice-Boltzmann Method. Finally, the 482.sphinx3 is essentially speech recognition software developed by Carnegie Mellon University, which processes a preprogrammed set of sounds [27].

The purpose of the SPEC benchmarks is to standardize performance comparisons for different computers. In order to do so, the results output by these benchmarks are ratios obtained by comparing the performance of the machine being tested to that of a standard set of computers used by the SPEC corporation. A higher ratio indicates that the processor is better at handling the given test's data set than the baseline machine, or machines with a lower ratio for that test.

2.6.5 MATLAB Case Study

Apart from using typical benchmarks, it was important to evaluate the performance of real-world programs that would generally be run in the WPI ECE department. In order to accomplish this, a computationally intensive MATLAB program was obtained from a researcher within the ECE department which would better emulate the typical loads experienced by their servers. This program was run ten times on all EC2 instances as well as Tesla and Henry, but was not considered a "benchmark." The program was instead used along with our MATLAB benchmark, in order to test for similarities in the behavior of MATLAB across the different machines.

The program itself calculates the passing of radiation through the 3-dimensional model of a human body, essentially mimicking a body scanner. The program is provided a time frame during which it depicts a human body being subjected to electromagnetic waves that slowly move towards and through the body within a simulated period of a few nanoseconds. When running MATLAB in GUI mode, the program displays the movement of the radiation and provides plots that display the intensity thereof. The output for fractions of each simulated nanosecond are written to data (*.mat*) files. This data is then analyzed by the experimenter in more detail, which is beyond the scope of our research.

It was decided earlier on, based on the above researcher's guidance, that this MATLAB program need not be run in GUI mode despite its several visual outputs. This was due to the fact that the visual illustrations were simply for the sake of understanding what was happening at a given point in time in the program, and not crucial to the data that was obtained at the end of the program's execution. Another important reason was that the GUI mode ran extremely slowly on the Linux-based systems that we were benchmarking, which posed an issue in terms of the monetary and time constraints being faced by the group. Therefore, it was decided that the program would only be run using the command-line mode of MATLAB, and the corresponding execution times were used for the analysis.

3 Methodology

A set of viable resources and standard benchmarking techniques were determined prior to proceeding with this study. This section discusses the process used to set up the EC2 instances, a comparison between the selected operating systems as well as the manner in which the aforementioned systems were benchmarked.

3.1 EC2 Configuration

In order to benchmark EC2, Tesla and Henry, it was necessary to make all of their environments the same. There were restrictions on the compilers and software that were already installed on Henry and Tesla that prevented them from being changed; as a result, the EC2 instances were modeled after the two ECE servers. Due to a version discrepancy with the Python interpreter, an updated version had to be installed and run from a local directory on the ECE machines. A list of the compiler and software versions can be seen in Table 3.

Software	**Version on Tesla**	**Version on Henry**	**Version on EC2**
C Compiler	4.1.2	4.1.2	4.1.2
Java Compiler	4.1.2	4.1.2	4.1.2
Python Interpreter	(Actual: 2.4.3) 2.5.1	(Actual: 2.4.3) 2.5.1	2.5.1
MATLAB	2009a	2009a	2009a

Table 3: Software versions used

The instances used in this project were created in December 2010, during which period Fedora 8 was the best available alternative to Red Hat Linux 5.5. The first instance that was created was used as a template for all of the others, to ensure that each future instance would have the same starting conditions. The 64-bit version of the Fedora 8 operating system, which was pre-installed on the instance, already had GCC (the standard open-source C compiler) and the Python interpreter. It did not, however, have the appropriate compilers for Java or C++, which were installed by the group. Finally, after installing MATLAB 2009a on the server, the cloning process to duplicate the instance template was initiated via the AWS console. Once the image creation was complete, six copies of the primary instance were created on each possible 64-bit instance. A detailed description of the steps followed when we created the instances on Amazon EC2 can be found in Appendix C.

3.2 Determination of the Performance Comparison Ratio

At the time that this analysis began, Fedora 8 was one of the standard operating system options on EC2. Both Fedora 8 and Red Hat Linux Enterprise 5.5 are designed by the Red Hat Corporation, and because of complications in setting up a Red Hat license on the EC2 servers, it was decided that Fedora 8 would be our target operating system. Although the two operating systems are both created by the same company, there are small dissimilarities between them—and it was for this reason that a Performance Comparison Ratio (PCR) was designed.

3.2.1 Purpose

The purpose of the PCR segment of this project was to minimize the differences between the two test systems (Red Hat and Fedora) by determining an accurate ratio of their relative performances for a given benchmark. Ideally, the results produced by the benchmarks on each system should not be affected by the fact that they ran on different operating systems. Therefore, the difference in performance between the two operating systems was quantitatively evaluated and used to compare the results obtained on EC2 and Tesla.

For each benchmark that was run on EC2 and Tesla (or Henry), over each trial j where $j = 1, 2, 3, \ldots n$, the resulting PCR was determined by the following formula, where $RH(j, i)$ and $F(j, i)$ are the functions that return the jth result for the ith benchmark for Red Hat and Fedora respectively:

$$PCR(i) = \frac{\frac{1}{n}\sum_{j=1}^{n} RH(j,i)}{\frac{1}{n}\sum_{j=1}^{n} F(j,i)} \tag{1}$$

This would give us a ratio in terms of Red Hat over Fedora, which means that if we multiply our Fedora results by the PCR, we will receive a ratio of Red Hat over Red Hat—leaving us with a "pure" ratio of their performance. To account for the differences between bandwidth rates and execution times—where a lower value for the former and a higher value for the latter are indicative of better performance—which are the two main types of results that our benchmarks returned, we used the inverse of Equation 1 handle rates. This gave a PCR value which represented the true difference in performance between operating systems, regardless of the data type. We then divided, rather than multiplied, the Fedora results by the PCR to adjust them.

For example, during the computation of the PCR for "Benchmark A," if Tesla or Henry (running Red Hat) were to observe a final execution time of 100s for the benchmark, and EC2 (running Fedora) was to observe a time of 100s as well, multiplying the Fedora result by the PCR for the benchmark—say the PCR was 0.95 in this case — would yield a final ratio of $\frac{100}{100 \times 0.95} = \frac{100}{0.95} = 1.0526$. This would mean that Tesla's (or Henry's) results were approximately 5.26% better than EC2's for "Benchmark A". The average PCR for each benchmark was multiplied with the values of our final results in order to yield a PCR-adjusted value — thereby standardizing all benchmarking results regardless of the operating system.

3.2.2 Methods

It is possible for the PCR to vary depending on the hardware specifications of the machine that it is derived from. In order to get around this issue, two separate machines with differing hardware specifications were chosen to run the benchmarks. Table 4 illustrates the hardware specifications of the two computers used for calculating the PCR.

System	CPU	RAM	CPU Type
Computer 1	3.0 GHz Dual Core	8GB DDR2	Intel
Computer 2	2.99 GHz Core 2 Duo	4GB DDR2	Intel

Table 4: System Specifications

By holding the hardware constant for both operating systems, it was possible to gather

data that directly compares the performance of both operating systems. Our approach was to install Fedora 8 and Red Hat 5.5 on one hard drive and use this hard drive for each test computer. When setting up the hard drive, it was necessary to create specific Linux partitions for each operating system, with each requiring its own swap, boot, and home partitions. Red Hat 5.5 and Fedora 8 were installed on the drive, and all of the system's libraries were updated to the most recent versions using the Red Hat / Fedora shell command: *yum update all*.

It was important to ensure that no extraneous processes were running in the background while the benchmarks were executed, as they would directly impact our results. To overcome this issue, the systems were booted directly into terminal mode from the GRUB menu, which cut out all of the GUI processes that could affect the performance of the system by congesting the CPU and memory. Fedora and Red Hat are both capable of booting directly into terminal mode from the GRUB boot loader. This was accomplished by creating a boot command in GRUB that makes the "runlevel" for the operating system to be "text only" as opposed to starting the operating system with the entire GUI loading when it boots. The command used is shown below. The '3' at the end of the command signifies that the operating system should boot in "runlevel" 3, which makes it boot directly into terminal mode.

```
kernel /vmlinuz-2.6.23.1-42.fc8 ro root=LABEL=/1 3
```

Linux shell scripts were written to streamline the process of executing benchmarks. The benchmarks were run by creating a loop that executed the benchmark on each iteration and dumped the output to a file. The following shell script is one of the scripts used to run the RAMSMP Float Reading benchmark, and demonstrates how the benchmarks were executed 50 times.

```
j=1
for ((j = 1; j <= 50; j++)
do
   echo "RAMSMP: 5 @ 1024:+ Test $j Started +:"
   time ./ramsmp -b 5 -m 1024 >> Results/Fedora8/ResultRamSMPFloatRead.txt
   echo "Test $j Done"
done
```

Due to variances amongst operating systems, such as kernel optimizations, that could affect each benchmark in different ways, it was necessary to calculate the ratios for each benchmark separately. This allowed for a more precise understanding of any differences that may have arisen in terms of the way Red Hat and Fedora handle the same test.

3.2.3 Results

In order to produce a ratio of the two operating systems, the results for Red Hat 5.5 were divided by the results for Fedora 8 for each benchmark. The ratios obtained are shown in Table 5 below.

i	Benchmark	PCR on Computer 1	PCR on Computer 2	Percent difference
1	Pystone	0.908	0.909	0.044
2	MATLAB Benchmark	1.060	1.058	0.216
3	RAMSMP FLOATmem	1.023	1.008	1.505
4	RAMSMP Float Reading (average)	0.996	0.993	0.295
5	RAMSMP Float Writing (average)	1.002	0.992	0.952
6	RAMSMP INTmem	1.019	1.005	1.409
7	RAMSMP Int Reading (average)	0.996	0.996	0.035
8	RAMSMP Int Writing (average)	1.003	0.998	0.485
9	SPEC			
int	400.perlbench	0.991	1.000	0.893
int	401.bzip2	1.000	1.006	0.576
int	403.gcc	1.348	1.340	0.528
int	429.mcf	0.961	0.983	2.244
int	445.gobmk	0.995	1.000	0.542
int	456.hmmer	0.991	0.983	0.881
int	458.sjeng	1.000	1.000	0.000
int	462.libquantum	1.037	1.010	2.722
int	464.h264ref	1.012	1.008	0.387
int	471.omnetpp	0.992	0.985	0.772
int	473.astar	1.000	1.008	0.791
int	483.xalancbmk	0.990	0.985	0.512
float	433.milc	0.970	0.969	0.095
float	444.namd	1.000	1.000	0.000
float	447.dealII	1.004	0.996	0.765
float	450.soplex	1.075	1.028	4.434
float	453.povray	1.034	1.019	1.478
float	470.lmb	1.005	1.055	4.852
float	482.sphinx3	1.045	1.013	3.120
10	Case Study	1.025	1.025	0.000

Table 5: Overall Summary of PCR Results

The above results demonstrate the almost negligible differences between Red Hat 5.5 and Fedora 8 across each benchmark, with the majority of the results differing by less than 1.5%. A few of the SPEC benchmarks extend beyond this range, but none differ by more than 5%. The overall PCR for each benchmark is an average of the values found on each system, which accounts for differences across both operating systems and hardware types. The PCRs obtained for each benchmark are analyzed in greater detail in the subsequent sections.

3.2.3.1 Pystone Results

From Table 6, it can be seen that the standard deviations observed for the Pystone benchmark were less than 2% in the case of both computers, which showed that the benchmark

runs fairly consistently on a single machine. The two PCRs were extremely close to each other, signifying that the benchmark results were reproducible across different computers as well. The PCRs show the general trend that Red Hat runs the Pystone benchmark considerably faster—almost 10%—than Fedora 8, which will be taken into account when analyzing the Pystone results obtained from the EC2 instances running Fedora.

	Red Hat			**Fedora**			
Computer 1							
Result	**Average**	**STDEV**	**% STDEV**	**Average**	**STDEV**	**% STDEV**	**PCR**
Execution Time (Seconds)	106.778	1.766	1.654	117.429	1.553	1.322	0.909
Pystones per Second	94845.205	1540.567	1.624	86034.022	1139.796	1.325	0.907
						Average:	**0.908**
Computer 2							
Result	**Average**	**STDEV**	**% STDEV**	**Average**	**STDEV**	**% STDEV**	**PCR**
Execution Time (Seconds)	107.470	1.797	1.672	118.131	1.873	1.585	0.910
Pystones per Second	94263.764	1560.671	1.656	85538.758	1345.585	1.573	0.907
						Average:	**0.909**

Table 6: Pystone Benchmark PCR Results

3.2.3.2 MATLAB benchmark Results

As we can see from Table 7, the standard deviation for the MATLAB results was less than 1% for both Fedora and Red Hat, demonstrating the consistency of the benchmark as a whole. The average PCR from the two machines, however, was almost 6%, which indicates that the MATLAB script was executed noticeably faster by Fedora than by Red Hat. This is contrary to what was observed in the case of Pystone, where Red Hat finished the benchmark significantly faster than Fedora. This difference was important to take into account when we compared the overall results obtained from running this benchmark, as it was possible for the EC2 instances to run these programs faster simply by virtue of their operating system.

Red Hat			Fedora			
Average Execution Time (Seconds)	**STDEV**	**% STDEV**	**Average Execution Time (Seconds)**	**STDEV**	**% STDEV**	**PCR**
Computer 1						
12.876	0.050	0.391	12.144	0.063	0.522	1.060
Computer 2						
12.967	0.041	0.313	12.257	12.257	0.064	1.058
					Average:	**1.059**

Table 7: MATLAB Benchmark PCR Results

3.2.3.3 RAMSMP FLOATmem Results

The RAMSMP FLOATmem benchmark tests a system's performance with respect to memory bandwidth by completing mathematical operations on floating-point matrices stored in memory. Table 8 contains a summary of all the FLOATmem results with their respective PCRs. The ratios that were calculated for both Computer 1 and Computer 2 are very close to one another, with a relative standard deviation of about 1.32%, and an average PCR of 1.015. As was the case with previous benchmarks, this shows consistency on and across each system, and that the differences between Red Hat and Fedora in this case were almost negligible.

Red Hat			Fedora			
Average Bandwidth (MB/s)	**STDEV**	**%STDEV**	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**PCR**
Computer 1						
4522.771	5.527	0.122	4627.131	2.410	0.052	1.023
Computer 2						
4320.456	3.121298	0.072	4354.504	3.256174	0.075	1.008
					Average:	**1.015**

Table 8: RAMSMP FLOATmem PCR Results

3.2.3.4 RAMSMP Float Reading Results

The Float Reading benchmark tests a system's performance with respect to memory access bandwidths by reading stored floating-point matrices that are stored in memory. Table 10 illustrates the results obtained in the PCR calculation process. Due to the nature of the Float Reading results, the geometric mean of the standard deviations was used in place of the normal standard deviation, to give a more accurate representation of the actual data set. The resulting PCR values for the Float Reading benchmark in Table 10 are extremely consistent across both computers, with a relative standard deviation of only 0.2%. The average PCR itself, 0.994, indicates that the two operating systems show almost identical performance for this benchmark.

Red Hat			Fedora			
Average Bandwidth (MB/s)	**Average STDEV**	**Average %STDEV**	**Average Bandwidth (MB/s)**	**Average STDEV**	**Average %STDEV**	**PCR**
Computer 1						
18355.120	64.750	0.353	18276.268	22.302	0.122	**0.996**
Computer 2						
18356.068	31.164	0.170	18223.048	23.617	0.122	**0.993**
					Average:	**0.994**

Table 9: RAMSMP Float Reading PCR Results

3.2.3.5 RAMSMP Float Writing Results

The RAMSMP Float Writing benchmark tests a system's performance with respect to memory speeds by writing floating-point data to various sizes of memory. As was the case with the previous RAMSMP benchmark, the PCRs for both systems were extremely similar, with an average of 0.997, as seen in Table 10. This further demonstrates the similarities across the operating systems, especially in regards to floating-point memory operations.

Red Hat			Fedora			
Average Bandwidth (MB/s)	**Average STDEV**	**Average %STDEV**	**Average Bandwidth (MB/s)**	**Average STDEV**	**Average %STDEV**	**PCR**
Computer 1						
10925.590	25.813	0.236	10942.719	62.746	0.122	**1.002**
Computer 2						
10755.972	40.435	0.376	10666.934	64.684	0.122	**0.992**
					Average:	**0.997**

Table 10: RAMSMP Float Writing PCR Results

3.2.3.6 RAMSMP INTmem Results

RAMSMP INTmem stores integer values in arrays in memory and then performs mathematical operations on those arrays to determine the overall average memory bandwidth for these operations. The results from running INTmem on Computer 1 and Computer 2 show that Red Hat was between 0.5% and 2% slower than Fedora for integer memory operations, resulting in a PCR of 1.012. The relative standard deviation between computers was only 1%, which further illustrates the accuracy of these results.

Red Hat			Fedora			
Average Bandwidth (MB/s)	**STDEV**	**%STDEV**	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**PCR**
Computer 1						
3985.977	6.868	0.172	4060.879	2.556	0.063	1.019
Computer 2						
3792.181	1.262	0.033	3809.450	1.176	0.031	1.005
					Average:	**1.012**

Table 11: RAMSMP INTmem PCR Results

3.2.3.7 RAMSMP Int Reading Results

When the RAMSMP Int Reading benchmark is run, it returns the bandwidth for reading different sized blocks of integers in memory. The average PCR over both systems was 0.996, which indicates that there is no practical difference in operating systems, or across the hardware, for reading integer values from memory.

Red Hat			Fedora			
Average Bandwidth (MB/s)	**Average STDEV**	**Average %STDEV**	**Average Bandwidth (MB/s)**	**Average STDEV**	**Average %STDEV**	**PCR**
Computer 1						
18350.542	39.220	0.214	18283.736	52.097	0.114	**0.996**
Computer 2						
18407.856	35.701	0.194	18332.991	91.457	0.114	**0.996**
					Average:	**0.996**

Table 12: RAMSMP Int Reading PCR Results

3.2.3.8 RAMSMP Int Writing Results

The RAMSMP Int Writing benchmark results reflect the bandwidth at which each system can write integers to memory. The average PCR was 1.000, which implies that the results were consistent across each operating system. Although the individual PCRs were reverse ratios, their relative standard deviation was 0.35%, which indicates that they are extremely consistent. The results for this test are shown in Table 13.

Red Hat			Fedora			
Average Bandwidth (MB/s)	**Average STDEV**	**Average %STDEV**	**Average Bandwidth (MB/s)**	**Average STDEV**	**Average %STDEV**	**PCR**
Computer 1						
11776.319	28.034	0.238	11805.731	25.099	0.114	**1.003**
Computer 2						
11486.470	9.483	0.083	11458.292	138.774	0.114	**0.998**
					Average:	**1.000**

Table 13: RAMSMP Int Writing PCR Results

3.2.3.9 SPEC Results

The SPEC2006 benchmark suite gives a user the option of selecting an operating system-specific tool set when installing the benchmarks. This is meant to reduce the overhead associated with running the benchmarks on different operating systems. However, for the sake of ensuring consistency across all of our results, we ran each of the SPEC benchmarks on our PCR machines as well; almost every PCR obtained was identical across both machines. The largest percent difference between the two sets of PCRs for the SPEC CINTtests was approximately 3.500%, which demonstrates the extreme accuracy of the benchmark suite. It is important to note that the 403.gcc benchmark, which emulates an optimizing C compiler, experienced a large difference between Red Hat and Fedora (almost 340%). This difference was consistent across both machines, with the difference in PCRs for this test being about 0.500%. This PCR could be attributed to the fact that the benchmark is designed to generate code for an AMD Opteron processor and all of the machines used in this study used Intel processors. The SPEC floating-point benchmarks were slightly less consistent on both computers, with two of the tests reaching a percent difference of about 4.500%. The average percent difference of the entire suite was approximately 1.200%, and the median was 0.752%, which further supports the consistency of these benchmarks. Tables 14 and 15 show the complete set of integer and floating-point benchmarks respectively.

	Computer 1			Computer 2		
Test	**Fedora Ratio**	**Red Hat Ratio**	**PCR**	**Fedora Ratio**	**Red Hat Ratio**	**PCR**
400.perlbench	22.5	22.3	0.991	22.3	22.3	1.000
401.bzip2	17.4	17.4	1.000	17.3	17.4	1.006
403.gcc	14.1	19	1.348	14.1	18.9	1.340
429.mcf	18.1	17.4	0.961	17.8	17.5	0.983
445.gobmk	18.5	18.4	0.995	18.4	18.4	1.000
456.hmmer	11.5	11.4	0.991	11.5	11.3	0.983
458.sjeng	18.3	18.3	1.000	18.2	18.2	1.000
462.libquantum	21.4	22.2	1.037	21	21.2	1.010
464.h264ref	25.6	25.9	1.012	25.6	25.8	1.008
471.omnetpp	13.3	13.2	0.992	13.2	13	0.985
473.astar	12.8	12.8	1.000	12.6	12.7	1.008
483.xalancbmk	19.8	19.6	0.990	19.8	19.5	0.985
		Average:	**1.026**		**Average:**	**1.026**
		Without GCC:	**0.997**		**Without GCC:**	**0.997**

Table 14: SPEC CINT2006 PCR Results

	Computer 1			Computer 2		
Test	**Fedora Ratio**	**Red Hat Ratio**	**PCR**	**Fedora Ratio**	**Red Hat Ratio**	**PCR**
433.milc	13.4	13	0.970	13	12.6	0.969
444.namd	15.1	15.1	1.000	15	15	1.000
447.dealII	26.2	26.3	1.004	26.1	26	0.996
450.soplex	17.4	18.7	1.075	17.8	18.3	1.028
453.povray	20.4	21.1	1.034	20.9	21.3	1.019
470.lbm	20.1	20.2	1.005	18.2	19.2	1.055
482.sphinx3	22	23	1.045	22.5	22.8	1.013
		Average:	**1.019**		**Average:**	**1.012**

Table 15: SPEC CFP2006 PCR Results

3.2.3.10 MATLAB Case Study Results

The PCRs obtained for our MATLAB case study are shown in Table 16. The difference between the two PCRs is almost negligible, and the standard deviations of the 10 runs on both computers is less than 1.5%. These results show that not only is the PCR reproducible for this particular program, but also that the case study tends to run consistently on different kinds of machines. We note that the results from the case study PCR are different from those for the MATLAB benchmark PCR, with the difference between the two operating systems for the former being approximately 2.5%, compared to approximately 6% for the latter. This may be attributed

to the different forms of executions or functions called in each program—the MATLAB benchmark runs purely mathematical operations, while the case study runs more complex, 3-dimension simulating algorithms. However, the general trend of Red Hat running MATLAB slower than Fedora is once again seen in these results, which must be taken into consideration when evaluating the results of both MATLAB programs in our final analysis.

Red Hat			**Fedora**			
Average Execution Time (Seconds)	**STDEV**	**% STDEV**	**Average Execution Time (Seconds)**	**STDEV**	**% STDEV**	**PCR**
Computer 1						
259.4906	3.0803	1.1871	253.0496	3.2384	1.2797	1.025
Computer 2						
272.3272	2.1640	0.7946	265.7577	2.8922	1.0883	1.025
					Average:	**1.025**

Table 16: Case Study PCR Results

3.3 Procedure

In this section we will describe in detail the procedure that was followed while benchmarking each of our machines and instances. Additionally, the methods used in the performance comparison as well as cost analyses have also been described.

3.3.1 Benchmarking

Prior to documenting our results, we must first discuss the process that was used to benchmark each instance with our set of benchmarks.

3.3.1.1 Pystone and MATLAB benchmarks

The most efficient and consistent way to run a series of benchmarks is to construct a shell script that runs each test in an automated loop. Our scripts were designed to run each benchmark 50 times and pass the results from each iteration to a text file. The UNIX "time" command was also used to record the overall execution time of each program, which was passed into a different text file. The scripts that handled the Pystone and MATLAB benchmarks only needed to control the number of times that each benchmark would be run, since neither benchmark received command-line arguments. An example of the Pystone shell script is shown below.

```
echo "Pystone benchmark for EC2 Large instance" >>
   /home/final_results/Pystone.txt
for (( i = 1; i < 51; i++))
do
  echo "Pystone benchmark $i Started +:" >> /home/final_results/Pystone.txt
  (time python pystone.py) >> /home/final_results/Pystone.txt 2>> /home/\
final_results/Pystone_time.txt

  echo "Test $i Done"
done
```

The main focus of these two benchmarks was on their runtime, so clock time—or real-world time—was originally calculated based on the results of the "time" command. However, the source code for the Pystone and MATLAB benchmarks (Appendices A and B) was later modified so that it would output the clock time taken by each test at the end of its execution—thereby eliminating the need to use the "time" command.

3.3.1.2 RAMSMP

While the RAMSMP benchmark suite offers a variety of different tests, our main focus for the scope of this project was on integer and floating-point operations. A total of 6 tests from the suite were chosen, making up our RAMSMP test set: Floating-point Reading/Writing (FLOATmark Read/Write), Integer Reading/Writing (INTmark Read/Write), Integer Memory (INTmem) and Floating-point Memory (FLOATmem). As with the Pystone and MATLAB benchmarks, each of these tests had a corresponding shell script that was used to execute it.

The benchmark suite was initially acquired from its source website [28]. The entire set of source code was contained in a compressed file, along with licensing information and the relevant README file. Upon extracting the container, the suite had to be built and installed by running the install script, "build.sh." By executing the benchmark suite with no additional command-line arguments—ie. issuing the command *./ramsmp* in the suite's root directory—it was possible to view which tests were available. The output was as follows:

```
RAMspeed/SMP (Linux) v3.5.0 by Rhett M. Hollander and Paul V. Bolotoff, 2002-09

USAGE: ramsmp -b ID [-g size] [-m size] [-l runs] [-p processes]
-b  runs a specified benchmark (by an ID number):
     1 -- INTmark [writing]          4 -- FLOATmark [writing]
     2 -- INTmark [reading]          5 -- FLOATmark [reading]
     3 -- INTmem                     6 -- FLOATmem
     7 -- MMXmark [writing]         10 -- SSEmark [writing]
     8 -- MMXmark [reading]         11 -- SSEmark [reading]
     9 -- MMXmem                    12 -- SSEmem
    13 -- MMXmark (nt) [writing]    16 -- SSEmark (nt) [writing]
    14 -- MMXmark (nt) [reading]    17 -- SSEmark (nt) [reading]
    15 -- MMXmem (nt)               18 -- SSEmem (nt)
-g  specifies a # of Gbytes per pass (default is 8)
-m  specifies a # of Mbytes per array (default is 32)
-l  enables the BatchRun mode (for *mem benchmarks only),
    and specifies a # of runs (suggested is 5)
-p  specifies a # of processes to spawn (default is 2)
-r  displays speeds in real megabytes per second (default: decimal)
-t  changes software prefetch mode (see the manual for details)
```

The set b of benchmarks that we chose to execute, as previously mentioned, was therefore the following: $b = \{1, 2, 3, 4, 5, 6\}$. An example of the command used in order to run the Floating-point Writing benchmark, as was used on Tesla, can be seen below. Each number in the command is preceded by a letter that denotes which parameter it modifies; the list of possible parameters can be seen in the output above.

```
./ramsmp -b 4 -m 512 -p 8
```

The array size supplied to each benchmark was kept constant across each test system, but the number of processes was modified so that each system would use its maximum number of physical cores. For example, 8 processes were used in the case of the HM4XLarge instance, since it has 8 cores, while 2 processes were used for the Large instance, which has 2 cores; the array sizes across both systems, however, were 512MB for the Reading and Writing tests, and 256MB for the INTmem and FLOATmem tests. In this way, it was possible to ensure that each system would write to RAM but would not exceed its total amount. All other variables were left as default.

3.3.1.3 SPEC2006 Benchmark Suite

We followed the instructions found on the SPEC website that detailed the installation of the suite for a Linux system. To expedite the process of sending all of the benchmarks on the DVD to the EC2 instances, a compressed *bzip2* file was used, which allowed for much faster data transfer. The *bzip2* file was extracted to the local directory by using the command:

```
tarxjvf cpu2006.tar.bz2
```

The suite was then installed using the following command:

```
./install.sh -e linux-suse101-AMD64 linux-suse101-i386
```

As was previously mentioned, during the installation process for the SPEC2006 suite, the user can select an operating system that is most like the one it is being installed on in order to optimize the execution. For our instances, we chose *"linux-redhat62-ia32"* for x86, IA-64, EM64T, and AMD64-based Linux systems with GLIBC 2.1.3+. The other two system choices, *"linux-suse101-AMD64"* and *"linux-suse101-i386"* were therefore manually excluded from the installation process. The configuration was then finally completed using the command *"./shrc"*.

The following command was obtained from the installation documentation, and was used in order to run each benchmark, depending on the name given to it by the suite.

```
runspec --config=Config.cfg --size=ref --noreportable --iterations=1 (benchmark)
```

This command was used to run all of the constituent benchmarks of the SPEC2006 suite. As mentioned earlier, the entire integer benchmark suite was run on all the EC2 instances, while only a few of the floating-point benchmarks could be run due to the absence of Fortran on our systems. It is also important to note that installing and running the SPEC2006 benchmarks required administrative privileges on the ECE department servers, which is against department policy for students to obtain; as a result we were unable to run these benchmarks on Tesla and Henry. In Section 2.6.4 we mentioned that while we were able to find both CINT2006 and CFP2006 results for the Dell PowerEdge R610 server in the SPEC results repository, we were unable to find CFP2006 results for the Sunfire X2200 M2 server. Therefore, the server equivalent for Henry has been excluded from the SPEC results.

The results output by the SPEC benchmarks were essentially ratios that were obtained by comparing the benchmarked system's performance against the baseline machines that the SPEC organization used. A higher ratio indicates better performance, and thus a more powerful machine in terms of its computing power [23, 27].

3.3.1.4 MATLAB Case Study

The case study that was obtained from the WPI ECE Department was run on each instance and machine a total of 10 times, and, as with many of our benchmarks, a shell script was employed to do so. Due to the "real-world" nature of this program, and the analysis that we performed thereof, we chose to only run the case study 10 times on each system. The program was run through MATLAB's command line mode, which was accomplished by using the following command:

```
matlab -nodisplay -nosplash -r (case study filename)
```

The case study consisted of a few individual MATLAB files of type *.m* and *.mat*. In order to run the main function 10 times through a shell script, we had to modify the MATLAB code to allow the program to exit once it was completed. As such, a new *.m* file was created that only had the name of the function we needed to execute, followed by "exit." This file was then used in the shell script created to run the program 10 times, so that each time the program ran, it would open MATLAB, execute, and exit out of MATLAB. This allowed each run of the test to be isolated from every other run, and ideally should have minimized the effects of data and instruction caching on the results.

The output of the program itself was the execution time for every step of the program. By taking the sum of these values, we were able to compute its total execution time. As with all time-based results, the machine that provided the smallest execution time had the best performance and therefore best handled our simulation of a real world application.

3.3.2 Analysis

In this section we will cover the process used to analyze all of the benchmarking results, as well as a brief explanation of the cost analysis.

3.3.2.1 Benchmarks

The performance for the Pystone and MATLAB benchmarks was based on the average clock time taken to run each test. The PCR-adjusted average of their execution times and their relative standard deviations were used to compare not only the average performance of the systems, but also any inconsistencies that were discovered. The machine that executed the program in the smallest amount of time therefore had the best performance.

While the Pystone benchmark also returns a value for the processing power per second (Pystones/second), the MATLAB benchmark does not. For the purposes of an overall analysis, we required the values to be in the same relative dimension—in this case, processing power per second—so we calculated an additional value for the MATLAB benchmark: tests per second. For this value, we simply used the total number of tests, $I = 15$, and divided it by the total execution time, T, giving us the processing power $P = \frac{I}{T}$.

For the RAMSMP benchmarks, the values returned were always the bandwidths associated with the targeted levels of memory. By using the average of these numbers, we were able to give an accurate representation of the different cache and RAM bandwidths achieved by the systems. In the case of the INTmem and FLOATmem benchmarks, the average bandwidth for each run was already calculated and returned by the programs. Since the results from the different tests in the RAMSMP suite were used in different ways, the values that they returned also had to be handled differently; the Read/Write bandwidths were used to graphically represent the different performances of the systems' cache levels, while the INTmem and FLOATmem tests were used to

give a big-picture representation of the system. As a result of this, the average bandwidths that were obtained from the latter two benchmarks across the total set of 50 runs were averaged and used as a general comparison. The machine with the highest bandwidth has the fastest cache and RAM speeds for integer or floating-point data manipulation.

The INTmark and FLOATmark benchmarks were examined by taking the average for each individual data point obtained. These values, along with their standard deviations, were used in order to differentiate the various cache levels present on the machine being benchmarked. To verify our findings, the cache information was also obtained from the systems themselves. The machine with the highest bandwidths once again demonstrated the best performance, although this was dependent on the level of cache.

The SPEC benchmarks returned both the ratio of the test system against the baseline machine as well as the total execution time for the benchmark. A larger ratio indicated better performance against the baseline, while a smaller execution time represented the same. The ratios obtained from these benchmarks were graphed and compared alongside Tesla's results in order to illustrate their relative performances.

Finally, the case study was analyzed in a manner similar to the MATLAB benchmark. This program, as mentioned earlier, ran 10 times and then its average execution times and overall standard deviations were used to analyze the data set. The machine or instance that had the smallest execution time with the least amount of standard deviation demonstrated the best performance overall.

With the analysis complete, the next task was to determine the Cost vs. Time to solution comparison. The Pystone and MATLAB benchmarks were used in order to evaluate the time versus dollar to solution analysis, as they can also be used to simulate massively parallel programs. This means that the cost vs. time solution can be estimated for both single and multi-threaded applications using these benchmarks. The analysis compared the number of hours versus the cost of running 1000 runs of each benchmark. This was used to find which instance would provide the quickest solution at the lowest cost for each type of benchmark.

Each of the benchmarks took between one and three minutes to fully run once. This was not a long enough period of time that would allow us to reasonably estimate the hourly cost of running such programs. In order to overcome this issue, the analysis was carried out by calculating the amount of time and money it would take to run each benchmark 1000 times, which would allow a large enough figure for both time as well as money. A detailed explanation of the cost versus time to solution portion of this report has been shown in Section 5.

3.3.2.2 Costs

In order to determine the overall practicality of choosing EC2 over in-house servers, a comprehensive analysis was conducted on all of the costs associated with purchasing and maintaining the current servers in the ECE department at WPI. These costs included the cost of purchasing, powering, and cooling all of the servers in the department. After speaking with a member of the department who maintains these servers, we determined that unlike average corporate servers, the lifespan of the ECE servers was around nine years. Also, all server maintenance is completed in-house in our department, so the costs that are typically associated with maintaining server infrastructure were not taken into account. Using this as our projected lifespan, the total cost of owning all of the servers was computed for the duration.

The next step was to calculate the cost of replacing all of the current servers with the most comparable EC2 instances over the same period of time. Using the reserved pricing for instances on EC2, we made a nine-year cost projection. What this cost analysis does not take into account is that the servers currently housed at WPI were all purchased when they were top of the line. It

is also safe to assume that Amazon will eventually upgrade their server hardware and lower the price per ECU for current instances.

We did not take software costs into account in our analysis because these will likely be the same in either scenario. The relevant analysis in this case is to find out if investing in EC2 instances for a period of 10 years or more would be more economically feasible than purchasing and utilizing physical servers for the same amount of time. In order to effectively evaluate this standpoint, the cost of purchasing and maintaining each of the primary ECE department servers was first determined. Following this, the number of instances that equated to each server being used in the department were determined, and their corresponding reserve costs were calculated. For example, one Tesla server is more or less equivalent to one HM4XLarge instance. Therefore, the cost of reserving one instance of the HM4XLarge was used to compare against the cost of purchasing and maintaining Tesla. Additionally, this analysis was extended to a period of 9 years, which is typically how long the ECE department maintains the servers that it purchases. The total costs would help in determining whether EC2 is advisable in the long run for well established research institutions such as WPI, that require a large number of servers to deal with their computational needs.

4 Results

As stated in Section 3, each benchmark was run a total of 50 times and the performance metrics for each benchmark were evaluated. In addition to the raw average results, we have also included two PCR-adjusted values: the averages and a ratio comparing each system's performance against Tesla.

4.1 Pystone Benchmark

The Pystone benchmark provides for an excellent comparison of the processing power of each system, based on both the execution time and number of Pystones per second that are returned. Due to the synthetic nature of "Pystones," their performance is in every way identical to the average execution time. Lower execution times indicate better performance by the test system. Table 17 displays the average results obtained from the entire set of machines that were benchmarked.

System	Average Single-Threaded Execution Time (seconds)	Relative Standard Devation of Execution Time (%)	PCR-Adjusted Average Time	PCR-Adjusted Performance Compared to Tesla
Tesla	91.811	0.531	-	1.000
Henry	135.56	1.09	-	0.677
Large	175.375	2.35	159.311	0.576
XLarge	174.947	2.396	158.922	0.578
HMXLarge	126.347	0.939	114.774	0.800
HM2XLarge	126.053	0.672	114.507	0.802
HM4XLarge	126.235	1.027	114.672	0.801
HCXLarge	151.521	2.414	137.642	0.667

Table 17: Summary of Pystone Benchmark Results

From the above table, it was calculated that Tesla outperformed the fastest instance (in this case, the HM2XLarge) by approximately 20%. The standard deviations observed by all of the test systems were extremely low, which supports the consistency of this data set. Henry performed significantly worse than both Tesla and the high-memory instances. Finally, the slowest instances were the Large and XLarge, which took nearly twice as long to complete as Tesla.

In addition to showing that Tesla is superior with CPU-intensive processing, our results also display three distinct tiers of performance: the standard instances are the worst, followed by the HCXLarge in the middle, and then high-memory instances at the top. Even though the three high-memory instances vary in cost, their processing power is almost identical. We also note that the HCXLarge instance does not appear to perform as well as the other instances despite the computational nature of this benchmark. While the high-compute instance may have a much larger number of ECUs, they are spread out over 8 cores—leaving fewer ECUs per core. Since Pystone is a single threaded benchmark, the HCXLarge is not able to make full use of its 7 additional cores, so it hangs up in the execution. Along with the results from the high-memory instances, the results for the high-compute instance will be explored in further detail in Section 5.

4.2 MATLAB benchmark

The completion times obtained from running the MATLAB benchmark on Tesla, Henry, and the EC2 instances are shown in Table 18. As in the case of the Pystone benchmark, better performance is determined by finishing the benchmark with a lower average execution time.

System	Average Single-Threaded Execution Time (seconds)	Relative Standard Devation of Execution Time (%)	PCR Adjusted Average Time	PCR-Adjusted Performance Compared to Tesla
Tesla	9.225	2.915	-	1.000
Henry	16.571	1.912	-	0.557
Large	22.544	2.663	23.875	0.386
XLarge	23.460	6.880	24.845	0.371
HMXLarge	16.248	15.539	17.207	0.536
HM2XLarge	14.991	6.666	15.876	0.581
HM4XLarge	14.086	13.852	14.918	0.618
HCXLarge	18.592	9.193	19.690	0.469

Table 18: Summary of MATLAB benchmark Results

The above results once again demonstrate Tesla's overall superiority of processing power. The next best system, HM4XLarge, is outperformed by nearly 40%, which is a significant increase from the Pystone results. This increase between performances suggests that Tesla is far superior to the EC2 instances at undertaking heavy matrix calculations, which are both computationally and memory intensive. Both HMXLarge and HM4XLarge experienced a large standard deviation, demonstrating the significant variabilty of performance on EC2. While the HM4XLarge had the best average execution time, it also had the second highest standard deviation, which could have pushed its results in either direction. Given a best case scenario, the HM4XLarge could not have achieved an execution time under $14.086 \times \frac{13.852}{100} = 12.135$ seconds, which would only have been 0.760, or 76%, of Tesla's performance.

As in the case of the Pystone results, all of the high-memory instances appear to have almost identical performance for this particular benchmark, along with a considerable amount of standard deviation. We also note that the HCXLarge instance once again does not appear to perform as well as the other instances despite its large amount of computing power. This may also be attributed to the fact that this benchmark runs on only a single core of the machine, which has been examined later in this report. Finally, the Large and XLarge instances continue to perform worst than the rest of the rest of the systems, with average execution times around 62% worse than Tesla's.

4.3 RAMSMP

The RAMSMP benchmark consisted of various memory tests that allow for the analysis of the systems' cache, memory levels and bandwidths. These tests consisted of the reading and writing as well as manipulation of both integer and floating-point numbers. For this benchmark, the recorded bandwidths represent how fast each system is able to perform specific memory operations, where higher rates are indicative of better systems.

4.3.1 FLOATmark and INTmark

The cache and RAM speeds were determined by the FLOATmark and INTmark Reading and Writing portions of the RAMSMP benchmarks. These benchmarks distinguish between the different cache levels (L1, L2, L3) and RAM available to the systems, with L1 being the fastest and smallest and RAM being the slowest and largest. Figures 1 through 4 show the speeds of the operations at each level of cache for each system. Rather than congesting this section of the report, the numerical results have been tabulated in Appendix E.3 for each of the figures shown below.

From the L1 cache results (Figure 1), we see that Tesla is much faster in reading and writing in all the benchmarks, followed by Henry, which is almost equally as fast. Tesla outperforms the fastest EC2 instance (HM4XLarge) by approximately 28% in terms of bandwidth. Henry is the closest to Tesla in terms of performance, with its bandwidths being on average approximately 80% that of Tesla's. The fastest EC2 instances are HM4XLarge and HCXLarge, performing at an average of approximately 70% and 65% of Tesla's performance respectively for the four tests. These instances are followed by much lower bandwidths for the remaining instances, with the slowest instance (Large), performing at an average of about 18% of Tesla's capabilities.

There is a sharp reduction in the bandwidth observed on Henry when dealing with floating-point values, which is not observable on Tesla or any of the the EC2 instances. Henry is the only machine being benchmarked that has an AMD Opteron processor. It was found in the AMD Opteron Software Optimization Guide that the processor contains a Load/Store unit, that loads data into the L1 cache (and if necessary, the L2 cache and system memory), depending on the data size. The unit is capable of performing up to 64-bit or 16-byte stores of data per clock cycle [29]. From the RAMSMP source code, it was found that the integer data type being used for INTmark was 32 bits, while the floating-point type being used for FLOATmark was 64 bits. Given this size discrepancy, it can be understood that more integer data can be loaded and stored as compared to the amount of floating-point data, given the rate of loading to be 64 bits per clock cycle. This can cause a decrease in the bandwidth associated with writing and reading of floating-point data as compared to integer data, resulting in the sort of behavior displayed by Henry in the figure below.

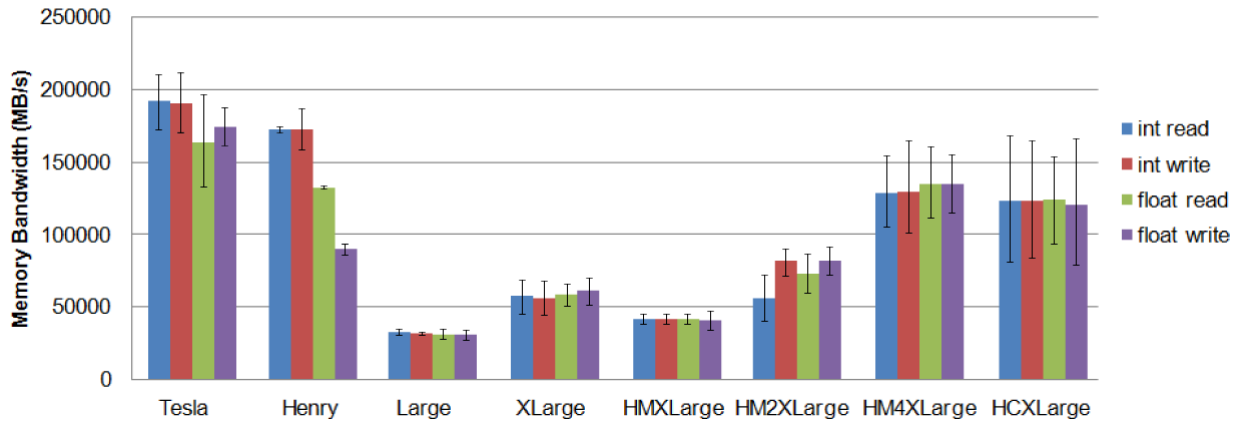

Figure 1: RAMSMP L1 Cache Results

Similarly, for the L2 Cache levels (Figure 2), Tesla once again dominates the rest of the instances in terms of bandwidth, followed by HM4XLarge, and not Henry. The bandwidth for some of the instances was drastically lower for the L2 cache, while Tesla remained relatively high. Unlike its L1 cache, Henry's L2 cache is extremely slow, with an average bandwidth that is less

than 30% of Tesla's. This is most likely due to the fact that Henry does not have an L3 cache, so this is essentially the boundary between RAM and cache. The only other instance that is remotely close in performance to Tesla is the HM4XLarge instance, which averages between 60% and 80% of Tesla's performance. The rest of the instances show a considerable slowdown in terms of L2 cache speeds, with the slowest instance (Large) showing a bandwidth on average about 12% that of Tesla.

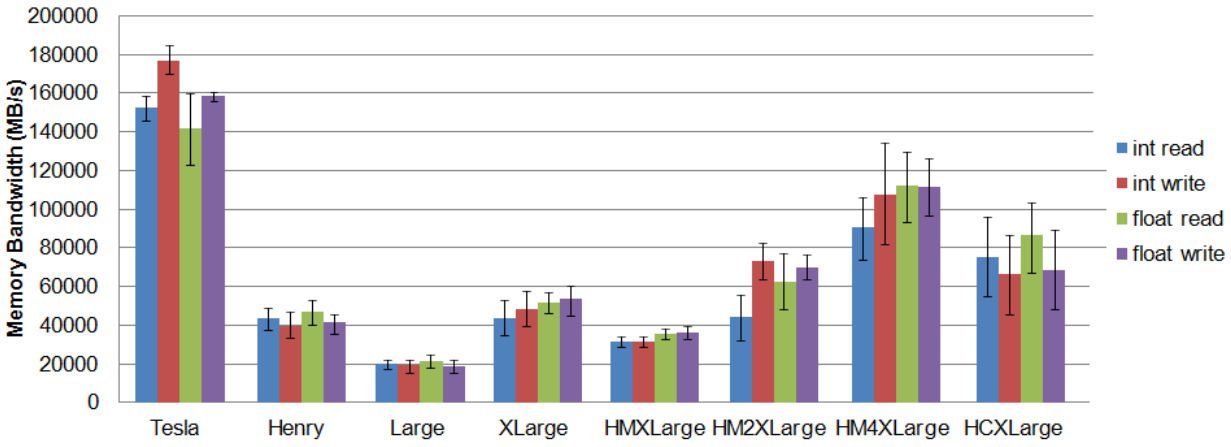

Figure 2: RAMSMP L2 Cache Results

The L3 cache (Figure 3) is not present on all instances, and was only found on six of the eight test systems. As with the prior two cache levels, Tesla outperforms the other instances in both of the reading tests, while it trails only slightly behind the HM4XLarge and HCXLarge instances in both of the writing tests. The HM4XLarge outperforms Tesla by approximately 9% on both of the writing tests, while its performance is significantly worse on the reading tests—averaging only 75% of Tesla's performance. This sort of behavior is also seen in the case of the HCXLarge instance, which outperforms Tesla by about 17% for the integer writing, and 5% for the floating-point writing. However, the reading bandwidths for this instance reduces sharply in comparison with Tesla, settling on average at about 60% of Tesla. The rest of the instances, once again, show a significant drop in reading and writing bandwidths compared to Tesla.

It can also be seen that there is a sharp decrease in writing speeds in the L3 cache in the case of all instances. This, however, can be explained by the fact that the writing operation is slower than the reading operation. This is because the system must select the right size of memory to allocate for the data being written before actually accessing the memory and writing to it (while reading only does memory accesses). Additionally, it is well known that bandwidth decreases while memory size increases, and the L3 cache, if it exists, is typically the largest cache level.

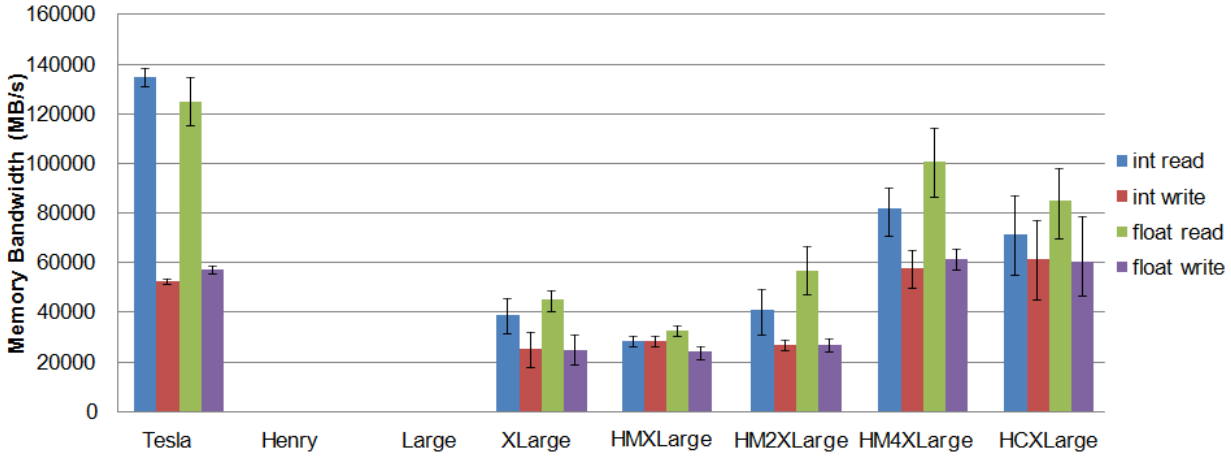

Figure 3: RAMSMP L3 Cache Results

After the cache comes the system's main memory: RAM (Figure 4). Once again Tesla clearly outperforms all of the other systems in every one of the tests. Henry's average bandwidths are around 27% of Tesla's, which is a significant difference in total bandwidth. While the HM4XLarge performs extremely close to Tesla in the two writing tests, it is overshadowed by Tesla's reading performance by nearly 25%. The rest of the instances appear significantly slower than Tesla and HM4XLarge, which further demonstrates both systems' superior memory handling capabilities. Of the remaining instances, the HCXLarge had a significant reduction in its relative performance between the L3 cache and RAM, hovering around 21% of Tesla on average. Surprisingly, the HM2XLarge is the only other instance that even remotely stands out, and its read speeds are barely above the HM4XLarge's write speeds.

Each of the plots in this section have demonstrated Tesla's dominant memory performance at every level, ranging from L1 cache to RAM. While Henry's L1 cache had comparable performance to Tesla's, Henry failed to deliver at the higher memory levels. The only two EC2 instances that stand out are the HM4XLarge (for all levels of memory), and the HCXLarge (primarily for the three cache levels). It was no surprise the the HM4XLarge performed as well as it did, since its memory structure is extremely similar to Tesla. The HCXLarge instance is also able to provide excellent bandwidths for every cache level, most likely due to its multi-core design; unfortunately, the instance greatly slows down when it reaches RAM.

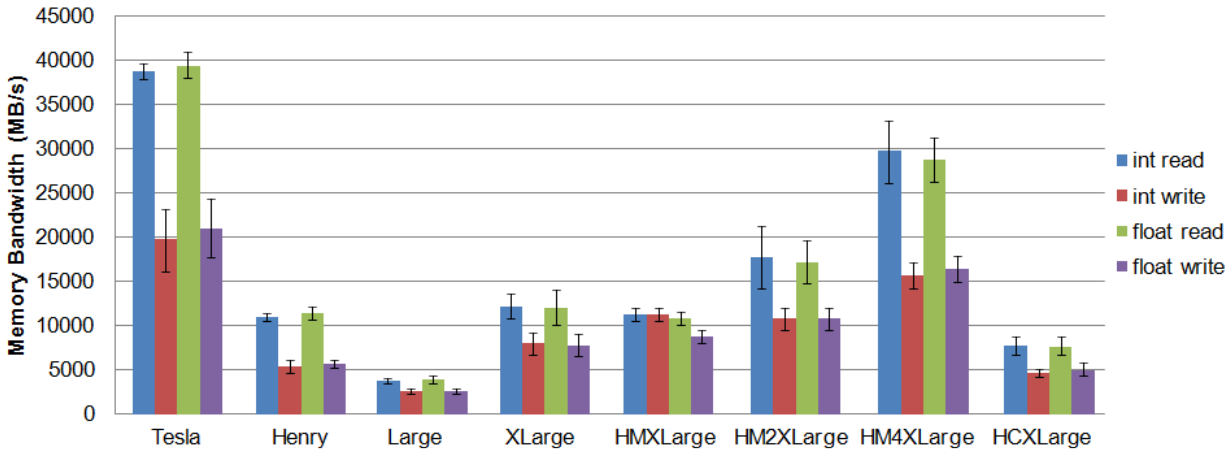

Figure 4: RAMSMP RAM Results

4.3.2 FLOATmem and INTmem

The FLOATmem test performed a series of manipulations on a specified amount of floating-point data by copying, scaling and writing the data into different memory locations. From the results in Table 19, Tesla once again outperformed the most comparable EC2 instance (HM4XLarge) by over 30% in terms of overall memory bandwidth. The execution times for the FLOATmem test are a function of the system's total memory, as well as the number of processes used to analyze it. Even though Tesla's execution time is about 21% greater than that of the HMXLarge instance, it is important to note that the bandwidth of the HMXLarge instance is significantly lower, at about 29% of the rate that Tesla achieves. Because the FLOATmem test performs various manipulations on the RAM, the HCXLarge instance—which we previously showed has terrible performance in RAM—is not even remotely comparable with Tesla, with an average performance around 20% of Tesla's.

System	Average Execution Time (seconds)	Relative Standard Deviation of Execution Time (%)	Average Memory Bandwidth (MB/s)	Relative Standard Deviation of Bandwidth (%)	PCR Adjusted Average Bandwidth	Performance Compared to Tesla
Tesla	28.481	18.089	28308.01	0.234	-	1.000
Henry	46.565	18.09	7975.794	0.455	-	0.282
Large	56.753	5.090	3213.911	4.043	3164.976	0.112
XLarge	54.157	13.370	7758.458	9.040	7640.327	0.270
HMXLarge	22.479	1.544	8222.597	1.143	8097.399	0.286
HM2XLarge	31.602	15.625	11961.620	5.960	11779.491	0.416
HM4XLarge	47.437	3.510	19413.844	2.851	19118.247	0.675
HCXLarge	141.611	4.722	5967.505	10.030	5876.643	0.208

Table 19: Summary of results for RAMSMP FLOATmem Test

The INTmem test was similar in function to the FLOATmem test, except that all of the operations were carried out on integer data rather than floating-point data. The results in Table 20 show that Tesla outperforms the HM4XLarge instance by about 28%. It should also be noted that as in the case of the FLOATmem results, the HCXLarge instance does not perform as well as the other instances, with its bandwidth measuring up to about 23% of Tesla's. On average, Tesla beat out all of the other test systems by between 30%-90% for memory performance. Both the INTmem and FLOATmem results will be used in an comprehensive analysis at the end of this section.

System	Average Execution Time (seconds)	Relative Standard Deviation of Execution Time (%)	Average Memory Bandwidth (MB/s)	Relative Standard Deviation of Bandwidth (%)	PCR Adjusted Average Bandwidth	Performance Compared to Tesla
Tesla	31.285	3.830	27533.830	0.727	-	1.000
Henry	50.193	0.227	7433.762	0.401	-	0.27
Large	56.753	5.090	3205.385	3.493	3168.381	0.115
XLarge	43.780	3.970	8943.317	1.880	8840.073	0.321
HMXLarge	19.966	9.137	9481.861	6.121	9372.400	0.34
HM2XLarge	30.442	5.520	12442.310	1.820	12298.673	0.447
HM4XLarge	46.897	3.248	19837.154	3.815	19608.149	0.712
HCXLarge	146.664	5.013	6382.179	11.453	6308.501	0.229

Table 20: Summary of results for RAMSMP INTmem Test

It should be noted that in most cases the floating-point operations are slightly slower than the integer operations. This is because the FLOATmem and INTmem tests perform mathematical operations during execution. Arithmetic operations for floating-point and integer data types are carried out by separate hardware units, with the Floating Point Unit (FPU) being used for the former, and the Arithmetic Logic Unit (ALU) being used for the latter. ALUs are typically much faster, since they are used by the CPU for almost every single operation, while the FPU is specifically designed for the more complex floating-point operations.

4.4 SPEC2006

The SPEC2006 benchmark suite was used to measure the computational performance of every test system. Each test in the suite returns both a ratio—which compares the performance of the test machine against a baseline system chosen by SPEC—and a total execution time. As was previously mentioned, a higher ratio and lower execution time are indicative of better performance.

We were unable to acquire the SPEC2006 results for the Sunfire X2200 M2 server (Henry), and therefore could make use of Henry in our comparisons of the SPEC results. These tests primarily test the CPU performance of a single core of the system. The results obtained from the integer and floating-point suites have been plotted in Figures 5 and 7, and tabulated in Tables 21 and 21.

SPEC CINT2006 Tests							
Results		**400 perlbench**	**401 bzip2**	**403 gcc**	**429 mcf**	**445 gobmk**	**456 hmmer**
Tesla	SPEC Ratio	25.5	19.9	24.7	42.2	24.2	46.8
	Runtime	384	484	326	216	433	199
	PCR-Ratio	-	-	-	-	-	-
	PCR-Runtime	-	-	-	-	-	-
	Tesla Ratio	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
Henry	SPEC Ratio	-	-	-	-	-	-
	Runtime	-	-	-	-	-	-
	PCR-Ratio	-	-	-	-	-	-
	PCR-Runtime	-	-	-	-	-	-
	Tesla Ratio	**-**	**-**	**-**	**-**	**-**	**-**
Large	SPEC Ratio	12.04	8.17	7.07	8.78	10.6	5.63
	Runtime	811.128	1181.429	1138.466	1038.733	989.451	1656.633
	PCR-Ratio	12.040	8.217	9.477	8.632	10.600	5.532
	PCR-Runtime	811.128	1188.258	1526.029	1021.226	989.451	1627.822
	Tesla Ratio	**0.473**	**0.323**	**0.252**	**0.376**	**0.388**	**0.236**
Xlarge	SPEC Ratio	19.56	15.13	9.84	14.22	16.31	10.24
	Runtime	499.484	637.864	818.155	641.326	643.075	911.099
	PCR-Ratio	19.560	15.217	13.190	13.980	16.310	10.062
	PCR-Runtime	499.484	641.551	1096.676	630.517	643.075	895.254
	Tesla Ratio	**0.769**	**0.599**	**0.350**	**0.609**	**0.597**	**0.429**
HMXLarge	SPEC Ratio	20.88	14.5	13.5	17.7	17.81	9.35
	Runtime	467.935	665.622	596.228	515.335	588.938	997.503
	PCR-Ratio	20.880	14.584	18.096	17.402	17.810	9.187
	PCR-Runtime	467.935	669.470	799.199	506.650	588.938	980.155
	Tesla Ratio	**0.821**	**0.574**	**0.480**	**0.758**	**0.652**	**0.392**
HM2XLarge	SPEC Ratio	21.68	14.92	13.72	18.17	18.08	9.37
	Runtime	450.726	646.626	586.831	502.027	580.067	995.359
	PCR-Ratio	21.680	15.006	18.391	17.864	18.080	9.207
	PCR-Runtime	450.726	650.364	786.603	493.566	580.067	978.048
	Tesla Ratio	**0.852**	**0.590**	**0.488**	**0.778**	**0.662**	**0.393**
HM4XLarge	SPEC Ratio	21.08	14.88	13.77	18.27	18.1	9.37
	Runtime	463.377	648.425	584.775	499.24	579.647	995.55
	PCR-Ratio	21.080	14.966	18.458	17.962	18.100	9.207
	PCR-Runtime	463.377	652.173	783.847	490.826	579.647	978.236
	Tesla Ratio	**0.829**	**0.589**	**0.490**	**0.782**	**0.662**	**0.393**
HCXLarge	SPEC Ratio	16.29	11.1	11.2	12.35	14.12	7.47
	Runtime	599.711	869.557	819.483	738.226	742.999	1248.538
	PCR-Ratio	16.290	11.164	15.013	12.142	14.120	7.340
	PCR-Runtime	599.711	874.583	1098.456	725.784	742.999	1226.824
	Tesla Ratio	**0.640**	**0.439**	**0.350**	**0.529**	**0.517**	**0.313**

Table 21: SPEC CINT2006 Results

Results		458 sjeng	462 libquantum	464 h264ref	471 omnetpp	473 astar	483 xalancbmk
Tesla	SPEC Ratio	25.9	477.0	37.1	21.6	20.0	36.1
	Runtime	468	43	96	290	350	191
	PCR-Ratio	-	-	-	-	-	-
	PCR-Runtime	-	-	-	-	-	-
	Tesla Ratio	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
Henry	SPEC Ratio	-	-	-	-	-	-
	Runtime	-	-	-	-	-	-
	PCR-Ratio	-	-	-	-	-	-
	PCR-Runtime	-	-	-	-	-	-
	Tesla Ratio	**-**	**-**	**-**	**-**	**-**	**-**
Large	SPEC Ratio	10.360	14.630	11.880	5.090	5.550	9.670
	Runtime	1168.364	1416.168	1863.604	1227.754	1265.458	713.778
	PCR-Ratio	10.360	14.769	11.973	5.013	5.594	9.523
	PCR-Runtime	1168.364	1429.655	1878.163	1209.152	1275.501	702.963
	Tesla Ratio	**0.401**	**0.030**	**0.051**	**0.240**	**0.274**	**0.272**
XLarge	SPEC Ratio	15.950	15.930	22.620	9.850	9.970	15.510
	Runtime	758.438	1300.477	978.441	634.717	703.764	444.782
	PCR-Ratio	15.950	16.082	22.797	9.701	10.049	15.275
	PCR-Runtime	758.438	1312.862	986.085	625.100	709.349	438.043
	Tesla Ratio	**0.617**	**0.033**	**0.097**	**0.464**	**0.493**	**0.436**
HMXLarge	SPEC Ratio	17.360	26.720	27.220	12.290	11.800	19.810
	Runtime	697.088	775.453	812.873	508.371	594.762	348.324
	PCR-Ratio	17.360	26.974	27.433	12.104	11.894	19.510
	PCR-Runtime	697.088	782.838	819.224	500.668	599.482	343.046
	Tesla Ratio	**0.671**	**0.055**	**0.117**	**0.579**	**0.584**	**0.557**
HM2XLarge	SPEC Ratio	17.530	27.290	27.410	12.690	12.080	20.630
	Runtime	690.245	759.124	807.289	492.553	581.222	334.538
	PCR-Ratio	17.530	27.550	27.624	12.498	12.176	20.317
	PCR-Runtime	690.245	766.354	813.596	485.090	585.835	329.469
	Tesla Ratio	**0.678**	**0.057**	**0.118**	**0.598**	**0.597**	**0.580**
HM4XLarge	SPEC Ratio	17.510	27.310	27.490	12.670	12.040	20.660
	Runtime	690.838	758.729	804.898	493.304	583.118	334.007
	PCR-Ratio	17.510	27.570	27.705	12.478	12.136	20.347
	PCR-Runtime	690.838	765.955	811.186	485.830	587.746	328.946
	Tesla Ratio	**0.677**	**0.057**	**0.118**	**0.597**	**0.595**	**0.581**
HCXLarge	SPEC Ratio	13.910	18.700	21.780	8.870	8.900	13.720
	Runtime	870.091	1108.004	1016.248	704.988	788.413	503.054
	PCR-Ratio	13.910	18.878	21.950	8.736	8.971	13.512
	PCR-Runtime	870.091	1118.556	1024.187	694.306	794.670	495.432
	Tesla Ratio	**0.538**	**0.039**	**0.094**	**0.418**	**0.440**	**0.386**

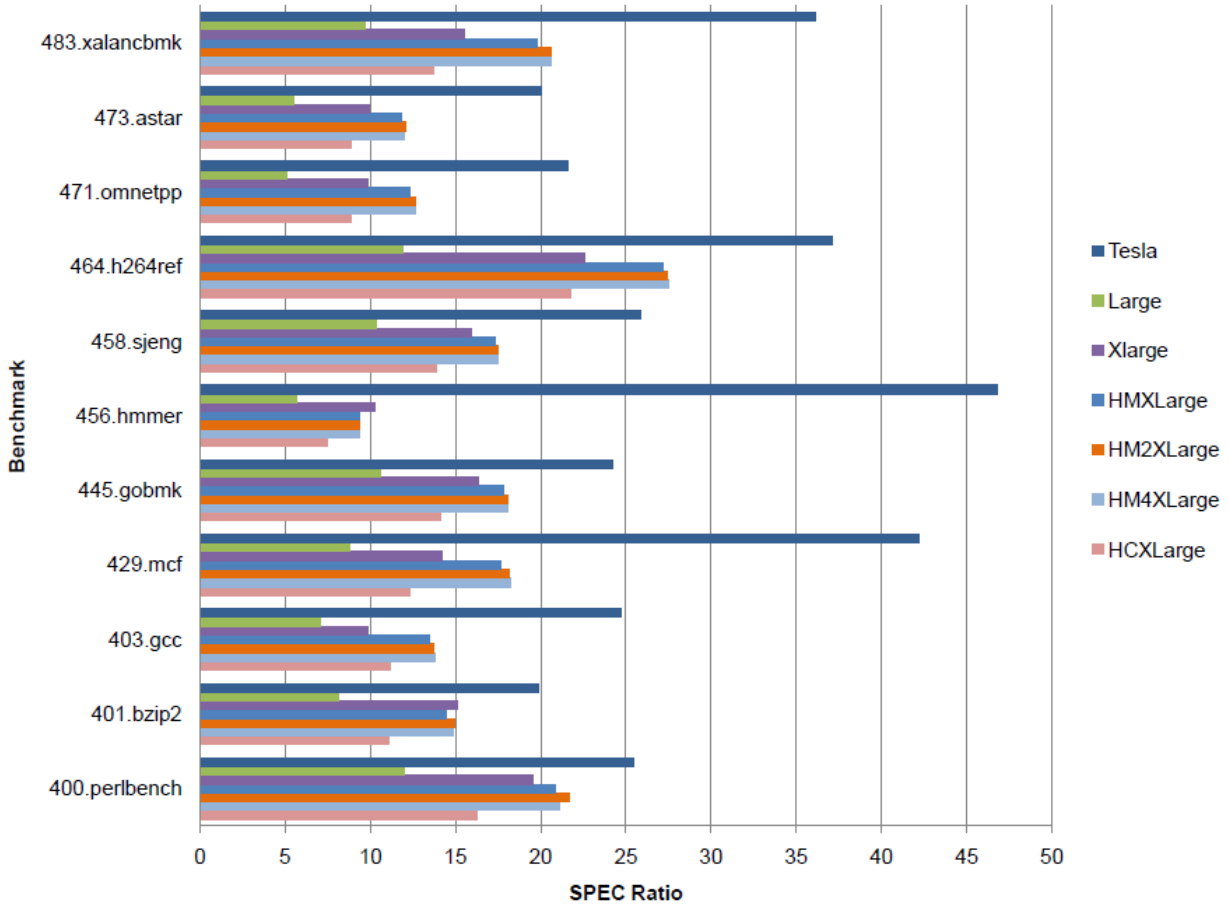

Figure 5: SPEC CINT2006 Results

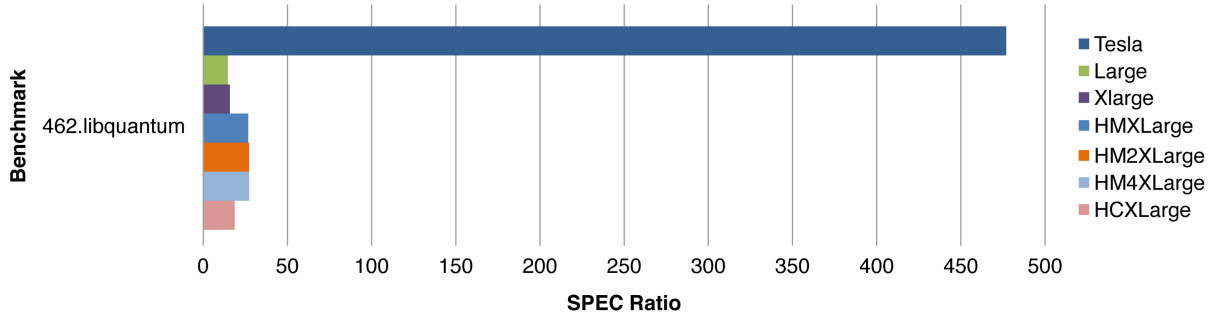

Figure 6: SPEC CINT2006 - Libquantum Results

The previous figures and tables show that Tesla is able to outperform every EC2 instance in all the benchmarks run from the integer benchmark suite. Each of the benchmarks ran differently on each instance, with the performance of the HM4XLarge instance ranging from almost 83% for the 400.perlbench benchmark, to only 6% of Tesla's performance for the 462.libquantum benchmark; it is important to note, however, that Tesla experienced a phenomenally high ratio for the libquantum benchmark, as such it has been displayed separately in Figure 6. The Large instance routinely performed worse than the other test systems, with performances ranging from 47% for the 400.perlbench benchmark, to 3% for the 462.libquantum benchmark, when compared to Tesla's performance. In general, the high memory instances, as well as the XLarge, are closest to Tesla in terms of their adjusted performance ratios.

SPEC CFP2006 Tests								
Results		**433 milc**	**444 namd**	**470 lbm**	**447 dealII**	**450 soplex**	**453 povray**	**482 sphinx3**
Tesla	SPEC Ratio	34.3	19.1	54.7	33.9	28.2	28.6	40.6
	Runtime	267	420	252	337	296	186	480
	PCR-Ratio	-	-	-	-	-	-	-
	PCR-Runtime	-	-	-	-	-	-	-
	Tesla Ratio	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
Henry	SPEC Ratio	-	-	-	-	-	-	-
	Runtime	-	-	-	-	-	-	-
	PCR-Ratio	-	-	-	-	-	-	-
	PCR-Runtime	-	-	-	-	-	-	-
	Tesla Ratio	**-**	**-**	**-**	**-**	**-**	**-**	**-**
Large	SPEC Ratio	7.18	10.13	5.09	16.41	9.88	14.23	14.16
	Runtime	1278.12	791.369	2701.489	697.305	844.086	373.804	1376.54
	PCR-Ratio	6.959	10.130	5.070	16.871	10.069	15.012	14.349
	PCR-Runtime	1238.797	791.369	2691.138	716.892	860.241	394.343	1394.889
	Tesla Ratio	**0.216**	**0.531**	**0.093**	**0.483**	**0.351**	**0.498**	**0.349**
XLarge	SPEC Ratio	9.45	9.38	14.2	14.26	12.11	13.03	12.44
	Runtime	971.491	854.925	967.925	801.985	688.852	40.441	1567.03
	PCR-Ratio	9.159	9.380	14.146	14.661	12.342	13.746	12.606
	PCR-Runtime	941.599	854.925	964.216	824.513	702.036	42.663	1587.927
	Tesla Ratio	**0.284**	**0.491**	**0.260**	**0.420**	**0.430**	**4.599**	**0.306**
HMXLarge	SPEC Ratio	13.63	14.19	28.37	22.11	21.83	19.72	25.67
	Runtime	673.592	565.371	484.265	517.526	381.96	269.824	759.357
	PCR-Ratio	13.211	14.190	28.261	22.731	22.248	20.804	26.012
	PCR-Runtime	652.866	565.371	482.410	532.063	389.270	284.649	769.482
	Tesla Ratio	**0.409**	**0.743**	**0.520**	**0.651**	**0.775**	**0.689**	**0.632**
HM2XLarge	SPEC Ratio	13.57	13.88	28.06	22.19	21.42	19.71	24.85
	Runtime	676.541	577.724	489.731	515.445	389.364	269.952	784.453
	PCR-Ratio	13.152	13.880	27.952	22.813	21.830	20.793	25.181
	PCR-Runtime	655.724	577.724	487.855	529.924	396.816	284.785	794.912
	Tesla Ratio	**0.407**	**0.727**	**0.515**	**0.654**	**0.760**	**0.689**	**0.612**
HM4XLarge	SPEC Ratio	13.72	14.2	28.41	22.17	21.86	19.68	25.56
	Runtime	668.879	564.982	483.647	516.075	381.562	270.376	762.64
	PCR-Ratio	13.298	14.200	28.301	22.793	22.278	20.761	25.901
	PCR-Runtime	648.298	564.982	481.794	530.571	388.865	285.232	772.809
	Tesla Ratio	**0.412**	**0.743**	**0.521**	**0.653**	**0.776**	**0.688**	**0.629**
HCXLarge	SPEC Ratio	11.2	11.32	14.49	17.17	14.21	15.74	16.28
	Runtime	819.483	708.52	948.551	666.14	586.877	338.095	1196.99
	PCR-Ratio	10.855	11.320	14.434	17.652	14.482	16.605	16.497
	PCR-Runtime	794.268	708.520	944.917	684.852	598.109	356.672	1212.946
	Tesla Ratio	**0.336**	**0.593**	**0.266**	**0.506**	**0.504**	**0.550**	**0.401**

Table 22: SPEC CFP2006 Results

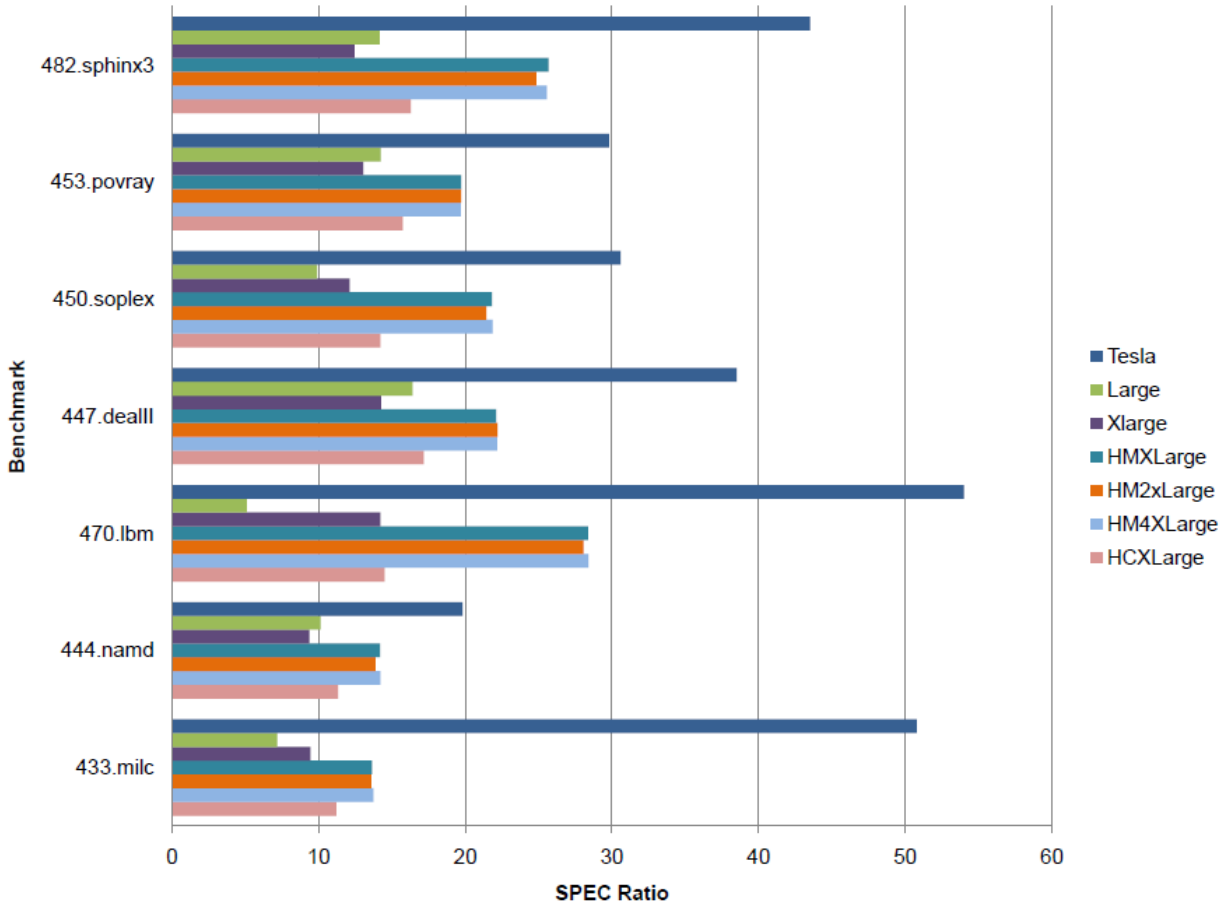

Figure 7: SPEC CFP2006 Results

Figure 7 shows the results obtained for the floating-point tests that were run from the SPEC suite. As was previously mentioned, none of the Fortran benchmarks were used, and our results only show those tests that were written in C and C++. Similar to our findings with the integer tests, Tesla's performance was the best out of the entire set of systems. The HM4XLarge's performance ranged from about 83% for the 450.soplex benchmark, to about 38% for the 433.milc benchmark, when compared with that of Tesla's. The other two high memory instances showed similar results for the floating-point benchmarks, followed by the HCXLarge instance and lastly by the Large instance.

4.5 MATLAB Case Study

The MATLAB case study was run a total of 10 times on each instance in order to simulate a real world application. The results of this case study have been shown in Table 23.

System	Average Execution Time (seconds)	Relative Standard Deviation of Execution Time (%)	PCR-Adjusted Average Execution Time (seconds)	Performance Compared to Tesla
Tesla	209.134	7.452	-	1.000
Henry	522.78	14.89	-	0.400
Large	367.83	3.23	377.058	0.555
XLarge	392.89	1.66	402.746	0.519
HMXLarge	221.61	0.35	227.169	0.921
HM2XLarge	218.92	0.21	224.412	0.932
HM4XLarge	220.44	0.25	225.97	0.925
HCXLarge	371.13	2.23	380.44	0.550

Table 23: Execution Times of MATLAB Case Study

As with most of the benchmarks, Tesla outperformed the fastest instance, in this case, the HM2XLarge, by a little over 6%. There was a considerable amount of variance in the execution times of Tesla and Henry, which was not observed in any of the EC2 instances. This can be attributed to the fact that at the time these tests were being run, there was one other member of the research community utilizing both Tesla and Henry, at approximately 700% and 350% of CPU respectively. The CPU usage of 700% for Tesla indicates that the program that was being run was multi-threaded, and was running over 7 cores. Since Tesla only has 8 physical cores, this likely caused a buildup of processes schedule to run on the one "available" core, resulting in sporadic and inexplicable program execution. We determined that our case study required a little over 1 full core (the program utilized about 105% CPU). It is not surprising that our results showed so much deviation, since the total amount of available resources was less than what our program required to properly execute.

Similarly, Henry consists of 4 physical cores, and with one person running their program at 350% CPU usage, our program was left with about half a core. Due to the relatively high variance with these results, we decided to wait until the servers were both empty to run the case study again. Table 24 summarizes the results.

System	Execution Time (seconds)	Standard Deviation (%)	PCR Adjusted Average	Performance Compared to Tesla
Tesla	160.350	4.770	-	1.000
Henry	411.037	10.670	-	0.390

Table 24: Case Study Results on Tesla and Henry (no load)

As the results of this second test show, the standard deviation for Tesla was reduced to about 5%, while Henry only fell to 11%. Even though a 5% deviation is still quite high compared to the results obtained from EC2, the accuracy that was gained in the second run helped to support the observed increase in performance of almost 13%. Regardless of the server's CPU usage, Henry's performance was still significantly worse than Tesla's.

While it is noteworthy that EC2 was able to run this case study much more consistently

than the ECE servers, it is extremely impressive that Tesla was capable of surpassing the EC2 instances despite already having a heavy processor load. Tesla's performance increased even further when it experienced no load, although there was still a noticeable amount of deviation in its results which could be attributed to various factors such as latencies associated with the campus license server, background tasks, and so on.

It is important to note that in all of the CPU-intensive benchmarks, Henry did not perform as well as the EC2 instances, let alone Tesla. This, however, can be explained by the fact that the hardware for Henry was purchased four years ago, as opposed to Tesla which was purchased about a year ago. It is possible that the processing units that support the EC2 instances are running newer hardware than Henry, which would account for this difference.

Although Tesla experienced inconsistent results for the case study, these variances did not skew the conclusions that were drawn from the result; this was ensured by the second case study execution. It should also be noted that the case study is not a formal benchmark, and has simply been included to determine whether or not our other findings are relevant for real-world applications.

4.6 Comprehensive Analysis

At this point, each individual component of the test systems that has been benchmarked has been discussed in detail without making any global judgments on system performance. With the exception of extremely specialized applications, real-world problems rarely rely on solely one aspect of a computer system. The purpose of this section is to present the overall findings from our benchmarks, and draw definitive conclusions about the relative performance of each system.

As was previously discussed, most of our computationally-heavy benchmarks returned execution times upon completion, while our memory benchmarks supplied the bandwidths associated with the given memory levels. In order to compare these different metrics, it was necessary to create uniform synthetic performance indicators based upon the execution times of the benchmarks. Unfortunately, even though Pystone already returned the number of Pystones per second, their values were inconsistent with the clock times. To amend this issue, a new value for Pystones per second was calculated based on the correct time value. In addition to the Pystone benchmark, the MATLAB benchmark and the case study also only returned an execution time.

For the MATLAB benchmark, the total number of tests that were executed was constant, and as such this was used as a base for calculating the computing power. The exact equation has already been documented in Section 3.3.2.1, and the resulting values are shown in Table 25.

System	Average Single-Threaded Execution Time (seconds)	PCR-Adjusted Average Time	Calculated PCR-Adjusted Computing Power (tests/second)
Tesla	9.225	-	1.626
Henry	16.571	-	0.905
Large	22.544	23.875	0.628
XLarge	23.460	24.845	0.604
HMXLarge	16.248	17.207	0.872
HM2XLarge	14.991	15.876	0.945
HM4XLarge	14.086	14.918	1.006
HCXLarge	18.592	19.690	0.762

Table 25: Synthesized MATLAB benchmark Performance

A similar process was used for converting the execution times of the MATLAB case study into a suitable metric for comparing computing power. Due to the nature of the case study—which returns over a thousand values—a large constant, $I = 10000$, was chosen to replace the number of tests used for the previous calculations. This ensured that the resulting metric would be larger than one, and because I is a constant, it will affect all of the results by the same amount. The values for each system are shown in Table 26.

System	Average Execution Time (seconds)	PCR-Adjusted Average Execution Time (seconds)	Calculated PCR-Adjusted Computing Power (10000/second)
Tesla	175.768	-	56.893
Henry	522.78	-	19.129
Large	367.83	377.058	26.521
XLarge	392.89	402.746	24.830
HMXLarge	221.61	227.169	44.020
HM2XLarge	218.92	224.412	44.561
HM4XLarge	220.44	225.97	44.254
HCXLarge	371.13	380.44	26.285

Table 26: Synthesized MATLAB Case Study Performance

These synthetic representations of computing power, along with the results from the previous sections—including the overall memory bandwidths returned by the INTmem and FLOATmem benchmarks, and the performance ratio returned by SPEC—were used to form the comprehensive data set. An overall metric was computed by taking the geometric mean of all values within a given system's data set, and from this metric we developed a ratio of true performance for each system. Table 27 shows the the raw performance metrics that were used to calculate the ratios in Table 28.

	Average Performance Metrics					
System	**Pystone (Pystones)**	**MATLAB Benchmark**	**INTmem**	**FLOATmem**	**SPEC (Ratio)**	**MATLAB Case Study**
Tesla	108918.604	1.626	27533.830	28308.010	34.187	175.768
Henry	72668.997	0.905	7433.762	7975.794	12.137	522.780
Large	62763.356	0.628	3131.805	3116.785	9.470	367.830
XLarge	62917.024	0.604	8738.021	7523.995	13.024	392.890
HMXLarge	87118.363	0.872	9264.203	7974.107	18.133	221.610
HM2XLarge	87321.409	0.945	12156.694	11600.135	18.424	218.920
HM4XLarge	87195.271	1.006	19381.787	18827.150	18.541	220.440
HCXLarge	72168.110	0.762	6235.674	5787.165	13.392	371.130

Table 27: Average Performance Metrics by System

	Geometric Mean of Performance Metrics	**Overall Performance Compared to Tesla**
Tesla	969.320	1.000
Henry	539.824	0.557
Large	332.068	0.343
XLarge	483.521	0.499
HMXLarge	531.505	0.548
HM2XLarge	600.576	0.620
HM4XLarge	712.435	0.735
HCXLarge	463.076	0.478

Table 28: Overall Performance Comparison

As the above data clearly demonstrates, Tesla is by far the best performing server in the entire set of test systems. Even by comparing every aspect of their performance, the next closest machine, HM4XLarge, has a best-case performance that is only 73.5% of Tesla's, followed by HM2XLarge at about 62%.

For most institutions, server selection comes down to more than just raw performance; there is usually a price tag. So, in addition to determining the difference in performance, an analysis was conducted to determine the cost per computing performance (CCP) for each of the test systems. The CCP was calculated by taking the annual cost of operating a given system and dividing it by the system's overall performance (as computed for Table 28). For the EC2 instances, all that was necessary to compute the annual associated costs C was to determine the price of running an instance for one year—which is function of the price per instance-hour P, the number of years Y, and an initial fee F charged by Amazon—such that $C = Y \times (24 \times 365 \times P) + F$. Determining the annual costs for Tesla and Henry was slightly more complicated, and draws from calculations made in Section 6 [Note: The following analysis assumes initial investments are made at the start of the one or three-year term]. The resulting

CCPs for each system are summarized in Tables 29 and 30, where a lower cost per computing performance is more desirable.

System	Annual Cost (USD)	Geometric Mean of Performance Metrics	Cost per Performance	Relative Performance
Tesla	13507.42	969.320	13.935	1.000
Henry	7064.90	539.824	13.087	0.939
Large	1961.20	332.068	5.906	0.424
XLarge	3922.40	483.521	8.112	0.582
HMXLarge	2814.20	531.505	5.295	0.380
HM2XLarge	5628.40	600.576	9.372	0.673
HM4XLarge	11256.80	712.435	15.800	1.134
HCXLarge	3922.40	463.076	8.470	0.608

Table 29: Cost per Performance Unit for 1-year Reserved Instances

For 1-year reserved instance pricing against the initial purchase and costs of an in-house server, look to Table 29. The CCP for Tesla is higher than all of the other systems except for the HM4XL, and is approximately 62% more expensive than the cheapest instance—HMXLarge. Tesla's exorbitant costs stem from the fact that this is based on an initial investment model, wherein the server hardware was just purchased, and its cost is reflected as a large percentage of the total annual cost. Not surprisingly, Henry also has a large CCP compared to the other servers. Some interesting features of the costs associated with the HMXLarge instance will be explored further in Section 5.

System	Annual Cost (USD)	Geometric Mean of Performance Metrics	Cost per Performance	Relative Performance
Tesla	5507.42	969.320	5.682	1
Henry	2844.90	539.824	5.270	0.928
Large	1517.87	332.068	4.571	0.804
XLarge	3035.73	483.521	6.278	1.105
HMXLarge	2155.87	531.505	4.056	0.714
HM2XLarge	4311.73	600.576	7.179	1.264
HM4XLarge	8623.47	712.435	12.104	2.130
HCXLarge	3035.73	463.076	6.556	1.154

Table 30: Cost per Performance Unit for 3-year Reserved Instances

An interesting and slightly unexpected aspect of our analysis was found after doing a three-year CCP comparison. As Table 30 shows, Tesla is actually a much better option over an extended duration. The relative performance for each of the EC2 instances was nearly doubled, while Henry remained about the same. It is important to note that while the relative performance for each instance went up, the actual CCP for all of the systems were reduced between the 1-year and 3-year analyses. As with the 1-year analysis, the HMXLarge still has a lower CCP than Tesla, by almost 30%, and could be a viable investment based on these results. These analyses will not cover active server replacement via Amazon EC2, as that will be described in great detail

in Section 6. Due to the drastic differences in the potential CCP between fully utilized and under-utilized EC2 instances, we conducted an extended cost analysis on the time that it takes to find a solution versus the cost to get to that point, which will be further documented in Section 5.

5 Cost vs. Time to Solution

Our goal for this section was to determine which instances were most cost effective for multi-threaded and single-threaded applications and to give an accurate representation of the cost associated with each possible scenario. This analysis will also demonstrate the most cost-effective way to run massively parallel applications on the Amazon EC2 service.

We derived a formula to compute the simplified cost C to run E executions of a benchmark B as a function of time T for each instance I, where $X(B, I)$ is the execution time in minutes for a given benchmark on an instance, $P(I)$ is the price per instance-hour, and $N(I)$ is the number of threads that a given instance has. The equation can be seen below:

$$C(E, B, T, I) = P(I) \cdot \left\lceil \frac{E \cdot X(B, I)}{T \cdot N(I)} \right\rceil \cdot \left\lceil \frac{T}{60} \right\rceil \tag{2}$$

The equation can be split into the products of three parts: the leftmost represents the price per instance-hour, the center represents the number of instances needed, and the rightmost represents the total number of hours that are charged. This equation represents the ideal case, where all instances will be launched at exactly the same time at the start of the trial. For single-threaded applications, $N(I)$ will return 1, rather than the number of threads that a given instances contains.

5.1 Single Threaded Applications

We first examine the time vs. dollars to solution for single-threaded applications using the Pystone and MATLAB benchmarks, as well as the MATLAB case study. By computing the cost at every available point in time using Equation 1 above, it was possible to graphically represent the cost to compute a given amount of benchmarks—in our case 1000—in different amounts of time. Our results for the Pystone benchmark can be seen in Figure 8 below.

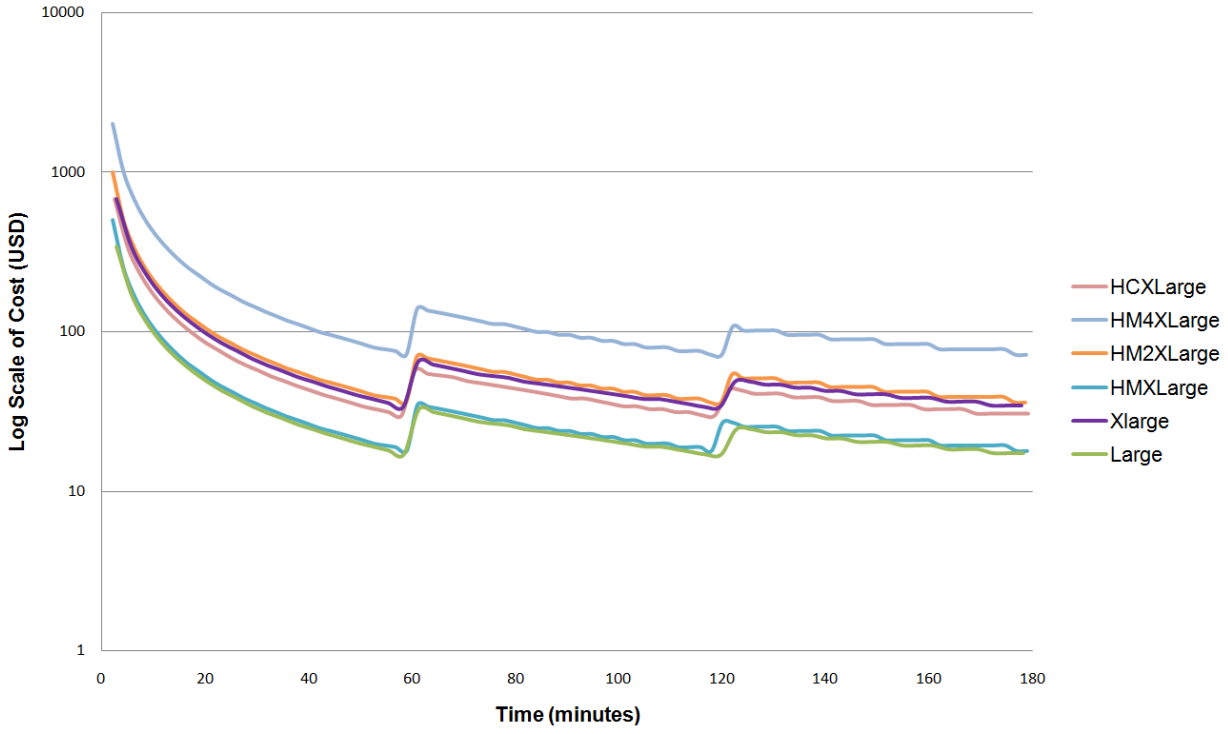

Figure 8: Single-Threaded Pystone Benchmark Cost vs. Time (1000 Runs)

Unless a solution is needed in under an hour, the most cost and time efficient choice of instance will always be the instance that has the lowest price just prior to the one-hour mark. In Figure 8, it is difficult to determine whether the HMXLarge or the Large instance is more cost-effective, as such either would be a suitable choice for single-threaded Pystone applications. Figure 9 below shows the MATLAB benchmark results.

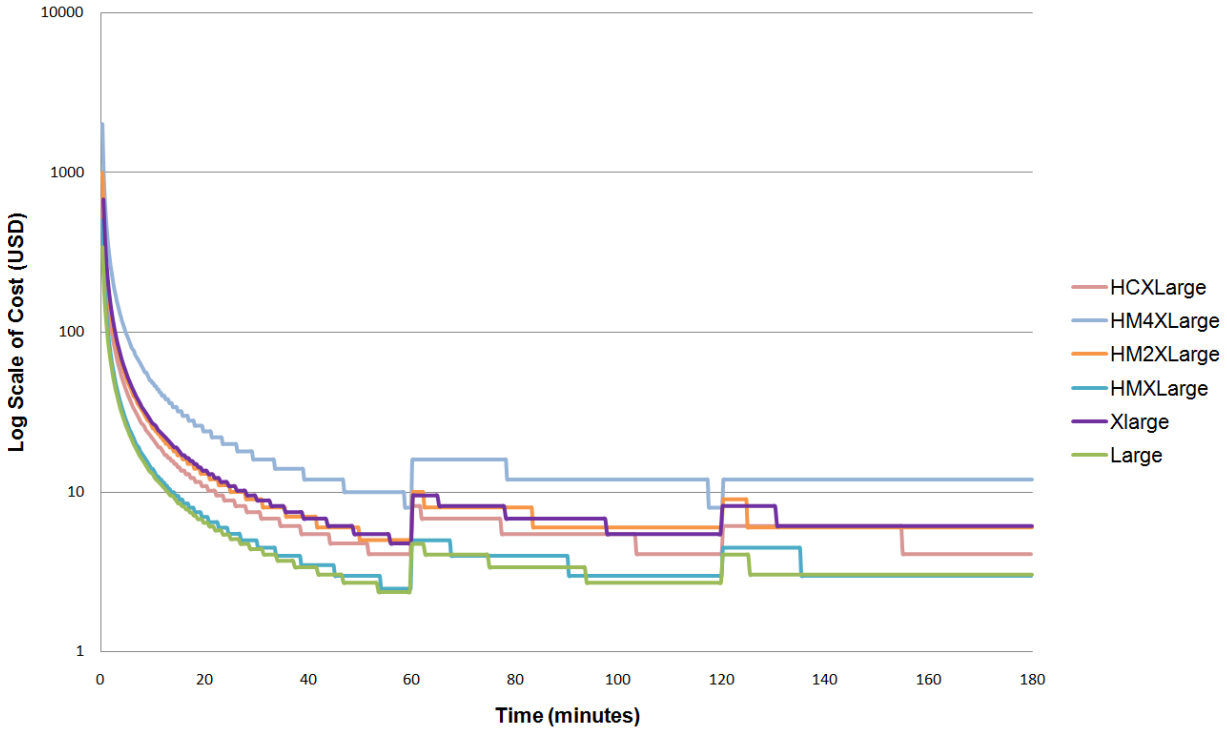

Figure 9: Single-Threaded MATLAB benchmark Cost vs. Time (1000 Runs)

Similar to the Pystone results, the HMXLarge and Large instances were once again the most cost-effective solutions for the MATLAB benchmark. An interesting component to each of the single-threaded plots is the grouping of the instance types into three different cost-efficiency tiers. HM4XLarge is always in the highest—and therefore least efficient—tier; HCXLarge, HM2XLarge and XLarge are always in the middle tier; HMXLarge and Large are always in the lowest tier, making them the most cost-efficient single-threaded instances. This is because they are wasting the lowest number of available cores when they are using a single thread, as they each only have two cores. Figure 10 shows this trend in the most detail for the MATLAB case study.

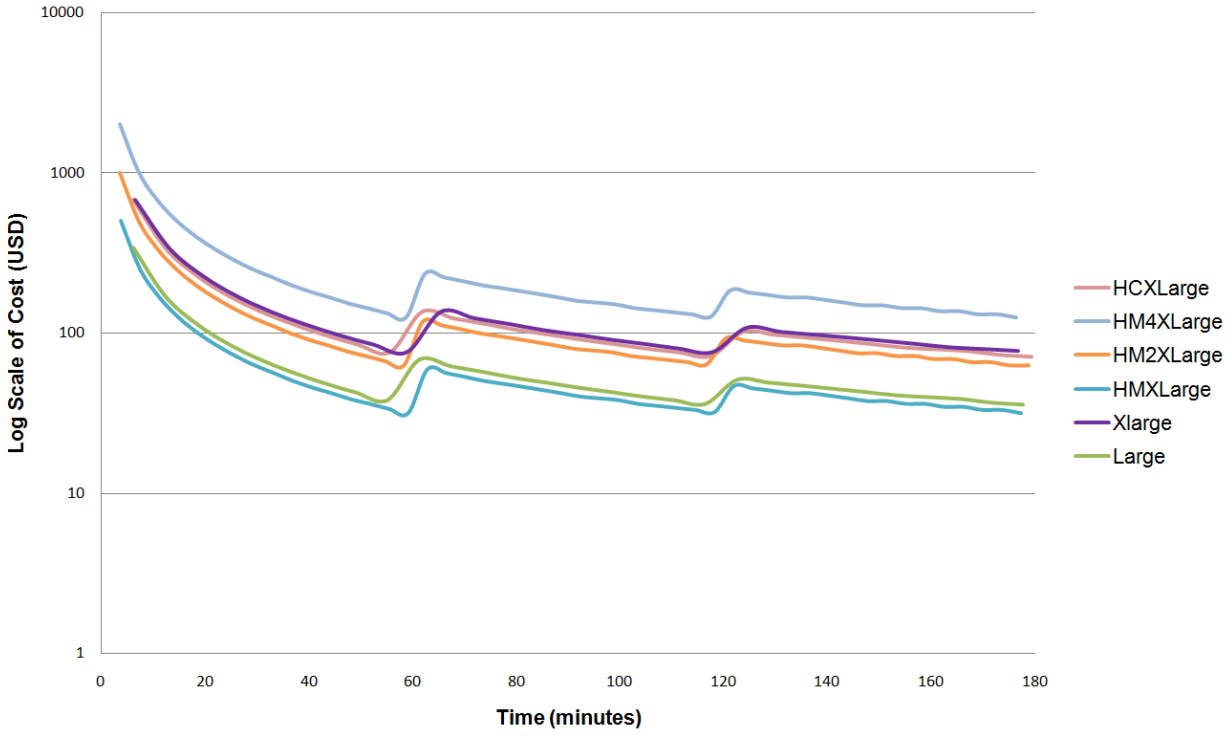

Figure 10: Single-Threaded MATLAB Case Study Cost vs. Time (1000 Runs)

The three tiers are most discernible in the results for the MATLAB case study, which shows that even for real-world applications the most cost-efficient single-threaded instance will be either the HMXLarge or Large. For the MATLAB case study, the HMXLarge actually had a lower absolute cost than the Large instance. This could imply that the HMXLarge is actually a more suitable choice for single-threaded applications, but the results for the other two single-threaded applications have been too close to declare either one as being the most cost-effective choice.

5.2 Multi-threaded Applications

The downside to the single-threaded analysis is that it is rarely more cost effective to use a single core of a machine, which is why we will now examine the costs associated with the same three applications running on multiple threads. The first case is the Pystone benchmark, running 1000 multi-threaded trials, as shown in Figure 11. Unlike the results obtained from the single-threaded applications, there is clearly one instance that out-performs all others: HCXLarge. With eight cores, and the same price as the XLarge instance (USD 0.68/hour), it is no surprise that even with its inferior execution time it is the most cost-effective instance. It costs $4.08 to complete 1000 trials in one hour on the HCXLarge, and the next most efficient instance is the Large at $8.50 to accomplish the same feat. Each of the plots in this section further demonstrates HCXLarge's superior multi-threading capabilities.

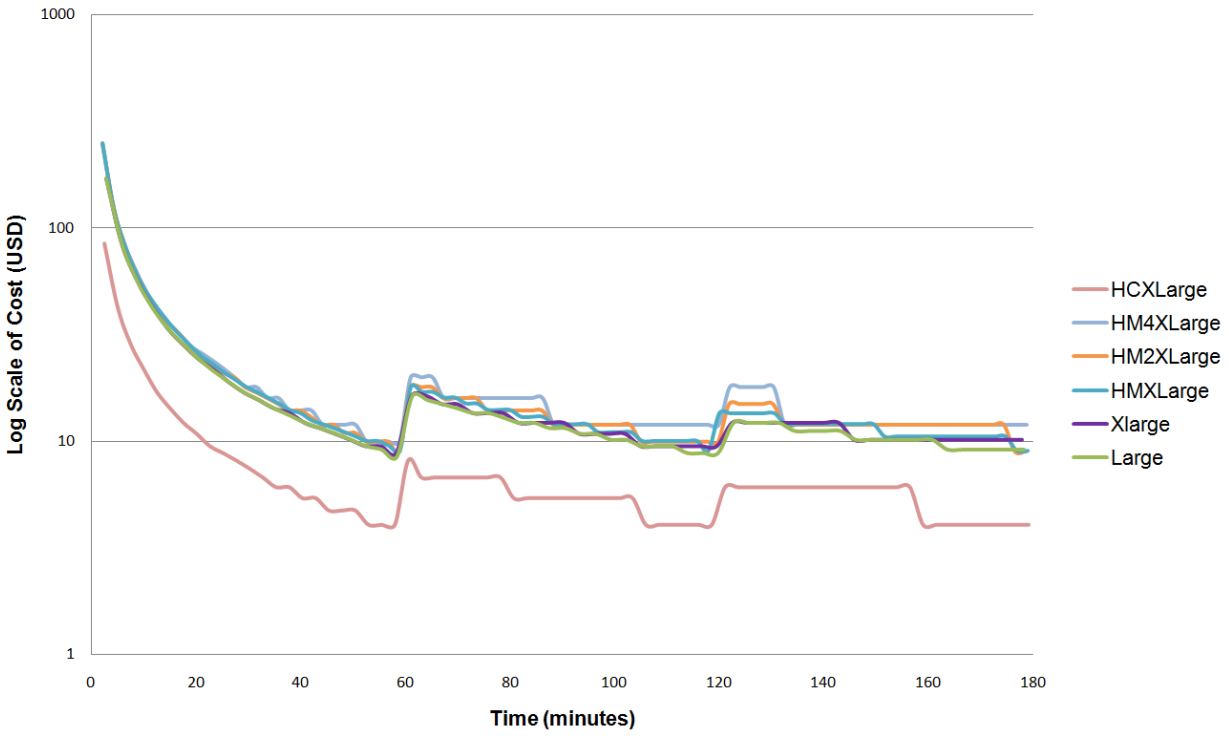

Figure 11: Multi-threaded Pystone Benchmark Cost vs. Time (1000 Runs)

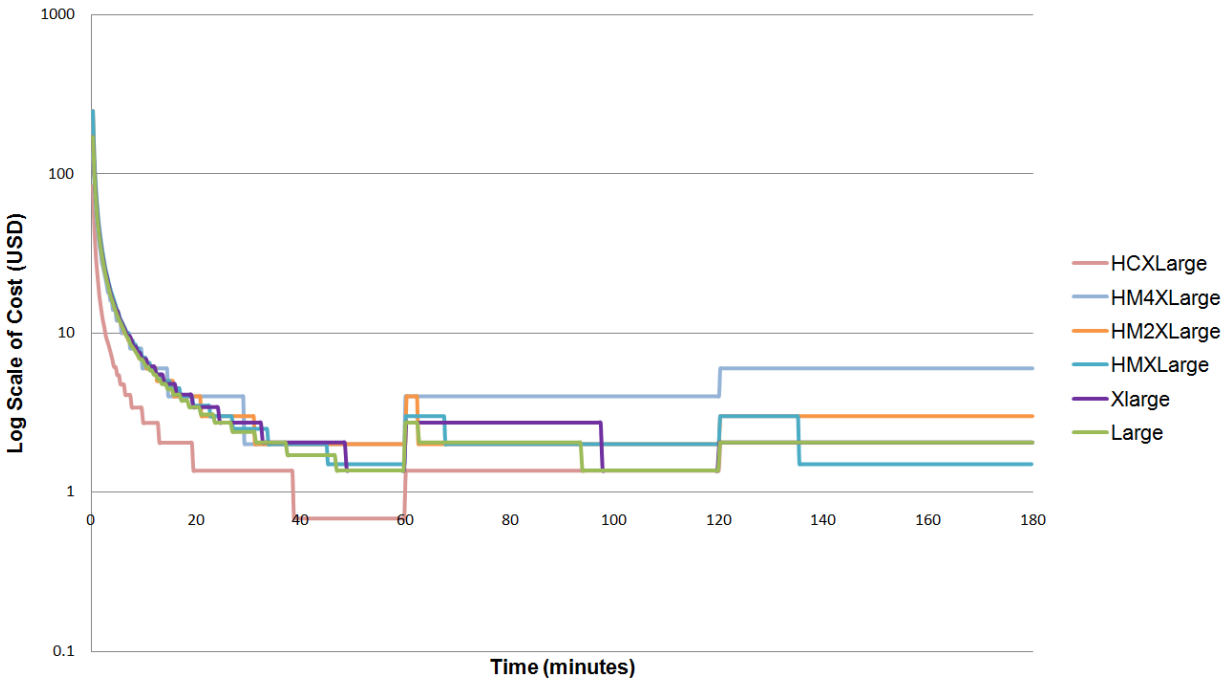

Figure 12: Multi-threaded MATLAB benchmark Cost vs. Time (1000 Runs)

In the case of a multi-threaded MATLAB benchmark, shown in Figure 12, HCXLarge once again outperforms all of the other instances. Please note that the increase in cost after one hour is a result of the HCXLarge's ability to complete 1000 trials of the MATLAB benchmark in a single hour; after that point, the instance will not utilize all of its cores, and will therefore be wasting efficiency.

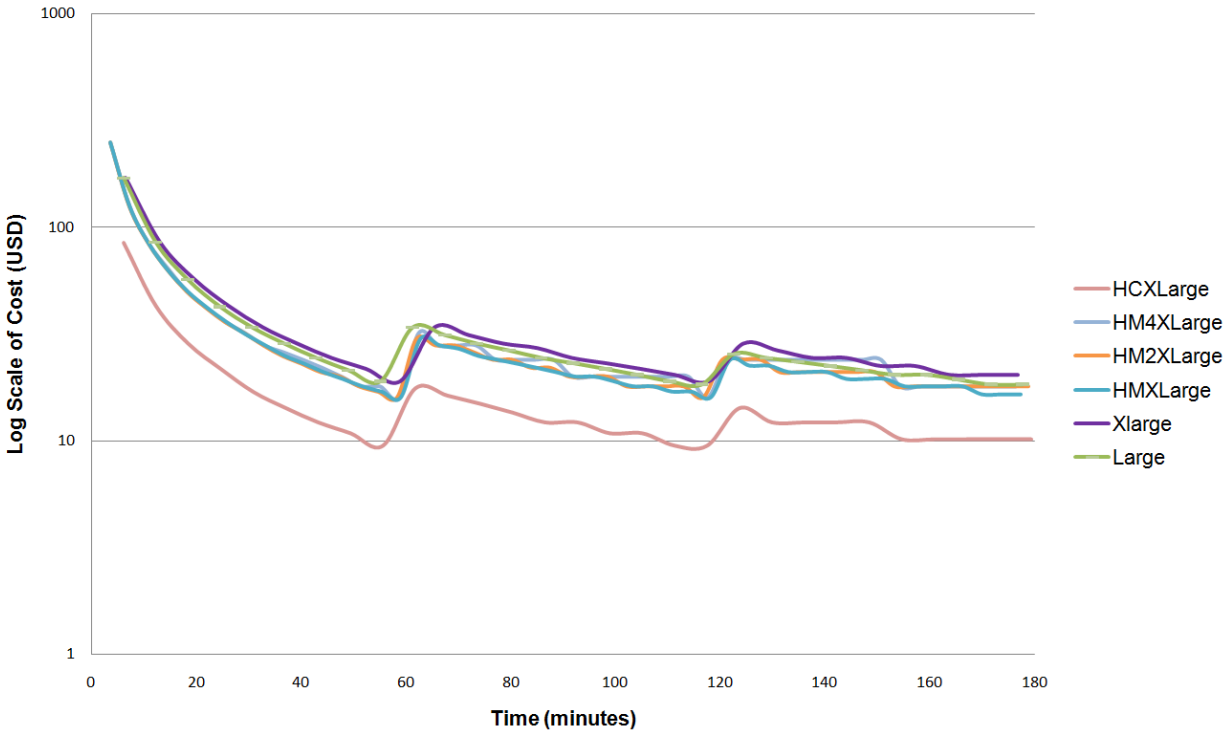

Figure 13: Multi-threaded MATLAB Case Study Cost vs. Time (1000 Runs)

As with the previous two benchmarks, the HCXLarge is also the most cost-efficient choice for the MATLAB case study (Figure 13). It also completes the execution of 1000 trials at a lower price earlier than most of the other instances reach their lowest cost. For real-world MATLAB applications, it would be most cost effective to choose the HCXLarge, provided that the application is multi-threaded.

5.3 Cost vs. Time to Solution Discussion

The analysis shown in Section 5.2 can help determine which of the various on-demand instances are suited to solving a given problem. Although much of the focus was on massively parallel applications, all of the concepts that were discussed can be applied to any single or multi-threaded program.

In the case of single-threaded applications, the Large and HMXLarge had by far the lowest cost to solve a given problem. As was previously mentioned, unless a problem is needed instantaneously, it is always advisable—and cheapest—to complete a task in one hour. This is due to the way that costs are associated with EC2. Every hour that an instance is in use, it costs a certain amount of money, and that instance can complete a given number of tasks. By offloading some of those tasks to an additional instance, it is possible to lower the time to solution proportionally to the number of instances used, while simultaneously keeping the cost constant. While this may seem counterintuitive, it is actually quite simple to prove.

Take, for example, the following scenario: an instance typically takes 5 hours to complete a batch of 50 tasks, and it costs \$0.50 per hour to run the instance. According to this information, it will cost $\$0.50 \times 5\,hours \times 1\,instance = \$2.50$ to complete the execution of all 50 tasks on a single instance. Suppose that we decided to create 4 new instances, and offloaded 10 tasks to each of them; the resulting cost to complete the problem set in $\frac{5\,hours}{5\,instances} = 1\,hour$ would be $\$0.50 \times 1\,hour \times 5\,instances = \$2.50$. This fixed cost per problem is the basis for the entire analysis in the previous section, due to its unique nature.

For multi-threaded applications, the HCXLarge instance was almost twice as cheap as the

next best instance. This is due to the fact that the HCXLarge instance provides the user with a large amount of computing power spread over 8 cores, all at the nominal rate of $0.68 per hour; this gives the HCXLarge a cost per core per hour of only $\frac{\$0.68}{8} = \$0.085$. Similar resources are offered by the HM4XLarge instance, but the marginally shorter execution times are almost never worth the associated costs.

The plots in section 5.2 can also be used to determine the best option for researchers based on their need. For example, if cost is not a concern but time to solution is of highest priority, the HM4XLarge instance would definitely finish in the least amount of time. The issue here is that the cost is exponentially greater as the time approaches its minimum value. It is always possible to complete a set of similar tasks in the execution time of the longest task, and while this may initially seem like a good idea, it is best to look at the cost.

Using the same sample as above, it will take each task approximately $\frac{5\,hours}{50\,tasks} = 6\,minutes$ to complete. If it was necessary to have the entire batch completed in 6 minutes, then it would cost $\$0.50 \times 1\,hour \times 50\,instances = \$25.00$ to complete. The minimum cost can only be achieved on hourly intervals, because it minimizes the amount of time wasted for each instance; in this way, even though it only takes 6 minutes to complete the execution, we are still charged for a full hour of usage for each instance that we started.

It is due to this concept, that problems have a certain minimum cost associated with them, that EC2 becomes an interesting option for embarrassingly parallel applications. There is no need to bog down in-house servers for a week at a time if it is possible to finish the same task on the cloud in an hour, for the absolute minimum price. These ideas will be further evaluated in Section 6.

6 Cost Analysis

The purpose of our cost analysis was to determine whether there are any benefits associated with using EC2 as a replacement to in-house servers. As discussed earlier, there has been limited research conducted in prior work that involves such a cost comparison. Both performance and cost had to be taken into consideration in order to fairly evaluate the advantages and disadvantages of migrating current resources to EC2.

6.1 Cost Comparison: EC2 vs. Tesla

We first consider whether there is an actual cost-benefit when using EC2 compared to a single physical server such as Tesla. The annual as well as long term costs pertinent to this analysis are calculated in this section.

6.1.1 Amazon EC2

The EC2 service has two primary costs associated with it: instance usage and storage costs. The instance-usage cost stems from the hourly rate that is applied to all instances, wherein partial hours are charged as full hours. It is possible to quickly exceed a budget if care is not taken when using the on-demand EC2 instances.

The storage cost is for the use of the Simple Storage Service (S3) and the Elastic Block Storage (EBS) provided by Amazon for the EC2 instances. In the event that large amounts of data are not being handled by this storage, the cost for using S3 is trivial. The standard rate for using EBS volumes is $0.10 per GB-month of provisioned storage, and $0.10 per million I/O requests [6]. Storage of EBS snapshots in S3 is charged at a rate of $0.15 GB-month of data stored. Data transferred "into" Amazon EC2 is charged at $0.10 per gigabyte transferred, while data transferred "out" of Amazon EC2 is charged based on the amount of data in gigabytes being transferred.

If one were to use the on-demand HM4XLarge instance on EC2 throughout the year instead of using Tesla, the cost would amount to approximately $17,520 per year. Although not obsolete, Tesla will likely be replaced by a newer server after 2 years, so the argument stands that if EC2 can be equal or lesser in value to Tesla's combined costs, it would be financially viable to invest in. We began our analysis by adding the initial cost of Tesla—about $12,000[30]—to the cost of cooling and powering the server for 2 years—which was $1508 per year. The cost of power was calculated by multiplying the server's wattage, 717 Watts as obtained from the Energy Star specifications sheet [31], by the amount that WPI currently pays for electricity ($0.12/kW-hr). Similarly, the BTU output of Tesla (2446.5 BTU/hr) was converted to watts and also multiplied by WPI's hourly cost for power. These methods used to calculate Tesla's yearly costs have been described in the following section as well. Both of these amounts came out to $754 per year, each—giving us the total 2 year cost for Tesla of $15,014.84. For the same amount of money, the largest on-demand instance of EC2 can be run for about one year, which indicates that it costs about twice as much as Tesla, or approximately $35,040. However, this is only the case for on-demand instances, like the ones used for our research.

Amazon also provides special reserved instances for 1 or 3 year contracts at reduced rates; they are identical to the on-demand instances in terms of computing power, but have significantly lower hourly rates. With reserved instances, it is possible to reserve an HM4XLarge instance and continually run it for one year for the price of $11,256.80 or three years for a price of $25,870.40. The three year cost of Tesla was estimated at $16,522.25, as shown in Section 6.2. We now see that there is a significant difference in terms of cost between Tesla and EC2 (with EC2's costs being approximately 1.566 times that of Tesla's), and even factoring in the costs of additional

hardware and floorspace will not tip the scale in favor of EC2. This supports the analysis of Section 4.6, which showed that the cost per compute unit of HM4XLarge was higher than that of Tesla. It is apparent, based on the above analysis, that the initial one-year cost of EC2 is cheaper than the one-year cost of purchasing an in-house server, but this is simply a result of the investment cost of buying the server. There are several managerial concerns that must be considered before considering migrating to the cloud, which will be discussed later in this section.

6.2 Cost Comparison: EC2 vs. Physical Servers

Of more relevance to the academic and institutional environments is the analysis presented in this section. We have examined the various costs associated with maintaining a large network of servers, much like those used by the ECE department at WPI. After speaking with a member of the department, we found that the ECE department purchases and uses its servers for periods of up to or exceeding 10 years for a variety of purposes, from remote login for students and faculty to mail servers to extreme computation. Based on this information, we developed a 9-year cost projection that accounted for all of the relevant costs associated with owning these servers and compared it to the costs associated with owning comparable EC2 servers for the same duration.

6.2.1 The ECE Department Servers

The ECE department owns a total of 21 servers in its building (including Tesla and Henry), each of which has been purchased and used for various purposes, such as those mentioned above. We selected a total of 10 servers that were being actively employed by the department for some form of public use.

The first step upon obtaining this list of servers was to estimate the amount that the ECE department spent at the time of purchasing these servers. It must be noted that some of these servers were purchased as far back as 10 years ago, and records for most of the machines were either lost or forgotten. We also had to account for various institutional discounts that the university received upon purchasing these servers. Once all of the prices had been researched, we then went into the specifications of each of the servers and found the wattage W required to power the unit as well as the thermal dissipation K in BTU/hour. The thermal dissipation was converted to watts in order to estimate how much it cost to cool each server. Once the wattage for both powering and cooling the servers was obtained, it was converted to kilowatts and multiplied by the rate that WPI pays for electricity ($0.12 per kW-hr), in order to estimate the total cooling and powering costs associated with each server. These costs were then multiplied by the number of hours in a year in order to estimate the total amount paid each year to cool and power the servers. To finish the cost projection, these costs were then multiplied by 9 to calculate the 9 year cost of cooling and powering the servers. Table 31 summarizes all of the costs calculated towards this effect, and Equation 3 shows how these values were acquired.

$$MC = \left(\frac{W}{1000} + \frac{K}{3412.1416} \right) \times 0.12 \times 24 \times 365 \times 9 \tag{3}$$

Server	Price When Purchased (USD)	Power (Watts)	Thermal Dissipation (BTU/hr)	9-Year Power Cost (USD)	9-Year Cooling Cost (USD)	9-Year Projected Cost (USD)
Tesla	12000.00	717.00	2,446.50	6,783.39	6783.38	25566.77
Hertz	6330.00	450.00	850.00	4,257.36	2356.78	12944.14
Henry	6330.00	450.00	850.00	4,257.36	2356.78	12944.14
Ohm	6330.00	450.00	850.00	4,257.36	2356.78	12944.14
Mho	6330.00	450.00	850.00	4,257.36	2356.78	12944.14
Amp	25600.00	760.00	1,228.00	7,190.21	3404.86	36195.07
Dot	8376.00	500.00	614.30	4,730.40	1703.26	14809.66
Volt	5200.00	465.00	1,100.00	4,399.27	3049.96	12649.23
Maxwell	3866.00	275.00	1,033.00	2,601.72	2864.19	9331.91
Hutt	21006.00	270.00	1,796.00	2,554.42	4979.75	28540.16
Projected Server	12671.00	478.70	1161.78	4,528.88	3221.25	20421.14
Total:	114039.00			49817.73	35433.77	**199290.51**

Table 31: ECE Department Server Costs

You will notice that in addition to the 10 department servers, we have also included costs for a "Projected Server," to anticipate future costs. To calculate the purchase cost of this "server", we took the sum of all of the server purchase costs and divided it by the number of years between the deployment of the oldest and newest servers. This gave us an average cost per year spent on buying new servers, which we multiplied by the number of additionally projected years, which in this case was 1 (since the span from the oldest to newest server was 8 years). From the above table, we calculated the total purchase costs for these 10 servers to be approximately $114,039.00. According to our estimations, the total powering and cooling costs for 1 year are approximately $5,535.30 and $3,937.09 respectively, and the corresponding costs for 9 years are approximately $49,817.73 and $35,433.77 respectively. We initially factored maintenance costs into the analysis, but after speaking with a member of the department who maintains the servers, we were informed that because all maintenance is done in-house, there are no real costs associated with maintaining the servers. Although this faculty member's salary could factored into the total cost, it would be unnecessary because the department would require their services just as much for maintaining network connections to the EC2 instances, as well as configuring and setting up new terminals to access the service.

We were also informed that the department makes use of its 3 or 4 year warranty that generally comes along with the purchase of the servers, and so if any maintenance is needed, it is normally covered by these. Since they house redundant systems, any other maintenance is generally done in-house without causing much impact to the students and faculty. Using all of these numbers, the total minimum expenses borne by the ECE department for purchasing and maintaining the above servers for 9 years is approximately $199,290.51. The cost does not include aspects such as storage costs or additional hardware, as these are typically much lower than the other associated costs.

In order for it to be financially viable to migrate over to the cloud, the minimum total cost of EC2 would have to be lower than the total cost of owning an in-house system.

6.2.2 EC2 Reserved Instances as Servers

We now examine the cost of reserving instances that equal the computing power currently provided by network infrastructure in the ECE department. This was done by first estimating which instance, or combination thereof, would align most closely with each of the physical servers. During our primary analysis we made the mistake of matching servers directly on their current hardware specifications. It is important to remember that each of the servers in the ECE department were purchased when they were some of the best on the market, and as Table 31 shows, we paid a premium. The cost analysis for this section is greatly complicated by Moore's law, which states that the number of transistors on an integrated circuit will double roughly every two years[32]. What this implies is that while the cost for high-end servers will remain about the same, their performance will double; a 2.8GHz dual-core processor in 2007 cost the same amount as a 2.8GHz quad-core processor in 2009.

What this meant for our analysis was that, for our server-instance estimates, we needed to determine which servers would have been considered top-of-the-line, and which would have been only average. Table 32 depicts the pairings of servers to historically comparable instances.

Calculating the initial 3-year cost was simply a matter of multiplying the cost per instance-hour by the number of hours in 3 years and adding the one-time fee. The cost for the second 3-year term should be significantly lower than the cost for the first term, and the best estimate of the costs for each server was to "downgrade" each server to the next lowest instance. This is the closest approximation to Moore's law, while staying within the EC2 pricing structure. In order to represent this data in an easily understandable format, we have divided a page into a flowchart of sorts which depicts the cost of each three year deployment of EC2 instances.

Server	Comparable Instance Type	3-Year Initial Cost (USD)	Cost per Instance Hour (USD)	Cost After 3 Years (USD)	Total Cost (USD)
1st Term:					
Tesla	HM4XLarge	8000	0.68	25870.4	
Hertz	HM2XLarge	4000	0.34	12935.2	
Henry	HM2XLarge	4000	0.34	12935.2	
Ohm	HM2XLarge	4000	0.34	12935.2	
Mho	HM2XLarge	4000	0.34	12935.2	
Amp	HM4XLarge	8000	0.68	25870.4	
Dot	HM2XLarge	4000	0.34	12935.2	
Volt	HMXLarge	2000	0.17	6467.6	
Maxwell	HMXLarge	2000	0.17	6467.6	
Hutt	HM4XLarge	8000	0.68	25870.4	
Projected Server	HM4XLarge	8000	0.68	25870.4	
				Total:	181092.8
	2nd Term:				
	HM2XLarge	4000	0.34	12935.2	
	XLarge	2800	0.24	9107.2	
	XLarge	2800	0.24	9107.2	
	XLarge	2800	0.24	9107.2	
	XLarge	2800	0.24	9107.2	
	HM2XLarge	4000	0.34	12935.2	
	XLarge	2800	0.24	9107.2	
	Large	1400	0.12	4553.6	
	Large	1400	0.12	4553.6	
	HM2XLarge	4000	0.34	12935.2	
	HM2XLarge	4000	0.34	12935.2	
				Total:	106384
	3rd Term:				
	XLarge	2800	0.24	9107.2	
	HMXLarge	2000	0.17	6467.6	
	HMXLarge	2000	0.17	6467.6	
	HMXLarge	2000	0.17	6467.6	
	HMXLarge	2000	0.17	6467.6	
	XLarge	2800	0.24	9107.2	
	HMXLarge	2000	0.17	6467.6	
	S	350	0.03	1138.4	
	S	350	0.03	1138.4	
	XLarge	2800	0.24	9107.2	
	XLarge	2800	0.24	9107.2	
				Total:	71043.6
				Overall Total:	**358520.4**

Table 32: EC2 to In-House Server Pairings

The previous table was a best case scenario, demonstrating the extreme prices associated with leasing out EC2 instances. No data transfer costs were taken into consideration, and the cheaper Linux pricing was used for all hourly instance costs. The total cost over the 9-year period was approximately $358,520, which was significantly greater than the costs associated with the in-house servers. It should also be noted that the step down from the HM2XLarge went to the XLarge, and likewise the XLarge stepped to the HMXLarge, instance, as it was the next lowest price.

6.3 Discussion

There are several aspects that must be taken into consideration when attempting to utilize the EC2 service as a cost-effective form of computing. This section discusses the significance of both of the analyses completed above, providing a better understanding of the overall cost-benefit of EC2.

6.3.1 Tesla vs. EC2

Prior to commencing our cost analysis, our group was informed that the annual server budget for the ECE department is roughly $12,000, which would later support our estimate of $12,600 for the average annual purchase costs of servers—as demonstrated by the initial cost of the projected server in Table 31. We also stayed in close contact with members of the department in order to verify the accuracy of different cost estimates, and ensured that all of the costs that we factored into our analysis were actually relevant to the department.

The most important point that must be considered when replacing a physical server with online instances is the overall cost-benefit to the department, since this analysis is directly aimed at educational research institutions such as WPI. From our analysis we note that it is much more cost effective to invest in private servers, rather than migrating to the cloud—regardless of whether or not there is already a server infrastructure in place. Although our findings indicate that it would not be advantageous to completely transition to the EC2 servers, by budgeting a certain percentage of department funds towards EC2-related activities it would enable researchers to quickly complete time-critical applications without completely locking up the department servers.

While it is still economical to do so, there are several usage constraints that can make such an option difficult. In the case of the ECE department, all servers are connected via the WPI network, not only to thousands of student and faculty accounts, but also to license and backup servers located on campus. The university currently allows access to most of these services, the licenses in particular, over only the WPI network due to various license agreement terms with software providers. Adding in one instance outside of the WPI network to the set of servers in the ECE department may cause difficulty in acquiring licenses for the instance reserved, as well as adding this external instance to the WPI network. Additionally, the data within the ECE department that is stored in its servers is backed up on a daily basis onto the department's privately owned storage servers. While backing up the information stored in the reserved instance is certainly possible using EC2's storage options such as S3, it could pose issues such as security of the information, as well as the fact that this backing up is occurring outside of WPI's realm of control.

An additional inconvenience posed by reserving instances is the limited 3 year contract which could potentially make it difficult for the department to back up data or provide continuous service throughout the year for several years at a time. It is necessary for the department to have servers up and running 24 hours a day all year due to the load that it experiences year round, not only from graduate students or professors running their research programs, but also

undergraduate students using the servers for their coursework. Finally, we must also not overlook the fact that the performance of the EC2 instances was still sub-par when compared to Tesla's performance. Apart from the usage convenience offered by purchasing one's own servers, the performance provided by the physical server may still be far more favorable than the EC2 instances, which can be a big deciding factor for research-intensive environments.

Based on these issues, it is important to note that while reserving an EC2 instance can not only provide very little cost-benefits, it may not be the best solution for well established research institutions such as WPI, that already have server labs and secured private networks set up. In order to examine this further, we extended our analysis to determining the costs associated with replacing an entire set of physical servers rather than only one, using reserved instances.

6.3.2 Server lab vs. EC2 instances

Our second cost analysis dealt with understanding the various expenses associated with investing in EC2 as a replacement to the all of the computational resources offered by the servers currently owned by the ECE department. We found that the overall costs associated with purchasing and maintaining a set of 10 servers for a period of 9 years is about 43% less than that of reserving a set of EC2 instances for the same purpose. From this very large difference, we observe that there are indeed little to no cost benefits that can be observed when reserving a large set of instances that provide the same amount of computational resources as physical servers.

As with the earlier analysis, there are usage constraints that could cause EC2 to be a less desirable option for research bodies with well established server setups. As mentioned earlier, WPI has access to a large set of software licenses that are specific to its secure and private network. If the ECE department alone was to migrate to the EC2 service for its computing needs, it would pose an issue with the campus-wide license agreements with various software distributors. Additionally, daily backups of the all the servers containing ECE department data could result in exorbitant amounts of storage (much more than only backing up one instance) that might not necessarily cost as much if physical storage was purchased along with physical computing resources. As we mentioned earlier, our server administrator stated that the ECE department spent about $1200 on the purchase of additional disk drives that were mainly used for disk failure backups, which does not add a significant amount to the total cost observed for all the servers. ind We must also keep in mind, as with the previous analysis, that the performance of the EC2 instances based on our benchmarks has not been as commendable as that of our most powerful (and newest) physical server found in the ECE department. There is always the concern that computing space provided by EC2 is shared with other users due to the very nature of the virtual machine usage. This can not only cause a reduction in performance and inconsistency of results, but can also bring the issue of information security. If the ECE department were to migrate entirely to EC2 for its computing and storage purposes, there lies a substantial risk of loss or sharing of data which can be problematic for researchers handling sensitive material or intellectual property matters. These issues could potentially hamper interest on the part of professors working on confidential material, as well as WPI's own policies about information security overall.

Based on these analyses, it would make sense to use EC2 in order to offset the extreme loads often experienced by the ECE department servers, rather than moving entirely to the EC2 service for all of our computing needs,. For example, there are times where the ECE department servers are loaded up to such a point that either their use is prevented or their performance is greatly compromised. In such cases, researchers can choose to invest in the on-demand instances offered by EC2, which could allow them to obtain their results far quicker than an overloaded campus server. Finally, our analysis shown in section 5 describes the instance that is best suited for different kinds of applications based on cost as well as time, which could help researchers pick the appropriate instance for their specific workloads.

An additional use of the EC2 reserved instances could be for smaller businesses or organizations, such as start-ups, that may not necessarily have the physical space or infrastructure to accommodate a large amount of servers. In such cases, it would prove useful for them to invest in the EC2 reserved instances, which would provide them with sufficient computing power and storage at potentially agreeable cost. These businesses would also benefit with the cloud environment's overall advantages, such as freedom from hassles dealing with server space, powering, maintenance and so on.

EC2 is therefore observed to be an excellent option for those who do not have access to powerful computing servers, or whose applications are not as intensive that they may require computation for large periods of time. In simpler terms, applications that are smaller and less intensive in nature can be effectively run on EC2, for a reasonable cost and rivaled performance.

7 Overall Discussion

Our primary focus with this research was to evaluate the feasibility of the Amazon EC2 service to address the computational needs experienced by scientific research-oriented environments, such as universities. In order to accomplish this in an impartial manner, it was important to take into account the performance as well as various costs associated with owning and maintaining a physical machine versus purchasing on-demand or reserved EC2 instances online. The results obtained from this report, and performance as well as cost evaluations have helped us in developing an overall conclusion on the usability of EC2.

In all the tests that were performed, whether they were compute intensive, memory based, or both, it was found that Tesla outperformed all the instances almost every single time. This is commendable as Tesla was quite often being used by other students that added a considerable load on to it. Although it was observed that there were times when Henry did not perform as well as Tesla in certain tests, it must be taken into account that Henry was purchased approximately 2 years prior to Tesla; the ECE department purchased Tesla about a year before the commencement of our research. Factoring in current technology trends, namely Moore's law, it is no surprise that Henry's hardware was slower than the hardware that supports the EC2 instances, let alone Tesla.

We also observed that despite lagging in performance, EC2 is capable of providing reasonable costs for the services it offers on a smaller scale. Computational power could be acquired from EC2 at a reasonable rate, provided the type of investment was confined to a single or a couple of instances, mostly of the reserved kind. Larger scale use of EC2, particularly to replace server facilities, would not be advisable based the economic analysis conducted in Section 6. We then noted the various managerial and usage issues that could come into play when using EC2 within a secure networked environment such as WPI, and found these as important aspects to take into account when evaluating one's need to migrate to cloud computing.

We now highlight some of the issues that came up while we benchmarked all of the systems chosen for this project. These are issues that one must be aware of when working with both types of resources, physical as well as virtual.

7.1 Issues

During the course of this project, it was realized that apart from the various performance drawbacks on the part of the EC2 instances, there was also a considerable amount of inconvenience associated with the on-demand instance use. For example, the AWS console repeatedly restarted instances on its own, that were previously shut down after being used. This glitch resulted in an unnecessary expenditure on the part of the group for resources that were not even used.

Another issue that arose was the selection of operating systems supplied by EC2 for launching instances. Although it is wise of Amazon to constantly upgrade the systems and software that they offer, it was particularly inconvenient when the Fedora 8 operating system that was offered was suddenly replaced without warning with SUSE Linux Enterprise. This was a problem as the group had already commenced work on the Fedora 8 instances that had been created while the operating system was being offered. Additionally, it was difficult to upload our own operating system onto their servers, causing greater inconvenience when setting up our instances. However, this issue was overcome by cloning the Fedora 8 image that was originally used, and generating other instances based off of this image in order to maintain uniformity of operating systems.

Occasionally there were instances that appeared to be "unavailable" in the geographic region specified by the group, although EC2 offers a 99.9% availability guarantee [6]. While the service does present to the user the option to open the instance in an alternative region, opting for

this alternative did not seem to launch the instance, thereby causing a standstill in benchmarking processes until the desired instance was once again available in our specified region. Additionally, there is always the possibility of sharing resources with other virtual instances on the EC2 machines. This is most commonly the reason behind inconsistent results of benchmarks due to large loads on physical machines that are being shared amongst various people's virtual machines – and can also lead to insecurity of information being utilized or stored on the virtual machine.

There were also issues present when using the ECE servers which were not observed on the EC2 instances. For example, when first attempting to run RAMSMP, the group accidentally exceeded the memory specifications of Tesla causing the server to force itself to shut down. This caused considerable inconvenience to those individuals that were currently running various academic-related workloads on the server. However, it was observed that when this sort of test was run on EC2, the instance could either recover from the situation on its own, or could be restarted via the AWS console thereby reloading the entire instance without causing anyone else any loss of information. It must be noted, though, that the restarting of the instance was possible only because the instance was privately owned by the group, and therefore was not being used by any outside entity whose work could be disrupted. This capability of EC2 was also particularly useful in testing the RAMSMP benchmark in order to establish appropriate boundaries when testing memory limits.

It was also not possible for the group to run the SPEC benchmarks on the ECE servers, as the installation of the benchmark required administrative privileges which the group was not permitted to obtain. It is fortunate that the SPEC repositories contained the results for the entire suite for the Dell PowerEdge R610 server (but not the Sunfire X2200 M2 2.80GHz server), that could be used when comparing Tesla to EC2.

Lack of administrative privileges also posed an issue when attempting to standardize all software versions across the instances and machines, as the versions that were installed on Henry and Tesla cannot be changed. The software versions that were found on the ECE servers were older compared to compiler and application versions that are currently in the market. Therefore, care had to be taken to find and install older versions of the software onto the EC2 instances in order to maintain uniformity when benchmarking.

Given these issues with the department servers, it was certainly convenient to have a machine that was dedicated only to the group, rather than be shared with several other users and their heavy workloads. This allowed much greater flexibility in terms of resource usage, administrative privileges, and so on, despite the instances' lacking in performance.

7.2 Cost Comparison

The cost analysis estimated by the group as well as the Amazon EC2 calculator both show that the cost of purchasing and running Tesla for 3 years is much less than purchasing a reserved instance of similar computing capabilities on EC2. These costs did not include any other additional expenses that may be incurred during the life-span of both the servers as well as the instances, but as far as the initial purchase and maintenance costs go, EC2 certainly provided the least economical results for the usage of a single reserved instance.

As part of our cost analysis, we compared the cost of maintaining not only one physical server, but also a complete server lab against purchasing reserved instances online to provide the same computing resources. It was found that purchasing and maintaining the ECE servers cost much less (by about 43%) when compared to purchasing and renewing contracts for reserved instances that could be put in place of the servers currently used by the ECE department. Following this cost-benefit analysis, we delved deeper into understanding the various factors that must be considered (other than expenses) when migrating the ECE department load to the EC2 service.

We discussed the managerial concerns associated with EC2 reserved instances, which could make its overall benefit appear less appealing. For example, permission to access licenses, personal network accounts as well as large scale data storage over EC2 may not be an option that institutions such as WPI would like to undertake. This can be for various reasons, mostly due to preservation of information security and intellectual property, as well as abiding by the terms of various software license agreements. It is much more preferred for the university to maintain its information within their secured network's protection, as well as more convenient for the various departments to have their servers in-house, rather than being concerned about external network connections, unknown power or system failures, limited support staff, and so on.

Despite these advantages, it may not always be possible to invest in a large server, particularly for small businesses, individual workers or researchers. In such cases, the cost vs. time to solution analysis plays a key role in helping the individual or organization in determining which instance would best suit their need. The cost versus time analysis provides the reader with the opportunity to set boundaries as to what sort of cost they are able to manage, or what kind of wait period they are willing to bear. The instances most suited for both single as well as multi-threaded applications at the most economical value were found using this analysis. As a result, the user can choose an instance that delivers the solution relatively quicker, with the trade-off being a large amount of money, or vice versa. This sort of comparison is particularly helpful for individuals that are limited on budget or time, be it for time sensitive research or cost-efficient results.

Overall, it appears as though that the EC2 service is more suited for sporadic use rather than continuous long term use. It would make more sense for a researcher to invest in on demand instances as and when their workload demands urgent response which may or may not be facilitated by on-campus servers. In such cases, the researcher could easily purchase an instance online for as long as they need, obtain their solution and close down the instance with no commitment to the same, and with adequate performance. Such flexibility was therefore the strongest point observed about the EC2 service.

8 Conclusion

This project was designed in order to identify the key differences between investing in cloud computing versus a physical machine to address the needs of a scientific research-based institution such as WPI. With the amount of flexibility and the wide variety of usage options offered by modern cloud computing providers such as Amazon EC2, it was important to evaluate whether this new, upcoming form of computing was in fact capable of replacing traditional means of processing information and complex problems. In addition to evaluating its performance, it was also important to take into account the various costs associated with cloud computing, specifically the EC2 service, and compare them against those expected from investing in physical servers.

It was found, as a result of the various analyses conducted in this project, that it is currently more practical to invest in a large computing server rather than in on-demand or reserved EC2 instances, specifically for research-oriented environments such as the WPI ECE department. This can be attributed to the much more reasonable costs, considerable performance boost observed in physical computing resources, along with the security that it provides for information, as well as flexibility in terms of networking with the rest of the university campus and using campus-wide shared utilities and software.

There are still issues with privately owned servers in terms of availability, load, and administrative privileges (or the lack thereof), which may hamper time-sensitive research, or skew results because of erratic performance by the server due to large loads. In times like this, it may be useful for the researcher to invest, for a brief period of time, in EC2 instances that can provide them their results at a quicker rate and at rivaled performance, rather than waiting until the physical server is able to process more load, thereby delaying research.

It is also important to take into consideration the various other resources that EC2 has to offer, that were not looked into by this project. These resources include the various cluster computing instances, the GPU instances, Spot instances, and so on. Looking into these instances can potentially provide additional insight as to whether the EC2 service is still capable of providing acceptable, if not better results when using any of these instance types, for a reasonable cost. Additionally, the aforementioned GPU instances were only a recent addition to the EC2 resources, thereby making them fair game for further performance and cost analysis of EC2's cluster computing resources.

Apart from this, given a long enough research period, it may be wise to invest in reserved instances in order to estimate the kind of performance one can expect. This includes aspects such as overall availability, frequency of system failures if any, network behavior and security, and various other concerns raised in this report about cloud computing overall. This could then be used to measure the similarities in behavior between the reserved instances as well as physical servers, in order to estimate which option provides a better quality for its price.

Future work can also take into account the costs associated with cloud computing versus those faced by different organizations, such as larger corporations or businesses. These establishments tend to face a larger cost when dealing with on-site servers, due to maintaining large server labs, leasing office space, and other such details that need not be taken into consideration when evaluating WPI's cost benefits. This sort of analysis can aid in determining whether there are conditions under which EC2 is in fact a viable computational resource for its associated costs. For example, in the case of a small private company, there may not be a largely established network (such as the WPI campus), that one would need to worry about in terms of licensing or networking issues. In such cases, EC2 would be ideal in terms of providing reasonable computing services at modest costs.

The results detailed in this project report were able to answer several questions that address the overall feasibility or performance comparison of EC2 versus a large dedicated physical server, based on single-node computing. We were also able to further endorse our results by using

the standardized and widely accepted SPEC benchmarks. Overall, we were able to demonstrate the differences in performance as well as cost between the EC2 instances and physical machines.

References

[1] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, and P. Maechling, "Scientific Workflow Applications on Amazon EC2," *2009 5th IEEE International Conference on EScience Workshops*, pp. 59–66, 2010. [Online]. Available: http://arxiv.org/abs/1005.2718

[2] J. Rehr, F. Vila, J. Gardner, L. Svec, and M. Prange, "Scientific Computing in the Cloud," *Computing in Science and Engineering*, vol. 99, no. PrePrints, 2010.

[3] A. Khajeh-Hosseini, D. Greenwood, and I. Sommerville, "Cloud Migration: A Case Study of Migrating an Enterprise IT System to IaaS," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, 2010, pp. 450 –457.

[4] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. P. Anderson, "Cost-benefit analysis of Cloud Computing versus desktop grids," in *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–12. [Online]. Available: http://portal.acm.org/citation.cfm?id=1586640.1587662

[5] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," *ACM*, vol. 53, no. No.4, pp. 50–58, 2010.

[6] Amazon Web Services, "Amazon Elastic Compute Cloud," 2011. [Online]. Available: http://aws.amazon.com/ec2/

[7] N. Guilbault and R. Guha, "Experiment setup for temporal distributed intrusion detection system on amazon's elastic compute cloud," in *Proceedings of the 2009 IEEE international conference on Intelligence and security informatics*, ser. ISI'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 300–302. [Online]. Available: http://portal.acm.org/citation.cfm?id=1706428.1706496

[8] S. Akioka and Y. Muraoka, "HPC Benchmarks on Amazon EC2," in *24th IEEE International Conference on Advanced Information Networking and Applications Workshops*.

[9] Z. Hill, J. Li, M. Mao, A. Ruiz-Alvarez, and M. Humphrey, "Early observations on the performance of Windows Azure," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 367–376. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851532

[10] J. Napper and P. Bientinesi, "Can cloud computing reach the top500?" in *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*, ser. UCHPC-MAW '09. New York, NY, USA: ACM, 2009, pp. 17–20. [Online]. Available: http://doi.acm.org/10.1145/1531666.1531671

[11] C. Evangelinos and C. Hill, "Cloud Computing for parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2," 2008.

[12] E. Walker, "Benchmarking Amazon EC2 for High-Performance Scientific Computing," *Login*, vol. 33, pp. 18–23, October 2008.

[13] MIT Information Services & Technology, "Athena at MIT." [Online]. Available: http://ist.mit.edu/services/athena

[14] D. P. Anderson and G. Fedak, "The Computational and Storage Potential of Volunteer Computing," *Cluster Computing and the Grid, IEEE International Symposium on*, vol. 0, pp. 73–80, 2006.

[15] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good, "On the Use of Cloud Computing for Scientific Workflows," in *ESCIENCE '08: Proceedings of the 2008 Fourth IEEE International Conference on eScience*. Washington, DC, USA: IEEE Computer Society, December 2008, pp. 640–645. [Online]. Available: http://dx.doi.org/10.1109/eScience.2008.167

[16] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in *2008 Grid Computing Environments Workshop*. IEEE, November 2008, pp. 1–10. [Online]. Available: http://dx.doi.org/10.1109/GCE.2008.4738445

[17] M. Klems, J. Nimis, and S. Tai, "Do Clouds Compute? A Framework for Estimating the Value of Cloud Computing," in *Designing E-Business Systems. Markets, Services, and Networks*, ser. Lecture Notes in Business Information Processing, W. Aalst, J. Mylopoulos, N. M. Sadeh, M. J. Shaw, C. Szyperski, C. Weinhardt, S. Luckner, and J. Ster, Eds. Springer Berlin Heidelberg, 2009, vol. 22, pp. 110–123. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01256-3_10

[18] S. Bibi, D. Katsaros, and P. Bozanis, "Application development: Fly to the clouds or stay in-house?" in *Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on*, 2010, pp. 60 –65.

[19] Amazon Web Services, "The Economics of the AWS Cloud vs. Owned IT Infrastructure," 2009. [Online]. Available: http://media.amazonwebservices.com/The_Economics_of_the_AWS_Cloud_vs_Owned_IT_Infrastructure.pdf

[20] ——, "Amazon EC2 Reserved Instances," 2011. [Online]. Available: http://aws.amazon.com/ec2/reserved-instances

[21] ——, "User Guide: Amazon EC2 Cost Comparison Calculator," 2011. [Online]. Available: http://aws.amazon.com/economics/

[22] J. Gustafson, D. Rover, S. Elbert, and M. Carter, "The design of a scalable, fixed-time computer benchmark," *J. Parallel Distrib. Comput.*, vol. 12, pp. 388–401, August 1991. [Online]. Available: http://portal.acm.org/citation.cfm?id=115832.115846

[23] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design, Fourth Edition: The Hardware/Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design)*. Morgan Kaufmann, 2008. [Online]. Available: http://www.amazon.com/Computer-Organization-Design-Fourth-Architecture/dp/0123744938%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0123744938

[24] "Python Core Development." [Online]. Available: http://coverage.livinglogic.de/index.html

[25] S. Appel and I. Petrov, "Performance Evaluation of Multi Machine Virtual Environments," in *Proceedings of the 2010 SPEC Benchmark Workshop, Paderborn, Germany*, 2010. [Online]. Available: http://www.dvs.tu-darmstadt.de/publications/pdf/ApPeBu2010.pdf

[26] "International SPEC Benchmark Workshop 2010." [Online]. Available: http://www.spec.org/workshops/2010/paderborn/program.html

[27] "Standard Performance Evaluation Corporation (SPEC)," [SPEC CPU2006]. [Online]. Available: http://www.spec.org/cpu2006/

[28] "RAMSpeed - A RAM and Cache Benchmarking Tool." [Online]. Available: http://alasir.com/software/ramspeed/

[29] AMD, "AMD Software Optimization Guide for AMD64 Processors," 2005. [Online]. Available: http://support.amd.com/us/Processor_TechDocs/25112.PDF

[30] Dell, "Server, Storage and Networking." [Online]. Available: http://www.dell.com/

[31] Energy Star, "ENERGY STAR Power and Performance Data Sheet: Dell PowerEdge R610 feaaturing High-output 717W Power Supply." [Online]. Available: http://www.dell.com/downloads/global/products/pedge/Dell_PowerEdge_R610_717W_Energy_Star_Datasheet.pdf

[32] Intel, "Moore's Law: Made by real Intel Innovations." [Online]. Available: http://www.intel.com/technology/mooreslaw/

A Pystone Code

```
"""
"PYSTONE" Benchmark Program

Version:        Python/1.1 (corresponds to C/1.1 plus 2 Pystone fixes)

Author:         Reinhold P. Weicker,  CACM Vol 27, No 10, 10/84 pg.
    1013.

                Translated from ADA to C by Rick Richardson.
                Every method to preserve ADA-likeness has been used,
                at the expense of C-ness.

                Translated from C to Python by Guido van Rossum.

Version History:

                Version 1.1 corrects two bugs in version 1.0:

                First, it leaked memory: in Proc1(), NextRecord ends
                up having a pointer to itself.  I have corrected this
                by zapping NextRecord.PtrComp at the end of Proc1().

                Second, Proc3() used the operator != to compare a
                record to None.  This is rather inefficient and not
                true to the intention of the original benchmark (where
                a pointer comparison to None is intended; the !=
                operator attempts to find a method __cmp__ to do value
                comparison of the record).  Version 1.1 runs 5-10
                percent faster than version 1.0, so benchmark figures
                of different versions can't be compared directly.

                                    Modified for the purposes of the Amazon
                                        EC2 Benchmarking Team @ WPI

"""

LOOPS = 10000000

from time import clock
from time import time

__version__ = "1.1"

[Ident1, Ident2, Ident3, Ident4, Ident5] = range(1, 6)

class Record:
```

```
    def __init__(self, PtrComp = None, Discr = 0, EnumComp = 0,
                       IntComp = 0, StringComp = 0):
        self.PtrComp = PtrComp
        self.Discr = Discr
        self.EnumComp = EnumComp
        self.IntComp = IntComp
        self.StringComp = StringComp

    def copy(self):
        return Record(self.PtrComp, self.Discr, self.EnumComp,
                      self.IntComp, self.StringComp)

TRUE = 1
FALSE = 0

def main():
    mstarttime = time()
    benchtime, stones = pystones()
    mdifftime = time() - mstarttime
    print "Pystone(%s) time for %d passes = %g, %g" % \
          (__version__, LOOPS, benchtime)
    print "Clock time = %g" % mdifftime
    print "This machine benchmarks at %g pystones/second" % stones

def pystones(loops=LOOPS):
    return Proc0(loops)

IntGlob = 0
BoolGlob = FALSE
Char1Glob = '\0'
Char2Glob = '\0'
Array1Glob = [0]*51
Array2Glob = map(lambda x: x[:], [Array1Glob]*51)
PtrGlb = None
PtrGlbNext = None

def Proc0(loops=LOOPS):
    global IntGlob
    global BoolGlob
    global Char1Glob
    global Char2Glob
    global Array1Glob
    global Array2Glob
    global PtrGlb
    global PtrGlbNext

    starttime = clock()
    for i in range(loops):
        pass
    nulltime = clock() - starttime
```

```
    PtrGlbNext = Record()
    PtrGlb = Record()
    PtrGlb.PtrComp = PtrGlbNext
    PtrGlb.Discr = Ident1
    PtrGlb.EnumComp = Ident3
    PtrGlb.IntComp = 40
    PtrGlb.StringComp = "DHRYSTONE PROGRAM, SOME STRING"
    String1Loc = "DHRYSTONE PROGRAM, 1'ST STRING"
    Array2Glob[8][7] = 10

    starttime = clock()

    for i in range(loops):
        Proc5()
        Proc4()
        IntLoc1 = 2
        IntLoc2 = 3
        String2Loc = "DHRYSTONE PROGRAM, 2'ND STRING"
        EnumLoc = Ident2
        BoolGlob = not Func2(String1Loc, String2Loc)
        while IntLoc1 < IntLoc2:
            IntLoc3 = 5 * IntLoc1 - IntLoc2
            IntLoc3 = Proc7(IntLoc1, IntLoc2)
            IntLoc1 = IntLoc1 + 1
        Proc8(Array1Glob, Array2Glob, IntLoc1, IntLoc3)
        PtrGlb = Proc1(PtrGlb)
        CharIndex = 'A'
        while CharIndex <= Char2Glob:
            if EnumLoc == Func1(CharIndex, 'C'):
                EnumLoc = Proc6(Ident1)
            CharIndex = chr(ord(CharIndex)+1)
        IntLoc3 = IntLoc2 * IntLoc1
        IntLoc2 = IntLoc3 / IntLoc1
        IntLoc2 = 7 * (IntLoc3 - IntLoc2) - IntLoc1
        IntLoc1 = Proc2(IntLoc1)

    benchtime = clock() - starttime - nulltime
    return benchtime, (loops / benchtime)

def Proc1(PtrParIn):
    PtrParIn.PtrComp = NextRecord = PtrGlb.copy()
    PtrParIn.IntComp = 5
    NextRecord.IntComp = PtrParIn.IntComp
    NextRecord.PtrComp = PtrParIn.PtrComp
    NextRecord.PtrComp = Proc3(NextRecord.PtrComp)
    if NextRecord.Discr == Ident1:
        NextRecord.IntComp = 6
        NextRecord.EnumComp = Proc6(PtrParIn.EnumComp)
        NextRecord.PtrComp = PtrGlb.PtrComp
        NextRecord.IntComp = Proc7(NextRecord.IntComp, 10)
```

```
    else:
        PtrParIn = NextRecord.copy()
    NextRecord.PtrComp = None
    return PtrParIn

def Proc2(IntParIO):
    IntLoc = IntParIO + 10
    while 1:
        if Char1Glob == 'A':
            IntLoc = IntLoc - 1
            IntParIO = IntLoc - IntGlob
            EnumLoc = Ident1
        if EnumLoc == Ident1:
            break
    return IntParIO

def Proc3(PtrParOut):
    global IntGlob

    if PtrGlb is not None:
        PtrParOut = PtrGlb.PtrComp
    else:
        IntGlob = 100
    PtrGlb.IntComp = Proc7(10, IntGlob)
    return PtrParOut

def Proc4():
    global Char2Glob

    BoolLoc = Char1Glob == 'A'
    BoolLoc = BoolLoc or BoolGlob
    Char2Glob = 'B'

def Proc5():
    global Char1Glob
    global BoolGlob

    Char1Glob = 'A'
    BoolGlob = FALSE

def Proc6(EnumParIn):
    EnumParOut = EnumParIn
    if not Func3(EnumParIn):
        EnumParOut = Ident4
    if EnumParIn == Ident1:
        EnumParOut = Ident1
    elif EnumParIn == Ident2:
        if IntGlob > 100:
            EnumParOut = Ident1
        else:
```

```
            EnumParOut = Ident4
    elif EnumParIn == Ident3:
        EnumParOut = Ident2
    elif EnumParIn == Ident4:
        pass
    elif EnumParIn == Ident5:
        EnumParOut = Ident3
    return EnumParOut

def Proc7(IntParI1, IntParI2):
    IntLoc = IntParI1 + 2
    IntParOut = IntParI2 + IntLoc
    return IntParOut

def Proc8(Array1Par, Array2Par, IntParI1, IntParI2):
    global IntGlob

    IntLoc = IntParI1 + 5
    Array1Par[IntLoc] = IntParI2
    Array1Par[IntLoc+1] = Array1Par[IntLoc]
    Array1Par[IntLoc+30] = IntLoc
    for IntIndex in range(IntLoc, IntLoc+2):
        Array2Par[IntLoc][IntIndex] = IntLoc
    Array2Par[IntLoc][IntLoc-1] = Array2Par[IntLoc][IntLoc-1] + 1
    Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc]
    IntGlob = 5

def Func1(CharPar1, CharPar2):
    CharLoc1 = CharPar1
    CharLoc2 = CharLoc1
    if CharLoc2 != CharPar2:
        return Ident1
    else:
        return Ident2

def Func2(StrParI1, StrParI2):
    IntLoc = 1
    while IntLoc <= 1:
        if Func1(StrParI1[IntLoc], StrParI2[IntLoc+1]) == Ident1:
            CharLoc = 'A'
            IntLoc = IntLoc + 1
    if CharLoc >= 'W' and CharLoc <= 'Z':
        IntLoc = 7
    if CharLoc == 'X':
        return TRUE
    else:
        if StrParI1 > StrParI2:
            IntLoc = IntLoc + 7
            return TRUE
        else:
```

```
            return FALSE

def Func3(EnumParIn):
    EnumLoc = EnumParIn
    if EnumLoc == Ident3: return TRUE
    return FALSE

if __name__ == '__main__':
    main()
```

B MATLAB Benchmark Code

```
% Modified for the purposes of the Amazon EC2 Benchmarking Team @ WPI
% Octave Benchmark 2 (8 March 2003)
% version 2, scaled to get 1 +/- 0.1 sec with R 1.6.2
% using the standard ATLAS library (Rblas.dll)
% on a Pentium IV 1.6 Ghz with 1 Gb Ram on Win XP pro
% Author : Philippe Grosjean
% eMail  : phgrosjean@sciviews.org
% Web    : http://www.sciviews.org
% License: GPL 2 or above at your convenience (see: http://www.gnu.org)
%
% Several tests are adapted from:
%
   ************************************************************************

%* Matlab Benchmark program version 2.0                                  *
%* Author : Stefan Steinhaus                                             *
%* EMAIL  : stst@informatik.uni-frankfurt.de                             *
%* This program is public domain. Feel free to copy it freely.           *
%
   ************************************************************************

% Escoufier's equivalents vectors (III.5) is adapted from Planque &
   Fromentin, 1996
% Ref: Escoufier Y., 1970. Echantillonnage dans une population de
   variables
% aleatoires relles. Publ. Inst. Statis. Univ. Paris 19 Fasc 4, 1-47.
%
% From the Matlab Benchmark... only cosmetic changes
%
% Type "cd('/<dir>')" and then "source('Octave2.m')" to start the test

clc

fact = 4; %numb
runs = 3;                          % Number of times the tests are
   executed
times = zeros(5, 3);

disp('   Octave Benchmark 2')
disp('   Modified for Amazon EC2 Benchmarking, WPI MQP 2010')
disp('   ==========')
disp(['Number of times each test is run------------------------: '
   num2str(runs)])
```

```
disp(' ')


disp('   I. Matrix calculation')
disp('   ----------------------')


% (1)
cumulate = 0; a = 0; b = 0;
for i = 1:runs
  tic;
    a = abs(randn(1500, 1500)/10);
    b = a';
    a = reshape(b, 750, 3000);
    b = a';
  timing = toc;
  cumulate = cumulate + timing;
end;
timing = cumulate/runs;
times(1, 1) = timing;
disp(['Creation, transp., deformation of a 1500x1500 matrix (sec): '
   num2str(timing)])
clear a; clear b;

% (2)
cumulate = 0; b = 0;
for i = 1:runs
  a = abs(randn(800*fact, 800*fact)/2);
  tic;
    b = a.^1000;
  timing = toc;
  cumulate = cumulate + timing;
end
timing = cumulate/runs;
times(2, 1) = timing;
disp(['800x800 normal distributed random matrix ^1000 ------- (sec): '
   num2str(timing)])
clear a; clear b;

% (3)
cumulate = 0; b = 0;
for i = 1:runs
  a = randn(2000000*fact, 1);
  tic;
    b = sort(a);
  timing = toc;
  cumulate = cumulate + timing;
end
timing = cumulate/runs;
times(3, 1) = timing;
```

```
disp(['Sorting of 2,000,0000 random values ------------------ (sec): '
    num2str(timing)])
clear a; clear b;

% (4)
cumulate = 0; b = 0;
for i = 1:runs
  a = randn(700*fact, 700*fact);
  tic;
    b = a'*a;
  timing = toc;
  cumulate = cumulate + timing;
end
timing = cumulate/runs;
times(4, 1) = timing;
disp(['700x700 cross-product matrix (b = a''*a) ----------- (sec): '
    num2str(timing)])
clear a; clear b;

% (5)
cumulate = 0; c = 0;
for i = 1:runs
  a = randn(600*fact, 600*fact);
  b = 1:600*fact;
  tic;
    c = a\b';
  timing = toc;
  cumulate = cumulate + timing;
end
timing = cumulate/runs;
times(5, 1) = timing;
disp(['Linear regression over a 600x600 matrix (c = a \\ b'') (sec): '
    num2str(timing)])
clear a; clear b; clear c;

times = sort(times);
disp('                    
    ------------------------------------------------------------')
disp(['                Trimmed geom. mean (2 extremes eliminated): '
    num2str(exp(mean(log(times(2:4,1)))))])
disp(' ')

disp('   II. Matrix functions')
disp('   ---------------------')


% (1)
cumulate = 0; b = 0;
for i = 1:runs
  a = randn(800000*fact, 1);
```

```
  tic;
    b = fft(a);
  timing = toc;
  cumulate = cumulate + timing;
end
timing = cumulate/runs;
times(1, 2) = timing;
disp(['FFT over 800,000 random values ---------- (sec): '
   num2str(timing)])
clear a; clear b;

% (2)
cumulate = 0; b = 0;
for i = 1:runs
  a = randn(320*fact, 320*fact);
  tic;
    b = eig(a);
  timing = toc;
  cumulate = cumulate + timing;
end
timing = cumulate/runs;
times(2, 2) = timing;
disp(['Eigenvalues of a 320x320 random matrix ---------- (sec): '
   num2str(timing)])
clear a; clear b;

% (3)
cumulate = 0; b = 0;
for i = 1:runs
  a = randn(650*fact, 650*fact);
  tic;
    b = det(a);
  timing = toc;
  cumulate = cumulate + timing;
end
timing = cumulate/runs;
times(3, 2) = timing;
disp(['Determinant of a 650x650 random matrix ---------- (sec): '
   num2str(timing)])
clear a; clear b;

% (4)
cumulate = 0; b = 0;
for i = 1:runs
  a = randn(900*fact, 900*fact);
  a = a'*a;
  tic;
    b = chol(a);
  timing = toc;
  cumulate = cumulate + timing;
```

```
end
timing = cumulate/runs;
times(4, 2) = timing;
disp(['Cholesky decomposition of a 900x900 matrix ---------- (sec): '
    num2str(timing)])
clear a; clear b;

% (5)
cumulate = 0; b = 0;
for i = 1:runs
  a = randn(400*fact, 400*fact);
  tic;
    b = inv(a);
  timing = toc;
  cumulate = cumulate + timing;
end
timing = cumulate/runs;
times(5, 2) = timing;
disp(['Inverse of a 400x400 random matrix ------------------ (sec): '
    num2str(timing)])
clear a; clear b;

times = sort(times);
disp('                    
    ---------------------------------------------------------')
disp(['                Trimmed geom. mean (2 extremes eliminated): '
    num2str(exp(mean(log(times(2:4,2)))))])
disp(' ')

disp('   III. Programmation')
disp('   ---------------------')

% (1)
cumulate = 0; a = 0; b = 0; phi = 1.6180339887498949;
for i = 1:runs
  a = floor(1000 * fact * rand(750000*fact, 1));
  tic;
    b = (phi.^a - (-phi).^(-a)) / sqrt(5);
  timing = toc;
  cumulate = cumulate + timing;
end
timing = cumulate/runs;
times(1, 3) = timing;
disp(['750,000 Fibonacci numbers calculation (vector calc)  (sec): '
    num2str(timing)])
clear a; clear b; clear phi;

% (1)
cumulate = 0; a = 5500; b = 0;
for i = 1:runs
```

```
  tic;
    b = ones(a, a)./((1:a)' * ones(1, a) + ones(a, 1) * (0:(a-1)));
  timing = toc;
  cumulate = cumulate + timing;
end
timing = cumulate/runs;
times(2, 3) = timing;
disp(['Creation of a 5550x5550 Hilbert matrix (matrix calc) (sec): '
   num2str(timing)])
clear a; clear b;

% (3)
cumulate = 0; c = 0;
for i = 1:3
  a = ceil(1000 * rand(70000, 1));
  b = ceil(1000 * rand(70000, 1));
  tic;
    c = gcd(a, b);                                % gcd2 is a recursive
        function
  timing = toc;
  cumulate = cumulate + timing;
end
timing = cumulate/runs;
times(3, 3) = timing;
disp(['Grand common divisors of 70,000 pairs (recursion)    (sec): '
   num2str(timing)])
clear a; clear b; clear c;
%
% (4)
cumulate = 0; b = 0;
for i = 1:runs
  b = zeros(220*fact, 220*fact);
  tic;
    for j = 1:220*fact
      for k = 1:220*fact
        b(k,j) = abs(j - k) + 1;
      end
    end
  timing = toc;
  cumulate = cumulate + timing;
end
timing = cumulate/runs;
times(4, 3) = timing;
disp(['Creation of a 220x220 Toeplitz matrix (loops)       (sec): '
   num2str(timing)])
clear b; clear j; clear k;

% (5)
cumulate = 0; p = 0; vt = 0; vr = 0; vrt = 0; rvt = 0; RV = 0; j = 0; k
    = 0;
```

```
x2 = 0; R = 0; Rxx = 0; Ryy = 0; Rxy = 0; Ryx = 0; Rvmax = 0; f = 0;
for i = 1:runs
  x = abs(randn(37, 37));
  tic;
    % Calculation of Escoufier's equivalent vectors
    p = size(x, 2);
    vt = [1:p];                                   % Variables to test
    vr = [];                                      % Result: ordered
        variables
    RV = [1:p];                                   % Result: correlations
    for j = 1:p                                   % loop on the variable
        number
      Rvmax = 0;
      for k = 1:(p-j+1)                           % loop on the variables
        if j == 1
          x2 = [x, x(:, vt(k))];
        else
          x2 = [x, x(:, vr), x(:, vt(k))];        % New table to test
        end
        R = corrcoef(x2);                         % Correlations table
        Ryy = R(1:p, 1:p);
        Rxx = R(p+1:p+j, p+1:p+j);
        Rxy = R(p+1:p+j, 1:p);
        Ryx = Rxy';
        rvt = trace(Ryx*Rxy)/((trace(Ryy^2)*trace(Rxx^2))^0.5); % RV
            calculation
        if rvt > Rvmax
          Rvmax = rvt;                            % test of RV
          vrt(j) = vt(k);                         % temporary held
              variable
        end
      end
      vr(j) = vrt(j);                             % Result: variable
      RV(j) = Rvmax;                              % Result: correlation
      f = find(vt~=vr(j));                        % identify the held
          variable
      vt = vt(f);                                 % reidentify variables
          to test
    end
  timing = toc;
  cumulate = cumulate + timing;
end
times(5, 3) = timing;
disp(['Escoufier''s method on a 37x37 matrix (mixed) --------- (sec): '
    num2str(timing)])
clear x; clear p; clear vt; clear vr; clear vrt; clear rvt; clear RV;
    clear j; clear k;
clear x2; clear R; clear Rxx; clear Ryy; clear Rxy; clear Ryx; clear
    Rvmax; clear f;
```

```
times = sort(times);
disp('                    ------------------------------------------------------------')
disp(['                Trimmed geom. mean (2 extremes eliminated): '
    num2str(exp(mean(log(times(2:4,3))))) ])
disp(' ')

disp(' ')
disp(['Total time for all 14 tests ------------------------ (sec): '
    num2str(sum(sum(times))) ])
disp(['Overall mean (sum of I, II and III trimmed means/3)  (sec): '
    num2str(exp(mean(mean(log(times(2:4,:))))))])
clear cumulate; clear timing; clear times; clear runs; clear i;
disp('                    ---  End of test  ---')

exit
```

C Setting up EC2 Instances

C.1 Creating Instances

The first step that had to be taken to get onto the Amazon Elastic Cloud Compute (EC2) system was to set up an Amazon Web Services (AWS) account, via the AWS registration page [2]. After creating an account, it was possible to log into the EC2 system by navigating to the AWS page, clicking on the *Account tab*, selecting *Amazon EC2*, and logging into the management console. Once logged in, new EC2 systems were created by clicking the *Launch New Instance* button in the center of the page.

This launched the *Request Instances Wizard* shown below:

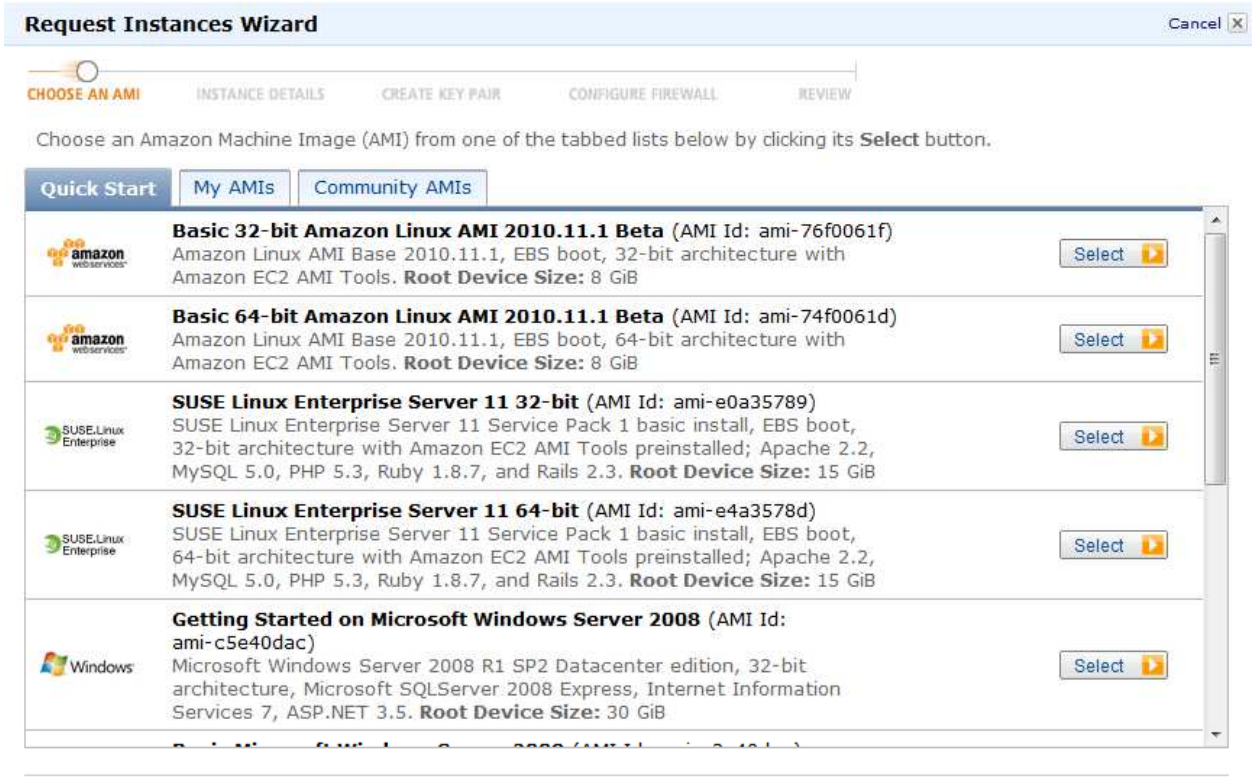

Figure 14: Request Instances Wizard (RIW)

It is important to note that Figure 14 is a screen shot as of February 2011, which was different from the original options that were displayed when creating the instances. At that point in time, Fedora 8 was the best available alternative to Red Hat Linux 5.5, which was why the 64-bit variant of Fedora 8 was chosen to use as the base-instance. In the next page of the wizard, the desired type of instance to be created was chosen (Large), leaving the availability zone set to the default (No Preference), and the number of instances to 1.

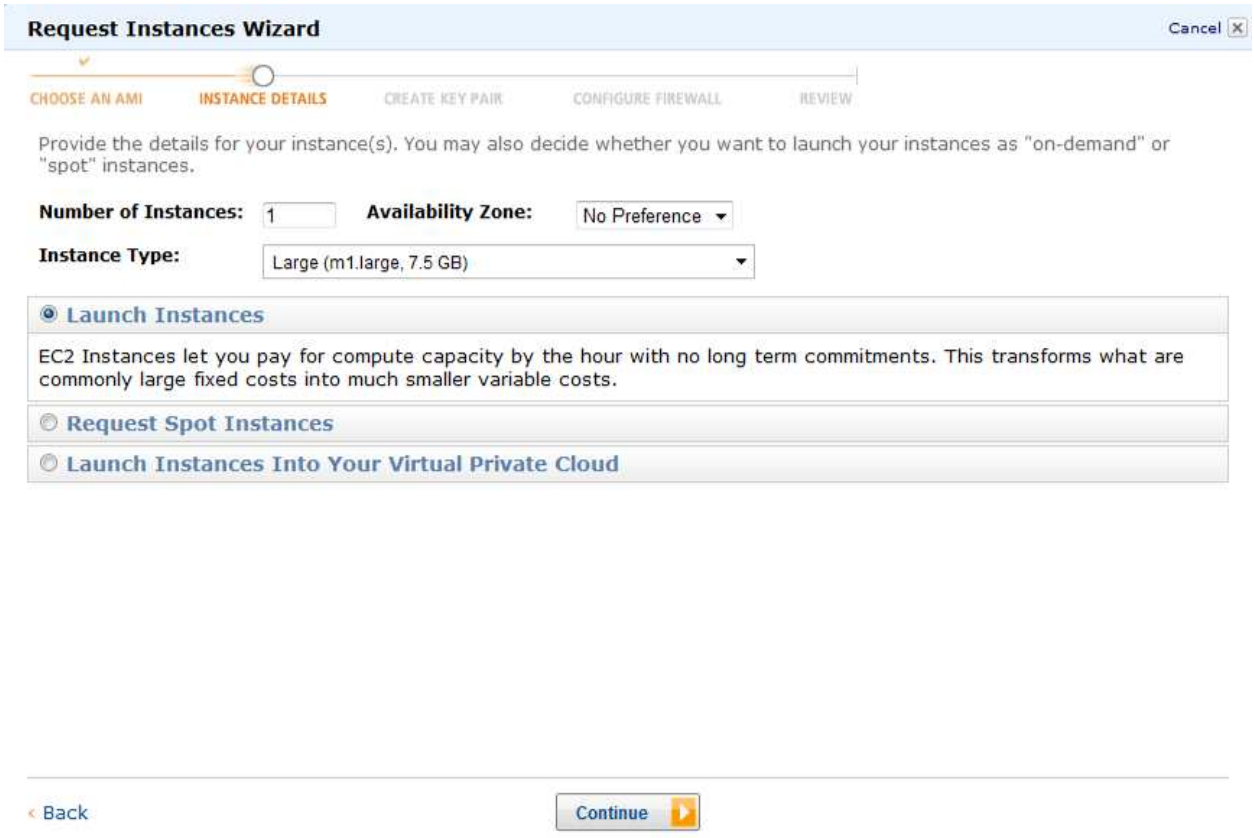

Figure 15: RIW: Select Instance Size & Zone

In the following page, all of the options were set to their default states. The Amazon EC2 instances all use specific Linux kernels that match up to RAM disks, and so these settings should only be adjusted if a custom instance has to be created, and prior research has been done. After the kernel selection page, an option to name the instance (Large) is provided, which is a useful feature for distinguishing between multiple instances of the same type. In addition to naming the instance, it is possible to tie together multiple key and value pairs (that is, key = name, value = large), which could be useful for managing large numbers of instances.

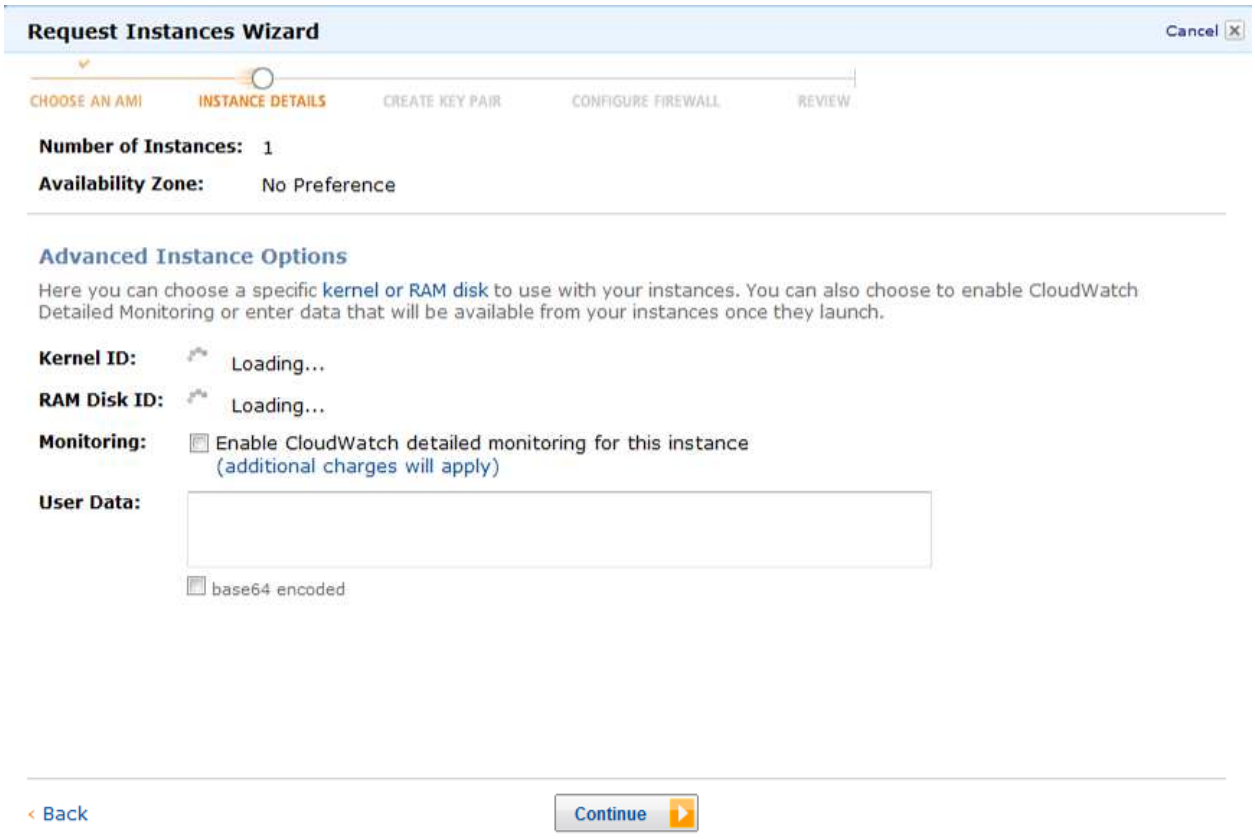

Figure 16: RIW: Select Kernel ID & RAM Disk ID

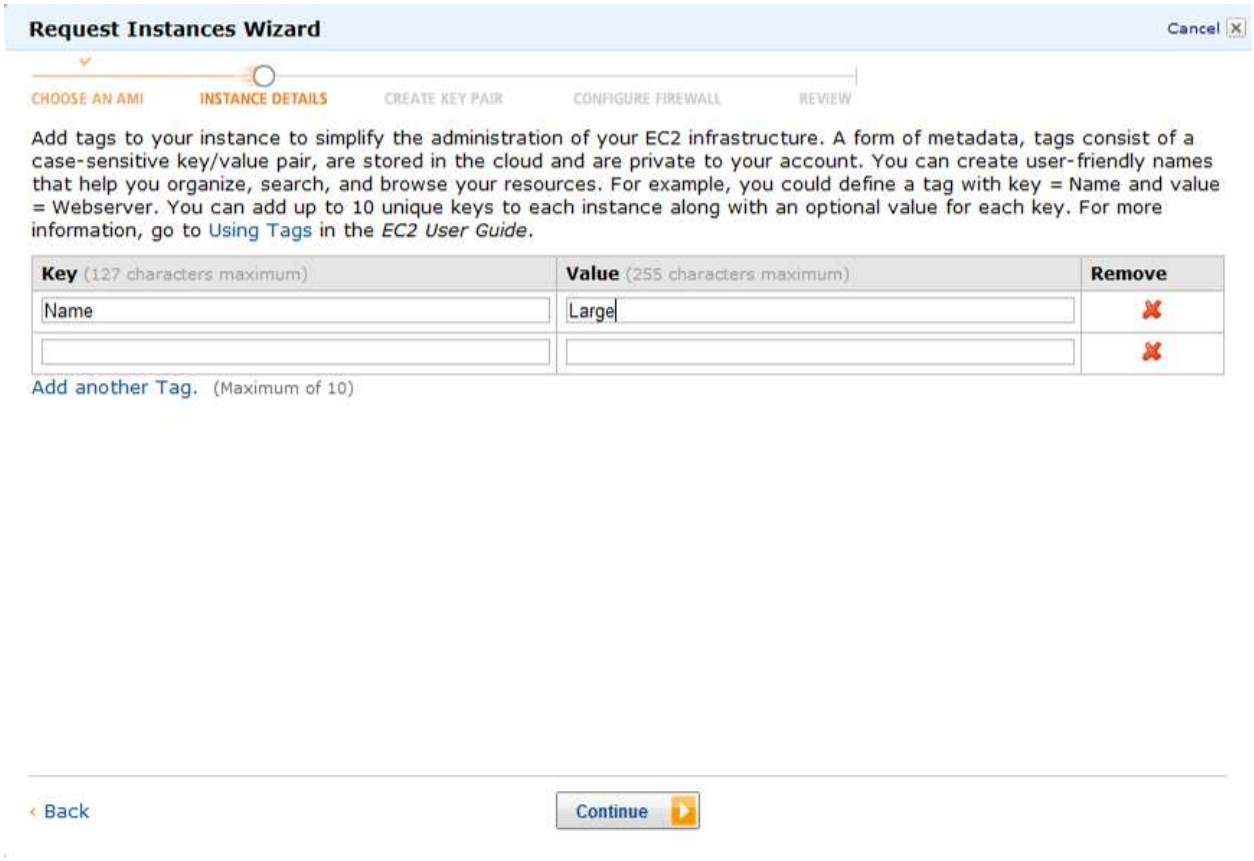

Figure 17: RIW: Key-Value Pairs

For the first instance that was created, a key pair file had to be created. The key pair file works much the same way as a key and a lock; only those who have access to the key pair file are able to connect to the EC2 server. Once a key pair is created, it can be linked with any number of EC2 instances. At first, a single copy of this key pair file was created, and then manipulated as necessary for connections to the EC2 server—which will be described in detail in the next section.

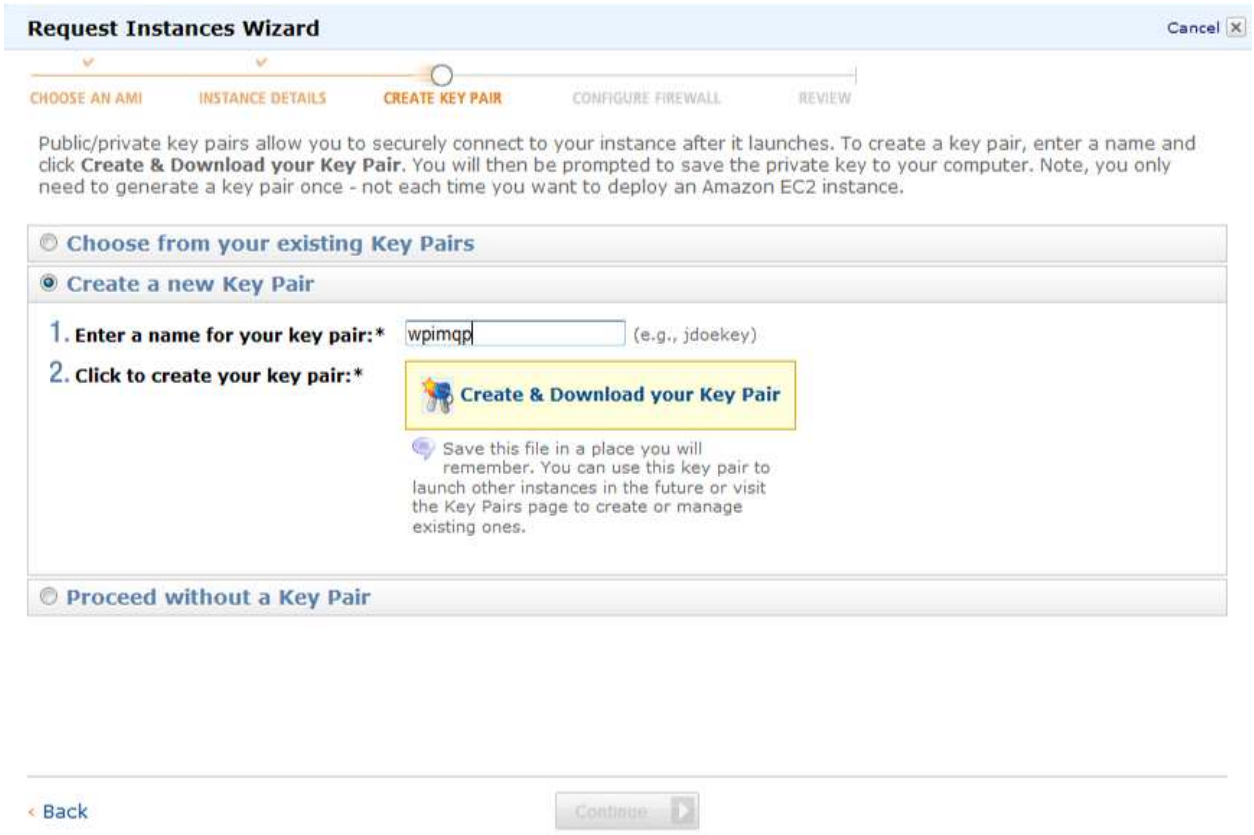

Figure 18: RIW: Create Key Pair

The next image is the firewall configuration for the instance, for which a "blank" security group was created. Although the default security group would have sufficed, it was important to ensure that no authorized connections would be lost to the server. After configuring the firewall, a final review of the instance and the choice to launch the instance was presented.

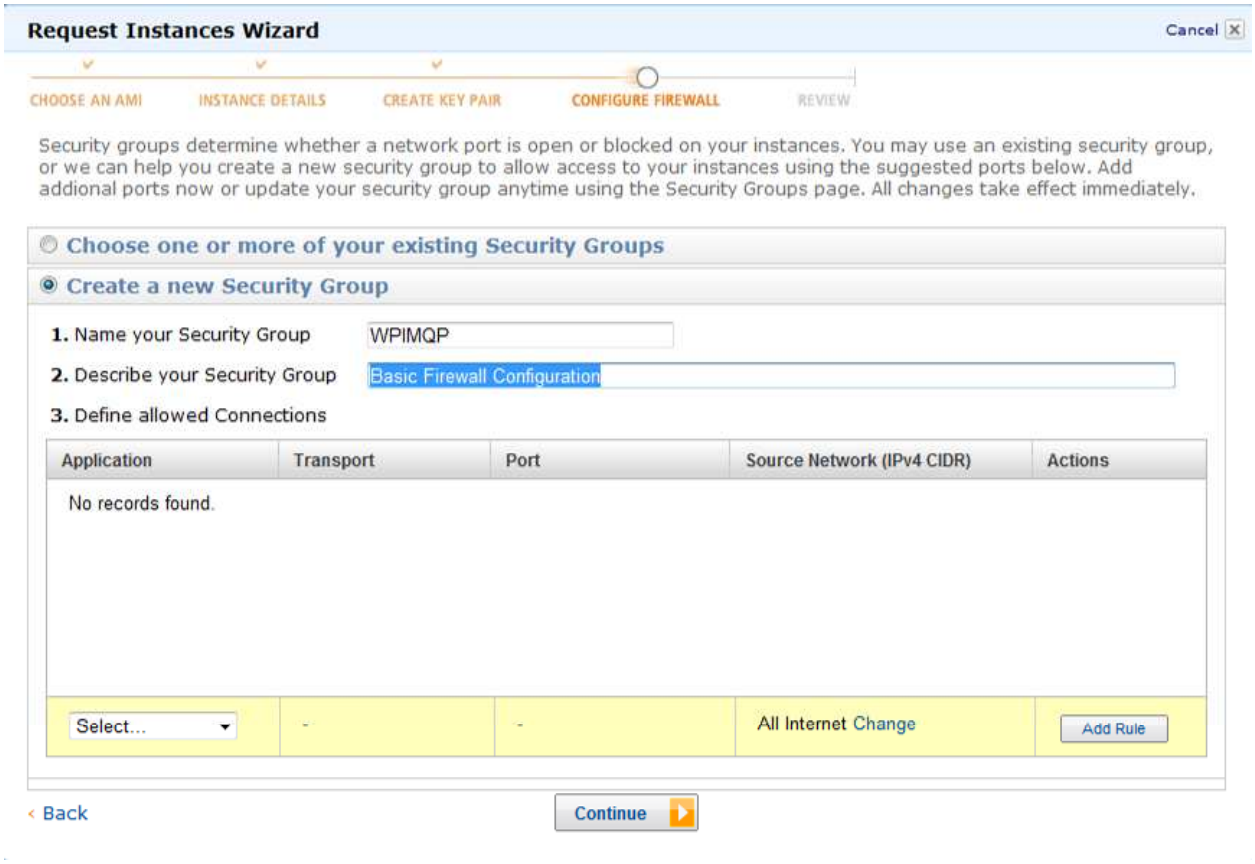

Figure 19: RIW: Create Security Group

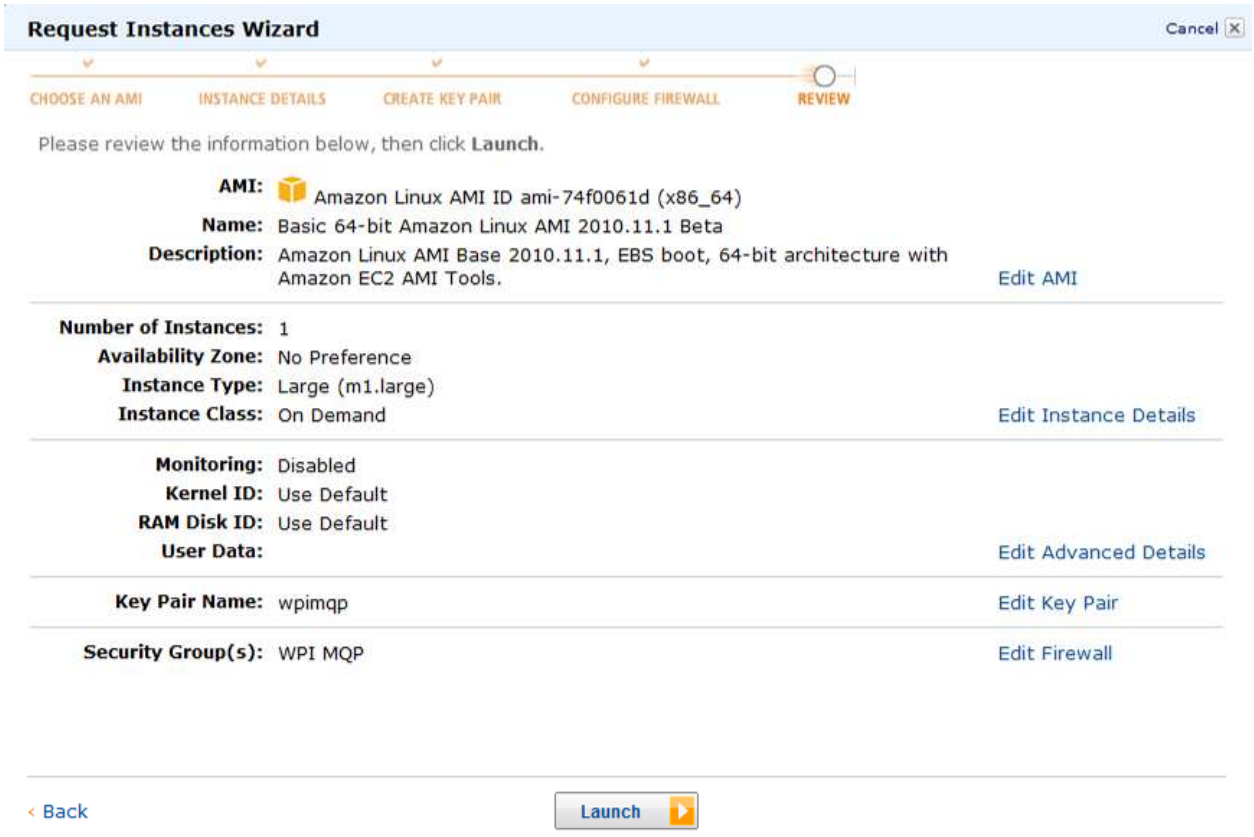

Figure 20: RIW: Review Instance

Once the instance was launched, it became available via the *instances* tab in the AWS management console. This is where all of the instances that were created could be accessed throughout the course of the project.

C.2 Connecting to Instances

There were two simple ways to connect to the instances, depending on the type of computer that is being used. Regardless of the chosen method, the first step was to acquire a copy of the key pair that was created in the initial instance creation process. Each member of the project group was provided a copy of the key, as well as the original copy on the secure remote server that was used for documentation. Prior to connecting, the instances were started by logging into the AWS management console, choosing *Instances*, right-clicking on the name of the instance, and selecting the *Start* option. Once the status of the instance is *running*, it is ready to connect.

C.2.1 The Linux/UNIX/Mac OSX Method

This was the simpler method of the two, as it required little to no additional resources. To connect to the remote EC2 servers, a terminal application was started, and the directory changed to that containing the key file—wpimqp.pem. Once in the directory and logged into the AWS management console, the instance was started by right-clicking on its name and then clicking *Start*. This brought up a window that displayed an SSH command to connect to the given instance. The connection is established with the EC2 server by copying the SSH command and pasting it into the terminal.

Figure 21: Connect Help—SSH

C.2.2 The Windows Method

Of the two approaches, the initial setup of the Windows system took longer. Using PuTTY on a Windows machine, it was necessary to create a different kind of secure key to log into the server—in this case, a ".ppk" file. This file will be used in lieu of the ".pem" file that was used for the Linux systems, and is compatible with both Putty and WinSCP—a common file transfer application.

The first thing to be done was to acquire a program that could convert a ".pem" file to ".ppk," which was accomplished my PuTTYgen. PuTTY and PuTTYgen were downloaded from their distribution sites, following which the PuTTYgen executable was launched. This brought up the main screen, from which we clicked *Conversions* and then *Import key*. After the browsing window appeared, the secure key chosen by the group was selected, in this case "example.pem," followed by clicking *Open*.

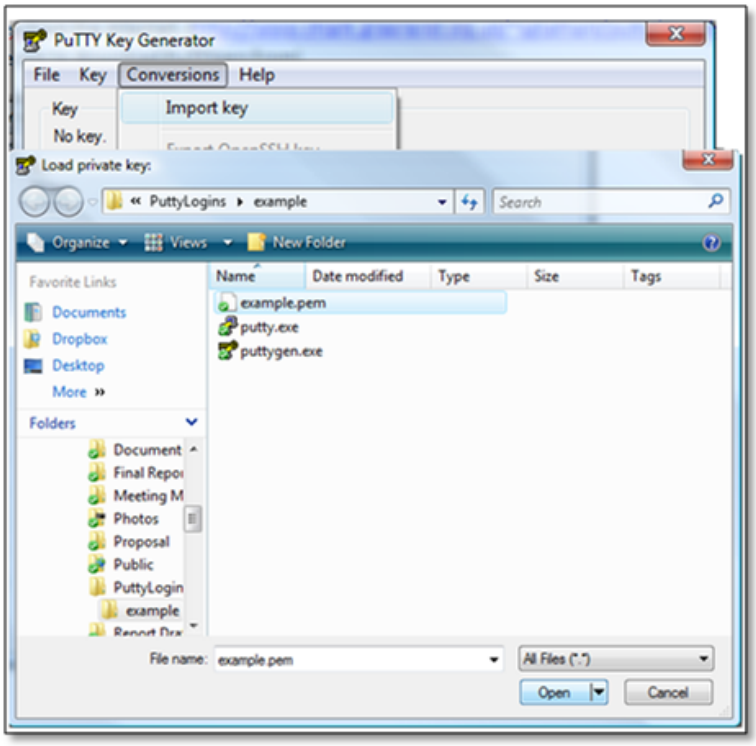

Figure 22: PuTTYgen: Import Key

This brought back the main screen, where *Save Private Key* was originally clicked. The

appropriate name and directory was entered, set to be saved as a *PuTTY Private Key File*, and saved. This generated a secure key, which was used to log into PuTTY.

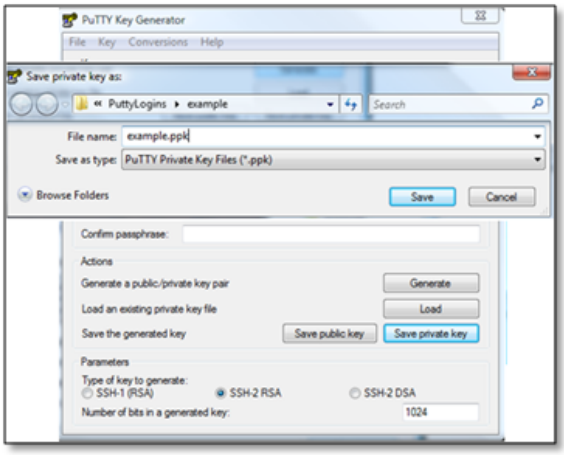

Figure 23: PuTTYgen: Save Private Key

After setting up the private key file, the PuTTY executable was launched. In the *Session* page, the public address for the EC2 instance into the *Host Name* field was entered. Next, PuTTY was navigated through in order to get to the *Connection* $\Rightarrow$ *SSH* $\Rightarrow$ *Auth* section, and the *Browse* button was clicked under *Private key file for authentication*. The ".ppk" file that had been saved earlier was selected and opened, in this case "example.ppk". At this point, the *Open* button was clicked, which started the session. The username *root* was used, as this is the only account that the EC2 instances come with default.

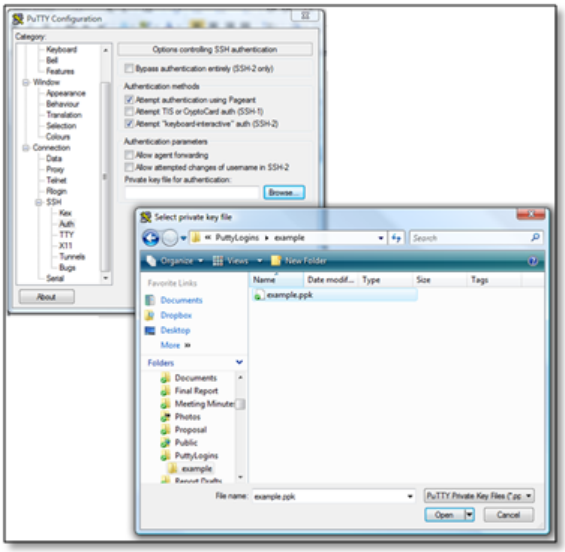

Figure 24: PuTTY: Select Private Key

D Performance Comparison Ratio Results

D.1 Pystone

Computer 1							
	Red Hat			Fedora			
Result	**Average**	**STDEV**	**% STDEV**	**Average**	**STDEV**	**% STDEV**	**PCR**
Execution Time (Seconds)	106.778	1.766	1.654	117.429	1.553	1.322	0.909
Pystones per Second	94845.205	1540.567	1.624	86034.022	1139.796	1.325	0.907
						Average:	**0.908**

Computer 2							
	Red Hat			Fedora			
Result	**Average**	**STDEV**	**% STDEV**	**Average**	**STDEV**	**% STDEV**	**PCR**
Execution Time (Seconds)	107.470	1.797	1.672	118.131	1.873	1.585	0.910
Pystones per Second	94263.764	1560.671	1.656	85538.758	1345.585	1.573	0.907
						Average:	**0.909**

D.2 MATLAB Benchmark

Computer 1							
	Red Hat			Fedora			
Test	**Average Execution Time (Seconds)**	**STDEV**	**% STDEV**	**Average Execution Time (Seconds)**	**STDEV**	**% STDEV**	**PCR**
1.000	1.999	0.030	1.481	1.513	0.034	2.268	1.321
2.000	0.850	0.001	0.111	0.829	0.000	0.020	1.026
3.000	0.920	0.001	0.123	0.895	0.002	0.258	1.028
4.000	1.162	0.007	0.614	1.136	0.004	0.390	1.022
5.000	0.828	0.009	1.108	0.809	0.007	0.815	1.024
6.000	0.969	0.004	0.404	0.945	0.003	0.268	1.026
7.000	0.361	0.010	2.800	0.315	0.009	2.879	1.147

8.000	3.377	0.023	0.690	3.394	0.043	1.252	0.995
9.000	0.879	0.002	0.226	0.855	0.002	0.211	1.028
10.000	0.992	0.001	0.067	0.969	0.003	0.352	1.025
11.000	0.582	0.003	0.512	0.580	0.003	0.445	1.005
12.000	0.798	0.001	0.183	0.783	0.002	0.295	1.019
13.000	0.470	0.013	2.797	0.471	0.007	1.537	0.998
14.000	0.195	0.014	7.138	0.129	0.005	3.788	1.510
15.000	-	-	-	-	-	-	-
16.000	0.014	0.001	4.219	0.014	0.000	2.973	1.004
17.000	0.245	0.001	0.338	0.236	0.000	0.125	1.041
18.000	0.087	0.003	2.974	0.075	0.001	1.744	1.164
19.000	**12.876**	**0.050**	**0.391**	**12.144**	**0.063**	**0.522**	**1.060**
20.000	0.406	0.004	1.030	0.381	0.002	0.644	1.067
Averages	**1.474**			**1.393**			**1.058**

Computer 2							
	Red Hat			**Fedora**			
Test	**Average Execution Time (Seconds)**	**STDEV**	**% STDEV**	**Average Execution Time (Seconds)**	**STDEV**	**% STDEV**	**PCR**
1.000	2.012	0.010	0.503	1.535	1.535	0.006	1.311
2.000	0.850	0.003	0.317	0.831	0.831	0.000	1.024
3.000	0.923	0.001	0.056	0.898	0.898	0.000	1.027
4.000	1.170	0.006	0.474	1.145	1.145	0.005	1.022
5.000	0.838	0.007	0.852	0.820	0.820	0.006	1.023
6.000	0.972	0.003	0.330	0.949	0.949	0.003	1.024
7.000	0.369	0.009	2.418	0.322	0.322	0.009	1.143
8.000	3.398	0.037	1.095	3.413	3.413	0.060	0.996
9.000	0.892	0.001	0.091	0.869	0.869	0.001	1.026
10.000	1.003	0.002	0.182	0.978	0.978	0.002	1.025
11.000	0.589	0.003	0.489	0.587	0.587	0.003	1.003
12.000	0.808	0.001	0.177	0.793	0.793	0.002	1.018
13.000	0.469	0.009	1.861	0.478	0.478	0.008	0.982
14.000	0.198	0.015	7.347	0.133	0.133	0.004	1.483
15.000	-	-	-	-	-	-	-
16.000	0.014	0.000	1.164	0.014	0.014	0.000	1.000
17.000	0.242	0.001	0.300	0.234	0.234	0.000	1.034
18.000	0.087	0.002	2.566	0.075	0.075	0.001	1.153
19.000	**12.967**	0.041	**0.313**	**12.257**	**12.257**	**0.064**	**1.058**

20.000	0.408	0.004	0.902	0.384	0.384	0.001	1.063
Averages	**1.485**			**1.406**			**1.056**

D.3 RAMSMP

D.3.1 Float Memory

Computer 1							
	Red Hat			**Fedora**			
Test	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**PCR**
Copy:	3989.932	9.207	0.231	4085.546	4.287	0.105	1.024
Scale:	3995.517	8.778	0.220	4086.738	4.819	0.118	1.023
Add:	5053.435	6.472	0.128	5169.554	6.283	0.122	1.023
Triad:	5052.189	4.296	0.085	5166.678	5.027	0.097	1.023
AVERAGE:	4522.771	5.527	0.122	4627.131	2.410	0.052	1.023
						Average:	**1.023**

Computer 2							
	Red Hat			**Fedora**			
Test	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**PCR**
Copy:	3817.966	2.921225	0.077	3844.573	5.03523	0.131	1.007
Scale:	3819.203	2.812004	0.074	3847.385	6.341245	0.165	1.007
Add:	4823.7	5.94919	0.123	4864.668	6.588147	0.135	1.008
Triad:	4820.951	6.079134	0.126	4861.375	7.099936	0.146	1.008
AVERAGE:	4320.456	3.121298	0.072	4354.504	3.256174	0.075	1.008
						Average:	**1.008**

D.3.2 Float Reading

Computer 1			
	Red Hat	**Fedora**	

Block Size	Average Bandwidth (MB/s)	STDEV	%STDEV	Average Bandwidth (MB/s)	STDEV	%STDEV	PCR
1 Kb	47655.241	786.322	1.650	47927.080	44.886	0.094	1.006
2 Kb	47941.366	36.637	0.076	47927.380	5.861	0.012	1.000
4 Kb	47911.876	74.597	0.156	47928.240	8.991	0.019	1.000
8 Kb	47930.435	43.965	0.092	47930.240	6.137	0.013	1.000
16 Kb	47932.405	42.372	0.088	47929.520	10.092	0.021	1.000
32 Kb	47044.608	52.280	0.111	47034.700	5.665	0.012	1.000
64 Kb	32424.056	20.920	0.065	32408.400	6.636	0.020	1.000
128 Kb	32371.452	128.768	0.398	32380.760	11.809	0.036	1.000
256 Kb	32408.608	22.734	0.070	32395.460	4.165	0.013	1.000
512 Kb	32400.680	25.222	0.078	32391.220	6.673	0.021	1.000
1024 Kb	32387.444	22.232	0.069	32375.210	12.635	0.039	1.000
2048 Kb	30375.091	1921.857	6.327	29903.400	1898.969	6.350	0.984
4096 Kb	7568.574	280.773	3.710	7567.538	342.773	4.530	1.000
8192 Kb	6556.492	38.776	0.591	6491.749	31.904	0.491	0.990
16384 Kb	6558.831	57.961	0.884	6495.715	28.944	0.446	0.990
32768 Kb	6567.572	56.795	0.865	6505.025	34.596	0.532	0.990
65536 Kb	6568.941	59.707	0.909	6509.626	41.948	0.644	0.991
131072 Kb	6600.540	55.786	0.845	6534.415	37.289	0.571	0.990
262144 Kb	6604.244	51.275	0.776	6539.864	27.607	0.422	0.990
524288 Kb	6632.648	34.428	0.519	6553.526	32.704	0.499	0.988
1048576 Kb	6641.208	37.327	0.562	6583.587	30.811	0.468	0.991
						Average:	**0.996**

Computer 2							
	Red Hat			**Fedora**			
Block Size	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**PCR**
1 Kb	47604.110	398.844	0.838	47511.860	1371.238	0.094	0.998
2 Kb	47826.270	17.538	0.037	47766.390	9.640	0.012	0.999
4 Kb	47822.720	53.907	0.113	47769.220	11.344	0.019	0.999
8 Kb	47830.820	29.144	0.061	47771.410	12.437	0.013	0.999
16 Kb	47827.860	16.930	0.035	47772.230	8.922	0.021	0.999
32 Kb	46959.690	14.130	0.030	46881.320	7.834	0.012	0.998
64 Kb	32351.040	12.633	0.039	32302.180	12.299	0.020	0.998
128 Kb	32325.410	25.172	0.078	32277.880	3.992	0.036	0.999
256 Kb	32339.790	9.590	0.030	32286.820	17.466	0.013	0.998
512 Kb	32337.610	12.074	0.037	32285.180	10.712	0.021	0.998

1024 Kb	32317.610	11.864	0.037	32272.020	1.815	0.039	0.999
2048 Kb	30391.710	1116.878	3.675	29711.320	2744.431	6.350	0.978
4096 Kb	7633.278	111.486	1.461	7689.616	305.783	4.530	1.007
8192 Kb	6560.193	23.884	0.364	6493.419	21.332	0.491	0.990
16384 Kb	6574.034	20.262	0.308	6496.869	21.692	0.446	0.988
32768 Kb	6602.597	22.305	0.338	6496.130	19.541	0.532	0.984
65536 Kb	6604.867	24.969	0.378	6490.263	19.489	0.644	0.983
131072 Kb	6611.157	21.830	0.330	6506.956	21.887	0.571	0.984
262144 Kb	6612.515	26.131	0.395	6478.888	19.896	0.422	0.980
524288 Kb	6623.563	32.077	0.484	6528.051	22.459	0.499	0.986
1048576 Kb	6629.294	32.980	0.497	6532.071	21.890	0.468	0.985
						Average:	**0.993**

D.3.3 Float Writing

Computer 1							
	Red Hat			**Fedora**			
Block Size	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**PCR**
1 Kb	46201.650	634.161	1.373	46356.190	1065.308	0.094	1.003
2 Kb	46964.440	40.540	0.086	46767.740	1240.196	0.012	0.996
4 Kb	47153.510	80.182	0.170	47110.180	411.589	0.019	0.999
8 Kb	47237.580	148.555	0.314	47094.860	1135.800	0.013	0.997
16 Kb	47327.410	37.813	0.080	47264.330	397.805	0.021	0.999
32 Kb	46262.360	48.409	0.105	46057.130	1029.894	0.012	0.996
64 Kb	21300.140	18.572	0.087	21268.800	7.662	0.020	0.999
128 Kb	21312.550	15.235	0.071	21270.050	1.625	0.036	0.998
256 Kb	21302.610	19.611	0.092	21231.640	221.608	0.013	0.997
512 Kb	21293.680	16.295	0.077	21227.220	165.011	0.021	0.997
1024 Kb	21289.850	13.662	0.064	21225.430	180.948	0.039	0.997
2048 Kb	20365.430	643.393	3.159	19648.870	1322.322	6.350	0.965
4096 Kb	3192.530	52.336	1.639	3186.670	101.396	4.530	0.998
8192 Kb	2627.613	5.432	0.207	2626.142	6.581	0.491	0.999
16384 Kb	2625.520	6.253	0.238	2626.732	6.437	0.446	1.000
32768 Kb	2623.591	10.059	0.383	2627.242	7.796	0.532	1.001
65536 Kb	2620.585	12.734	0.486	2626.952	11.000	0.644	1.002
131072 Kb	2614.151	11.703	0.448	2626.416	12.534	0.571	1.005
262144 Kb	2606.087	8.705	0.334	2630.822	9.743	0.422	1.009
524288 Kb	2586.935	6.604	0.255	2648.481	7.736	0.499	1.024
1048576 Kb	2545.045	6.527	0.256	2682.871	5.853	0.468	1.054
						Average:	**1.002**

Computer 2							
	Red Hat			Fedora			
Block Size	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**PCR**
1 Kb	45616.150	1471.976	3.227	45397.89	2247.8170	0.094	0.995
2 Kb	46722.090	789.580	1.690	46799.19	55.7225	0.012	1.002
4 Kb	47016.520	242.890	0.517	47002.36	45.1257	0.019	1.000
8 Kb	47137.280	150.374	0.319	46906.91	1306.0590	0.013	0.995
16 Kb	47218.080	20.206	0.043	47096.76	464.0466	0.021	0.997
32 Kb	46252.080	288.478	0.624	45902.64	1178.1800	0.012	0.992
64 Kb	22617.910	6827.961	30.188	21502.25	3528.4020	0.020	0.951
128 Kb	21262.910	12.775	0.060	21179.76	191.7609	0.036	0.996
256 Kb	21261.110	11.970	0.056	21158.39	291.5292	0.013	0.995
512 Kb	21251.310	8.684	0.041	21164.57	148.7571	0.021	0.996
1024 Kb	21237.290	16.347	0.077	21167.3	161.3955	0.039	0.997
2048 Kb	20198.260	938.995	4.649	20423.87	570.2100	6.350	1.011
4096 Kb	3499.644	4810.118	137.446	3076.583	2514.1060	4.530	0.879
8192 Kb	2514.165	137.113	5.454	2469.011	66.1581	0.491	0.982
16384 Kb	2477.096	1.047	0.042	2460.43	2.3430	0.446	0.993
32768 Kb	2477.183	1.076	0.043	2461.815	1.1056	0.532	0.994
65536 Kb	2475.996	0.940	0.038	2465.134	2.5085	0.644	0.996
131072 Kb	2471.572	1.706	0.069	2468.814	0.9446	0.571	0.999
262144 Kb	2462.975	2.834	0.115	2475.892	1.1759	0.422	1.005
524288 Kb	2445.357	5.196	0.212	2489.342	1.9695	0.499	1.018
1048576 Kb	2414.592	9.129	0.378	2518.077	4.1274	0.468	1.043
						Average:	**0.992**

D.3.4 Int Memory

Computer 1							
	Red Hat			Fedora			
Test	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**PCR**
Copy:	3975.610	14.699	0.370	4065.247	5.691	0.140	1.023
Scale:	3972.357	8.472	0.213	4067.983	5.155	0.127	1.024
Add:	3999.469	5.681	0.142	4057.769	3.418	0.084	1.015
Triad:	3996.476	5.774	0.144	4052.514	4.118	0.102	1.014

AVERAGE:	3985.977	6.868	0.172	4060.879	2.556	0.063	1.019
						Average:	**1.019**

Computer 2							
	Red Hat			**Fedora**			
Test	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**PCR**
Copy:	3794.1198	2.091277	0.055	3820.1817	1.954316	0.051	1.007
Scale:	3791.8569	1.589752	0.042	3819.7259	2.029807	0.053	1.007
Add:	3792.8253	1.890267	0.050	3799.96	2.098101	0.055	1.002
Triad:	3789.918	1.792579	0.047	3797.9327	2.015338	0.053	1.002
AVERAGE:	3792.1806	1.261712	0.033	3809.45	1.176464	0.031	1.014
						Average:	**1.006**

D.3.5 Int Reading

Computer 1							
	Red Hat			**Fedora**			
Block Size	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**PCR**
1 Kb	47389.700	342.646	0.723	47263.800	1387.313	0.094	0.997
2 Kb	47558.640	41.375	0.087	47538.720	110.829	0.012	1.000
4 Kb	47544.820	52.249	0.110	47323.830	1280.886	0.019	0.995
8 Kb	47556.260	43.827	0.092	47521.600	247.218	0.013	0.999
16 Kb	47540.740	136.584	0.287	47054.230	2773.283	0.021	0.990
32 Kb	46336.210	50.462	0.109	46242.540	565.306	0.012	0.998
64 Kb	21299.240	31.621	0.148	21546.330	3565.003	0.020	1.012
128 Kb	21307.930	33.102	0.155	21268.850	15.713	0.036	0.998
256 Kb	21300.180	31.581	0.148	21266.490	23.078	0.013	0.998
512 Kb	21288.730	30.957	0.145	21247.540	4.449	0.021	0.998
1024 Kb	21284.390	35.658	0.168	21256.420	26.920	0.039	0.999
2048 Kb	20374.380	900.627	4.420	20261.170	923.302	6.350	0.994
4096 Kb	3132.524	85.894	2.742	3201.734	2540.083	4.530	1.022
8192 Kb	2624.786	9.363	0.357	2635.428	70.637	0.491	1.004
16384 Kb	2626.047	7.808	0.297	2628.842	3.385	0.446	1.001
32768 Kb	2627.056	5.612	0.214	2632.915	4.118	0.532	1.002

65536 Kb	2625.920	5.677	0.216	2635.873	3.626	0.644	1.004
131072 Kb	2619.925	3.581	0.137	2640.740	4.304	0.571	1.008
262144 Kb	2608.513	3.619	0.139	2647.971	3.880	0.422	1.015
524288 Kb	2587.500	5.330	0.206	2660.407	3.741	0.499	1.028
						Average:	**1.003**

Computer 2							
	Red Hat			**Fedora**			
Block Size	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**PCR**
1 Kb	47647.320	361.894	0.760	47518.110	1153.128	0.094	0.997
2 Kb	47826.350	26.016	0.054	47691.050	506.776	0.012	0.997
4 Kb	47841.660	11.191	0.023	47667.860	683.449	0.019	0.996
8 Kb	47829.820	19.088	0.040	47750.870	146.501	0.013	0.998
16 Kb	47819.530	91.676	0.192	47769.000	21.267	0.021	0.999
32 Kb	46959.590	16.036	0.034	46745.990	820.251	0.012	0.995
64 Kb	28497.730	33.834	0.119	28396.060	334.295	0.020	0.996
128 Kb	28450.140	34.324	0.121	28306.800	564.115	0.036	0.995
256 Kb	28464.190	32.353	0.114	28417.170	34.527	0.013	0.998
512 Kb	28463.740	66.362	0.233	28426.820	23.326	0.021	0.999
1024 Kb	28457.050	6.305	0.022	28383.240	183.444	0.039	0.997
2048 Kb	27252.500	897.628	3.294	27320.790	765.291	6.350	1.003
4096 Kb	7398.064	111.631	1.509	7345.678	87.767	4.530	0.993
8192 Kb	6373.727	22.045	0.346	6336.512	23.378	0.491	0.994
16384 Kb	6405.545	23.270	0.363	6358.008	19.731	0.446	0.993
32768 Kb	6413.602	22.882	0.357	6390.188	21.073	0.532	0.996
65536 Kb	6433.593	22.236	0.346	6394.373	17.816	0.644	0.994
131072 Kb	6450.704	23.597	0.366	6402.321	16.996	0.571	0.992
262144 Kb	6447.826	18.662	0.289	6399.539	15.980	0.422	0.993
524288 Kb	6453.546	21.133	0.327	6401.160	18.658	0.499	0.992
						Average:	**0.996**

D.3.6 Int Writing

Computer 1							
	Red Hat			**Fedora**			
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Average Bandwidth**	**STDEV**	**%STDEV**	**PCR**

	(MB/s)			(MB/s)			
1 Kb	47396.500	337.242	0.712	47559.910	9.169	0.094	1.003
2 Kb	47557.330	43.202	0.091	47551.490	13.015	0.012	1.000
4 Kb	47545.390	51.631	0.109	47557.760	7.581	0.019	1.000
8 Kb	47556.380	42.932	0.090	47421.740	889.464	0.013	0.997
16 Kb	47540.140	134.452	0.283	47558.370	6.500	0.021	1.000
32 Kb	46337.340	49.928	0.108	46171.760	817.209	0.012	0.996
64 Kb	21299.760	31.108	0.146	21246.180	152.334	0.020	0.997
128 Kb	21308.280	32.470	0.152	21256.000	93.104	0.036	0.998
256 Kb	21300.650	31.021	0.146	21246.210	127.609	0.013	0.997
512 Kb	21289.230	30.419	0.143	21251.540	2.858	0.021	0.998
1024 Kb	21284.880	35.019	0.165	21235.170	121.419	0.039	0.998
2048 Kb	20314.600	942.334	4.639	19924.870	868.922	6.350	0.981
4096 Kb	3133.732	84.470	2.696	3224.534	64.037	4.530	1.029
8192 Kb	2624.591	9.221	0.351	2625.091	4.583	0.491	1.000
16384 Kb	2625.879	7.692	0.293	2629.364	6.080	0.446	1.001
32768 Kb	2626.880	5.569	0.212	2632.503	5.839	0.532	1.002
65536 Kb	2625.735	5.637	0.215	2633.960	5.043	0.644	1.003
131072 Kb	2619.722	3.652	0.139	2638.413	3.874	0.571	1.007
262144 Kb	2608.370	3.618	0.139	2644.516	4.910	0.422	1.014
524288 Kb	2587.317	5.302	0.205	2658.493	4.223	0.499	1.028
						Average:	**1.003**

Computer 2							
	Red Hat			**Fedora**			
Block Size	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**Average Bandwidth (MB/s)**	**STDEV**	**%STDEV**	**PCR**
1 Kb	47268.450	568.637	1.203	45497.370	3627.006	0.094	0.963
2 Kb	47457.100	17.033	0.036	47170.260	580.258	0.012	0.994
4 Kb	47458.550	19.034	0.040	47206.340	375.511	0.019	0.995
8 Kb	47459.270	16.244	0.034	46935.590	1495.163	0.013	0.989
16 Kb	47449.230	24.016	0.051	47092.110	587.033	0.021	0.992
32 Kb	46257.380	15.287	0.033	45629.560	1330.055	0.012	0.986
64 Kb	21254.410	17.718	0.083	21487.590	3535.762	0.020	1.011
128 Kb	21264.870	10.816	0.051	21145.130	197.889	0.036	0.994
256 Kb	21258.760	14.481	0.068	21154.520	115.589	0.013	0.995
512 Kb	21245.810	14.570	0.069	21142.360	86.037	0.021	0.995
1024 Kb	21237.970	16.751	0.079	21148.650	87.157	0.039	0.996
2048 Kb	20002.040	959.456	4.797	19607.150	1620.399	6.350	0.980
4096 Kb	2972.731	57.641	1.939	3151.949	2423.122	4.530	1.060
8192 Kb	2476.401	0.950	0.038	2475.692	82.950	0.491	1.000

16384 Kb	2476.496	0.931	0.038	2463.470	5.934	0.446	0.995
32768 Kb	2475.616	0.921	0.037	2464.407	5.099	0.532	0.995
65536 Kb	2475.014	0.863	0.035	2466.229	4.918	0.644	0.996
131072 Kb	2471.535	1.577	0.064	2470.068	2.934	0.571	0.999
262144 Kb	2462.696	1.018	0.041	2465.085	12.653	0.422	1.001
524288 Kb	2444.841	0.745	0.030	2484.761	6.970	0.499	1.016
						Average:	**0.998**

D.4 SPEC2006

SPEC CINT2006						
	Computer 1			**Computer 2**		
Test	**Fedora Ratio**	**Red Hat Ratio**	**PCR**	**Fedora Ratio**	**Red Hat Ratio**	**PCR**
400.perlbench	22.5	22.3	0.991	22.3	22.3	1.000
401.bzip2	17.4	17.4	1.000	17.3	17.4	1.006
403.gcc	14.1	19	1.348	14.1	18.9	1.340
429.mcf	18.1	17.4	0.961	17.8	17.5	0.983
445.gobmk	18.5	18.4	0.995	18.4	18.4	1.000
456.hmmer	11.5	11.4	0.991	11.5	11.3	0.983
458.sjeng	18.3	18.3	1.000	18.2	18.2	1.000
462.libquantum	21.4	22.2	1.037	21	21.2	1.010
464.h264ref	25.6	25.9	1.012	25.6	25.8	1.008
471.omnetpp	13.3	13.2	0.992	13.2	13	0.985
473.astar	12.8	12.8	1.000	12.6	12.7	1.008
483.xalancbmk	19.8	19.6	0.990	19.8	19.5	0.985
		Average:	**1.026**		**Average:**	**1.026**
		Without GCC:	**0.997**		**Without GCC:**	**0.997**
SPEC CFP2006						
	Computer 1			**Computer 2**		
Test	**Fedora Ratio**	**Red Hat Ratio**	**PCR**	**Fedora Ratio**	**Red Hat Ratio**	**PCR**
433.milc	13.4	13	0.970	13	12.6	0.969
444.namd	15.1	15.1	1.000	15	15	1.000
447.dealII	26.2	26.3	1.004	26.1	26	0.996
450.soplex	17.4	18.7	1.075	17.8	18.3	1.028
453.povray	20.4	21.1	1.034	20.9	21.3	1.019
470.lbm	20.1	20.2	1.005	18.2	19.2	1.055
482.sphinx3	22	23	1.045	22.5	22.8	1.013
		Average:	**1.019**		**Average:**	**1.012**

D.5 MATLAB Case Study

Red Hat			Fedora			
Average Execution Time (Seconds)	**STDEV**	**% STDEV**	**Average Execution Time (Seconds)**	**STDEV**	**% STDEV**	**PCR**
Computer 1						
259.4906	3.0803	1.1871	253.0496	3.2384	1.2797	1.0255
Computer 2						
272.3272	2.1640	0.7946	265.7577	2.8922	1.0883	1.0247
					Average:	**1.0251**

E Benchmark Results

E.1 Pystone

Tesla				
	Average	**STDEV**	**% STDEV**	**Ratio to Tesla**
Execution Time (seconds)	91.812	0.531	0.579	1.000
Pystones/Second	109930.704	637.231	0.580	1.000

Henry				
	Average	**STDEV**	**% STDEV**	**Ratio to Tesla**
Execution Time (seconds)	137.610	1.097	0.797	0.667
Pystones/Second	73768.628	587.924	0.797	0.671

Large				
	Average	**STDEV**	**% STDEV**	**Ratio to Tesla**
Execution Time (seconds)	175.375	2.350	1.340	0.524
Pystones/Second	74808.135	6148.454	8.219	0.681

XLarge				
	Average	**STDEV**	**% STDEV**	**Ratio to Tesla**
Execution Time (seconds)	174.947	2.396	1.369	0.525
Pystones/Second	73983.666	8236.479	11.133	0.673

High Memory XLarge				
	Average	**STDEV**	**% STDEV**	**Ratio to Tesla**
Execution Time (seconds)	126.347	0.939	0.743	0.727
Pystones/Second	80010.168	596.063	0.745	0.728

High Memory 2XLarge				
	Average	**STDEV**	**% STDEV**	**Ratio to Tesla**
Execution Time (seconds)	126.053	0.672	0.533	0.728
Pystones/Second	80201.698	427.310	0.533	0.730

High Memory 4XLarge				
	Average	**STDEV**	**% STDEV**	**Ratio to Tesla**
Execution Time (seconds)	126.236	1.027	0.814	0.727
Pystones/Second	80079.743	645.996	0.807	0.728

High Compute XLarge				
	Average	**STDEV**	**% STDEV**	**Ratio to Tesla**
Execution Time (seconds)	152.521	2.415	1.583	0.602
Pystones/Second	66330.420	1045.005	1.575	0.603

E.2 MATLAB

We created the following key to save space when displaying the MATLAB benchmark results. The mapping numbers are simply tied to each of the output values from the program, and the total time for all of the tests will be boldface.

Test	**Mapping**
Creation, transp., deformation of a 1500x1500 matrix	1
800x800 normal distributed random matrix ^1000	2
Sorting of 2,000,0000 random values	3
700x700 cross-product matrix (b = a' * a)	4
Linear regression over a 600x600 matrix (c = a \\ b')	5
Trimmed geom. mean (2 extremes eliminated):	6
FFT over 800,000 random values	7
Eigenvalues of a 320x320 random matrix	8
Determinant of a 650x650 random matrix	9
Cholesky decomposition of a 900x900 matrix	10
Inverse of a 400x400 random matrix	11
Trimmed geom. mean (2 extremes eliminated):	12
750,000 Fibonacci numbers calculation (vector calc)	13
Creation of a 5550x5550 Hilbert matrix (matrix calc)	14
Grand common divisors of 70,000 pairs (recursion)	15
Creation of a 220x220 Toeplitz matrix (loops)	16
Escoufier's method on a 37x37 matrix (mixed)	17
Trimmed geom. mean (2 extremes eliminated):	18
Total time for all 14 tests	19
Overall mean (sum of I, II and III trimmed means/3)	20

Tesla				
Test	**Average Execution Time (Seconds)**	**STDEV**	**% STDEV**	**Ratio to Tesla**
1	0.073	0.004	5.217	1
2	0.222	0.010	4.544	1
3	0.956	0.038	3.992	1
4	0.872	0.031	3.54	1
5	0.604	0.014	2.25	1
6	0.489	0.010	1.949	1
7	0.224	0.008	3.471	1
8	2.982	0.239	8.005	1
9	0.622	0.010	1.608	1
10	0.665	0.010	1.491	1
11	0.485	0.009	1.919	1
12	0.585	0.007	1.177	1
13	0.131	0.006	4.47	1
14	0.477	0.019	4.024	1
15	0.585	0.035	6.001	1
16	0.010	0.000	3.986	1
17	0.306	0.028	9.093	1
18	0.268	0.009	3.424	1
19	**9.225**	**0.269**	**2.915**	**1**
20	0.425	0.006	1.487	1
Average Tests/Sec	**1.518**	**-**	**-**	1

Henry				
Test	**Average Execution Time (Seconds)**	**STDEV**	**% STDEV**	**Ratio to Tesla**
1	2.604	0.169	6.495	0.028
2	0.489	0.008	1.567	0.454
3	1.154	0.050	4.344	0.829
4	1.643	0.011	0.688	0.530
5	1.240	0.060	4.845	0.487
6	1.330	0.033	2.498	0.368
7	0.532	0.060	11.262	0.421
8	5.797	0.439	7.571	0.514
9	1.270	0.043	3.365	0.490
10	1.510	0.030	1.994	0.440
11	0.952	0.029	3.036	0.510

12	1.222	0.032	2.58	0.479
13	0.286	0.010	3.612	0.459
14	0.197	0.008	4.121	2.419
15	0.015	0.000	1.359	40.081
16	0.335	0.004	1.299	0.029
17	0.094	0.002	2.113	3.265
18	18.039	0.711	3.941	0.015
19	**0.534**	**0.008**	**1.555**	**17.272**
20	0.425	0.006	1.487	1.000
Average Tests/Sec	**26.213**	**-**	**-**	**17.272**

Large				
Test	**Average Execution Time (Seconds)**	**STDEV**	**% STDEV**	**Ratio to Tesla**
1	0.166	0.004	2.631	0.438
2	1.701	0.528	31.015	0.130
3	1.386	0.012	0.868	0.690
4	1.937	0.227	11.702	0.450
5	1.417	0.055	3.891	0.426
6	1.448	0.097	6.666	0.338
7	0.609	0.006	0.91	0.367
8	6.655	0.209	3.143	0.448
9	1.464	0.059	4.043	0.425
10	1.979	0.099	5.015	0.336
11	1.117	0.055	4.905	0.434
12	1.479	0.045	3.072	0.396
13	0.720	0.078	10.839	0.182
14	1.828	0.034	1.875	0.261
15	0.935	0.011	1.171	0.625
16	0.020	0.003	12.494	0.486
17	0.514	0.073	14.236	0.596
18	0.702	0.039	5.539	0.381
19	**22.544**	**0.601**	**2.664**	**0.409**
20	1.146	0.033	2.848	0.371
Average Tests/Sec	**0.621**	**-**	**-**	**0.409**

XLarge

Test	Average Execution Time (Seconds)	STDEV	% STDEV	Ratio to Tesla
1	0.160	0.007	4.124	0.455
2	1.468	0.433	29.495	0.151
3	1.381	0.015	1.070	0.692
4	1.719	0.432	25.157	0.507
5	1.510	0.120	7.947	0.400
6	1.414	0.134	9.501	0.346
7	0.659	0.012	1.835	0.340
8	7.471	0.844	11.291	0.399
9	1.534	0.124	8.102	0.405
10	2.007	0.183	9.141	0.331
11	1.217	0.160	13.128	0.399
12	1.553	0.126	8.085	0.377
13	0.641	0.196	30.594	0.204
14	1.795	0.074	4.122	0.266
15	0.940	0.021	2.255	0.622
16	0.020	0.003	12.755	0.490
17	0.722	0.545	75.443	0.424
18	0.754	0.091	12.068	0.355
19	**23.460**	**1.614**	**6.882**	**0.393**
20	1.183	0.079	6.699	0.359
Average Tests/Sec	**0.597**	**-**	**-**	**0.393**

High Memory XLarge				
Test	**Average Execution Time (Seconds)**	**STDEV**	**% STDEV**	**Ratio to Tesla**
1	0.115	0.004	3.647	0.632
2	1.362	0.409	30.014	0.163
3	1.148	0.005	0.450	0.833
4	2.185	0.692	31.673	0.399
5	1.110	0.282	25.385	0.544
6	1.195	0.178	14.927	0.409
7	0.325	0.015	4.550	0.689
8	3.408	0.318	9.343	0.875
9	1.222	0.348	28.461	0.509
10	1.624	0.533	32.794	0.409
11	0.893	0.249	27.857	0.543
12	1.210	0.359	29.645	0.484
13	0.571	0.167	29.267	0.230

14	1.071	0.051	4.788	0.446
15	0.728	0.011	1.496	0.804
16	0.012	0.000	0.672	0.843
17	0.299	0.049	16.346	1.022
18	0.499	0.045	9.078	0.536
19	**16.248**	**2.525**	**15.540**	**0.568**
20	0.897	0.123	13.688	0.473
Average Tests/Sec	**0.862**	**-**	**-**	**0.568**

High Memory 2XLarge				
Test	**Average Execution Time (Seconds)**	**STDEV**	**% STDEV**	**Ratio to Tesla**
1	0.113	0.003	2.762	0.642
2	1.333	0.372	27.901	0.167
3	1.136	0.002	0.165	0.842
4	1.720	0.328	19.045	0.507
5	1.010	0.081	8.043	0.598
6	1.136	0.101	8.914	0.430
7	0.317	0.014	4.519	0.706
8	3.219	0.185	5.739	0.926
9	1.119	0.097	8.709	0.556
10	1.438	0.208	14.458	0.462
11	0.804	0.133	16.516	0.603
12	1.090	0.125	11.477	0.537
13	0.544	0.083	15.234	0.241
14	1.043	0.036	3.490	0.458
15	0.713	0.011	1.531	0.821
16	0.012	0.000	0.379	0.850
17	0.368	0.093	25.335	0.832
18	0.522	0.054	10.292	0.512
19	**14.991**	**0.999**	**6.666**	**0.615**
20	0.865	0.065	7.501	0.491
Average Tests/Sec	**0.934**	**-**	**-**	**0.615**

High Memory 4XLarge				
Test	**Average Execution Time (Seconds)**	**STDEV**	**% STDEV**	**Ratio to Tesla**

1	0.114	0.003	2.585	0.639
2	1.367	0.305	22.318	0.162
3	1.137	0.001	0.074	0.841
4	1.513	0.470	31.034	0.576
5	0.912	0.281	30.840	0.662
6	1.071	0.188	17.543	0.457
7	0.317	0.002	0.714	0.706
8	3.213	0.365	11.355	0.928
9	0.913	0.281	30.724	0.681
10	1.075	0.400	37.173	0.618
11	0.672	0.204	30.331	0.722
12	0.870	0.269	30.855	0.673
13	0.460	0.149	32.480	0.285
14	1.012	0.053	5.279	0.472
15	0.715	0.011	1.549	0.818
16	0.012	0.000	0.263	0.849
17	0.425	0.109	25.541	0.719
18	0.519	0.069	13.205	0.516
19	**14.088**	**1.951**	**13.852**	**0.655**
20	0.785	0.140	17.871	0.541
Average Tests/Sec	**0.994**	**-**	**-**	**0.655**

High Compute XLarge				
Test	**Average Execution Time (Seconds)**	**STDEV**	**% STDEV**	**Ratio to Tesla**
1	0.136	0.005	3.395	0.536
2	1.321	0.330	25.008	0.168
3	1.184	0.001	0.100	0.808
4	1.552	0.389	25.056	0.561
5	1.200	0.200	16.661	0.503
6	1.213	0.140	11.518	0.403
7	0.488	0.040	8.177	0.458
8	5.819	0.560	9.621	0.512
9	1.156	0.218	18.837	0.538
10	1.322	0.298	22.547	0.503
11	0.940	0.163	17.322	0.516
12	1.129	0.181	16.024	0.519
13	0.512	0.126	24.554	0.256
14	1.456	0.050	3.466	0.328
15	0.812	0.009	1.071	0.720

16	0.018	0.000	0.475	0.559
17	0.531	0.152	28.667	0.577
18	0.604	0.080	13.310	0.443
19	**18.592**	**1.709**	**9.193**	**0.496**
20	0.939	0.104	11.045	0.452
Average Tests/Sec	**0.753**	**-**	**-**	**0.496**

E.3 RAMSMP

E.3.1 Float Memory

Tesla				
Test	**Average Bandwidth**	**STDEV**	**%STDEV**	**Ratio to Tesla**
Copy:	26811.296	156.187	0.583	1.000
Scale:	26827.098	149.121	0.556	1.000
Add:	29829.270	64.634	0.217	1.000
Triad:	29763.706	60.009	0.202	1.000
AVERAGE:	28308.015	66.276	0.234	1.000

Henry				
Test	**Average Bandwidth**	**STDEV**	**%STDEV**	**Ratio to Tesla**
Copy:	7567.733	50.121	0.662	0.282
Scale:	7569.323	46.038	0.608	0.282
Add:	8383.630	59.220	0.706	0.281
Triad:	8382.109	16.563	0.198	0.282
AVERAGE:	7975.794	36.327	0.455	0.282

Large				
Test	**Average Bandwidth**	**STDEV**	**%STDEV**	**Ratio to Tesla**
Copy:	3099.067	135.437	4.37	0.116
Scale:	3106.549	128.886	4.149	0.116
Add:	3325.480	160.062	4.813	0.111
Triad:	3320.193	134.231	4.043	0.112
AVERAGE:	3213.911	112.276	3.493	0.114

XLarge				
Test	**Average Bandwidth**	**STDEV**	**%STDEV**	**Ratio to Tesla**
Copy:	7084.444	699.217	9.87	0.264
Scale:	7227.356	737.715	10.207	0.269
Add:	8315.533	869.029	10.451	0.279
Triad:	8375.658	763.634	9.117	0.281
AVERAGE:	7758.458	701.678	9.044	0.274

High Memory XLarge				
Test	**Average Bandwidth**	**STDEV**	**%STDEV**	**Ratio to Tesla**
Copy:	7632.920	96.149	1.26	0.285
Scale:	7759.231	104.441	1.346	0.289
Add:	8740.896	109.482	1.253	0.293
Triad:	8756.819	107.907	1.232	0.294
AVERAGE:	8222.597	94.016	1.143	0.290

High Memory 2XLarge				
Test	**Average Bandwidth**	**STDEV**	**%STDEV**	**Ratio to Tesla**
Copy:	12443.641	187.104	1.504	0.464
Scale:	12513.761	539.197	4.309	0.466
Add:	12321.083	134.198	1.089	0.413
Triad:	12377.618	241.457	1.951	0.416
AVERAGE:	12442.312	227.064	1.825	0.440

High Memory 4XLarge				
Test	**Average Bandwidth**	**STDEV**	**%STDEV**	**Ratio to Tesla**
Copy:	18392.175	392.422	2.134	0.686
Scale:	18317.267	953.457	5.205	0.683
Add:	20423.929	1137.661	5.57	0.685
Triad:	20476.688	817.531	3.992	0.688
AVERAGE:	19413.844	553.463	2.851	0.686

High Compute XLarge				
Test	**Average Bandwidth**	**STDEV**	**%STDEV**	**Ratio to Tesla**
Copy:	6335.888	679.037	10.717	0.236
Scale:	6253.073	782.520	12.514	0.233
Add:	6074.514	790.595	13.015	0.204
Triad:	5945.458	740.356	12.452	0.200
AVERAGE:	5967.505	598.514	10.03	0.211

E.3.2 Float Reading

Tesla				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	150949.024	35487.900	23.510	1.000
2 Kb	155037.087	21400.002	13.803	1.000
4 Kb	164922.755	22144.399	13.427	1.000
8 Kb	168294.454	19505.457	11.590	1.000
16 Kb	167562.347	20247.290	12.083	1.000
32 Kb	160533.859	21572.459	13.438	1.000
64 Kb	140757.656	19847.639	14.101	1.000
128 Kb	139516.252	19332.246	13.857	1.000
256 Kb	133094.761	16664.532	12.521	1.000
512 Kb	125557.103	14239.994	11.341	1.000
1024 Kb	121651.791	16168.554	13.291	1.000
2048 Kb	78590.105	8182.766	10.412	1.000
4096 Kb	40951.349	1427.093	3.485	1.000
8192 Kb	39231.229	2951.555	7.523	1.000
16384 Kb	38899.364	3047.553	7.834	1.000
32768 Kb	39031.374	2705.305	6.931	1.000
65536 Kb	38831.420	3308.944	8.521	1.000
131072 Kb	38950.614	3157.213	8.106	1.000
262144 Kb	38997.202	3060.449	7.848	1.000
524288 Kb	39118.685	3333.496	8.521	1.000

Henry				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	131005.302	705.116	0.538	0.868
2 Kb	132300.808	25.673	0.019	0.853

4 Kb	132810.786	19.427	0.015	0.805
8 Kb	133077.242	20.074	0.015	0.791
16 Kb	131813.031	174.431	0.132	0.787
32 Kb	132494.662	24.409	0.018	0.825
64 Kb	131250.485	25.993	0.020	0.932
128 Kb	50418.560	9.489	0.019	0.361
256 Kb	47021.668	5582.941	11.873	0.353
512 Kb	41427.649	7392.155	17.844	0.330
1024 Kb	19874.531	1443.653	7.264	0.163
2048 Kb	11554.638	561.112	4.856	0.147
4096 Kb	10808.572	1013.457	9.376	0.264
8192 Kb	11342.321	626.829	5.526	0.289
16384 Kb	11022.463	896.496	8.133	0.283
32768 Kb	11173.119	686.575	6.145	0.286
65536 Kb	11416.298	385.957	3.381	0.294
131072 Kb	11440.812	242.166	2.117	0.294
262144 Kb	11584.256	50.285	0.434	0.297
524288 Kb	11608.901	61.956	0.534	0.297

Large				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	29190.719	4720.553	16.171	0.193
2 Kb	31508.804	3063.888	9.724	0.203
4 Kb	31689.377	3124.365	9.859	0.192
8 Kb	31369.737	3444.822	10.981	0.186
16 Kb	30896.744	3958.256	12.811	0.184
32 Kb	30156.446	4104.534	13.611	0.188
64 Kb	21348.931	3232.918	15.143	0.152
128 Kb	21187.548	3336.291	15.746	0.152
256 Kb	21362.986	2997.335	14.031	0.161
512 Kb	21335.823	3023.580	14.171	0.170
1024 Kb	21323.939	2973.511	13.944	0.175
2048 Kb	20989.165	2825.751	13.463	0.267
4096 Kb	15931.778	5322.393	33.407	0.389
8192 Kb	4156.852	446.816	10.749	0.106
16384 Kb	3816.750	471.815	12.362	0.098
32768 Kb	3807.691	437.803	11.498	0.098
65536 Kb	3795.106	463.331	12.209	0.098
131072 Kb	3798.868	447.385	11.777	0.098
262144 Kb	3814.332	519.551	13.621	0.098
524288 Kb	3691.799	377.147	10.216	0.094

XLarge				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	56187.126	8393.210	14.938	0.372
2 Kb	62166.956	7183.644	11.555	0.401
4 Kb	61346.898	7048.301	11.489	0.372
8 Kb	62726.913	8072.249	12.869	0.373
16 Kb	62267.404	8068.652	12.958	0.372
32 Kb	59704.511	7216.391	12.087	0.372
64 Kb	51873.813	6717.670	12.950	0.369
128 Kb	50967.348	6942.900	13.622	0.365
256 Kb	46834.659	5320.351	11.360	0.352
512 Kb	42564.101	3559.527	8.363	0.339
1024 Kb	31348.201	5908.298	18.847	0.258
2048 Kb	14066.003	2411.896	17.147	0.179
4096 Kb	11778.819	1152.165	9.782	0.288
8192 Kb	11665.170	995.914	8.537	0.297
16384 Kb	11427.646	659.875	5.774	0.294
32768 Kb	11462.429	945.864	8.252	0.294
65536 Kb	11507.519	815.588	7.087	0.296
131072 Kb	11699.688	1022.213	8.737	0.300
262144 Kb	11543.503	609.875	5.283	0.296
524288 Kb	11757.280	476.898	4.056	0.301

High Memory XLarge				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	40471.073	5130.702	12.677	0.268
2 Kb	41451.619	3917.390	9.451	0.267
4 Kb	41403.684	3914.652	9.455	0.251
8 Kb	41312.034	3916.688	9.481	0.245
16 Kb	41410.584	3922.272	9.472	0.247
32 Kb	40443.245	3801.220	9.399	0.252
64 Kb	34997.937	3409.048	9.741	0.249
128 Kb	35063.609	3298.995	9.409	0.251
256 Kb	33949.613	3291.302	9.695	0.255
512 Kb	32795.957	3137.644	9.567	0.261
1024 Kb	32698.692	3133.179	9.582	0.269
2048 Kb	32302.361	3199.585	9.905	0.411
4096 Kb	18779.495	492.826	2.624	0.459
8192 Kb	11107.443	108.764	0.979	0.283
16384 Kb	10780.594	738.224	6.848	0.277

32768 Kb	10759.854	754.104	7.008	0.276
65536 Kb	10769.082	789.918	7.335	0.277
131072 Kb	10779.037	774.227	7.183	0.277
262144 Kb	10772.703	770.053	7.148	0.276
524288 Kb	10773.774	770.106	7.148	0.275

High Memory 2XLarge				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	70483.778	13705.124	19.444	0.467
2 Kb	71962.230	12831.771	17.831	0.464
4 Kb	72187.012	12644.118	17.516	0.438
8 Kb	72327.142	12481.225	17.257	0.430
16 Kb	73780.244	11158.135	15.123	0.440
32 Kb	71809.635	11172.673	15.559	0.447
64 Kb	63167.743	8602.781	13.619	0.449
128 Kb	61580.715	10350.950	16.809	0.441
256 Kb	58920.385	10494.673	17.812	0.443
512 Kb	55855.450	9175.440	16.427	0.445
1024 Kb	56387.530	7698.959	13.654	0.464
2048 Kb	37342.459	8374.611	22.427	0.475
4096 Kb	18735.866	3909.012	20.864	0.458
8192 Kb	15956.081	594.260	3.724	0.407
16384 Kb	16167.784	1348.461	8.340	0.416
32768 Kb	16640.884	2168.819	13.033	0.426
65536 Kb	16985.647	2093.795	12.327	0.437
131072 Kb	17004.445	2263.309	13.310	0.437
262144 Kb	17128.374	2165.121	12.641	0.439
524288 Kb	17132.847	2142.146	12.503	0.438

High Memory 4XLarge				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	121189.983	19277.775	15.907	0.803
2 Kb	143905.347	16870.100	11.723	0.928
4 Kb	137161.899	17868.712	13.027	0.832
8 Kb	135241.295	23756.477	17.566	0.804
16 Kb	131679.805	24513.760	18.616	0.786
32 Kb	128545.893	24224.239	18.845	0.801
64 Kb	110414.636	19316.576	17.495	0.784
128 Kb	110238.554	17544.924	15.915	0.790

256 Kb	103664.853	16575.046	15.989	0.779
512 Kb	101409.038	16745.486	16.513	0.808
1024 Kb	101175.800	17572.298	17.368	0.832
2048 Kb	90715.888	12204.713	13.454	1.154
4096 Kb	38911.506	6718.956	17.267	0.950
8192 Kb	28634.755	1818.407	6.350	0.730
16384 Kb	28565.788	2141.313	7.496	0.734
32768 Kb	27908.568	1786.294	6.401	0.715
65536 Kb	28460.950	1893.375	6.653	0.733
131072 Kb	28159.540	1841.120	6.538	0.723
262144 Kb	29001.204	1687.808	5.820	0.744
524288 Kb	30556.091	2759.657	9.031	0.781

High Computing Xlarge				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	118583.384	26476.397	22.327	0.786
2 Kb	131339.318	16702.327	12.717	0.847
4 Kb	108564.625	24875.368	22.913	0.658
8 Kb	131300.988	16183.819	12.326	0.780
16 Kb	111793.789	23125.797	20.686	0.667
32 Kb	130587.150	17395.428	13.321	0.813
64 Kb	74677.178	14218.435	19.040	0.531
128 Kb	82653.390	18200.011	22.020	0.592
256 Kb	86996.593	17479.625	20.092	0.654
512 Kb	81704.257	15042.245	18.411	0.651
1024 Kb	93010.086	12547.199	13.490	0.765
2048 Kb	76400.783	16170.675	21.166	0.972
4096 Kb	38141.851	24530.435	64.314	0.931
8192 Kb	8327.210	1513.914	18.180	0.212
16384 Kb	7514.257	1071.826	14.264	0.193
32768 Kb	7366.140	941.453	12.781	0.189
65536 Kb	7464.702	804.408	10.776	0.192
131072 Kb	7071.306	877.504	12.409	0.182
262144 Kb	7392.783	1084.726	14.673	0.190
524288 Kb	7423.085	876.769	11.811	0.190

E.3.3 Float Writing

Tesla

Block Size	Average Bandwidth	STDEV	%STDEV	Raw Ratio to Tesla
1 Kb	170699.680	11302.500	6.621	1.000
2 Kb	166888.178	13579.292	8.137	1.000
4 Kb	174930.460	6515.033	3.724	1.000
8 Kb	177447.986	4364.107	2.459	1.000
16 Kb	178501.650	3560.775	1.995	1.000
32 Kb	175560.034	1025.807	0.584	1.000
64 Kb	158780.835	1820.582	1.147	1.000
128 Kb	158254.189	930.801	0.588	1.000
256 Kb	126728.006	7544.802	5.954	1.000
512 Kb	58491.587	1920.338	3.283	1.000
1024 Kb	55427.483	1364.582	2.462	1.000
2048 Kb	38201.083	1670.225	4.372	1.000
4096 Kb	18884.854	3151.541	16.688	1.000
8192 Kb	20576.670	2294.814	11.153	1.000
16384 Kb	21314.625	1554.597	7.294	1.000
32768 Kb	21543.937	938.979	4.358	1.000
65536 Kb	21536.691	833.047	3.868	1.000
131072 Kb	21387.278	872.168	4.078	1.000
262144 Kb	21259.640	600.263	2.823	1.000
524288 Kb	21247.230	448.534	2.111	1.000

Henry				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	89401.884	2160.275	2.416	0.524
2 Kb	89903.789	132.326	0.147	0.539
4 Kb	89907.005	155.545	0.173	0.514
8 Kb	89903.600	130.409	0.145	0.507
16 Kb	89295.888	118.404	0.133	0.500
32 Kb	89595.327	117.526	0.131	0.510
64 Kb	89527.741	122.836	0.137	0.564
128 Kb	44207.599	78.720	0.178	0.279
256 Kb	41788.450	4122.984	9.866	0.330
512 Kb	38472.256	6077.257	15.796	0.658
1024 Kb	13505.104	945.250	6.999	0.244
2048 Kb	5529.737	468.280	8.468	0.145
4096 Kb	5307.456	595.817	11.226	0.281
8192 Kb	5409.749	580.969	10.739	0.263
16384 Kb	5688.192	403.649	7.096	0.267
32768 Kb	5727.449	329.014	5.745	0.266
65536 Kb	5743.094	226.070	3.936	0.267

131072 Kb	5704.779	83.390	1.462	0.267
262144 Kb	5699.297	37.833	0.664	0.268
524288 Kb	5649.730	21.220	0.376	0.266

Large				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	29151.753	4585.910	15.731	0.171
2 Kb	31709.205	1590.746	5.017	0.190
4 Kb	32031.987	1672.481	5.221	0.183
8 Kb	30481.934	4013.438	13.167	0.172
16 Kb	31101.720	3436.892	11.050	0.174
32 Kb	29605.093	4386.363	14.816	0.169
64 Kb	18452.632	2830.270	15.338	0.116
128 Kb	18571.737	3138.687	16.900	0.117
256 Kb	18421.991	3441.717	18.683	0.145
512 Kb	18626.022	3398.958	18.248	0.318
1024 Kb	18599.774	3166.032	17.022	0.336
2048 Kb	16617.839	2665.645	16.041	0.435
4096 Kb	12362.992	4912.866	39.738	0.655
8192 Kb	2916.307	174.321	5.977	0.142
16384 Kb	2555.292	121.065	4.738	0.120
32768 Kb	2555.730	121.266	4.745	0.119
65536 Kb	2559.344	110.846	4.331	0.119
131072 Kb	2516.822	124.141	4.932	0.118
262144 Kb	2497.306	118.292	4.737	0.117
524288 Kb	2436.319	129.260	5.306	0.115

XLarge				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	56037.305	9280.369	16.561	0.328
2 Kb	60458.498	7976.606	13.194	0.362
4 Kb	59514.496	8224.087	13.819	0.340
8 Kb	63098.926	8768.010	13.896	0.356
16 Kb	62113.464	7940.319	12.784	0.348
32 Kb	61444.475	9018.191	14.677	0.350
64 Kb	53177.128	7802.385	14.672	0.335
128 Kb	53284.131	7573.215	14.213	0.337
256 Kb	41765.402	5196.581	12.442	0.330
512 Kb	17658.919	1365.059	7.730	0.302

1024 Kb	14713.370	2697.779	18.336	0.265
2048 Kb	9415.729	2027.419	21.532	0.246
4096 Kb	7921.425	575.823	7.269	0.419
8192 Kb	7569.497	243.836	3.221	0.368
16384 Kb	7695.381	607.477	7.894	0.361
32768 Kb	7512.110	406.030	5.405	0.349
65536 Kb	7535.495	483.730	6.419	0.350
131072 Kb	7584.972	571.689	7.537	0.355
262144 Kb	7326.014	415.755	5.675	0.345
524288 Kb	7094.219	349.996	4.934	0.334

High Memory XLarge				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	39514.220	6259.908	15.842	0.231
2 Kb	39862.597	5909.432	14.825	0.239
4 Kb	39642.626	6065.734	15.301	0.227
8 Kb	39849.421	5985.357	15.020	0.225
16 Kb	39976.102	5822.745	14.566	0.224
32 Kb	40086.597	5391.377	13.449	0.228
64 Kb	35973.019	4705.783	13.081	0.227
128 Kb	35698.720	5004.752	14.019	0.226
256 Kb	35302.448	4579.393	12.972	0.279
512 Kb	24627.305	2271.572	9.224	0.421
1024 Kb	24313.309	2017.209	8.297	0.439
2048 Kb	24058.590	2083.144	8.659	0.630
4096 Kb	15656.755	639.784	4.086	0.829
8192 Kb	9406.808	175.274	1.863	0.457
16384 Kb	8950.184	674.216	7.533	0.420
32768 Kb	8943.709	657.240	7.349	0.415
65536 Kb	8874.987	669.867	7.548	0.412
131072 Kb	8733.066	670.239	7.675	0.408
262144 Kb	8471.884	681.547	8.045	0.398
524288 Kb	7997.946	676.310	8.456	0.376

High Memory 2XLarge				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	80728.501	7119.121	8.819	0.473
2 Kb	82034.776	5923.324	7.221	0.492
4 Kb	82080.070	5897.341	7.185	0.469

8 Kb	82051.511	5863.716	7.146	0.462
16 Kb	81931.490	5605.320	6.841	0.459
32 Kb	81366.342	5442.467	6.689	0.463
64 Kb	72754.866	5224.679	7.181	0.458
128 Kb	72556.435	5611.730	7.734	0.458
256 Kb	63447.604	4489.591	7.076	0.501
512 Kb	27612.158	1678.962	6.081	0.472
1024 Kb	26236.122	1871.054	7.132	0.473
2048 Kb	19438.233	3705.682	19.064	0.509
4096 Kb	11432.161	1062.611	9.295	0.605
8192 Kb	10878.513	395.189	3.633	0.529
16384 Kb	10869.487	381.026	3.505	0.510
32768 Kb	10832.721	347.658	3.209	0.503
65536 Kb	10753.519	329.292	3.062	0.499
131072 Kb	10866.004	882.906	8.125	0.508
262144 Kb	10585.818	304.559	2.877	0.498
524288 Kb	10309.877	260.681	2.528	0.485

High Memory 4XLarge				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	122200.129	10603.494	8.677	0.716
2 Kb	141915.140	12772.982	9.000	0.850
4 Kb	137790.777	13585.060	9.859	0.788
8 Kb	133956.498	20325.247	15.173	0.755
16 Kb	132682.999	19981.956	15.060	0.743
32 Kb	126265.445	16965.435	13.436	0.719
64 Kb	114457.934	12644.362	11.047	0.721
128 Kb	109282.792	13064.547	11.955	0.691
256 Kb	107938.411	11116.527	10.299	0.852
512 Kb	64066.013	2303.008	3.595	1.095
1024 Kb	62304.844	2350.099	3.772	1.124
2048 Kb	58400.524	3935.100	6.738	1.529
4096 Kb	26071.035	5534.754	21.230	1.381
8192 Kb	16482.038	973.042	5.904	0.801
16384 Kb	16987.289	1752.633	10.317	0.797
32768 Kb	16656.161	1474.482	8.852	0.773
65536 Kb	16522.247	1234.182	7.470	0.767
131072 Kb	16272.674	1323.641	8.134	0.761
262144 Kb	16116.035	1025.764	6.365	0.758
524288 Kb	15836.382	1115.831	7.046	0.745

High Computing Xlarge				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	107537.342	33060.293	30.743	0.630
2 Kb	128373.905	16526.671	12.874	0.769
4 Kb	103684.734	24577.289	23.704	0.593
8 Kb	132575.328	15936.932	12.021	0.747
16 Kb	107826.546	23859.557	22.128	0.604
32 Kb	129745.757	16322.466	12.580	0.739
64 Kb	55950.243	11498.234	20.551	0.352
128 Kb	61052.230	13493.386	22.101	0.386
256 Kb	71770.950	16891.688	23.536	0.566
512 Kb	58379.311	10919.411	18.704	0.998
1024 Kb	62178.662	14894.915	23.955	1.122
2048 Kb	55009.898	13946.651	25.353	1.440
4096 Kb	29494.205	25010.300	84.797	1.562
8192 Kb	5348.969	1080.394	20.198	0.260
16384 Kb	4835.171	660.402	13.658	0.227
32768 Kb	4867.721	548.189	11.262	0.226
65536 Kb	4775.990	644.289	13.490	0.222
131072 Kb	4795.498	645.913	13.469	0.224
262144 Kb	4925.755	602.319	12.228	0.232
524288 Kb	4767.248	564.328	11.838	0.224

E.3.4 Int Memory

Tesla				
Test	**Average Bandwidth**	**STDEV**	**%STDEV**	**Ratio to Tesla**
Copy:	27420.512	513.351	1.872	1.000
Scale:	27647.866	450.404	1.629	1.000
Add:	27526.623	565.490	2.054	1.000
Triad:	27525.806	442.909	1.609	1.000
AVERAGE:	27533.834	200.232	0.727	1.000

Henry				
Test	**Average Bandwidth**	**STDEV**	**%STDEV**	**Ratio to Tesla**
Copy:	7599.968	55.048	0.724	0.277
Scale:	7492.633	36.316	0.485	0.271

Add:	7330.112	14.200	0.194	0.266
Triad:	7312.007	22.722	0.311	0.266
AVERAGE:	7433.743	16.897	0.227	0.270

Large				
Test	**Average Bandwidth**	**STDEV**	**%STDEV**	**Ratio to Tesla**
Copy:	3194.812	170.443	5.335	0.117
Scale:	3202.596	123.310	3.85	0.116
Add:	3215.647	134.578	4.185	0.117
Triad:	3206.159	126.915	3.958	0.116
AVERAGE:	3205.385	126.854	3.958	0.116

XLarge				
Test	**Average Bandwidth**	**STDEV**	**%STDEV**	**Ratio to Tesla**
Copy:	8863.889	187.382	2.114	0.323
Scale:	9180.613	331.001	3.605	0.332
Add:	8905.891	181.157	2.034	0.324
Triad:	8907.543	241.981	2.717	0.324
AVERAGE:	8934.317	167.827	1.878	0.324

High Memory XLarge				
Test	**Average Bandwidth**	**STDEV**	**%STDEV**	**Ratio to Tesla**
Copy:	9713.445	606.023	6.239	0.354
Scale:	9707.500	614.921	6.334	0.351
Add:	9252.426	554.846	5.997	0.336
Triad:	9253.648	551.332	5.958	0.336
AVERAGE:	9481.861	580.425	6.121	0.344

High Memory 2XLarge				
Test	**Average Bandwidth**	**STDEV**	**%STDEV**	**Ratio to Tesla**

Copy:	11311.445	772.289	6.828	0.413
Scale:	11441.751	588.259	5.141	0.414
Add:	12603.741	664.967	5.276	0.458
Triad:	12475.635	948.919	7.606	0.453
AVERAGE:	11961.621	712.735	5.959	0.434

High Memory 4XLarge				
Test	**Average Bandwidth**	**STDEV**	**%STDEV**	**Ratio to Tesla**
Copy:	20849.552	1021.750	4.901	0.760
Scale:	19838.194	1348.434	6.797	0.718
Add:	19254.640	1817.860	9.441	0.699
Triad:	19277.745	1199.067	6.22	0.700
AVERAGE:	19837.154	756.844	3.815	0.720

High Compute XLarge				
Test	**Average Bandwidth**	**STDEV**	**%STDEV**	**Ratio to Tesla**
Copy:	6314.395	725.529	11.49	0.230
Scale:	6352.578	715.255	11.259	0.230
Add:	6418.422	790.773	12.32	0.233
Triad:	6260.427	683.219	10.913	0.227
AVERAGE:	6382.179	730.958	11.453	0.232

E.3.5 Int Reading

Tesla				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	198018.630	5596.630	2.826	1.000
2 Kb	188728.279	21573.080	11.431	1.000
4 Kb	181163.267	13363.087	7.376	1.000
8 Kb	191648.691	9579.838	4.999	1.000
16 Kb	197018.296	9405.058	4.774	1.000
32 Kb	193899.987	8562.046	4.416	1.000
64 Kb	151474.144	6449.027	4.258	1.000
128 Kb	152701.997	2371.550	1.553	1.000

256 Kb	143356.156	4091.702	2.854	1.000
512 Kb	135610.031	761.606	0.562	1.000
1024 Kb	133470.482	3322.074	2.489	1.000
2048 Kb	63480.750	1315.308	2.072	1.000
4096 Kb	37802.712	1533.244	4.056	1.000
8192 Kb	37856.541	1138.410	3.007	1.000
16384 Kb	38889.955	1077.240	2.770	1.000
32768 Kb	38907.179	854.610	2.197	1.000
65536 Kb	38953.875	866.367	2.224	1.000
131072 Kb	39133.996	593.272	1.516	1.000
262144 Kb	39269.243	369.593	0.941	1.000
524288 Kb	39350.954	192.872	0.490	1.000

Henry				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	171054.001	1680.315	0.982	0.864
2 Kb	173059.291	221.138	0.128	0.917
4 Kb	173746.444	62.749	0.036	0.959
8 Kb	174077.636	31.753	0.018	0.908
16 Kb	172117.970	50.780	0.030	0.874
32 Kb	173260.698	29.101	0.017	0.894
64 Kb	170853.400	103.346	0.060	1.128
128 Kb	44669.205	57.050	0.128	0.293
256 Kb	42563.061	3569.061	8.385	0.297
512 Kb	38670.987	5400.982	13.966	0.285
1024 Kb	19411.194	1507.770	7.768	0.145
2048 Kb	10941.889	809.367	7.397	0.172
4096 Kb	10750.656	803.730	7.476	0.284
8192 Kb	10834.351	743.622	6.864	0.286
16384 Kb	11112.021	375.529	3.379	0.286
32768 Kb	11078.413	408.768	3.690	0.285
65536 Kb	10970.553	345.182	3.146	0.282
131072 Kb	11073.384	253.523	2.289	0.283
262144 Kb	11202.259	97.977	0.875	0.285
524288 Kb	11215.776	67.458	0.601	0.285

Large				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	31910.111	1882.426	5.899	0.161

2 Kb	32485.873	319.177	0.983	0.172
4 Kb	32463.335	382.074	1.177	0.179
8 Kb	32443.703	394.518	1.216	0.169
16 Kb	32155.955	1985.735	6.175	0.163
32 Kb	31639.443	1578.853	4.990	0.163
64 Kb	19929.317	1204.306	6.043	0.132
128 Kb	19927.296	976.493	4.900	0.130
256 Kb	19923.989	1000.543	5.022	0.139
512 Kb	19896.101	991.580	4.984	0.147
1024 Kb	19974.409	474.510	2.376	0.150
2048 Kb	19539.485	628.731	3.218	0.308
4096 Kb	14525.234	4615.740	31.777	0.384
8192 Kb	4110.152	222.427	5.412	0.109
16384 Kb	3636.510	120.269	3.307	0.094
32768 Kb	3638.657	118.010	3.243	0.094
65536 Kb	3628.437	127.249	3.507	0.093
131072 Kb	3624.562	138.797	3.829	0.093
262144 Kb	3634.953	121.626	3.346	0.093
524288 Kb	3610.837	147.906	4.096	0.092

XLarge				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	51437.362	11079.493	21.540	0.260
2 Kb	58332.823	7985.153	13.689	0.309
4 Kb	58884.120	8585.994	14.581	0.325
8 Kb	58792.857	11979.186	20.375	0.307
16 Kb	55790.268	11757.400	21.074	0.283
32 Kb	56543.777	10381.851	18.361	0.292
64 Kb	43768.080	7628.046	17.428	0.289
128 Kb	42142.597	7747.981	18.385	0.276
256 Kb	39123.444	6688.884	17.097	0.273
512 Kb	37087.420	5990.301	16.152	0.273
1024 Kb	30336.959	5275.493	17.390	0.227
2048 Kb	15021.674	3260.380	21.705	0.237
4096 Kb	11780.031	1127.702	9.573	0.312
8192 Kb	11810.291	1394.300	11.806	0.312
16384 Kb	12126.778	1508.652	12.441	0.312
32768 Kb	12043.654	1648.336	13.686	0.310
65536 Kb	11913.390	1583.062	13.288	0.306
131072 Kb	12396.603	1763.350	14.224	0.317
262144 Kb	12129.478	1441.468	11.884	0.309
524288 Kb	12068.207	1058.256	8.769	0.307

High Memory XLarge				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	40605.553	4717.912	11.619	0.205
2 Kb	41399.142	3938.581	9.514	0.219
4 Kb	41260.022	4048.297	9.812	0.228
8 Kb	41294.204	3909.766	9.468	0.215
16 Kb	41342.799	3910.903	9.460	0.210
32 Kb	40419.759	3807.190	9.419	0.208
64 Kb	31527.565	2993.813	9.496	0.208
128 Kb	31428.213	2998.351	9.540	0.206
256 Kb	29721.733	2882.545	9.698	0.207
512 Kb	28320.708	2759.356	9.743	0.209
1024 Kb	28321.821	2745.052	9.692	0.212
2048 Kb	28073.202	2676.248	9.533	0.442
4096 Kb	17594.238	815.532	4.635	0.465
8192 Kb	11422.539	339.206	2.970	0.302
16384 Kb	11103.424	800.387	7.208	0.286
32768 Kb	11133.299	814.983	7.320	0.286
65536 Kb	11167.214	822.001	7.361	0.287
131072 Kb	11153.675	823.704	7.385	0.285
262144 Kb	11161.950	832.694	7.460	0.284
524288 Kb	11193.759	751.449	6.713	0.284

High Memory 2XLarge				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	50809.296	13848.763	27.256	0.257
2 Kb	51890.610	14967.436	28.844	0.275
4 Kb	52376.373	14770.360	28.200	0.289
8 Kb	53998.307	14981.014	27.743	0.282
16 Kb	56480.289	15509.467	27.460	0.287
32 Kb	54986.422	14477.161	26.329	0.284
64 Kb	42923.235	11994.101	27.943	0.283
128 Kb	42185.498	12621.112	29.918	0.276
256 Kb	39870.584	11654.823	29.232	0.278
512 Kb	36101.703	10123.283	28.041	0.266
1024 Kb	39047.024	11394.913	29.183	0.293
2048 Kb	39004.219	12561.563	32.206	0.614
4096 Kb	22408.552	5518.364	24.626	0.593
8192 Kb	16492.824	2995.137	18.160	0.436
16384 Kb	15724.420	3181.643	20.234	0.404

32768 Kb	17199.105	4127.258	23.997	0.442
65536 Kb	17193.595	4285.527	24.925	0.441
131072 Kb	17529.750	4472.379	25.513	0.448
262144 Kb	17910.893	4520.403	25.238	0.456
524288 Kb	18038.342	4501.417	24.955	0.458

High Memory 4XLarge				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	115771.894	12513.284	10.809	0.585
2 Kb	138364.395	17416.962	12.588	0.733
4 Kb	134526.106	17156.508	12.753	0.743
8 Kb	126829.919	20006.416	15.774	0.662
16 Kb	123301.517	24380.086	19.773	0.626
32 Kb	123047.519	21534.979	17.501	0.635
64 Kb	91949.929	12874.221	14.001	0.607
128 Kb	88143.275	11660.953	13.230	0.577
256 Kb	83875.536	11746.480	14.005	0.585
512 Kb	81013.901	10965.785	13.536	0.597
1024 Kb	81150.583	10995.898	13.550	0.608
2048 Kb	81239.069	6952.424	8.558	1.280
4096 Kb	43356.230	4856.559	11.202	1.147
8192 Kb	28881.148	1869.471	6.473	0.763
16384 Kb	29001.475	2101.960	7.248	0.746
32768 Kb	28974.664	1926.799	6.650	0.745
65536 Kb	28890.884	1803.170	6.241	0.742
131072 Kb	29513.331	2017.812	6.837	0.754
262144 Kb	30362.248	1415.824	4.663	0.773
524288 Kb	32333.314	2687.568	8.312	0.822

High Computing Xlarge				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	115174.747	29394.274	25.521	0.582
2 Kb	135882.471	12522.144	9.215	0.720
4 Kb	116843.841	22006.591	18.834	0.645
8 Kb	136433.215	11000.801	8.063	0.712
16 Kb	116050.238	19884.988	17.135	0.589
32 Kb	131754.416	12624.223	9.582	0.679
64 Kb	67720.921	11675.880	17.241	0.447
128 Kb	71411.713	13454.425	18.841	0.468

256 Kb	79622.752	9449.330	11.868	0.555
512 Kb	71013.685	12320.521	17.350	0.524
1024 Kb	69706.132	8737.546	12.535	0.522
2048 Kb	71339.236	10746.823	15.064	1.124
4096 Kb	39789.911	23894.617	60.052	1.053
8192 Kb	8551.982	1340.624	15.676	0.226
16384 Kb	7371.077	641.784	8.707	0.190
32768 Kb	7556.404	755.979	10.004	0.194
65536 Kb	7602.567	752.839	9.902	0.195
131072 Kb	7748.740	885.204	11.424	0.198
262144 Kb	7648.946	851.049	11.126	0.195
524288 Kb	7892.440	1210.892	15.342	0.201

E.3.6 Int Writing

Tesla				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	195960.565	14549.605	7.425	1.000
2 Kb	182994.057	25568.087	13.972	1.000
4 Kb	178020.885	16466.275	9.250	1.000
8 Kb	192477.490	8291.741	4.308	1.000
16 Kb	196191.248	7395.624	3.770	1.000
32 Kb	195611.536	7721.859	3.948	1.000
64 Kb	175329.589	9350.220	5.333	1.000
128 Kb	177294.822	3506.821	1.978	1.000
256 Kb	146486.966	7722.077	5.272	1.000
512 Kb	53629.373	591.842	1.104	1.000
1024 Kb	50924.815	477.446	0.938	1.000
2048 Kb	31188.376	548.693	1.759	1.000
4096 Kb	18150.681	3084.447	16.994	1.000
8192 Kb	18267.802	2732.219	14.956	1.000
16384 Kb	20100.745	1058.784	5.267	1.000
32768 Kb	20111.976	686.868	3.415	1.000
65536 Kb	19999.201	667.127	3.336	1.000
131072 Kb	20079.564	515.742	2.568	1.000
262144 Kb	20245.574	339.518	1.677	1.000
524288 Kb	20288.638	161.454	0.796	1.000

Henry

Block Size	Average Bandwidth	STDEV	%STDEV	Raw Ratio to Tesla
1 Kb	165559.591	16957.833	10.243	0.845
2 Kb	171731.959	8762.836	5.103	0.938
4 Kb	174627.941	6494.579	3.719	0.981
8 Kb	176393.314	59.293	0.034	0.916
16 Kb	174330.871	295.615	0.170	0.889
32 Kb	174901.834	4984.286	2.850	0.894
64 Kb	170911.348	5697.554	3.334	0.975
128 Kb	44031.233	1137.063	2.582	0.248
256 Kb	39349.159	5741.088	14.590	0.269
512 Kb	35119.780	6383.686	18.177	0.655
1024 Kb	12601.817	2081.756	16.519	0.247
2048 Kb	5236.304	897.843	17.147	0.168
4096 Kb	5046.351	922.508	18.281	0.278
8192 Kb	4941.022	997.430	20.187	0.270
16384 Kb	5127.035	992.843	19.365	0.255
32768 Kb	5314.478	939.823	17.684	0.264
65536 Kb	5343.971	737.093	13.793	0.267
131072 Kb	5330.244	729.676	13.689	0.265
262144 Kb	5432.117	684.986	12.610	0.268
524288 Kb	5384.526	690.716	12.828	0.265

Large				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	31749.811	1110.994	3.499	0.162
2 Kb	32228.768	416.815	1.293	0.176
4 Kb	32243.096	499.417	1.549	0.181
8 Kb	32202.253	572.536	1.778	0.167
16 Kb	32296.375	390.395	1.209	0.165
32 Kb	31497.442	394.580	1.253	0.161
64 Kb	18888.289	2000.646	10.592	0.108
128 Kb	19135.500	2308.658	12.065	0.108
256 Kb	19447.455	2217.512	11.403	0.133
512 Kb	19608.251	2305.376	11.757	0.366
1024 Kb	19354.487	2126.076	10.985	0.380
2048 Kb	17399.303	1679.035	9.650	0.558
4096 Kb	12953.873	4737.382	36.571	0.714
8192 Kb	2968.628	159.102	5.359	0.163
16384 Kb	2632.438	84.763	3.220	0.131
32768 Kb	2628.967	66.723	2.538	0.131
65536 Kb	2617.713	76.462	2.921	0.131

131072 Kb	2601.436	66.641	2.562	0.130
262144 Kb	2575.644	72.355	2.809	0.127
524288 Kb	2523.837	68.958	2.732	0.124

XLarge				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	50551.187	9425.434	18.645	0.258
2 Kb	55704.509	9044.819	16.237	0.304
4 Kb	54997.837	9235.048	16.792	0.309
8 Kb	56511.169	12231.671	21.645	0.294
16 Kb	55287.709	10161.593	18.379	0.282
32 Kb	55544.351	10801.327	19.446	0.284
64 Kb	46949.799	8512.786	18.132	0.268
128 Kb	47190.674	8373.690	17.744	0.266
256 Kb	40590.464	7088.540	17.464	0.277
512 Kb	18094.875	1340.821	7.410	0.337
1024 Kb	16374.072	2336.926	14.272	0.322
2048 Kb	10487.141	2133.149	20.341	0.336
4096 Kb	8123.011	493.148	6.071	0.448
8192 Kb	7753.873	451.014	5.817	0.424
16384 Kb	7738.798	479.521	6.196	0.385
32768 Kb	7744.077	528.620	6.826	0.385
65536 Kb	7682.168	541.378	7.047	0.384
131072 Kb	7688.222	565.864	7.360	0.383
262144 Kb	7590.289	530.816	6.993	0.375
524288 Kb	7276.936	461.022	6.335	0.359

High Memory XLarge				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	41207.274	3917.055	9.506	0.210
2 Kb	41202.400	4062.753	9.860	0.225
4 Kb	41323.609	3923.252	9.494	0.232
8 Kb	41292.808	3922.135	9.498	0.215
16 Kb	41299.926	3922.860	9.498	0.211
32 Kb	40888.054	3862.999	9.448	0.209
64 Kb	36732.784	3384.162	9.213	0.210
128 Kb	36574.390	3597.417	9.836	0.206
256 Kb	35920.800	3423.994	9.532	0.245
512 Kb	24924.141	1542.188	6.188	0.465

1024 Kb	24445.149	1531.048	6.263	0.480
2048 Kb	24303.762	1452.207	5.975	0.779
4096 Kb	15182.900	733.473	4.831	0.836
8192 Kb	9195.665	253.101	2.752	0.503
16384 Kb	8898.616	550.343	6.185	0.443
32768 Kb	8861.416	578.433	6.528	0.441
65536 Kb	8774.797	658.900	7.509	0.439
131072 Kb	8667.144	583.099	6.728	0.432
262144 Kb	8406.469	576.870	6.862	0.415
524288 Kb	7925.990	566.712	7.150	0.391

High Memory 2XLarge				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	81918.195	5941.384	7.253	0.418
2 Kb	82269.222	5352.534	6.506	0.450
4 Kb	82480.096	5377.647	6.520	0.463
8 Kb	82360.808	5433.125	6.597	0.428
16 Kb	82578.045	5145.477	6.231	0.421
32 Kb	81649.317	5334.556	6.533	0.417
64 Kb	73059.842	4848.978	6.637	0.417
128 Kb	72870.025	5243.743	7.196	0.411
256 Kb	64095.737	3618.645	5.646	0.438
512 Kb	27490.935	1569.513	5.709	0.513
1024 Kb	26136.281	1712.513	6.552	0.513
2048 Kb	19226.068	3516.301	18.289	0.616
4096 Kb	11411.251	1047.005	9.175	0.629
8192 Kb	10877.069	366.403	3.369	0.595
16384 Kb	10897.930	496.938	4.560	0.542
32768 Kb	10826.775	359.097	3.317	0.538
65536 Kb	10767.430	325.618	3.024	0.538
131072 Kb	10734.812	543.134	5.060	0.535
262144 Kb	10560.848	267.174	2.530	0.522
524288 Kb	10330.636	483.899	4.684	0.509

High Memory 4XLarge				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	108995.009	26880.239	24.662	0.556
2 Kb	131337.241	28142.124	21.427	0.718
4 Kb	131065.904	25668.028	19.584	0.736

8 Kb	125659.440	33387.179	26.570	0.653
16 Kb	124452.760	35573.637	28.584	0.634
32 Kb	119160.071	31032.778	26.043	0.609
64 Kb	104208.469	27639.681	26.523	0.594
128 Kb	101247.342	27061.429	26.728	0.571
256 Kb	100055.552	25682.344	25.668	0.683
512 Kb	59122.458	10782.671	18.238	1.102
1024 Kb	57369.448	10192.252	17.766	1.127
2048 Kb	51574.916	10701.134	20.749	1.654
4096 Kb	23881.945	6726.716	28.167	1.316
8192 Kb	15909.417	1717.622	10.796	0.871
16384 Kb	15331.685	2576.194	16.803	0.763
32768 Kb	15503.649	2828.638	18.245	0.771
65536 Kb	15649.229	2397.424	15.320	0.782
131072 Kb	15199.779	2084.049	13.711	0.757
262144 Kb	15388.319	2439.584	15.853	0.760
524288 Kb	15039.421	2297.362	15.276	0.741

High Computing Xlarge				
Block Size	**Average Bandwidth**	**STDEV**	**%STDEV**	**Raw Ratio to Tesla**
1 Kb	113232.891	30569.393	26.997	0.578
2 Kb	135267.480	15332.489	11.335	0.739
4 Kb	101643.394	30937.621	30.437	0.571
8 Kb	135271.616	13417.258	9.919	0.703
16 Kb	102578.923	27236.923	26.552	0.523
32 Kb	131133.888	15927.656	12.146	0.670
64 Kb	56942.225	13468.332	23.653	0.325
128 Kb	61955.725	16609.875	26.809	0.349
256 Kb	75058.223	14387.523	19.168	0.512
512 Kb	58778.486	14468.540	24.615	1.096
1024 Kb	61567.866	16323.686	26.513	1.209
2048 Kb	57026.582	14081.136	24.692	1.828
4096 Kb	24283.214	23293.180	95.923	1.338
8192 Kb	4958.236	674.010	13.594	0.271
16384 Kb	4496.934	634.467	14.109	0.224
32768 Kb	4546.730	556.891	12.248	0.226
65536 Kb	4564.510	570.342	12.495	0.228
131072 Kb	4574.956	746.890	16.326	0.228
262144 Kb	4676.805	544.985	11.653	0.231
524288 Kb	4578.970	770.536	16.828	0.226

E.4 SPEC Benchmarks

SPEC CINT2006 Tests							
Results		**400 perlbench**	**401 bzip2**	**403 gcc**	**429 mcf**	**445 gobmk**	**456 hmmer**
Tesla	SPEC Ratio	25.5	19.9	24.7	42.2	24.2	46.8
	Runtime	384	484	326	216	433	199
	Tesla Ratio	1.000	1.000	1.000	1.000	1.000	1.000
Henry	SPEC Ratio	10.9	9.14	9.1	11.4	10.4	15.4
	Runtime	895	1056	884	804	1009	606
	Tesla Ratio	0.429	0.458	0.369	0.269	0.429	0.328
HM4XLarge	SPEC Ratio	21.08	14.88	13.77	18.27	18.1	9.37
	Runtime	463.377	648.425	584.775	499.24	579.647	995.55
	Tesla Ratio	0.829	0.746	0.557	0.433	0.747	0.200
HM2XLarge	SPEC Ratio	21.68	14.92	13.72	18.17	18.08	9.37
	Runtime	450.726	646.626	586.831	502.027	580.067	995.359
	Tesla Ratio	0.852	0.749	0.556	0.430	0.746	0.200
HMXLarge	SPEC Ratio	20.88	14.5	13.5	17.7	17.81	9.35
	Runtime	467.935	665.622	596.228	515.335	588.938	997.503
	Tesla Ratio	0.821	0.727	0.547	0.419	0.735	0.199
HCXLarge	SPEC Ratio	16.29	11.1	11.2	12.35	14.12	7.47
	Runtime	599.711	869.557	819.483	738.226	742.999	1248.538
	Tesla Ratio	0.640	0.557	0.398	0.293	0.583	0.159
XLarge	SPEC Ratio	19.56	15.13	9.84	14.22	16.31	10.24
	Runtime	499.484	637.864	818.155	641.326	643.075	911.099
	Tesla Ratio	0.769	0.759	0.398	0.337	0.673	0.218
Large	SPEC Ratio	12.04	8.17	7.07	8.78	10.6	5.63
	Runtime	811.128	1181.429	1138.466	1038.733	989.451	1656.633
	Tesla Ratio	0.473	0.410	0.286	0.208	0.438	0.120

SPEC CINT2006 Tests cont.							
Results		**458 sjeng**	**462 libquantum**	**464 h264ref**	**471 omnetpp**	**473 astar**	**483 xalancbmk**
Tesla	SPEC Ratio	25.9	477	37.1	21.6	20	36.1
	Runtime	468	43.4	96	290	350	191
	Tesla Ratio	1.000	1.000	1.000	1.000	1.000	1.000
Henry	SPEC Ratio	11.1	32.9	16.9	9.86	9.14	11.1
	Runtime	1086	631	1310	634	768	624
	Tesla Ratio	0.431	0.069	0.073	0.457	0.456	0.306
HM4XLarge	SPEC Ratio	17.51	27.31	27.49	12.67	12.04	20.66
	Runtime	690.838	758.729	804.898	493.304	583.118	334.007
	Tesla Ratio	0.677	0.057	0.119	0.588	0.600	0.572
HM2XLarge	SPEC Ratio	17.53	27.29	27.41	12.69	12.08	20.63
	Runtime	690.245	759.124	807.289	492.553	581.222	334.538
	Tesla Ratio	0.678	0.057	0.119	0.589	0.602	0.571
HMXLarge	SPEC Ratio	12.36	26.72	27.22	12.29	11.8	19.81
	Runtime	697.088	775.453	812.873	508.371	594.762	348.324
	Tesla Ratio	0.671	0.056	0.118	0.570	0.588	0.548
HCXLarge	SPEC Ratio	13.91	18.7	21.78	8.87	8.9	13.72
	Runtime	870.091	1108.004	1016.248	704.988	788.413	503.054
	Tesla Ratio	0.538	0.039	0.094	0.411	0.444	0.380
XLarge	SPEC Ratio	15.95	15.93	22.62	9.85	9.97	15.51
	Runtime	758.438	1300.477	978.441	634.717	703.764	444.782
	Tesla Ratio	0.617	0.033	0.098	0.457	0.497	0.429
Large	SPEC Ratio	10.36	14.63	11.88	5.09	5.55	9.67
	Runtime	1168.364	1416.168	1863.604	1227.754	1265.458	713.778
	Tesla Ratio	0.401	0.031	0.052	0.236	0.277	0.268

SPEC CFP2006 Tests							
Results	**433 milc**	**444 namd**	**470 lbm**	**447 dealII**	**450 soplex**	**453 povray**	**482 sphinx3**

Tesla	SPEC Ratio	34.3	19.1	54.7	33.9	28.2	28.6	40.6
	Runtime	267	420	252	337	296	186	480
	Tesla Ratio	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Henry	SPEC Ratio	-	-	-	-	-	-	-
	Runtime	-	-	-	-	-	-	-
	Tesla Ratio	-	-	-	-	-	-	-
HM4XLarge	SPEC Ratio	13.72	14.2	28.41	22.17	21.86	19.68	25.56
	Runtime	668.879	564.982	483.647	516.075	381.562	270.376	762.64
	Tesla Ratio	0.399	0.743	0.521	0.653	0.776	0.688	0.629
HM2XLarge	SPEC Ratio	13.57	13.88	28.06	22.19	21.42	19.71	24.85
	Runtime	676.541	577.724	489.731	515.445	389.364	269.952	784.453
	Tesla Ratio	0.395	0.727	0.515	0.654	0.760	0.689	0.612
HMXLarge	SPEC Ratio	13.63	14.19	28.37	22.11	21.83	19.72	25.67
	Runtime	673.592	565.371	484.265	517.526	381.96	269.824	759.357
	Tesla Ratio	0.396	0.743	0.520	0.651	0.775	0.689	0.632
HCXLarge	SPEC Ratio	11.2	11.32	14.49	17.17	14.21	15.74	16.28
	Runtime	819.483	708.52	948.551	666.14	586.877	338.095	1196.99
	Tesla Ratio	0.326	0.593	0.266	0.506	0.504	0.550	0.401
XLarge	SPEC Ratio	9.45	9.38	14.2	14.26	12.11	13.03	12.44
	Runtime	971.491	854.925	967.925	801.985	688.852	40.441	1567.03
	Tesla Ratio	0.275	0.491	0.260	0.420	0.430	4.599	0.306
Large	SPEC Ratio	7.18	10.13	5.09	16.41	9.88	14.23	14.16
	Runtime	1278.12	791.369	2701.489	697.305	844.086	373.804	1376.54
	Tesla Ratio	0.209	0.531	0.093	0.483	0.351	0.498	0.349