

2006-274: TEACHING COMPUTER ARCHITECTURE PERFORMANCE ANALYSIS

Gregory Palmier, Purdue University

Gregory M. Palmier

Gregory Palmier was a graduate student in the College of Technology working on computer architecture and performance.

Jeffrey Honchell, Purdue University

Jeffrey W. Honchell

Professor Honchell is an Associate Professor in the Department of Electrical and Computer Engineering technology at Purdue University, West Lafayette. His professional area of interest is teaching and applied research in the area of RF communications. Professor Honchell also has 10 years of combined industrial experience with IBM and The Johns Hopkins University Applied Physics Laboratory. He has also been actively involved in the ASEE and IEEE holding various positions over the last 10 years.

Teaching Computer Architecture Performance Analysis

Introduction

The field of Computer Engineering continues to make great strides as computer hardware consistently reduces to a fraction of its former size while increasing in speed and capabilities. Education in this field is demanding and competitive as the complexity of today's computer technology increases steadily. While the College of technology at Purdue University offers many courses teaching the use of a computer as a tool, there are few classes offering a technical "hands-on" approach to analyzing the performance of different computer architectures and fostering an understanding how computer systems function. This field of education is dominated by theoretical computer architecture classes in the School of Electrical and Computer Engineering and has little to offer students with technical backgrounds interesting in the subject of Computer engineering and Performance Analysis. Therefore a meaningful laboratory supplement to the theoretical knowledge applied to future technology is necessary for a better understanding of a computer system's architecture and performance.

The role of Electrical and Computer Engineering Technology (ECET 325) at Purdue University will provide students with learning experience of introductory computer architecture designs and theories with a required laboratory experiment each week. The goal of these laboratory experiments will be to reinforce the lecture topics of computer architecture for technology students. Students in technology curriculums need to establish proper methodologies for understanding computer performance with statistical analysis using software tools for benchmarking and analyzing computer system configurations. This will be accomplished as a part of the laboratory experience.

These performance measurements include properly analyzing the CPU, memory, bus and operating system in terms of similarities and differences. Conducting the laboratory experiments will provide students with a "real world" view of computer processing in real-time. Analyzing these characteristics of computer systems will provide students with an understanding of how to properly assess the performance of new computer systems for task specific applications.

The objective of benchmarking is to attempt to define and measure machine power in such a way that they can compare one machine with another [1]. Students in the Computer Engineering Technology field need the experience of working on a laboratory experiment to answer questions like; "What is being measured? How is performance properly analyzed? How is this affected by the operating system?" The human desire for faster technology fuels the problem facing analysts in performance tuning: it is the adaptation of the speed of a computer system to the speed requirements imposed by the real world [2]. It is these topics in performance analysis that need to be applied to keep the design requirements of computer systems moving forward.

Background

To create a method for developing exercises to analyze computer performance and modeling, a literature review process is used to organize an approach for solving the problem while providing focus on how the projects research will contribute to the technology field based on previous related work. Current computer engineering programs contain a significant amount of material for implementation of complex hardware/software systems, however the use of performance modeling to analyze computer design alternatives has not made its way into the undergraduate curriculum [3].

The literature review process involves four methods of finding literature relevant to the problem, purpose, significance and methodology for the study. The first goal of this literature review is to find resources that are relevant to the understanding of computer system benchmarking, modeling, and analysis as a tool. While there are many resources available to computer architecture design, resources applicable to the understanding of the computer benchmark, modeling and analysis topics themselves are researched. The topics of these resources include the role of benchmarking in technology, a comparison of different benchmarking types, and the interpretation of performance data. This research is an analysis of the traditional ideas and methods of benchmarking that do not focus directly on the benchmark programming, instead how the benchmarking tools can be used. This includes researching computer system analysis and the definition of computer system performance analysis as a field of study.

Secondly, the representation of numbers and visual data must be understood to properly quantify the results of the analysis of a given computer system. Performance graphs are used to depict data to a human being as a method of understanding what the numbers are representing. Data represented visually can be skewed according to a manufacturer's, author's or programmer's bias towards certain architectures or computer system features, therefore understanding analysis reports and visual data is researched in conjunction with how the computer analysis data is obtained.

Next, the methods of education undergraduate students, topics in computer architecture and performance analysis are researched. More recent publications including journals and conference papers are highly relevant because they represent a more recent observation of current industrial and educational focus. Educational goals of the College of Technology provide a greater focus for this portion of the research by discerning information relevant to solving the problem of this study.

The last research goal involves gathering methods of using computer software tools to analyze computer systems in a limited time instructional setting. Resources applicable to the use of software tools include operating system manuals, software package manuals, system performance guides, and instructional computer analysis printed works. The use of the Internet is helpful for the high level software tools readily available to solve the problem of the study. This includes using benchmarking tools to analyze the computer system as a whole or in part. More research pertaining to this area will come from more recent publications such as journal articles and conference papers. Due to the fact, that

there are many different computer architectures and these analysis methods may only pertain to new, more sophisticated techniques of analyzing performance properly. There are two types of computer benchmarks, system and component. The use of benchmark software distinguishing the two types will be up-to-date with today's current computer technology; therefore, traditional computer performance ideas will not aide this portion of the literature finding. .

Literature Review

The results of the literature search indicate a need for more conceptualized analysis of computer systems, to bring the critical topics of implementing alternative computer system designs into technology education. Design alternatives must be evaluated early in the design process before detailed information about the system exists; therefore the use of abstract architecture models is used to create performance models [3]. The relevance of teaching these topics have been of interest since the establishment of the *Joint Task Force on Computing Curricula 2001* by the *Computer Society of the Institute for Electrical and Electronic Engineers (IEEE)* and the *Association for Computer Machinery (ACM)* to evaluate the computer curriculum in undergraduate programs. Outside of computer engineering principles, objectives of this evaluation include important issues for designers, and tools needed for carrying out research [4]. The education of computer system design using modeling and evaluation methods not only includes computer architecture as seen through the eyes of a programmer, but how the implementation and organization of the computer system will dictate its design [4]. Implementation and organization of a computer system design includes all components of the system which affect the computer architecture. This project will analyze multiple computer system component designs which include operating system choices, memory and motherboard architecture, file systems I/O devices such as data storage devices and computer process handling best suited for computer system design applications. Tools for performance analysis in place have been successful in analyzing hardware and software components of a design for computer professionals, however, use of these tools by undergraduates requires ease of understanding so that design alternatives of computer systems can be applied.

In a paper describing the evolution of computer performance evaluation since the 1960's, benchmarking was described as the only technique to receive consistent use as a performance evaluation tool [5]/ Research emphasized the increasing importance of performance evaluation by the establishment of the ACM in 1971. The ACM recognized a need for understanding the relationship between computer system models and the tolls useful for performance evaluation. More research shows that most undergraduate engineering programs spend significant time on implementing complex hardware and software designs, but do not utilize performance modeling as a tool for analyzing design alternatives. Computer manufacturers often use benchmarks with an agenda to show superiority in a competitive market, however, results often mislead system designers with inaccurate depictions of component performance. Research shows that using traditional statistical methods causes these bad interpretations and designers much distinguish between traditional statistics and computer-related statistical methods which accurately

depict the computer performance. Software tools often summarize computer performance into one number for conclusions, but can only be useful to a system designer if considered properly [6]. More findings indicate the use and abuse of performance monitoring systems. A similar study's results, from the *Joint Task Force on Computing Curricula 2001 (CC2001)*, reveals a semester long timeline for computer architecture topics and also outlines course objectives and amount of time spent on topics. However, the CC2001 study focused more specifically on processor architecture by lecture and had no laboratory exercises. The goal of this project is to continue with a similar model produced by the CC2001 and reinforce the lecture topics with laboratory instruction for better comprehension. Compared to lecture-only instruction, reinforcement with laboratory exercises can increase the comprehension of computer system performance topics.

Previous studies related to the education environment of computer engineering undergraduates have determined that the success of this project strategy depends on completing the goal considering the following variables. First, the material must be understandable on a level comparable to the complexity of similar computer system undergraduate projects. Second, all of the reference material necessary for completing laboratory exercises must be available in what [3] calls a "prefabricated" environment. With two hour per week time constraints as a limitation of potential student success, quick access to material references aiding student exercises is essential. Third, the material needs to be flexible so that it can be applied to different applications and repeatable in different projects. Limitations of this variable are defined by the computer system architectures that will be used in the laboratory experiments and the breadth of abilities the different modeling analysis tools provide for analyzing systems. For instance, there is no "all encompassing" computer benchmark tool available for any computer architecture and operating system, so different tools will be utilized for analyzing different components of the computer system. The laboratory experiments will utilize Microsoft Windows, Debian Linux and Sun Solaris which are widely used for both personal workstations and large scale computer systems. Another variable relating to the success of the student's understanding of the laboratory material will be the depth of requirements completed outside of lab in preparation for the lab session. This method is used by the ECET department's strategy for preparing the students to be ready to utilize the short time in lab efficiently. Limitations of the "pre-lab" need to be concise with providing information necessary to complete the laboratory experiment, without requiring the use of any computer performance tools.

Variables in selecting the analysis software used in the laboratory experiments will depend on the following characteristics. The amount of time running the analysis tools must be efficient and provide for the limitations for the tools that will be acceptable for laboratory exercises. Software tools that require large amounts of time are not useful for laboratory experiments if the students are waiting for results during the entire laboratory session. Also, the computer tools must be able to meet the minimum hardware requirements for the computer system. This limitation included ensuring hardware compatibility with the operating system and analysis tools. Findings of software tools available show that **sar**, **vmstat**, and tools offered by SiSoftware® and Veritest®

companies are some examples of software that are free to use and analyze different aspects of computer system components.

After completing the laboratory experiments, students should have an understanding of computer architecture by performing the “hands-on” work and were able to integrate theoretical designs and practical experiences to performance metrics.

Definition of Terms

Benchmark – The process of performance comparison for two or more systems by measurements [8].

Computer Architecture – The structure of a computer that a machine language programmer must understand to write a correct (timing independent) program for that machine [9].

Organization – In terms of computer, includes the high- level aspects of a computer’s design, such as the memory system, the bus structure, and design of the internal CPU [7].

Performance analysis – As applied to experimental computer science and engineering, should be thought of as a combination of measurement, interpretation and communication of a computer system’s ‘speed’ or ‘size’ [10].

Throughput – Measure of the amount of computation performed per unit time [10].

Introduction to the Experimental Process

To prepare the laboratory experiments, each computer architecture topic presented in lecture was researched. Performance analysis methods of each topic were discovered through the analysis of software tools. The analysis tools were gathered, tested and critiqued on the basis of compatibility, ease of use, completion time and results. Figure 1 shows a flowchart diagram of the experiment creation process.

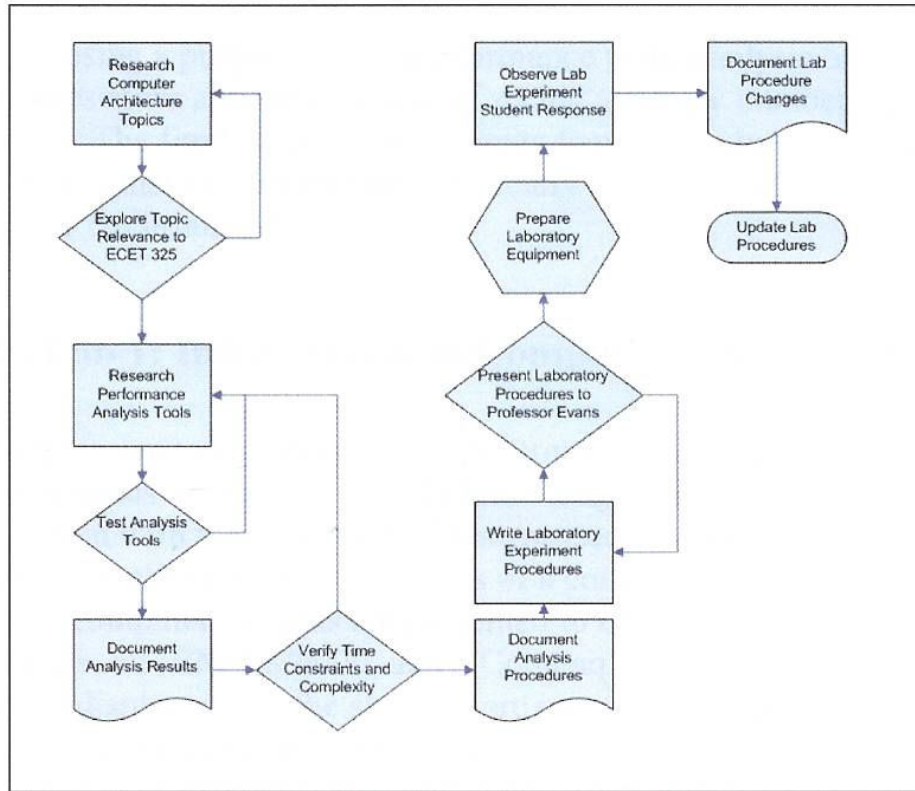


Figure1: Procedure Creation Process

Each of the lab procedures focus on the ease of gathering basic data. The use of the **script** command under the Linux platform is used consistently to allow users to record and import data easily into an analysis program (e.g. Microsoft Excel). The ability to gather data easily was a focus in this project considering the time allotted and student knowledge required for the experiments to be completed.

Experiments one through four were designed for students to become acquainted with the computer and operating systems before being able to analyze the operation. Experiments five through eight were designed to understand the importance and purpose of computer performance analysis. Table1 shows the semester long laboratory experiment plan with duration.

Table1: ECET 325 Laboratory Experiments

Experiment #	Title	Duration (weeks)
1	Introduction to Computer Hardware Platforms	1
2	Installing Operating Systems	2
3	Data Types and Time-keeping	1
4	Operating a Computer	1
5	Introduction to Benchmarking	1
6	Linux Performance Analysis	1
7	File System Performance	1
8	Multi-Platform Analysis	3
9	Memory Logic Analyzer	1
10	Introduction to Computer Clusters	2

As imposing a purpose of the performance test, analysis of what the data represents becomes more apparent instead of seeing arbitrary number reports from analysis software. The final class of experiments focused analysis of single components (e.g. CPU, memory) that can be pedagogically successful after creating competence with the computer environment. For brevity only two of the experiments are discussed in detail.

Data Types and Time-Keeping

The objective of this experiment is to identify differences in CPU instruction time and become acquainted with system analysis tools. A pre-laboratory assignment was created to prepare students to run a program and keep track of its execution time. The laboratory procedures are to modify, compile and run a small 'C' language program used to calculate the time of integer and floating point processing power of each platform. The program is a simple number addition loop, increasing the number of addition operations through loop iteration by a factor of 10. Figure 2 displays the syntax to compile the code for Linux. Figure 3 displays the compile syntax in the Windows environment.

```
ecet325@debian:~$ gcc lab3.c -o lab3
ecet325@debian:~$ gcc lab3all.c -o lab3all
```

Figure 2: Compiling Two Programs Under Linux

```
>bcc32.exe -Ic:\Borland\BCC55\Include -Lc:\Borland\BCC55\Lib ../lab3.c
```

Figure 3: Compiling Programs Under Windows

The duration of the integer and floating point calculations were recorded using a timer in a C program. The lab 3 integer benchmark programs were run under the Linux and Windows operating systems. The results for each program are shown in Figure 4 and Figure 5.


```

ecet325@debian:~$ ./lab3
addition operations: integers
type    number          time
int:    1.000000e+04   0.000000e+00
int:    1.000000e+05   0.000000e+00
int:    1.000000e+06   2.000000e-02
int:    1.000000e+07   1.600000e-01
int:    1.000000e+08   1.610000e+00
int:    1.000000e+09   1.617000e+01

```

Figure 4: Linux Integer analysis

```

C:\Borland\BCC55\Bin>lab3.exe
addition operations: integers
type    number          time
int:    1.000000e+04   0.000000e+00
int:    1.000000e+05   0.000000e+00
int:    1.000000e+06   1.000000e-02
int:    1.000000e+07   8.000000e-02
int:    1.000000e+08   7.610000e-01
int:    1.000000e+09   7.581000e+00

```

Figure 5: Windows Integer Analysis

Next, the students were instructed to modify the code to analyze both integer and floating point functions. Figure 6 and figure 7 display the results of the integer and floating point program under each operating system.

```

ecet325@debian:~$ ./lab3all
addition operations: ints and floats
type    number          time
int:    1.000000e+04   0.000000e+00
float:  1.000000e+04   0.000000e+00
int:    1.000000e+05   0.000000e+00
float:  1.000000e+05   0.000000e+00
int:    1.000000e+06   2.000000e-02
float:  1.000000e+06   2.000000e-02
int:    1.000000e+07   1.800000e-01
float:  1.000000e+07   2.000000e-01
int:    1.000000e+08   1.800000e+00
float:  1.000000e+08   2.010000e+00
int:    1.000000e+09   1.783000e+01
float:  1.000000e+09   2.013000e+01

```

Figure 6: Linux Integer and Floating Point Results

```
C:\Borland\BCC55>lab3all.exe
addition operations: ints and floats
type      number          time
int:      1.000000e+04  0.000000e+00
float:    1.000000e+04  1.000000e-02
int:      1.000000e+05  0.000000e+00
float:    1.000000e+05  0.000000e+00
int:      1.000000e+06  1.000000e-02
float:    1.000000e+06  5.100000e-02
int:      1.000000e+07  7.000000e-02
float:    1.000000e+07  4.900000e-01
int:      1.000000e+08  6.710000e-01
float:    1.000000e+08  4.947000e+00
int:      1.000000e+09  6.730000e+00
float:    1.000000e+09  4.941100e+01
```

Figure 7: Windows Integer and Floating Point Results

The base install of Debian is packages with the GCC compiler and a free Borland compiler was configured for Windows. Using GCC under Windows would have involved configuring Cygwin, a Linux-like environment for Windows on the test computers, but this wasn't done to give a more broad difference in results for comparison in the laboratory report. Not only are computer performance results skewed due to the platform, but also the compilers too, depending on how the functions of the processor are utilized by the compiler. The results of the integer and floating point tests on the two different platforms using different compilers were very significant, which showed the floating point calculations to take up to ten times longer with the Borland compiler on Windows versus the GCC compiler using Linux. The results of these procedures show that there is more than one way to manage the architecture of a computer system not only by the operating system, but also the compiler and the subtle differences in data types. The laboratory report includes presenting integer and floating point performance results. The data is analyzed, detailing the differences between using the two platforms for analysis.

Filesystem Performance Analysis

The objectives of this experiment are for the student to benchmark and analyze the performance of the hard disk drive and to quantify the performance metrics of hard drive performance characteristics. The first procedure [appendix] involves installing the I/O performance tools used in the experiment. Retrieving the **tiobench**, **systat**, and **hdparm** software tools requires using the internet to install the software. Figure 8 shows the syntax required to install the software.

```

ecet325@debian:~$ su
Password:
debian:/home/ecet325# apt-get update
 [apt-get update status feedback]
debian:/home/ecet325# apt-get install tiobench hdparm
Reading Package Lists... Done
Building Dependency Tree... Done
The following NEW packages will be installed:
  tiobench hdparm
0 upgraded, 2 newly installed, 0 to remove and 9 not upgraded.
 [apt-get download and install status feedback]
debian:/home/ecet325#

```

Figure 8: Installing Hard Disk Analysis Tools

The **sysat** software suite includes more than one program for performance analysis, therefore verifying **vmstat** installation is required for this experiment. The installation of the software tools was then verified and is shown in figure 9.

```

debian:/home/ecet325# which hdparm tiobench vmstat
/sbin/hdparm
/usr/bin/tiobench
/usr/bin/vmstat
debian:/home/ecet325#

```

Figure 9: Software Tool Installation Verified

The first performance analysis test to be run included using the **tiotest** benchmark program. The data generated by this program is shown in Figure 10.

```

debian:/home/ecet325# tiotest
Tiotest results for 4 concurrent io threads:
-----
| Item                | Time      | Rate           | Usr CPU | Sys CPU | |
|---|---|---|---|---|---|
| Write               | 40 MBs   | 4.7 s | 8.544 MB/s | 0.4 % | 13.5 % |
| Random Write       | 16 MBs   | 22.9 s | 0.682 MB/s | 0.1 % | 0.6 % |
| Read               | 40 MBs   | 0.2 s | 252.952 MB/s | 0.0 % | 88.5 % |
| Random Read        | 16 MBs   | 0.1 s | 235.572 MB/s | 30.2 % | 60.3 % |
-----

Tiotest latency results:
-----
| Item                | Average latency | Maximum latency | % >2 sec | % >10 sec |
-----|-----|-----|-----|-----|
| Write               | 0.198 ms | 190.143 ms | 0.00000 | 0.00000 |
| Random Write       | 4.855 ms | 15366.723 ms | 0.05000 | 0.02500 |
| Read               | 0.014 ms | 0.095 ms | 0.00000 | 0.00000 |
| Random Read        | 0.014 ms | 0.059 ms | 0.00000 | 0.00000 |
-----
| Total               | 0.760 ms | 15366.723 ms | 0.00702 | 0.00351 |
-----

```

Figure 10: Tiotest Benchmark Results

To analyze different performance enhancing features of the hard disk, the disk write-caching feature is disabled and the **tiotest** benchmark is re-analyzed. Figure 11 shows how write-caching is disabled and Figure 12 displays the **tiotest** results.

```

debian:/home/ecet325# hdparm -W0 /dev/hda

/dev/hda:
  setting drive write-caching to 0 (off)
  
```

Figure 11: Disabling Hard Disk Write-Caching

```

debian:/home/ecet325# tiotest
Tiotest results for 4 concurrent io threads:
-----
| Item                | Time      | Rate      | Usr CPU | Sys CPU |
-----+-----+-----+-----+-----+
| Write               | 40 MBs | 11.7 s | 3.423 MB/s | 0.2 % | 5.2 % |
| Random Write       | 16 MBs | 25.3 s | 0.618 MB/s | 0.0 % | 0.6 % |
| Read               | 40 MBs | 0.2 s | 251.964 MB/s | 18.9 % | 75.6 % |
| Random Read       | 16 MBs | 0.1 s | 236.527 MB/s | 0.0 % | 90.8 % |
-----

Tiotest latency results:
-----
| Item                | Average latency | Maximum latency | % >2 sec | % >10 sec |
-----+-----+-----+-----+-----+
| Write               | 0.190 ms | 220.121 ms | 0.00000 | 0.00000 |
| Random Write       | 4.391 ms | 17382.327 ms | 0.02500 | 0.02500 |
| Read               | 0.014 ms | 0.095 ms | 0.00000 | 0.00000 |
| Random Read       | 0.014 ms | 0.064 ms | 0.00000 | 0.00000 |
-----
| Total               | 0.692 ms | 17382.327 ms | 0.00351 | 0.00351 |
-----
  
```

Figure 12: **Tiotest** Results with Write-Caching Disabled

By disabling the write-caching features on the hard disk, the write process time increased over 100% for 40MB of write data. The net task was to disable the Direct Memory Access (DMA) feature on the computer system. The DMA function on the computer can be enabled and disabled by typing the commands shown in Figure 13.

```

Enable: hdparm -d 1 /dev/hda
Disable: hdparm -d 0 /dev/hda
  
```

Figure 13: Enabling and Disabling DMA

A comparison of the effect DMA had on hard disk performance was then compared using **hdparm**, a much simpler hard drive tool, available for nearly all UNIX and Linux based computer systems. Figure 14 displays the performance results with DMA enabled and disabled.

```
Enabled:
debian:/home/ecet325# hdparm -Tt /dev/hda
/dev/hda:
Timing cached reads:   480 MB in  2.01 seconds = 238.81 MB/sec
Timing buffered disk reads:  40 MB in  3.09 seconds = 12.94 MB/sec
Disabled:
debian:/home/ecet325# hdparm -Tt /dev/hda
/dev/hda:
Timing cached reads:   476 MB in  2.00 seconds = 238.00 MB/sec
Timing buffered disk reads:  14 MB in  3.20 seconds =  4.38 MB/sec
```

Figure 14: DMA Performance Impact using **Hdparm**

Analyzing the data, the performance impact of enabling and disabling the DMA feature on the hard disk was very significant, increasing the transfer rate up to 195% when enabled.

The next test conducted analyzed the differences between transferring data sequentially and synchronously. The difference between sequential and synchronous data write is that the processor does not wait for the hard drive to complete the write function before starting the next process thread in the sequential mode transfer so the data is in the exact order as was transferred to the hard disk from the CPU. In a synchronous transfer, the data is transferred to memory and is written with multiple process threads, but causes a significant performance decrease and fragmentation due to multiple concurrent write functions. Figure 15 displays the performance results of a sequential data transfer compared to a synchronous data transfer shown in Figure 16.

```

debian:/home/ecet325# tiotest -S
Tiotest results for 4 concurrent io threads:
-----
| Item                | Time      | Rate      | Usr CPU | Sys CPU |
-----+-----+-----+-----+-----+
| Write               | 40 MBs | 91.2 s | 0.439 MB/s | 0.0 % | 6.4 % |
| Random Write        | 16 MBs | 37.2 s | 0.420 MB/s | 0.1 % | 0.7 % |
| Read                | 40 MBs | 0.2 s | 247.072 MB/s | 0.0 % | 98.8 % |
| Random Read         | 16 MBs | 0.1 s | 236.732 MB/s | 15.2 % | 75.8 % |
-----

Tiotest latency results:
-----
| Item                | Average latency | Maximum latency | % >2 sec | % >10 sec |
-----+-----+-----+-----+-----+
| Write               | 35.615 ms | 592.636 ms | 0.00000 | 0.00000 |
| Random Write        | 36.952 ms | 132.978 ms | 0.00000 | 0.00000 |
| Read                | 0.015 ms | 0.099 ms | 0.00000 | 0.00000 |
| Random Read         | 0.014 ms | 0.062 ms | 0.00000 | 0.00000 |
-----
| Total               | 18.003 ms | 592.636 ms | 0.00000 | 0.00000 |
-----

```

Figure 15: Synchronous Hard Disk Transfer Performance

```

debian:/home/ecet325# tiotest -W
Tiotest results for 4 concurrent io threads:
-----
| Item                | Time      | Rate      | Usr CPU | Sys CPU |
-----+-----+-----+-----+-----+
| Write               | 40 MBs | 10.3 s | 3.886 MB/s | 0.3 % | 5.9 % |
| Random Write        | 16 MBs | 23.9 s | 0.653 MB/s | 0.0 % | 0.6 % |
| Read                | 40 MBs | 0.2 s | 251.629 MB/s | 12.6 % | 81.8 % |
| Random Read         | 16 MBs | 0.1 s | 234.009 MB/s | 0.0 % | 89.9 % |
-----

Tiotest latency results:
-----
| Item                | Average latency | Maximum latency | % >2 sec | % >10 sec |
-----+-----+-----+-----+-----+
| Write               | 0.061 ms | 0.250 ms | 0.00000 | 0.00000 |
| Random Write        | 4.851 ms | 19225.733 ms | 0.02500 | 0.02500 |
| Read                | 0.022 ms | 79.401 ms | 0.00000 | 0.00000 |
| Random Read         | 0.014 ms | 0.062 ms | 0.00000 | 0.00000 |
-----
| Total               | 0.713 ms | 19225.733 ms | 0.00351 | 0.00351 |
-----

```

Figure 16: Sequential Hard Disk Transfer Performance

The next procedure was to analyze the random process usage of the hard disk when using the computer system to check email, browse the internet and use other random computer functions. This data was analyzed over a fifteen minute time period and recorded using the **vmstat** performance analyzer. The **vmstat** tool analyzer the transfer of I/O block

data and reports samples according to user input parameters. The **vmstat** program sample rate (in seconds) is initiated by appending a number parameter to the command. Figure 17 displays an example of **vmstat** sample reporting where I/O is reported with data blocks in (bi) and data blocks out (bo). Aside from strict I/O usage, the **vmstat** program also reports usage of the swap partition.

```
Debian:/home/ecet325# vmstat 10
procs -----memory----- ---swap-- -----io----- --system-- ----cpu----
 r b  swpd  free  buff  cache   si   so   bi   bo   in   cs us sy id wa
 0 0     0 82596 25468 109412    0    0   19   45  108   18  1  1 98  0
```

Figure 17: **Vmstat** I/O Statistical analysis

The report data was then graphed showing blocks in and blocks out versus time for each lab group. Each student executed different functions of the computer during the fifteen minute time period; therefore all of the data between the groups was different. Figure 18 shows a **vmstat** analysis plot of hard disk usage use versus time while running the database update program, **updatedb**.

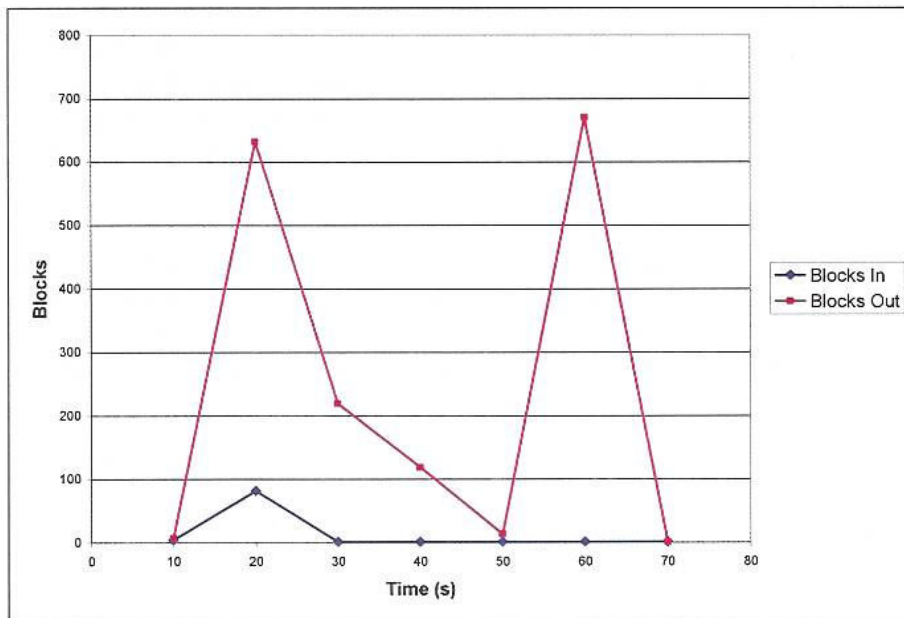


Figure 18: I/O Usage Versus Time

The goal of this portion of the laboratory procedure was to reinforce the random process usage of computer systems and how this is effected by student interaction.

At the end of the lab, the students could quantify the performance metrics of a hard disk by using the formula shown in Figure 19.

$$Speed = \frac{tracks / sectors \cdot RPM \cdot 512}{60s \cdot 1000000} = MB / s \quad [2]$$

Figure 19: Quantifying Hard Disk Speed

The different functions of hard drive operations were easily manipulated and analyzed with a small set of basic tools to modify system parameters and initiating short, simple performance tests instead of merely using a complex data transfer tool analyzing the default system configuration performance. The students understood hard drive writing and reading function differences such as sequential, synchronous, cached write and direct memory access by modifying these system parameters and analyzing the performance results individually. While most single hard drive computer systems utilize the same read/write methods to maximize performance, the students understood how and why each of these design methods are utilized by modifying the parameters and seeing direct results from performance analysis.

Conclusions and Recommendations

While writing the lab procedures and testing the tools used for performance analysis, testing the tools used to gather data for computer system analysis didn't require too much down time waiting for results. Students' knowledge of Linux was gauged to be very low, after gathering responses from questions about computer knowledge and observing response to early computer tests. Over the course of the semester when writing the labs, more incorporation of common usages of Linux and data recorded over longer periods of time gathered by simple analysis tools were implemented. Doing so, the students could more easily interpret how a computer is engineered by how the software used the systems. By the end of the semester, the students were comfortable editing, compiling and running code, using the internet, configuring the network interface, and transferring files to each other for later analysis. Incorporation of experiments early in the semester proved to be a good foundation of computer component understanding. Observation of the component orientation on the motherboard reinforced the computer operation by visualizing the model of process operation. Minimizing waiting time for results of the performance analysis programs kept students involved and attentive to what they were trying to accomplish. Students learned how data types and computer architecture affected the results shown because the students understood the operation of their own programs and connected the meaning of their results to the characteristics of the computer architecture regardless of how tedious the procedure steps were. Each objective was completed using small steps which keep students constantly thinking of what they were trying to accomplish, no matter how mundane or predictable the results were. This coincided with the approach for writing the lab procedures, which were originally planned to include only seven over the course of the semester but turned into ten. The single-week laboratory experiments proved to be more beneficial because the previous week's work wasn't forgotten over the course of the week. The performance tests were more concise and the topic coverage proved more effective when completed during one lab session. This also lends the lab material more flexibility by managing the smaller parts for future semester of ECET 325.

After completing all of the experiments, the students changed their perspective of computer performance to a degree based on their experience in the laboratory. Each of the systems were comparable in terms of raw processing power but had subtle difference

which included stability, compiling code, transferring data, user interfaces, etc. A great majority had never used either Linux or Sun so this experience was eye opening to the world of computer engineering outside of webpage browsing and media. Some of the experiments were very hard to complete using the Windows environment, while others were very challenging with Linux, so their options definitely changed regarding which computer system is best suited for a certain application.

In the future, using the Sarge release of Debian Linux would provide the students with an easier installation process along with a more updated kernel and software packages. This wasn't selected because during test phases the use of the internet was required. A lab of 5+ computers would saturate the network and extend past the allotted lab time.

References

- [1] Benwell, N. (1975). *Benchmarking: Computer Evaluation and Measurement*. Washington, D.C. Hemisphere Publishing Corporation.
- [2] Musumeci, G.D., Loukides, M. (2002). *System Performance Tuning*. Sebastopol, CA: O'Reilly & Associates, Inc.
- [3] Klenke, R.H., Belvins, J.M., Aylor, J.H., Hein, J.J. (2003). A Performance Modeling and Design Environment for Senior Computer Engineering Undergraduates, Proceedings of ASEE/IEEE Frontiers in Education 2003 Conference, F2F-23-F2F-28.
- [4] Nelson, V.P., Theys, M.D., Clements, A. (2003). Computer Architecture and Organization in the Model Computer Engineering Curriculum, Proceedings of ASEE/IEEE Frontiers in Education 2003 Conference, F2F-11 – F2F-16.
- [5] Lewis, B.C., Crews, A.E. (1985). The Evolution of Benchmarking as a Computer Performance Evaluation Technique. *MIS Quarterly* 9(1), 7-16.
- [6] Fleming, P.J., Wallace, J.J. (1986). How Not To Lie With Statistics: The Correct Way to Summarize Benchmark Results. *Communications of the ACM* 29(3), 218-221.
- [7] Hennessy, J.L. & Patterson, D.A. (2003). *Computer Architecture: A Quantitative Approach*. San Francisco, CA: Morgan Kaufmann.
- [8] Jain, R. (1991). *The Art of computer Systems Performance Analysis*. New York, NY: John Wiley & Sons, Inc.
- [9] Amdahl, G.M., Blaauw, G.A., Brooks Jr., F.P. (1964). Architecture of the IBM System 360. *IBM J. Research and Development* 8, 2 (April) 87-101.
- [10] Lilja, D.J. (2000). *Measuring computer Performance: A Practitioner's Guide*.

Cambridge, MA: Cambridge University Press.