Central Processing Unit Performance

William R. Salinas
Computer Science Department
Saint John's University
Collegeville, MN
wrsalinas@csbsju.edu

***Abstract* – High performance computing often requires that algorithms are run on large data sets or that other demanding tasks are performed. If we were to use a CPU from 10 years ago to run the operations we perform today, the run time would be exponentially longer than they already are. In order to keep run times reasonable we must continue advancing our hardware technology. This research will consist of running the quicksort algorithm on two different computers to demonstrate the difference in processing power and performance between two CPUs. Quicksort will be run under different conditions such as sequentially, in parallel, and on data sets of varying sizes to observe the processor's capabilities under light and heavy workloads. The results will consist of run time averages to ensure consistency and they will be graphed for analysis. The contributions of modern technology should make themselves apparent and the CPUs' different strengths and weaknesses can be determined. The hypothesized superior performance of the parallel quicksort algorithm and the CPUs in SJU's 'Beefy' are the focuses of the study along with the contributions given to their performance by technological advances.**

Table of Contents

Table of Figures

I. INTRODUCTION

The field of central processing units is one of rapid evolution, changing demands and innovation. The central processing unit, known as a CPU, is an essential computer component that performs nearly all of the calculations for the machine. It is called upon to perform basic arithmetic functions, gene sequences for the human genome and everything in between. A task such as mapping the genome requires a great deal of processing power and would not have been feasible 25 years ago. The creation of tasks like this is among the driving forces behind the fast evolution of the CPU. In order to maintain a near constant rate of growth, new methods of increasing CPU processing power are being developed and incorporated in processors. The benefits or high performance processing chips are widespread and impact everyone who uses common devices such as computers, cellular phones, or automobiles. Benefits include reducing the waiting time for tasks to complete, making the advancement of human knowledge possible, multitasking, or even playing the latest video game.

II. BACKGROUND

Computing has been a necessary task for humanity far before the invention of the modern computer. Calculations were performed on paper, with an abacus and mechanical devices in the past. The purpose of these devices was to perform computations that were too complex for humans to do themselves or to perform computations faster than could be done in our brains.

1. Early Advances

Early computers required the rewiring of circuits in order to function, however in the 1940s John Von Neumann, J. Presper Eckert and John Mauchly came up with the idea of storing instructions inside the computer [1]. The device that carried out these instructions was called the central processing unit.

Among the first major advances in CPU technology was the introduction of Intel's 4004 microprocessor in 1971 which could be programmed to perform many different calculations [1]. A second important advance was the insertion of cache Random Access Memory (RAM) between the microprocessor and main memory. This addition meant that the microprocessor would be able to execute at improved speed as it would not need to wait as frequently while accessing the slower main memory because its instructions or data could be stored in the cache RAM [1].

2. Performance Growth

Throughout the late 20th century, the CPU had steady and predictable growth. During this time the method used to increase processing power was to decrease transistor size, increase the number of transistors in a CPU and decrease the overall size of the computer chip. The growth in processing power over this time had been quite consistent with Moore's Law which stated that the number of transistors on a microchip would double every two years. David House later adapted Moore's Law to state that computer performance would double every 18 months [2].

The measure of performance for CPUs was generally by their clock speed, such that higher frequency indicated better performance [2]. In the 1990s microprocessor performance grew by approximately 60% each year. In the four years that followed the turn of the century, performance only grew by 40% each year and in 2004 it increased by merely 20% [3].

3. The Problem

As of late 2004, commercially available processors boasted frequencies up to 3.8GHz [4]. At this point a marginal increase in clock rate would have required a substantial increase in energy. It became apparent that processors' clock speeds were reaching a plateau.

This was not due to issues related to transistor size or number but rather power requirements and heat dispersion. Increasing CPUs' frequencies causes them to require more energy and run at hotter temperatures [3]. Additional cooling functions to combat the heat would have been necessary and these functions would have required even more energy to power them. Had we continued using the same method of enhancing performance, it would not be unrealistic for laptop computers to require expensive features such as water cooling systems in order to run at an acceptable temperature [5]. It was time for the computing world to take a different approach towards performance.

4. The Solution

The need to keep up with the expected rate of performance improvement drove CPU manufacturers to introduce one of the most dramatic changes that the microprocessor has seen in its evolutionary history. Prior to 2006, the computing world ran on high frequency single-core CPUs. From the humble beginnings of the 740kHz Intel 4004 processor in 1971 [1] the single-core CPU remained dominant until manufacturing giants AMD and Intel each put forward commercial dual-core processors in 2005 [4].

Multicore processors host two or more processing units, referred to as cores, inside a single integrated circuit. These cores were not as powerful as those found in single-core processors but they generated better overall performance because they could handle more work by operating in parallel [3]. This is a technique known as multiprocessing.

The introduction of multicore processors was met with a positive reception by the computing industry and they were praised for their power-saving nature as two lower frequency cores consumed less energy than a single high frequency core. The IEEE Review noted in September of 2005 that for every 400MHz increase in clock speed, power consumption would rise by 60% but that dual-core chips provided a significant performance boost without the need to run at ruinous clock speeds [2].

5. Challenges

The shortcomings of multicore processors were not ignored, as many were quick to point out that most software was not optimized for dual-core CPUs so there would be little if any improvement running applications on dual-core units over single-core ones. It can be difficult for programmers to write applications that scale across multiple cores and they synchronize correctly while ensuring some calculations were executed in the correct order [6]. Despite this drawback, multicore processors were still seen as the future of computing. [3] John Williams, a technical director for AMD, demonstrated that sentiment with his 2005 statement that “Multiple cores are the new megahertz. Multicore will be the transition from brute-force performance to architectural elegance.”

Hardware is not the only aspect of high performance computing that is evolving towards parallelism; software is beginning to be written to be compatible with the parallel nature of current CPUs and graphics processing units (GPUs). Writing software that can be run on multiple threads is a challenge currently facing the computing industry [3].

III. TECHNICAL ANALYSIS

1. Architecture

The physical configuration of functional units within a microprocessor varies from one design to another; however the collection of parts is quite similar across the spectrum of microprocessors. Figure 1 shows a generic microprocessor with the different units labeled.

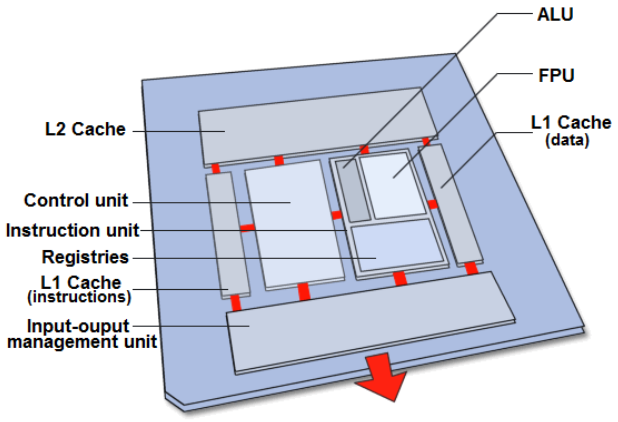

Figure 1. Diagram of a microprocessor’s components.

There are several generic terms that should be understood before individual devices are explained. The first of these is a *register*, which is very small and very fast memory housed within the CPU. *Cache* is local memory that serves to reduce the waiting time for data stored in RAM also known as main memory.

The *control unit* is the device that directs the flow of data within the processor and it sends instructions to the execution unit. There are several other devices housed within the control unit. Among these is the *sequencer* which may also be called the monitor and logic unit. It synchronizes the execution of instructions with the clock rate and sends control signals which instruct the components involved in executing an instruction. Additionally, the *program counter* (or ordinal counter) is a register stored within the control unit and it holds the registry address of the instruction currently being performed. The *instruction register* is a small amount of memory in the control unit and contains the next instruction to be executed [7].

The *execution unit* is where the calculations take place within the CPU. This element is also known as the processing unit and it takes directions from the instruction unit. It is composed of the *arithmetic and logic unit* (ALU), floating-point unit (FPU), status register, and accumulator register. The ALU carries out basic mathematic calculations and logic functions. When the CPU needs to add, subtract, multiply, divide, or perform AND, OR, NOT, and XOR functions the ALU is the device that completes the task. *The floating-point unit* is similar to the ALU in that it carries out mathematic operations, however what distinguishes the FPU is its ability to perform these on floating point numbers, which are too complex for the ALU to handle. The *status register* is memory that holds system status indicators, or flags, for cases where a carryover in needed, the result is zero, and when there is overflow. The *accumulator register* is where the result of mathematic or logical operations is stored [7]. Both of these registers are contained in the box labeled registries in Figure 1.

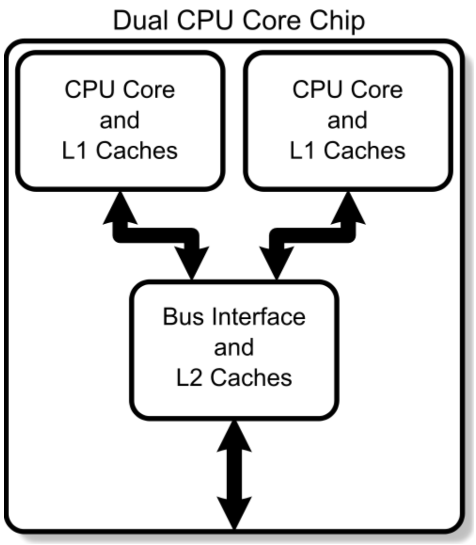

Figure 2. Dual core CPU architecture.

The *bus management unit* manages the flow of incoming and outgoing information for the CPU. It is labeled as the input-output manager in Figure 1 and it interfaces with main memory. The last component to be discussed is cache memory. *Level one (L1) cache* is directly integrated into the processor whereas *level two (L2) cache* is separate from the processor but still located within the microprocessor. In multicore processors the L2 cache is often shared between two processors, as seen in Figure 2, which can lead to contention for resources. L2 cache serves as an intermediary between the processor and main memory. L1 cache is slower and larger than a register but it is smaller and faster than L2 cache which in turn is smaller and faster than main memory [7].

2. CPU Operation

A CPU performs four primary steps over the course of its operation: fetch, decode, execute, and writeback. A sequence of these instructions is known as a program. These instructions are stored in the program memory and accessed by the fetch command. The instruction retrieved by the fetch determines what the processor does next. This instruction is then decoded into smaller pieces that will have meaning to certain components of the CPU. Once decoded, the CPU will check for operands that will tell it what to do in the execution stage. During execution the arithmetic or logical function specified by the instruction is performed. The final step is the writeback in which the result of the execution stage is stored into memory [7].

3. Multicore

A multicore processor is one in which multiple complete CPUs are placed onto a single integrated circuit die with significant parts of their memory hierarchy. Multicore chips do not run at as high of a clock speed as a single core chip yet their overall performance is increased due to the augmented number of processing cores [3]. This gives the microprocessor true multitasking ability and the multicore chips require less power and cooling because their cores operate at lower frequencies.

4. Performance

This section explains how CPU performance has been able to growth throughout their period of dramatic change. Figure 3 illustrates CPU trends through the mass conversion to multicore processors. The green line represents the number of transistors in CPUs, which is increasing at a steady rate in accordance with Moore's Law.

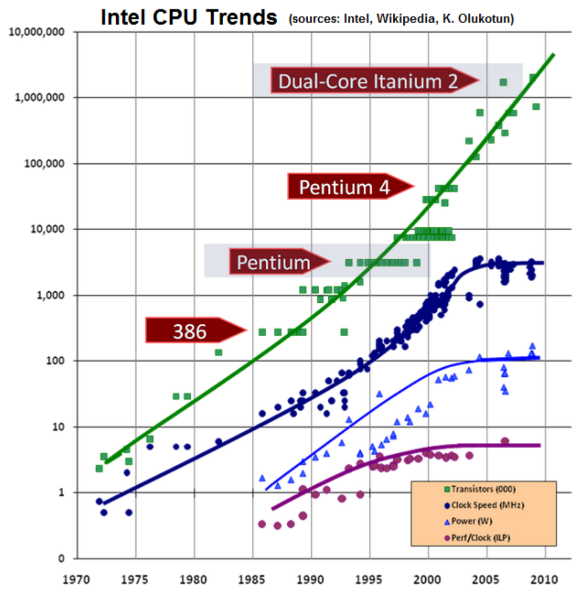

Figure 3. CPU trends over time.

The blue line shows the clock speeds over time. Clock speeds are still approximately what they were in 2005. Instead of increasing the speed for greater performance, manufacturers have taken to adding more cores. This also accounts for why the number of transistors is growing consistently. Instead of crowding more transistors into each core, the number of transistors can be doubled by adding a second core.

The lighter blue line is the power required to operate the CPU which appears to have stabilized along with the clock speeds. The performance per clock cycle is the purple line which also evened out with clock speeds. Individuals who desire a greater clock speed have an option to overclock their CPUs which involves modifying the hardware settings so the clock speed is higher than what the manufacturers sold it at.

1.) Multithreading

Multiprocessing was not the only technique utilized to gain a performance boost. Multithreading technology allows a processor to keep multiple hardware threads on the chip and ready for execution. The threads will share resources and the idea behind this is to maximize the overall throughput without duplicating the existing resources.

Most manufacturers design these chips to issue instructions from several threads each cycle rather than switching between the threads on a core. This process is known as simultaneous multithreading and will cause an operating system (OS) to identify twice as many physical processors as actually exist [7]. Like using multiple cores, multithreading is also prone to competition for cache memory and time executing on the processor. Both multiprocessing and multithreading are forms of parallelism.

2.) Clock Rate

Clock rate was discussed earlier as a common measure of a CPU's processing power. This is because the clock rate is the speed at which a CPU executes instructions measured by clock cycles per second which gives a value in hertz. Thus, the higher clock speed indicates better performance because a CPU would be able to execute more instructions per second.

In the past, clock speeds were able to increase rapidly, as predicted by Moore's Law, due to the inclusion of more transistors in processors as they were becoming increasingly smaller [3]. With the introduction of multicore CPUs the clock speeds began to drop however the number of transistors in CPUs continued to rise.

Many CPUs today have the ability to temporarily increase their clock speed during computationally intensive tasks [8]. This can be thought of as a type of 'selective overclocking.' It is not counter-productive towards finding alternative techniques for performance enhancement because rather than increasing the clock speed outright, this method normally operates at a lower and more efficient frequency and only increases it on an as-needed basis.

3.) Drawbacks

The multicore method is not without its weakness. The ability for a single program to take advantage of multiple processors is based on whether or not it was written optimized for parallelism [3]. As long as the serial portion of a program is kept small then the larger number of simple cores is advantageous, however if a significant portion of it is serial then fewer and more complex cores would be desired. A single core processor uses brute force, which many multicore processors do not have, to execute serial programs. Another drawback is that multiple cores may be competing for the same resources such as L2 cache [9].

IV. DEMONSTRATION

1. Overview

The demonstration we developed aims to show that with parallelization CPUs experience a performance boost over serialized programs. The quicksort algorithm was chosen for the testing portion of the prototype. It was coded in C++ with the addition of OpenMP for the parallel version of the program. Each program was run with a randomly generated input that was stored into arrays of sizes 2^5, 2^{10},2^{15}, 2^{20}, and 2^{21} which was the largest data set that could be used without causing a core dump.

The testing was conducted on a computer from St. John's University's Linux lab using a dual core 2.40Ghz CPU and then it was repeated on the institution's supercomputer 'Beefy' which has two quad core 2.67Ghz processors. The data collected were the runtimes taken to sort the input list and an average of five outputs was used for each program at each size and on each machine. Taking an average of five outputs allowed us to ensure consistency in the results and integrity of the data.

Our predictions prior to the experiment were that Beefy would complete the algorithm faster than the Linux lab computer and that the parallel version of the algorithm would finish more quickly than the sequential version.

2. Quicksort

Quicksort follows a divide-and-conquer method which includes three phases. The first phase splits the problem into smaller sub-problems of relatively equal size. The second phase solves the sub-problems and the last phase merges the solutions from phase two in order to achieve the solution to the original problem.

The quicksort algorithm has best case runtime of $O(n \log n)$ which is impressive as this is also the average runtime. The worst case runtime completes in $O(n^2)$ which is quite poor but also very uncommon. Quicksort runs well on random inputs and poorly on nearly sorted inputs [10].

The sorting method required that a pivot value be chosen from the input. All elements that were smaller than the value of the pivot were moved to its left side and all larger elements were moved to its right. Pivot selection was important to ensure that the two resulting sub-arrays were of approximately equal size. The technique used to choose a pivot location was to consider the first, middle, and last elements from the input and use the median of the three values as the pivot.

Once the pivot was selected, it was moved to the far right end of the array for the partitioning portion of the process. The algorithm worked from the left end towards the right. Two position variables that we will call *i* and *j* were initialized with their values as the position of the leftmost element. Variable *i* was incremented until it has traversed the entire array while *j* served as a position marker. If *i* encountered an element whose value was less than the pivot's, it exchanged that element with the element at position *j* and *j* was incremented. Once the array had been traversed by *i*, the pivot was exchanged with the element at position *j* and all of the elements to the left of the pivot were less than it and all the elements to the right were greater.

Then quicksort was applied recursively on the sub-array from the leftmost element to the pivot index -1 and on sub-array from pivot index+1 to the rightmost element. This was repeated while the rightmost element of a sub-array was greater than the leftmost of the sub-array, or while the sub-array was greater than one element.

Code for parallelization, using OpenMP, was cast around quicksort's recursive calls to itself because once the elements were sorted around the pivot the two sides remained independent of one another until the algorithm's completion.

3. Results

1.) Beefy vs. Lab PC

The hypothesis that Beefy would complete the quicksort algorithm faster than the lab computer was shown to be correct. At input sizes of 2^5 and 2^{10} there was no time difference between the machines when rounded to the nearest hundredth of a second, but at an input size of 2^{20} Beefy ran over a full second faster than the lab machine. For the sequential and parallel quicksort, Beefy's speedups over the lab computer were approximately 1.29 and 1.41 respectively. On the largest data set of 2^{21} elements, Beefy finished over 4 seconds faster than the lab machine both in parallel and sequential tests which resulted in approximate speedups of 1.29 for the sequential quicksort again and 1.36 for the parallel algorithm. Figure 4 shows that Beefy even completed the algorithm faster sequentially than the lab computer could in parallel.

Table 1. Demonstration runtimes in seconds and rounded to the nearest hundredth.

Machine & Algorithm	Input Size		
	2^15	2^20	2^21
Linux Lab Sequential	0.01	5.21	20.47
Beefy Sequential	0.01	4.04	15.85
Linux Lab Parallel	0.02	5.19	16.45
Beefy Parallel	0.01	3.69	12.06

Beefy had a clear advantage in both tests due to its higher clock speed which allowed it to use more brute force in sorting the sequential algorithm. It held an advantage in the parallel test for the same reason in addition to having six more cores at its disposal than the lab machine.

2.) Sequential vs. Parallel

It was predicted that the parallel tests would typically finish sooner than the sequential ones. At lower input sizes both algorithms completed too quickly for any differentiation in the runtimes. The first difference noted was at input size of 2^{15} when all the machine and algorithm combinations finished in 0.01 seconds except for the parallel quicksort on the lab computer. It was surprising that the parallel algorithm ran slower than the sequential one at this size, however this occurrence did not repeat. At the input size of 2^{20} the parallel quicksort on the lab computer averaged only 0.02 seconds faster than the sequential algorithm. We had expected to see a greater difference between the algorithms at this point. It is possible that other tasks running at the time had interfered with the tests.

During the testing for the largest input we saw the result that had been expected. The lab computer's parallel algorithm competed over four seconds faster than its sequential one at that size for a speedup of approximately 1.24 while Beefy's finished sorting over three seconds faster than its corresponding sequential algorithm with a speedup of about 1.31.

This outcome was predictable because the workload was distributed amongst the computers' cores so that portions of the code were executed simultaneously. Figure 4 shows that both parallel implementations' slopes are lower than for their sequential counterparts between the two largest data sets. This indicates that with even larger data sets the speedup for the parallel algorithm would continue to increase.

A limitation was encountered due to the implementation of quicksort that was used. This implementation used a single array that was constantly being modified and passed rather than creating sub-arrays with each iteration of quicksort. The effect this had on the demonstration was that, at most, two processing cores could be utilized at one time. The lab computer was unaffected by this limitation, but for Beefy's speedup was limited to two despite having an ideal speedup of eight.

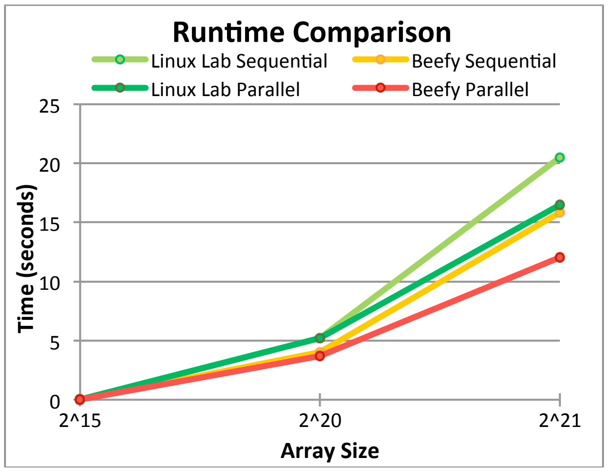

Figure 4. Graph of runtimes by input size and machine/algorithm.

V. STATE-OF-THE-FIELD

The field of CPU performance is still in the early stages of one of the most dramatic changes that it has undergone in its short history [3]. Transitioning to multicore processors allowed for the continued rapid growth of microprocessor performance through the use of parallelization. Increasing processing power through the addition of CPU cores is a method that scales well [11] and provides abilities such as true multitasking. The demand for CPUs with high clock rates has been addressed both in the natural evolution of multicore processors and through short-term boost to the clock rate built into many new processors.

There is also a current focus on writing efficient software that utilizes CPUs' multiprocessing ability and improving parallel programming languages [12]. Several major Internet browsers already include this feature. The desire for efficiency goes beyond just software.

In addition to taking advantage of multiple cores there is an interest in utilizing a GPU's processing power to perform some of the calculations for a system's CPU [13]. Processor performance is at a comfortable point in this post-transitional period where the path of parallelization seems the obvious route but there is still room for experimentation.

VI. FUTURE TRENDS

We predict that current trend of placing additional cores in a microprocessor will continue over many years while CPUs maintain a relatively stable clock speed as they have over the better part of the last decade [14]. Secondly, we think that there will be an increase in the usage of graphics processing units (GPUs) assisting CPUs with computations [15] and interest in quantum computing will begin to rise as advancements are made in the area. Additionally, we believe that the field of supercomputing will begin focusing on more sophisticated and efficient designs rather than FLOPS (floating point operations per second) [16].

1. Parallelism

The future of CPUs development appears to have committed to the path of parallelism. Manufacturers are adding additional cores to CPUs such that quad-core processors have started to become common.

There is evidence that the number of cores contained within processors will continue to increase with time. As early as 2005 Intel was working on creating 16-core CPUs [3]. Last year, at the Supercomputing 2010 conference, Intel researcher Timothy Mattson stated that their 48-core Single Chip Cloud Computer (SCC) processor could be scaled over 20 times and produce a 1,000-core processor. [17] Mattson also states that there is no theoretical limit to the number of cores that can be used.

The clock speeds of earlier generations of multicore CPUs were noticeably slower than their single-core counterparts. Since then, multicore chips have caught up in clock speed and experienced the same plateau. We predict that the dwindling number of single-core CPUs on the market will disappear [5] over the next three to five years because multicore chips can now match their clock speeds.

2. GPGPU

A second trend is the combination of CPUs and GPUs running together in parallel. The director of the National Center for Supercomputing Applications, Thom Dunning, believes that GPUs are the future of supercomputing [15]. Using the immense computational power of a GPU to perform calculations for the CPU often results in a significant speedup.

The results of this method are easily seen in the capabilities of China's Tianhe-1A which utilized this strategy in order to become the top ranked supercomputer in the world in terms of floating point operations per second [15].

We expect that many others will follow this route in order to supplement CPU performance and that almost all supercomputers built in the next few years will use this approach.

3. Quantum Computing

Quantum computing is another direction that is continuing to be explored however the field is still in its infancy and is error-prone. The potential to be the fastest computational method in the world keeps researchers interested in the idea of quantum computing [18].

While no quantum computers yet exist, advances have been made such that an architecture called RezQu has been developed for a quantum processor. The architecture is highly scalable and University of California researcher Erik Lucero feels that his team is on the verge of actually having a quantum processor [19].

4. Efficiency

The computing world is also experiencing a shift towards parallelism in its software aspect as well as hardware in an effort for greater efficiency in addition to performance.

The use of clock speed boosting is an effort to provide enough processing power while keeping the overall power consumption down thus making processors more energy efficient.

Regarding software, the Multicore Association is seeking to establish standards to assist the coding of software for multicore chips [6]. Their goal is to reduce the challenge faced by programmers attempting to write applications that scale across multiple processing cores and synchronize correctly afterwards. The creation of such standards would be beneficial to the computing community as a whole, assuming they are implemented well. We anticipate their completion will facilitate the creation of a wave of effective parallel applications which coincides with the overall direction that software coding is moving towards.

In the realm of supercomputing, it is possible that the computers' construction will become more specific towards the machine's intended purpose with a greater focus on efficiency and clever software designs rather than the number of petaFLOPS it is capable of [16].

Some members of the computing community are comparing the petaFLOPS measurement of supercomputers to only considering the top speed of an automobile. They put forth the argument that a Ferrari's speed advantage over a Volvo station wagon wouldn't matter if you needed to take two children to soccer practice. [16]

VII. CONCLUSION

The multicore processor was embraced by the computing world and the ideals of parallelism it brought with it have become the new standards within the industry. The shift towards parallelization is leading towards a new era of high performance computing that is based on distributed workloads and efficient coding rather than brute force. The assortment of techniques for improving CPU performance and implementing parallelism pave the way for future processors with greater computational power and efficiency than those used today.

The results of our demonstration indicate that parallel programming is advantageous opposed to sequential. The runtimes in parallel for small data sets were comparable to sequential times on both machines tested, but on large data sets the parallel runtimes were clearly faster. The demonstration also showed that high clock speeds on CPUs are still beneficial as Beefy's average sequential runtime on the data set of size 2^{21} was 0.60 seconds quicker than that of the lab computer's parallel time at the same size. The runtime difference due to the higher clock speed of Beefy and its greater potential for running parallel processes are evidence of the impact that technological advances have on the field of performance computing.

REFERENCES

[1] Davis, Roy. "Microprocessor History (Part 1, The Basics)." Tech-tips. 22 Nov. 2005. <http://www.geeks.com/techtips/2005/techtips-NOV22-05.htm>.

[2] Schauer, Brian. "A Brief History of Microprocessors." ProQuest. Sept. 2008. <http://www.csa.com/discoveryguides/multicore/review2.php>.

[3] Geer, David. "Industry Trends: Chip Makers Turn to Multicore Processors," Computer, 38.5 (2005) pp. 11-13.

[4] Polsson, Ken. “Chronology of Microprocessors.” Processor Time Line. 1 Jan. 2011. <http://processortimeline.info/proc2004.htm>.

[5] Field, David. “The Impact of Multi-core Processors on Application Performance.” HPCWire. 21 Mar. 2008. <http://www.hpcwire.com/offthewire/17910484.html>.

[6] Shah, Agam. “Multicore Coding Standards Aim to Ease Programming.” ACM News. 30 March 2011. 12 April 2011. <http://cacm.acm.org/news/107011-multicore-coding-standards-aim-to-ease-programming/fulltext>.

[7] Shah, Hemant, “Processor.” Hardware Information. Blogspot. 14 April 2009. 27 March 2011 <http://hardwaresangli.blogspot.com/2009/04/processor.html>.

[8] Glaskowsky, Peter. “Explaining Intel’s Turbo Boost Technology.” CNet News. 28 Sep. 2009. <http://news.cnet.com/8301-13512_3-10362882-23.html>.

[9] Angela Sodan, Jacob Machina, Arash Deshmeh, Kevin Macnaughton, Bryan Esbaugh, "Parallelism via Multithreaded and Multicore CPUs,"*Computer*, 20 Nov. 2009. IEEE computer Society Digital Library. IEEE Computer Society, <http://doi.ieeecomputersociety.org/10.1109/MC.2009.377>.

[10] Joseph JaJa, "A Perspective on Quicksort," Computing in Science and Engineering, vol. 2, no. 1, pp. 43-49, Jan./Feb. 2000, doi:10.1109/5992.814657

[11] Clark, Jack. “Intel: Why a 1,000-Core Chip Is Feasible.” ACM News. 29 Dec 2010. 12 April 2011. <http://cacm.acm.org/news/103399-intel-why-a-1000-core-chip-is-feasible/fulltext>.

[12] Asanovic, K., Bodik, R., Demmel, J., Keaveny, T., Keutzer, K., Kubiatowicz, J., Morgan, N., Patterson, D., Sen, K., Wawrzynek, J., Wessel, D., Yelick, K. “A View of the Parallel Computing Landscape.” Communications of the ACM. 52.10 (2009) pp.56-67.

[13] Bauer, D., McMahon, M., Page, E. “An approach for the effective utilization of GP-GPUs in parallel combined simulation.” Winter Simulation Conference. (2008) pp. 695-702.

[14] Mark D. Hill, Michael R. Marty, "Amdahl's Law in the Multicore Era," Computer, vol. 41, no. 7, pp. 33-38, July 2008, doi:10.1109/MC.2008.209

[15] Crothers, Brooke. “NCSA Director: GPU Is Future of Supercomputing.” ACM News. 2 November. 2010. 12 April 2011. <http://cacm.acm.org/opinion/interviews/101045-ncsa-director-gpu-is-future-of-supercomputing/fulltext>.

[16] Young, Jeffery. “Supercomputers Let Up on Speed.” ACM News. 8 April 2011. 12 April 2011. <http://cacm.acm.org/news/107282-supercomputers-let-up-on-speed/fulltext>.

[17] Clark, Jack. “Experimental Intel chip could scale to 1,000 cores.” ZDNet UK. 22 Nov 2010. 12 April 2011. <http://www.zdnet.co.uk/blogs/mapping-babel-10017967/experimental-intel-chip-could-scale-to-1000-cores-10021129/>.

[18] John Preskill. “Proceedings: Mathematical, Physical and Engineering Sciences.” Vol. 454, No. 1969, Quantum Coherence and Decoherence. 8 Jan. 1998. pp. 469-486. <http://www.jstor.org/stable/53176>.

[19] Palmer, Jason. “Quantum computing device hints at powerful future.” BBC News. 22 March 2011. 12 April 2011. <http://www.bbc.co.uk/news/science-environment-12811199>.

A. Reflection

Over the course of this research project I found it necessary to utilize many of the skills that I developed over the course of my education.

The first of these is my ability to do research and find the types of information that I am looking for. This ability has been particularly useful in the communications classes that I had been taking for my minor. Each of those classes required several research papers, however none of them were of the magnitude of this project. In my research for the communications classes I learned how to utilize the school's library's search functions on their website. It was quite rare that I would need to use them as a computer science student so it was fortunate that another aspect of my education helped me to further develop that skill.

In addition to research skills, the communications papers also helped me keep my writings skills from deteriorating. A typical computer science class requires little more writing than lab reports so again I feel that it was beneficial for me to have taken courses that allow for more creative writing styles.

Other relevant coursework would primarily include the algorithms class I took last year in which I learned about sorting algorithms and I first used C++ which I wrote in for the demonstration portion of my project. I also learned a small amount about coding in parallel with OpenMP during the algorithms course. I again applied this knowledge to my demonstration.

Additional experiences that helped me work on this project include the programming classes I took as a freshman and sophomore because they helped prepare me to spend hours in the lab attempting to get a program to execute correctly. I think that my problem-solving ability has gotten better during my education as well. Reflecting on the time I spent coding my demonstration, I realize that I have become much more perceptive as to what is actually happening when my coding is executing correctly or incorrectly and when it is the latter it has become much easier to fix than it would have been one or two years ago.

The writing of this draft also required me to call upon the coursework that was completed over the course of this last semester. I used the previous assignments of the course to help bring the project together as a whole. The sources of each paper were revisited during the writing process with the exceptions of the papers from IEEE which would not let me view them. The article reviews that were due at every class period served indirectly as a method of promoting research as I found on multiple occasions while looking for an article to review I found one that was potentially helpful towards some aspect of the project.

Lastly, this project required me to combine the research and writing skills I developed outside my major with the coding and problem solving skills that I learned within it. The project has broadened my understanding of a device that I had used every day and taken for granted.