Journal of Information Systems Education, Vol. 16(2)

Demonstration of Parallel Processing Computing: A Scalable Linux Personal Computer Cluster Approach

Jay Rine
Virginia Franke Kleist
Brian McConahey
Department of Management
West Virginia University
Morgantown, WV 26506
jayrine@msn.com virginia.kleist@mail.wvu.edu b_mcconah@hotmail.com

ABSTRACT

In this paper, we describe an innovative approach to teaching parallel computing concepts in a lab setting using a master and slave cluster of Pentium PCs strapped together using Scyld Corporation's Beowulf software, applying a straightforward, custom written prime number test analytical program. This classroom based parallel processing application serves to illustrate three useful topics for the advanced decision sciences student: 1) the Linux operating system and programming concepts, 2) Beowulf cluster computing, and 3) the importance of Linux based parallel processing using low level PCs to solve complex computing applications. It is likely that the results described here can be replicated at low cost in most academic computing environments, yielding enhanced student understanding and ownership of previously less accessible information systems programming concepts. Further, learning the described cluster computing technology tool may build improved problem solving skills for students faced with large, non-trivial computational requirements. Finally, we believe that the demonstrated approach is inherently scalable, thus, deploying this method in larger and larger clusters would be additionally instructive.

Keywords: Parallel Processing Computing, Linux Platform, Scalable Personal Computer Cluster

1. INTRODUCTION

The processing power of personal computers has grown exponentially over the years, yet the personal computer can still complete only one instruction at a time. If given a sufficiently large number of tasks to complete, such as frequently encountered during financial simulation processing, a considerable amount of computing power and time are required to finish the assigned work. In a parallel processing environment, such tasks can be delegated to various processors and worked on simultaneously, greatly improving overall run time to completion. Historically, parallel processing has been the realm of mainframe computers, an expensive and often financially out-of-reach solution that requires specialized personnel and custom written software. Few students have access to these facilities, and fewer still obtain any hands on experience in the parallel processing computing domain. This paper investigates and demonstrates results from an innovative, lab based approach used for teaching the concept of clustering in simulation problems, in which everyday personal computers are networked together using a Linux platform, and used like a single parallel processing machine for solving massive computer processing tasks. We suggest that such a hands-on approach to teaching parallel processing computing will yield effective graduate student learning in an area that is rapidly changing, prohibitively expensive, yet of critical technological importance in the near future.

Since the introduction of the IBM PC in 1981 (Bellis, 1999), personal computers (PCs) have become increasingly powerful and well-networked with each other. Many of these machines, however, are utilized for mostly narrow scope computing tasks such as e-mail, spreadsheet and word processing applications (Taschek, 2003). While many organizations have a need for harnessing large amounts of computing power, they have historically elected to utilize mainframe computers or in more recent times implemented banks of the latest equipped PCs. However, both of these solutions are quite expensive and prohibitive for many organizational computing tasks.

When confronted with a requirement for large computing processing power in 1994, a contractor for NASA established a "Beowulf project" with the goal of harnessing the computational power of a group of personal computers (Taschek, 2003). As a result of this project, the first Beowulf cluster was created, in which sixteen computers running under the Linux operating system were networked together (Gropp et al., 2003). This cluster was constructed such that its collective processing power could be harnessed

as one computer and utilized by programs able to take advantage of such parallel processing.

2. GRID COMPUTING

Conceptually similar to Beowulf clusters, grid computing has emerged as a more commercially viable method of utilizing the vast amounts of idle computing power that resides within many corporations. Although both methods are based upon an underlying Linux operating system, grid computing is mostly implemented by commercial vendors such as IBM (Middlemiss, 2004), while Beowulf clusters tend to be more of a build–it–yourself experience that is advocated mainly by computer hobbyists and academic institutions.

Unlike Beowulf clusters, which tend to be intentionally isolated from outside networks, grid computing uses external contact to its advantage; its component computers need not be in the same physical location (Lonsdale). The "SETI at Home" project is an example of grid computing (Taschek, 2003) which is currently tapping the idle computing power of hundreds of thousands of computers throughout world (SETI@home). A more economically oriented, albeit smaller scale, use of grid computing was implemented at Charles Schwab. This firm used the aggregated computational power of several machines strapped together in a successful effort to reduce the run time of certain financial scenarios by an order of magnitude (Middlemiss, 2004).

2.1 Selecting Beowulf Software

A Beowulf cluster can be constructed using one of any number of software options, although all are based upon an underlying Linux distribution. For this case, the "Basic Edition" of Scyld Corporation's Beowulf software was purchased through the Linux Central e–commerce web site for $7.80 (Linux Central). Its selection was driven by the low price of the software, the simplicity of how the cluster was to be constructed and the positive comments found on the Internet attesting to both its ease of installation and use. In fact, the subsequent installation required approximately thirty minutes and consumed slightly less than 700 megabytes of hard drive space on the master node, while the slave node(s) need not have a hard drive. It should be noted here that the "Basic Edition" contains neither documentation nor user support. In addition, the version available through Linux Central is the outdated Release 27BZ–8 with a copyright of 2001; the Scyld Corporation has subsequently issued Releases 28 and 29 of its Beowulf software.

2.2 Building a Beowulf Cluster

For this deployment of parallel processing using a Linux platform, a total of seven computers were employed in the construction of the cluster with one designated as master and six as slaves. Table 1 summarizes the base specifications of each computer.

As can be seen in the above table, the cluster is not made up of homogenous computers, and the machines used for this project were abandoned and relegated to storage. The lack of consistent computing power across the cluster lead to some interesting performance characteristics during the benchmarking phase of the project and will be addressed later in this paper.

Table 1: Base Specifications of Computers

Node	Processor	RAM
Master	Intel Pentium II 450 MHz	192 MB
Slave	Intel Pentium III 450 MHz	256 MB
Slave	Intel Pentium III 450 MHz	256 MB
Slave	Intel Pentium III 450 MHz	256 MB
Slave	Intel Pentium III 450 MHz	256 MB
Slave	Intel Pentium III 733 MHz	256 MB
Slave	Intel Pentium 200 MHz	64 MB

During the construction of the cluster, a concern became evident. In this lab application, there were several computers operating in a confined space, and the room temperature began to rise quickly and noticeably. A larger cluster composed of dozens of computers would generate a considerable amount of heat and would likely need to be housed in a location with sufficient air conditioning to prevent overheating, which is discussed in greater detail by Brown (pp. 15–16). Since the master node is likely to have access to other parts of the network, it is also suggested that this location be secured from unauthorized personnel.

3. WRITING A PROGRAM TO RUN IN PARALLEL

Once construction of the master slave cluster was complete, it was necessary to design and code a parallel program that could be used to test the computational speed of the cluster. Since a Message Passing Interface (MPI) had been designed to parallelize programs within the C programming language, C was selected. While any computationally intensive program would suffice for testing purposes, prime numbers had been listed as a prospective target for parallel processing (Blaise, 2001).

A simplistic method for determining if a test value is prime involves an iterative test using modular arithmetic. A list of prospective factors is generated, using each whole number greater than or equal to two but less than the square root of the test value. Each of these prospective factors is divided into the test value. If the remainder is zero, then a factor has been found and the test value cannot be prime. If the remainder is non–zero for all prospective factors, then the test value must be prime. The designed algorithm was purposely inelegant, requiring brute force to determine if a number is prime. For example, since all even numbers are divisible by two, the only even prime number is two; as a result, there is no need to test any even number greater than two. However, it was not the purpose of this program to efficiently calculate prime numbers. The objective of the program was to provide a facility with which the cluster could be measured for computational speed, and to demonstrate to the student the functionality of cluster computing in a lab setting.

3.1 Sourcecode and Subroutines

Although the program can be run across multiple nodes, the program need only exist on the master node. The program

was directly modeled after a sample program in Gropp (pp. 215–218) and contains the following three subroutines:

1. Main – initializes the MPI structure and directs the master and slave nodes to execute their respective subroutines, based upon their MPI–defined node rank.
2. Master – choreographs the data being sent to the various slave nodes, as well as collates the subsequent responses; determines when all prospective prime numbers have been evaluated, at which point termination signals are sent to the slave nodes and it.
3. Slave – based upon data received from the main subroutine and master node, calculates if a series of values are prime or non–prime; subsequently reports its results back to the master node. (NOTE: The source code is available upon request from the authors).

The program was also written to require that two command line arguments be provided at run time. The first argument defines the upper limit that will be tested as a prospective prime number. The second argument determines how many values will be assigned to each slave node for testing during each iteration. By manipulating these two arguments, a range of run times can be generated. These data can then be analyzed with respect to computation and communication time over more than one slave node, if the instructor so chooses.

To assist in the analysis of computation and communication time, the MPI timer functionality was incorporated into the "Main" subroutine of the program. In particular, the time is noted immediately before the "Master" subroutine is called, as well as immediately after the "Master" subroutine is completed. Locating the timing elements around the "Master" subroutine is most appropriate for this purpose, as the "Master" subroutine is responsible for initiating the calculation processes on the slave nodes and does not terminate until all prospective prime numbers have been evaluated. However, it should be noted that there is some MPI housekeeping that is required prior to the "Master" subroutine and after the "Master" subroutine's completion; this time is not captured.

It should be noted that in a standalone (non–cluster) computing environment, debugging a program may be difficult and time consuming, but is a relatively straightforward exercise. However, a program being written in a parallel processing environment presents a much more difficult debugging task. In particular, it becomes significantly more difficult to isolate the nature and source of the underlying errors, as the algorithm is no longer the sole source of potential problems; instead, the problem may be a result of inappropriately programmed message handling by either the master node or the slave nodes, or even an inability of the various nodes to physically communicate with each other due to network difficulties.

3.2 Program Efficiency

As stated previously, the program was not written with efficiency as its primary goal. This is confirmed through an analysis of program run times using a single slave, in which the communication aspect was minimized by setting the two aforementioned command line arguments equal; this forced the single slave node to evaluate all prospective prime numbers using just one communication cycle. As shown in Figure 1, the run times are exponential with respect to the quantity of values being evaluated, when run in the single slave arrangement.

In particular, a mere 1.3 seconds were needed to evaluate the prime numbers up to and including 100,000. Increasing the threshold to 1,000,000 required a total of 25 seconds, which demonstrates that the program has already begun to exhibit exponential scaling characteristics; ten times as many numbers were evaluated, but nearly twenty times as much run time was needed. Finally, 588 seconds were required to evaluate all prospective prime numbers through 10,000,000; once again, ten times as many numbers were evaluated, but over twenty times as much run time was needed.

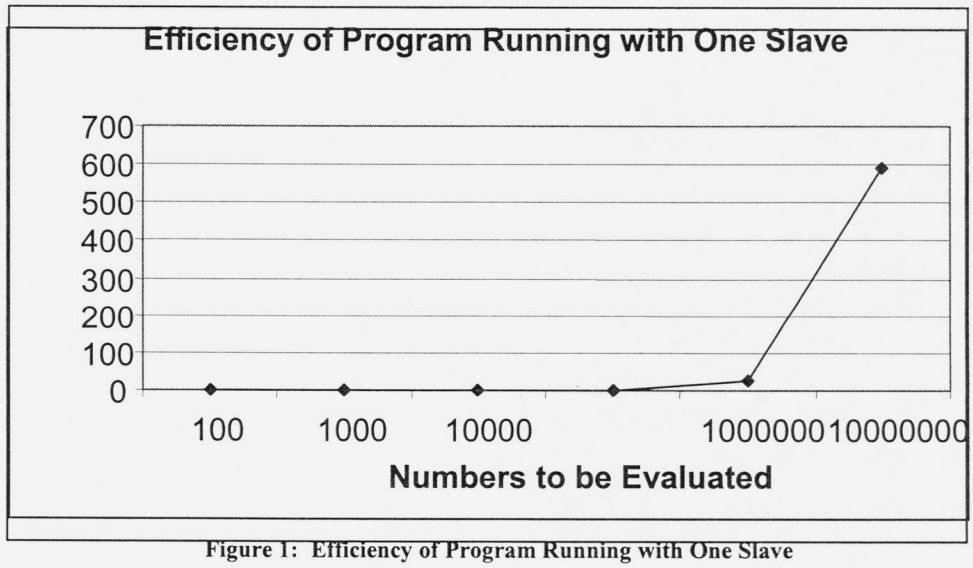


Figure 1: Efficiency of Program Running with One Slave

3.3 Impact Of Communication On Run Time

We test the impact of intra–cluster communications on program run time in the single node arrangement. During each communication cycle of the prime number program, the master node sends one integer to the slave nodes, representing the starting value for the next iteration of prime number calculations. Upon completing its calculations, the slave nodes reply to the master node with seven integers, representing the number of discovered prime numbers less than or equal to 10; less than or equal to 100; less than or equal to 1,000; less than or equal to 10,000; less than or equal to 100,000; less than or equal to 1,000,000; and less than or equal to 10,000,000. The master node collates the summarized data from these replies until all requested numbers, as defined by the first argument in the command line, have been evaluated.

To confirm that the program had calculated the prime numbers correctly, the results of the program were compared with known primes, as obtained from the University of Tennessee at Martin web site (Caldwell), as found in Table 2.

It should be noted that it is possible for the program to incorrectly label one or more particular numbers (i.e. determining that a prime number is not prime or the reverse), while still generating summarized values that are consistent with the above table. However, the odds of this occurring were deemed to be insignificant due to the simplicity of the underlying algorithm, as well as unimportant to the overall purpose of the program.

Table 2: Known Primes (Caldwell)

Upper Limit	Prime Numbers Less Than or Equal To Upper Limit
10	4
100	25
1,000	168
10,000	1,229
100,000	9,592
1,000,000	78,498
10,000,000	664,579

The total run time of the program is equal to the sum of the total calculation time and the total communication time. Since the initial cluster contained only one slave node, the total calculation time is effectively constant. By adjusting the value of the second command line argument, while holding the first command line argument constant, the program could be evaluated over a range of communication cycles. As expected, the shortest total run time of 588 seconds was obtained when only one communication cycle was required, and total run time did increase as additional communication cycles were incurred, but was fairly stable over a range of communication cycles.

Through 10,000 communication cycles, only 11 additional seconds of run time were needed with a single slave. At this point, however, additional communication cycles begin to materially impact the run time of the program. At the level of 100,000 and 1,000,000 communication cycles, the total program run time spikes to 682 and 1,443 seconds, respectively. These represent 16% and 145% increases over the minimum run time of 588 seconds. Thus, we were able to demonstrate a strictly increasing relationship between run time and communication cycles in our single node deployment. The following section of the paper explores the differences that the student will be expected to see in processing time and efficiencies as compared to the single slave design, when deploying the prime number program in a multi-node Beowulf cluster.

3.4 Analysis of Run Times for Six Slave Cluster

In a multi–slave node cluster systems design, should the instructor wish to further demonstrate the potential scalability efficiencies from parallel processing, it was expected that the run time of the program would improve as the utilization of the computing power of the entire cluster increases. As shown in Figure 2, the number of communication cycles greatly impact the total run time of the six slave cluster when calculating prime numbers up to six million. At fewer communication cycles, the cluster's efficiency is impeded by large amounts of idle time. For example, in looking at Figure 2, it is apparent when only one communication cycle is permitted, then one slave node calculates furiously, while the other five slave nodes remain idle.

If six communication cycles are used, then all six slave nodes will be utilized. However, since some scenarios may require more calculation time than other scenarios, it is likely that some slave nodes will finish relatively quickly and remain idle until the last slave node has finished. For example, the time required to calculate the primes between 1 and 1,000,000 is significantly less than the time required to calculate the primes between 5,000,001 and 6,000,000. As additional communication cycles are permitted, the job becomes more evenly distributed amongst the slave nodes, decreasing the quantity of slave node idle time, as well as total program run time. This is evidenced by the flattening of the above run time curve.

An interesting event occurred at the sixth communication cycle – the run time spiked up to a level nearly equal to the one communication cycle run. This is a result of the heterogeneous nature of the machines in the larger cluster. As the communication cycles increase from one to six, additional slave nodes are sequentially put into use. The cluster was structured in such a way that the most powerful of the idle slave nodes is added to the cluster; as such, the least powerful slave node is added last. The sixth slave node is not only the least powerful of the slave nodes, but significantly inferior. As such, adding the slower slave node to the cluster will actually impair the cluster's performance

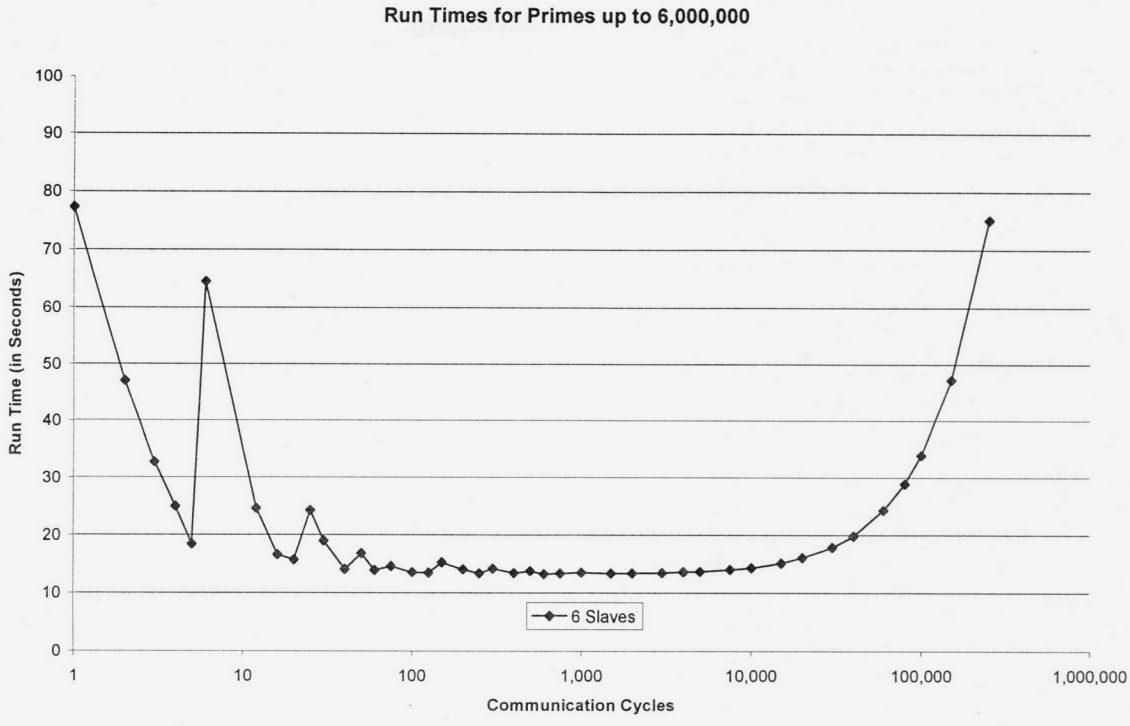


Figure 2: Run Times for Primes up to 6 Million with One Cycle

until sufficient communication cycles are available to more evenly distribute the job. In short, the first five slaves were finishing their assigned tasks rather quickly (i.e. in less than twenty seconds), but then needed to wait an additional forty seconds for the sixth, less efficient, slave to finish its task.

It should also be noted that there is a demonstrable trade off between slave node idle time and communication time. At some point along the domain of communication cycles, the reduction in slave node idle time will no longer offset the incurred increase in communication time, at which point the total program run time will begin to increase. For primes up to six million, this increase was first noticeable around ten thousand communication cycles and became exponentially more severe as the communication cycles exceeded one hundred thousand.

Had the six slave nodes been relatively homogenous, then it is theoretically possible for the program to be completed in as little as one–sixth of the one communication cycle run time (Brown, 2003). However, the heterogeneous slave nodes make it more difficult to estimate the theoretical lower bound of program run time. In reality, the theoretical minimum run time is unlikely to be achieved due to the additional communication time incurred, as well as the differing calculation times required to process different scenarios.

3.5 Run Times for Different Cluster Configurations

By removing one or more of the slower slave nodes from the cluster, a "new" cluster could be formed and its performance benchmarked for comparison against the six slave cluster. Figure 3 illustrates the impact of cluster size on run time.

It is notable that the five slave cluster run time was not appreciably slower than the six slave cluster run time. Once again, this is a result of the heterogeneous nature of the cluster and in particular, the significant performance inferiority of the sixth slave node. It is also notable that the removal of the slowest slave node also resulted in generally smoother run time curves, since the remaining nodes were relatively more homogenous. In addition, the total calculation for the one slave cluster is effectively constant. As such, additional communication cycles merely impede the progress of the one slave cluster, resulting in a strictly increasing run time curve.

It is notable that the five slave cluster run time was not appreciably slower than the six slave cluster run time. Once again, this is a result of the heterogeneous nature of the cluster and in particular, the significant performance inferiority of the sixth slave node. It is also notable that the removal of the slowest slave node also resulted in generally smoother run time curves, since the remaining nodes were relatively more homogenous. In addition, the total calculation for the one slave cluster is effectively constant. As such, additional communication cycles merely impede the progress of the one slave cluster, resulting in a strictly increasing run time curve.

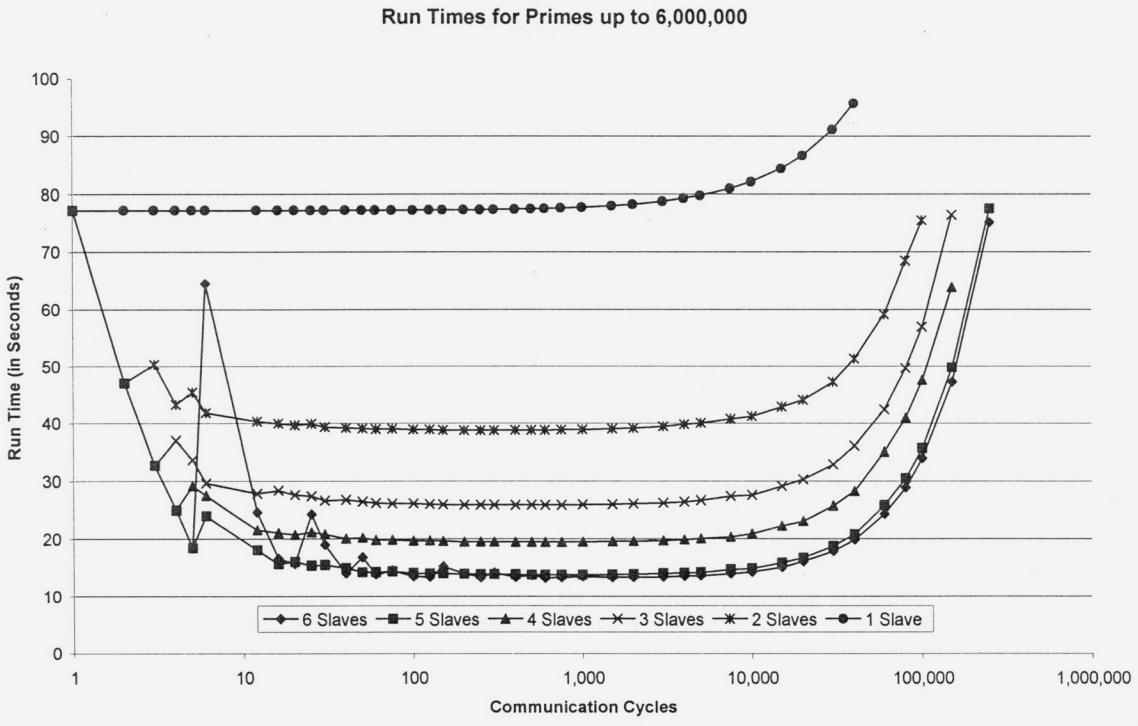


Figure 3: Run Times for Primes up to 6 Million with Six Cycles

In general, all of the clusters experience their minimal (or near minimal) run times over a large range of communication cycles. Furthermore, despite the heterogeneous nature of the underlying slave nodes, increasing the size of the cluster always resulted in improvements to the minimum run time.

Prime numbers up to sixty million were also evaluated for two to six slaves, to determine if this would have a material impact on the shape or relative dispersion of the various run time curves. As shown in Figure 4, the relative dispersion of the run time curves seems to be consistent with the behavior experienced during the previous (six million prime) runs. However, it appears that the deleterious impact of excessive communication cycles is delayed; noticeable impact to the run time does not occur until the communication cycles exceed one hundred thousand. Even at one million communication cycles, the impact of the additional communication time has not yet become critical.

3.6 Impact Of Cluster Overhead On Run Time

Calculating all of the prime numbers up to six million and sixty million is an inherently arduous task. It is unsurprising that increases to the cluster size were able to generate considerable savings in run time. However, not every job is a mammoth calculation. To better understand the time lost during the initiation of the cluster's message passing logic, the prime number program was modified to work within a standalone (i.e. non–cluster) environment. Four different runs were initiated, requiring the calculation of prime numbers up to 100,000; 1,000,000; 10,000,000; and 100,000,000. The standalone version of the prime number program was run on two different computers: a Pentium 4 at 2.4GHz with 192 MB of RAM and a Pentium II at 450 MHz with 192 MB of RAM. The total run time for these tasks was compared with the corresponding run times for various cluster configurations utilizing 500 communication cycles; the results are summarized in Figures 5 and 6.

For smaller tasks, the standalone computing jobs dominate the cluster–based run times. The greater run times for the clusters are a result of the overhead required to initialize the message passing within the cluster at the beginning of the prime number program's execution. Due to the relatively small amount of calculation involved, the cluster's greater computational power cannot offset the time lost during the initialization process. In fact, some of the smaller clusters dominated the larger clusters, as less time is required to initialize a smaller cluster.

However, once the tasks become sufficiently intensive from a computational standpoint, the greater power of the clusters begin to dominate the standalone jobs by an increasingly significant margin, as demonstrated in the Figures 7 and 8.

In short, the five to fifteen seconds of cluster initialization is a considerable impediment for a job requiring a minimal calculation time. However, for jobs requiring calculation time of hundreds or thousands of seconds, this overhead is inconsequential.

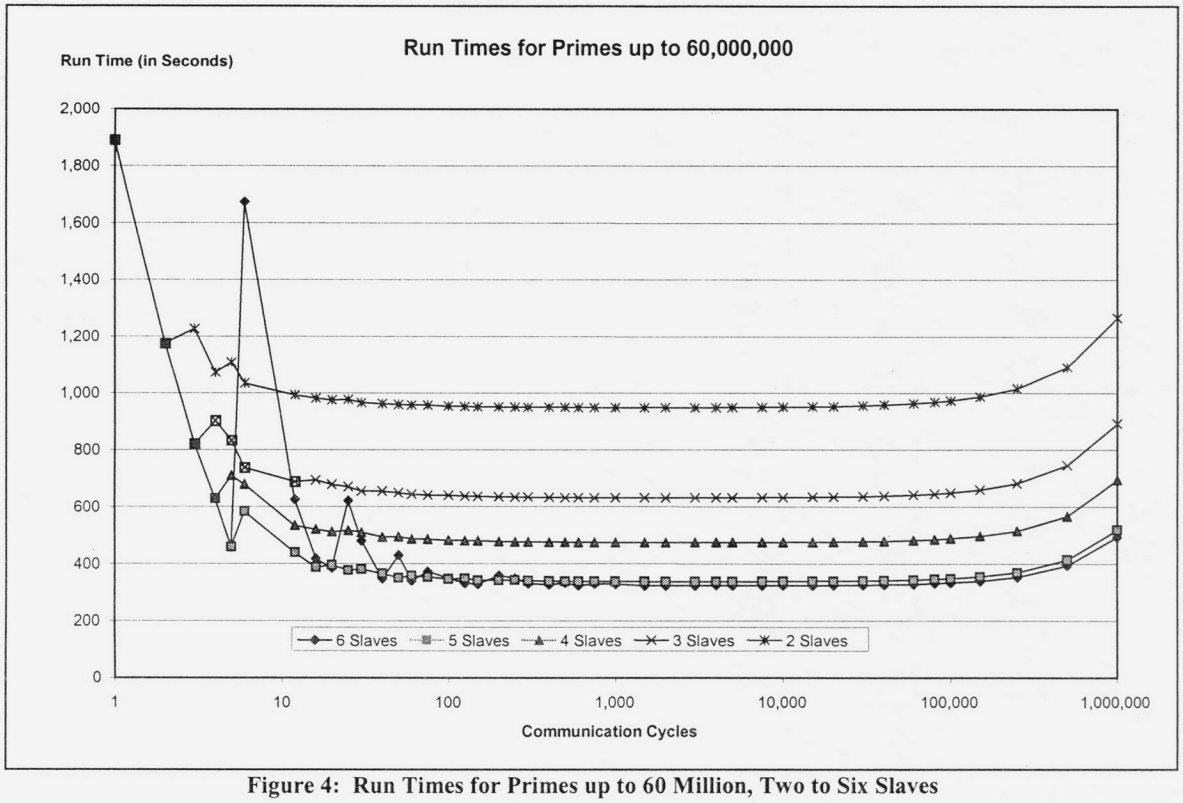


Figure 4: Run Times for Primes up to 60 Million, Two to Six Slaves

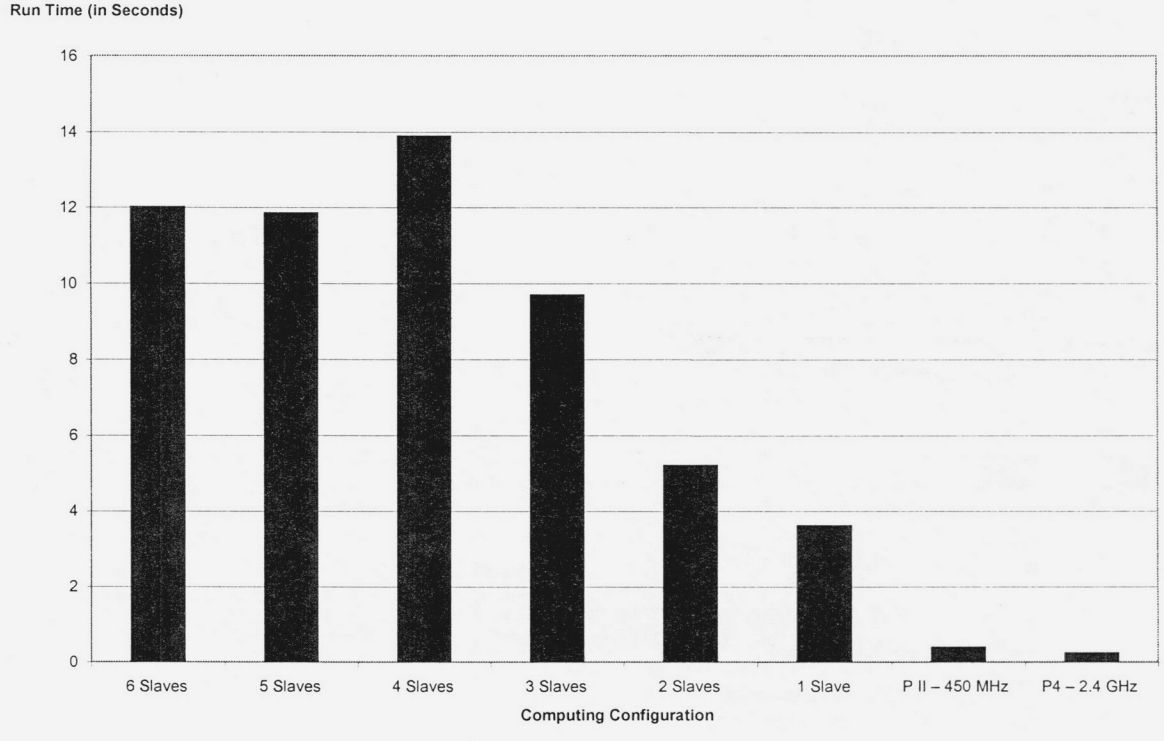


Figure 5: Run Time for Primes up to 100,000

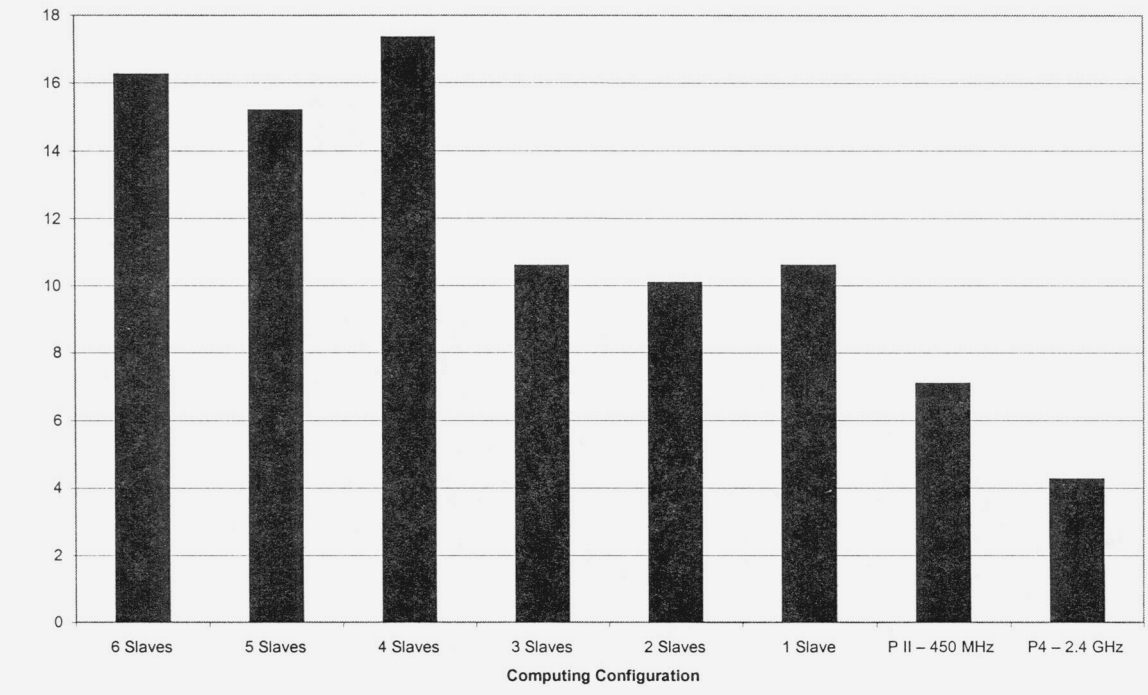


Figure 6: Run Time for Primes up to 1,000,000

4. CONCLUSION

Clustering is an interesting, new approach in which the collective computing power of multiple personal computers is harnessed and applied as one. The viability of deploying this technology in a lab setting was demonstrated with the creation of a two node cluster consisting of a master and slave node, as well as with a seven node cluster with one master and six slaves. A relatively simple prime number program was developed and confirmed to be executable within the parallel processing environment of the cluster. The use of the program for teaching purposes illustrated the importance of the underlying algorithm's efficiency and scalability, as well as the potential for intra–cluster communications to materially impact the execution time of the program. We suggest that this innovative approach will be useful for adding to student understanding of the concepts of the Linux platform, parallel processing applications, and issues related to multiple node scalability, all while presented in the traditionally resource limited classroom information sciences environment.

5. REFERENCES

Bellis, M. (1999). "Inventors of the Modern Computer." Retrieved April 24, 2004 from http://inventors.about.com/library/weekly/aa031599.htm

Blaise, A. (2001). "Beowulf clusters: e pluribus unum." Retrieved March 7, 2004 from http://www-106.ibm.com/developerworks/linux/library/l-beow

Brown, R. G. (2003). "Introduction to the BEOWULF design." Retrieved March 9, 2004 from http://www.phy.duke.edu/brahma/brahma_old/beowulf_intro_2003.pdf

Caldwell, C. K. "How Many Primes Are There?" Retrieved March 20, 2004 from http://www.utm.edu/research/primes/howmany.shtml

Gropp, W., Lusk, E., & Sterling, T. (Eds.). (2003). "Beowulf Cluster Computing with Linux." (2nd Ed.). Cambridge MA: The MIT Press.

Laudon, K. & Laudon, J. (2004). "Management Information Systems." (8th Ed.) Upper Saddle River NJ: Pearson Prentice Hall.

"Linux Central." Retrieved March 9, 2004 from http://www.linuxcentral.com/catalog/index.php3?prod_code=L000-114&id=C1C7KDl0iuZCQ

Middlemiss, J. (2004). "Gearing up for Grid." [Electronic version]. Wall Street & Technology, 22(2), 46–47.

Lonsdale, G. "Activities and Perspectives for Cluster and Grid Computing at NEC." Retrieved April 22, 2004 from http://www.isc2002.org/download/tutorial/lonsdale.pdf

Rine, J. (2004). "Brute Force Calculation of Prime Numbers in a Parallel Processing Environment." [Computer software].

Scyld Corporation. (2001). Beowulf CD–ROM (Release 27BZ–8) [Computer software]. Annapolis MD.

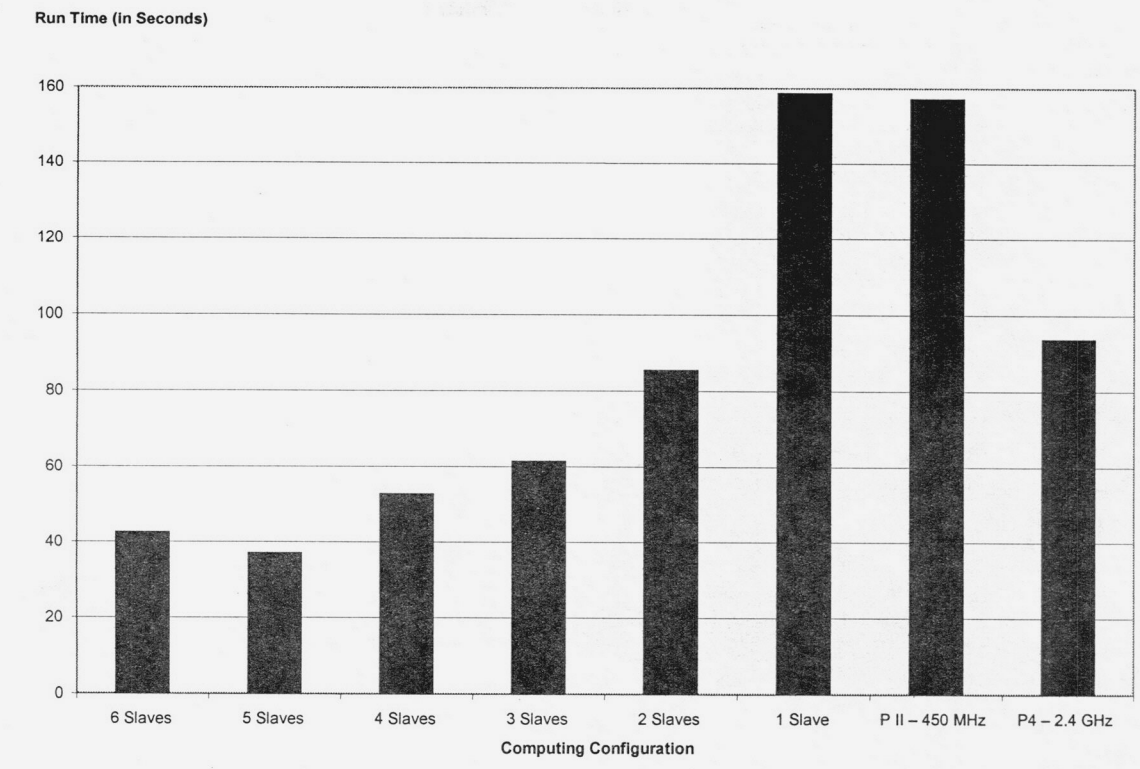


Figure 7: Run Time for Primes up to 10,000,000

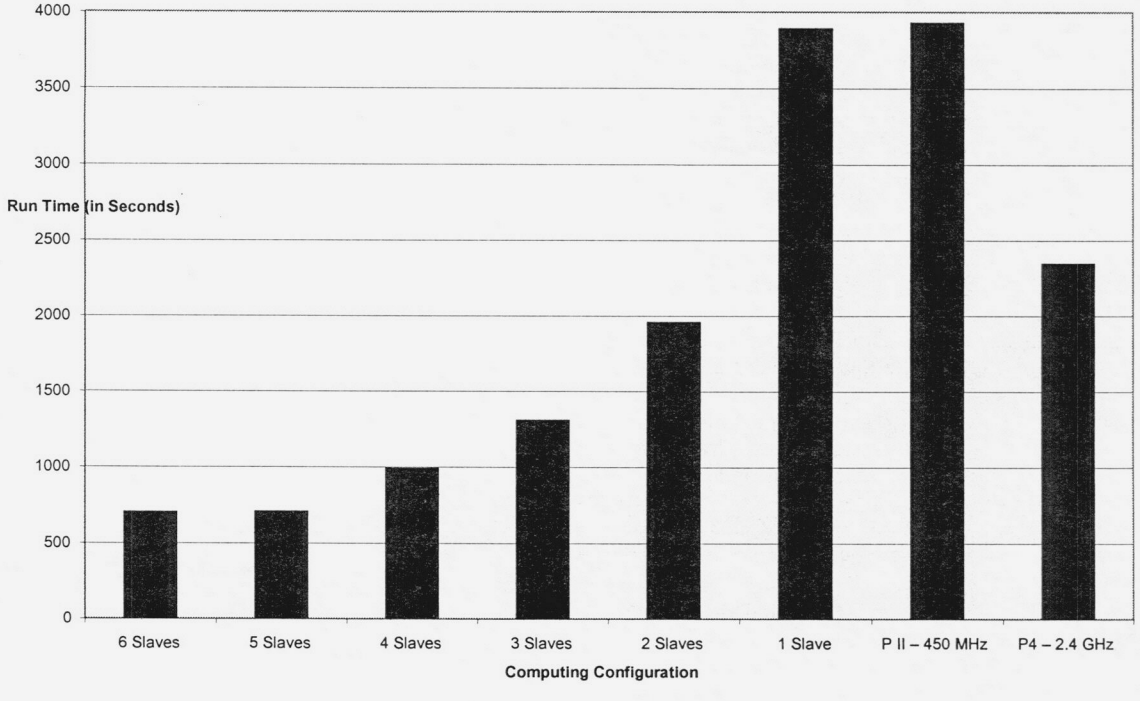


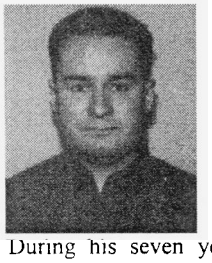
Figure 8: Run Time for Primes up to 100,000,000

SETI@home. (2004). "Active Users." Retrieved April 24, 2004 from http://setiathome.ssl.berkeley.edu/numusers.html

Taschek, J. (2003). "Time to end gridlock." [Electronic version]. eWeek, 20(22), 63.

YarKhan, A. & Manne, L. (1997). "Beginners Guide to MPI Message Passing Interface." Retrieved April 11, 2004 from http://www.jics.utk.edu/MPI/MPIguide/MPIguide.html

AUTHOR BIOGRAPHIES



Jay Rine is a Senior Consultant in the Finance Department of a Fortune 500 company in the financial services industry. His professional interests include risk management, with an emphasis on using Monte Carlo simulations as an analysis tool. Jay holds a BS in Mathematical Sciences from Worcester Polytechnic Institute and an MBA from West Virginia University. During his seven years as a practicing actuary, Jay had attained Membership in the American Academy of Actuaries, as well as Fellowship in the Society of Actuaries.

Dr. Virginia F. Kleist is an Assistant Professor of MIS at the College of Business and Economics at West Virginia University. Her research sits at the intersection of economics and information systems, investigating the information goods industries and cost versus benefit issues of biometrics, network security and knowledge management technologies. Recent publications include work on modeling technological based electronic trust and security in the digital economy and research on the adoption, assessment and payoff of electronic commerce business information systems projects. She holds a BA from Duke University in Economics, an MBA from Marquette University, and an MA in Economics, MS in MIS and PhD in MIS from the University of Pittsburgh. Dr. Kleist spent 10 years as a strategic analyst, manager, and director of information technology systems applications. Dr. Kleist was awarded the Best Doctoral Dissertation Award at ICIS in December 2000, and the WVU Foundation Outstanding Teaching Award in April 2003.

Brian McConahey recieved his BA from West Virginia University in 1996. He spent several years working in marketing, then returned to WVU in 2003 to earn his MBA. Brian has held a long interest in technology, particularly in the areas of hardware, operating systems and networking. Currently, he is living in the Pittsburgh, PA area working in the investment services industry.

APPENDICES

The programming for the Beowulf Computer Cluster is available at http://www.be.wvu.edu/divmim/mgmt/kleist/

Information Systems & Computing
Academic Professionals

STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the Journal of Information Systems Education have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN 1055-3096