A Low-Cost Computer Cluster for High-Performance Computing Education

Aaron M. Pfalzgraf and Joseph A. Driscoll
Department of Electrical and Computer Engineering
Bradley University
Peoria, IL 61625
E-mail: {apfalzgraf, jadriscoll}@bradley.edu

***Abstract*—Skills in high-performance computing are increasingly desired by many employers, such as Google [1] and Intel [2]. However, traditional curricula often do not provide adequate preparation. This is often due to the high cost of HPC equipment needed to provide students with needed hand-on experience. In this paper we describe a HPC system that is small, low cost, and yet powerful enough to support educational activities in HPC.**

Keywords*—*High-performance computing, parallel computing, computing education.

I. Introduction

In virtually all areas of science and engineering, and also areas such as economics and finance, computers are indispensable tools used to understand and predict complex phenomena. The scale of these calculations is limited by the computational power available, and the time one is willing to wait for a result. Many techniques exist that can improve the speed and efficiency of a computer program, and these methods are part of the field known as high-performance computing (HPC).

The use of HPC is rapidly increasing in both science and industry [3], [4], but current curricula do not adequately prepare enough students for this field [5]. A major obstacle for academic departments is the cost of HPC hardware. Low-cost alternatives such as LittleFe [6] provide one option. In this paper we describe our 25-node parallel computing cluster constructed with small single-board computers. Our system costs less than other educational HPC systems, and provides more processors.

A cluster consists of multiple "worker" computers that do the actual processing in parallel. These worker nodes are usually hidden from direct user access. Users log into a "head node" machine that provides system management. A user submits a parallel processing task on the head node, which then distributes the parallel tasks to some (or all) worker nodes. The user provides specifications such as how many workers are needed, etc. To allow multiple users to use the cluster in an organized way, open-source cluster management software is available. Examples include Torque [7] and Maui [8], which provide queue-based job scheduling, email notifications, etc.

A. Prior work

One of the main inspirations for our system was the LittleFe project [6]. LittleFe is a six-node parallel processing system made up of six small form factor motherboards. The cost for parts is approximately $3000, and relies on a frame that must be custom machined. LittleFe shares our goal of providing inexpensive HPC technology for education, and there are several sets of materials designed to aid an instructor. Our system has many more nodes than LittleFe (25 vs. 6) and costs much less (under $1000). In addition, no machining is needed: all parts can be ordered from standard vendors and assembled by hand in a relatively short amount of time.

Other groups have designed clusters of Raspberry Pi SBCs [9], [10], [11]. However, these tend to use custom case designs and other difficult to reproduce aspects. Our design goals explicitly included the ability for a user to simply order parts and hand assemble them, with very little tool use required.

The rest of the paper is organized as follows. The next section provides an overview of the design methodology used to create the cluster. This is followed by a discussion of various benchmark tests used to verify the operation of the system. Finally, we make concluding remarks and discuss possible directions for future work.

II. Methods

We have designed an inexpensive parallel processing system to allow education in high-performance computing. Of course, this system cannot be expected to compete with much larger (and much more expensive) systems. Still, the principles of HPC can be explored, and this is the major contribution: a cost-effective platform able to support most aspects of an HPC course. Since connecting the system to a network is trivial, multiple users can simultaneously log in to the system to work. Standard cluster management tools such as Torque [7], Maui [8], etc. can be installed on the head node to provide an organized, efficient way for users to submit jobs for execution on the worker nodes. This section provides details on the hardware and software components used in the system. The cluster is shown in Figure 1.

A. Hardware

Our goal was to design a system that could be easily assembled from off-the-shelf components, would be relatively inexpensive, and would provide a programming environment suitable for learning about high-performance computing.

The cluster is a collection of 25 Raspberry Pi Model B single-board computers (SBCs) [12]. One board serves as the cluster's head node, and the rest are worker nodes. The SBC

978-1-4799-4774-4/14/$31.00 ©2014 IEEE

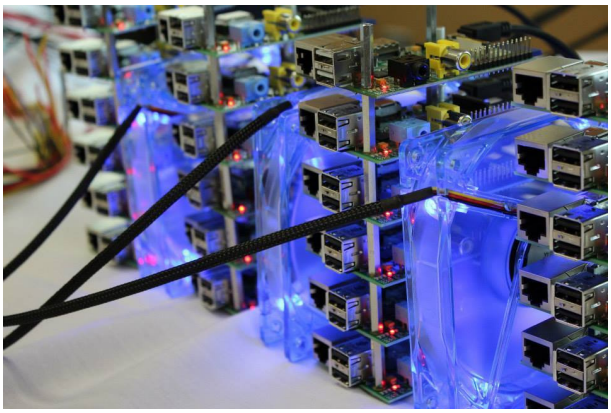

Fig. 1. The cluster described in this paper. Visible are the blue cooling fans, the black power cables, and the Raspberry Pi boards arranged in vertical stacks.

uses a lightweight Linux operating system called Raspbian [13]. Standard programming tools such as compilers, MPI libraries, etc. are either installed by default or can be easily installed.

Fig. 2. The Raspberry Pi Model B single-board computer. 25 of these were used in the cluster. For scale, notice the Ethernet port in the far right of the image. (Image source: Wikimedia Commons).

Other SBCs could be used to make a similar cluster. For example, one could use the Beaglebone Black, which features more RAM and a faster processor than the Raspberry Pi [14], [12]. Our choice between the two was based simply on the wider availability of the Raspberry Pi during development.

Each SBC has an Ethernet port, which allowed connection to a network switch via standard network cabling. This network connectivity allows all nodes (head node and the worker nodes) to exchange information needed during a parallel computation. In addition, the network is used to send status and other management messages between worker nodes and the head node. The switch also was connected to the outside network, allowing remote access to the cluster, for example by using an SSH client [15].

The Raspberry Pi boards have holes pre-drilled, and these were used to connect them in vertical stacks using standoffs and screws. We allowed enough space between boards to allow for adequate airflow. The stacks were typically no more than six boards high, to match the approximate height of cooling fans placed between the stacks.

The cluster was powered by a standard PC ATX 600W power supply. The 5V rail was used connected to 25 micro USB plugs, which connected to each board to provide power. Note that often Raspberry Pi boards are powered through pins on the expansion header. The problem with this is that it bypasses a protection fuse. For this reason, we chose to power the boards via the micro USB port, which allows fuse protection. The power supply also supports four 120mm PC case fans, which are used between the stacks of SBCs to provide airflow for cooling.

The number of nodes in our cluster is somewhat arbitrary, and could be adjusted to provide less cost (fewer nodes) or more processing power (more nodes). A reasonable minimum number of boards would be 4-6, which would still allow basic parallel processing concepts to be studied. As for increasing the number of nodes, one must be cautious. Having too many nodes will saturate the network and internode communication will be a bottleneck, limiting performance. We empirically determined 25 nodes to be a reasonable compromise between performance, network saturation, and cost.

B. *Software*

All software used in our system is open-source, further reducing the overall cost. The Raspberry Pi SBCs can use a variety of Linux operating systems; we chose to use Raspbian, which is a variant of the Debian Linux distribution [16] that has been modified for the Raspberry Pi. A variety of compilers, text editors, and other tools are available on the system. The worker nodes are configured to be identical. The head node differs in that it is the node that users log into via SSH, while the workers are hidden. The head node is also responsible for sending computational tasks to the workers, collecting the partial results, and providing the final results to the user.

The Message Passing Interface (MPI) library [17] was used to develop parallel processing tasks to serve as benchmark tests for the system. Results of these tests are presented in Section III. While a wide variety of programming languages can be used in HPC, we chose to use Python [18] for our tests. This was a deliberate choice, as Python is recognized [19] as an easy to learn, yet powerful programming language. It is used widely in education to teach programming, and yet is powerful enough to be well-regarded in industry. The “mpi4py” Python library [20] provided access to the MPI library’s various functions. We also used the “numpy” Python library [21] to provide mathematical tools needed for the benchmark tests.

Often, more complex benchmark suites such as LINPACK [22] are used to profile the performance of an HPC system. In this case, we wanted to develop a total system for education, both hardware and software. So, by creating example tests in an easy to learn language, we lower barriers to adoption. LINPACK is a good test, but is better suited to testing powerful, expensive systems that care about computational speed. In our case, we know our system is less powerful than top supercomputers. Our goal in the tests is to demonstrate that the system is suitable for use in teaching standard HPC topics, such as the trends seen in the test results (see Section III).

III. Results

In this section we discuss results of several benchmark tests applied to the system. These tests were not designed to demonstrate the computational power of the cluster, since that is not the system's purpose. Instead, we present three tests written in accessible Python code. The tests demonstrate that the systems do in fact display typical parallel computing behavior, and so establish that the system would be useful in an educational setting.

Three tests were used, all of which involve basic linear algebraic operations between scalars, vectors, and matrices. These tests were chosen because of both their wide use in HPC educational materials, and also for how common these mathematical operations are in HPC application software. For a given test, we varied the problem size n, which is used as the vector length or the dimension of a square matrix. For each size we measured the time needed to complete the operation. To mitigate the influence of noise in the timing, we repeated each measurement 10 times and used the average value.

A. Vector Triad

The vector triad algorithm uses four vectors of size n and computes the following:

$$a(n) = b(n) + c(n) * d(n) \tag{1}$$

Figure 3 shows data collected using varying values of n. The vertical axis represents average elapsed time for the test, so a lower value is better. These results display the well-known

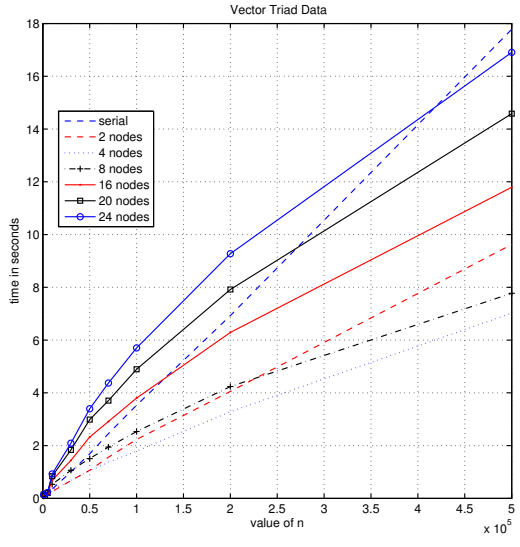

Fig. 3. Data collected for the vector triad benchmark test.

[23] concept that with parallel processing, the problem size must be large enough to justify the internode communications overhead. In Figure 3 we see that initially, the serial version outperforms several of the parallel implementations. Once the problem size grows beyond approximately $n = 45000$, all of the parallel versions outperform the serial code.

B. Matrix-Vector Multiplication

The matrix-vector multiply test makes use of a matrix of size $n \times n$ and a vector of length n to compute the following product:

$$B = A * x(n) \tag{2}$$

The matrix-vector multiplication test shows a much more clear distinction between serial and parallel; see Figure 4. The parallel execution of the matrix vector multiply computation finished in 1/4 of the time it took to run in serial (for $n = 1000$). The exponential nature of the data is from the number of memory accesses as the size of the matrix increases. This clear distinction between serial and parallel performance

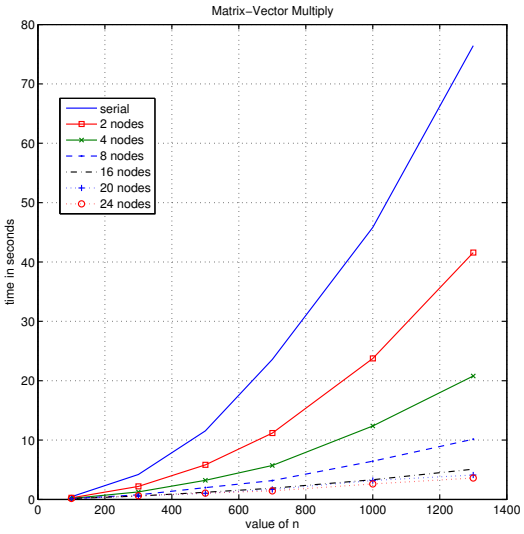

Fig. 4. Data collected for the matrix-vector multiplication benchmark test.

is key to understanding why parallel programming is beneficial. Algorithms such as this give students a clear and practical example of the benefits of programming in parallel. This particular algorithm transitions very easily from serial to parallel forms, making it accessible for use in the classroom. Starting with this simple algorithm, it is possible to extend the concept and realize how much time would be saved on a similar application that takes days or weeks to run in a serial form.

In addition to the results from a fixed number of nodes with a varying n, a test with a fixed n was performed as well. It was determined that the differences between the number of nodes would be most clearly seen by using the matrix-vector multiplication algorithm. The operation was run on the cluster while varying the number of active workers. Figure 5 shows results for 0, 2, 4, 8, 12, 16, 20, and 24 worker nodes.

C. Matrix-Matrix Multiplication

The matrix-matrix multiply makes use of two $n \times n$ matrices to compute

$$C = A * B \tag{3}$$

The parallel code ran slower in the case of the matrix-matrix multiplication test, as shown in Figure 6. The slowdown is

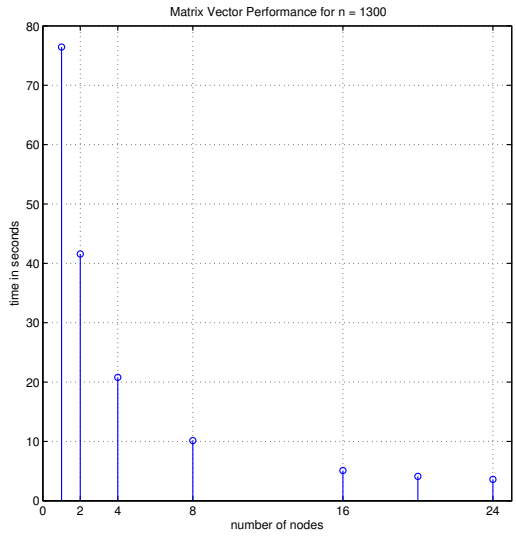

Fig. 5. Results of varying number of nodes with fixed problem size n ($n = 1300$).

most likely caused by the data access pattern of the program. If the algorithm was not written properly, the program may not benefit from any CPU cache speeds and may have resided in RAM. This would have slowed down the acquisition of the data. The results appear to be serialized parallel. Such results

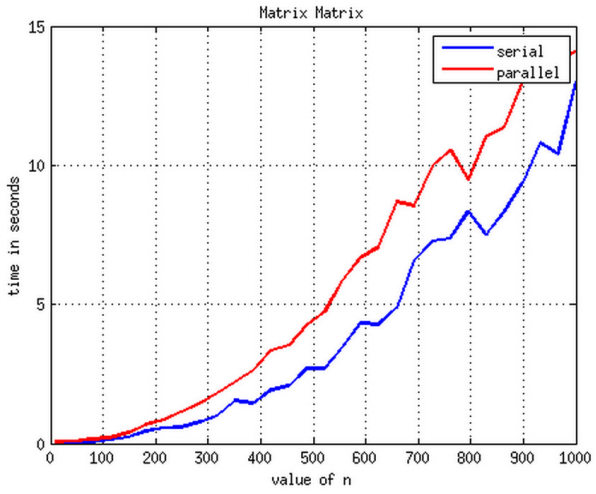

Fig. 6. Data collected for the matrix-matrix multiplication benchmark test.

present students with a unique opportunity to explore what went wrong. It is often simple to see why something worked; this is not so when something fails. The results shown can be attributed data acquisition pattern in the software. There are many possible reasons for this; the algorithm is not very straightforward to code. The matrix-matrix multiplication algorithm, unlike the matrix-vector multiply, does not transition as easily from serial to parallel [24]. If the program does not access the data efficiently, it will result in increasing amounts of computation time.

IV. Conclusions and Future Work

We have presented a low-cost, easy to construct, and effective system to aid in high-performance computing education. Compared to similar systems such as LittleFe [6], our system has more nodes, costs less, and is simpler to construct. The increased number of nodes is a major advantage in an instructional environment, where varying the number of nodes in a task, and seeing how the results change, is a common task.

Our benchmark materials in Python serve as both performance tests and also as sample educational materials. The tests demonstrate that the system does display the well known behavior of parallel systems. While many other languages can be used in the system, our use of Python takes advantage of its well-noted [19] ease with which it can be learned, further lowering educational barriers.

Future work includes expanding the number of nodes until network congestion proves to be too much of a bottleneck. This will establish the practical limit of the system's size. Also, we plan to develop another version of the cluster using the Beaglebone Black [14], which is rapidly growing in popularity and is more powerful than the Raspberry Pi [14], [12].

Acknowledgement

The authors gratefully acknowledge the financial support of Bradley University's Caterpillar College of Engineering and Technology and the Department of Electrical and Computer Engineering.

References

[1] B. Reed. (2007) Google, ibm launch parallel-computing initiative. http://www.networkworld.com/.

[2] J. Kruszewski. (2014) University of bristol joins intel parallel computing center program. http://newsroom.intel.com/docs/DOC-4942.

[3] I. Research. (2013) High performance computing market to exceed $40 billion by 2017. http://www.intersect360.com.

[4] K. A. Frenkel. (2013) Preparing a parallel programming workforce. http://cacm.acm.org/news/.

[5] D. Lyons. (2011) The u.s. is busy building supercomputers, but needs someone to run them. http://www.thedailybeast.com.

[6] D. A. Joiner, P. Gray, T. Murphy, and C. Peck, "Teaching parallel computing to science faculty: Best practices and common pitfalls," in *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '06. New York, NY, USA: ACM, 2006, pp. 239–246. [Online]. Available: http://doi.acm.org/10.1145/1122971.1123007

[7] G. Staples, "Torque resource manager," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. ACM, 2006, p. 8.

[8] D. Jackson, Q. Snell, and M. Clement, "Core algorithms of the maui scheduler," in *Job Scheduling Strategies for Parallel Processing*. Springer, 2001, pp. 87–102.

[9] S. Cox, J. Cox, R. Boardman, S. Johnston, M. Scott, and N. OBrien, "Iridis-pi: a low-cost, compact demonstration cluster," *Cluster Computing*, pp. 1–10, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10586-013-0282-7

[10] J. Kiepert. (2013, May) Creating a raspberry pi-based beowulf cluster. http://coen.boisestate.edu/ece/raspberry-pi/.

[11] F. P. Tso, D. R. White, S. Jouet, J. Singer, and D. P. Pezaros, "The glasgow raspberry pi cloud: A scale model for cloud computing infrastructures," in *Distributed Computing Systems Workshops (ICDCSW), 2013 IEEE 33rd International Conference on*. IEEE, 2013, pp. 108–112.

[12] E. Upton and G. Halfacree, *Raspberry Pi User Guide*. John Wiley & Sons, 2013.

[13] M. Thompson and P. Green. (2012) Raspbian operating system. http://www.raspbian.org.

[14] G. Coley, "Beaglebone black system reference manual," 2013.

[15] D. J. Barrett and R. E. Silverman, *SSH, the Secure Shell: the definitive guide*. O'Reilly Media, Inc., 2001.

[16] I. M. et al. (2014) Debian operating system. http://www.debian.org.

[17] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: portable parallel programming with the message-passing interface*. MIT press, 1999, vol. 1.

[18] G. Van Rossum *et al.*, "Python programming language." in *USENIX Annual Technical Conference*, 2007.

[19] A. Radenski, ""python first": A lab-based digital introduction to computer science," *SIGCSE Bull.*, vol. 38, no. 3, pp. 197–201, Jun. 2006. [Online]. Available: http://doi.acm.org/10.1145/1140123.1140177

[20] L. Dalcin, "mpi4py," 2007.

[21] T. E. Oliphant, *A Guide to NumPy*. Trelgol Publishing USA, 2006, vol. 1.

[22] J. Dongarra and P. Luszczek, "Linpack benchmark," *Encyclopedia of Parallel Computing*, pp. 1033–1036, 2011.

[23] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. Von Eicken, *LogP: Towards a realistic model of parallel computation*. ACM, 1993, vol. 28, no. 7.

[24] J. Gunnels, C. Lin, G. Morrow, and R. Van De Geijn, "A flexible class of parallel matrix multiplication algorithms," in *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International... and Symposium on Parallel and Distributed Processing 1998*. IEEE, pp. 110–116.