USING PHOENIX++ MAPREDUCE TO INTRODUCE UNDERGRADUATE STUDENTS TO PARALLEL COMPUTING *

Suzanne J. Matthews
Department of Electrical Engineering & Computer Science
United States Military Academy
West Point, NY 10996
845-938-5577
suzanne.matthews@usma.edu

ABSTRACT

With the release of CS2013, departments around the country are injecting more parallelism into their core computer science courses. MapReduce is an attractive option for introducing undergraduate students to parallelism. However, most approaches to teaching MapReduce require students to access a Hadoop cluster. This is often too costly to be a viable option at many institutions, especially if MapReduce represents a minor topic in a course. In this paper, we discuss the use of Phoenix++ MapReduce for introducing undergraduate students to parallel computing and the MapReduce paradigm. We publish an open-source module on Phoenix++, enabling instructors at other institutions to rapidly adopt Phoenix++ MapReduce for use in their own curricula.

INTRODUCTION

The introduction of the Parallel and Distributed Computing (PDC) knowledge area in the IEEE/ACM Curriculum Guidelines for Undergraduate Degree Programs in Computer Science (CS2013) [9] spurs many undergraduate programs to explore how to incorporate more PDC topics into their curricula. A main challenge is figuring out how to "fit" PDC concepts into existing core courses. While electives are a convenient vehicle

for introducing PDC topics, CS2013 recommends fifteen hours of PDC education in core offerings in the CS curriculum [9]. However, faculty are sometimes resistant to modify existing courses due to a perceived lack of room for PDC material. MapReduce is an attractive option to introduce students to parallel computing, since it requires students to only implement two serial functions (map() and reduce()); the underlying framework automates the rest. The general concept of MapReduce can typically be covered in one lesson period and it is applicable to several fields.

However, open-source commercial MapReduce systems such as Hadoop [19] require a dedicated cluster, which can be too expensive to justify a brief use in a course or by small departments. Not having access to such systems can prevent students from gaining valuable practical experience writing MapReduce jobs. As an alternative, we propose the use of Phoenix++ [18], a multicore implementation of the MapReduce framework. Given the ubiquity of multicore architectures, almost all computer science departments have existing lab equipment that are multicore. This makes Phoenix++ easy to deploy in a classroom or laboratory setting.

To aid educators at other institutions in using Phoenix++ in their own courses, we design a MapReduce module specifically for Phoenix++. It is publicly available through CSinParallel [1, 2], an NSF-funded initiative which curates and hosts open-source modules for PDC education. The use of modules is championed by several researchers [1, 6, 11, 14] for the rapid integration of parallel and distributed computing concepts into the undergraduate curriculum. Modules are hands-on learning experiences that can be implemented in one to two lesson periods or a lab hour. Our contribution represents the 25th module in CSinParallel's collection, and can be completed within a single 55-minute lesson period.

We have used Phoenix++ for the last three years to introduce students to parallel computing in two separate courses and three year-long undergraduate capstone projects. Our students in our capstone course have created novel MapReduce applications using the Phoenix++ software, yielding multiple posters and peer-reviewed publications. Our experiences suggest that Phoenix++ is successful in helping students quickly grasp the concepts of MapReduce, and enables them to solve real-world problems with little overhead. Given our success, we believe that Phoenix++ is a great alternative to Hadoop for teaching MapReduce concepts at undergraduate institutions. We believe Phoenix++ will be a boon especially to smaller departments, who often cannot afford to set up their own Hadoop clusters or gain access to one.

RELATED WORK

The MapReduce paradigm was popularized [5] by Dean and Ghemawat of Google in 2004. Inspired by the map and reduce constructs found in functional programming languages, MapReduce requires the programmer to serially implement the contents of the map() and reduce() functions. The framework automates the scheduling of map and reduce tasks, enabling the programmer to focus on the parallelization of the algorithm at hand, rather than spending valuable time programming synchronization constructs and other details [5].

Google's implementation of MapReduce remains closed-source. Apache Hadoop [19] is the most well-known open-source implementation of the MapReduce framework, and has been widely used in the classroom. For example, the University of Washington [10] and the University of Maryland [12] used Hadoop clusters provided through the Academic Cloud Computing Initiative to teach cloud computing courses. Note that neither institution was responsible for creating their own Hadoop cluster. Students also worked on MapReduce projects over several weeks in a dedicated elective course.

Creating a Hadoop cluster locally at an institution can be challenging and expensive. Clemson [13] pursued a two-prong approach, having students run virtualized instances of Hadoop and carving out 8 nodes of their university supercomputer as a dedicated Hadoop cluster. Students spent a significant time getting their VMs to work. While the dedicated Hadoop cluster worked better, the cluster became overloaded as deadlines approached, and experienced data errors and crashes [13]. The researchers concluded that maintaining an in-house Hadoop cluster was not practical due to technical issues. St. Olaf [3] explored virtualization as a mechanism to have a non-dedicated Hadoop cluster; they note however that large in-home clusters require non-trivial cooling and power costs.

For the above reasons, the cost and setup time of a Hadoop cluster may be excessive for courses covering MapReduce for a small unit or single lesson. While using a resource like Amazon EC2 may seem like an attractive option, inexperience with such systems can cause students to quickly exceed account limits. Researchers at Berkeley [15] used Amazon EC2 to teach MapReduce concepts to undergraduates. While they were able to demonstrate good speedups, they questioned the suitability of cloud-based billing services for the classroom [15]. Students in the course indicated that they would not be willing to provide their own credit cards in case they exceeded their account limits (Amazon provided funding in the amount of $100 per student).

A MODULE FOR TEACHING MAPREDUCE WITH PHOENIX++

Phoenix++ [18] is a shared-memory implementation of MapReduce created by researchers at Stanford University. To the best of our knowledge, we are the first to use Phoenix++ for educational purposes and have used it extensively as a vehicle to introduce students to undergraduate research. There are several advantages to using Phoenix++ to cover MapReduce concepts. First, given the universality of multicore architectures, Phoenix++ can be run on individual workstations, without any need for additional networking or hardware. Next, Phoenix++ is capable of yielding speedups for smaller datasets, something that Hadoop is not designed for. Thirdly, its implementation in C/C++ makes Phoenix++ a very attractive option for courses that target those languages. This enables the coverage of MapReduce in a systems course, a popular place for injecting parallelism in the CS curriculum. Lastly, it does not prevent teaching the distributed aspects of MapReduce, which can still be discussed at a high level.

To aid educators at other institutions in adopting Phoenix++ in their own courses, we create a module that is currently published on the CSinParallel website: http://csinparallel.org/csinparallel/modules/PhoenixMRIntro.html. Our contribution represents the 25th module available in CSinParallel's collection. In addition to HTML, the module is available in PDF or Word form. The module also includes a custom

distribution of the Phoenix++ WordCount example that allows students to see how a standalone project written in Phoenix++ can be organized. The WordCount example is simplified to improve students' ability to read the code.

Our module is split into three parts. The first, “What is MapReduce?” gives the historical motivations and origins of the MapReduce paradigm, the most popular implementations of MapReduce, and Phoenix++. The next part, “Counting Pease with MapReduce”, discusses the classic WordCount example using a popular Mother Goose poem, along with the purpose of the map() function, the reduce() function, and the combiner. The third part, “Getting Started with Phoenix++”, walks the reader through the actual implementation of the Phoenix++ WordCount program, line by line, and explains what the code is doing in detail. The section concludes with a set of exercises for students to complete. For example, students can vary the number of application threads and note the speedup on a 100 MB text file composed of concatenated Project Gutenberg texts. Students can then vary the size of the data file (from 100 MB to 500 MB in increments of 100 MB) and note improvements in speedup.

CS2013 mentions the teaching of MapReduce to reinforce concepts in Computational Science (CN), Information Management (IM), and Parallel and Distributed Computing (PDC). Crucially, CS2013 does not advocate the teaching of any particular flavor of MapReduce. Instead, it calls on educators to emphasize MapReduce's role in processing large amounts of data and its use of the data-parallel decomposition strategy. Phoenix++ enables educators to cover such concepts in small doses, without requiring a Hadoop cluster.

In Table 1, we give an overview of the Core Tier 1/2 learning outcomes from the Parallel and Distributed Computing (PDC) knowledge area from ACM CS2013 [9] that can be satisfied by our module. Please note that the numbering of the outcomes correspond to those specific outcomes listed in CS2013. We note that our module also satisfies outcomes from the CN and IM knowledge areas. However, coverage of MapReduce is elective in those areas, and therefore the corresponding outcomes are not listed in Table 1. A key advantage of Phoenix++ is that instructors can use it to demonstrate application speedup, while still covering outcomes related to MapReduce, such as data parallelism and decomposing a problem into map and reduce tasks. This enables instructors to cover several outcomes in a single lesson period. In our department, we reinforce several of the listed outcomes with our other lessons dealing with concurrency.

Table 1. Applicable PDC Core Tier 1/2 Topic Areas from ACM CS2013.

PDC Topic	Learning Outcome
PD/Parallel Decomposition	1. Explain why synchronization is necessary in a specific parallel program. 5. Parallelize an algorithm by applying data-parallel decomposition.

PD/Parallel Algorithms, Analysis, and Programming	3. Define "speed-up" and explain the notion of an algorithm's scalability in this regard. 4. Identify independent tasks in a program that may be parallelized. 5. Characterize features of a workload that allow or prevent it from being naturally parallelized. 7. Decompose a problem (e.g., counting the number of occurrences of some word in a document) via map and reduce operations.
PD/Parallel Architecture	1. Explain the differences between shared and distributed memory. 8. Describe the key performance challenges in different memory and distributed system topologies.

RESULTS

We have used the material in our Phoenix++ module to introduce undergraduate students to parallel computing for the past three years. Phoenix++ was used as part of a lesson on MapReduce in a parallel computing elective (CS485) and a required computer organization course (CS380). We have also used Phoenix++ to introduce undergraduates to parallel computing and research methods in the context of year-long capstone experiences (CS401/CS402). In the subsections below we present some sample benchmarks, observed benefits of using Phoenix++ to teach MapReduce concepts, and observed benefits for introducing undergraduates to research. We base our observations on three years of experience using Phoenix++ for each of these different contexts.

Observed Speedups Using Phoenix++

To illustrate the level of speedup that can be observed, we perform some benchmarking on one of the lab workstations typically used by our students. During class, each student has access to separate, identical workstations in the lab. Each of our lab workstations consists of a quad-core Intel i7-3770 CPU @ 3.40 GHz, with 32 GB of RAM. Due to hyper-threading, there are 8 virtual cores on each system. Using the collected works of Sherlock Holmes ("sherlock.txt" in the data/ folder of the files distributed with the module), we can use concatenation to create files that range from 100 MB to 500 MB, in increments of 100 MB.

We measure average run time, speedup, and efficiency. Speedup is calculated using the equation: $s_n = \frac{Time(1\,core)}{Time(n\,core)}$. In an ideal scenario, running a parallel program on n cores will yield a speedup of n. Efficiency is calculated using the equation $E_n = \frac{s_n}{n}$. Ideally, we would want efficiencies to be close to 1. In our experiments, we keep the file sizes relatively small to ensure students can complete the exercise in a reasonable amount of time.

Table 2. Run time Benchmarks.

Time (s)	100 MB	200 MB	300 MB	400 MB	500 MB
1 core	1.00	1.96	2.95	3.92	4.86
2 cores	0.55	1.05	1.57	2.08	2.56
4 cores	0.32	0.61	0.89	1.18	1.46
8 cores	0.23	0.45	0.66	0.87	1.07

Table 2 illustrates the results of our benchmarking. Each cell *(i, j)* represents the average of three runs of dataset j on i cores. As we increase the size of the dataset, the run time increases by a near-linear amount. Increasing the number of cores to 2 results in speedups varying from 1.82 to 1.90, which increase with the size of the dataset. The efficiencies vary from 0.91 to 0.95. On 4 cores, speedups on our datasets range from 3.12 to 3.32, increasing as the size of the dataset increases. The efficiency on 4 cores varies from 0.78 to 0.83. Further increasing the number of cores to 8 increases our speedups to a range of 4.29 to 4.52, while our efficiencies range from 0.53 to 0.57.

We note that our results are typical for a threaded application running on a multicore system. The observed speedups for each dataset increase with the number of cores, and the efficiencies for a particular number of cores increase with the size of the data. We note the low efficiencies on 8 cores is unsurprising [17]. While increasing the number of threads can improve application run time on a hyper-threaded system, the best efficiencies are observed when each thread can be assigned to separate physical cores. Overall, while one could theoretically achieve better speedups with a hand-optimized threaded implementation, the observed speedups of Phoenix++ WordCount are acceptable for an application that automates its parallelism.

The observed results can lead to many interesting in-class discussions on important topics. For example: the tradeoffs between abstraction and performance; performance on physical cores vs virtual cores; Amdahl's Law; efficiency vs. speedup. The list goes on. However, perhaps the most important lesson is showing that it is possible to achieve speedup with MapReduce, when provided with an appropriate application space and framework.

Observed Benefits of using Phoenix++ to teach MapReduce Concepts

In both CS380 and CS485, students were primarily programming in the C language, and spent a unit on programming POSIX threads (Pthreads). Phoenix++ was introduced as part of a single lesson on MapReduce toward the end of both courses. This is not coincidental; the CS485 elective offering was developed to help facilitate the design of parallel computing education material that could then be injected into required CS courses in our curriculum. The Pthreads unit and MapReduce lesson in CS380 were taken verbatim from CS485.

In both courses, the unit on Pthreads helped set the stage for the later discussion of MapReduce. After explicitly spawning/joining threads and dealing with synchronization

constructs, the students are eager to learn about an automated system for parallel programming. Note that a knowledge of Pthreads is not required to complete the module. In CS380 and CS485, we simply leverage students' existing knowledge of Pthreads (and first-hand experience with the complexities and frustrations of programming with them) as a motivational device for automated frameworks for parallelization. The module material is covered in lecture form over 55 minutes, with the last 20 minutes of the period devoted to measuring speedup with Phoenix++ on varying datasets.

There are several advantages in using Phoenix++ over Hadoop in our two courses. First, the time and expense to create a Hadoop cluster far exceeds its planned use in our class (a single lesson period). Using Phoenix++ is very simple: students download and decompress the provided archive, and type "make" in the root directory. Students are then able to run the WordCount application and measure speedup. Second, as all the code is in C/C++, students are able to quickly comprehend the contents of the map() and reduce() functions by tracing through the code. Lastly, since Phoenix++ was built using POSIX threads, it demonstrates how an automated framework can be built on such primitives (though this is perhaps not interesting to those students unfamiliar with Pthreads).

Using Phoenix++ in conjunction with our lessons on POSIX threads have allowed us to give our students a hands-on introduction to distributed computing and data parallelism concepts without needing to rely on additional hardware, such as GPUs or a cluster. Note that Phoenix++ does not prevent the discussion of how MapReduce is implemented on a distributed system; we do discuss this in our MapReduce lesson. The primary benefit is that it enables us to give students a "hands-on" experience with MapReduce at virtually no cost, by leveraging existing multicore lab equipment.

Observed Benefits of using Phoenix++ for Undergraduate Research

All computer science seniors at West Point are required to complete a team-based, two-semester capstone experience. Students typically choose their capstone project from an available list at the end of their junior year. In each of the MapReduce projects, students start with the topics covered in the Phoenix++ module. This enables them to learn basic benchmarking concepts, such as how to take run-time measurements, calculate speedup, and plot results. To make it more challenging, we give students the original Phoenix++ code (rather than the simplified version in the module) and have them figure out how to vary the number of threads. This forces students to dive into the code early on and understand how components interact with each other.

Given MapReduce's applicability to problems that involve distributed hashing, our students have used the paradigm to work on many eclectic projects. For example, the capstone group in Year 1 used Phoenix++ to compare phylogenetic trees. A separate group of students in Year 2 used Phoenix++ for authorship verification, parallelizing an existing algorithm [8] for write-print creation. In Year 3, another group of students used Phoenix++ to implement a novel, parallel anomaly detection algorithm to detect alarm events on a 1000:1 scale emulation of a power distribution grid.

In the context of our year-long capstone course, our experiences over the last three years indicate that Phoenix++ is useful for creating small, manageable (yet compelling) undergraduate research projects related to parallel computing that can be completed in

a single year. Students have gained a lot of useful skills fundamental to the research process, such as reproducing the results of existing research (and its difficulties), learning unfamiliar code-bases, describing research results by writing academic papers, and presenting at national conferences. For the faculty advisor, the use of Phoenix++ allowed for the design of projects that enable undergraduate students to gain practical experience with the MapReduce paradigm and parallel computing without access to a Hadoop cluster.

The projects have yielded three peer-reviewed posters at venues such as the ACM Student Research competition and the Grace Hopper conference. Two projects led to peer-reviewed conference papers, and a third yielded a peer-reviewed technical report. Five students from the Year 1 and Year 2 projects have since been accepted to graduate computer science programs; three of the five are currently attending graduate school on competitive fellowships. Our experiences support existing literature indicating that successful research experiences at the undergraduate level assist in recruiting and retaining students in graduate computer science programs [4, 16].

CONCLUSIONS

In this paper, we discuss the use of Phoenix++ to teach MapReduce concepts to undergraduates. Phoenix++ is a multicore implementation of the MapReduce paradigm. Unlike Hadoop, Phoenix++ enables students to observe application speedup on a single multicore workstation. This is especially valuable for small departments for whom accessing or building and maintaining a cluster is too expensive an option.

In our courses, we use Phoenix++ to cover salient features of MapReduce in one lesson period. In the context of our year-long capstone experiences, students have been able to build novel MapReduce applications using the framework, and publish and present their results at national conferences.

To facilitate the adoption of Phoenix++ in the curricula at other institutions, we design a free module that is publicly accessible from the CSinParallel website. Beyond parallel computing and computer organization, we believe our module would be appropriate in other courses, such as computer architecture, operating systems, and programming languages.

Our experiences with Phoenix++ suggest that it is a very effective tool for introducing students to parallel computing. In the future, we hope to conduct a multi-institutional study comparing the effectiveness of Phoenix++ and Hadoop. We also hope to partner with the creators of WebMapReduce [7] to help create a Phoenix++ backend for the platform. We believe this will increase the ability for WebMapReduce to be deployed in a greater number of CS1 classrooms, as Phoenix++ will enable students to use WebMapReduce on multicore workstations.

ACKNOWLEDGEMENTS

We would like to sincerely thank Dr. Elizabeth Shoop for her assistance in deploying the Phoenix++ module on the CSinParallel website. The opinions expressed

in this paper are solely of the author and do not necessarily reflect those of the U.S. Military Academy, the U.S. Army, or the Department of Defense.

REFERENCES

[1] Brown, R., Shoop E., CSinParallel and synergy for rapid incremental addition of PDC into CS curricula. In *Proceedings of the 2012 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, 1329-1334, 2012.

[2] Brown, R., Shoop, E., Adams J., CSinParallel: Parallel computing in the CS curriculum. Internet Website, last accessed, August 2016. http://csinparallel.org/

[3] Brown, R., Hadoop at home: large-scale computing at a small college. *ACM SIGCSE Bulletin*, 41(1), 106-110, 2009.

[4] Dahlberg, T., Barnes, T., Rorrer, A., Powell, E., Cairco, L. Improving retention and graduate recruitment through immersive research experiences for undergraduates. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education* (SIGCSE '08). 2008.

[5] Dean J., Ghemawat, S., MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1), 107-113, 2008.

[6] Ernst D.J., Stevenson D.E., Concurrent CS: preparing students for a multicore world. *ACM SIGCSE Bulletin*, 40, 230-234, 2008.

[7] Garrity, P., Yates, T., Brown, R., Shoop, E., WebMapReduce: an accessible and adaptable tool for teaching map-reduce computing. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, 183-188, 2011.

[8] Iqbal F., Hadjidj R, Fung B.C., et. al. A novel approach of mining write-prints for authorship attribution in e-mail forensics. *Digital Investigation*, 5, S42-51, 2008.

[9] Final report of the joint ACM/IEEE-CS task force on computing curricula 2013 for computer science, 2013.

[10] Kimball, A., Michels-Slettvet, S., Bisciglia. C., Cluster computing for web-scale data processing. *ACM SIGCSE Bulletin*, 40(1), 116-120, 2008.

[11] Läufer, K., Sekharan, C.N., Thiruvathukal, G.K., PDC modules for every level: A comprehensive model for incorporating PDC topics into the existing undergraduate curriculum. In *NSF/TCPP Workshop on Parallel and Distributed Computing Education (EduPar'11)*, 2011.

[12] Lin, J., Exploring large-data issues in the curriculum: A case study with MapReduce. In *Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics*, pages 54-61, 2008.

[13] Ngo, L.B., Duy, E.B., Apon, A.W., Teaching HDFS/MapReduce systems concepts to undergraduates. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, 1114-1121, 2014.

[14] Ortiz-Ubarri J., Arce-Nazario R., Modules to teach parallel computing using Python and the LittleFe cluster. *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC'13)*, 2013. Poster.

[15] Rabkin, A.S., Reiss, C., Katz, R., Patterson, D., Experiences teaching MapReduce in the cloud. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, (12), 601-606, 2012.

[16] Raicu, D.S., Furst, J.D. Enhancing undergraduate education: a REU model for interdisciplinary research. In *Proceedings of the 40th ACM technical symposium on Computer science education (SIGCSE '09)*. 2009.

[17] Sutter, H. “The free lunch is over: A fundamental turn toward concurrency in software.” *Dr. Dobb's journal* 30(3), 202-210. 2005.

[18] Talbot, J., Yoo, R.M., Kozyrakis, C., Phoenix++: modular MapReduce for shared-memory systems. In *Proceedings of the second international workshop on MapReduce and its applications*, 9-16, 2011.

[19] White, T., Hadoop: *The definitive guide*, O'Reilly Media, Inc., 2012.