Teaching Parallel and Distributed Computing Using a Cluster Computing Portal

Hong Lin
Department of Computer and Mathematical Sciences
University of Houston-Downtown
Houston, USA
linh@uhd.edu

Abstract— **This paper presents the establishment of cluster computing lab at a minority serving institution that aims to provide computing resources to support undergraduate computer science curriculum. We present a case study of using this platform to teach parallel and distributed computing topics in the operating system course. The evaluation of the teaching effectiveness is presented thereafter.**

Keywords-component; lab platform, cluster computing, user portal, undergraduate education, parallel computing

I. INTRODUCTION

In recent years, cluster computing proved itself to be a very efficient way to serve as a high performance computing platform with low cost. It also addresses the flexibility and scalability issues by allowing the upgrading of individual components in the cluster. With the support from an NSF grant, UHD established a computing cluster to meet educational and research needs. The cluster has four segments, composed of different types of computers acquired in different times. A master server node connects all the segments together to form a bigger cluster or a grid. The portal on the server allows remote access to the computing resources. The portal is accessible from the webpage at http://grid.uhd.edu [1-2].

The intent of the presented project is to develop and implement a web interface for the cluster job distribution and multi-platform source handling. The goal of the project is to allow users of the computational cluster to remotely manage their files, and perform application deployment, execution and cluster job scheduling on the cluster. The means of remote access to the cluster resources are provided by the use of a web browser. The project should serve as framework allowing limited platform processing, compilation and execution of C, C++, and Java source code, and its effective distribution on the cluster. The framework can then serve for further expansion and development of modules to handle additional programming languages and platforms.

Needless to say, the project has an expected impact on utilization of the computational resources provided by the cluster; the ability to access and use the computational resources among faculty members, research personnel, and students remotely from web browser tremendously increases the access to harness the computational power of the cluster. Further, an anticipated outcome of the project is increased student participation in research and increased interest in fields like parallel and distributed programming (PDC), grid and high-performance computing [3].

Among the uses of this multi-purpose platform, we present the use of this platform in teaching PDC topics in an undergraduate computer science curriculum. With the user portal of the cluster, students are able to access the computing resources online from their classrooms and experiment on various PDC topics. We present the use of this platform in teaching PDC topics in an operating systems course and evaluate the teaching effectiveness of the relevant course modules.

The organization of this paper is as follows: We will firstly present the computing cluster and the user portal. Secondly, we present the course modules designed to teach PDC topics and the teaching activities in an operating system course. We conclude thereafter.

II. DESIGN OF THE CLUSTER COMPUTING PORTAL

The cluster has four segments, each having sixteen slave nodes and a master node. A master server node connects all the clusters together to form a bigger cluster or a grid. The portal on the server allows remote access to the computing resources. The portal is accessible from the webpage at http://grid.uhd.edu.

Intrinsically, the cluster portal allows the access to compilers and executors and the means to distribute content on the cluster. Through the web interface, a user is able to upload the source code (or any other content) of an application to the directory structure nested in their home directory, compile it, and/or execute it on one or multiple nodes of the cluster. The latter execution is dependent on whether the target application is sequential or parallel in nature. The web interface serves as a remote access tool to the backend workhorse. It takes the needed information from a user, it then creates a compilation and/or executor object, which in turn upon success contacts a job distributor to allocate resources on the cluster and finally dispatch the job onto those resources. The web interface allows the user to monitor the standard streams, and even provide input, if so the target application requires it.

The requirements of the application were set to:

- provide means of user distinction, through the method of user authentication
- provide intuitive navigation through the application
- provide facilities for file manipulation, like directory browsing, file uploading and downloading
- provide client access to the functionality delivered of the backend application, i.e. the compilation, and execution of user programs on the cluster.

In the following section, we present the use of this platform to support teaching PDC topics in an operating systems course.

III. Teaching Parallel Computing Topics Using the Platform

We partnered with NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing in a project to incorporate PDC topics in undergraduate curriculum. The project involves national and international participants to enhance computer science curricula in their institutions by adopting advanced PDC topics in relevant courses. Since PDC now permeates most computing activities, "it is no longer sufficient for even basic programmers to acquire only the traditional, conventional sequential programming skills. The preceding trends point to the need for imparting a broad-based skill set in PDC technology at various levels in the educational fabric woven by Computer Science (CS) and Computer Engineering (CE) programs as well as related computational disciplines." Therefore, it is necessary for the stakeholder experts to work together and periodically provide guidance on restructuring standard curricula across various courses and modules related to PDC. "A primary benefit would be for CS/CE students and their instructors to receive periodic guidelines that identify aspects of PDC that are important to cover, and suggested specific core courses in which their coverage might find an appropriate context." [4]

We proposed the integration of the TCPP Core Curriculum into the Operating System course offered in Spring 2012 at the University of Houston-Downtown. The Operating System course (CS 4315) is a required undergraduate course. We believe the experience in teaching PDC topics in the operating system course will benefit teaching PDC topics in other courses.

The operating system course focuses on traditional systems utilizing a single processor and the issues that arise when multiple processes must share the processor, memory and I/O resources. The course focuses on three main operating system functions: device management, process management, and storage management (main memory and file system management). Programming assignments are carried out on UNIX. After the evaluation of the possibility for adopting PDC topics in this course, it was determined that it can cover some of the architecture and programming topics of the TCPP core, in addition to the PDC topics already covered by the current syllabus. The detailed integration plan is as follows.

A. *Integration Plan*

The Computer Organization module covers pipeline, SIMD, and MIMD. Synchronization tools such as spin lock and test-and-set instructions were added. Deadlock problems were discussed. Some Message Passing topics such as topology, latency, and routing were introduced as well.

The Operating System Organization module covers multithreading. The Simultaneous Multi-Threading were studied and the differences between SMT and multicore addressed.

On the Memory Management module, Topics about Memory Hierarchy, including Cache, Consistency, Coherence and Impact on Software were added. In addition, UMA and NUMA systems were also studied.

Likewise, some of the Programming topics in TCPP curriculum were already covered by CS 4315. These include Shared memory and Task/thread spawning. Additional course modules were designed to cover Distributed memory, Hybrid, SPMD, and Data parallel topics. Since this is not a parallel computing course, the programming components were handled in the form of closed labs. Sample codes were provided to students to skip learning language constructs specific to parallel programming. This approach especially applies to topics in shared memory and message-passing programming.

Evaluation was done in three forms. Firstly, the difficulty to integrate the new topics into the existing course was examined. For example, what topics in the existing version of the course have to be upgraded to accommodate new topics. This part was the front-end evaluation. Because the current setting of the course concentrated more on concepts around traditional single processor system than practical issues arisen from modern multi-core systems, interweaving PDC topics into the existing uni-processor based topics must be done in a way that traditional topics are still covered and it proceeding of the course goes naturally into the multi-core topics.

Secondly, feedbacks were collected from the students at the beginning and the end of the semester. UHD is featured by small classes (normally around 15-20 students per class) and a strong interactive teaching mode. This allows the instructor to help students get onto the new materials by tutoring in closed labs. Student feedbacks were collected during the class about whether the topics were interesting, the depth of the topics, and the pace of the class.

Finally, TCPP topics became inherent part of the course content, closed labs, and exams. It was expected that student grades on the labs and exams would reflect the learning effectiveness with the new course settings.

B. *Design of the Course Modules*

In addition to existing course modules that are related to PDC topics, e.g., synchronization and deadlock, more materials were included in the course to cover the aforementioned topics. We tried to supplement all PDC

topics with hands-on experiences. Closed labs were added to allow the students to experiment on the concepts. With the web portal of the cluster computing lab, students have the access to the Linux computer cluster with duo-core and quad-core machines and a GPU machine. This allows the experiments on a spectrum of PDC topics. The following is a list of the labs used.

1) Multicore Lab 1 Synchronization with Java – Using Java Synchronized method to ensure timely access to a counter shared by two threads.

   Students need to find out why synchronization is necessary for two threads to ensure correct reading of the counter. A pre-written Java program was given to the students with the code for synchronization missing. Students experimented with the given erroneous program and checked the incorrect output of the program. Student were required to fix the problem and turn in a correct version of the program.

2) Multicore Lab 2 Spin Lock and Cache Coherence – Simulate cache invalidation and updating using TAS Lock.

   This lab was designed to experiment on the use of the TAS lock mechanism to solve the cache coherence problem. A Java program with multiple threads was created to facilitate the experiment. A shared variable was used to simulate the main copy of the shared data in the main memory and each thread has a local copy of the shared variable, which represents the copy in the local cache. TAS lock methods were provided in a class package. Students need to use the TAS lock methods to correctly implement the cache invalidation and update operations.

3) Multicore Lab 3 UMA and NUMA Access – Using Pthread and MPI to simulate and evaluate the access times to local shared memory and the access times to remote memory.

   Uniform Memory Access (UMA) and Non-Uniform Memory Access (NUMA) are two types of memory systems used in SIMD and MIMD machines. With a computer cluster that accommodates multi-core machines, both UMA and NUMA modes are used in the system: UMA mode is used among threads that run on multi-cores of the same processor, while NUMA mode is used when a process needs to read data located in a remote processor. This lab allows the students to measure the timing features of UMA and NUMA read/write operations and understanding the differences between UMA and NUMA.

4) Lab for Process and Thread Management (Chapter 6) – Using fork() routine and Pthread to create a multi-process program and a multithreading program to solve a producer-consumer problem.

   In this lab, students are asked to write a program that creates two threads, one reading a text file that contains a series of none-zero numbers ended by a special number -1 and stores the numbers, including the ending -1, into an array, while the other thread write the numbers in the array to a newly created text file in the same directory. Synchronization must be imposed to make sure the thread that writes the numbers to the file comes back to read the array until -1 is encountered, if the writing is faster than the reading. In addition, students need to figure out how to tell the writing thread where is the last item that has been read into the array when the reading is still ongoing.

5) Lab for Basic Synchronization Methods (Chapter 8) – Using Pthread to implement a banking account program that allows concurrent deposit/withdraw activities.

   This lab requires the students to create a program that allows asynchronous read and write operations. Students need to go through a certain number of steps to create a scenario in which asynchronous read and write operations lead to incorrect ending balance on the bank account, and then develop a solution to ensure orderly execution of read and write operations so that the ending balance is correct. The following are the steps.

   i. Write a program to simulate deposit/withdraw activities on a banking account: Initialize the beginning balance to 1 million, withdraw 600 thousands, and then deposit 500 thousands. Finally print out the ending balance. What is the output?
   ii. Write two functions, one for withdraw, the other for deposit. Declare the balance as a global variable. Both withdraw and deposit functions have one parameter, which represent the amount to withdraw or deposit. Then call the two functions from main().
   iii. Modify the withdraw and the deposit functions to make them deduct the balance and add to the balance one dollar at a time, respectively. Therefore, to withdraw 600 thousands, for example, the withdraw function has to execute a loop with 600 thousand iterations. The deposit function works in the same way, in the reverse direction.
   iv. Create two Posix threads in main(), which call the withdraw and the deposit function respectively. You can create these two threads in any order. However, before you create the second thread, use pthread_join() to wait for the first thread to terminate. What is the ending balance?
   v. Move the calls to pthread_join() function after the creation of both pthreads. Run the program several times. Do you see different result?

vi. Use pthread-mutex_lock() and pthread-mutex_unlock() functions to ensure mutual exclusion between the two pthreads. Is the ending balance correct now?

6) Lab for Deadlock (Chapter 10) – Using Pthread to experiment the dining philosophers' problem and the solution.

This lab asks the students to implement a program for the dining philosophers' problem. The program should use five Pthreads to simulate five philosophers and declare an array of five semaphores to represent five forks. Two versions of the program are to be written to experiment on a method for deadlock avoidance.

Firstly, write the program without considering deadlock. The program should create five Pthreads that run the same procedure for a philosopher to try to acquire the two forks for eating in the same order. Repeatedly run the program to see that deadlock occurs when the philosophers run to a cyclic hold and wait situation. Deadlock situation can be observed by making each philosopher print a message at every event (viz., request, allocation, release). The message should show the philosopher number and the relevant fork number.

Then, write another program that makes Philosopher 4 request the forks in the other order so that the cyclic hold and wait condition is prevented. Observe that the deadlock will never occur.

7) Programming Assignment 3 Bounded Buffer Problem – Using Mutex Lock and Semaphore to solve a bounded buffer problem.

In this assignment, students are provided with a program of the producer-consumer problem using threads. It uses POSIX *mutex locks (or mutexes)* to support synchronization between the producer and consumer threads. The program attempts to solve the bounded-buffer problem for 1 producer and 1 consumer, but is not a correct solution. Students are required to read the program carefully, and provide a scenario in which it produces an incorrect answer, and then execute the program and verify the scenario provided. The students are then required to modify the program so that it solves the *bounded-buffer problem* using (a) mutex locks, (b) semaphores.

C. *Learning Outcomes Assessment:*

The following analysis aims to compare students' performing in meeting the learning objectives of the PDC topics. Assessment was done in Spring 2012. The size of the class was 19. We analyze the programming assignments (labs and projects) and exams separately. The following table shows the passing rate of the programming assignments. The first three labs are add-ons labs for PDC topics and the others are existing assignments that are related to PDC topics. There is no significant difference between these two sets of the assignments, except that the 3rd add-on lab (UMA and NUMA Access) has a lower passing rate. The reason might be due to its difficulty. It involves the use of both MPI and Pthread.

Multicore Hands-on Experience	**Passing rate***
Multicore Lab 1 Synchronization with Java	50%
Multicore Lab 2 Spin Lock and Cache Coherence	67%
Multicore Lab 3 UMA and NUMA Access	39%
Lab for Process and Thread Management	44%
Lab for Basic Synchronization Methods	61%
Lab for Deadlock	50%
Programming Assignment 3 Bounded Buffer Problem	56%

Note*: Passing rate is the percentage of the students who have scored at least 70 out of 100

The following table shows the passing rate on multicore questions on the midterm and the final exams. We show the passing rate among all students and the passing rate among only the students who eventually received a passing grade (C or up) for the course. The passing rate among all students is low, which indicates the difficulty of the multicore topics. The passing rate among passing students varied drastically, bringing difficulty to reach a conclusion. However, both passing rates indicated improvements from the students along the progress of the course.

Exams	**Passing rate 1***	**Passing rate 2****
Midterm	17%	33%
Final	22%	80%

Note*: Passing rate 1 is the percentage of the students who have scored at least 70 out of 100 among the entire class.
Note**: Passing rate 2 is the percentage of the students who have scored at least 70 out of 100 among the students who received a passing grade (C or up) for the course.

One of the students commented in the course survey:

> *"I find the topic interesting as well as useful when programming applications for multi-core devices, but I personally find the topic to be complex and too much to cover in an Introduction class. I understand the need to push out as much information as possible, but from a students perspective that is still very new learning about multi-core processing and the CS curriculum being so broad, I would of liked to gone through more examples during class time to get a better understanding and take more time to learn each topic. Also programming some labs and projects in different languages is extremely hard when*

you don't know the programming language that well. …"

This comment should reflect the difficulty in incorporating PDC topics into a CS course due to the wide coverage of PDC topics and the use of various tools in experiments.

An entrance survey and an exit survey were done at the beginning and the end of the semester. Similar questions were asked in two surveys.

1. How much do you think you know about PDC technology?
    1) A lot
    2) Somewhat
    3) Only a little
    4) Not at all
1. Do you think the traditional operating systems course, which is based on single processor systems, still provide sufficient knowledge in the related domain of computer science?
    1) Yes
    2) Not sure
    3) No
2. What do you think about how multi-core topics are relevant in computer science curriculum at large?
    1) It is highly important in computer science curriculum.
    2) It is relevant but optional.
    3) It is not important.
3. What is your point of view about the usefulness of skills in multi-core programming in career development in industry and/or graduate studies?
    1) Multi-core programming skills are very useful.
    2) Multi-core programming skills are somewhat useful.
    3) Multi-core programming skills are not useful.
4. In a scale of 1 to 5, rate your knowledge about message-passing computing systems with 1 being the least knowledge and 5 being the full knowledge?
    1) 1
    2) 2
    3) 3
    4) 4
    5) 5
5. In a scale of 1 to 5, rate your knowledge about multi-threading using Pthread with 1 being the least knowledge and 5 being the full knowledge?
    1) 1
    2) 2
    3) 3
    4) 4
    5) 5

The following table shows the comparison of the student responses in two surveys.

Question #	Mean on Entrance Survey	Mean on Exit Survey
1	3.00	2.00
2	2.56	2.38
3	1.33	1.38
4	1.44	1.38
5	2.00	2.75
6	2.22	3.00

Question 1: How much do you think you know about PDC technology? The mean of the self scoring goes from 3.00 (only a little) to 2.00 (somewhat).

Question 2: Do you think the traditional operating systems course, which is based on single processor systems, still provide sufficient knowledge in the related domain of computer science? The mean goes from 2.56 (between "Not sure" and "No") to 2.38, which indicates that while the students tend to agree that traditional single processor based OS course is no longer sufficient, but the course didn't seem to help them reaffirm this opinion.

Question 3: What do you think about how multi-core topics are relevant in computer science curriculum at large? The mean goes from 1.33 (between "highly important" and "relevant but optional") to 1.38. Again, the students tend to believe multi-core topics are highly important but the course didn't seem to make them believe more.

Question 4: What is your point of view about the usefulness of skills in multi-core programming in career development in industry and/or graduate studies? The mean goes from 1.44 (between "very useful" and "somewhat useful") to 1.38. They seem to believe more the skills in multi-core programming are useful through the study of the course.

While the students' responses to Question 3 and 4 are a little bit contradictory, the difference in the entrance survey and the exit survey might be due to randomness when taking into account that the means in two surveys are very close.

Question 5: In a scale of 1 to 5, rate your knowledge about message-passing computing systems with 1 being the least knowledge and 5 being the full knowledge. The mean goes from 2.00 to 2.75, indicating some improvement.

Question 6: In a scale of 1 to 5, rate your knowledge about multi-threading using Pthread with 1 being the least knowledge and 5 being the full knowledge. The mean goes from 2.22 to 3.00, again indicating some improvement.

The above student self-evaluation shows that teaching PDC topics in the OS course improves student's awareness and confidence in PDC technology to some limited extend.

In general, the evaluation of teaching PDC topics in the operating system course is positive and students accept that including those topics in the course is necessary and will help them succeed in their career path in computer technology. It is also deemed necessary to continue teaching PDC topics in a wider range of CS courses and adjusting the course materials to fit the needs of each individual course.

IV. Concluding Remarks

We present a project that implemented a web interface for a computing cluster at a small minority serving institution. We implemented a web application to allow access to cluster computational power, and to utilize and further develop the backend for compilation, execution and distribution of sequential, parallel and interactive applications on the clusters available resources. Further, the project incorporated a file browser allowing the download, and upload of multiple files, their editing and basic file manipulations like copy, move, rename.

Facilitated by the cluster computing portal, we are able to renovate computer science curriculum by incorporating topics in parallel and distributed computing in various CS courses. We participated in an early adopter program sponsored by NSF/IEEE that aims to promote teaching PDC topics in undergraduate CS curricula. We present a case study of using an operating system course as the adopting course to teaching a set of selected PDC topics. The evaluation of teaching PDC topics indicated positive learning outcomes and student feedback.

Our project sets an example of establishing necessary computing platforms in an undergraduate institution to support experiments on newly emerged computer architectures such as multi-core computers, and incorporating PDC topics in CS curriculum to provide students with knowledge and skills on current state-of-art technologies. The students are able to acquire indispensible experience and knowledge in all of the utilized technologies and programming principles. This demonstrates an unlimited potential of improving learning outcomes in an undergraduate education institution.

Acknowledgment

The teaching of PDC topics in the operating system course was partially supported by an early adopter program of the NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing. Former UHD student Borislav Ganov designed the web portal for the cluster computing lab.

References

[1] H. Lin, O. Sirisaengtaksin, and P. Chen, “The Establishment of a Cluster Computing Environment at a Small Institution,“ Nagib Callaos, Kamran Eshraghian, Michita Imai, William Lesso, C. Dale Zinn (Eds.), Proceedings of the 14th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI 2010), Orlando, Florida, June 29 - July 2, 2010, Volume I, pp. 161-166.

[2] H. Lin, O. Sirisaengtaksin, and P. Chen, “A Cluster Computing Environment at a Small Institution to Support Faculty/Student Projects,“ Proceedings of the South Central Conference/ Journal of Computing in Small Colleges, Volume 25, Number 4, April 2010, pp. 30-36.

[3] H. Lin, B. Ganov, J. Kemp, P. Gilbert, “A Grid Computing Environment for Undergraduate Research,” Proceedings of IEEE 3rd International Conference on Software Engineering and Service Science (ICSESS 2012), June 22-24, 2012, Beijing, China, pp. 16-19.

[4] S.K. Prasad, et al, NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing - Core Topics for Undergraduates, Version I, Dec 2012, Website: http://www.cs.gsu.edu/~tcpp/curriculum/index.php