Using Microlabs to Teach Modern Distributed Computing

Barry L. Kurtz, Rahman Tashakkori, John Helfert, Michael Sage
Appalachian State University
Boone, NC 28608
828-262-2370
{BLK,RT}@cs.appstate.edu

ABSTRACT
Closed labs have become a common feature in computer science education because they provide hands-on experience in a supervised setting. Microlabs extend this approach into the lecture format with very short hands-on activities in the “middle of” the lecture. This approach was developed for a modern distributed computing course that integrated all levels of parallelism (multicore, cluster, and grid) into a single course and required a substantial course project that used all these levels. After presenting the current status of our activities, we will discuss future directions that include stand-alone distributed computing modules and a distributed computing repository. This work is supported, in part, by a National Science Foundation grant.

Categories and Subject Descriptors
K.3.2 [Computer and Information Science Education] Computer Science Education, Curriculum

General Terms
Algorithms, Performance, Human Factors

Keywords
Distributed Computing, Computer Science Labs, Multicore Computing, Cluster Computing, Grid Computing

1. INTRODUCTION
We first review some historical trends in teaching distributed computing followed by a discussion on the use of labs in computer science education.

1.1 Historical Trends in Teaching Distributed Computing
In the 1980s and the early 1990s instruction in high performance computing emphasized computer architectures for parallelism. Topics included Flynn’s taxonomy (SIMD, MIMD, etc.) and very expensive, special purpose computers that supported high performance computing. By the mid-1990s it became evident that special purpose machines could not be justified on a cost basis and a workstation “farm” was a much more cost effective approach to distributed computing within the reach of ordinary universities. Courses in cluster computing, at first using PVM [11] and then MPI [8], emphasized a message passing paradigm in a distributed memory system. Threads are lightweight processes that can distribute work over multiple processors or cores. Modern object-oriented languages, such as Java and C#, support threads directly. In other environments, libraries, such as PThreads [10] and OpenMP [9], supported shared memory, thread programming. As multicore chips become the norm rather than the exception, there is renewed interest in shared memory approaches to distributed computing.

Grid computing emerged in the early 21st century largely in response to the needs of businesses to tie together their diverse computing resources. Globus [6] became a de facto standard in the academic community because it was a freeware application; however, due to its steep learning curve and lack of support, it has not been a commercial success. Courses on grid computing have emerged in the undergraduate curriculum [14]. The Common Object Request Broker Architecture (CORBA) was an early attempt at using heterogeneous computers hooked to a network transparently, but it has been replaced by the widespread acceptance of web services based on the Simple Object Access Protocol (SOAP) model. The Internet is too slow to support high performance computing directly, so grid computing applications often stress data parallelism. For some “embarrassingly parallel” applications that don’t require constant message passing, a worldwide network of computers “at home” has proved to be very effective; perhaps the two best known examples are set@home [12] and folding@home [5]. Software frameworks, such as BOINC [2], have been developed that allow this approach to be extended into computer science education.

In this paper we use the term “distributed computing” in an umbrella sense to include all forms of parallelism ranging from shared memory approaches (threads, OpenMP) and distributed memory approaches (clusters, web services). Many of the example labs are classical; our use of the word “modern” refers to the integration of all these levels in a single course and a single course project.

1.2 Use of Labs to Teach Computer Science
The traditional approach to computer science education is to have students complete programming assignments outside of class. Open lab environments, perhaps with a lab monitor to provide general assistance, are provided so students can access computing resources. The closed lab environment is modeled after Chemistry or Physics.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGCSE’10, March 10–13, 2010, Milwaukee, Wisconsin, USA.
Copyright 2010 ACM 978-1-60558-885-8/10/03...$10.00.
labs where students meet once or twice a week at a scheduled time to complete programming activities under supervised conditions. These closed labs are common in the introductory CS1 and CS2 courses. Students are often required to complete a lengthy sequence of activities with a lot of hand holding to guarantee early success in computer science. These labs are usually supplemented with traditional programming assignments to be completed outside of scheduled class time.

A variation of the closed lab is a testing environment where students have to complete small program components under strictly monitored and possibly timed conditions. Ken Bowles [3] at the University of California San Diego (UCSD) introduced the use of drills and programming quizzes to support instruction in UCSD Pascal in the CS1 course. Kurtz has used this approach at three universities: the University of California Irvine, New Mexico State University, and Appalachian State University [4].

An increasingly popular approach to computer science instruction is to hold the entire class, both lecture and lab, in a classroom/labatory environment. In this environment there is usually a projection system to support lecture, the seats are arranged in a forward-facing position to make viewing the lecture convenient, and every student has immediate access to a computer. Typical classroom environments may be limited to 20-30 students and does not support large lecture formats. The instructor’s computer is often running software that allows monitoring, displaying and/or taking control of student screens. The microlabs described in this paper were used in this environment.

2. THE STRUCTURE OF MICROLABS

2.1 Characteristics of a Microlab

Our microlab approach is a form of active learning and in this sense is nothing new. However the consistent application of this approach throughout a course integrating distributed and parallel computing is unique. Microlabs are designed to be completed in the middle of a normal lecture, thus they are short duration and require a limited amount of coding. A 5 minute microlab may require only a few lines of new code to be put into a skeleton program while a 15 minute microlab might involve writing one or more methods. Students are given the opportunity to work in pairs but our experience indicates that most students prefer to work individually. Not everyone will finish a microlab during the allotted time period, but these students are allowed to work on the microlabs after class and check them off the next class period. Initially students were hesitant to give up trying and go back to lecture mode, but once they realized the ground rules after the first few microlabs, they were willing to switch back to lecture mode even when their microlab wasn’t completed.

Development of microlabs does require extensive work by the instructor and lab assistants (if available). An entire framework must be provided to incorporate the student modifications into a complete program and testing environment. The student must correct all compile errors on his/her own. This is relatively easy for a programming environment a student is familiar with, but when using a new language (e.g., C) and a new set of library routines (e.g., MPI), this can be a challenge. Developing carefully chosen test cases assists students in finding any logical errors once the program compiles and executes. In our development environment the instructor and lab assistant were available to help with problems in a small class, but this approach may not scale well to a lecture hall environment with a large number of students. We will discuss these issues in more detail in section 2.3.

2.2 How Microlabs Differ from Closed Labs

Microlabs differ from typical closed labs in several ways. First, they occur during a lecture and are intended to give students some immediate hands-on experience with the lecture topic. They are much shorter and more focused than a closed lab activity. After the brief interlude, the class must continue on in lecture mode. As explained above, students who have not finished the activity must be willing to suspend their efforts until after the class period. The microlabs also require a more substantial support environment than normal closed lab activities in order to be completed in a short period of time.

2.3 The Feasibility of Using Microlabs in a Large Lecture Environment

The mixture of lecture and lab activities is most easily carried out in a lab environment that is also set up as a classroom (projector, forward-facing seats, etc.). What about a pure lecture environment with no installed equipment and a large class of students? Won’t the microlabs just be demonstrations by the instructor or work by a single student asked to come work in front of the class? Five years ago these might have been the only choices, but that is not true today. A majority of students, particularly computer science majors, own laptop computers that can connect wirelessly to a local area network. Students can be asked to work with a classmate if they don’t have a laptop with them. This approach can establish a virtual network of machines even in a large lecture hall.

The next problem to be solved is to insure the students use their computers in the manner intended by the instructor. Fortunately commercial companies have developed software, such as DyKnow Monitor or SmartSync, to solve these management problems. Not only can instructors observe what students are doing, they can also block access to undesired activities.

A final problem is to provide feedback to students who are working on a microlab. As discussed in section 1.2, there is a long history of automated testing environment to administer, grade, and provide feedback to students working on programming quizzes. We are adapting this approach to provide similar support for microlabs in a distributed computing environment. This is discussed in section 5.2.

3.0 THE DISTRIBUTED COMPUTING COURSE

3.1 Multicore Computing

The first topic covered in Multicore Computing was the use of threads in Java. There were ten lecture presentations where topics ranged from mutual exclusion and semaphores to message passing using array blocking queues.

There were ten microlabs for threads in Java, as listed in Table 1. As a detailed example, in ML05 students had to complete a class called Semaphore that used the lock and condition variables available in Java 1.5 to simulate the behavior of a counting semaphore. The constructor and the methods for P and V in the class are initially empty. The skeleton program contains a simple application that runs two threads each incrementing a shared global variable fifty times. With a non-functioning semaphore class the results varied from 50 to 100. Once the students get the Semaphore
simulation running correctly, the result is always 100. The steps students had to complete were:

- complete the constructor that stores the number of allowed permits, creates a re-entrant lock and a local condition variable called nonzero
- complete the P method that obtains the lock, waits until the semaphore count is greater than zero, decrements the semaphore count, and releases the lock
- complete the V method that obtains the lock, signals the condition variable nonzero if the semaphore count is zero, increments the semaphore count, and releases the lock

Most students completed this microlab in 20 minutes.

<table>
<thead>
<tr>
<th>Lab</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>ML01</td>
<td>Taking Turns</td>
</tr>
<tr>
<td>ML02</td>
<td>Dekker’s Algorithm</td>
</tr>
<tr>
<td>ML03</td>
<td>Dinning Philosopher</td>
</tr>
<tr>
<td>ML04</td>
<td>Sleeping Barbers</td>
</tr>
<tr>
<td>ML05</td>
<td>Semaphore Simulation with a Monitor</td>
</tr>
<tr>
<td>ML06</td>
<td>Bounded Buffer</td>
</tr>
<tr>
<td>ML07</td>
<td>Prime Number Sieve Using Message Passing</td>
</tr>
<tr>
<td>ML08</td>
<td>Bounded Buffer Using Message Passing</td>
</tr>
<tr>
<td>ML09</td>
<td>Laplace Grid Using Message Passing</td>
</tr>
<tr>
<td>ML10</td>
<td>Laplace Grid Using Shared Memory</td>
</tr>
</tbody>
</table>

There were five programming assignments for this portion of the course. These assignments ranged from P01 (The Bakery Algorithm) through P05 (The Game of Life).

The next approach for shared memory threads was using OpenMP [9] in C. Our students are Java literate and, at this stage, only have limited experience in C. This section of the course provided a good introduction to C before more intensive experiences with MPI using C. There were six lecture topics: an introduction to OpenMP, clauses, directives, synchronization, handling global data, and runtime commands.

There were five microlabs for OpenMP, as listed in Table 2. In ML15 students were given a skeleton program that contained the following code fragment:

```c
#pragma omp for reduction(+:sum)
for (i=0; i < N; i++)
{
    sum = sum + (a[i]*b[i]);
}
```

The directions to the student state “if you run the program multiple times you will find that the sum printed out may differ between each execution. You need to introduce one more pragma to control the updating of sum so that the correct result is returned each time.” (Note: students were not told which pragma to use.)

The easiest solution was to put `#pragma omp atomic` before the assignment to sum; an alternative was to use the pragma named critical. This is an example of a very short microlab that most students completed in 5 minutes. There were three programming assignments using OpenMP: Matrix Multiplication, Calculating PI using Monte Carlo Techniques and Gray Scale Conversion.

<table>
<thead>
<tr>
<th>Table 2: Microlabs for OpenMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>ML11</td>
</tr>
<tr>
<td>ML12</td>
</tr>
<tr>
<td>ML13</td>
</tr>
<tr>
<td>ML14</td>
</tr>
<tr>
<td>ML15</td>
</tr>
</tbody>
</table>

Cluster Computing

The MPI lectures covered Basic Operations, Global Operations, an MPI Application, Derived Datatypes and Communicators. There were seven microlabs in MPI, as listed in Table 3.

<table>
<thead>
<tr>
<th>Table 3: Microlabs for MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>ML16</td>
</tr>
<tr>
<td>ML17</td>
</tr>
<tr>
<td>ML18</td>
</tr>
<tr>
<td>ML19</td>
</tr>
<tr>
<td>ML20</td>
</tr>
<tr>
<td>ML21</td>
</tr>
<tr>
<td>ML22</td>
</tr>
</tbody>
</table>

In ML17 students were told: “You are to calculate sine for the values 0.1, 0.3, 0.5, 0.7, 0.9, 1.1, 1.3, 1.5. Put these values in a vector and scatter them to eight processes. Have each process calculate the sine using the Taylor expansion.” An algorithm in high level psuedocode was provided. “Gather the results back into the main program and print out the values:

\[
\begin{align*}
\sin(0.1) & = 0.0998334 \\
\sin(0.3) & = 0.29552
\end{align*}
\]

Students had 15-20 minutes to complete this microlab; they were given a skeleton program that contained:

```c
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &id);
//your code goes here
if(id == 0)
    // the for loop to print results was given
```

Most students were able to complete this lab in the allotted time.

There were four programming assignments in MPI: Total Count of Images (later integrated into the course project), Numerical Integration using the Trapazoid Method, Sparse Matrices, and Fox's Algorithm for Matrix Multiplication.

Grid Computing and Web Services

These lectures covered Condor, Directed Acyclic Graph Manager, an Overview of Globus [6], and Web Services. There were only two microlabs, as listed in Table 4, that involve Condor, a workload scheduler that can be used under Globus.

<table>
<thead>
<tr>
<th>Table 4: Microlabs for Grid Computing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>ML23</td>
</tr>
<tr>
<td>ML24</td>
</tr>
</tbody>
</table>
At this time, students had been introduced to the course project involving image processing; programming assignments now became project assignments. There were two intermediate checkpoints on the course project: “Searching for a SubImage Using MPI” and “Searching for a SubImage at Multiple Websites Using Web Services.”

The Course Project
The project used several of the tools developed during the course. Students had to access two URLs that each contained a distinct collection of grayscale mammograms. Each website had four quadcore computers connected by a 1 Ghz LAN. These images were approximately 3000 x 4000 pixels. Students were given a variety of target images, ranging from 9 x 9 pixels to 17 x 17 pixels, and they had to search for the best match in the mammograms. Each web service was implemented in Java; it had to call a C program that used MPI over the local area network. The images had to be striped horizontally with overlapping rows (the number of overlapping rows was (t-1)/2 where t was the target image size). Recent versions of MPI detect whether MPI is running on a multicore machine and automatically distribute tasks over these cores. In the latest version of this project we have introduced some OpenMP pragmas to increase parallelism at the single machine level. Java threads were used by the client program that called the web services; there was a separate thread for each web service call. An initial web service call started the search process. A repeated web service call checked for results. If null was returned, it tried again after a short delay (useful processing at the client could have been done at this time, but for simplicity students put in sleep statements). Results for the best match for each image were passed back from the C program to the Java program, which in turn returned this non-null result from the web service (if the results were not known yet, null was returned). The printing process had to be put into a separate thread using a lock to insure that printing results was atomic and not intermixed.

3.5 Other Activities
There was one week remaining in class by the time all topics necessary to complete the project were finished. Students expressed interest in learning about tools available in the .NET development environment. There were two lectures: use of the Semaphore Class in C# and use of the Task Parallel Library [13] (similar to OpenMP in C). Two microlabs allowed students to experience hands-on activities, as shown in Table 5.

<table>
<thead>
<tr>
<th>Topic</th>
<th>SD</th>
<th>D</th>
<th>N</th>
<th>A</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threads</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>33.3</td>
<td>66.7</td>
</tr>
<tr>
<td>OpenMP</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>66.7</td>
<td>33.3</td>
</tr>
<tr>
<td>MPI</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>50.0</td>
<td>50.0</td>
</tr>
<tr>
<td>Web Services</td>
<td>0.0</td>
<td>0.0</td>
<td>33.3</td>
<td>33.3</td>
<td>33.3</td>
</tr>
</tbody>
</table>

There were no programming assignments associated with C# and the .NET environment.

EVALUATION OF THE MICROLAB APPROACH
A few microlabs had two separate activities that were graded independently. There were 31 microlab grades spread throughout the course; each component was graded on a satisfactory/not satisfactory basis. If a student did not finish a microlab during the allotted time, the student could complete it after class and check it off the next lecture period. Eighty four percent of students earned 90% or higher on the microlabs; sixteen percent of students earned between 80 to 90%. The most frequent cause of not completing a microlab was absenteeism during the lecture that introduced the lab activities.

Our first year grant activities were evaluated by an independent external evaluator [see Acknowledgements]. In the introduction of the report, the evaluator said:

The course developers followed the Chinese proverb, “I hear and I forget, I see and I remember, I do and I understand”, by allowing the students to be active learners during the class period through the use of the microlabs.

There was a sequence of questions associated with each of the major topics covered in the course. Table 6 shows the results for the question dealing with microlabs.

Table 6: Evaluation Results

It can be seen that students reacted very favorably to the use of microlabs. Only the web services microlabs, which did not relate to the course project, received any neutral evaluations. When asked in a free-form input text box the question “What was the best part of the course?” 50% of the students explicitly said the microlabs. The evaluator concluded:

Overall, the short-lecture/microlab approach designed by the grant participants is a strong pedagogical tool for teaching students distributed computing.

FUTURE DEVELOPMENTS
The Distributed Computing Modules
Our NSF grant is organized into two phases:

- During year one materials were developed as part of a complete course that was taught in the summer 2008 and the spring 2009
- During year two materials will be modularized and introduced into existing courses as separate learning modules

We strongly believe that distributed computing education has to permeate the entire computer science curriculum and not just be an upper division elective course. These modules will allow instructors to introduce distributed computing concepts into many courses in the curriculum. Here are some characteristics of these modules:

- They last 2 to 3 weeks
- They include presentations, microlabs, and program assignments
- There will be a wide variety of materials available that will allow instructors to pick and choose particular items to suit the course needs
Materials can be easily modified and expanded to allow the instructor to have a feeling of ownership. During the Fall 2009 semester we have tested the following modules:

- Java threads in our CS2 course; these results were mixed as the instructor did not want to use the more complex examples (he said students “were not ready yet”)
- MPI in our Programming Languages course plus one lecture on Java threads; these results appear to be more positive with students completing the assigned microlabs
- An expanded version of our .NET materials in our software engineering course; all levels of parallelism (threads, the task parallel library, MPI.NET, and web services) were integrated into a single programming environment using C# and used in the course project; trouble with setting up Windows Server and HPC Server limited the use of MPI.NET

Our external reviewer had the following comment about module development:

*It will be interesting to see how the course materials will be used as independent modules or lectures in lower-level computer science courses during the second year of this grant.*

### The Use of Microlabs

To make the microlabs practical in a large lecture section we need both a virtual wireless network with associated management software and an automated grading system. We have developed programming quiz grading systems in the past [4] but adaption of such systems to a distributed computing environment is challenging. We have started this work and here are some of the problems that need to be overcome:

- Extend the automatic compilation to handle multiple languages such as Java, C, and C#
- Check intermediate results returned by individual program components as well as the final result
- Allow for some non-determinism regarding the order results are reported
- Check for correct deployment of a web service even if the client program is not functioning properly
- Take into account any randomness used to generate test data

We have made a start on tackling these issues by introducing an automated grading program for MPI programs written in C during a summer workshop. Work is currently in progress during the second year of our grant.

### 5.3 The Distributed Computing Repository

Another component of our grant activities is the establishment of a distributed computing repository. It will not only contain materials directly developed as part of this grant but will also contain materials developed by researchers at other universities. Our initial focus has been on lecture presentations that can be searched using a keyword system. In addition to the lectures referenced in this paper, the repository also includes lectures from the following researchers:

- Dr. Barry Wilkinson, UNC Charlotte, Parallel Computing (ITCS 4145)
- Dr. Yaohang Li, North Carolina A&T, Distributed Systems (COMP750)
- Dr. Barry Wilkinson, UNC Charlotte, Grid Computing (ITCS 4146)

We will expand these materials to include programming assignments and projects associated with these courses. The repository can be accessed at:


Our external reviewer reacted positively to our initial version of the distributed computing repository as indicated by the following comment:

*In conclusion, the course materials developed for the integrated course and their initial inclusion in the distributed computing repository, started by the grant participants, is a wonderful beginning of an invaluable resource for instructors of distributed computing.*

### ACKNOWLEDGMENTS

The activities presented in this paper were partially supported by a NSF grant (0737355) entitled *Modern Distributed Computing Education* funded as a Phase 1 (Exploratory) proposal under the Course, Curriculum, and Laboratory Improvement (CCLI) program. The funding level was $150,000 for two years commencing July 2008. The graduate assistant position associated with this grant was funded 50% by the NSF and 50% by the Office of Research and Sponsored Programs at Appalachian State University.

The annual evaluation of our first year activities was performed by Joel Hollingsworth at Elon University.

### REFERENCES


[2] BOINC - BOINC.berkeley.edu


[12] Seti@home - http://setiathome.ssl.berkeley.edu/
