A Flipped Classroom Approach to Teaching Concurrency and Parallelism

Shirley V. Moore
Computer Science Department
University of Texas at El Paso
El Paso, TX 79968 USA
e-mail: svmoore@utep.edu

Steven R. Dunlop
Purdue NExT
Purdue University
West Lafayette, IN USA
e-mail: dunlops@purdue.edu

***Abstract*—Concurrency and parallelism are difficult concepts for computer science students, even those who are experienced programmers. The traditional teaching approach of lecturing and giving assignments to be done outside of class can leave students frustrated and unable to complete assignments successfully. The flipped classroom approach promotes collaborative learning during class meetings, with the goal of leading to a clearer understanding of concurrency and parallelism concepts and increased ability to apply and synthesize the knowledge. Outcomes-based assessment has shown improvement over the traditional lecture approach.**

Keywords- flipped classroom; concurrency; parallelism; outcomes-based assessment

I. INTRODUCTION

Research on effective teaching has shown that students learn more if they are actively engaged and if they can make connections between new concepts and what they already know [1]. Teaching concurrency and parallelism is challenging since most students have little or no experience with these concepts in prior computer science courses. The *flipped classroom* is a pedagogical model in which the typical lecture and homework elements of a course are reversed. Short video lectures are viewed by students at home before the class session, while in-class time is devoted to exercises and discussion [2]. We compared achievement of learning outcomes between offerings of a parallel programming course in two consecutive years using the traditional approach and the flipped classroom approach, respectively. We found that achievement of learning outcomes improved, but that some students disliked the flipped classroom approach.

II. RELATED WORK

A flipped classroom approach to teaching a high performance computing course in an interdisciplinary program is described in [3]. The course is about finite element methods and is project based. The main motivation for going to a flipped classroom format was for the instructors to be able to spend more time working with students individually on their projects. Reflective writing and journaling were used to connect the material from the video lectures with the students' projects. The authors found that they needed to model the reflective writing and journaling for the students, many of whom were from other cultures and were not used to this kind of exercise. They authors were not able to provide quantitative data due to the small sample size, but they reported that the approach was successful from the instructor's point and that the students found the new format more enjoyable.

Other literature on the flipped classroom approach in science and engineering courses reports on using this approach in first and second year undergraduate classes. A comparison of the traditional lecture approach and an approach called "deliberate practice" in teaching different sections of a large-enrollment second-semester undergraduate physics class is given in [4]. There were approximately 270 students in each section. Deliberate practice is similar to flipped classroom in that students do readings before class, there are no lectures, and students work on problem solving in small groups during class. The results showed increased student attendance, higher engagement, and twice the learning in the deliberate practice section.

Evaluation of the flipped classroom approach has shown both pros and cons [5]. The pros are that the students spend more time preparing for class and are more engaged during class. The cons are that preparing lectures and activities typically takes more time than preparing lectures and that merely assigning videos or reading does not guarantee that students will do the assignments or understand them. In our experiment with the flipped classroom approach, we addressed some of the cons by using a set of lectures prepared by internationally known experts and providing follow-up quizzes for the videos.

III. COURSE ORGANIZATION AND LEARNING OUTCOMES

The course Parallel and Concurrent Programming is offered during the spring semester at the University of Texas at El Paso (UTEP). It is listed as a graduate course with course number CS 5334, but it is cross-listed as a CS 4390 Special Topics course for undergraduates. In addition to a Computer Science Department that offers bachelors, masters, and PhD programs, UTEP has a graduate interdisciplinary Computational Science Program (CPS) that offered masters and PhD programs. The students enrolled in CS 5334 are typically a mix of graduate and upper-level undergraduate

computer science students, together with graduate CPS students. The prerequisites for the course are either 1) the undergraduate data structures (CS3) and computer architecture courses or 2) the graduate Introduction to Computational Science course.

Prior to spring 2015, CS 5334 had been offered in the typical lecture format. The instructor used a parallel programming textbook, lectured during class, and gave programming assignments to be done outside of class. Starting in spring 2015, the instructor switched to using a flipped classroom format. Students were required to watch video lectures and/or do reading assignments prior to class. An online quiz for each video lectures was also required to be completed before class. Class time was spent on group work on paper-and-pencil activities and programming exercises. Groups were formed in different ways, depending on the activity. Sometimes we assigned students to groups randomly by having them pick numbers. Other times we grouped more experienced students with less experienced students. For the final programming project, we allowed students to self-select their groups. Programming assignments similar to those in previous semesters were assigned to be completed outside of class. As in previous semesters, a midterm and a final examination were given. Graduate students were given some more challenging questions or tasks on assignments and exams.

In both spring of 2014 and spring of 2015, there were 16 students enrolled in the course. For spring of 2016, there are currently 19 students enrolled. In each year, there has been approximately an even mix of graduate computer science (CS) students vs. undergraduate CS students plus graduate computational science (CPS) students, with CPS graduate students having similar programming maturity to undergraduate CS students.

The video lectures and online quizzes for the spring 2015 course were provided by Purdue NExT (http://purduenext.purdue.edu/). The video lectures were by the author of the textbook used in the previous lecture-based offering of the course and the content and learning outcomes were basically the same. Some of the quizzes include questions that require running a concurrent or parallel program, where the program is either provided or the student is required to complete it. PurdueNext also provided a parallel computing system on which all the required software for the exercises was installed. The PurdueNext curriculum includes a course entitled Principles of Concurrency, taught by Suresh Jagannathan, Professor of Computer Science at Purdue University, and a course entitled Principles of Parallelism, taught by Ananth Grama, Director of the Computational Science and Engineering Program at Purdue University. Each course consists of five modules, with a module containing two to four lectures and associated quizzes, and the pace is intended to be one module per week. The two Purdue NExT courses were followed by teamwork on a parallel programming project. The separation of the course into the concurrency and parallelism modules allows the students to first acquire a sound foundation in correctness of concurrent programming before moving on to performance and speedup issues with parallel programming. Some of the principles of concurrency overlap with what is taught in the required CS operating system course, but since the CPS students are not required to take this course, and CS students may not have taken it prior to CS 5334, these concepts are included and/or reinforced in CS 5334.

The local UTEP instructor supplemented the PurdueNext materials with in-class activities intended to reinforce the concepts and engage the students. Because she noticed that a number of students did not seem to have watched the first few videos, she began subsequent classes by asking if there were questions on the video and following with a short quiz over the video to enforce the requirement to watch the videos. For the spring 2016 semester, she has developed viewing guides with questions for the students to ponder and answer as they watch the videos. She has also removed some of the videos that were not relevant to the learning outcomes for the course. In addition to class time, the instructor held twice-weekly office hours and several programming help sessions.

The UTEP Computer Science Department faculty develop detailed learning outcomes for all courses offered by the department. The outcomes are specified at level 1 (define), level 2 (apply, implement), and level 3 (evaluate, design, synthesize). In keeping with this policy, the CS 5334 instructor developed detailed learning outcomes for the course, drawing on previous semesters, the PurdueNext statement of outcomes, and the ACM/IEEE Model CS Curriculum [6]. The outcomes were then used to guide development of activities and assignments and to assess learning. The high-level outcomes are shown below. More detailed learning outcomes have been developed for specific class sessions and assignments.

Level 1
1.1 Distinguish between concurrency and parallelism.
1.2 Distinguish multiple sufficient programming constructs for synchronization that may be inter-implementable but have complementary advantages.
1.3 Explain the difference between processes and threads.
1.4 Describe a design technique for avoiding liveness failures in programs using multiple locks or semaphores.
1.5 Explain the concept of interconnection networks and characterize different approaches.
1.6 Discuss performance advantages of multithreading offered in an architecture along with the factors that make maximum performance difficult to achieve.
1.7 Describe the relevance of scalability to performance.
1.8 Define “critical path”, “work”, and “span”.
1.9 Explain the differences between shared and distributed memory.

Level 2
2.1 Use mutual exclusion to avoid a given race condition. [
2.2 Give an example of an ordering of accesses among concurrent activities that is not sequentially consistent.
2.3 Write a program that correctly terminates when all of a set of concurrent tasks have completed.

2.4 Give an example of a scenario in which deadlock can occur.
2.5 Compute the work and span, and determine the critical path with respect to a parallel execution diagram.
2.6 Identify issues that arise in producer-consumer algorithms and mechanisms that may be used for addressing them.
2.7 Measure throughput and latency in a given network.

Level 3
3.1 Use formal techniques to show that a concurrent algorithm is correct with respect to a safety or liveness property.
3.2 Decide if a specific execution is linearizable or not.
3.3 Write a correct and scalable parallel algorithm.
3.4 Parallelize an algorithm by applying task-based decomposition.
3.5 Parallelize an algorithm by applying data-parallel decomposition.
3.6 Implement a parallel matrix algorithm.
3.7 Develop and validate an analytical parallel performance model.

IV. Example activities

A. Dining Philosophers Problem

This activity addressed the learning outcomes 1.4, 2.1, 2.4, and 3.1. The students were given a naive implementation of the Dining Philosophers problem in Java that is subject to deadlock. The program was written by Barbara Lerner at Mount Holyoke College. The students were instructed to explain why the deadlock occurs. Further instructions were to design and implement a solution that prevents deadlock and starvation.

The students initially worked in groups during class to understand the problem and physically act it out to see how the deadlock occurs. Since some of the students were familiar with Java programming and some were not, they were given a pointer to an online reference on Java threads programming. Also, during class, students familiar with Java were grouped with those who were not.

In the program, each philosopher runs in a thread. A philosopher alternates between thinking and eating. To eat, the philosopher needs to pick up the left chopstick and then the right chopstick. The philosopher shares chopsticks with its neighbors, so it cannot eat at the same time as either neighbor. The chopsticks are implemented as reentrant locks. The program gives output to indicate when a philosopher has picked up one chopstick, is eating, or is thinking. A random number generator is used to determine how long a philosopher eats or thinks.

The students had seen a lecture on locking that discussed the conditions under which deadlock can occur. All students in the class were able to device a solution that prevented deadlock. Most students broke the cycle of philosophers waiting on each other by having even and odd numbered philosophers pick up their chopsticks in different orders. However, fewer than half the students were able to give a formal proof that deadlock would not occur with their solution. Also, fewer than half the students were able to implement a solution that prevented starvation. One possible solution to prevent starvation, which was discovered by a few students, is to specify fairness for the Java reentrant lock for the chopsticks.

B. Producer-Consumer using Pthreads Condition Variables

This activity addressed learning outcomes 1.2, 2.1, and 2.6. The students were given an implementation of a producer-consumer problem in Pthreads that uses condition variables for synchronization. As written, the code creates one producer thread and one consumer thread, but it can be modified easily to create multiple producers and consumers. The producer places items in a FIFO queue and the consumer removes them. The producer waits on a notFull condition variable and signals a notEmpty conditional variable. The consumer waits on the notEmpty condition variable and signals the notFull condition variable.

The students were instructed to work in groups, running the Pthreads code and modifying it if needed, to answer the following questions:

1. Explain why the tests on fifo->full and fifo->empty are inside while loop conditions rather than if statement conditions. Describe a scenario of how the code could fail if if statements were used.
2. Assume just one producer and one consumer. Would the code still be guaranteed to work correctly if just the condition variables were used and the associated mutex were not locked? If so, argue why it is still correct. If not, give an example where it fails.
3. Assume just one producer and one consumer. Would the code still work correctly if just one condition variable were used, rather than two – that is, both producer and consumer wait on and signal the same condition variable? If so, argue why it is still correct. If not, give an example where it fails.
4. Now assume more than one producer and/or consumer – for example, one producer and two consumers.
   a. Would the code still work correctly if just one condition variable were used, rather than two. If so, argue why it is still correct. If not, give an example where it fails.
   b. As in part a, assume one condition variable is used, rather than two. Also change the code to use pthread_cond_broadcast() instead of pthread_cond_signal() in both the producer and consumer routines. Now will the code work correctly? If so, argue why it is correct. If not, give an example illustrating how it can fail. Even if it is correct, is it a good solution?

The students had the entire class period of 80 minutes to work on this activity. After much lively discussion within and between groups and with the instructor who circulated about the room, all groups were able to give correct answers to the questions. Each group presented the answer to one of

the questions to the rest of the class. A question on the final exam followed up on this learning outcome and 10 out of 16 students answered that question correctly.

C. Recursive Mergesort using Cilk

This assignment addressed learning outcomes 1.8, 2.5, and 3.4. The exercise was to write and analyze a concurrent algorithm for recursive mergesort using Cilk. The students were instructed to first design and implement a serial recursive merge sort algorithm and to find the size for the base case that gave the fastest runtime. They were asked to draw the binary tree of recursive and find the span and critical path. Next they were asked to convert their program to a concurrent recursive Cilk program, run the program in parallel, and compare the runtime with the theoretically predicted speedup.

After some class discussion and individual help for some students, all students were able to implement a correct serial recursive mergesort program. However, fewer than half the students found the optimal base case size. Eighty percent of the students were able to implement a correct recursive Cilk version. Only a quarter of the student drew the binary tree and analyzed the span and critical path length correctly.

D. Collective Communication Analysis and Benchmarking

This assignment addressed learning outcomes 2.7 and 3.7. Students worked in groups of two to three. Each group was assigned a different collective communication operation. The assignment was to develop a detailed analytical model for the communication costs of the collective operation and to validate the model using both shared and distributed memory on the Stampede supercomputer. Each group was required to write a report and present their results to the class. The benchmark codes implementing the various collective operations were provided as part of the PurdueNext exercises.

The collective operations studied were the following:

1. One-to-all broadcast
2. All-to-one broadcast
3. All-to-all broadcast
4. All-reduce
5. Gather and scatter
6. All-to-all personalized communication
7. Different implementations of one-to-all broadcast

The videos on this topic contain explanations of the optimal algorithms for each collective, as well as an analysis of the communication costs, although some of the detailed steps of the analysis are not included. As a class exercise, students acted out the various collective communication operations and analyzed the number of steps and the communication costs they incurred. As a result, all groups had a good understanding of their operation and were able to explain the algorithm clearly. However, only half the groups were able to fill in the detailed mathematical analysis of the communication costs. For the experimental validation, the students were asked to use regression modeling to fit the experimental data to the analytical models. After some instruction and examples on regression modeling, all groups were able to complete this task successfully.

E. Parallel Matrix Operations

This assignment addressed learning outcomes 3.5, 3.6, and 3.7. The students were asked to implement two parallel matrix operations – matrix-vector multiplication and matrix-matrix multiplication – on a distributed memory system using C+MPI with a 2D block distribution of the matrix data. For each of the two problems, they were required to write a function to perform the matrix operation and a driver program to distribute the data and call the function. They worked through an implementation of matrix-vector multiplication using a 1D block distribution as a class activity. A straightforward implementation of matrix-matrix multiplication was required, with use of Cannon's space-efficient algorithm as extra credit. A further requirement for graduate students was to construct an analytical performance model for the two problems. They were also asked to measure the runtimes of the computational portions of their programs for various values of the dimension size and the number of processors and compare the experimental and analytical results. The analytical modeling was extra credit for undergraduates.

This assignment was given to both the spring 2014 and the spring 2015 classes. The spring 2014 class completed the matrix-vector multiplication part after a large amount of coaching from the instructor, including outlining the code on the board. No one in the class completed the matrix-matrix multiplication part, and the students voiced their opinion that the assignment was too difficult. None of the students completed the analytical modeling part. No class time was devoted to collaborative work on the assignment during class, since most of the class time was taken up by lectures. In the spring 2015 class, half of two 80-minute class periods were devoted to collaborative work on the assignment. During this time, the instructor circulated about the room answering questions and giving feedback, but she did not provide detailed guidance as in the previous year. Over half the class was able to complete the matrix-matrix multiplication portion. One group of two graduate CS students completed the Cannon's multiplication extra credit and successfully completed the analytical modeling part.

V. Outcomes-based evaluation

In addition to using in-class activities and out-of-class assignments, we assessed the learning outcomes across two years of the course, the first year that used the traditional approach and the second year that used the flipped classroom approach. Similar but not identical assignments and test questions were used to assess the same learning outcomes across classes while limiting the opportunity to obtain answers from the previous year (e.g., using a program written by a student from the previous year, perhaps with modifications to avoid detection, or studying from a test obtained from a previous year). The learning outcomes and their levels of achievement for similar test questions for the

two years are summarized in Table 1. The detailed test questions are given in Appendix 1.

TABLE 1. Comparison of achievement of learning outcomes

Outcome	Achievement rate	
	2014	2015
1.1 and 1.3	.88	.94
2.1	.63	.81
2.4	.81	.81
2.5	.63	.81
3.7	.56	.63

Although the sample size of sixteen students per year is small, and the starting skill and experience levels of the two classes may be different, the percentage of students achieving each learning outcome is the same or higher for the class with the flipped format.

In addition to our outcomes-based evaluation, we also compared students' evaluations of the course across the two years for the common questions asked. The results are shown in Table 2. The ratings go from 1 to 5, with 5 being the highest. The students in the flipped classroom felt that the instructor communicated less effectively and was less well prepared, although the instructor actually spent much more time preparing than the previous year. Interestingly, the students in the flipped classroom felt they learned more, thus agreeing with our assessment of the learning outcomes, and that they were challenged more intellectually. It is somewhat alarming that the overall rating of the course went down so much. The overall rating of the instructor also went down somewhat.

TABLE 2. Comparison of course evaluations

Question	2014 rating	2015 rating
Instructor communicated information effectively	4.625	4.375
Instructor prepared for each activity	4.625	4.167
Instructor encouraged me to take active role in my own learning	4.667	4.875
How much I learned in this course	4.000	4.133
Course challenged me intellectually	4.250	4.580
Overall rating of course	4.250	3.500
Overall rating of instructor	4.875	4.545

Students were also able to make free-form comments on for the course evaluation. Some students commented that they did not like the flipped classroom format because it increased their workload. Others said they had trouble understanding the video lectures. Two students commented that they thought the flipped classroom was effective. We are addressing the perception on the part of the students that course is more work by emphasizing the benefits of the flipped classroom approach. We are addressing the problem of not understanding the videos by preparing viewing guides and not using some lectures that are not directly relevant to the learning outcomes.

We saw an overall increase in the amount of engagement that students had with each other with the flipped classroom offering. Because of the nature of the format, students engaged more in class because of being put in groups and assigned activities to do as a group. However, we also noticed more interaction between students outside of class in working and studying together. Some watched the videos together. They also often came to office hours and to help sessions as a group, rather than individually.

VI. Conclusions and Future Work

Although we saw an improvement in achievement of learning outcomes with the flipped classroom format, students still struggled with outcomes related to analysis of algorithms and performance and with proofs of correctness. In general, a high percentage of the students achieved lower and mid-level outcomes, while many failed to achieve higher-level outcomes. We have extended our previous activities and assignments to try to address this deficiency. We have supplemented the assignments that involve modeling and analysis with more specific instructions and examples to help guide the students. We will re-assess the outcomes to determine whether these strategies help during our current second offering of the course in the flipped classroom format. For example, in the first assignment that involves analysis of different scheduling schemes (first-come first-served, shortest job first, and pre-emptive shortest job first) for a concurrent server, we worked through an example of first-come first-served using a graphical timeline showing intervals of time when tasks were executing or waiting in order to analyze and compute the average service time. We then had the students work in groups to analyze the other two scheduling schemes. We followed up with a quiz question in the next class to assess achievement of the learning outcome of correctly modeling and analyzing different scheduling policies for a concurrent server. 90 percent (17 out of 19) of the students answered the question correctly, compared with 69 percent (11 out of 16) on a similar question the previous year. We will have completed our evaluation of the second year of offering the course in the flipped classroom format before the workshop and will be able to present the results at that time.

Another interesting question to explore would be how a completely online offering of the course compares with a flipped classroom offering. UTEP has begun offering online sections of a number of undergraduate and graduate courses in order to accommodate students who have difficulty physically attending class because of distance or schedules. Our evaluation so far indicates that offering the PurdueNext video lectures and quizzes in their current form would be less effective in achieving the desired learning outcomes than a flipped classroom in-person instructor guidance and student interaction. Possibilities for increasing the effectiveness of the online offering include guided solving of example

problems and an online student discussion forum where students and instructors could post and answer questions.

The course format and activities from the three offerings of the course are available on the UTEP instructor's course website at http://svmoore.pbworks.com/. We welcome suggestions on how to improve the course and our flipped classroom approach. We are interested in collaborating with other instructors of concurrent and parallel programming on how to effectively use this approach.

ACKNOWLEDGMENTS

We acknowledge the use of an Education Allocation on the Stampede supercomputer at Texas Advanced Computing Center that was used for parallel programming assignments. We would like to thank William White and Angela Opie for assisting with administration and coordination of the PurdueNext courses.

REFERENCES

[1] L. B. Nilson, Teaching at its best: a research-based resource for college instructors, 3rd ed., John Wiley & Sons, 2010.

[2] Seven things you should know about flipped classrooms, Educause 2012, https://net.educause.edu/ir/library/pdf/ELI7081.pdf

[3] Jill Zarestky and Wolfgang Bangerth. Teaching high performance computing: lessons from a flipped classroom, project-based course on finite element methods. EduHPC@SC 2014: 34-41.

[4] L. Deslauriers, E. Schelew, and C. Wieman. Improved learning in a large-enrollment physics class. Science 332, May 2011: 862-876.

[5] M. D. Estes, R. Ingram, and J.C. Liu (2014). A review of flipped classroom research, practice, and technologies.*International HETL Review*, Volume 4, Article 7, URL: https://www.hetl.org/feature-articles/a-review-of-flipped-classroom-research-practice-and-technologies

[6] Computer Science Curricula 2013: Curriculum guidelines for undergraduate programs in computer science, by the Joint Task Force on Computing Curricula – ACM/IEEE, December 2013, https://www.acm.org/education/CS2013-final-report.pdf

APPENDIX 1
Test questions to assess learning outcomes

Outcomes 1.1 and 1.3
Spring 2014
a. Explain the difference between concurrency and parallelism.
b. Explain the difference between threads and processes.

Spring 2015
Answer True or False and explain your answer: Processes can run in parallel but threads can only run concurrently.

Outcomes 2.1 and 2.4

Spring 2014
Consider the code below for a task queue/thread pool implementation. Here is the code for the thread that is producing the tasks:

```
void AssignWork(WorkItem work)
{
   pthread_mutex_lock(&work_mutex);
   add_work_to_queue(work); // put work item on queue
   pthread_cond_signal(&work_cv); // wake worker thread
   pthread_mutex_unlock(&work_mutex);
}
```

and here is the code executed by the threads in the thread pool:

```
while(1) {
   pthread_mutex_lock(&work_mutex);
   while (work_queue_empty()) // wait for work
      pthread_cond_wait(&work_cv, &work_mutex);
   work = get_work_from_queue(); // get work
   pthread_mutex_unlock(&work_mutex);
   do_work(work); // do that work }
```

a. Is it possible for deadlock to occur with this code? Why or why not?

b. The semantics of pthread_cond_signal() is that it
unblocks at least one of the threads that are blocked on the specified condition. Given these semantics, in the code above, could it happen that two threads in the thread pool might try to remove a single piece of work from the task queue (i.e., a race condition)? If so, show how this might occur. If not, explain why not.

Spring 2015
Consider a linked list structure with a mutex for each node. The delete function searches the list starting at ListHead, which is never removed, until it finds the node containing the value to be deleted. The code for the data structure and the delete function is shown below.
a. Can concurrent calls to delete result in a race condition? Why or why not?
b. Can concurrent calls to delete ever deadlock? If so, show how. If not, argue why not.

```
typedef struct node1 {
   int value;
   struct node1 *link;
   pthread_mutex_t lock;
} node1_t;
node1_t ListHead;
```

```
node_t *delete(int value)
{
  node1_t *prev, *current;

  prev = &ListHead;
  pthread_mutex_lock(&prev->lock);
  while ((current = prev->link) != NULL) {
    pthread_mutex_lock(&current->lock);
    if (current->value == value) {
      prev->link = current->link;
      pthread_mutex_unlock(&current->lock);
      pthread_mutex_unlock(&prev->lock);
      current->link = NULL;
      return(current);
    }
    pthread_mutex_unlock(&prev->lock);
    prev = current:
  }
  pthread_mutex_unlock(&prev->lock);
  return(NULL);
}
```

Outcome 2.5

Spring 2014

Consider the recursive C code below.

a. Add keywords to convert the program into a recursive Cilk program.
b. Assuming a list of length n and BASE = sqrt(n), find the average parallelism T_1/T_∞ .

```
void vadd (real *A, real *B, int n) {
    if (n <= BASE) {
        int i;
        for (i=0; i<n; i++)
            A[i] +=B[i];
    } else {
        vadd (A, B, n/2);
        vadd (A+n/2, B+n/2, n-n/2);
    }
```

Spring 2015

Consider the recursive unordered search algorithm programmed below in C.

a. Modify the program to make it a recursive Cilk program.
b. Given a list of size 16, sketch the execution graph. Analyze the critical path length and upper and lower bounds on runtime for a list of size n.

```
void unordered_search(int list[], int lo, int hi, int key) {
    int mid;
    if (lo > hi)
        return;
    mid = (lo + hi) / 2;
    if (list[mid] == key)
        printf("Key found at position %d\n", mid);
    unordered_search(list, lo, mid - 1, key);
    unordered_search(list, mid + 1, hi, key);
    return;    }
```

Outcome 3.7

Spring 2014

a. Draw a picture to show how a 128 x 128 matrix A would be distributed among 16 processors using a 2D block-cyclic distribution with a block size of 32 x 32 .
b. Sketch an algorithm for multiplying the matrix with a 128 x 1 vector x using collective communication algorithms. You may assume that the vector x and the result vector b are stored on the diagonal processors.
c. Write expressions for the computation and communication costs of your algorithm.

Spring 2015

You are given an n x n matrix that is block-row partitioned across p processors. That is, each processor has a block of n/p rows. The transposed matrix should also be block-row partitioned after the operation. What is the parallel time to perform the transpose of the matrix on a hypercube? Include both local and communication operations in the runtime.