Integrating Parallel and Distributed Computing Topics into an Undergraduate CS Curriculum

Andrew Danner and Tia Newhall

Computer Science Department
Swarthmore College
Swarthmore, PA 19081, USA
{adanner,newhall}@cs.swarthmore.edu

***Abstract*—We present changes to our undergraduate computer science curriculum for a small liberal arts college. The changes are designed to incorporate parallel and distributed computing topics into all levels of our curriculum, with the goal of ensuring that all graduating CS majors have exposure to, and experience with, parallel and distributed computing. Our effort is motivated by the ACM/IEEE Ironman Curriculum, which includes a increased focus on these important topics. In addition, we use the NSF/IEEE-TCPP model curriculum as a guide in our effort. Because of the small size of our department, and the breadth constraints of a liberal arts college, we face some unique challenges. Our multi-year effort involves at least six courses in our curriculum. Of these courses, one is a new introductory-level course, while the others are existing courses whose content has been modified to include more focus on these important topics. We present our curricular changes and we discuss an initial evaluation of the first implementation of these changes.**

I. Introduction

The current ubiquity of multi-core computers, GPGPU computing, and cluster computing has brought parallel and distributed computing from an isolated research topic to a core part of many areas of computer science. While these parallel architectures introduce new memory hierarchies, interprocess communication mechanisms, and new programming environments, traditional computer science education at the undergraduate level focuses on single threaded programming and algorithmic analysis. Occasionally, parallel topics were included in upper-level elective courses. The ACM/IEEE's Ironman [1] draft for the 2013 CS education curriculum has added a new knowledge area of Parallel and Distributed computing to stress the importance of teaching parallel computation throughout the undergraduate curriculum and not just in a single optional course.

To further encourage integrating parallel computing into undergraduate education, Prasad et al. [2] developed the NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing–Core Topics for Undergraduates. Through their support, we are integrating core TCPP topics into six courses in our undergraduate curriculum at Swarthmore College. Our goals are to convey the increasing importance in parallel and distributing computing to undergraduates; to better prepare students for research opportunities in parallel computing; and to integrate parallel topics throughout the curriculum–including introductory-level, systems, theory, and application courses.

One of the primary challenges of our proposal is that Swarthmore College is an undergraduate-only institution. Students must take 20 of their 32 courses outside of their major, and our limited staffing prevents us from offering either the wide variety of courses, or the depth of courses, offered at large research universities. To ensure that students have sufficient background in parallel and distributing computing, it is essential that we cover parallel topics throughout the curriculum, including introductory courses. A key component of our work is the addition of a new course in our curriculum that is focused on introducing computer systems and introducing some parallel topics. It is designed to better prepare students for upper-level systems courses and will allow us to discuss more advanced topics in those courses by requiring this new course as a prerequisite.

A. Overview

In Fall 2012, we began to integrate elements of the TCPP curriculum into our courses. We added a new course required for all CS majors and offered every semester, *CS31: Introduction to Computer Systems*. The course includes introductory TCPP material and is now a prerequisite for upper-level courses that build on parallel and distributed topics in different contexts.

Existing upper-level electives are offered every other year. We introduced or developed additional parallel material for *CS41: Algorithms* in Fall 2012. We will be further developing material in *CS40: Graphics* (Spring 2013), *CS45: Operating Systems* (Fall 2013), and *CS87: Parallel and Distributed Computing* (Spring 2014). In addition, we plan to add parallel topics to *CS75: Compilers* and potentially to *CS44: Databases* in the future. The proposed curricular changes and course schedule will provide at least one introductory and one upper-level course containing parallel core topics every semester.

B. Project Goals

Our main goal is to ensure that every Swarthmore CS major and minor is exposed to parallel and distributed computing. In particular, we focus on teaching students "parallel thinking". We want every student to be exposed to fundamental issues in parallel and distributed computing from the algorithmic, systems, architecture, and programming perspectives. Students should also develop skills to analyze and problem solve in parallel and distributed environments.

In addition to our primary goal, we also want to increase opportunities for students to participate in parallel and distributed research projects. We plan to share course materials that we develop such as lectures, lab assignments, and syllabi, with colleagues at other CS departments who are interested in adding parallel content to similar courses.

In the rest of this paper, we provide a brief background of our department and highlight recent curricular changes in Section II. Next, we describe changes made to our Fall 2012 courses and proposed changes to future courses in Section III. Finally, in Section IV we provide initial evaluation of our changes to our Fall 2012 courses.

II. Swarthmore Background

Swarthmore is a small, elite liberal arts college. The Computer Science Department consists of five tenure track faculty and offers CS major and minor degrees. Because of our small size, we are not able to cover all areas of computer science. We provide a set of core introductory courses, and a set of upper-level electives designed to provide some depth and breadth to students.

Our curriculum is based on several factors including the small size of our department, the expertise of our faculty, and the nature of a computer science curriculum in the context of a liberal arts college [3]. Our pedagogical methods include a mix of lectures, active in-class exercises, and labs. A large number of our graduates eventually go on to CS graduate school; for this reason, our curriculum includes a focus on preparing students for graduate study by providing them instruction and practice in reading and discussing CS research papers, in technical writing, in oral presentation, and in independent research projects. We offer some seminar-style courses, much like graduate seminars, that are a mix of discussion and projects–designed to help prepare our students for graduate study and for summer research experiences.

The main goal of our curriculum is to increase proficiency in computational thinking and practice. We believe this will help both majors and non-majors in any further educational or career endeavor. We teach students to think like computer scientists by teaching algorithmic problem solving, by developing their analytical thinking skills, by teaching them the theoretical basis of our discipline, and by giving them practice applying the theory to solve real-world problems. We feel that by teaching our students how to learn CS, they master the tools necessary to adapt to our rapidly changing discipline.

A. Our Curriculum Prior to 2012

Our most recent previous curriculum was last offered during the 2011-12 academic year, when we had only four tenure lines. It included three introductory-level courses: a CS1 course taught in Python; a CS2 course taught in C++; and a Machine Organization course with some C programming. Because of the constraints of being in a liberal arts setting, as well as constraints with how frequently we can offer upper-level courses (typically once every two years for a given course), we cannot have a deep prerequisite structure. As a result, all of our upper-level courses only had CS1 and CS2 as prerequisites. After taking CS2, students needed to take one of Theory or Algorithms, one of Programming Languages or Compilers, our senior seminar course, and three upper-level electives. We also require two math courses beyond second semester Calculus.

Prior to our new curriculum, students were only exposed to systems topics and to parallel and distributed computing topics in upper-level electives. This meant that students could graduate with a CS major and have little to no systems background and no exposure to parallel or distributed computing topics.

B. Our New Curriculum

Our previous introductory sequence prepared students well in algorithmic problem solving, programming, and algorithmic analysis, and thus prepared students well for about one half of our upper-level courses. However, we found that our students lack of computer systems background made them less well prepared for many of our upper-level courses in systems related areas and, as a result, we had to spend time in each of these courses teaching introductory systems material and C programming. It made these courses seem more difficult to the students new to this material, and repetitive to the students who had seen this material in other upper-level courses. It also meant that there was often advanced material we could not get to in these courses.

Our revised curriculum [4] is designed to ensure that all students have some basic computer systems background so they can be better prepared for all our upper-level courses. In Fall 2012, we added a new required introductory sequence course, CS31, that is an introduction to computer systems. This course replaces our machine organization course and is now a prerequisite to one half of our upper-level courses. CS31 also includes a focus on parallel systems and parallel computing.

We additionally reorganized our upper-level courses into three groups. Students must take at least one course in each group. This change increases breadth of our major and minor. The groupings are as follows (courses requiring CS31 as a new prerequisite are starred):

- **Group 1: Theory and Algorithms**: Algorithms, Theory.
- **Group 2: Systems**: Networking*, Databases*, Operating Systems*, Compilers*, Parallel and Distributed Computing*.
- **Group 3: Applications**: Graphics*, AI, Natural Language Processing, Information Retrieval, BioInformatics, Software Engineering, Adaptive Robotics, Programming Languages.

Students take two additional upper-level electives, including any from the three groups, and including Computer Architecture, Mobile Robotics, or Vision, taught in the Engineering department.

III. Incorporating The NSF/IEEE-TCPP Curriculum

Our current curricular focus is to incorporate most of the NSF/IEEE-TCPP curriculum into our undergraduate CS curriculum and to re-structure major and minor requirements to ensure that every student has exposure to these important topics.

Our effort is a multi-year plan, involving at least six courses, one of which is new, while the others are existing courses to which parallel topics will be added or expanded. This set of courses include the following (the first semester in which each will be taught within our new curriculum is listed):

- CS31: Introduction to Computer Systems, Fall 2012
- CS41: Algorithms, Fall 2012
- CS40: Graphics, Spring 2013
- CS45: Operating Systems, Fall 2013
- CS75: Compilers, Spring 2014
- CS87: Parallel and Distributed Computing, Spring 2014

In addition, we may introduce parallel and distributed computing topics in a new Databases course, which will first be offered in Spring 2014.

A. *Details of our Affected Courses*

We discuss in detail the curricular changes to each of the six courses in which we are implementing the NSF/IEEE-TCPP curriculum. Two of the courses, Algorithms and Introduction to Computer Systems, were first taught in their new form in Fall 2012. The others will be taught in subsequent semesters.

CS31: Introduction to Computer Systems. CS31 is a new introductory course first offered in Fall 2012. It is required for all CS majors and minors. It is also a new prerequisite to many of our upper-level courses, providing students with appropriate background in systems, assembly, and parallel computing.

There are three main learning goals associated with this course: the first is to understand how a program goes from being expressed in a high-level parallel or sequential programming language to being executed on a computer; the second is to understand computer systems costs associated with program performance and to be able to evaluate trade-offs in system design; and the third is to understand parallel computing, including programming, algorithmic and systems issues, with a focus on shared memory parallelism and threaded programming. Secondary goals include: learning C, assembly, and Pthreads programming; learning debugging and debugging tools such as gdb and valgrind; designing and carrying out performance experiments; and working collaboratively in small groups.

The course is structured much like a vertical slice through the computer, starting from the building blocks of binary data representation and circuits, through how programs written in high-level languages are executed. We consider both sequential programs written in C and parallel programs running on multi-core computers written using Pthreads.

The course includes an associated weekly lab section. These labs are used to teach students the programming tools necessary for carrying out lab assignments, to provide practice on lecture content, and to help students with their lab work. The lab assignments provide student practice with practical application of the topics covered in lecture. Table I lists the labs assigned in our first offering of CS31, and includes learning goals associated with each assignment. We used the Bryant and O'Hallaron textbook [5] for the course. Two of the lab assignments used in CS31 came from these authors: the binary bomb and the Unix shell labs. In addition to being the CS31 textbook, we anticipate that it will serve as a useful reference for students in many of our upper-level courses. The web page for CS31 [6] includes more details including the weekly schedule with reading assignments, full lab assignments, weekly lab assignments, and links to programming resources and references.

CS31 includes many topics from the TCPP curriculum, with a focus on covering much of the minimal skill set. Topics covered span the Architecture, Programming and Algorithms topics from the TCPP curriculum. Table II lists the specific topics covered and a description of how they are covered.

The course serves as a first introduction to machine organization and computer systems, and to parallel architecture, systems and programming. There is a strong focus on analyzing problems from a systems perspective. For example, students use what they learn about the memory hierarchy to evaluate the performance of code based on its memory costs in addition to its algorithmic complexity. The course introduces experimentation and performance measurement and analysis at various levels, including applying performance metrics such as latency, bandwidth, I/O and synchronization costs, Amdahl's Law, space-time tradeoffs, locality, and speed-up. The focus on systems topics, including virtual memory, processes and threads, in combination with assembly programming, allows students to have a good background for understanding issues associated with shared memory and synchronization in threaded programs.

CS41: Algorithms. Our upper-level algorithms course [7] explores fundamentals of algorithmic design and analysis, including formalizing an algorithmic statement of a problem from abstract descriptions, developing algorithmic solutions, proving correctness, and analyzing both runtime and space complexity. The course previously covered some TCPP topics including asymptotic analysis, time and space complexity, divide and conquer, and recursive techniques. In Fall 2012, we introduced two weeks of material to 43 students covering alternative computational models including parallel (PRAM, BSP/CILK) models, and the out-of-core (I/O-efficient) model. We used merge sort as a primary example revisiting the analysis of its complexity in the RAM and out-of-core contexts, as well as discussing the work and span of parallel merge sort. We added core TCPP concepts including speedup, scalability, work, and span in the discussion of our analysis. A more detailed summary of topics is shown in Table III, and course materials including topics, lab exercises,

TABLE I
CS31 Lab Assignments.

ASSIGNMENT	TOPIC	GOALS
Data Representation	Binary data representation, Binary arithmetic and operations	understand binary representation of different C types convert between hex, decimal, binary binary Arithmetic and bit-wise operations, overflow intro to C programming and gdb
Building an ALU	Digital Logic, Circuits, Executing Machine code	to build and test circuits from basic gates understand how machine code instrs are executed
Bit compare, Bit vectors	Bit-wise operations Memory, Assembly Code	writing assembly code disassembling code in gdb understanding bit-wise operators and encodings C programming and debugging
Binary Bomb	IA32 Assembly, The Stack Scope, Functions	reading and tracing IA32 assembly understanding C to IA32 translation practice with tools for examining binary files
Game of Life	C Programming, Timing Experiments	understand dynamic memory, C pointers writing and designing larger C programs understanding memory layout of 2D arrays learning how to add timing measurement to C code
Python lists in C	C pointers, C structs, Low-level Memory	implementing and using C-style libraries understanding memory storage layout of different C types C operations on memory (memcpy, void *, recasting, pointers)
Unix Shell	Processes, Unix Process Creation, Signals, Race Conditions	understand how a Unix shell works understand processes and the process hierarchy understand signals practice using fork, exec, signal handlers
Parallel Game of Life Using Pthreads and Experimental Scalability Study	Threads, Shared Memory Programming, Synchronization, Scalability Analysis	understanding shared memory programming understanding and solving synchronization problems pthread programming experience developing a parallel algorithm designing and carrying out scalability experiments analyzing data and explaining results in written report

TABLE II
NSF/IEEE-TCPP Curricular Topics Covered in CS31.

MAIN TOPIC	DETAILS	PEDAGOGICAL METHODS
The Memory Hierarchy	Storage Circuits, RAM, Disk, Caching and Cache Organizations, Paging, Replacement Policies, Cache Coherence	Lecture, Lab Assignments, Exams, Written Assignments
Multicore and Threads	Architecture, Buses, Coherency, Explicit Parallelism, Threads and Threaded Programming	Lecture, Lab Assignments, Exams, Written Assignments
Operating Systems	Overview, Goals, Processes, Threads, Synchronization Primitives (locks, semaphores), Virtual Memory, Efficiency, Mechanism/Policy and Space/Time Trade-offs	Lecture, Lab Assignments, Exams, Written Assignments
Parallel Algorithms and Programming	Shared Memory Programming, Threads, Synchronization, Deadlock, Race Conditions, Critical Sections, Producer-Consumer, Amdahl's Law, Scalability, Speed-up	Lecture, Lab Assignments, Exams, Written Assignments
Other Topics Covered In-Depth	Machine Organization Topics, Assembly programming, C to IA32, The Stack, Function Call Mechanics	Lecture, Lab Assignments, Exams, Written Assignments
Other Topics Covered	Distributed Computing, Message passing basics TCP-IP sockets, Pipelining, Super-scalar, Implicit parallelism	Lecture

TABLE III
NSF/IEEE-TCPP Curricular Topics Covered in CS41.

MAIN TOPIC	DETAILS	PEDAGOGICAL METHODS
Parallel and Distributed Models and Complexity	Asymptotic Bounds, Time, Memory, Space, Scalability, PRAM, Task graphs, Work, Span	Lecture, Lab Exercises, Homework, Exams
Algorithmic Paradigms	Divide and Conquer, Recursion, Scan, Blocking, Out-of-Core (I/O-Efficient) Algorithms	Lecture, Lab Exercises, Homework, Exams
Algorithmic Problems	Sorting, Selection, Matrix Computation	Lecture, Lab Exercises, Exams

and homework assignments are available on the publicly accessible course web page [7]. While the primary text for the course was Kleinberg and Tardos [8], we used parallel material from Cormen et al. [9]. Students explored parallel algorithms in group lab exercises, and as part of their final exam.

CS40: Computer Graphics. The primary focus for the Graphics course is on data structures and algorithms for representing and rendering 3D models. We have gradually introduced parallel topics, primarily CUDA, into the course. In two weeks of instruction in Spring 2011 we provided a brief introduction into CUDA and covered TCPP core topics including SIMD and stream architectures, memory organization (CPU memory, GPU memory, shared memory), hybrid computing, GPU threads, synchronization, scheduling on CUDA GPUs, data layout, and speedups. Practical examples and assignments explored both graphics and general purpose applications, including parallel reductions on large arrays.

In Spring 2013, we plan to further develop these topics, possibly expanding into a third week of coverage. By redesigning some of the introductory material and introducing programmable shaders from the start of the course, students can begin to understand GPU programming well before we begin a full discussion of GPGPU programming using CUDA. This will allow us to better integrate TCPP topics without sacrificing more traditional core material.

Furthermore, making CS31 a prerequisite to CS40 allows us to skip some introductory parallel concepts in graphics, and explore more advanced CUDA topics. As parallel concepts become more commonplace throughout the curriculum, we may be able to offer a large multi-week project in which students develop a hybrid MPI/CUDA ray tracer to run on GPU clusters.

CS45: Operating Systems. CS45 covers a fairly standard undergraduate OS curriculum, including processes, threads, synchronization, memory management, file systems, I/O, protection and security and an introduction to distributed systems. The course strives to have a good balance of theory and practice. It includes a strong focus on analyzing performance based on systems costs, trade-offs in system design, layered design, and the separation of mechanism and policy.

Prior to CS31 being added to our curriculum, CS45 also provided an introduction to C programming and C programming tools, to computer architecture, and to Unix utilities. With the addition of CS31 as a new prerequisite to CS45, we will be able to cover more advanced operating systems topics in OS. In particular to the NSF/IEEE-TCPP Curriculum, we will add more coverage of distributed systems, distributed file systems, networking and security. In addition, because students will have experience with Pthread programming, there may be an opportunity to have students implement and test some of the complicated synchronization problems that they solve in OS, but have only evaluated as written problems in the past.

CS87: Parallel and Distributed Computing. We first added CS87 to our curriculum in Spring 2010, and offered it a second time in Spring 2012. This course is a broad survey of parallel and distributed computing topics. CS87 includes both lecture and in-class discussion of CS research papers. The course includes many short lab assignments in the first half of the semester to give students practice using different parallel and distributed programing languages and paradigms. These have included Pthreads, MPI, OpenMP, C socket client-server, CUDA, and practice using XSEDE [10] resources for MPI and hybrid MPI-CUDA programming. The short labs are designed to teach students tools for carrying out independent course projects during the second half of the course. The course is about 1/3 systems topics, 1/3 programming languages, and 1/3 algorithms. There is an emphasis on paper reading, writing, oral presentation, experimentation, and researching, proposing, and carrying out an independent project.

The course covers many of the TCPP topics in all four Areas. Topics covered in the most recent offering include: the memory hierarchy, pipelining, multi-core, SMPs, false sharing, vector processors, GPUs, MPPs, clusters, grid, P2P, cloud computing, SIMD, MIMD, data parallel, client-server, distributed memory, shared memory, threads, synchronization, MPI, CUDA, OpenMP, Map-Reduce, hybrid CPU-GPU-MPI programming, messaging communication, parallel programming patterns, parallel reduce and scan, trade-offs, speedup, scalability, dependencies, time, power, parallel algorithms, fault tolerance, distributed file systems, distributed shared memory, security, and networking.

Because students have a wide range of computer systems backgrounds entering this course, we have had to provide instruction in basic systems, architecture and C programming. With the addition of CS31 as a new prerequisite to CS87, we will be able to assume that all students have background in computer architecture, operating systems, C programming, shared memory systems and programming, threads, and synchronization as well as practice applying a systems perspective to performance analysis. With this background, much of the introductory material in CS87 can be replaced with more advanced parallel and distributed computing topics. This will allow for increasing both the breadth and depth of coverage of these two fields. It will also allow for at least one additional parallel or distributed short lab to replace a C warm-up lab that was necessary prior to CS31 being a prerequisite. Most likely the additional lab will involve using Hadoop.

CS75: Compilers. CS75 covers a fairly standard undergraduate compilers curriculum, including detailed coverage of parts of the front-end and back-end of a compiler. Students implement a compiler for most of the C programming language over the course of the semester, implementing a lexical analyzer, parser, and a code generator with some optimizations, in a multi-part semester-long project.

Prior to CS31 being added to our curriculum, CS75 also served as an introduction to C programming and C and Unix programming tools, an introduction to computer architecture, and and introduction to assembly language and assembly language programming. With the addition of CS31 as a new prerequisite, our students will come in with extensive background in C and IA32 assembly, stack and function call implementation, and a strong understanding of scope. This will enable us to expand the content on compiler optimization and on advanced topics. In particular to the TCPP effort, we will add content about optimization for super-scalar, multi-core and SMP systems, and we can discuss techniques for solving false-sharing issues. We may also include some coverage of just-in-time and dynamic compilation, and compilation issues for general purpose GPU computing.

CS44: Databases. With our new tenure hire in Fall 2012, we will be adding Databases to our set of regularly offered upper-level courses. The course will be taught with a heavy focus on the systems side of databases. Although not originally included as part of our TCPP Early Adoptor effort, we expect that this course will include some coverage of parallel and distributed database systems, potentially including coverage of parallel join algorithms, distributed transactions, and distributed hash tables.

IV. Initial Evaluation

Our primary evaluation goals for this project are to assess how well we integrate TCPP topics across the curriculum and how our modifications influence student ability to think effectively using parallel concepts. Immediate assessment of individual courses is done through lab assignments, exams, and end-of-course surveys. However, we are particularly interested in how topics introduced in CS31 prepare students for upper-level courses that explore advanced parallel topics. Naturally, as we introduce topics across multiple courses and phase in the prerequisite of CS31, there will be some redundancy and repetition across upper-level courses. We plan to evaluate and identify these common topics and consider them for inclusion in later iterations of CS31.

Ideally, we would like students to apply knowledge of parallel topics to other courses in the curriculum which do not necessarily emphasize TCPP core topics. Spring 2013 presents an opportunity to evaluate how topics in CS31 and CS41 (Algorithms) are retained and applied, as there will likely be some students in CS40 (Graphics) who have had one or both of these courses in the previous semester. Since CS31 will be a prerequisite for Graphics, but not for Algorithms, we plan to assess the level of overlap in topic coverage amongst these courses and potentially adjust the coverage in future iterations of these courses.

As we coalesce common TCPP core topics from upper-level courses in CS31, there may be opportunities to move smaller, simpler elements of parallel programming into other introductory level courses, including our CS1 and CS2 equivalent courses that currently do not emphasize parallel concepts.

A. *Evaluation of CS41: Algorithms*

Adding parallel topics to CS41 was a success from the perspective of both faculty and students. An end of course evaluation specifically asked students to comment on alternate models of computation including the I/O model and parallel models. Feedback was generally positive and several students commented on how these models seem more applicable to current computational problems. Two students mentioned that studying alternate models of computation was one of their favorite parts of the course, while no one listed parallel models as their least favorite (the overall least favorite topics were NP Completeness, solving recurrences, and writing proofs, all features that are likely not going away in an Algorithms course). Several students mentioned that the material on I/O-efficient and parallel algorithms was covered too quickly, and that they wished there was more coverage of parallel algorithms.

From the faculty perspective, the new parallel material fit naturally in the course syllabus, and the introduction of alternate models after completing sections on divide and conquer and solving recurrences made for a smooth transition from traditional to parallel material. The use of merge sort as a unifying algorithm helped students make connections between the multiple models of computation. To address concerns that the material was covered too quickly or that there was not enough parallel material, it may be beneficial to skip coverage of the I/O-model and introduce additional examples in a parallel model and add homework problems on parallel algorithms instead of only covering the topics in lecture and lab sessions. Overall, integrating parallel topics into an undergraduate algorithms course was successful and something we plan to expand in future offerings of CS41.

B. *Evaluation of CS31: Introduction to Computer Systems*

Based on faculty assessments and student course evaluations, our overall assessment of the first offering of CS31 is that it was very successful in meeting its goals. Comments from student course evaluations helped to reinforce our faculty's evaluation of the courses. In particular, in student answers to a very open-ended question on the course evaluation, we received numerous comments specifically mentioning course goals. The question was stated as "How did this course contribute to your intellectual growth? Were you exposed to new concepts or new perspectives? Explain.". The following are just a few of the student comments that specifically mentioned CS31 learning goals:

- "I feel that I understand how a computer functions from the ground up".
- "I liked learning how to boost performance in terms of systems costs".
- "I feel much more knowledgeable about the inner workings of computers. I think this knowledge will contribute to my implementation of higher level programs".
- "I learned to think about efficiency in ways other than algorithm efficiency".

- "I think this course was essential to exposing me to systems concepts".
- "Exposed to looking at runtime of programs differently. I now know the essence behind how a computer actually works".
- "I think in a much more informed way about how computers work under normal daily OS stuff. That is really nice for both general information and for future CS work".

CS31 is designed to be a next course after our CS1 course, so we want to ensure that students entering CS31 having taken only our CS1 course, are not overwhelmed by the content or by the C programming assignments. Based on feedback from student course evaluations, we learned that the course was a bit too ambitious and challenging for this group of students. In addition, both the instructor and many students felt that there should be more time spent on operating systems, parallel computing, and Pthread programming. We plan to modify the course in the following ways to address these issues:

1) Reduce by one lecture the coverage of IA32 assembly, reduce the depth of coverage of compilers topics, and eliminate heap memory management to make room for expanding the coverage of other topics.
2) Add a first week of introduction to C programming to better prepare students for lab assignments.
3) Lengthen the coverage of parallel programming by at least one lecture and possibly two. This will also include adding additional Pthreads programming exercises either to be done during a weekly lab session or as one of the assigned labs in the courses.
4) Provide more practice with synchronization problems and with solving them using Pthread synchronization primitives.

Overall, we are very pleased with this course. Students really enjoyed the course and felt that they learned a lot. The next step in our evaluation will be to evaluate how well CS31 prepares our students for the upper-level courses that newly require it as a prerequisite.

V. Conclusion and Future Work

The Fall 2012 semester represents our first steps of including parallel topics in our undergraduate curriculum at Swarthmore College. The major curricular change was the introduction of CS31, a new introductory course which better prepares students for systems work in upper-level courses while introducing concepts in parallel computing. Initial evaluations from both student and faculty perspectives indicate that CS31 met its proposed goals.

Additionally, we added two weeks of theoretical parallel topics into CS41: Algorithms. The two weeks of new material replaced other optional advanced topics in algorithms without sacrificing core material. Evaluation of student lab work and exams indicated that students could recognize the difference between parallel and sequential algorithm analysis and identify the benefits of parallel algorithms from a theoretical perspective.

Our plan is to continue implementing changes in our curriculum to incorporate parallel concepts, starting with Computer Graphics in Spring 2013 and continuing with Operating Systems in Fall 2013, and Compilers and Parallel and Distributed Computing in Spring 2014.

Because many of our upper-level courses now require CS31 as a prerequisite, we anticipate that we can easily add parallel and distributed material in these courses without sacrificing core material. As we evaluate our changes to these courses, we expect that some common introductory material in these upper-level courses will continue to be integrated into CS31 to provide an appropriate background for further student exploration of parallel topics in advanced CS courses.

Overall, we feel our initial implementation and evaluation of our curricular changes were a success, and we plan to continue integrating parallel topics throughout the curriculum so students have an opportunity to take courses with parallel and distributed topics every semester. Furthermore, we expect students to be better prepared for research in parallel computing, or to apply these new skills to opportunities in industry.

References

[1] ACM/IEEE-CS Joint Task Force, "Computer science curricula 2013, ironman draft," http://cs2013.org, 2012.
[2] Prasad S. et al., "NSF/IEEE-TCPP curriculum initiative on parallel and distributed computing - core topics for undergraduates," http://www.cs.gsu.edu/ tcpp/curriculum/, 2012.
[3] LACS Consortium, "A 2007 model curriculum for a liberal arts degree in computer science," in *Journal on Educational Resources in Computing (JERIC)*, vol. 7, 2007.
[4] Computer Science Department, Swarthmore College, "Swarthmore computer science department curriculum," http://www.swarthmore.edu/cc_computerscience.xml, 2012.
[5] Bryant and O'Hallaron, *Computer Systems: A Programmer's Perspective, 2/E.* Prentice Hall, 2011.
[6] "CS31: Introduction To Computer Systems, Fall 2012, Course Webpage," http://www.cs.swarthmore.edu/ newhall/cs31/f12/, 2012.
[7] "CS41: Algorithms, Fall 2012, Course Webpage," http://www.cs.swarthmore.edu/ adanner/cs41/f12/, 2012.
[8] J. Kleinberg and E. Tardos, *Algorithm Design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.
[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009.
[10] National Science Foundation grant number OCI-1053575, "XSEDE Extreme Science and Engineering Discovery Environment," http://www.xsede.org, 2011.