Addressing Parallelism in Undergraduate Computer Science Courses

David Monismith
Northwest Missouri State University
800 University Dr., 2210 Colden Hall
Maryville, MO 64468
(660) 562-1802, +1
monismi@nwmissouri.edu

ABSTRACT

Over the past four semesters, several instructors at Northwest Missouri State University have made an effort to integrate parallel and distributed computing into multiple classes. This included the following undergraduate courses: Data Structures (cs242), Computer Organization (cs345), Operating Systems (cs550), and Software Engineering I and II (cs561/cs562). This poster provides an overview of parallel content, assignments, and assessment tools used in such courses. It also covers the relationship of the content to the CS2013 Curriculum Guidelines for Parallel and Distributed Computing (PD) and related Knowledge Areas such as Architecture and Organization (AR), Operating Systems (OS), and System Fundamentals (SF) [1].

In Data Structures, during Spring 2014, students spent several class and lab periods learning to use Linux on a LittleFe cluster, and writing threaded programs in the Java programming language. Students in this course gained hands-on experience with thread-based fork and join, with an embarrassingly parallel threaded matrix multiplication example, with a simple example of race conditions, and engaged in a discussion of data structures that lend themselves to straightforward parallelism. A one-week module was dedicated to these topics allowing for students to cover much of the SF Parallel Fundamentals Knowledge Unit [1].

In Computer Organization, aside from standard topics related to Parallel Architecture and Performance such as Amdahl's law, speedup, Flynn's Taxonomy, cache coherence, and NUMA, students were taught about the layout of a cluster computer. This included both large scale (TACC Stampede) and small-scale (LittleFe) examples. Review of cluster components involved lecture and hands-on experience with the LittleFe cluster computer (a six-node desktop-sized cluster). Labs covered Linux, OpenMP, and CUDA. Students were also provided a demonstration of the N-Body problem using the MPI GalaxSee module on LittleFe, which shows the effects of scaling and network latency by using different numbers of CPU cores and nodes [2]. Topics taught in this course cover various Knowledge Units from AR, OS, PD, and SF Knowledge Areas [1].

In Operating Systems, students completed assignments with Steele (Purdue) and Ranger (TACC) in 2012, and LittleFe and Stampede (TACC) in 2013. Lectures, labs, and assignments were developed to cover many PD and OS Knowledge Unit topics such as synchronization, critical sections, mutual exclusion, deadlock, race conditions, task and data decomposition, scalability, message passing, blocking vs. non-blocking operations, scheduling, memory management, and parallel algorithm patterns [1]. The course included many hands-on activities requiring that students use C, pthreads, MPI. Students in the most recent cohort of this class completed labs, projects, and homework assignments that required the use of local resources and high performance computers. As an example, assignments and lectures covered topics such as simple load balancing via a Monte Carlo simulation, map-reduce via a producer-consumer simulation, and multi-process memory management with worker processes that performed matrix multiplication, word counts, and decryption.

In Software Engineering, a two-semester capstone course, students completed a research paper and in-class presentation during their first semester. Students were given the opportunity to research topics such as NoSQL databases, Big Data, Machine Learning, Image Processing, and even Professional Certification. Many students (more than 1/3 of the class) chose to cover technical topics related to either Big Data or High Performance Computing. In their second semester, students undertook a semester-long project in teams of four or five. All of these projects involved a database, multi-user capability, and network interactivity, thus requiring knowledge and understanding of fundamental parallelism topics.

This poster includes an overview of how the topics listed above were integrated into the Computer Science curriculum example assignments, assessment tools, and how others might do the same.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Computer Science Education and Curriculum.

General Terms

Algorithms, Measurement, Performance, Design, Experimentation, Languages, Theory.

Keywords

Parallelism, High Performance Computing, Education.

1. ACKNOWLEDGMENTS

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1053575.

2. REFERENCES

[1] Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM), and IEEE Computer Society 2013. *Computer Science 2013: Curriculum Guidelines for Undergraduate Programs in Computer Science.* ACM, New York, NY. DOI= http://dx.doi.org/10.1145/2534860.

[2] Joiner, D., Gray, P., Murphy, T., and Peck, C. 2006. Teaching parallel computing to science faculty: best practices and common pitfalls. In *Proceedings of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. PPoPP '06. ACM New York, NY, 239-246. DOI= http://dx.doi.org/10.1145/1122971.1123007.