Panel

Multicore Education: Pieces of the Parallel Puzzle

Organizer and Chair:

Joel C. Adams
Calvin College
Grand Rapids, MI 49546
+001-616-526-8666
adams@calvin.edu

Daniel J. Ernst
U. of Wisconsin–Eau Claire, WI, USA
ernstdj@uwec.edu

Thomas Murphy
Contra Costa College, CA, USA
tmurphy@contracosta.edu

Ariel Ortiz
Tecnológico de Monterrey, Mexico
ariel.ortiz@itesm.mx

PANEL SUMMARY

Although *Moore's Law* continues to hold at present, *Moore's Dividend* – where software developers could rely on increasingly faster CPUs to make their software faster – has expired [5]. Instead of manufacturing uni-core CPUs with faster clocks, hardware manufacturers are producing *multi-core* CPUs, and *many-core* CPUs (with 32 or more cores) have begun appearing. Traditional sequential applications will not take advantage of these new hardware capabilities, and thus will not run any faster. To gain performance on these new and future hardware platforms, applications must be designed and written in pieces that run simultaneously on different cores. Ideally, the performance of such *parallel applications* should *scale* as the number of available cores increases.

As computer science educators, it behooves us to prepare our students for this brave new parallel world. In this session, the panelists will discuss different aspects of doing so, including:

- How do we integrate parallelism into the CS curriculum? What aspects of parallelism do we cover, and where?
- What available technologies (e.g., programming languages, libraries, etc.) facilitate parallel application development?
- What resources are available for CS faculty members to learn how to design and build parallel applications?

Each panelist will focus on one of these aspects of the problem.

Categories and Subject Descriptors

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

General Terms

Algorithms, Design, Languages, Performance

Keywords

Concurrency, Curriculum, Faculty, Many-core, Multi-core, Parallel, Programming, Languages, Training, Workshops

SIGCSE'10, March 10-13, 2010, Milwaukee, Wisconsin, USA.

ACM 978-1-60558-885-8/10/03.

Chair Joel Adams, *Calvin College*

Joel Adams is Professor and Chair of the Department of Computer Science at Calvin College. He has been teaching students about concurrency and parallelism for 20 years. To provide platforms for his students to experience the benefits of distributed parallelism, he has designed and built a variety of Beowulf clusters over the past decade, ranging from small clusters like Microwulf [1] to a terascale cluster at *dahl.calvin.edu*.

As chair, he will introduce the session and each of the other panelists. If (and only if) time permits, he will provide a brief overview of how OpenMP (see *openmp.org*) can be used to parallelize hotspots in legacy (i.e., sequential) C or C++ code.

Panelist Daniel Ernst, *U. of Wisconsin – Eau Claire*

The recent shift in computing towards ubiquitous parallelism raises several questions for computer science educators. Most of these questions relate to how we can best prepare students for a world where parallel resources are always available, and where, as the number of cores increases, sequential applications and single-threaded performance will become less and less acceptable in practice.

Traditionally, parallel programming concepts have been kept in the realm of upper-level electives and/or reserved for students who plan to work in the realm of high-performance scientific computing. This curricular placement within computer science has naturally led to these concepts being taught in an advanced way, appropriate to the level of the students.

As has been observed by others [6], in order to bring parallelism and concurrency to the mainstream, educators need to re-frame parallelism using simpler concepts, preferably ones that students can think about and understand in a natural way.

A few years ago, Daniel initiated and helped begin the implementation of changes to the curriculum at the University of Wisconsin – Eau Claire to attempt to address this deficiency in their curriculum [4]. The approach they adopted was to provide a consistent and integrated exposure to parallelism, spread across the curriculum, in a platform-agnostic manner.

In his presentation, Daniel will share guidelines and lessons he and his colleagues at Wisconsin – Eau Claire have learned. In particular, he will share their initial experiences with their recent curricular changes, and insights stemming from the many discussions they have had with other educators on this topic.

Panelist Thomas Murphy, *Contra Costa College*

Thomas Murphy is a professor and chair of Computer Science at Contra Costa College (CCC). He is director of the CCC High Performance Computing Center, which has supported both the Linux cluster administration program and the computational science education program. Since 2003, Thomas has been an instructor with the National Computational Science Institute Parallel and Distributed Working Group, which presents several three to seven day faculty workshops each year. He also helps coordinate the SC07-11 Education Program. He helps maintain the Bootable Cluster CD software platform, the LittleFe hardware platform, and the CSERD (Computational Science Education Reference Desk) curricular platform.

As a panelist, Thomas will describe multicore and parallel computing resources and opportunities for computer science educators. These include:

- *The SC Education Program*, which provides summer workshops and other opportunities for educators to learn about parallel computing and develop curricular materials;
- *BCCD*, a mechanism for turning a lab of personal computers into a network of workstations (NOW) multiprocessor;
- *LittleFe*, a portable and inexpensive Beowulf cluster;
- *CSERD*, a collection of resources for computational science, including parallel / cluster computing.

If time permits, Thomas will provide a brief overview of success integrating parallelism into a community college CS program.

Panelist Ariel Ortiz, *Tecnológico de Monterrey*

Ariel Ortiz is a fulltime faculty member at the Tecnológico de Monterrey, Campus Estado de México. He has been teaching different programming styles and languages for more than two decades. In 2007, he started to use the *Erlang* language (see *erlang.org*) in his Programming Language course in order to teach functional, concurrent and distributed programming. He has also recently begun covering Google's *MapReduce* model (see *labs.google.com/papers/mapreduce.html*) in the concurrent portion of this course. In this presentation, he will compare and contrast these technologies.

Erlang simplifies parallel programming by modeling a problem as a collection of parallel processes that can only communicate with each other by sending and receiving messages. Erlang has parallel processes, but no locks, and no possibility of shared memory corruption (since there is no shared memory). Concurrency is supported by the Erlang virtual machine and not by the operating system. This means that applications behave the same way regardless of the deployment platform.

What follows is a list with some guidelines for efficient multi-core programming in Erlang. By following them, a program might run *n* times faster on an *n*-core processor – without any changes to the program itself [2]:

- Avoid side effects.
- Use lots of processes.
- Avoid sequential bottlenecks.
- Favor CPU bound processes and small messages.

MapReduce is a programming model that can be used in order to adhere to these guidelines. It involves specifying a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key [3]. Mappings and reductions are executed in parallel.

REFERENCES

[1] J. Adams, T. Brom. Microwulf: A Beowulf Cluster For Every Desk. *39th SIGCSE Technical Symposium on Computer Science Education*, March 2008. 121-125.

[2] J. Armstrong. *Programming Erlang: Software for a Concurrent World.* Pragmatic Bookshelf, Raleigh, NC. 2007.

[3] J. Dean and S. Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters.* http://labs.google.com/papers/mapreduce.html Accessed September 2, 2009.

[4] Daniel J. Ernst and Daniel E. Stevenson. Concurrent CS: Preparing Students for a Multicore World. *13th Annual Conference on Innovation and Technology in Computer Science.* June 2008. 230-234.

[5] J. Laurus. Spending Moorre's Dividend. Communications of the ACM (52) 5. May 2009. 62-69.

[6] Michael L. Scott, "Don't Start with Dekker's Algorithm: Top-Down Introduction of Concurrency," *Workshop on Multicore Programming Education* (held in conjunction with ASPLOS 2009), March 2009.