

Invited Presentations

Title: Opportunities in Many Core Algorithms

Guy Blelloch, Carnegie-Mellon University

Abstract: The advent of many-core machines presents an important opportunity for research in algorithms. These machines are becoming prevalent and unlike most of the large-scale parallel machines developed in the 90s, many-core machines have shared memory making them significantly easier to design algorithms for. Perhaps we can dust off all the great PRAM algorithms developed in the 80s, put them back on the shelf and continue filling the rest of the shelf. I'll argue, however, that the PRAM model itself is probably not the right model, but that many of the ideas, techniques and algorithms developed in the PRAM are still very relevant. Furthermore there are many further areas open for exploration, some leveraging this previous work and some completely new. In the talk I'll describe some such areas.

Title: Theory: Asleep at the Switch to Many-Core

Phillip B. Gibbons (Intel Research Pittsburgh)

Abstract: Two decades after the peak of Theory's interest in parallel computing, the age of many-core is finally underway. Moore's Law is driving a steady doubling of cores per chip, forcing everyone to jump aboard the parallelism train (or be left behind). To overcome the challenges in realizing the potential of many-core, research is needed in all aspects of parallel computing. Yet, judging by the lack of parallel computing papers in FOCS, SODA, and STOC, Theory is asleep at the switch. This talk provides a wake-up call for Theory to regain a leadership role in parallel computing, before it's too late. I will advocate five areas in which Theory can (and should) have an important impact, by playing to its strengths. Illustrative examples of initial progress in several of these areas will be drawn from our work on the Hierarchy-savvy Parallel Algorithm Design (Hi-Spade) project.

Title: Parallel Algorithm Issues in TBB

Arch Robison, Intel

Abstract: Intel(R) Threading Building Blocks (TBB) is an open-source C++ library for writing shared-memory parallel programs. One part of TBB is a set of parallel algorithms based on recursively divisible ranges, implemented via classic work stealing. This talk introduces these algorithms and then focuses on some aspects of their implementation that are necessary in practice, but outside the PRAM theoretical model. Examples are:

- o A conflict between industry standard calling conventions and ideal task stealing.
- o Gains from cache considerations can trump gains from parallelism.

- o Inserting local delays in work stealing can speed up some parallel programs.
- o Efficient parallel scan on multi-core is harder in practice than in theory.

Discussing these issues to light may spark better theoretical models for multi-core and finding better solutions.

Title: A Bridging Model for Multi-Core Computing

Leslie Valiant, Harvard University

Abstract: Writing software for one parallel system is not an impossible task. Reusing the intellectual property so generated for a second system has proved much more challenging.

In order to address this problem for multicore architectures we propose a bridging model aimed at capturing the most basic resource parameters of these architectures. We suggest that the considerable intellectual effort needed for designing efficient algorithms for such architectures may be most fruitfully expended as an effort in designing portable algorithms for such a bridging model. Portable algorithms would contain efficient designs for all reasonable ranges of the basic resource parameters and input sizes, and would form the basis for implementation or compilation for particular machines.

Our Multi-BSP model is a multi-level model that has explicit parameters for processor numbers, memory/cache sizes, communication costs, and synchronization costs at each level. The lowest single level instance of the model corresponds to the PRAM, and in that natural way acknowledges the relevance of that model for whatever limitations on memory and processor numbers it may be practicable to implement it at the lowest level.

We propose parameter-aware algorithms that run efficiently on all relevant architectures with any number or levels or combinations of parameters at each level. We propose a parameter-free notion of optimality for such portable algorithm. We show for several fundamental problems, including standard matrix multiplication, the Fast Fourier Transform, and comparison sorting, that they have portable algorithms that are optimal in that sense for all combinations of machine parameters. Thus algorithmic elegance can be found in this multicore context.

Title: The PRAM-On-Chip Proof-of-Concept

Uzi Vishkin, University of Maryland

Abstract: As the heart of the field is being reinvented for parallelism the main technical challenge is timely convergence to an easy-to-program highly-scalable many-core platform. The wealth of the parallel random-access machine/model (PRAM) theory of algorithms is well documented. The University

of Maryland (UMD) explicit Multi-Threading (XMT) project has been driven by a PRAM-On-Chip vision, seeking to build an easy-to-program parallel computer comprising thousands of processors on a single chip, using a PRAM-like programming model. XMT has gone through extensive hardware (64-processor machine) and software prototyping. A software release allows experimentation with the XMT environment on any standard computer platform.

Interestingly, starting with a PRAM might not have been an obvious choice. Technology constraints guide us away from tightly coupled concurrency in programs; e.g., away from the PRAM and towards multi threading. On the other hand, relaxed concurrency in programs is notoriously difficult to design or analyze for correctness or performance. The XMT programming approach incorporates an elegant workaround. First, XMT provides a (refinement) ***“work flow” from a PRAM algorithm to an XMT program***, and even to fine-tuning an XMT program for performance. Given a problem, a PRAM style parallel algorithm is developed for it using the Shiloach-Vishkin 1982 Work-Depth (WD) Methodology. All the operations that can be concurrently performed in the first round are noted, followed by those that can be performed in the second round, and so on. Such synchronous description of a parallel algorithm makes it easy to reason about correctness and analyze for work (the total number of operations) and depth (number of rounds). The XMT programmer is then expected to use the XMT language (basically C with two additional commands) for producing a multi-threaded program. Reasoning about correctness or performance can **now be restricted to just comparison of the program with the WD algorithm, assuming that correctness and performance of the algorithm have been established**, often a much easier task than directly analyzing the program. This workaround allowed college freshmen and even high-school students solve the same problems they get in typical freshmen serial programming course assignments using (XMT) parallel programming.

Our proof-of-concept demonstrates that the limited diversity of many-core vendor hardware may miss an important opportunity for the timely convergence sought. Theory can play a key role for this or other opportunities.