Concurrency for the Application Programmer

Vijay Saraswat
IBM TJ Watson Research Center
vijay@saraswat.org

Doug Lea
SUNY, Oswego
dl@cs.oswego.edu

Abstract

Forced by architectural and commercial considerations, programmers now have to confront multi-core systems, heterogeneity, clusters, clouds.

What does this revolution mean for the application programmer, typically removed from the hardware through many layers of middle-ware (often on top of managed run-time environments)? How should the capabilities of heterogeneous processors (including GPUs, FPGAs, streaming processors) and heterogeneous memory (including non-coherent memory) be made available to the application programmer? Should abstractions for the application programmer focus primarily on application-level concurrency rather than implementation-level concurrency? Should application-level concurrency abstractions be fundamentally determinate? Fundamentally declarative? Resilient in the face of node- and network- failure? How can high-performance concurrent programs be written in garbage-collected languages? How can they *not* be written in garbage-collected languages?

This workshop aims to bring together practitioners and thinkers to address all topics around concurrency for the application programmer.

Categories and Subject Descriptors K.3 Computers and Education [*K.3.2 Computer and Information Science Education*]: Curriculum

General Terms Algorithms, Design, Experimentation, Measurement, Performance, Theory

Keywords Concurrency, Parallelism, Curricula, Multicore, Applications

1. Introduction

The concurrency revolution is upon us. Forced by architectural and commercial considerations, programmers now have to confront multi-core systems, heterogeneity, clusters, clouds. Programmers must address the challenge of productively and reliably implementing computational systems that deal internally with a large number of threads and massive amounts of data. Fundamental notions (what does it mean to read and write a shared memory location?) must be addressed. Long-standing data-access models (e.g. ACID transactions, relational querying) must be revisited (e.g. to take advantage of mostly read-only data, streaming).

SPLASH'10, October 17–21, 2010, Reno/Tahoe, Nevada, USA.
ACM 978-1-4503-0240-1/10/10.

What does this revolution mean for the application programmer? Typically such a programmer is removed from the hardware through many layers of middle-ware (often on top of managed runtime environments), and works with high levels of abstraction (e.g. in high-level or custom, domain-specific languages or frameworks such as J2EE).

Nevertheless the need to get performance and scale and exploit heterogenous and non-uniform architectures are as vital for such programmers as for the systems programmer. Application programmers face the additional challenge that they have to contend for the same computational resources with layers of the system on top of which they are built. This can cause significant problem with performance transparency and performance portability. (For instance it is well known that some kinds of Single-program/Multiple data programs are very susceptible to jitter that might be introduced by unpredictably scheduled computations on the same system, such as network interrupts, garbage collections etc.)

On the other hand, application programmers can often take advantage of regularities in their domains to simplify the computational task. For instance Hadoop is a concurrent application framework that permits its users to specify sequential pieces of code that are scaled out to run in parallel across multiple nodes of a cluster (data-parallelism). Typically the Map/Reduce computations realized in Hadoop are determinate. It is much easier to build tooling for determinate systems (as opposed to general concurrent, reactive concurrent systems).

This workshop aims to bring together practitioners and thinkers to address the topic of concurrency for the application programmer. Speficially the following questions are targeted:

> How should the capabilities of heterogeneous processors (including GPUs, FPGAs, streaming processors) and heterogeneous memory (including non-coherent memory) be made available to the application programmer? Should abstractions for the application programmer focus primarily on application-level concurrency rather than implementation-level concurrency? Should application-level concurrency abstractions be fundamentally determinate? Fundamentally declarative? Resilient in the face of node- and network- failure? How can high-performance concurrent programs be written in garbage-collected languages? How can they *not* be written in garbage-collected languages?

The workshop will be organized around the presentation of position papers selected by the PC, and a panel discussion. The results of the workshop will be made available online.

The Program Committee for this workshop consists of Bob Blainey (IBM), Joshua Bloch (Google), Ras Bodik (UC Berkeley), Amol Ghoting (IBM), Kevlin Henney (Curbralan Ltd), David Holmes (Oracle), Jim Larus (Microsoft), Doug Lea (SUNY Oswego – co-chair), Martin Odersky (EPFL), Bill Pugh (U Maryland), Vijay Saraswat (IBM – co-chair), Adam Welc (Intel).