Scribe notes for Concurrent Programming

Armando Intro:

How to use the ExCAPE methodology to tackle concurrency/parallelism? There has been a lot of work like the work in the compiler. Problems: cost, correctness, performance portability

How to automate low-level low-level decisions. Keep the programmers in control. Reusability of insights to prevent reinventing the wheel again and again.

Some success stories related to CAPE:

(1) Project named Paraglide, the work at IBM by Vechev et al, which attempts to write correct code with respect to a memory model. (2) Project named Spiral.

Madhusudhan:

Migrating from sequential to parallel code is a good domain for synthesis. Well understood that it is crucial to exploit programmer's intuition on what the code does to parallelize.

Restructuring the code is required to migrate from sequential to parallel code. Specification already exists: Sequential code is the specification.

Few current technologies to assist parallelization:
(1) Intel Parallel Adviser:
   Tells programmers the program segments which can provide benefits of parallelization
(2) Refactoring techniques to transform sequential code to use parallel libraries.

Challenge: Can we combine performance analyzers and refactoring methodologies?

Audience comments/questions/observations:
(1) Object-oriented programming is an enemy of parallelism.
(2) Fine grained concurrency is generally used by hardware developers
(3) Where you store your data is important

Armando:

Problem of parallelism in the context map-reduce parallel distributed memory machines.

Partition the problem and communicate at boundaries. The key idea is avoid communication by computing things locally.

How to automate this problem of writing reusable loop nests?
Idea 1: Make program refinement explicit.
Idea 2: Don't write implementations, write generators
Idea 3: Synthesis + autotuning == better generators

The advantage of this approach over Spiral/Phoenix and other approaches is little domain knowledge. In contrast, investing in domain specific knowledge is a huge capital investment.

Generating complicated data structures was an initial step towards try your own stuff approach.

Audience observation/comment/question?

(1) High performance computing community generally batch their messages in MPI. The methodology still does not provide ability to batch their messages.
(2) How does it compare to Phoenix? Spiral?
(3) Are you going to synthesize your parallel synthesizer?

Ras Bodik:

Talk about synthesis about systems biology. Understanding stem cell differentiation helps in understanding cancer. Understanding cell communication helps cell differentiation.

Research goal: What is the cell's algorithm for robustly deciding cell fates through communication?

Executable Biology is taking the diagrams from the Biologist, and model checking the computational model extracted out of the diagrams. Analysis of models can show the inconsistency with experimental data which can allow one to fix the model probably leading to the discovery of new interactions.

Cell as reactive system. Biologist had written models as a Reactive Module. However it took months to tweak, hard to understand, and was difficult to fix bugs.

Motivation for a new language:

Attempt 1: Synthesizing parts of the RM code
   -- scalability problems
   -- hard to understand

Attempt 2: A new high level language
   -- easier synthesis

The language consists of nodes and some communication. Synthesize a program for each node. The key challenge is determining the semantics. Nodes modeled as finite automata as a first try.

Synthesis approach is based on CEGIS [Armando PLDI 2008].
The language is implemented as an embedded DSL in Scala. Interface to the Z3 SMT solver via Scala library is available. Verification is fast.

Audience questions:

(1) Using the approach as the method to identify the next biological experiments?

Discussion on the challenge questions:

(1) Skepticism on moving from sequential cores to parallel code.

(a) Large investment has been made to parallelize existing code. An interesting research perspective would be to use synthesize technology to assist the creation of new parallel code.

(b) What should be staring point? Using existing code is interesting as it's already there and acts a specification.

(c) Sequential languages don't have a notion of where the data is located. Starting with a sequential program, we need equivalence checking.

(d) Trying to nail down where are the opportunities for automation. Identifying array indices and iteration bounds are the hard things in scientific code. Why not automate them? Another important thing to consider is data locality.

(e) In a concurrent program, introducing memory fences in hard. That's where synthesis has been successful.

(f) Speculative computation is another dimension that parallel computation can exploit that is generally absent in the sequential domain.

(g) Suggestion was to pick one problem. Data layout and data communication are interesting.

(h) Get the users involved early. Examples and scenarios are useful.

(i) Programmers know GDB. Concrete traces and execution is useful.