Models of Computation

brief intro with pointers

Verónica Gaspes

School of Information Science, Computer and Electrical Engineering

CERES

March 19th, 2008

Concurrent programming

Simon Peyton Jones, Microsoft Research, Cambridge

The free lunch is over. We have grown used to the idea that our programs will go faster when we buy a next-generation processor, but that time has passed. While that next-generation chip will have more CPUs, each individual CPU will be no faster than the previous year's model. If we want our program to run faster, we must learn to write parallel programs. Parallel programs execute in a non-deterministic way, so they are hard to test, and bugs can be almost impossible to reproduce.

The quote addresses general purpose programming. In our project we look at specialized architectures and restricted application domains.

Computation models

Attempts at formalizing what notions should be used to express computations and interactions with the environment and other computations.

Programming languages are implementations of computational models. They automatize essential parts of the model so that the programmer can rely on them:

- The compiler can translate source code to assembler so that we can program with reasonable abstractions
- The run-time system organizes execution and things that have to be done in the background while the program is running.

Courses

Models of concurrent computation, at Imperial College.

http://www.doc.ic.ac.uk/teaching/coursedetails/481

Multithreaded Parallelism: Languages and Compilers, at MIT.

http://csg.csail.mit.edu/6.827/

Concurrent models of computation for embedded software, at Berkeley

http://embedded.eecs.berkeley.edu/concurrency/

First case: Streams

Computation model

- Filters repeatedly perform the same computation on each of the elements of an input stream and generate an output stream of results.
- As soon as the filter is ready with one computation and there is data in the input stream it takes this in for a new computation.
- Filters can be combined in sequence and in parallel.

Looks very much like filters in signal processing!

As filters only see local memory they can be executed in parallel!

Reasonable sources for our architectures.

R. Stephens, Stream Processing, A Survey.
Synchronous data flow and some programming languages like StreamIt

Second case: Communicating Sequential Processes

Computation model

- Processes execute sequential code and can handshake with any other process to exchange data.
- Even processes can be exchanged (mobile code)
- Processes can be combined sequentially, in parallel and using non-deterministic choice.

Became a programming language (Occam) to program the *transputer*, an early parallel architecture.

There is lots of good theory including tools for verification.

C.A.R. Hoare, Communicating Sequential Processes
R. Milner and others about the calculus of communicating systems and the π-calculus
The programming language Occam-PI

Third case: Reactive objects

Computation model

- Objects encapsulate state and provide an interface of methods.
- Each object is also a process that is at rest until some other object requests some action from it by calling one of its methods.
- The process serves calls in mutual exclusion.

Very suitable for embedded systems, more modern in that it provides better ways of composing programs.

There are implementations that include real time constraints. There are implementations of the run time system that can be used stand alone.

J. Nordlander, M. Jones and others on Reactive Objects
The programming language Timber.

Transactional memory

Computation model

- Computations are organized as processes that might share some data structures.
- Parts of computations can be qualified as atomic.
- Atomic computations can be retried (blocking).
- There is a choice operator.

Provides for fine grain synchronization that scales well!

Programs become easier to compose that with locks and condition variables (what we usually get for implementing monitors).

S. Peyton Jones, Beautiful Concurrency.
The programming language Haskell with STM.

Opportunities

All these models of computation were put forward as for general purpose computing. In the context of more constrained application domains e.g. signal processing, it might be fruitful to explore more dedicated models that can be used

- to make life easier to the programmer,
- to achieve even better exploitation of hardware resources,
- to provide better tools for analysis and verification (one specification can be used by many tools!).

Some very good examples exist: cryptol for cryptography, lava for fpga configurations and lexifi for financial instruments.