Using Critical-Path Length as a Practical Performance Metric
or
Dataflow Microprocessors

Bradley C. Kuszmaul
bradley@ee.yale.edu
Yale University
August 24, 1998

One of our new research directions at Yale is to explore the hypothesis that explicit-dataflow instruction-set architectures provide significant advantages for high-performance microprocessors.

Our explicit-dataflow instruction-set architecture can be thought of as a VLIW variation on Nikhil and Arvind's PRISC architecture. In our architecture, a thread is a single instruction. We maintain a queue of ready instructions, and the processor runs them using its own scheduler. A typical instruction might perform an arithmetic operation and conditionally spawns some successor instructions. A typical instruction might look like this:

```
Ra := Rb + Rc; if (!Rd--) spawn PC+I; if (!Rd--) spawn PC+J;
```

There are several interesting challenges for dataflow.

- Resource management: Can a dataflow processor run a parallel program without running out of memory? We plan to study hardware support for provably good schedulers, like the MIT Cilk scheduler, that simultaneously achieve good time and space bounds. One of the challenges for Dataflow is to provide hardware support for a provably good scheduling algorithm.
- Code-size blowup: Can a compiler efficiently represent the dependencies in a program without blowing up the size of the program? Under some conditions a good memory disambiguator can result in a squaring of the code size in order to express all the parallelism of the program. The code-size blowup comes from the possibility that there are $\Theta(n^2)$ edges connecting a program of $\Theta(n)$ instructions. Code-size blowup affects the performance by requiring larger memory systems and requiring that the processor handle more synchronization events.
- Branch prediction: Can a dataflow processor exploit branch prediction?
- Compiling C: Does compiling C for an explicit dataflow processor expose more parallelism than does compiling for a superscalar, or VLIW processor, or for pthreads?
- A big picture for parallelism: Can the traditional dataflow vision of a single mechanism for exposing parallelism work? Is there parallelism in future programs?

We are measuring the average parallelism of several ordinary C programs by compiling them into dataflow graphs and measuring the graphs' critical-path lengths. Given the average parallelism, we plan to determine how well various instruction-scheduling mechanisms exploit the available parallelism, and to evaluate the performance contributions made by various compiler optimizations and source-code transformations.

Our intuition is that the technology is finally ready for dataflow processors.