Configuration by Combustion: Online Simulated Annealing for Dynamic Hardware Configuration

Steven Swanson, Ken Michelson and Mark Oskin
Department of Computer Science and Engineering, University of Washington

The Computation Cache is an execution substrate for scalable execution on future silicon fabrication processes. It is a grid-based reconfigurable processor architecture for general purpose computation and is composed of an array of thousands of processing elements that communicate over a switched, grid-based network. Each processing element contains a single 64-bit functional unit, space for a single instruction, and a small amount of storage for input and output queues. Programs for the Computation Cache are made up of normal RISC-type instructions (with a few additions), but with an explicit dataflow encoding of their operands. Execution proceeds in a dataflow fashion with instructions passing their results to the instructions that require them.

One of the central challenges of this architecture is mapping the application onto the grid. We are presently investigating real-time, on-line simulated annealing to dynamically layout and optimize the Computation Cache's reconfigurable hardware. The goal is to inject a description of an application and have the substrate dynamically decide how to implement it and then reorganize itself to adapt to changing application behavior. There are two principle advantages to optimizing layout dynamically at runtime: First, the layout can be routed around defective hardware components. Future silicon processes will be increasingly unreliable and routing around defective hardware is essential if the Computation Cache is to be a fault tolerant, scalable architecture. Second, the layout can change over time to match the changing requirements of the computation so that the current "hot-spot" executes most efficiently.

We envision simply "pouring" the executable onto the substrate and then optimizing the initial layout while this application executes using simulated annealing. When each instruction arrives at a processing element, it broadcasts its location and an identifier so that dependent instructions know its location. Once this stage is complete execution can begin, but it will likely be inefficient because the instructions are placed semi-randomly.

The Computation Cache then dynamically reorganizes the instructions during execution to improve performance. Since the Computation Cache design strives to be scalable, optimization can use only local information. Fortunately, each processing element can collect a wealth of local information about local behavior: Each instruction knows the locations of the other instructions it exchanges messages with and the round-trip latencies to each of them. It also knows how frequently it executes. Branch instructions can collect taken/not-taken statistics useful for optimizing for the common case.

Using this information we plan to use simulated annealing, a randomized technique, to allow more global changes to the layout. In general, simulated annealing starts with a configuration and searches for better solutions by making random modifications to it and seeing how they perform. If the modification is an improvement, it becomes the "current" configuration and the process continues. To escape from local optima, simulated annealing will also make non-optimizing changes with some probability. The probability varies over the course of optimization with a *temperature* parameter: The higher the temperature, the higher the probability it will make a non-improving change. As the optimization process proceeds, the temperature is lowered and the layout settles into an efficient solution.

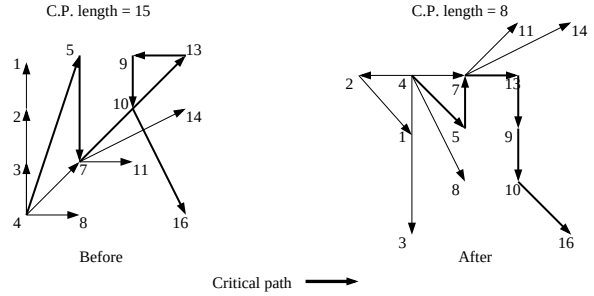

Figure 1: Effects of online optimization.

In the Computation Cache an instruction will move to a new location in the grid. If the performance improves or the temperature is high enough it will stay put, otherwise it will return to its original location and try again. Since the instruction layout should adapt and change as program behavior changes, the temperature parameter will rise and fall over time. The rate of change for the temperature and period of its oscillations are open questions.

Currently we are implementing an execution simulator for the Computation Cache architecture. Therefore, evaluating the online real-time performance of simulating annealing algorithms in the actual architecture will take some time. However, we have constructed a trace-based simulator with a perfect memory system to investigate just the layout algorithms and their effect on program performance. The results are extremely encouraging. For example, for a core function of bzip2, the "ideal" (1 cycle communication between all instructions, which is admittedly unrealistic with 2D grid network) ILP is 6.68 (on a trace of 30M instructions). Several initial placement algorithms where designed and the best performing of these achieved in ILP of 3.04. However, after simulated annealing this same trace achieves an ILP of 6.07, much closer to the "ideal". Figure 1 depicts a simple dataflow graph and the intended effects of optimizations; the critical path (dark lines) are substantially shorter.

As new process technologies change the way we design microprocessors and force wire delay and defect tolerance to the forefront, the need for new techniques to utilize large (and possibly flawed) dies will increase. At the same time, transistor budgets will soon be large enough to allow real-time monitoring and reconfiguration of the processor to match workload conditions. The Computation Cache, using a simulated annealing algorithm for configuration and optimization, is one example of a design built to address these problems and exploit these opportunities.