Wavenets

John Chapin
MIT Laboratory for Computer Science

ASPLOS new ideas session, October 1998

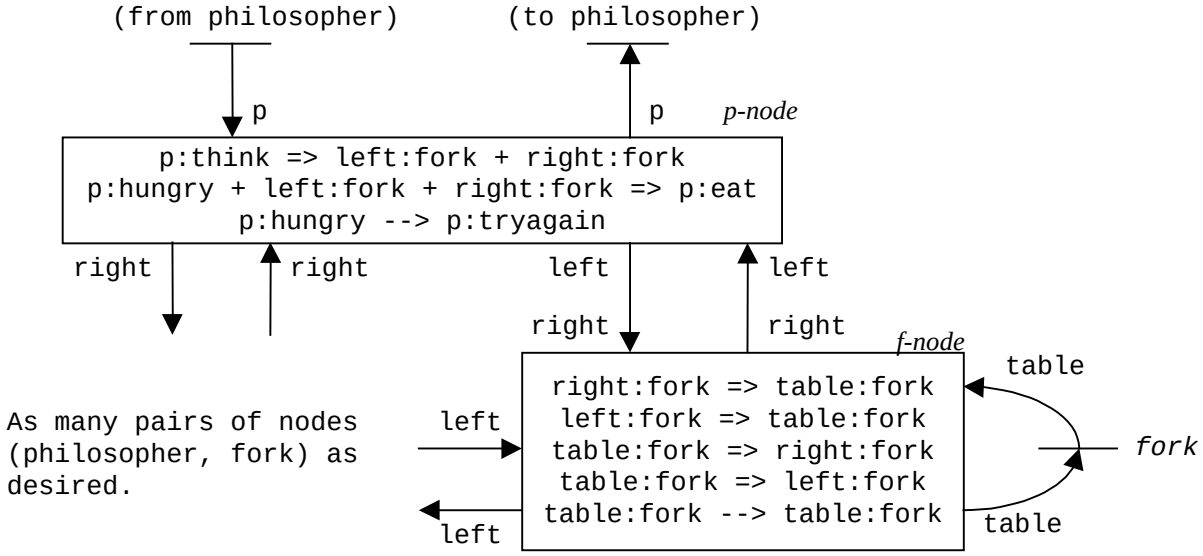
http://sdg.lcs.mit.edu/~jchapin

Current programming languages and instruction sets are based on computational models well-tuned to implement batch programs: programs that read input, process it, generate output, and halt. Wavenets are an attempt at a clean-sheet design tuned instead to implement parallel servers: programs that run forever and respond to multiple asynchronous inputs in parallel while maintaining their internal invariants. Wavenets are inspired by quantum mechanics, specifically state superposition, wave function collapse, and conservation of energy. However, wavenets are a computational model, *not* a proposal for a quantum computer architecture.

A wavenet is a synchronous circuit with the following special properties. When a signal arrives at a computational element *N* where one of multiple outputs must be chosen, the signal initially propagates to all outputs (superposition). Eventually one of the outputs is latched by a latch or contributes to a further computation whose output is latched. All other outputs generated by *N* for that signal are retracted, undoing any further computations in which they have participated (wave function collapse). Finally, all signals output by latches at the start of each machine cycle must be consumed at the end of the cycle, either by being latched or by contributing to a computation whose output is latched (conservation of energy). Surprisingly, conservation of energy is a powerful computational primitive that provides global concurrency control and constraint satisfaction.

Here is a wavenet that solves the dining philosophers problem. In this formalism, defined precisely in a white paper available on my website, wires carry symbols rather than binary signals. Bars across wires represent latches, which may have an initial state denoted by writing a symbol name next to the latch. Input from the external world is expressed as a latch with no input wire; output to the external world is a latch with no output wire. Production rules are the only operation. Production rules contain input terms, followed by an arrow and output terms. A rule activates when all the symbols named in its input terms have arrived on the corresponding named input ports of the node. When a rule activates, it produces the symbols named in its output terms on the corresponding named output ports of the node.

All production rules in a node whose input terms have been satisfied activate in parallel. However, after wave function collapse, each input symbol can only contribute to a single production in a node. That is, collapsing requires choosing which rule in each node will consume each symbol that has arrived on an input port of that node during the cycle. => indicates a regular production rule, while --> indicates a default rule that only fires if required to avoid violating conservation of energy.



Discussion

- The programmer or compiler need not express the usual locking and resource reservation protocols to solve the problem. In each machine cycle all input requests and available internal resources are considered simultaneously to find a wave function collapse that does not violate conservation of energy.

  Consider the fork symbol output by the latch at the right edge of the diagram. If neither the philosopher to its right nor the one to its left has sent a hungry request in this cycle, the default rule in the f-node must fire, leaving the fork on the table, because the outputs of the other rules in the f-node with `table:fork` as an input term cannot be consumed.

  Conversely, if a philosopher has sent a hungry request, the fork resource from the philosopher's left can only be consumed if the fork resource from the philosopher's right is also consumed. Therefore if the fork on the right is not available or is consumed elsewhere, the fork on the left will remain on the table and the default rule in the p-node will fire telling the philosopher to try again.
- By automating the global matching of requests with resources, wavenets promise much better composability of components than traditional computational models. An individual component like the f-node or p-node in the dining philosopher's solution merely states what it is capable of computing. Which of its possible computations will actually occur in a given cycle is determined by the global arrangement of components, available resources, and input requests. This suggests that wavenets may form the basis of a useful abstraction at the programming language level.
- The production rules in the above formalism can represent arbitrary functional computations, while the symbols can be arbitrarily complex data structures. One could design a system with a hybrid architecture where the wavenet component provides global control and the traditional component does all the processing work. Given a program expressed partially in terms of the wavenet abstraction, the hardware may execute it efficiently by mating a traditional ISA processing core to a FPGA or other application-specific custom circuit component.