CSE548: Readings for 1/14/13

Dan Butler
A Preliminary Architecture for a Basic Data-Flow Processor

• Basic idea:
  – Express a program as data consumer / producer nodes, wired together

• Exploits parallelism to overcome data latency
  – Nodes activate whenever their inputs are ready

• Architecture designed to simulate these nodes in an efficient way, storing / retrieving state as needed
\begin{verbatim}
input a, b
y := (a+b)/x
x := (a*(a+b)) + b
output y, x
\end{verbatim}
Packet erands, that is the processor will value the instantiated (Figure available).

In Figure 2, the value of the processor has been determined to be practical.

The value of the processor has been determined to be practical.

Data packets are directed to the operation unit where they are processed. The operation unit is composed of several operation units, each capable of performing a specific operation.

The operation unit is composed of several operation units, each capable of performing a specific operation.

Cells of the processor are connected to each other through the network. The network is responsible for distributing the data packets to the appropriate cells.

The network is responsible for distributing the data packets to the appropriate cells.

Each cell contains an instruction register which holds the operands for the current operation. The instruction register is connected to the memory, allowing the operands to be loaded from memory.

The instruction register is connected to the memory, allowing the operands to be loaded from memory.

The memory is responsible for storing the program instructions and the operands. The memory is accessed by the instruction register and the instruction units.

The memory is accessed by the instruction register and the instruction units.

The instruction units are responsible for performing the specified operations on the operands stored in the instruction register. The results of the operations are then sent back to the instruction register.

The instruction units are responsible for performing the specified operations on the operands stored in the instruction register. The results of the operations are then sent back to the instruction register.

The result of the operation is then sent back to the instruction register, completing the cycle.

The result of the operation is then sent back to the instruction register, completing the cycle.
Logic and Control Flow (special-cased)

Two-Level Memory Hierarchy
Questions on Dataflow Processor

• Why is control flow a special case?

• What is it like to program for this processor?
  – Seems tough!

• How would file I/O work?
Two Fundamental Limits on Dataflow Multiprocessing

• Criticizes dataflow processors for not paying attention to the memory hierarchy

• Argues:
  1. Optimal # of virtual processors limited by size of top-level cache (swapping context is costly)
  2. Purely local scheduling of processes is doomed
  3. Dataflow idea hides cost of swapping contexts
     - should be made explicit
Two Fundamental Limits on Dataflow Multiprocessing

‘Dataflow architectures essentially replace the small register number with a large tag that serves to “name” the value. A realistic view of the storage hierarchy requires that only a small number of such name/value pairs can be resident at a time. Once the number of VPs exceeds the capacity of the top level matching store, the synchronization cost increases dramatically, since some form of overflow store must be used[12, 6]’

• “overflow” is the two-level memory hierarchy
• Argues that this sort of switching can be very costly because processor speed comes from using the high-level cache effectively
Two Fundamental Limits on Dataflow Multiprocessing

• The propose another architecture, “Threaded Abstract Machine”
• No virtual processors
• Exposes the “fixed resources” of the actual processor
• Exposes scheduling to the compiler to allow for higher-level planning
Two Fundamental Limits on Dataflow Multiprocessing

• What are “throttling” and “k-bounding”?  
  – Help dataflow use the right amount of parallelism

• Why isn’t the dataflow two-level memory hierarchy a sufficient solution?

• How does this TAM business work in detail?
WaveScalar

• Dataflow architecture that does not switch contexts as much
  – Keeps data cells in high-level cache for long periods of time
• Allows users to program in a traditional von Neumann-style language
• Proposes a new type of locality – “dataflow locality” – and exploits it
  – Basically the fact that some data nodes need to talk to each other more than others
WaveScalar

• Consists of a physical network of processors with fast local caches
  – Divided into clusters so that dataflow-locality can be exploited
• Procedural code is compiled into chunks
• Each chunk is a dataflow-style node network
  – Called a “wave”
• Dataflow network distributed across processors
WaveScalar

• How does the compiling process really work, in detail?
  – loop unrolling?