**Dataflow Computers**

Motivation:
- exploit *instruction-level parallelism* on a massive scale
- more fully utilize all processing elements

Believed this was possible if:
- expose instruction-level parallelism by using a functional-style programming language
- no side effects; only restrictions were producer-consumer
- scheduled code for execution on the hardware greedily
- hardware support for data-driven execution

**Brief Review of Instruction-Level Parallelism (ILP)**

Fine-grained parallelism

Obtained by:
- instruction overlap
- executing instructions in parallel

In contrast to:
- loop-level parallelism (medium-grained)
- process-level or task-level or thread-level parallelism (coarse-grained)
Dataflow Execution

All computation is data-driven.
- binary is represented as a directed graph
- nodes are operations
- values travel on arcs
- WaveScalar instruction

Dataflow Closing

Data-dependent operations are connected, producer to consumer
Code & initial values loaded into memory
Execute according to the dataflow firing rule
- when operands of an instruction have arrived on all input arcs, instruction may execute
- value on input arcs is removed
- computed value placed on output arc

Dataflow Example

A[j + i*i] = i;
b = A[i*j];

Dataflow Closing

Control
- steer (s)
- merge (m)

Dataflow Closing

• convert control dependence to data dependence with value-steering instructions
• execute one path after condition variable is known (steer)
or
• execute both paths & pass values at end (merge)
Problems with Dataflow Computers

Language compatibility
- dataflow cannot guarantee a global ordering of memory operations
- dataflow computer programmers could not use mainstream programming languages, such as C
- developed special languages in which order didn’t matter

Scalability: large token store
- side-effect-free programming language with no mutable data structures
- each update creates a new data structure
- 1000 tokens for 1000 data items even if the same value
- delays in processing (only so many functional units, arbitration delays, etc.) meant delays in operand arrival
- associative search impossible; accessed with slower hash function
- aggravated by the state of processor technology at the time

Partial Solutions

Data representation in memory
- I-structures:
  - write once; read many times
  - early reads are deferred until the write
- M-structures:
  - multiple reads & writes, but they must alternate
  - reusable structures which could hold multiple values

Local (register) storage for back-to-back instructions in a single thread

Cycle-level multithreading
Partial Solutions

Frames of sequential instruction execution
• create "frames", each of which stored the data for one iteration or
  one thread
• not have to search entire token store (offset to frame)
• dataflow execution among coarse-grain threads

Partition token store & place each partition with a PE

Many solutions led away from pure dataflow execution