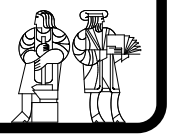
von Neumann vs. Dataflow

von Neumann: control-driven computing
- program as linear sequence of instructions
- directives: load, store, add, etc.
- ILP, if present, is implicit
- expensive inter-thread synchronization typical

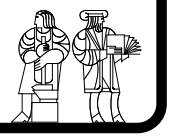
Dataflow: data-driven computing
- program as data-dependence graph
  + graph nodes are operations (add, etc.)
  + graph edges are "token" (data) queues
  + operations "fire" when all incoming arcs have tokens
- ILP explicit in the program representation
- very cheap synchronization (I-structures) essential



Dataflow Hardware

Pure dataflow:
- purely dynamic (data-driven) instruction scheduling
- program represented as "templates"
  + templates look like OOE scoreboards
- instruction executed when template filled in
- token "queues" of depth 1 (or small C)

Hybrid von Neumann/dataflow:
- compiler-scheduled RISC-y instruction sequences
- hardware-supported I-structures (synchronization)
- cheap thread creation/destruction



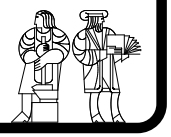
Some Issues in Dataflow Architectures

- synchronization
- moderating excessive parallelism
- efficient use of registers
- static vs. dynamic instruction scheduling
- hardware complexity
- side-effect management (or lack thereof)

Dataflow-related Papers

Papers you already have:

- pH programming language
- StarT architecture
- Tera architecture

Papers you just got:

- Arvind -- "The Evolution of Dataflow Architectures..."
- Iannucci -- "Toward a Dataflow/von Neumann Hybrid..."
- Culler -- "Resource Requirements of Dataflow Programs"