Data Flow Architecture

By Matthew Johnson
Overview

- History
- Importance
- Comparison to Tomasulo
- Demo
- Dataflow Overview:
  - Static
  - Dynamic
  - Hybrids
- Problems:
  - CAM
  - I-Structures
- Potential Applications
- Conclusion
Timeline

1964– 1st Scoreboard Computer: CDC 6600
1967– Tomasulo
1970– MIT Static Dataflow Processors: Tokens trigger next instruction
1970– Kahn Dataflow Processors: Sequential Processes communicate with each via messages through FIFO’s (Hybrid)
1980– MIT Dynamic Dataflow Processors
1985 – Exception Solutions in Out of Order Execution : Smith & Pleszkun devise precise exception handling
1986– Restricted Out of Order Execution > RISC: Yale Pratt publishes paper on in which he simulates a restricted OO architecture and it outperforms RISC benchmarks
1988– MIT Monsoon Processors
1990– IBM POWER1: 1st Out of Order Microprocessor (floating point only)
1995– Out of Order goes Mainstream

KEY
Data Flow
Hybrid
Out of Order
Why is Dataflow Important?

- Naturally “latency tolerant” because of its ability to dynamically switch between threads
- Eliminate processor switching problems
- Eliminates Data dependencies
- Naturally takes advantage of Inherent parallelism in code
## Dataflow Comparison

<table>
<thead>
<tr>
<th></th>
<th>Tomasulo</th>
<th>Dataflow</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Issue</strong></td>
<td>Issue instructions in order to form dependency graph</td>
<td>Dependency graph is preloaded into memory before launch</td>
</tr>
<tr>
<td><strong>Execution Launch</strong></td>
<td>Once operands arrive execution begins</td>
<td>Once operands arrive execution begins</td>
</tr>
<tr>
<td><strong>Result Broadcast</strong></td>
<td>Common data bus</td>
<td>Distribution Network (CAM)</td>
</tr>
<tr>
<td><strong>Data Hazards</strong></td>
<td>Extra hardware needed to avoided: RAW, WAR &amp; WAW</td>
<td>None (Driven by RAW)</td>
</tr>
<tr>
<td><strong>Programming Languages</strong></td>
<td>Instructions Sequential</td>
<td>Instructions are represented as nodes in a dependency graph</td>
</tr>
</tbody>
</table>


Notation

- **Token** – Output of an operation and its destination

- **Operation Packet** – contains instructions, destination and 2 operands

- **Instruction Cell** – The location in main memory that holds the operation packet
Example

\[
Y = 4
\]

\[Y = 1 + 3\]
\[X = 4 \times Y\]
\[Z = 4 \times X\]
Run Through

\[ Y = 1 + 3 \]
\[ X = 4 \times Y \]
\[ Z = 4 \times X \]
Flow Control Operators

MERGE node

SWITCH node
Branch
Dataflow Languages

- Describes dataflow graph
- Data tokens propagate along the arc
- Single Assignment Rule: variables can only be assigned once
- Examples: VAL, LUCID, ...

\[
\text{Mean} = \frac{x + y + z}{3}; \\
\text{StDev} = \sqrt{\frac{x^2 + y^2 + z^2}{3} - \text{Mean}^2};
\]
Two Main Types of Dataflow Architecture

- Static
- Dynamic
Static Dataflow

- Allows at most 1 data token on the arc of data flow
- To Ensure 1 token per arc, must use hand shaking
- Can only run 1 iteration of a loop at a time
- Static Pipeline:
  - Check instructions for operands
  - Select subset of available instruction
  - Update instructions that are complete
- Example: MIT Static Dataflow Machines

Diagram:
- Data tokens
- Acknowledge signals
- Data arcs
- Acknowledgement arcs
Static Dataflow Pros/Cons

Pros:
- Simple Model

Cons:
- Loops can not be run in parallel
- Token Traffic Double
- Hand shaking wastes time
- No support for procedure calls
Dynamic Dataflow

- Allows multiple data tokens on the arc of data flow
- Allows multiple loop iterations at one time
- An extra tag is added to tokens and contains address context and loop number
- Instruction Cells are changed to accept multiple tokens of different tags
- Overall effect is the creation of reentrant sub-graphs
- Examples: MIT Tagged-Token Data Flow, Manchester Dataflow
Dynamic Dataflow Pros/Cons

Pros:
- Greater Performance (No handshaking, multiple tokens/arc, loops can run in parallel)
- Supports procedure calls

Cons:
- Tokens need to be buffered because matching tokens takes LOTS of time (Large Buffer Needed)
- No instruction locality (can’t use registers)
Other Hybrids

- Threaded Dataflow– subgraphs with low degrees of parallelisms are converted into sequential thread

- Monsoon– Replaces associative memory with ram
  - Uses eight stage pipeline to inject instructions and execute enabled instructions
Problems

- Lack of locality make current storage hierarchy ineffective
- Matching tokens takes a large amount of time and requires buffering
- Building CAMs large enough to hold dependencies of a real program
- Data Structures are difficult to implement
CAM

- Content Addressable Memory
- Read—returns the address of the data
- Size is small, RAM is 8X (DRAM: 64 MB CAM: 8MB –single chip)
- Expensive
- Large footprint
- Excessive power
I – Structures

- Main problem: every write to a structure requires the entire structure to be rewritten

- Solution:
  - Define size of data structure
  - Each element has a series of status bits (states)
  - Can only write an element once
Applications

- Digital Signal Processing
- Network routing
- Graphics processing
- Telemetry
- Data warehousing
- Everywhere Parallel computing is used
Conclusion

- Potential to unlock the inherent parallelism of the algorithm
- Naturally eliminates data hazards
- Current technologies in CAM are not advanced enough to machines based in RAM
- Serious Problems still remain such as implementing data structures
Questions?
References