Encoding Dataflow Graphs and Tokens

**Conceptual**

```
   op1
     ^
   op2
     |  +  |
   op3  op4
```

**Encoding of graph**

<table>
<thead>
<tr>
<th>Program memory:</th>
<th>Op−code</th>
<th>Destination(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>109</td>
<td>op1</td>
<td>120L</td>
</tr>
<tr>
<td>113</td>
<td>op2</td>
<td>120R</td>
</tr>
<tr>
<td>120</td>
<td>+</td>
<td>141, 159L</td>
</tr>
<tr>
<td>141</td>
<td>op3</td>
<td>...</td>
</tr>
<tr>
<td>159</td>
<td>op4</td>
<td>... , ...</td>
</tr>
</tbody>
</table>

**Encoding of token:**

A "packet" containing:

- **120R** 6.847 Destination instruction address, Left/Right port Value

**Re−entrancy ("dynamic" dataflow):**

- Each invocation of a function or loop iteration gets its own, unique, "Context"
- Tokens destined for same instruction in different invocations are distinguished by a context identifier

- **120R Ctxt** 6.847 Destination instruction address, Left/Right port Context Identifier Value
MIT Tagged Token Dataflow Architecture

- Designed by Arvind et. al., MIT, early 1980’s
- Simulated; never built

Global view:

- Processor Nodes
  (including local program and "stack" memory)

- Interconnection Network

- Resource Manager Nodes

"I−Structure" Memory Nodes (global heap data memory)

- Resource Manager Nodes responsible for
  - Function allocation (allocation of context identifiers)
  - Heap allocation
  - etc.

- Stack memory and heap memory: globally addressed
Wait-Match Unit:
- Tokens for unary ops go straight through
- Tokens for binary ops: try to match incoming token and a waiting token with same instruction address and context id
  - **Success:** Both tokens forwarded
  - **Fail:** Incoming token $\rightarrow$ Waiting Token Mem, Bubble (no-op) forwarded
### Output Unit routes tokens:
- Back to local Token Queue
- To another Processor
- To heap memory based on the addresses on the token

■ **Tokens from network are placed in Token Queue**
MIT Tagged Token Dataflow Architecture
Support for "Remote Loads"

Conceptual:

Encoding of graph:

Program memory:

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Destination(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Fetch, 207</td>
</tr>
<tr>
<td>207</td>
<td>op1, ...</td>
</tr>
</tbody>
</table>
Multiple remote loads are no problem:
- Can be issued in parallel
- "Join" of responses is implicit in Wait-Match
Heap memory locations have FULL/EMPTY bits
- When "I-Fetch" request arrives (instead of "Fetch"), if the heap location is EMPTY, heap memory module queues request at that location
- Later, when "I-Store" arrives, pending requests are discharged
- "I-structure semantics"
  Note: no busy waiting, no extra messages
Unification of Dataflow & von Neumann Designs

**pure Dataflow**
- Static (Dennis & Misunas '74)
- Manchester Dataflow (Gurd & Watson 82)
- Sigma-1 (Shimada & Hiraki 87)
- Epsilon (Grafe et.al. 88)
- Epsilon 2 (Grafe & Hoch 89)
- "Static" (Dennis 91)
- Monsoon (Papadopoulos & Culler 88)
- EM 4 (Sakai 89)
- EM 5 (Kodama & Sakai 91)
- Monsoon Threads (Traub & Papadopoulos 91)

**multithreaded von Neumann**
- CDC 6600 1964
- HEP (B.Smith 78)
- Cosmic Cube (Seitz 85)
- iPSCs, CM-5
- J-Machine (Dally 87)
- Horizon Tera (B.Smith, et.al. 88–93)
- Alewife (Agarwal 89)
- MASA (Halstead & Fujita 88)
- EM 4
- VNDF Hybrid (Iannucci 88)
- Hybrid (Buehner & Ekanadham 87)
- Hybrid
- EMPIRE (Iannucci et. al. 89–91)
- TAM (Culler, et. al. 90)
- *T (Nikhil, Papadopoulos, Arvind & Greiner 91)

**Unified**
- Argument Fetching (Gao, et. al. 88)
- Alpha
- "Static" Message Passing

(adapted from a slide by Greg Papadopoulos)