Dynamic Dataflow

Arvind
Computer Science & Artificial Intelligence Lab
Massachusetts Institute of Technology

December 5, 2006
http://csg.csail.mit.edu/6.827/
Outline

- Static Dataflow Machines
  - Not general-purpose enough
- Dynamic Dataflow Machines
  - As easy to build as a simple pipelined processor
- The software view
  - The memory model: I-structures
- Monsoon and its performance

Dataflow Graphs

\[
\{ x = a + b; y = b \times 7; (x-y) \times (x+y) \}\]

- Values in dataflow graphs are represented as tokens
- An operator executes when all its input tokens are present; copies of the result token are distributed to the destination operators
- No separate control flow
Static Dataflow Machine: 
*Instruction Templates*

Each arc in the graph has a operand slot in the program.

```
<table>
<thead>
<tr>
<th>Opcode</th>
<th>Destination 1</th>
<th>Destination 2</th>
<th>Operand 1</th>
<th>Operand 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+</td>
<td></td>
<td>3L</td>
<td>4L</td>
</tr>
<tr>
<td>2</td>
<td>*</td>
<td></td>
<td>3R</td>
<td>4R</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td></td>
<td>5L</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>+</td>
<td></td>
<td>5R</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>*</td>
<td></td>
<td></td>
<td>out</td>
</tr>
</tbody>
</table>
```

Presence bits:

- 1
- 2
- 3
- 4
- 5

Op   dest1  dest2  p1  src1   p2   src2

Op   dest1  dest2  p1  src1   p2   src2

Many such processors can be connected together.

Programs can be statically divided among the processor.

December 5, 2006  http://csg.csail.mit.edu/6.827/  L22-5

December 5, 2006  http://csg.csail.mit.edu/6.827/  L22-6
Static Dataflow: Problems/Limitations

- Mismatch between the model and the implementation
  - The model requires *unbounded FIFO token queues* per arc but the architecture provides storage for one token per arc
  - The architecture *does not ensure FIFO order* in the reuse of an operand slot
  - The *merge* operator has a unique firing rule
- The static model *does not support*:
  - Function calls
  - Data Structures

Outline

- Static Dataflow Machines ✓
  - Not general-purpose enough
- Dynamic Dataflow Machines ←
  - As easy to build as a simple pipelined processor
- The software view
  - The memory model: I-structures
- Monsoon and its performance
Dynamic Dataflow Architectures

- Allocate instruction templates, i.e., a frame, dynamically to support each loop iteration and procedure call
  - termination detection needed to deallocate frames
- The code can be shared if we separate the code and the operand storage

The code can be shared if we separate the code and the operand storage.

A Frame in Dynamic Dataflow

Need to provide storage for only one operand/operator.
Monsoon Processor

Greg Papadopoulos

Temporary Registers & Threads

Robert Iannucci

n sets of registers (n = pipeline depth)

Registers evaporate when an instruction thread is broken

Registers are also used for exceptions & interrupts
Actual Monsoon Pipeline:  
*Eight Stages*

**Diagram:**
- Instruction Memory
- Presence bits
- Frame Memory
- Registers
- Instruction Fetch
- Effective Address
- Presence Bit Operation
- Frame Operation
- ALU
- Form Token
- User Queue
- System Queue
- Network

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction</td>
<td>Fetch</td>
</tr>
<tr>
<td>Memory</td>
<td></td>
</tr>
<tr>
<td>Presence bits</td>
<td></td>
</tr>
<tr>
<td>Frame Memory</td>
<td></td>
</tr>
<tr>
<td>Registers</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>Fetch</td>
</tr>
<tr>
<td>Memory</td>
<td></td>
</tr>
<tr>
<td>Presence bits</td>
<td></td>
</tr>
<tr>
<td>Frame Memory</td>
<td></td>
</tr>
<tr>
<td>Registers</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>Fetch</td>
</tr>
<tr>
<td>Memory</td>
<td></td>
</tr>
<tr>
<td>Presence bits</td>
<td></td>
</tr>
<tr>
<td>Frame Memory</td>
<td></td>
</tr>
<tr>
<td>Registers</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>Fetch</td>
</tr>
<tr>
<td>Memory</td>
<td></td>
</tr>
<tr>
<td>Presence bits</td>
<td></td>
</tr>
<tr>
<td>Frame Memory</td>
<td></td>
</tr>
<tr>
<td>Registers</td>
<td></td>
</tr>
</tbody>
</table>

Instructions directly control the pipeline

The opcode specifies an operation for each pipeline stage:

- **opcode**
- **r**
- **dest1**
- **[dest2]**

### EA - effective address
- **FP + r**: frame relative
- **r**: absolute
- **IP + r**: code relative (not supported)

### WM - waiting matching
- **Unary**: Normal; Sticky; Exchange; Imperative
- **Pbs X port**: Pbs X Frame op X ALU inhibit

### Register ops:
- **ALU**: \( V_L \times V_R \rightarrow V'_L \times V'_R, CC \)
- **Form token**: \( V_L \times V_R \times Tag_1 \times Tag_2 \times CC \rightarrow Token_1 \times Token_2 \)

*Easy to implement; no hazard detection*
Procedure Linkage Operators

Like standard call/return but caller & callee can be active simultaneously

Dynamic Dataflow Machines
- As easy to build as a simple pipelined processor

The software view
- The memory model: I-structures

Monsoon and its performance

Outline
- Static Dataflow Machines
  - Not general-purpose enough
- Dynamic Dataflow Machines
  - As easy to build as a simple pipelined processor
- The software view
  - The memory model: I-structures
  - Monsoon and its performance
Parallel Language Model

Global Heap of Shared Objects

Tree of Activation Frames

Active threads

Loop

asynchronous and parallel at all levels

Id World

*implicit parallelism*

Dataflow Graphs + I-Structures + ...

TTDA

Monsoon

*T

*T-Voyager
Data Structures in Dataflow

- Data structures reside in a structure store
  ⇒ tokens carry pointers

- I-structures: Write-once, Read multiple times or
  - allocate, write, read, ..., read, deallocate
   ⇒ No problem if a reader arrives before the writer at the memory location
I-Structure Storage: Split-phase operations & Presence bits

- Need to deal with multiple deferred reads
- Other operations: fetch/store, take/put, clear

Next time

Compiling Id/pH into dataflow graphs

Monsoon Performance