Dataflow: Passing the Token

Arvind
Computer Science and Artificial Intelligence Lab
M.I.T.

ISCA 2005, Madison, WI
June 6, 2005
Inspiration: Jack Dennis

General purpose parallel machines based on a dataflow graph model of computation

Inspired all the major players in dataflow during seventies and eighties, including Kim Gostelow and I @ UC Irvine
Dataflow Machines

Static
- mostly for signal processing
- NEC - NEDIP and IPP
- Hughes, Hitachi, AT&T, Loral, TI, Sanyo
- M.I.T. Engineering model...

Dynamic
- Manchester ('81)
- M.I.T. - TTDA, Monsoon ('88)
- M.I.T./Motorola - Monsoon ('91) (8 PEs, 8 IS)
- ETL - SIGMA-1 ('88) (128 PEs, 128 IS)
- ETL - EM4 ('90) (80 PEs), EM-X ('96) (80 PEs)
- Sandia - EPS88, EPS-2
- IBM - Empire...

Related machines:
- Burton Smith's
- Denelcor HEP, Horizon, Tera

Shown at
- Supercomputing 96
- Supercomputing 91

EM4: single-chip dataflow micro

Sigma-1: The largest dataflow machine

K. Hiraki
T. Shimada
S. Sakai
Y. Kodama

EM4: single-chip dataflow micro

John Gurd
Greg Papadopoulos
Boughton
Chris Joerg
Jack Costanza
Monsoon
Software Influences

• Parallel Compilers
  – Intermediate representations: DFG, CDFG (SSA, φ,...)
  – Software pipelining
    Keshav Pingali, G. Gao, Bob Rao, ..

• Functional Languages and their compilers

• Active Messages
  David Culler

• Compiling for FPGAs, ...
  Wim Bohm, Seth Goldstein...

• Synchronous dataflow
  – Lustre, Signal
    Ed Lee @ Berkeley
This talk is mostly about MIT work

- Dataflow graphs
  - A clean model of parallel computation
- 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

- Musings
Dataflow Graphs

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

- Values in dataflow graphs are represented as tokens
  
  token < ip, p, v >

- An operator executes when all its input tokens are present; copies of the result token are distributed to the destination operators

\begin{itemize}
  \item \textit{no separate control flow}
\end{itemize}
Dataflow Operators

- A small set of dataflow operators can be used to define a general programming language
Well Behaved Schemas

- Conditional
  - one-in-one-out & self cleaning
  - Needed for resource management

- Bounded Loop
  - Needed for resource management
Outline

• Dataflow graphs ✓
  – A clean model of parallel computation
• 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

• Musings
Static Dataflow Machine: 
Instruction Templates

Each arc in the graph has a operand slot in the program
Static Dataflow Machine

Jack Dennis, 1973

- Many such processors can be connected together
- Programs can be statically divided among the processor
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

- No easy solution in the static framework
- Dynamic dataflow provided a framework for solutions
Outline

• Dataflow graphs ✓
  – A clean model of parallel computation
• 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

• Musings
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

```
a token
<fp, ip, port, data>
```

```
frame pointer
instruction pointer
```
A Frame in Dynamic Dataflow

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3L, 4L</td>
<td>3R, 4R</td>
<td>5L, 5R</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Program:

1. + 1
2. * 2
3. - 3
4. + 4
5. * 5

Frame:

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

Instruction Fetch
Operand Fetch
Form Token
Network

op r d1,d2
Code
Frames

ip

fp+r

Token Queue

ALU

Network
Temporary Registers & Threads

Robert Iannucci

Registers evaporate when an instruction thread is broken.

Registers are also used for exceptions & interrupts.

n sets of registers (n = pipeline depth)

Robert Iannucci
Actual Monsoon Pipeline: Eight Stages

- Instruction Fetch
- Effective Address
- Presence Bit Operation
- Frame Operation
- Form Token

Diagram nodes:
- Instruction Memory
- Presence bits
- Frame Memory
- Registers
- User Queue
- System Queue
- ALU
- Network

Arvind - 18
Instructions directly control the pipeline

The opcode specifies an operation for each pipeline stage:

\[
\text{opcode} \quad r \quad \text{dest1} \quad [\text{dest2}]
\]

- **EA**: effective address
  - \( \text{FP} + r \): frame relative
  - \( r \): absolute
  - \( \text{IP} + r \): code relative (not supported)

- **WM**: waiting matching
  - Unary; Normal; Sticky; Exchange; Imperative

- **Register ops**:
  - **ALU**: \( V_L \times V_R \rightarrow V'_L \times V'_R \), CC

- **Form token**: \( V_L \times V_R \times \text{Tag}_1 \times \text{Tag}_2 \times \text{CC} \rightarrow \text{Token}_1 \times \text{Token}_2 \)

Easy to implement; no hazard detection
Procedure Linkage Operators

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

Graph for f

1: 

n: 

change Tag 0

change Tag 1

Fork

change Tag 0

change Tag 1
Outline

• Dataflow graphs √
  – A clean model of parallel computation
• 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
• Musings
Parallel Language Model

Tree of Activation Frames

Global Heap of Shared Objects

asynchronous and parallel at all levels

active threads

loop

f: g: h:
Id World

*implicit parallelism*

---

Dataflow Graphs + I-Structures + . . .

- TTDA
- Monsoon
- *T
  - *T-Voyager
Id World people

- Rishiyur Nikhil
- Keshav Pingali
- Vinod Kathail
- David Culler
- Ken Traub
- Steve Heller
- Richard Soley
- Dinart Mores
- Jamey Hicks
- Alex Caro
- Andy Shaw
- Boon Ang
- Shail Anditya
- R Paul Johnson
- Paul Barth
- Jan Maessen
- Christine Flood
- Jonathan Young
- Derek Chiou
- Arun Iyargar
- Zena Ariola
- Mike Bekerle
- K. Eknadham (IBM)
- Wim Bohm (Colorado)
- Joe Stoy (Oxford)
- ...

ISCA, Madison, WI, June 6, 2005
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
Outline

- Dataflow graphs ✓
  - A clean parallel model of computation
- 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 ←

- Musings
The Monsoon Project
Motorola Cambridge Research Center + MIT

Research Prototypes
16 2-node systems (MIT, LANL, Motorola, Colorado, Oregon, McGill, USC, ...)
2 16-node systems (MIT, LANL)

Id World Software

I-structureMonsoon

Processor
64-bit
10M tokens/sec 4M 64-bit words
100 MB/sec

16-node Fat Tree

Tony Dahbura
Id Applications on Monsoon @ MIT

• Numerical
  – Hydrodynamics - SIMPLE
  – Global Circulation Model - GCM
  – Photon-Neutron Transport code -GAMTEB
  – N-body problem

• Symbolic
  – Combinatorics - free tree matching, Paraffins
  – Id-in-Id compiler

• System
  – I/O Library
  – Heap Storage Allocator on Monsoon

• Fun and Games
  – Breakout
  – Life
  – Spreadsheet
Id Run Time System (RTS) on Monsoon

- **Frame Manager**: Allocates frame memory on processors for procedure and loop activations
  
  Derek Chiou

- **Heap Manager**: Allocates storage in I-Structure memory or in Processor memory for heap objects.
  
  Arun Iyengar
### Single Processor Monsoon Performance Evolution

One 64-bit processor (10 MHz) + 4M 64-bit I-structure

<table>
<thead>
<tr>
<th></th>
<th>Feb. 91</th>
<th>Aug. 91</th>
<th>Mar. 92</th>
<th>Sep. 92</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Matrix Multiply</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>500x500</td>
<td>4:04</td>
<td>3:58</td>
<td>3:55</td>
<td>1:46</td>
</tr>
<tr>
<td><strong>Wavefront</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>500x500, 144 iters.</td>
<td>5:00</td>
<td>5:00</td>
<td>3:48</td>
<td></td>
</tr>
<tr>
<td><strong>Paraffins</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n = 19</td>
<td>.50</td>
<td>.31</td>
<td></td>
<td>.02.4</td>
</tr>
<tr>
<td>n = 22</td>
<td></td>
<td></td>
<td></td>
<td>.32</td>
</tr>
<tr>
<td><strong>GAMTEB-9C</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40K particles</td>
<td>17:20</td>
<td>10:42</td>
<td>5:36</td>
<td>5:36</td>
</tr>
<tr>
<td>1M particles</td>
<td>7:13:20</td>
<td>4:17:14</td>
<td>2:36:00</td>
<td>2:22:00</td>
</tr>
<tr>
<td><strong>SIMPLE-100</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 iterations</td>
<td>.19</td>
<td>.15</td>
<td>.10</td>
<td>.06</td>
</tr>
<tr>
<td>1K iterations</td>
<td>4:48:00</td>
<td></td>
<td></td>
<td>1:19:49</td>
</tr>
</tbody>
</table>

*Need a real machine to do this*

- hours:minutes:seconds

ISCA, Madison, WI, June 6, 2005
## Monsoon Speed Up Results

*Boon Ang, Derek Chiou, Jamey Hicks*

<table>
<thead>
<tr>
<th></th>
<th>speed up</th>
<th>critical path (millions of cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1pe</td>
<td>2pe</td>
</tr>
<tr>
<td>Matrix Multiply 500 x 500</td>
<td>1.00</td>
<td>1.99</td>
</tr>
<tr>
<td>Paraffins n=22</td>
<td>1.00</td>
<td>1.99</td>
</tr>
<tr>
<td>GAMTEB-2C 40 K particles</td>
<td>1.00</td>
<td>1.95</td>
</tr>
<tr>
<td>SIMPLE-100 100 iters</td>
<td>1.00</td>
<td>1.86</td>
</tr>
</tbody>
</table>

September, 1992

*Could not have asked for more*
## Base Performance?
### Id on Monsoon vs. C / F77 on R3000

<table>
<thead>
<tr>
<th></th>
<th>MIPS (R3000) (x 10\text{e}6 cycles)</th>
<th>Monsoon (1pe) (x 10\text{e}6 cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix Multiply</td>
<td>954 +</td>
<td>1058</td>
</tr>
<tr>
<td>500 x 500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paraffins</td>
<td>102 +</td>
<td>322</td>
</tr>
<tr>
<td>n=22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAMTEB-9C</td>
<td>265 *</td>
<td>590</td>
</tr>
<tr>
<td>40 K particles</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMPLE-100</td>
<td>1787 *</td>
<td>4682</td>
</tr>
<tr>
<td>100 iters</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **MIPS codes won’t run on a parallel machine without reccompilation/recoding**
- **8-way superscalar?**
  - Unlikely to give 7 fold speedup

R3000 cycles collected via Pixie

* Fortran 77, fully optimized + MIPS C, O = 3

64-bit floating point used in Matrix-Multiply, GAMTEB and SIMPLE
The Monsoon Experience

- Performance of implicitly parallel Id programs scaled effortlessly.

- Id programs on a single-processor Monsoon took 2 to 3 times as many cycles as Fortran/C on a modern workstation.
  - Can certainly be improved

- Effort to develop the *invisible software* (loaders, simulators, I/O libraries,....) *dominated* the effort to develop the *visible software* (compilers...*)
Outline

• Dataflow graphs √
  – A clean parallel model of computation
• 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 √

• Musings ←
What would we have done differently - 1

• Technically: Very little
  – Simple, high performance design, easily exploits fine-grain parallelism, tolerates latencies efficiently
  – Id preserves fine-grain parallelism which is abundant
  – Robust compilation schemes; DFGs provide easy compilation target

• Of course, there is room for improvement
  – Functionally several different types of memories (frames, queues, heap); all are not full at the same time
  – Software has no direct control over large parts of the memory, e.g., token queue
  – Poor single-thread performance and it hurts when single thread latency is on a critical path.
What would we have done differently - 2

- Non technical but perhaps even more important
  - It is difficult enough to cause one revolution but two? Wake up?
  - Cannot ignore market forces for too long – may affect acceptance even by the research community
  - Should the machine have been built a few years earlier (in lieu of simulation and compiler work)? Perhaps it would have had more impact (had it worked)
  - The follow on project should have been about:
    1. Running conventional software on DF machines, or
    2. About making minimum modifications to commercial microprocessors (We chose 2 but perhaps 1 would have been better)
Imperative Programs and Multi-Cores

• Deep pointer analysis is required to extract parallelism from sequential codes
  – otherwise, extreme speculation is the only solution

• A multithreaded/dataflow model is needed to present the found parallelism to the underlying hardware

• Exploiting fine-grain parallelism is necessary for many situations, e.g., producer-consumer parallelism
Locality and Parallelism: Dual problems?

- Good performance requires exploiting both

- Dataflow model gives you parallelism for free, but requires analysis to get locality

- C (mostly) provides locality for free but one must do analysis to get parallelism
  - Tough problems are tough independent of representation
Parting thoughts

• Dataflow research as conceived by most researchers achieved its goals
  – The model of computation is beautiful and will be resurrected whenever people want to exploit fine-grain parallelism

• But installed software base has a different model of computation which provides different challenges for parallel computing
  – Maybe possible to implement this model effectively on dataflow machines – we did not investigate this but is absolutely worth investigating further
  – Current efforts on more standard hardware are having lots of their own problems
  – Still an open question on what will work in the end
Thank You!

and thanks to

R.S.Nikhil, Dan Rosenband, James Hoe, Derek Chiou, Larry Rudolph, Martin Rinard, Keshav Pingali

for helping with this talk
DFGs vs CDFGs

- Both Dataflow Graphs and Control DFGs had the goal of structured, well-formed, compositional, executable graphs.

- CDFG research (70s, 80s) approached this goal starting with original sequential control-flow graphs ("flowcharts") and data-dependency arcs, and gradually adding structure (e.g., $\phi$-functions).

- Dataflow graphs approached this goal directly, by construction:
  - Schemata for basic blocks, conditionals, loops, procedures.

- CDFGs is an Intermediate representation for compilers and, unlike DFGs, not a language.