Merrimac
Scientific Computing with Streams

Bill Dally
Computer Systems Laboratory
Stanford University
November 20, 2003
I’m not going to talk about

- Optimizing topology
- Worst-case routing
- Best-known adaptive routing
- High-radix routers
Just in time for Christmas
## The Efficiency Gap

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Raw</th>
<th>Realized</th>
</tr>
</thead>
<tbody>
<tr>
<td>$/GFLOPS</td>
<td>0.50</td>
<td>1,000</td>
</tr>
<tr>
<td>W/GFLOPS</td>
<td>0.05</td>
<td>10</td>
</tr>
</tbody>
</table>
Streaming scientific computation exploits the capabilities of VLSI

- Modern VLSI technology makes arithmetic cheap
  - 100s of GFLOPS/chip today TFLOPS in 2010
- But bandwidth is expensive
- Streams change the ratio of arithmetic to bandwidth
  - By exposing producer-consumer locality
    - Cannot be exploited by caches – no reuse, no spatial locality
- Streams also expose parallelism
  - To keep 100s of FPUs per processor busy
- High-radix networks reduce cost of bandwidth when its needed
  - Simplifies programming
- Net result
  - 10x to 100x performance per unit cost
  - Scalable from workstation to supercomputer
To Exploit VLSI Technology We Need:

- **Parallelism**
  - To keep 100s of ALUs per chip (thousands/board millions/system) busy

- **Latency tolerance**
  - To cover 500 cycle remote memory access time

- **Locality**
  - To match 20Tb/s ALU bandwidth to ~100Gb/s chip bandwidth

- **Moore’s Law**
  - Growth of transistors, not performance

Arithmetic is cheap, global bandwidth is expensive
Local << global on-chip << off-chip << global system
Stream Architecture Makes Communication Explicit – Exploits Parallelism and Locality
Stream Programming Exposes Parallelism and Locality

- **Locality**
  - Producer-consumer within a kernel (local register file)
  - Producer-consumer across kernels (on-chip SRF)
  - Stream locality (spatial)

- **Parallelism**
  - Data Level Parallelism across stream elements
  - Task Parallelism across kernels
Streamed Applications Exploit Producer-Consumer Locality to Exploit BW Hierarchy

Stream program matches application to BW Hierarchy 1:1:6:100
A streaming supercomputer exploits the arithmetic density of VLSI to realize an efficiency of $6/GFLOPS and $3/M-GUPS

Capability AND Capacity
A PFLOPS machine with only 8,192 nodes (or less)
A TFLOPS workstation for < $20,000 (parts cost)
Architecture of a Streaming Supercomputer

Board 32
Backplane 2
32 Boards
512 Nodes
64K FPUs
64TFLOPS
1TByte

Board 2
16 Nodes
1K FPUs
2TFLOPS
32GBytes

Node 16

Node 2

Board 2

Node

16 x DRDRAM
2GBytes

Stream Processor
128 FPUs
128GFLOPS

16GBytes/s

16GBytes/s
32+32 pairs

16GBytes/s

32GBytes

64GBytes/s
128+128 pairs
6" Teradyne GbX

64GBytes/s

1TBytes/s
2K+2K links
Ribbon Fiber

All links 5Gb/s per pair or fiber
All bandwidths are full duplex

E/O

O/E

On-Board Network

Intra-Cabinet Network

Inter-Cabinet Network

Bisection 32TBytes/s

Merrimac: 11
Nov 20, 2003
Bandwidth taper matches capabilities of VLSI to demands of scientific applications.
High Radix Routers Enable Economical Global Memory

- Flat memory bandwidth within a 16-node board
- 4:1 Concentration within a 32-node backplane, 8:1 across a 32 backplane system
- Routers with bandwidth $B=640\text{Gb/s}$ route messages with length $L=128\text{b}$
  - Requires high radix to exploit
Merrimac Processor

- 90nm tech (1 V)
- ASIC technology
- 1 GHz (37 FO4)
- 128 GOPs
- Inter-cluster switch between clusters
- 127.5 mm² (small ~12x10)
  - Stanford Imagine is 16mm x 16mm
  - MIT Raw is 18mm x 18mm
- 25 Watts (P4 = 75 W)
  - ~41W with memories
Merrimac Power Estimates

- MADD ALU: 7.0W
- Network Controller: 5W
- Memory Controller: 5W
- Switches: 1.3W
- Cache: 1W
- SRF: 1.4W
- LRF: 1.3W
- MADD ALU: 7.0W
- Scalar CPU: 2.2W
- Microcontroller: 0.6W
- DRAM: 16W
### Bandwidth Hierarchy Enabled by Streams

<table>
<thead>
<tr>
<th>Level</th>
<th>Bandwidth per Node (GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster registers</td>
<td>3,840</td>
</tr>
<tr>
<td>Stream register file</td>
<td>512</td>
</tr>
<tr>
<td>Stream cache</td>
<td>64</td>
</tr>
<tr>
<td>Local Memory</td>
<td>16</td>
</tr>
<tr>
<td>Board Memory (16 Nodes)</td>
<td>16</td>
</tr>
<tr>
<td>Cabinet Memory (1K Nodes)</td>
<td>4</td>
</tr>
<tr>
<td>Global Memory (16K Nodes)</td>
<td>2</td>
</tr>
</tbody>
</table>
## Rough Per-Node Budget

<table>
<thead>
<tr>
<th>Item</th>
<th>Cost</th>
<th>Per Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor chip</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>Router chip</td>
<td>200</td>
<td>69</td>
</tr>
<tr>
<td>Memory chip</td>
<td>20</td>
<td>320</td>
</tr>
<tr>
<td>Board</td>
<td>1000</td>
<td>63</td>
</tr>
<tr>
<td>Router Board</td>
<td>1000</td>
<td>2</td>
</tr>
<tr>
<td>Backplane</td>
<td>5000</td>
<td>10</td>
</tr>
<tr>
<td>Global Router Board</td>
<td>5000</td>
<td>5</td>
</tr>
<tr>
<td>Power</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>Per-Node Cost</td>
<td></td>
<td>718</td>
</tr>
<tr>
<td>$/GFLOPS (128/node)</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>$/M-GUPS (250/node)</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

Preliminary numbers, parts cost only, no I/O included
Merrimac Borrows from Imagine Prototype

- **Imagine**
  - Stream processor for image and signal processing
  - 16mm die in 0.18um TI process
  - 21M transistors
  - Collaboration with TI ASIC

- **Software tools based on Stream-C/Kernel-C**
  - Stream scheduler
  - Communication scheduling

- **Many Applications**
  - 3 Graphics pipelines
  - Image-processing apps – depth, MPEG
  - 3G Cellphone (Rice)
  - STAP
Gentle Slope to Port Applications to Merrimac

like OpenMP
Brook and Brooktran – Stream Extensions with Multiple Dimensions and Irregular Grids

• Stream code intermixed with “scalar” code
  – C – Brook Fortran – Brooktran
  – Easy migration - only port the time-consuming kernels
• Locality and Parallelism
  – Producer-consumer locality
  – No global memory references within a kernel
  – No retained state in kernels (reductions allowed)
  – Stream elements are processed independently
• Stream operators for defining neighbors
• Gather/Scatter and GatherOp/ScatterOp
Stream Compiler Achieves Near Optimum Kernel Performance

ComputeCellInterior Kernel from StreamFEM application
Stream Compiler Reduces Bandwidth Demand Compared to Caching

StreamFEM application
Capable Hardware Makes Software Easier

- Nearly flat global bandwidth (10:1)
  - Supports non-local applications
  - Reduces need to partition and place application

- Fine grain remote access
  - Single word gather/scatter
  - Eliminates need to restructure application for spatial locality

- Leaves the programmer to focus on
  - Writing a correct program
  - Expressing parallelism
Several scientific applications have been demonstrated on a stream processor simulator.

They all exhibit stream locality and achieve a high fraction of peak performance.
Scientific Streamed Applications

- **StreamFLO** is a streaming version of FLO82, [Jameson], for the solution of the inviscid flow around an airfoil
  - Uses a cell centered finite volume formulation with a **multigrid** acceleration to solve the 2D Euler equations
- **StreamFEM** implementation of the Discontinuous Galerkin (DG) Finite Element Method (FEM) (Tim Barth, NASA)
  - 2D or 3D triangulated domains
  - Increasingly complex PDEs
    - Scalar advection (1 PDE), Euler (4 PDEs), Magnetohydrodynamics (6 PDEs)
  - Piecewise polynomial function
    - Constant (1 dof), Linear (3 dof), Quadratic (6 dof), Cubic (10 dof)
- **StreamMD** molecular dynamics simulation
  - Box of water molecules
  - Electrostatic and Van der Waals interactions
  - Gridded to accelerate approximate force calculation
Summary of Application Results

<table>
<thead>
<tr>
<th>Application</th>
<th>Sustained GFLOPS</th>
<th>FP Ops / Mem Ref</th>
<th>LRF Refs</th>
<th>SRF Refs</th>
<th>Mem Refs</th>
</tr>
</thead>
<tbody>
<tr>
<td>StreamFEM3D¹</td>
<td>31.6</td>
<td>17.1</td>
<td>153.0M (95.0%)</td>
<td>6.3M (3.9%)</td>
<td>1.8M (1.1%)</td>
</tr>
<tr>
<td>(Euler, quadratic)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>StreamFEM3D¹</td>
<td>39.2</td>
<td>13.8</td>
<td>186.5M (99.4%)</td>
<td>7.7M (0.4%)</td>
<td>2.8M (0.2%)</td>
</tr>
<tr>
<td>(MHD, constant)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>StreamMD¹</td>
<td>14.2²</td>
<td>12.1²</td>
<td>90.2M (97.5%)</td>
<td>1.6M (1.7%)</td>
<td>0.7M (0.8%)</td>
</tr>
<tr>
<td>(grid algorithm)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROMACS¹</td>
<td>22.0²</td>
<td>7.1²</td>
<td>181M (95.4%)</td>
<td>5.3M (2.8%)</td>
<td>3.4M (1.8%)</td>
</tr>
<tr>
<td>StreamFLO</td>
<td>12.9²</td>
<td>7.4²</td>
<td>234.3M (95.7%)</td>
<td>7.2M (2.9%)</td>
<td>3.4M (1.4%)</td>
</tr>
</tbody>
</table>

1. Simulated on a machine with 64GFLOPS peak performance
2. The low numbers are a result of many divide and square-root operations
Summary

• The problem is bandwidth – arithmetic is cheap
  – Forget GFLOPS,
  – forget traditional P:M:C ratios – balance by incremental return
• Stream processing can provide cost-effective scientific computing
  – Streams expose concurrency and locality
  – The register hierarchy of a stream processor exploits them
  – Enables compiler optimization at a larger scale than scalar processing
  – $6/GFLOPS sustained on ‘local’ parts of applications
• A high-radix network provides high global bandwidth
  – $3/M-GUPS on ‘non-local’ parts of applications
• Demonstrated on several applications
  – 50% of peak except where divide/sqrt limited
• Scalable
  – 2 TFLOPS board for to 32-backplane 2 PFLOPS machine
Acknowledgements

• Faculty
  – Bill Dally
  – Ron Fedkiw
  – Pat Hanrahan
  – Mark Horowitz
  – Mendel Rosenblum
  – Juan Alonso
  – Eric Darve

• Collaborators
  – Massimiliano Fatica (ME)
  – Tim Barth (NASA)
  – Alan Wray (NASA)
  – Rich Lethin (Reservoir)
  – Peter Mattson (Reservoir)
  – Charlie Garret (Reservoir)

• Sponsor
  – DOE, NNSA – ASCI Alliance Program

• Students
  (Architecture & Prog Sys)
  – Jung Ho Anh
  – Ian Buck
  – Abhishek Das
  – Mattan Erez
  – Jayanth Gummaraju
  – Amit Gupta
  – Nuwan Jayasena
  – Tim Knight
  – Francois Labonte
  – Manman Ren
  – Brian Towles
  – Arjun Singh

* Names in blue contributed primarily to applications