Stream Processing
Efficient Computing in the Many-Core Era

Bill Dally
Computer Science Department
Stanford University
EE380 Seminar
November 1, 2006
Computer Architecture in the Many-Core Era

• We are rapidly moving into the many-core era
• Data movement is what matters, not arithmetic
• Hierarchical architecture enables optimization of data movement
• Need explicit control over data movement (bulk operations)
  – Caches themselves are not the solution
• Experience shows this works for signal/image processing (Imagine/SPI) and Scientific Computing (Merrimac)
• Carrying the hierarchy to a fine-grain will close the efficiency gap with ASICs
Computer Architecture in the Many-Core Era

• We are rapidly moving into the many-core era
• Data movement is what matters, not arithmetic
• Hierarchical architecture enables optimization of data movement
• Need explicit control over data movement (bulk operations)
  – Caches themselves are not the solution
• Experience shows this works for signal/image processing (Imagine/SPI) and Scientific Computing (Merrimac)
• Carrying the hierarchy to a fine-grain will close the efficiency gap with ASICs
60 years of Von Neumann Architecture is at an end
ILP is mined out – end of superscalar processors
Time for a new architecture

Computers are power limited

Source: S Borkar, Intel
Cores/chip expected to double every 18 months
Intel says 80 cores by 2010
Many core from embedded devices to desktops to supercomputers
Do we have all the answers?

- 50 years of parallel computing research
- We know
  - Interconnection networks
  - Comm & Sync mechanisms
  - Simple parallel programming
- We still don’t know
  - How to exploit locality
    - Beyond caches and domain decomposition
  - How to convert serial programs
    - May not be relevant in the long run
- And the problem has changed
Reinvention of multi-core should benefit from past research

MAP Chip (1997)
3 cores x 6 threads/core
Efficient mechanisms
It’s not hard to write a parallel program.

It is hard to write an *efficient* parallel program.

The right hardware and programming system can make this much easier.
Computer Architecture in the Many-Core Era

• We are rapidly moving into the many-core era
• **Data movement is what matters, not arithmetic**
• Hierarchical architecture enables optimization of data movement
• Need explicit control over data movement (bulk operations)
  - Caches themselves are not the solution
• Experience shows this works for signal/image processing (Imagine/SPI) and Scientific Computing (Merrimac)
• Carrying the hierarchy to a fine-grain will close the *efficiency gap* with ASICs
Arithmetic is cheap, Communication is expensive

- **Arithmetic**
  - Can put 100s of FPUs on a chip
  - $0.50/GFLOPS, 50mW/GFLOPS
  - Exploit with **parallelism**

- **Communication**
  - Dominates cost
    - $8/GW/s 2W/GW/s (off-chip)
  - BW decreases (and cost increases) with distance
  - Power increases with distance
  - Latency increases with distance
  - But can be hidden with parallelism
  - Need **locality** to conserve global bandwidth
Cost of data access varies by 1000x

<table>
<thead>
<tr>
<th>From</th>
<th>Energy</th>
<th>Cost*</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Register</td>
<td>10pJ</td>
<td>$0.50</td>
<td>1ns</td>
</tr>
<tr>
<td>Chip Region (2mm)</td>
<td>50pJ</td>
<td>$2</td>
<td>4ns</td>
</tr>
<tr>
<td>Global on Chip (15mm)</td>
<td>200pJ</td>
<td>$10</td>
<td>20ns</td>
</tr>
<tr>
<td>Off chip (node mem)</td>
<td>1nJ</td>
<td>$50</td>
<td>200ns</td>
</tr>
<tr>
<td>Global</td>
<td>5nJ</td>
<td>$500</td>
<td>1us</td>
</tr>
</tbody>
</table>

*Cost of providing 1GW/s of bandwidth
All numbers approximate
Performance = Parallelism

Efficiency = Locality
Computer Architecture in the Many-Core Era

- We are rapidly moving into the many-core era
- Data movement is what matters, not arithmetic
- Hierarchical architecture enables optimization of data movement
- Need explicit control over data movement (bulk operations)
  - Caches themselves are not the solution
- Experience shows this works for signal/image processing (Imagine/SPI) and Scientific Computing (Merrimac)
- Carrying the hierarchy to a fine-grain will close the efficiency gap with ASICs
So we should build chips that look like this
An abstract view

Global Memory

Switch

LM
CM

Switch

RM

Switch

RM

Switch

RM

Switch

RM

Switch

Switch

Switch

Global Memory
Now that we’ve enabled control of data movement, how do we optimize it?
Computer Architecture in the Many-Core Era

• We are rapidly moving into the many-core era
• Data movement is what matters, not arithmetic
• Hierarchical architecture enables optimization of data movement
• Need explicit control over data movement (bulk operations)
  – Caches themselves are not the solution
• Experience shows this works for signal/image processing (Imagine/SPI) and Scientific Computing (Merrimac)
• Carrying the hierarchy to a fine-grain will close the efficiency gap with ASICs
Conventional Wisdom: Use caches

Global Memory

Switch

LM
CM

Switch

RM
RM
RM

Switch

RRRR
AAA
Caches squander bandwidth – our scarce resource

- Unnecessary data movement
- Poorly scheduled data movement
  - Idles expensive resources waiting on data

- More efficient to map programs to an explicit memory hierarchy
Example – Simplified Finite-Element Code

loop over cells
  flux[i] = ...

loop over cells
  ... = f(flux[i], ...)

Diagram:

- **Cells**
  - Gather to **Cells**
  - Scatter to **Cells**
- **Flux**
  - Connect to **Cells**
- **fn1**
- **fn2**
- **SRFs**
- **LRFs**
- **DRAM**
Explicitly block into SRF

loop over cells
flux[i] = ...

loop over cells
... = f(flux[i],...)

Flux passed through SRF, no memory traffic
Explicitly block into SRF

loop over cells
flux[i] = ...

loop over cells
... = f(flux[i],...)

Explicit re-use of Cells, no misses
Stream loads/stores (bulk operations) hide latency
(1000s of words in flight)
Explicit storage enables simple, efficient execution

All needed data and instructions on-chip
no misses
Caches lack predictability
(controlled via a “wet noodle”)
Caches are controlled using a “wet noodle”

99% hit rate, 1 miss costs 100s of cycles, 10,000s of ops
So how do we program an explicit hierarchy?
Stream Programming: Parallelism, Locality, and Predictability

- **Parallelism**
  - Data parallelism across stream elements
  - Task parallelism across kernels
  - ILP within kernels

- **Locality**
  - Producer/consumer
  - Within kernels

- **Predictability**
  - Enables scheduling
Evolution of Stream Programming

1997  StreamC/KernelC
Break programs into kernels
Kernels operate only on input/output streams and locals
Communication scheduling and stream scheduling

2001  Brook
Continues the construct of streams and kernels
Hides underlying details
Too “one-dimensional”

2005  Sequoia
Generalizes kernels to “tasks”
Tasks operate on local data
Local data “gathered” in an arbitrary way
“Inner” tasks subdivide, “leaf” tasks compute
Machine-specific details factored out
Explicit storage enables simple, efficient execution unit scheduling

One iteration

- ComputeCellInt kernel from StreamFem3D
- Over 95% of peak with simple hardware
- Depends on explicit communication to make delays predictable
Stream scheduling exploits explicit storage to reduce bandwidth demand

StreamFEM application

Prefetching, reuse, use/def, limited spilling
Sequoia – Generalize Kernels into Leaf Tasks

- Perform actual computation
- Analogous to kernels
- “Small” working set

```c
void __task matmul::leaf( __in    float A[M][P],
                          __in    float B[P][N],
                          __inout float C[M][N] )
{
    for (int i=0; i<M; i++) {
        for (int j=0; j<N; j++) {
            for (int k=0; k<P; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}
```
Inner tasks

- Decompose to smaller subtasks
  - Recursively
- “Larger” working sets

```c
void __task matmul::inner( __in float A[M][P],
                           __in float B[P][N],
                           __inout float C[M][N] )
{
    tunable unsigned int U, X, V;
    blkset Ablks = rchop(A, U, X);
    blkset Bblks = rchop(B, X, V);
    blkset Cblks = rchop(C, U, V);

    mappar (int i=0 to M/U, int j=0 to N/V)
    mapreduce (int k=0 to P/X)
        matmul(Ablks[i][k],Bblks[k][j],Cblks[i][j]);
}
```
Computer Architecture in the Many-Core Era

• We are rapidly moving into the many-core era
• Data movement is what matters, not arithmetic
• Hierarchical architecture enables optimization of data movement
• Need explicit control over data movement (bulk operations)
  – Caches themselves are not the solution
• Experience shows this works for signal/image processing (Imagine/SPI) and Scientific Computing (Merrimac)
• Carrying the hierarchy to a fine-grain will close the efficiency gap with ASICs
Imagine VLSI Implementation

• **Chip Details**
  - 2.56cm² die, 0.15um process, 21M transistors, 792-pin BGA
  - Collaboration with TI ASIC
  - Chips arrived on April 1, 2002

• **Dual-Imagine test board**
Application Performance (cont.)

Diagram showing execution time breakdown for different applications.

- Host bandwidth stalls
- Stream controller overhead
- Memory stalls
- Cluster stalls
- Kernel non main loop
- Kernel main loop overhead
- Operations
Applications match the bandwidth hierarchy
Merrimac Supercomputer

- **8 x XDR-DRAM**
  - 2GBytes

- **Stream Processor**
  - 64 FPU
  - 128 GFLOPS

- **On-Board Network**
  - 64GBytes/s
  - 32+32 pairs

- **E/O**
  - 48GBytes/s
  - 128+128 pairs
  - 6” Teradyne GbX

- **Inter-Cabinet Network**
  - 768GBytes/s
  - 2K+2K links
  - Ribbon Fiber

- **Bisection**
  - 24TBytes/s

- **Backplane**
  - 32 Boards
  - 512 Nodes
  - 16K FPUs
  - 32TFLOPS
  - 512GBytes

- **Board 32**
  - 16GBytes
  - 1TFLOPS

- **Backplane 64**
## Merrimac Application Results

<table>
<thead>
<tr>
<th>Application</th>
<th>Developed</th>
<th>Compiled</th>
<th>Sustained GFLOP/s</th>
<th>Speedup over Pentium4</th>
<th>Efficiency vs. Pentium4</th>
</tr>
</thead>
<tbody>
<tr>
<td>StreamFEM</td>
<td>Brook</td>
<td>StreamC</td>
<td>69</td>
<td></td>
<td></td>
</tr>
<tr>
<td>StreamMD</td>
<td>C</td>
<td>StreamC</td>
<td>46</td>
<td>16.1</td>
<td>34.5</td>
</tr>
<tr>
<td>StreamFLO</td>
<td>Brook</td>
<td>Metacompiled</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>StreamCDP</td>
<td>Fortran</td>
<td>StreamC</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DGEMM</td>
<td>StreamC</td>
<td>StreamC</td>
<td>117</td>
<td>17.7</td>
<td>37.9</td>
</tr>
<tr>
<td>CONV2D</td>
<td>StreamC</td>
<td>StreamC</td>
<td>79</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFT3D</td>
<td>StreamC</td>
<td>StreamC</td>
<td>37</td>
<td>18.7</td>
<td>40.1</td>
</tr>
<tr>
<td>StreamSPAS</td>
<td>C</td>
<td>StreamC</td>
<td>3.1</td>
<td>6</td>
<td>12.6</td>
</tr>
</tbody>
</table>
Imagine – 20x perf/cost and perf/power as best DSPs

Merrimac – 10-20x perf/cost and 30x perf/power as clusters and supercomputers
Many-Core Architecture

- Communication is expensive, arithmetic is cheap
  - Parallelism to exploit arithmetic
  - Locality to conserve bandwidth
- Hierarchical architecture enables optimization of data movement
- Explicitly manage this hierarchy
  - Makes efficient use of scarce, expensive resources
  - Enables optimization
- Experience shows this works for signal/image processing (Imagine/SPI) and Scientific Computing (Merrimac)
  - 10x -30x perf/cost and perf/power advantage
  - Efficient programming systems
- Parallelism and Locality are the **big** issues with multicore processors
  - Stream organization and programming make it easy to deal with them