Parallelism isn't Enough: An Architect's Perspective on Building and Programming Terascale Processors and Petascale Systems

Mattan Erez
The University of Texas at Austin

LACSS 2008, Programming Models Workshop
October 15, 2008
Outline

• Power is the number one concern
  – A word on reliability and cost in general
• Parallelism isn’t enough
  – Properties of efficient VLSI
• Locality, Parallelism, and Hierarchical control
• Threading and streaming models
• Memory systems
• Programming models
Supercomputer performance outpaces Moore’s law VLSI scaling

The figure shows the sustained FLOP/s ( Floating Point Operations Per Second ) of supercomputers from 1993 to 2017. The lines represent different categories of supercomputers:

- **Num. 1**
- **Num. 10**
- **Num. 500**
- **Idealized VLSI**

The y-axis represents the sustained FLOP/s, ranging from $1 \times 10^8$ to $1 \times 10^{18}$. The x-axis represents the years from 1993 to 2017.
Power is the Dominant Architectural Problem

- Bad news: power scaling is slowing down
  - Can’t scale Vt much in order to control leakage
    - New technology helps
  - Can’t scale Vdd as much
  - Power doesn’t go down as it used to

- Energy/device decreases slower than devices/chip

- Power goes up if performance scaling continues
  - For same processor architecture

- Roadrunner: 1PFLOP/2MW, BG/L 0.5PFLOP/2MW
  - How much for many PFLOPS?
There's more to a system than power

• Building systems is about optimizing utility/cost
  • Power plays an increasing role
    – Power determines much of operating cost
    – Power determines much of acquisition cost
      • Cooling and facilities

• Reliability
  – Likelihood of faults is growing, especially soft errors
  – Fault-tolerance == opportunity cost
  – Fault tolerant techniques are scalable

• Higher power leads to more failures (soft and hard)

• Bandwidth and compute density
How Can We Reduce Power?

• Compute less
  – Use better algorithms

• Waste less
  – Don’t build/use unnecessary hardware
  – No unnecessary operations
  – No unnecessary data movement
  – Tuning can help – minimize power per acceptable performance goal

• Specialize more
  – Specialized circuits are more efficient
  – Tuning can help decide when
Parallelism isn’t enough

Parallelism, Locality, and efficient Hierarchical control
Wasting Less – Effective Performance in VLSI

- **Parallelism**
  - 10s of FPUs per chip
  - Efficient control

- **Locality**
  - Locality lowers power
  - Reuse reduces global BW

- **Throughput Design**
  - Throughput oriented I/O
  - Tolerate Increasing on-/off-chip latencies

- **Minimum control overhead**
### Bandwidth Dominates Energy Consumption

<table>
<thead>
<tr>
<th>Operation</th>
<th>65nm</th>
<th>32nm</th>
<th>16nm</th>
</tr>
</thead>
<tbody>
<tr>
<td>64b FP Operation</td>
<td>38</td>
<td>12.5</td>
<td>4.2</td>
</tr>
<tr>
<td>Read 64b from 16KB Cache</td>
<td>17.5</td>
<td>5.3</td>
<td>2</td>
</tr>
<tr>
<td>Transfer 64b across chip (10mm, Rep.)</td>
<td>179</td>
<td>179</td>
<td>179</td>
</tr>
<tr>
<td>Transfer 64b across chip (10mm, Cap.)</td>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>Transfer 64b off chip</td>
<td>154</td>
<td>115</td>
<td>100</td>
</tr>
</tbody>
</table>

Locality/Communication are key; Even then, performance is power-bound
Building for Locality, Parallelism, and Efficient Control

ALUs

on-chip memory

registers

control

interconnect
Locality & parallelism are easy to first order
Control offers a few more options

- Data Level Parallelism
  - Amortize control with SIMD

- Instruction Level Parallelism
  - Amortize control with static scheduling

- Thread (Task) Level Parallelism
  - Scalable
Data-Level Parallelism

- SIMD
- Independent indexing per FPU
- Full crossbar between FPUs
- No sub-word operation
Data- and Instruction-Level Parallelism

- A group of FPUs = A Processing Element (PE) = A Cluster
- VLIW
- Hierarchical switch provides area efficiency
Data-, Instruction- and Thread-Level

• Sequencer group
  – Each instruction sequencer runs different kernels
Many reasonable hardware options for 64-bit
Application Performance

Small performance differences for “good streaming” applications
So far so good

- Fairly generic with some nice results
- Describes just about all throughput architectures
  - NVIDIA
  - ATI
  - Stream processors (Merrimac)
  - Cell
  - Niagara
  - Larrabee
- So where are the differences?
Wasting Less – Effective Performance in VLSI

- **Parallelism**
  - 10s of FPUs per chip
  - Efficient control

- **Locality**
  - Locality lowers power
  - Reuse reduces global BW

- **Throughput Design**
  - Throughput oriented I/O
  - Tolerate Increasing on-/off-chip latencies

- Minimum control overhead
Another level of control hierarchy

• Different sequencer groups (threads/tasks) need to coordinate
  – Typically done by a single master
    – Scalar core (Cell, Merrimac)
    – Thread dispatcher (NVIDIA, ATI)
    – Program (Larrabee)

• Parallel program = sequence of parallel steps
The temporal dimension complicates things

• Need to hide latency

• Need parallelism in time
  – Threading
  – Streaming
Threading and streaming are duals with respect to sharing and partitioning state.
Threading and streaming are duals with respect to sharing and partitioning state

- Tradeoff in managing state
  - Threading: partitioned registers – the best memory
  - Streaming: partition local memory – problems with dynamic reuse

Differences in namespaces \(\rightarrow\) SPs can have more efficient control and memory systems
Finally, Programming Models: Expose what’s important to hardware

Ignore what isn’t!
Hierarchical programming model view

Application layer: numerical methods, DSLs

Portability and tuning layer: locality, parallelism, hierarchical control

Architecture layer: locality, parallelism, hierarchical control

Physical component layer: power, bandwidth, performance
Portability and tuning layer

• Need massive parallelism
  – Spatial and temporal
• Locality is critical
  – Doesn’t imply streaming or threading
• Hierarchy is key
• Arbitrarily communicating threads are insane
• Nested bulk synchronous
• Atomic regions (or operations)

Common canonical model for both streaming and threading!
Summary: What should and shouldn’t be exposed?

• Should not:
  – Inter-node communication
    • Hierarchy targets distribution, not directly exposed
    • Single global address space within each level
  – Intricacies of memory system
    • #channels, #banks, line-sizes, …
  – Explicit synchronization
    • Just atomics and barriers

• Should:
  – Locality, parallelism, and hierarchical control
  – Precision/accuracy
    • Word size
    • Fault tolerance
  – Dynamic irregularity?
Conclusions

• Power is everything
  – Bandwidth and performance requirements also

• Locality, parallelism, and hierarchical control
  – Good proxy for power, bandwidth, and performance

• Convergence/divergence
  – Throughput-architecture “dominate”
  – Threading and streaming are duals

• Layered system/programming model
  – Portability and tuning layer is key 😊

• Nested bulk synchronous + atomics
  – Target both stream and thread variants and enable opt.

• Don’t expose memory details, do expose locality
Conclusions

• Power is everything
  – Bandwidth and performance requirements also

• Locality, parallelism, and hierarchical control
  – Good proxy for power, bandwidth, and performance

• Convergence/divergence
  – Throughput-architecture “dominate”
  – Threading and streaming are duals

• Layered system/programming model
  – Portability and tuning layer is key 😊

• Nested bulk synchronous + atomics
  – Target both stream and thread variants and enable opt.

• Don’t expose memory details, do expose locality
Backup

• Stream processors are more efficient
Stream Processors have minimalistic dynamic control

Decoupling enables efficient static architecture
Separate address spaces (MEM/SRF/LRF)
Stream Architecture Features

• Exposed deep locality hierarchy
  – explicit software control over data allocation and data movement
  – flexible on-chip storage for capturing locality
  – staging area for long-latency bulk memory transfers

• Exposed parallelism
  – large number of functional units
  – latency hiding
Stream Architecture Features

• Exposed deep locality hierarchy
  – software managed data movement (communication)

• Exposed parallelism
  – large number of functional units and latency hiding

• Predictable instruction latencies
• Optimized static scheduling
• High sustained performance
Stream Architecture Features

- Exposed locality hierarchy
  - software managed data movement
- Exposed parallelism
  - high sustained performance
- Most instructions manipulate data
- Minimal hardware control structures
  - no branch prediction
  - no out-of-order execution
  - no trace-cache/decoded cache
  - simple bypass networks
  - ...

Efficient hardware → greater software responsibility
Current State of the Art in Stream Software Systems

• Kernel/Stream 2-level programming model
  – Good kernel scheduling
SPs decouples memory and execution enables static optimization and reduces hardware
Current State of the Art in Stream* Software Systems

* Stream model as defined earlier

- Kernel/Stream 2-level programming model
  - Good kernel scheduling
  - Decent SRF allocation and stream operation scheduling
  - Minor success otherwise

- Sequoia
  - Extends to more than 2 levels

- Great auto-tuning opportunities
  - Perfect knowledge of execution pipeline timing
  - Explicit communication
  - Experiments in Sequoia and StreamC

Stream processing simplifies tuning but demands more from the software system and programmer
Stream Compiler Reduces Bandwidth Demand Compared to Caching

StreamFEM application

- Gathered Elements
- Face Geometry
- Read-Only Table Lookup Data (Master Element)
- Compute Flux States
- Compute Numerical Flux
- Gather Cell
- Numerical Flux Gather
- Cell Geometry
- Compute Cell Interior
- Elements (Current)
- Elements (New)
- Advance Cell

- Cell Orientations

- 690
- 700
- 710
- 720
- 730
- 740
- 750
- 760
- 770
- 780
- 790
- 800

LACSS'08, Programming Models

© 2008 Mattan Erez
Explicit stream architecture enables effective resource utilization
What Streams Well?

• Data parallel in general?

• Data – control decoupled algorithms
  – No data $\rightarrow$ control $\rightarrow$ data dependence

• Work in progress
  – Traversing data structures in general
  – Dynamic block sizes (data-dependent output rates)

• Later on
  – Building data structures
  – Dynamic data structures