CoDesign

John Shalf
Computer Science and Data Sciences (CSDS) Department Head
NERSC CTO
Lawrence Berkeley National Laboratory Computing Research Division

Technology Disruptions on the Path to Exascale

Gigaflops to Teraflops was highly disruptive
• Moved from vector machines to MPPs with message passing
• Required new algorithms and software

Teraflops to Petaflops was not very disruptive
• Continued with MPI+Fortran/C/C++ with incremental advances

Petaflops to Exaflops will be highly disruptive
• No clock increases $\rightarrow$ hundreds of simple “cores” per chip
• Less memory and bandwidth $\rightarrow$ cores are not MPI engines
• x86 too energy intensive $\rightarrow$ more technology diversity (GPUs/accel.)
• Programmer controlled memory hierarchies likely

Computing at every scale will be transformed (not just exascale)
Traditional Sources of Performance Improvement are Flat-Lining

- Moore's Law is alive and well
- 15 years of exponential clock speed growth has ended
- How to use the transistors?
  - Industry Response: #cores per chip doubles every 18 months instead of clock frequency!
  - Power is now the leading design constraint!!
  - Resulting technology disruption forces redesign of many aspects of our computing environment!

But It's Going to Be Very Hard!
(Requires a Laboratory-Wide Strategy for Next Decade)

Current Technology Roadmaps will Depart from Historical Performance Gains

Without major changes, computing cannot continue historical trends of performance improvement
… and the power costs will still be staggering

$1M per megawatt per year! (with CHEAP power)

Technology Disruptions for Next Decade

Power limits growth in clock rates
- Moving to exponential growth in parallelism

Data movement costs exceeding cost of flops
- Wire Power= Bitrate * Length / Wiresize
- Diminished memory and interconnect bandwidth

Memory density not scaling at historical rates
- Diminished memory per core due to cost
- Slower growth in system memory (motivates hybrid model)

More components means failure rates
- Per-component reliability based on Windows fail rates
- Cannot scale current practices for checkpoint/restart

Can’t scale disk spindles or strict POSIX I/O

Disruptive changes require redesign of machine hand-in-hand with redesign of our software and algorithms
Power is an Industry Wide Problem
(2% of US power consumption and growing)

“Hiding in Plain Sight, Google Seeks More Power”,
by John Markoff, June 14, 2006

New Google Plant in The Dulles, Oregon,
from NYT, June 14, 2006

Relocate to Iceland?

The Move Towards Manycore

Cubic power improvement with lower clock rate due to $V^2F$

Slower clock rates enable use of simpler cores

Simpler cores use less area (lower leakage) and reduce cost

Tailor design to application to REDUCE WASTE

This is how iPhones and MP3 players are designed to maximize battery life and minimize cost
The Move Towards Manycore

Power5 (server)
- 120W@1900MHz
- Baseline

Intel Core2 sc (laptop):
- 15W@1000MHz
- 4x more FLOPs/watt than baseline

Intel Atom (handhelds)
- 0.625W@800MHz
- 80x more

Tensilica XTensa DP (Moto Razor):
- 0.09W@600MHz
- 400x more (80x-120x sustained)

Even if each simple core is 1/4th as computationally efficient as complex core, you can fit hundreds of them on a single chip and still be 100x more power efficient.
Future of On-Chip Architecture
(San Diego Meeting, December 2009)

Reduce power using simpler cores
• Power = C * V^2 * ClockFrequency
• Simpler Cores == fewer transistors

~1000-10k simple cores /Chip
• 4-8 wide SIMD or VLIW bundles
• Either 4 or 50+ HW threads

On-chip communication Fabric
• Low-degree topology for on-chip communication (torus or mesh)
• Scale cache coherence?
• Global (nonCC memory)
• Shared register file (clusters)

Off-chip communication fabric
• Integrated directly on an SoC
• Reduced component counts
• Coherent with TLB (no pinning)

Scale-out for Planar geometry

Move to Massive Parallelism

Future computing must move to simpler power-efficient core designs
• Embedded/consumer electronics technology is central to the future of HPC
• Convergence inevitable because it optimizes both cost and power efficiency

Consequence is massive on-chip parallelism
• A thousand cores on a chip by 2018
• 1 Million to 1 Billion-way System Level Parallelism
• Need to think in terms of “nodes” instead of “cores” or will go crazy
• Need to consider MPI+X models (with locality awareness)
The problem with Wires:
Energy to move data proportional to distance

Cost to move a bit on copper wire:
- \[ \text{Power} = \text{bitrate} \times \frac{\text{Length}}{\text{cross-section-area}} \]

Wire data capacity constant as feature size shrinks

Cost to move bit proportional to distance

\(~1-5\text{TByte/sec max feasible off-chip BW (10-20GHz/pin)}\)

Photonics is a wildcard

- Photonics requires no redrive and passive switch little power
- Copper requires to signal amplification even for on-chip connections

Cost of Data Movement

\[ \text{Power} = \text{bitrate} \times \frac{\text{Length}}{\text{cross-section-area}} \]

Energy Efficiency will require careful management of data locality

Important to know when you are on-chip and when data is off-chip!
**Locality Management is Key**

**Vertical Locality Management**

**Horizontal Locality Management**

*Can no longer assume everything is “equidistant”*

---

**Industry Trends in Fault Resilience**

*It’s proportional to # discrete chips in system -- not # cores (it’s going to get worse, but not ridiculously worse)*

Industry must maintain constant FIT rate per node
- ~1000 failures in time

Moore’s law gets us 100x improvement
- But still have to increase node count by 10x

So we will own 10x worse FIT rate
- MTTI 1 week to 1 day
- MTTI 1 day to 1 hour

Localized checkpointing
- LLNL SCR to node-local NVRAM
- More user-assistance in identifying what data to checkpoint
The Challenge

*How to get 1000x performance in 10 years? with a finite development budget?*

*How do you make it “programmable?”*

A Revolution is Underway
*(and it’s already happening…)*

Rapidly Changing Technology Landscape

- **Evolutionary** change between nodes *(10x more explicit parallelism)*
- **Revolutionary** change within node *(100x more parallelism, with diminished memory capacity and bandwidth)*
- Multiple Technology Paths *(GPU, manycore/embedded, x86/PowerX)*

The technology disruption will be pervasive *(not just exascale)*

- **Assumptions that our current software infrastructure is built upon are no longer valid**
- Applications, Algorithms, System Software will all break
- As significant as migration from vector to MPP *(early 90’s)*

Need a new approach to ensuring continued application performance improvements

- This isn’t just about Exaflops – this is for all system scales
Potential System Parameters for Exascale
(DOE Computer Architecture Summit, Nov. 2009)

<table>
<thead>
<tr>
<th>Systems</th>
<th>2009</th>
<th>2011</th>
<th>2015-2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>System peak</td>
<td>2 Peta</td>
<td>20 Peta</td>
<td>100-200 Peta</td>
</tr>
<tr>
<td>System memory</td>
<td>0.3 PB</td>
<td>1.6 PB</td>
<td>5 PB</td>
</tr>
<tr>
<td>Node performance</td>
<td>125 GF</td>
<td>200 GF</td>
<td>200-400 GF</td>
</tr>
<tr>
<td>Node memory BW</td>
<td>25 GB/s</td>
<td>40 GB/s</td>
<td>100 GB/s</td>
</tr>
<tr>
<td>Node concurrency</td>
<td>12</td>
<td>32</td>
<td>O(100)</td>
</tr>
<tr>
<td>Interconnect BW</td>
<td>1.5 GB/s</td>
<td>22 GB/s</td>
<td>25 GB/s</td>
</tr>
<tr>
<td>System size (nodes)</td>
<td>18,700</td>
<td>100,000</td>
<td>500,000</td>
</tr>
<tr>
<td>Total concurrency</td>
<td>225,000</td>
<td>3,200,000</td>
<td>O(50,000,000)</td>
</tr>
<tr>
<td>Storage</td>
<td>15 PB</td>
<td>30 PB</td>
<td>150 PB</td>
</tr>
<tr>
<td>IO</td>
<td>0.2 TB/s</td>
<td>2 TB/s</td>
<td>10 TB/s</td>
</tr>
<tr>
<td>MTTI</td>
<td>days</td>
<td>days</td>
<td>days</td>
</tr>
<tr>
<td>Power</td>
<td>6 MW</td>
<td>~10 MW</td>
<td>~10 MW</td>
</tr>
</tbody>
</table>

Some Exascale Proposals

<table>
<thead>
<tr>
<th>Property</th>
<th>Discussed Today</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Nodes</td>
<td>O(M)</td>
</tr>
<tr>
<td>FLOPS/Node</td>
<td>O(1-10 TF)</td>
</tr>
<tr>
<td>Threads/Node</td>
<td>O(1,000)</td>
</tr>
<tr>
<td>Total Memory Capacity</td>
<td>10 PB</td>
</tr>
<tr>
<td>Memory B/W (node)</td>
<td>&gt;400 GB/sec</td>
</tr>
<tr>
<td>Network B/W (node)</td>
<td>50 GB/sec/direction/link</td>
</tr>
<tr>
<td>Power</td>
<td>20 MW (firm)</td>
</tr>
</tbody>
</table>

YIKES!
Channels, Memory Parallelism, etc? Better be a 1 TF node
Depends on # of links and Injection B/W
### Exascale Proposals

<table>
<thead>
<tr>
<th>Property</th>
<th>Discussed Today</th>
<th>NVIDIA/Daily Proposal</th>
<th>SNL App Portfolio Design Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Nodes</td>
<td>O(M)</td>
<td>49,152 ~2E single/.65E double</td>
<td>100k-250k</td>
</tr>
<tr>
<td>FLOPS/Node</td>
<td>O(1-10TF)</td>
<td>40 TF Single 13 TF Double</td>
<td>4TF - 10 TF</td>
</tr>
<tr>
<td>Threads/Node</td>
<td>O(1,000)</td>
<td>2,400 O(1,000s)</td>
<td></td>
</tr>
<tr>
<td>Total Memory Capacity</td>
<td>10 PB</td>
<td>6.4 PB 0.003-0.01 B/F</td>
<td>50-100 PB 0.05 - 1 B/F</td>
</tr>
<tr>
<td>Memory B/W node</td>
<td>&gt;400 GB/sec</td>
<td>2 TB/sec 0.05 - 0.15 B/F</td>
<td>2-4 TB/sec 0.2 - 1 B/F</td>
</tr>
<tr>
<td>Network B/W node</td>
<td>50 GB/sec/direction/link</td>
<td>1 TB/sec 0.025 - 0.07 B/F</td>
<td>500-1,000 GB/sec 0.05 - 0.25 B/F</td>
</tr>
<tr>
<td>Power</td>
<td>20 MW (firm)</td>
<td>10 MW</td>
<td>20-30 MW</td>
</tr>
</tbody>
</table>

Even an aggressive proposal by a major vendor should be considered too timid for this program!

---

### Systems

<table>
<thead>
<tr>
<th>Systems</th>
<th>2009</th>
<th>2015 +1/-0</th>
<th>2018 +1/-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>System peak</td>
<td>2 Peta</td>
<td>100-300 Peta</td>
<td>1 Exa</td>
</tr>
<tr>
<td>Power</td>
<td>6 MW</td>
<td>~15 MW</td>
<td>~20 MW</td>
</tr>
<tr>
<td>System memory</td>
<td>0.3 PB</td>
<td>5 PB</td>
<td>64 PB (+)</td>
</tr>
<tr>
<td>Node performance</td>
<td>125 GF</td>
<td>0.5 TF or 7 TF</td>
<td>1-2 or 10TF</td>
</tr>
<tr>
<td>Node memory BW</td>
<td>25 GB/s</td>
<td>1-2TB/s</td>
<td>2-4TB/s</td>
</tr>
<tr>
<td>Node concurrency</td>
<td>12</td>
<td>O(100) O(1k) or 10k</td>
<td></td>
</tr>
<tr>
<td>Total Node Interconnect BW</td>
<td>3.5 GB/s</td>
<td>100-200 GB/s</td>
<td>10:1 vs memory bandwidth 2:1 alternative 200-400 GB/s (1:4 or 1:8 from memory BW)</td>
</tr>
<tr>
<td>System size (nodes)</td>
<td>18,700</td>
<td>50,000 or 500,000</td>
<td>O(100,000) or O(1M)</td>
</tr>
<tr>
<td>Total concurrency</td>
<td>225,000</td>
<td>O(100,000,000) *O(10) - O(50) to hide latency O(billion) * O(10) to O(100) for latency hiding</td>
<td></td>
</tr>
<tr>
<td>Storage</td>
<td>15 PB</td>
<td>150 PB</td>
<td>500-1000 PB (&gt;10x system memory is min)</td>
</tr>
<tr>
<td>IO</td>
<td>0.2 TB</td>
<td>10 TB/s</td>
<td>60 TB/s (how long to drain the machine)</td>
</tr>
<tr>
<td>MTTI</td>
<td>days</td>
<td>O(1day)</td>
<td>O(1 day)</td>
</tr>
</tbody>
</table>

60 MW over budget 2-3x$ over budget OOPPs!
## Potential System Architectures

### What is Possible

<table>
<thead>
<tr>
<th></th>
<th>2009</th>
<th>2015 +1/-0</th>
<th>2018 +1/-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>System peak</td>
<td>2 Peta</td>
<td>100-300 Peta</td>
<td>1 Exa</td>
</tr>
<tr>
<td>Power</td>
<td>6 MW</td>
<td>~15 MW</td>
<td>~20 MW</td>
</tr>
<tr>
<td>System memory</td>
<td>0.3 PB</td>
<td>5 PB</td>
<td>64 PB (+)</td>
</tr>
<tr>
<td>Node performance</td>
<td>125 GF</td>
<td>0.5 TF or 7 TF</td>
<td>2 TF or 10TF</td>
</tr>
<tr>
<td>Node memory BW</td>
<td>25 GB/s</td>
<td>0.2TB/s or 0.5TB/s</td>
<td>0.4TB/s or 1TB/s</td>
</tr>
<tr>
<td>Node concurrency</td>
<td>12</td>
<td>O(100)</td>
<td>O(1k) or 10k</td>
</tr>
<tr>
<td>Total Node Interconnect BW</td>
<td>3.5 GB/s</td>
<td>100-200 GB/s</td>
<td>200-400GB/s (1:4 or 1:8 from memory BW)</td>
</tr>
<tr>
<td>System size (nodes)</td>
<td>18,700</td>
<td>50,000 or 500,000</td>
<td>O(100,000) or O(1M)</td>
</tr>
<tr>
<td>Total concurrency</td>
<td>225,000</td>
<td>O(100,000,000) *O(10)-O(50) to hide latency</td>
<td>O(billion) * O(10) to O(100) for latency hiding</td>
</tr>
<tr>
<td>Storage</td>
<td>15 PB</td>
<td>150 PB</td>
<td>500-1000 PB (&gt;10x system memory is min)</td>
</tr>
<tr>
<td>IO</td>
<td>0.2 TB</td>
<td>10 TB/s</td>
<td>60 TB/s (how long to drain the machine)</td>
</tr>
<tr>
<td>MTTI</td>
<td>days</td>
<td>O(1day)</td>
<td>O(1 day)</td>
</tr>
</tbody>
</table>

### Changing Notion of “System Balance”

If you pay 5% more to double the FPUs and get 10% improvement, it’s a win (despite lowering your % of peak performance)

If you pay 2x more on memory BW (power or cost) and get 35% more performance, then it’s a net loss (even though % peak looks better)

**Real example**: we can give up ALL of the flops to improve memory bandwidth by 20% on the 2018 system

We have a fixed budget
- Sustained to peak FLOP rate is wrong metric if FLOPs are cheap
- Balance involves balancing your checkbook & balancing your power budget
- Requires a application co-design make the right trade-offs
Design Space Exploration and CoDesign

Let's come up with a more organized way to assess design trade-offs.

Use architecture simulation to predict costs of design choices.

Understand impact of design choices on algorithms!
Understand impact of algorithms on design choices!

Technology Challenges for the Next Decade

Power is leading constraint for future performance growth.

Parallelism is growing at exponential rate.

Reliability going down for large-scale systems, but also to get more energy efficiency for small systems.

Memory Technology improvements are slowing down.

By 2018, cost of a FLOP will be less than cost of moving 5nm across the chip’s surface (locality will really matter).
What's wrong with current SW env.?

Current programming systems have WRONG optimization targets

<table>
<thead>
<tr>
<th>Old Constraints</th>
<th>New Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Peak clock frequency</strong> as primary limiter for performance improvement</td>
<td><strong>Power</strong> is primary design constraint for future HPC system design</td>
</tr>
<tr>
<td><strong>Cost:</strong> FLOPs are biggest cost for system: optimize for compute</td>
<td><strong>Cost:</strong> Data movement dominates: optimize to minimize data movement</td>
</tr>
<tr>
<td><strong>Concurrency:</strong> Modest growth of parallelism by adding nodes</td>
<td><strong>Concurrency:</strong> Exponential growth of parallelism within chips</td>
</tr>
<tr>
<td><strong>Memory scaling:</strong> maintain byte per flop capacity and bandwidth</td>
<td><strong>Memory Scaling:</strong> Compute growing 2x faster than capacity or bandwidth</td>
</tr>
<tr>
<td><strong>Locality:</strong> MPI+X model (uniform costs within node &amp; between nodes)</td>
<td><strong>Locality:</strong> must reason about data locality and possibly topology</td>
</tr>
<tr>
<td><strong>Uniformity:</strong> Assume uniform system performance</td>
<td><strong>Heterogeneity:</strong> Architectural and performance non-uniformity increase</td>
</tr>
<tr>
<td><strong>Reliability:</strong> It’s the hardware’s problem</td>
<td><strong>Reliability:</strong> Cannot count on hardware protection alone</td>
</tr>
</tbody>
</table>

Fundamentally breaks our current programming paradigm and computing ecosystem

These Trends Affect ALL users at ALL SCALES

The disruptions are primarily within the node
- Only resilience and interconnect scaling are exclusively HPC
- Exponential growth of parallelism, power, and memory trends have pervasive impact

Worse yet, the changes are already underway!
- There is no need point in waiting for the “ExaFLOP computer”.
- These trends are happening NOW!

Emerging solutions are;
- Difficult to program
- Offer non-portable programming environments
- Non-durable (have to rewrite again when the next machine comes along)

We need to design the software environment together with the hardware
There are many possible realizations of the machine architecture: Need agile simulation/modeling capability.

- Cores (many simple cores)
  - Flat clock rate
  - Multithreaded (n-threads)
  - SIMD (n-slots)
  - Fat+Thin cores (ratio)
- NoC
  - Constrained Topology (2D)
- Cache Hierarchy (size, type, assoc)
  - Automatic caches
  - Scratchpad/software managed
  - NVRAM
- Alternative coherency methods

- Non-uniform memory access (NUMA) between cores and memory channels
  - Topology may be important
  - Or perhaps just distance
- Memory
  - Increased NUMA domains
  - Intelligence in memory (or not)
- Fault Model for node
  - FIT rates, kinds of faults, granularity of faults/recovery
- Interconnect
  - Constrained Topology (Torus, Tapered Dragonfly)
  - Bandwidth/latency/overhead for communication
  - Primitives for data movement/sync
  - Global Address Space or messages only
  - Memory fences
  - Transactions / remote atomics

Design Space Exploration for Exascale Computing

The Trade Space for Exascale is Very Complex

- 20 MW power envelope
- $200M cost envelope
- Exascale Performance envelope
- bytes/core envelope

nodes vs. memory
Solution: Hardware/Software Codesign

Goal: Rapidly evaluate hardware/software/algorithmic formulation together (it's not just design space exploration... it's considering HW changes together with SW)

Architectural Simulation for Design Space Exploration

- **Enabling a New Model for Vendor Interaction**
- **Insert Applications into the Design Process**
  - Simulate hardware before it is built!
  - Break slow feedback loop for system designs
  - Insert applications and algorithms into the tightly coupled hardware/software CoDesign process
Define and Develop the Co-design Methodology for HPC

- Key Co-design Capabilities
  - HPC Architectural Simulators
  - Proxy Applications
  - Advanced Architecture Testbeds
  - Proxy Architectures

- Proxy Applications
  - See http://www.mantevo.org

- HPC Simulation
  - http://code.google.com/p/sst-simulator

Proxy Applications

Full Workload

Integration (reality) Increases

Understanding Increases

Coupled Multiphysics Application

Hydro, radiation transport, etc…

All Application Kernels, but stripped down to essentials

Just the communication (halo exchange) or per-core compute load

CG, Elliptic Solve, Stencil, PIC particle push vs. particle

Wasserman 2006
Codesign workflow for Proxy Applications

Proxy Hardware (via simulation models)
cycle accurate and energy-accurate models
Rapid Design Synthesis
(Rapid HW prototyping for Design Evaluations)

Processor configuration
1. Select from menu
2. Automatic instruction discovery (XPRES Compiler)
3. Explicit instruction description (TIE)

This stuff is essential!

Tailored SW Tools: Compiler, debugger, simulators, Linux, other OS Ports (Automatically generated together with the Core)

Build with any process in any fab

Application-optimized processor implementation (RTL/Verilog)
Base CPU, OCD, Apps, Datapaths, Cache, Timer, Extended Registers, FPU

Processor Generator (Tensilica)

Application-optimized processor implementation (RTL/Verilog)
Base CPU, OCD
Apps, Datapaths, Cache, Timer
Extended Registers, FPU

RAMP FPGA-accelerated Emulation of ASIC

Build with any process in any fab
Or "tape out" To FPGA

Cycle Time 1-2 days

Synthesize SoC (hours)

Cycle Time 4-6+ years

Build Hardware (2 years)

Tune Software (2 years)

Build application

Cycle Time 2 year concept phase

Design New System

Port Application

Autotune Software (hours)

This is central to CoDesign (and it ain’t new)
A tour of the Processor Generator
Rapid Prototyping of Processor Designs

Checkboxes to add or remove features from processor design (everything from ISA, to endian-ness to cache hierarchy. TIE language (verilog like) to design ISA extensions. Compiler back-end automatically modified by the environment to understand extensions.

And cycle-accurate simulation (software based + can tape out to FPGA hardware) for detailed understanding of performance implications of design. Full introspection of the hardware to understand performance.

CoDesign to Reduce Waste

Biggest win was in what we do NOT include in an HPC Design

Mark Horowitz 2007: “Years of research in low-power embedded computing have shown only one design technique to reduce power: reduce waste.”

Seymour Cray 1977: “Don’t put anything in to a supercomputer that isn’t necessary.”
Peel Back the Historical Growth of Instruction Sets (accretion of cruft)

Chris Rowen: Tensilica

Traditional Processor Family

Configurable Processor Family

A Short List of x86 Opcodes that HPC Applications Don’t Need!

<table>
<thead>
<tr>
<th>opcode</th>
<th>size</th>
<th>num</th>
<th>mnemonic</th>
<th>opcode</th>
<th>size</th>
<th>num</th>
<th>mnemonic</th>
<th>opcode</th>
<th>size</th>
<th>num</th>
<th>mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>AXX</td>
<td>32</td>
<td>0</td>
<td>push</td>
<td>BXX</td>
<td>32</td>
<td>0</td>
<td>pop</td>
<td>CXX</td>
<td>32</td>
<td>0</td>
<td>not</td>
</tr>
<tr>
<td>AX</td>
<td>16</td>
<td>0</td>
<td>...</td>
<td>BX</td>
<td>16</td>
<td>0</td>
<td>...</td>
<td>CX</td>
<td>16</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AXx</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BXx</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CXx</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AXp</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BXp</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CXp</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AXr</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BXr</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CXr</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AXs</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BXs</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CXs</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AXl</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BXl</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CXl</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AXo</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BXo</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CXo</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AXn</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BXn</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CXn</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AXw</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BXw</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CXw</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AXi</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BXi</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CXi</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AXe</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BXe</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CXe</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AXh</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BXh</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CXh</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AXt</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BXt</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CXt</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AX1</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BX1</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CX1</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AX2</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BX2</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CX2</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AX3</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BX3</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CX3</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AX4</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BX4</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CX4</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AX5</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BX5</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CX5</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AX6</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BX6</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CX6</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AX7</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BX7</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CX7</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AX8</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BX8</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CX8</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AX9</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BX9</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CX9</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AXA</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BXA</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CXA</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AXB</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BXB</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CXB</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AXC</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BXC</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CXC</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AXD</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BXD</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CXD</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AXE</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BXE</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CXE</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>AXF</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>BXF</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>CXF</td>
<td>32</td>
<td>0</td>
<td>...</td>
</tr>
</tbody>
</table>

Area = silicon cost and power

Traditional Processor Family

Time per variant: years

Configurable Processor Family

Time per variant: days

A Short List of x86 Opcodes that HPC Applications Don’t Need!

<table>
<thead>
<tr>
<th>mnemonic</th>
<th>opcode</th>
<th>size</th>
<th>num</th>
<th>op1</th>
<th>op2</th>
<th>op3</th>
<th>ret</th>
<th>Tentalize notes</th>
<th>need very much</th>
<th>verbose</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>32</td>
<td>0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>16</td>
<td>0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>8</td>
<td>0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>4</td>
<td>0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>2</td>
<td>0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>48</td>
<td>0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>64</td>
<td>0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>128</td>
<td>0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>256</td>
<td>0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>512</td>
<td>0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>1024</td>
<td>0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

A Short List of x86 Opcodes that HPC Applications Don’t Need!
More Unused Opcodes

- We only need 80 out of the nearly 300 ASM instructions from the x86 instruction set!
- Still have all of the 8087 and 8088 instructions!
- Wide SIMD Doesn't Make Sense with Small Cores
- Neither does Cache Coherence
- Neither does HW Divide or Sqrt for loops
  - Creates pipeline bubbles
  - Better to unroll it across the loops (like IBM MASS libraries)
- Move TLB to memory interface because its still too huge (but still get precise exceptions from segmented protection on each core)

Revisiting Data Locality Management

Vertical Locality Management
(spatio-temporal optimization)

Horizontal Locality Management
(topology optimization)
Hardware/Software for Managing Vertical Data Locality

Memory Bandwidth

- Memory that exceeds 20MW is not practical.
- Memory Technology Investment enables improvement in bandwidth (and hence improves application breadth).

Application performance and breadth pushes us to higher bandwidth.
Power pushes us to lower bandwidth.

Memory Power Consumption in Megawatts (MW)
Memory Bandwidth

- Stacked JEDEC 30pJ/bit 2018 ($20M)
- Advanced 7pJ/bit Memory ($100M)
- Enhanced 4pJ/bit Advanced Memory ($150M cumulative)
- Feasible Power Envelope (20MW)
Loop Fusion To Reduce Memory Bandwidth
“use cache as bandwidth filter”

Baseline
2.9 GB/sweep
1.78 Bytes/Flop

Simple Fusion
1.6 GB/sweep (−46%)
0.96 Bytes/Flop

Aggressive Fusion
0.48 GB/sweep (−84%)
0.29 Bytes/Flop

Note: This is not traditional fusion. Current compilers models are not up to this task.

But how much is it worth to fix them?

Codesign Question:
How Much Cache Should I have?

More Cache reduces memory bandwidth requirements
But consumes surface area, so need to give up some processor cores or other services

Memory Traffic vs Cache Size for Loop Fusion Scenarios
("best" block size)

Select “best” strategy for each cache size
Codesign Question:
How many registers should I have?

If not enough registers available to hold state, registers spill into the L1 cache, increasing cache traffic and possibly affecting performance.

x86 has 16 integer and 16 FP registers!

Allocate to registers

Leave in L1 cache

x86 has 16 FP named registers!
Conclusions on Vertical Locality Management

Aggressive Fusion is essential to lower memory bandwidth requirements
• But to get the advantage need large L1 cache (would need to be scratchpad to be feasible)
• Also requires larger register file
• And requires new programming paradigm to enable aggressive fusion (functional semantics or other hints to facilitate compiler analysis)

Benchmarking on current architectures would have missed this opportunity
• Requires predictive modeling and architectural simulation
• This is the center of codesign

Many of the most valuable hardware opportunities identified by codesign will have major impact on our programming paradigm!
• Its not just about transforming code and algorithms
• Choices affect our entire paradigm for programming these systems!
• Must think deeper about ramifications to programming ecosystem (just as we did in the transition from vec to MPI)

Software/Hardware Mechanisms for Managing Horizontal Data Locality

FLOPs cost more than on-chip data movement! (NUMA)
Problems with Existing Abstractions for Expressing Locality

Our current programming models assume all communicating elements are equidistant (PRAM)
- OpenMP, and MPI each assume flat machine at their level of parallelism

But the machine is not flat!!!
- Lose performance because expectation and reality are mismatched
- Pmodel does not match underlying machine model!!

What is wrong with Flat MPI?
- 10x higher bandwidth between cores on chip
- 10x lower latency between cores on chip
- If you pretend that every core is a peer (each is just a generic MPI rank) you are leaving a lot of performance on the table
- You cannot domain-decompose things forever

Good News!
Benefits of expressing Two-levels of locality

But OMP offers no management of data locality
- Huge performance penalty for ignoring NUMA effects
- Then programmer responsible for matching up computation with data layout!! (UGH!)
- Makes library writing difficult and Makes AMR nearly impossible!

Current Practices (MPI+OpenMP)

MPI+OMP Hybrid recognizes huge cost for going off-chip
Hybrid Model improves 3D FFT communication performance
- Enables node to send larger messages between nodes
- Substantial improvements in communications efficiency

Bad News!
It’s the Revenge of the SGI Origin2000

Good News!
Benefits of expressing Two-levels of locality
**Expressing Hierarchical Layout**

Hierarchical layout statements
- Express mapping of "natural" enumeration of an array to the unnatural system memory hierarchy
- Maintain unified "global" index space for arrays (A[x][y][z])
- Support mapping to complex address spaces
- Convenient for programmers

Iteration expressions more powerful when they bind to *data locality* instead of *threadnumber*
- instead of upc_forall(;;;<threadnumber>)
- Use upc_forall(;;;<implicitly where Array A is local>)

```
upc_forall(i=0;i<NX;i++;A)
  C[j]+=A[j]*B[i][j]);
```

---

**Example from UPC**

1D Decomp
- Shared [blocksize] int [nx][ny][nz]

3D Decomp
- Struct gridcell_s { int cell[cellsize] }
- Shared [blocksize] gridcell_t cellgrids[nthreads];
- #define grids(gridno,z,y,z) cell_grids[gridno][((z)/DIMZ)*NO_ROWS*NO_COLS+ etc.....
Hierarchical Layout Statements

Building up a hierarchical layout

- Layout block coreblk {blockx,blocky};
- Layout block nodeblk {nnx,nny,nnz};
- Layout hierarchy myheirarchy {coreblk,nodeblk};
- Shared myhierarchy double a[nx][ny][nz];

- Then use data-localized parallel loop

\[
\begin{align*}
doall_at(i=0;i<nx;i++;a) & \\
doall_at(j=0;j<ny;j++;a) & \\
doall_at(k=0;k<nz;k++;a) & \\
a[i][j][k] = C*a[i+1] & \ldots
\end{align*}
\]

- And if layout changes, this loop remains the same

Satisfies the request of the application developers (minimize the amount of code that changes)

Conclusions on Data Layout

Failure to express data locality has substantial cost in application performance
- Compiler and runtime cannot figure this out on its own given limited information current languages and programming models provide

Hierarchical data layout statements offer better expressiveness
- Must be hierarchical
- Must be multidimensional
- Support composable build-up of layout description

Data-centric parallel expressions offer better virtualization of # processors/threads
- Don’t execute based on “thread number”
- Parallelize & execute based on data locality
- Enables layout to be specified in machine-dependent manner without changing execution
Interconnects

Technology Trends and Effects on Application Performance

Interconnect Design Considerations for Message Passing Applications

- Application studies provide insight to requirements for Interconnects (both on-chip and off-chip)
  - On-chip interconnect is 2D planar (crossbar won’t scale!)
  - Sparse connectivity for most apps.; crossbar is overkill
  - No single best topology
  - Most point-to-point message exhibit sparse topology + often bandwidth bound
  - Collectives tiny and primarily latency bound
- Ultimately, need to be aware of the on-chip interconnect topology in addition to the off-chip topology
  - Adaptive topology interconnects (HFAST)
  - Intelligent task migration?

Opportunity
CCSM Performance Variability
(trials of embedding communication topologies)

Result of 311 runs of the coupled climate model showing model throughput as a function of completion date.

Data from Harvey Wasserman

Node placement of a fast, average and slow run

Failure to exploit opportunity (when virtualization of topology goes wrong)

Fast run: 940 seconds Average run: 1100 seconds Slow run: 2462 seconds
Topology Optimization
(turning Fat-trees into Fit-trees)

A Fit-tree uses OCS to prune unused (or infrequently used) connections in a Fat-Tree
Tailor the interconnect bandwidth taper to match application data flows

A 2-ary 4-tree with 16 nodes.

Figure 2: A (2, 2, 4)-TL fit-tree with 16 nodes.

Architectural Simulation of 100k-endpoint Interconnect
Communication Times for Various Network Link BWs

Varied network link bandwidths: 10, 40, 100 GB/s
Network link bandwidth is the bottleneck in communication
Increasing link bandwidth improves the performance even when the jobs are canonically placed (see Slide 12 for NIC bw = 400 GB/s)
Conclusions on Interconnect

Huge opportunity for communication topology optimization to improve performance
- Runtime information gathering for active task migration, circuit switching
- Use intelligent runtime to remap for locality or to use circuit switching to optimize switch topology

Current Programming Models do not provide facility to express topology
- OpenMP topology un-aware
- MPI has topology directives (tedious, rarely implemented or used)
- Results in substantial measurable losses in performance (within node/OpenMP and inter-node/MPI)

Need to provide the compiler, runtime & resource manager more information about topology

HW/SW Co-Tuning for Energy Efficiency

The approach: Use auto-tuned code when evaluating architecture design points

Co-Tuning can improve power-efficiency and area-efficiency by ~4x
Full Design Studies

Fully Integrated Design Studies

Green Flash Overview

Research effort: study feasibility of designing an application-targeted supercomputer and share insight w/community

- Elements of the approach
- Choose the science target first (climate)
- Design systems for applications (rather than the reverse)
- Design hardware, software, scientific algorithms together using hardware emulation and auto-tuning

- What is NEW about this approach
  - Leverage commodity processes used to design power efficient embedded devices (redirect the tools to benefit scientific computing!)
  - Auto-tuning to automate mapping of algorithm to complex hardware
  - RAMP: Fast hardware-accelerated emulation of new chip designs
Global Cloud System Resolving Climate Models

Surface Altitude (feet)

- 200km: Typical resolution of IPCC AR4 models
- 25km: Upper limit of climate models with cloud parameterizations
- 1km: Cloud system resolving models

- Direct simulation of cloud systems replacing statistical parameterization.
- This approach recently was called for by the 1st WMO Modeling Summit

1km-Scale Global Climate Model Requirements

- Simulate climate 1000x faster than real time
- 10 Petaflops sustained per simulation (~200 Pflops peak)
- 10-100 simulations (~20 Exaflops peak)
- E3SGS report suggests exaflop requires 180MW

Some specs:
- Advanced dynamics algorithms: icosahedral, cubed sphere, reduced mesh, etc.
- ~20 billion cells → Massive parallelism
- 100 Terabytes of Memory
- Can be decomposed into ~20 million total subdomains

Requires New Algorithmic Approach to Achieve 20M-way concurrency
Collaborating with CSU on Icosahedral Model
Example Design Study (2006 to 2009)

**Green Flash: Global Cloud-Resolving Climate Models**


200km
Typical resolution of IPCC AR4 models

25km
Upper limit of climate models with cloud parameterizations

1km
Cloud system resolving models with cloud parameterizations

- Direct simulation of cloud systems replacing statistical parameterization.
- This approach recently was called for by the 1st WMO Modeling Summit.

**Demonstrated during SC ’08**

**Proof of concept**

- CSU limited-area atmospheric model ported to Tensilica architecture
- Single Tensilica processor running atmospheric model at 50MHz

**Actual code running - not representative benchmark**

---

Example Design Study

**Green Wave: Seismic Imaging**

Seismic imaging used extensively by oil and gas industry
- Dominant method is RTM (Reverse Time Migration)

RTM models acoustic wave propagation through rock strata using explicit PDE solve for elastic equation in 3D
- High order (8th or more) stencils
- High computational intensity

- Typical survey requires months of computing on petascale-sized resources
Example Design Study
Green Wave: Seismic Imaging

Developed RTL design for SoC in 45 nm technology using off-the-shelf embedded technology + simulated with RAMP FPGA platform

- **Tensilica LX2 processor core**
  - off-the-shelf 4-slot SP SIMD, ECC
  - 4-slot VLIW (FLIX)
  - cache hierarchy with Local store + conventional cache.
  - TIE queues interprocessor messaging

- **NoC fabric for SoC services**
  - 128 cores
  - 4 DDR3 1600 memory controllers
  - 4x 10Gig Ethernet

**Area Breakdown (240 mm² for SoC)**
- Core: 12%
- Local Store: 13%
- OEA controller: 3%
- NoC: 7%
- ISA Extensions: 7%
- Cache: 34%
- Risting overhead: 2%

**Power Breakdown (45W total for SoC+ memory)**
- Core + ISA Extensions: 19%
- Local Store + Cache: 9%
- DRAM + Controllers: 23%
- NoC: 49%

Compare to Nehalem, Fermi
- All 40-45nm technology
- Nehalem and Green Wave are 240mm² die area with conventional DDR3 memory (same memory perf)
- Fermi c2050 is 540mm² die with DDR5 memory (3x higher memory bandwidth)

8th order RTM kernel performance
- Nehalem auto-tuned to within 15% of theoretical performance limit by Sam Williams
- Fermi hand-tuned by Paulius Mickavelius of Nvidia

Off-the-shelf embedded ASIC unable to beat Fermi (but within 30%)

However, offers huge gain in energy efficiency
- Fermi burdened by host (green triangle is if you had Fermi without host)
- Green wave also has advantage of not including anything you don’t need for RTM
  - Fermi and Nehalem have to include a lot of extra stuff for other markets such as Graphics.
  - Nehalem also must maintain legacy binary compatibility
Basic Tenets of Codesign

Start with application
- Example, we chose Cloud Resolving Climate Model for Green Flash
- Use to drive all hardware design decisions

Use Architectural simulation to prototype
- Leverage tools from embedded market
- Low power, rapid design cycles
- Enables us to include math & CS in hardware design process

Tune algorithm, hardware and software together
- Autotuning to accelerate SW optimization
- Algorithm redesign and comparison of alternatives

Achieve 100x energy efficiency improvement over mainstream HPC approach

Conclusions

Emerging hardware constraints are increasingly mismatched with our current Software Environment (algorithms, pmodels, systems)
- Current emphasis is on preserving FLOPs
- The real costs now are not FLOPs... it is data movement
- Requires shift to a data-locality centric programming paradigm and hardware features to support it

Codesign is NOT just design optimization
- The programming environment and associated “abstract machine model” is a reflection of the underlying machine architecture
- Therefore, design decisions can have deep effect your entire programming paradigm
- Hardware/Software Codesign MUST consider ergonomic decisions about your programming environment together with performance

Performance Portability Should be Top-Tier Metric for CoDesign process
- Know what to IGNORE, what to ABSTRACT, and what to make more EXPRESSIVE
Cache and Block Size Effects on Memory Traffic

Smaller blocks fit into cache, but require more redundant traffic for ghost zones

Memory Traffic vs. Block Size for 128 x 128 x 128 CNS Code

- 16 kB Cache
- 64 kB Cache
- 256 kB Cache
- 1 MB Cache
- 4 MB Cache
- Unlimited Cache

Optimal block size depends on available cache
Loop Fusion To Reduce Memory Bandwidth
“use cache as bandwidth filter”

Baseline
2.9 GB/sweep
1.78 Bytes/Flop

Simple Fusion
1.6 GB/sweep (–46%)
0.96 Bytes/Flop

Aggressive Fusion
0.48 GB/sweep (–84%)
0.29 Bytes/Flop

Note: This is not traditional fusion. Current compilers models are not up to this task.

But how much is it worth to fix them?

Codesign Question:
How Much Cache Should I have?

More Cache reduces memory bandwidth requirements
But consumes surface area, so need to give up some processor cores or other services

Memory Traffic vs Cache Size for Loop Fusion Scenarios
("best" block size)
Codesign Question:
How many registers should I have?

If not enough registers available to hold state, registers spill into the L1 cache, increasing cache traffic and possibly affecting performance.

Modeling, Simulation, and Computer Architecture Research for Co-design of Energy Efficient Computing

Goal: Rapidly evaluate technology alternatives to enable codesign

Innovations
- Compiler-assisted automated performance model extraction
- Flexible hardware/software simulation/modeling environment

History
- Microbenchmarks (2002-2005)
- Ultra-Efficient Hardware Architecture Research: Green Flash/Wave (2006-present), RAMP, CoDEX
- New initiative: Computer Architecture Laboratory (CAL) with Sandia

Impact
- Industry partnerships: DARPA UHPC (NVIDIA Echelon), Fast Forward, and Design Forward
- Critical support for ExaCT codesign center and DEGAS
Because of cost and power issues, we cannot have both high memory bandwidth and large memory capacity. We evaluate the colored region which is feasible in 2017.

**Compute intensive architecture concentrates power and $’s on upper-left**

**Data Intensive architecture concentrates more power and $’s on lower right**

<table>
<thead>
<tr>
<th>Bandwidth\Capacity</th>
<th>16 GB</th>
<th>32 GB</th>
<th>64 GB</th>
<th>128 GB</th>
<th>256 GB</th>
<th>512 GB</th>
<th>1 TB</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 TB/s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 TB/s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 TB/s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>512 GB/s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>256 GB/s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128 GB/s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

成本（随着容量的增加而增加）和成本/位（随着带宽的增加而增加）