Rebooting Computing: Parallelism

Pete Beckman
Argonne National Laboratory & Northwestern University
Parallelism Track

- **Innovation**: What are the innovations needed for the exascale, zettascale, and beyond regimes? These could include (but are not limited to) energy efficiency, memory bandwidth, device scaling, and packaging.

- **Programming**: How can parallel programming be made simpler? Since Parallel computing is becoming ubiquitous, should parallel programming be taught at an earlier stage?

- **Other Computing**: How can other computing trends such as neuromorphic, approximate, and adiabatic computing affect the future direction of parallelism?
MPI Runs Successfully at Full Scale on the Largest Supercomputers of Today
Example: HACC Cosmology Code

- HACC cosmology code from Argonne (Salman Habib) achieved **14 PFlops/s** on Sequoia (Blue Gene/Q at LLNL)
  - Ran on full Sequoia system using MPI + OpenMP hybrid
  - Used 16 MPI ranks * 4 OpenMP threads per rank on each node, which matches the architecture: 16 cores per node with 4 hardware threads each
  - ~6.3 million way concurrency: **1,572,864 MPI ranks * 4 threads/rank**
  - SC12 Gordon Bell prize finalist

*The HACC code has been used to run one of the largest cosmological simulations ever, with 1.1 trillion particles*
Will Computing Be Rebooted?

- **Mira**: Blue Gene/Q System
  - 20 times faster than BG/P Intrepid (10 PF)
  - ~4 times more power (~4 MW)
  - ~5X more power efficient than BG/P

- Repeat twice to reach Exascale?
  - 400 times faster than BG/Q Mira (4 EF)
  - ~16 times more power (~64 MW)
  - ~25X more power efficient than BG/Q

Infinite number of transistors only helps if they take zero energy
Data from Peter Kogge, Notre Dame

We’ve Hit a Power Ceiling

Sockets and Cores Growing

The Clock Ceiling

Argonne 30 Years: May 14, 2013
Supercomputing in the Next 5-8 Years

- Evolution toward exascale (x100 performance increase)
- Leverage continued evolution of CMOS, advances in packaging (3D stacks), and Non-volatile memory (NVRAM)
- Increased specialization of HPC technology
  - Intel Phi + NIC + stacked memory, GPU + CPU + NIC, Fat ARM + lean ARM + NIC
  - Modify and reuse IP serving broader market but build unique chips and unique packages
- Exascale in 2022 seems feasible;
  - Possibly not for $200M and at 20 Mwatts

Observation: More Parallelism
More Hierarchy, More Complexity
Looking forward by looking at history for a moment
The 1990 Big Extinction: The Attack of the Killer Micros  
(Eugene Brooks, 1990)

Shift from bipolar vector machines & to clusters of MOS micros

- **Roadblock:** bipolar circuits leaked too much current – it became too hard to cool them (even with liquid nitrogen)
- MOS was leaking very little – did not require aggressive cooling
- MOS was used in fast growing markets: controllers, workstations, PCs
- MOS had a 20 year history and clear evolution path (“Moore’s Law”)
- MOS was slower
  - Cray C90 vs. CM5 in 1991: 244 MHz vs. 32 MHz

• Perfect example of “good enough” technology  
(Christensen, *The Innovator’s Dilemma*)
The CMOS Age: Killer Micros, Moore’s Law & Dennard Scaling (1990-2020 ?)
Microprocessor Transistor Counts 1971-2011 & Moore’s Law

Dennard Scaling:
- Decrease feature size by a factor of \( \lambda \) and decrease voltage by a factor of \( \lambda \); then
  - \# transistors increase by \( \lambda^2 \)
  - Clock speed increases by \( \lambda \)
- Energy consumption does not change
  (in reality, voltage decrease was slower; clock speed and energy consumption increased faster)
The Future Is Not What It Was

"Herbert Stein's Law: "If something cannot go on forever, it will stop,"
Have We Been There?

- **History repeats itself:**
  - CMOS technology has hit a power wall, same as ECL in late 80’es
    - Clock speed is not raising
  - Alternative materials are not ready (gallium arsenide and other III-V materials; nanowires, nanotubes)

- **History does not repeat itself:**
  - ✔ There is a much larger industrial base investing in continued improvements in current technologies
  - ✗ An alternative “good enough” technology (such as MOS in 1990)
  - ✗ There is much more code that needs to be rewritten if a new model is needed (>200MLOCs)
Will there be another Mass Extinction?

What can we say for certain about the future?

Bet on Parallelism... But What **Kind** of Parallelism?
BG/Q Water Temps

Fault-Tolerance is Already Here

- We have already seen the future
  - 8 years ago in fact.
- Persistent ECC memory faults are the norm, not the exception
  - Machines need to stay up to satisfy their contracts.
  - Over the course of a day or two these parts can be replaced, but not over the life of your batch job.

Patch Hyperbolic Integration Time

Cray XT4

Time (seconds)

Processor
Our Systems Are Adaptive...
But we don’t usually program that way:
We must re-imagine programming...

On Scalable Systems
Equal Work is not Equal Time

- Dynamic parallelism and decomposition
  - We cannot hand-pick granularity / resource mapping
  - Machine Learning?

The future is even more dynamic

- Seek new latency tolerant algorithms and methods.
- Create new tools that measure and predict latency tolerance and execution distribution

Pete Beckman  Argonne National Laboratory / Northwestern University
This is not new...
Dynamic Lightweight Parallelism

But what will pervasive look like?

PLASMA: Parallel Linear Algebra s/w for Multicore Architectures

- High utilization of each core
- Scaling to large number of cores
- Shared or distributed memory

Methodology
- Dynamic DAG scheduling
- Explicit parallelism
- Implicit communication
- Fine granularity / block data layout

Arbitrary DAG with dynamic scheduling

Croniesky
4 x 4

Fork-join parallelism

Charm++
(the run-time and execution model)

Parallelization Using Charm++
The computation is decomposed into “natural” objects of the application, which are assigned to processors by Charm ++ RTX

- Charm++/AMPI style “virtual processors”
  - Decompose into natural objects of the application
  - Let the runtime map them to processors
  - Decouple decomposition from load balancing

Benefits of Temperature Aware LB

Courtesy: Laxmikant Kale
Distance ≠ Equal Time

Human Learning...
Machine Learning...

Chicago commute one of nation's most unpredictable, study suggests

February 05, 2013 | By Jon Hilkevitch, Chicago Tribune

You can predict with a high degree of confidence that the time it takes to drive from Point A to B on any given day is unpredictable.

And it’s not just snowy or rainy days. It can be any day.

If there is a bright side, it’s that Chicago was not the worst.

- Over 420 Million Travel Times Collected Since October 2004 - All Presented In Real Time

http://www.travelmidweststats.com
Serious Cognitive Dissonance

Even Today, we have the “Dynamic Deniers”
(We want low runtime variance)

- Trinity/NERSC-8:
  “The system shall provide correct and consistent runtimes. An
application’s runtime (i.e. wall clock time) shall not change by
more than 3% from run-to-run in dedicated mode and 5% in
production mode.”
<table>
<thead>
<tr>
<th>Name</th>
<th>Best</th>
<th>Average</th>
<th>Worst</th>
<th>Memory</th>
<th>Stable</th>
<th>Method</th>
<th>Other notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timsort</td>
<td>$n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n$</td>
<td>Yes</td>
<td>Insertion &amp; Merging</td>
<td>$n$ comparisons when the data is already sorted.</td>
</tr>
<tr>
<td>Bubble sort</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>1</td>
<td>Yes</td>
<td>Exchanging</td>
<td>Tiny code</td>
</tr>
<tr>
<td>Cocktail sort</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>1</td>
<td>Yes</td>
<td>Exchanging</td>
<td></td>
</tr>
<tr>
<td>Comb sort</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>1</td>
<td>No</td>
<td>Exchanging</td>
<td>Small code size</td>
</tr>
<tr>
<td>Gnome sort</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>1</td>
<td>Yes</td>
<td>Exchanging</td>
<td>Tiny code size</td>
</tr>
<tr>
<td>Selection sort</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>1</td>
<td>No</td>
<td>Selection</td>
<td></td>
</tr>
<tr>
<td>Insertion sort</td>
<td>$n$</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>1</td>
<td>Yes</td>
<td>Insertion</td>
<td>In-place with theoretically optimal number of writes</td>
</tr>
<tr>
<td>Cycle sort</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>1</td>
<td>No</td>
<td>Insertion</td>
<td></td>
</tr>
<tr>
<td>Shell sort</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>1</td>
<td>No</td>
<td>Insertion</td>
<td></td>
</tr>
<tr>
<td>Binary tree sort</td>
<td>$n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n$</td>
<td>Yes</td>
<td>Insertion</td>
<td>When using a self-balancing binary search tree</td>
</tr>
<tr>
<td>Library sort</td>
<td>$n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n$</td>
<td>Yes</td>
<td>Insertion</td>
<td></td>
</tr>
<tr>
<td>Merge sort</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>Depends</td>
<td>Yes</td>
<td>Selection</td>
<td></td>
</tr>
<tr>
<td>In-place merge sort</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>1</td>
<td>No</td>
<td>Selection</td>
<td></td>
</tr>
<tr>
<td>Heapsort</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>1</td>
<td>No</td>
<td>Selection</td>
<td></td>
</tr>
<tr>
<td>Smoothsort</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>1</td>
<td>No</td>
<td>Selection</td>
<td></td>
</tr>
<tr>
<td>Quicksort</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>No</td>
<td>Selection</td>
<td></td>
</tr>
<tr>
<td>Patience sorting</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>No</td>
<td>Selection</td>
<td></td>
</tr>
<tr>
<td>Strand sort</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>1</td>
<td>Yes</td>
<td>Selection</td>
<td></td>
</tr>
<tr>
<td>Tournament sort</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>1</td>
<td>Selection</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fault? Dynamic Execution? Parallelism?
Abstractions Matter
Question: Abstractions for the Future Massively Parallel, Dynamic Machine?

Ignore Device Technology
Ignore Cartoons of future chips

Focus on Parallelism
Unpredictable Performance,
Interacting control systems
Unpredictable Fault
OpenMP and Performance

- The transparency of OpenMP is a mixed blessing
  - Makes things pretty easy
  - May mask performance bottlenecks
- In the ideal world, an OpenMP application just performs well
- Unfortunately, this is not the case
- Two of the more obscure effects that can negatively impact performance are cc-NUMA behavior and False Sharing
- Neither of these are restricted to OpenMP, but they are important enough to cover in some detail here
A 3D matrix update

```c
do k = 2, n
    do j = 2, n
        !$omp parallel do default(shared) private(i) &
        !$omp schedule(static)
            do i = 1, m
                x(i,j,k) = x(i,j,k-1) + x(i,j-1,k)*scale
            end do
        !$omp end parallel do
    end do
end do
```

The performance

- Inner loop over `i` has been parallelized.

Performance Analyzer data

**Using 10 threads**

<table>
<thead>
<tr>
<th>Name</th>
<th>Excl. User</th>
<th>Incl. User</th>
<th>Excl. CPU</th>
<th>User CPU</th>
<th>Wall sec.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>mt_EndOfTask_Barrier</strong></td>
<td>10.590</td>
<td>10.590</td>
<td>1.550</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>mt_WaitForWork</strong></td>
<td>3.860</td>
<td>3.860</td>
<td>0.480</td>
<td>0.480</td>
<td>0.800</td>
</tr>
<tr>
<td>RAM</td>
<td>0.400</td>
<td>0.400</td>
<td>0.040</td>
<td>0.040</td>
<td>0.010</td>
</tr>
<tr>
<td>block_3d -- MP deal from line 14 <strong>sd1A14</strong></td>
<td>0.010</td>
<td>0.010</td>
<td>0.010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>memset</td>
<td>0.030</td>
<td>0.030</td>
<td>0.030</td>
<td>0.030</td>
<td>0.010</td>
</tr>
</tbody>
</table>

**Using 20 threads**

<table>
<thead>
<tr>
<th>Name</th>
<th>Excl. User</th>
<th>Incl. User</th>
<th>Excl. CPU</th>
<th>User CPU</th>
<th>Wall sec.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>mt_EndOfTask_Barrier</strong></td>
<td>25.700</td>
<td>25.700</td>
<td>0.980</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>mt_WaitForWork</strong></td>
<td>19.880</td>
<td>19.880</td>
<td>0.470</td>
<td>0.470</td>
<td>0.800</td>
</tr>
<tr>
<td>RAM</td>
<td>0.450</td>
<td>0.450</td>
<td>0.170</td>
<td>0.170</td>
<td>0.330</td>
</tr>
<tr>
<td>block_3d -- MP deal from line 14 <strong>sd1A14</strong></td>
<td>0.010</td>
<td>0.010</td>
<td>0.010</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>mt_Setup_doJob_int</strong></td>
<td>0.010</td>
<td>0.010</td>
<td>0.010</td>
<td>0.010</td>
<td>0.010</td>
</tr>
<tr>
<td>block_3d</td>
<td>0.010</td>
<td>0.010</td>
<td>0.010</td>
<td>0.010</td>
<td>0.010</td>
</tr>
</tbody>
</table>

**Question:** Why is __mt_WaitForWork__ so high in the profile?

This is False Sharing at work!
Google (re-discovers) Noise

Component-Level Variability Amplified By Scale
A common technique for reducing latency in large-scale online services is to parallelize sub-operations across many different machines, where each sub-operation is co-located with its portion of a large dataset. Parallelization happens by fanning out a request from a root to a large number of leaf servers and merging responses via a request-distribution tree. These sub-operations must all complete within a strict deadline for the

Reducing Component Variability
Interactive response-time variability can be reduced by ensuring interactive requests are serviced in a timely manner through load balancing and other techniques.

Living with Latency Variability
The careful engineering techniques in the preceding section are essential for building high-performance interactive services, but the scale and complexity of modern Web services make it infeasible to eliminate all latency variability. Even if such perfect behavior could

Software techniques that tolerate latency variability are vital to building responsive large-scale Web services.

BY JEFFREY DEAN AND LUÍZ ANDRÉ BARROSO

The Tail at Scale

Probability of one-second service-level response time as the system scales and frequency of server-level high-latency outliers varies.

<Graph showing probability of response time exceeding 1 second as system scales.>

Fault-Tolerance is Already Here

- We have already seen the future 8 years ago in fact.
- Persistent ECC memory built are the norm, not the exception.
- Machines need to stay up to satisfy their contracts.
- Over the course of a day or two these parts will be replaced, but not the life of your batch job.

<Graph showing processor performance and fault tolerance.>
What Next?
To Reboot Computing

- We must reboot our machine abstractions
  - Dynamic control system
    - run-time view of large programs?
  - Data flow?
  - Power, Fault, Variation as first class design pieces.

- Change Programming to be parallel everywhere

- Prepare for exotic technology to force a mass extinction
What Prevents Scalability?

- Insufficient parallelism
- Insufficient latency hiding
- Insufficient resources (Memory, BW, Flops)
What Prevents Scalability?

- **Insufficient parallelism**
  - As the problem scales, more parallelism must be found

- **Insufficient latency hiding**
  - As the problem scales, more latency must be hidden

- **Insufficient resources** (Memory, BW, Flops)
  - As the problem scales, so must the resources needed