Software for Exascale

Pete Beckman
Argonne National Laboratory
Northwestern University

Exascale Applications and Software Conference
EASC 2015, Edinburgh
April 21, 2015
Argonne: Part of DOE National Laboratory System
Argonne National Laboratory

- $675M /yr budget
- 3,200 employees
- 1,450 scientists/eng
- 750 Ph.D.s
HPC has been pretty successful...

- Tianhe-2
- Sequoia
- K Computer
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*
Old Wisdom:
Moore’s Law = free exponential speedups!

But the industry’s costs keep rising, with new chip-fabrication plants costing as much as $10 billion. Cost pressures led International Business Machines Corp. last year to pay $1.5 billion to another company to take over its semiconductor operations.

Companies that can afford to keep pushing Moore’s Law are finding it increasingly hard to keep up the pace. Intel’s introduction of 14-nanometer technology was two quarters late because of the economic downturn.

While companies say they likely can keep shrinking the size of silicon chips for another decade or so, that work is bringing diminishing financial returns. Some chip designers already are limiting their use of the newest technology to high-end products where performance is more important than cost.

Intel: Moore’s Law will continue through 7nm chips
**Reality: Computing improvements have slowed dramatically over the past Decade**

Transistors you can buy for a fixed # of dollars in leading technology is no longer increasing!

"Herbert Stein's Law: "If something cannot go on forever, it will stop,""

*"No Moore?", Economist, Nov 2013.

Src: Linley Group

Pete Beckman  Argonne National Laboratory / Northwestern University

"Intel has done a little better over this period, Increasing at 21% per year.

Courtesy: Andrew Chien
Old Wisdom: Efficient Algorithms minimize operations

Classic Analysis of Algorithms: \( \text{Ops} = \text{Time} \)

Make algorithm quicker: minimize flops, compares

Ops: Best, Worst, Average, \textit{Space}
Reality:
Efficient = optimize data movement (and power)

Comparing Data Movement to Operations

We’ve Hit a Power Ceiling

The Clock Ceiling

Courtesy: John Shalf

Pipelining, load/store, GPGU...

Courtesy: Peter Kogge

Pete Beckman  Argonne National Laboratory / Northwestern University
Old Wisdom: 
Parallel Algorithms: **Equal Work = Equal Time**
(computers run at predictable speeds)

SPMD Code: Divide data into equal sized chunks across $p$ processors

For all timesteps {
   exchange data with neighbors
   compute on local data
   barrier
}

Courtesy: Andrew Chien
Reality: Performance is Highly Variable

Memory Hierarchy Depth (1-150-?)

Turbo Boost (1.2-2.5-?)

+ new Non-volatile memory (3,000 cycles)
+ old Non-volatile memory Flash (150,000 cycles)
The New Exascale Reality

- Computing rapidly gets faster and cheaper for free
  - Rapid exponential improvement is over, slow improvement will continue for awhile... Parallelism explodes, SQUEEEEZE!

- Efficient programs minimize operations
  - More operations can better, optimize for locality, data movement, power

- Computers run at fixed, predictable speed
  - Increasing dynamic and flexible, complication and advantage
What Prevents Scalability?
(in the large and in the small)

- **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
What Prevents Scalability?
(in the large and in the small)

- **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

As we scale machine, system becomes more dynamic
As we squeeze power, system becomes more dynamic
As we address resilience, system becomes more dynamic
As we share networks, system becomes more dynamic
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

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
Exploring Dynamic

Dynamic Choices: Fast and Variable..... Slow and Steady...

“Scalability challenges for massively parallel AMR applications”, IPDPS 2009

Pete Beckman Argonne National Laboratory / Northwestern University

Credit: Brian Van Straalen, John Shalf
Exploring Dynamic

Dynamic Choices: Fast and Variable..... Slow and Steady...

“Scalability challenges for massively parallel AMR applications”, IPDPS 2009

Pete Beckman Argonne National Laboratory / Northwestern University
Dynamic Power and Temp from Turboboost

pkg0=2600; pkg1=2600

pkg0=2600; pkg1=2601
We live with dynamic now... More examples
Our Hardware is Dynamic, Adaptive Today!
(the future is even more dynamic)

• Bulk Synchronous is our scaling problem
  • ≠MPI (library that moves data with put/get or send/recv)
  • We must focus on dynamic behavior
• “OS Noise” and “jitter” is a legacy distraction
  • OS & Runtime must be VERY active...
  • Forget that old-school “get out of the way” stuff
• Load balancing is necessary, but not sufficient...

• How do we design software in this new era?
• How do we build latency tolerant algs?
• Can we create tools that measure, learn, predict, and then improve performance?
How Pliable does node code need to be?
How do we measure pliability?

- What is the shape of performance distribution?
- How much latency do we need to hide?
- What is the cost of dynamic execution?
- Can we build in predictive models?
But yet, We Pretend our World is Not Dynamic

- **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.”

**ASCAC Top 10 Research Challenges for Exascale**

- “[...] power management [...] through dynamic adjustment of system balance to fit within a fixed power budget”
- “[...] Enabling [...] dynamic optimizations [...] (power, performance, and reliability) will be crucial to scientific productivity.”
- “[...] Next-generation runtime systems are under development that support different mixes of several classes of dynamic adaptive functionality.”

“dynamic” mentioned 43 times in 86 pg report
Exascale Lesson:

- **Code should be as static as possible, but no more so**
- **1) Prepare:** Create flexibility via over-decomposition, clear expression of dependencies
- **2) Take small steps to becoming more pliable.... statically**
  - (static) mapping of resource (slow/fast; heat)
  - (static) load balancing (periodic repartitioning)
  - (static) dependency graph tiling of stencils to match communication
- **3) Find goal-oriented optimization**
  - Dynamic lightweight work-sharing
  - Dynamic power management
  - Dynamic data movement across hierarchy

**Code should not consider dynamic a performance error**
(e.g. NERSC)
Online temperature predictions (blue solid line) versus actual sensor readings (red dotted line)
Distance ≠ Equal Time

Human Learning...
Machine Learning...

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

February 05, 2013 | By Jon Kull

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

And it's not just snowy or rainy, it's unpredictable any day.

If there is a bright side, it's the worst.

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

http://www.travelmidweststats.com
Automatically Tuned Linear Algebra Software (ATLAS) … 15 yrs ago…

Primitive Machine Learning: “Search and Select” (no humans)

But embarrassingly static….
“The figure shows the performance of MPI, OpenMP and MPI+OpenMP versions of CP2K. The blue diamonds show the original performance with poor task placement. The green line shows the final result with optimal placement. This obtained better performance than both the MPI and OpenMP versions and enabled more virtual threads to be used. The best placement was found to be a balanced approach where each of the 60 physical cores have as few threads as possible whilst also keeping the threads belonging to a particular MPI process physically close to one another.”
Why Mixed OpenMP/MPI Code is Sometimes Slower?

- OpenMP has less scalability due to implicit parallelism while MPI allows multi-dimensional blocking.
- All threads are idle except one while MPI communication.
  - Need overlap comp and comm for better performance.
  - Critical Section for shared variables.
- Thread creation overhead
- Cache coherence, false sharing.
- Data placement, NUMA effects.
- Natural one level parallelism problems.
- Pure OpenMP code performs worse than pure MPI within node.
- Lack of optimized OpenMP compilers/libraries.

Debug and Tune Hybrid Codes

- Debugger tools: DDT, Totalview, gdb, Valgrind.
- Profiling: IPM, CrayPat, TAU.
- Decide which loop to parallelize. Better to parallelize outer loop. Decide whether Loop permutation, fusion or exchange is needed. Use NOWAIT clause if possible.
- Choose between loop-based or SPMD.
- Use different OpenMP task scheduling options.
- Experiment with different combinations of MPI tasks and number of threads per MPI task. Less MPI tasks may not saturate inter-node bandwidth.
- Adjust environment variables.
- Aggressively investigate different thread initialization options and the possibility of overlapping communication with computation.
- Try OpenMP TASK.
- Leave some cores idle on purpose: memory capacity or bandwidth capacity.
- Try different compilers.
Argonne’s Next Big Machine: Aurora
# Argonne’s Aurora Details

<table>
<thead>
<tr>
<th>System Feature</th>
<th>Aurora</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak System performance (FLOPs)</td>
<td>180 - 450 PetaFLOPS</td>
</tr>
<tr>
<td>Processor</td>
<td>3rd Generation Intel® Xeon Phi™ processor (code name Knights Hill)</td>
</tr>
<tr>
<td>Number of Nodes</td>
<td>&gt;50,000</td>
</tr>
<tr>
<td>Compute Platform</td>
<td>Cray Shasta next generation supercomputing platform</td>
</tr>
<tr>
<td>High Bandwidth On-Package Memory, Local Memory, and Persistent Memory</td>
<td>&gt;7 PetaBytes</td>
</tr>
<tr>
<td>System Interconnect</td>
<td>2nd Generation Intel® Omni-Path Architecture with silicon photonics</td>
</tr>
<tr>
<td>Interconnect interface</td>
<td>Integrated</td>
</tr>
<tr>
<td>Burst Storage Buffer</td>
<td>Intel® SSDs, 2nd Generation Intel® Omni-Path Architecture</td>
</tr>
<tr>
<td>File System</td>
<td>Intel Lustre* File System</td>
</tr>
<tr>
<td>File System Capacity</td>
<td>&gt;150 PetaBytes</td>
</tr>
<tr>
<td>File System Throughput</td>
<td>&gt;1 TeraByte/s</td>
</tr>
<tr>
<td>Intel Architecture (x86-64) Compatibility</td>
<td>Yes</td>
</tr>
<tr>
<td>Peak Power Consumption</td>
<td>13 Megawatts</td>
</tr>
<tr>
<td>FLOPS/watt</td>
<td>&gt;13 GFLOPS/watt</td>
</tr>
<tr>
<td>Delivery Timeline</td>
<td>2018</td>
</tr>
<tr>
<td>Facility Area</td>
<td>~3,000 sq. ft.</td>
</tr>
</tbody>
</table>
Conclusions: The Times They are A-Changin’

- Embrace DYNAMIC!
  - Work ≠ Time
- Optimize algorithms for data movement
- Learn to love runtime systems
- Explore adaptive, learning, predictive software stacks that takes humans out of the loop...
  - Sorry humans, you are too slow.
  - Reject human tuning papers...
  - System software stack must stop being forgetful.....

- mpiexec -n 1048576 a.out
- mpiexec -n 1048576 a.out
Questions?