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

Tianhe-2

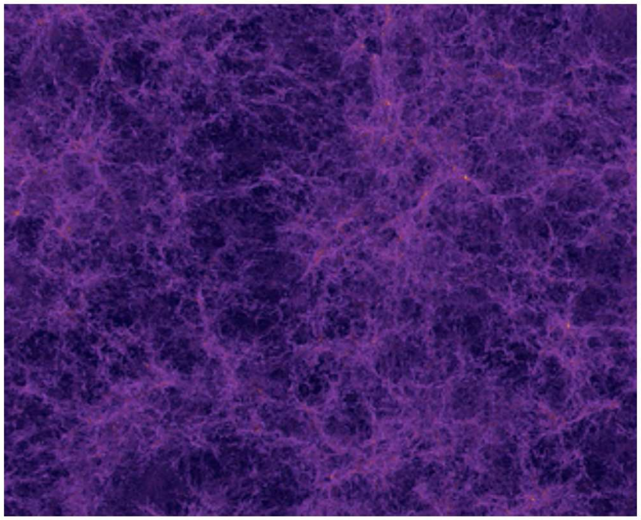
Sequoia

K Computer

©RIKEN

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**
  - http://www.hpcwire.com/hpcwire/2012-11-29/sequoia_supercomputer_runs_cosmology_code_at_14_petaflops.html
  - 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

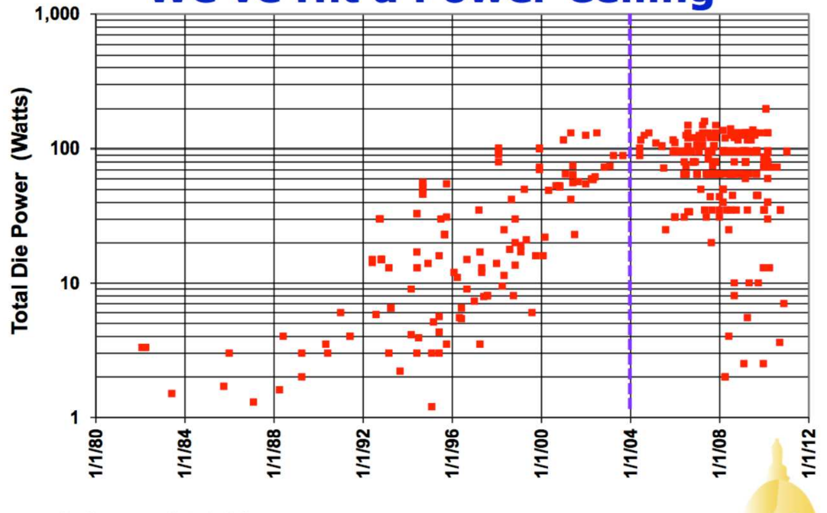
Will Computing Be Rebooted?

- *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

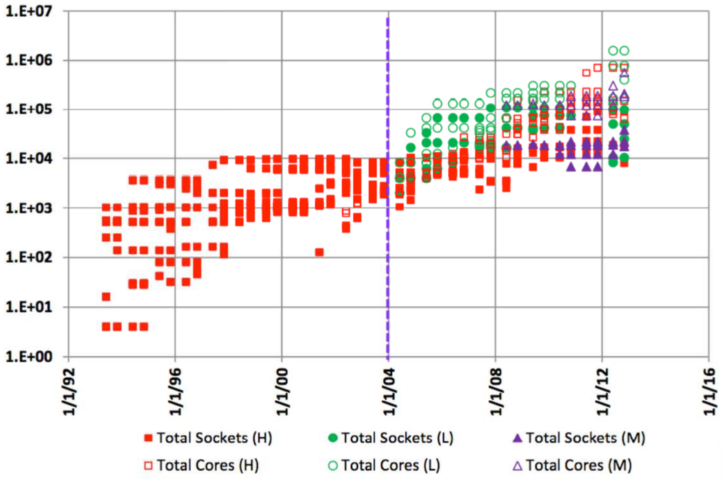
We've Hit a Power Ceiling



data from www.cpudb.stanford.edu

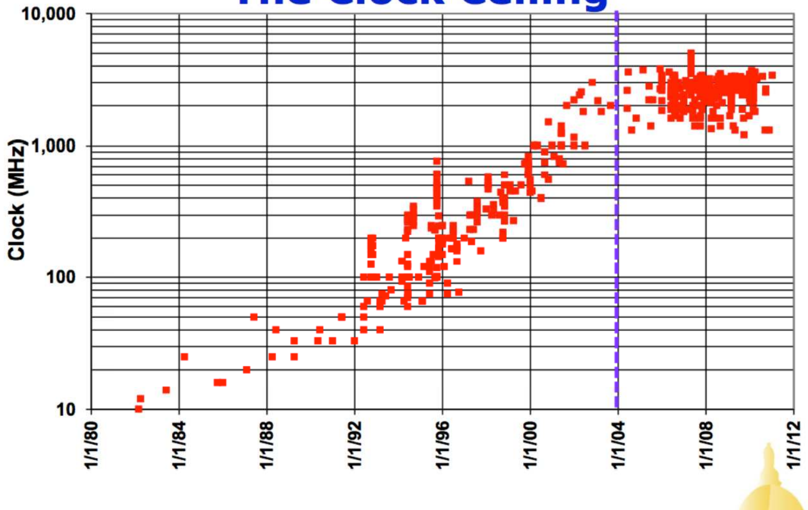
UNIVERSITY OF NOTRE DAME Argonne 30 Years:

Sockets and Cores Growing



e 30 Years: May 14, 2013 ENABLING INNOVATION 26

The Clock Ceiling



data from www.cpudb.stanford.edu

UNIVERSITY OF NOTRE DAME Argonne 30 Years: May 14, 2013 ENABLING INNOVATION 13

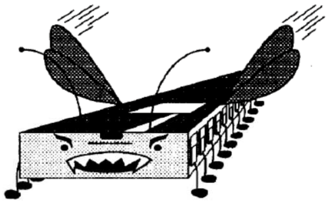
Supercomputing in the Next 5-8 Years

- 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)



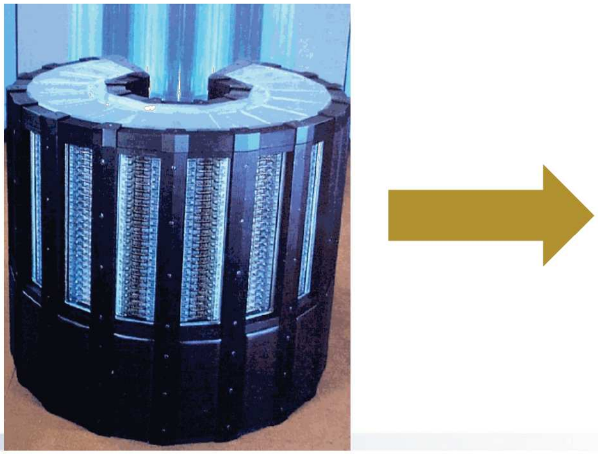
Shift from bipolar vector machines & to clusters of MOS micros

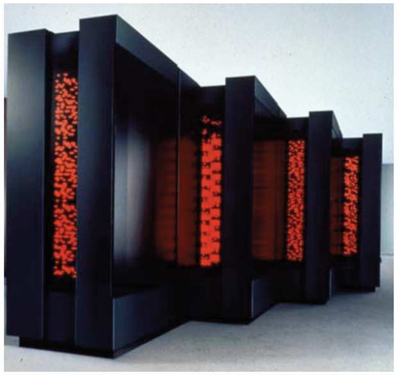
- *Roadblock*: bipolar circuits leaked too much current – it became too hard to cool them (even with liquid nitrogen)
- MOS was leaking very little – did not require aggressive cooling
- MOS was used in fast growing markets: controllers, workstations, PCs
- MOS had a 20 year history and clear evolution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

Courtesy Marc Snir

- Perfect example of "good enough" technology

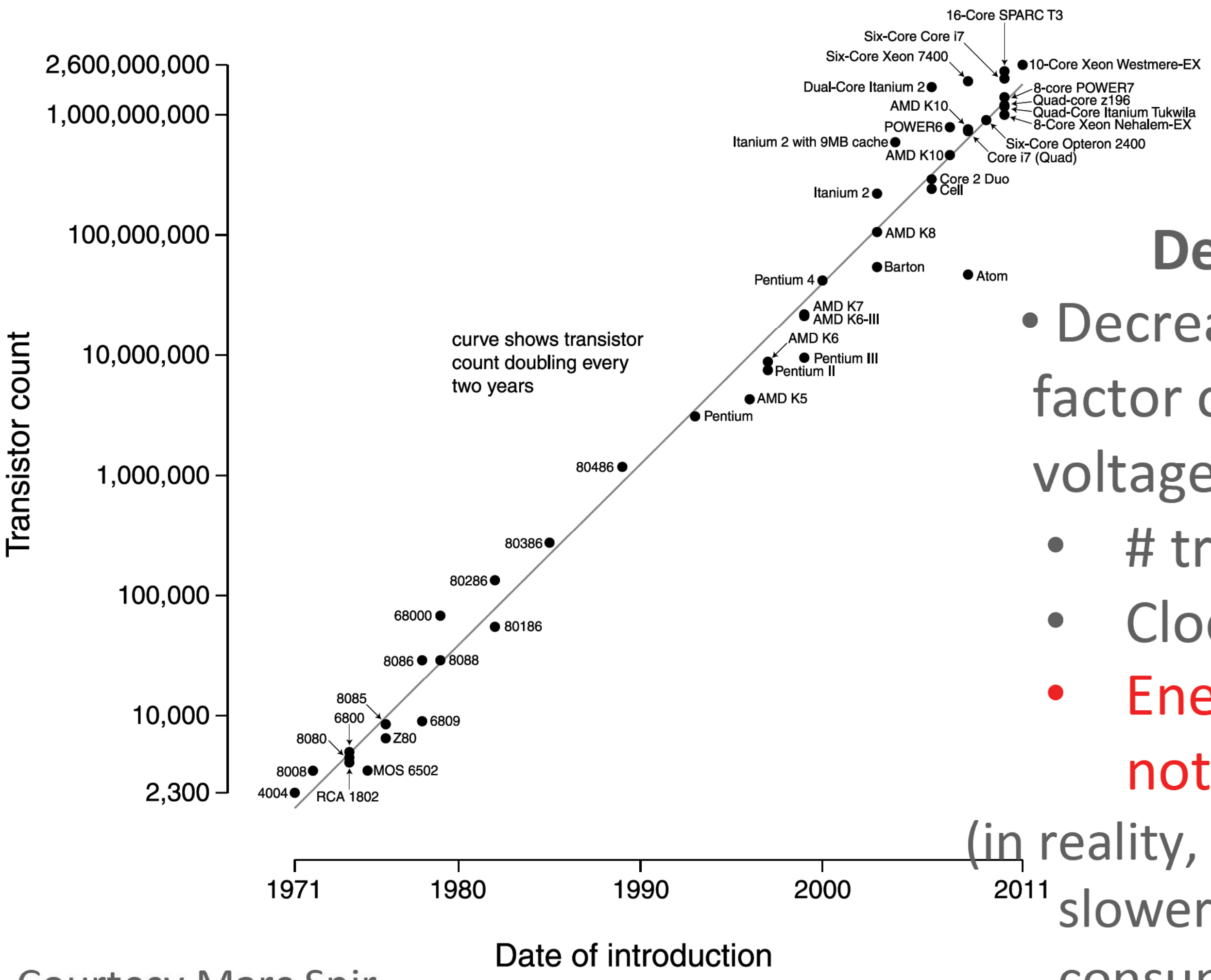
(Christensen, *The Innovator's Dilemma*)

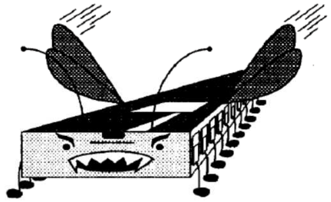




The CMOS Age: Killer Micros, Moore's Law & Dennard Scaling (1990-2020 (?))

Microprocessor Transistor Counts 1971-2011 & Moore's Law





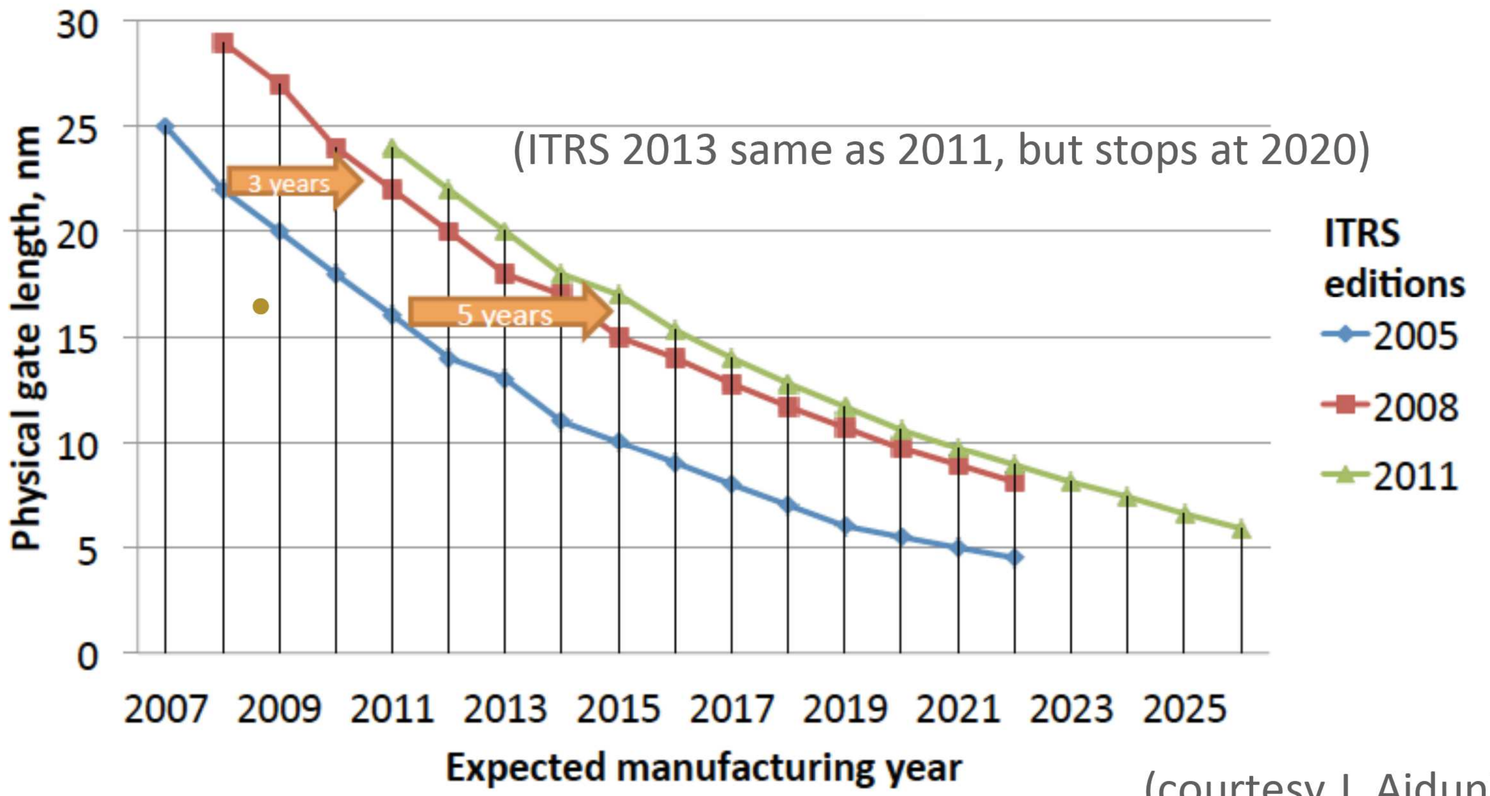
Dennard Scaling:

- Decrease feature size by a factor of λ and decrease voltage by a factor of λ; then
  - # transistors increase by λ^2
  - Clock speed increases by λ
  - Energy consumption does not change

(in reality, voltage decrease was slower; clock speed and energy consumption increased faster)

Courtesy Marc Snir

The Future Is Not What It Was



(ITRS 2013 same as 2011, but stops at 2020)

(courtesy J. Aidun)

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

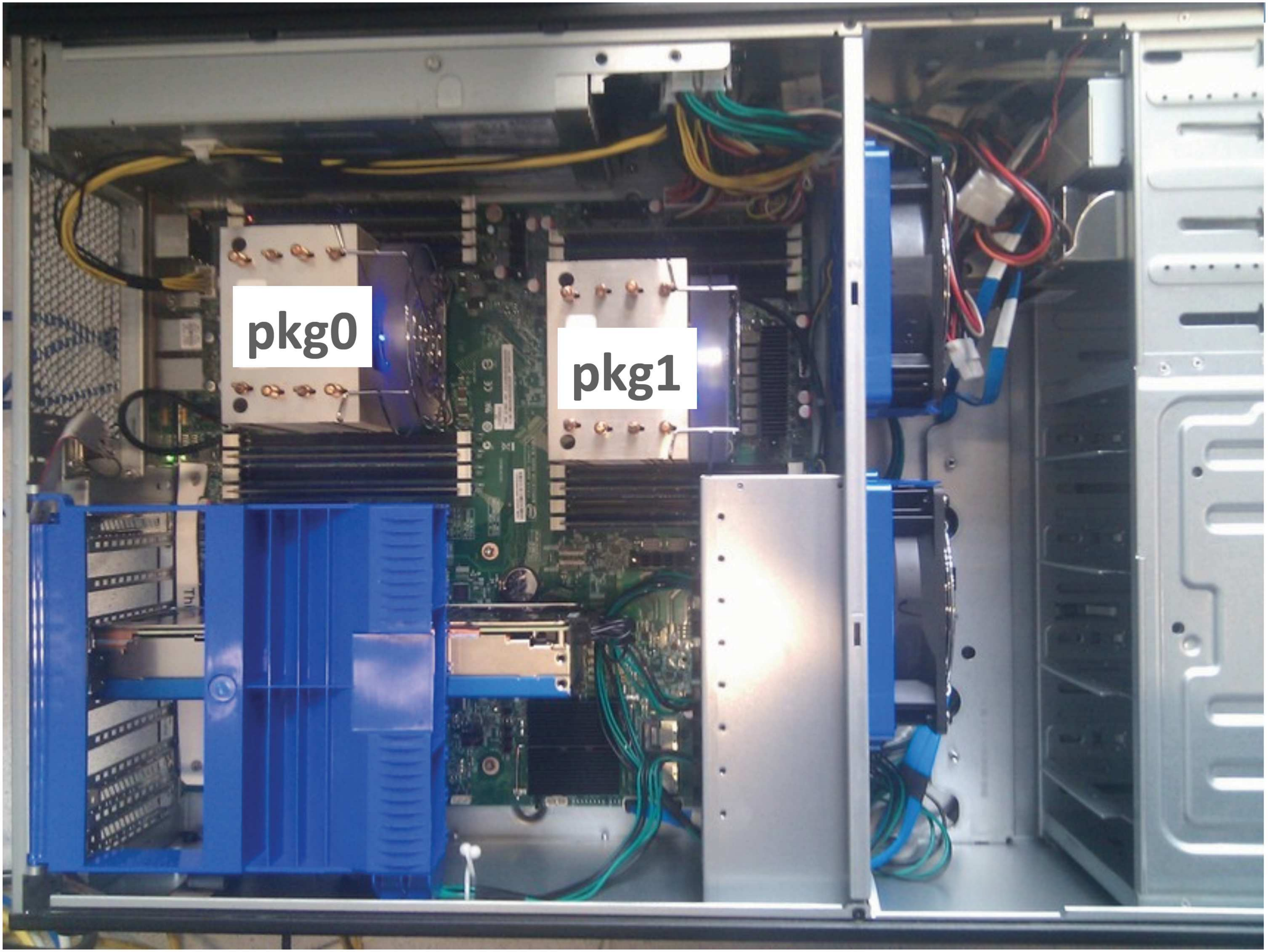
Have We Been There?

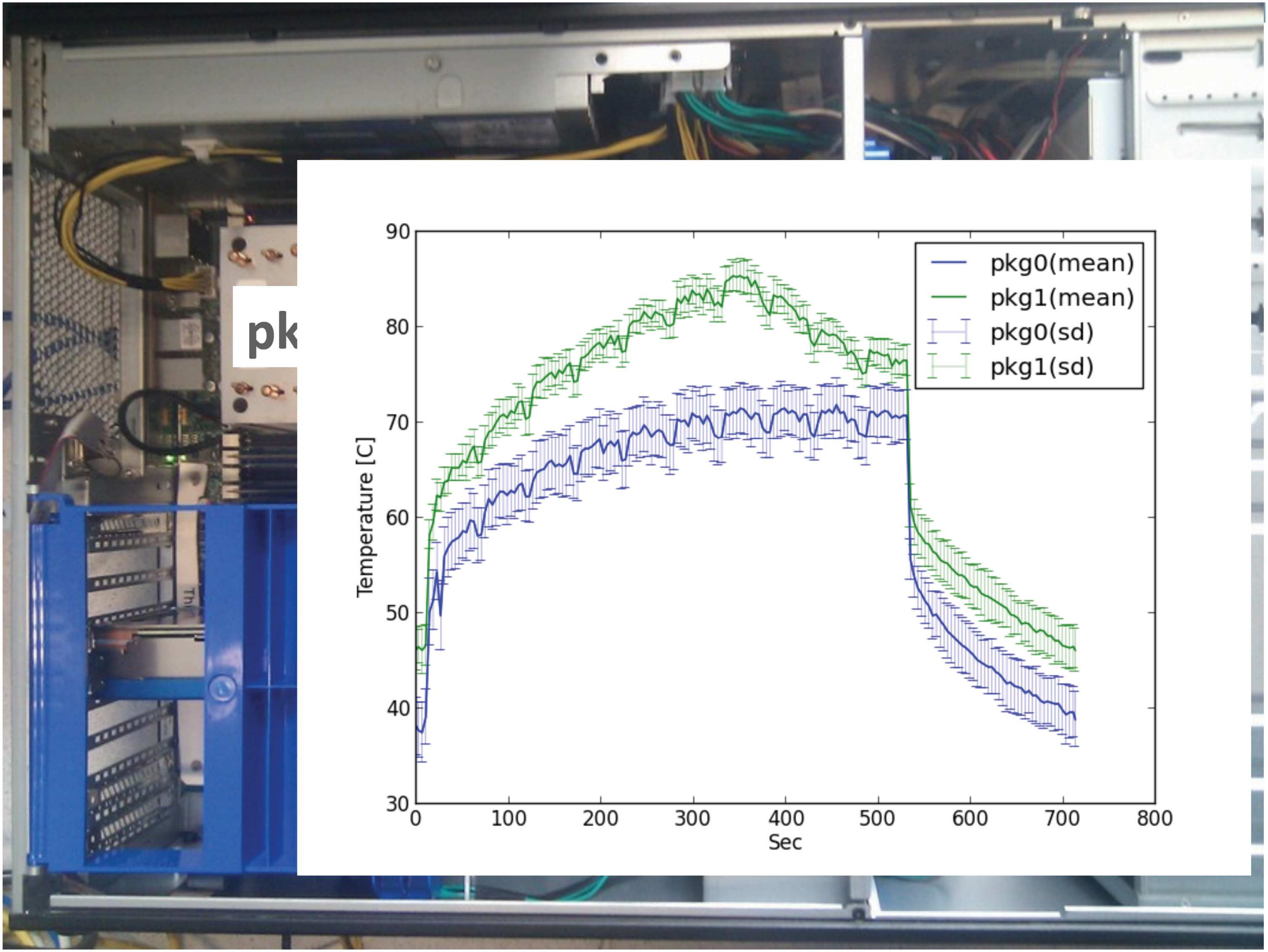
- **History repeats itself:**
  - CMOS technology has hit a power wall, same as ECL in late 80’es
    - Clock speed is not raising
  - Alternative materials are not ready (gallium arsenide and other III-V materials; nanowires, nanotubes)
- **History does not repeat itself:**
  - ✔ There is a much larger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

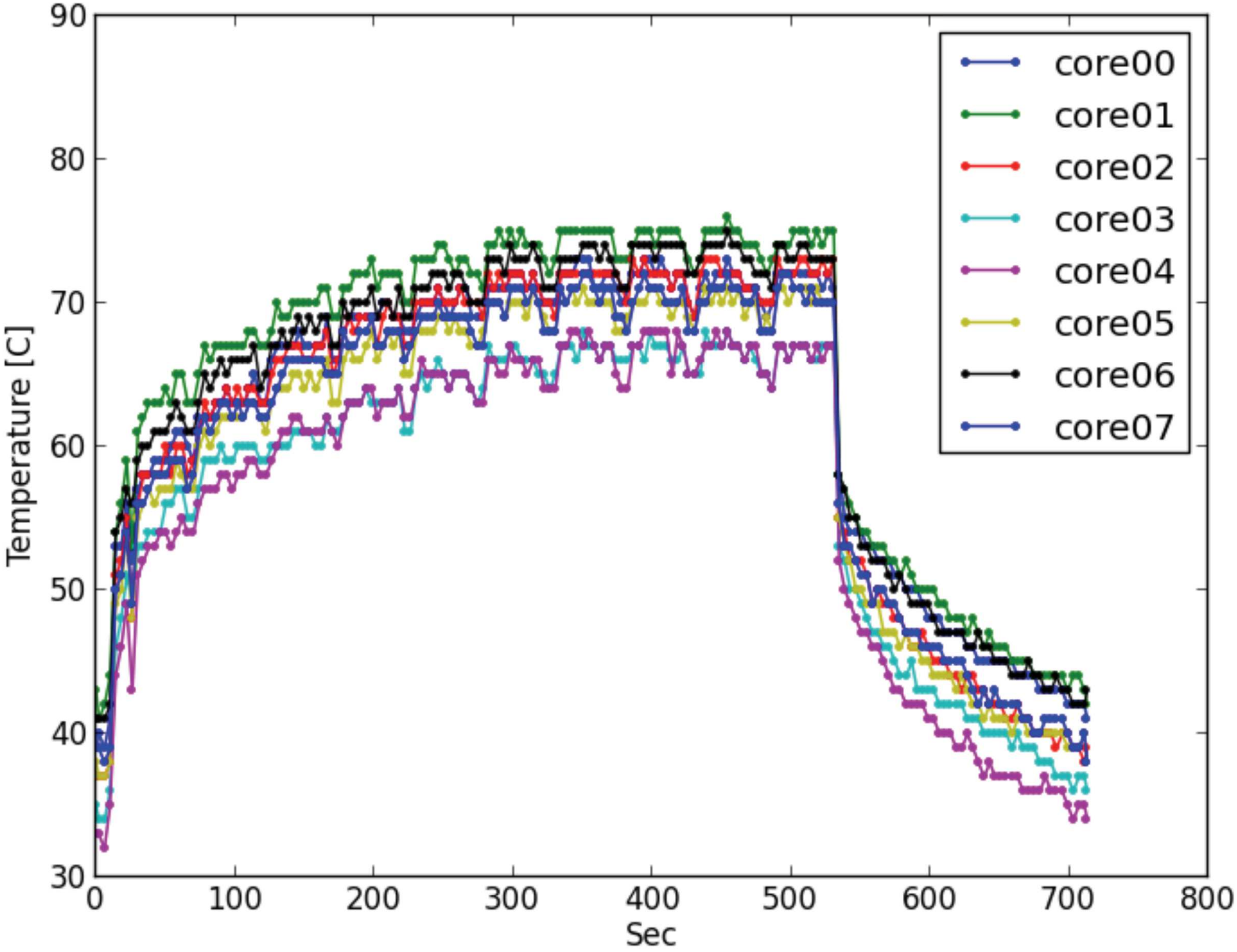
Will there be another Mass Extinction?

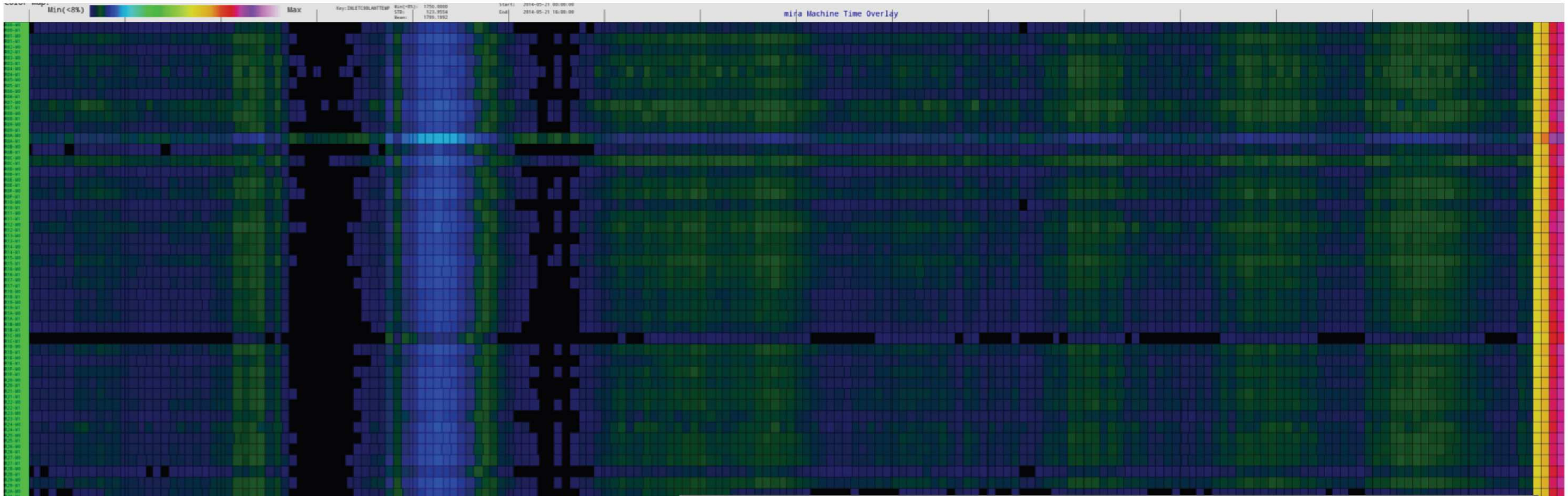
What can we say for certain about the future?

Bet on Parallelism…
But What Kind of Parallelism?

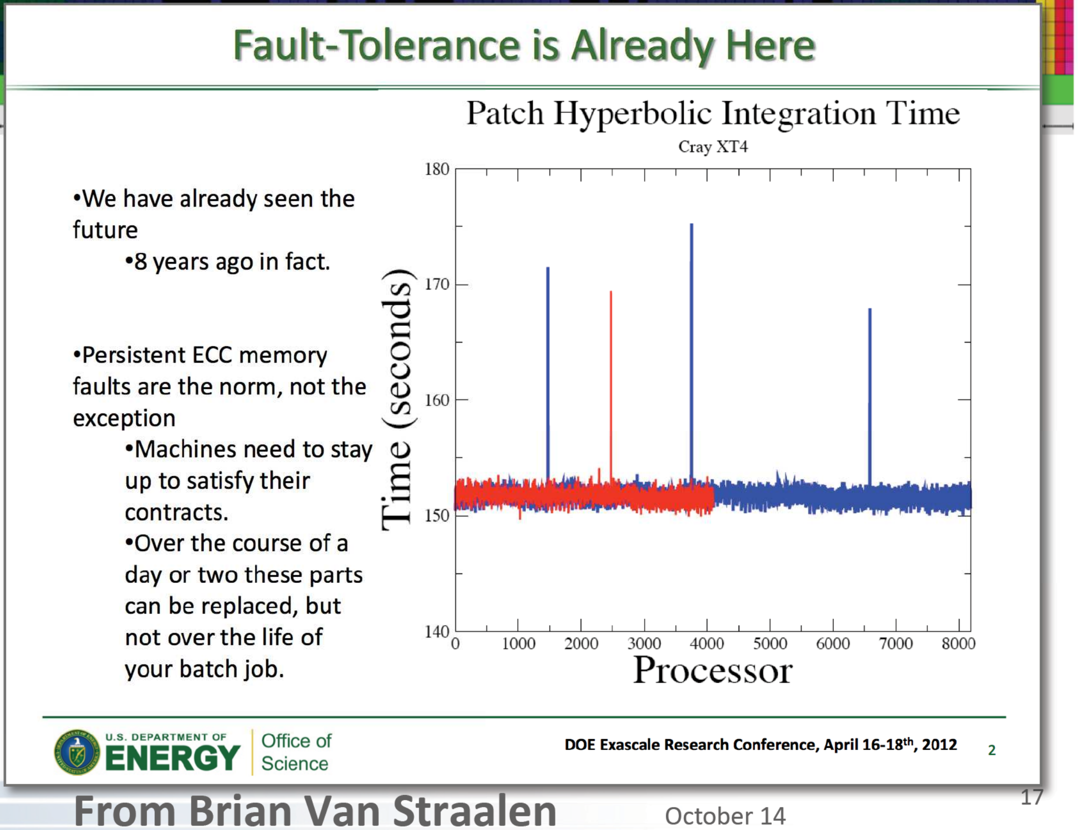

pkg0
pkg1


pk
90
80
70
60
50
40
30
Temperature [C]
0
100
200
300
400
500
600
700
800
Sec
pkg0(mean)
pkg1(mean)
pkg0(sd)
pkg1(sd)


90
80
70
60
50
40
30
Temperature [C]
0
100
200
300
400
500
600
700
800
Sec
core00
core01
core02
core03
core04
core05
core06
core07



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.

U.S. DEPARTMENT OF ENERGY | Office of Science

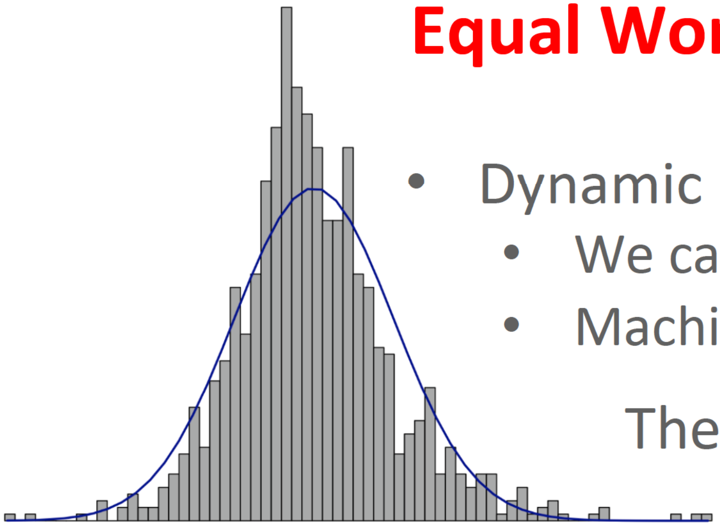
DOE Exascale Research Conference, April 16-18th, 2012 2

From Brian Van Straalen

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



Variability is the new norm:
Power
Resilience
Intranode Contention

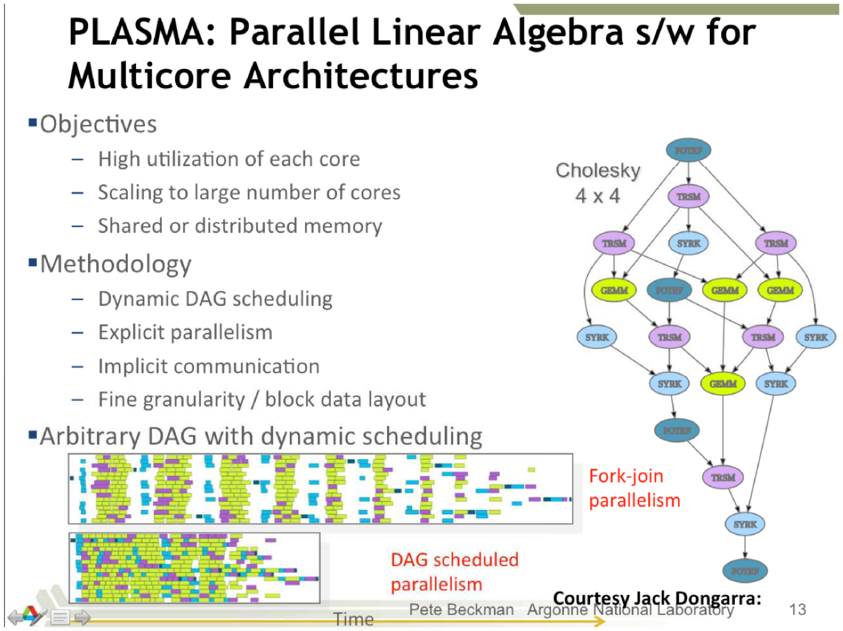
- Dynamic parallelism and decomposition
  - We cannot hand-pick granularity / resource mapping
  - Machine Learning?

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

This is not new…
Dynamic Lightweight Parallelism

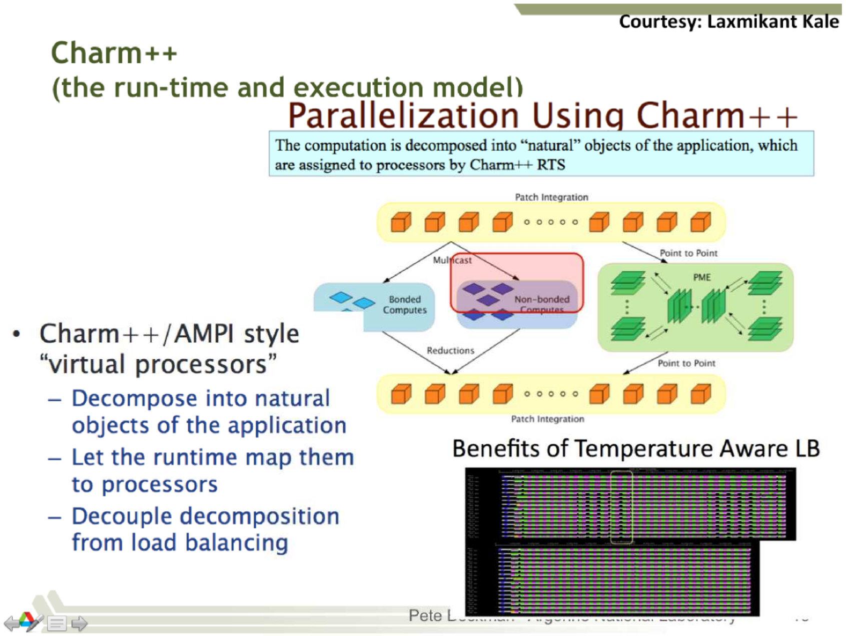
But what will pervasive look like?



PLASMA: Parallel Linear Algebra s/w for Multicore Architectures

- Objectives
  - 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

Courtesy Jack Dongarra:



Courtesy: Laxmikant Kale

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++ RTS

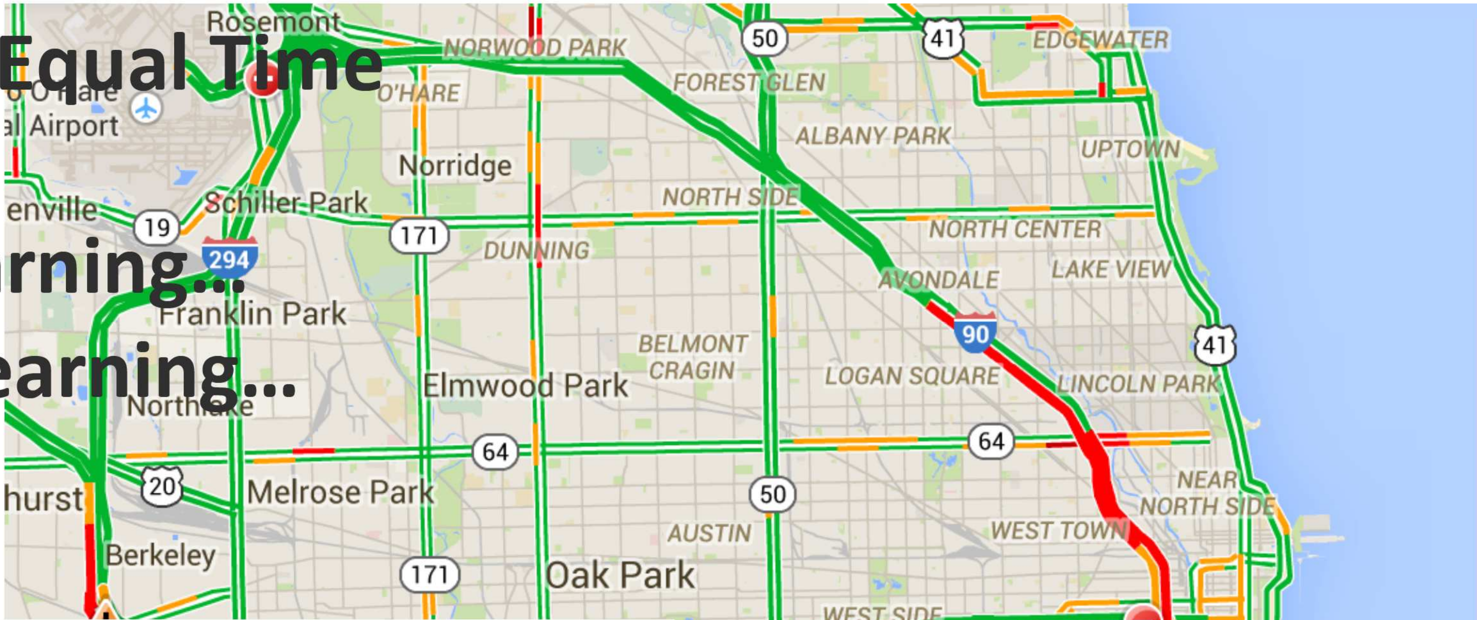
- Charm++/AMPI style "virtual processors"
  - Decompose into natural objects of the application
  - Let the runtime map them to processors
  - Decouple decomposition from load balancing

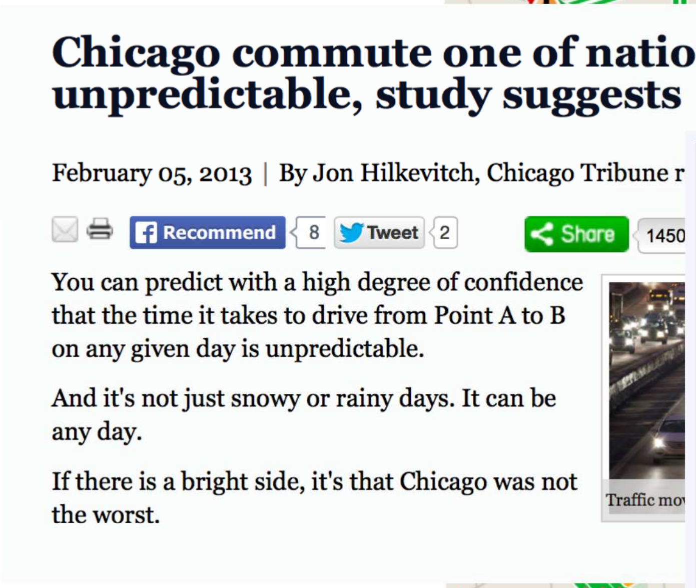
Benefits of Temperature Aware LB

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 r

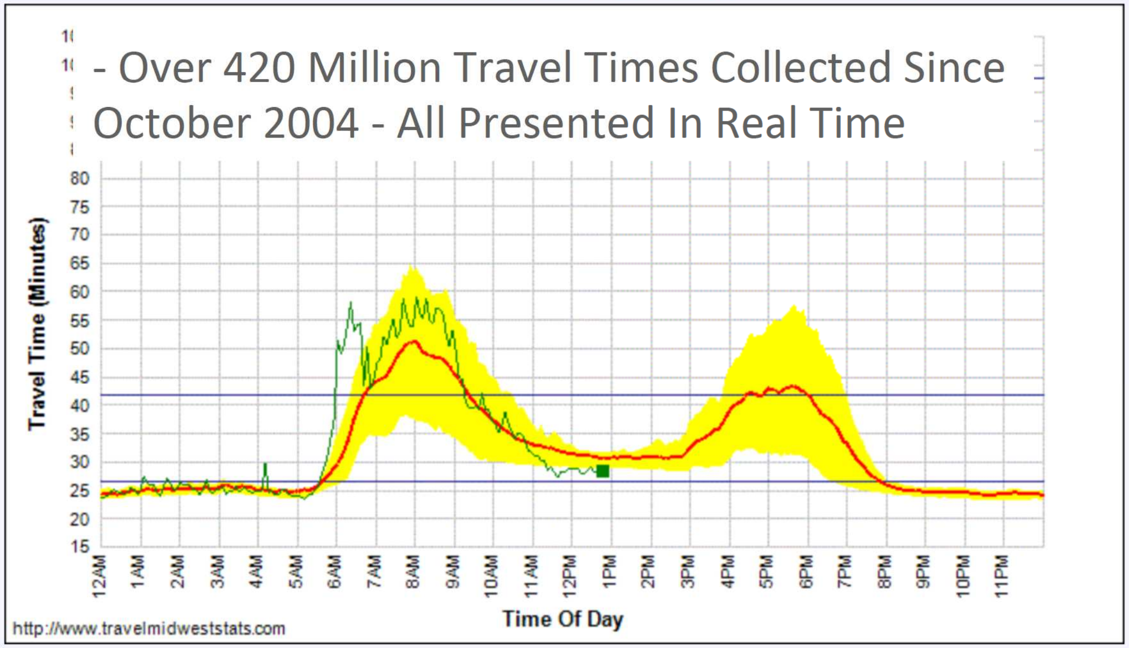
Recommend 8 Tweet 2 Share 1450

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.

Traffic mov



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

Serious Cognitive Dissonance

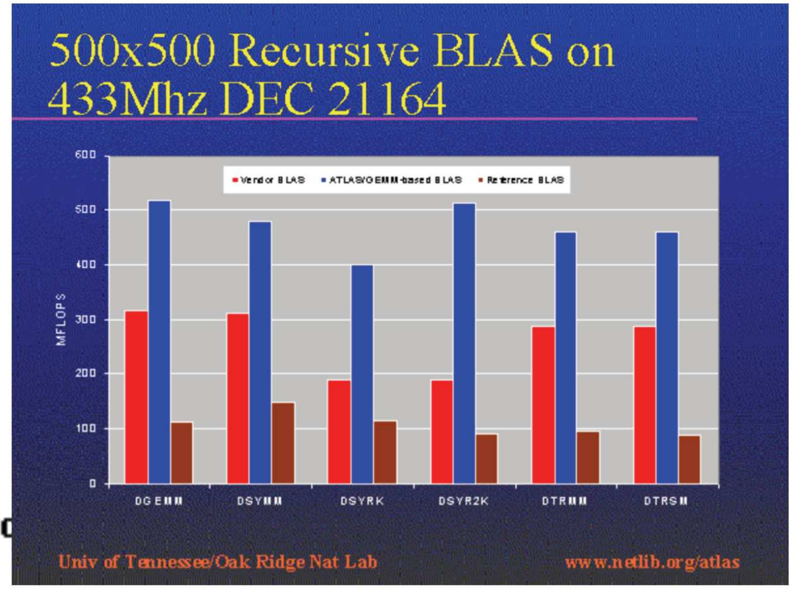
Even Today, we have the “Dynamic Deniers”

(We want low runtime variance)

- **Trinity/NERSC-8:**

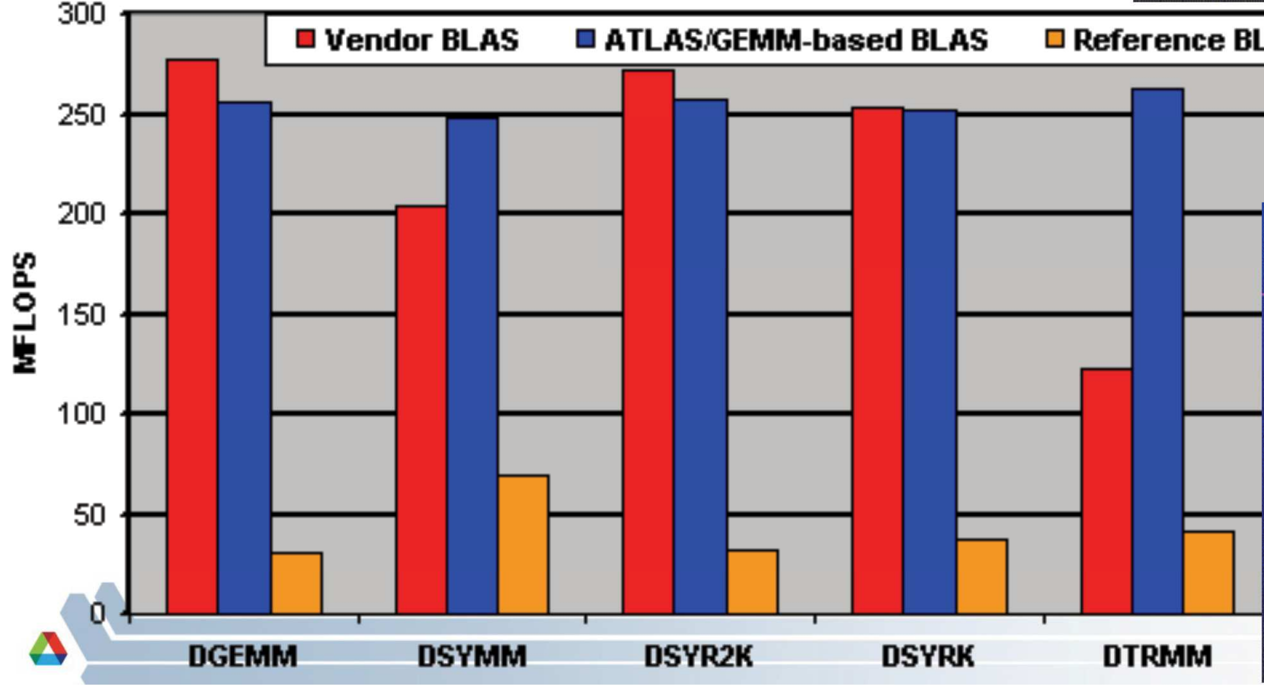
“The system shall provide correct and consistent runtimes. An application’s runtime (i.e. wall clock time) shall not change by more than 3% from run-to-run in dedicated mode and 5% in production mode.“

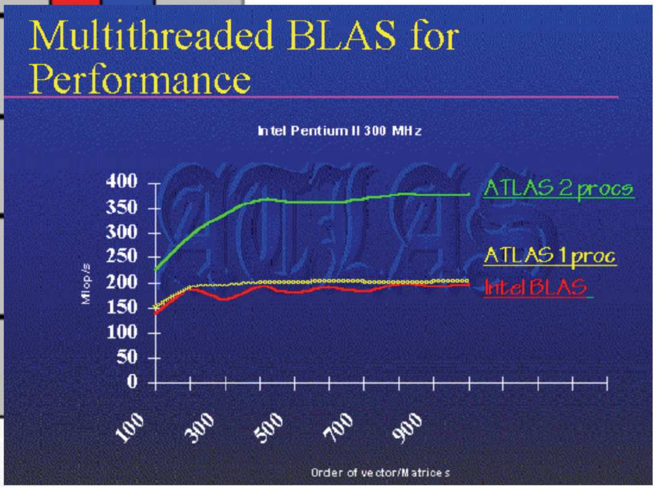
Automatically Tuned Linear Algebra Software (ATLAS) 15 yrs ago…



But static….

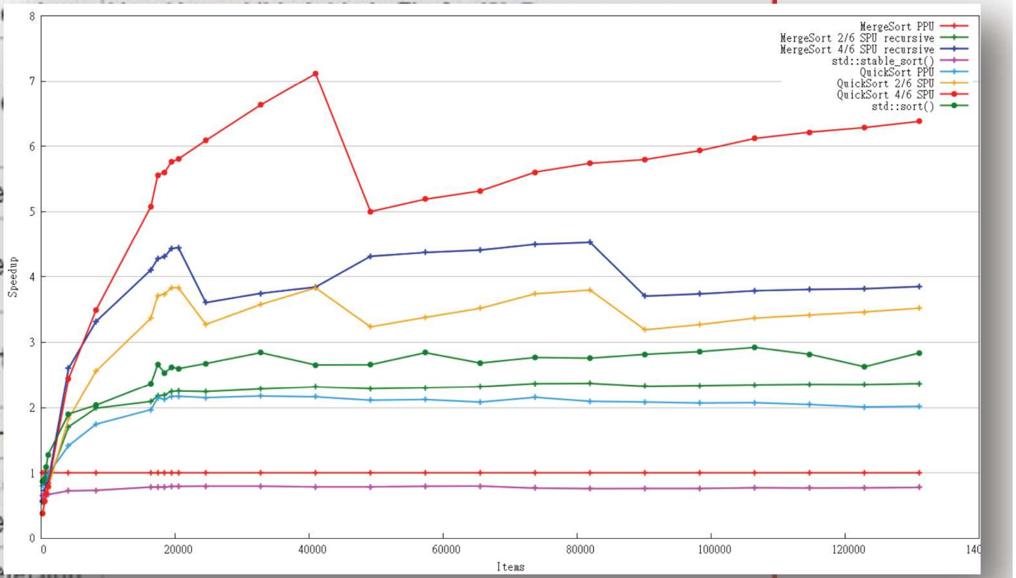
Level 3 BLAS On One Processor of a Sun UltraSparc



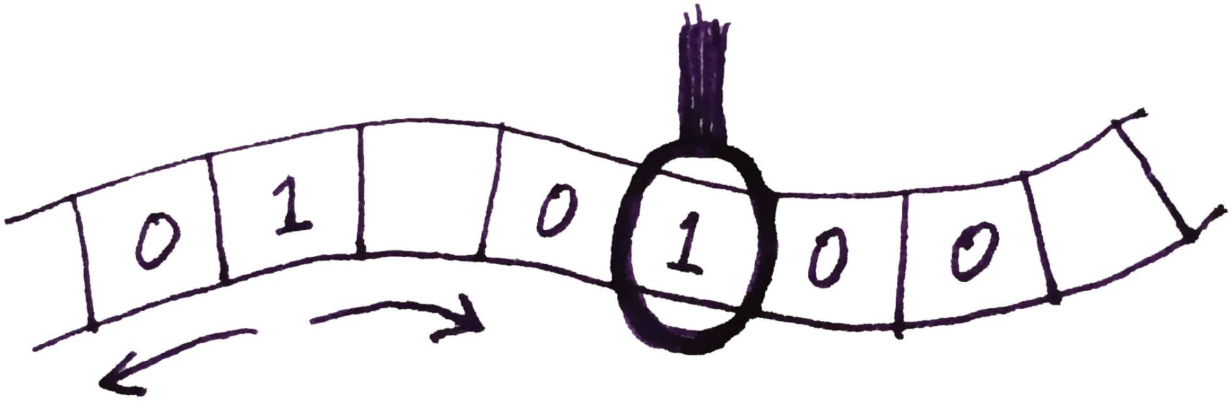


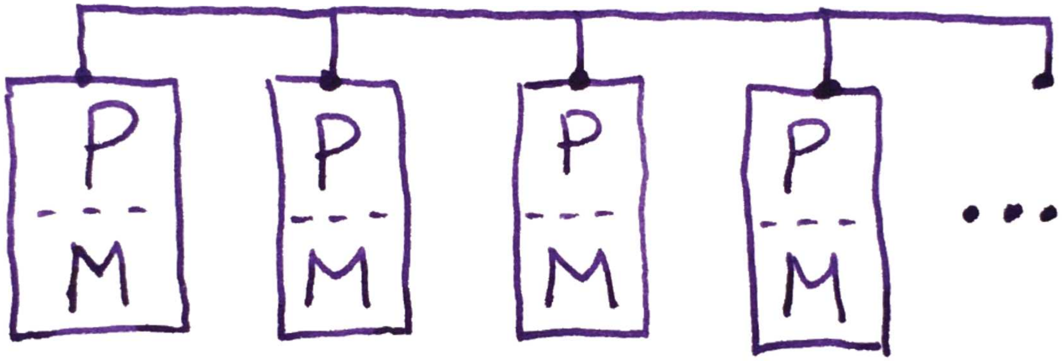
Name	Best	Average	Worst	Memory	Stable	Method	Other notes
Timsort	—	$n \log n$	$n \log n$	n	Yes	Insertion & Merging	n comparisons when the data is already sorted.
Bubble sort	n	n^2	n^2	1	Yes	Exchanging	Tiny code
Cocktail sort	—	—	n^2	1	Yes	Exchanging	
Comb sort	—	—	—	1	No	Exchanging	Small code size
Gnome sort	—	—	n^2	1	Yes	Exchanging	Tiny code size
Selection sort	n^2	n^2	n^2	1	No	Selection	Its stability depends on this table in Safari or ot
Insertion sort	n	n^2	n^2	1	Yes	Insertion	Average case is also O number of inversions
Cycle sort	—	n^2	n^2	1	No	Insertion	In-place with theoretically optimal number of writes
Shell sort	—	—	$n \log^2 n$	1	No	Insertion	
Binary tree sort	—	$n \log n$	$n \log n$	n	Yes	Insertion	When using a self-balancing binary search tree
Library sort	—	$n \log n$	n^2	n	Yes	Insertion	
Merge sort	$n \log n$	$n \log n$	$n \log n$	Depends	Yes	M	
In-place merge sort	—	$n \log n$	$n \log n$	1	Depends	M	
Heapsort	$n \log n$	$n \log n$	$n \log n$	1	No	Se	
Smoothsort	—	—	$n \log n$	1	No	Se	
Quicksort	$n \log n$	$n \log n$	n^2	$\log n$	Depends	Par	
Introsort	—	$n \log n$	$n \log n$	$\log n$	No	H	
Patience sorting	—	—	$n \log n$	n	No	Ins Se	
Strand sort	—	$n \log n$	n^2	n	Yes	Selection	
Tournament sort	—	$n \log n$	$n \log n$			Selection	

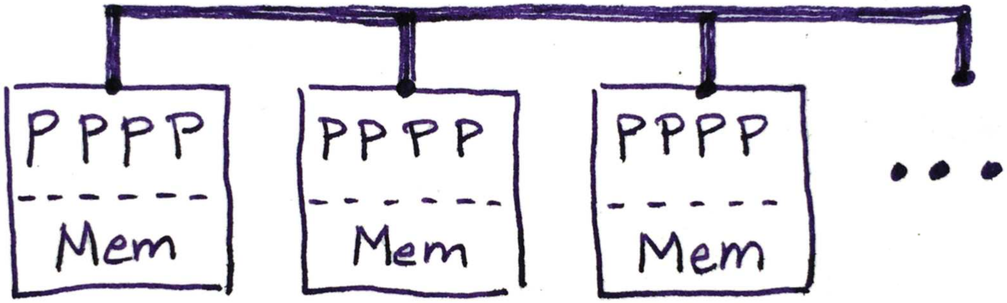
Fault?
Dynamic Execution?
Parallelism?



Abstractions Matter




P
M
P
M
P
M
P
M
...


PPPP
Mem
PPPP
Mem
PPPP
Mem
...

Question: Abstractions for the
Future Massively Parallel, Dynamic Machine?

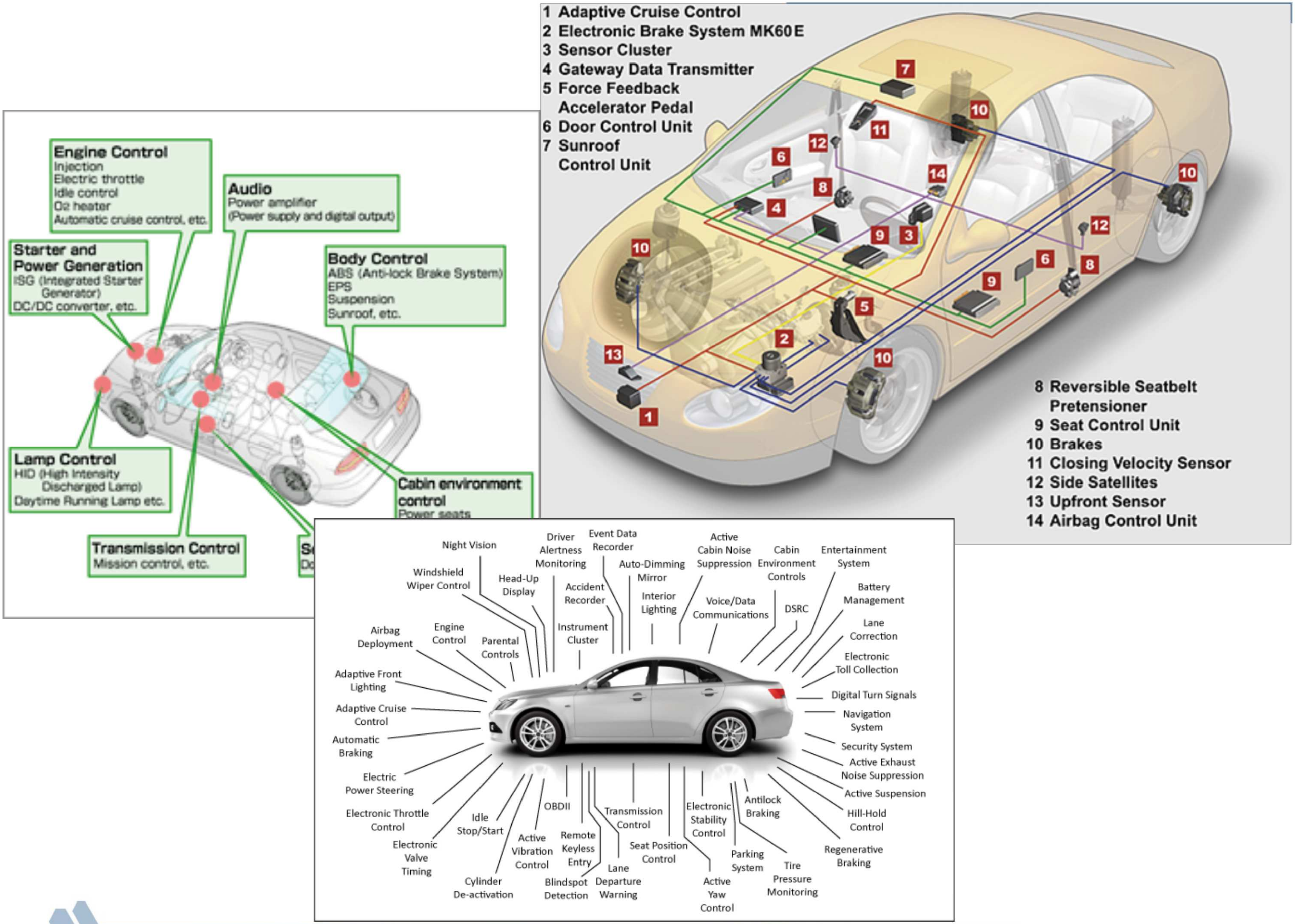
Ignore Device Technology
Ignore Cartoons of future chips

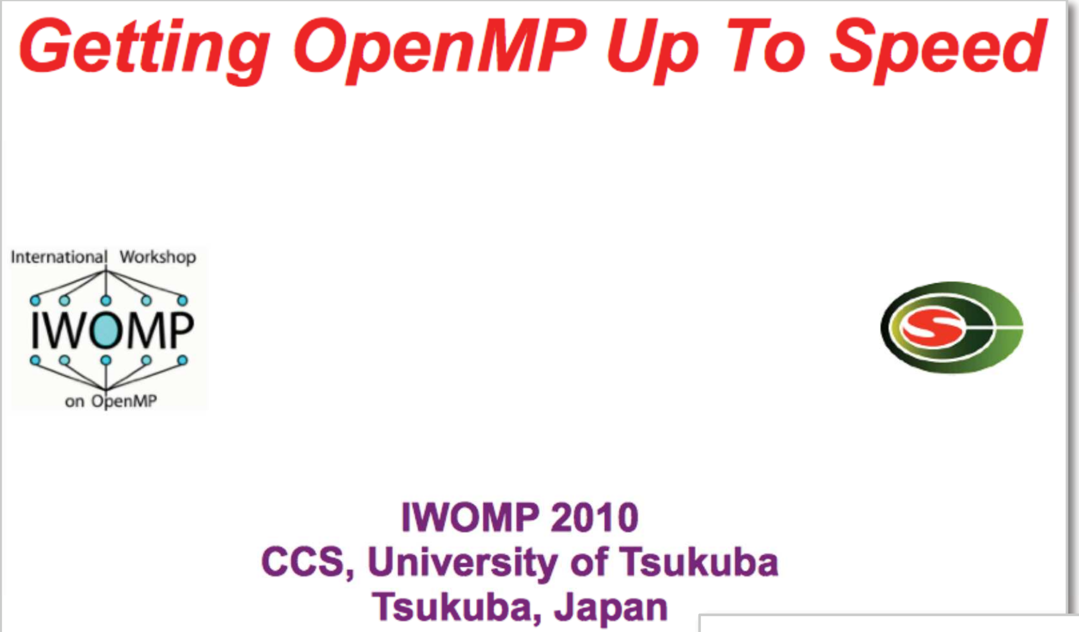
Focus on Parallelism
Unpredictable Performance,
Interacting control systems
Unpredictable Fault

Harold's
TEXACO
BOTHELL, WN.
HU. 6-6441
Ford

MSD IGNITION
Multiple Spark Discharge
Ford Racing
HOLLEY
POWERED BY FORD
Stant
TEST OR REPLACE WHEN CHANGING COOLANT
CAUTION PRESSURE DO NOT OPEN WHEN HOT
R 30 16 LBS
TAYLOR SPIRO PRO SILICONE MADE IN U.S.A.

AMG
AMG
AMG

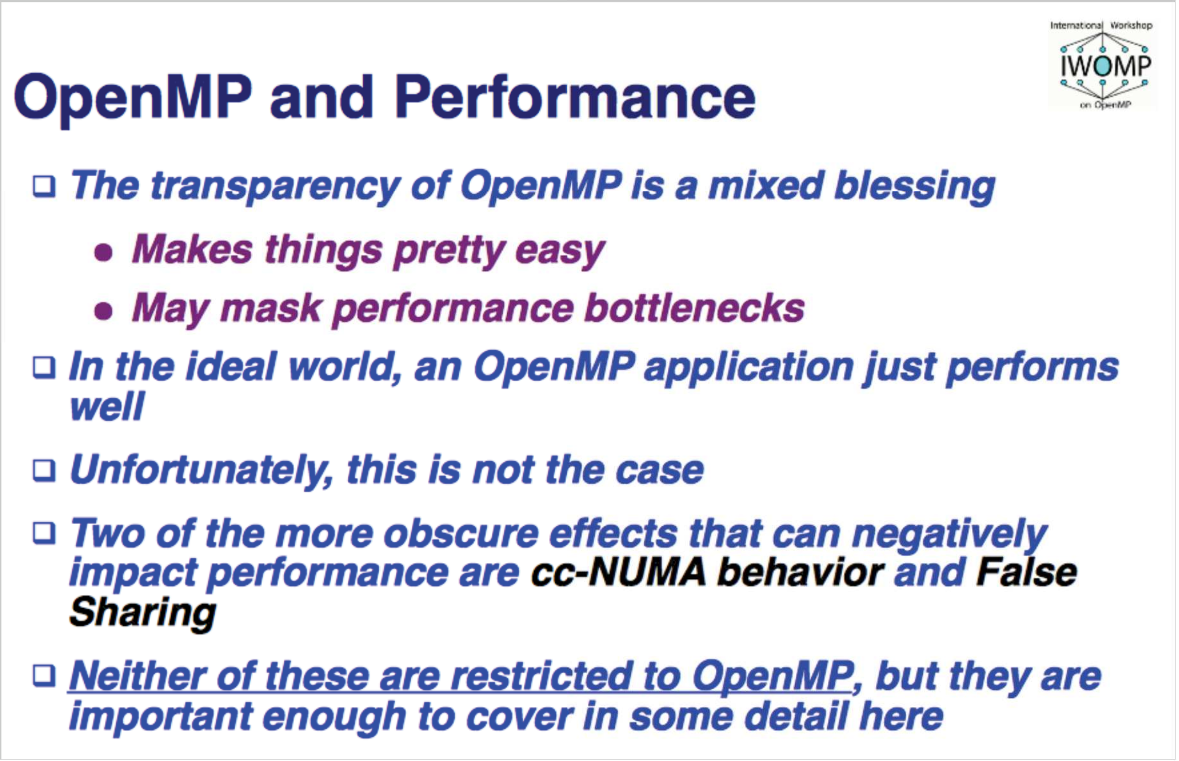

Engine Control
Injection
Electric throttle
Idle control
O2 heater
Automatic cruise control, etc.
Audio
Power amplifier
(Power supply and digital output)
Starter and Power Generation
ISG (Integrated Starter Generator)
DC/DC converter, etc.
Body Control
ABS (Anti-lock Brake System)
EPS
Suspension
Sunroof, etc.
Lamp Control
HID (High Intensity Discharged Lamp)
Daytime Running Lamp etc.
Cabin environment control
Power seats
Transmission Control
Mission control, etc.
Se
Do
1 Adaptive Cruise Control
2 Electronic Brake System MK60E
3 Sensor Cluster
4 Gateway Data Transmitter
5 Force Feedback Accelerator Pedal
6 Door Control Unit
7 Sunroof Control Unit
8 Reversible Seatbelt Pretensioner
9 Seat Control Unit
10 Brakes
11 Closing Velocity Sensor
12 Side Satellites
13 Upfront Sensor
14 Airbag Control Unit
7
10
11
12
6
8
14
4
9
3
10
12
6
8
9
5
2
13
10
1
10
Night Vision
Driver Alertness Monitoring
Event Data Recorder
Active Cabin Noise Suppression
Cabin Environment Controls
Entertainment System
Windshield Wiper Control
Head-Up Display
Accident Recorder
Auto-Dimming Mirror
Interior Lighting
Voice/Data Communications
DSRC
Battery Management
Lane Correction
Airbag Deployment
Engine Control
Parental Controls
Instrument Cluster
Electronic Toll Collection
Adaptive Front Lighting
Digital Turn Signals
Adaptive Cruise Control
Navigation System
Automatic Braking
Security System
Electric Power Steering
Active Exhaust Noise Suppression
Active Suspension
Electronic Throttle Control
Idle Stop/Start
OBDII
Transmission Control
Electronic Stability Control
Antilock Braking
Hill-Hold Control
Electronic Valve Timing
Active Vibration Control
Remote Keyless Entry
Seat Position Control
Parking System
Regenerative Braking
Cylinder De-activation
Blindspot Detection
Lane Departure Warning
Active Yaw Control
Tire Pressure Monitoring



Getting OpenMP Up To Speed

International Workshop
IWOMP
on OpenMP

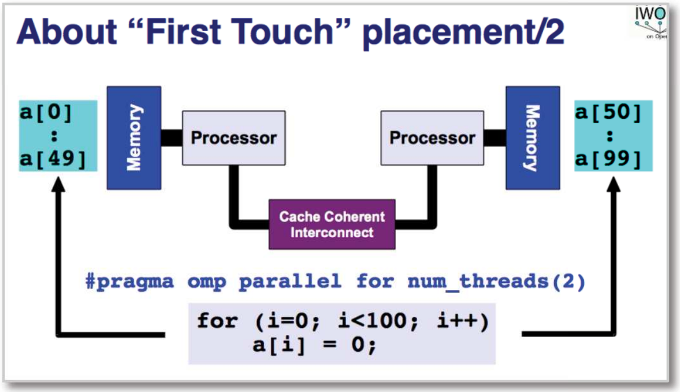
IWOMP 2010
CCS, University of Tsukuba
Tsukuba, Japan
June 14-16, 2010



OpenMP and Performance

International Workshop
IWOMP
on OpenMP

- ❑ *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*



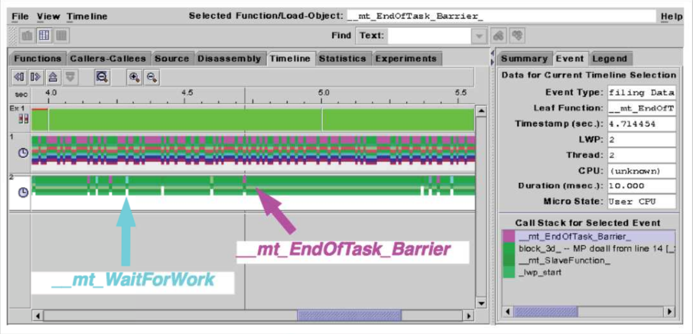
About “First Touch” placement/2

```
#pragma omp parallel for num_threads(2)
for (i=0; i<100; i++)
   a[i] = 0;
```

A 3D matrix update

```
      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
```

This is False Sharing at work !



Selected Function/Load-Object: __mt_EndOfTask_Barrier_

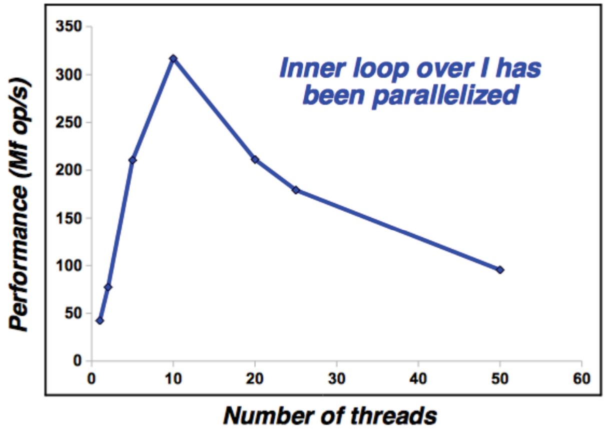
Data for Current Timeline Selection

Event Type: filing Data
Leaf Function: __mt_EndOfT
Timestamp (sec.): 4.714454
LWP: 2
Thread: 2
CPU: (unknown)
Duration (msec.): 10.000
Micro State: User CPU

Call Stack for Selected Event

__mt_EndOfTask_Barrier_
block_3d_ -- MP doall from line 14 [_
__mt_SlaveFunction_
_lwp_start

The performance



Scaling is very poor (as to be expected)

Performance Analyzer data

Using 10 threads

```
Name                                                  Excl. User    Incl.     Excl.
                                                      CPU           User CPU  Wall
                                                        sec.     %    sec.     sec.
<Total>                                               10.590 100.0  10.590    1.550
__mt_EndOfTask_Barrier_                                5.740  54.2   5.740    0.240
__mt_WaitForWork_                                      3.860  36.4   3.860    0.
__mt_MasterFunction_                                   0.480   4.5   0.680    0.480
MAIN_                                                  0.230   2.2   1.200    0.470
block_3d_ -- MP doall from line 14 [_$d1A14       3d_]  0.170   1.6   5.910    0.170
block_3d_                                              0.040   0.4   6.460    0.040
memset                                                 0.030   0.3   0.030    0.080
```

Using 20 threads

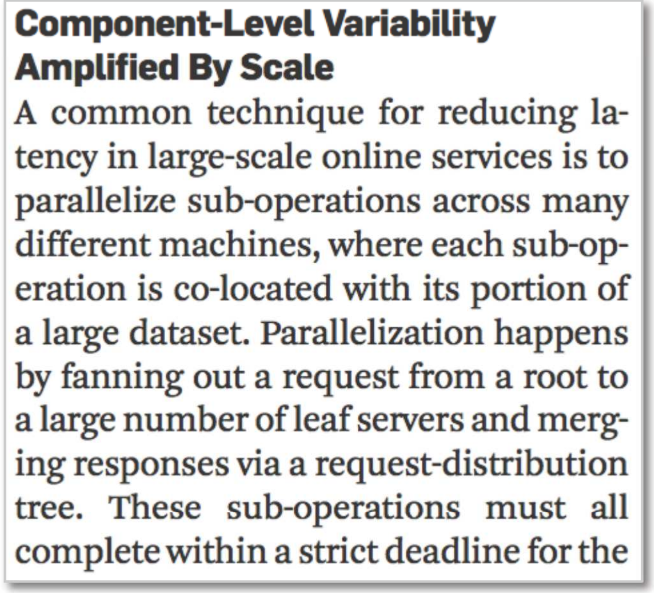
```
Name                                                  Excl. User    Incl.     Excl.
                                                       CPU          User CPU  Wall
                                                        sec.     %    sec.     sec.
<Total>                                               47.120 100.0  47.120    2.900
__mt_EndOfTask_Barrier_                               25.700  54.5  25.700    0.980
__mt_WaitForWork_                                     19.880  42.2  19.880    0.
__mt_MasterFunction_                                   1.100   2.3   1.320    1.100
MAIN_                                                  0.190   0.4   2.520    0.470
block_3d_ -- MP doall from line 14 [_$d1A14.block_3d_] 0.100   0.2  25.800    0.100
__mt_setup_doJob_int_                                  0.080   0.2   0.080    0.080
__mt_setup_job_                                        0.020   0.0   0.020    0.020
block_3d_                                              0.010   0.0  27.020    0.010
```

do not scale at all

scales somewhat

Question: Why is __mt_WaitForWork so high in the profile ?

Google (re-discovers) Noise

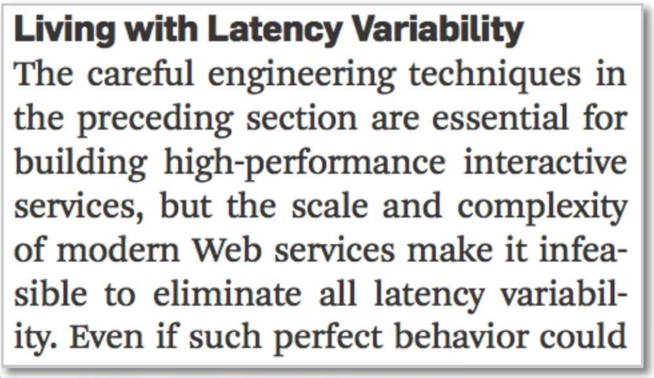


Component-Level Variability Amplified By Scale

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

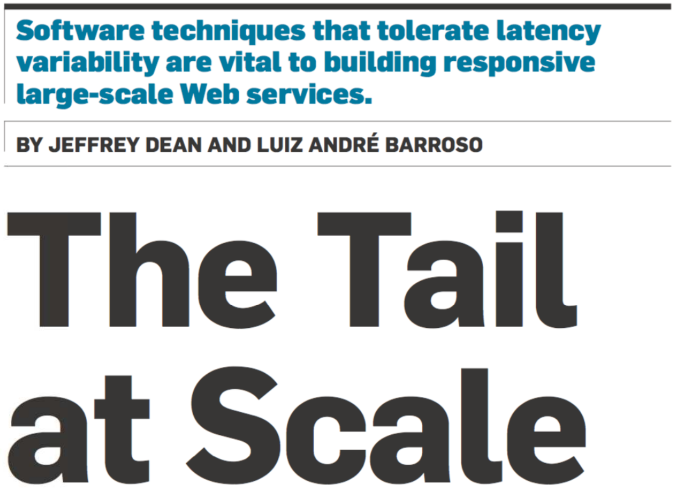
Reducing Component Variability

Interactive response-time variability can be reduced by ensuring interactive requests are serviced in a timely manner



Living with Latency Variability

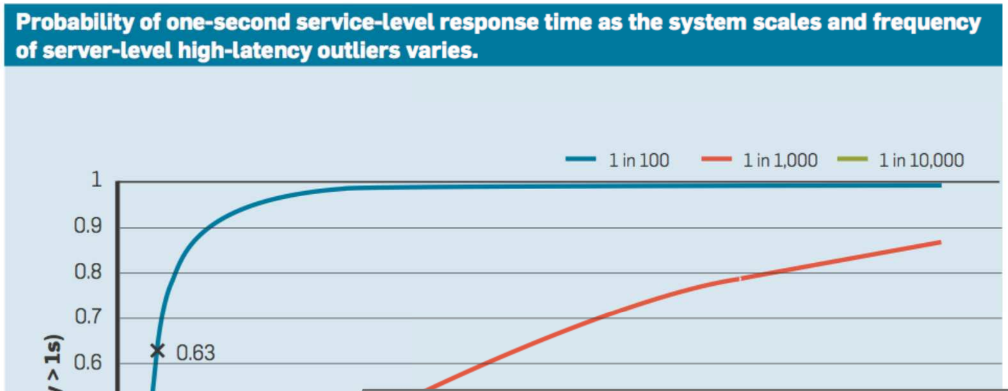
The careful engineering techniques in the preceding section are essential for building high-performance interactive services, but the scale and complexity of modern Web services make it infeasible to eliminate all latency variability. Even if such perfect behavior could



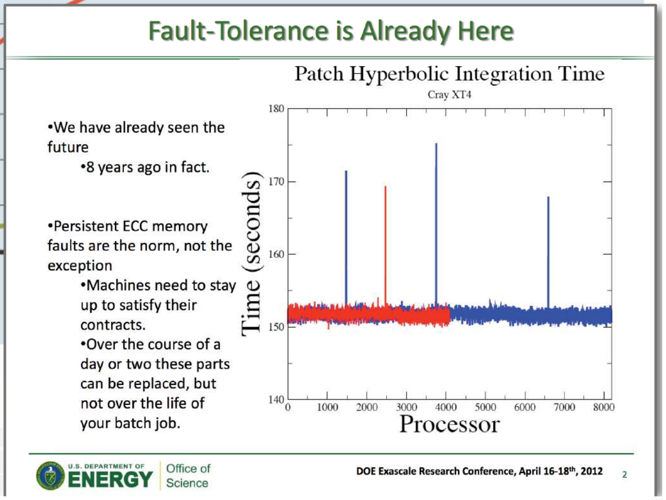
Software techniques that tolerate latency variability are vital to building responsive large-scale Web services.

BY JEFFREY DEAN AND LUIZ 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.



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.

U.S. DEPARTMENT OF ENERGY | Office of Science

DOE Exascale Research Conference, April 16-18th, 2012 2

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