Tux Faces the Rigors of Terascale Computation:

High-Performance Computing and the Linux Kernel

David Cowley

Pacific Northwest National Laboratory

Information Release #PNNL-SA-65780

THE LINUX FOUNDATION

EMSL is a national scientific user facility at the Pacific Northwest National Laboratory

EMSL—the Environmental Molecular Science Laboratory—located in Richland, Washington, is a national scientific user facility funded by the DOE. EMSL provides integrated experimental and computational resources for discovery and technological innovation in the environmental molecular sciences to support the needs of DOE and the nation.

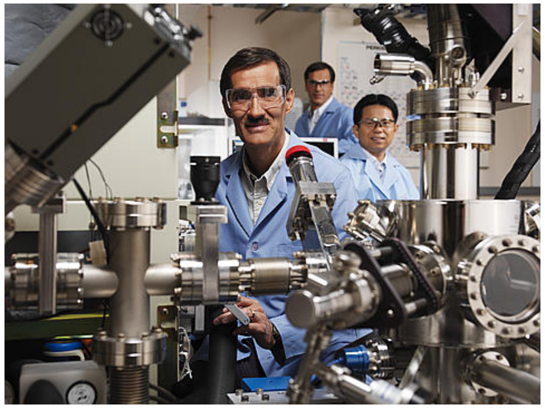
Washington
Sequim
Seattle
Spokane
Pacific Northwest National Laboratory
Pasco
Snake River
Richland
Kennewick
Walla Walla
Columbia River

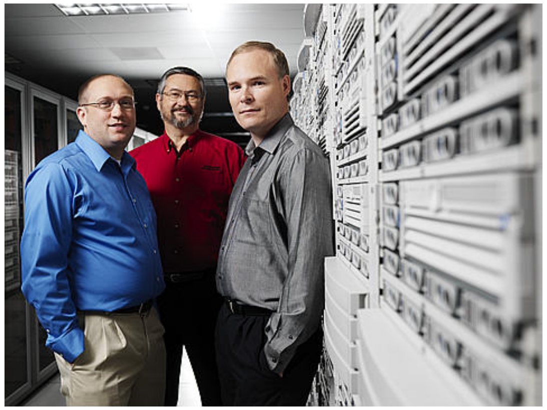
William R. Wiley, founder

William R. Wiley's Vision:
An innovative multipurpose user facility providing *"synergism between the physical, mathematical, and life sciences."*

Visit us at
www.emsl.pnl.gov

Characteristics of EMSL



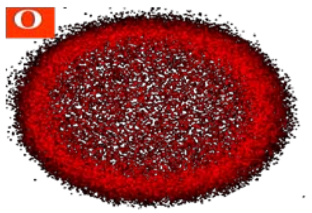


- Scientific **expertise** that enables scientific discovery and innovation.
- Distinctive focus on **integrating** computational and experimental capabilities and collaborating among disciplines.
- A **unique collaborative environment** that fosters synergy between disciplines and a complimentary suite of tools to address the science of our users.
- An impressive suite of **state-of-the-art instrumentation** that pushes the boundaries of resolution and sensitivity.
- An **economical** venue for conducting non-proprietary research.

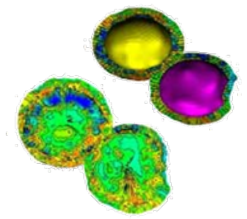
High-performance computing in EMSL

- **EMSL uses high-performance computing for:**
  - Chemistry
  - Biology (which can be thought of as chemistry on a larger scale)
  - Environmental systems science.
- **We will focus primarily on quantum computational chemistry**

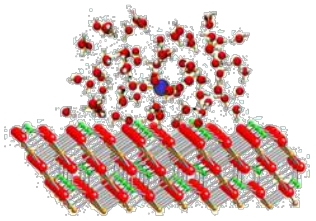
Atmospheric Aerosol Chemistry (developing science theme)



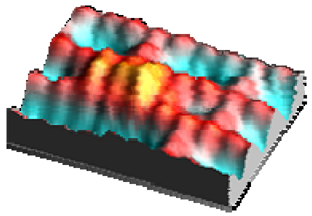
Biological Interactions & Dynamics



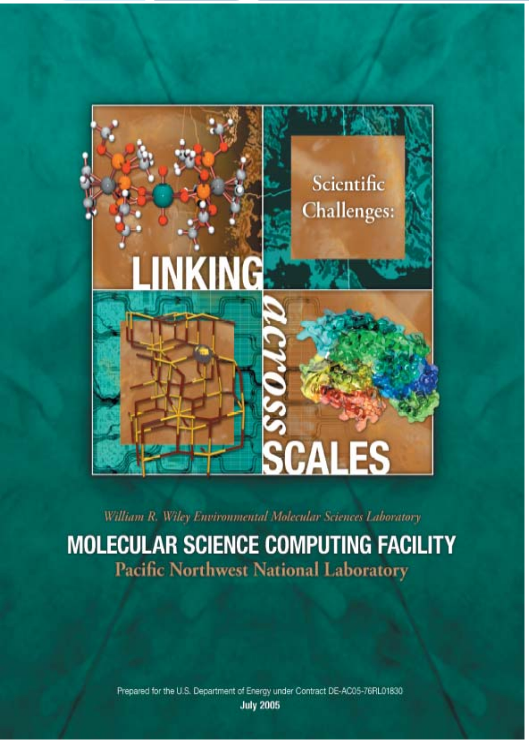
Geochemistry/ Biogeochemistry & Subsurface Science



Science of Interfacial Phenomena



Defining high-performance computing hardware for EMSL science


Scientific Challenges:
LINKING across SCALES
William R. Wiley Environmental Molecular Sciences Laboratory
MOLECULAR SCIENCE COMPUTING FACILITY
Pacific Northwest National Laboratory
Prepared for the U.S. Department of Energy under Contract DE-AC05-76RL01830
July 2005

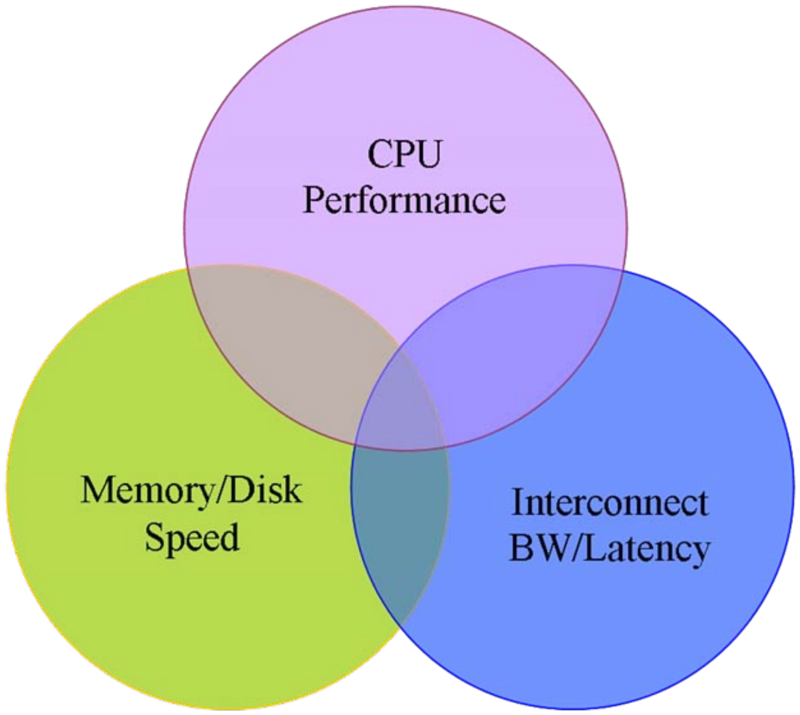
Scientists project science needs in a 'Greenbook'

Hardware Features Summary

Hardware Feature	B	C	E
Memory hierarchy (bandwidth, size and latency)	X	X	X
Peak flops (per processor and aggregate)	X	X	X
Fast integer operations	X		
Overlap computation, communication and I/O	X	X	X
Low communication latency	X	X	X
High communication bandwidth		X	
Large processor memory	X	X	
High I/O bandwidth to temporary storage		X	
Increasing global and long-term disk storage needs (size)	X		X
B = Biology; C = Chemistry; E = Environmental Systems Science			

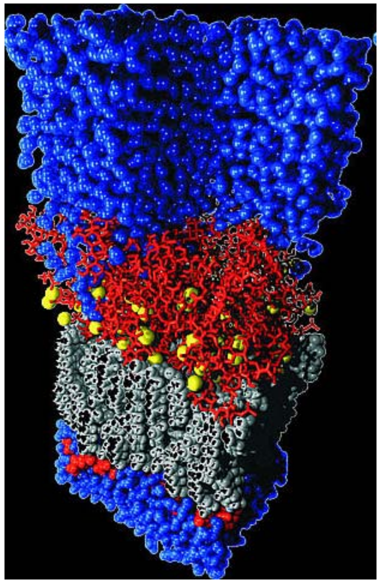
The need for a *balanced* system

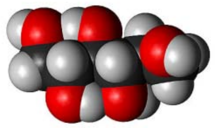
- **From a certain point of view, the idea is to get the most math done in the least amount of time**
- **We need a good *balance* of system resources to accomplish this**
- **That data may need to come from many far-flung places**
  - CPU cache
  - Local RAM
  - Another node's RAM
  - Local disk
  - Non-local disk.
- **RAM, disk, and interconnect all need to be fast enough to keep processors from starving**

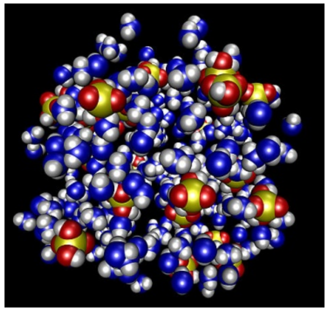


So what's the big deal about quantum chemistry?

- **We want to understand the properties of molecular systems**
- **Quantum models are very accurate, but**
  - Properties from tens or hundreds of atoms are possible, but they want more
  - Biologists need *many* more atoms.
- **The more atoms, the more compute intensive!**
  - *We can get very accurate results*
  - *We can do it in a reasonable amount of time*
  - *Pick one!*







Quantum chemistry 101 x 10^{-3}

- **We want to understand the behavior of large molecular systems**
- **The number of electrons governs the amount of calculation Electrons are represented mathematically by *basis functions***
  - Basis functions combine forming wave functions, which describe the probabilistic behavior of a molecule's electrons
  - More basis functions make for better results, but *much* more computation.
- **N is a product of atoms and basis functions**
- **The chemist chooses a computational method, trading off accuracy against speed:**

Computational Method	Order of Scaling
Empirical Force Fields	O(N) (number of atoms only)
Density Functional Theory	$O(N^3)$
Hartree-Fock	$O(N^4)$
Second-Order Hartree-Fock	$O(N^5)$
Coupled Cluster	$O(N^7)$
Configuration Interaction	O(N!)

THE LINUX FOUNDATION

The awful arithmetic of scaling

- **“This scales on the order of N^7”**
- **How bad is that?**
- **Consider two values:**
  - N = 40 (2 water molecules, 10 basis functions per oxygen, 5 per hydrogen)
  - N = 13200 (C_6H_{14}, 264 basis functions, 50 electrons)

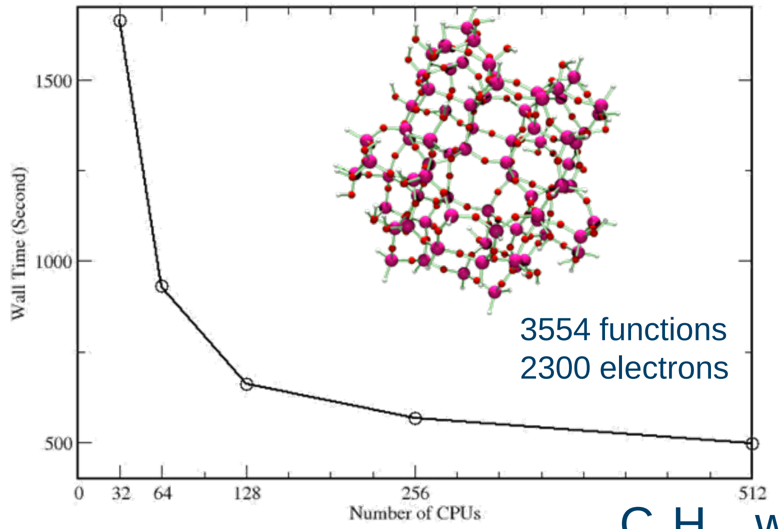
Computational Method	Order of Scaling	“Difficulty” of N=40	“Difficulty” of N=13200	How many atoms can we do?
Empirical Force Fields	O(N)	40	13,200	1,000,000
Density Functional Theory	$O(N^3)$	64,000	2,299,968,000,000	3,000
Hartree-Fock	$O(N^4)$	2,560,000	30,359,577,600,000,000	2,500
Second-Order Hartree-Fock	$O(N^5)$	102,400,000	400,746,424,320,000,000,000	800
Coupled Cluster	$O(N^7)$	163,840,000,000	69,826,056,973,516,800,000,000,000,000	24
Configuration Interaction	O(N!)	8.15915×10^{47}	Just forget it!	4

Pfister tells us there are three ways to compute faster

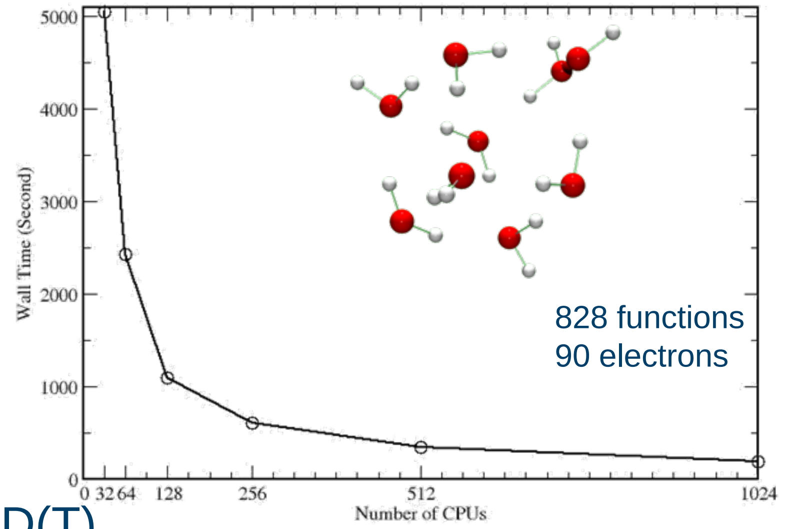
- **Use faster processors**
  - Moore’s law gives us 2x the transistor count in our CPUs every 18 months
  - That’s not a fast enough rate of acceleration for us.
- **Use faster code**
  - Optimizing code can be slow, expensive, dirty work
  - It doesn’t pay off very consistently.
- **Use more processors**
  - The good news: Chip manufacturers are passing out cores like candy!
  - The bad news: Bandwidth ain’t keeping up!
  - Still, this gives us the biggest payoff
  - GPUs? There may be some promise there.

Sample scaling curves, 32 to 1024 cores

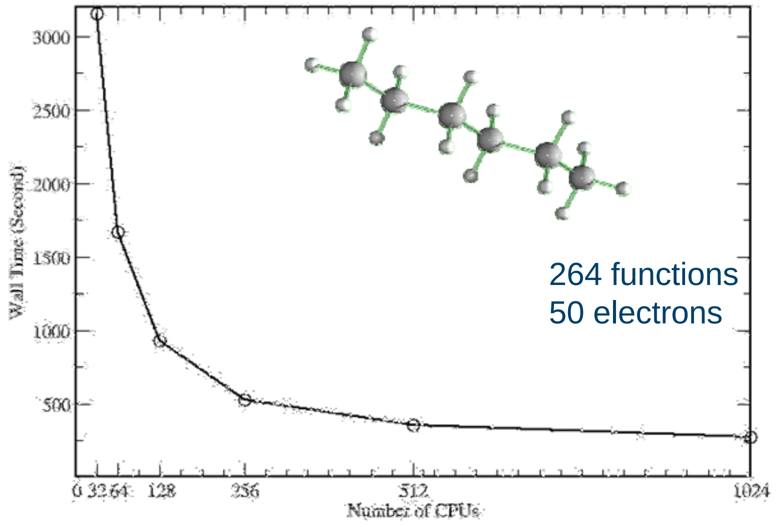
$Si_{75}O_{148}H_{66}$ with DFT



$(H_2O)_9$ with MP2



C_6H_{14} with CCSD(T)



The method of choice is clearly to use more processors, and wow, do we need them!

- User input tells us they want "several orders of magnitude" more computation in a new system
- Cell membrane simulations need to be at least *thousands* of atoms, with many electrons per atom
- We can just now, with a 160-teraflop system, start to simulate systems with several hundred molecules and get reasonable accuracy
- We want to do more than that. *Much* more than that!

Introducing a high-performance computing cluster: Compute nodes

- **Our clusters have hundreds or thousands of compute nodes**
- **Each node has had:**
  - One or more processor cores
  - Its own instance of the Linux kernel
  - Some gigabytes of RAM
  - A high-performance cluster interconnect (QSNet, Infiniband, etc.)
  - Local disk
  - Access to a shared parallel filesystem.
- **We are currently at a RHEL 4.5 code base with a 2.6.9-67 kernel**
  - (We'd like to be much more current).

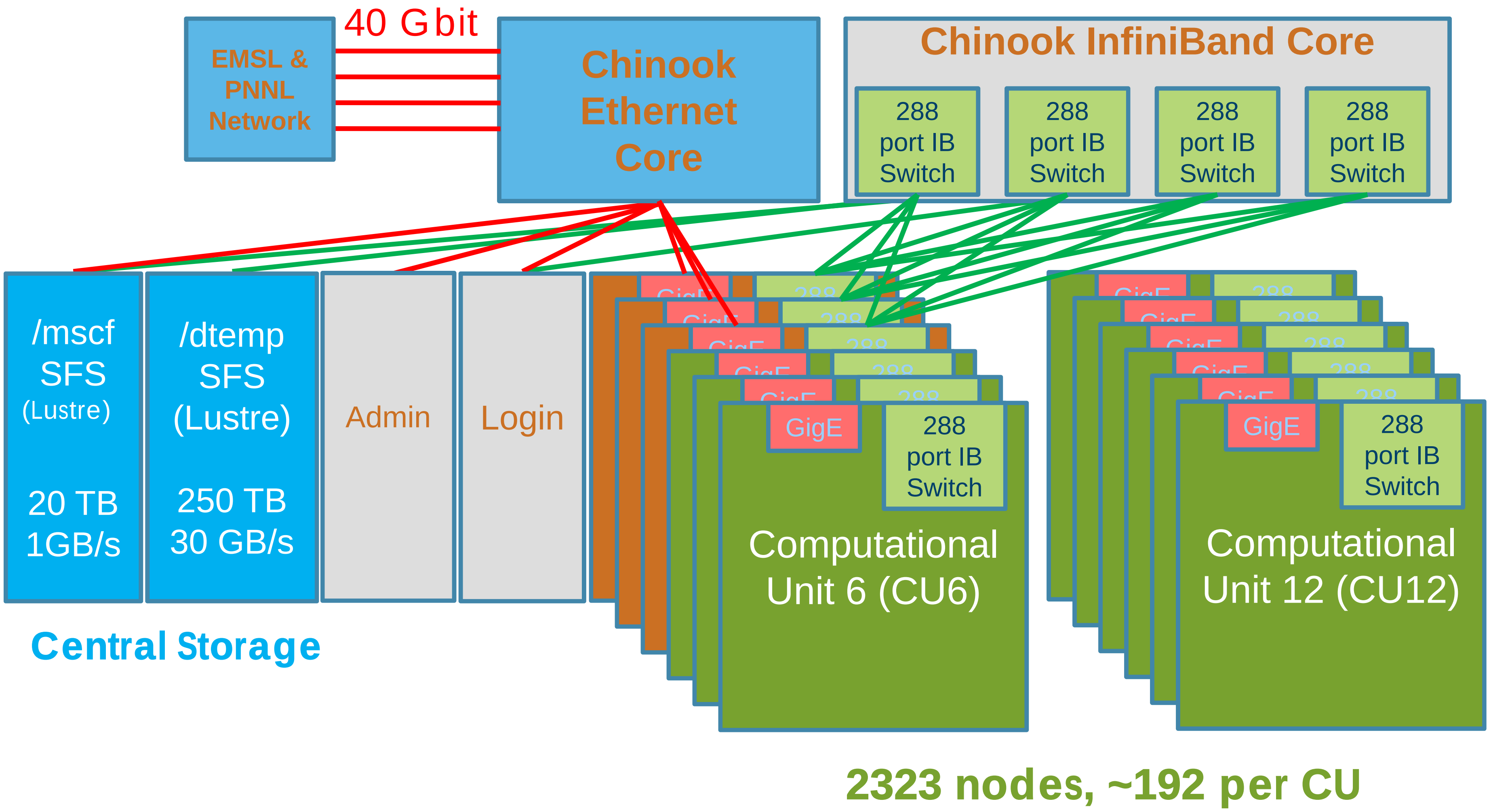
Chinook

2323 node HP cluster

Feature	Detail
Interconnect	DDR InfiniBand (Voltaire, Mellanox)
Node	Dual Quad-core AMD Opteron 32 GB memory
Local scratch filesystems	440 MB/s, 1 TB/s aggregate 440 GB per node. 1 PB aggregate
Global scratch filesystem	30 GB/s 250 TB total
User home filesystem	1 GB/s 20 TB total

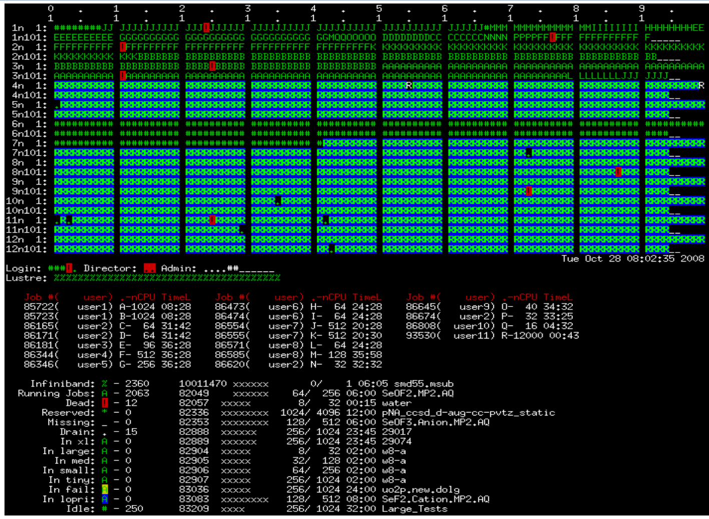
THE LINUX FOUNDATION

Chinook cluster architecture



Typical cluster infrastructure

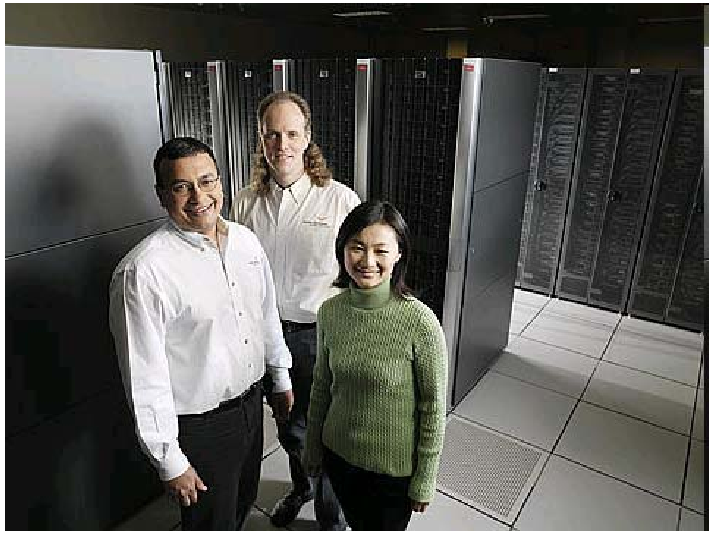
- **Parallel batch jobs are our stock in trade**
  - Jobs run on anywhere from 64 to 18,000 cores
  - Jobs get queued up and run when our scheduler software decides it’s time
  - The user gets the results at the end of the job.
- **To support them, we provide:**
  - High-performance shared parallel filesystem
  - Shared home filesystem
  - Batch queueing/scheduling software
  - Interconnect switches
  - System administrators
  - Scientific consultants
  - Parallel software.



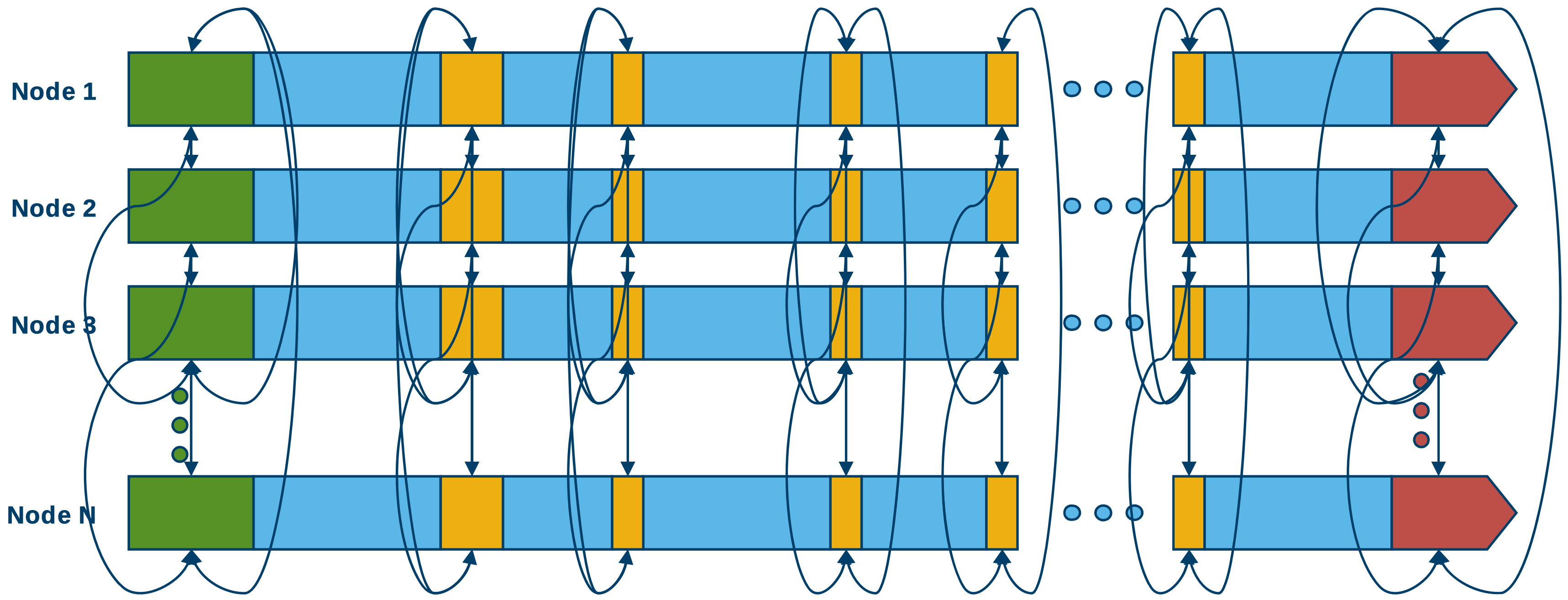
```
         0    .    1    .    2    .    3    .    4    .    5    .    6    .    7    .    8    .    9    .
         1    .    1    .    1    .    1    .    1    .    1    .    1    .    1    .    1    .    1    .
  1n   1: ########JJ JJJJJJJJJJ JJJ!JJJJJJ JJJJJJJJJJ JJJJJJJJJJ JJJJJJJJJJ JJJJJJ#MMM MMMMMMMMMM MMIIIIIIII HHHHHHHHEE
  1n101: EEEEEEEEEE GGGGGGGGGG GGGGGGGGGG GGGGGGGGGG GGMQQOOOOO DDDDDDDDCC CCCCCCNNNN PPPPPF!FFF FFFFFFFFFF F_____
  2n   1: FFFFFFFFFF !FFFFFFFFF FFFFFFFFFF FFFFFFFFFF FFFFFFFFFK KKKKKKKKKK KKKKKKKKKK KKKKKKKKKK KKKKKKKKKK KKKKKKKKKK
  2n101: KKKKKKKKKK KKKBBBBBBB BBBBBBBBBB BBBBBBBBBB BBBBBBBBBB BBBBBBBBBB BBBBBBBBBB BBBBBBBBBB BBBBBBBBBB BB____
  3n   1: BBBBBBBBBB BBBBBBBBBB BBBB!BBBBB BBBBBBBBBB BBBBBBBBBB AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
  3n101: AAAAAAAAAA !AAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAL LLLLLLLJJJ JJJJ__
  4n   1: RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR
  4n101: RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRR__
  5n   1: .RRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR
  5n101: RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRR__
  6n   1: ########## ########## ########## ########## ########## ########## ########## ########## ########## ##########
  6n101: ########## ########## ########## ########## ########## ########## ########## ########## ########## ####__
  7n   1: ########## ########## ########## ########## #RRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR
  7n101: RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RR.RRRRRRR RRRRRRRRRR RRRR__
  8n   1: RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR
  8n101: RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRR!RRRR RRRR__
  9n   1: RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR
  9n101: RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RR!RRRRRRR RRRRRRRRRR RRRR__
 10n   1: RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRR.RRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR
 10n101: RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRR__
 11n   1: .R.RRRRRRR RRRRRRRRRR RRRR!RRRRR RRRRRRRRRR R.RRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR
 11n101: RRRRRRRRRR RRRRRRRRRR RRRRRRRRR. RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRR__
 12n   1: RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR
 12n101: RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RR.RRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRRRRRRRR RRRR__
                                                                                   Tue Oct 28 08:02:35 2008
Login: ###!. Director: .. Admin: ....##______
Lustre: %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   Job #(    user) .-nCPU TimeL     Job #(    user) .-nCPU TimeL     Job #(    user) .-nCPU TimeL
   85722(   user1) A-1024 08:28     86473(   user6) H-   64 24:28     86645(   user9) O-   40 34:32
   85723(   user1) B-1024 08:28     86474(   user6) I-   64 24:28     86674(   user2) P-   32 33:25
   86165(   user2) C-   64 31:42     86554(   user7) J-  512 20:28     86808(  user10) Q-   16 04:32
   86171(   user2) D-   64 31:42     86555(   user7) K-  512 20:30     93530(  user11) R-12000 00:43
   86181(   user3) E-   96 36:28     86571(   user8) L-   64 24:28
   86344(   user4) F-  512 36:28     86585(   user8) M-  128 35:58
   86346(   user5) G-  256 36:28     86620(   user2) N-   32 32:32

    Infiniband: % - 2360     10011470 xxxxxx        0/    1 06:05 smd55.msub
  Running Jobs: A - 2063     82049    xxxxxx      64/  256 06:00 SeOF2.MP2.AQ
          Dead: ! - 12       82057  xxxxx          8/   32 00:15 water
      Reserved: * - 0        82336  xxxxxxxx   1024/ 4096 12:00 pNA_ccsd_d-aug-cc-pvtz_static
       Missing: _ - 0        82353  xxxxxxxx    128/  512 06:00 SeOF3.Anion.MP2.AQ
         Drain: . - 15       82888  xxxxxx      256/ 1024 23:45 29017
         In xl: A - 0        82889  xxxxxx      256/ 1024 23:45 29074
      In large: A - 0        82904  xxxxx         8/   32 02:00 w8-a
        In med: A - 0        82905  xxxxx        32/  128 02:00 w8-a
      In small: A - 0        82906  xxxxx        64/  256 02:00 w8-a
       In tiny: A - 0        82907  xxxxx       256/ 1024 02:00 w8-a
       In fail: A - 0        83036  xxxxx       256/ 1024 24:00 uo2p.new.dolg
      In lopri: A - 0        83083  xxxxxxxx    128/  512 08:00 SeF2.Cation.MP2.AQ
          Idle: # - 250      83209  xxxx        256/ 1024 32:00 Large_Tests
```



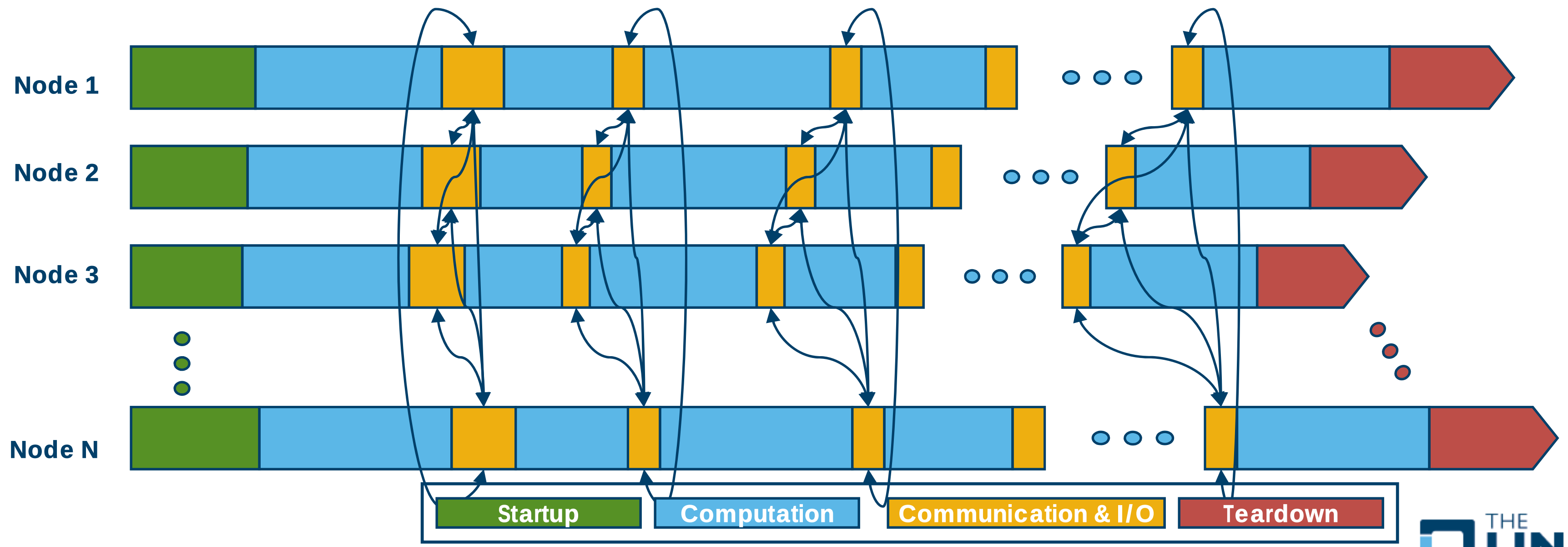
Anatomy of a tightly coupled parallel job



🟩 Startup 🟦 Computation 🟨 Communication & I/O 🟥 Teardown

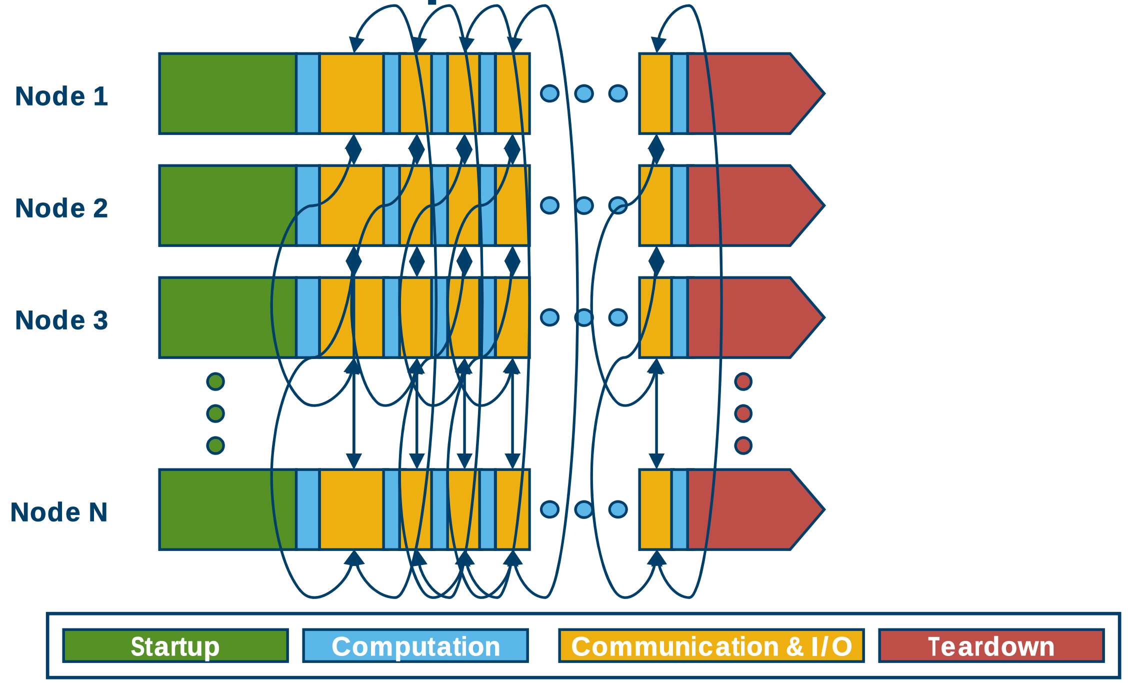
Characterizing the computation and data generation

- **A typical chemistry job:**
  - Starts with a small amount of data
  - Generates hundreds of gigabytes per node during computation
  - Condenses back down to kilobytes or megabytes of results.
- **This requires us to provide large amounts of disk space and disk bandwidth on the nodes**
- **Data have to come to a processor core from many places**
- **We are running tightly coupled computations, so at some point, everybody waits for the slowest component!**



Amdahl Bites!

- Here’s where parallelism breaks down
- Assume we have all the processors we want
- I/O and Communications come to dominate runtime, and you can’t go any faster unless you speed those up
- *That’s* where we need help from the kernel community!



Our data problem

- **We would ideally have memory bandwidth of 2-4 bytes per FLOP/second per processor core**
- **Some systems in the 80's were close to this**
- **Ever since, we have been going the wrong direction!**
  - Core clock speeds have far outstripped RAM speeds
  - Multicore processors make this much worse, since there more cores to feed, but not significantly more memory bandwidth
- **CPU caches help somewhat, but they have drawbacks:**
  - Complicated memory hierarchies
  - Drastically different levels of performance, depending on where data needs to come from.
- **More often than not, data needs to come from off-node, which is relatively slow**

Good things the kernel does for us

- **Well… It works!**
  - That shouldn’t be overlooked, given that it’s a general-purpose kernel
  - It’s certainly no more troublesome than vendor-supplied closed solutions
  - We do like looking “under the covers” and tweaking and tuning.
- **We love the idea of Asynchronous I/O**
- **Kernel RAID helps us very much**
  - We have used mirrored pairs of disks for the OS
  - We use RAID-5 for our scratch filesystems on compute nodes
  - When a disk dies (this happens more than once per day), the MD device carries on in degraded mode, allowing the job to finish
  - XFS is our scratch filesystem, we do a mkfs on it before each job to clean it out.

Keeping up performances

- **We exploit all features of the node as much as we possibly can**
- **Our applications "own" the node for the duration of a job, and they are *greedy***
  - No more than one job is scheduled on a node at a time
  - They allocate all the RAM they can grab right up front and don't give it back until the job is done
  - They may take all the cores (though for reasons of memory bandwidth, that may be inefficient)
- **We map and pin large contiguous regions of RAM for Infiniband RDMA**
- **We frequently pre-calculate integrals and save them to local disk for lookup later**

Pressure on the kernel: Memory

- **There's never enough RAM!**
- **There is lots of competition for memory**
  - OS/kernel
  - Daemons
  - Block I/O buffering & caching
  - Infiniband Queue Pairs
  - Application-shared memory segments
  - Mapped/pinned memory for Infiniband RDMA.
- **We frequently have problems due to memory pressure**
  - "It makes me really happy when linux denies malloc requests when it has 19GB of cached data still (note this is with overcommit turned off)."

Pressure on the “kernel”: I/O

- **A Lustre example**
  - Lustre prior to 1.8 (by design) bypasses the kernel buffer cache on its storage servers
  - This means if all the nodes in a parallel job need to copy a file, **every** node reads **every** block of that file from disk!
  - We’ve seen 400 MB/sec on a single disk volume (good)
  - The bad part was that the volume was asked to do this (600 nodes all reading every block of a 70-MB file because of poor caching).

Pressure on the kernel: OOM

- **OOM conditions are too common and painful**
  - Our experience is that the OOM-killer makes bad decisions
  - Currently, if we have OOM activity on the node, we mark it untrustworthy and don't use it until it's rebooted
  - Can we tag user processes with a "kill me first" flag?
- **We have seen behavior that makes it look like any paging activity on a highly committed node causes random-looking crashes**
- **The overcommit sysctls don't seem to be doing us any favors, and we would like to understand them better**

Appealing new technologies: Huge pages

- **Good use of the Translation Lookaside Buffer (TLB) is vitally important to us**
  - TLB misses cause 2,000 – 3,000 cycles to be wasted on our AMD “Barcelona” processors
  - The TLB has a fixed number of entries, but using larger pages lets us map *much* more memory and avoid misses
  - Fortunately, the processors support large pages (2MB, 4MB, 1GB)
  - The hugepages feature in the kernel enables large page support.
- **We have written test programs that show 4x - 7x speedup in matrix multiply operations on random memory locations with 2MB page size (vs. the default 4 KB size)**
  - This looks like a huge win, but it is cumbersome for our users to have to change their code, guess how many hugepages to map, and run root-level commands to set them up for each job they run
  - On the IA64, page size was a kernel compile-time setting. While this was not very flexible, it was simple to deal with.
  - A compile-time or boot-time setting for default large page sizes sounds very appealing to us.

Appealing new technologies: GPU

- **Floating point operations can be offloaded to modern GPUs**
  - The GPUs have *lots* of lightweight processor cores optimized for floating point math
  - If these are fast enough, maybe we can quit using local disk in all our nodes.
- **High points**
  - There's a lot of floating point performance on tap!
  - They're relatively cheap and plentiful.
- **The key challenges are:**
  - They can be power hungry
  - Does your algorithm lend itself to the parallelism these devices are good at?
  - Is there enough bandwidth to RAM to keep these from starving for data?
- **Chemistry papers are starting to come out now citing speedups of up to 10^2 if the algorithm is adapted to GPU!**

Appealing new technologies: Bus-resident SSD devices

- **Using Solid State Disk (SSD) devices (e.g. Fusion-IO) as some kind of cache**
- **They should be blindingly fast (compared to rotating disk) if:**
  - They reside in a bus slot, *not* on a disk controller
  - They aren’t hampered by elevators or schedulers that assume they have sector layouts and rotational latency.
- **How would the kernel support these?**
  - sd block I/O?
  - Shmem()?
  - Mmap()?
  - Other?

Appealing new technologies: POSIX filesystem extensions for HPC

- **Shared filesystems on large clusters frequently bottleneck on metadata operations**
  - This gets even worse on striped parallel filesystems
  - For example, stat() calls may have to talk to multiple servers to get the latest [c,a,m]time, file size, etc.
  - That gets very slow if there's a lot of contention.
- **A lot of time can be saved if a quicker, less-accurate result is "good enough"**
- **The proposed HEC POSIX I/O API Extensions (http://www.pdl.cmu.edu/posix) implement I/O calls that allow relaxed semantics, or inform the filesystem about access patterns to improve performance**
  - readx(), writex() de-serialize vector I/O
  - statlite(), readdirplus() allow faster, less accurate stat() operations
  - "Lazy" I/O (O_LAZY flag, lazyio_propagate(), lazyio_synchronize()) relaxes coherency restraints.
- **This is a proposed solution, perhaps controversial, but it's an option we'd like to have!**

THE LINUX FOUNDATION

Takeaway messages

- Our applications have an insatiable demand for cycles
- Memory bandwidth is crucial to us
- Parallelization (probably on many levels) is the only way to get where we want to go
- The kernel is the “gatekeeper” to high performance, since it mediates the I/O and communications that hold us back
- If we can use other features in the node to save compute time, we will use them to the fullest
- Making huge pages easier to use should help our application performance tremendously
- GPUs and SSDs look very promising to us, provided they are not hamstrung by antiquated assumptions

Questions?

David Cowley

Pacific Northwest National Laboratory

david.cowley@pnl.gov

This research was performed using EMSL, a national scientific user facility sponsored by the Department of Energy's Office of Biological and Environmental Research and located at Pacific Northwest National Laboratory.