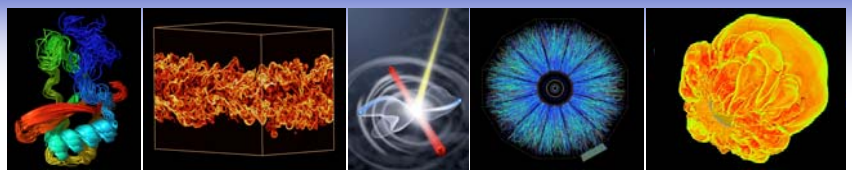


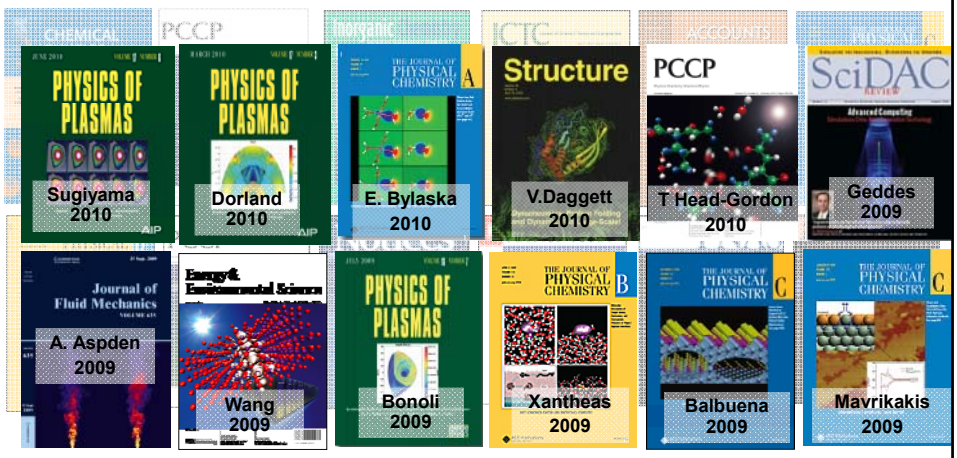
Bringing Users Along the Road to Billion Way Concurrency

Kathy Yelick

NERSC Director, Lawrence Berkeley National Laboratory

EECS Department, UC Berkeley

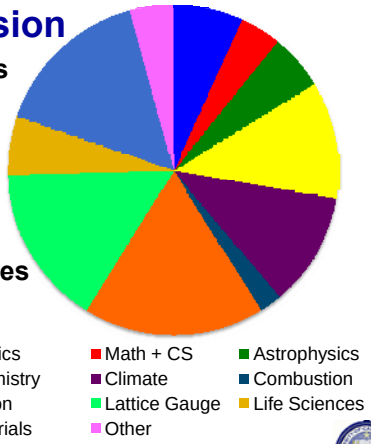
Cover Stories from NERSC Research



NERSC is enabling new high quality science across disciplines, with over *1,600 refereed publications* last year

NERSC is the Primary Computing Center for DOE Office of Science

- **NERSC serves a large population**
  Over 3000 users, 400 projects, 500 codes
- **NERSC Serves DOE SC Mission**
  - **Allocated by DOE program managers**
  - **Not limited to largest scale jobs**
  - **Not open to non-DOE applications**
- **Strategy: Science First**
  - **Requirements workshops by office**
  - **Procurements based on science codes**
  - **Partnerships with vendors to meet science requirements**



3

NERSC Systems for Science

Large-Scale Computing Systems

Franklin (NERSC-5): Cray XT4
- 38,128 cores (quad core), ~25 Tflop/s on applications; 356 Tflop/s peak

Hopper (NERSC-6): Cray XE6

- Phase 1: Cray XT5, 668 nodes, 5344 cores
- Phase 2: > 1 Pflop/s peak (late 2010), 24-core nodes

Clusters

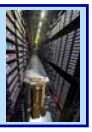
105 Tflops combined

Carver
- IBM iDataplex cluster

PDSF (HEP/NP)
- Linux cluster (~1K cores)

Magellan Cloud testbed
- IBM iDataplex cluster

NERSC Global Filesystem (NGF)



- Uses IBM's GPFS
- 1.5 PB; 5.5 GB/s

HPSS Archival Storage

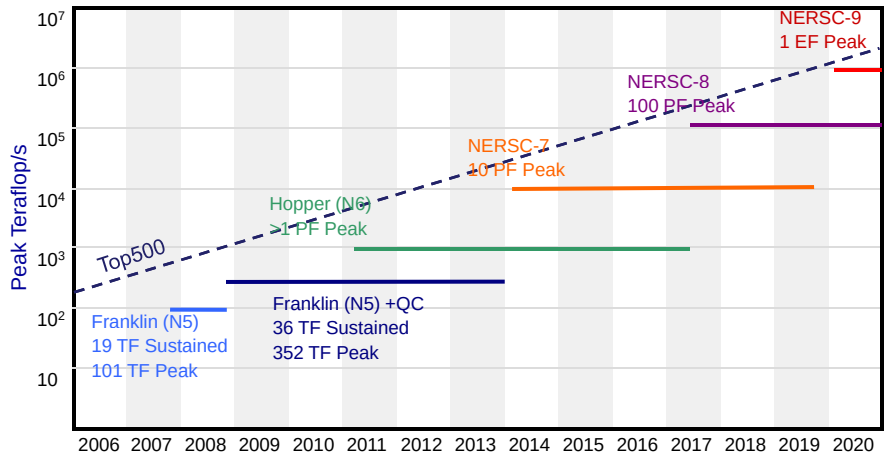
- 40 PB capacity
- 4 Tape libraries
- 150 TB disk cache

Analytics

- Euclid (512 GB shared memory)
- Dirac GPU testbed (48 nodes. Fermi)

4

NERSC Roadmap



Users expect 10x improvement in capability every 3-4 years

5

Challenges to Exascale

- **System power** is the primary constraint for exascale (MW ~= $M)
- **Concurrency** (1000x today) driven by system power and density
- M**emory** bandwidth and capacity are under pressure
- **Processor** architecture is an open question, but heterogeneity is likely
- **Algorithms** need to be designed to minimize data movement, not flops
- **Programming model** memory & concurrency → new chip-level model
- **Reliability and resiliency** will be critical at this scale
- **I/O bandwidth** unlikely to keep pace with machine speed

Why are these NERSC problems?

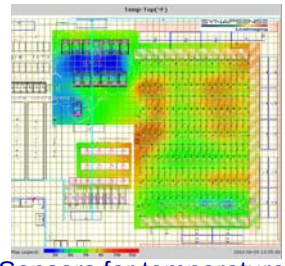
- **All** are challenges for 100 "capacity" 100PF machines, except:
  - System wide outages and bisection bandwidth
- NERSC needs to guide the transition of its user community
- NERSC represents broad HPC market better than any other DOE center
- Hardware revolutions affect procurement strategies (Giga->Tera)

NERSC Response to Exascale

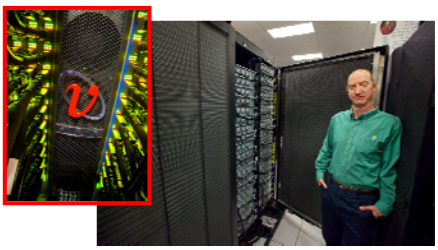
- Vendor partnerships
- NERSC workload and procurements
- Testbeds & training
- Anticipating and changing the future
  - Programming models
  - Autotuning
  - Algorithms
  - Co-Design

7

Energy Efficiency Partnerships with Synapsense and IBM



600 Sensors for temperature, etc.



Rear door heat exchangers

- Monitoring for energy efficiency (and reliability!)
- Liquid cooling on IBM system uses return water from another system, with modified CDU design
  - Reduces cooling costs to as much as ½
  - Reduces floor space requirements by 30%

Air is colder coming out than going in!

8

Center of Excellence with Cray

Multicore Era: Massive on-chip concurrency necessary for reasonable power use

What should we tell NERSC users to do ?

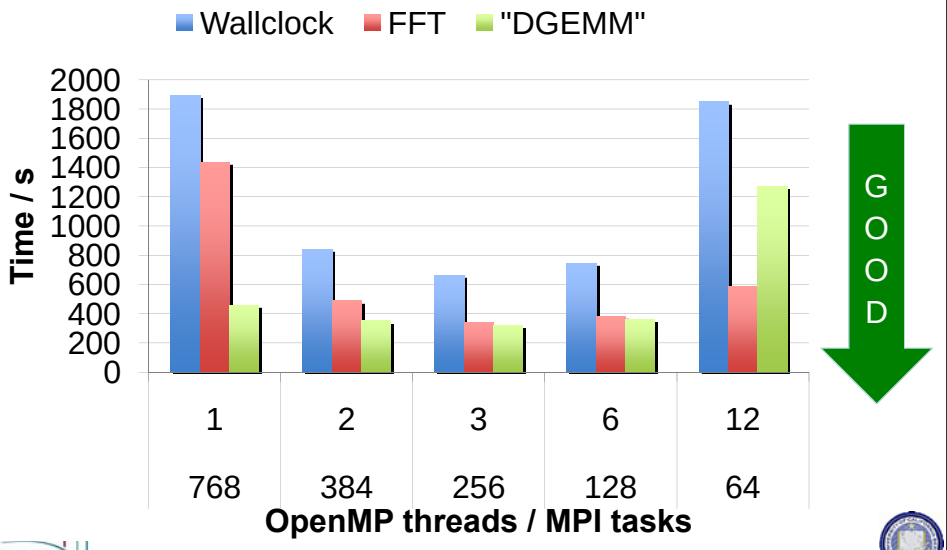
NERSC/Cray "Programming Models Center of Excellence" combines:

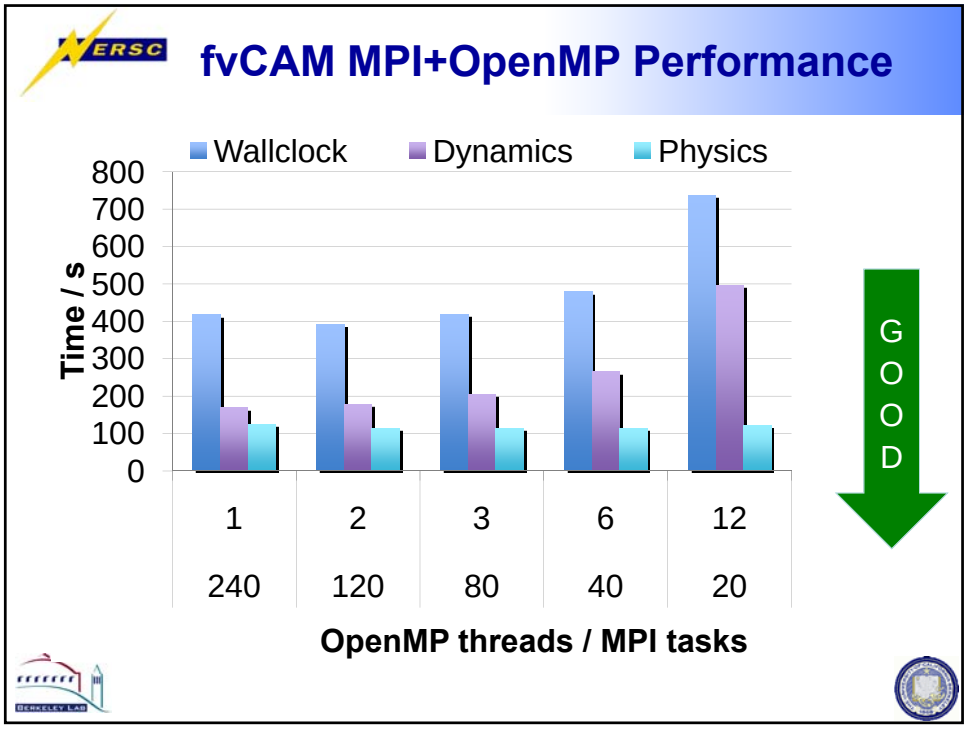
- **Berkeley Lab strength in advanced programming models, multicore tuning, and application benchmarking**
- **Cray strength in advanced programming models, optimizing compilers, and benchmarking**

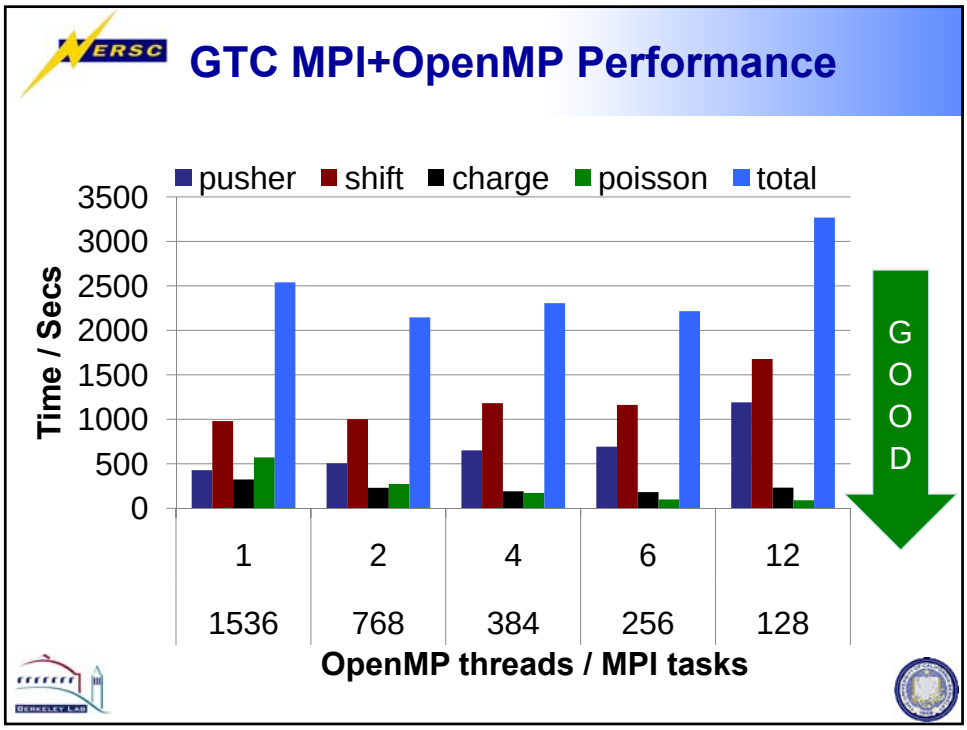
Immediate question: What is the best way to use Hopper node?

- **Flat MPI - Today's preferred mode of operation**
- **MPI + OpenMP, MPI + pthreads, MPI + PGAS, MPI+OpenCL, PGAS…**

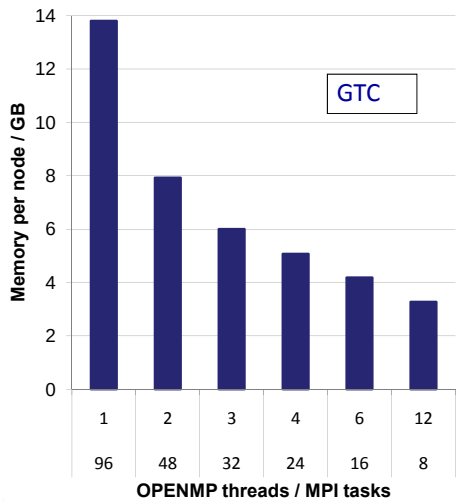
Paratec MPI+OpenMP Performance

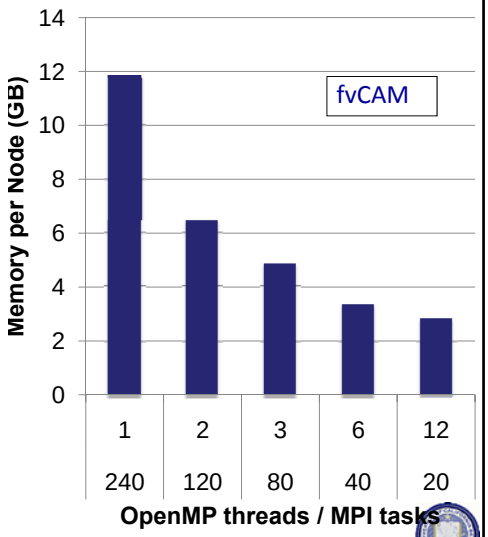



fvCAM MPI+OpenMP Performance
Wallclock
Dynamics
Physics
Time / s
800
700
600
500
400
300
200
100
0
1
2
3
6
12
240
120
80
40
20
OpenMP threads / MPI tasks
GOOD


GTC MPI+OpenMP Performance
pusher
shift
charge
poisson
total
Time / Secs
3500
3000
2500
2000
1500
1000
500
0
1
2
4
6
12
1536
768
384
256
128
OpenMP threads / MPI tasks
GOOD

Less Memory Usage with OpenMP Compared to Flat MPI





Current Procurement Strategy

Use Science Applications to Measure System Performance!

NERSC-6 "Sustained System Performance (SSP)" Benchmarks

CAM Climate	*GAMESS* Quantum Chemistry	*GTC* Fusion	*IMPACT-T* Accelerator Physics	*MAESTRO* Astro-physics	*MILC* Nuclear Physics	*PARATEC* Material Science

Peak FLOPs offer little insight into delivered application performance
SSP applications model delivered performance of REAL workload

"Benchmarks are only useful insofar as they model the intended workload"

Ingrid Bucher (LANL 1978)

14

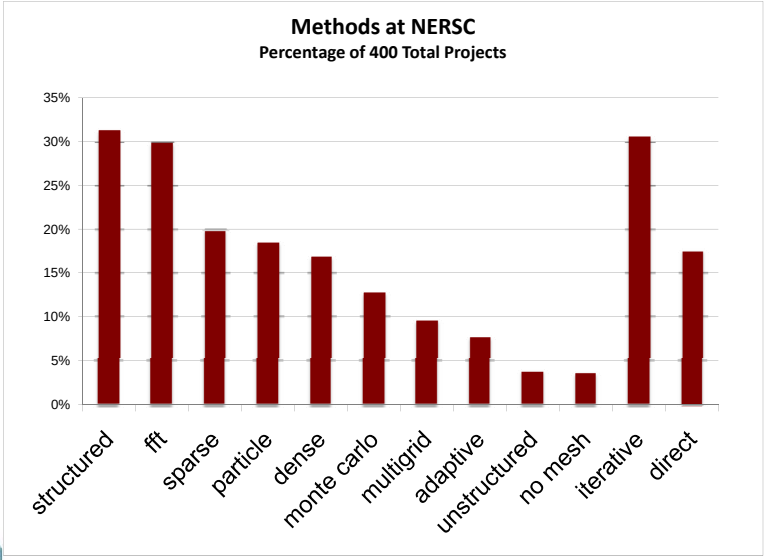
Algorithm Diversity

Science areas	Dense linear algebra	Sparse linear algebra	Spectral Methods (FFT)s	Particle Methods	Structured Grids	Unstructured or AMR Grids
Accelerator Science		X	X	X	X	X
Astrophysics	X	X	X	X	X	X
Chemistry	X	X	X	X		
Climate			X		X	X
Combustion					X	X
Fusion	X	X		X	X	X
Lattice Gauge		X	X	X	X	
Material Science	X		X	X	X	

NERSC users require a system which performs adequately in all areas

Numerical Methods at NERSC

(Caveat: survey data from ERCAP requests)



16

Revolutions Require a New Strategy

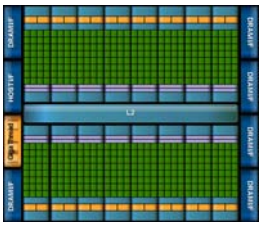
- **Preserve Application Performance as goal**
  - But, allow significant optimizations
  - Produced optimized versions of some codes
  - Understand performance early
- **E.g., for Fermi-based GPU system**
  - Fermi is nearly as expensive as host node
  - 48 nodes w/Fermi or 2x more nodes without
  - Minimum: Fermi must be 2x faster than host
- **Estimating value of acceleration for SSP**
  - Estimate "best possible performance"
  - Successive refinement of estimates
  - Stop if estimate < min threshold (2x host)

17

Dirac Testbed at NERSC

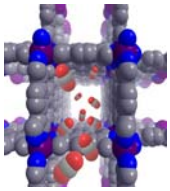
- **Dirac is a 48 nodes GPU cluster**
- **44 Fermi nodes**
  - 3GB memory per card
  - @~144GB/s bandwidth + ECC
  - ~1TF peak single precision and 500 GF/s DP
  - 24 GB of memory per node
- **4 Tesla nodes**
- **QDR Infiniband network**

Paul Dirac, Nobel prize-winning Theoretical Physicist

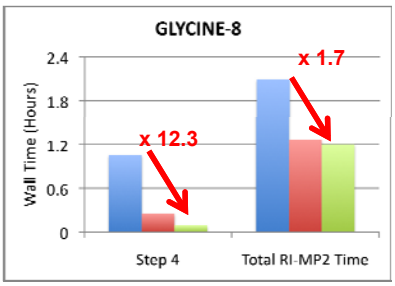


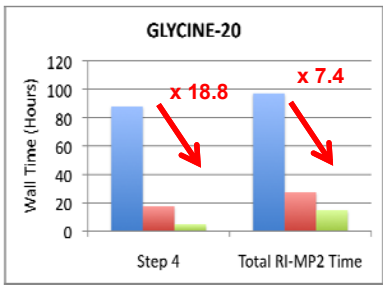
Ri-MP2: Molecular Equilibrium Structure

- **Goal: Optimize Q-Chem RI-MP2 Routine**
  - Start with "step 4" kernel, which dominates the running time
- **Observations**
  - Impressive single node speedups
  - Performance results highly dependent on molecular structure



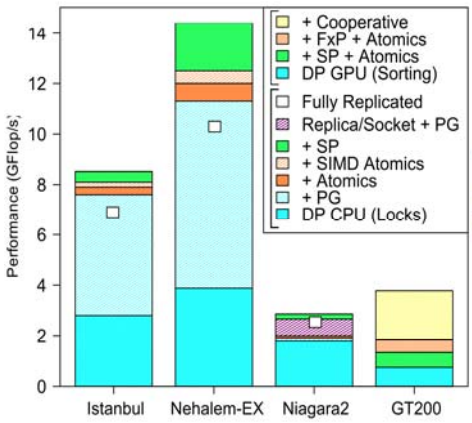
Fermi GPU Racks - NERSC

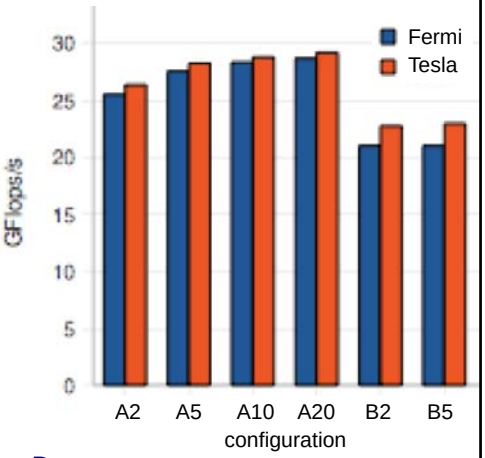




Jihan Kim (SciDAC-E NERSC Postdoc)

GTC (Fusion) Charge Deposition Performance

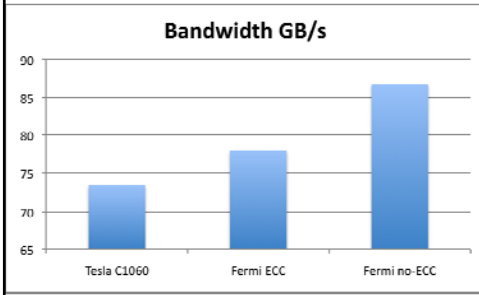


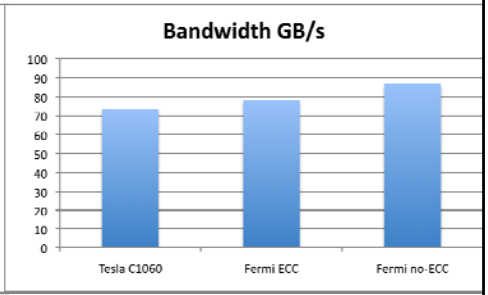


Optimizing GTC (Fusion code) Charge Deposition for Multicore and GPUs

K. Ibrahim, K. Madduri, L. Oliker, S. Williams, E.-J. Im, E. Strohmaier, S. Ethier, J. Shalf

Fermi Memory Bandwidth





Anticipating and Influencing the Future

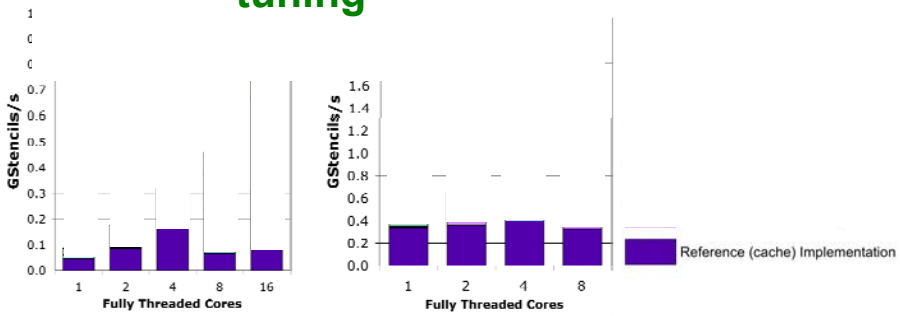
22

Autotuning: Write Code Generators for Nodes

Nearest-neighbor 7point stencil on a 3D array

Use Autotuning!

Write code generators and let computers do tuning



23

Example pattern-specific compiler: Structured grid in Ruby

- **Ruby class encapsulates SG pattern**
  - **body of anonymous lambda specifies filter function**
- **Code generator produces OpenMP for multicore x86**
  - **~1000-2000x faster than Ruby**
  - **Minimal per-call runtime overhead**

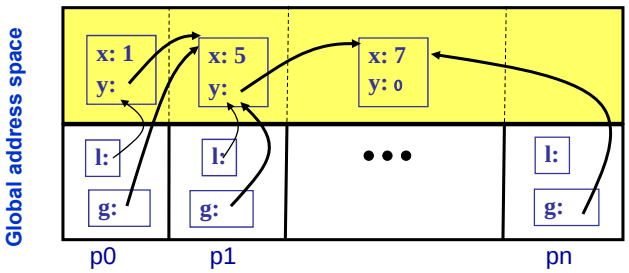
```
class LaplacianKernel < Kernel
 def kernel(in_grid, out_grid)
  in_grid.each_interior do |point|
   in_grid.neighbors(point,1).each
     do |x|
     out_grid[point] += 0.2*x.val
   end
 end
end
```

```
VALUE kern_par(int argc, VALUE* argv, VALUE
self) {
unpack_arrays into in_grid and out_grid;

#pragma omp parallel for default(shared)
private (t_6,t_7,t_8)
for (t_8=1; t_8<256-1; t_8++) {
 for (t_7=1; t_7<256-1; t_7++) {
  for (t_6=1; t_6<256-1; t_6++) {
   int center = INDEX(t_6,t_7,t_8);
   out_grid[center] = (out_grid[center]
      +(0.2*in_grid[INDEX(t_6-
1,t_7,t_8)]));
   ...
   out_grid[center] = (out_grid[center]
   +(0.2*in_grid[INDEX(t_6,t_7,t_8+1)]));
```

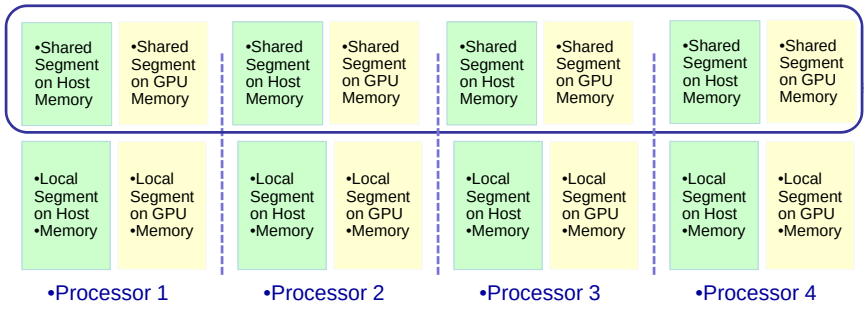
PGAS Languages: Why use 2 Languages (MPI+X) when 1 will do?

- *Global address space:* thread may directly read/write remote data
- *Partitioned:* data is designated as local or global



- Remote put and get: never have to say "receive"
- No less scalable than MPI!
- Permits sharing, whereas MPI rules it out!
- Gives affinity control, useful on shared and distributed memory

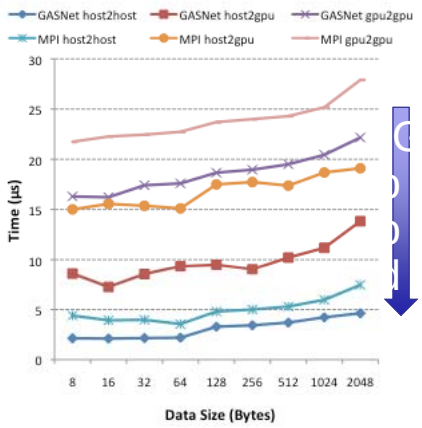
Hybrid Partitioned Global Address Space



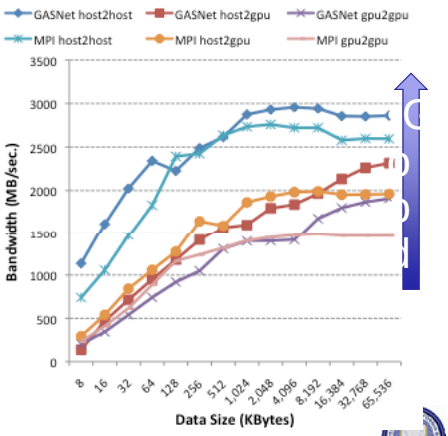
- Each thread has only two shared segments
- Decouple the memory model from execution models; one thread per CPU, vs. one thread for all CPU and GPU "cores"
- Caveat: type system and therefore interfaces blow up with different parts of address space

GASNet GPU Extension Performance

Latency

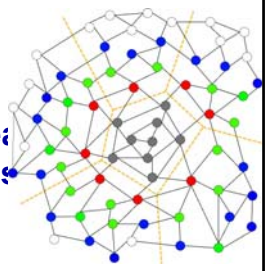


Bandwidth



Communication-Avoiding Algorithms

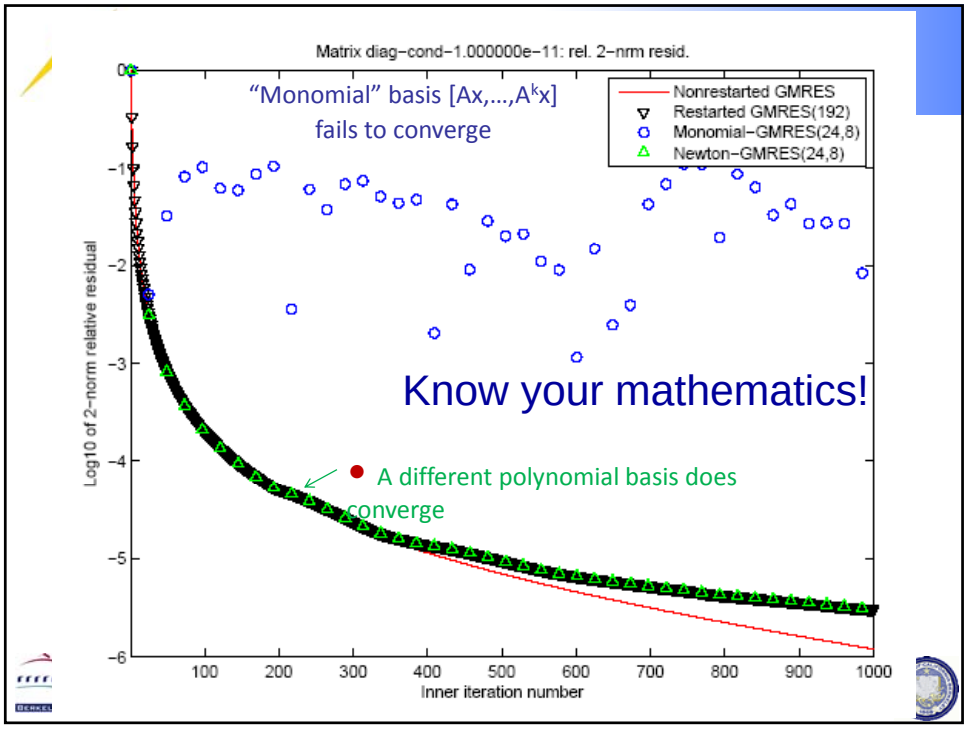
- Consider Sparse Iterative Methods
  - Nearest neighbor communication on a mesh
  - Dominated by time to read matrix (edges) from DRAM
  - And (small) communication and global synchronization events at each step

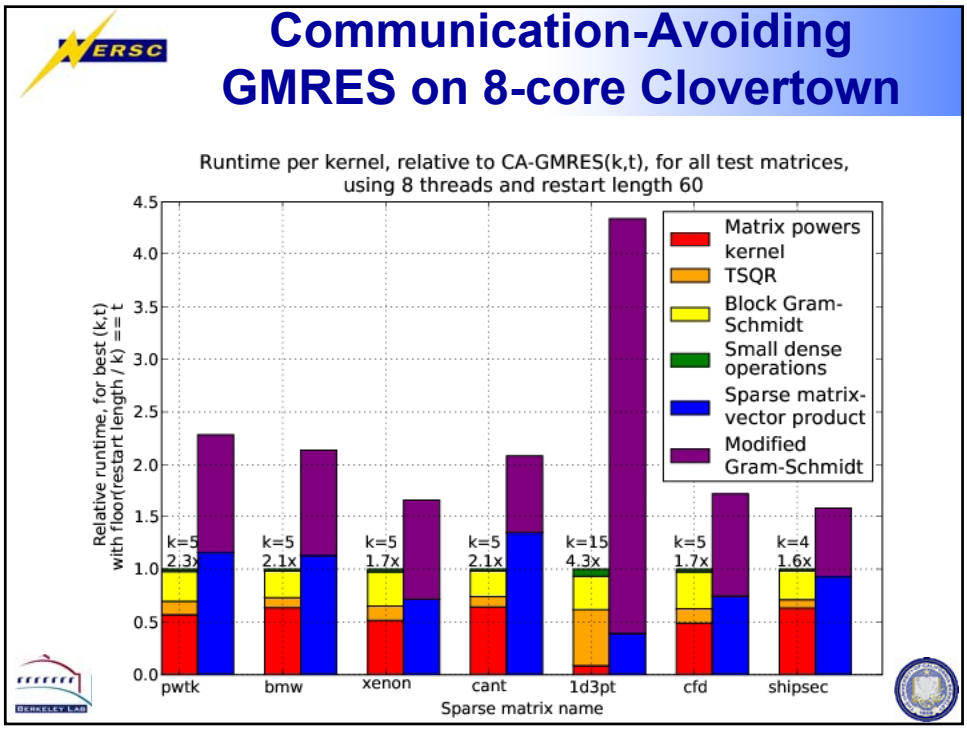


Can we lower data movement costs?

- Take *k* steps "at once" with one matrix rea from DRAM and one communication phas
  - Parallel implementation
    O(log p) messages vs. O(k log p)
  - Serial implementation
    O(1) moves of data moves vs. O(k)

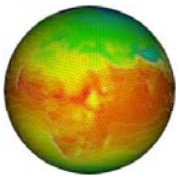
Joint work with Jim Demmel, Mark Hoemman, Marghoob Mohiyuddin

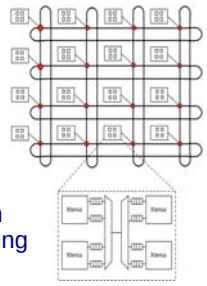

Matrix diag−cond−1.000000e−11: rel. 2−nrm resid.
"Monomial" basis [Ax,...,A^k x]
fails to converge
Nonrestarted GMRES
Restarted GMRES(192)
Monomial−GMRES(24,8)
Newton−GMRES(24,8)
Know your mathematics!
A different polynomial basis does converge
Log10 of 2−norm relative residual
Inner iteration number


Communication-Avoiding GMRES on 8-core Clovertown
Runtime per kernel, relative to CA-GMRES(k,t), for all test matrices, using 8 threads and restart length 60
Matrix powers kernel
TSQR
Block Gram-Schmidt
Small dense operations
Sparse matrix-vector product
Modified Gram-Schmidt
k=5 2.3x
k=5 2.1x
k=5 1.7x
k=5 2.1x
k=15 4.3x
k=5 1.7x
k=4 1.6x
pwtk
bmw
xenon
cant
1d3pt
cfd
shipsec
Sparse matrix name
Relative runtime, for best (k,t) with floor(restart length / k) == t

Co-Design Before its Time

- Demonstrated during SC '09
- CSU atmospheric model ported to low-power core design
  - Dual Core Tensilica processors running atmospheric model at 25MHz
  - MPI Routines ported to custom Tensilica Interconnect
- Memory and processor Stats available for performance analysis
- Emulation performance advantage
  - 250x Speedup over merely function software simulator
- *Actual code running - not representative benchmark*





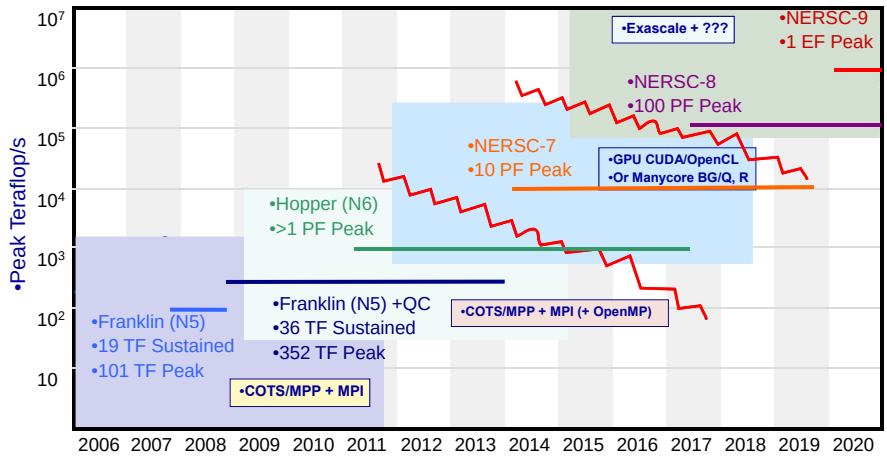
Icosahedral mesh for algorithm scaling

General Lessons

- Early intervention with hardware designs
- Optimize for what is important:

  energy → data movement
- Anticipating and changing the future
  - Programming models
  - Autotuning
  - Algorithms
  - Co-Design

NERSC Aggressive Roadmap



•Users expect 10x improvement in capability every 3-4 years

33

ICCS
International Center for Computational Science

•Three continents, three institutions

•UC Berkeley/LBNL/NERSC
•University of Heidelberg and
•National Astronomical Observatories (CAS)

•Horst Simon
•Hemant Shukla
•John Shalf
•Rainer Spurzem

•ICCS Projects

•ISAA

•Infrastructure for Astrophysics Applications Computing

•ISAAC is a three-year (2010-2013) NSF funded project to focus on research and development of infrastructure for accelerating physics and astronomy applications using and multicore architectures.

•Goal is to successfully harness the power of the parallel architectures for compute-intensive scientific problems and open doors for new discovery and revolutionize the growth of science via, **Simulations**, **Instrumentations** and **Data processing /analysis**

•GRACE II

•SILK ROAD

•ICCS Activities

•**Summer School** Aug 2-6, 2010

•Proven Algorithmic Techniques for Many-core Processors

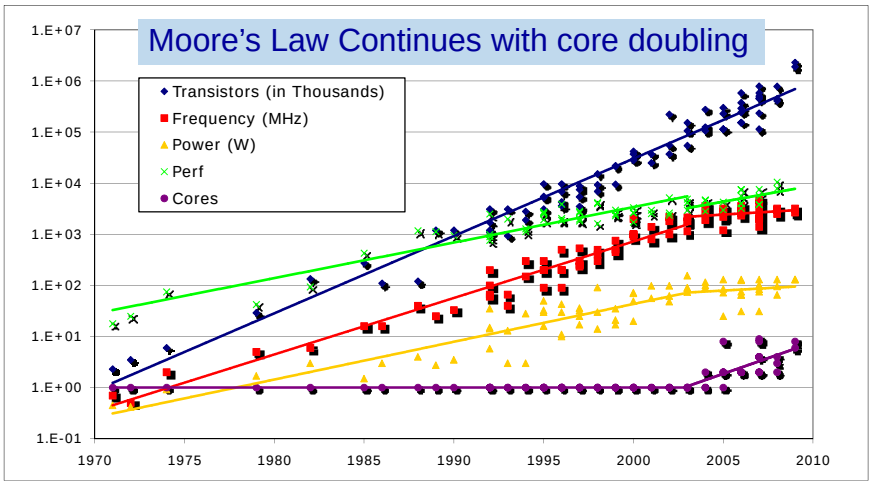
•Wen-Mei Hwu (UIUC) and David Kirk (NVIDIA)

•**Workshop** Nov 30 – Dec 2, 2009

•Many-core and Accelerator-based Computing for Physics and Astronomy Applications

•Visit us – http://iccs.lbl.gov

Performance Has Also Slowed, Along with Power

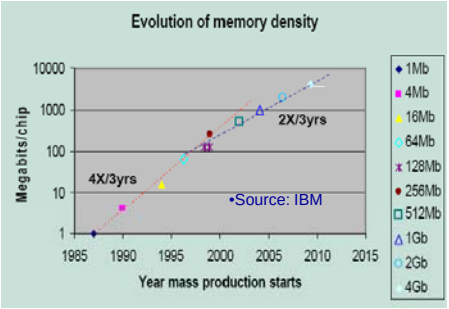


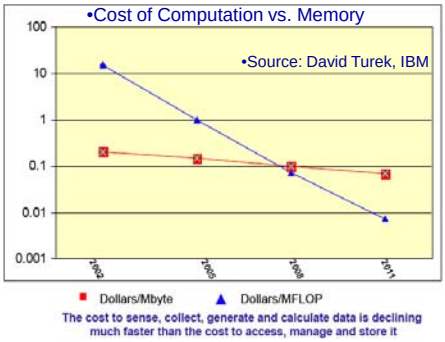
•Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanoviç

35

Memory is Not Keeping Pace

- **Technology trends against a constant or increasing memory per core**
- Memory density is doubling every three years; processor logic is every two
- Storage costs (dollars/Mbyte) are dropping gradually compared to logic costs





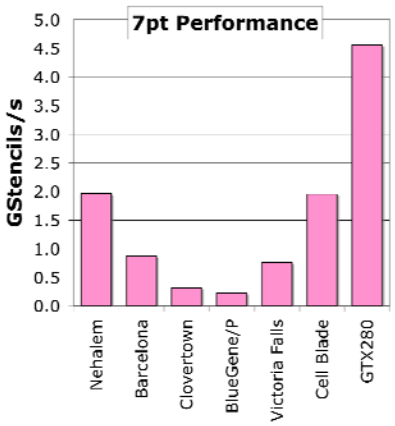
- Question: *Can you double concurrency without doubling memory?*

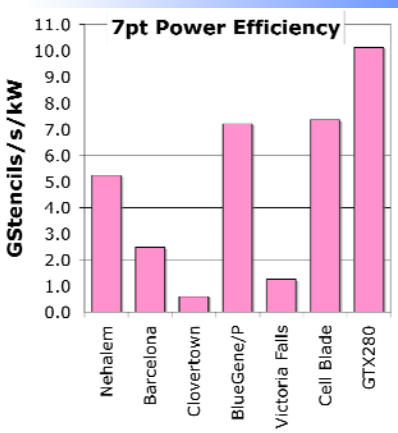
36

How to make use of 100,000 (or more!) cores?

37

7 Point Stencil Revisited





- Cell and GTX280 are notable for both performance and energy efficiency

•Joint work with Kaushik Datta, Jonathan Carter, Shoaib Kamil, Lenny Oliker, John Shalf, and Sam Williams

38

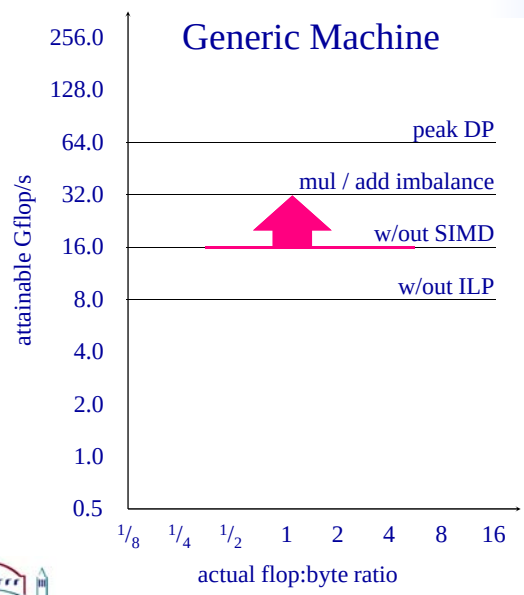
#2: Understand your machine limits

The "roofline" model

S. Williams, D. Patterson, L. Oliker, J. Shalf, K. Yelick

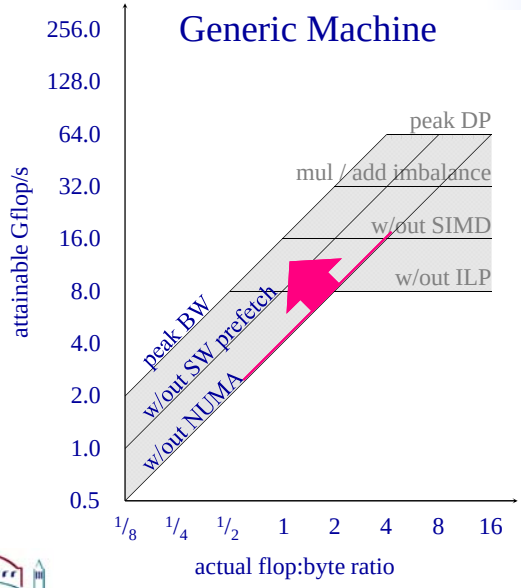
39

The Roofline Performance Model



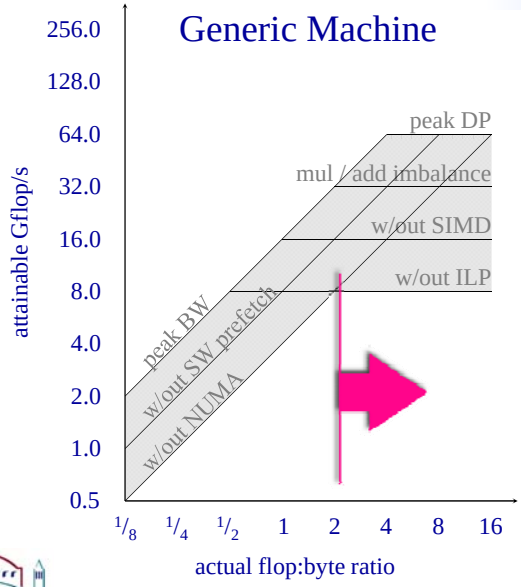
- **The top of the roof is determined by peak computation rate (Double Precision floating point, DP for these algorithms)**
- **The instruction mix, lack of SIMD operations, ILP or failure to use other features of peak will lower attainable**

The Roofline Performance Model



- The sloped part of the roof is determined by peak DRAM bandwidth (STREAM)
- Lack of proper prefetch, ignoring NUMA, or other things will reduce attainable bandwidth

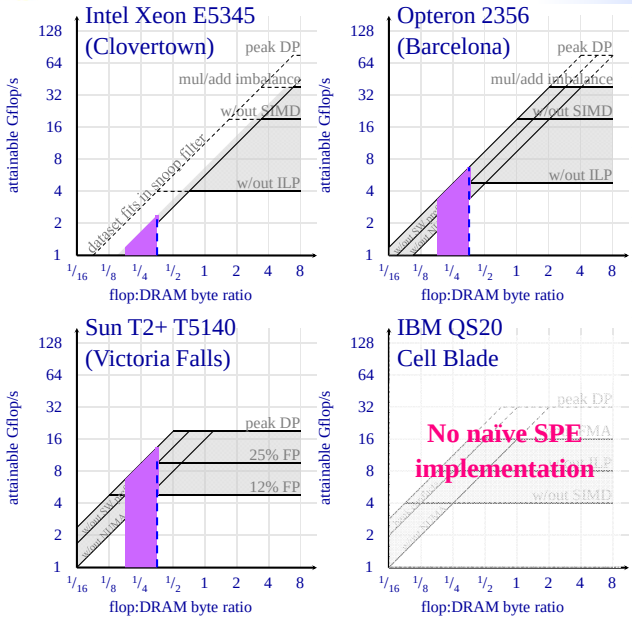
The Roofline Performance Model



- Locations of posts in the building are determined by algorithmic intensity
- Will vary across algorithms and with bandwidth-reducing optimizations, such as better cache re-use (tiling), compression techniques

Roofline model for Stencil

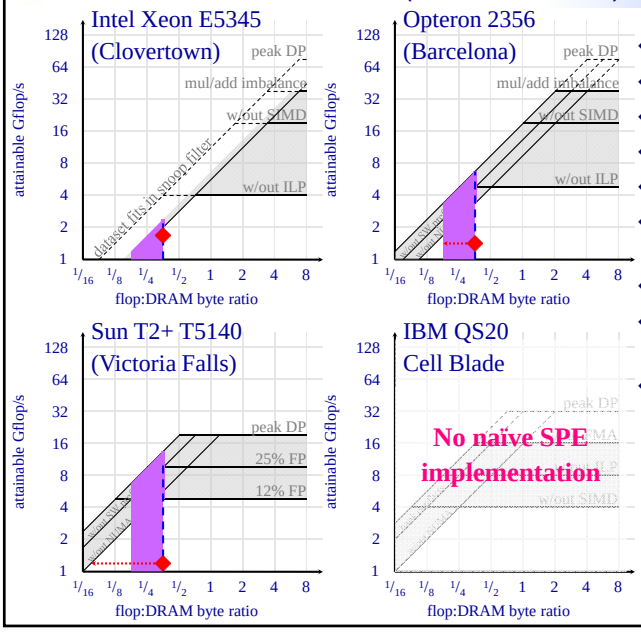
(out-of-the-box code)



- Large datasets
- 2 unit stride streams
- No NUMA
- Little ILP
- No DLP
- Far more adds than multiplies (imbalance)
- Ideal flop:byte ratio $^1/_3$
- High locality requirements
- Capacity and conflict misses will severely impair flop:byte ratio

Roofline model for Stencil

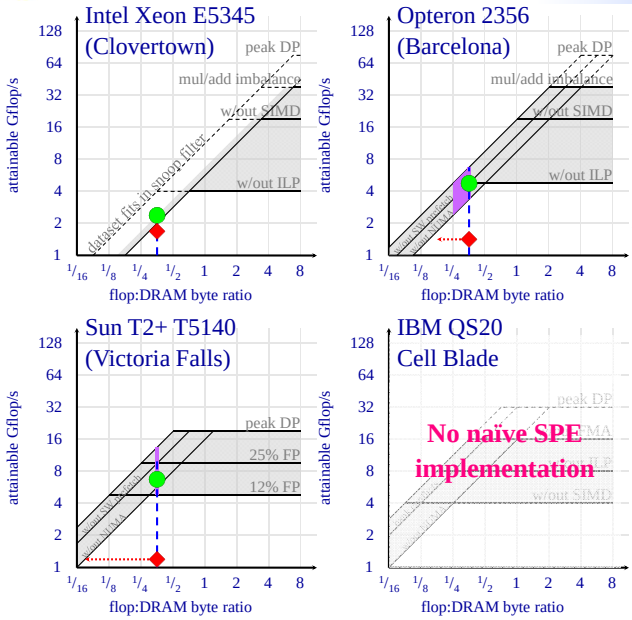
(out-of-the-box code)



- Large datasets
- 2 unit stride streams
- No NUMA
- Little ILP
- No DLP
- Far more adds than multiplies (imbalance)
- Ideal flop:byte ratio $^1/_3$
- High locality requirements
- Capacity and conflict misses will severely impair flop:byte ratio

Roofline model for Stencil

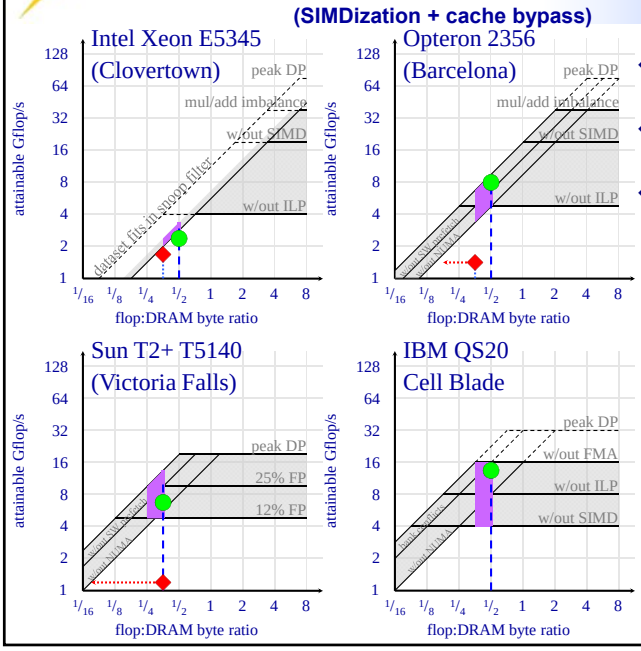
(NUMA, cache blocking, unrolling, prefetch, …)



- Cache blocking helps ensure flop:byte ratio is as close as possible to $^1/_3$
- Clovertown has huge caches but is pinned to lower BW ceiling
- Cache management is essential when capacity/thread is low

Roofline model for Stencil

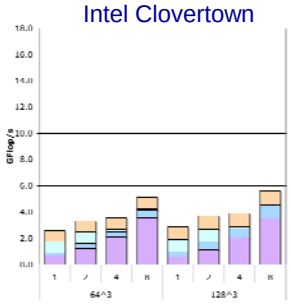
(SIMDization + cache bypass)



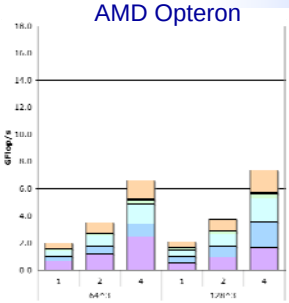
- Make SIMDization explicit
- Use cache bypass instruction: ***movntpd***
- Increases flop:byte ratio to ~0.5 on x86/Cell

#3) Write Code Generators Rather Than Code (LBMHD)

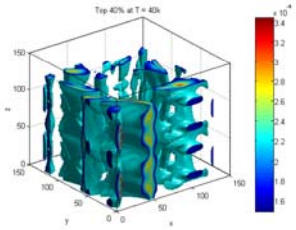
Intel Clovertown



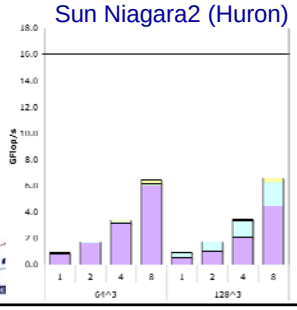
AMD Opteron



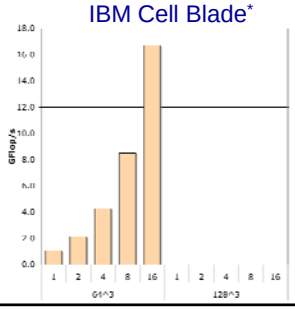
LBMHD is not always bandwidth limited: used SIMD, etc.



Sun Niagara2 (Huron)



IBM Cell Blade*

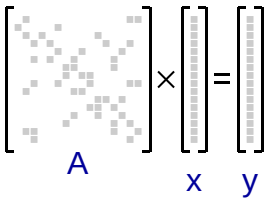


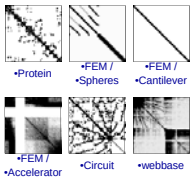
- +SIMDization
- +SW Prefetching
- +Unrolling
- +Vectorization
- +Padding
- Naïve+NUMA

•Joint work with Sam Williams, Lenny Oliker, John Shalf, and Jonathan Carter

#4) Used Optimized Librarites: (Sparse Matrix Vector Multiplication)

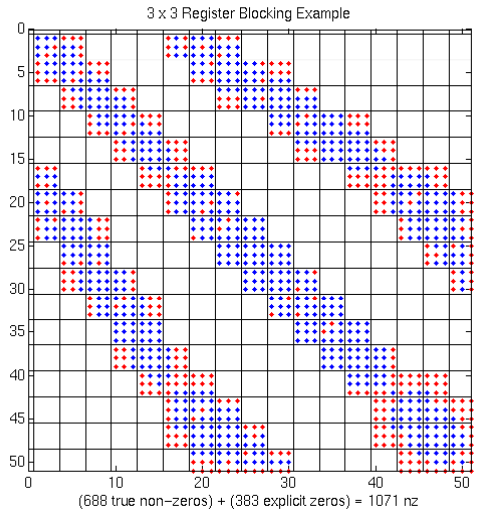
- **Sparse Matrix**
  - **Most entries are 0.0**
  - **Performance advantage in only storing/operating on the nonzeros**
  - **Requires significant meta data**
- **Evaluate y=Ax**
  - **A is a sparse matrix**
  - **x & y are dense vectors**
- **Challenges**
  - **Difficult to exploit ILP(bad for superscalar),**
  - **Difficult to exploit DLP(bad for SIMD)**
  - **Irregular memory access to source vector**
  - **Difficult to load balance**
  - **Very low computational intensity (often >6 bytes/flop) = likely memory bound**





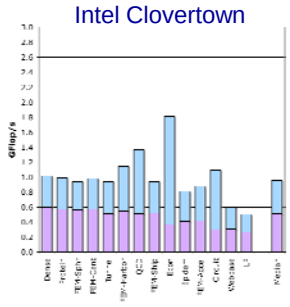
48

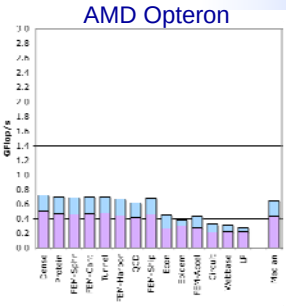
Extra Work Can Improve Efficiency!



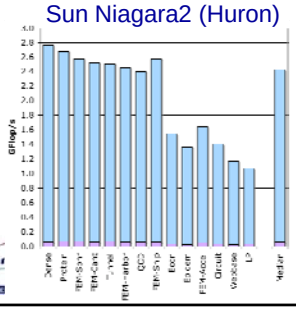
- **Example: 3x3 blocking**
  - **Logical grid of 3x3 cells**
  - **Fill-in explicit zeros**
  - **Use only 1 index per block, rather than per nonzero**
  - **Unroll 3x3 block multiplies**
  - **"Fill ratio" = 1.5**
- **On Pentium III: 1.5x speedup!**
  - **(Actual mflop rate = 2.25 higher)**

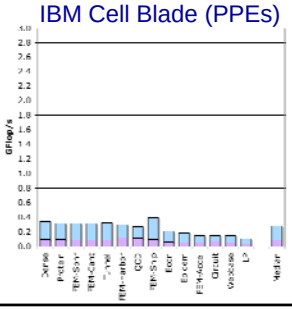
Naïve Parallel Implementation





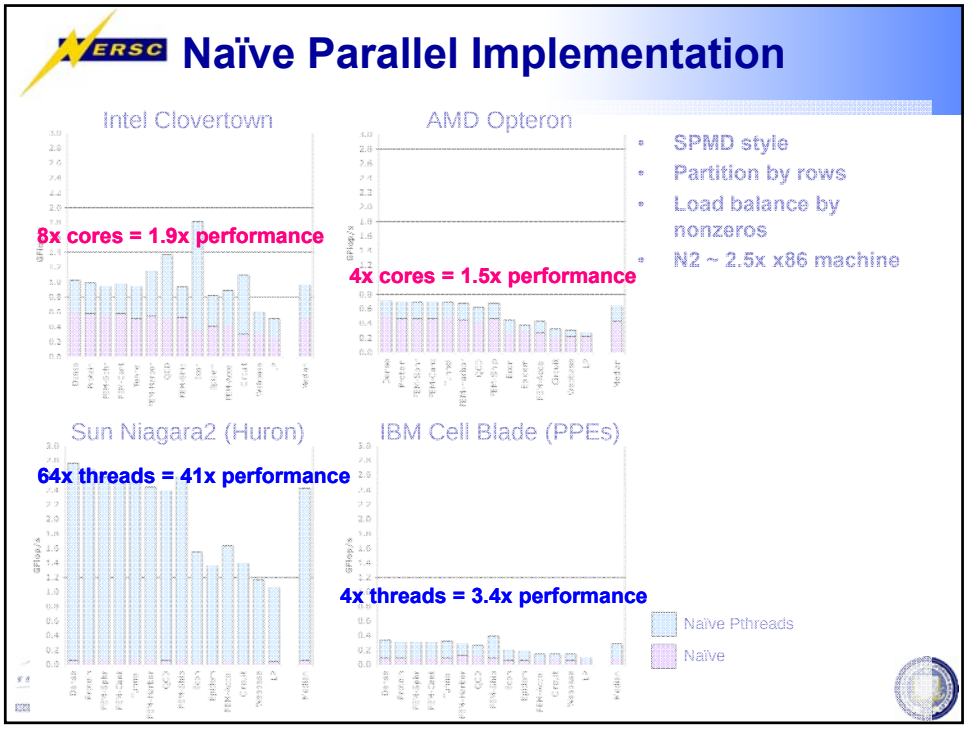
- **SPMD style**
- **Partition by rows**
- **Load balance by nonzeros**
- **N2 ~ 2.5x x86 machine**

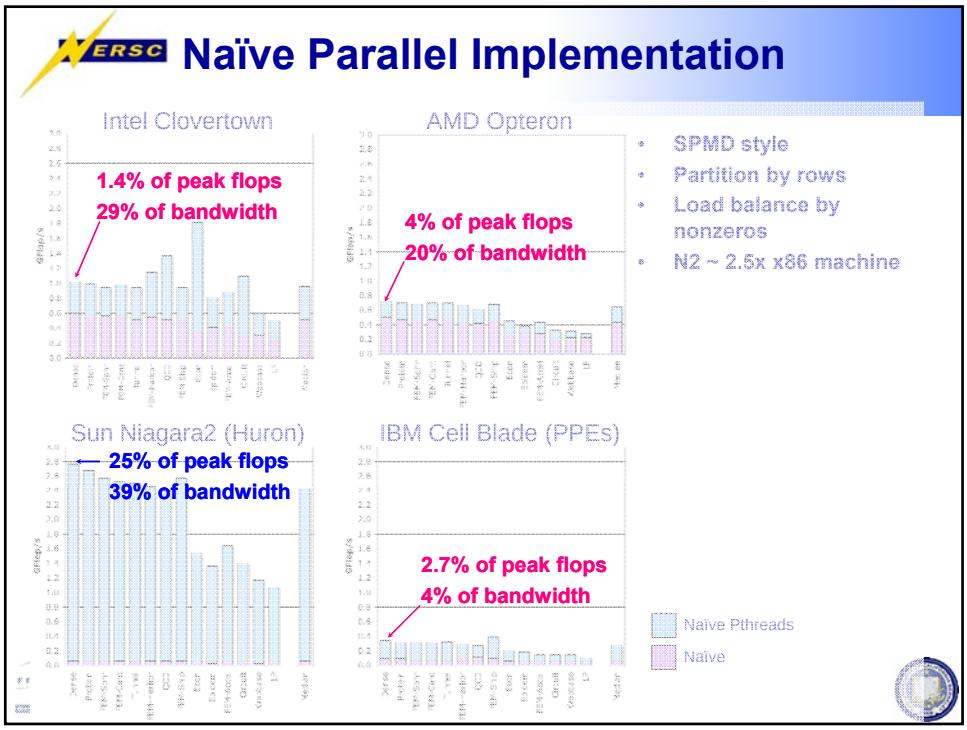




Naïve Pthreads

Naïve

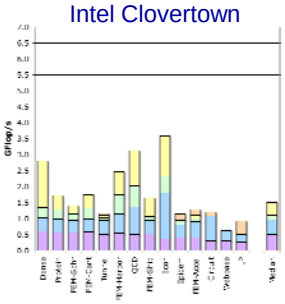

Naïve Parallel Implementation
Intel Clovertown
AMD Opteron
8x cores = 1.9x performance
4x cores = 1.5x performance
SPMD style
Partition by rows
Load balance by nonzeros
N2 ~ 2.5x x86 machine
Sun Niagara2 (Huron)
IBM Cell Blade (PPEs)
64x threads = 41x performance
4x threads = 3.4x performance
Naïve Pthreads
Naïve


Naïve Parallel Implementation
Intel Clovertown
AMD Opteron
1.4% of peak flops
29% of bandwidth
4% of peak flops
20% of bandwidth
SPMD style
Partition by rows
Load balance by nonzeros
N2 ~ 2.5x x86 machine
Sun Niagara2 (Huron)
IBM Cell Blade (PPEs)
25% of peak flops
39% of bandwidth
2.7% of peak flops
4% of bandwidth
Naïve Pthreads
Naïve

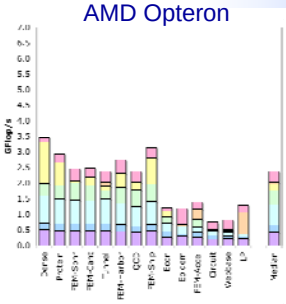
Autotuned Performance

(+DIMMs, Firmware, Padding)

Intel Clovertown

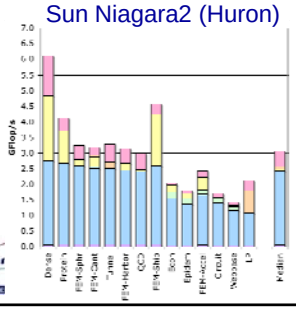


AMD Opteron

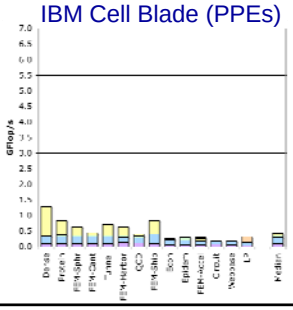


- **Clovertown was already fully populated with DIMMs**
- **Gave Opteron as many DIMMs as Clovertown**
- **Firmware update for Niagara2**
- **Array padding to avoid inter-thread conflict misses**
- **PPE's use ~1/3 of Cell chip area**

Sun Niagara2 (Huron)



IBM Cell Blade (PPEs)

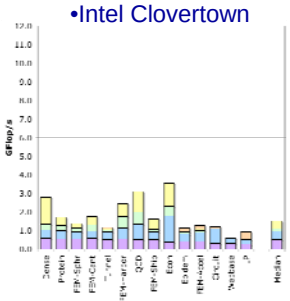


- +More DIMMs(opteron), +FW fix, array padding(N2), etc…
- +Cache/TLB Blocking
- +Compression
- +SW Prefetching
- +NUMA/Affinity
- Naïve Pthreads
- Naïve

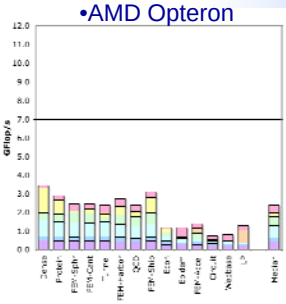
Autotuned Performance

(+Cell/SPE version)

•Intel Clovertown

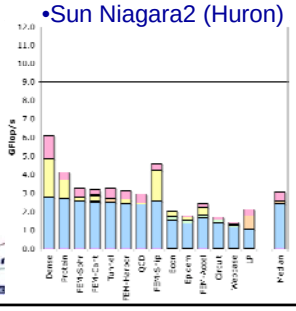


•AMD Opteron

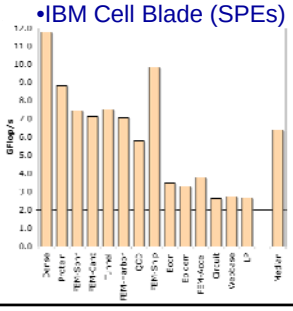


- **Wrote a double precision Cell/SPE version**
- **DMA, local store blocked, NUMA aware, etc…**
- **Only 2x1 and larger BCOO**
- **Only the SpMV-proper routine changed**
- **About 12x faster (median) than using the PPEs alone.**

•Sun Niagara2 (Huron)



•IBM Cell Blade (SPEs)

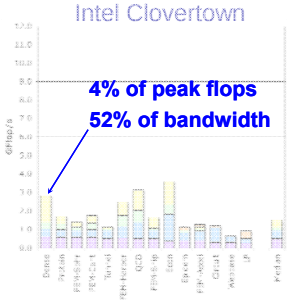


- •+More DIMMs(opteron), •+FW fix, array padding(N2), etc…
- •+Cache/TLB Blocking
- •+Compression
- •+SW Prefetching
- •+NUMA/Affinity
- •Naïve Pthreads
- •Naïve

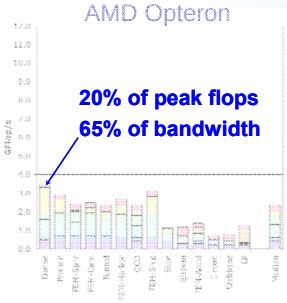
Autotuned Performance

(+Cell/SPE version)

Intel Clovertown

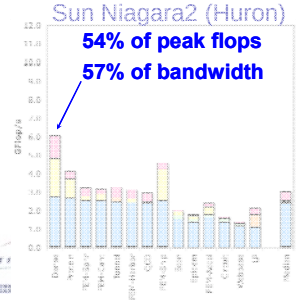


AMD Opteron

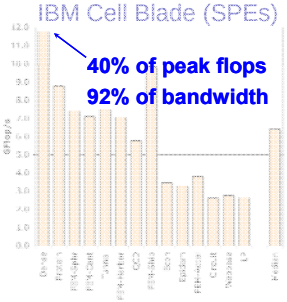


- Wrote a double precision Cell/SPE version
- DMA, local store blocked, NUMA aware, etc…
- Only 2x1 and larger BCOO
- Only the SpMV-proper routine changed
- About 12x faster than using the PPEs alone.

Sun Niagara2 (Huron)



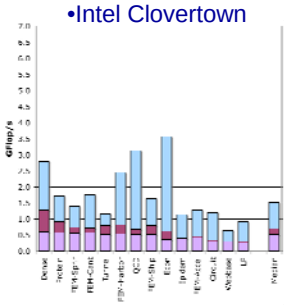
IBM Cell Blade (SPEs)



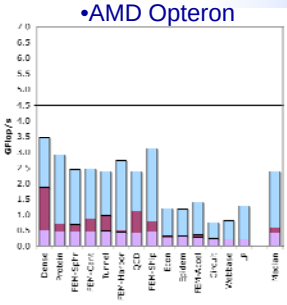
- +More DIMMs(opteron), +FW fix, array padding(N2), etc…
- +Cache/TLB Blocking
- +Compression
- +SW Prefetching
- +NUMA/Affinity
- Naïve Pthreads
- Naïve

MPI vs. Threads

•Intel Clovertown

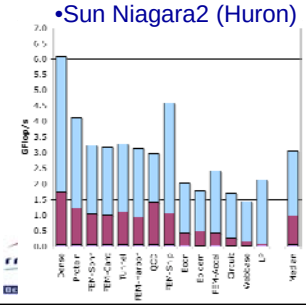


•AMD Opteron



- **On x86 machines, autotuned(OSKI) shared memory MPICH implementation rarely scales beyond 2 threads**
- **Still debugging MPI issues on Niagara2, but so far, it rarely scales beyond 8 threads.**

•Sun Niagara2 (Huron)



- •Autotuned pthreads
- •Autotuned MPI
- •Naïve Serial

56

Optimized Sparse Kernel Interface - OSKI

- **Provides sparse kernels automatically tuned for user's matrix & machine**
  - BLAS-style functionality: SpMV, Ax & A^Ty, TrSV
  - Hides complexity of run-time tuning
  - Includes new, faster locality-aware kernels: A^TAx, A^kx
- **Faster than standard implementations**
  - Up to 4x faster matvec, 1.8x trisolve, 4x A^TA*x
- **For "advanced" users & solver library writers**
  - Available as stand-alone library (OSKI 1.0.1h, 6/07)
  - Available as PETSc extension (OSKI-PETSc .1d, 3/06)
  - Bebop.cs.berkeley.edu/oski

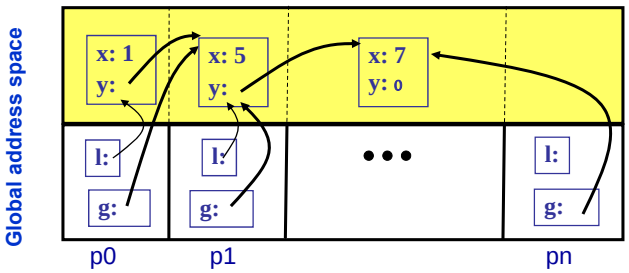
Programming Model for Multicore

- **These autotuned implementations, use:**
  - Fixed set of threads (pthreads)
    - "Parallel all the time"
  - Shared memory
    - Avoid unnecessarily replication
  - Logically partitioned memory with affinity
    - Avoid unnecessary cache coherence traffic
- **What programming model offers these?**

58

PGAS Languages: Why use 2 Languages (MPI+X) when 1 will do?

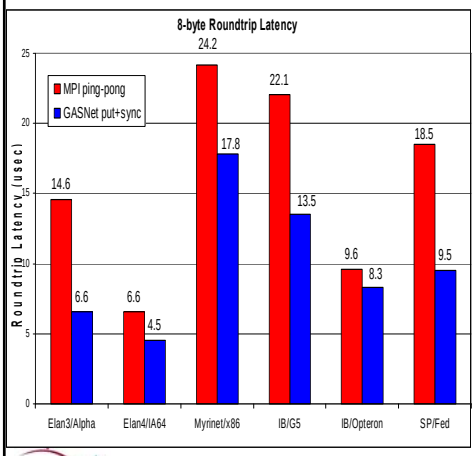
- *Global address space:* thread may directly read/write remote data
- *Partitioned:* data is designated as local or global

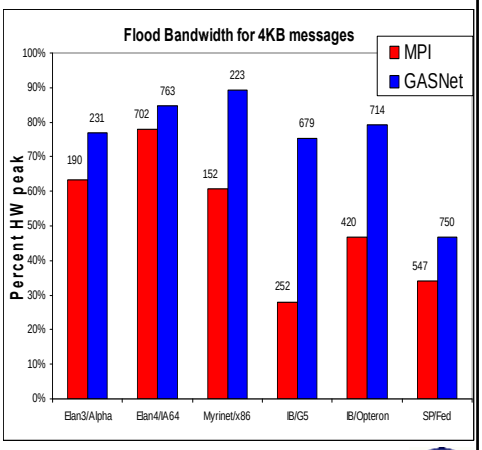


- Remote put and get: never have to say "receive"
  - Remote function invocation? See HPCS languages
- No less scalable than MPI!
- Permits sharing, whereas MPI rules it out!
- One model rather than two, but if you insist on two:
  - Can call UPC from MPI and vice verse (tested and used)

#5) Used Lightweight Communication

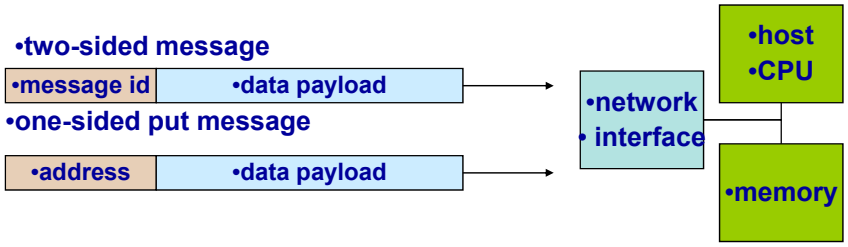
Use a programming model in which you can utilize bandwidth and "low" latency





Joint work with Berkeley UPC Group

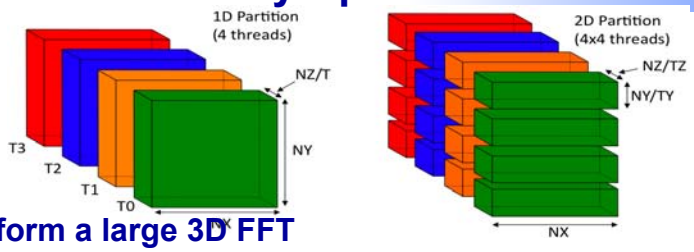
Two-sided vs One-sided Communication



- Two-sided message passing (e.g., MPI) requires matching a send with a receive to identify memory address to put data
  - Wildly popular in HPC, but cumbersome in some applications
  - Couples data transfer with synchronization
- Using global address space decouples synchronization
  - Pay for what you need!
  - Note: Global Addressing ≠ Cache Coherent Shared memory

Joint work with Dan Bonachea, Paul Hargrove, Rajesh Nishtala and rest of UPC group

Case Study Update: NAS FT

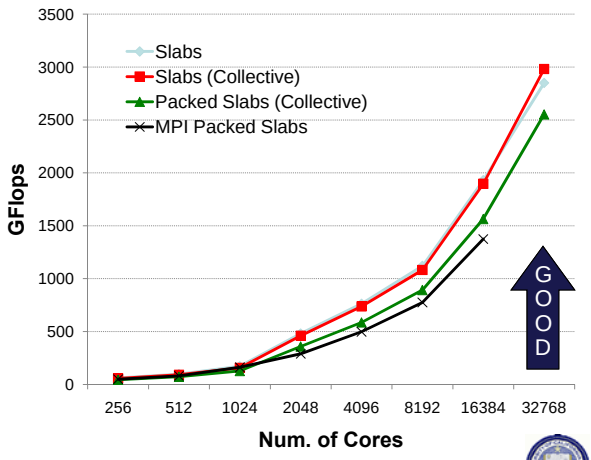


- Perform a large 3D FFT
  - Represents bisection-bandwidth limited algorithms
- Builds on our previous work, but with a 2D partition
  - Requires two rounds of communication rather than one
  - Each processor communicates with O(√T) threads
- Leverage nonblocking communication
  - Packed: minimize messages, no overlap
  - Slab: no packing, one message per plane per "other" thread

FFT Performance on BlueGene/P

- PGAS implementations consistently outperform MPI
- Leveraging communication/computation overlap yields best performance
  - More collectives in flight and more communication leads to better performance
  - At 32k cores, overlap algorithms yield 17% improvement in overall application time
- Numbers are getting close to HPC record
  - Future work to try to beat the record

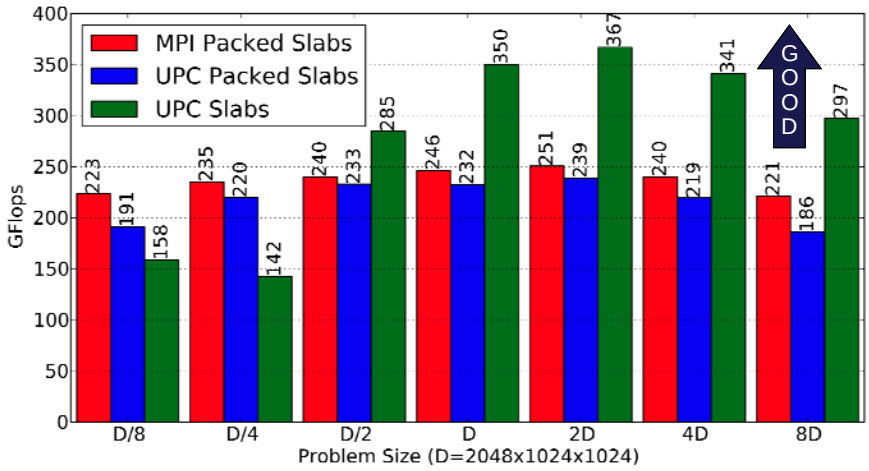
HPC Challenge Peak as of July 09 is ~4.5 Tflops on 128k Cores



63

FFT Performance on Cray XT4

- **1024 Cores of the Cray XT4**
  - **Uses FFTW for local FFTs**
  - **Larger the problem size the more effective the overlap**

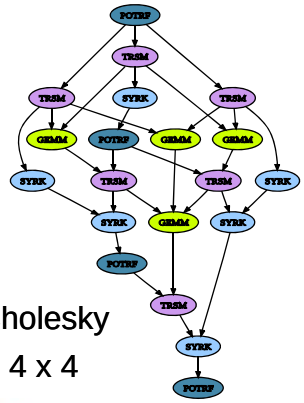


64

#6) Avoid Synchronization

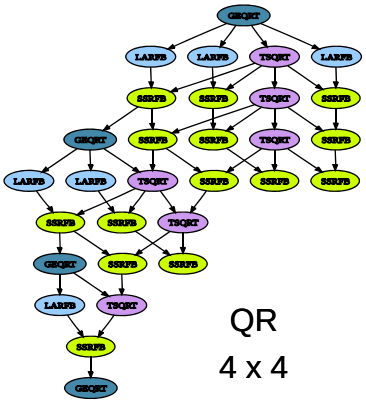
Computations as DAGs

View parallel executions as the directed acyclic graph of the computation



Cholesky

4 x 4



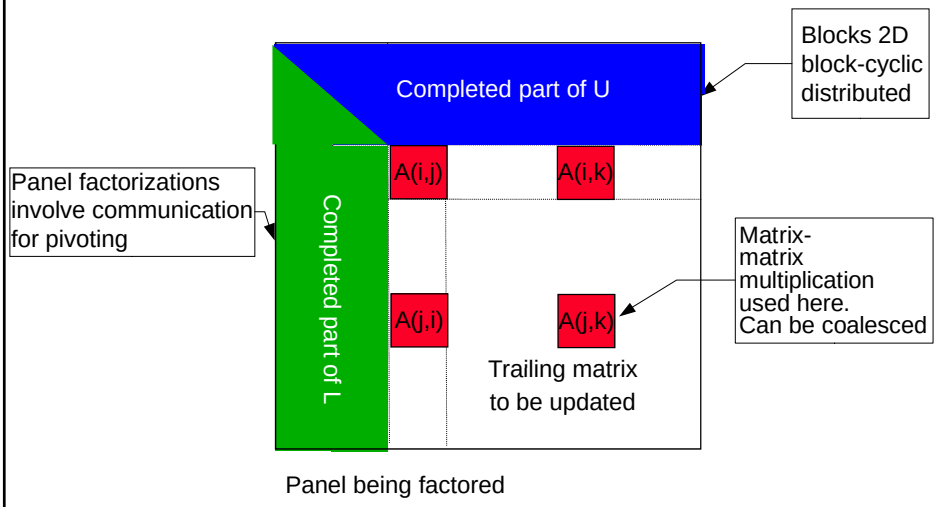
QR

4 x 4

Slide source: Jack Dongarra

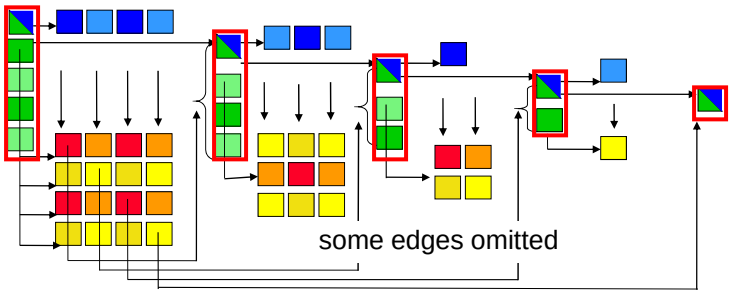
65

Parallel LU Factorization



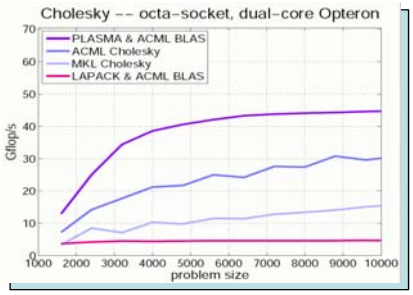
Event Driven Execution of Dense LU

- Ordering needs to be imposed on the schedule
- Critical operation: Panel Factorization
  - need to satisfy its dependencies first
  - perform trailing matrix updates with low block numbers first
  - "memory constrained" lookahead
- General issue: dynamic scheduling in partitioned memory
  - Can deadlock memory allocator!

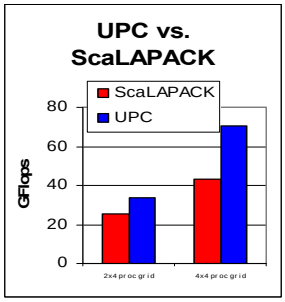


DAG Scheduling Outperforms Bulk-Synchronous Style

PLASMA on shared memory



UPC on partitioned memory



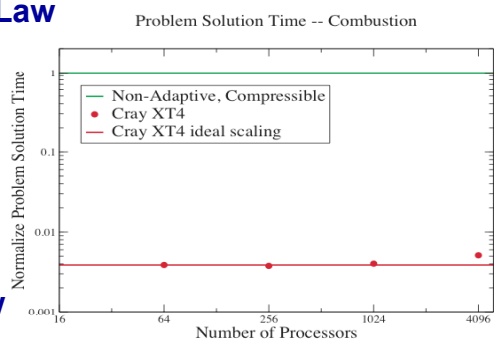
- UPC LU factorization code adds cooperative (non-preemptive) threads for latency hiding
  - New problem in partitioned memory: allocator deadlock
  - Can run on of memory locally due tounlucky execution order

PLASMA by Dongarra et al; UPC LU joint with Parray Husbands

68

#7) Use Scalable Algorithms

- **Algorithmic gains in last decade have far outstripped Moore's Law**
  - Adaptive meshes rather than uniform
  - Sparse matrices rather than dense
  - Reformulation of problem back to basics
- **Algorithmic gains have outstripped Moore's Law**



- **Example of canonical "Poisson" problem on n points:**
  - Dense LU: most general, but $O(n^3)$ flops on $O(n^2)$ data
  - Multigrid: fastest/smallest, $O(n)$ flops on $O(n)$ data

Performance results: John Bell et al

Conclusions

- **Use your communication systems effectively**
  - On-chip communication between cores
  - Communication between processor and memory
  - Communication between sockets
  - Communication between nodes
- **Work with experts on hardware, software, algorithms, applications**

More Info

- The Berkeley View/Parlab
  - http://view.eecs.berkeley.edu
  - http://parlab.eecs.berkeley.edu/
- Berkeley Autotuning and PGAS projects
  - http://bebop.cs.berkeley.edu
  - http://upc.lbl.gov
  - http://titanium.cs.berkeley.edu
- NERSC System Architecture Group
  - http://www.nersc.gov/projects/SDSA
- LBNL Future Technologies Group

  http://crd.lbl.gov/ftg