

## Scientific Computing in the Americas: the challenge of massive parallelism

3–14 January 2011, Valparaiso, Chile

# Exaflop/s, seriously!

**David Keyes**

**Division of Mathematical and Computer Sciences and Engineering, KAUST**

**Fu Foundation Professor of Applied Mathematics, Columbia University**

**Based on an article “The Exascale: Why and How” to appear in 2011 in**

***Comptes Rendus de l’Academie des Sciences (CRAS)***

# Credits for this talk include the IESP team

[www.exascale.org](http://www.exascale.org)

## INTERNATIONAL EXASCALE ROADMAP 1.0 SOFTWARE PROJECT



The International Exascale  
Software Roadmap,  
J. Dongarra, P. Beckman, et al.,  
*International Journal of High  
Performance Computer  
Applications* **25**(1), 2011 (to  
appear), ISSN 1094-3420.

- |                       |                 |                  |                   |                   |
|-----------------------|-----------------|------------------|-------------------|-------------------|
| Jack Dongarra         | Alok Choudhary  | Sanjay Kale      | Matthias Mueller  | Bob Sugar         |
| Pete Beckman          | Sudip Dosanjh   | Richard Kenway   | Wolfgang Nagel    | Shinji Sumimoto   |
| Terry Moore           | Thom Dunning    | David Keyes      | Hiroshi Nakashima | William Tang      |
| Patrick Aerts         | Sandro Fiore    | Bill Krumpal     | Michael E. Papka  | John Taylor       |
| Giovanni Aloisio      | Al Geist        | Jesus Labarta    | Dan Reed          | Rajeev Thakur     |
| Jean-Claude Andre     | Bill Gropp      | Alain Lichnewsky | Mitsuhsisa Sato   | Anne Trefethen    |
| David Barkai          | Robert Harrison | Thomas Lippert   | Ed Seidel         | Mateo Valero      |
| Jean-Yves Berthou     | Mark Hereld     | Bob Lucas        | John Shalf        | Aad van der Steen |
| Taisuke Boku          | Michael Heroux  | Barney Maccabe   | David Skinner     | Jeffrey Vetter    |
| Bertrand Braunschweig | Adolfy Hoisie   | Satoshi Matsuoka | Marc Snir         | Peg Williams      |
| Franck Cappello       | Koh Hotta       | Paul Messina     | Thomas Sterling   | Robert Wisniewski |
| Barbara Chapman       | Yutaka Ishikawa | Peter Michielsse | Rick Stevens      | Kathy Yelick      |
| Xuebin Chi            | Fred Johnson    | Bernd Mohr       | Fred Streit       |                   |

### SPONSORS



## The G8 Research Councils Initiative on Multilateral Research Funding



### The first Call for Proposals: Interdisciplinary Program on Application Software towards Exascale Computing for Global Scale Issues



#### The G8 Research Councils Initiative on Multilateral Research Funding

The Heads of Research Councils from the G8 countries (G8-HORCs) Canada, France, Germany, Japan, Russia, the UK, and the USA (Italy is not participating) have established a new joint funding initiative.

Global challenges need global solutions and this pilot initiative provides a new framework for co-operation across a broad range of disciplines and on a multilateral and multinational basis.

The initiative begins with a first call on Exascale Computing, further calls on other topics are expected in 2011 and 2012.



## Questions to consider

- Why the push to the exascale?
- What will systems a thousand times faster than today's petascale systems look like architecturally?
- What will be the implications of their budgets for power and acquisition?
- What should we do about it to prepare, algorithmically?



# Why push to extreme scale?

- Better resolve model's full, natural range of length or time scales
- Accommodate physical effects with greater fidelity
- Allow the model degrees of freedom in all relevant dimensions
- Better isolate artificial boundary conditions (e.g., in PDEs) or better approach realistic levels of dilution (e.g., in MD)
- Combine multiple complex models
- Solve an inverse problem, or perform data assimilation
- Perform optimization or control
- Quantify uncertainty
- Improve statistical estimates
- Operate without models (machine learning)



**“Third paradigm”**



**“Fourth paradigm”**



# The third paradigm



*The “third paradigm” paper (1986)*

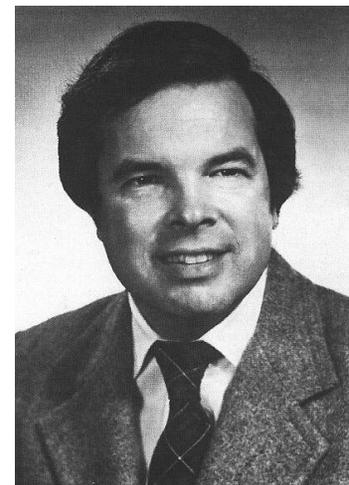
“During its spectacular rise, the computational has joined the theoretical and the experimental branches of science...”

– Peter D. Lax, in *J. Stat. Phys.*, 43:749-756

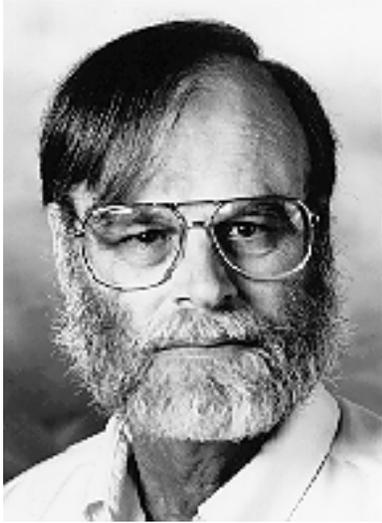
*The “Grand Challenge” paper (1989)*

"Grand Challenges of Computational Science" ... define opportunities to open up vast new domains of scientific research, domains that are inaccessible to traditional experimental or theoretical models of investigation.” – Kenneth G. Wilson,

in *Future Generation Computer Systems*, 5:171-189



# The fourth paradigm



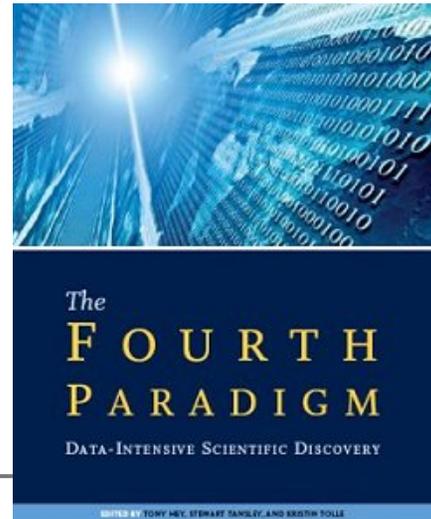
*The Data-centric world paper (2006)*

“The data need to be curated with metadata, stored under a schema with a controlled vocabulary, and indexed and organized for quick and efficient temporal, spatial, and associative search.”

— James N. Gray *et al.*, in *IEEE Computer*, 39:110-112

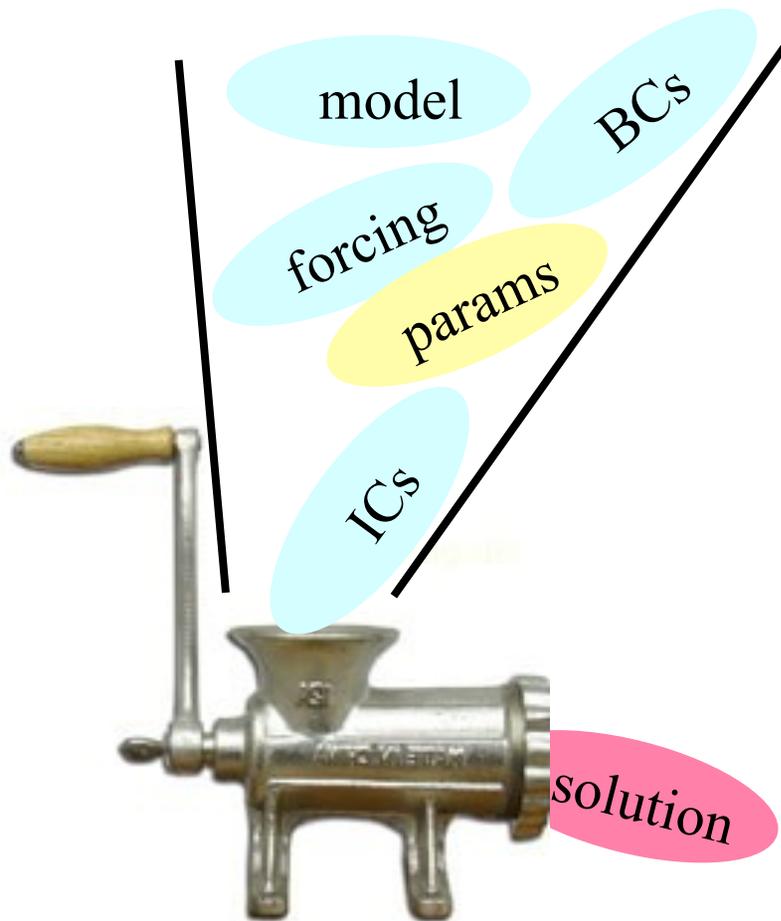
“....Authors in this volume ... refine an understanding of this new paradigm from a variety of disciplinary perspectives.”

— Gordon Bell, *Microsoft Research*

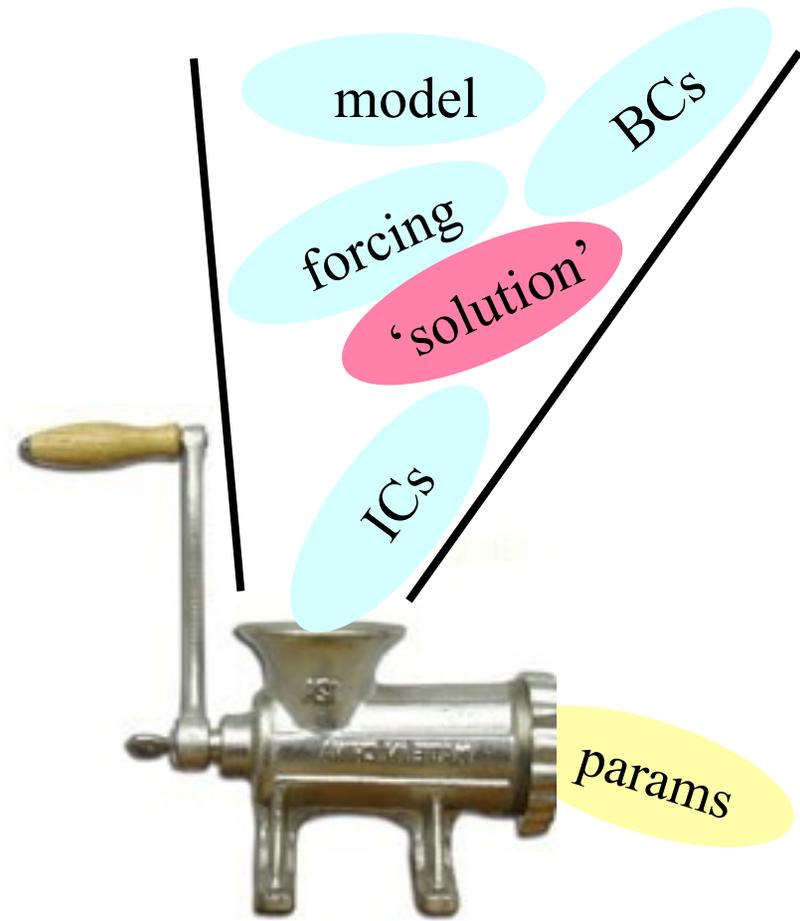


# Combining the paradigms: forward vs. inverse problems

forward problem



inverse problem



+ regularization

# How would these needs be felt at, *e.g.*, an oil company?

- Better resolve model's full, natural range of length or time scales
  - ◆ Discretize a reservoir into more layers and horizontal cells
- Accommodate physical effects with greater fidelity
  - ◆ Replace black oil assumption with fuller compositional effects
- Allow the model degrees of freedom in all relevant dimensions
  - ◆ Here, all three space dimensions, plus time
- Better isolate artificial boundary conditions or better approach realistic levels of dilution
  - ◆ Use additional cells outside of the production and injection zones to buffer the zones where data is needed from unknown geology and fluxes



# How would these needs be felt at, *e.g.*, an oil company?

- Solve an inverse problem, or perform data assimilation
  - ◆ Match simulation with drilling and historical production records to better estimate unknown parameters in the model and nudge the simulation towards reality, where and when reality is known
- Combine multiple complex models
  - ◆ Unify the simulations of adjacent fields to capture the effects of producing one of them on the other, monitor production through changes in seismic profiles, or model surface subsidence
- Perform optimization or control
  - ◆ Select a strategy for injection and pumping in the thousands of wells per reservoir

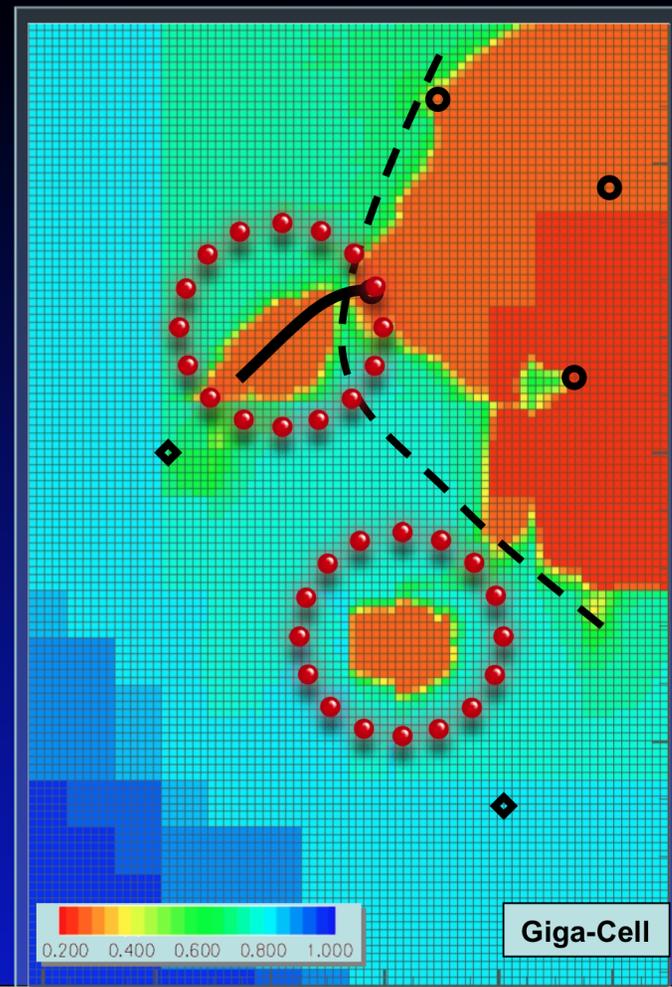
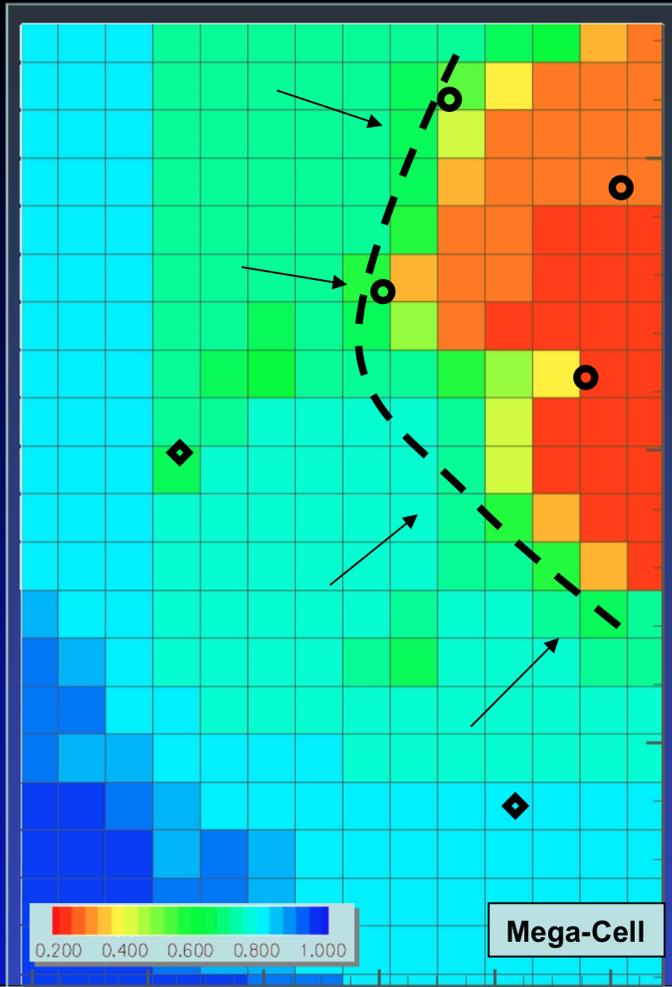


# How would these needs be felt at, *e.g.*, an oil company?

- Quantify uncertainty
  - ◆ Given the many unresolvable uncertainties in program inputs, bound the error in the outputs in terms of errors in the inputs
- Improve statistical estimates
  - ◆ Treating uncertain inputs as random, improve output estimates



# GigaPOWERS™ Impact High Resolution - Higher Recovery



© Copyright 2010, Saudi Aramco. All Rights Reserved.

Dhahran, Saudi Arabia

November 25, 2008

Page 1

c/o A. Dogru (used by permission)

PASI, 4 Jan 2011



# Need for extreme scale goes far beyond these, however!

- Oil companies are vast, with dozens of reservoirs “upstream” and many refineries and transportation systems “downstream”
- Oil companies function under many constraints for product supply and must seek to *maximize profit* while *satisfying output demands* in many different product streams producible from the same crude
- Tens of thousands of “valves” (literal and figurative, like workforce and other controls) need to be scheduled continuously
- Mathematically, this is a massive, nonlinear and possibly nonrobust constrained optimization problem – insatiably power hungry



# How should such resources be managed?

- They are too complex for a self-contained, self-consistent theory
- They are unsuitable for experiment, because you can only do the experiment (*e.g.*, exploiting a reservoir) once
- Engineering heuristics are useful (and used!), but
  - ◆ gone are the days when employees spent decades understanding a single reservoir
  - ◆ the external forcings (*e.g.*, world economy) change daily, making history less useful
- *Simulation* is an incredibly useful tool for exploring scenarios experimentally in a virtual world
- *Data mining and machine learning* may be even more useful tools in the future



**NEXT 9 SLIDES CONTAIN  
RAW OUTPUT FROM THE  
OCTOBER 2010 IESP WORKSHOP**



# IESP roadmapping: 19 candidate exascale applications

- Caveat: Neither a complete nor an independent basis, but productive to consider
  - ◆ Interesting feature: 7 of the apps represented at IESP are Gordon Bell prize winners or finalists
- Discrete algorithms are underrepresented, so far
- Not necessarily representative of IESP collaborating country research priorities or proportions for exascale simulation-enabled or data-enabled computing



# Applications inventory

- **Magnetically Confined Fusion**
  - ◆ Princeton PPL
  - ◆ Max Planck Inst.
  - ◆ LBNL
  - ◆ Kyoto University
- **Molecular Dynamics**
  - ◆ Chinese Academy of Sciences
  - ◆ LANL
  - ◆ LLNL
- **Climate**
  - ◆ CMCC
- **Combustion**
  - ◆ ORNL
- **Radio Astronomy**
  - ◆ CSIRO
- **Aerodynamics**
  - NASA
  - CERFACS
- **Fluid Dynamics and Heat Transfer**
  - ANL
- **Neutron Transport**
  - ANL
- **Nuclear Fuel Assemblies**
  - EDF
- **HEDP and Rad Hydro**
  - U Chicago
  - ORNL
- **Electronic Structure**
  - Fritz-Haber-Inst.
  - ORNL



# Applications survey questions

- Programming Models
  - ◆ message-passing, shared memory, hybrid
- Current scaling performance and percentage of peak
  - ◆ high-water mark scalability on a node and between nodes
- Algorithm dependencies
  - ◆ PDEs, IEs, N-body, FFT, FMM, etc.
- Software library dependencies
  - ◆ ScaLAPACK, PETSc, Zoltan, VisIt, etc.
- Balance of hardware resources
  - ◆ Input/Output vs. Computation
  - ◆ Communication vs. Computation
  - ◆ Synchronization vs. Computation
- Scaling requirements of memory with flop/s
  - ◆ Strong (if relevant)
  - ◆ Weak



PI	Application	Programming Model	Scaling	Cores per node	% Peak BW limited?	Algorithmic Dependencies	Libraries	I/O vs Comp	Comm vs. Comp.	Synch vs. Comp.	Memory vs. Flop/s
Ethier	Fusion, PIC	hybrid DD and particle dist.	no limit particle grid replicated	12	10%	PDEs: equil w MG	ParMetis PETSc HyPre	important: pre/post		frequent	log-linear
Guenter	Fusion, PIC	OpenMP/MPI hybrid GPUs under investigation	260K no known limit	32	10%	FFTs PDE: 3+2D PIC, Monte Carlo	BLAS ScaLAPACK	important: pre/post latency crit. BW less crit.		frequent	log-linear
Koniges	Fusion, Gyro	MPI w/hierarchical splitt	no known limit	4 to 8	10%	PDE: 3+2D, struct collision, unstruct	FFTW UMFPACK BLAS ScaLAPACK	not limiting			log-linear
Nakashima	Fusion, PIC	OpenMP/MPI hybrid GPUs useful in phases bulk synch SPMD DD	10K vulnerable to imbalance	16	10%	yes PDE: equil PIC	RNG		linear	per step	log-linear
Aloisio	Climate	OpenMP/MPI hybrid nested SPMD within MPMD DD with space-filling curves	86K	32		PDEs: evol PDEs: evol	FFTW PETSc ScaLAPACK Trilinos	important: pre/post superlinear		per step	log-linear
Zhong	MD	DD and particle dist. GPUs	2K	8		N-body FFT FMM	FFTW	not limiting	superlinear	frequent	log-linear
Swaminarayar	MD	hybrid GPU	100K	48	18-36%	yes		not limiting			log-linear
Cornwell	Radio Astro	SPMD DD	1M threads no known limit	3	20-40%	FFT	BLAS GRAPHLAB	EB per day	can be limiting		log-linear
Keyes	Aerodynamics	partial hybrid bulk synch SPMD DD	no known limit	4	5-15%	yes PDE: equil	ParMetis PETSc	not limiting	const w/weak	frequent	log-linear (min 1K vert/proc)
Fischer	Fluid/Heat Transf	partial hybrid bulk synch SPMD DD MPI	260K no known limit	4		yes PDE: equil	VisIt	important: pre/post const w/weak		frequent	log-linear (min 5K vert/proc)
Siegel	Neutron Transport	domain/energy/angle decomp MPI			5-8%		PETSc ParMetis Triangle	important: pre/post			log-linear (min 1K vert/proc)
Graziani	HEDP with FLASH	bulk synch SPMD DD MPI	130K, incl I/O		8-15%	PDE: equil AMR	PARAMESH Chombo, PETSc	important: pre/post output order of mag smaller than 1		frequent amenable to weaker	log-linear
Scheffler	Electronic Structure	OpenMP/MPI hybrid DFT	40K		10-15%	eigensolver	BLAS ScaLAPACK	important: pre/post		frequent	linear
Berthou	CFD for Nuclear Reactor	SPMD DD	>10K			PDE: equil PDE: evol					log-linear
Andre	Gas Turbines	nested SPMD within MPI	100K				PALM	important: pre/post			log-linear
Sankaran/Che	Combustion	bulk synch SPMD DD limited hybrid	224K	6	5-10%	yes unrolled integrator PDE: evol		periodic dump small input		per step	log-linear
Eisenbach	First Principle The	MC "walkers" outside of bulk synch SPMD DD	224K	6	75%		BLAS (ZGEMM) ScaLAPACK	important: pre/post			log-linear (multiple atoms per wa)
Messer	Astro Rad Hydro	bulk synch SPMD DD OpenMP for energy/angle	90K	6	5-15%	yes PDE: equil FFT AMR pointwise	PETSc SILO	important: pre/post			log-linear
Streitz	Classical MD	bulk synch SPMD DD hybrid possible over physics	300K	4	30%	N-body FFT				per step	linear



## Dominant findings on models and scaling

- Predominantly bulk synchronous, MPI and SPMD, either domain-, particle-, or other object-decomposed
- Electronic Structure codes, however, are dominantly *not* MPI, but global shared memory (e.g., GlobalArrays, Linda)
- Other models are Charm++ (NAMD) and distributed objects built on top of active messages and Pthreads
- Some codes have multiple phases with not necessarily compatible load-balanced decompositions
- Typically already running hybrid MPI/OpenMP with “small” numbers of cores, up to 48
- We can typically weak-scale without serious loss of scaling out to the edge of today’s machines (300K cores), and theoretically beyond



# Dominant findings on resources

- Memory bandwidth
  - ◆ Majority of the apps are already BW limited in most phases even with relatively few cores
- Flop/s per byte of storage
  - ◆ Generally linear or log-linear, in weak scaling
- I/O versus computation
  - ◆ With notable exceptions that are I/O intensive, the majority of our peta-apps need intensive I/O only at start-up and in dumping the output
  - ◆ We expect check-pointing to become much more intensive at the exascale, and perhaps limiting, even with users taking charge of check-pointing minimal state



## Dominant findings on resources, cont.

- Communication versus computation
  - ◆ In weak scaling there is generally a constant ratio of near-neighbor communication to computation, represented by a minimum collections of vertices, cells, particles, etc., per processor
- Synchronization versus computation
  - ◆ We require frequent global collectives
    - At least once per timestep in evolutionary codes
    - Even more frequently when implicit linear algebra needs to be performed within each timestep



# Core algorithms

- PDEs: equilibrium (implicit)
- PDEs: evolution (explicit)
- FFT
- FMM
- Particle pushing
- Adaptive mesh refinement
- Sparse and dense linear algebra
- ODE integrators



# Minimal library dependencies

- MPI, GlobalArrays, GasNet
- ParMetis
- BLAS, ScaLAPACK
- UMFPACK, SuperLU, MUMPS
- PETSc, hypre, Trilinos, SUNDIALS
- Chombo, SAMRAI
- FFTW
- GraphLib
- VisIt, VTK
- Triangle
- PALM
- SILO, ADIOS, HDF5
- BOOST



**END OF PRIMARY SOURCE DATA**



# Remaining presentation plan

- Reflect briefly on progress in high-end scientific computing
  - ◆ as captured in ACM Gordon Bell prize trends
  - ◆ as analyzed in some of the eight U.S. DOE “extreme scale” workshops of 2009-2010 ([extremescale.labworks.org](http://extremescale.labworks.org))
- Peek briefly at structure of a core motivating application
- Take a look at a few hurdles and possible solutions in algorithmic and programming model development arenas



# Exascale considerations

- Applications
  - ◆ What do we want to simulate at the exascale, why, and in what best formulation or sets of formulations?
- Architectures
  - ◆ What are the hurdles of granularity, cost, power, programmability, systems software, reliability (resilience) in delivering exascale systems, and how can they be surmounted?
- Algorithms
  - ◆ How can we get the applications to run on architectures that are physically and fiscally achievable (“co-design” of architecture and application)



# Philosophy of an algorithmicist

- Applications are *given* (as function of time)
- Architectures are *given* (as function of time)
- Algorithms and software *must be adapted or created* to bridge to hostile architectures for the sake of the complex applications
  - ◆ as important as ever today, with transformation of Moore's Law from speed-based to concurrency-based, due to power considerations
  - ◆ scalability still important, but new memory-bandwidth stresses arise when on-chip memories are shared
  - ◆ greatest challenge is lack of performance robustness of individual cores, which can spoil load balance
- Knowledge of algorithmic capabilities can usefully influence
  - ◆ the way applications are formulated
  - ◆ the way architectures are constructed
- Knowledge of application and architectural opportunities can usefully influence algorithmic development



# Tracking third paradigm progress: Gordon Bell Prize “peak performance”

<i>Year</i>	<i>Type</i>	<i>Application</i>	<i>No. Procs</i>	<i>System</i>	<i>Gflop/s</i>
1988	PDE	Structures	8	Cray Y-MP	1.0
1989	PDE	Seismic	2,048	CM-2	5.6
1990	PDE	Seismic	2,048	CM-2	14
1992	NB	Gravitation	512	Delta	5.4
1993	MC	Boltzmann	1,024	CM-5	60
1994	IE	Structures	1,904	Paragon	143
1995	MC	QCD	128	NWT	179
1996	PDE	CFD	160	NWT	111
1997	NB	Gravitation	4,096	ASCI Red	170
1998	DFT	Magnetism	1,536	T3E-1200	1,020
1999	PDE	CFD	5,832	ASCI BluePac	627
2000	NB	Gravitation	96	GRAPE-6	1,349
2001	NB	Gravitation	1,024	GRAPE-6	11,550
2002	PDE	Climate	5,120	Earth Sim	26,500
2003	PDE	Seismic	1,944	Earth Sim	5,000
2004	PDE	CFD	4,096	Earth Sim	15,200
2005	MD	Solidification	131,072	BG/L	101,700
2006	DFT	Elec. Struct.	131,072	BG/L	207,000
2007	MD	Kelvin-Helm.	131,072	BG/L	115,000
2008	DFT	Crystal Struct.	150,000	XT-5	1,352,000
2009	DFT	Nanoparticles	223,000	XT-5	2,330,000
2010	FMM	Physiology	196,608	XT-5	780,000



Six orders of  
magnitude in  
20 years

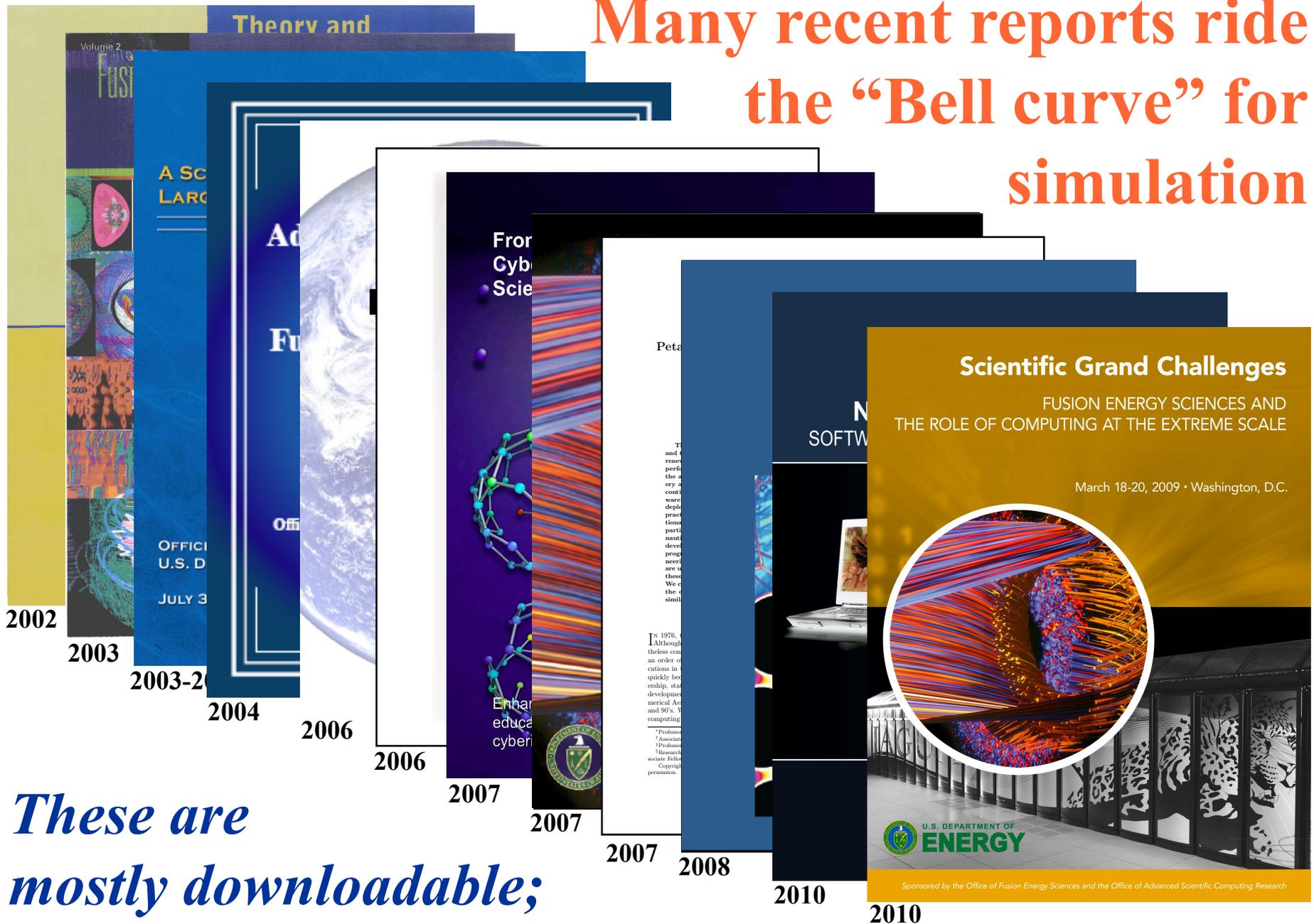
# Tracking third paradigm progress: Gordon Bell Prize: “price performance”

<i>Year</i>	<i>Application</i>	<i>System</i>	<i>\$ per Mflop.</i>
1989	Reservoir modeling	CM-2	2,500
1990	Electronic structure	IPSC	1,250
1992	Polymer dynamics	cluster	1,000
1993	Image analysis	custom	154
1994	Quant molecular dyn	cluster	333
1995	Comp fluid dynamics	cluster	278
1996	Electronic structure	SGI	159
1997	Gravitation	cluster	56
1998	Quant chromodyn	custom	12.5
1999	Gravitation	custom	6.9
2000	Comp fluid dynamics	cluster	1.9
2001	Structural analysis	cluster	0.24
2009	Gravitation & turbulence	cluster (GPU)	0.0081

5.5 orders of  
magnitude in  
20 years



# Many recent reports ride the “Bell curve” for simulation



*These are mostly downloadable; (e-mail me if interested)*

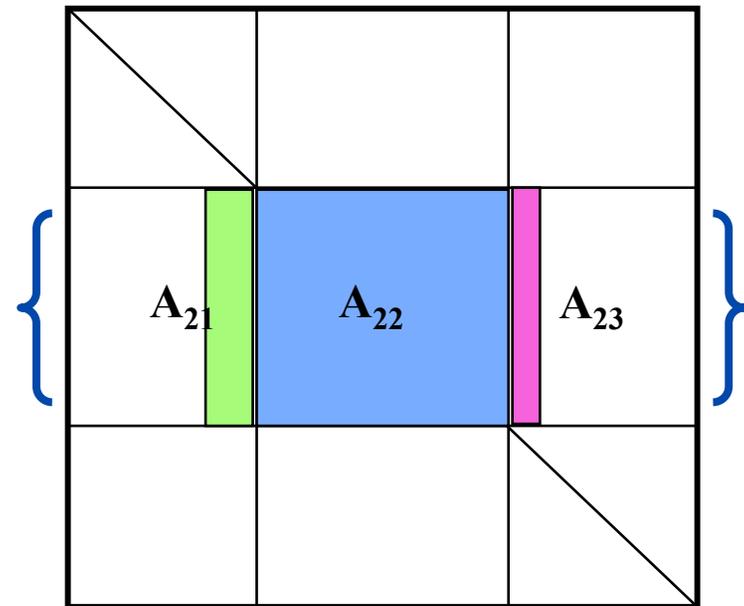
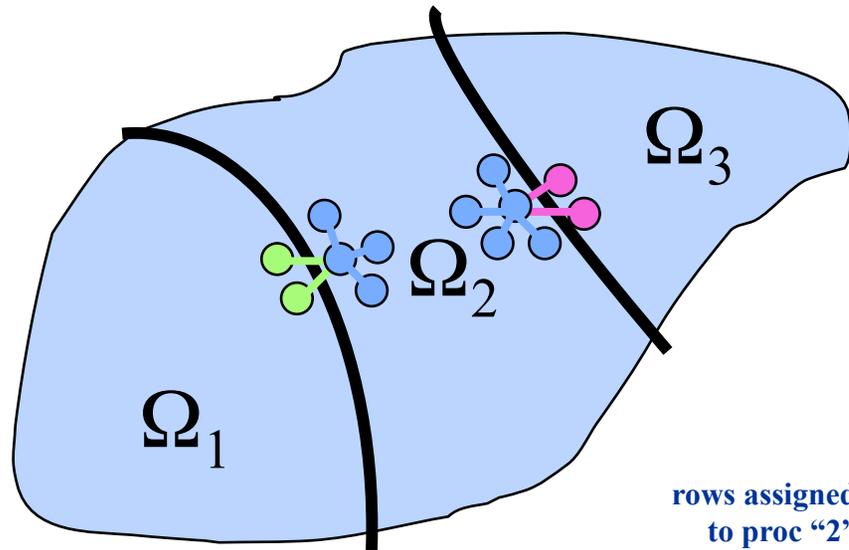
PASI, 4 Jan 2011



# How are problems like these solved at the petascale today?

- *Iterative methods based on domain decomposition and message-passing*
  - Each individual processor works on a subdomain of the original problem and exchanges information at its boundaries with other processors that own subdomains with which it interacts causally, to evolve in time or to establish equilibrium (steady state)
- **The programming model is SPMD/BSP/CSP**
  - **Single Program, Multiple Data**
  - **Bulk Synchronous Programming** refers to alternating nearly uniformly sized chunks of work on each processor with bursts of exchanges of data
  - **a.k.a. Communicating Sequential Processes**
- **Nearly all successful large-scale simulations are built this way today – and this may change radically at the exascale**

# SPMD parallelism with domain decomposition



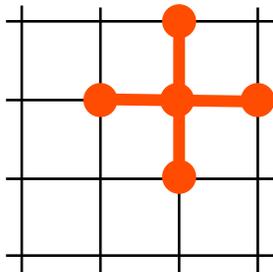
Partitioning of the grid  
induces block structure on  
the system matrix  
(Jacobian)

# Domain decomposition is relevant to any local stencil formulation

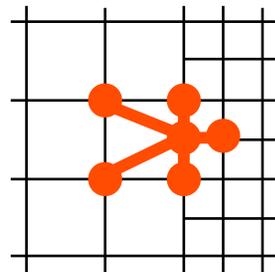
finite differences

finite elements

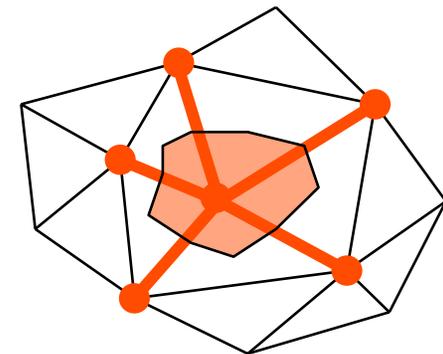
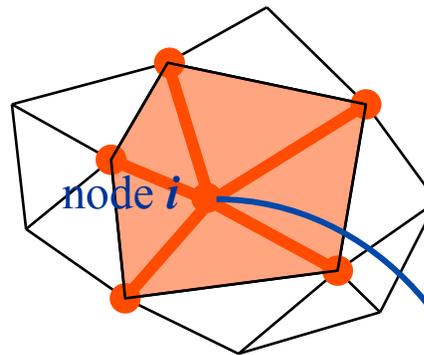
finite volumes



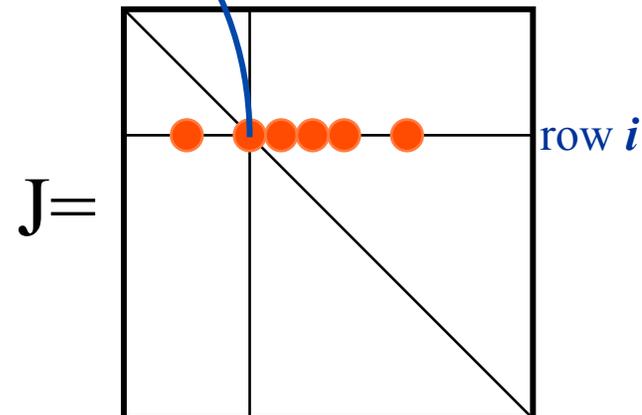
uniform



Cartesian  
adaptive



- All lead to sparse Jacobian matrices
- However, the inverses are generally dense; even the factors suffer unacceptable fill-in in 3D
- Want to solve in subdomains only, and use to precondition full sparse problem



# Newton-Krylov-Schur-Schwarz: a solver “workhorse”

$$F(u) \approx F(u_c) + F'(u_c)\delta u = 0$$

$$u = u_c + \lambda \delta u$$

$$J\delta u = -F$$

$$\delta u = \arg \min_{x \in V = \{F, JF, J^2F, \dots\}} \{Jx + F\}$$

$$B^{-1}J\delta u = -B^{-1}F$$

$$B^{-1} = \left( \begin{bmatrix} \tilde{A}_{ii} & 0 \\ A_{\Gamma i} & I \end{bmatrix} \begin{bmatrix} I & \tilde{A}_{ii}^{-1}A_{\Gamma i} \\ 0 & M \end{bmatrix} \right)^{-1} \quad \tilde{A}^{-1} = \sum_i R_i^T (R_i A R_i^T)^{-1} R_i$$



Newton

nonlinear solver

*asymptotically  
quadratic*



Krylov

accelerator

*spectrally  
adaptive*



Schur

preconditioner

*parallelizable  
by structure*



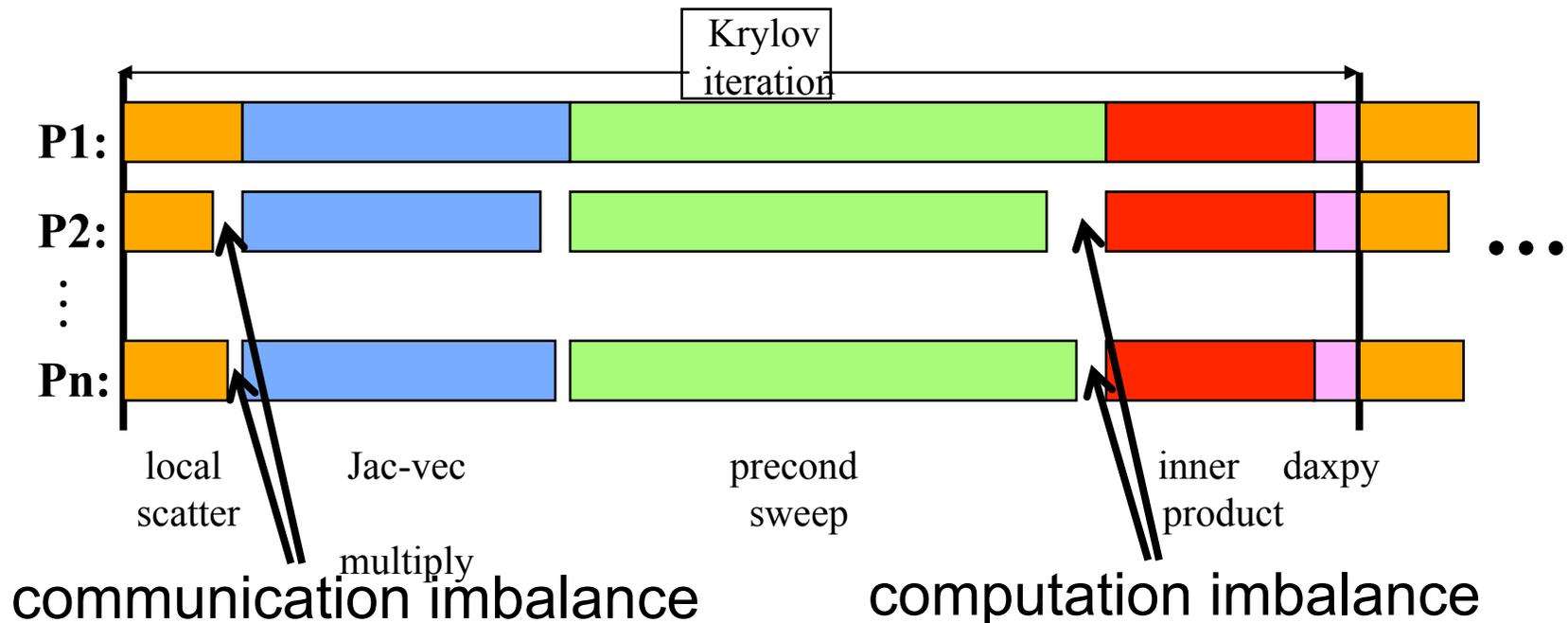
Schwarz

preconditioner

*parallelizable  
by domain*



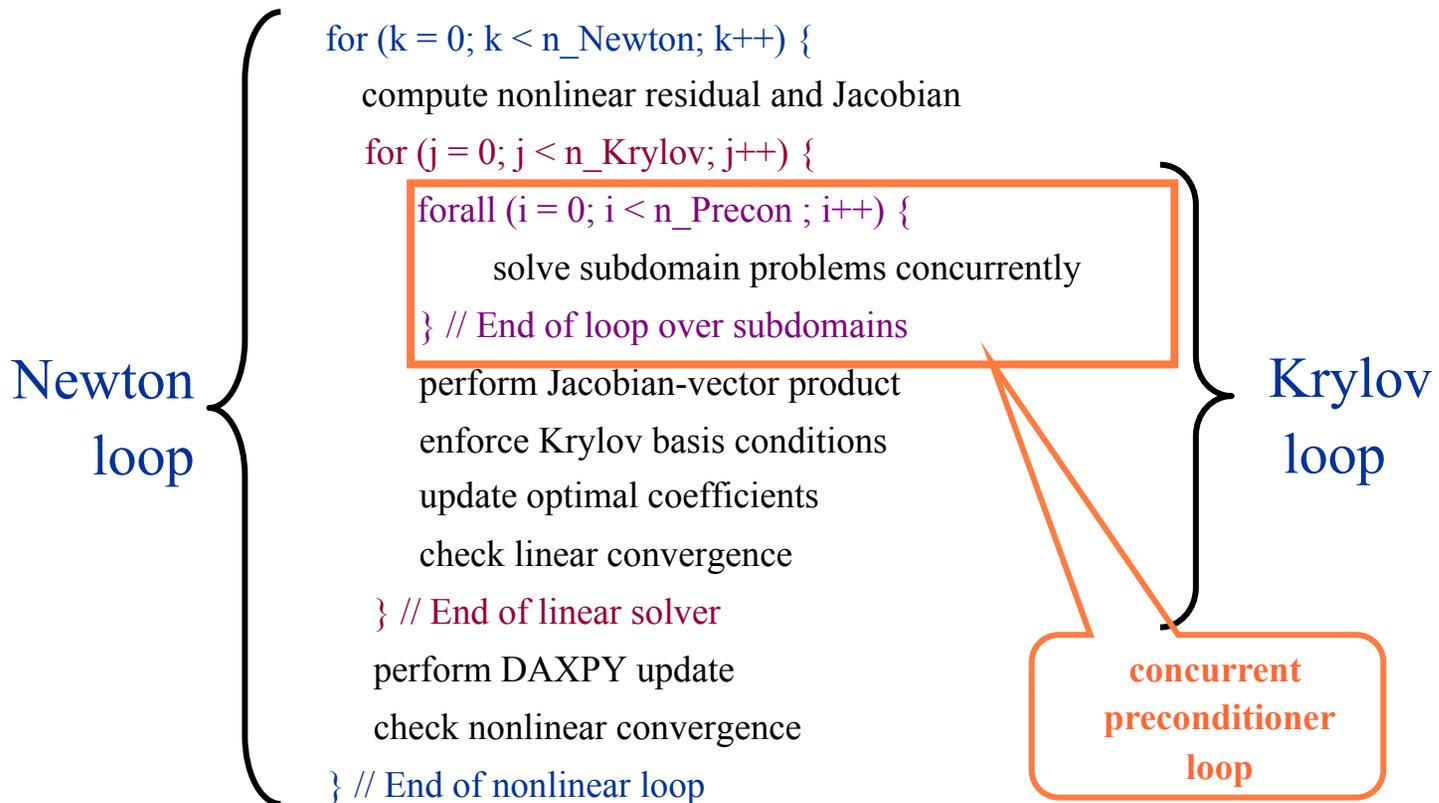
# Workhorse innards: Krylov-Schwarz, a Bulk Synchronous Process (“BSP”)



Idle time due to load imbalance becomes a challenge at, say, one million cores, when *one* processor can hold up *all* of the rest at a synchronization point



# Newton-Krylov-Schwarz



Outer loops (not shown): continuation, implicit timestepping, optimization



# What will first “general purpose” exaflop/s machines look like?

- Many paths beyond today’s CMOS silicon-based logic
- Earliest and most significant post-CMOS device improvement *may* be carbon nanotube memory, but not in 10 years
  - ◆ up to tens of GB on a 1 cm-square die
  - ◆ will deal directly with the “memory wall” problem
- Two paths from peta- to exa-
  - ◆ IBM: BlueGene’s successor, maybe at 22nm linewidth technology – some architectural merger of BlueGene, Power, and Cell
  - ◆ All others: GPGPU-based spinoff
- At least for PDE-based scientific codes:
  - ◆ programming model will still be message-passing (due to large legacy code base), adapted to multicore processors beneath the MPI interface, and made less synchronous



# Potential System Architectures

## *What is Possible?*

Systems	2009	2011	2015	2018
System Peak Flops/s	2 Peta	20 Peta	100-200 Peta	1 Exa
System Memory	0.3 PB	1 PB	5 PB	10 PB
Node Performance	125 GF	200 GF	400 GF	1-10 TF
Node Memory BW	25 GB/s	40 GB/s	100 GB/s	200-400 GB/s
Node Concurrency	12	32	O(100)	O(1000)
Interconnect BW	1.5 GB/s	10 GB/s	25 GB/s	50 GB/s
System Size (Nodes)	18,700	100,000	500,000	O(Million)
Total Concurrency	225,000	3 Million	50 Million	O(Billion)
Storage	15 PB	30 PB	150 PB	300 PB
I/O	0.2 TB/s	2 TB/s	10 TB/s	20 TB/s
MTTI	Days	Days	Days	O(1Day)
Power	6 MW	~10 MW	~10 MW	~20 MW

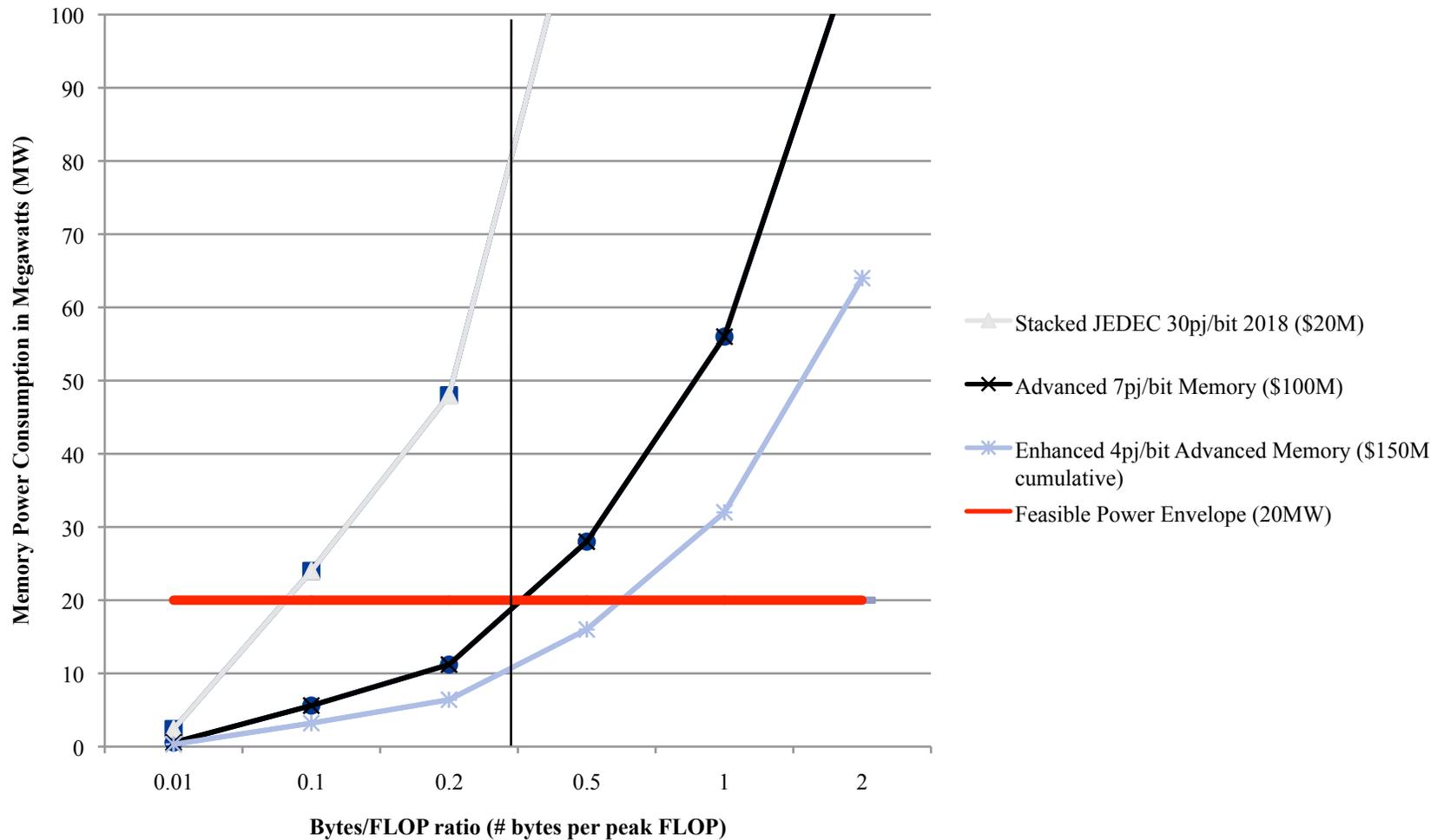


# Prototype exascale hardware: a heterogeneous, distributed memory *GigaHz KiloCore MegaNode* system

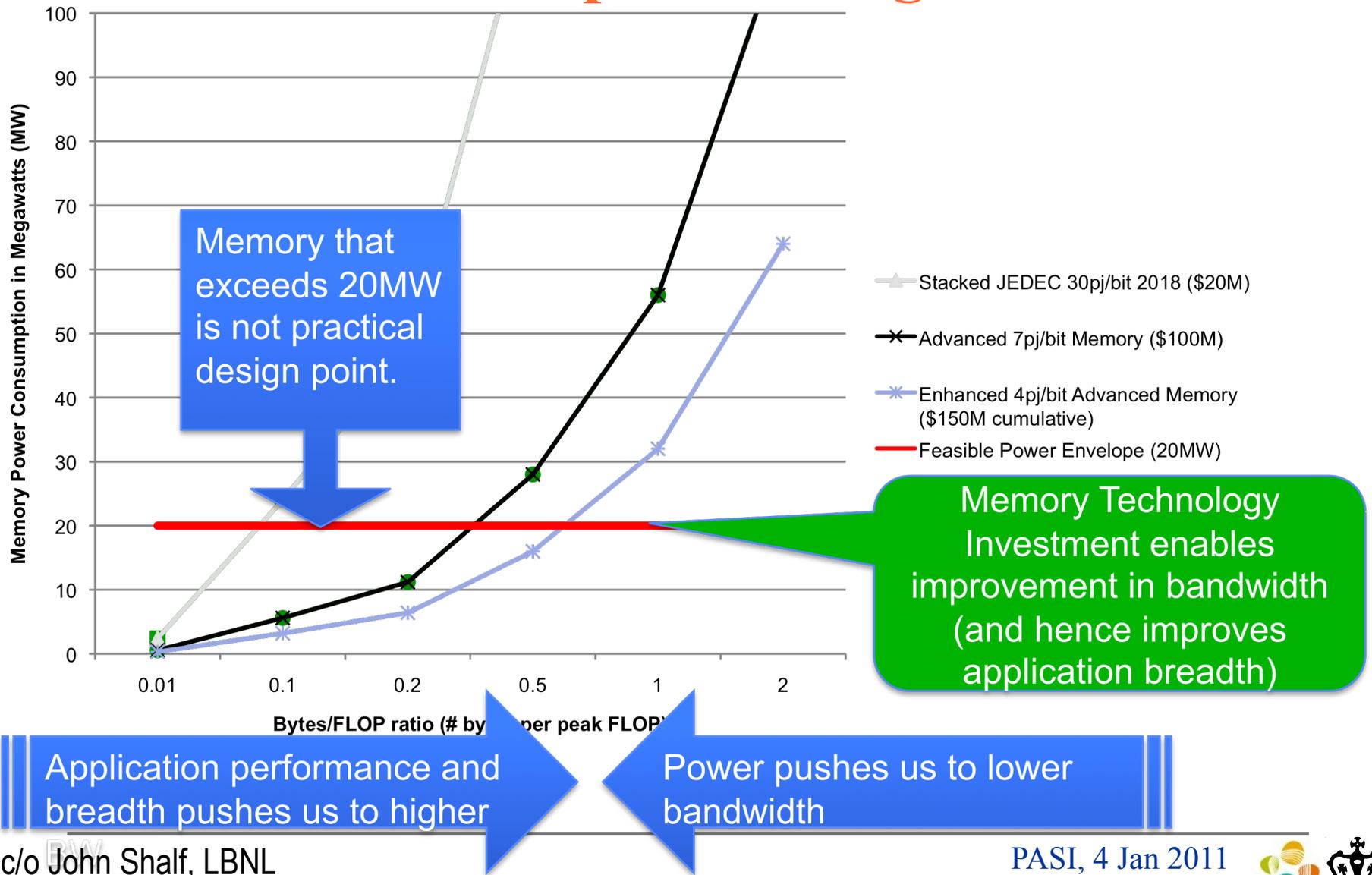
Systems	2009	2018	Difference Today & 2018
System peak	2 Pflop/s	1 Eflop/s	O(1000)
Power	6 MW	~20 MW	~3
System memory	0.3 PB	32 - 64 PB [ .03 Bytes/Flop ]	O(100)
Node performance	125 GF	1,2 or 15TF	O(10) – O(100)
Node memory BW	25 GB/s	2 - 4TB/s [ .002 Bytes/Flop ]	O(100)
Node concurrency	12	O(1k) or 10k	O(100) – O(1000)
Total Node Interconnect BW	3.5 GB/s	200-400GB/s (1:4 or 1:8 from memory BW)	O(100)
System size (nodes)	18,700	O(100,000) or O(1M)	O(10) – O(100)
Total concurrency	225,000	O(billion) [O(10) to O(100) for latency hiding]	O(10,000)
Storage	15 PB	500-1000 PB (>10x system memory is min)	O(10) – O(100)
IO	0.2 TB	60 TB/s (how long to drain the machine)	O(100)
MTTI	days	O(1 day)	- O(10)



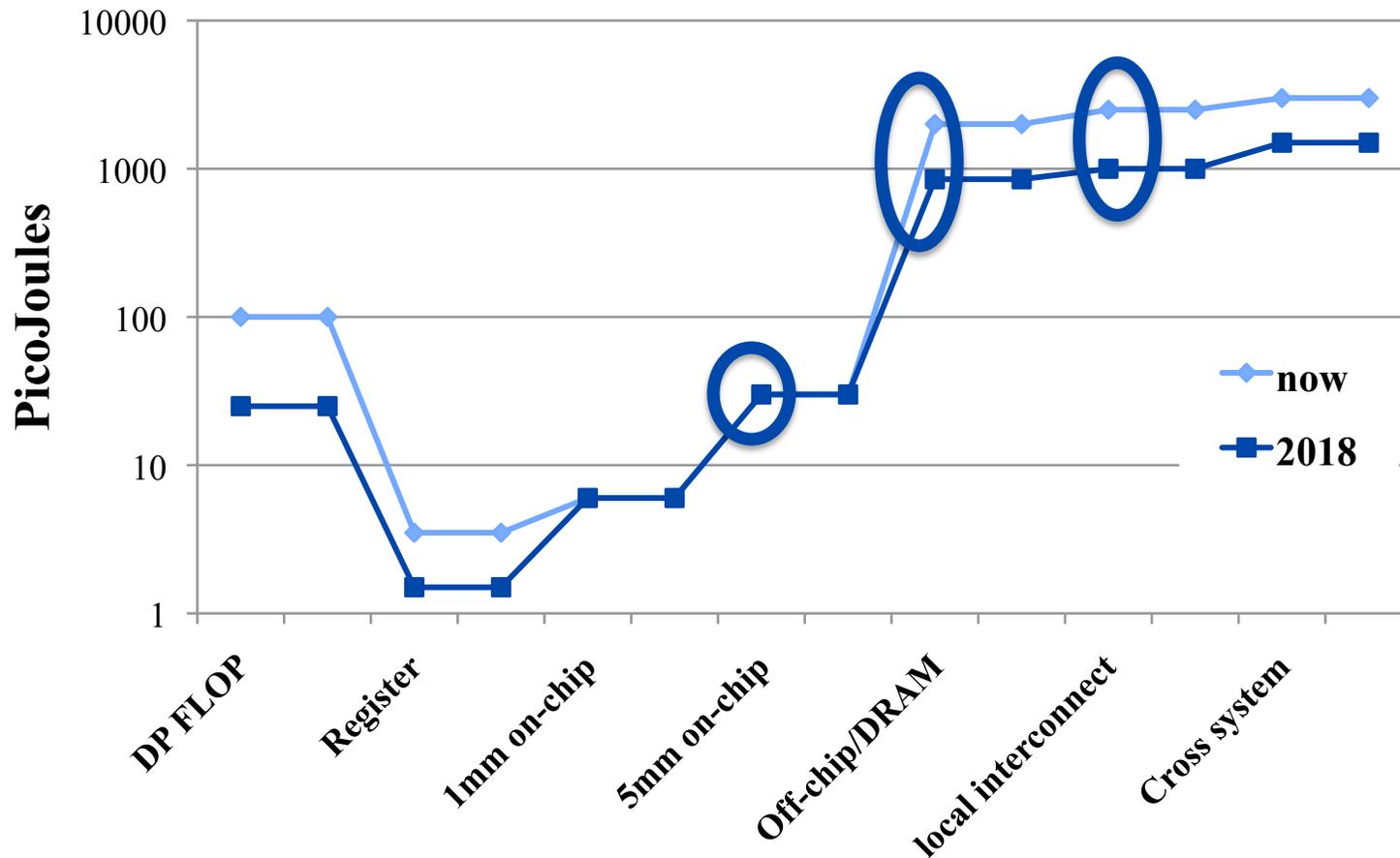
# Hurdle #1: memory bandwidth eats up the entire power budget



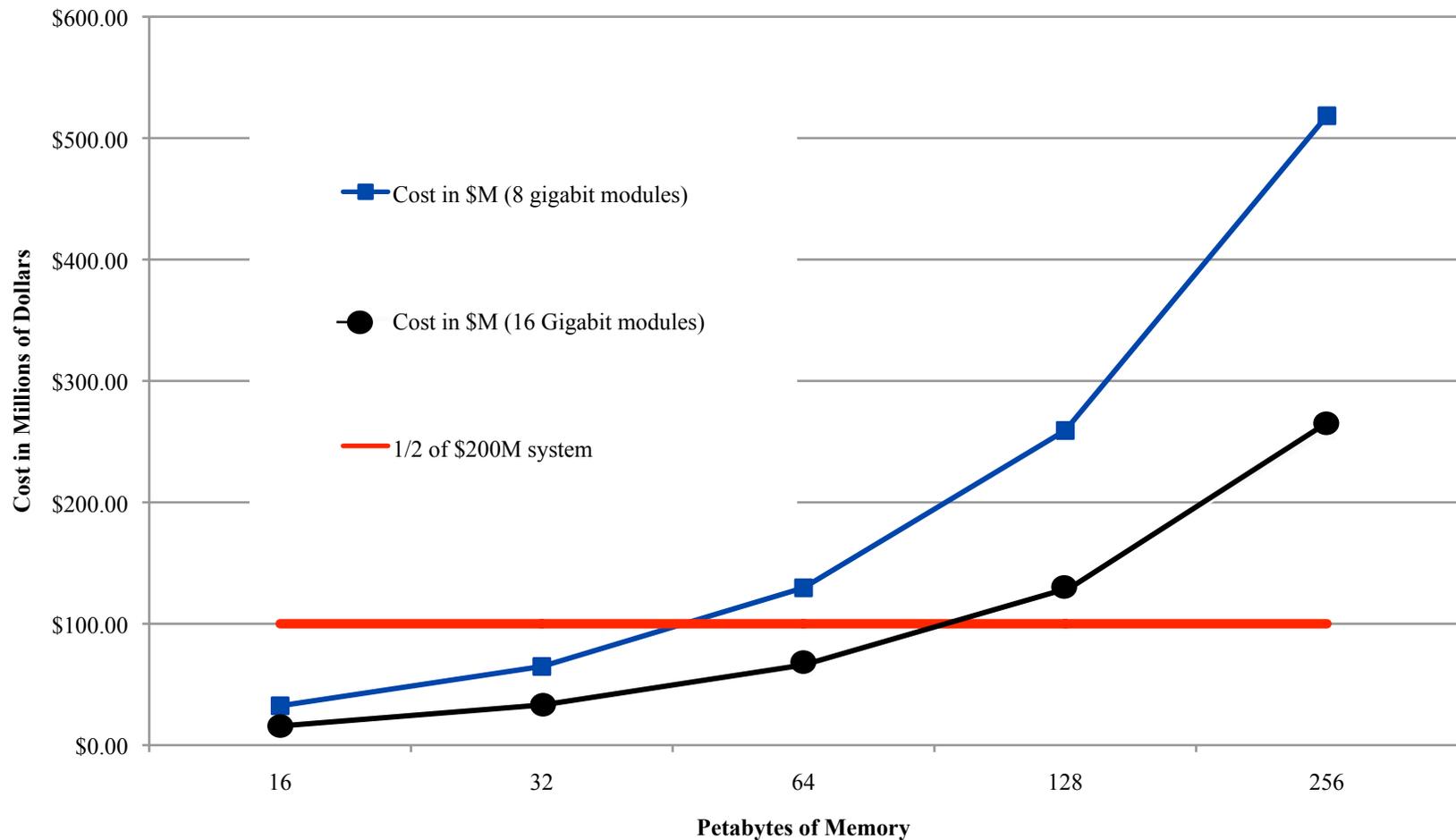
# Hurdle #1: memory bandwidth eats up the entire power budget



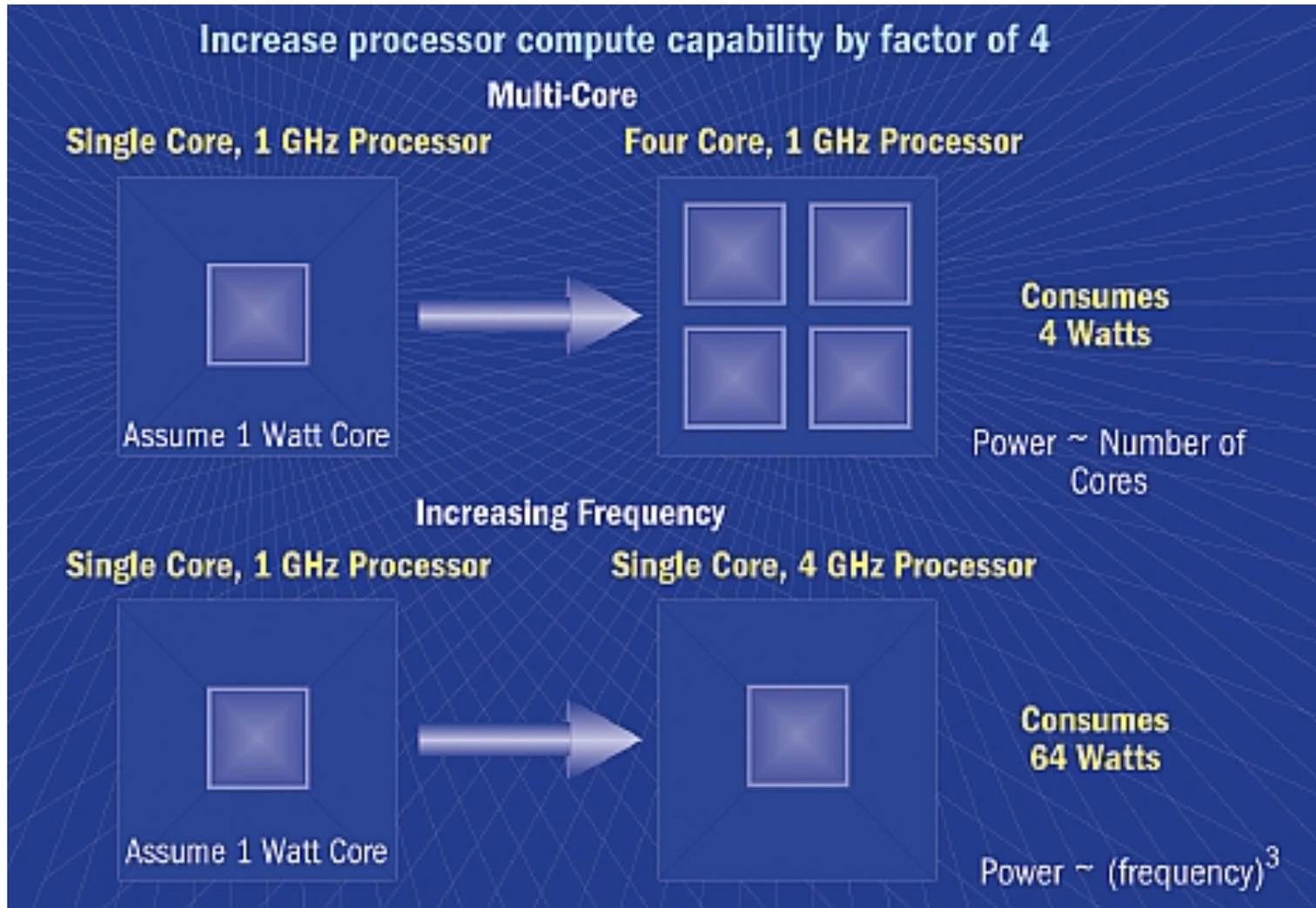
# This situation will not improve enough by 2018 to permit us to be profligate with memory



# Hurdle #2: memory capacity eats up the entire fiscal budget



# Hurdle #3: power requires slower clocks and greater concurrency



# Implications

- Expanding the number of nodes (processor-memory units) to  $10^6$  is *not* a serious threat to algorithms that lend themselves to well-amortized precise load balancing (like PDEs)
  - ◆ Provided that the nodes are performance reliable
- The real challenge is expanding the number of cores on a node to  $10^3$ 
  - ◆ Must be done while memory and memory bandwidth per node expand by (at best) ten-fold less
- It is already about  $10^3$  slower to retrieve an operand from main DRAM memory than to perform an arithmetic operation
  - will get worse by a factor of ten
  - ◆ Almost all operands must come from registers or upper cache



## Implications, cont.

- Draconian reduction required in power per flop and per byte will make computing and copying data less reliable
  - ◆ voltage difference between “0” and “1” will be reduced
  - ◆ circuit elements will be smaller and subject to greater physical noise per signal
  - ◆ there will be more errors that must be caught and corrected
- Power will have to be cycled off and on or clocks slowed and speeded based on compute schedules and based on cooling capacity
  - ◆ makes per node performance rate unreliable



# Sources of nonuniformity

- System
  - ◆ Manufacturing, dynamic power management, soft errors, hard component failures, OS jitter, software-mediated resiliency, TLB/cache performance variations, network contention, etc.
- Algorithmic
  - ◆ Physics at gridcell scale (e.g., table lookup, equation of state, external forcing), discretization adaptivity, solver adaptivity, precision adaptivity, etc.
- Effects are similar when it comes to waiting at synchronization points
- Possible solutions for system nonuniformity will improve programmability, too



## Implications, cont.



# Moans from application developers

- The present consensus path to exascale is thousand-fold manycore
  - ◆ However, memory bandwidth is already limiting today's low core count nodes to less than 10% of peak on most apps, whose kernels offer little cache reuse (e.g., stencil ops or sparse matvecs)
  - ◆ Processors are cheap and (relative to memory) small in chip area and relatively low in power, so there is no harm in having them in excess most of the time, but the opportunities for exploiting the main new source for performance are undemonstrated for most applications
- While there is opportunity for combining today's individually high capability simulations into complex simulations, there is no silver bullet for merging the data structures of the separate applications
  - ◆ The data copying inherent in the code coupling will likely prevent exploitation of the apparent concurrency opportunities



# Chief issues identified by apps groups of the International Exascale Software Project (IESP)

- I/O
- Fault tolerance
- Reproducibility of computations
- Programming models and algorithms



# I/O

- For some important apps, I/O is a likely bottleneck
  - ◆ For input, for output (including visualization), or for checkpointing, or any combination
- I/O must be acknowledged as primary for many apps (though certainly not all), but is beyond the scope of this talk



# Fault tolerance

- IESP users reluctantly recognize that fault tolerance is a shared responsibility
  - ◆ It is too wasteful of I/O and processing cycles to handle faults purely automatically
- Different types of faults may be handled different ways, depending upon consequences evaluated by scientific impact
- Strategic, minimal workingset checkpoints can be orchestrated by application developers and users



# Reproducibility

- IESP users realize that bit-level reproducibility is unnecessarily expensive most of the time
- Though scientific outcomes must be run- and machine-independent, we have no illusions about bit-level reproducibility for individual pairs of executions with the same inputs
  - ◆ Since operands may be accessed in different orders, even floating point addition is not commutative in parallel and on inhomogeneous hardware platforms; this has been true for a long time
  - ◆ A new feature, with an emphasis on low power (low voltage switching), is that lack of reproducibility may emerge for many *other* (hardware-based) reasons
  - ◆ If applications developers are tolerant of irreproducibility for their own reasons, e.g., for validation and verification through ensembles, then this has implications for considering less expensive, less reliable hardware



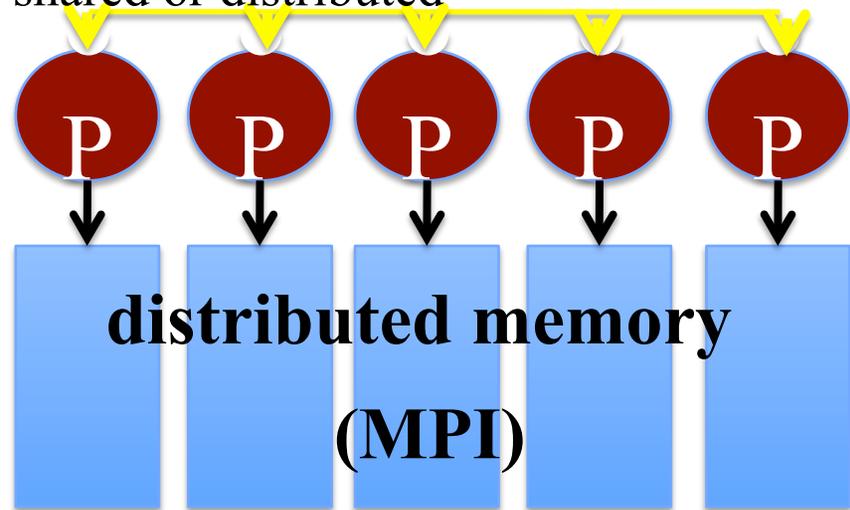
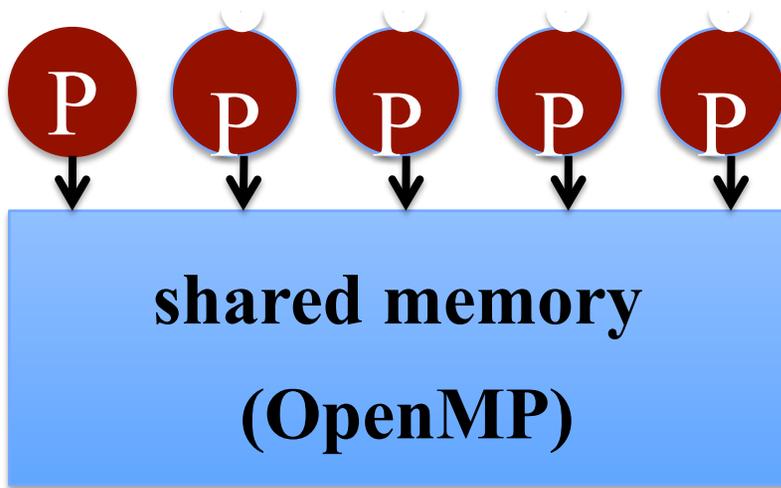
# Programming model

- Prior to possessing exascale hardware, users can prepare themselves by exploring new programming models
  - ◆ on manycore and heterogeneous nodes
- Attention to locality and reuse is valuable at all scales
  - ◆ will produce performance paybacks today *and* in the future
- New algorithms and data structures can be explored under the assumption that flop/s are cheap and moving data is expensive
- Bandwidth pressure can be reduced by considering mixed-precision algorithms, using lower precision wherever possible
- ~~Relaxation of synchrony could relieve pressure on load~~  
balance

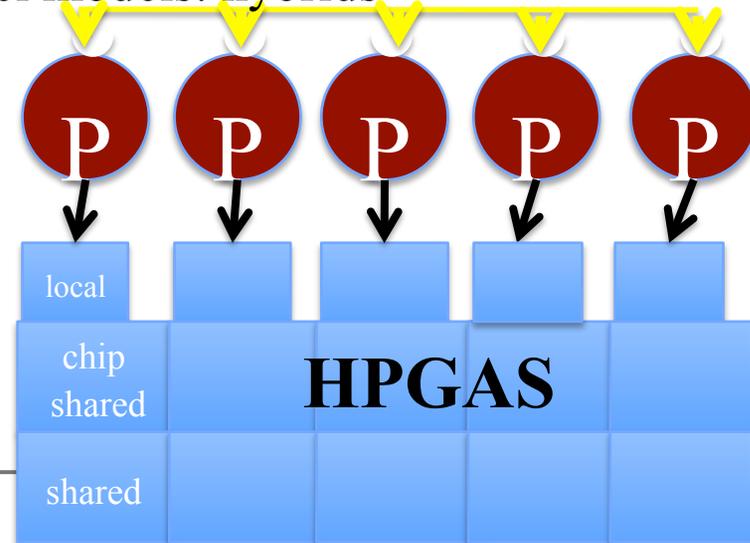
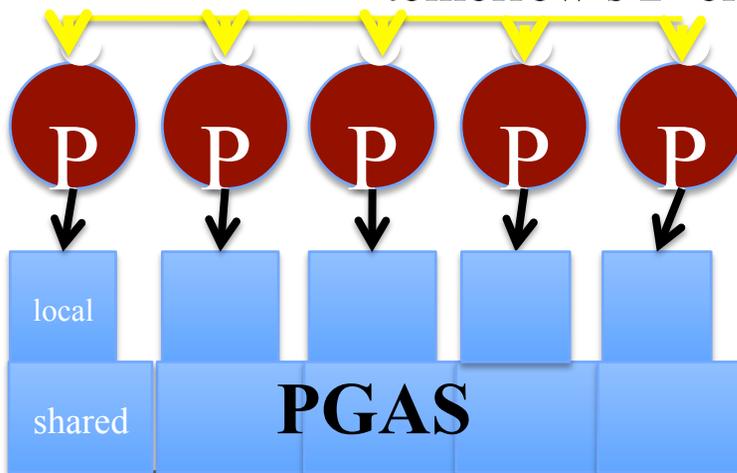


# Evolution of parallel programming models: strong scaling within a node

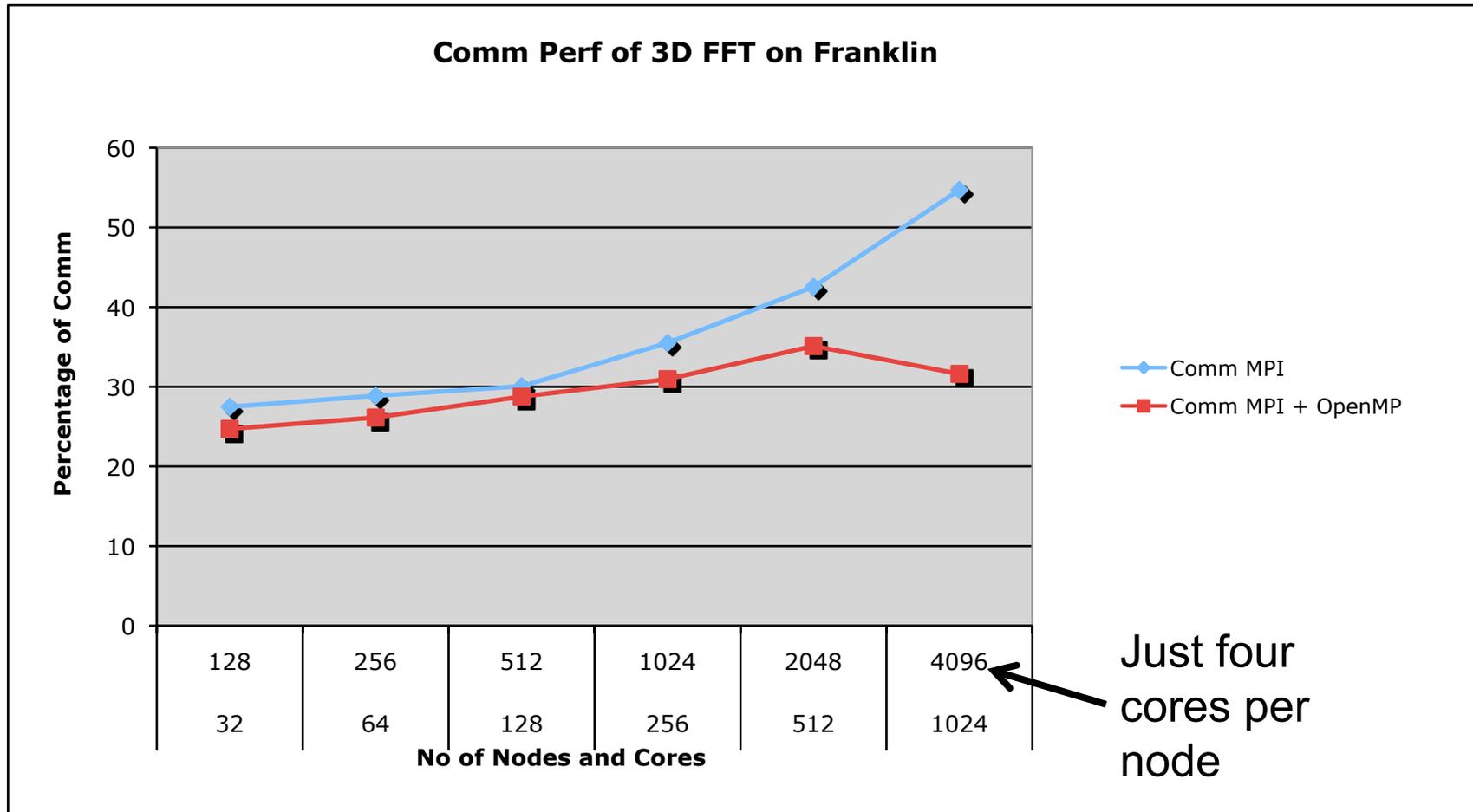
today's 1-level models: shared or distributed



tomorrow's 2- or 3-level models: hybrids



# Benefits of 2-level parallelism for 3D FFT



# Hybrid programming model for a full application: petascale bio-electro-magnetics

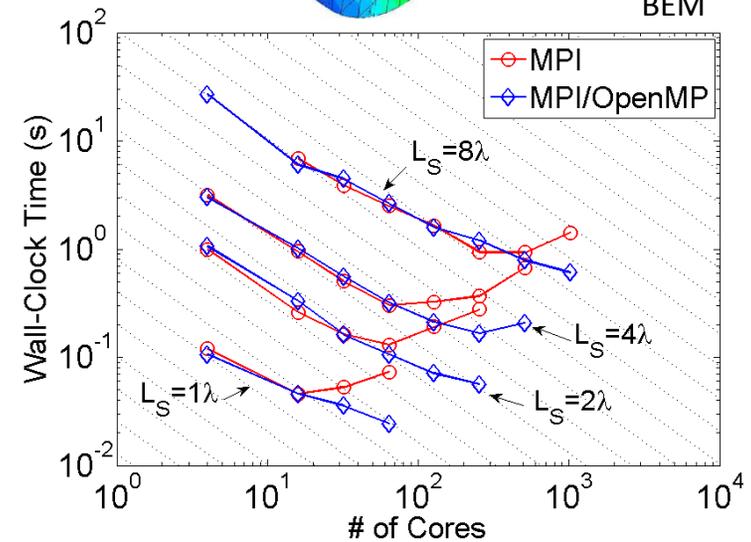
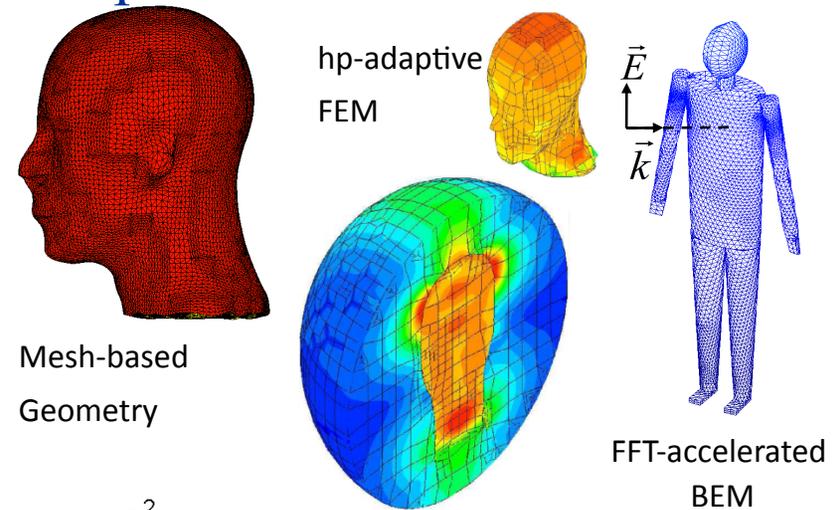
## High-Fidelity BIOEM Simulation

- ◆ Minimize health risks & increase efficiency of wireless devices
- ◆ Existing results contradictory, e.g., do children's smaller heads absorb more radiation or allow deeper penetration? Need reproducible, reliable, high-accuracy, high-resolution simulations
- ◆ Solve coupled Maxwell's EM and Penne's Bio-Heat Transfer equations : Human & device models must resolve geometry, material, EM wavelength, and thermal mechanisms (*petascale problem*)
- ◆ Advance the state-of-the-art: Develop novel petascale BIOEM simulators, investigate and quantify modeling & analysis errors

## Two-pronged Approach (FEM/BEM)

- ◆ Cross-validate, verify, & build confidence in unprecedented petascale results
- ◆ Evolve sub-mm resolution human body models
- ◆ Next generation EM solvers: FFT-accelerated integral-equation solvers (BEM prong); HP-adaptive differential-equation solvers (FEM prong); novel preconditioned iterative solvers
- ◆ Target multi-core clusters: Nested distributed-/shared-memory parallelism and hybrid programming (MPI/OpenMP)
- ◆ High-fidelity petascale BIOEM simulations
- ◆ Shed light to controversial scientific and engineering questions

## Samples



# Hybrid programming models not enough

- Tools for monitoring the availability and predicted performance of resources within an architecture-adaptive and application-adaptive are improving
- However, even perfect knowledge of resource capabilities at every moment and perfect load balancers will not rescue billion-thread SPMD implementations of PDE simulations, etc.
  - ◆ cost of rebalancing frequently is too large
  - ◆ Amdahl penalty of failing to rebalance is fatal



# Evolution of parallel programming models: breaking the synchrony stronghold

- Can write code in styles that do not require artifactual synchronization
- Critical path of a nonlinear implicit PDE solve is essentially  
... lin\_solve, bound\_step, update; lin\_solve, bound\_step, update ...
- However, we often insert into this path things that could be done less synchronously, because we have limited language expressiveness
  - ◆ Jacobian and preconditioner refresh
  - ◆ Convergence testing
  - ◆ Algorithmic parameter adaptation
  - ◆ I/O, compression
  - ◆ Visualization, data mining



# Adaptation to asynchronous programming styles

- To take full advantage of such asynchronous algorithms, we need to develop greater expressiveness in scientific programming
  - ◆ Create separate threads for logically separate tasks, whose priority is a function of algorithmic state, not unlike the way a time-sharing OS works
  - ◆ Join priority threads in a directed acyclic graph (DAG), a task graph showing the flow of input dependencies; fill idleness with noncritical work or steal work
- Steps in this direction
  - ◆ Asynchronous Dynamic Load Balancing (ADLB) [Lusk (Argonne), 2009]
  - ◆ Asynchronous Execution System [Steinmacher-Burrow (IBM), 2008]



# Algorithmic adaptation to asynchronous programming styles

- Additive versions of algorithms are often available that significantly relax synchronicity
- Such algorithms have received a bad rap historically
  - ◆ “chaotic relaxation,” Chazan & Miranker, 1969, for instance
- However, they can sometimes be made virtually as good as their multiplicative cousins
  - ◆ “AFACx” versus “AFAC,” Lee, McCormick, Philip & Quinlan, 2003, for instance



# Synchronization reducing algorithms will be a week-long “Hot Topic” at ICERM, 2011 ...

Brown University



Media Relations • PUBLIC AFFAIRS AND UNIVERSITY RELATIONS

## Brown University News

PRESS RELEASES

FEATURED EVENTS

TODAY AT BROWN

Working with  
Media Relations

MEDIA STAFF

TV STUDIO/ISDN LINE

## SEARCH

BEFORE 2008

August 5, 2010 | Contact: Richard Lewis | (401) 863-3766 | Print

## Brown University awarded \$15.5-million math institute

Brown University has been awarded \$15.5 million to create a national mathematics research institute. The Institute for Computational and Experimental Research in Mathematics is the eighth math institute in the United States funded by the National Science Foundation and the only one of its kind in New England. The institute will bring in leading scholars from around the world to explore the frontier of mathematics and computation.

**PROVIDENCE, R.I.** [Brown University] — Brown University has been awarded \$15.5 million to form a national mathematics research institute, where scholars will explore the intersection of computation and mathematics, the University announced today.

The award funds the Institute for Computational and Experimental Research in Mathematics and comes from



**Jill Pipher**

Professor of Mathematics  
Director, Institute for Computational and  
Experimental Research in Mathematics

*Credit: Mike Cohea/Brown University*

PASI, 4 Jan 2011



# Peta to exa for algorithms

- Things we need to do for exascale will help us at petascale and terascale
  - ◆ Reducing memory requirements and memory traffic
  - ◆ Exploiting hybrid and less synchronous parallel programming models
  - ◆ Co-design of hardware and software (for, e.g., power management)
- Though it inveighs against the CS aesthetic of “separation of concerns”, and involves more issues, co-design requires similar attitude and aptitude as in, say, MPI programming today
  - ◆ Applications programmers have “bit the bullet” and designed excellent MPI-based codes, by using quality libraries designed and ported by specialists
  - ◆ Hopefully, we will be able to isolate applications programmers from many of the hardware and software architectural details, just as we do today from message-passing details



## Peta to exa

- Billion-way parallelism of GigaHertz cores will not significantly expand today's million-way flat parallelism at the node level
  - ◆ MPI legacy code will still be usable on the “outside” on a million nodes
  - ◆ Changes will be mainly *within* a node, where we will need to evolve thousand-way parallelism: “MPI+X”
- Principal challenges from peta to exa are within the node, and the burden is *shared by the marketplace* at all scales of node aggregation



## Peta to exa

- Loosely speaking, the algorithmic path from peta to exa preserves the focus on weak scaling between nodes and adds (mostly) strong scaling within nodes
  - ◆ The big challenge is the strong scaling within the nodes
- The synchronization cost of global reductions in implicit methods propagates inside the node with a much smaller coefficient
  - ◆ Hierarchical reductions
- Dynamic load balance through “work stealing” is sufferable since remote DRAM access is no longer much slower than local DRAM access



## Some major challenges that stay the same in peta to exa

- Poor scaling of collectives due to internode latency
- Poor performance of SpMV due to shared intranode bandwidth
- Poor scaling of coarse grids due to insufficient concurrency
- Lack of reproducibility due to floating point noncommutativity and algorithmic adaptivity (including autotuning) in efficient production mode
- Difficulty of understanding and controlling vertical memory transfers



# Exascale drivers

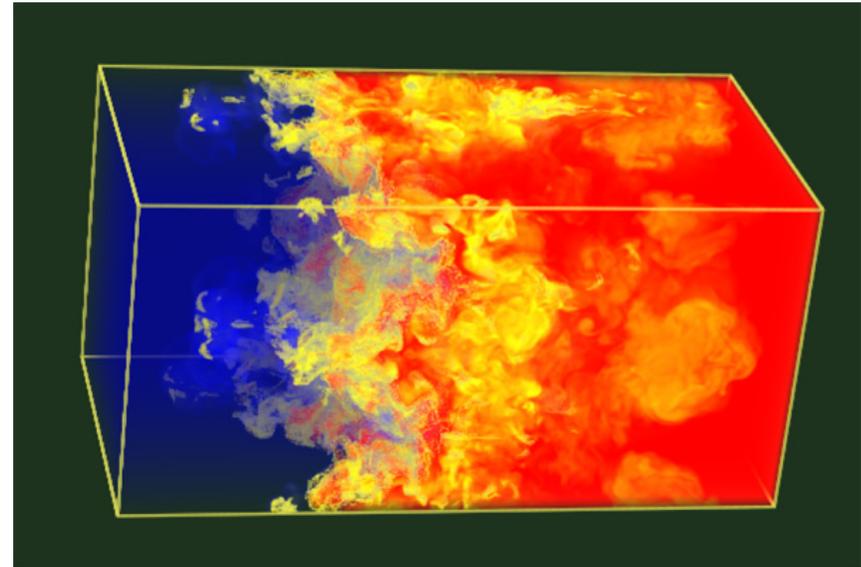
- *Climate, combustion, subsurface flow, nanoscience, and other applications* depend algorithmically upon a fairly common core of mathematical formulations
  - mainly partial differential equations (PDEs) and molecular dynamics (MD)
- These are multiscale-, multiphysics-based problems
  - *“Problems that are not multiscale or multiphysics have already been solved.”* – John Bell, LBNL
- Typical formulations of these problems share a feature that work requirements grow faster than memory requirements
  - This allows memory (Bytes) to grow more slowly than operations (Flops), for cost- and power-feasible exascale hardware design

## Exascale drivers, cont.

- **Three motivations**
  - **Intrinsic interest in exascale for the individual codes (“capability”)**
  - **Additional interest in their combination (“complexity”)**
  - **Still further interest in solving inverse problems, sensitivity analysis, and UQ (“understanding”)**

# Capability – multiple scales

- **Multiple spatial scales**
  - interfaces, fronts, layers
  - thin relative to domain size,  $\delta \ll L$
- **Multiple temporal scales**
  - fast waves
  - small transit times relative to convection or diffusion,  $\tau \ll T$
- Analyst must first *isolate dynamics of interest* and *model the rest* in a system that can be discretized over modest range of scales
- Often involves filtering of high frequency modes, quasi-equilibrium assumptions, etc.
- May lead to infinitely “stiff” subsystem requiring implicit treatment
- Resulting implicit subsystem may be very ill-conditioned



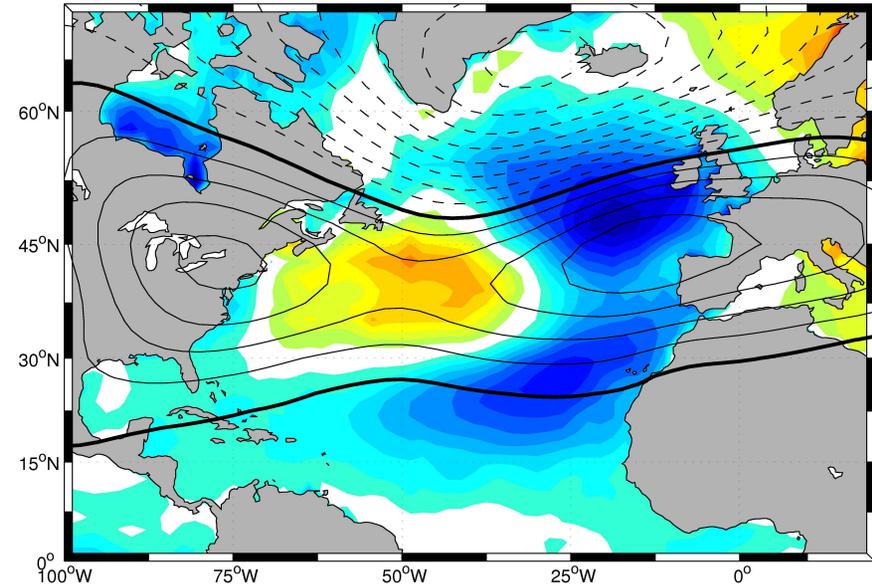
● Richtmyer-Meshkov instability, c/o A. Mirin, LLNL

# Complexity – multiple components

- **Interfacial coupling examples**

- **Ocean-atmosphere coupling in climate**
- **Core-edge coupling in tokamaks**
- **Fluid-structure vibrations in aerodynamics**
- **Boundary layer-bulk phenomena in fluids**
- **Surface-bulk phenomena in solids**

- **Coupled systems may admit destabilizing modes not present in either system alone**

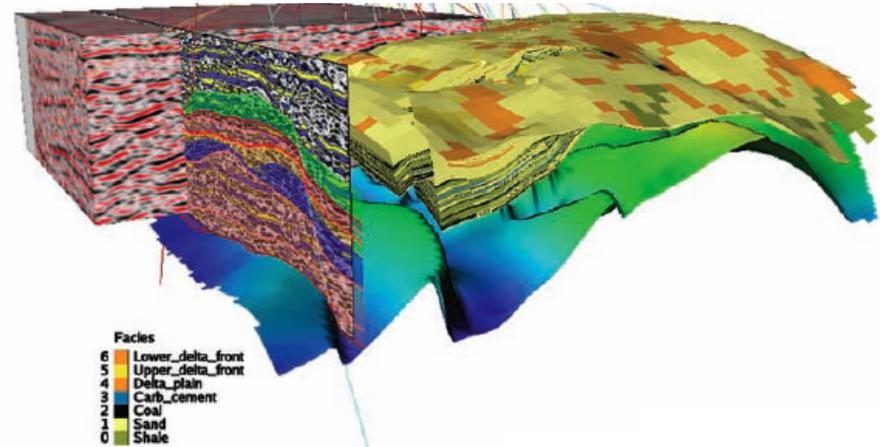


- **Bulk-bulk coupling examples**

- **Radiation-hydrodynamics**
- **Magneto-hydrodynamics**

# Understanding – multiple parameters

- Subsurface contaminant transport and petroleum recovery
- Climate prediction
- Medical imaging
- Stellar dynamics, e.g., supernovae
- Waves in inhomogeneous media
- Nondestructive evaluation of bridges, other structures
- Sensitivity, optimization, parameter estimation, boundary control all require the ability to apply the inverse action of the Jacobian of the system – a capability present in all Newton-like implicit methods



● *Subsurface property estimation, c/o Roxar*

- Uncertainty can be in
  - constitutive laws
  - initial conditions
  - boundary conditions

# Path for scaling up applications

- Weak scale applications up to distributed memory limits
  - ◆ Proportional to number of nodes
- Strong scale applications beyond this
  - ◆ Proportional to cores per node/memory unit
- Scale the workflow, itself
  - ◆ Proportional to the number of instances (ensembles)
  - ◆ Integrated end-to-end simulation
- Co-design process is staged, with any of these types of scaling valuable by themselves
- Big question: does the software for co-design factor? Or is all the inefficiency at the data copies at interfaces between the components after a while?



# Limitations to be explored by early apps

- Strong scaling algorithms
  - ◆ sufficient coarse grain parallelism but
  - ◆ expect load imbalance to be the main problem due to irregular task/data size
  - ◆ bulk synchronous algorithms on one million nodes do not tolerate load imbalance worse than one part per million for a synchronous task
- Single node performance
  - ◆ Compiler-generated code for hybrid/multicore
  - ◆ linear algebra kernels
    - typically come with autotuning
  - ◆ nonstandard linear algebra kernels
    - we need the autotuning tools, not just their output



## Needs to be explored by early apps

- Tools to generate domain-specific languages and for powerful source-to-source transformations
  - ◆ to enhance composability
    - want to enable new science and expand developer and user communities, so we must \*decrease\* complexity as we go to exascale
  - ◆ for writing performance-portable code (retargetable)
  - ◆ to extend effective lifetime of code over generations of hardware
  - ◆ to implement domain-specific frameworks
    - frameworks are widely perceived as successful solutions of many HPC problems
    - the future is increasingly multidisciplinary, so these must interoperate



## Needs to be explored by early apps, cont.

- Expanded/different programming models
  - ◆ move more of the burden of managing scheduling of computation and placement of data to runtime
  - ◆ expand intrinsically fault tolerant programming models to be relevant to a broader class of algorithms
  - ◆ interoperability of programming models (GAS, MPI, Cilk, HPCS, etc.)
- Understanding the design space
  - ◆ tradeoffs associated with options for power consumption and resilience
  - ◆ the nature of expected faults, including common signaled faults and especially silent faults



# Criteria for early apps

- Application running at at least the terascale today, with obvious need for more
- In progressing to the exascale, should be able to achieve scientific goals in its own domain
  - ◆ amenability to experimental validation
- Should possess a well-defined set of steps to progress to exascale
- Should have a community supporting the application that has the skills and experience to engage with the co-design process
- Has to be open and should ideally spawn modules for common use
- Overall portfolio should attempt span application space



# Exascale apps candidates

- Multiscale, multiphysics problems
- Problems that have a superlinear scaling of work with memory capacity
  - ◆ for cost- and power-feasible exascale hardware design, which is memory- and communication bandwidth-limited



# Required software enabling technologies

## Model-related

- ◆ Geometric modelers
- ◆ Meshers
- ◆ Discretizers
- ◆ Partitioners
- ◆ Solvers / integrators
- ◆ Adaptivity systems
- ◆ Random no. generators
- ◆ Subgridscale physics
- ◆ Uncertainty quantification
- ◆ Dynamic load balancing
- ◆ Graphs and combinatorial algs.
- ◆ Compression

## Development-related

- ◆ Configuration systems
- ◆ Source-to-source translators
- ◆ Compilers
- ◆ Simulators
- ◆ Messaging systems
- ◆ Debuggers
- ◆ Profilers

High-end computers come with little of this stuff. Most has to be contributed by the user community

## Production-related

- ◆ Dynamic resource management
- ◆ Dynamic performance optimization
- ◆ Authenticators
- ◆ I/O systems
- ◆ Visualization systems
- ◆ Workflow controllers
- ◆ Frameworks
- ◆ Data miners
- ◆ Fault monitoring, reporting, and



# Algorithmic Priority Research Directions (1)

- Advanced mathematical methods for scientific understanding in exascale simulations, including *in situ*
  - ◆ Uncertainty quantification, intrusive and nonintrusive
  - ◆ Optimization, inverse problems, sensitivity
  - ◆ Analysis and Visualization
  - ◆ Validation and Verification



## Algorithmic Priority Research Directions (2)

- Exascale algorithms that expose and exploit multiple levels of parallelism
  - ◆ Communication-reducing algorithms
  - ◆ Synchronization-reducing algorithms
  - ◆ Fault resilient algorithms
- Support for multiphysics, multiscale methods
  - ◆ Break the SPMD and BSP paradigms when joining multiple different codes
  - ◆ Stability of coupling



## Algorithmic Priority Research Directions (3)

- Exascale algorithms for constructing and adapting discrete objects
  - ◆ Algorithms that deal with unpredictable, dynamic workloads and have few flops to hide
- Mixed precision arithmetic, to use the lowest precision required to achieve a given accuracy outcome
  - ◆ Improves runtime, power consumption
  - ◆ Reformulate algorithms to find corrections, rather than solutions



## Progressive by-product

- The consolation for the architecture-induced hard work of reducing synchrony is that algorithms have been waiting for this freedom for a long time
  - ◆ freedom to adapt the mesh or vary the timestep locally
  - ◆ freedom to vary the physical models at the cells or particle level
  - ◆ Freedom to vary the precision
- Once the synchronization is thrown on the programming model and runtime system, developers are less constrained



# The wrong reward

- Since a flop is by far cheaper to provision and to power up, relative to memory and memory transfer rate, good designs over-provision flops, so that they are never limiting
  - ◆ Percentage of peak is therefore a counter-productive metric, since it penalizes for over-provisioning something cheap
  - ◆ Percentage of the instantaneously limiting resource, typically memory bandwidth, is the most interesting figure of merit for a given execution of the “right problem”
  - ◆ The “right problem” is the one that provides the requisite accuracy with the requisite confidence at some combination of lowest energy and shortest execution time



# Summary:

## High reward R&D themes for algorithms

- Mixed precision arithmetic, to use the lowest precision required to achieve a given accuracy outcome
  - ◆ Improves runtime, power consumption
  - ◆ Reformulate algs to find corrections, rather than solutions
- Prioritization of critical path and noncritical tasks
  - ◆ DAG scheduling of critical path tasks
  - ◆ Allows taking advantage of asynchronicity between major steps and adaptive load balancing for noncritical tasks
- Communication-reducing algorithms
- And, of course, better mathematical formulations



# Where have you seen these strategies before?

- Maximize parallel execution
  - ◆ Expose as much data parallelism and data reuse as possible
  - ◆ Map algorithm to hardware as efficiently as possible
  - ◆ Maximize concurrent communication and computation
- Maximize memory bandwidth
  - ◆ Minimize data transfers, especially access to main memory
  - ◆ Consider recomputing rather than storing and retrieving
  - ◆ Optimize memory layout so transferred blocks are fully used
- Maximize instruction throughput
  - ◆ Avoid low-throughput arithmetic instructions
  - ◆ Trade precision for speed when precision doesn't matter
  - ◆ Avoid variable control flows wherever possible



# Reminder about the source of simulations

- Computational science and engineering is not about individual large-scale analyses, done fast and “thrown over the wall”
- Both “results” and their sensitivities are desired; often multiple operation points to be simulated are known *a priori*, rather than sequentially
- Sensitivities may be fed back into optimization process
- Full PDE analyses may also be inner iterations in a multidisciplinary computation
- In such contexts, “exaflop/s” may mean 1,000 analyses running somewhat asynchronously with respect to each other, each at 1 PF/s – clearly a less daunting challenge than 1 analysis running at 1 EF/s



# Recapitulation

- Reflected briefly on progress in high-end scientific computing
- Peeked briefly at some motivating applications
- Looked generically at PDE-based simulation and the basis of continued optimism for its growth – capability-wise
- Looked at some specific hurdles to and opportunities for PDE-based simulation posed by high-end architecture



## Kennedy's Challenge, 1962



“We choose to do [these] things, not because they are easy, but because they are hard, *because that goal will serve to organize and measure the best of our energies and skills,* because that challenge is one that we are willing to accept, one we are unwilling to postpone, and one which we intend to win...”



# Acknowledgment: today's Peta-op/s machines



$10^{12}$  neurons @ 1 KHz = 1 PetaOp/s



# Zetta-scale, anyone?



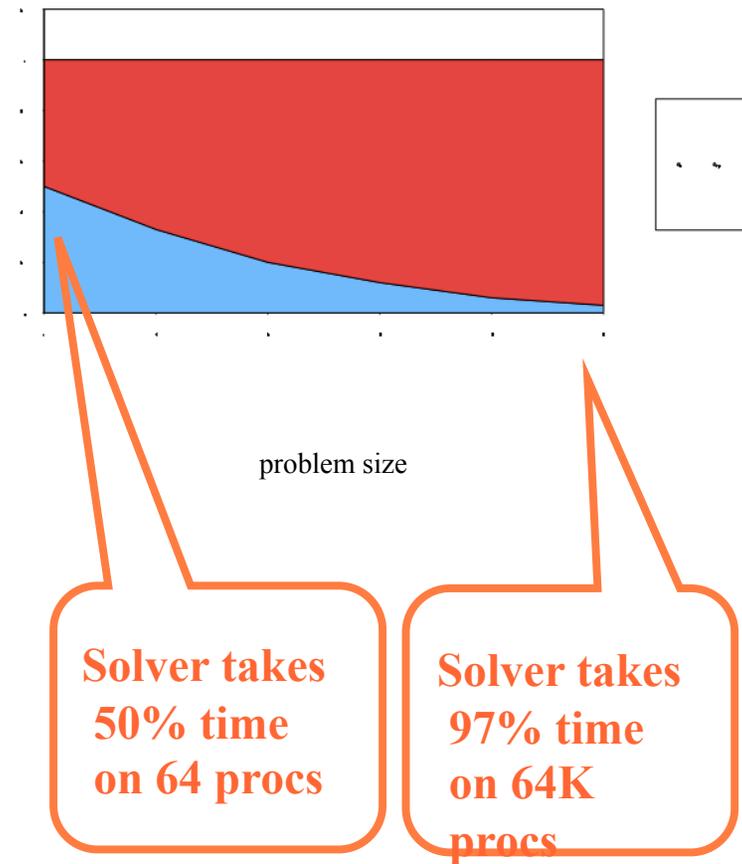
# Extra slides



# It's *all* about the solver (at the ultrascale)

- Given, for example:
  - ◆ a “physics” phase that scales as  $O(N)$
  - ◆ a “solver” phase that scales as  $O(N^{3/2})$
  - ◆ computation is almost all solver after several doublings
- Most applications groups have not yet “felt” this curve in their gut
  - ◆ this is changing

Weak scaling limit, assuming efficiency of 100% in both physics and solver phases



# Some noteworthy algorithmic adaptations to distributed memory architecture

- Multi-step relaxation and Krylov schemes (long history + Demmel'07)
  - ◆ Exchange several steps worth of neighbor data at once – replaces many small messages with one large (requires extra computing to preserve stability)
- Restricted Schwarz (Cai & Sarkis)
  - ◆ omit every other local communication (actually leads to *better* convergence, now proved)
- Extrapolated Schwarz (Garbey & Tromeur-Dervout)
  - ◆ hide interprocessor latency by extrapolating messages received in time integration, with rollback if actual messages have discrepancies in lower Fourier modes (higher mode discrepancies decay anyway)
- Nonlinear Schwarz (Cai & Keyes)
  - ◆ reduce global Krylov-Schwarz synchronizations by applying NKS within well-connected subdomains and performing *few* global outer Newton iterations (interchange of loops, move synchronization outside)
- Aggressive coarsening in AMG (Falgout, Yang, et al.)
  - ◆ reduce size of coarse problems to trade-off cost per iteration with number of iterations (and many other such preconditioner quality ideas)



# Some noteworthy algorithmic adaptations to hierarchical memory architecture

- ATLAS/Sparsity (Whalley & Dongarra, Demmel & Yelick)
  - ◆ block (and and selectively fill and reorder for sparse) for optimal cache performance of linear kernels
- Block-vector Krylov methods (Baker *et al.*)
  - ◆ amortize the unavoidable streaming of large sparse Jacobian through cache over several matrix-vector multiplies
- Block relaxation methods (Ruede, Stals, Douglas)
  - ◆ similar to above, but for triangular backsolves
- Reduced precision preconditioning (Smith *et al.*)
  - ◆ double effective bandwidth by truncating precision of already approximate operators



# Often neglected algorithmic possibilities for more scalability

- Parallelization in the time (or generally causal) dimension, particularly in nonlinear problems after spatial concurrency is exhausted
  - ◆ Converts a time-like variable to space-like
  - ◆ Unlikely to be less fruitful in future architectural balances, since this concurrency comes at the expense of memory
- Creating independent ensembles for asynchronous evaluation (parameter exploration or stochastic model) after space-time concurrency is exhausted on the direct problem
- Trading finely resolved discretizations (very sparse) for higher-order discretizations (block dense), or other algorithmic innovations that alter the granularity of bulk synchronous work between data movements

