The Evolution of Programming Models in Response to Energy Efficiency Constraints

John Shalf

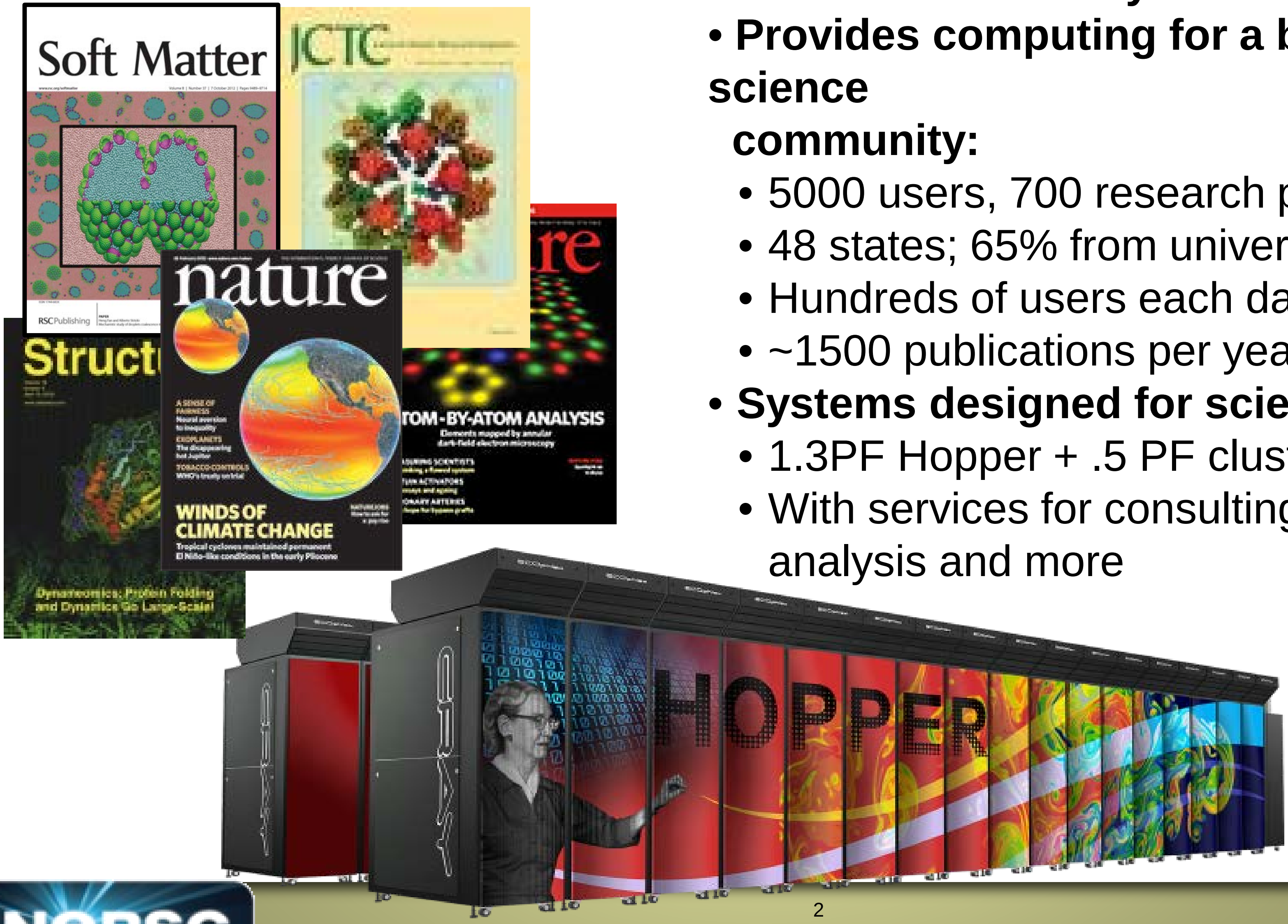
Department Head: Computer Science and Data Sciences (CSDS)

CTO: National Energy Research Scientific Computing Center (NERSC)

Oklahoma Supercomputing Symposium 2013

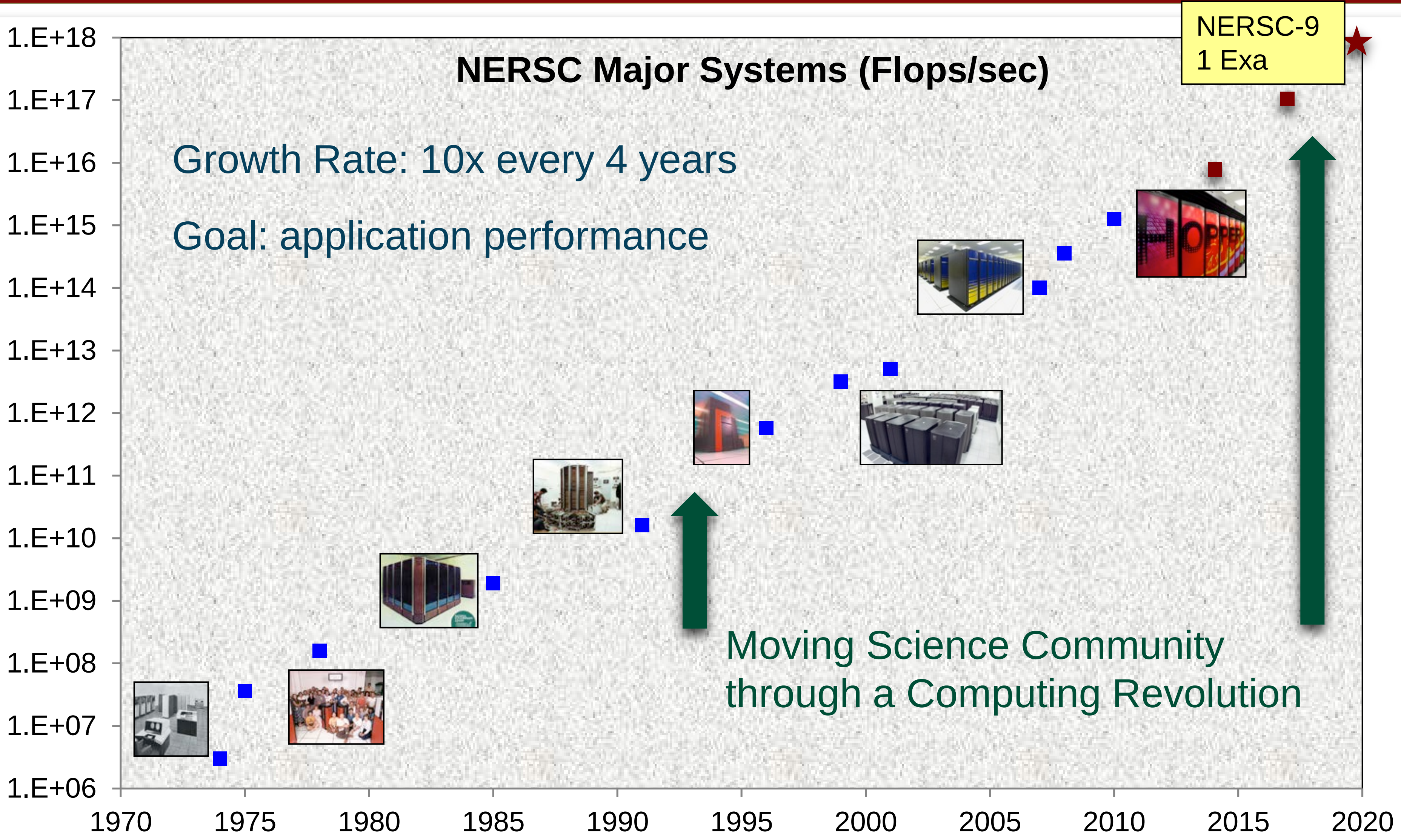
October 2, 2013

National Energy Research Scientific Computing Center (NERSC)


Soft Matter
RSC Publishing
JCTC
nature
A SENSE OF FAIRNESS
EXOPLANETS
TOBACCO CONTROLS
WINDS OF CLIMATE CHANGE
Tropical cyclones maintained permanent El Niño-like conditions in the early Pliocene
Struct
Dynameomics: Protein Folding and Dynamics Go Large-Scale!
re
ATOM-BY-ATOM ANALYSIS
Elements mapped by annular dark-field electron microscopy
HOPPER
CRAY

- **Located at Berkeley Lab**
- **Provides computing for a broad science community:**
  - 5000 users, 700 research projects
  - 48 states; 65% from universities
  - Hundreds of users each day
  - ~1500 publications per year
- **Systems designed for science:**
  - 1.3PF Hopper + .5 PF clusters
  - With services for consulting, data analysis and more

NERSC's 40 Year History of Production High Performance Computing

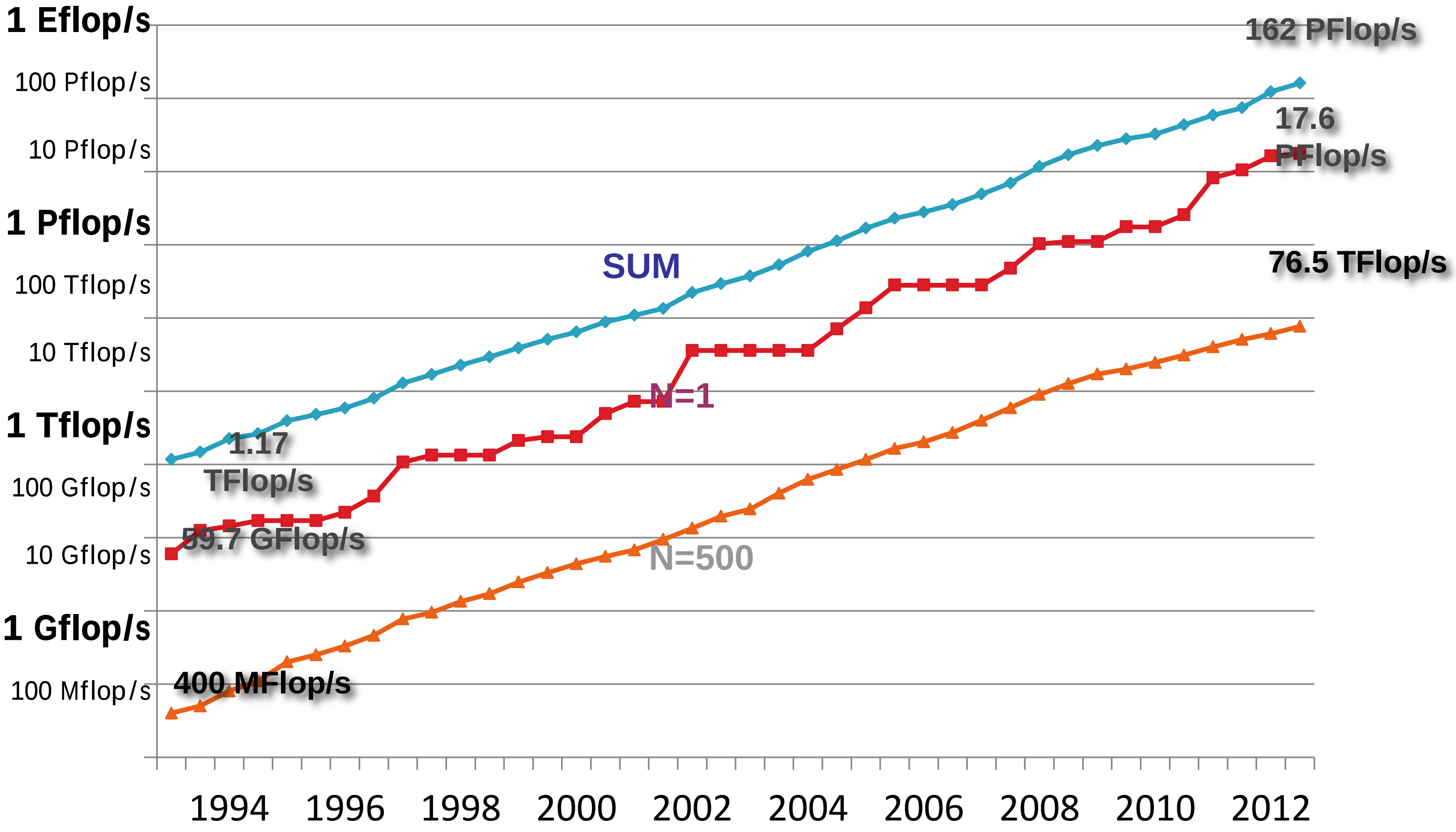


Growth Rate: 10x every 4 years

Goal: application performance

Moving Science Community through a Computing Revolution

30 Years of Exponential Performance Improvement



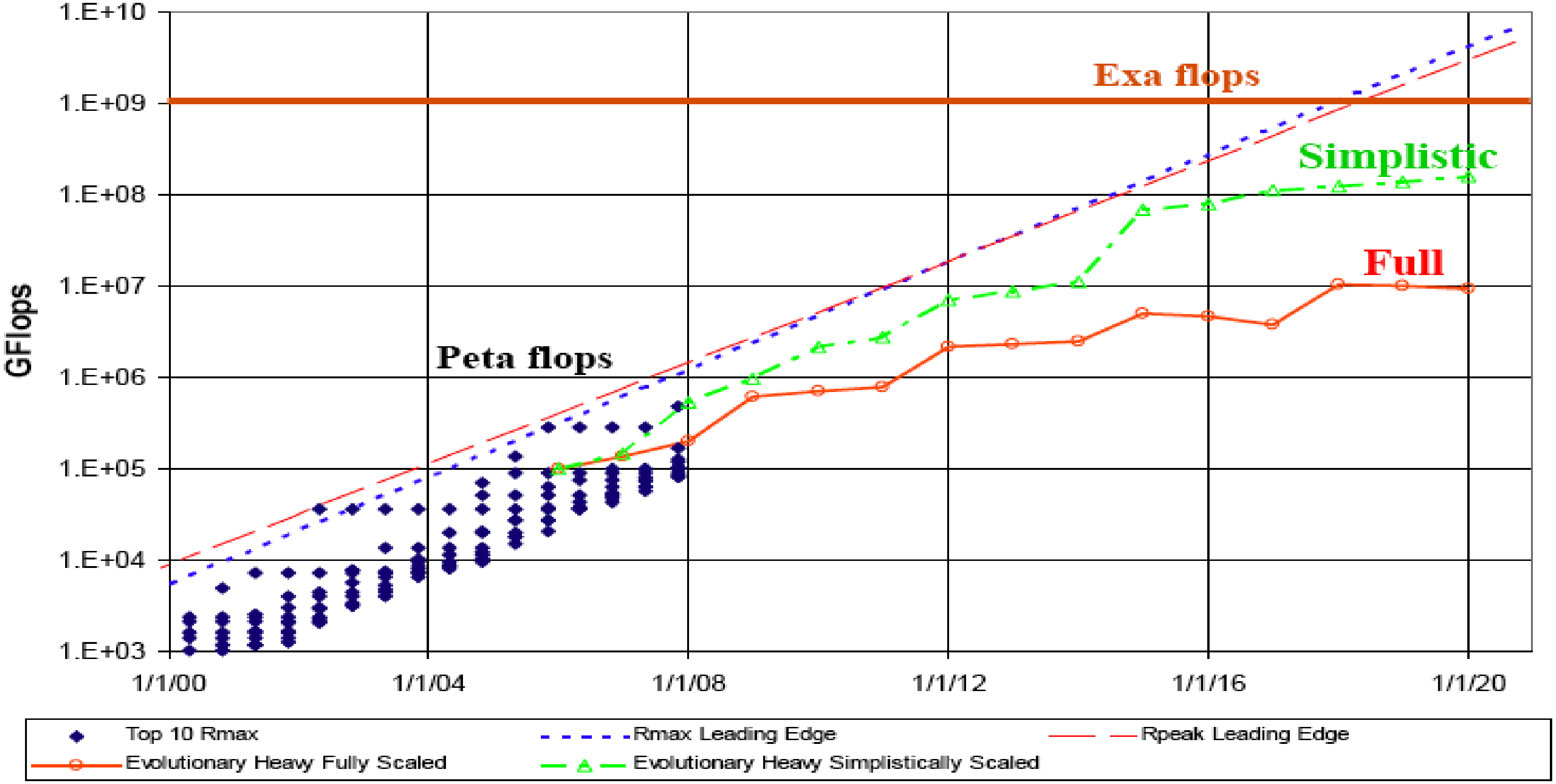
Source: TOP500 November 2012

But Its Going to Be Very Hard to Continue!

(Requires a Laboratory-Wide Strategy for Next Decade)

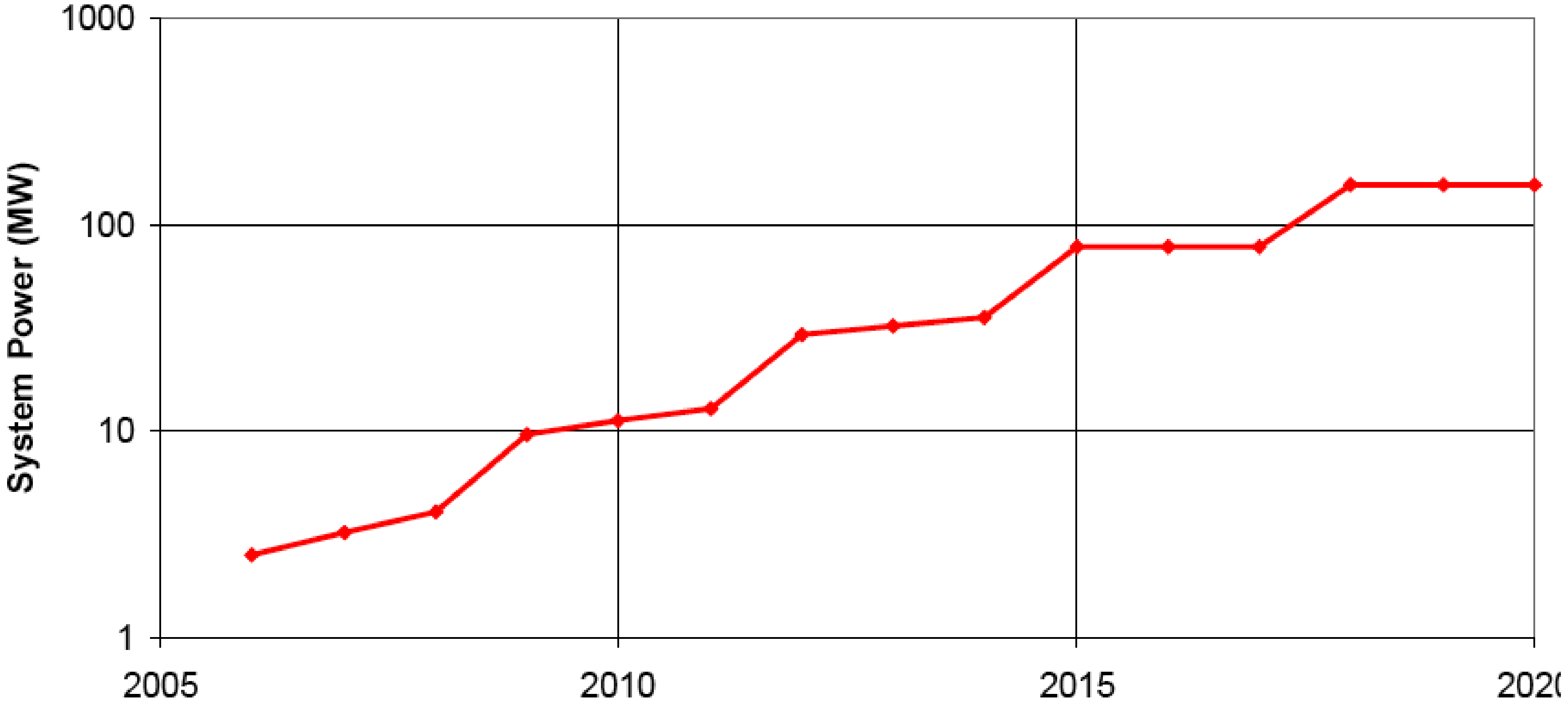
From Peter Kogge, DARPA Exascale Study

Current Technology Roadmaps will Depart from Historical Performance Gains



Without major changes, computing cannot continue historical trends of performance improvement

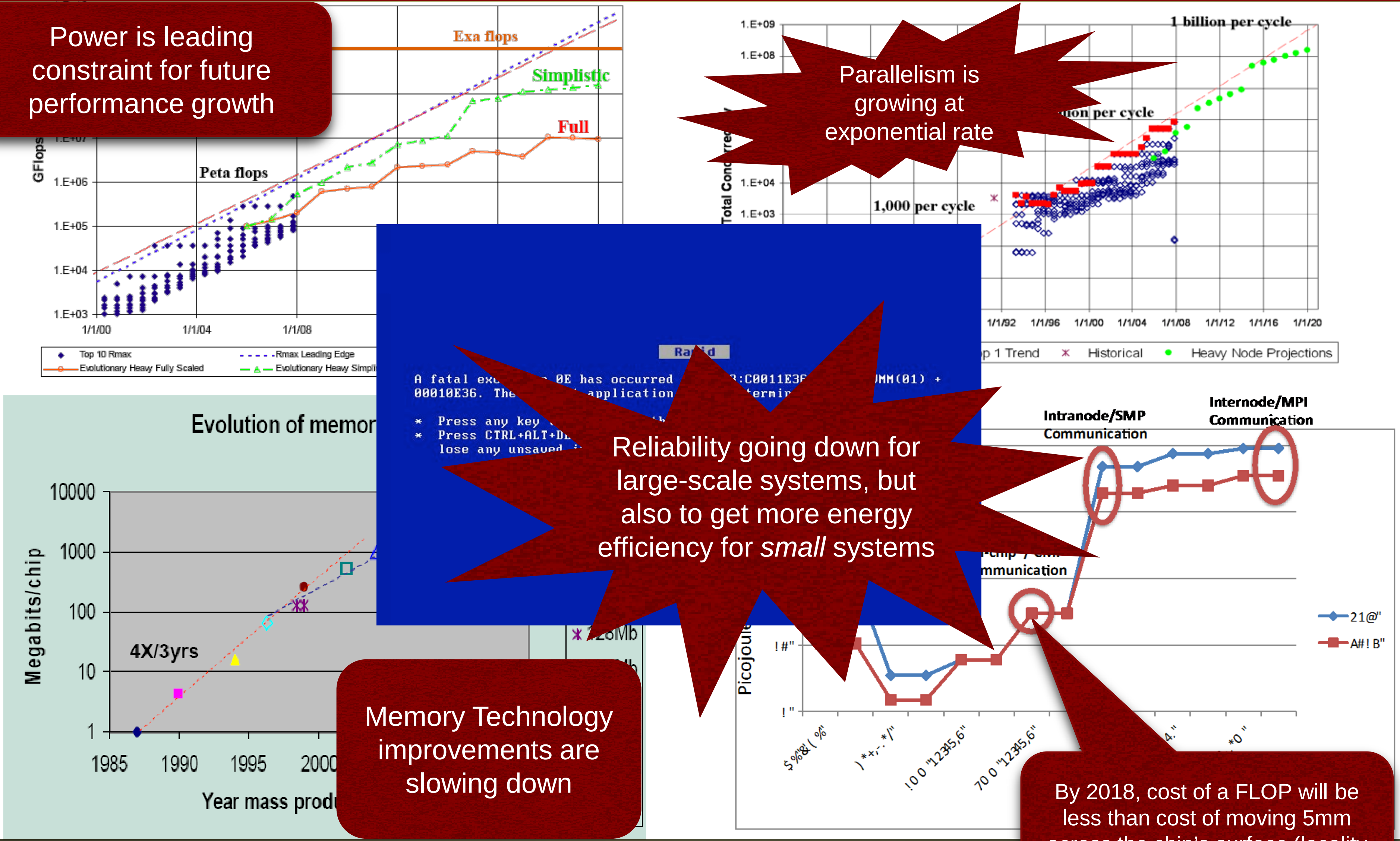
… and the power costs will still be staggering



From Peter Kogge,
DARPA Exascale Study

$1M per megawatt per year! (with CHEAP power)

Technology Challenges for the Next Decade



Power is leading constraint for future performance growth

Parallelism is growing at exponential rate

Reliability going down for large-scale systems, but also to get more energy efficiency for *small* systems

Memory Technology improvements are slowing down

By 2018, cost of a FLOP will be less than cost of moving 5mm across the chip's surface (locality will *really* matter)

These Trends Affect ALL users at ALL SCALES

*(this is **NOT** just about exascale)*

The disruptions are primarily within the node

- Only resilience and interconnect scaling are exclusively HPC
- Exponential growth of parallelism, power, and memory trends have pervasive impact on computing for the coming decade

Worse yet, the changes are already underway!

- There is no need point in waiting for the "ExaFLOP computer".
- These trends are happening NOW!

The Challenge of our Decade: *Performance growth in fixed power budget*

- The challenge is as dramatic as transition from vector to MPP
- This transition affects *all* computing for science from smallest to the largest scale
- Fundamentally breaks our software infrastructure (need to re-architect)

Top 10 Systems in November 2012

#	Site	Manufacturer	Computer	Country	Cores	Rmax [Pflops]	Power [MW]
1	Oak Ridge National Laboratory	Cray	Titan Cray XK7, Opteron 16C 2.2GHz, Gemini, NVIDIA K20x	USA	560,640	17.6	8.21
2	Lawrence Livermore National Laboratory	IBM	Sequoia BlueGene/Q, Power BQC 16C 1.6GHz, Custom	USA	1,572,864	16.3	7.89
3	RIKEN Advanced Institute for Computational Science	Fujitsu	K Computer SPARC64 VIIIfx 2.0GHz, Tofu Interconnect	Japan	795,024	10.5	12.66
4	Argonne National Laboratory	IBM	Mira BlueGene/Q, Power BQC 16C 1.6GHz, Custom	USA	786,432	8.16	3.95
5	Forschungszentrum Juelich (FZJ)	IBM	JuQUEEN BlueGene/Q, Power BQC 16C 1.6GHz, Custom	Germany	393,216	4.14	1.97
6	Leibniz Rechenzentrum	IBM	SuperMUC iDataPlex DX360M4, Xeon E5 8C 2.7GHz, Infiniband FDR	Germany	147,456	2.90	3.52
7	Texas Advanced Computing Center/UT	Dell	Stampede PowerEdge C8220, Xeon E5 8C 2.7GHz, Intel Xeon Phi	USA	204,900	2.66	
8	National SuperComputer Center in Tianjin	NUDT	Tianhe-1A NUDT TH MPP, Xeon 6C, NVidia, FT-1000 8C	China	186,368	2.57	4.04
9	CINECA	IBM	Fermi BlueGene/Q, Power BQC 16C 1.6GHz, Custom	Italy	163,840	1.73	0.82
10	IBM	IBM	DARPA Trial Subset Power 775, Power7 8C 3.84GHz, Custom	USA	63,360	1.52	3.57

Systems	2009	2015 +1/-0	2018 +1/-0
System peak	**2 Peta**	**100-300 Peta**	**1 Exa**
Power	**6 MW**	**~15 MW**	**~20 MW**
System memory	0.3 PB	5 PB	64 PB (+)
Node performance	125 GF	0.5 TF or 7 TF	2 TF or 10TF
Node memory BW	25 GB/s	0.2TB/s or 0.5TB/s	0.4TB/s or 1TB/s
Node concurrency	12	O(100)	O(1k) or 10k
Total Node Interconnect BW	3.5 GB/s	100-200 GB/s 10:1 vs memory bandwidth 2:1 alternative	200-400GB/s (1:4 or 1:8 from memory BW)
System size (nodes)	18,700	50,000 or 500,000	O(100,000) or O(1M)
Total concurrency	225,000	O(100,000,000) *O(10)-O(50) to hide latency	O(billion) * O(10) to O(100) for latency hiding
Storage	15 PB	150 PB	500-1000 PB (>10x system memory is min)
IO	0.2 TB	10 TB/s	60 TB/s (how long to drain the machine)
MTTI	days	O(1day)	O(1 day)

Why can't we keep doing what we've been doing?

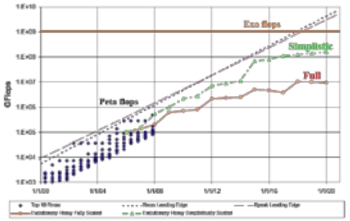
Optimization target for hardware has evolved to new direction (but pmodels have not kept up)

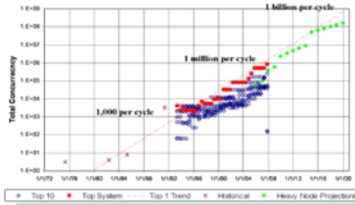
Old Constraints

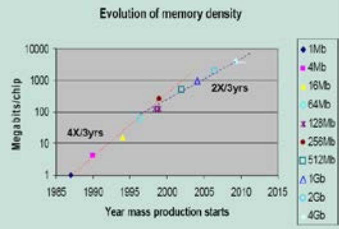
- ***Peak clock frequency** as primary limiter for performance improvement*
- **Cost:** *FLOPs* are biggest cost for system: *optimize for compute*
- **Concurrency**: Modest growth of parallelism by adding nodes
- **Memory scaling**: *maintain byte per flop capacity and bandwidth*
- **Locality**: *MPI+X model (uniform costs within node & between nodes)*
- ***Uniformity**: Assume uniform system performance*
- **Reliability**: *It's the hardware's problem*

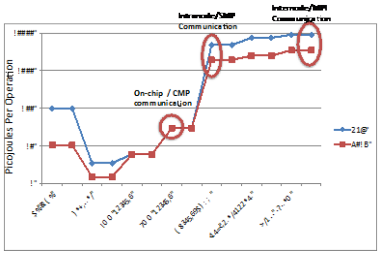
New Constraints

- ***Power** is primary design constraint for future HPC system design*
- **Cost:** *Data movement dominates: optimize to minimize data movement*
- **Concurrency:** *Exponential growth of parallelism within chips*
- **Memory Scaling:** *Compute growing 2x faster than capacity or bandwidth*
- **Locality**: *must reason about data locality and possibly topology*
- ***Heterogeneity**: Architectural and performance non-uniformity increase*
- **Reliability**: *Cannot count on hardware protection alone*









Fundamentally breaks our current programming paradigm and computing ecosystem

Effect of Hardware on Programming Models

Programming Models are a Reflection of the Underlying Machine Architecture

- *Express what is important for performance*
- *Hide complexity that is not consequential to performance*

Programming Models are Increasingly Mismatched with Underlying Hardware Architecture

- *Changes in computer architecture trends/costs*
- *Performance and programmability consequences*

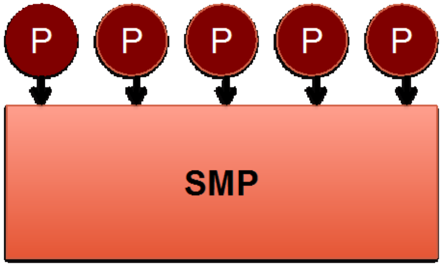
Recommendations on where to Reformulate Programming Models for the Future of HPC

- *Emphasis on **Performance Portability***
- *What to **virtualize***
- *What to make more **expressive/visible***
- *What to **ignore***

The Programming Model is a Reflection of the Underlying *Abstract Machine Model*

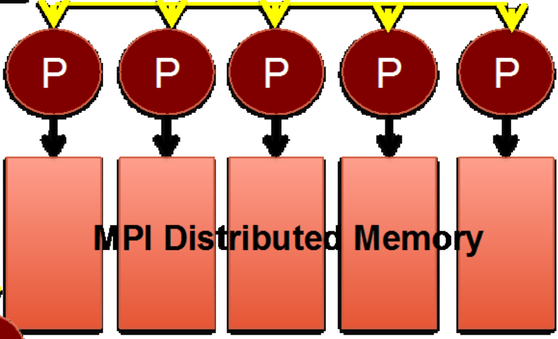
Equal cost SMP/PRAM model

- No notion of non-local access
- int [nx][ny][nz];



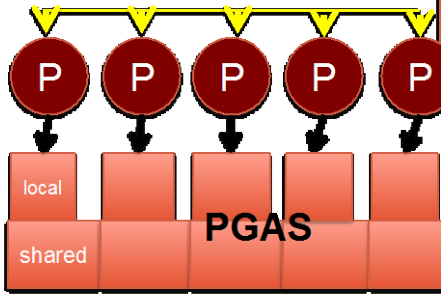
Cluster: Distributed memory model

- No unified memory
- int [localNX][localNY][localNZ];



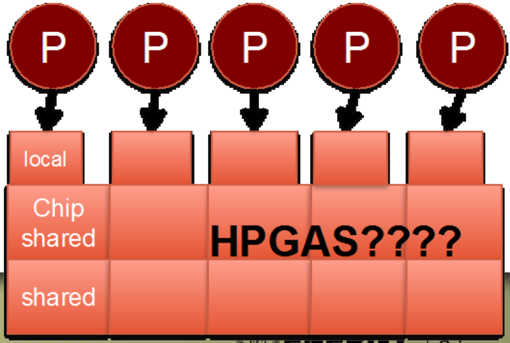
PGAS for horizontal locality

- Data is LOCAL or REMOTE
- shared [Horizontal] int [nx][ny][nz];



HPGAS for vertical data movement

- Depth of hierarchy also matters now
- shared [Vertical][Horizontal] int [x][y][z];?



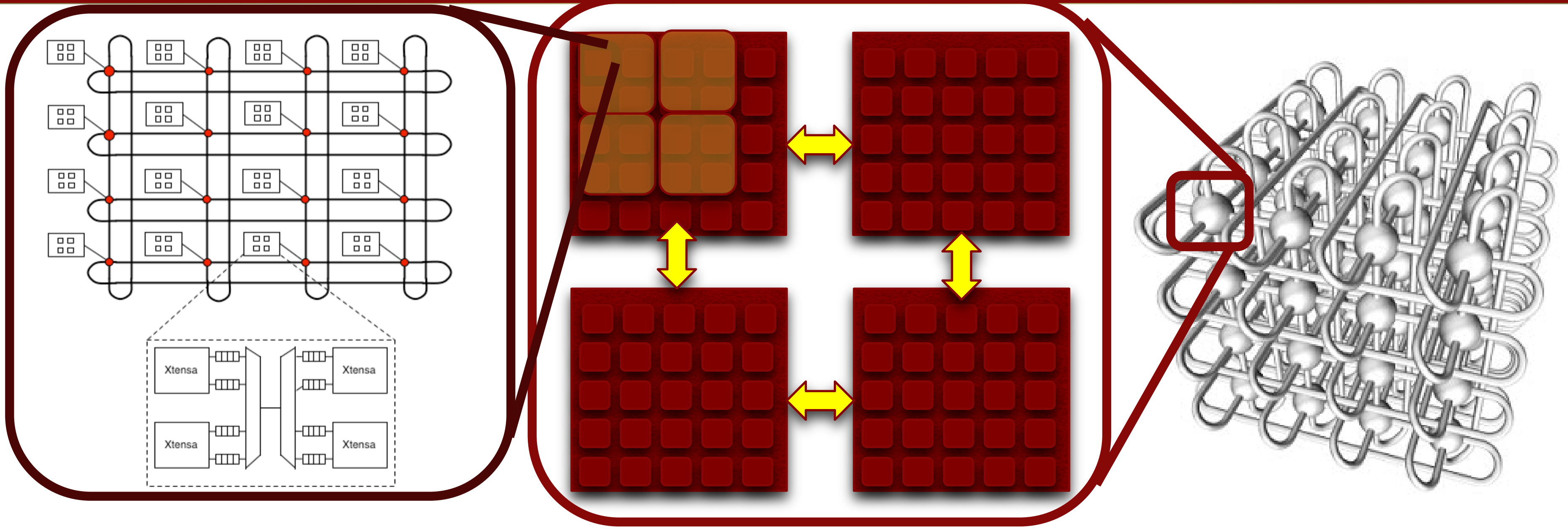
Abstract Machine Models

Definition: ***An Abstract Machine model represents the machine attributes that will be important to reasoning about code performance***

- Enables us to reason about how to map algorithm onto underlying machine architecture
- Enables us to reason about power/performance trade-offs for different algorithm or execution model choices
- Want model to be as simple as possible, but not neglect any aspects of the machine that are important for performance

Notional Multi-Scale Machine Model

(what do we need to reason about when designing a new code?)



Cores

- How Many
- Heterogeneous
- SIMD Width

Network on Chip (NoC)

- Are they equidistant or
- Constrained Topology (2D)

On-Chip Memory Hierarchy

- Automatic or Scratchpad?
- Memory coherency method?

Node Topology

- NUMA or Flat?
- Topology may be important
- Or perhaps just distance

Memory

- Nonvolatile / multi-tiered?
- Intelligence in memory (or not)

Fault Model for Node

- FIT rates, Kinds of faults
- Granularity of faults/recovery

Interconnect

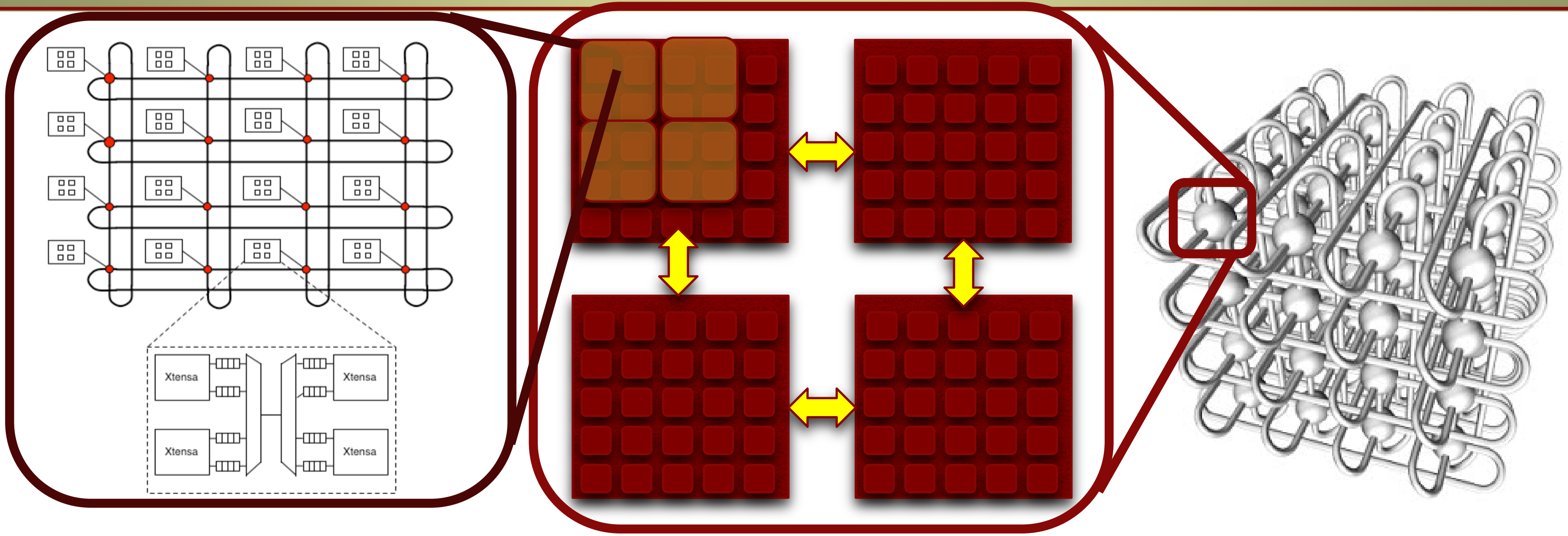
- Bandwidth/Latency/Overhead
- Topology

Primitives for data movement/sync

- Global Address Space or messaging?
- Synchronization primitives/Fences

Notional Multi-Scale Abstract Machine Model

(what do we need to reason about when designing a new code?)



For each parameterized machine attribute, can

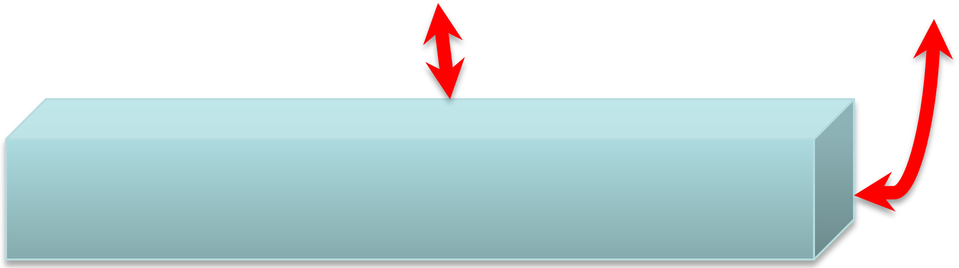
- **Ignore it**: If ignoring it has no serious power/performance consequences
- **Abstract it (*virtualize*)**: If it is well enough understood to support an automated mechanism to optimize layout or schedule
  - This makes programmers life easier (one less thing to worry about)
- **Expose it (*unvirtualize*)**: If there is not a clear automated way of make decisions
  - Must involve the human/programmer in the process ***(make pmodel more expressive)***
  - Directives to control data movement or layout (for example)

Want model to be as simple as possible, but not neglect any aspects of the machine that are important for performance

Data Movement

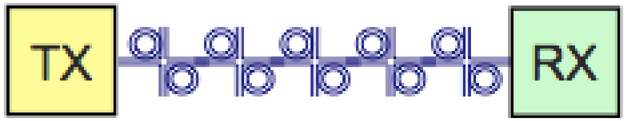
The Problem with Wires:
Energy to move data proportional to distance

- **Cost to move a bit on copper wire:**
  - **power= bitrate * Length / cross-section area**

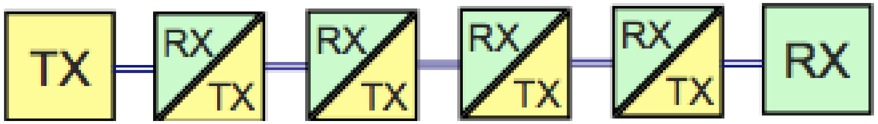


- **Wire data capacity constant as feature size shrinks**
- ***Cost to move bit proportional to distance***
- ***~1TByte/sec max feasible off-chip BW (10GHz/pin)***
- ***Photonics reduces distance-dependence of bandwidth***

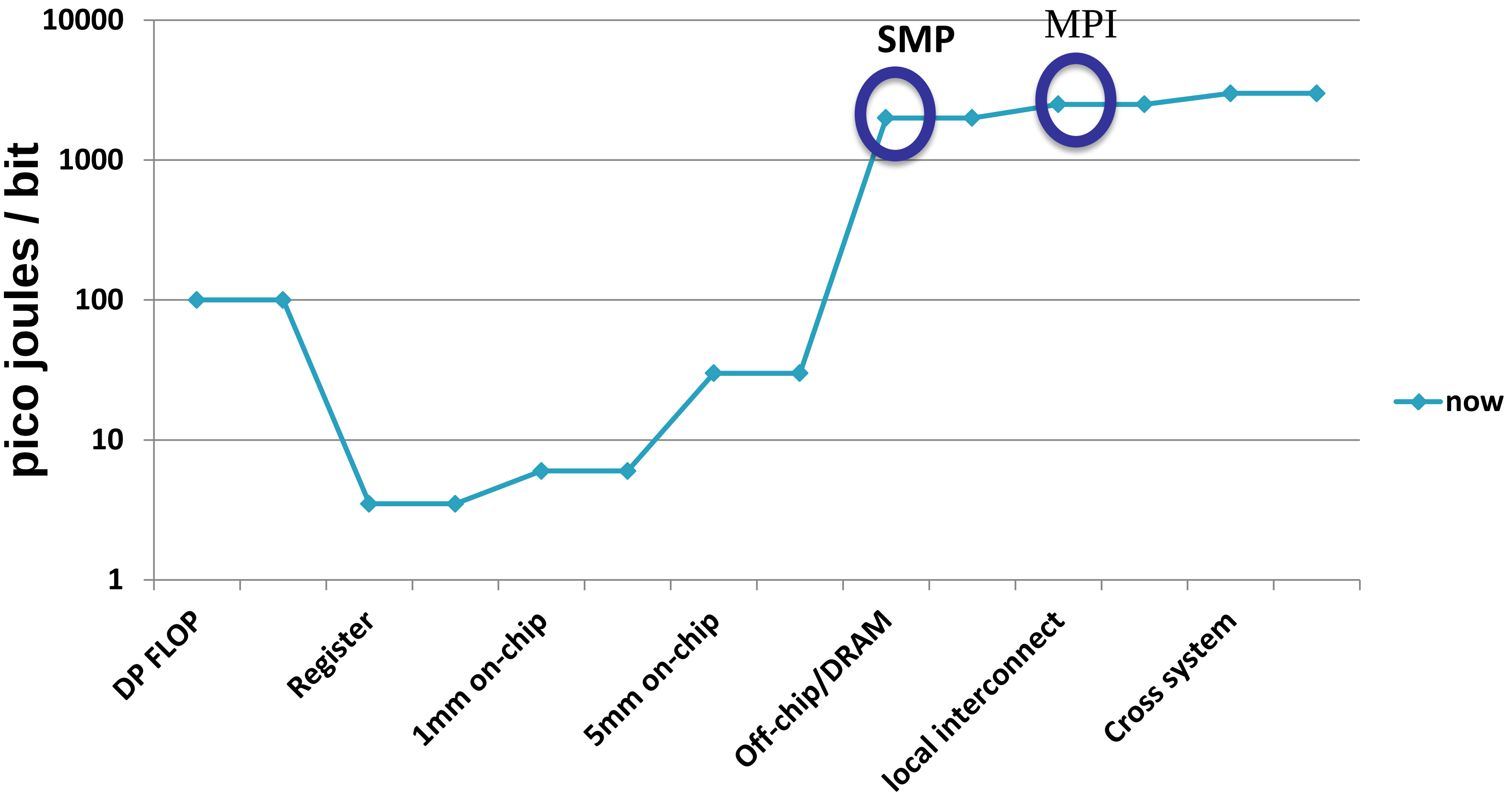
Photonics requires no redrive and passive switch little power



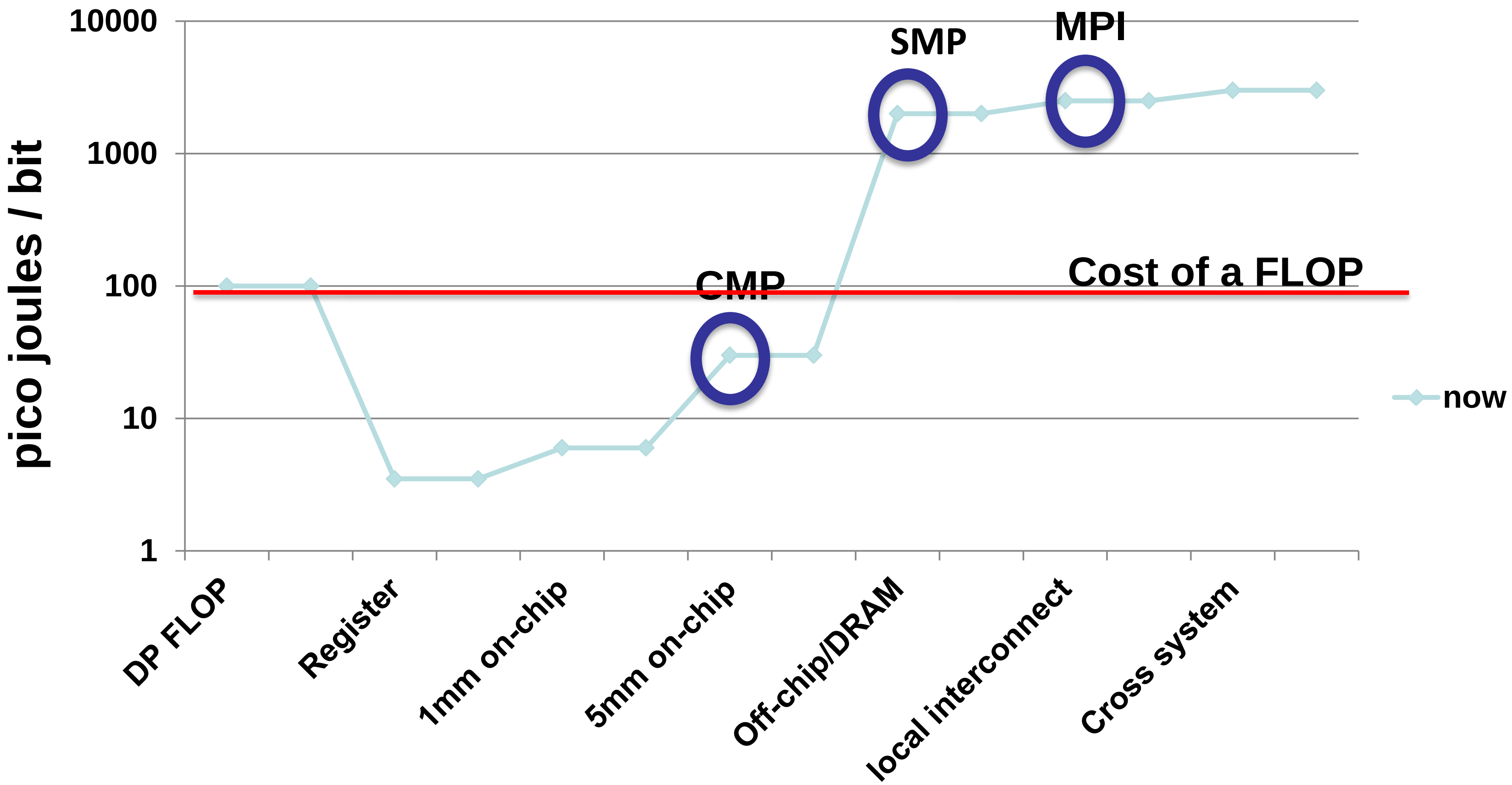
Copper requires to signal amplification even for on-chip connections



The Cost of Data Movement

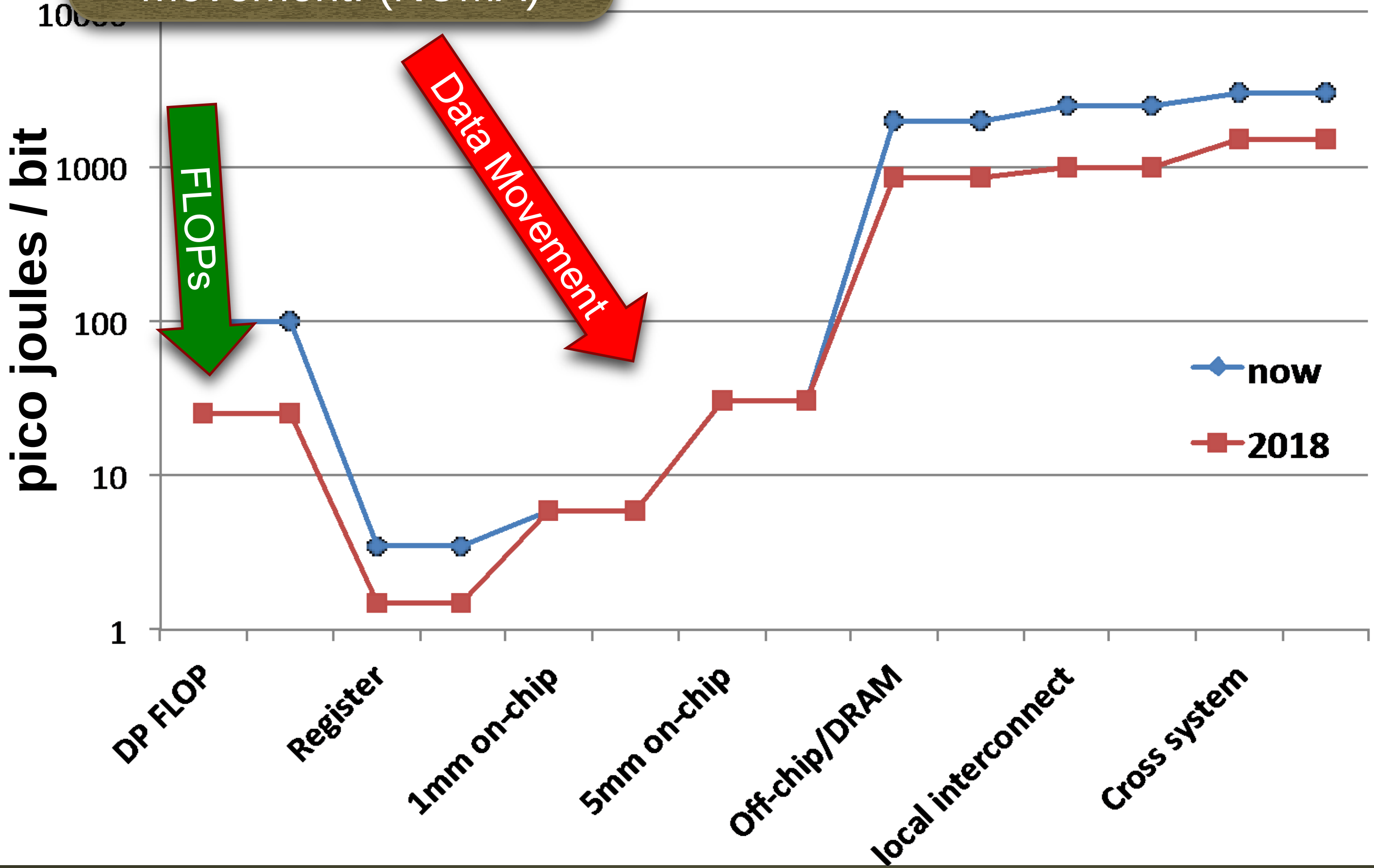


The Cost of Data Movement



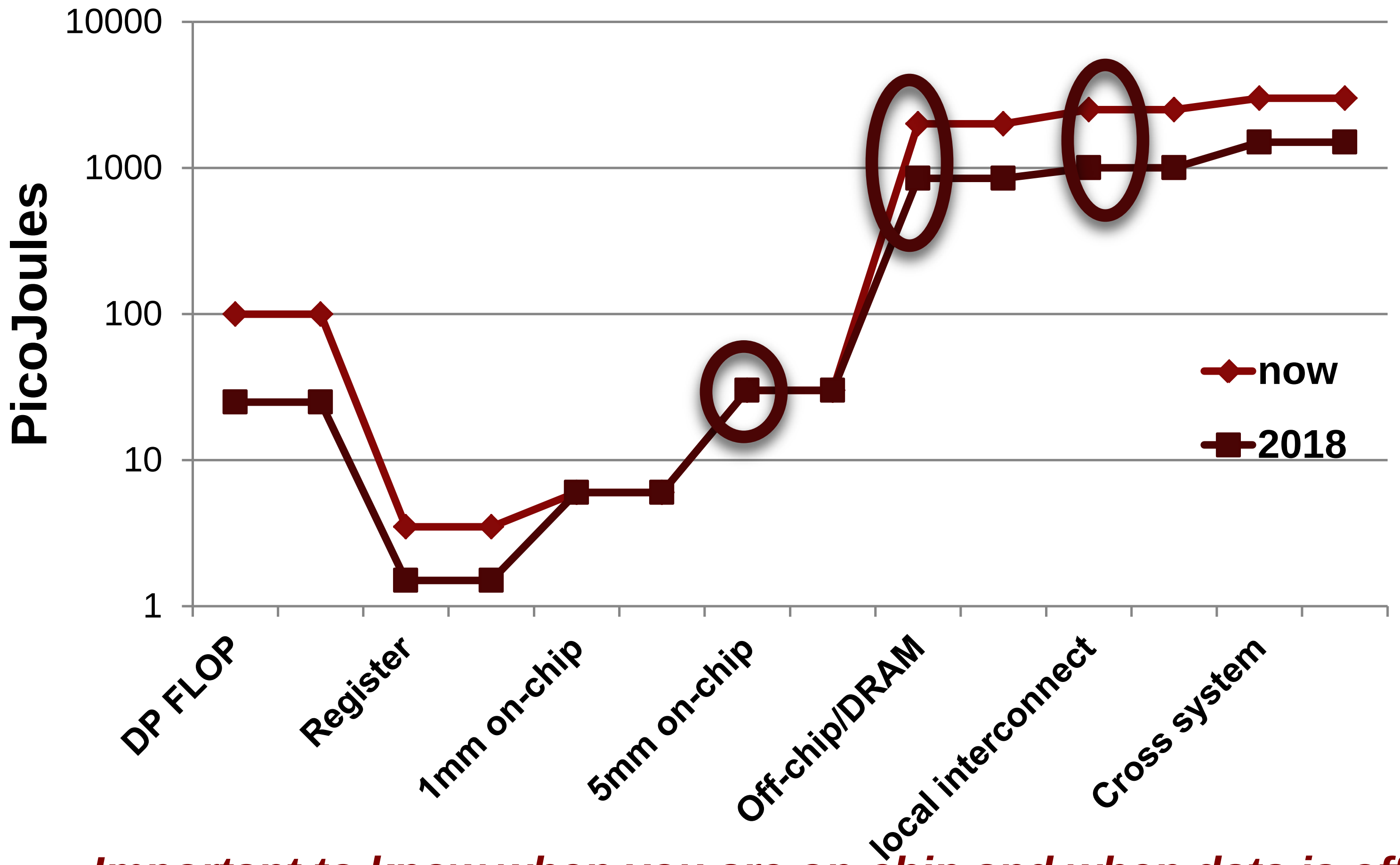
The Cost of Data Movement in 2018

FLOPs will cost less than on-chip data movement! (NUMA)



Data Movement Costs

Energy Efficiency will require careful management of data locality



Important to know when you are on-chip and when data is off-chip!

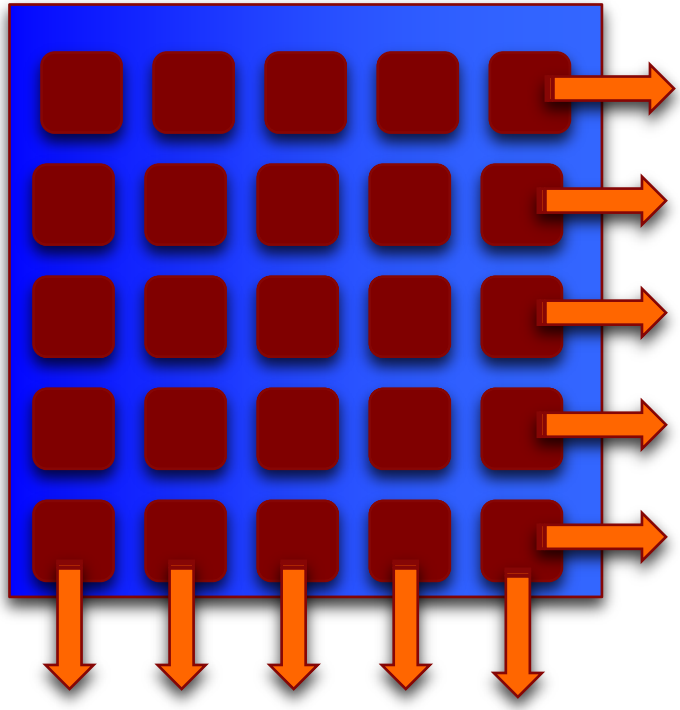
Consequences for Algorithm Design

- **Current Algorithms are designed to minimize FLOPs**
  - Often at the expense of data movement

- **But if data movement costs more than FLOPs, we are using the wrong metric for optimizing algorithms!**

- **Future of algorithm design**
  - Incorporate data movement costs as a metric for “algorithm complexity”
  - Consume additional flops to AVOID data movement (Communication Avoiding Algorithms)
  - And make communication “locality aware” *(will talk more about that)*
  - *Even communication topology may be important*

Future of On-Chip Architecture
(Nov 2009 DOE arch workshop)



Scale-out for Planar geometry

~1000-10k simple cores

4-8 wide SIMD or VLIW bundles

Either 4 or 50+ HW threads

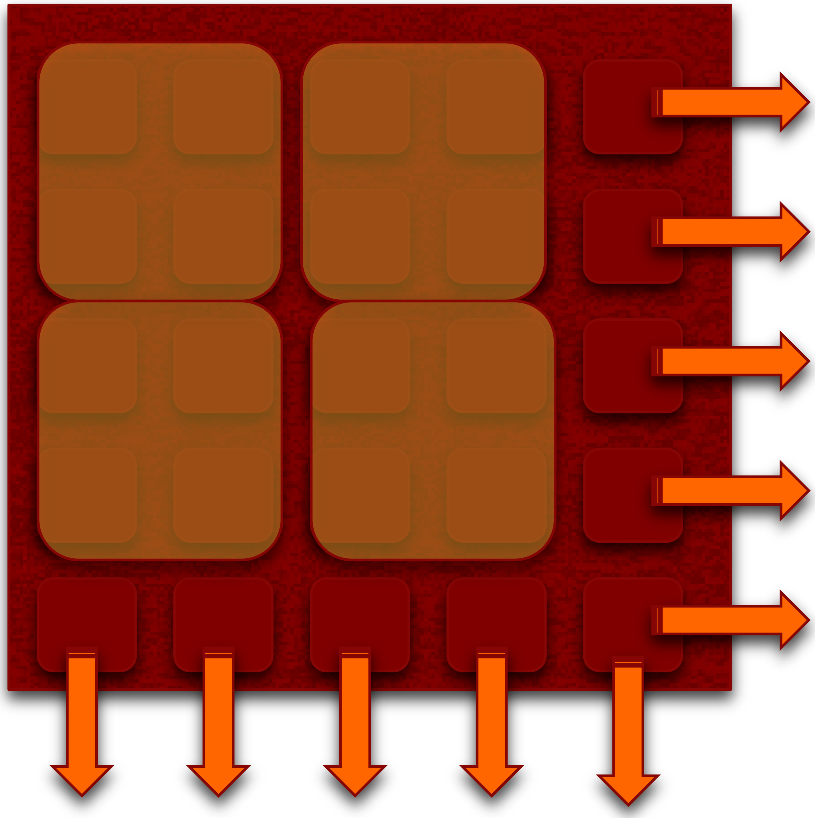
On-chip communication Fabric

- Low-degree topology for on-chip communication (torus or mesh)
- *Can we scale cache-coherence?*
- HW msg. passing
- Global (possibly nonCC memory)
- Shared register file (clusters)

Off-chip communication fabric

- Integrated directly on an SoC
- Reduced component counts
- Coherent with TLB (no pinning)

Cost of Data Movement
(towards "coherence domains" on chip)



Cost of moving long-distances on chip motivates clustering on-chip

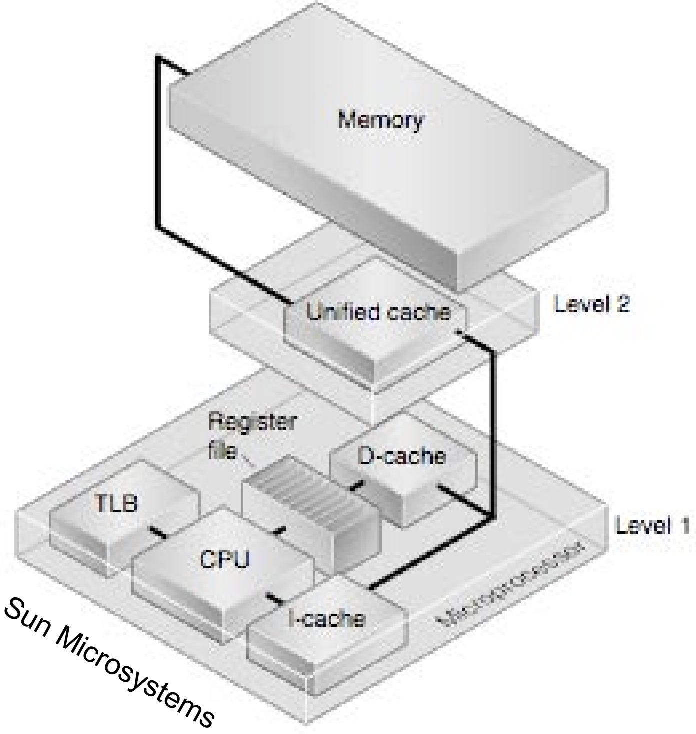
- 1mm costs ~6pj (today & 2018)
- 20mm costs ~120 pj (today & 2018)
- FLOP costs ~100pj today
- FLOP costs ~25pj in 2018

Different Architectural Directions

- GPU: WARPs of hardware threads clustered around shared register file
- CMP: limited area cache-coherence
- CMT: hardware multithreading clusters

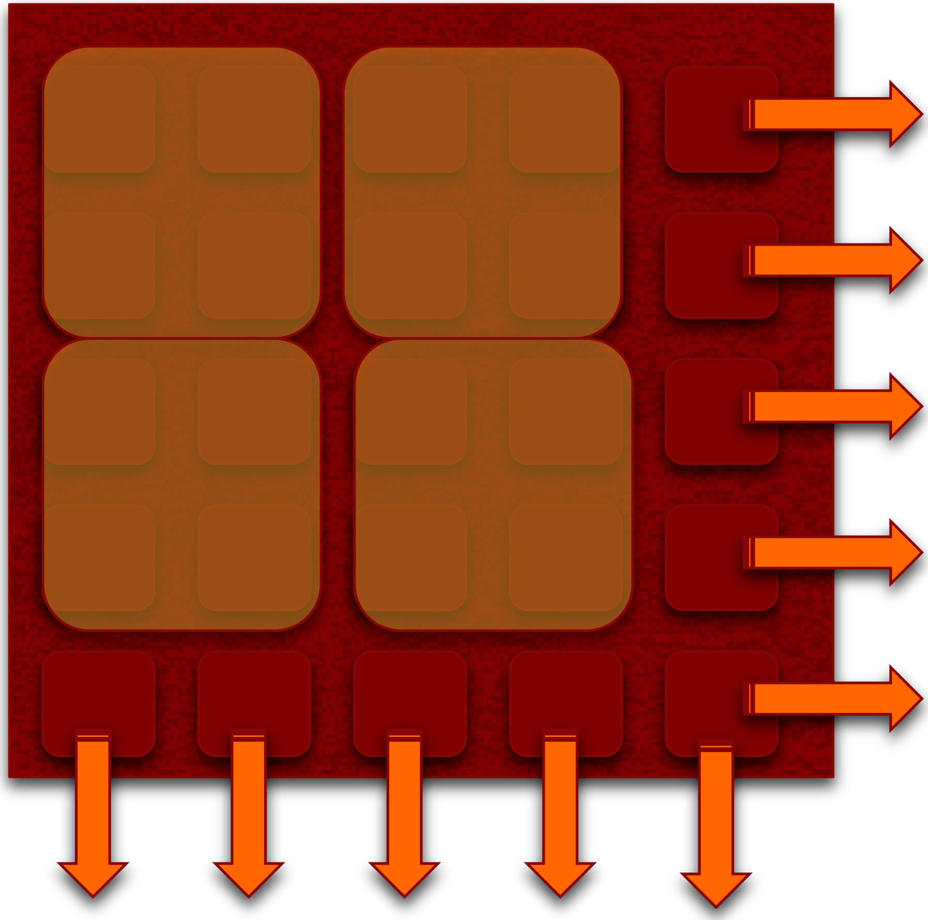
Data Locality Management

Vertical Locality Management
(spatio-temporal optimization)



Sun Microsystems

Horizontal Locality Management
(topology optimization)



Software Mechanisms for Expressing Locality

Problems with Existing Abstractions for Expressing Locality

Our current programming models assume all communicating elements are equidistant (PRAM)

- OpenMP, and MPI each assume flat machine at their level of parallelism

But the machine is not flat!!!

- Lose performance because expectation and reality are mismatched
- *Pmodel does not match underlying machine model!!*

What is wrong with Flat MPI?

- 10x higher bandwidth between cores on chip
- 10x lower latency between cores on chip
- If you pretend that every core is a peer (each is just a generic MPI rank) you are leaving a lot of performance on the table
- You cannot domain-decompose things forever

Two-level Parallelism? (MPI+X?)

Hybrid Model (MPI+X)

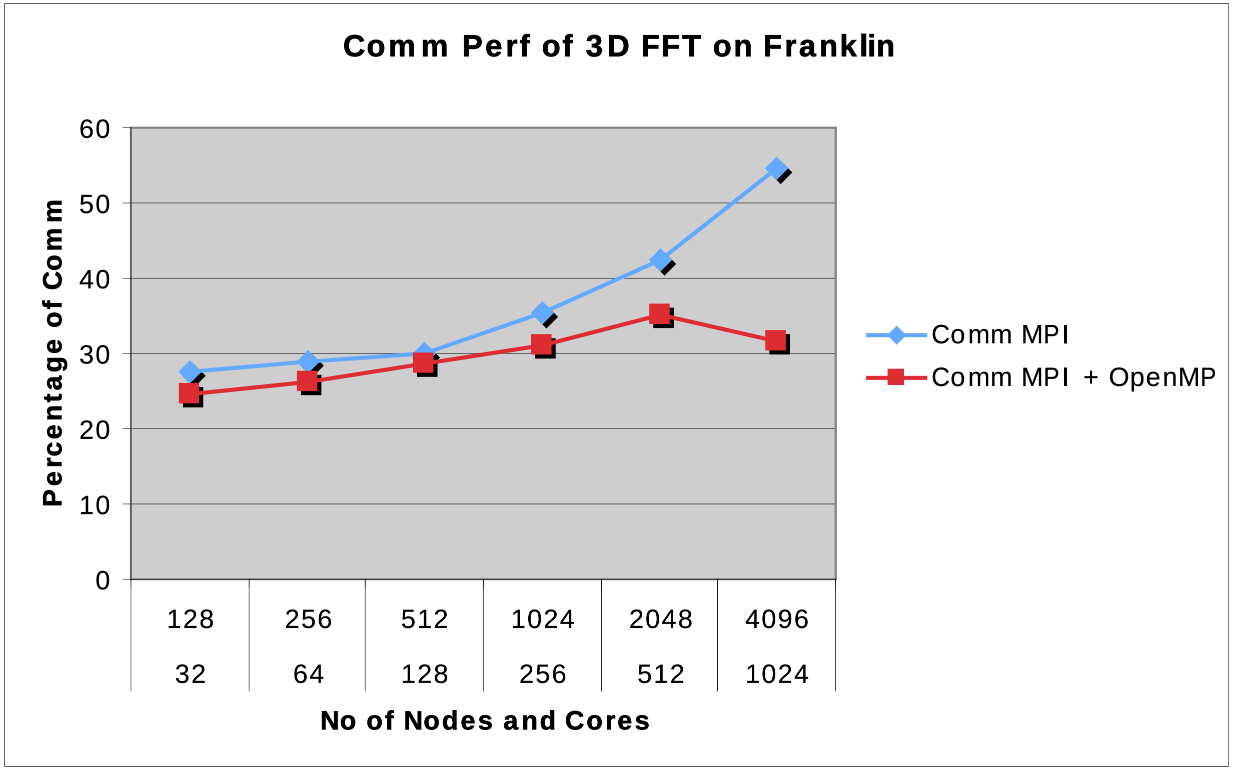
- Recognizes biggest cost delta is when you go off-chip
- This is not the same as old SMPs
  - 10x-100x higher bandwidth between cores on chip and 10x-100x lower latency
- *Failure to exploit hierarchical machine architecture will drastically inhibit ability to efficiently exploit concurrency! (requires code structure changes)*

If this abstraction is sufficient to capture performance (within factor of 2x) then why make things more complicated by having hierarchical abstraction?

Current Practices (2-level Parallelism)

Hybrid Model improves 3D FFT communication performance

- Enables node to send larger messages between nodes
- Substantial improvements in communications efficiency



Good News!

Benefits of expressing Two-levels of locality

Current Practices (2-level Parallelism)

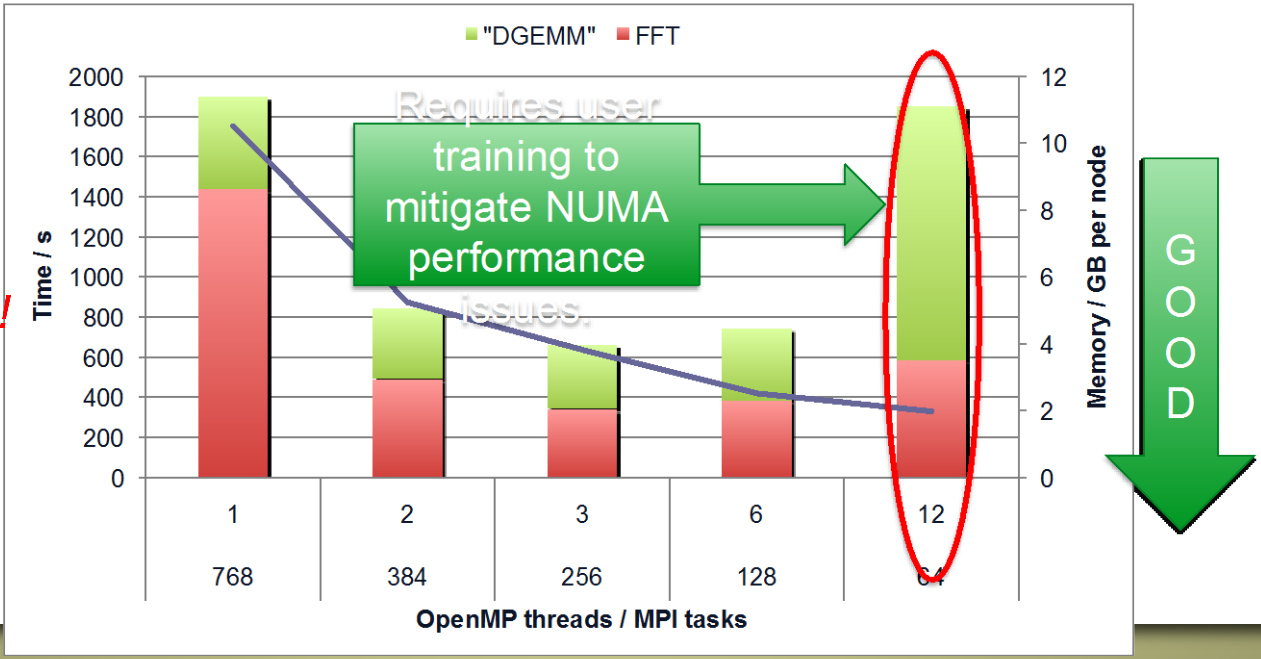
NUMA Effects Ignored (with huge consequence)

MPI+OMP Hybrid

It's the Revenge of the SGI Origin2000

- Reduces memory footprint
- Increases performance up to NUMA-node limit
- *Then programmer responsible for matching up computation with data layout!! (UGH!)*
- *Makes library writing difficult and Makes AMR nearly impossible!*

Bad News!



Requires user training to mitigate NUMA performance issues.

Partitioned Global Address Space (PGAS)

better expression of data locality

Implicitly binds compute location to data layout

Data Layout in PGAS understands two categories of data access performance

- Local
- Not local

Enables powerful locality aware looping constructs

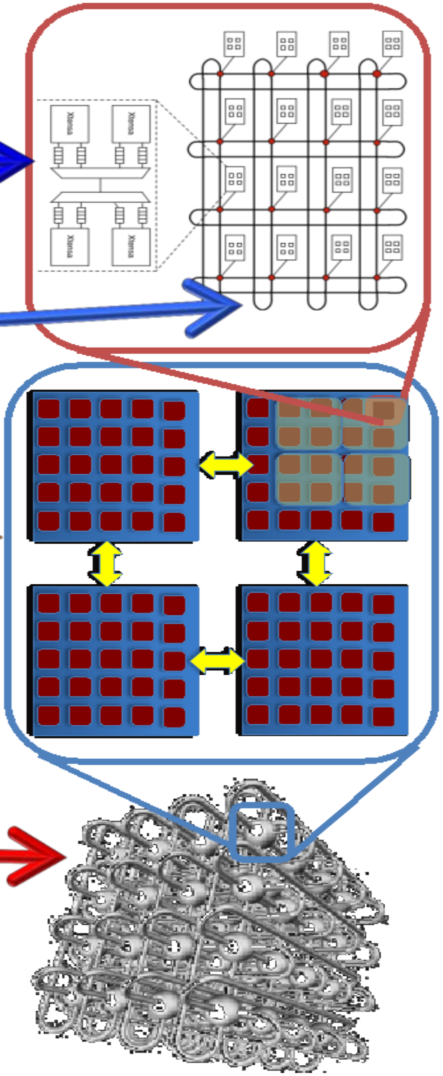
- Can write loop in conventional form, while typesystem determines data layout
- UPC_FORALL() will execute iterations where data is local (affine renumbering of loop iterations)
  - *this is HUGE because totally abstracts/virtualizes # cores*
  - *It also implicitly binds execution locality to data layout*

This is better than flat model, but . . .

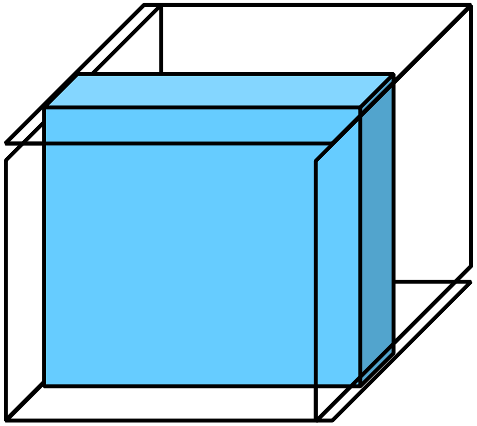
PGAS 1D partitions May Be Insufficient for Expressing hierarchical energy and locality cost

Hierarchical Energy Costs

- FP Op: 100pj
- Register: 3.5pj
- 1mm on chip: 6pj
- 20mm on chip: 120pj
- Off-chip (SMP): 250pj
- Off-chip (DRAM): 2000pj
- Off-chip (neighboring node): ~2500pj
- Off-chip (cross-system): ~3000pj



Example from UPC

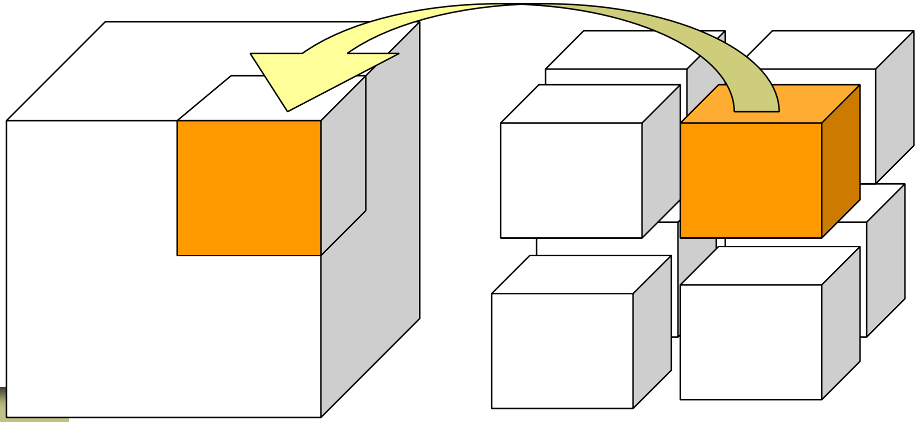


1D Decomp

- Shared [blocksize] int [nx][ny][nz]

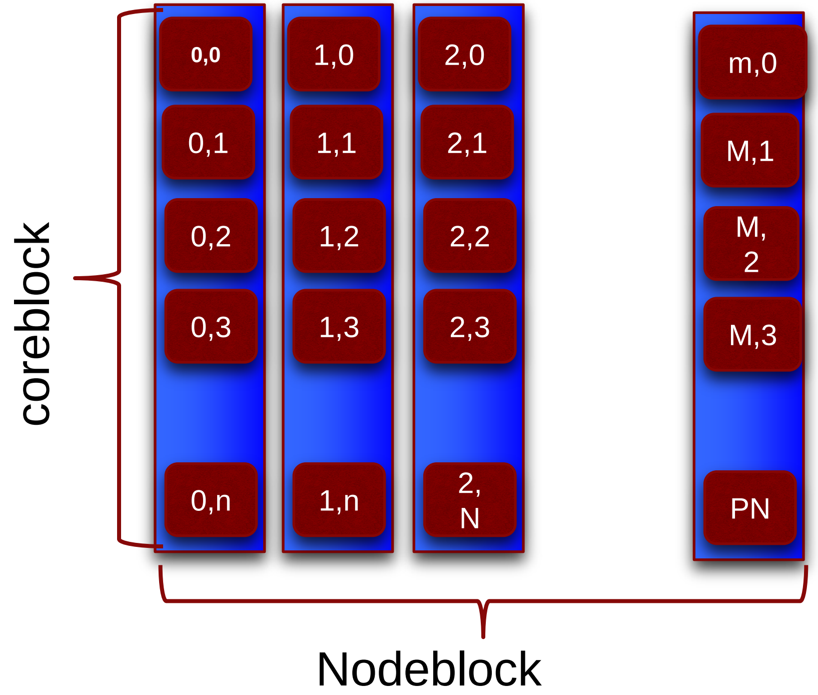
3D Decomp

- Struct gridcell_s { int cell[cellsize] }
- Shared [blocksize] gridcell_t cellgrids[nthreads];
- #define grids(gridno,z,y,z) cell_grids[gridno][((z)/DIMZ) *NO_ROWS*NO_COLS+ etc…..

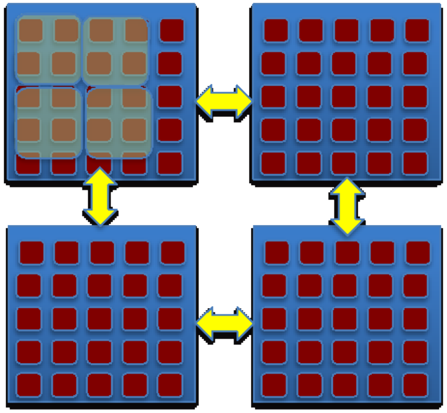


Multidimensional Blocking?

- **Shared [coreblock][nodeblock] int x[nx][ny];**



Our target abstract machine



Doesn't really match our target machine

Expressing Hierarchical Layout

Hierarchical layout statements

- Express mapping of “natural” enumeration of an array to the unnatural system memory hierarchy
- Maintain unified “global” index space for arrays (A[x][y][z])
- Support mapping to complex address spaces
- Convenient for programmers

Iteration expressions more powerful when they bind to ***data locality*** instead of ***threadnumber***

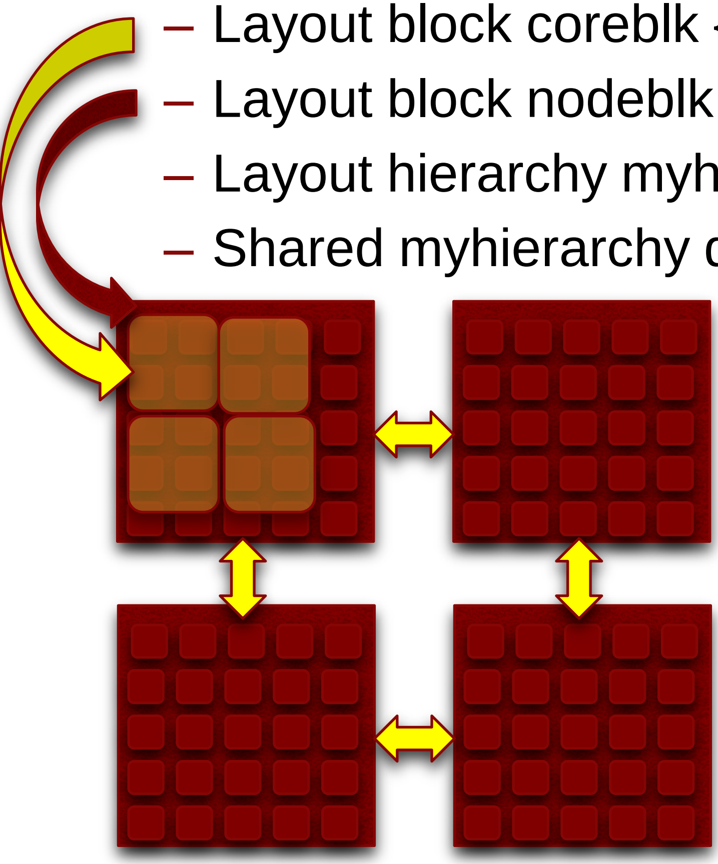
- instead of upc_forall(;;;<threadnumber>)
- Use upc_forall(;;;<implicitly where Array A is local>)

```
upc_forall(i=0;i<NX;i++;A)
  C[j]+=A[j]*B[i][j]);
```

Hierarchical Layout Statements

Building up a hierarchical layout

- – Layout block coreblk {blockx,blocky};
- – Layout block nodeblk {nnx,nny,nnz};
- – Layout hierarchy myheirarchy {coreblk,nodeblk};
- – Shared myhierarchy double a[nx][ny][nz];



- **Then use data-localized parallel loop**

```
    doall_at(i=0;i<nx;i++;a){
        doall_at(j=0;j<ny;j++;a){
          doall_at(k=0;k<nz;k++;a){
            a[i][j][k]=C*a[i+1]…>
```

- ***And if layout changes, this loop remains the same***

Satisfies the request of the application developers (minimize the amount of code that changes)

Conclusions on Data Layout

Failure to express data locality has substantial cost in application performance

- Compiler and runtime cannot figure this out on its own given limited information current languages and programming models provide

Hierarchical data layout statements offer *better expressiveness*

- Must be hierarchical
- Must be multidimensional
- Support composable build-up of layout description

Data-centric parallel expressions offer *better virtualization* of # processors/threads

- Don’t execute based on “thread number”
- Parallelize & execute based on data locality
- Enables layout to be specified in machine-dependent manner without changing execution

Interconnects

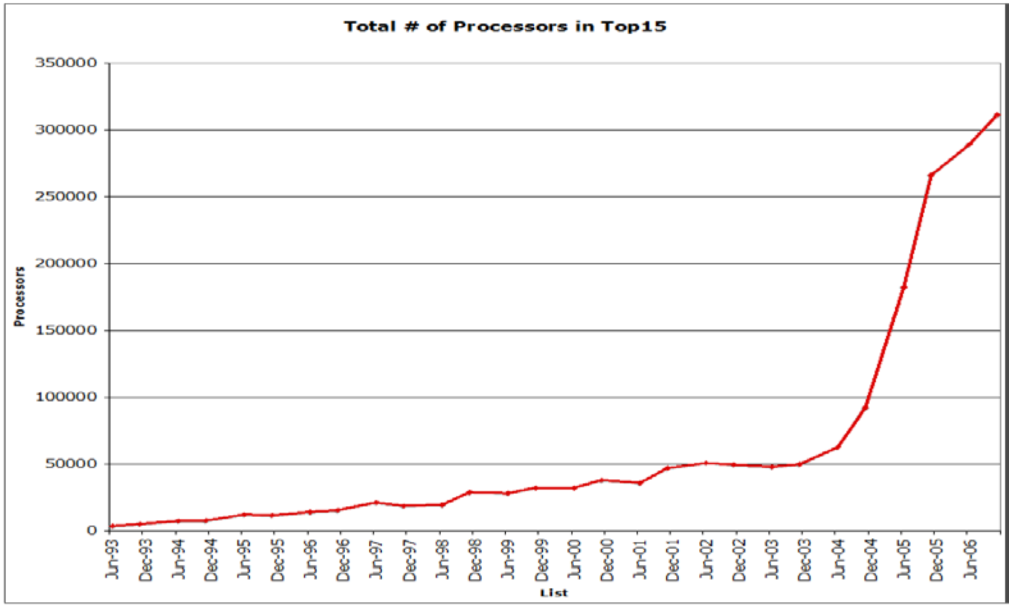
Technology Trends and Effects on Application Performance

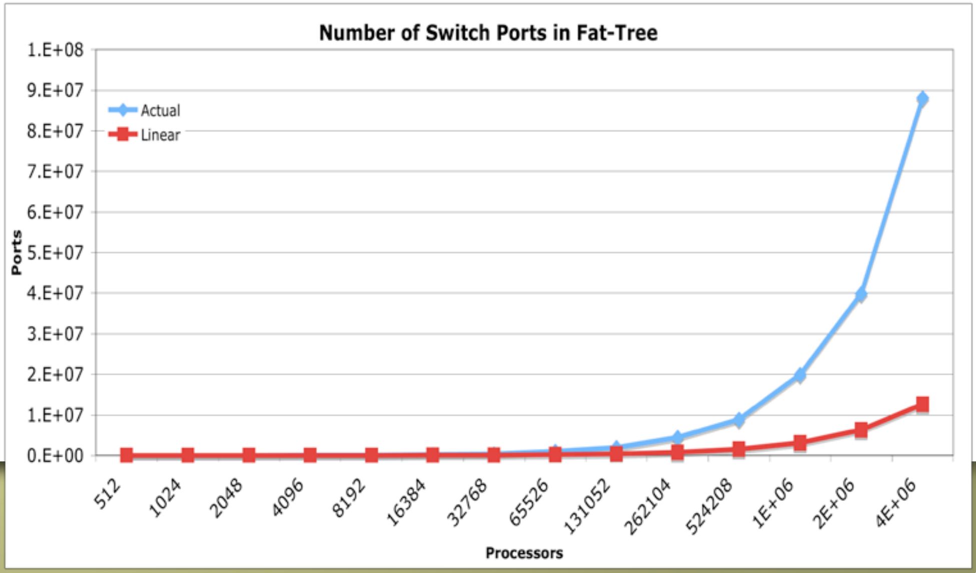
Scalable Interconnects

Can’t afford to continue with Fat-trees or other Fully-Connected Networks (FCNs)

But will Ultrascale applications perform well on lower degree networks like meshes, hypercubes or torii. Or high-radix routers/tapered dragonfly?

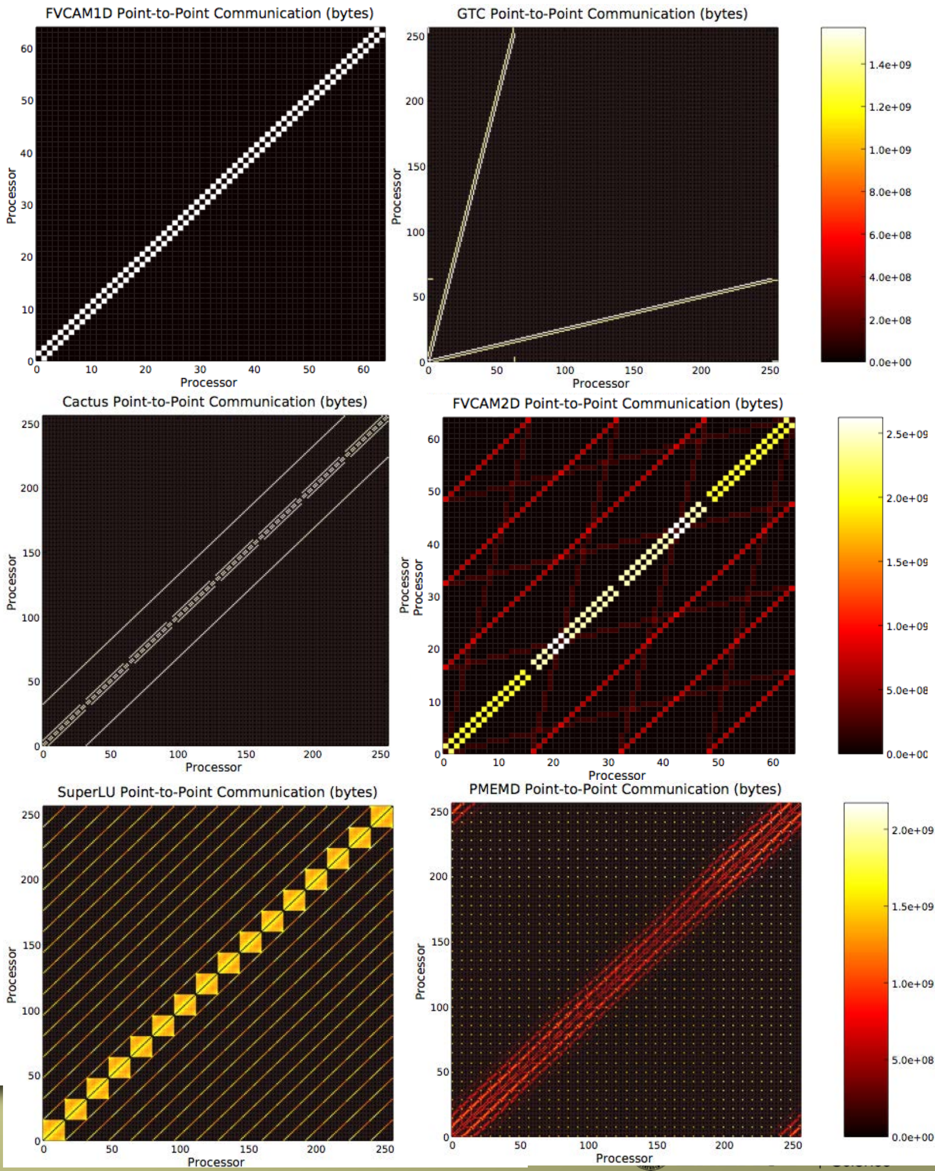
How can we wire up a custom interconnect topology for each application?





Interconnect Design Considerations for Message Passing Applications

- **Application studies provide insight to requirements for Interconnects (both on-chip and off-chip)**
  - On-chip interconnect is 2D planar (crossbar won’t scale!)
  - Sparse connectivity for most apps.; crossbar is overkill
  - No single best topology
  - Most point-to-point message exhibit sparse topology + often bandwidth bound
  - Collectives tiny and primarily latency bound
- **Ultimately, need to be aware of the on-chip interconnect topology in addition to the off-chip topology**
  - Adaptive topology interconnects (HFAST)
  - Intelligent task migration?



FVCAM1D Point-to-Point Communication (bytes)

GTC Point-to-Point Communication (bytes)

Cactus Point-to-Point Communication (bytes)

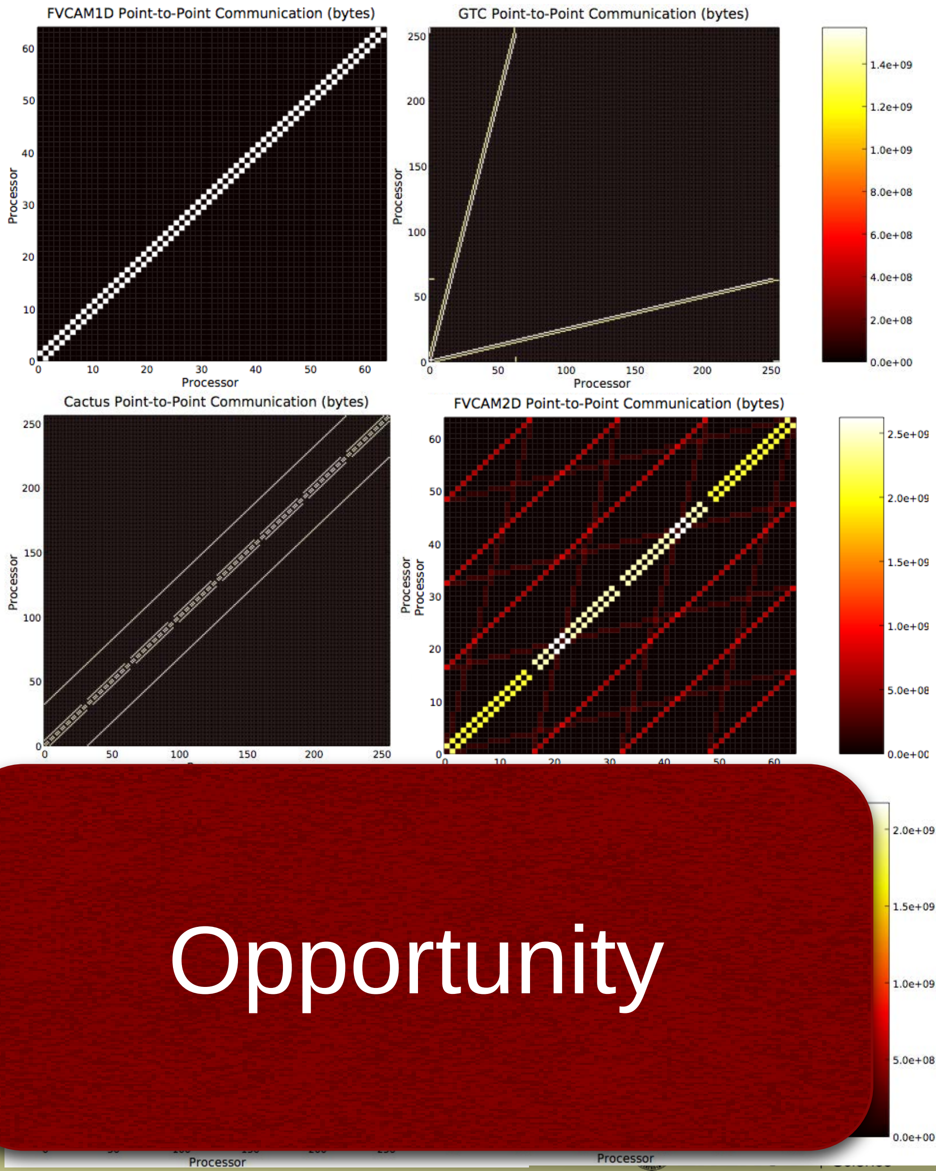
FVCAM2D Point-to-Point Communication (bytes)

SuperLU Point-to-Point Communication (bytes)

PMEMD Point-to-Point Communication (bytes)

Interconnect Design Considerations for Message Passing Applications

- **Application studies provide insight to requirements for Interconnects (both on-chip and off-chip)**
  - On-chip interconnect is 2D planar (crossbar won't scale!)
  - Sparse connectivity for most apps.; crossbar is overkill
  - No single best topology
  - Most point-to-point message exhibit sparse topology + often bandwidth bound
  - Collectives tiny and primarily latency bound
- **Ultimately, need to be aware of the on-chip interconnect topology in addition to the off-chip topology**
  - Adaptive topology interconnects (HFAST
  - Intelligent task migration?

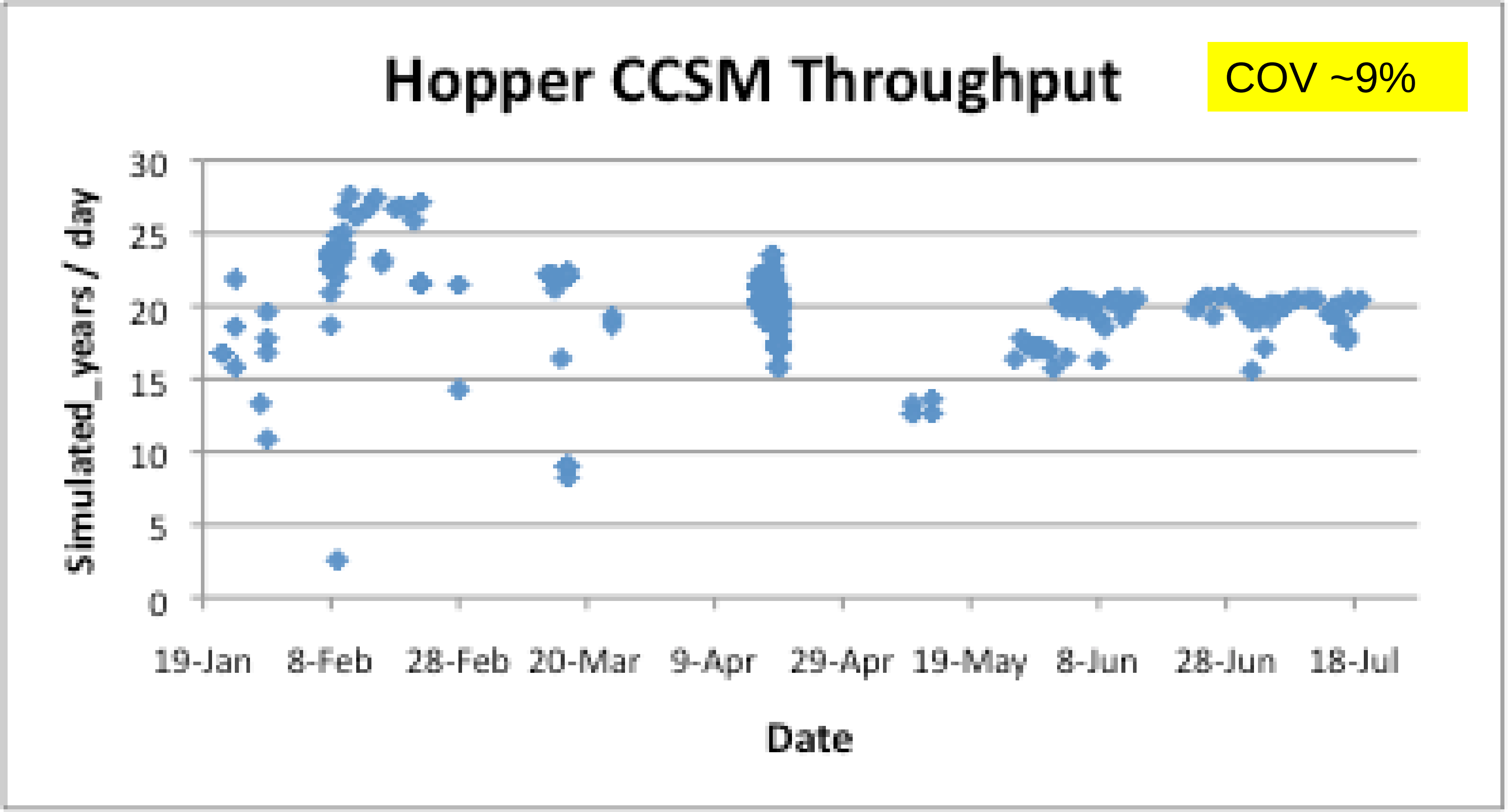


Opportunity

CCSM Performance Variability

(trials of embedding communication topologies)

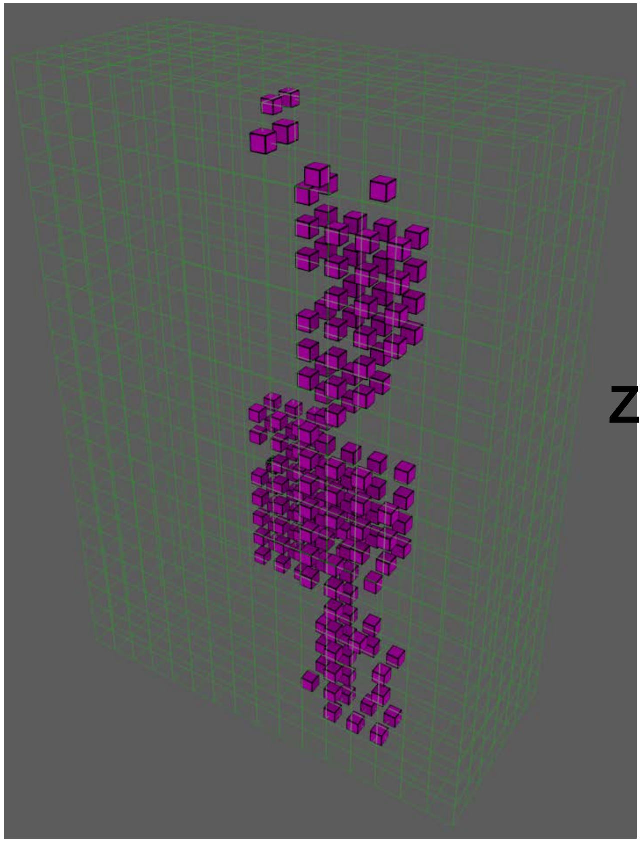
Result of 311 runs of the coupled climate model showing model throughput as a function of completion date.

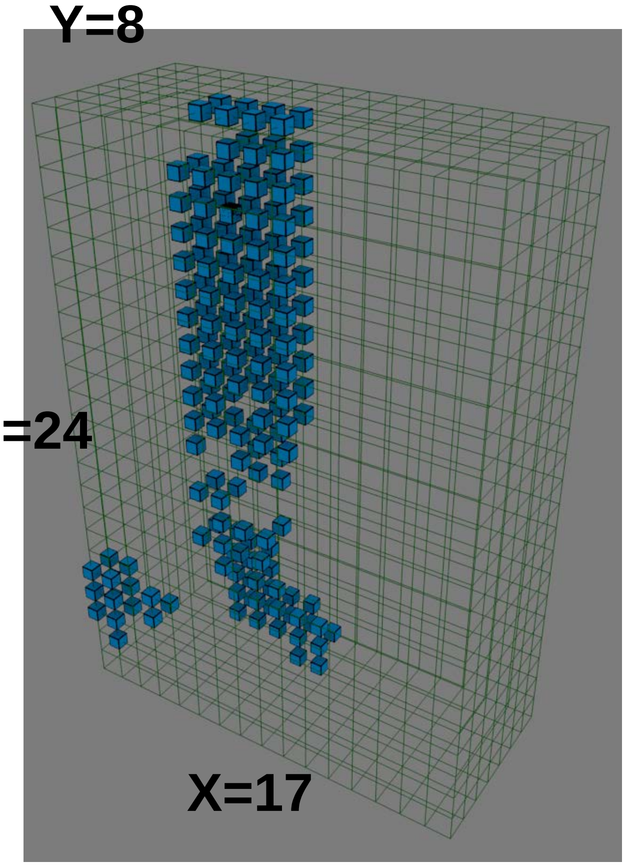


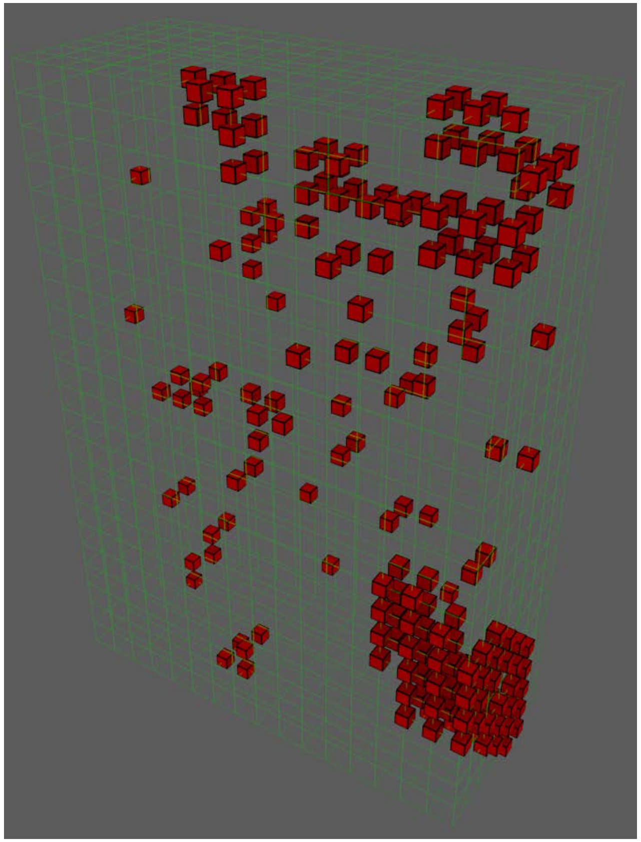
Data from Harvey Wasserman

Node placement of a fast, average and slow run

from Katie Antypas



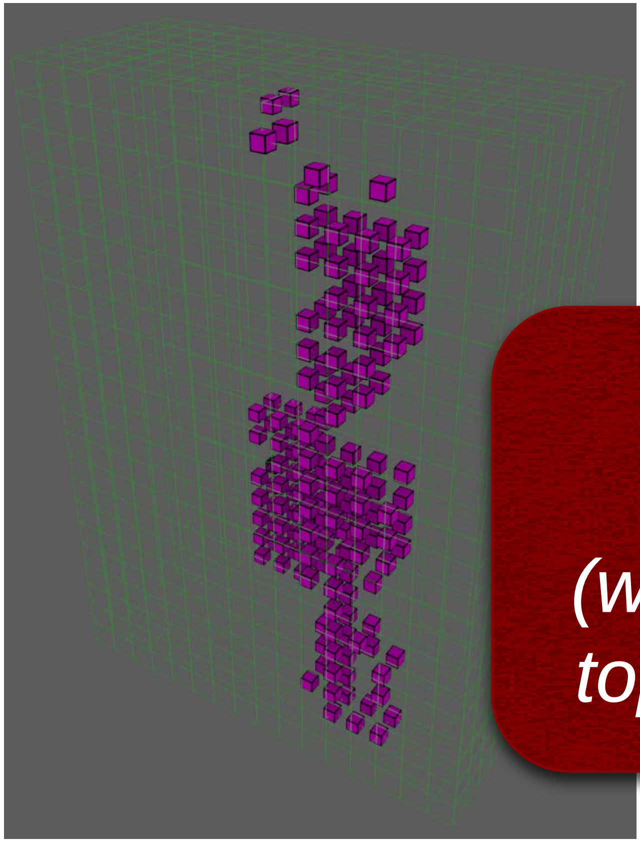


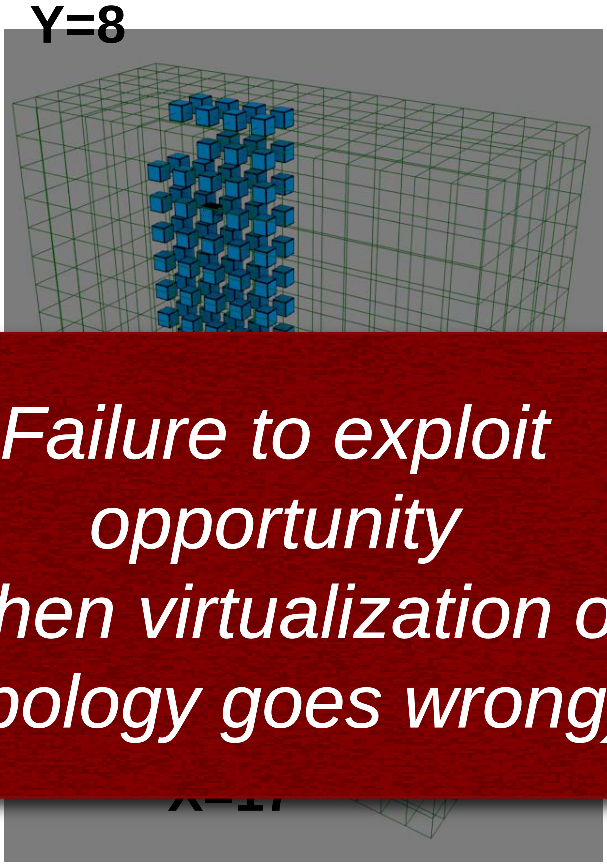


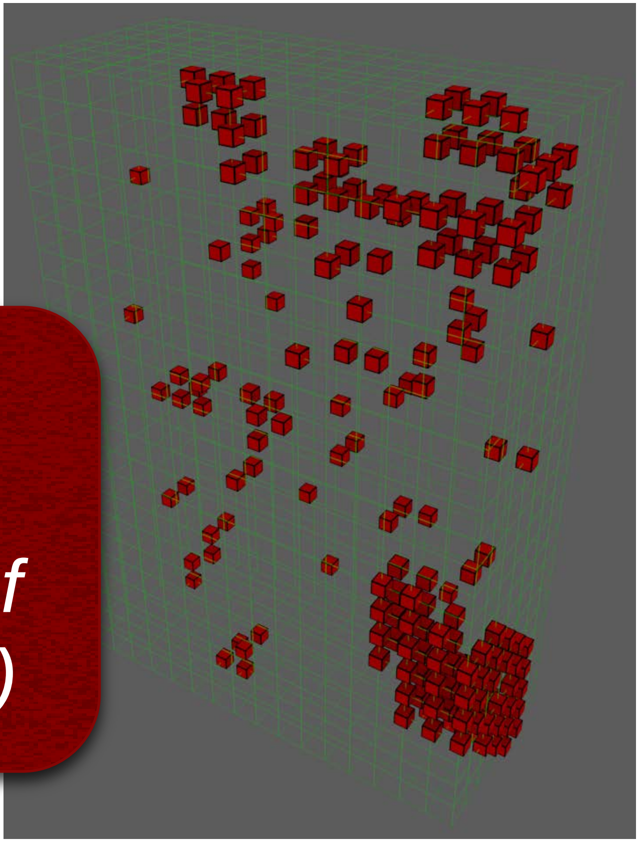
Fast run: 940 seconds **Average run: 1100 seconds** **Slow run: 2462 seconds**

Node placement of a fast, average and slow run

from Katie Antypas



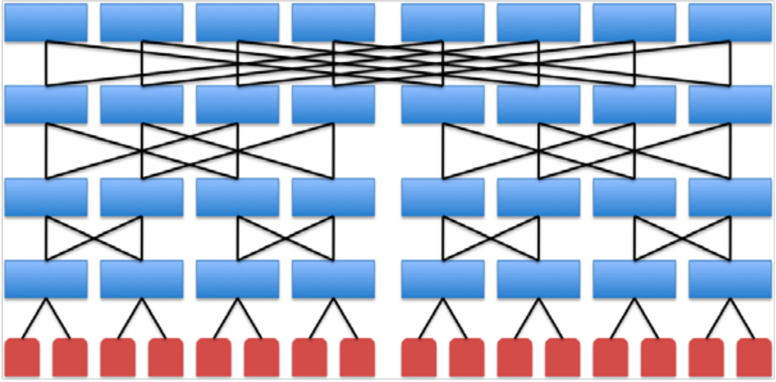




Failure to exploit opportunity (when virtualization of topology goes wrong)

Fast run: 940 seconds **Average run: 1100 seconds** **Slow run: 2462 seconds**

Topology Optimization
(turning Fat-trees into Fit-trees)



A 2-ary 4-tree with 16 nodes.

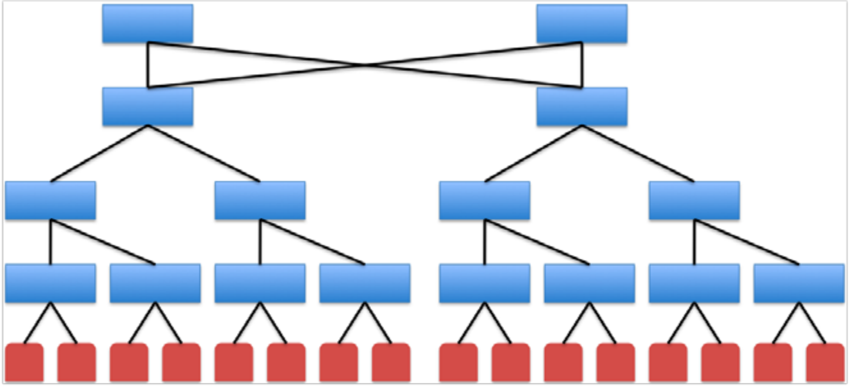
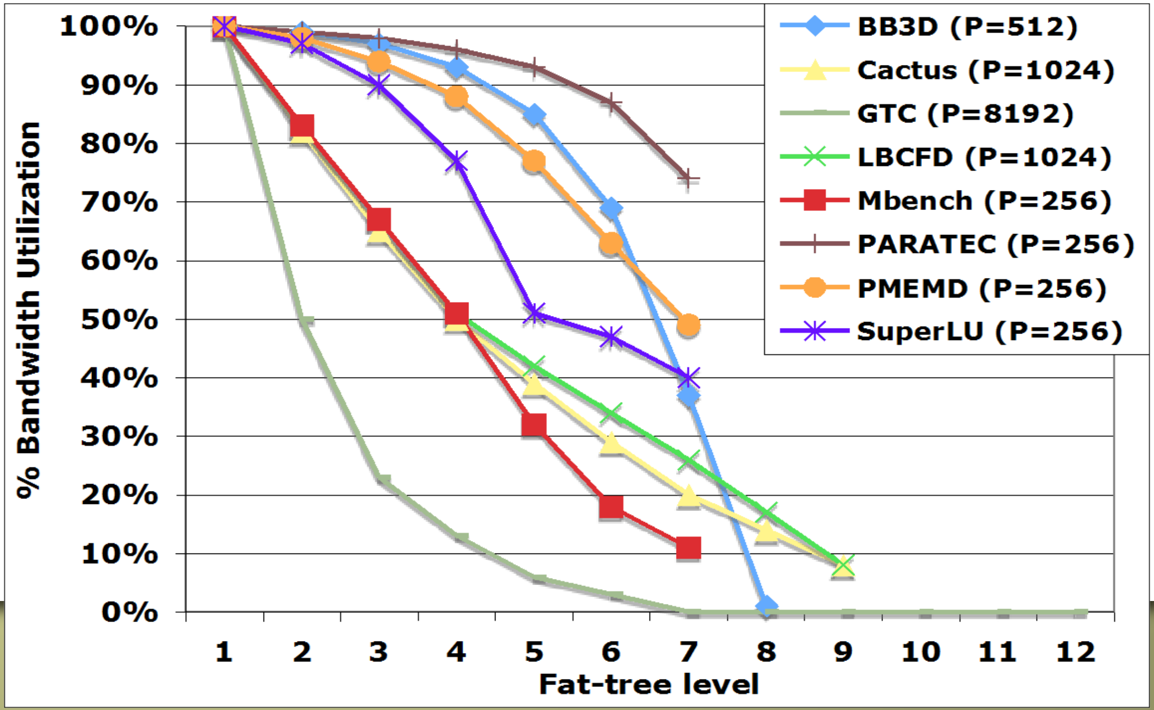


Figure 2: A (2, 2, 4)-TL fit-tree with 16 nodes.

A Fit-tree uses OCS to prune unused (or infrequently used) connections in a Fat-Tree

Tailor the interconnect bandwidth taper to match application data flows



Conclusions on Interconnect

Huge opportunity for communication topology optimization to improve performance

- Runtime information gathering for active task migration, circuit switching
- Use intelligent runtime to remap for locality or to use circuit switching to optimize switch topology

Current Programming Models do not provide facility to express topology

- OpenMP topology un-aware
- MPI has topology directives (tedious, and rarely implemented or used)
- **Results in substantial measurable losses in performance *(within node/OpenMP and inter-node/MPI)***

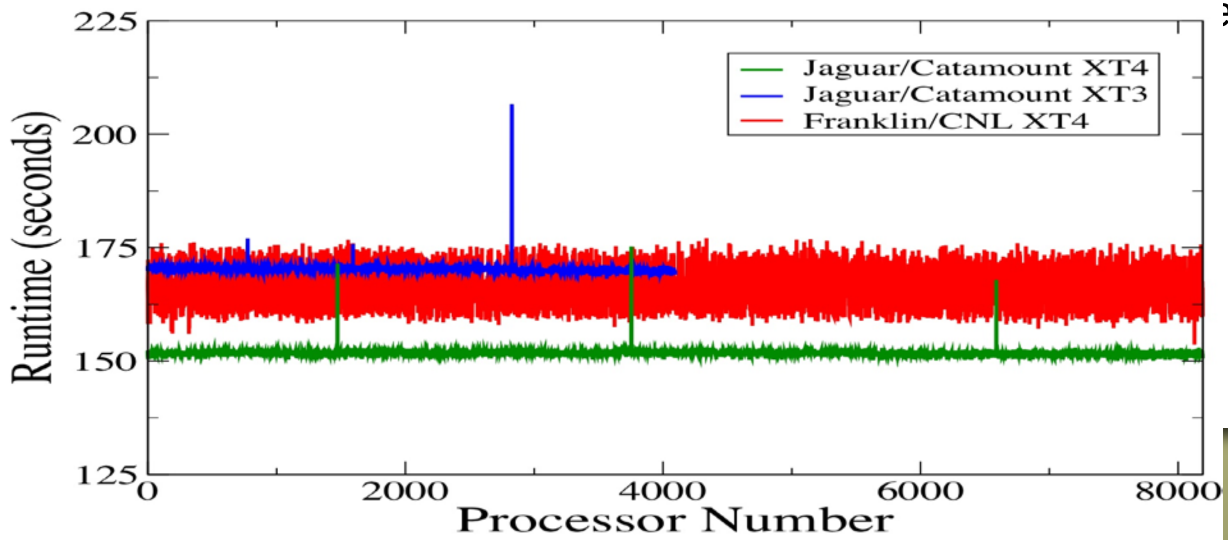
Need to provide the compiler, runtime & resource manager more information about topology

Heterogeneity / Inhomogeneity

The case for asynchronous runtime systems
(aka “execution models”)

Assumptions of Uniformity is Breaking *(many new sources of heterogeneity)*

- **Heterogeneous compute engines (hybrid/GPU computing)**
- **Irregular algorithms**
- **Fine grained power mgmt. makes homogeneous cores look heterogeneous**
  - *thermal throttling on Sandybridge – no longer guarantee deterministic clock rate*
- N**onuniformities in process technology creates non-uniform operating characteristics for cores on a CMP**
- **Fault resilience introduces inhomogeneity in execution rates**
  - *error correction is not instantaneous*
  - *And t* *ence*



Conclusions on Heterogeneity

Sources of performance heterogeneity increasing

- Heterogeneous architectures (accelerator)
- Thermal throttling
- Performance heterogeneity due to transient error recovery

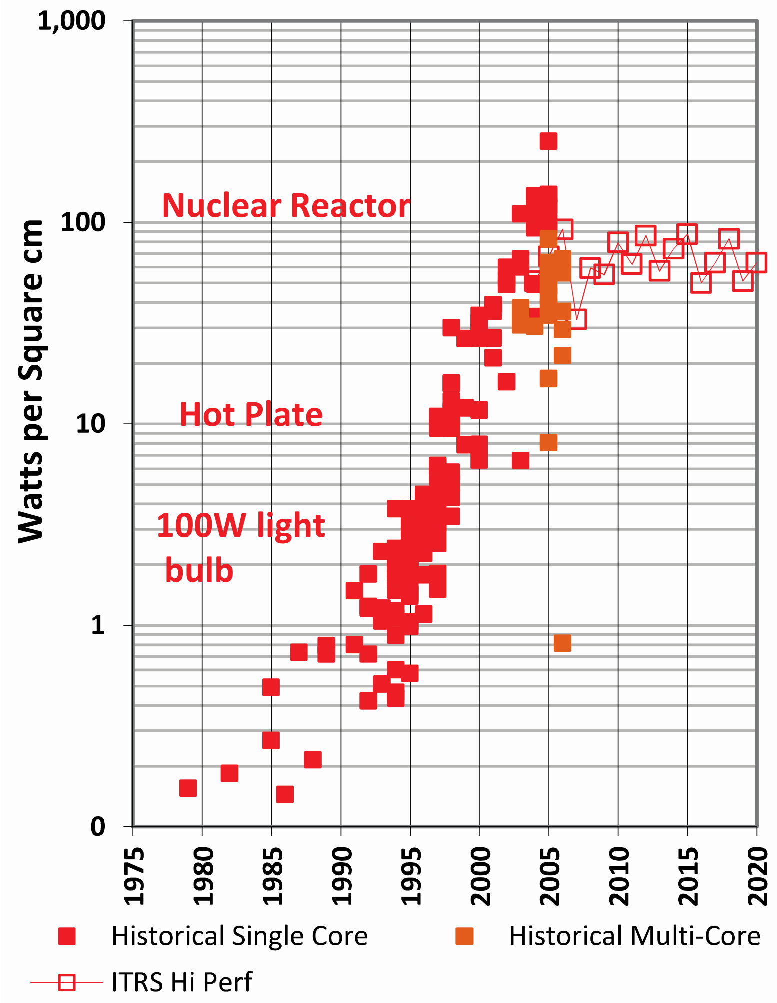
Current Bulk Synchronous Model not up to task

- Current focus is on removing sources of performance variation (jitter), is increasingly impractical
- Huge costs in power/complexity/performance to extend the life of a purely bulk synchronous model

Embrace performance heterogeneity: Study use of asynchronous computational models (e.g. SWARM, HPX, and other concepts from 1980s)

Why Wait for Exascale everything is breaking NOW!

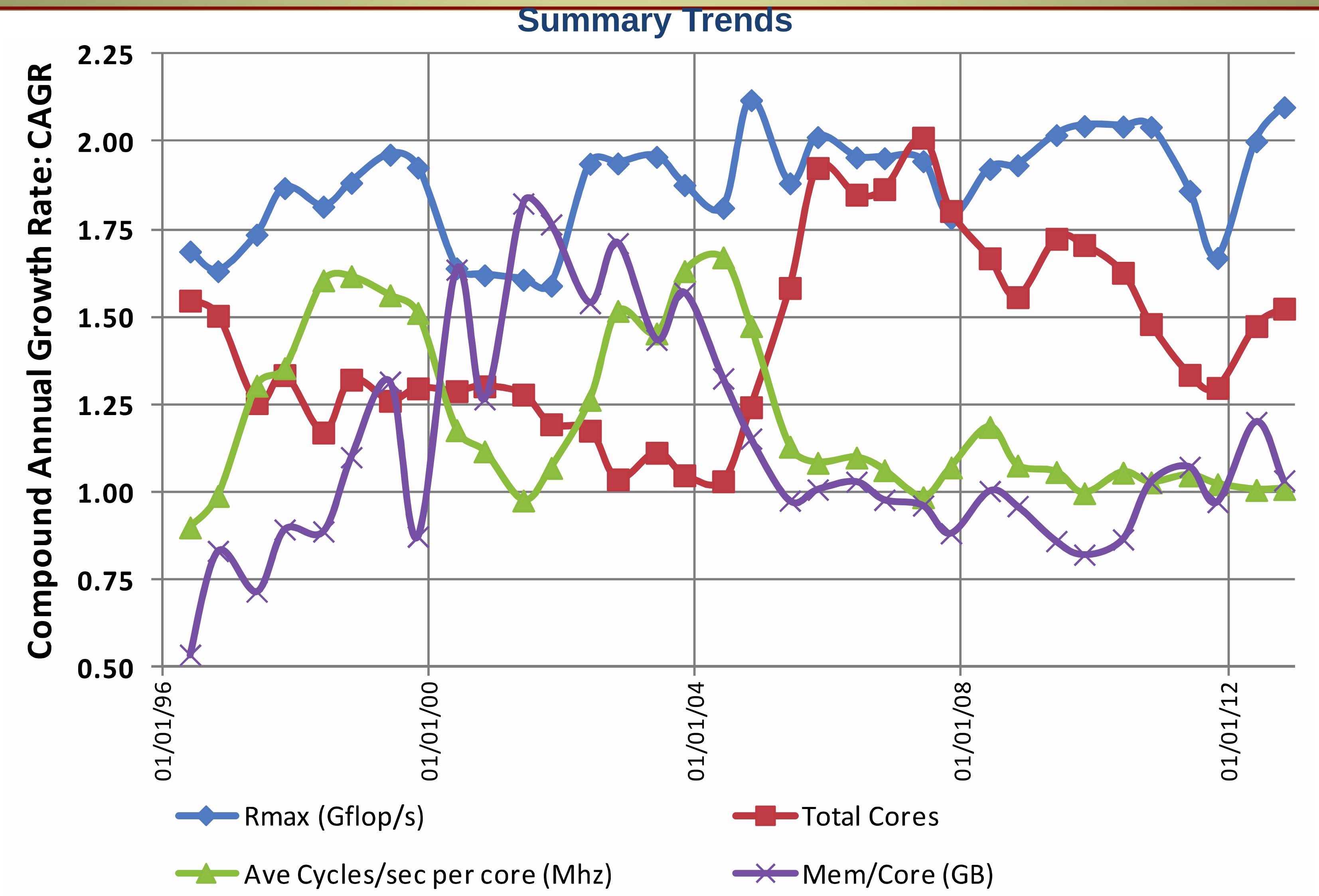
The Power and Clock Inflection Point in 2004



re Clock (MHz) (M)

Source: Kogge and Shalf, IEEE CISE

It’s the End of the World as We Know It!



Source: Kogge and Shalf, IEEE CISE

Why Wait for Exascale Machine?

The changes we are concerned about are underway NOW

Specifics of exascale machine are not as important as the design TRENDS

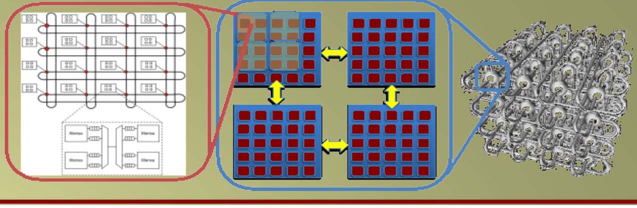
Focus on the *first derivative* of change rather than the target point design

- If you focus on target design, then it will create a brittle non-performance-portable solution
- Performance portability SHOULD be the right metric (how little code to change between generations of machines)
- *Architectural Simulation helps us with sensitivity analysis and extrapolation*

To this end, we should concentrate on what hardware abstractions correctly minimize the impact of these design trends

- How do I make # cores seemingly go away? (or scale without intervention)
- How do I express communication costs in a way that makes it easier to reason about data placement/locality without being pinned down to the specifics of one machine

Remember the Abstract Machine Model



Programming model IS, and SHOULD BE a proper reflection of the underlying machine architecture

Machine attributes are parameterized

- •Changes with each generation of machine and between different vendor implementations
- •Pmodel should target the parameterized attributes

For each parameterized machine attribute

- Ignore it: If ignoring it has no serious power/performance consequences
- Abstract it (*virtualize*): If it is well enough understood to support an automated mechanism to optimize layout or schedule
- Expose it (*unvirtualize*): If there is not a clear automated way of make decisions

Conclusions

Data layout (currently, make it more expressive)

- Need to support hierarchical data layout that closer matches architecture
- Automated method to select optimal layout is elusive, but type-system can support minimally invasive user selection of layout

Horizontal locality management (virtualize)

- Flexibly use message queues and global address space
- Give intelligent runtime tools to dynamically compute cost of data movement

Vertical data locality management (make more expressive)

- Need good abstraction for software managed memory
- Malleable memories (allow us to sit on fence while awaiting good abstraction)

Heterogeneity (virtualize)

- Its going to be there whether you want it or not
- Pushes us towards async model for computation (post-SPMD)

Parallelism (virtualize)

- Need abstraction to virtualize # processors (but must be cognizant of layout)
- For synchronous model (or sections of code) locality-aware iterators or loops enable implicit binding of work to local data.
- For async codes, need to go to functional model to get implicit parallelism
  - Helps with scheduling
  - Does not solve data layout problem