Computer Architecture for the Next Decade

(nee.. Exascale)

Adjusting to the new normal for computing

John Shalf

Department Head: Computer Science and Data Sciences (CSDS)

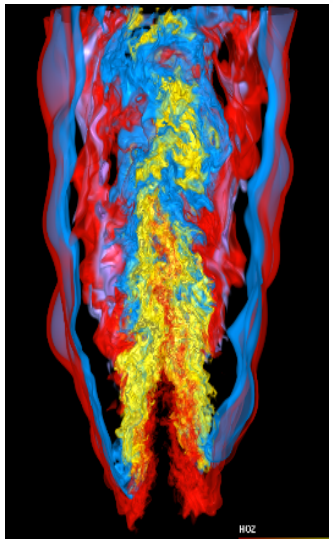
CTO: National Energy Research Scientific Computing Center (NERSC)

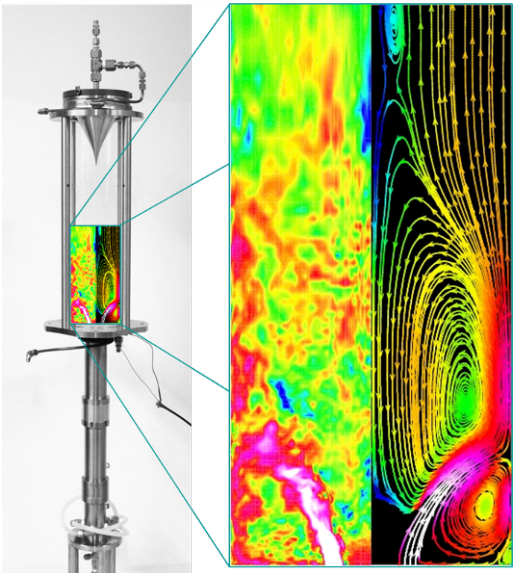
EEHPC Workshop, November 18, 2013

High End Modeling and Data Assimilation For Advanced Combustion Research

Approach: Combine unique codes and resources to maximize benefits of high performance computing for turbulent combustion research

Advanced "capability-class" solvers





DNS to investigate combustion phenomena at smallest scales

no modeling

limited applicability

LES to investigate coupling over full range of scales in experiments

minimal modeling

full geometries

Access to leading edge computational resources

CRF Computational Combustion and Chemistry Laboratory

Combustion Research and Computational Visualization Facility

EERE System:

256 Opteron™ processors, InfiniBand, 10 terabytes NFS disk storage.

Visualization Cluster:

34 Opteron™ processors with high-end graphics cards, Gigabyte Ethernet, 50 terabyte parallel file system.

BES System:

284 Opteron™ processors, InfiniBand, 15 terabytes NFS disk storage.

Joint OS-EERE Funding

DOE Office of Science Laboratories

- LBNL NERSC
- ORNL OLCF
- ANL ALCF

INCITE Program

Image co

Ofelein, Chen: Sandia 2009

Scientific Breakthroughs Enabled by Algorithms, Applications, and HPC Capability

Combustion Research has demonstrated a long history of scientific breakthroughs resulting from joint advances in Algorithms, Applications, and HPC Capability

Need for more simulation fidelity drives insatiable need for larger scale systems.

Gigaflops

Teraflops

Petaflops

Exaflops

Chemical Dynamics Theory

Kinetic Experiments

Mechanistic Experiments

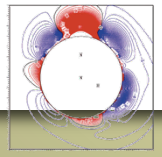
Mechanism Development

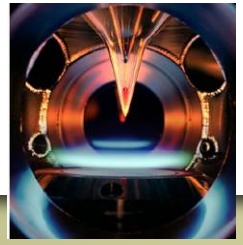
Mechanism Reduction

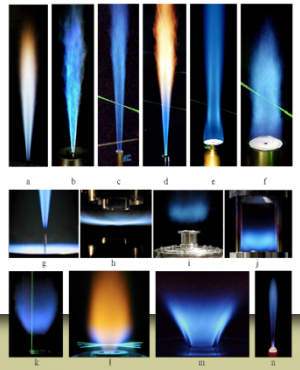
Turbulent Flame Experiments

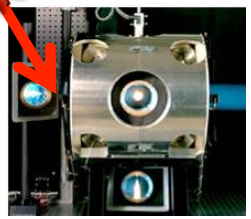
Sub-Model Validation Experiments

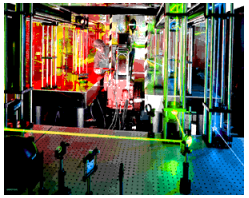
Device Validation Experiments

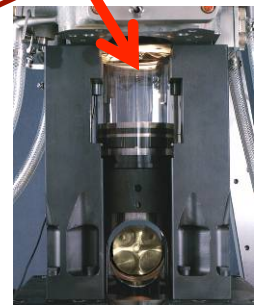








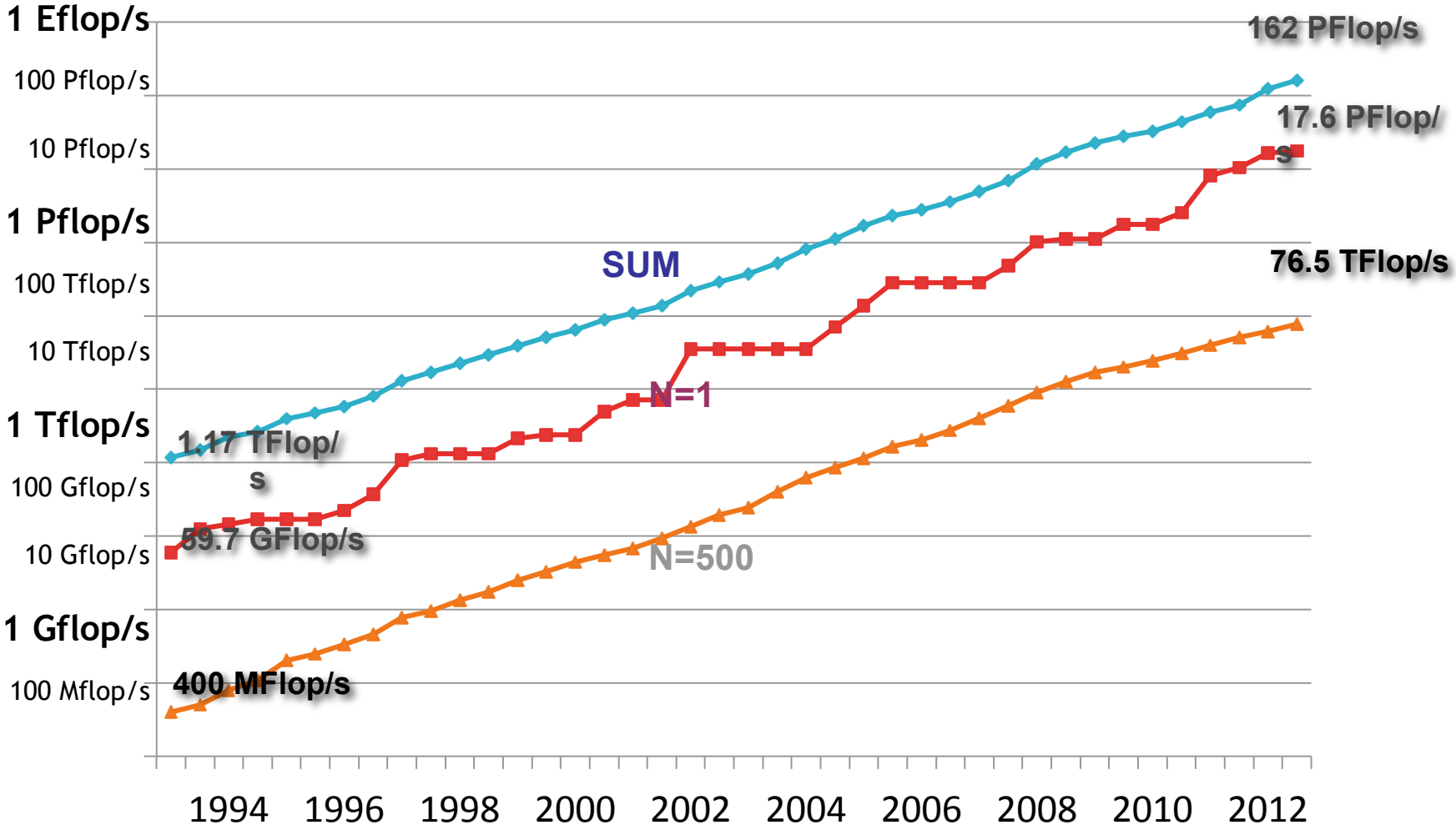




Detailed Device Models

- e.g. Gas Turbines, IC Engines, Liquid Rockets

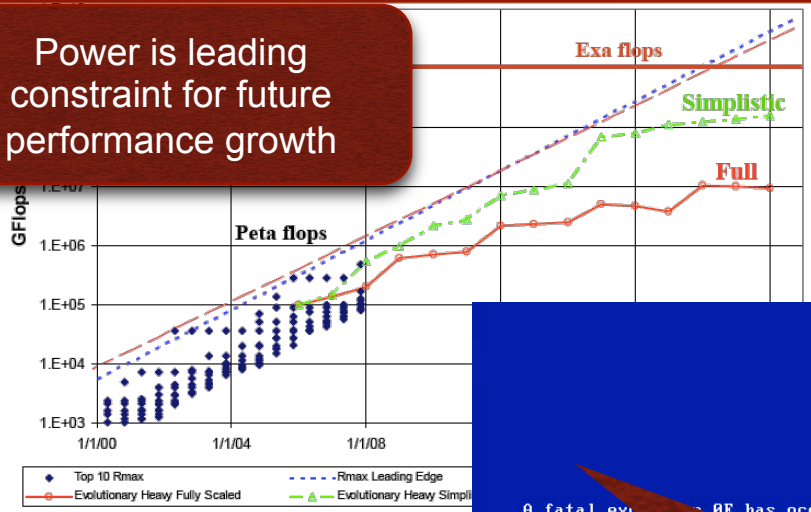
Two Decades of Exponential Performance Improvements



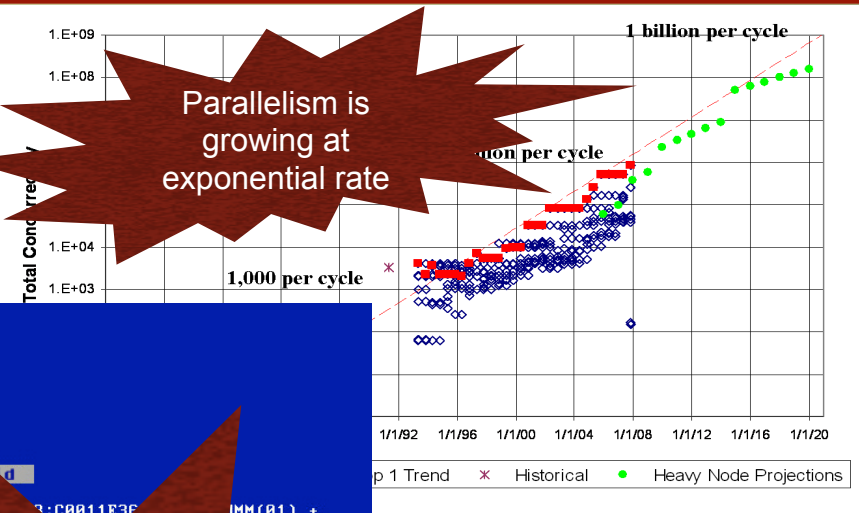
Source: TOP500 November 2012

Technology Challenges for the Next Decade

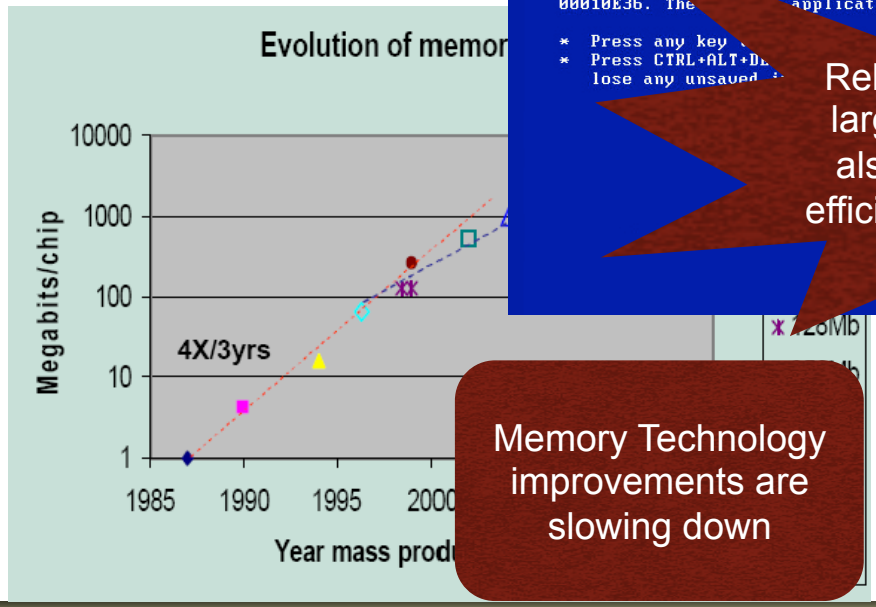
Power is leading constraint for future performance growth



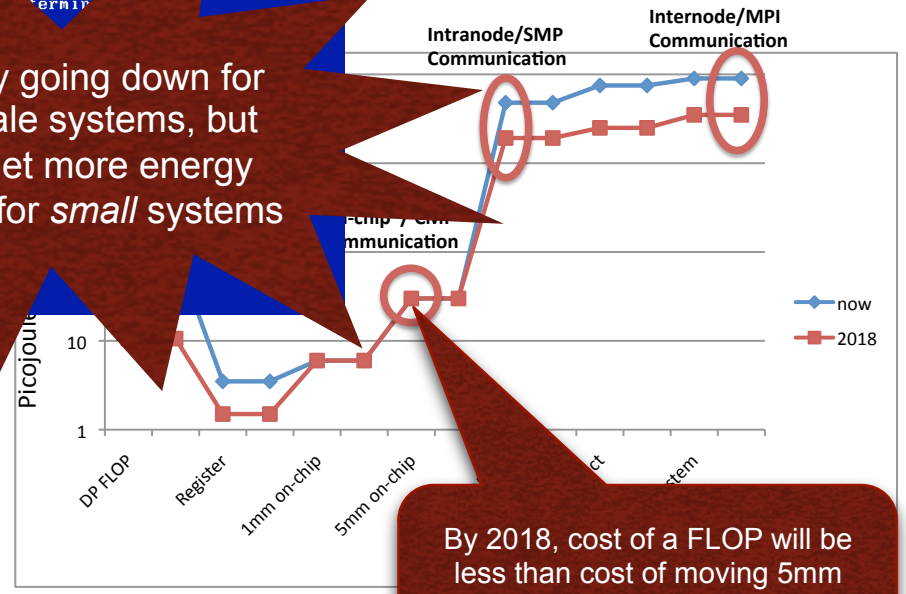
Parallelism is growing at exponential rate



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

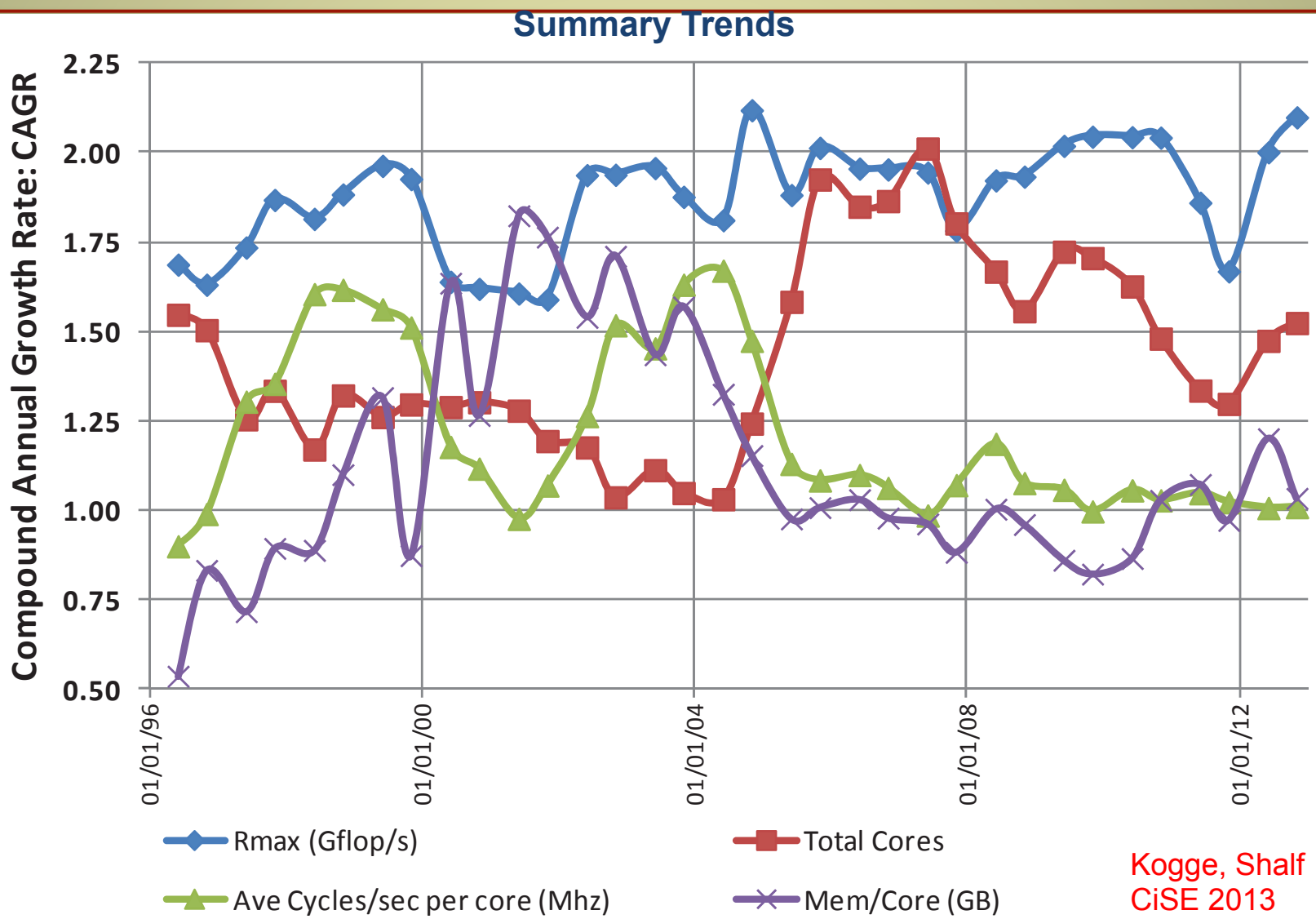


Memory Technology improvements are slowing down



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

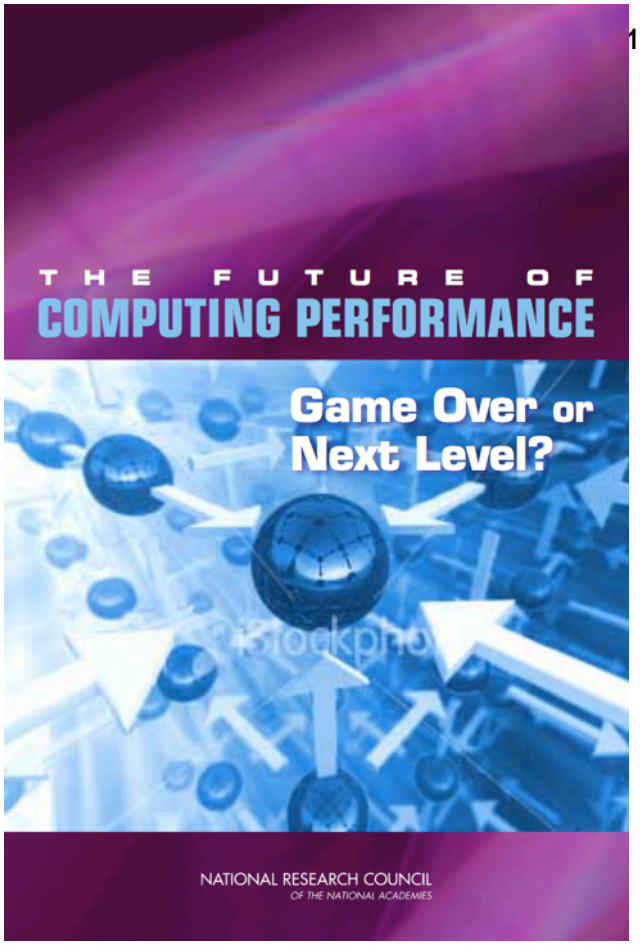
It's the End of the World as We Know It!

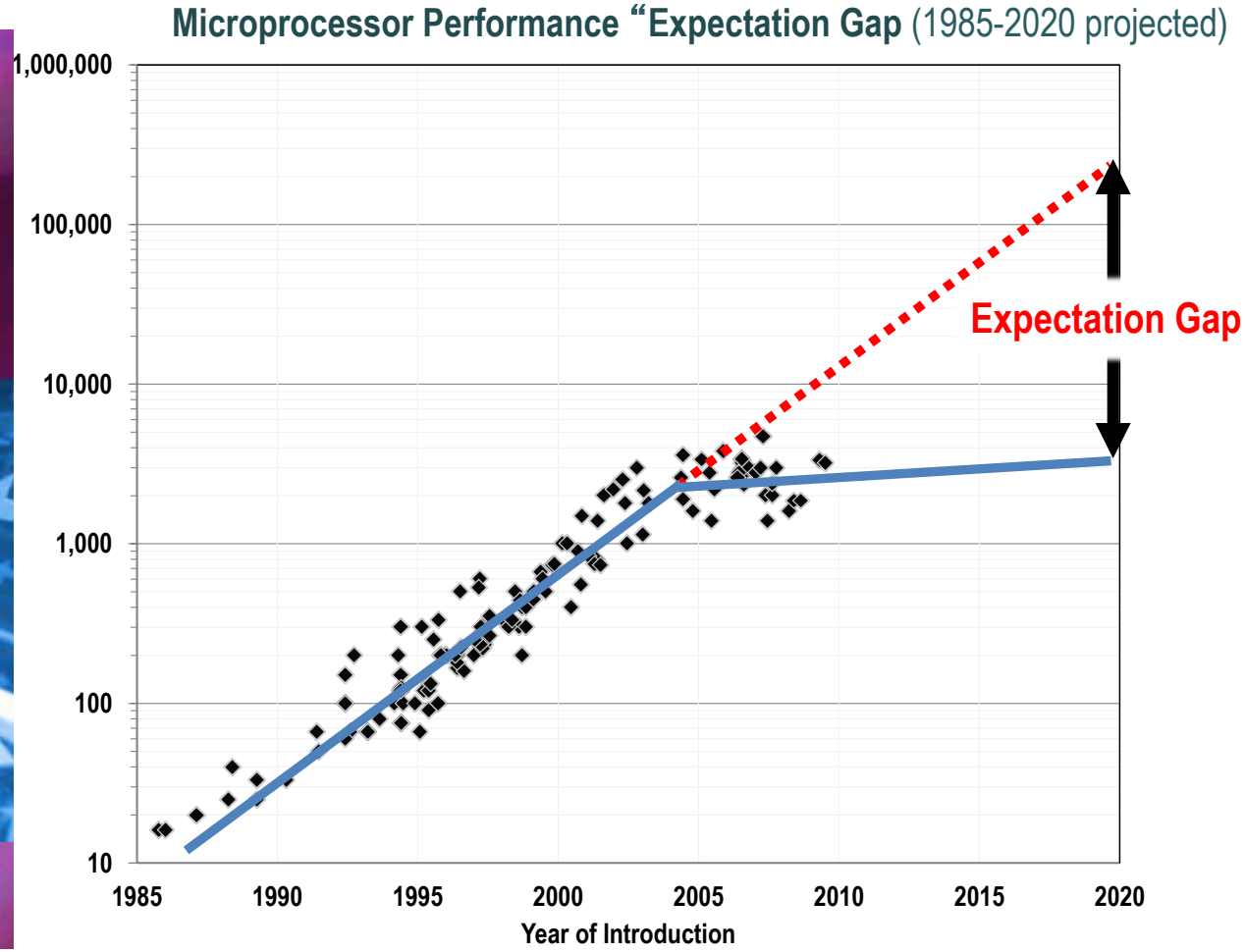


Kogge, Shalf
CiSE 2013

Source: Kogge and Shalf, IEEE CISE 2013

Computing Crisis is Not Just about Exascale





Industry motivated, path forward is unclear

Whats wrong with current HPC Systes?

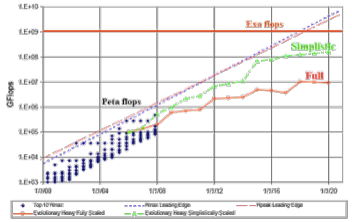
Designed for Constraints from 30 years ago! ***(wrong target!!)***

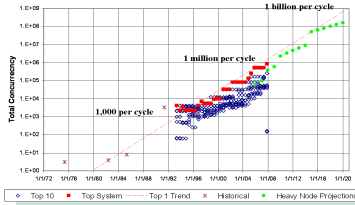
Old Constraints

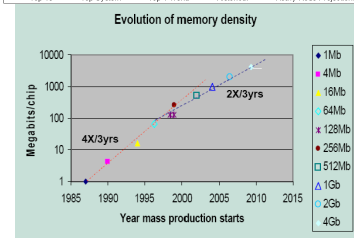
- ***Peak clock frequency*** *as primary limiter for performance improvement*
- **Cost:** *FLOPs* are biggest cost for system: *optimize for compute*
- **Concurrency**: Modest growth of parallelism by adding nodes
- **Memory scaling***: maintain byte per flop capacity and bandwidth*
- **Locality***: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

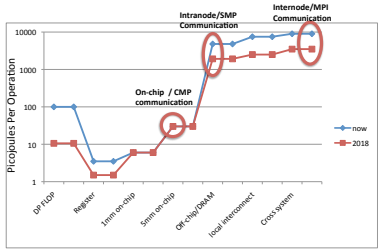
New Constraints

- ***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

The Programming Systems Challenge

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*

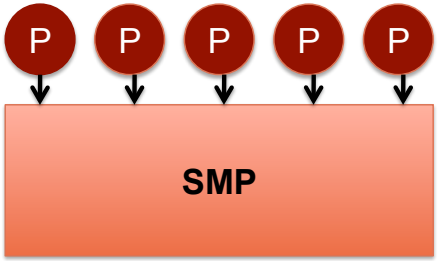
Technology changes have deep and pervasive effect on ALL of our software systems *(and how we program them)*

- *Change in costs for underlying system affect what we expose*
- *What to **virtualize***
- *What to make more **expressive/visible***
- *What to **ignore***

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

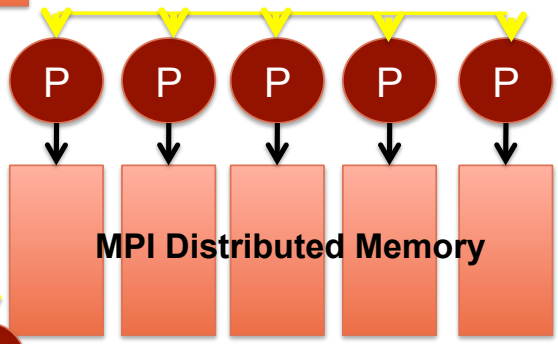
Equal cost SMP/PRAM model

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



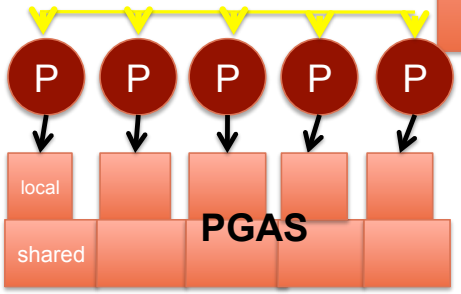
Cluster: Distributed memory model

- CSP: Communicating Sequential Processes
- No unified memory
- int [localNX][localNY][localNZ];



MPI+X: (HCSP)

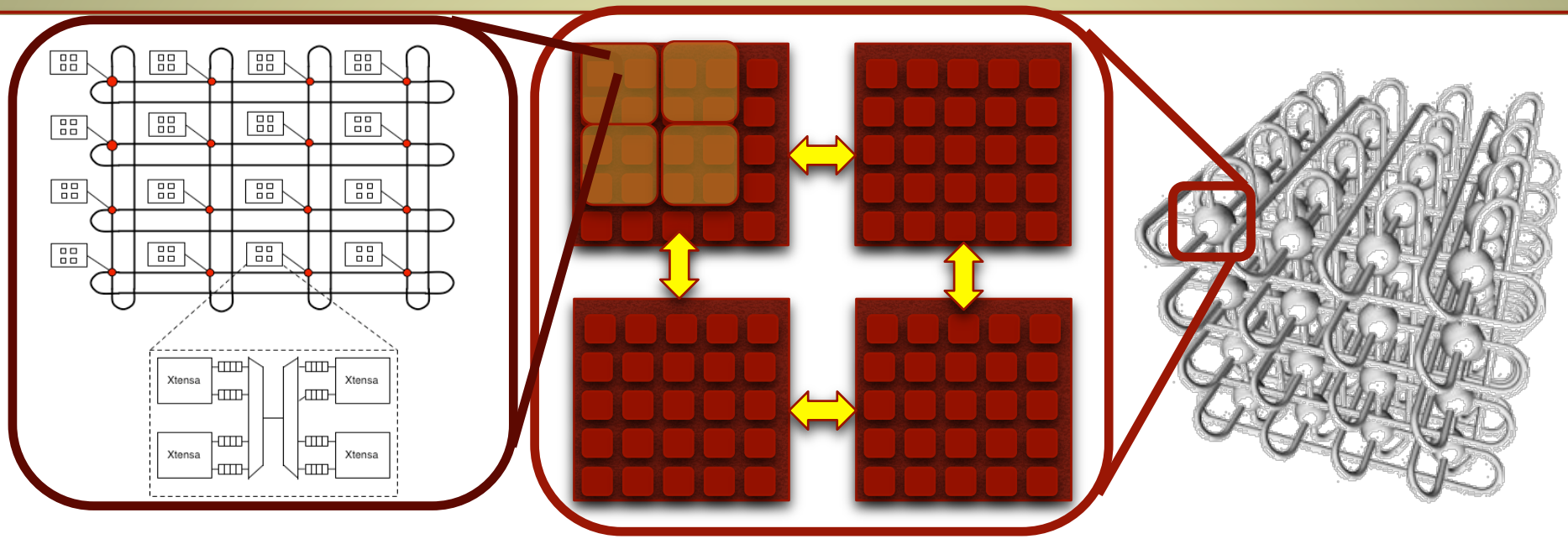
- Data is LOCAL or REMOTE
- node[#] int [nx][ny][nz];



Whats Next?

Parameterized 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)

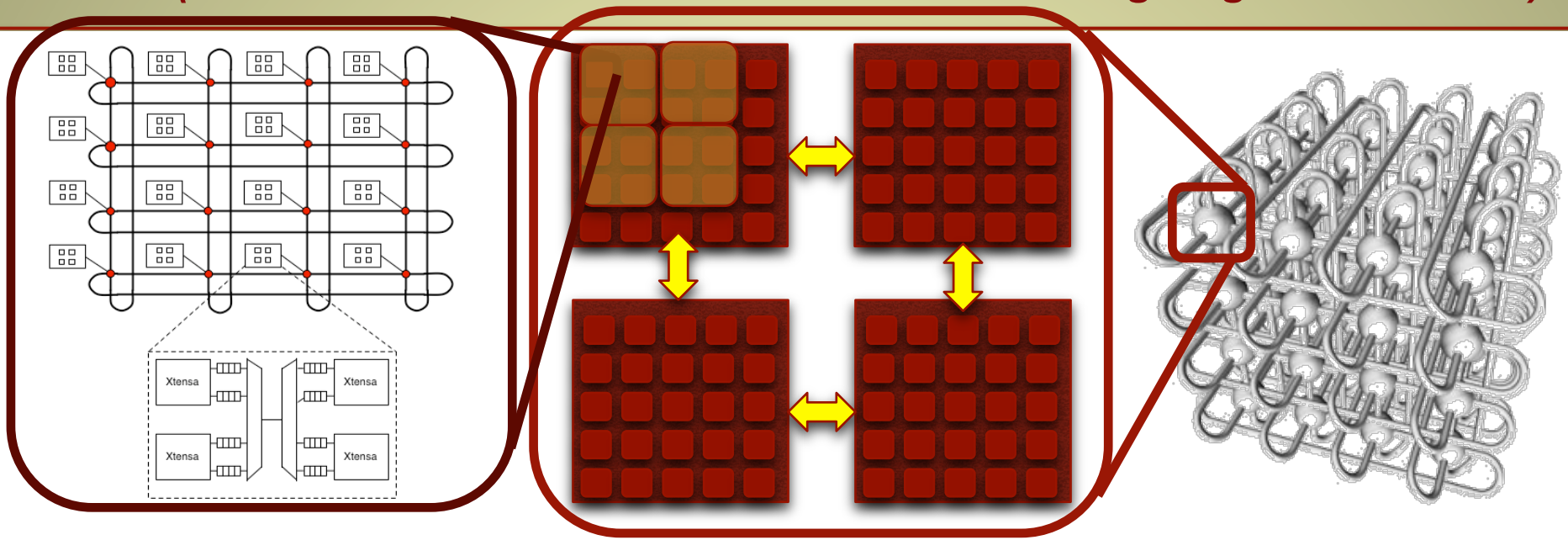
Fault Model for Node
- FIT rates, Kinds of faults
- Granularity of faults/recovery

Interconnect
- Bandwidth/Latency/Overhead
- Topology

Primitives for data movement/ sync
- Global Address Space or messaging?
- Synchronization primitives/Fences

Abstract Machine Model

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



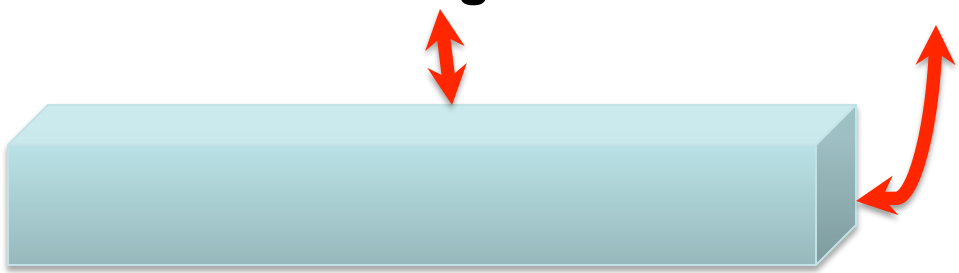
For each parameterized machine attribute, can

- **Ignore it:** *If ignoring it has no serious power/performance consequences*
- **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)
- **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)

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

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

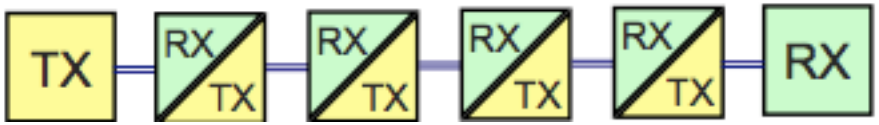
- **Cost to move a bit on copper wire:**
  - **Power = Bitrate * Length / cross-section area**



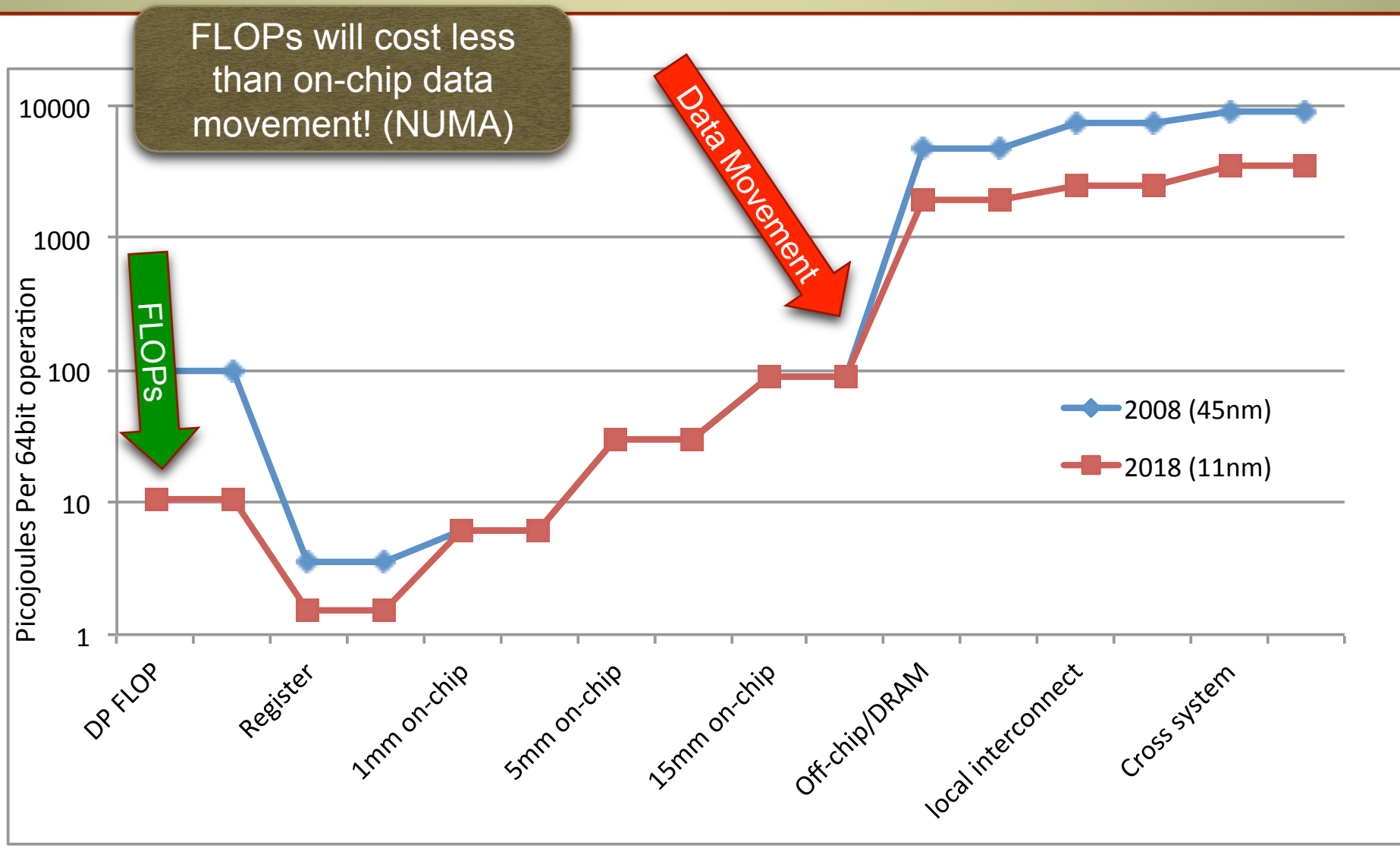
- **Wire data capacity constant as feature size shrinks**
- ***Cost to move bit proportional to distance***
- ***~1TByte/sec max feasible off-chip BW (10GHz/pin)***
- ***Photonics reduces distance-dependence of bandwidth***

Photonics requires no redrive and passive switch little power

Copper requires to signal amplification even for on-chip connections



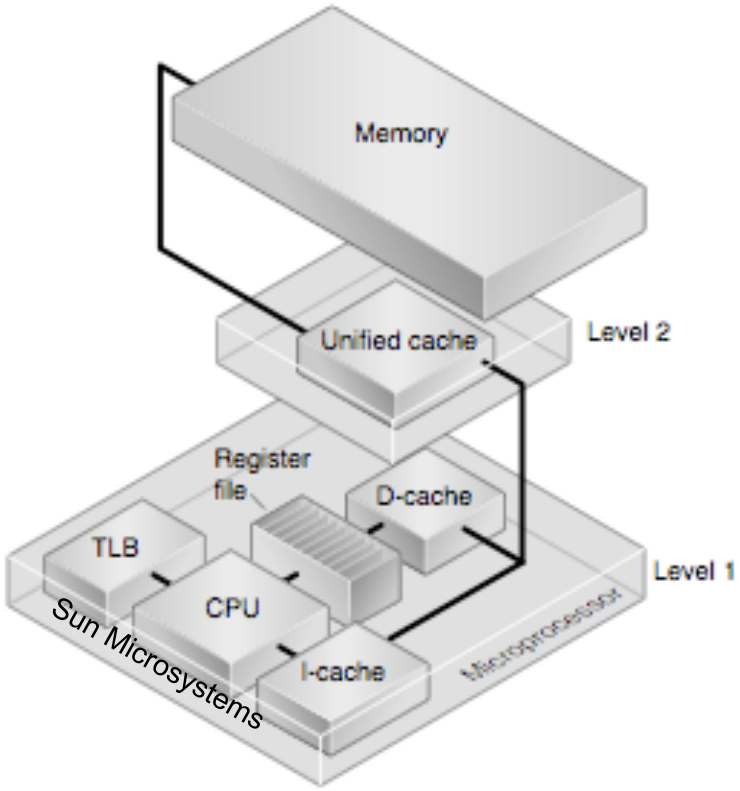
Cost of Data Movement Increasing Relative to Ops

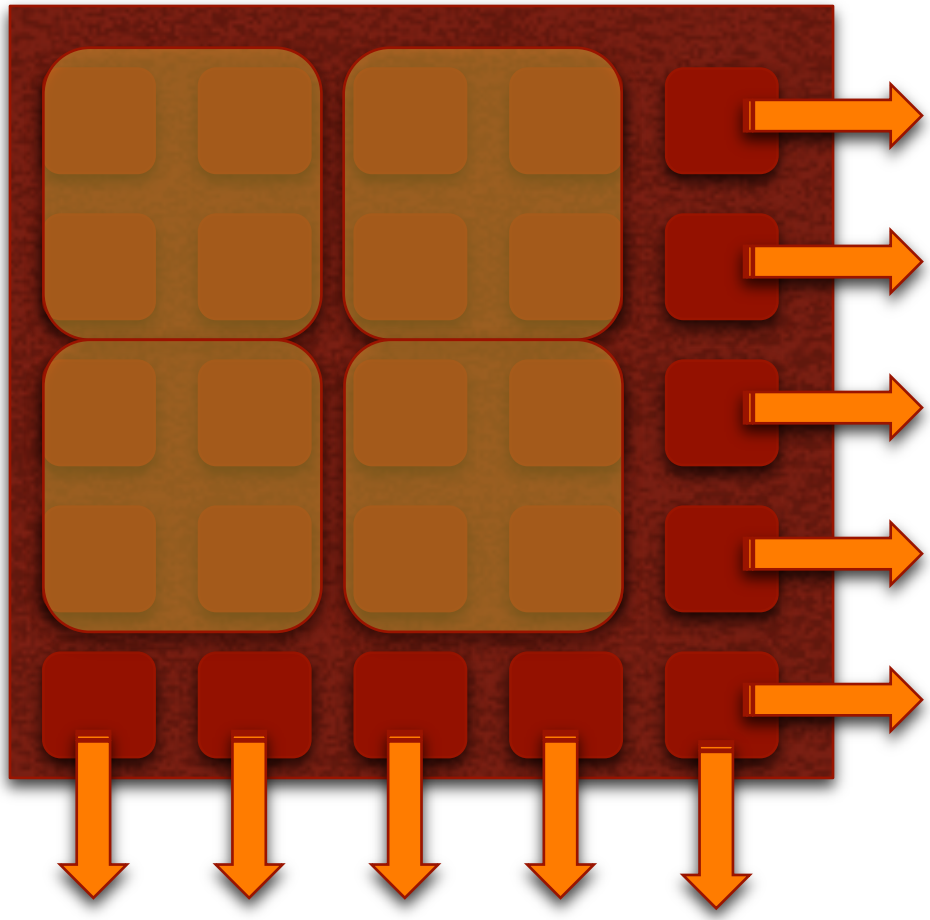


Data Locality Management

Vertical Locality Management
(spatio-temporal optimization)

Horizontal Locality Management
(topology optimization)





Current Practices (2-level Parallelism)

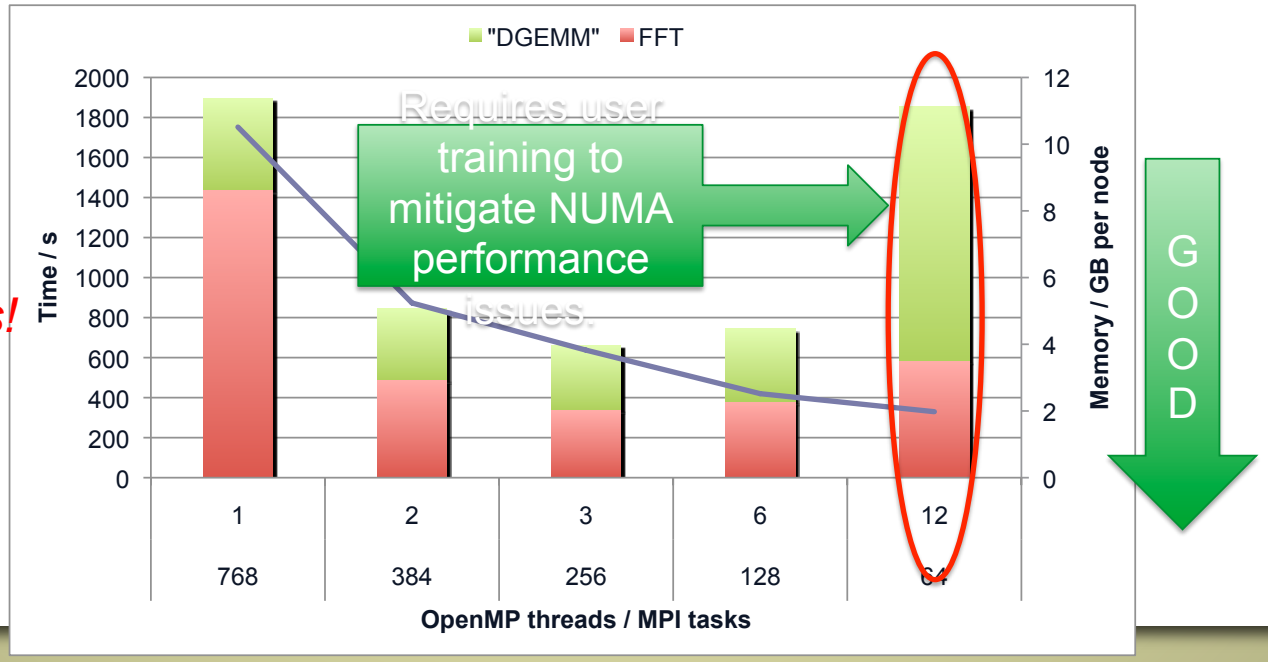
NUMA Effects Ignored (with huge consequence)

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!



Expressing Hierarchical Layout

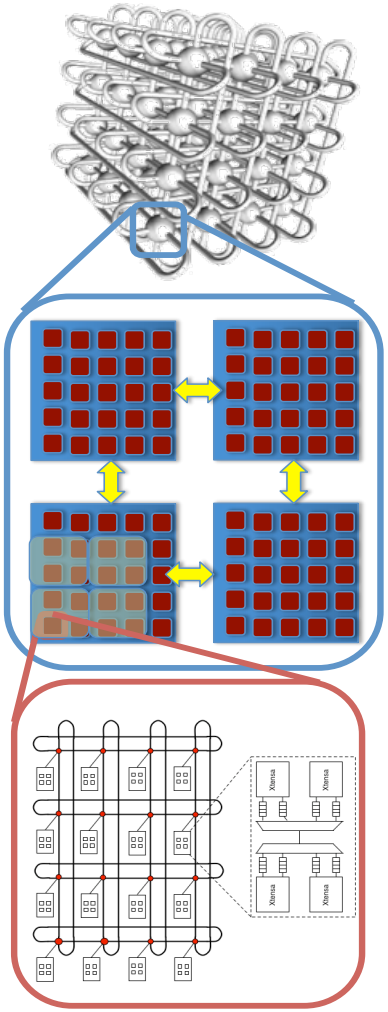
Old Model (OpenMP)

- Describe how to parallelize loop iterations
- Parallel “DO” divides loop iterations evenly among processors
- . . . but where is the data located?

New Model (Data-Centric)

- Describe how data is laid out in memory
- Loop statements operate on data where it is located
- Similar to MapReduce, but need more sophisticated descriptions of data layout for scientific codes

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

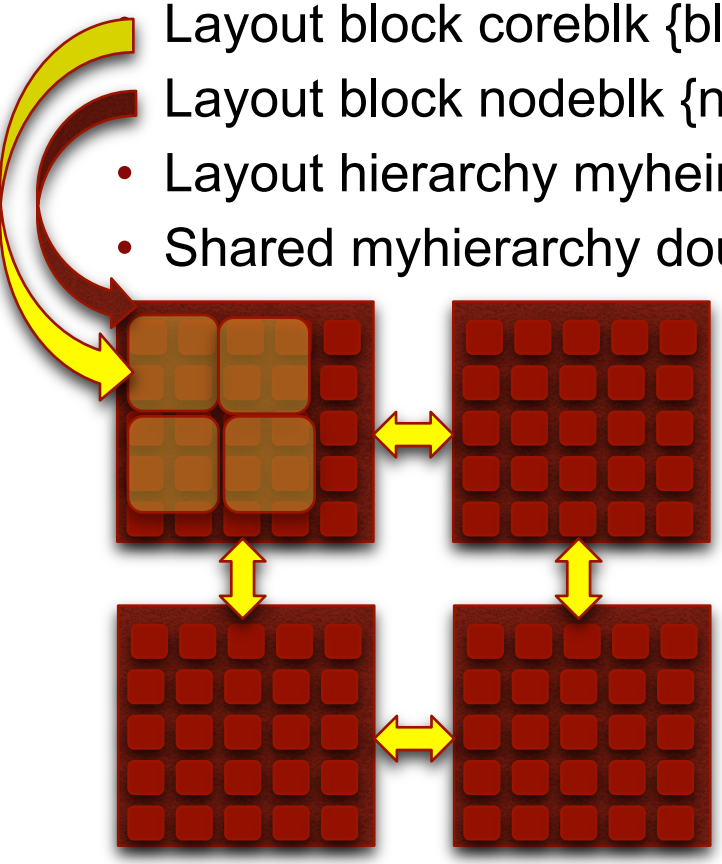


Data-Centric Programming Model

(current compute-centric models are mismatched with emerging hardware)

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)

U.S. DEPARTMENT OF ENERGY | Office of Science

Tiling Formulation: *abstracts both data locality and massive parallelism (both exascale challenges)*

Expose massive degrees of parallelism through domain decomposition

- Represent an atomic unit of work
- Task scheduler works on tiles

Core concept for data locality

- Vertical data movement
  - *Hierarchical partitioning*
- Horizontal data movement
  - *Co-locate tiles sharing the same data by respecting tile topology*

Multi-level parallelism

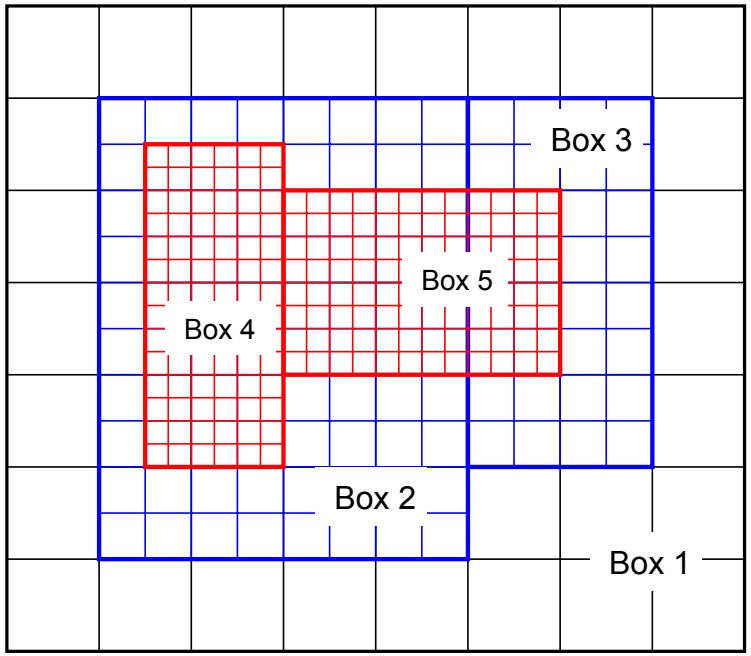
- Coarse-grain parallelism: across tiles
- Fine-grain parallelism: vectorization, instruction ordering etc. within a tile

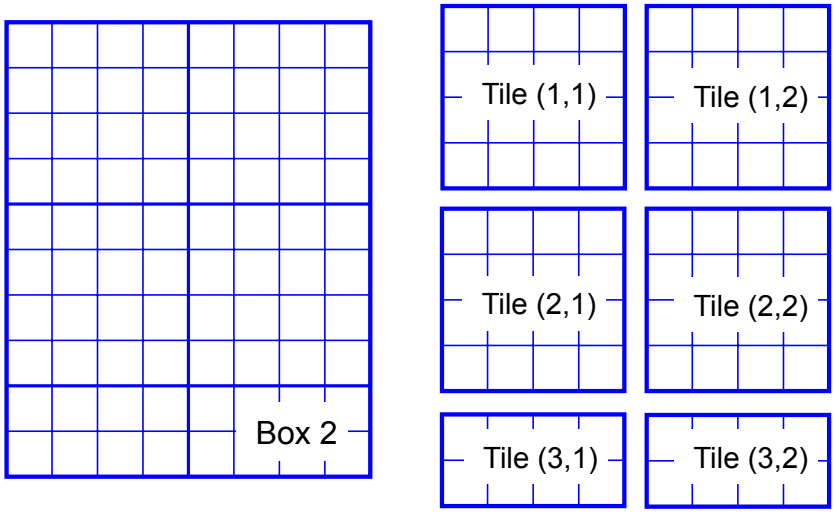
TiDA: Tiling as a Durable Abstraction

(Didem Unat, SC13)

TiDA centralizes and parameterizes the tiling information at the data structure

- Direct approach for memory affinity management for data locality
- Expose massive degrees of parallelism through domain decomposition



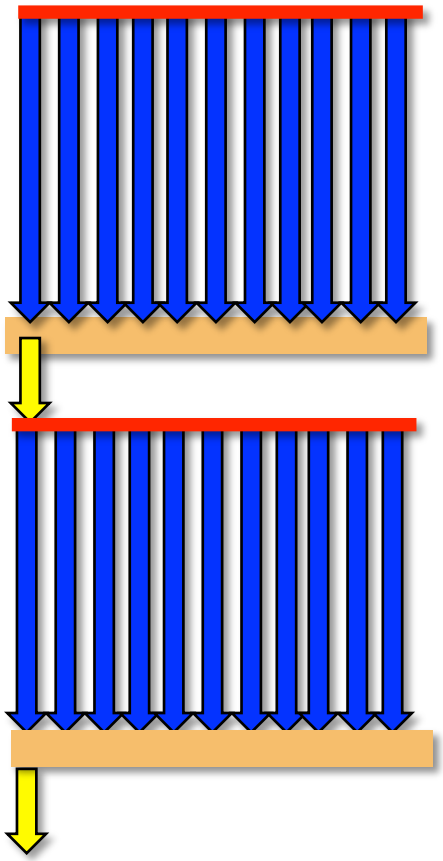


Tiled Box 2

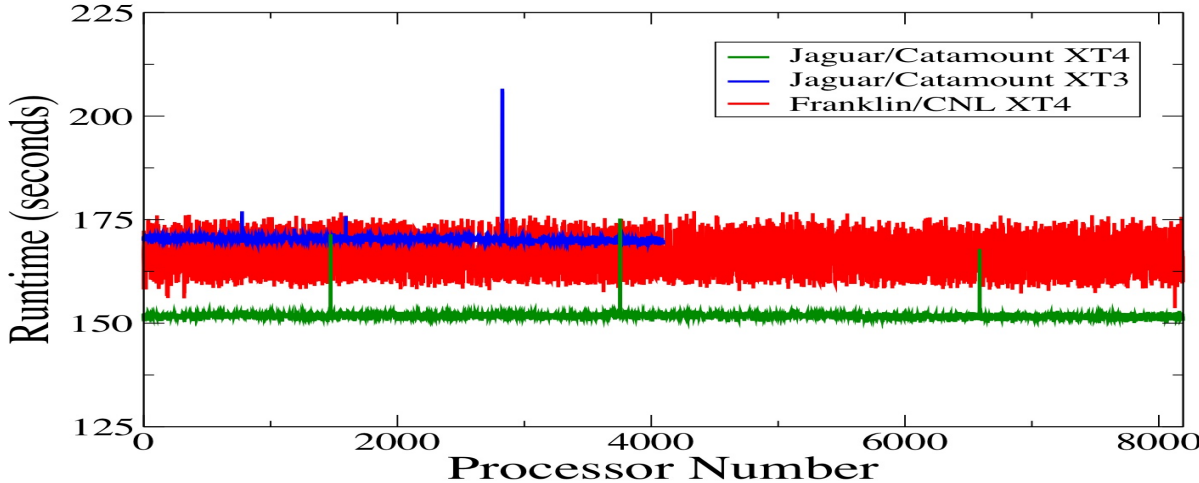
Heterogeneity / Inhomogeneity Async Programming Models?

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

Bulk Synchronous Execution

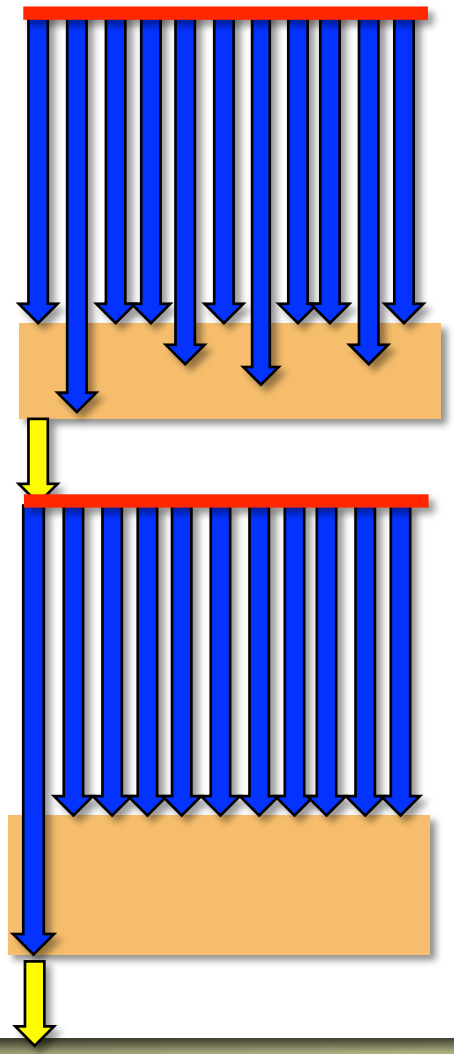


- **Heterogeneous compute engines (hybrid/GPU computing)**
- **Fine grained power mgmt. makes homogeneous cores look heterogeneous**
  - *thermal throttling – no longer guarantee deterministic clock rate*
- **Nonuniformities in process technology creates non-uniform operating characteristics for cores on a CMP**
  - Near Threshold Voltage (NTV)
- **Fault resilience introduces inhomogeneity in execution rates**
  - *error correction is not instantaneous*
  - *And this will get WAY worse if we move towards software-based resilience*

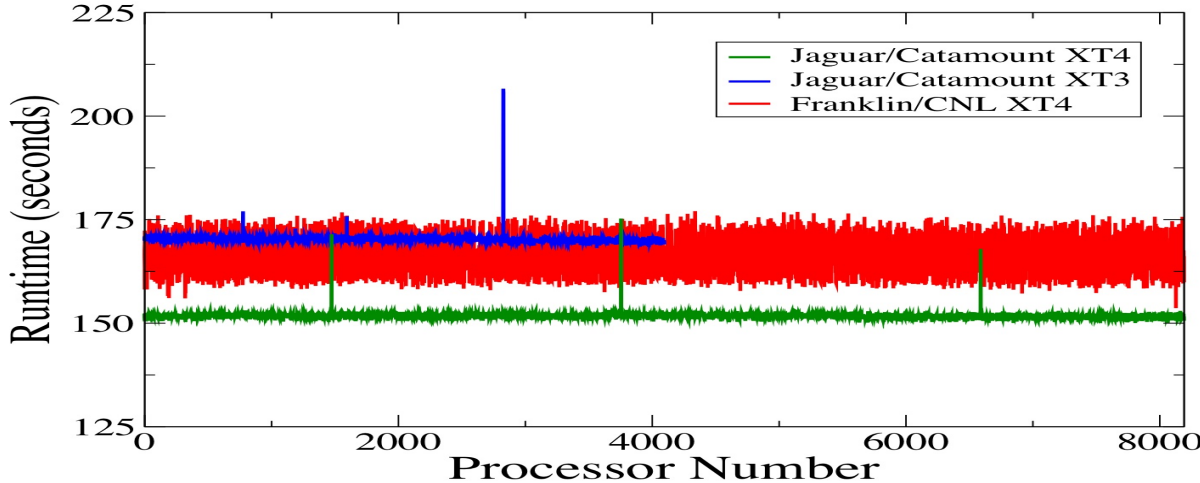


Assumptions of Uniformity is Breaking
(many new sources of heterogeneity)

Bulk Synchronous Execution



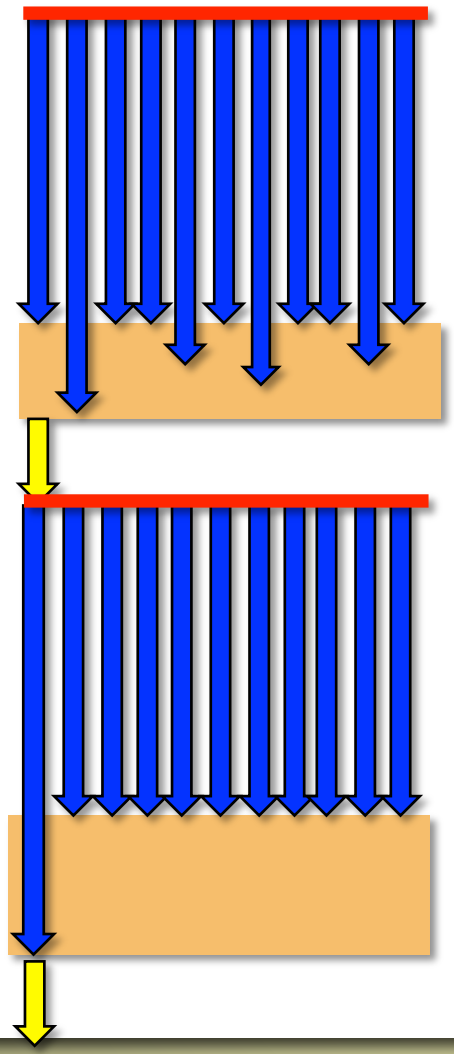
- **Heterogeneous compute engines (hybrid/GPU computing)**
- **Fine grained power mgmt. makes homogeneous cores look heterogeneous**
  - *thermal throttling – no longer guarantee deterministic clock rate*
- **Nonuniformities in process technology creates non-uniform operating characteristics for cores on a CMP**
  - Near Threshold Voltage (NTV)
- **Fault resilience introduces inhomogeneity in execution rates**
  - *error correction is not instantaneous*
  - *And this will get WAY worse if we move towards software-based resilience*



Near Threshold Voltage (NTV): *Shekhar Borkar (Intel)*

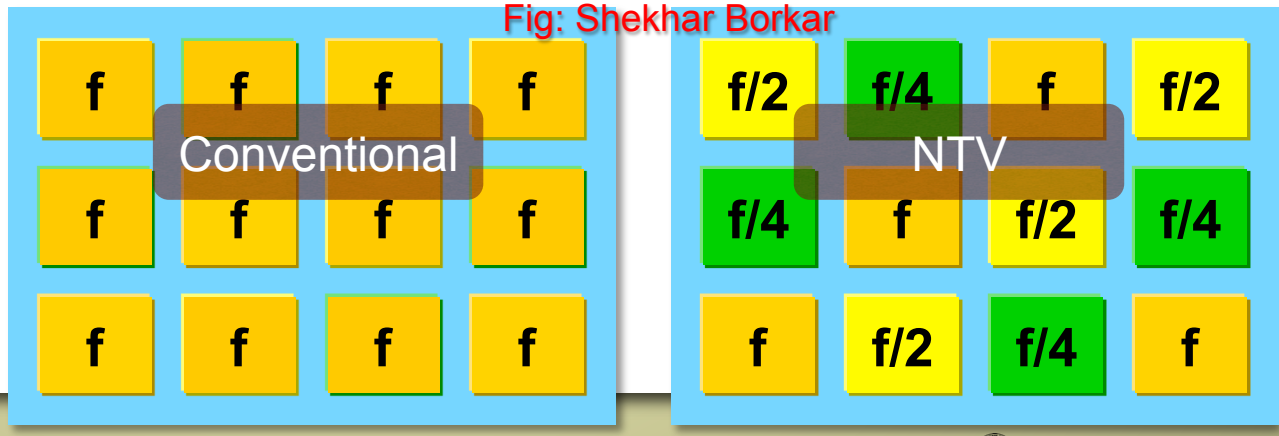
The really big opportunities for energy efficiency require codesign!

Bulk Synchronous Execution



- **Heterogeneous compute engines (hybrid/GPU computing)**
- **Fine grained power mgmt. makes homogeneous cores look heterogeneous**
  - *thermal throttling – no longer guarantee deterministic clock rate*
- **Nonuniformities in process technology creates non-uniform operating characteristics for cores on a CMP**
  - **Near Threshold Voltage (NTV)**
- **Fault resilience introduces inhomogeneity in execution rates**
  - *error correction is not instantaneous*
  - *And this will get WAY worse if we move towards software-based resilience*

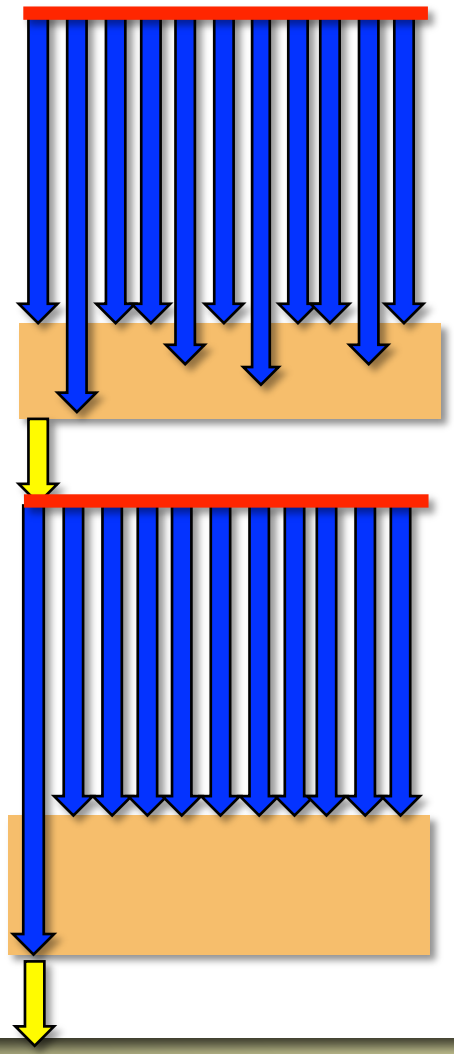
Fig: Shekhar Borkar



Near Threshold Voltage (NTV): *Shekhar Borkar (Intel)*

The really big opportunities for energy efficiency require codesign!

Bulk Synchronous Execution



Improving energy efficiency or performance of individual components doesn't really need co-design

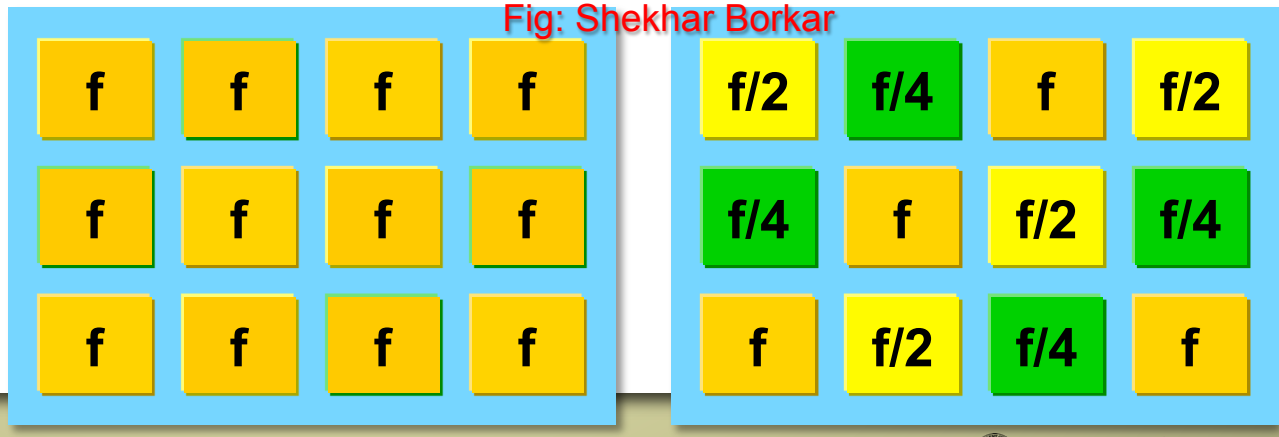
- Memory is faster, then odds are that the software will run faster
- if its better, that's good!

The really *big* opportunities to improve energy efficiency may require a shift in how we program systems

- This requires codesign to evalute the hardware and new software together
- HW/SW Interaction unknown (requires HW/SW codesign)

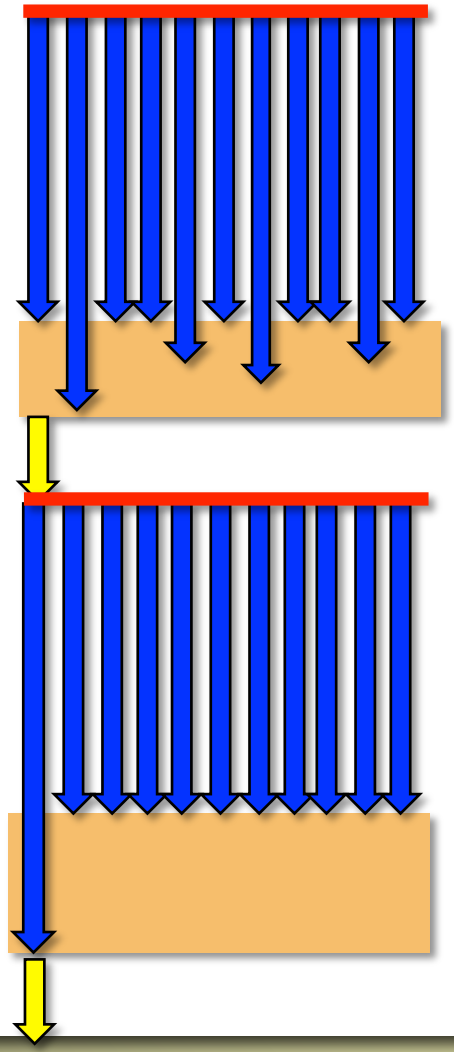
If software CANNOT exploit these radical hardware concepts (such as NTV), then it would be better to not have done anything at all!

Fig: Shekhar Borkar

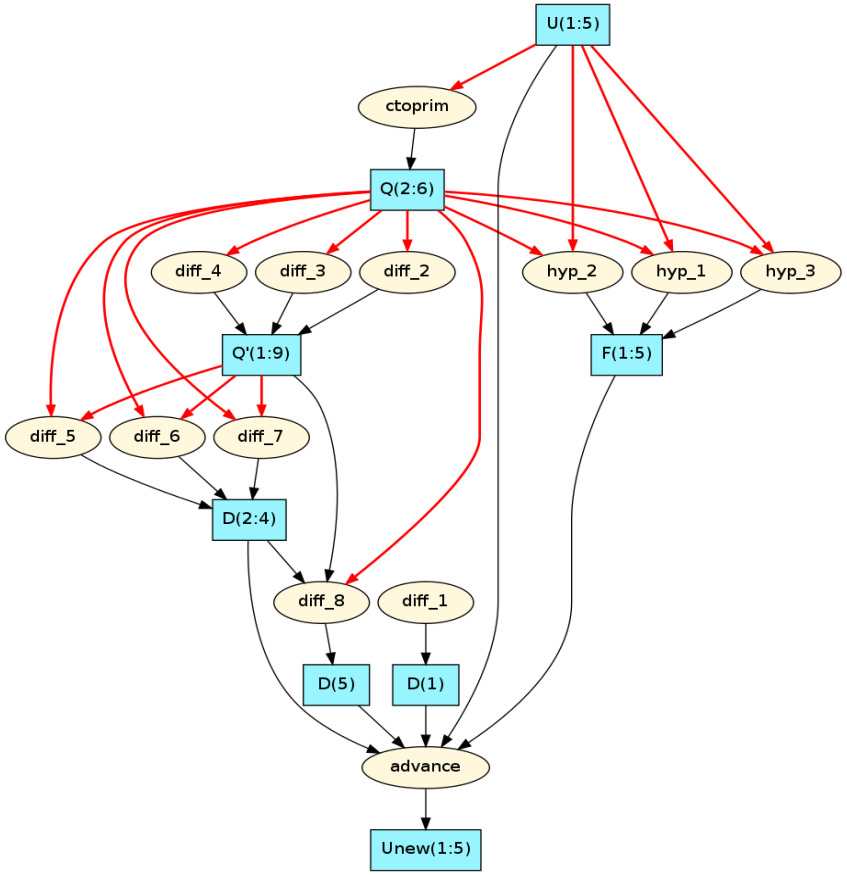


Assumptions of Uniformity is Breaking
(many new sources of heterogeneity)

Bulk Synchronous Execution Model



Asynchronous Execution Model



Conclusions on Heterogeneity

Sources of performance heterogeneity increasing

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

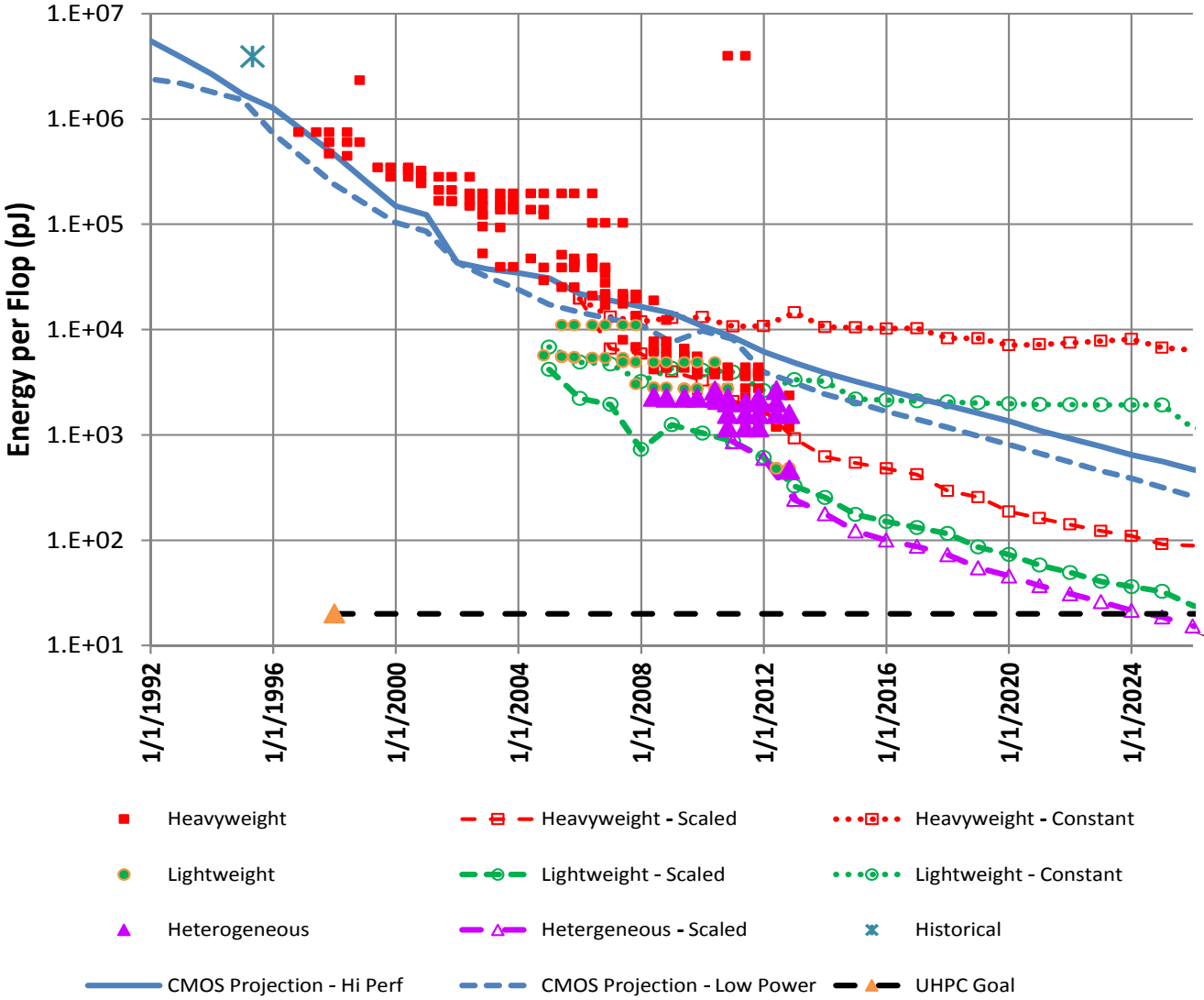
Current Bulk Synchronous Model not up to task

- 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)

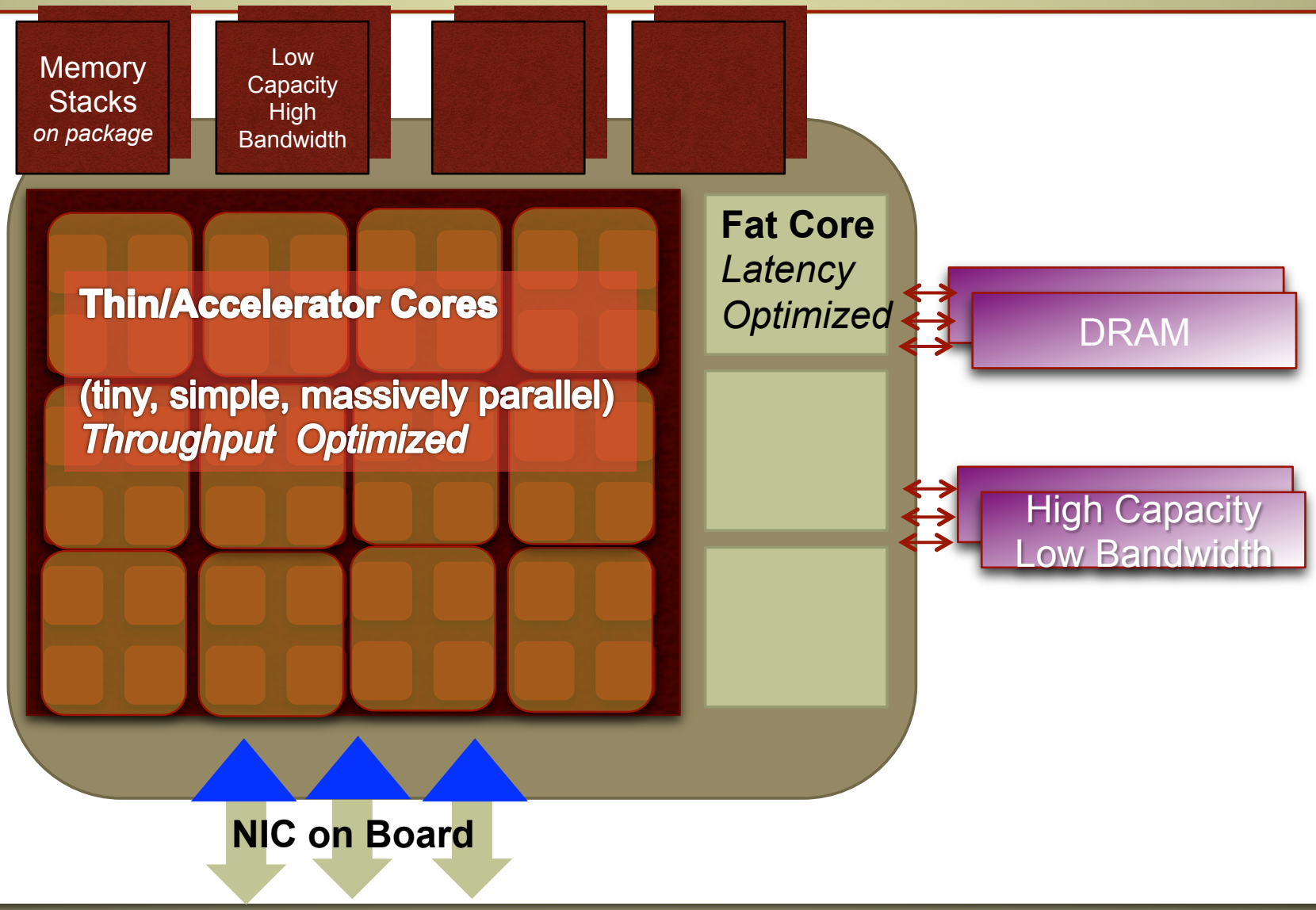
Hybrid Architectures:

Moving from side-show to necessity



Hybrid is the only approach that crosses the exascale finish line

Future Node Architecture (System on Chip)



OpenSoC: Abstract Fabric

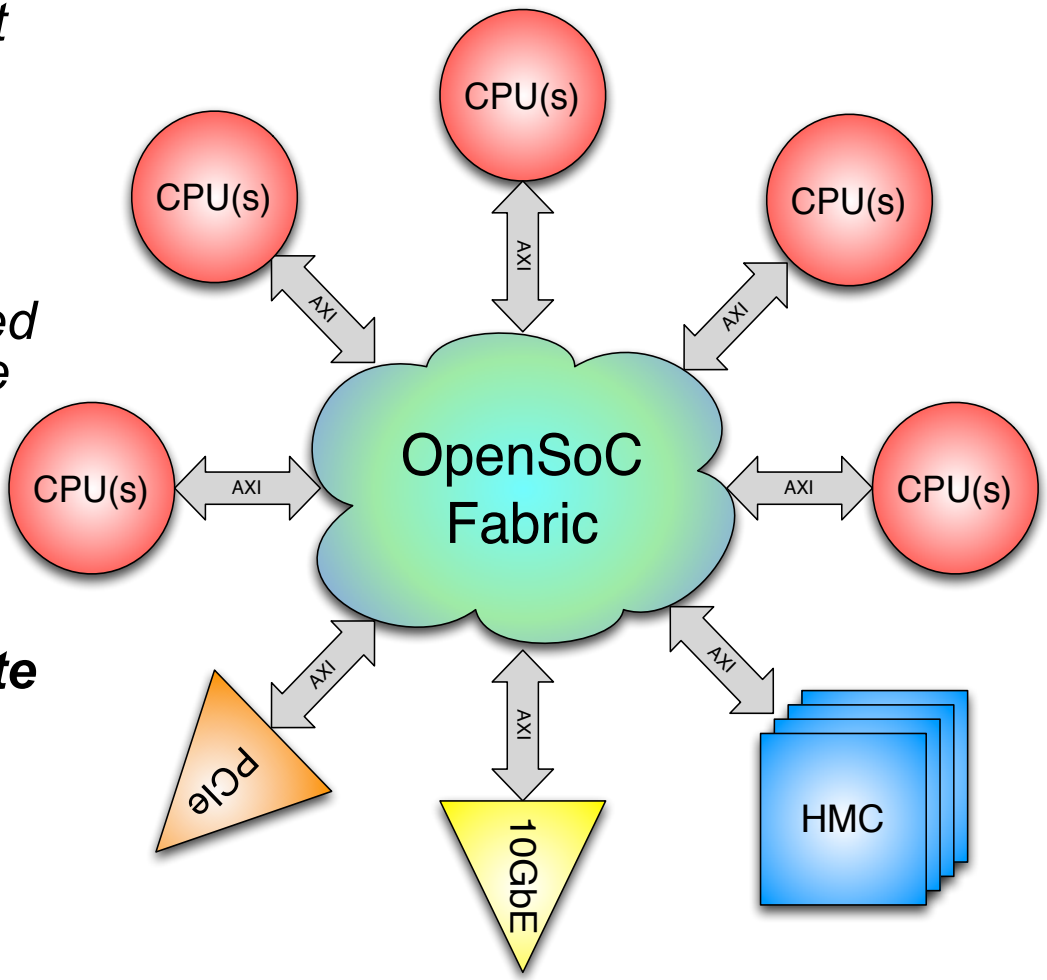
System-on-Chip (SoC) could revolutionize energy efficient computing

Seymour Cray 1977: *"Don't put anything in to a supercomputer that isn't necessary."*

Mark Horowitz 2007*: "Years of research in low-power embedded computing have shown only one design technique to reduce power: reduce waste."*

SoC Revolution enables us to achieve goal of reducing waste

- *Enable us to include ONLY what we need for HPC.*
- *Tighter component integration*
- *Fewer losses for inter-chip wiring for peripherals*



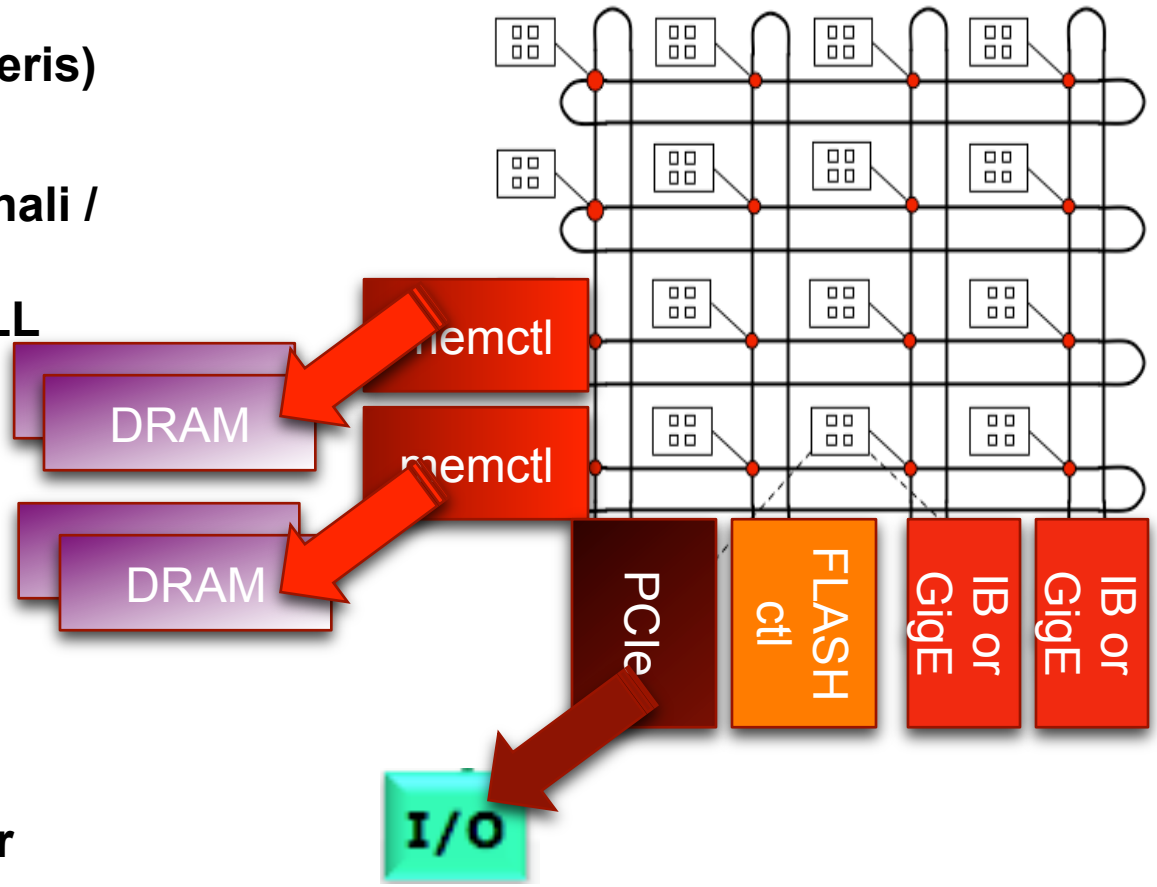
Building an SoC (System on Chip) from IP Logic Blocks

Lego circuit blocks with a some extra integration and verification cost
Include only what you need (and no more).

Processor Core (ARM, Tensilica, MIPS deriv)
With extra "options" like DP FPU, ECC

OpenSoC Fabric (ARM or Arteris)

DDR memory controller (Denali / Cadence, SiCreations)
+ Phy and Programmable PLL



PCIe Gen3 Root complex

Integrated FLASH Controller

10GigE or IB DDR 4x Channel

Conclusions

Emerging hardware constraints are increasingly mismatched with our current programming paradigm

- Current emphasis is on preserving FLOPs
- The real costs now are not FLOPs… it is data movement
- Requires shift to a data-locality centric programming paradigm and hardware features to support it

Technology Changes Fundamentally Disrupt our Programming Environments

- The programming environment and associated "abstract machine model" is a reflection of the underlying machine architecture
- Therefore, design decisions can have deep effect your entire programming paradigm
- The BIGGEST opportunities in energy efficiency and performance improvements require HW and SW considered together (codesign)

Performance Portability Should be Top-Tier Metric for codesign

- Know what to **IGNORE**, what to **ABSTRACT**, and what to make more **EXPRESSIVE**

The End

For more information go to

http://www.cal-design.org/
http://www.nersc.gov/
http://crd.lbl.gov/

Bonus:

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

Summary

- There is progress in Exascale with many projects now focused and on their way, e.g. FastForward, Xstack, and Co-Design Centers in the U.S.
- HPC has moved to low power processing, and the processor growth curves in energy-efficiency could get us in the range of exascale feasibility
- Memory and data movement are still more open challenges
- Programming model needs to address heterogeneous, massive parallel environment, as well as data locality
- Exascale applications will be challenge just because their sheer size and the memory limitations

DOE Strategy for Exascale Computing

Designing the computing environment for the future

Objective: *Enable DOE scientists and engineers to use the most advanced computational hardware and software for discovery science.*

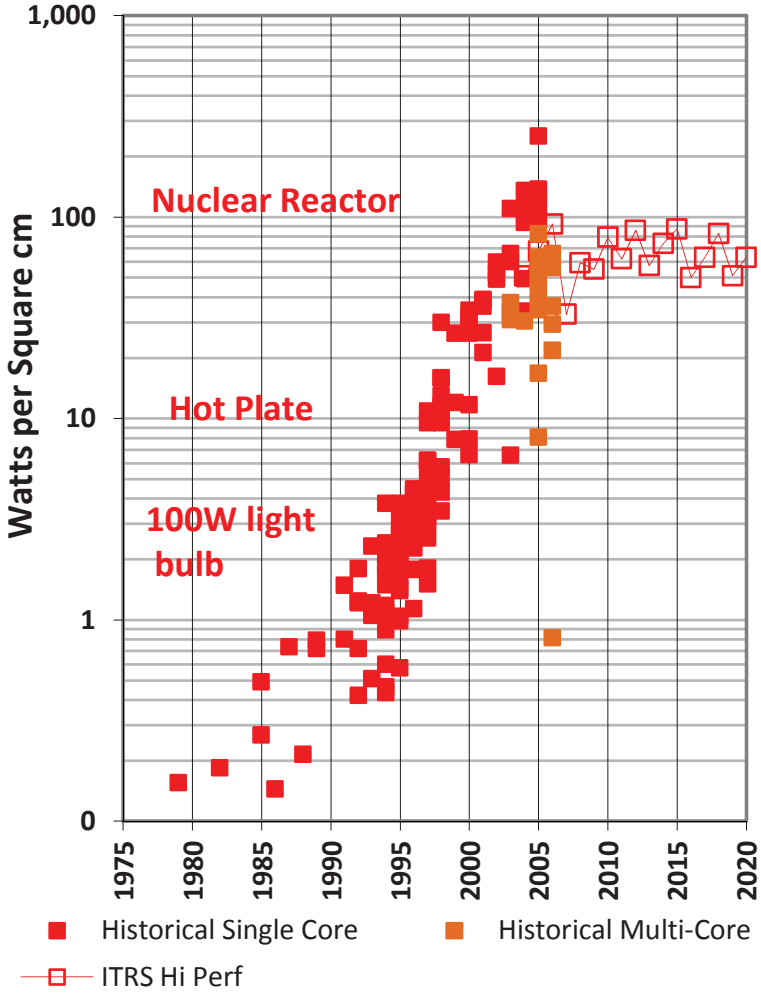
The Challenge of our Decade: *Performance growth in fixed power budget*

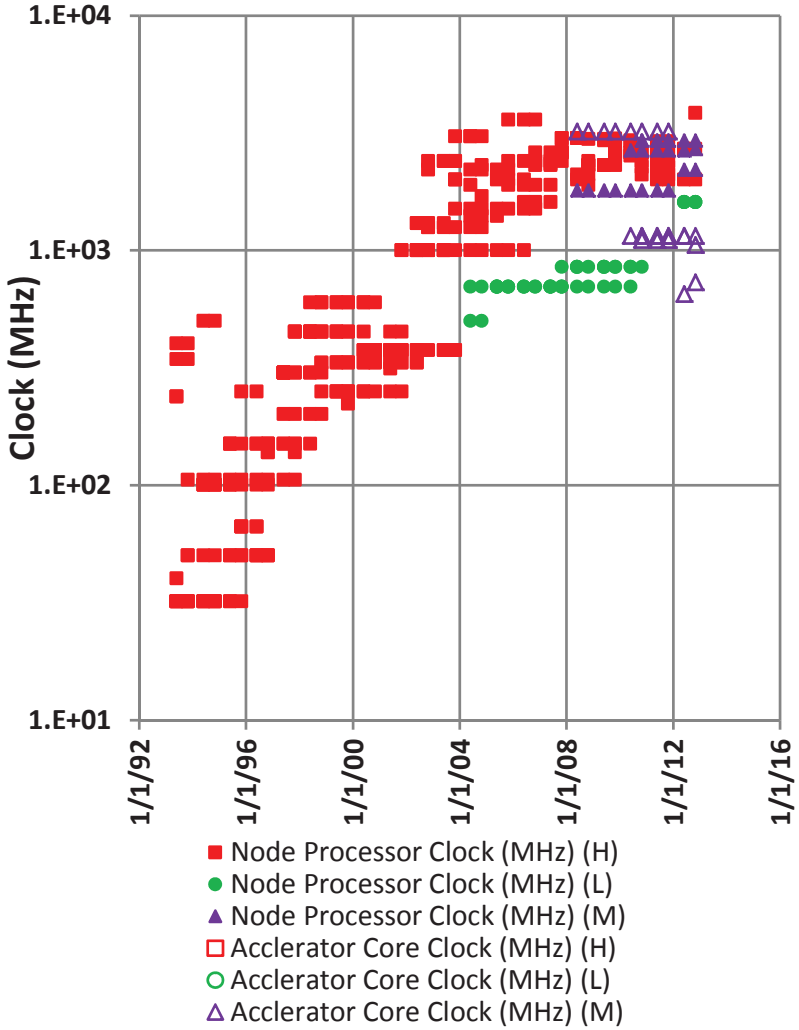
- The challenge is as dramatic as transition from vector to MPP
- This transition affects *all* computing for science from smallest to the largest scale
- Fundamentally breaks our software infrastructure (need to re-architect)

Approach: *Components of CoDesign Process*

- **XStack**: Translate emerging architectural trends into advanced software technology *(operating systems, communications libraries, programming systems)*
- **Fast Forward:** *$60M* public/private partnerships to accelerate development of computing technologies to deliver 100x more *usable* operations per watt in 10 yrs
- **CoDesign Centers**: Software Design Space Exploration, “proxy applications” and application prototyping to facilitate codesign
- **Hardware Design Space Exploration**: CAL hardware design space and “proxy hardware” using architectural simulation and modeling to facilitate codesign

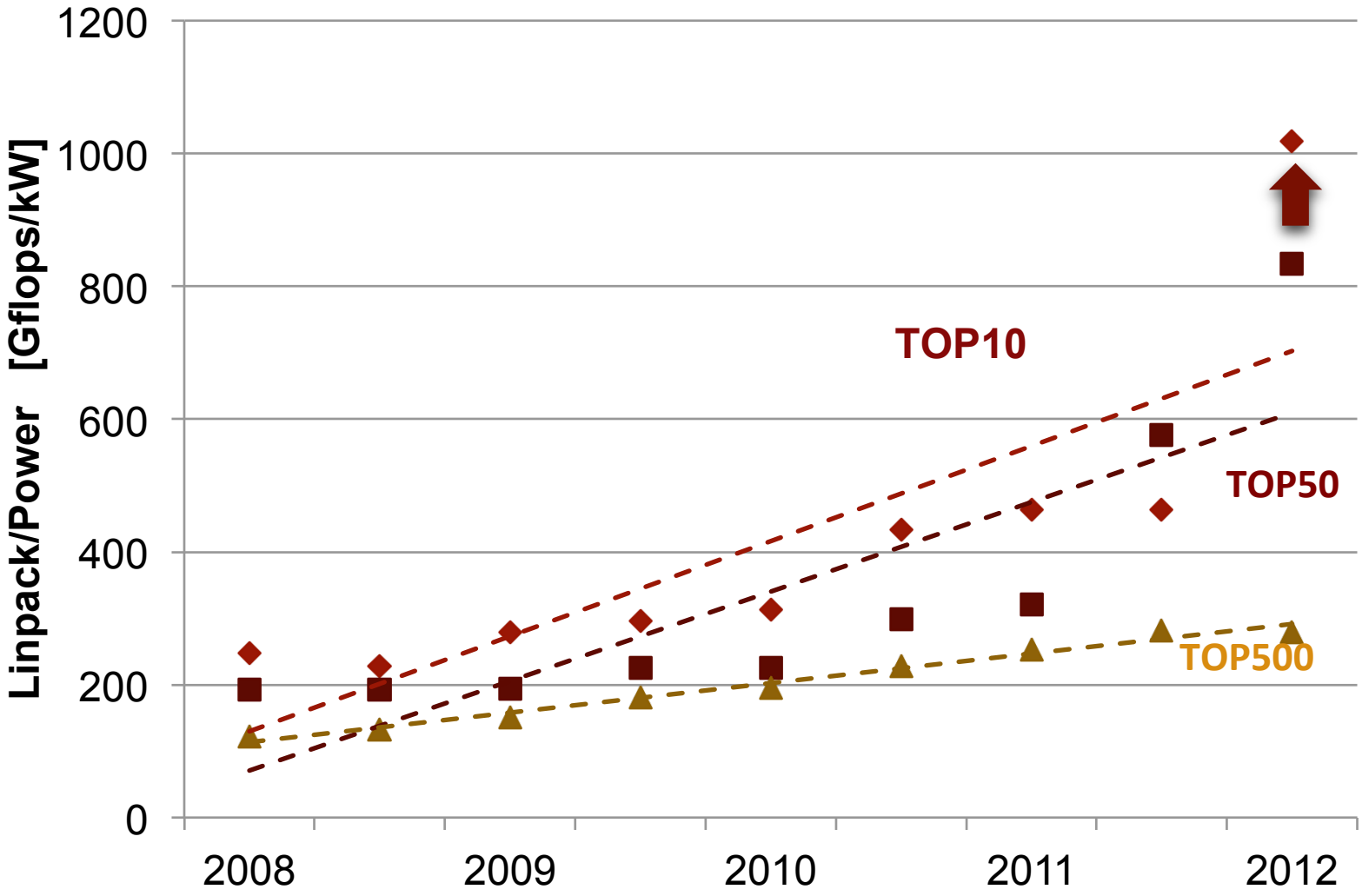
The Power and Clock Inflection Point in 2004





Source: Kogge and Shalf, IEEE CISE

Power Efficiency has gone up significantly in 2012



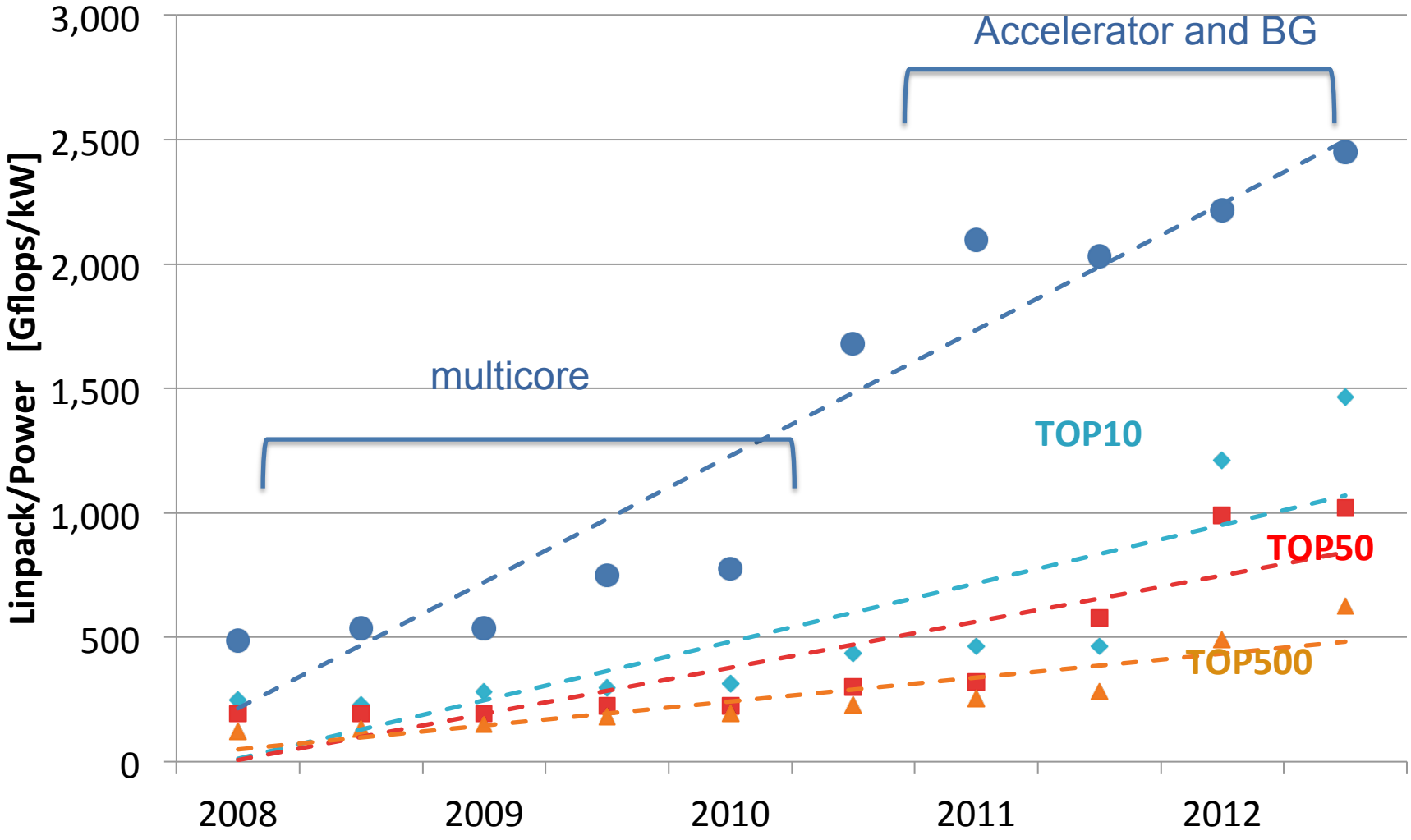
Data from: TOP500 November 2012

Most Power Efficient Architectures

Computer	Rmax/ Power
Appro GreenBlade, Xeon 8C 2.6GHz, Infiniband FDR, **Intel Xeon Phi**	**2,450**
Cray XK7, Opteron 16C 2.1GHz, Gemini, **NVIDIA Kepler**	**2,243**
BlueGene/Q, Power BQC 16C 1.60 GHz, Custom	**2,102**
iDataPlex DX360M4, Xeon 8C 2.6GHz, Infiniband QDR, **Intel Xeon Phi**	**1,935**
RSC Tornado, Xeon 8C 2.9GHz, Infiniband FDR, **Intel Xeon Phi**	**1,687**
SGI Rackable, Xeon 8C 2.6GHz, Infiniband FDR, **Intel Xeon Phi**	**1,613**
Chundoong Cluster, Xeon 8C 2GHz, Infiniband QDR, **AMD Radeon HD**	**1,467**
Bullx B505, Xeon 6C 2.53GHz, Infiniband QDR, **NVIDIA 2090**	**1,266**
Intel Cluster, Xeon 8C 2.6GHz, Infiniband FDR, **Intel Xeon Phi**	**1,265**
Xtreme-X , Xeon 8C 2.6GHz, Infiniband QDR, **NVIDIA 2090**	**1,050**

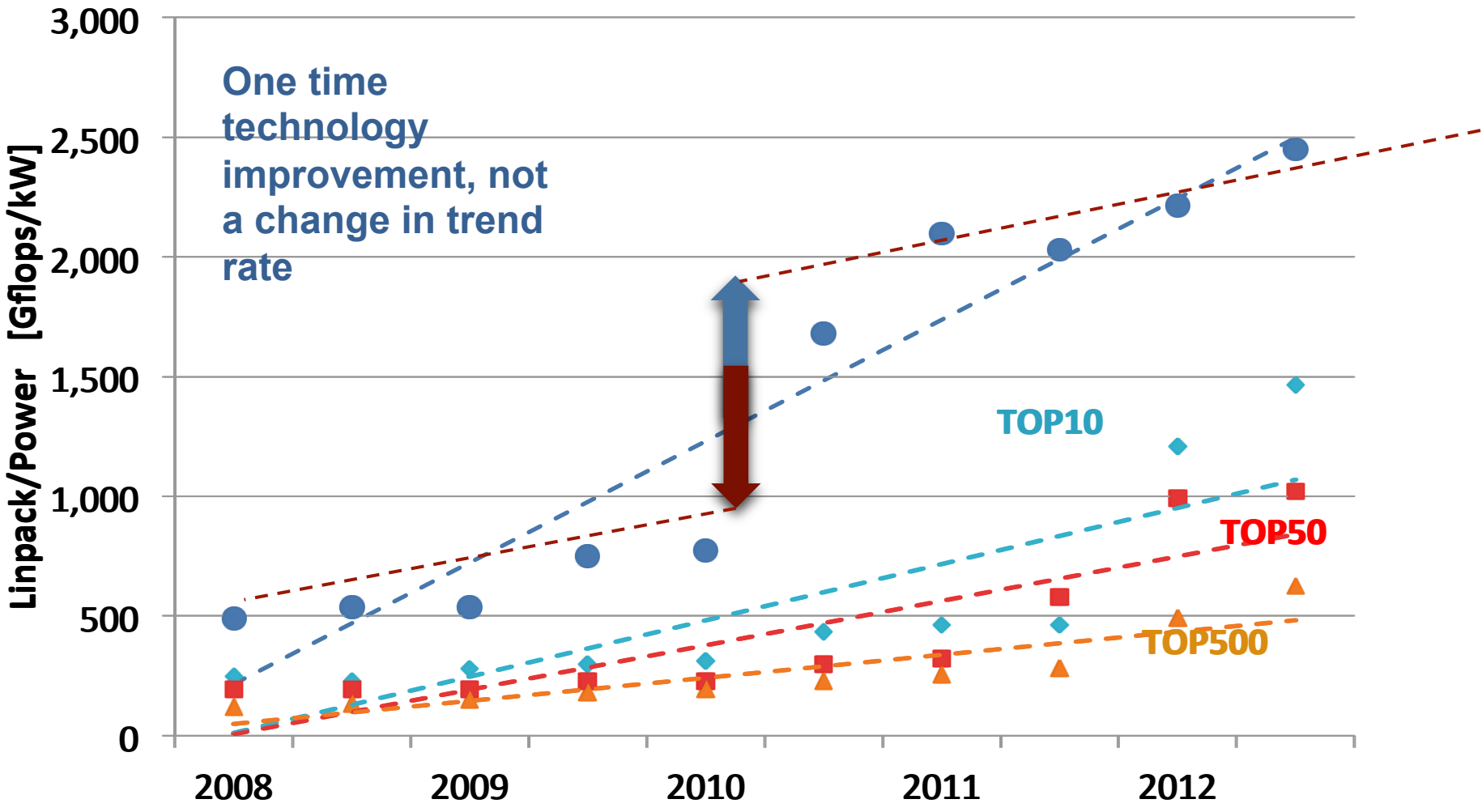
[Tflops/MW] = [Mflops/Watt]

Power Efficiency over Time



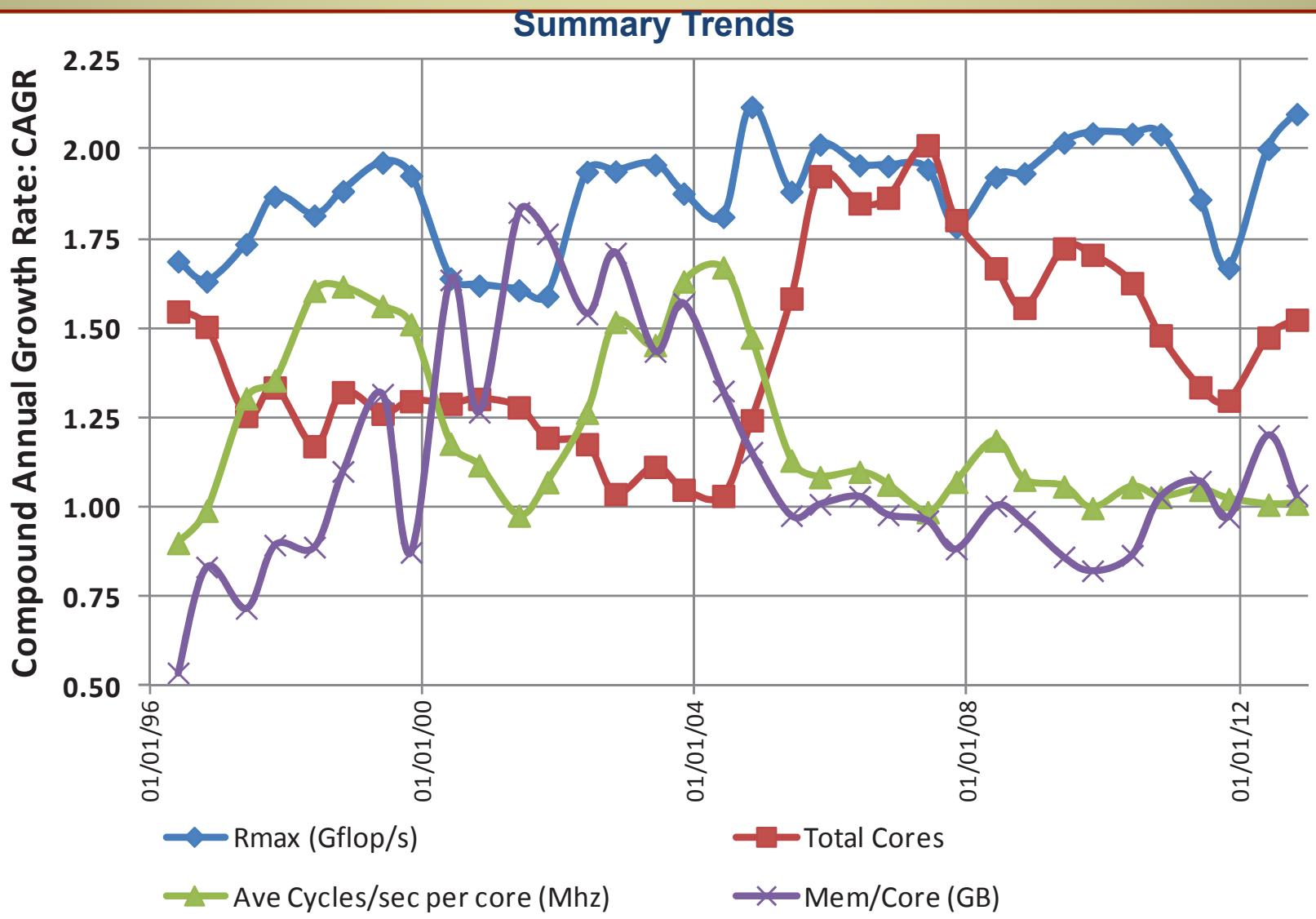
Data from: TOP500 November 2012

Power Efficiency over Time



Data from: TOP500 November 2012

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



Source: Kogge and Shalf, IEEE CISE 2013