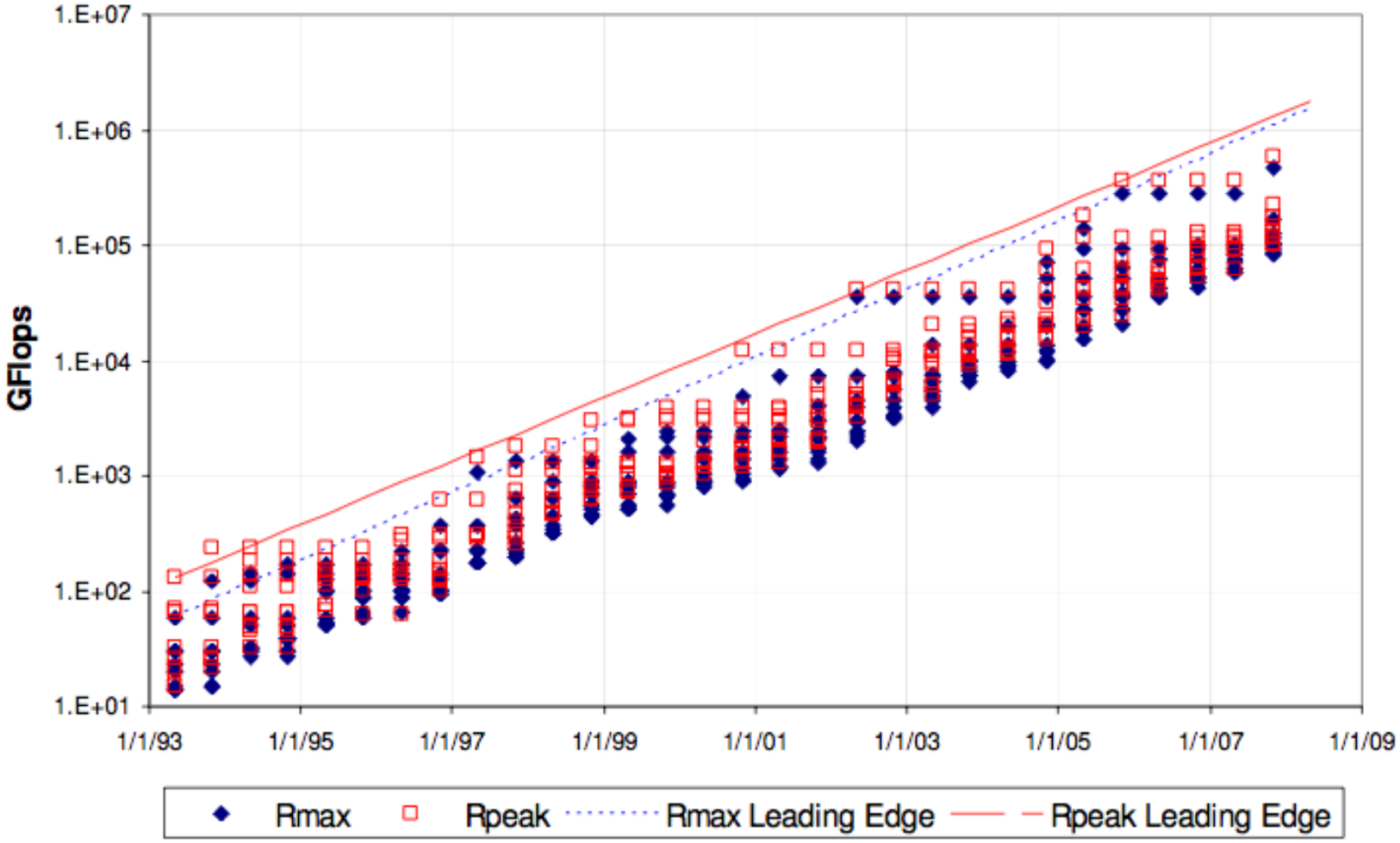
Software at Petascale and Exascale

Supercomputing – Next 10 Years

PARALLEL@ILLINOIS

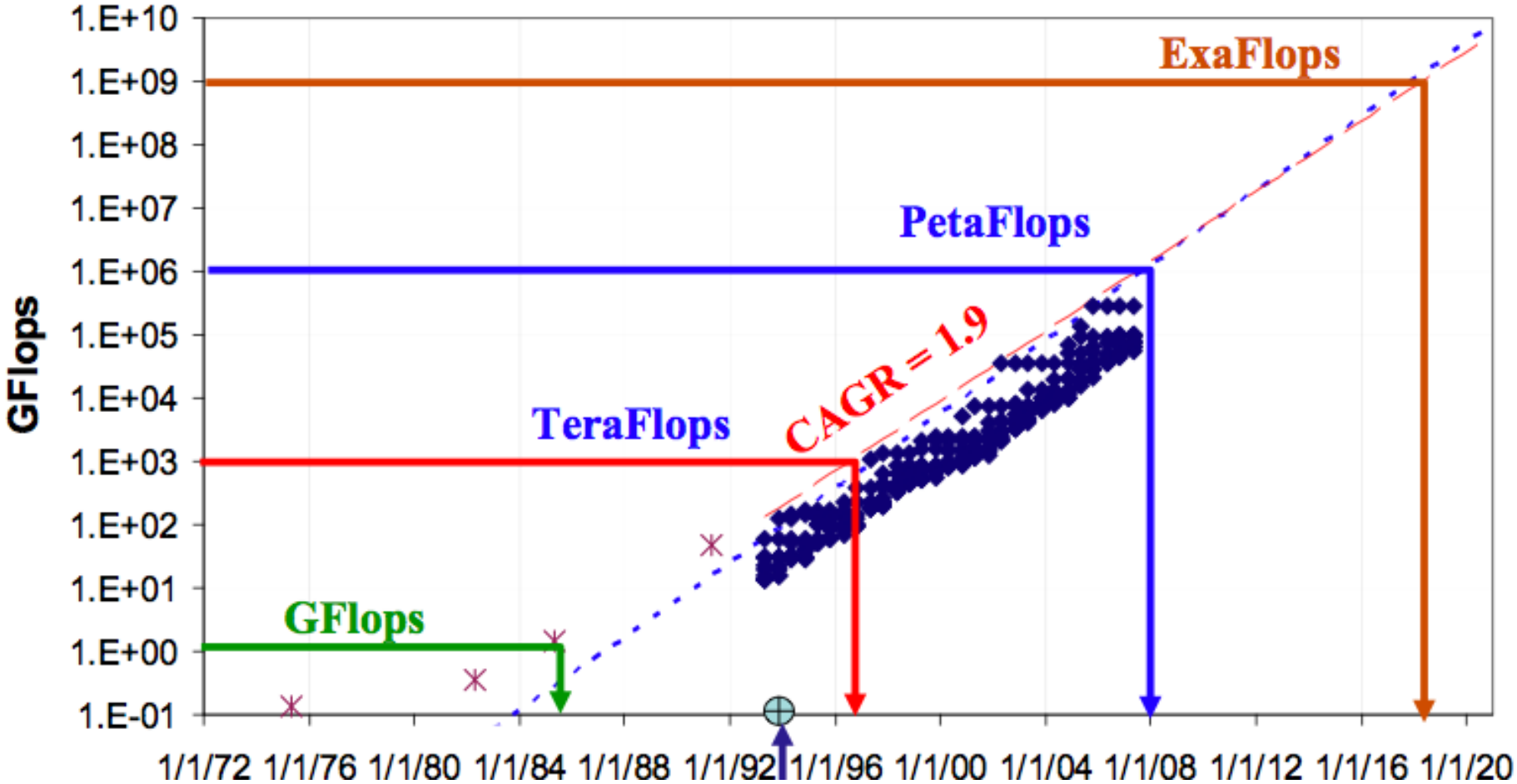
www.parallel.illinois.edu

Supercomputer Performance Evolution



(Kogge et al.)

Performance growths 1,000-fold every 11 years

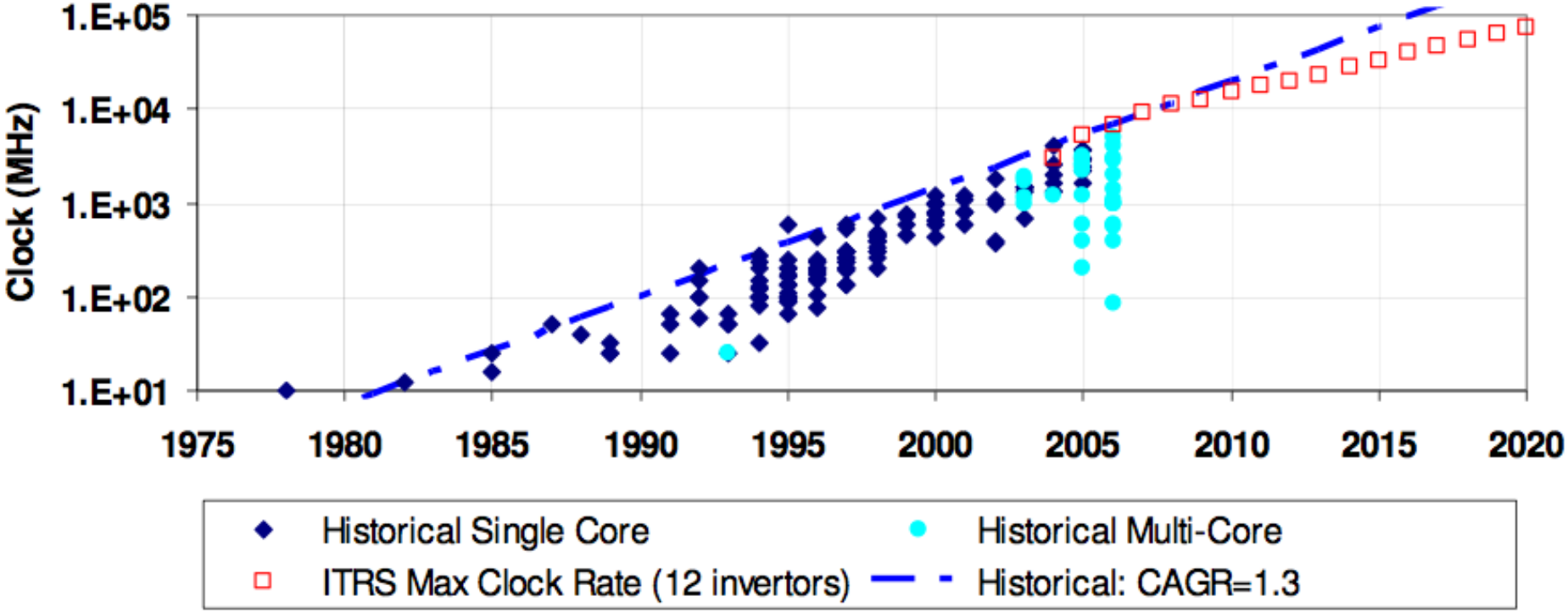


(Kogge et al.)

PARALLEL@ILLINOIS

Factors of Performance Growth

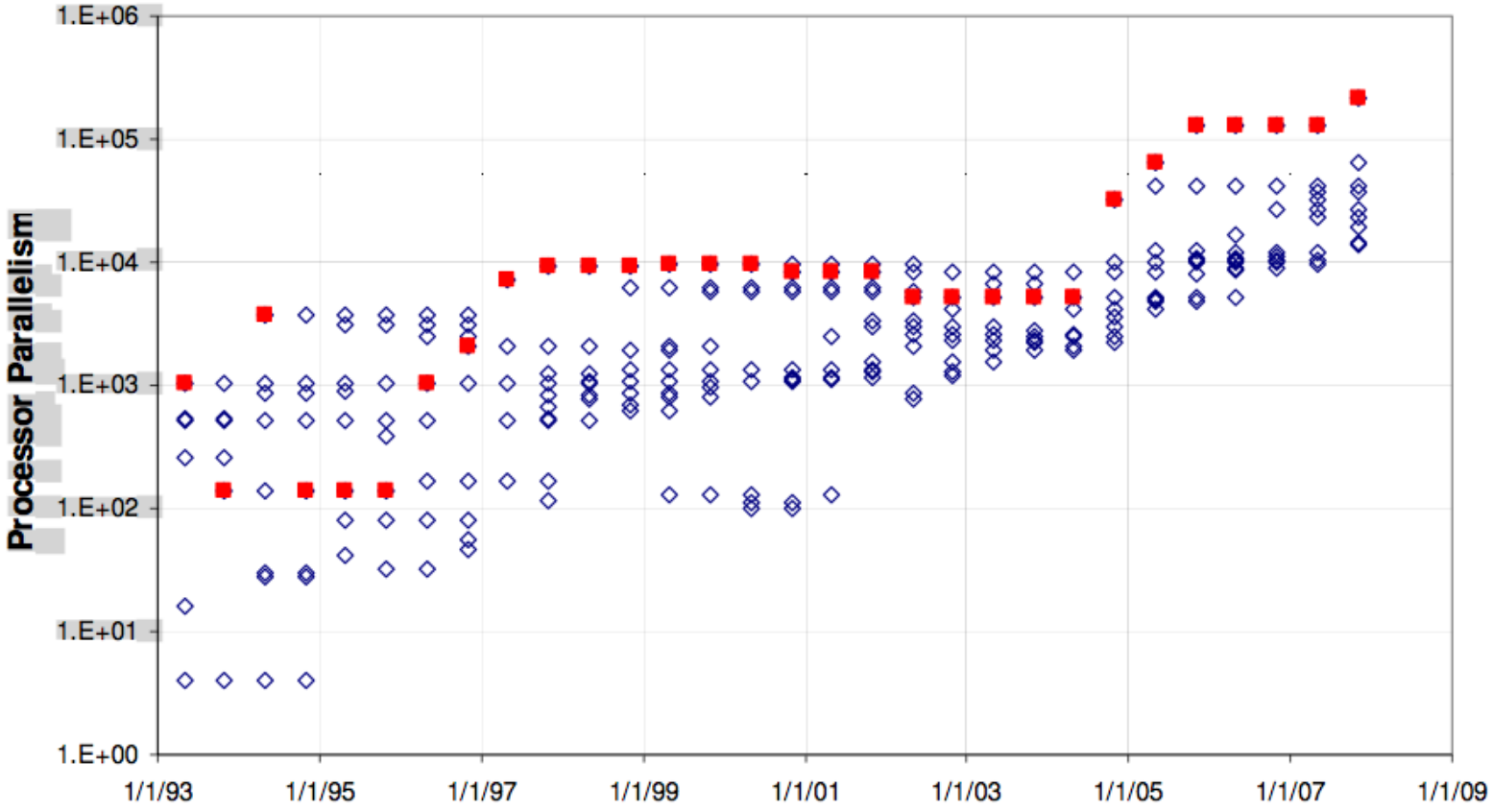
- Growth in clock rate



(Kogge et al.)

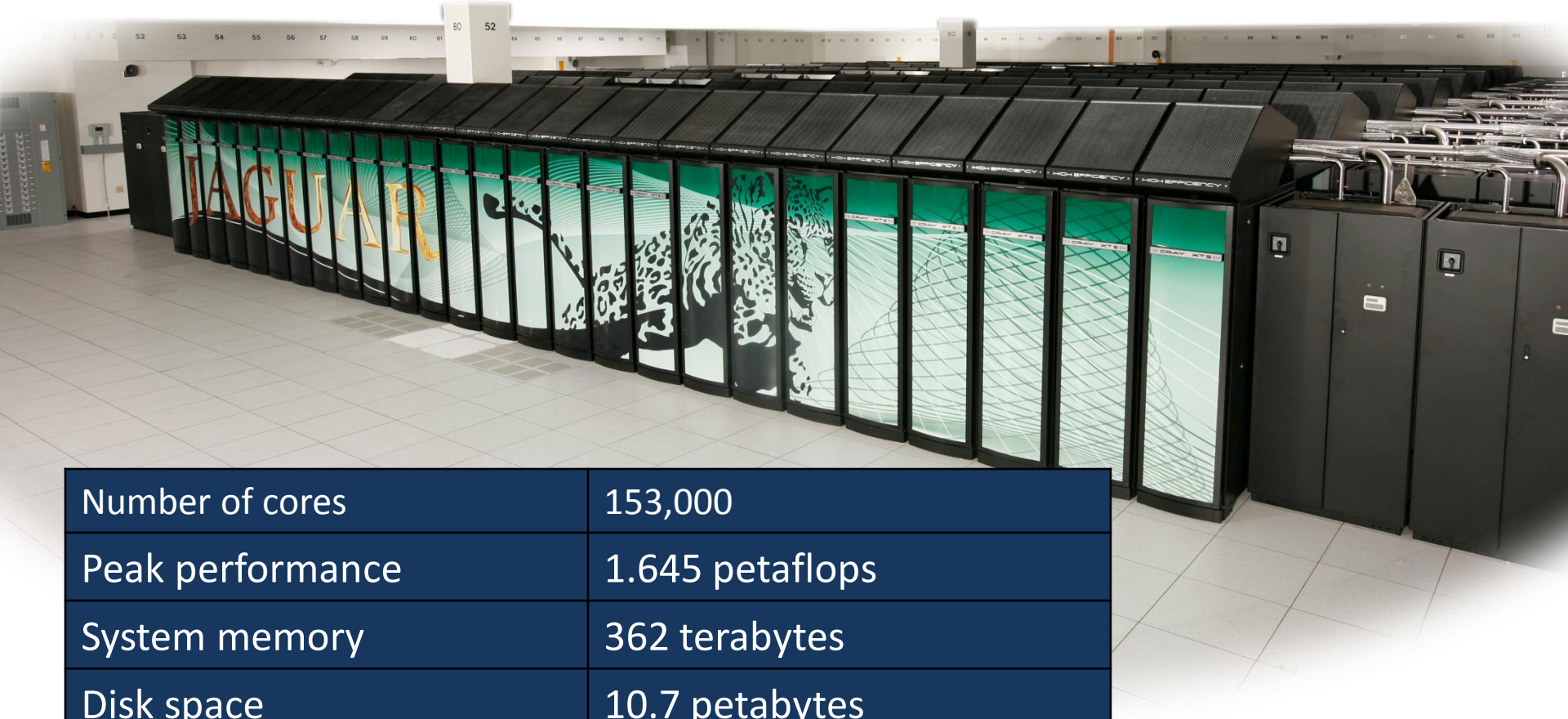
Factors of Performance Growth

- Growth in number of processors



(Kogge et al.)

Current Leader: Jaguar Cray XT5 at Oak Ridge



Number of cores	153,000
Peak performance	1.645 petaflops
System memory	362 terabytes
Disk space	10.7 petabytes
Disk bandwidth	200+ gigabytes/second
Power	Megawatts

(Band)

Science Applications on Jaguar

Science Area	Code	Cores	Total Performance
Materials	DCA++	150,144	1.3 PF*
Materials	LSMS	149,580	1.05 PF
Seismology	SPECFEM3D	149,784	165 TF
Weather	WRF	150,000	50 TF
Climate	POP	18,000	20 sim yrs/ day
Combustion	S3D	144,000	83 TF
Fusion	GTC	102,000	20 billion Particles / sec
Materials	LS3DF	147,456	442 TF
Chemistry	NWChem	96,000	480 TF
Chemistry	MADNESS	140,000	550+ TF

- Many of the applications can leverage exascale performance and more
- Use-driven science

(Band)

Toward Exascale

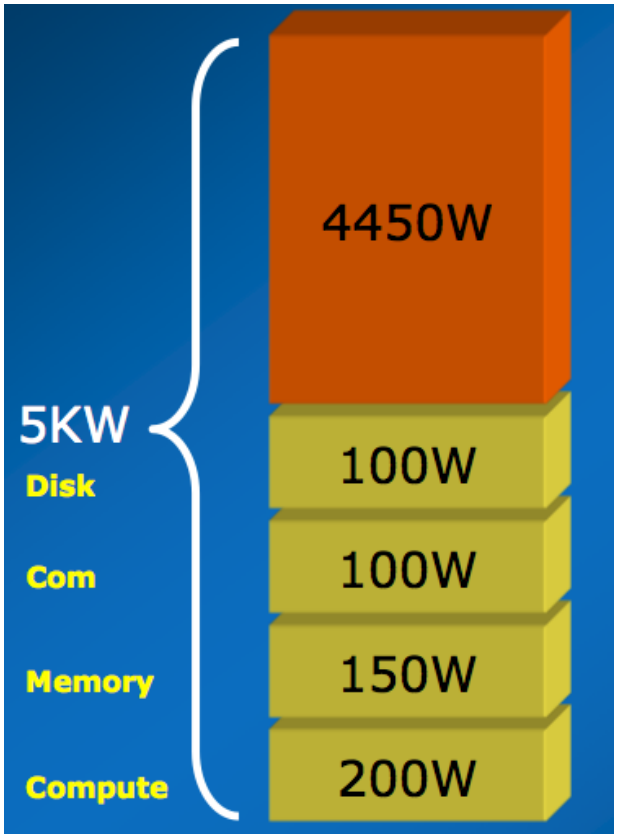
- Transistor density continues to increase (Moore's law)
  - up to 2020 – 8 nm; not clear what happens beyond
- Clock frequency does not increase
  - Power barrier
- Increased performance comes only from increased number of cores per chip, and increased number of chips

Exascale in 2020

- Extrapolation of current technology
  - 100M- 1B threads
  - 100-500 MWatts
- Energy consumption might be reduced one order of magnitude with aggressive technology and architecture change
  - Low power cores (more cores)
  - Aggressive voltage scaling (more errors)
  - Aggressive DRAM redesign (less bandwidth)

Challenges and Opportunities

Current power budget per Tflop



- Compute is 4% of power budget
- "Other" is 90% (decode, control … power supply, cooling)

- Opportunities for special-purpose architectures & simplified processor design

(Borkar)

Main Issues

- Increased parallelism
- Resiliency
- Variability
- Virtualization
- Hybrid HW

Blue Waters

- Petascale: sustained petaflop/s on complex applications
  - More than 200,000 cores
  - More than 1 petabyte of memory
  - More than 10 petabytes of user disk storage
  - Half an exabyte of archival storage
  - Up to 400 Gbps external connectivity
  - IBM PERCS technology

Managing with 1M-1B Threads

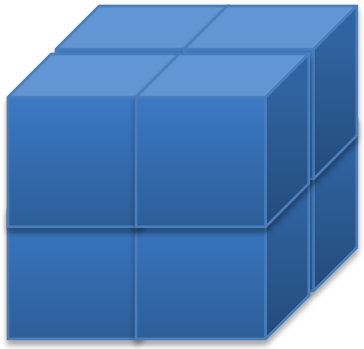
- Increased parallelism
- Resiliency
- Variability
- Virtualization
- Hybrid HW

Scaling Applications

- Weak scaling: use more powerful machine to solve larger problem
  - increase application size and keep running time constant; e.g., refine grid
  - Larger problem may not be of interest
  - May want to scale time, not space (molecular dynamics)
  - Cannot scale space without scaling time (iterative methods): granularity decreases and communication increases

Scaling Iterative Methods

- Assume that number of cores (and compute power) are increased by factor of k^4
- Space and time scales are refined by factor of k
- Mesh size increased by factor of $k \times k \times k$
- Local cell dimension decreases by factor of $k^{1/4}$
- Cell volume decreases by factor of $k^{3/4}$ while area decreases by factor of $k^{2/4}$; area to volume ratio (communication to computation ratio) increases by factor of $k^{3/2}$.



Debugging and Tuning: Observing 1B Threads

- Scalable infrastructure to control and instrument 1B threads
- Parallel information compression to identify “anomalies”

- Need to ability to express “normality” (global correctness and performance assertions)

Main Issues

- Increased parallelism
- Resiliency
- Variability
- Virtualization
- Hybrid HW

Decreasing Mean Time to Failure

- Problem:
  - More transistors
  - Smaller transistors
  - Lower voltage
  - More manufacturing variance
- Current technology: global, synchronized checkpoint
- Future technology:
  - Increased HW redundancy & error checking
    - OK if number of components stays constant
  - Integrated HW/SW/application approach for fault isolation and local recovery
    - Later probably needed if number of components per system increases

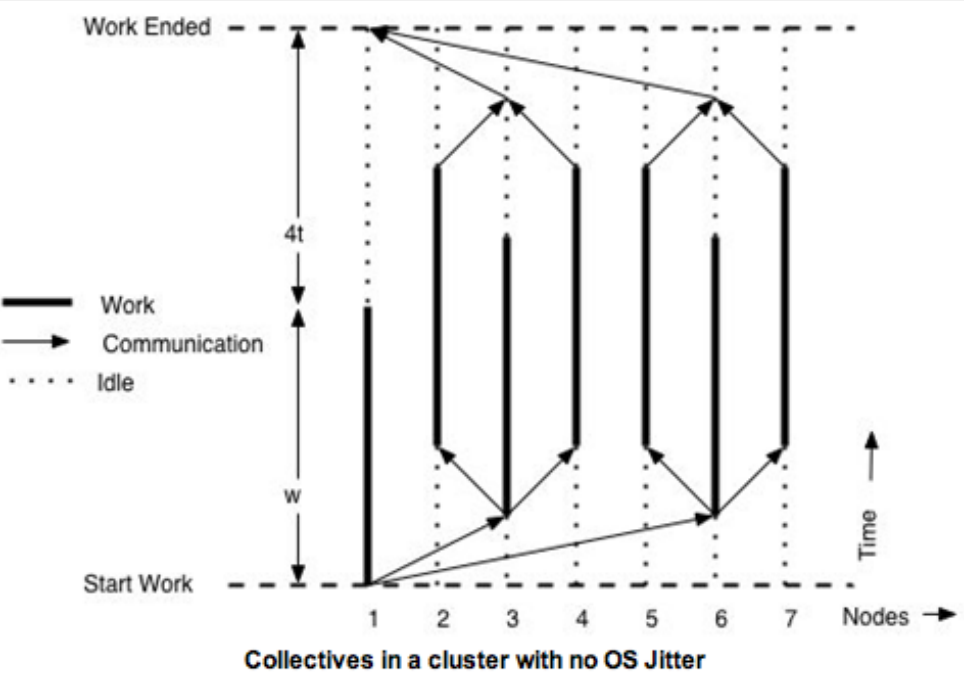
Main Issues

- Increased parallelism
- Resiliency
- Variability
- Virtualization
- Hybrid HW

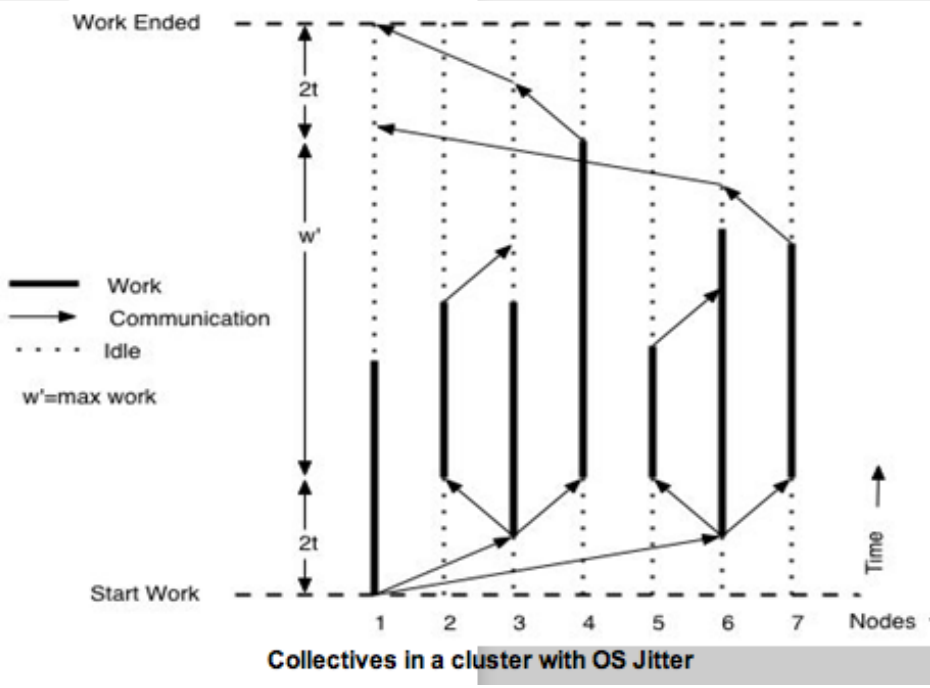
Bulk Synchronous

- Many parallel applications are written in a “bulk-synchronous style”: alternating stages of local computation and global communication
- Models implicitly assumes that all processes advance at the same compute speed
- Assumptions breaks down for an increasingly large number of reasons
  - Black Swans
  - OS jitter
  - Application jitter
  - HW jitter

Jitter Illustrated



Collectives in a cluster with no OS Jitter



Collectives in a cluster with OS Jitter

OS jitter has been empirically measured to slow down computations by a factor of 2 or more

(IBM)

Jitter Causes

- Black Swans
  - If each thread is unavailable (busy) for 1 msec once a month, than most collective communications involving 1B threads take > 1 msec (the black swan effect)
- OS jitter
  - Background OS activities (daemons, heartbeats…)
- HW jitter
  - Background error recovery activities (e.g., memory scrubbing & error handling); power management; management of manufacturing variability; degraded operation modes
- Application jitter
  - Input-dependent variability in computation intensity
- **Need to move away from bulk model**

Possible Approaches

- User helped source code optimization
  - Replace blocking communication (including collective communication) with non blocking communication
  - Refactoring tools help user make changes correctly

MPI_Barrier

MPI_Barrier_start

….

MPI_Barrier_end

  - Code between start—end should not conflict with code at other processes not separated by full barrier

Possible Approaches (2)

- Compiler optimizations (no change in source code)
  - execute “sends” as early as possible; execute “receives” as late as possible
  - tradeoff with communication aggregation
- Run-time optimization: virtualization

Main Issues

- Increased parallelism
- Resiliency
- Variability
- Virtualization
- Hybrid HW

Task Virtualization

- Multiple logical tasks are scheduled on each physical core; tasks are scheduled nonpreemptively; task migration is supported
  - Hides variance and communication latency
  - Helps with scalability (decouples # tasks from # cores)
  - Helps with resiliency
  - *Needed for modularity* (multiphysics/multiscale codes – handling parallel coupling of modules)
  - *Improves performance* (better locality)
  - *Scales* (Charm++/AMPI)
  - Can be implemented below MPI or PGAS languages

Task Virtualization Styles

- Varying, user controlled number of tasks (AMPI)
  - Locality achieved by load balancer
- Recursive (hierarchical) range splitter (TBB)
  - Method to split (recursively) problem in two sub-problems
  - Method to combine two sub-solutions
  - Method to decide when sub-problem is small enough tob e solved sequentially
  - Method to solve sub-problem sequentially
  - Locality is achieved implicitly

Main Issues

- Increased parallelism
- Resiliency
- Variability
- Virtualization
- Hybrid HW

Hybrid Communication

- Multiple levels of caches and of cache sharing
- Different communication models intra and inter node
  - Coherent shared memory inside chip (node)
  - rDMA (put/get/update) across nodes
- Hybrid features change every HW generation
- Need to be able to easily adjust number of cores & replace inter-node communication with intra-node communication
- Easy to “downgrade” (use shared memory for message passing); hard to “upgrade”; hence tend to use lowest commonality (message passing)
- No good interoperability between shared memory (e.g., OpenMP) and message passing (MPI)

Possible Directions

- Express cache oblivious algorithms using recursive range splitting
  - May provide 3 methods:
    - Distributed memory splitting/merging
    - Shared memory splitting/merging
    - Sequential
- (Low hanging fruit) Enable shared memory communication across MPI tasks

Hybrid Computation

- Vector instructions
- Different core types
- Accelerators

- Can significantly reduce energy per flop
- Require (now) different source code
- Easy to compile CUDA to multicore; hard to compile OpenMP to GPU

Possible Approaches

- Use library auto-tuning
- Reduce semantic gap at architecture level; use (static or dynamic) compilation

Societal Changes in Supercomputing

- How would a supercomputer that costs >> $1B be built and managed? An international consortium?
- How should access to such an expensive apparatus be controlled? How does one ensure efficient use?
- Should we have teams with more specialized members (expert debugger; expert tuner; expert modeler…)? Should we build tools for experts or for the informed dilettante?
- What changes when “waiting” is not, anymore, a strategy for getting more performance?