Presentation to the:
Virtual School for Computational Science and Engineering

*Scaling to Petascale: Concepts & Beyond*

# Scaling to the Present and Future


Thomas Sterling

Arnaud and Edwards Professor of Computer Science

*Louisiana State University*

Visiting Associate, *California Institute of Technology*

Distinguished Visiting Scientist, *Oak Ridge National Laboratory*

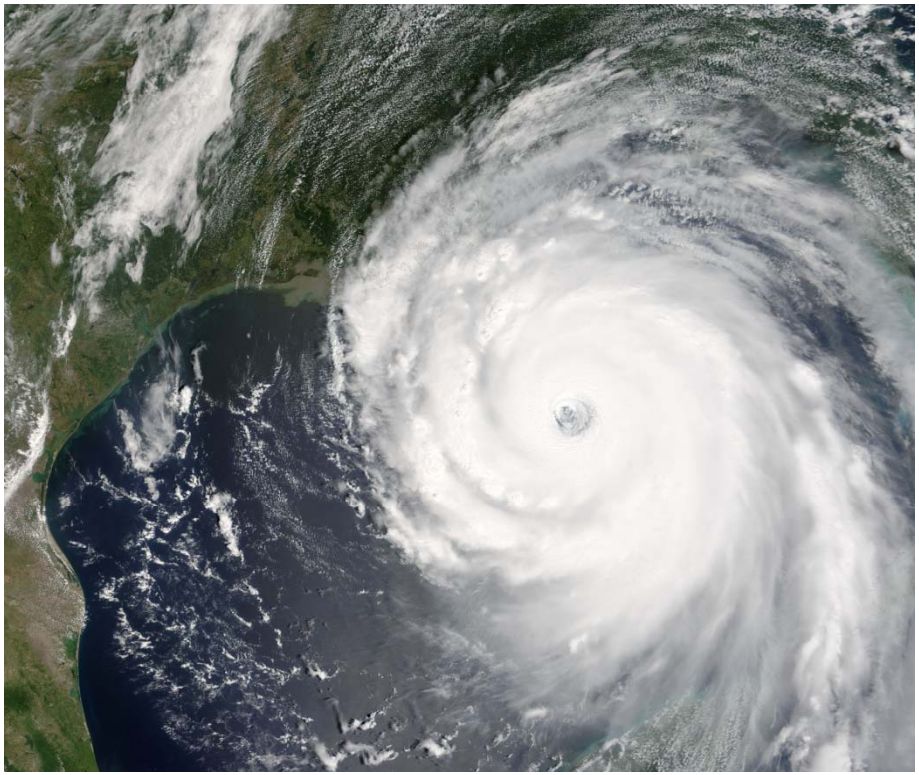
CSRI Fellow, *Sandia National Laboratory*


August 3, 2009

# Topics

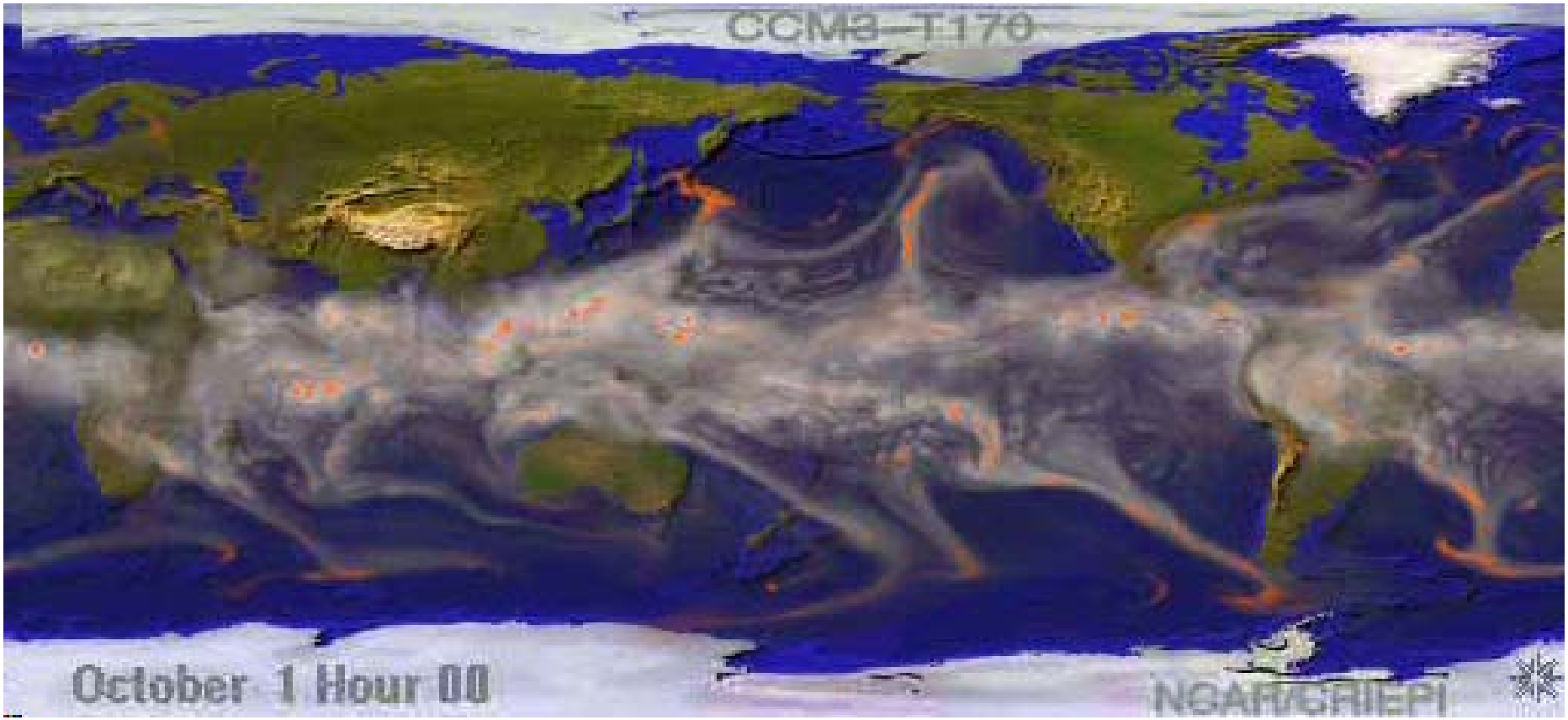
- HPC Applications
- Petaflops Systems
- Enabling Technologies and Trends
- HPC Architectures
- Scaling Factors
- Sources of Performance Degradation
- Parallel Programming Models – Condor
- Parallel Programming Models - MPI
- Parallel Programming Models – OpenMP
- Scaling to a Thousand Petaflops

# Aerial & Satellite  of Hurricane Katrina



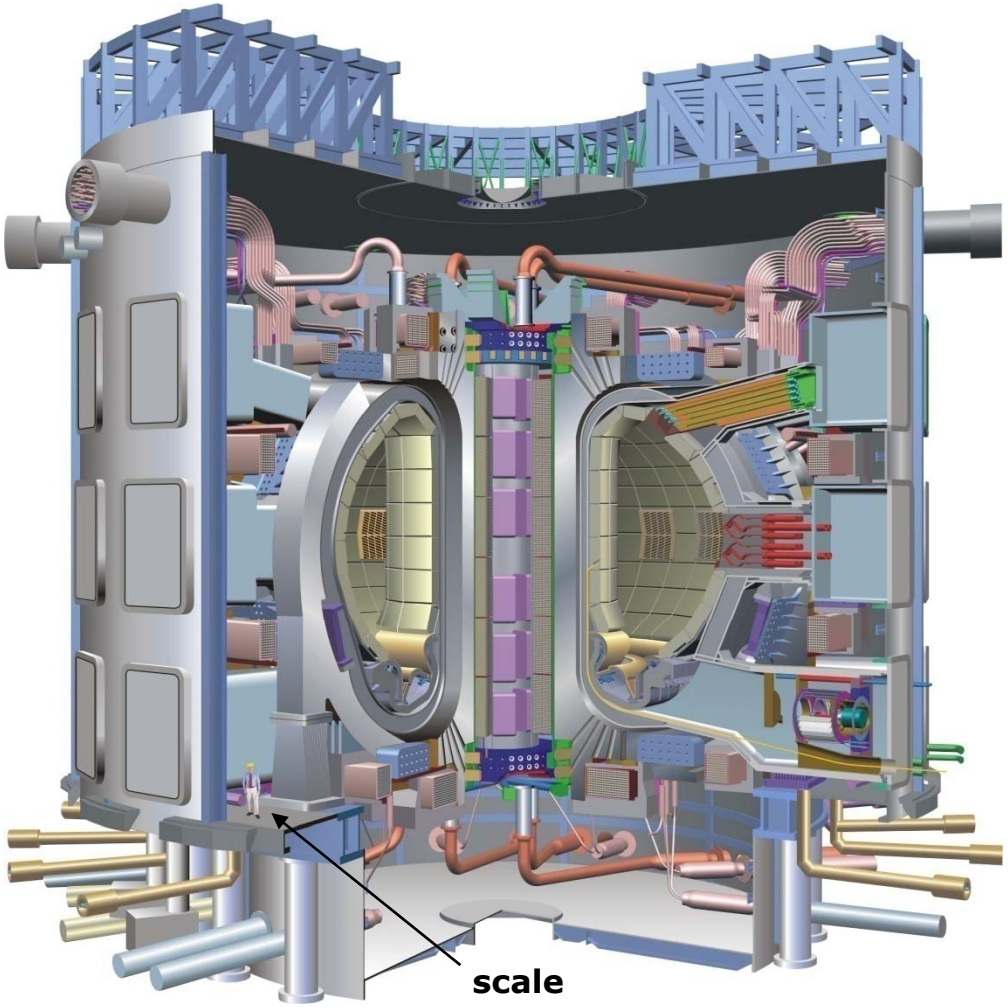
# *Example of Global Climate Model Simulation*

Precipitable Water (gray scale) and Precipitation Rate (orange)




*Animation courtesy of NCAR SCD Visualization and Enabling Technologies Section*

# The U.S. is an official partner in ITER

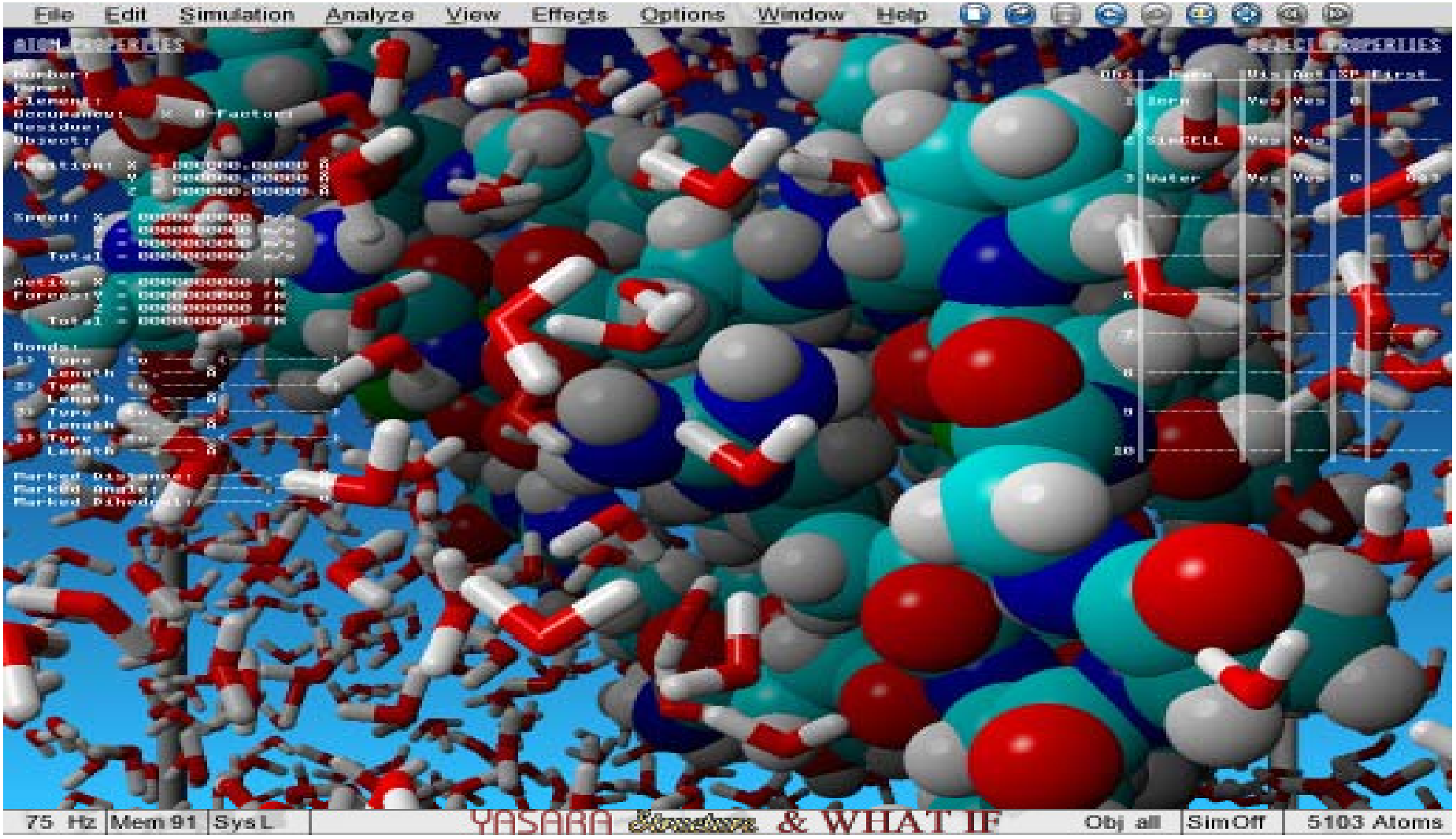



**I**nternational
**T**hermonuclear
**E**xperimental
**R**eactor

- European Union
- Japan
- United States
- Russia
- Korea
- China

- 500 MW fusion output
- Cost: $5-10 B
- To begin operation in 2015

# Molecular Dynamics

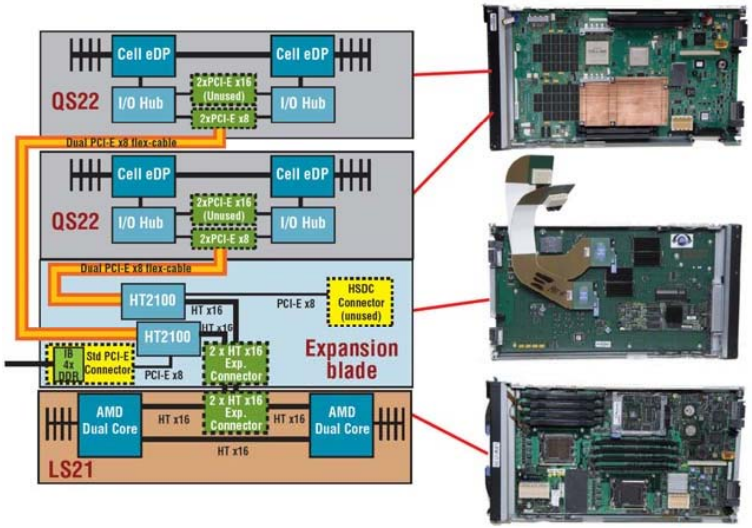


# Topics

- HPC Applications
- Petaflops Systems
- Enabling Technologies and Trends
- HPC Architectures
- Scaling Factors
- Sources of Performance Degradation
- Parallel Programming Models – Condor
- Parallel Programming Models - MPI
- Parallel Programming Models – OpenMP
- Scaling to a Thousand Petaflops

# Road Runner

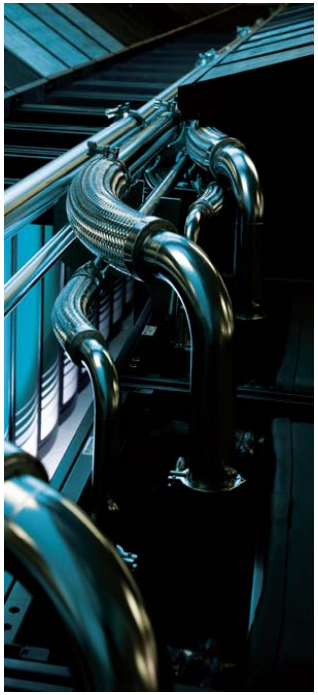
- First supercomputer to reach sustained 1 PFLOPS performance and current #1 on TOP500
- First hybrid supercomputer (Cell+Opteron)
- System information:
  - 296 racks on 5,200 sq.ft. footprint
  - 18 *connected units* with 180 nodes each
  - 1.46 PFLOPS peak, 1.1 PFLOPS $R_{max}$
  - 6,480 AMD Opteron processors
  - 12,960 IBM PowerXCell 8i processors
  - 101 TB memory
  - 216 System x3755 I/O nodes
  - 26 288-port ISR2012 Infiniband 4x DDR switches
  - 2.5 MW power consumption
- Tri-blade compute node:
  - Two QS22 blades (Cell) and one LS21 blade (Opteron)
  - Cell blades host four 3.2 GHz processors with aggregate peak of 435.2 DP GFLOPS
  - Opteron blade contains 2 dual-core 1.8 GHz Opterons 2210 delivering peak of 14.4 GFLOPS
  - Every Opteron core and Cell processor uses 4 GB memory (32 GB per node)



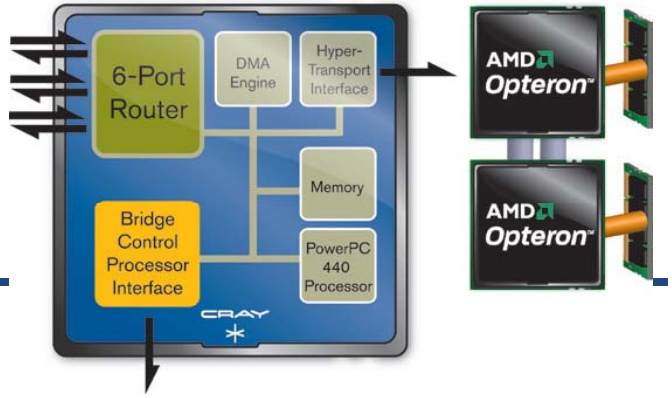

CENTER FOR COMPUTATION & TECHNOLOGY

Los Alamos NATIONAL LABORATORY EST. 1943

# ORNL Jaguar (Cray XT 5)

- Second sustained PFLOPS machine (#2 on TOP500)
- System overview
  - Cray XT4 and XT5 nodes over shared network (SION)
  - 284 cabinets in 5,800 sq.ft. floorspace
  - 1.38 PFLOPS peak, 1.06 PFLOPS $R_{max}$ (XT5 only)
  - 26,604 compute nodes with over 181,000 cores
  - 362 TB memory
  - Cray Seastar2+ interconnect configured as 3-D torus
  - 374 TB/s interconnect bandwidth (XT5)
  - 10 PB of RAID6 storage with 240+44 GB/s bandwidth respectively from 214 XT5 and 116 XT4 I/O nodes
  - Power: 6.95 MW, liquid cooling (XT5)
- XT5 node specifications
  - Two quad-core 2356 Barcelona Opterons at 2.3 GHz
  - Peak performance of 73.6 GFLOPS
  - 16GB ECC DDR2-800 SDRAM
  - 6 port Seastar2+ ASIC, 9.6 GB/s per port



# FZJ JUGENE (IBM BG/P)

- Most powerful BG/P system (#3 in TOP500)
- System information:
  - 72 racks with 130 $m^2$ footprint
  - 73,728 compute nodes with 294,912 processors
  - 1.0 PFLOPS peak, 825.5 TFLOPS $R_{max}$
  - 144 TB memory
  - Multiple interconnects: 3-D torus, scalable collective network and fast barrier network
  - 600 I/O nodes
  - 1 PB of storage at 16 GB/s
  - Power: 2.5 MW (35 kW per rack)
- Node specification:
  - 4 PowerPC 450 cores at 850 MHz
  - Dual DP FPUs per core
  - 2 GB memory
  - 32-bit mode operation


JÜLICH FORSCHUNGSZENTRUM

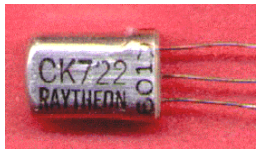
LSU CENTER FOR COMPUTATION & TECHNOLOGY

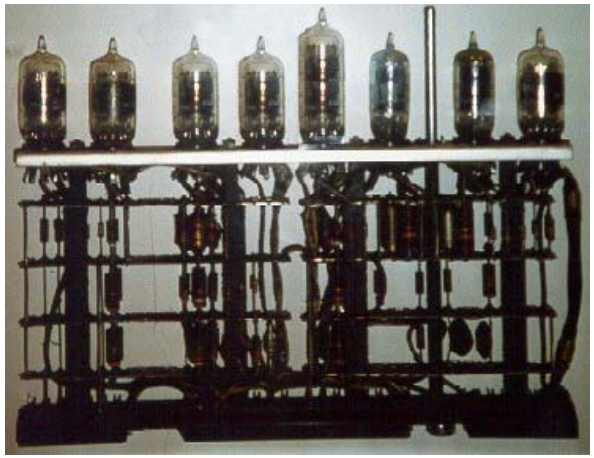
# Topics

- HPC Applications
- Petaflops Systems
- Enabling Technologies and Trends
- HPC Architectures
- Scaling Factors
- Sources of Performance Degradation
- Parallel Programming Models – Condor
- Parallel Programming Models - MPI
- Parallel Programming Models – OpenMP
- Scaling to a Thousand Petaflops

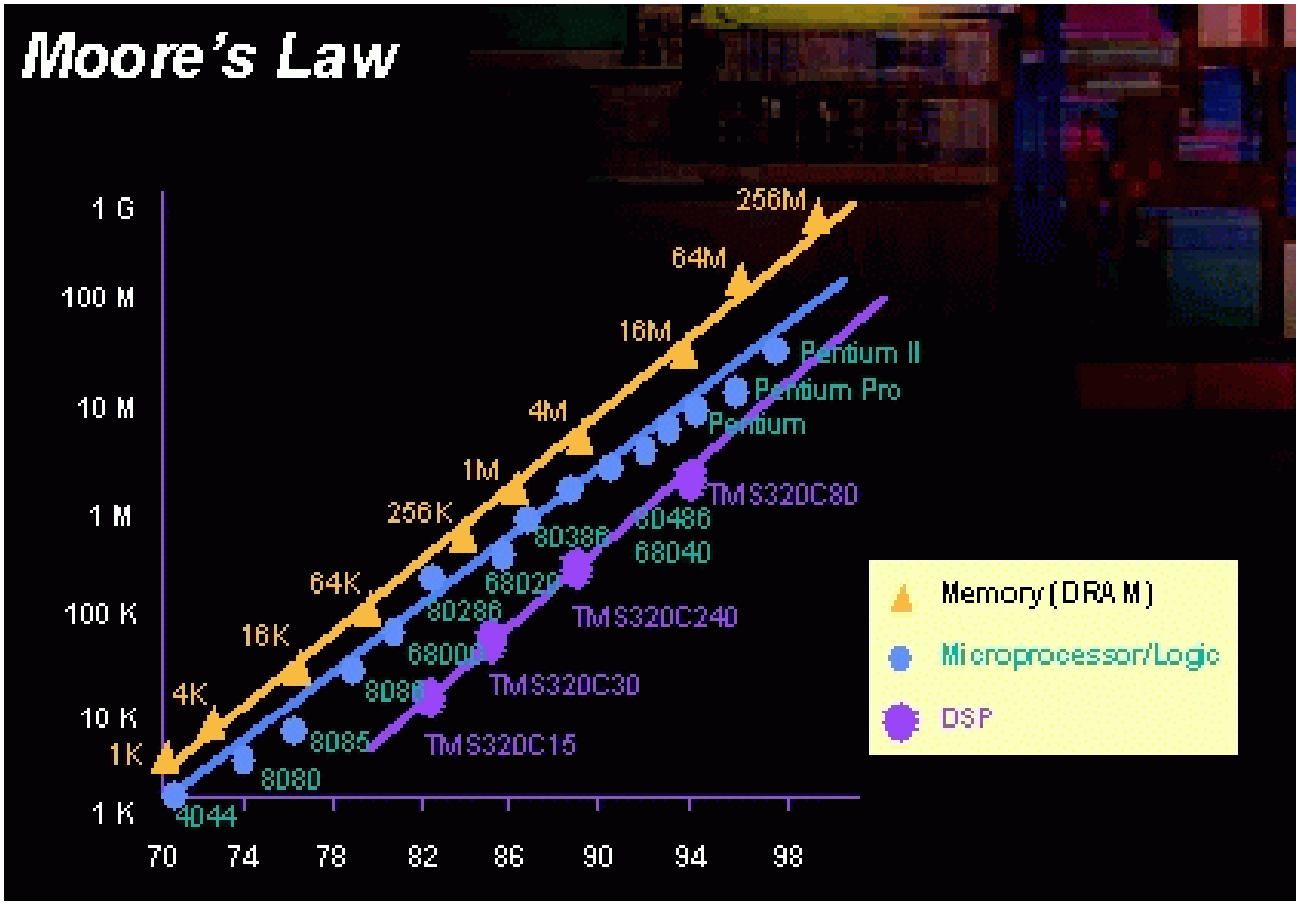
# Where Does Performance Come From?

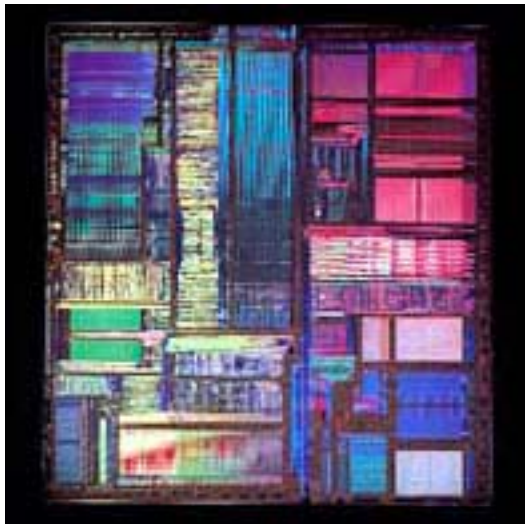
- Device Technology
  - Logic switching speed and device density
  - Memory capacity and access time
  - Communications bandwidth and latency
- Computer Architecture
  - Instruction issue rate
    - Execution pipelining
    - Reservation stations
    - Branch prediction
    - Cache management
  - Parallelism
    - Operations per cycle per processor
      - Instruction level parallelism (ILP)
      - Vector processing
    - Processors per node
    - Number of nodes in a system

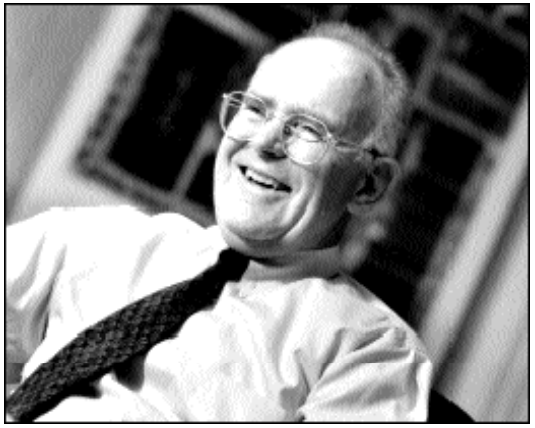


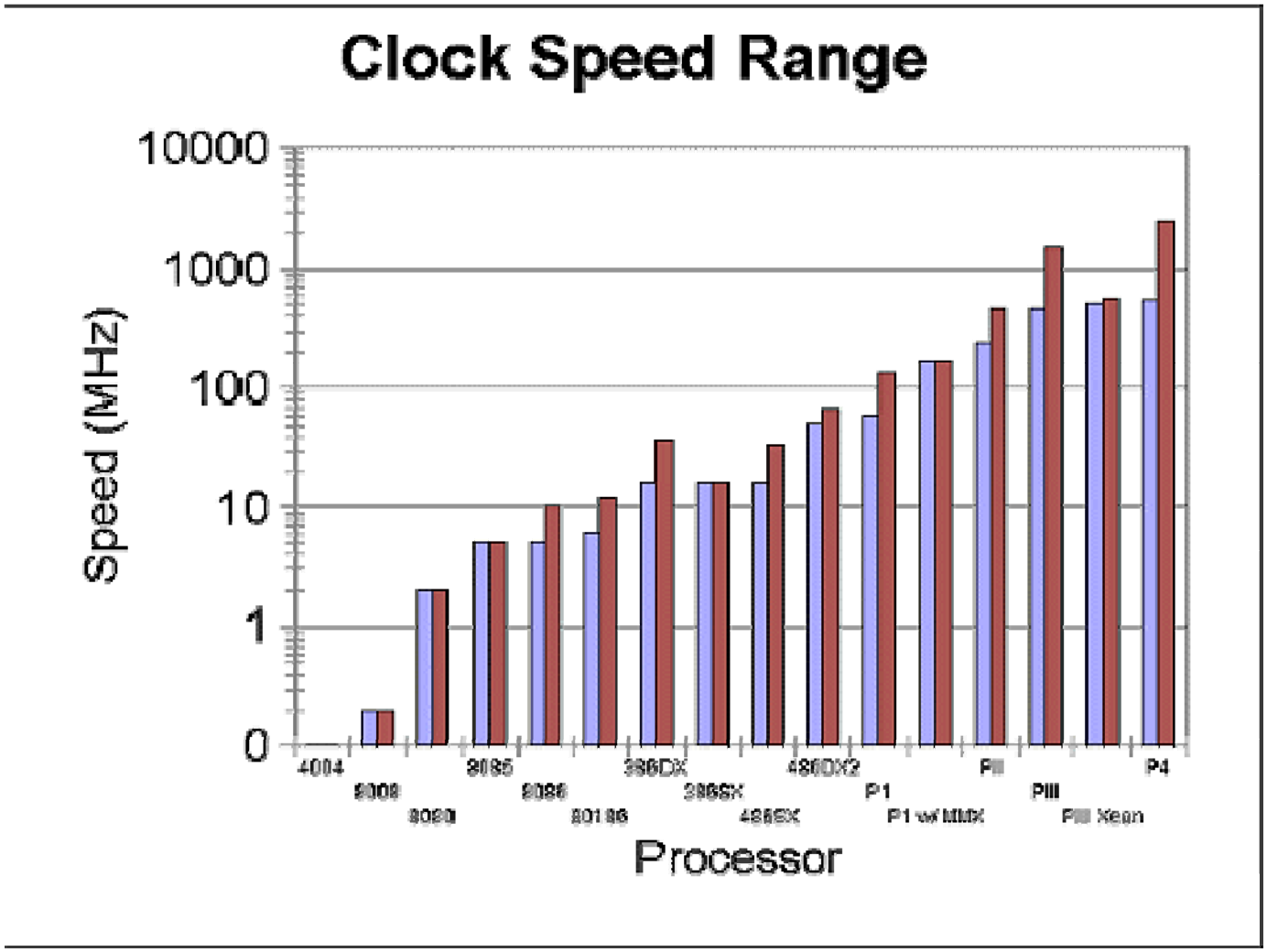
# Moore's Law


Moore's Law
1 G
100 M
10 M
1 M
100 K
10 K
1 K
70
74
78
82
86
90
94
98
256M
64M
16M
4M
1M
256K
64K
16K
4K
1K
Pentium II
Pentium Pro
Pentium
TMS320C80
80486
80386
68040
68020
80286
TMS320C240
68000
TMS320C30
8086
8085
TMS320C15
8080
4044
Memory (DRAM)
Microprocessor/Logic
DSP

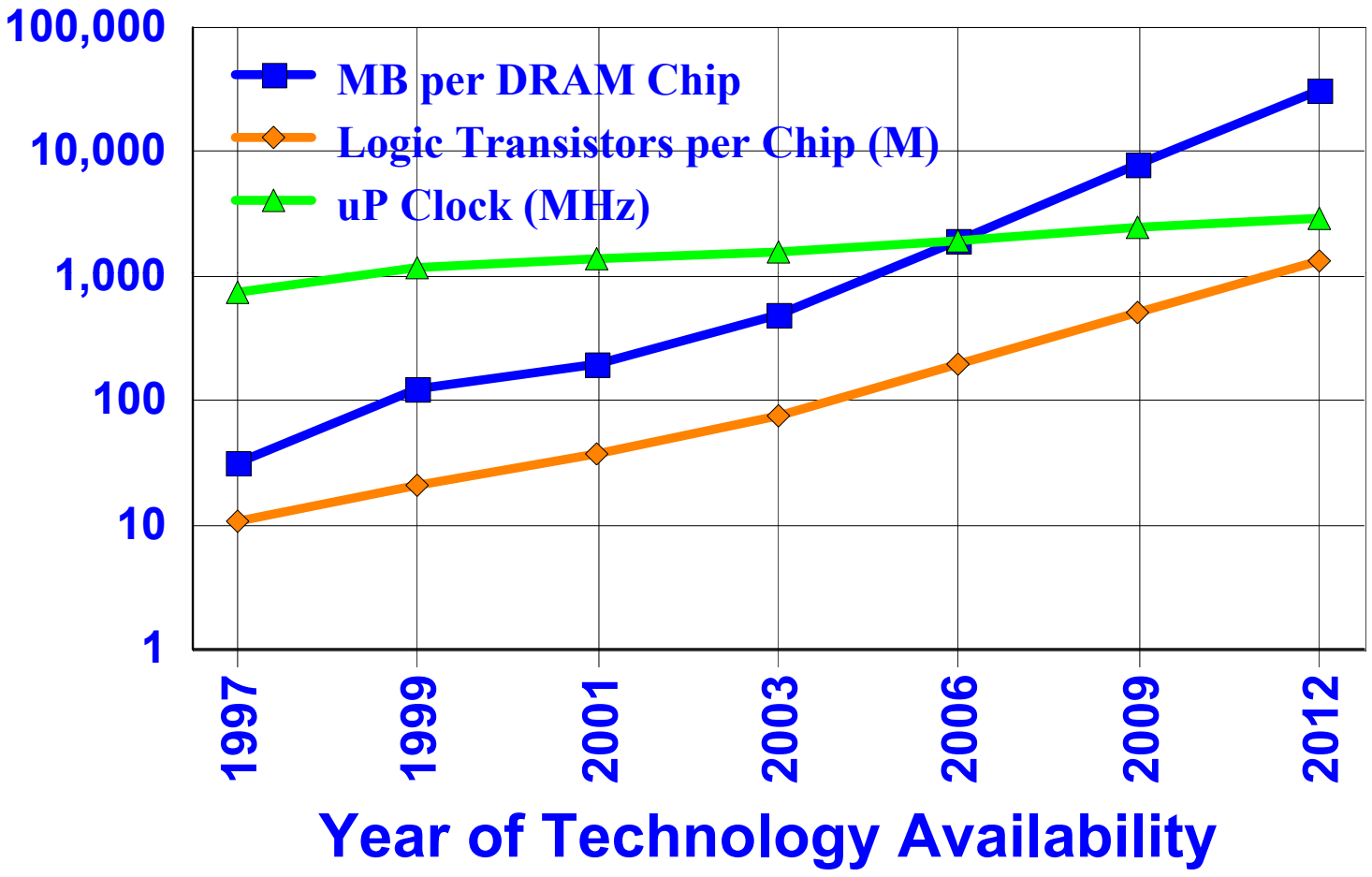




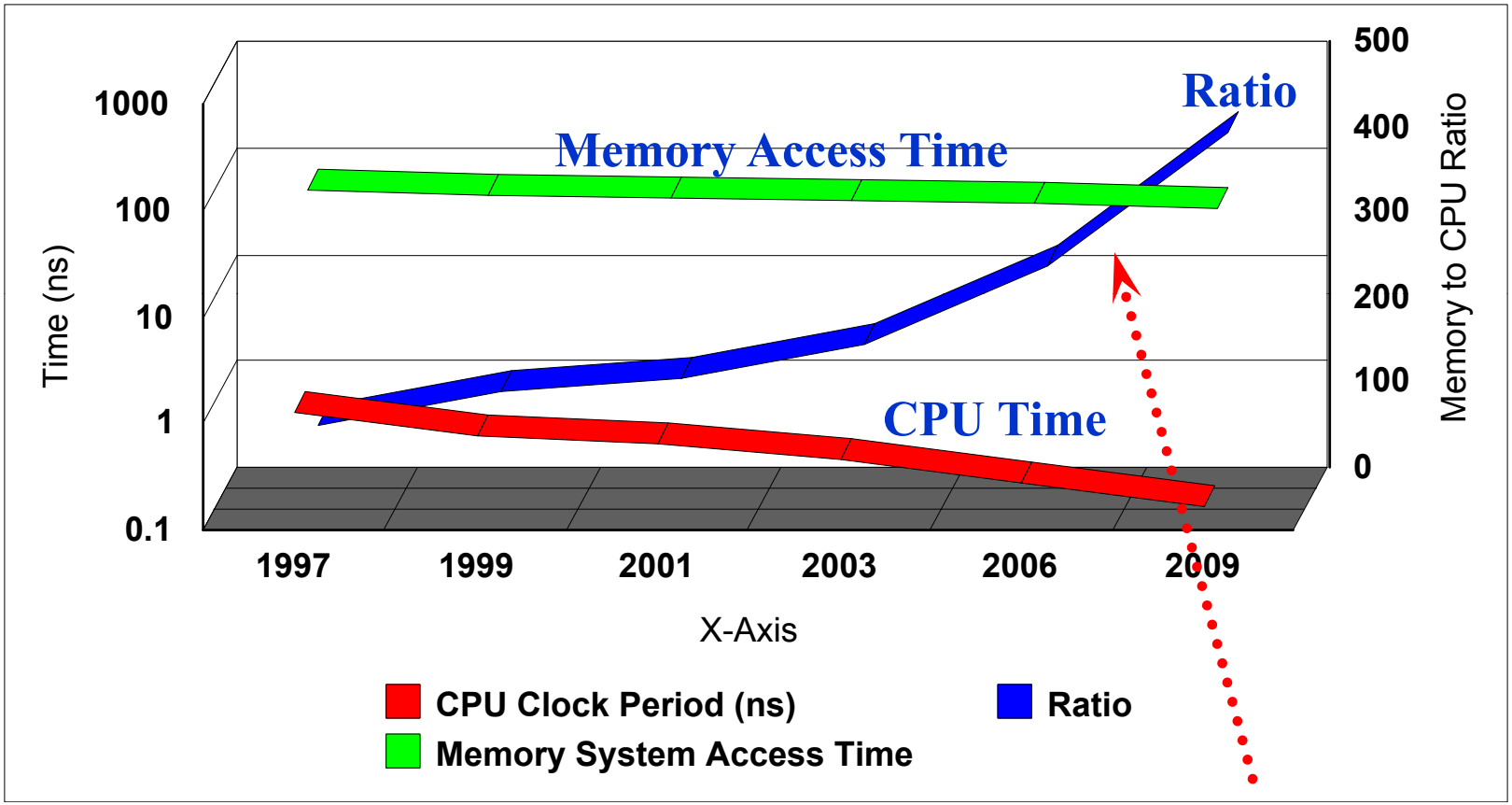
# Microprocessor Clock Speed



# The *SIA ITRS* Roadmap

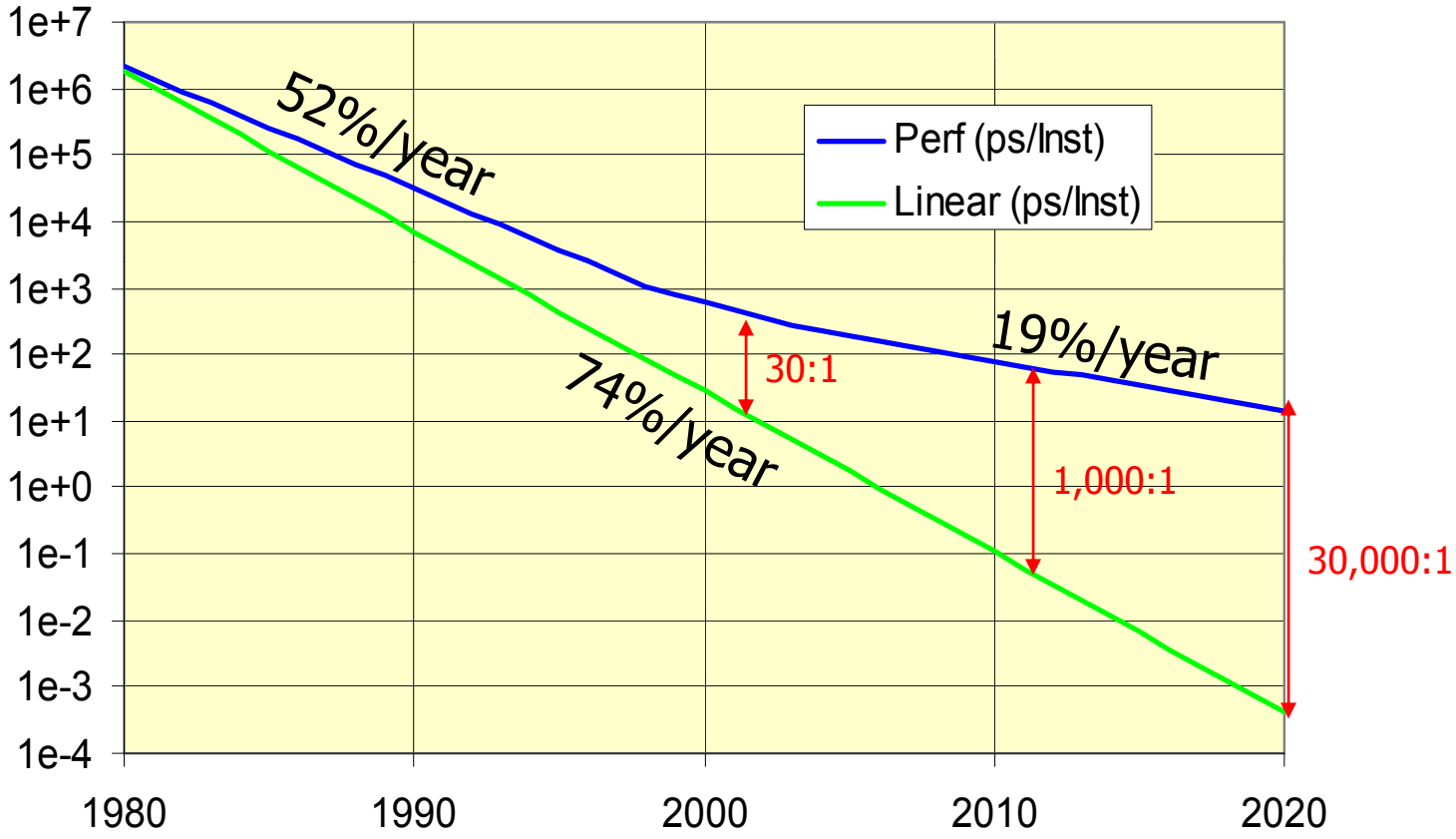


# The Memory Wall




THE *WALL*

# Microprocessors no longer realize the full potential of VLSI technology



# Topics

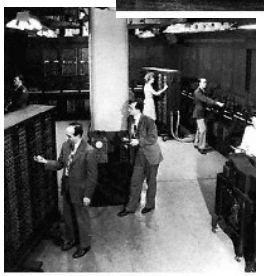
- HPC Applications
- Petaflops Systems
- Enabling Technologies and Trends
- HPC Architectures
- Scaling Factors
- Sources of Performance Degradation
- Parallel Programming Models – Condor
- Parallel Programming Models - MPI
- Parallel Programming Models – OpenMP
- Scaling to a Thousand Petaflops

# The Evolution of HPC









1823
Babbage Difference Engine

1943
Harvard Mark 1

1949
Edsac

1951
Univac 1

1959
IBM 7094

1964
CDC 6600

1976
Cray 1

1982
Cray XMP

1988
Cray YMP

1991
Intel Delta

1996
T3E

1997
ASCI Red

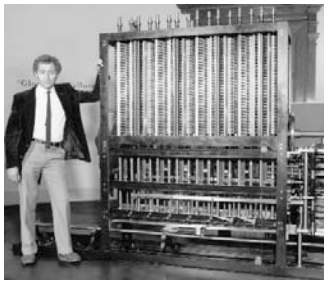
2001
Earth Simulator

2003
Cray X1

2006
Blue Gene/L

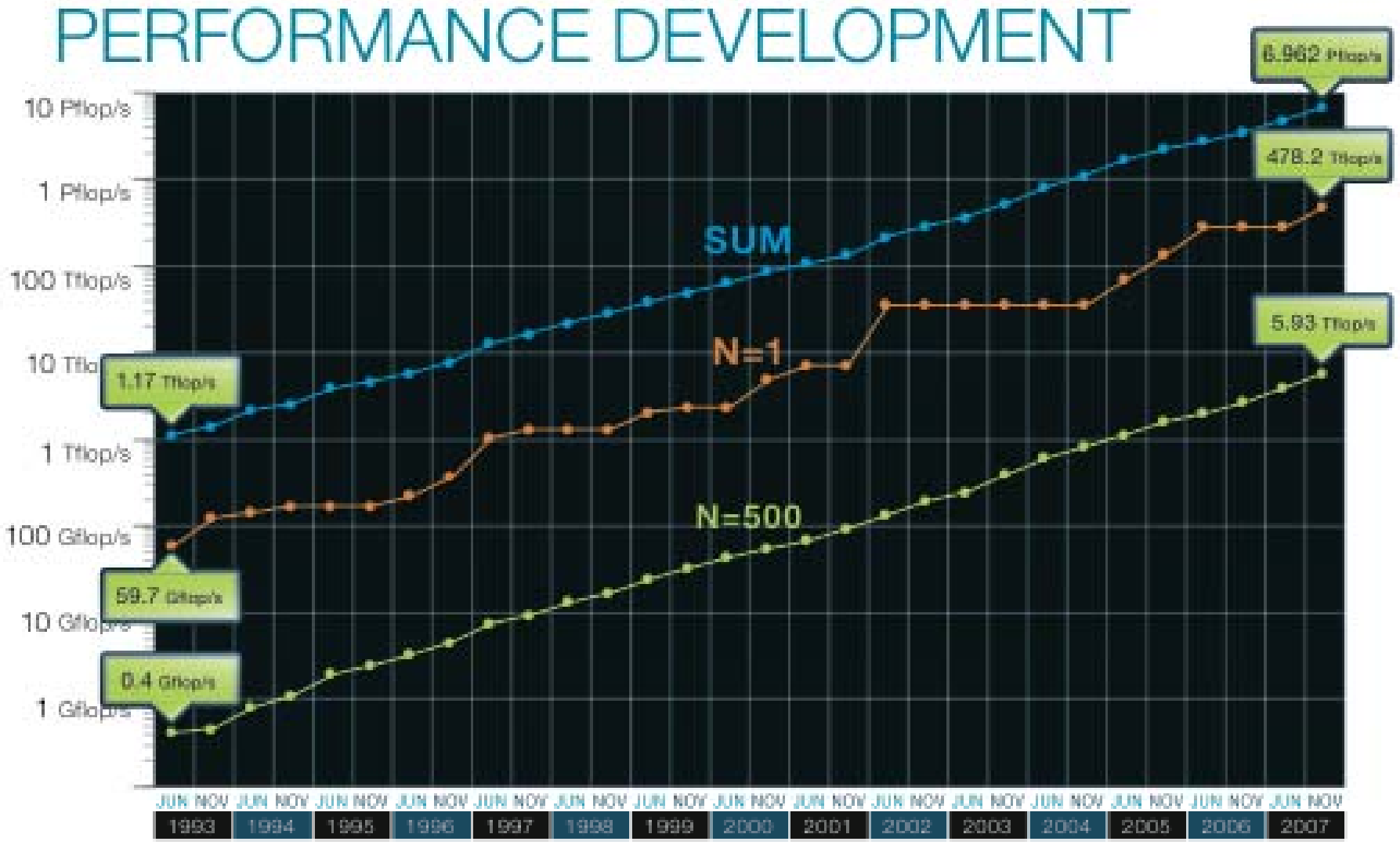
2008
Roadrunner

1 — One OPS — $10^3$ — KiloOPS — $10^6$ — MegaOPS — $10^9$ — GigaOPS — $10^{12}$ — TeraOPS — $10^{15}$ — PetaOPS

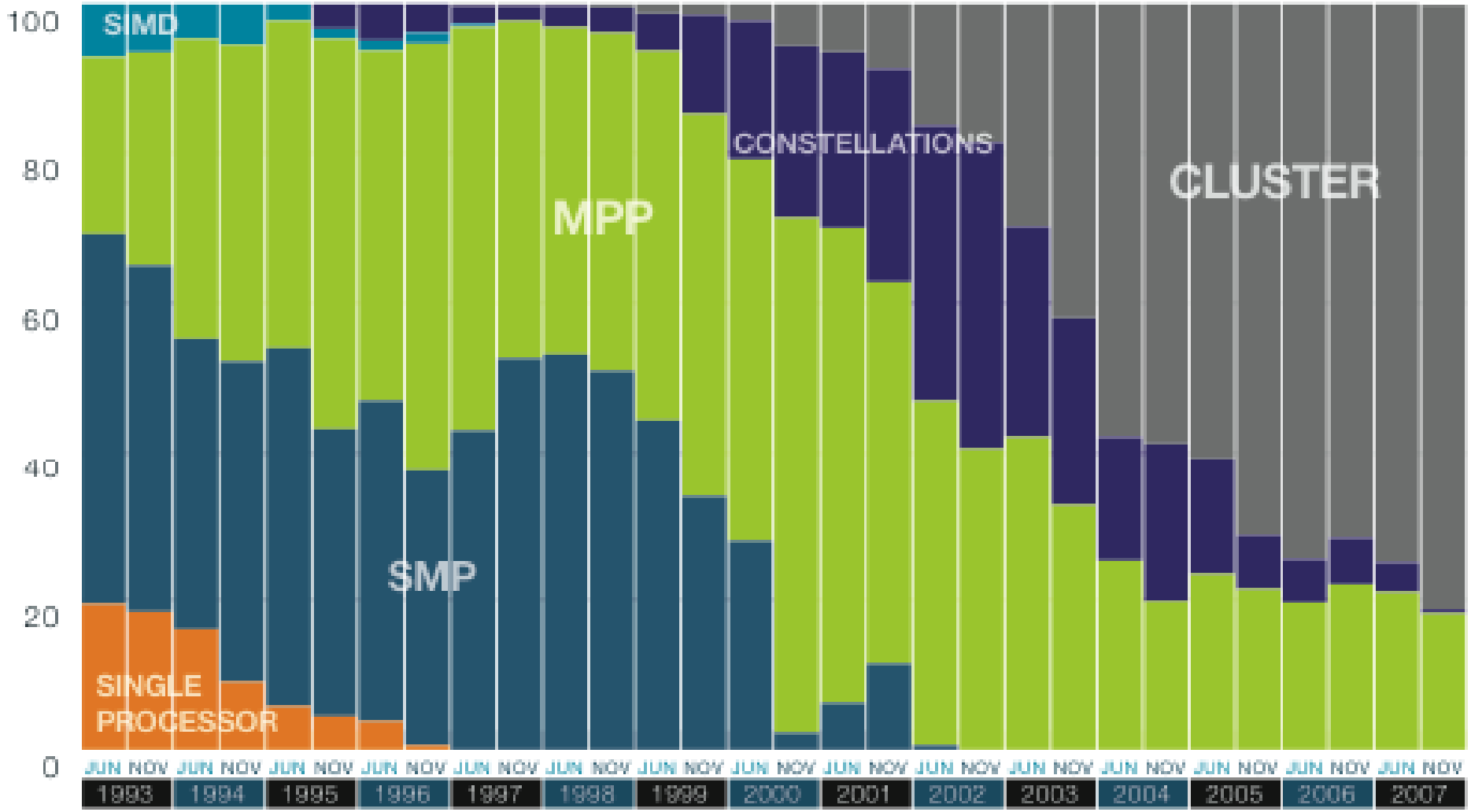




# Top 500 List


PERFORMANCE DEVELOPMENT
6.962 Pflop/s
478.2 Tflop/s
5.93 Tflop/s
SUM
N=1
N=500
1.17 Tflop/s
59.7 Gflop/s
0.4 Gflop/s
10 Pflop/s
1 Pflop/s
100 Tflop/s
10 Tflo
1 Tflop/s
100 Gflop/s
10 Gflop/s
1 Gflop/s
JUN NOV JUN NOV JUN NOV JUN NOV JUN NOV JUN NOV JUN NOV JUN NOV JUN NOV JUN NOV JUN NOV JUN NOV JUN NOV JUN NOV JUN NOV
1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007

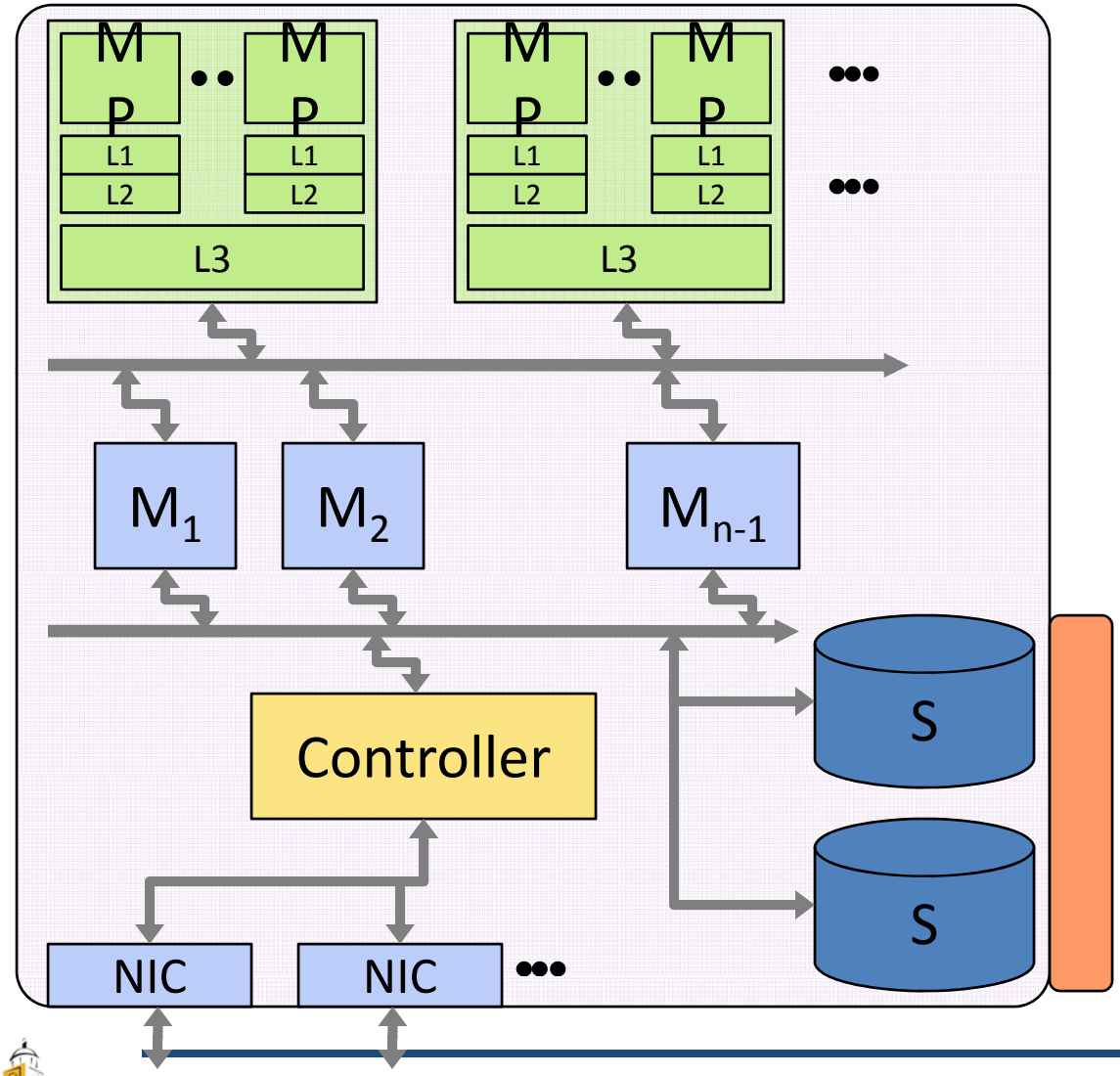
# Top 500: System Architecture



# TSUBAME – A Cluster

| Rank | Site | Computer |
|---|---|---|
| 1 | DOE/NNSA/LLNL<br>United States | BlueGene/L - eServer Blue Gene Solution<br>IBM |
| 2 | Forschungszentrum Juelich (FZJ)<br>Germany | JUGENE - Blue Gene/P Solution<br>IBM |
| 3 | SGI/New Mexico Computing Applications Center (NMCAC)<br>United States | SGI Altix ICE 8200, Xeon quad core 3.0 GHz<br>SGI |
| 4 | Computational Research Laboratories, TATA SONS<br>India | EKA - Cluster Platform 3000 BL460c, Xeon 53xx 3GHz, Infiniband<br>Hewlett-Packard |
| 5 | Government Agency<br>Sweden | Cluster Platform 3000 BL460c, Xeon 53xx 2.66GHz, Infiniband<br>Hewlett-Packard |
| 6 | NNSA/Sandia National Laboratories<br>United States | Red Storm - Sandia/ Cray Red Storm, Opteron 2.4 GHz dual core<br>Cray Inc. |
| 7 | Oak Ridge National Laboratory<br>United States | Jaguar - Cray XT4/XT3<br>Cray Inc. |
| 8 | IBM Thomas J. Watson Research Center<br>United States | BGW - eServer Blue Gene Solution<br>IBM |
| 9 | NERSC/LBNL<br>United States | Franklin - Cray XT4, 2.6 GHz<br>Cray Inc. |
| 10 | Stony Brook/BNL, New York Center for Computational Sciences<br>United States | New York Blue - eServer Blue Gene Solution<br>IBM |

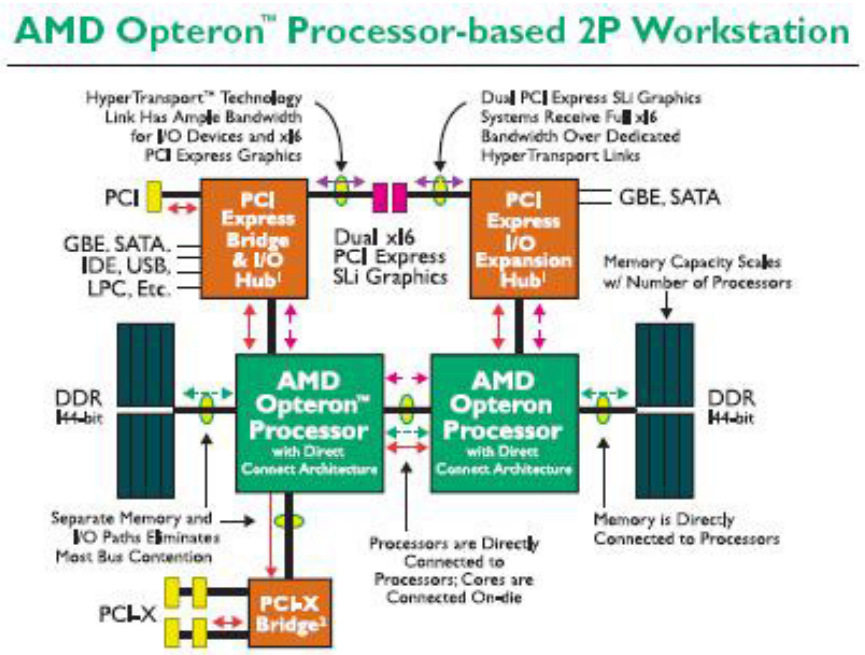
# SMP Node Diagram

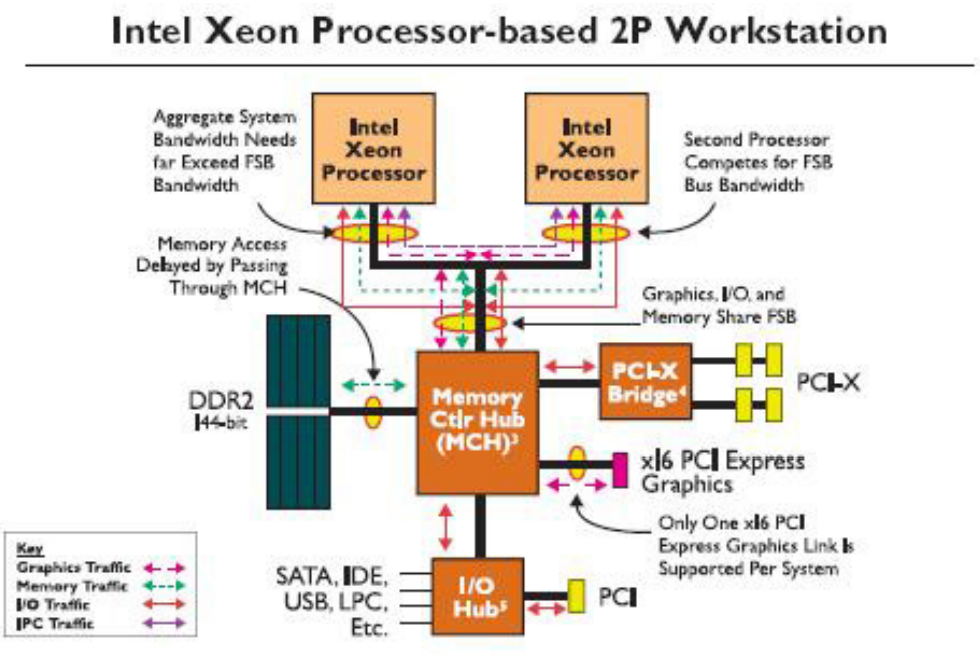



Legend :
MP : MicroProcessor
L1,L2,L3 : Caches
M1.. : Memory Banks
S : Storage
NIC : Network Interface Card

PCI-e
JTAG
Ethernet
Peripherals
USB

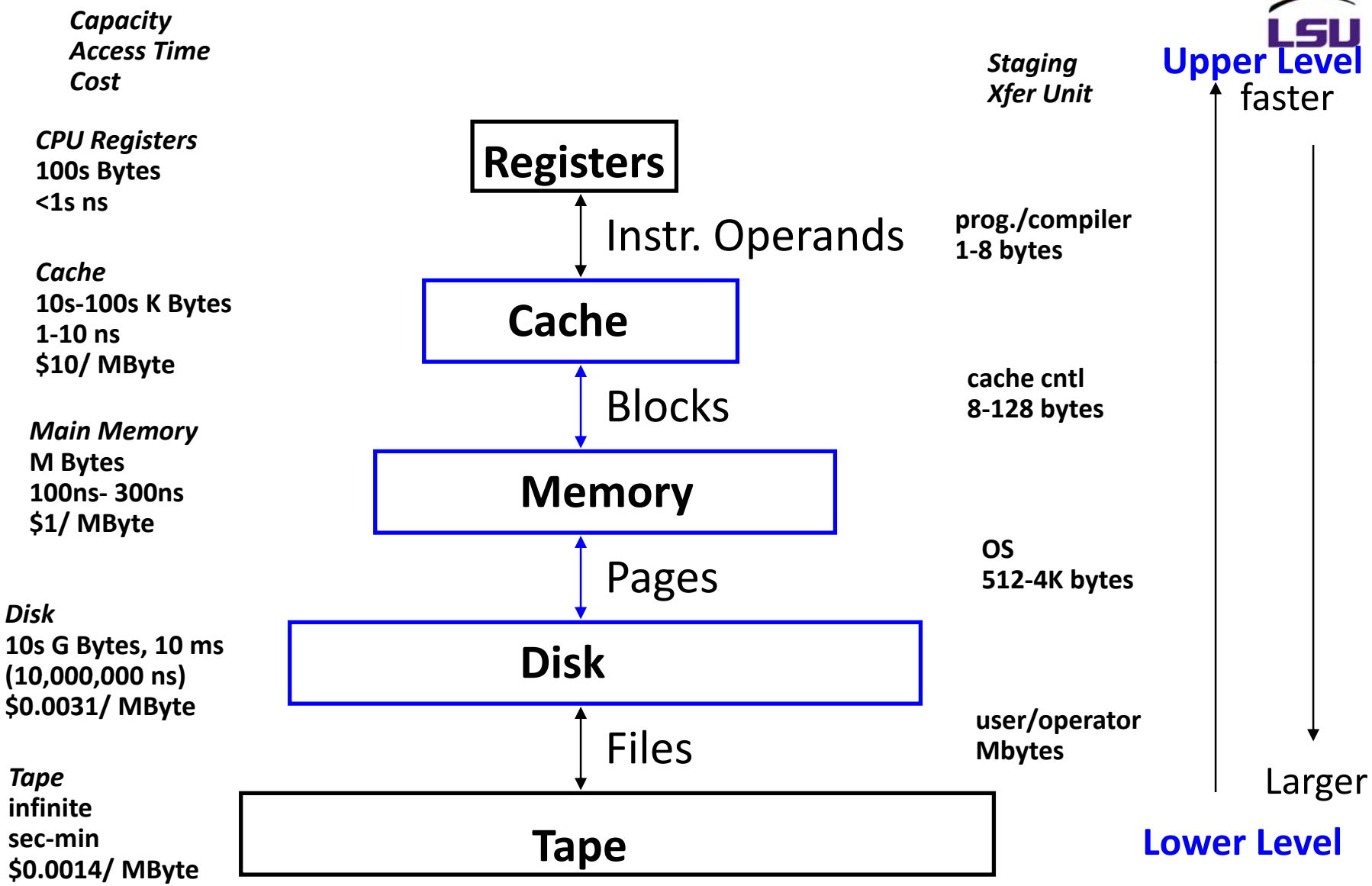
# Comparison of Opteron and Xeon SMP Systems



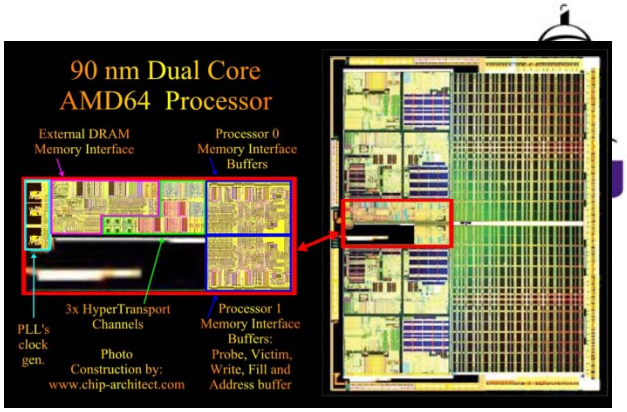

Source: http://www.devx.com/amd/Article/17437

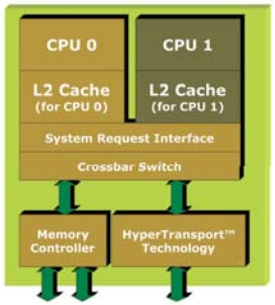
# Levels of the Memory Hierarchy



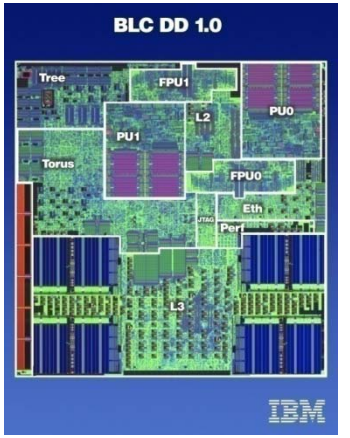
# Multi-Core

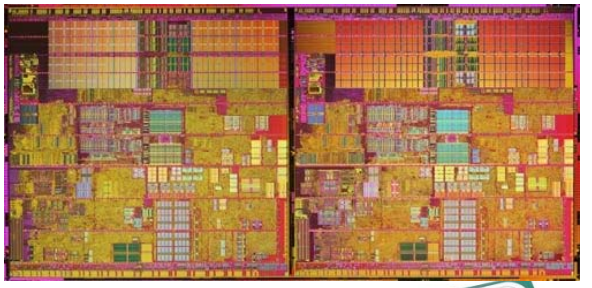
- Motivation for Multi-Core
  - Exploits improved feature-size and density
  - Increases functional units per chip (spatial efficiency)
  - Limits energy consumption per operation
  - Constrains growth in processor complexity
- Challenges resulting from multi-core
  - Relies on effective exploitation of multiple-thread parallelism
    - Need for parallel computing model and parallel programming model
  - Aggravates memory wall
    - Memory bandwidth
      - Way to get data out of memory banks
      - Way to get data into multi-core processor array
    - Memory latency
    - Fragments (shared) L3 cache
  - Pins become strangle point
    - Rate of pin growth projected to slow and flatten
    - Rate of bandwidth per pin (pair) projected to grow slowly
  - Requires mechanisms for efficient inter-processor coordination
    - Synchronization
    - Mutual exclusion
    - Context switching



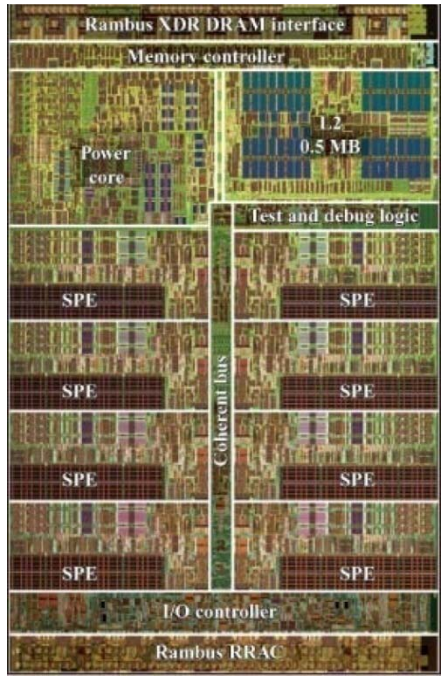
AMD Athlon™ 64 X2
Dual-Core Processor Design



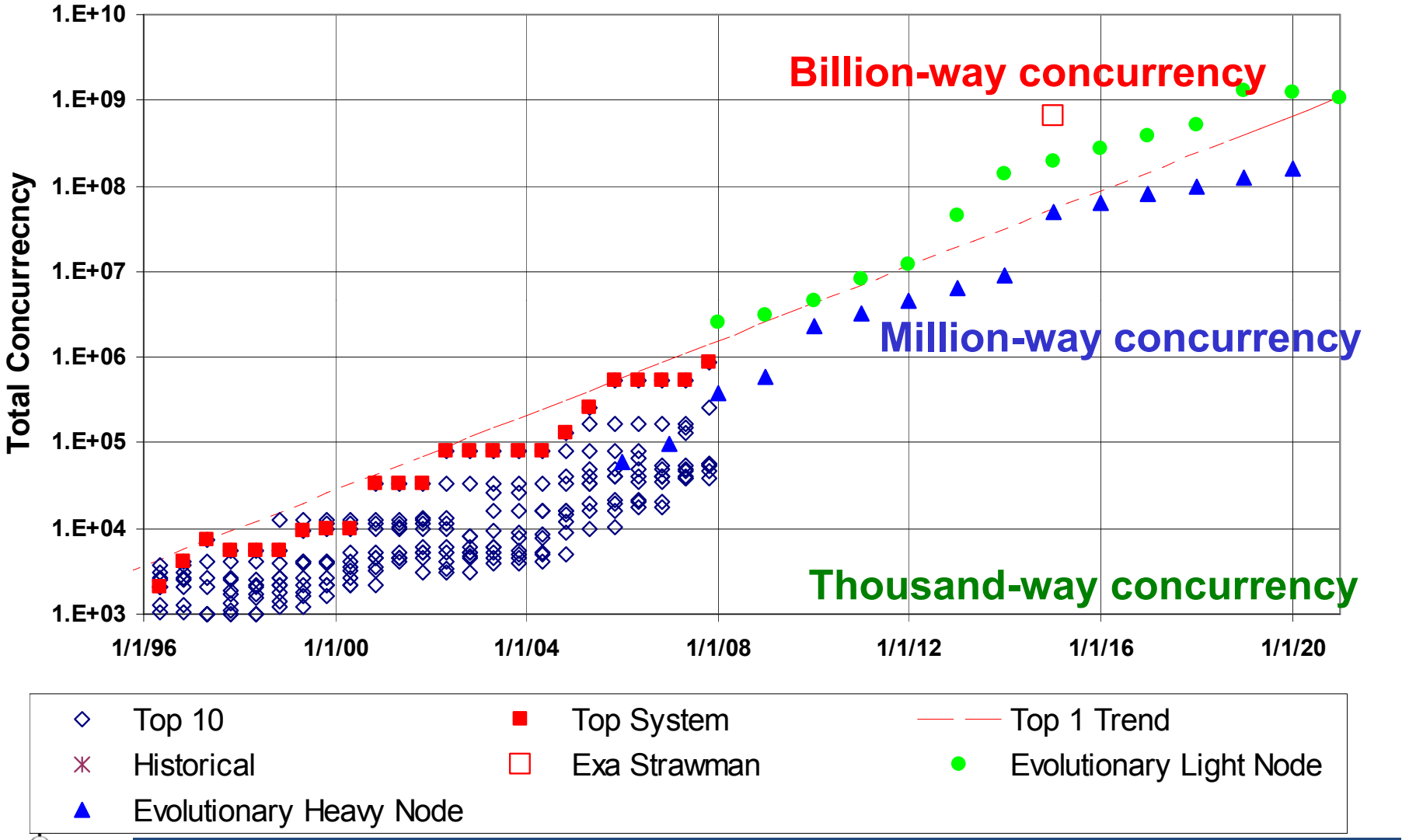



# Heterogeneous Multicore Architecture

- Combines different types of processors
  - Each optimized for a different operational modality
    - Performance > *N*x better than other *N* processor types
  - Synthesis favors superior performance
    - For complex computation exhibiting distinct modalities
- Conventional co-processors
  - Graphical processing units (GPU)
  - Network controllers (NIC)
  - Efforts underway to apply existing special purpose components to general applications
- Purpose-designed accelerators
  - Integrated to significantly speedup some critical aspect of one or more important classes of computation
  - IBM Cell architecture
  - ClearSpeed SIMD attached array processor



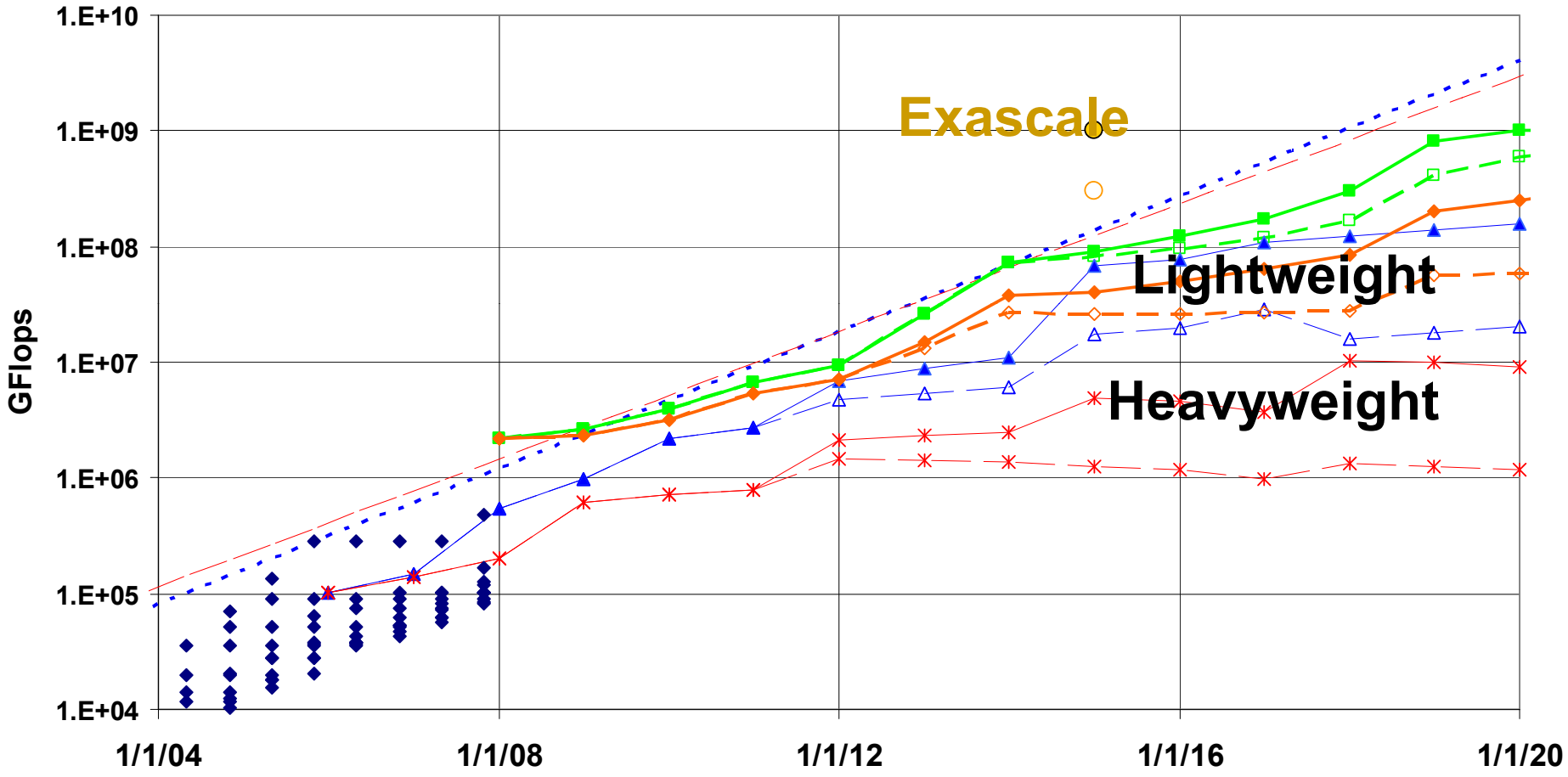

# Data Center Total Concurrency






Courtesy of Peter Kogge, UND

# Data Center Performance Projections




*But not at 20 MW!*

CENTER FOR COMPUTATION & TECHNOLOGY

Courtesy of Peter Kogge, UND

# Topics

- HPC Applications
- Petaflops Systems
- Enabling Technologies and Trends
- HPC Architectures
- Scaling Factors
- Sources of Performance Degradation
- Parallel Programming Models – Condor
- Parallel Programming Models - MPI
- Parallel Programming Models – OpenMP
- Scaling to a Thousand Petaflops

# Key Terms and Concepts

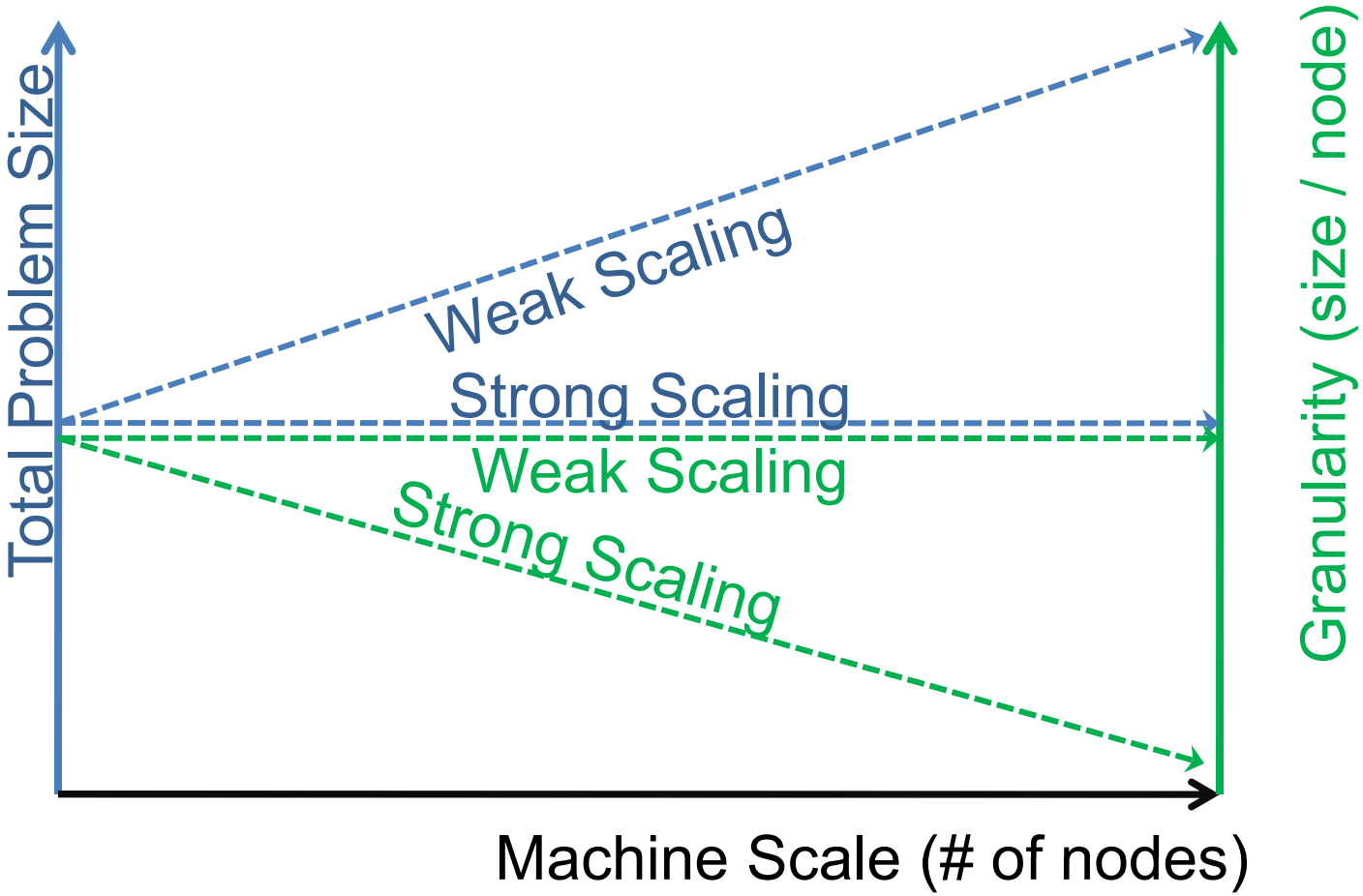
- *Scalable Speedup:* Relative reduction of execution time of a fixed size workload through parallel execution

$$Speedup = \frac{execution\_time\_on\_one\_processor}{execution\_time\_on\_N\_processors}$$

- *Scalable Efficiency :* Ratio of the actual performance to the best possible performance.

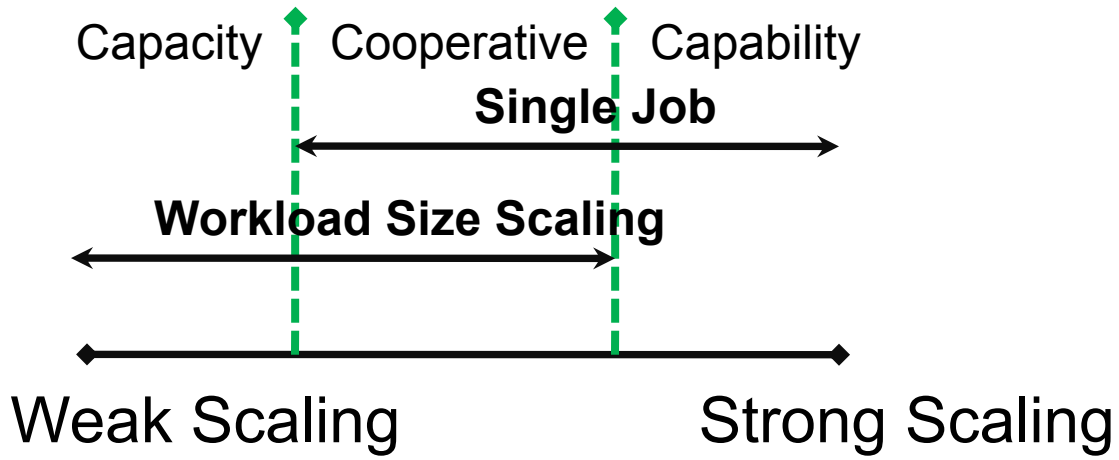
$$Efficiency = \frac{execution\_time\_on\_one\_processor}{(execution\_time\_on\_multiple\_processors \times number\_of\_processors)}$$

# Strong Scaling, Weak Scaling



# Strong Scaling, Weak Scaling

- Capability
  - Primary scaling is decrease in response time proportional to increase in resources applied
  - Single job, constant size – scaling proportional to machine size
- Cooperative
  - Single job, (different nodes working on different partitions of the same job)
  - Job size scales proportional to machine
  - Granularity per node is fixed
- Capacity
  - Primary scaling is increase in throughput  proportional to increase in resources applied
  - Decoupled concurrent tasks, increasing in number of instances – scaling proportional to machine.



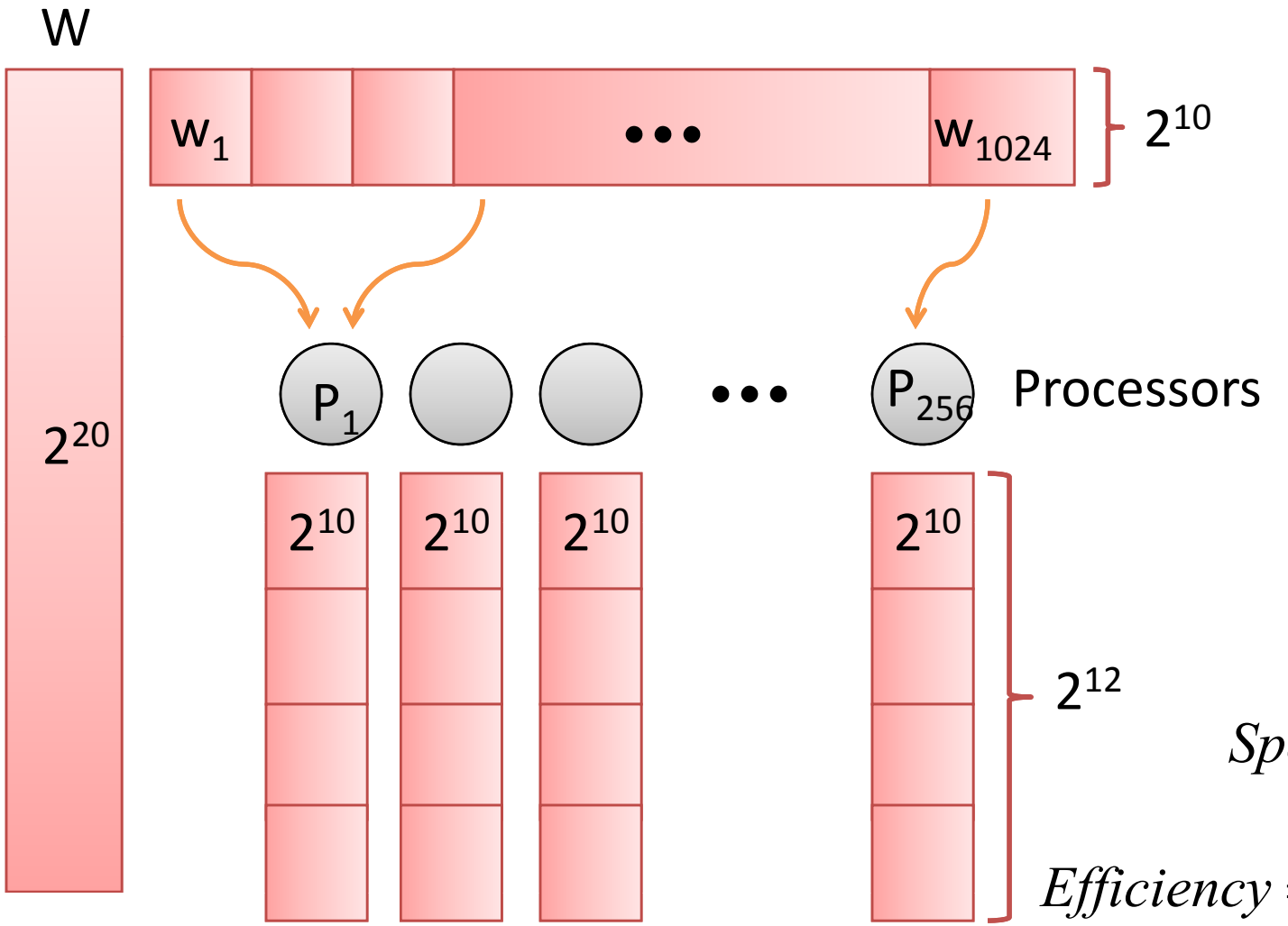
# Topics

- HPC Applications
- Petaflops Systems
- Interlude: the System “hour-glass” Stack
- Enabling Technologies and Trends
- HPC Architectures
- Scaling Factors
- Sources of Performance Degradation
- Parallel Programming Models – Condor
- Parallel Programming Models - MPI
- Parallel Programming Models – OpenMP
- Scaling to a Thousand Petaflops

# Sources of Performance Degradation
# SLOW

- *Starvation*
  - Not enough work to do due to insufficient parallelism or poor load balancing among distributed resources
- *Latency*
  - Waiting for access to memory or other parts of the system
- *Overhead*
  - Extra work that has to be done to manage program concurrency and parallel resources the real work you want to perform
- *Waiting for Contention*
  - Delays due to fighting over what task gets to use a shared resource next. Network bandwidth is a major constraint.

# Ideal Speedup Example




Units : steps

$$W = \sum_i w_i$$

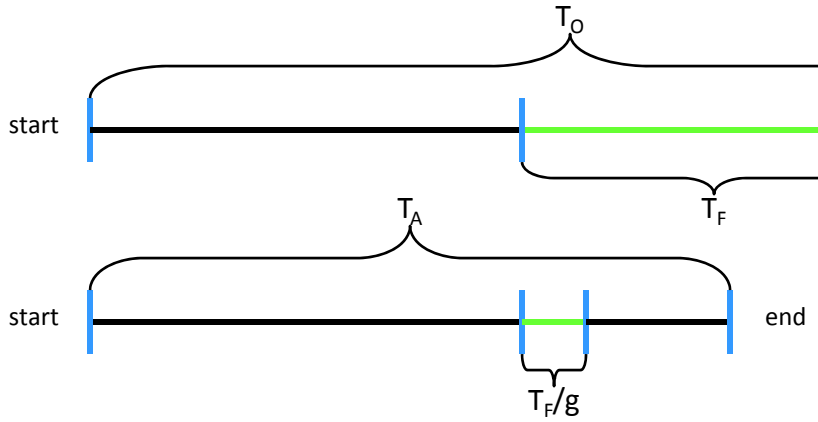
$T(1)=2^{20}$

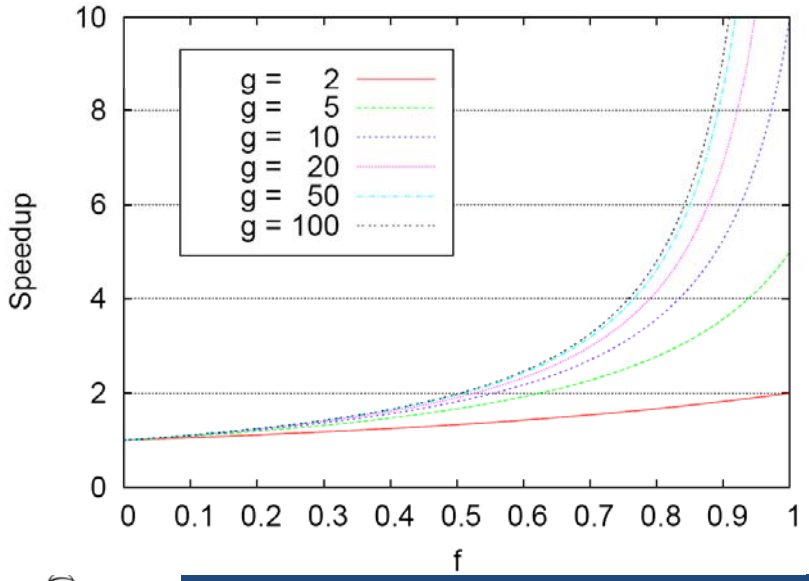
$T(2^8)=2^{12}$

$$Speedup = \frac{2^{20}}{2^{12}} = 2^8$$

$$Efficiency = \frac{2^{20}}{2^{12} \times 2^8} = 2^0 = 1$$

# Amdahl's Law




$T_O \equiv$ time for non - accelerated computation

$T_A \equiv$ time for accelerated computation

$T_F \equiv$ time of portion of computation that can be accelerated

$g \equiv$ peak performance gain for accelerated portion of computation

$f \equiv$ fraction of non - accelerated computation to be accelerated

$S \equiv$ speed up of computation with acceleration applied

$$S = T_O / T_A$$

$$f = T_F / T_O$$

$$T_A = (1 - f) \times T_O + \left(\frac{f}{g}\right) \times T_O$$

$$S = \frac{T_O}{(1 - f) \times T_O + \left(\frac{f}{g}\right) \times T_O}$$

$$\boxed{S = \frac{1}{1 - f + \left(\frac{f}{g}\right)}}$$

# Overhead

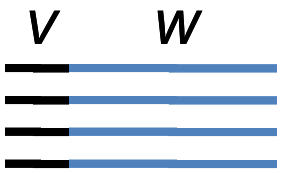
$$W = \sum_{i=1}^{P} w_i \qquad w_i = \frac{W}{P}$$

Assumption : Workload is infinitely divisible

$$T = v + w$$

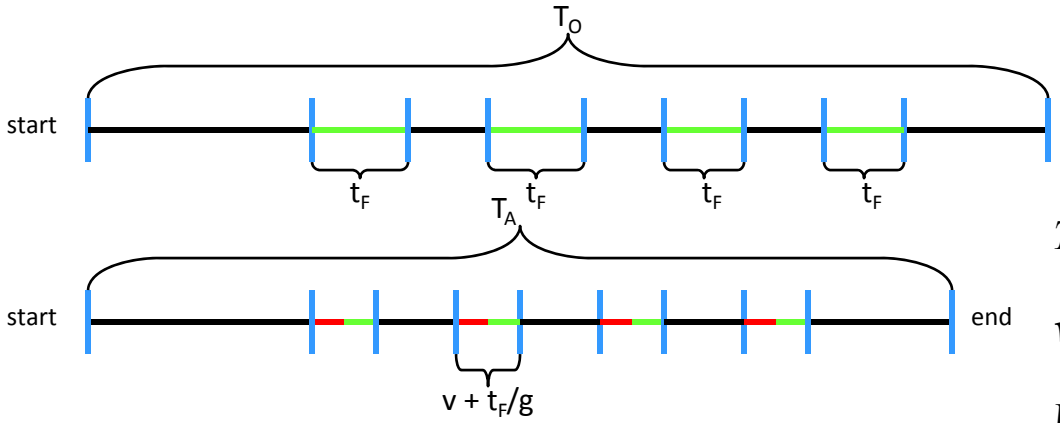
$$S = \frac{T_1}{T_P} = \frac{W + v}{\frac{W}{P} + v} \cong \frac{W}{\frac{W}{P} + v} = \frac{P}{1 + \frac{P \times v}{W}} = \frac{P}{1 + \frac{v}{W/P}}$$

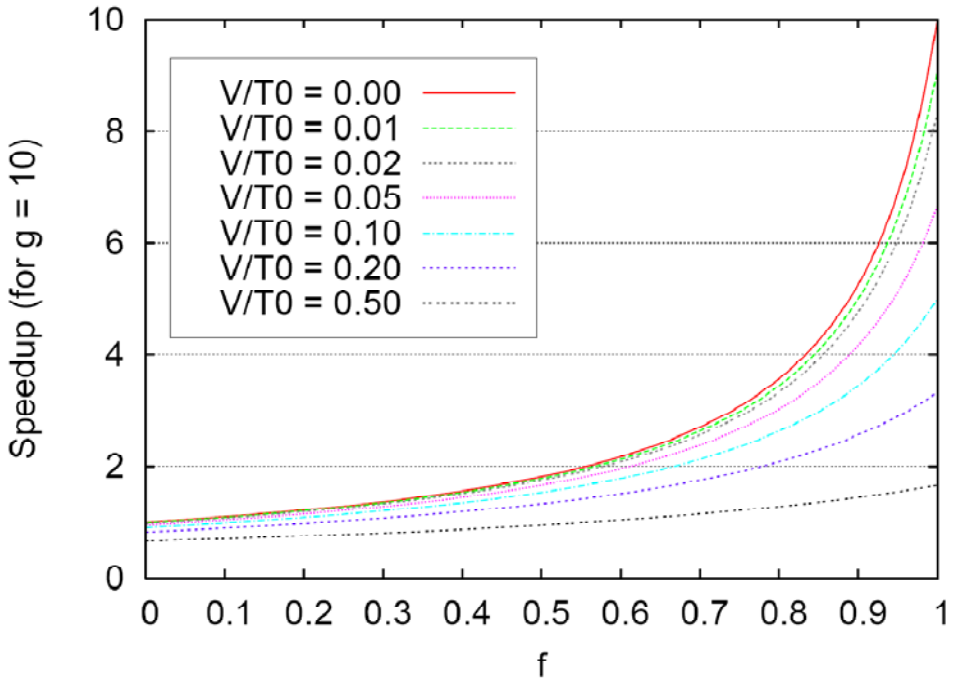
v = overhead
w = work unit
W = Total work
$T_i$ = execution time with i processors
P = # processors




W=4v+4w

# Amdahl's Law with Overhead




$$T_F = \sum_i^n t_{Fi}$$

$v \equiv$ overhead of accelerate d work segment

$V \equiv$ total overhead for accelerate d work $= \sum_i^n v_i$

$$T_A = (1-f)\times T_O + \frac{f}{g}\times T_O + n\times v$$

$$S = \frac{T_O}{T_A} = \frac{T_O}{(1-f)\times T_O + \frac{f}{g}\times T_O + n\times v}$$

$$\boxed{S = \frac{1}{(1-f)+\frac{f}{g}+\frac{n\times v}{T_O}}}$$

# Cache Performance

$$T = I_{count} \times CPI \times T_{cycle}$$

$$I_{count} = I_{ALU} + I_{MEM}$$

$$CPI = \left( \frac{I_{ALU}}{I_{count}} \right) \times CPI_{ALU} + \left( \frac{I_{MEM}}{I_{count}} \right) \times CPI_{MEM}$$

T = total execution time
$T_{cycle}$ = time for a single processor cycle
$I_{count}$ = total number of instructions
$I_{ALU}$ = number of ALU instructions (e.g. register – register)
$I_{MEM}$ = number of memory access instructions ( e.g. load, store)
CPI = average cycles per instructions
$CPI_{ALU}$ = average cycles per ALU instructions

$CPI_{MEM}$ = average cycles per memory instruction
$r_{miss}$ = cache miss rate
$r_{hit}$ = cache hit rate
$CPI_{MEM\text{-}MISS}$ = cycles per cache miss
$CPI_{MEM\text{-}HIT}$=cycles per cache hit
$M_{ALU}$ = instruction mix for ALU instructions
$M_{MEM}$ = instruction mix for memory access instruction

# Cache Performance

$$CPI_{MEM} = CPI_{MEM-HIT} + r_{MISS} \times CPI_{MEM-MISS}$$

$$T = I_{count} \times \left[ \left( M_{ALU} \times CPI_{ALU} \right) + \left( M_{MEM} \times \left( CPI_{MEM-HIT} + r_{MISS} \times CPI_{MEM-MISS} \right) \right) \right] \times T_{cycle}$$

T = total execution time
$T_{cycle}$ = time for a single processor cycle
$I_{count}$ = total number of instructions
$I_{ALU}$ = number of ALU instructions (e.g. register – register)
$I_{MEM}$ = number of memory access instructions ( e.g. load, store)
CPI = average cycles per instructions
$CPI_{ALU}$ = average cycles per ALU instructions

$CPI_{MEM}$ = average cycles per memory instruction
$r_{miss}$ = cache miss rate
$r_{hit}$ = cache hit rate
$CPI_{MEM-MISS}$ = cycles per cache miss
$CPI_{MEM-HIT}$=cycles per cache hit
$M_{ALU}$ = instruction mix for ALU instructions
$M_{MEM}$ = instruction mix for memory access instruction

# Cache Performance: Example

$$I_{count} = 10^{11}$$
$$I_{MEM} = 2 \times 10^{10}$$
$$CPI_{ALU} = 1$$
$$T_{cycle} = 0.5ns$$
$$CPI_{MEM-MISS} = 100$$
$$CPI_{MEM-HIT} = 1$$

$$I_{ALU} = I_{count} - I_{MEM} = 8 \times 10^{10}$$
$$M_{ALU} = \frac{I_{ALU}}{I_{count}} = \frac{8 \times 10^{10}}{10^{11}} = \frac{8}{10} = 0.8$$
$$M_{MEM} = \frac{I_{MEM}}{I_{count}} = \frac{2 \times 10^{10}}{10^{11}} = 0.2$$

$$r_{hitA} = 0.9$$
$$CPI_{MEM-A} = CPI_{MEM-HIT} + r_{MISS-A} \times CPI_{MEM-MISS}$$
$$= 1 + (1 - 0.9) \times 100 = 11$$
$$T_A = 10^{11} \times ((0.8 \times 1) + (0.2 \times 11)) \times 5 \times 10^{-10}$$
$$= 150\,\text{sec}$$

$$r_{hitB} = 0.5$$
$$CPI_{MEM-B} = CPI_{MEM-HIT} + r_{MISS-B} \times CPI_{MEM-MISS}$$
$$= 1 + (1 - 0.5) \times 100 = 51$$
$$T_B = 10^{11} \times ((0.8 \times 1) + (0.2 \times 51)) \times 5 \times 10^{-10}$$
$$= 550\,\text{sec}$$

# Performance: Locality


- *Temporal Locality* is a property that if a program accesses a memory location, there is a much higher than random probability that the same location would be accessed again.
- *Spatial Locality* is a property that if a program accesses a memory location, there is a much higher than random probability that the nearby locations would be accessed soon.
- Spatial locality is usually easier to achieve than temporal locality
- A couple of key factors affect the relationship between locality and scheduling :
  - Size of dataset being processed by each processor
  - How much reuse is present in the code processing a chunk of iterations.

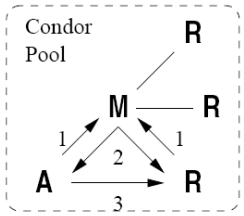
# Topics

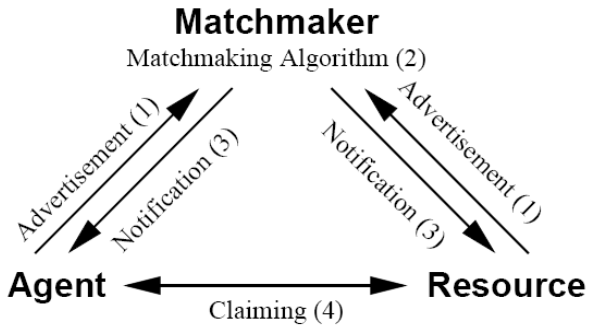
- HPC Applications
- Petaflops Systems
- Enabling Technologies and Trends
- HPC Architectures
- Scaling Factors
- Sources of Performance Degradation
- Parallel Programming Models – Condor
- Parallel Programming Models - MPI
- Parallel Programming Models – OpenMP
- Scaling to a Thousand Petaflops

# Management Middleware: Condor

## Condor MatchMaker

- MatchMaker, a crucial part of the Condor architecture, uses the job description classAd provided by the user and matches the Job to the best resource based on the Machine description classAd
- MatchMaking in Condor is performed in 4 steps :
  1. Job Agent (**A**) and resources (**R**) advertise themselves.
  2. Matchmaker (**M**) processes the known classAds and generates pairs that best match resources and jobs
  3. Matchmaker informs each party of the job-resource pair of their prospective match.
  4. The Job agent and resource establish connection for further processing. (Matchmaker plays no role in this step, thus ensuring separation between selection of resources and subsequent activities)



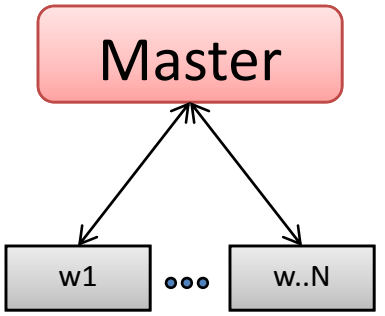



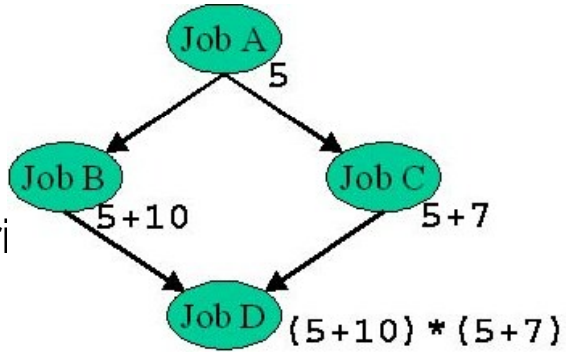

Src : Douglas Thain, Todd Tannenbaum, and Miron Livny, "Distributed Computing in Practice: The Condor Experience" *Concurrency and Computation: Practice and Experience*, Vol. 17, No. 2-4, pages 323-356, February-April, 2005. http://www.cs.wisc.edu/condor/doc/condor-practice.pdf

# Management Middleware: Condor

## Condor Problem Solvers

- **Master-Worker (MW)** is a problem solving system that is useful for solving a coarse grained problem of indeterminate size such as parameter sweep etc.
- The MW Solver in Condor consists of 3 main components : work-list, a tracking module, and a steering module. The work-list keeps track of all pending work that master needs done. The tracking module monitors progress of work currently in progress on the worker nodes. The steering module directs computation based on results gathered and the pending work-list and communicates with the matchmaker to obtain additional worker processes.
- **DAGMan** is used to execute multiple jobs that have dependencies represented as a Directed Acyclic Graph where the nodes correspond to the jobs and edges correspond to the dependencies between the jobs. DAGMan provides various functionalities for job monitori and fault tolerance via creation of rescue DAGs.



# Condor: A Walkthrough of Condor commands


condor_status  : provides current pool status

condor_q : provides current job queue

condor_submit : submit a job to condor pool

condor_rm : delete a job from job queue

# Topics

- HPC Applications
- Petaflops Systems
- Enabling Technologies and Trends
- HPC Architectures
- Scaling Factors
- Sources of Performance Degradation
- Parallel Programming Models – Condor
- Parallel Programming Models - MPI
- Parallel Programming Models – OpenMP
- Scaling to a Thousand Petaflops

# Basic MPI Calls


- In review, the 6 main MPI calls:
  - MPI_Init
  - MPI_Finalize
  - MPI_Comm_size
  - MPI_Comm_rank
  - MPI_Send
  - MPI_Recv
- Include MPI Header file
  - #include “mpi.h”
- Basic MPI Datatypes
  - MPI_INT, MPI_FLOAT, ….

# Example : communicators

```
#include "mpi.h"
#include <stdio.h>

int main( int argc, char *argv[])
{
    int rank, size;
    MPI_Init( &argc, &argv);
    MPI_Comm_rank( MPI_COMM_WORLD, &rank);
    MPI_Comm_size( MPI_COMM_WORLD, &size);
    printf("Hello, World! from %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

Determines the rank of the current process in the communicator-group MPI_COMM_WORLD

Determines the size of the communicator-group MPI_COMM_WORLD

```
…
Hello, World! from 1 of 8
Hello, World! from 0 of 8
Hello, World! from 5 of 8
…
```

# Example : Communicator & Rank


- Compiling :

```
mpicc -o hello2 hello2.c
```

- Result :

```
Hello, World! from 4 of 8
Hello, World! from 3 of 8
Hello, World! from 1 of 8
Hello, World! from 0 of 8
Hello, World! from 5 of 8
Hello, World! from 6 of 8
Hello, World! from 7 of 8
Hello, World! from 2 of 8
```

# MPI : Point to Point Communication primitives


- The basic communication mechanism of MPI between a pair of processes in which one process is sending data and the other process receiving the data, is called "*point to point communication*"
- Message passing in MPI program is carried out by 2 main MPI functions
  - MPI_Send – sends message to a designated process
  - MPI_Recv – receives a message from a process
- Each of the *send* and *recv* calls is appended with information to the data that needs to be exchanged between application programs
- The message envelope consists of the following information
  - The rank of the receiver
  - The rank of the sender
  - A tag
  - A communicator
- The source argument is used to distinguish messages received from different processes
- Tag is user-specified *int* that can be used to distinguish messages from a single process

# Collective Calls

- A communication pattern that encompasses all processes within a communicator is known as collective communication
- MPI has several collective communication calls, the most frequently used are:
  - Synchronization
    - Barrier
  - Communication
    - Broadcast
    - Gather & Scatter
    - All Gather
  - Reduction
    - Reduce
    - AllReduce

# Topics


- HPC Applications
- Petaflops Systems
- Enabling Technologies and Trends
- HPC Architectures
- Scaling Factors
- Sources of Performance Degradation
- Parallel Programming Models – Condor
- Parallel Programming Models - MPI
- Parallel Programming Models – OpenMP
- Scaling to a Thousand Petaflops

# OpenMP: Basic Constructs

To invoke library routines in C/C++ add

```
#include <omp.h>
```

near the top of your code

**OpenMP Execution Model (FORK/JOIN):**

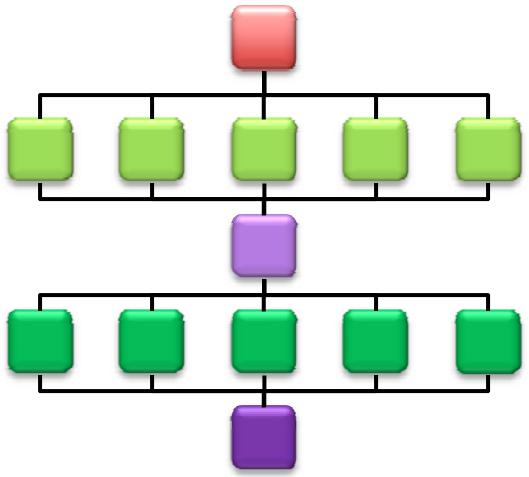
Sequential Part (master thread)

Parallel Region (**FORK :** group of threads)

Sequential Part (**JOIN:** master thread)

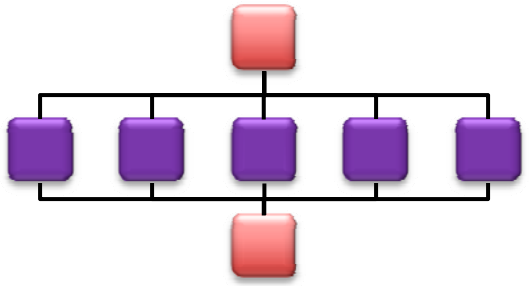
Parallel Region (**FORK:** group of threads)

Sequential Part (**JOIN :** master thread)



**C / C++ :**

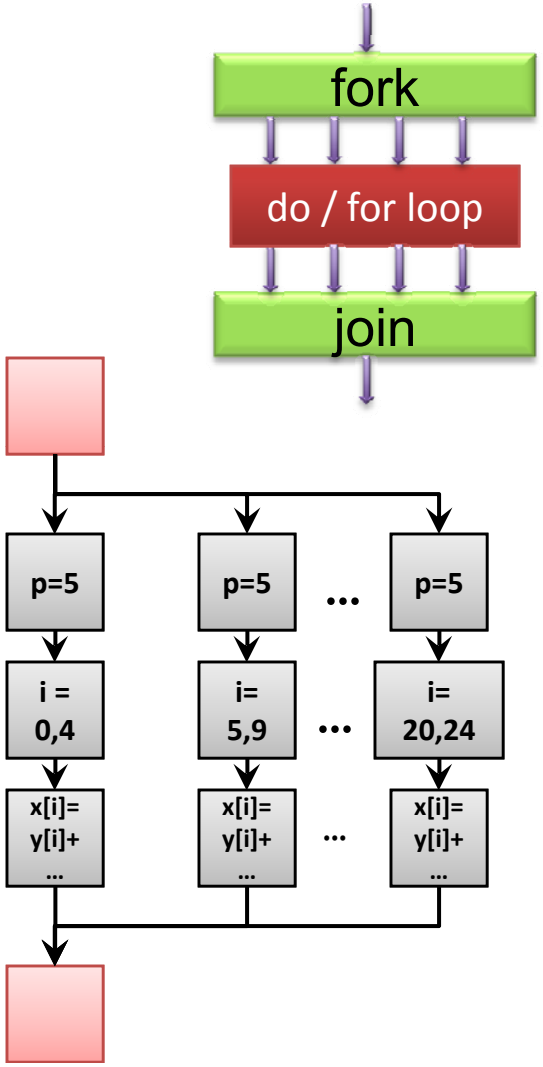
```
#pragma omp parallel {
   parallel block
} /* omp end parallel */
```



# OpenMP *for* directive

- *for* directive helps share iterations of a loop between a group of threads
- If *nowait* is specified then the threads do not wait for synchronization at the end of a parallel loop
- The *schedule* clause describes how iterations of a loop are divided among the threads in the team (discussed in detail in the next few slides)

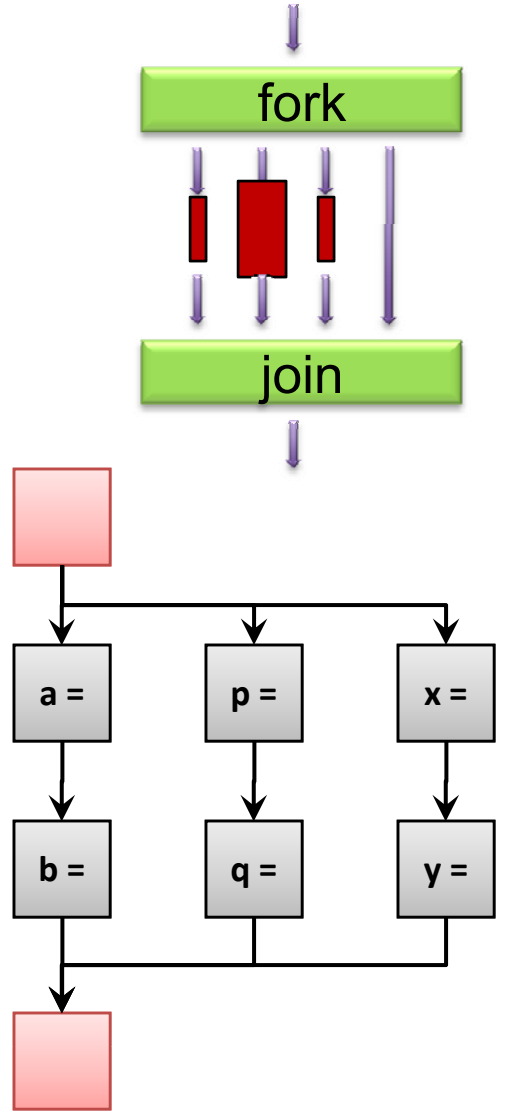
```
#pragma omp parallel
{
  p=5;
  #pragma omp for
     for (i=0; i<24; i++)
         x[i]=y[i]+p*(i+3)
      …
  …
} /* omp end parallel */
```



# OpenMP *sections* directive

- *sections* directive is a non iterative work sharing construct.
- Independent *section* of code are nested within a *sections* directive
- It specifies enclosed *section* of codes between different threads
- Code enclosed within a *section* directive is executed by a thread within the pool of threads

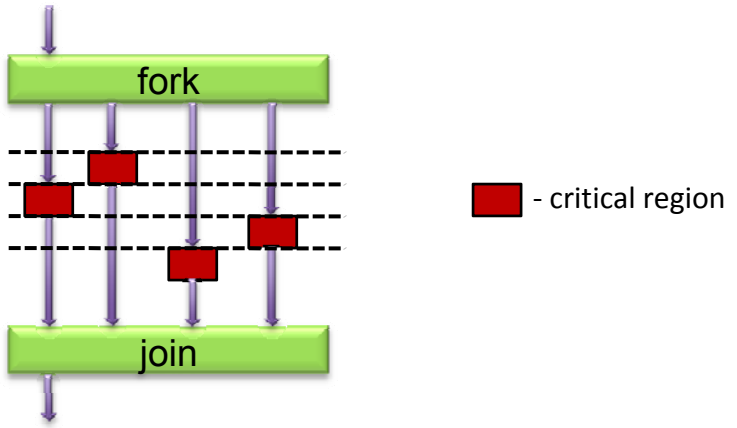
```
#pragma omp parallel private(p)
{
#pragma omp sections
{{  a=…;
   b=…;}
 #pragma omp section
 {  p=…;
   q=…;}
 #pragma omp section
 {  x=…;
   y=…;}
 } /* omp end sections */
} /* omp end parallel */
```



# OpenMP *critical* directive: Explicit Synchronization

- Race conditions can be avoided by controlling access to shared variables by allowing threads to have exclusive access to the variables
- Exclusive access to shared variables allows the thread to *atomically* perform read, modify and update operations on the variable.
- *Mutual exclusion* synchronization is provided by the *critical* directive of OpenMP
- Code block within the *critical region* defined by critical /end critical directives can be executed only by one thread at a time.
- Other threads in the group must wait until the current thread exits the critical region. Thus only one thread can manipulate values in the critical region.

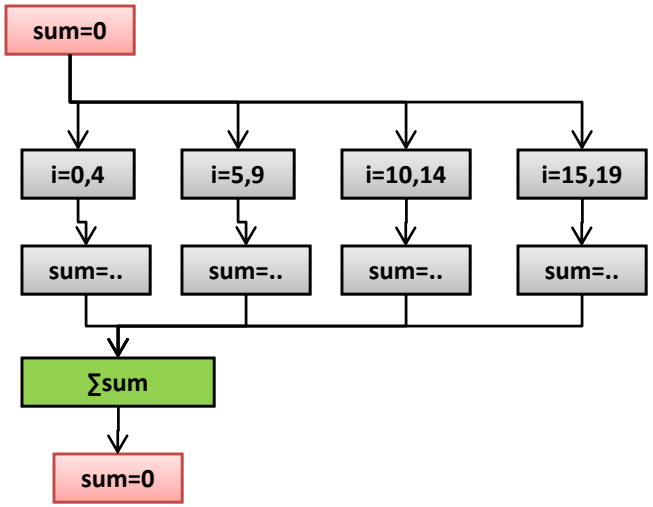
```
int x
x=0;
#pragma omp parallel shared(x)
{
  #pragma omp critical
      x = 2*x + 1;
} /* omp end parallel */
```



# OpenMP: Reduction

- performs reduction on *shared variables* in list based on the operator provided.
- for C/C++ operator can be any one of :
  - +, *, -, ^, |, ||, & or &&
  - At the end of a reduction, the shared variable contains the result obtained upon combination of the list of variables processed using the operator specified.

```
sum = 0.0
#pragma omp parallel for
  reduction(+:sum)
  for (i=0; i < 20; i++)
    sum = sum + (a[i] * b[i]);
```

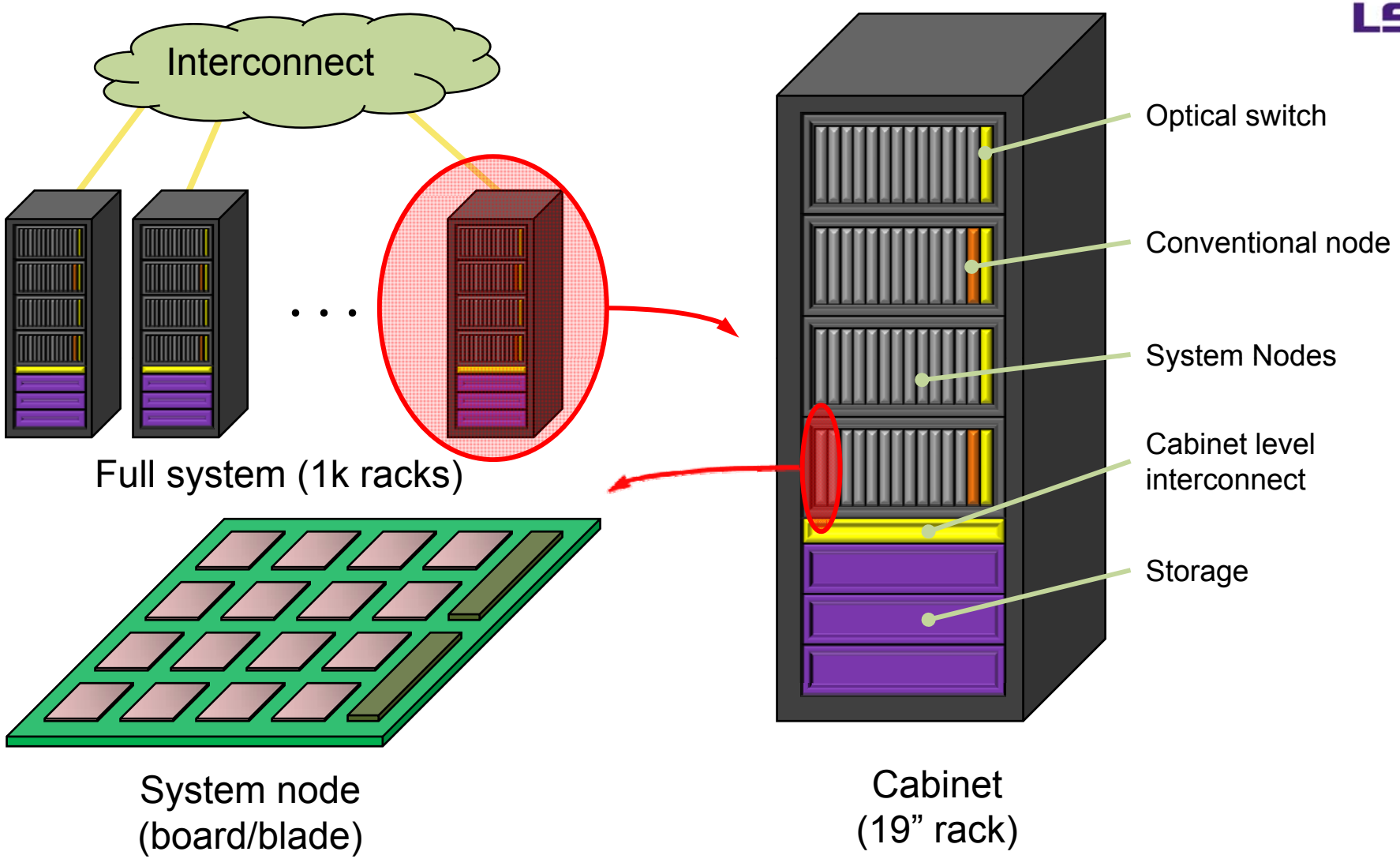


# Topics

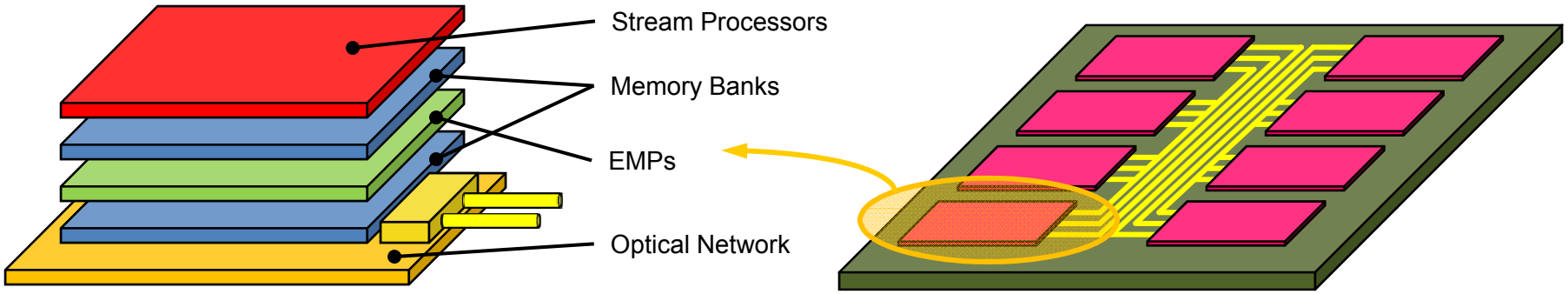
- HPC Applications
- Petaflops Systems
- Enabling Technologies and Trends
- HPC Architectures
- Scaling Factors
- Sources of Performance Degradation
- Parallel Programming Models – Condor
- Parallel Programming Models - MPI
- Parallel Programming Models – OpenMP
- Scaling upto a 1000X Petaflops

# In Review

- 3 classes of parallel/distributed computing
  - Capacity
  - Capability
  - Cooperative
- 3 classes of parallel architectures (respectively)
  - Loosely coupled clusters and workstation farms
  - Tightly coupled vector, SIMD, SMP
  - Distributed memory MPPs (and some clusters)
- 3 classes of parallel execution models (respectively)
  - Workflow, throughput, SPMD
  - Multithreaded with shared memory semantics
  - Communicating Sequential Processes
- 3 classes of programming models
  - Condor
  - MPI
  - OpenMP


Interconnect
Full system (1k racks)
System node
(board/blade)
Optical switch
Conventional node
System Nodes
Cabinet level
interconnect
Storage
Cabinet
(19” rack)

| Cabinet | |
|---|---|
| Organization | 4x 8U compute cages<br>1x 1U optical switch<br>3x 4U secondary storage array |
| Peak performance | 1.18 PFLOPS |
| Memory | 192 TB |
| Secondary storage (effective) | 2.25 PB |
| **Compute Cage** | |
| Blades per cage | 14 |
| Blade housing width | 30 mm |
| Blade type | 12x system nodes<br>1x spare system node or conventional node<br>1x cage-level optical switch |
| **Storage Array** | |
| Raw capacity | 960 TB |
| Effective capacity | 768 TB |
| Number of disks | 80 |
| Disk type | 3.5”, 12 TB, 7200 RPM |
| Redundancy | RAID 6, 8+2 parity |

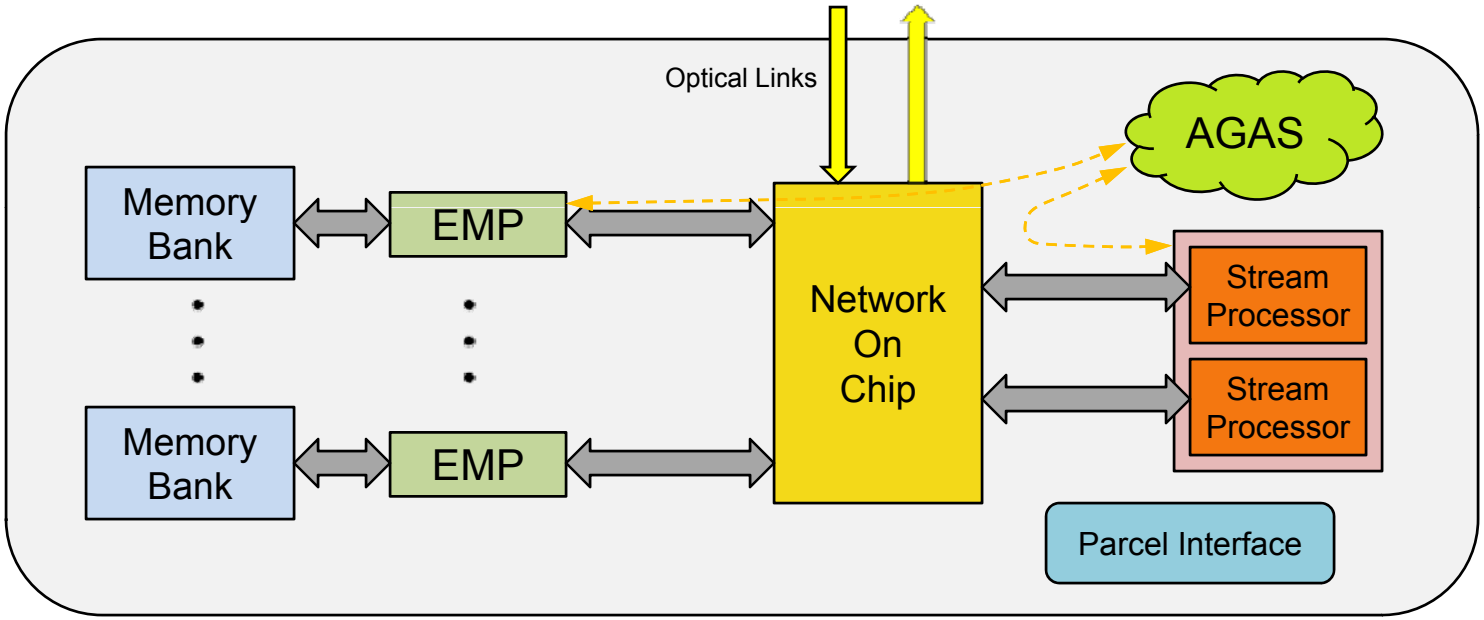



Processor Module

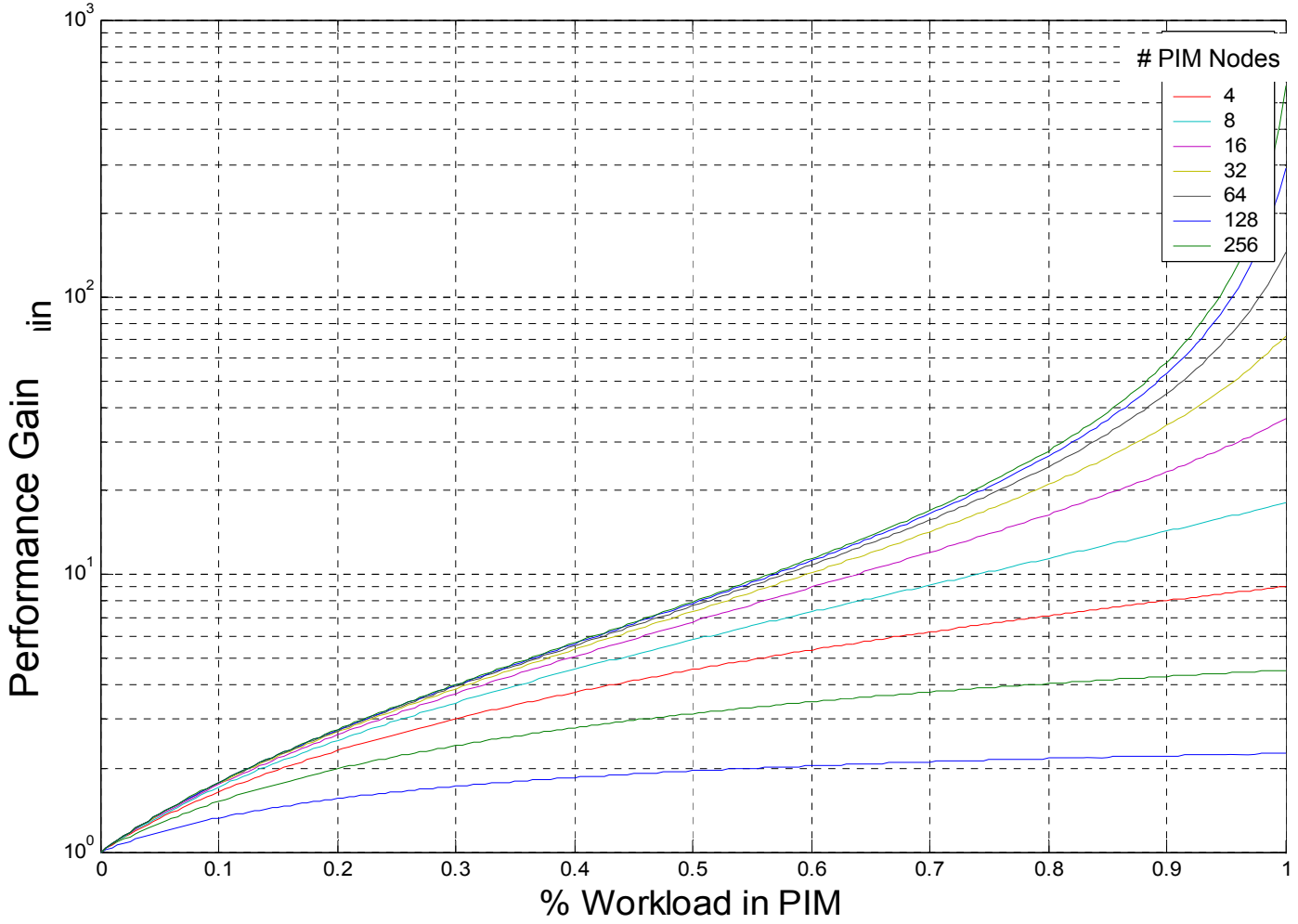
Processor Node
(locality)

| System Node | |
|---|---|
| Processor nodes | 16 |
| Peak performance | 24.5 TFLOPS |
| Memory | 4 TB |
| Board dimensions | 450x330 mm |
| Fraction of board area for processor nodes with local interconnect | 75% |

| Processor Node | |
|---|---|
| Processor modules | 8 |
| Peak performance | 1536 GFLOPS |
| Total memory | 256 GB |
| Aggregate streaming performance | 128 GFLOPS |
| Carrier dimensions | 58.4x86.2 mm |


Optical Links
AGAS
Memory Bank
EMP
Memory Bank
EMP
Network On Chip
Stream Processor
Stream Processor
Parcel Interface

# Speedup with Memory Accelerators



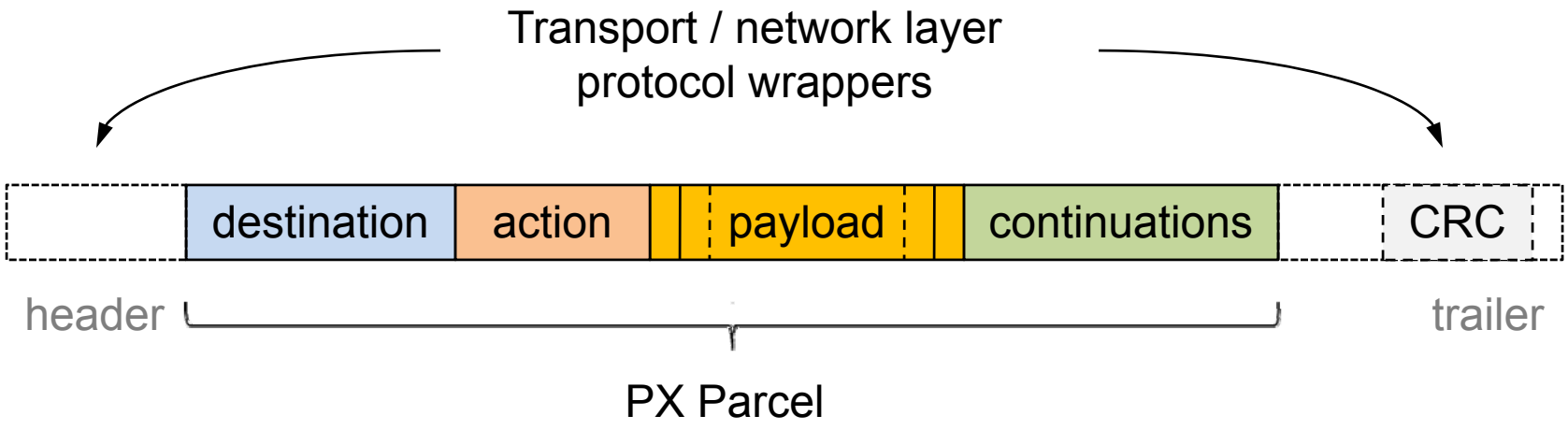

PIM Workload

| Processor Module | |
|---|---|
| Organization | 1 TSV stack (low power density)<br>1 standalone chip (high power density), side by side |
| Aggregate performance | 192 GFLOPS |
| Aggregate EMP performance | 64 GFLOPS |
| Aggregate streaming performance | 128 GFLOPS |
| Number of EMPs | 128 |
| Number of streaming processors | 8 |
| Memory capacity | 32 GB |
| TSV stack footprint | 150 $mm^2$ |
| Standalone chip footprint | 150 $mm^2$ |
| Combined silicon outline | 14.6x20.6 mm |
| Silicon outline to package outline area ratio | 90% |
| Package dimensions | 15.3x21.7 mm |
| **EMP** | |
| Clock frequency | 512 MHz |
| Floating-point op. issue per cycle | 1 |
| Silicon area | 0.5 $mm^2$ |
| **Streaming Processor** | |
| Clock frequency | 2 GHz |
| Peak floating-point ops per cycle | 8 |

# Active Global Address Space (AGAS)


- Distributed
- Assumes no coherence between localities
- User variables
- Synchronization variables and objects
- Threads as first-class objects
- Moves virtual named elements in physical space
- Parcel sets (but not parcels!)
- Process
  - First class object
  - Specifies a broad task
  - Defines a distributed environment
    - Spans multiple localities
    - Need not be contiguous

# Parcel Structure




Parcels may utilize underlying communication protocol fields to minimize the message footprint (e.g. destination address, checksum)

# Latency Hiding with Parcels

## Idle Time with respect to Degree of Parallelism