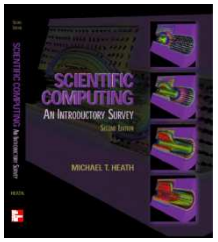
High Performance Computing: An Introduction

Alvaro L.G.A. Coutinho

High Performance Computing Center
COPPE/Federal University of Rio de Janeiro, Brazil
E-mail: alvaro@nacad.ufrj.br
Web: http://www.coc.ufrj.br/~alvaro

Recommended Skills

- **Programming, FORTRAN, C, C++**
- **On-line programming book:**
  - **How to Design Programs**
    - http://www.htdp.org/2001-09-22/
- **Scientific Computing**



» Michael Heath's: http://www.cse.illinois.edu/heath/scicomp/

- **Numerical Methods for PDE's, FD, FV, FE, BE ...**

Contents:

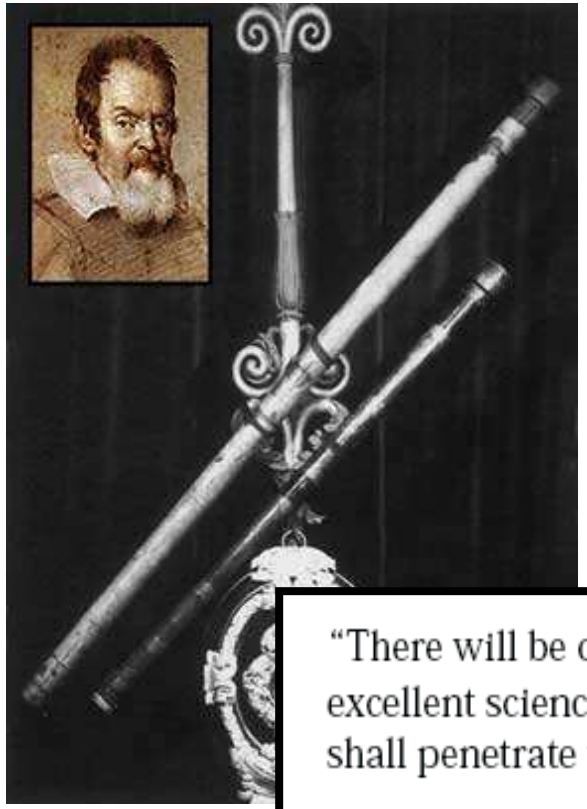
- **Computational Science and Engineering**
- **Basics of Computer Architecture:**
  - What a normal programmer should know about
- **High Performance Computers**
  - What is this? What's on? Why should I care?
- **High Performance Applications**
  - What I can expect? What are the tools? What should I learn?
- **HPC in Computational Mechanics**
  - What are the difficulties? Is it worth doing?
- **Final Comments and Discussion**

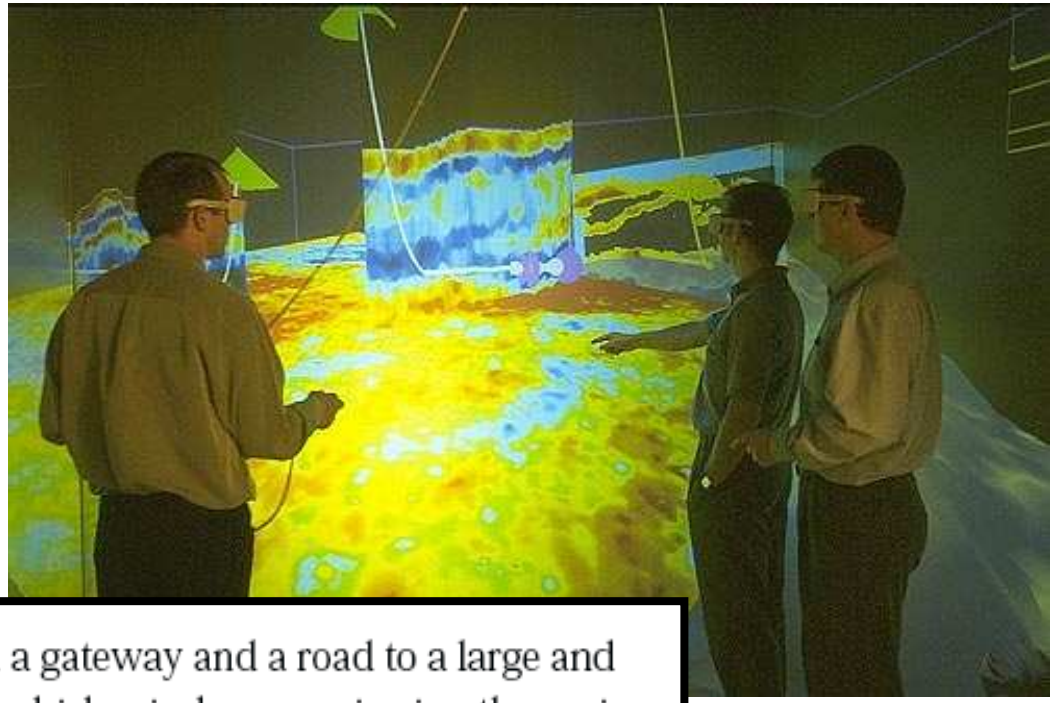
COMPUTATIONAL SCIENCE AND ENGINEERING

Computational Science Engineering

- In broad terms it is about using computers to analyze scientific problems.
- Thus we distinguish it from computer science, which is the study of computers and computation, and from theory and experiment, the traditional forms of science.
- Computational Science and Engineering seeks to gain understanding principally through the analysis of mathematical models on high performance computers.

Computational Science and Engineering





"There will be opened a gateway and a road to a large and excellent science, into which minds more piercing than mine shall penetrate to recesses still deeper." Galileo (1564–1642)

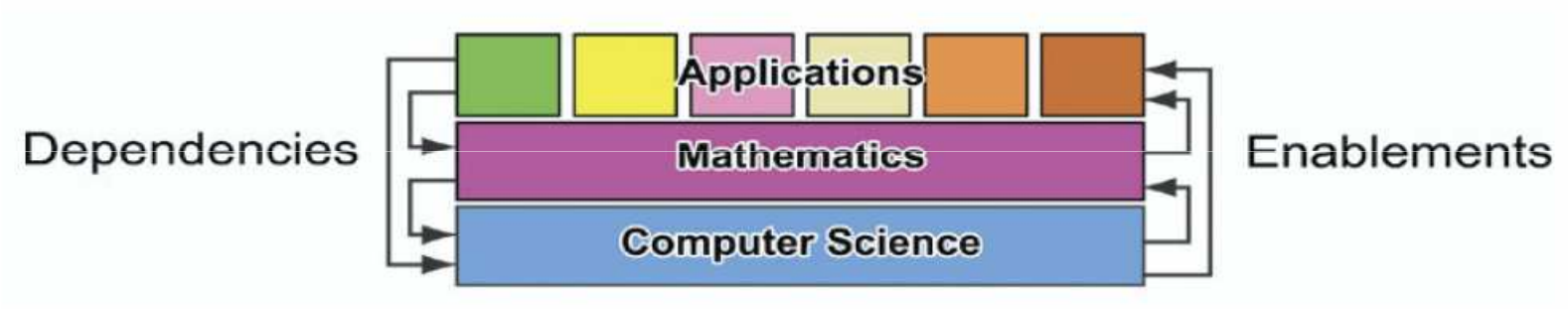
[*on the "experimental mathematical analysis of nature," appropriated here for "computational simulation"*]

Web Material and Food for Thought

- **IEEE Computing in Science and Engineering**
- **Strang's book in CSE**

- **SIAM's Activity Group in CSE**
  - http://www.siam.org/activity/cse/
- **Simulation Based Engineering Science: the future**
  - http://www.ices.utexas.edu/events/SBES_Final_Report.pdf

Layered Structure of CSE



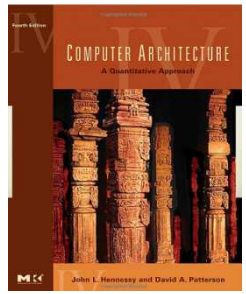
From: A SCIENCE-BASED CASE FOR LARGE-SCALE SIMULATION, DOE, 2003

BASICS OF COMPUTER ARCHITECTURE

Basics of Computer Architecture

- Processors
- Memory
- Buses
- I/O
- Operational Systems
- Performance Model

- See: Hennessy and Patterson, Computer Architecture



The main components of a computer system are:

- ***Processors***
- ***Memory***
- ***Communications Channels***

These components of a computer architecture are often summarized in terms of a PMS diagram.
(P = "processors", M = "memory", S = "switches".)

Processors

Fetch-Decode-Execute Cycle

The essential task of computer processors is to perform a *Fetch-Decode-Execute* cycle:

1. In the *fetch* cycle the processor gets an instruction from memory; the address of the instruction is contained in an internal register called the **Program Counter** or PC.
2. While the instruction is being fetched from memory, the PC is incremented by one. Thus, in the next Fetch-Decode-Execute cycle the instruction will be fetched from the next sequential location in memory (unless the PC is changed by some other instruction in the interim.)
3. In the *decode phase* of the cycle, the processor stores the information fetched from memory in an internal register called the **Instruction Register** or IR.
4. In the *execution phase* of the cycle, the processor carries out the instruction stored in the IR.

Classification of Processor Instructions

Instructions for the processor may be classified into three major types:

1. **Arithmetic/Logic** instructions apply primitive functions to one or two arguments; an example is the addition of two numbers.
2. **Data Transfer** instructions move data from one location to another, for example, from an internal processor register to a location in the main memory.
3. **Control** instructions modify the order in which instruction are executed, for example, in loops or logical decisions.

Clock Cycles

Operations within a processor are controlled by an external clock (a circuit generating a square wave of fixed period).

The quantum unit of time is a *clock cycle*. The clock frequency (e.g., 3GHz would be a common clock frequency for a modern workstation) is one measure of how fast a computer is, but the length of time to carry out an operation depends not only on how fast the processor cycles, but how many cycles are required to perform a given operation.

This is a rather involved function of topics to be discussed below.

Computer Memory

Memory Classifications

Computers have hierarchies of memories that may be classified according to

- Function
- Capacity
- Response Times

Memory Function

"Reads" transfer information from the memory; "Writes" transfer information to the memory:

- ***Random Access Memory*** (RAM) performs both reads and writes.
- ***Read-Only Memory*** (ROM) contains information stored at the time of manufacture that can only be read.

Memory Capacity

bit = smallest unit of memory (value of 0 or 1); commonly abbreviated by "b"
byte = 8 bits; commonly abbreviated by "B"
Common prefixes: k=kilo=1000, M=mega=10^6, G=giga=10^9, Tera=10^12. In modern computers, the total memory may range from say 4Gb in a small personal computer to several TB (terabytes) in large supercomputers. This total memory is divided into sections with different functions (see below).

Memory Response

Memory response is characterized by two different measures:

- ***Access Time*** (also termed *response time* or *latency*) defines how quickly the memory can respond to a read or write request.
- ***Memory Cycle Time*** refers to the minimum period between two successive requests of the memory

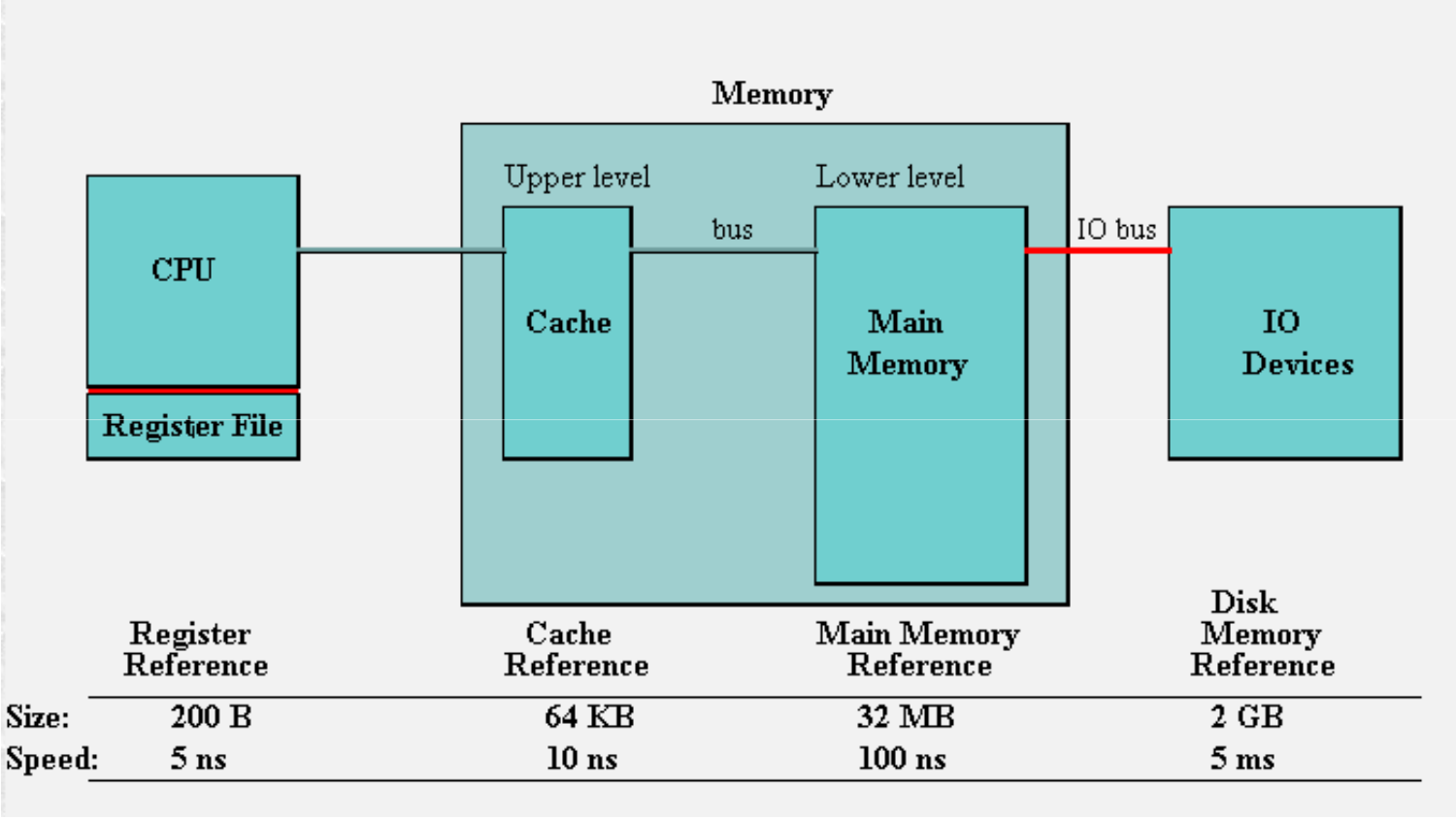
Access times vary from about 80 ns [ns = nanosecond = 10^(-9) seconds] for chips in small personal computers to about 10 ns or less for the fastest chips in caches and buffers (see below). For various reasons, the memory cycle time is more than the speed of the memory chips (i.e., the length of time between successive requests is more than the 80 ns speed of the chips in a small personal computer).

Locality of Reference and Memory Hierarchies

In practice, processors tend to access memory in a patterned way. For example, in the absence of logical branches, the Program Counter is incremented by one after each instruction. Thus, if memory location *x* is accessed at time *t*, there is a high probability that the processor will request an instruction from memory location *x*+1 in the near future. This clustering of memory references into groups is termed **Locality of Reference**.

Locality of reference can be exploited by implementing the memory as a *hierarchy of memories*, with each level of the hierarchy having characteristic access times and capacity.

Memory Hierarchy



fast, expensive slow, cheap

Now the processor sends its request to the fastest, smallest partition of memory (**cache**). If what it wants is there, it can be quickly loaded. If it isn't, the request is forwarded to the next lowest level of the hierarchy and so on.

The key idea is that when the lower (slower and larger and cheaper) members of the hierarchy answer a request from higher levels for the content of location x, they also send at the same time the content of x+1, x+2, ... Because of locality of reference, it is likely that these will be needed in short order, and if they are, they can be loaded quickly from faster memory.

By such a hierarchical scheme, one can improve the effective speed of the memory, even though only a small part of it is composed of fast (and expensive) chips.

Terminology

- *Cache* = fast memory closest to processor.
- *Split Cache* = separate caches for instructions and data.
- *Instruction Buffer* = sophisticated cache for instructions that also performs other functions optimizing the fetching of instructions.
- *Primary Memory* = the main memory below cache.
- *Secondary Memory* = memory at lower end of hierarchy; often disk.
- *Block* = unit of information transferred between items in memory hierarchy.
- *Cache Lines* = blocks transferred to or from cache.
- *Pages* = units transferred between primary and secondary memory.
- *Replacement Strategy* = algorithm for deciding which items in higher memory are discarded to make room for items moved from lower memory. Common possibilities are random, first-in-first-out (FIFO), or least recently used (LRU).
- *Hit* = request satisfied at particular level of memory.
- *Miss* = request that must be passed on to lower memory levels.
- *Hit Rate* = percentage of requests resulting in a hit.

Effective Access Time

The performance of a hierarchical memory is characterized by an **Effective Access Time**. If *T* = effective access time, *H* = cache hit rate, *T(cache)* = cache access time, and *T(main)* = main memory access time,

$$T = H*T(cache) + (1-H)*T(main)$$

For example, if the hit rate is 98% (not uncommon on modern computers), cache speed is 10 ns, and main memory has a speed of 100 ns,

$$T = 0.98*10ns + 0.02*100ns = 11.8\ ns$$

The memory behaves as if it were composed entirely of fast chips with 11.8 ns access time, even though it is composed mostly of cheap 100 ns chips!
Hierarchical memories are complex, but efficient hardware algorithms that work in parallel with other processes to implement the replacement strategy mean that the fetch-decode-execute cycle time is not appreciably lengthened by the implementation of hierarchical memory.

Data Buses

Buses transfer information between parts of a computer. Smaller computers have a single bus; more advanced computers have complex interconnection strategies.

Transaction = Unit of communication on bus.
Bus Master = The module controlling the bus at a particular time.
Arbitration Protocol = Set of signals exchanged to decide which of two competing modules will control a bus at a particular time.
Communication Protocol = Algorithm used to transfer data on the bus.
Asynchronous Protocol = Communication algorithm that can begin at any time; requires overhead to notify receivers that transfer is about to begin.
Synchronous Protocol = Communication algorithm that can begin only at well-know times defined by a global clock.
Transfer Time = Time for data to be transferred over the bus in single transaction.
Bandwidth = Data transfer capacity of bus; usually expressed in ***bits per second*** (bps). Sometimes termed ***throughput***.

Bandwidth and Transfer Time measure related things, but bandwidth takes into account required overheads and is usually a more useful measure of the speed of the bus.

Computer I/O

There are various methods of input and output for a computer. In this section we will discuss only one: output to a video display.

The dominant technology for displaying an image is the ***raster scan***, where a beam of electrons is swept across a screen of phosphors line by line. The beam can be turned on and off very quickly; a phosphor hit by the beam glows white; otherwise, it remains dark.
Refresh Rate = Number of times per second the entire screen is swept by the beam. Rates of 30-60/second are common.
Pixels = "Picture Cells" = individual locations on the screen that can be painted by the beam.
Resolution = Number of pixels per unit length. Modern large screens are often 1280 pixels across and 1024 pixels high. The controller for the electron gun reads from a memory that has 1 bit per pixel. If the bit is 1, the pixel is painted, if it is 0, it isn't painted.

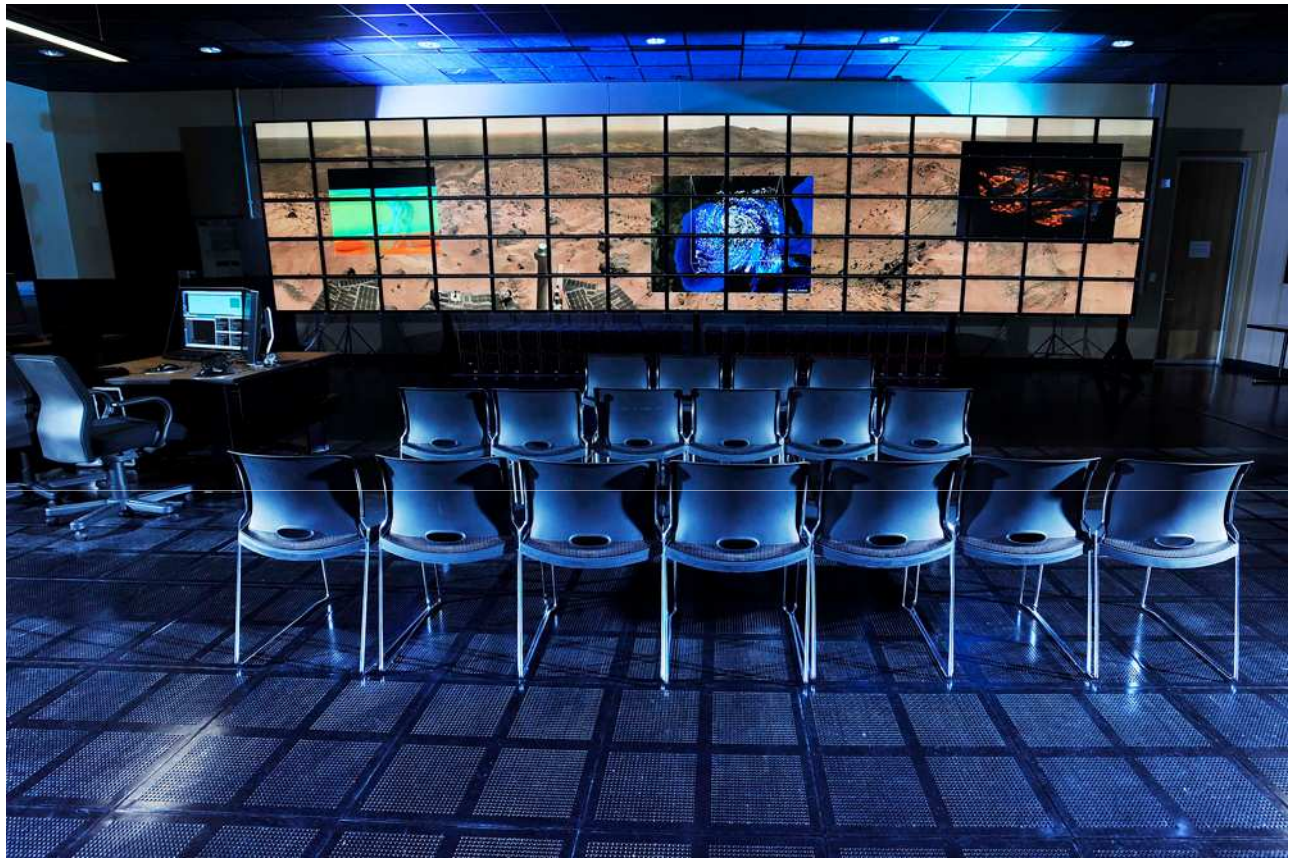
Modern displays use a dedicated memory termed the ***frame buffer*** to hold the bit patterns controlling the display. On inexpensive systems, the main processor computes these patterns and transfers them to the frame buffer. On more sophisticated systems, a dedicated processor called the ***Graphics Engine*** accomplishes this task.

- For ***Color Displays*** the same principles apply, but there are 3 guns (one for each primary color, R = Red, G = Green, B = Blue) and 3 phosphors for each location. To get a large range of colors, the guns must have variable intensities (for example, violet results from 0.61R + 0.24G + 0.80B, where the fractions refer to fractions of full electron intensity).

- Typical systems divide the range of intensities into 2^8 = 256 discrete values. Thus, the intensity can be represented by an 8-bit number for each color, and each pixel requires an 8 x 3 = 24 bit number in the frame buffer to characterize it.

- But 1024 x 1280 x 24 = 32 MB of RAM. Instead, most systems create a ***Color Map*** with a fixed number of 24-bit entries (often 256). Each pixel then maps to one of these entries. Only 8 bits are required to characterize a color map with 256 entries, so 24 - 8 = 16 bits of memory are saved for each pixel. Only 256 colors can be displayed, but that is sufficient for many applications.

Modern Viz Systems: Tiled Wall Displays



Stallion, tiled wall display with 45 Dell monitors, resolution of 184 million pixels
“Visualization Laboratory”, Advanced Computational Engineering and Sciences (ACES), The University of Texas at Austin.

Operating Systems

Most modern computers are ***Multitasking***: they run several ***Processes*** or ***Tasks*** at the same time. The most common operating system for workstations and high-performance computers is ***Unix***.

Active Processes are being executed by the Processing Unit(s)
Idle Processes are waiting to execute
Blocked Processes are waiting for some external event (e.g., the reading of data from a file). When this is accomplished, they become idle, waiting their turn for execution.

Multitasking operating systems let each process run for a short time termed the ***Time Slice*** (a typical time slice might be 20 ms), stops it and changes its status to idle, and then installs one of the idle tasks as the new active program. The system cycles through the tasks in the ***Process Queue***, giving each time slices of a size dictated by some allocation algorithm.
On many Unix systems, the command *top* will give a dynamic display of the jobs in the processing queue and the percentage of processor time each is receiving.
In multitasking systems, one must be careful to distinguish elapsed "wall time" from actual CPU time for a task in evaluating code performance.

Performance Models

Measures of Machine Performance

The *clock cycle time* is a simple, but rather inadequate measure of the performance of a modern compute:

· Processors must act in conjunction with memories and buses.

· The efficiency of executing instructions for each clock cycle can vary widely.

The basic performance for a single-processor computer system can be expressed in terms of

$$T = n \times CPI \times t$$

where T is the time to execute, n is the number of instructions executed, t is the time per instruction, and CPI is the number of cycles per instruction.

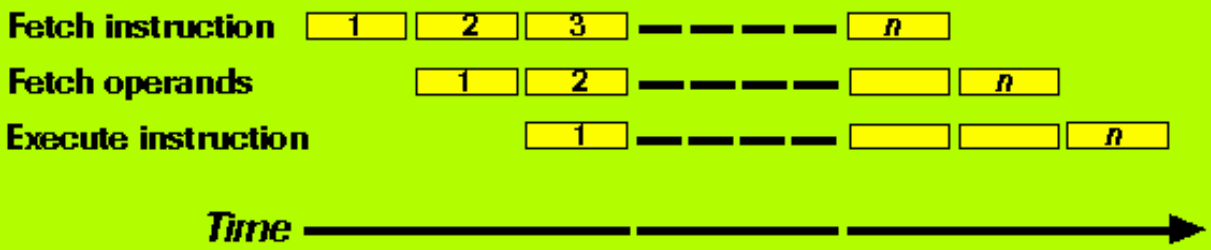
RISC vs. CISC Architectures

RISC (Reduced Instruction Set Computer): Implement a few very simple instructions.
CISC (Complex Instruction Set Computer): Implement a larger instruction set that does more complicated things.

In recent years, the RISC architecture has proven a better match with modern developments in VLSI (Very Large Scale Integration) chip manufacture:

- Simple instructions allow powerful implementation techniques such as ***pipelining*** *(see below).*
- Simple instructions allow more stuff on the chip: on-board cache, CPUs with multiple arithmetic units, etc.
- Simple instruction sets mean cycle times can often be *much* faster.
- Simple instruction sets need less logic, smaller space required on chip, and smaller circuits run faster and cooler.

Overlapping Instructions: Pipelining



A processor can overlap the execution of several instructions. In this example the first instruction is fetched during the first cycle. In the seond cycle, instruction 1 is handed to the part of the processor that prepares operands and the second instruction is fetched from memory. In cycles 3 through n the processor is working on three instructions at a time. Note also that one instruction is completed every cycle. $n + 2$ cycles are required to execute n instructions, so the average number of cycles per instruction is CPI = $((n + 2)/n) = 1.0$.

Figure 3 Pipelined Execution

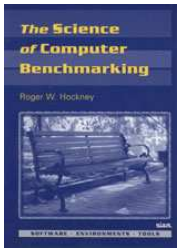
Some Performance Metrics

MIPS = "Millions of instructions per second"

MFLOP/S = "Millions of floating-point operations per second", GigaFlop/s, TeraFlop/s, ExaFlop/s …

Theoretical Peak MFLOP/S = MFLOP/S if the machine did nothing but numerical operations

Benchmarks = Programs designed to determine performance metrics for machines. *Examples:* HPL, NASA-NPB's



See: R.W. Hockney, The Science of Computer Benchmarking, SIAM, 1995

Parallel Processing

The basic performance for a single-processor computer system can be expressed in terms of

$T = n * CPI * t$

where T is the time to execute, n is the number of instructions executed, t is the time per instruction, and CPI is the number of cycles per instruction.
Decreasing the clock time t is a matter of engineering. Generally smaller, faster circuits lead to better clock speed. Decreasing the other two factors involves some version of *parallelism*. There are several levels of parallelism:

1. ***Job-Level Parallelism:*** The computer center purchases more computers so more jobs can be run in a given period.
2. ***Program-Level Parallelism:*** A single program is broken into constituent parts, and different processors compute each part.
3. ***Instruction-Level Parallelism:*** Techniques such as pipelining allow more throughput by the execution of overlapping instructions.
4. ***Arithmetic and Bit-Level Parallelism:*** Low-level parallelism primarily of interest to designers of the arithmetic logic units; relatively invisible to user.

Example: Program-Level Parallelism

This form of parallelism manifested in:

- Independent sections of a program.
- Individual iterations of a loop.

Such parallelism may be exploited by employing multiple processors. For example, consider the loop

```
   do 10 i=1,n
   A(i)=B(i) + C(i)
10 continue
```

This calculates *n* sums that are **independent**: *A(i) + B(i)* does not depend on *A(j) + B(j)* for $j < i$.

Thus, the n sums can be done in any order. In particular, they could be done simultaneously by assigning each of the sums to one of n processors or functional units.

Example: Instruction-Level Parallelism

There are two basic kinds of Instruction-Level Parallelism:

- Individual instructions are ***overlapped*** (executed at the same time) in the processor.
- A given instruction is decomposed into ***sub-operations*** and the sub-operations are overlapped.

A common example of the former type is the overlap of a load instruction that copies something from memory to an internal CPU register, and an arithmetic instruction. The second type is exemplified by ***pipelines***, which will be discussed further later.

Granularity of Tasks

Parallel operations may be classified according to the size of the operations running in parallel.

Large-Grain System: Operations running in parallel are large (of the order of program size).
Small-Grain System: Operations running in parallel are small (of the order of a few instructions).

The preceding example of parallel execution of a do-loop is very *small-grain*.

Pipelined Architectures

Pipelines are now standard components even of smaller computer systems.

The common analogy for a pipeline in a processing unit is the assembly line of normal manufacturing: productivity is increased by dividing the overall task into pieces that can be performed in parallel by separate workers in successive stages of the line.

Pipelines are used for both instruction processing and arithmetic operations. A system is a candidate for pipelined operation if,

- It repeatedly executes a ***basic function.***
- The basic function must be divided into independent ***stages*** having minimal overlap with each other.
- The stages must be of similar complexity.

The number of stages is termed the ***Depth of the Pipeline.***

Example: Pipelined Adder

Consider floating-point addition of two numbers of the form *m*2^e*. This operation could be broken into the following stages:

1. If *e(1) < e(2)*, swap the operands. Find the difference in exponents *e = e(1) - e(2)*.
2. Shift *m(2)* to the right by *e* bits.
3. Compute the mantissa of the sum *m(1) + m(2)*. The exponent of the sum is *e(1)*.
4. Normalize the sum.

The added complexity of such a ***pipelined adder*** pays off if long sequences of numbers are being added: one stage of the adder can be comparing the exponents of one number pair while another stage is adding the mantissas of a ***different pair of numbers*** .

Resource Conflicts and Dependencies

Overlapping operations in a pipeline require that the operations be *independent of each other*. There are various ways in which this condition may be violated. Such *Resource Conflicts* or *Dependencies* inhibit the pipelining efficiency. For example, suppose a code implements the instructions

R2 = R0 + R1

R4 = R2 + R1

This is an example of a *data dependency:* the processor cannot send the second pair of operands to the pipeline adder until the result of the first addition has exited the pipeline (because only then will the correct value of R2 be known). Such dependencies lead to periods when the pipeline stages are empty that are termed *bubbles*. These are well represented in terms of what are called Gantt Charts.

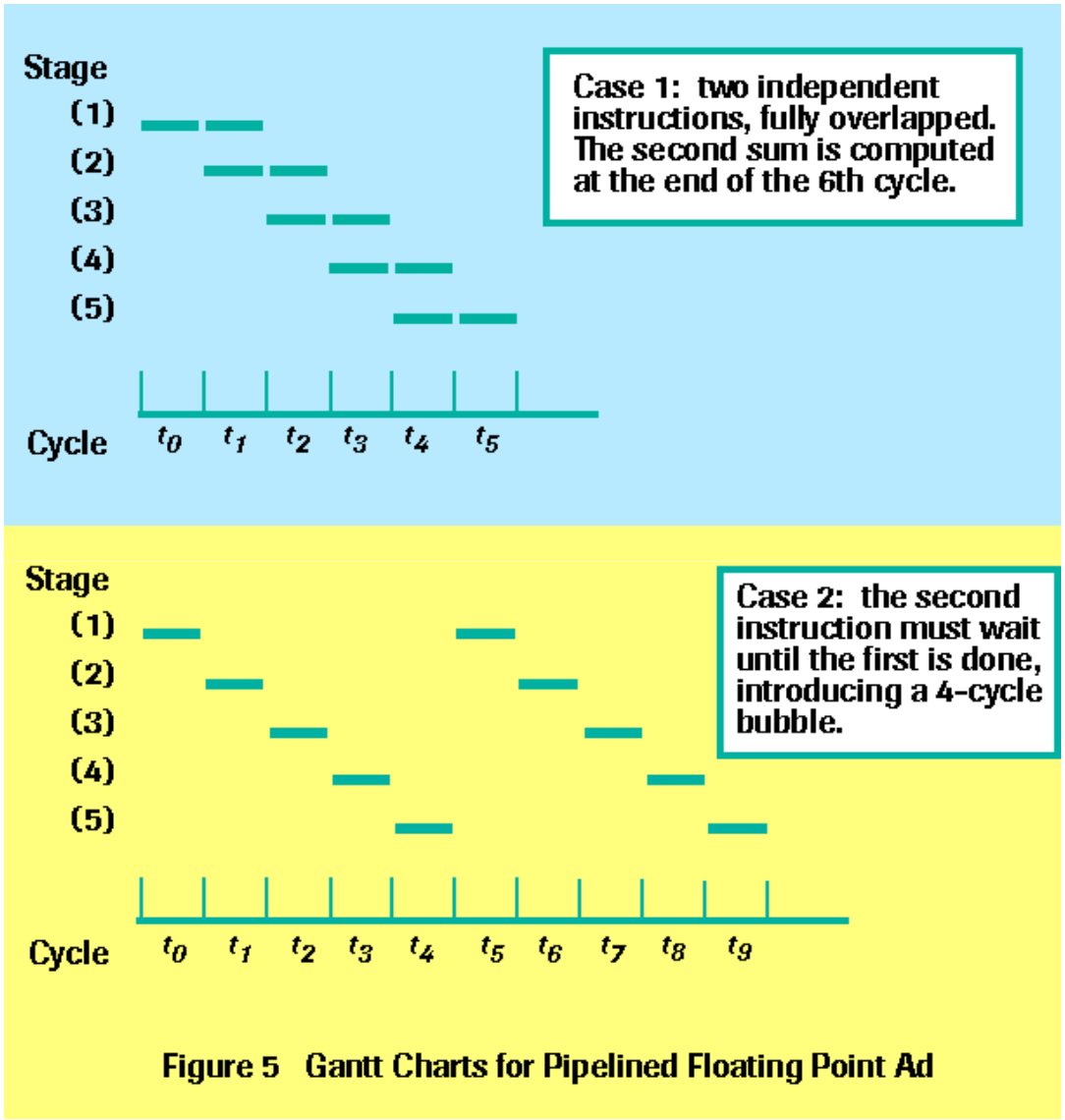


Figure 5 Gantt Charts for Pipelined Floating Point Ad

Speedup for Pipelined Adder

A pipeline of depth d produces n items in $n + d$ steps in the absence of bubbles.

Without a pipeline, $n \cdot d$ cycles would be required to process the same data.

$$\text{Speedup} = \frac{n \cdot d}{n + d} \approx d \qquad \text{(for large } n\text{)}$$

Example: Addition of two 1000-component vectors for 5-stage pipelined adder on machine for which an addition takes 5 machine cycles:

$$\text{Speedup} = \frac{1000 \times 5}{1000 + 5} = 4.97$$

Instruction pipelines are used to speed the fetch-decode-execute cycle. The pipeline is constantly exploiting locality of reference by "looking ahead" and fetching instructions it thinks the processor will soon need. If a branch or loop instruction in the program invalidates this look-ahead, a bubble appears in the instruction pipeline while the fetch stage goes to look for the new instructions. This is called a ***control dependency.***

Memory Organization

In high-performance computing, it is important to match the (generally slower) memory accesses as well as possible with the (generally faster) processor cycling. This is particularly true for pipelined units that derive their efficiency from a constant supply of fresh operands for the pipeline. The primary difficulty is the *memory cycle time*, during which the memory is not accessible by subsequent operations.

For parallel systems there are two general memory designs:

- ***Shared Memory Systems*** for which there is one large virtual memory that all processors have equivalent access to.
- ***Distributed Memory Systems*** for which each processor has its own *local memory* not directly accessible from other processors.

Interconnect Topologies for Parallel Systems

A major consideration for parallel systems is the manner in which the processors, memories, and switches communicate with each other. The connections among these define the *topology* for the machine.

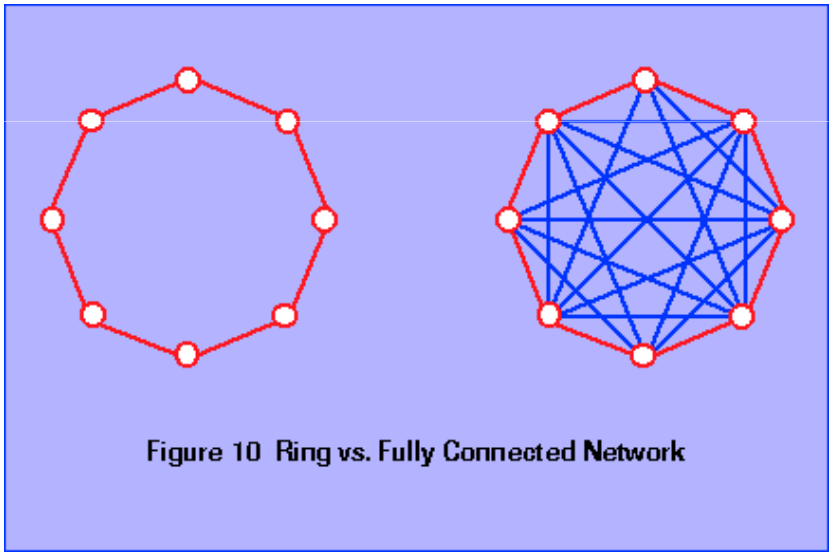
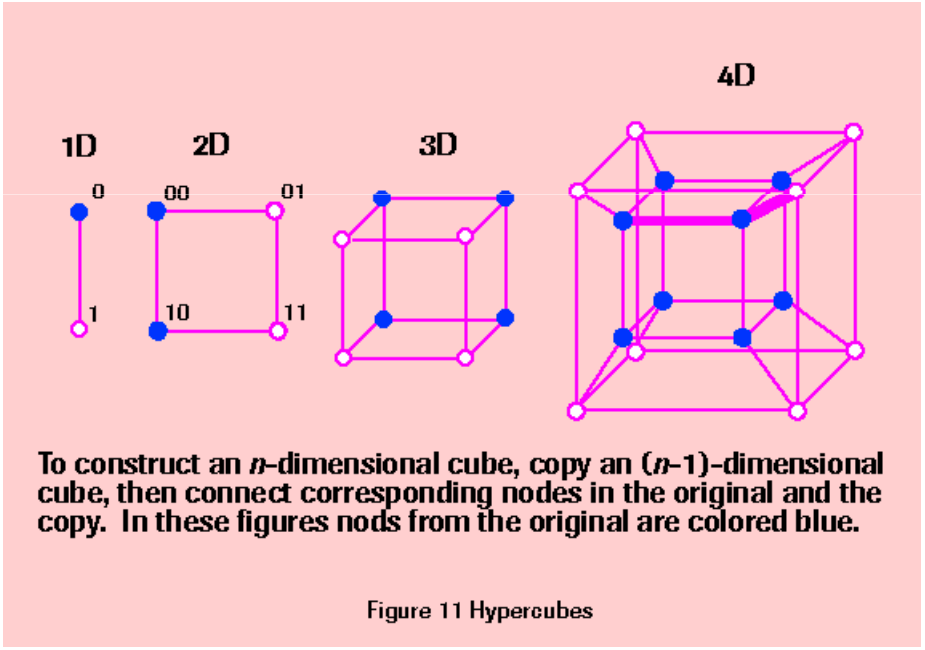


Figure 10 Ring vs. Fully Connected Network

Ring vs. Fully Connected Network



To construct an *n*-dimensional cube, copy an (*n*–1)-dimensional cube, then connect corresponding nodes in the original and the copy. In these figures nods from the original are colored blue.

Figure 11 Hypercubes

Hipercubes

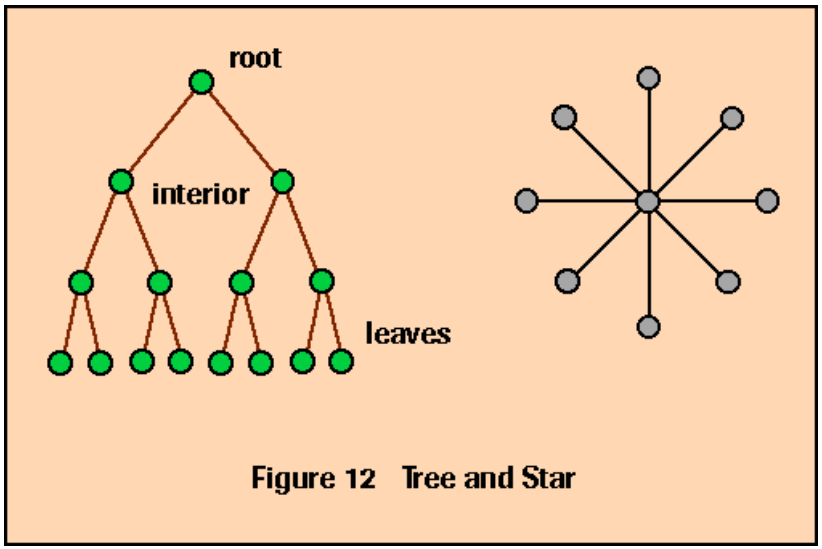


Figure 12 Tree and Star

Tree and Star Topologies

Mesh Topologies

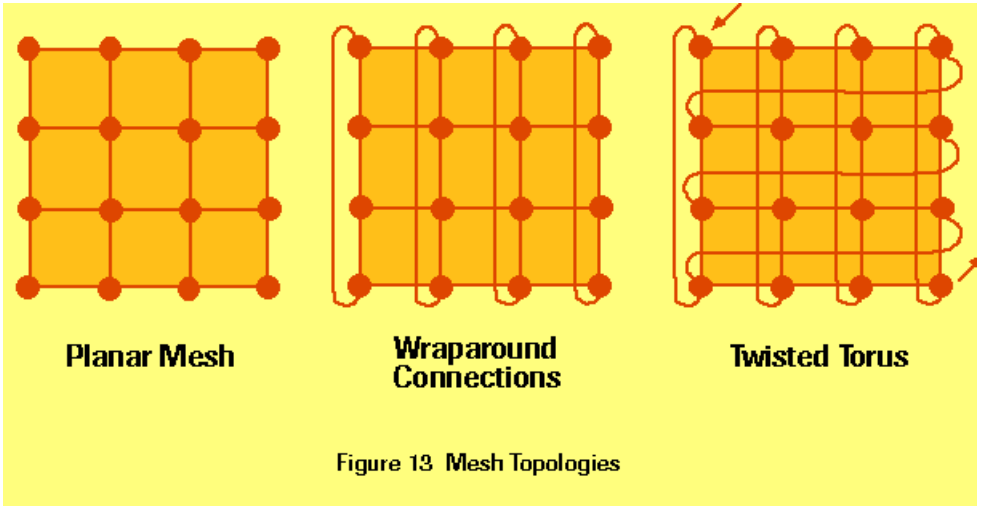


Figure 13 Mesh Topologies

Basic Types of Parallel Architectures

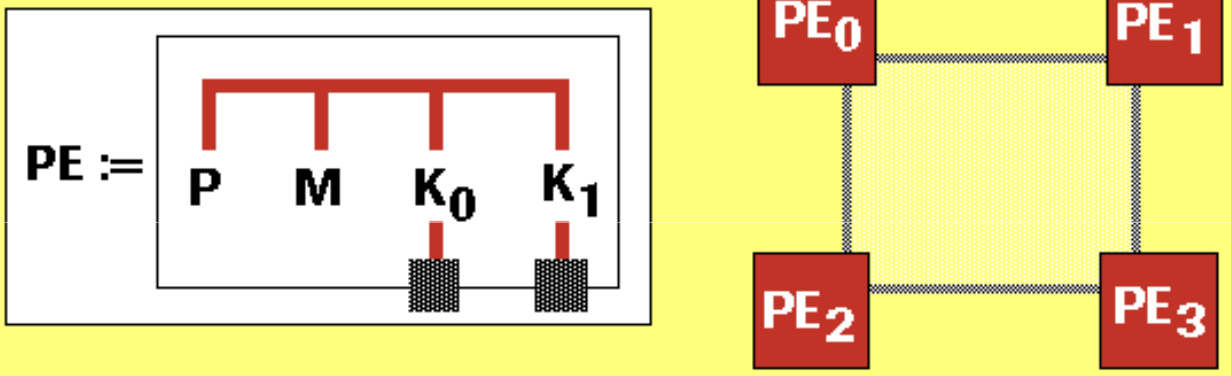
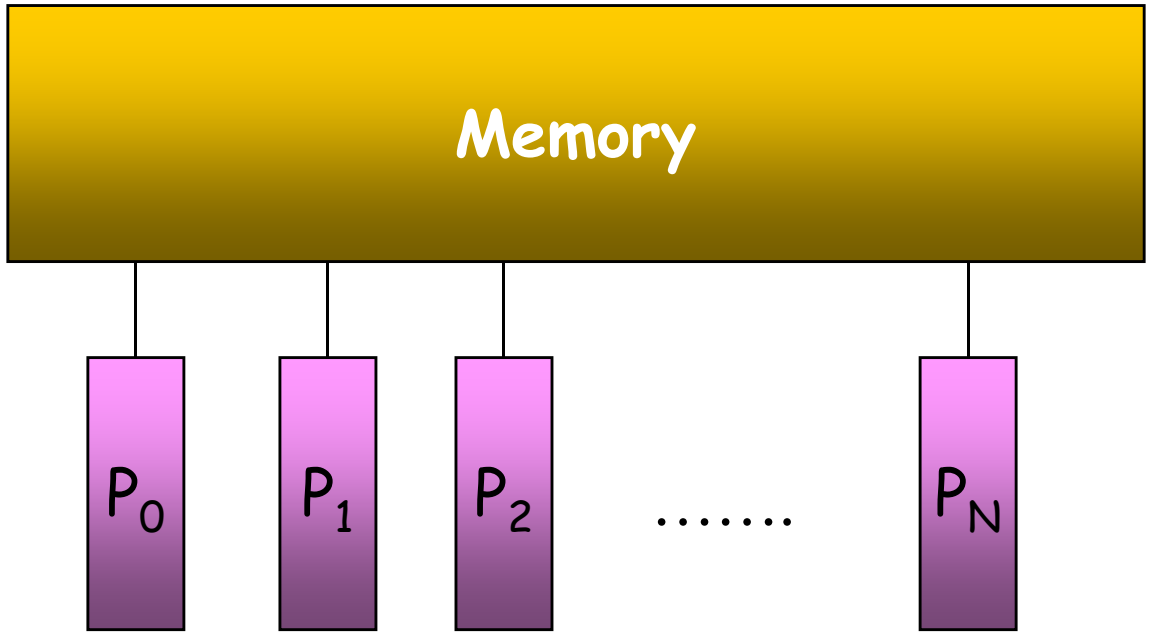


Figure 9 Distributed Memory Parallel Processor

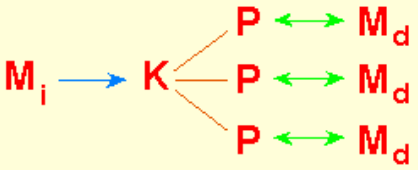
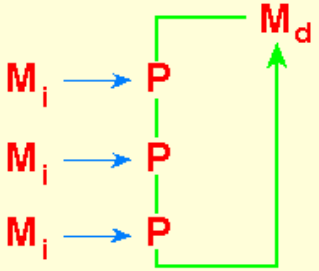
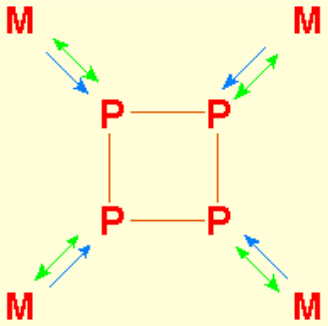
http://csep1.phy.ornl.gov/csep.html

Basic Types of Parallel Architectures



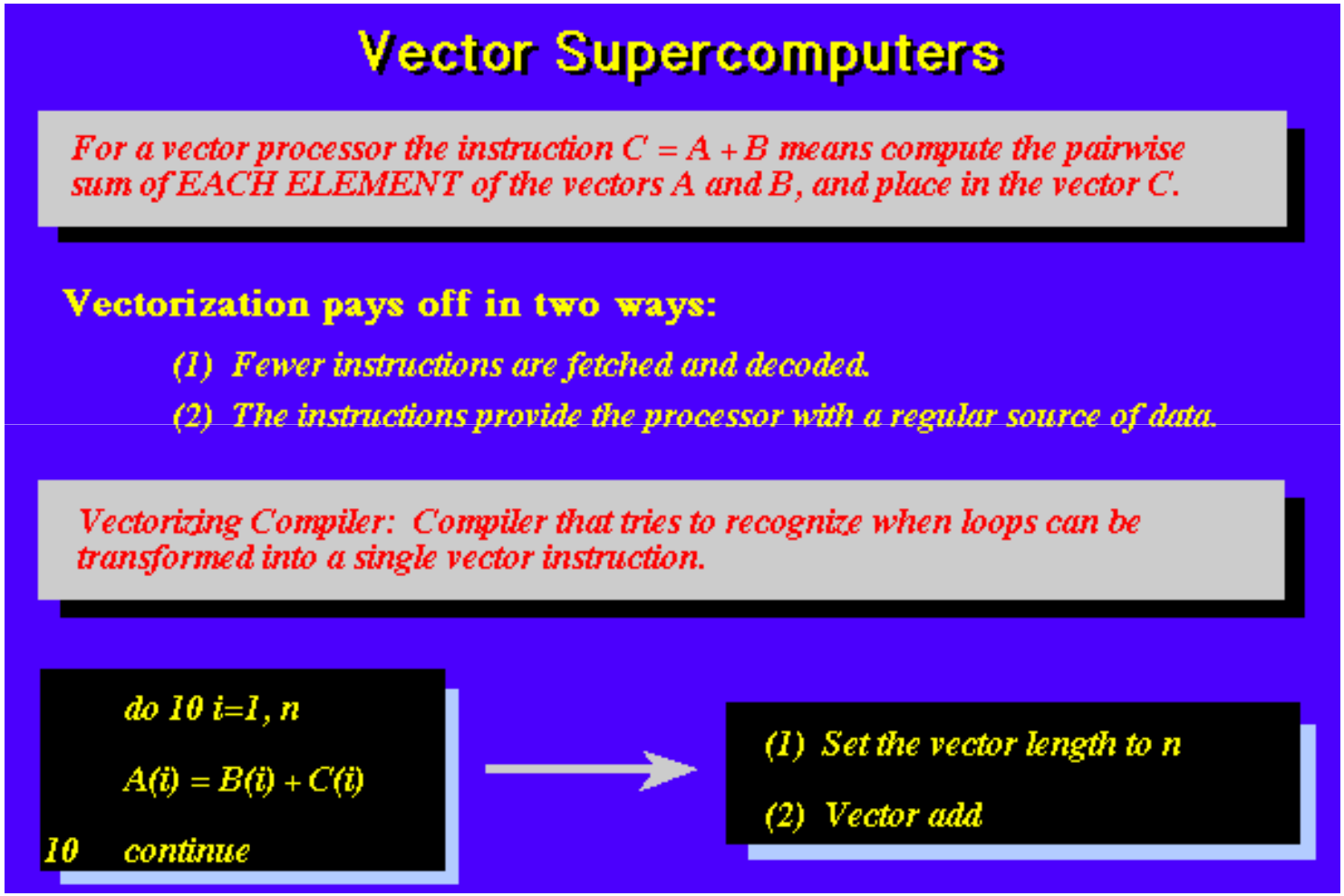
Shared memory Machine

Flinn's Taxonomy of Parallel Architectures

	Single Data Stream	*Multiple Data Streams*
Single Instruction Stream	 *SISD: A single processor fetches instructions and performs all data processing operations*	 *SIMD: One instruction processor K fetches instructions, broadcasts orders to processing elements P. Typically each P has its own data memory.*
Multiple Instruction Streams	 *MISD: A single data stream is operated on by several processors, each with an instruction stream from its own instruction memory.*	 *MIMD: Processors independently fetch instructions and operate on data. Processors communicate directly (as shown), or through shared memory.*

http://csep1.phy.ornl.gov/csep.html

Vector Supercomputers



Cluster - Beowulf Systems

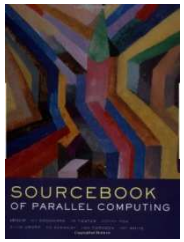
Work on this type of distributed parallel commodity-off-the-shelf system was initiated at NASA Goddard in 1993 using Intel 486 processors. These systems are also being investigated at several of the other National Laboratories and major Universities. Some of the main attributes are:

- Promising Price-Performance for small and moderate scale parallel processing.
- Easily upgraded to new PC technology and different PC vendors.
- Operating System Software in place (e.g. Linux). Now has TCP/IP, NFS, NIS, Gnu compilers (C, C++, F77).
- Message Passing Libraries such as MPI.

See: T.L. Sterling, J. Salmon, D.J. Becker, D. Savarese, How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters, MIT Press, 1999.

More Reading Material

- **Barney's Online Tutorial on Introduction to Parallel Computing**
  - https://computing.llnl.gov/tutorials/parallel_comp/
- **Jim Demmel's Applications of Parallel Computing course at Berkeley, USA**
  - http://www.cs.berkeley.edu/~demmel/cs267_Spr10/
- **Jack Dongarra, Ian Foster, Geoffrey C. Fox, William Gropp, Ken Kennedy, Linda Torczon, Andy White, The Sourcebook of Parallel Computing, 2002**

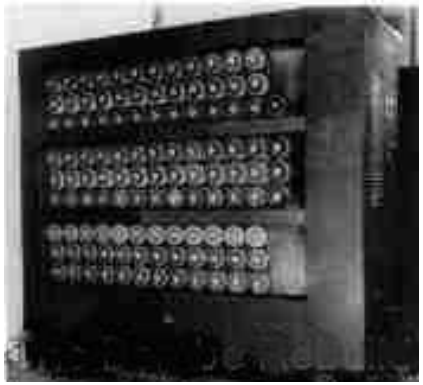


HIGH PERFORMANCE COMPUTERS

High Performance Computers or Supercomputers

Supercomputers are the fastest and most powerful general purpose scientific computing systems available at any given time.

Dongarra et al, “Numerical Linear Algebra for High-Performance Computers”, SIAM, 1998



Turing´s Bombe, UK, 1941

CRAY
THE SUPERCOMPUTER COMPANY

Cray XT5 at Oak Ridge, USA, 2009
2.3 Petaflops, 224K AMD cores

The TOP500 List
www.top500.org

- The main objective of TOP500 is to provide a ranked list of general purpose systems that are in common use for high end applications
- It is based on LINPACK Benchmark, that solves a dense system of linear equations by LU factorization
- A parallel implementation of the LINPACK benchmark and instructions on how to run it can be found at http://www.netlib.org/benchmark/hpl/
- TOP500 uses the benchmark version that allows the user to scale the size of the problem and to optimize the software in order to achieve the best performance for a given machine

The LINPACK Benchmark

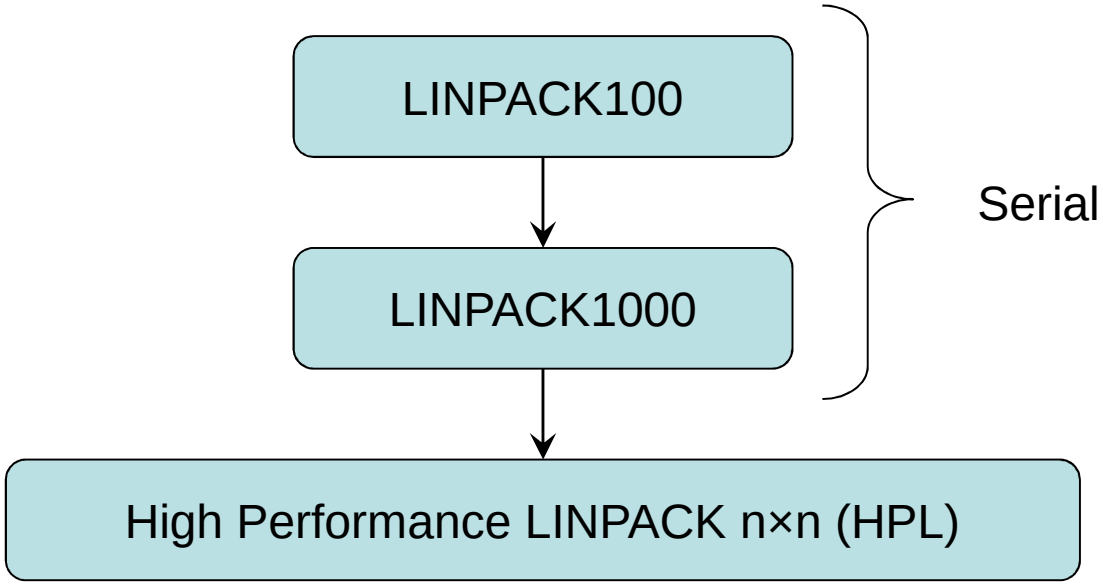
- **Direct solver for dense linear systems based on LU factorization**

- **Initially used as a tool to predict execution time only. Original report listed execution times for a matrix of order 100 in 23 computers (LINPACK100)**

$$tempo_n = \frac{tempo_{100} \cdot n^3}{100^3}$$

- **Today lists more than 1300 systems**
- **Web: http://www.netlib.org/benchmark/hpl**
- **See:** Jack J. Dongarra, Piotr Luszczek and Antoine Petitet, The LINPACK Benchmark: past, present and future, *Concurrency and computation: practice and experience.* 2003; 15:803–820

LINPACK's Evolution



Other packages for dense linear algebra (www.netlib.org):

LAPACK and ScaLAPACK

LINPACK's Building Blocks

- **BLAS (Basic Linear Algebra Subprograms): http://www.netlib.org/blas**
  - Level 1: vector-vector operations; $y = y + ax$
  - Level 2: matrix-vector operations; $y = y + Ax$
  - Level 3: matrix-matrix operations; $A=B+C$

- **Highly optimzed BLAS:**
  - ATLAS (free optmized BLAS generator); http://www.netlib.org/atlas
  - Intel's MKL
  - GOTO BLAS
    - K. Goto, R. A. Van de Geijn, Anatomy of High-Performance Matrix Multiplication, ACM Transactions on Mathematical Software, 34(3): 2008
    - K. Goto, R. A. Van de Geijn, High-performance implementation of the level-3 BLAS, ACM Transactions on Mathematical Software, 35(1): 2008
    - http://www.tacc.utexas.edu/?id=402

HPL Benchmark Highlights

- **Extract MAXIMUM sustained performance of a given system**
- **Results listed in TOP500:**
  - R_{max} = Performance in Gflop/s for the biggest problem ran
  - N_{max} = Size of biggest problem
  - $N_{1/2}$ = Size of problem where half of R_{max} is sustained
  - R_{peak} = Theoretical peak performance
- **How to Run?**
  - HowTo – HPL Over Intel MPI
    - http://software.intel.com/en-us/articles/running-the-hpl-benchmark-over-intel-mpi/
- **Factors affecting HPL performance**
  - Implementation; human effort; operational system; hardware, network, compiler, BLAS, etc.

The TOP500 List

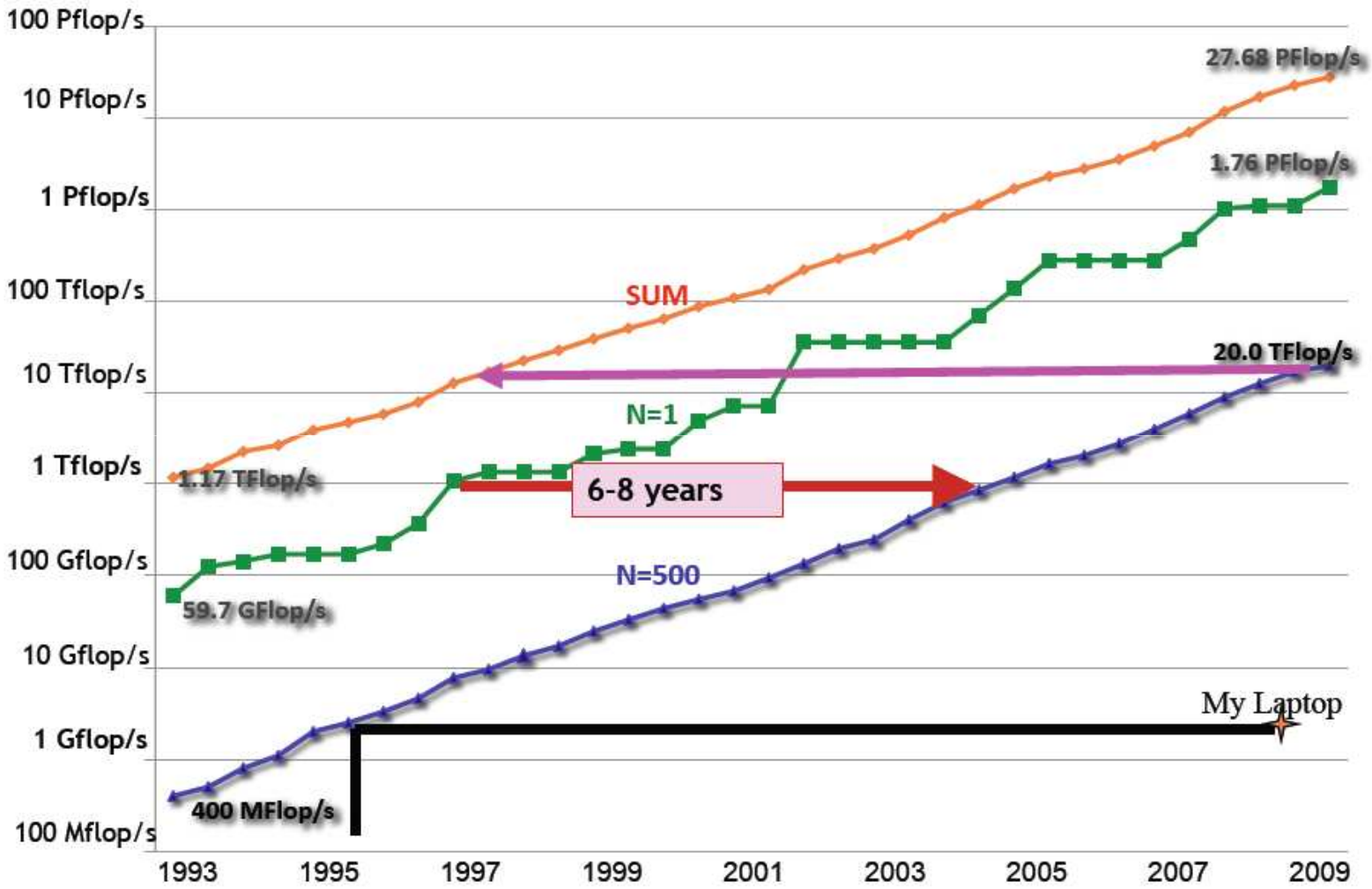
TOP 10 Sites for November 2009

For more information about the sites and systems in the list, click on the links or view the complete list.

Rank	Site	Computer
1	Oak Ridge National Laboratory United States	Jaguar - Cray XT5-HE Opteron Six Core 2.6 GHz Cray Inc.
2	DOE/NNSA/LANL United States	Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband IBM
3	National Institute for Computational Sciences/University of Tennessee United States	Kraken XT5 - Cray XT5-HE Opteron Six Core 2.6 GHz Cray Inc.
4	Forschungszentrum Juelich (FZJ) Germany	JUGENE - Blue Gene/P Solution IBM
5	National SuperComputer Center in Tianjin/NUDT China	Tianhe-1 - NUDT TH-1 Cluster, Xeon E5540/E5450, ATI Radeon HD 4870 2, Infiniband NUDT
6	NASA/Ames Research Center/NAS United States	Pleiades - SGI Altix ICE 8200EX, Xeon QC 3.0 GHz/Nehalem EP 2.93 Ghz SGI
7	DOE/NNSA/LLNL United States	BlueGene/L - eServer Blue Gene Solution IBM
8	Argonne National Laboratory United States	Blue Gene/P Solution IBM
9	Texas Advanced Computing Center/Univ. of Texas United States	Ranger - SunBlade x6420, Opteron QC 2.3 Ghz, Infiniband Sun Microsystems
10	Sandia National Laboratories / National Renewable Energy Laboratory United States	Red Sky - Sun Blade x6275, Xeon X55xx 2.93 Ghz, Infiniband Sun Microsystems

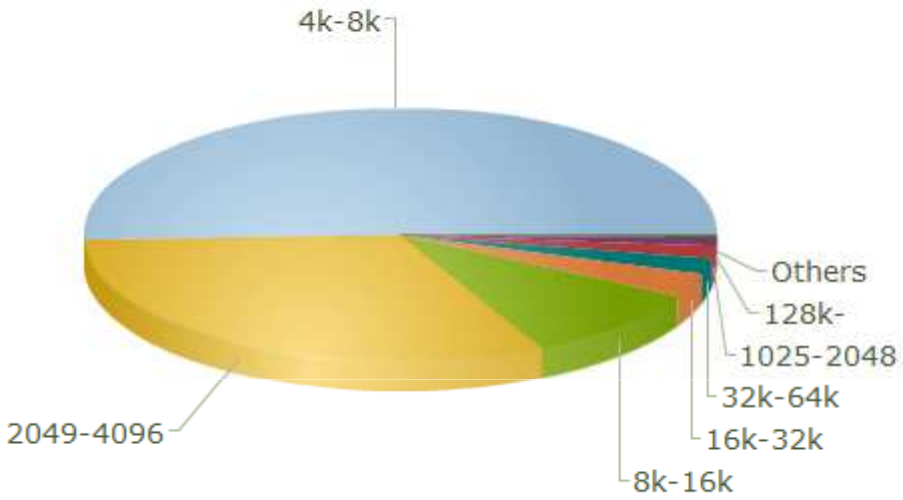
http://www.top500.org
Lists the top 500 supercomputers
Updated in 06/XX and 11/XX

Performance Development

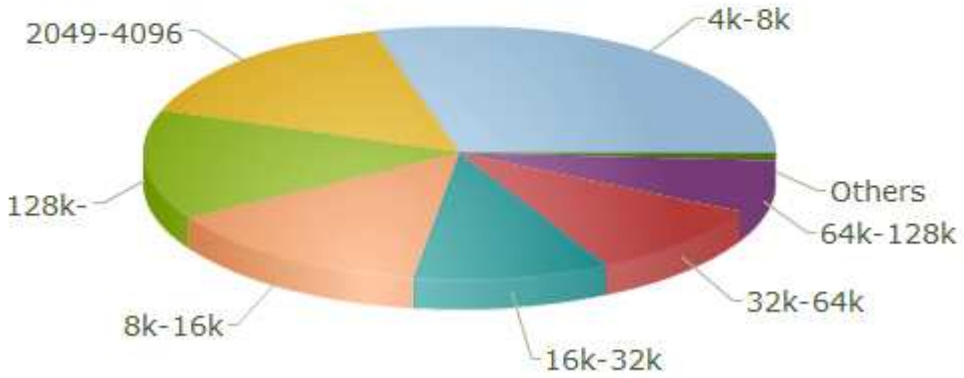


source: J. J. Dongarra, http://www.netlib.org/utk/people/JackDongarra/talks.htm

Number of Processors / Systems
November 2009



Number of Processors / Performance
November 2009

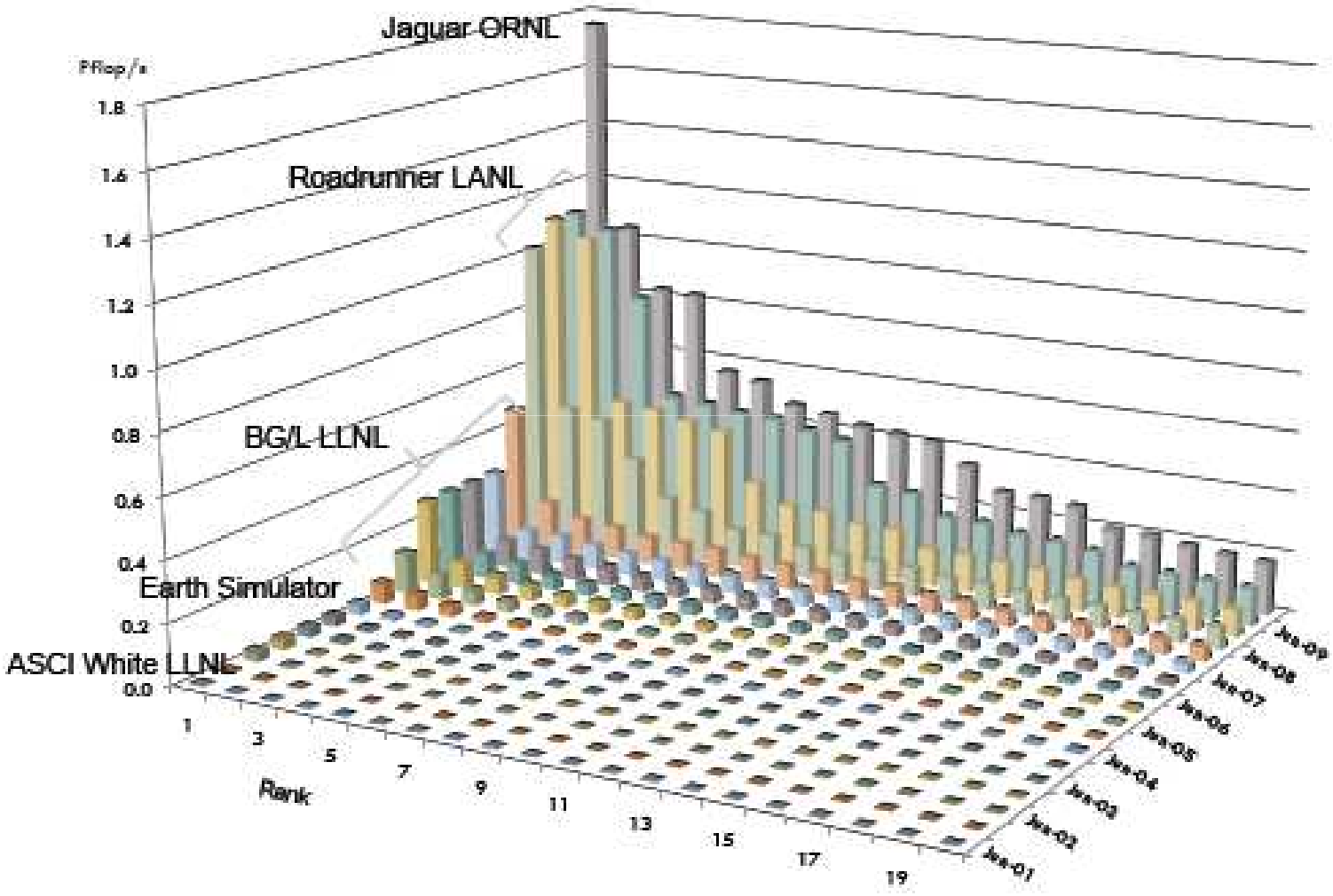


Processor Family share for 11/2009

In addition to the table below, you can view the visual charts using the TOP500 charts page. A direct link to the charts is also available.

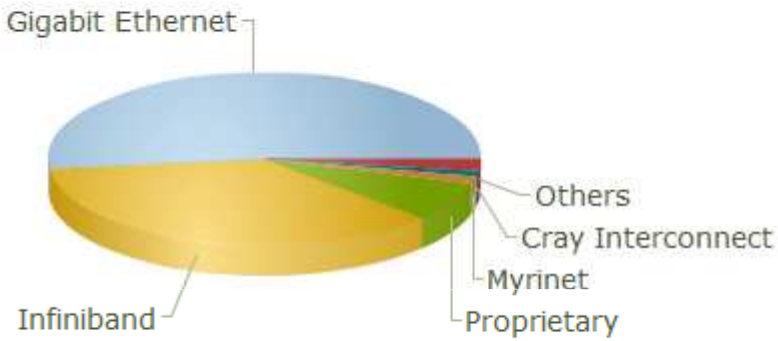
Processor Family	Count	Share %	Rmax Sum (GF)	Rpeak Sum (GF)	Processor Sum
Power	52	10.40 %	6020408	7574901	1470752
NEC	1	0.20 %	122400	131072	1280
Sparc	2	0.40 %	139110	152247	15104
Intel IA-64	6	1.20 %	289948	343346	54512
Intel EM64T	396	79.20 %	15173465	24492101	2217612
AMD x86_64	42	8.40 %	6210211	8171974	897175
Others	1	0.20 %	21960	84480	8192
Totals	**500**	**100%**	**27977501.79**	**40950122.01**	**4664627**

Performance of TOP20 over time

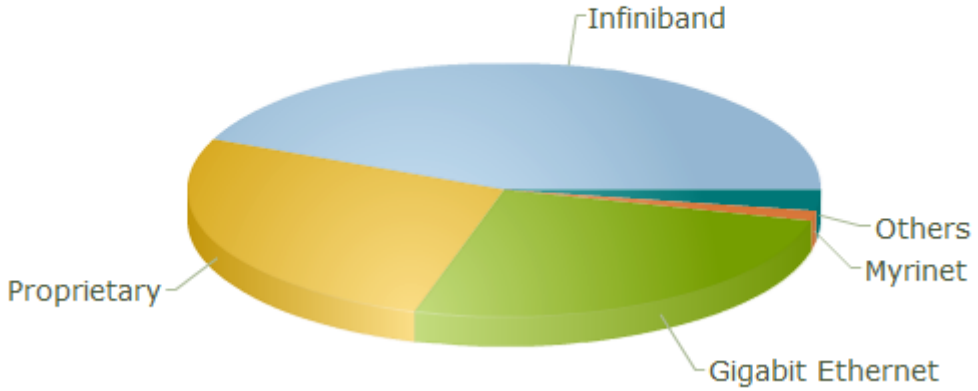


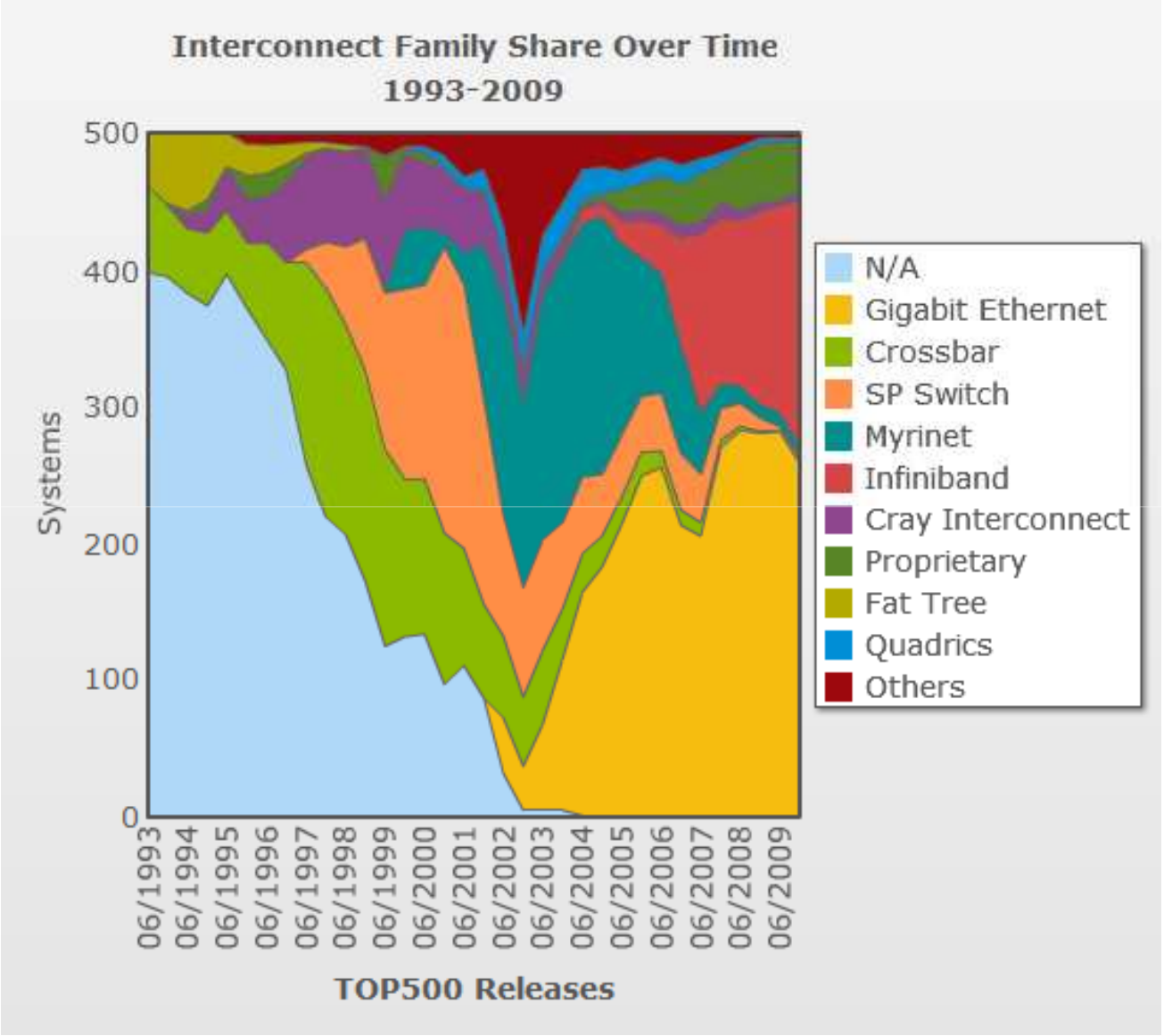
source: J. J. Dongarra, http://www.netlib.org/utk/people/JackDongarra/talks.htm

Interconnect Family / Systems
November 2009

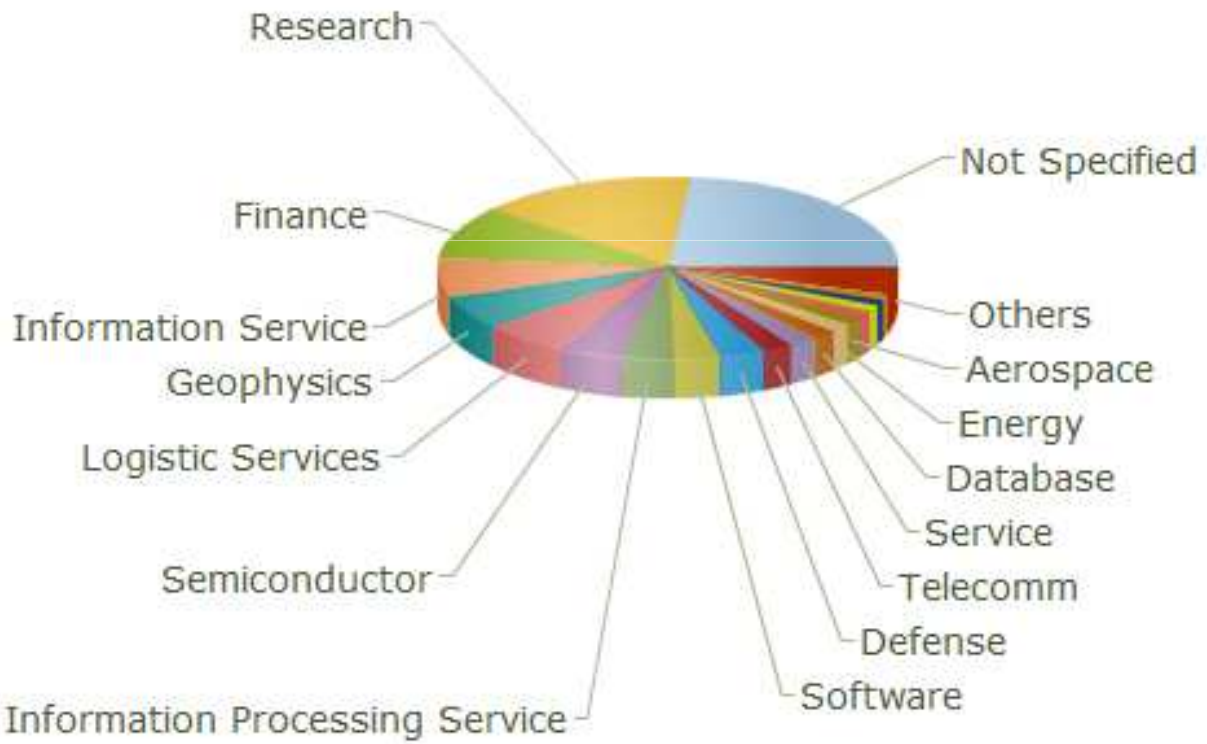


Interconnect Family / Performance
November 2009

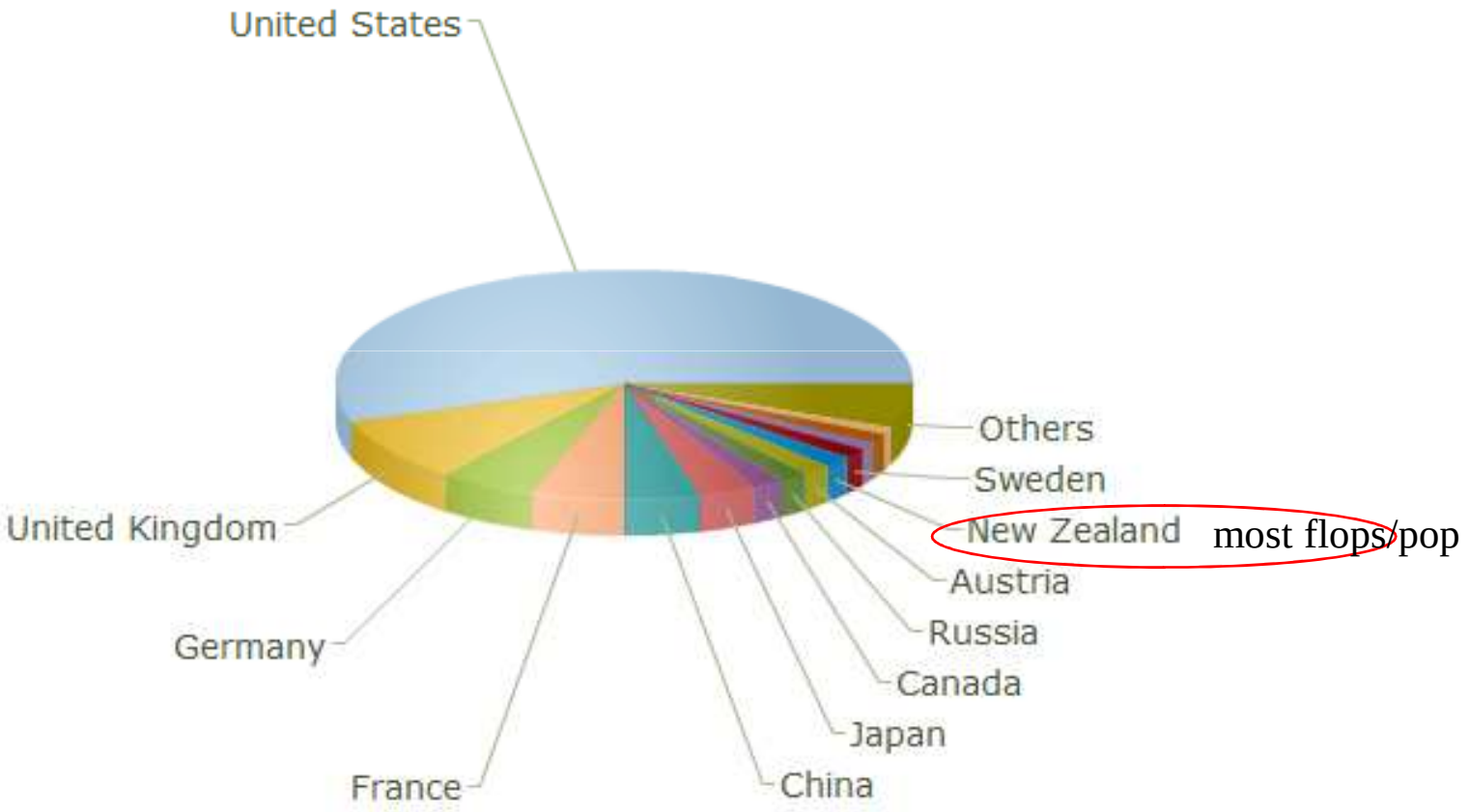



Interconnect Family Share Over Time
1993-2009
Systems
500
400
300
200
100
0
06/1993
06/1994
06/1995
06/1996
06/1997
06/1998
06/1999
06/2000
06/2001
06/2002
06/2003
06/2004
06/2005
06/2006
06/2007
06/2008
06/2009
TOP500 Releases
N/A
Gigabit Ethernet
Crossbar
SP Switch
Myrinet
Infiniband
Cray Interconnect
Proprietary
Fat Tree
Quadrics
Others

Application Area / Systems
November 2009



Countries / Systems
November 2009



most flops/pop

Brazil & Spain in TOP500/Nov 2009

TOP500 Sublist Generator

R_{max} and **R_{peak}** values are in TFlops. For more details about other fields, check the TOP500 description.

1 entries found.

Rank	Site	System	Cores	R_{max}	R_{peak}
76	NACAD/COPPE/UFRJ Brazil	Sun Blade x6048, Xeon X5560 2.8 Ghz, Infiniband QDR Sun Microsystems	6464	64.63	72.4

TOP500 Sublist Generator

R_{max} and **R_{peak}** values are in TFlops. For more details about other fields, check the TOP500 description.

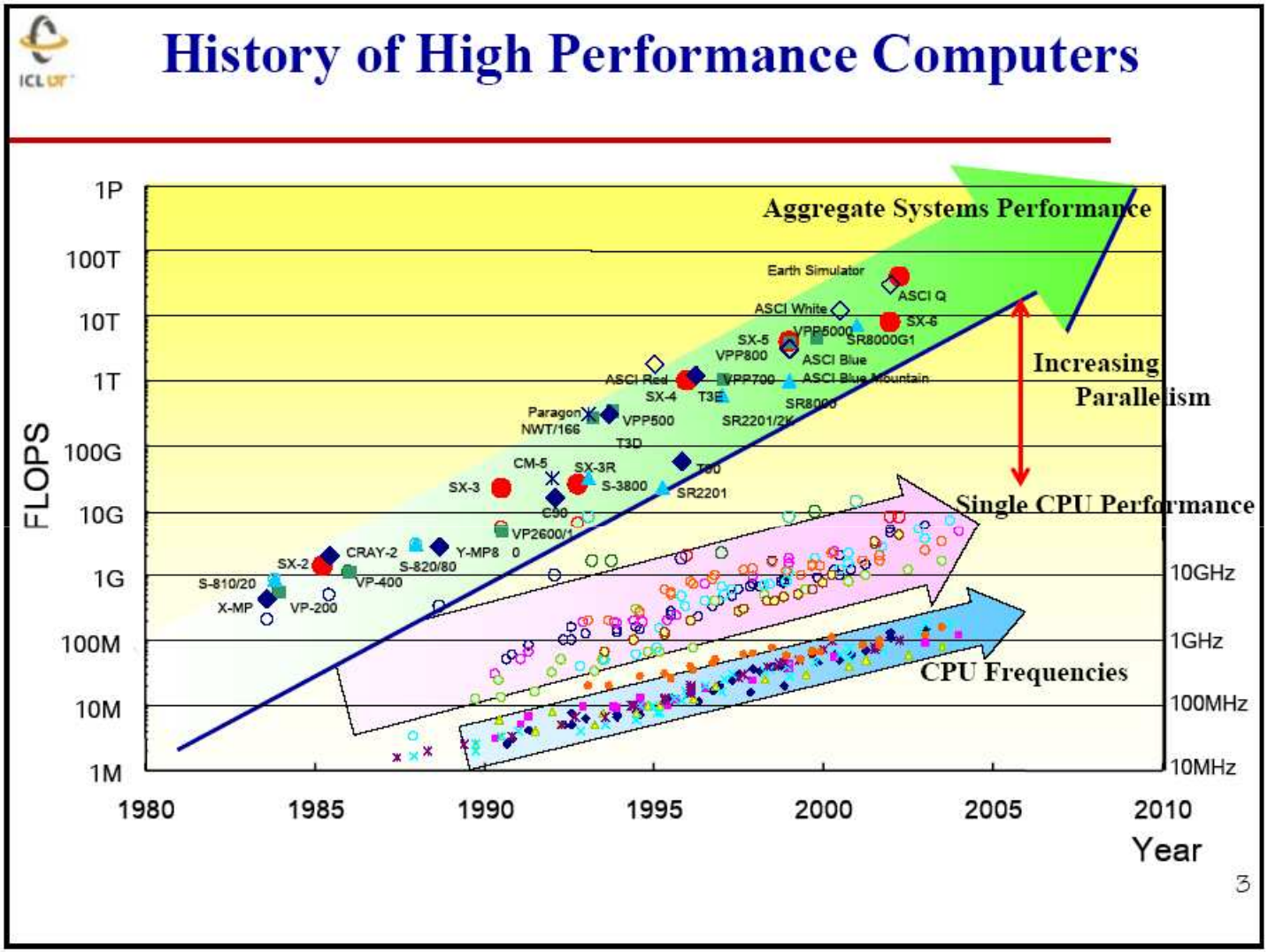
6 entries found.

Rank	Site	System	Cores	R_{max}	R_{peak}
77	Barcelona Supercomputing Center Spain	BladeCenter JS21 Cluster, PPC 970, 2.3 GHz, Myrinet IBM	10240	63.83	94.21
135	IT Service Provider Spain	Cluster Platform 3000 BL2x220, E54xx 3.0 Ghz, Infiniband Hewlett-Packard	4864	38.29	48.64
151	Financial Institution Spain	Cluster Platform 3000 BL460c G1, Xeon L5410 2.33 GHz, GigE Hewlett-Packard	7392	36.39	68.89
392	Research Institution Spain	Cluster Platform 3000 BL460c, Xeon 54xx 2.66GHz, Infiniband Hewlett-Packard	2816	23.4	30.04
405	IT Service Provider Spain	Cluster Platform 3000 BL2x220, E54xx 3.0 Ghz, Infiniband Hewlett-Packard	2432	23.19	29.18
483	Financial Institution (S1) Spain	Cluster Platform 3000 BL460c, Xeon 53xx 1.86GHz, GigEthernet Hewlett-Packard	5248	20.79	39.05

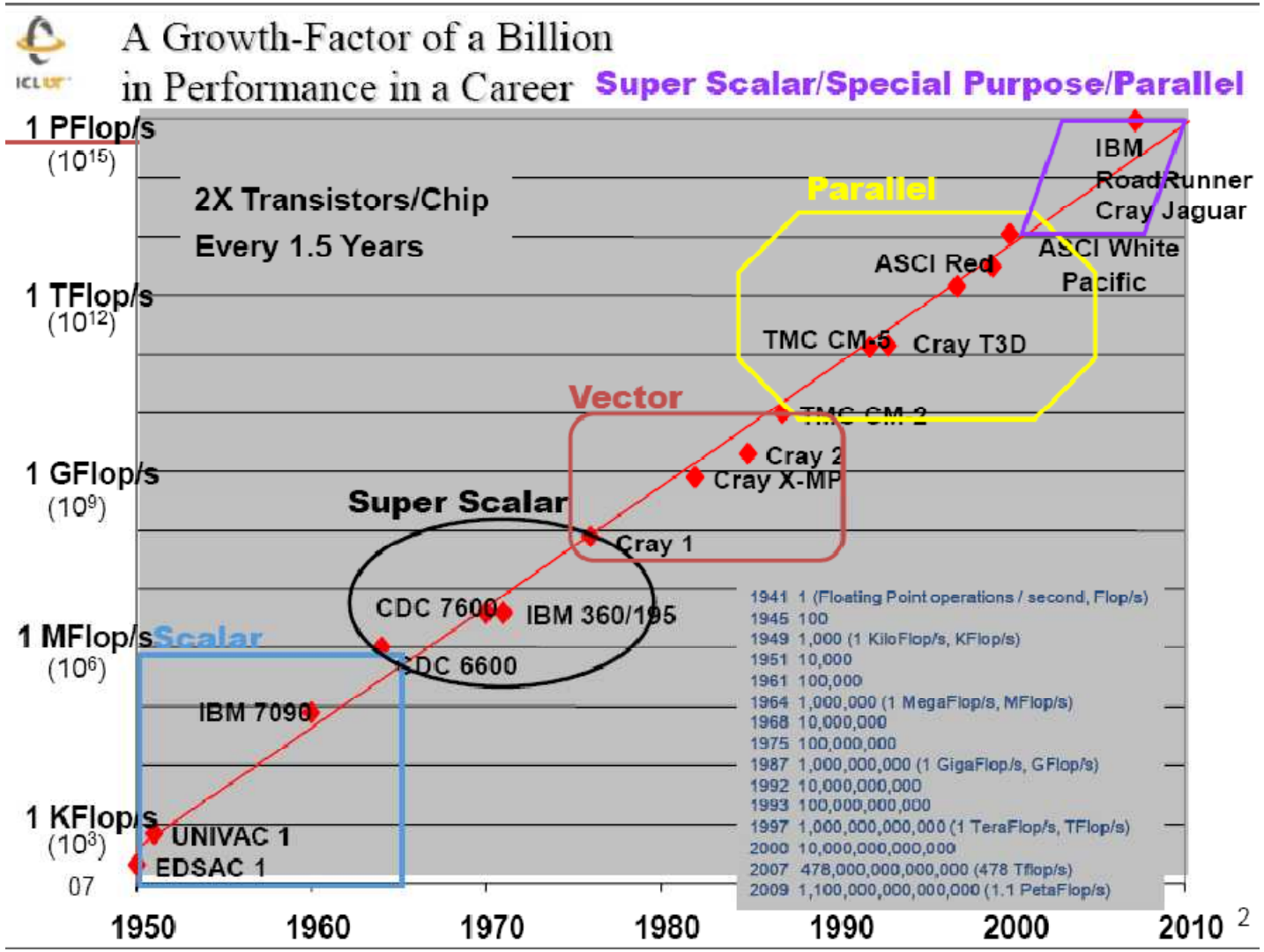
The Power Wall

THE GREEN 500

Green500 Rank	MFLOPS/W	Site*	Computer*	Total Power (kW)	TOP500 Rank*
1	722.98	Forschungszentrum Juelich (FZJ)	QPACE SFB TR Cluster, PowerXCell 8i, 3.2 GHz, 3D-Torus	59.49	110
1	722.98	Universitaet Regensburg	QPACE SFB TR Cluster, PowerXCell 8i, 3.2 GHz, 3D-Torus	59.49	111
1	722.98	Universitaet Wuppertal	QPACE SFB TR Cluster, PowerXCell 8i, 3.2 GHz, 3D-Torus	59.49	112
4	458.33	DOE/NNSA/LANL	BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Infiniband	276	29
4	458.33	IBM Poughkeepsie Benchmarking Center	BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Infiniband	138	78
6	444.25	DOE/NNSA/LANL	BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband	2345.5	2
7	428.91	National Astronomical Observatory of Japan	GRAPE-DR accelerator Cluster, Infiniband	51.2	445
8	379.24	National SuperComputer Center in Tianjin/NUDT	NUDT TH-1 Cluster, Xeon E5540/E5450, ATI Radeon HD 4870 2, Infiniband	1484.8	5
9		#1 TOP500, Jaguar, 7MW, MFLOPS/W=251		504	18
9	378.77	EDF R&D	Blue Gene/P Solution	252	49
9	378.77	Ecole Polytechnique Federale de Lausanne	Blue Gene/P Solution	126	99

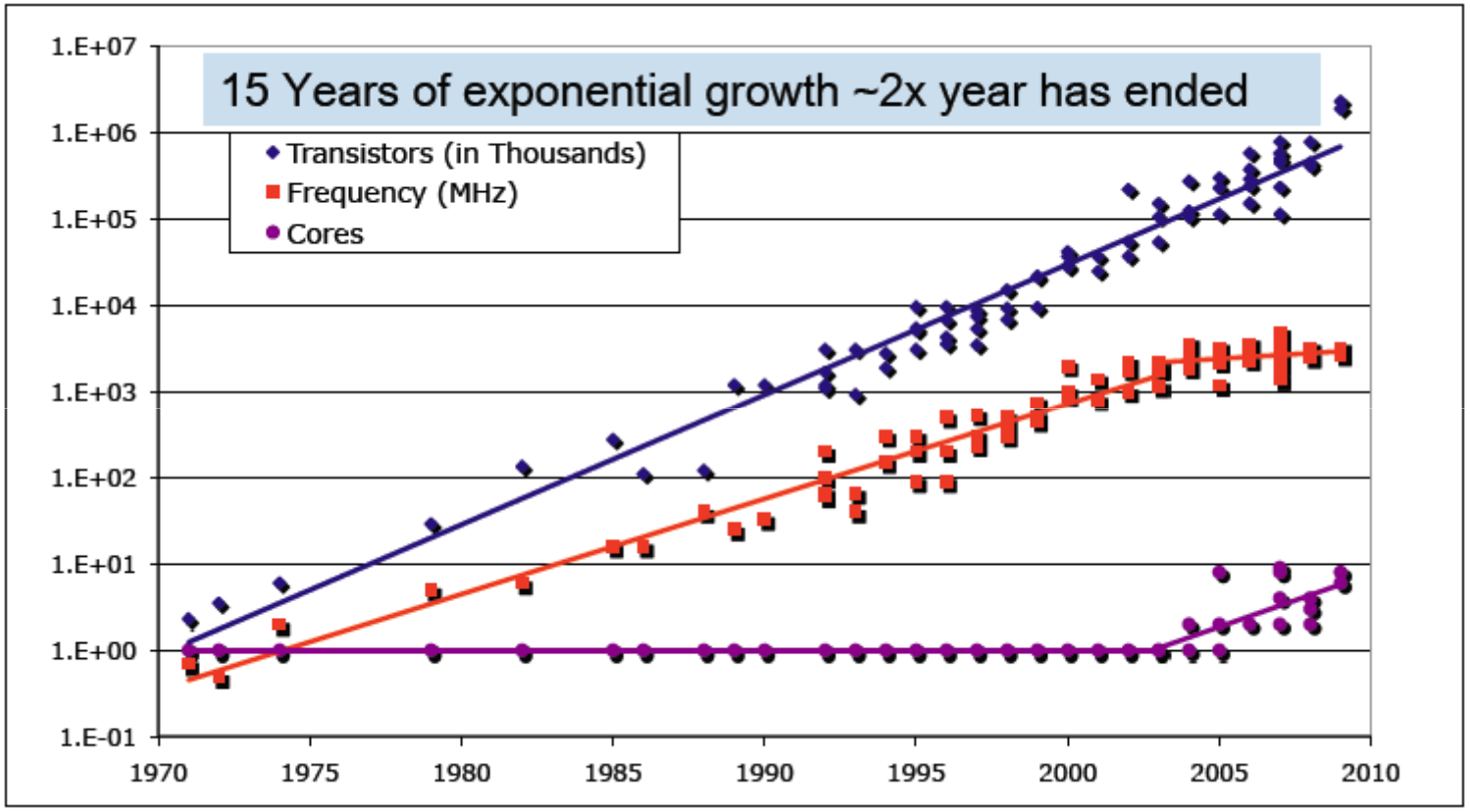


Fonte: J. Dongarra, http://www.netlib.org/utk/people/JackDongarra/talks.html



Fonte: J. Dongarra, http://www.netlib.org/utk/people/JackDongarra/talks.html

Moore's Law revisited

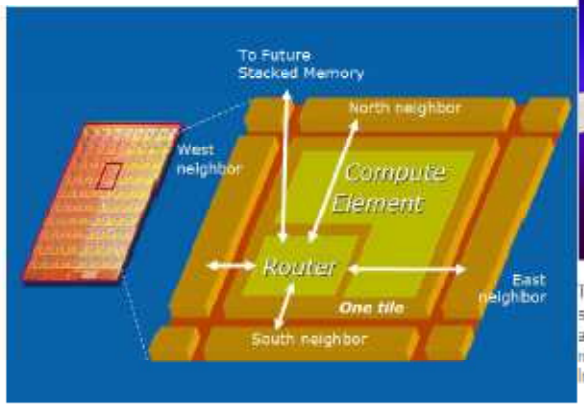


Clock frequency scaling replaced by scaling cores / chip

source: J. J. Dongarra, http://www.netlib.org/utk/people/JackDongarra/talks.htm

80 Core

- Intel's 80 Core chip
  - 1 Tflop/s
  - 62 Watts
  - 1.2 TB/s internal BW



The New York Times

Technology

WORLD | U.S. | N.Y. / REGION | BUSINESS | TECHNOLOGY | SCIENCE | HEALTH | SPORTS | OPINION

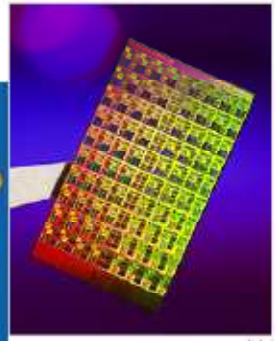
CAMCORDERS | CAMERAS | CELLPHONES | COMPUTERS | HANDHELDS | HOME VIDEO | MUSIC | PERIPHERALS

Intel Prototype May Herald a New Age of Processing

By JOHN MARKOFF
Published: February 12, 2007

E-MAIL
PRINT
REPRINTS
SAVE
SHARE

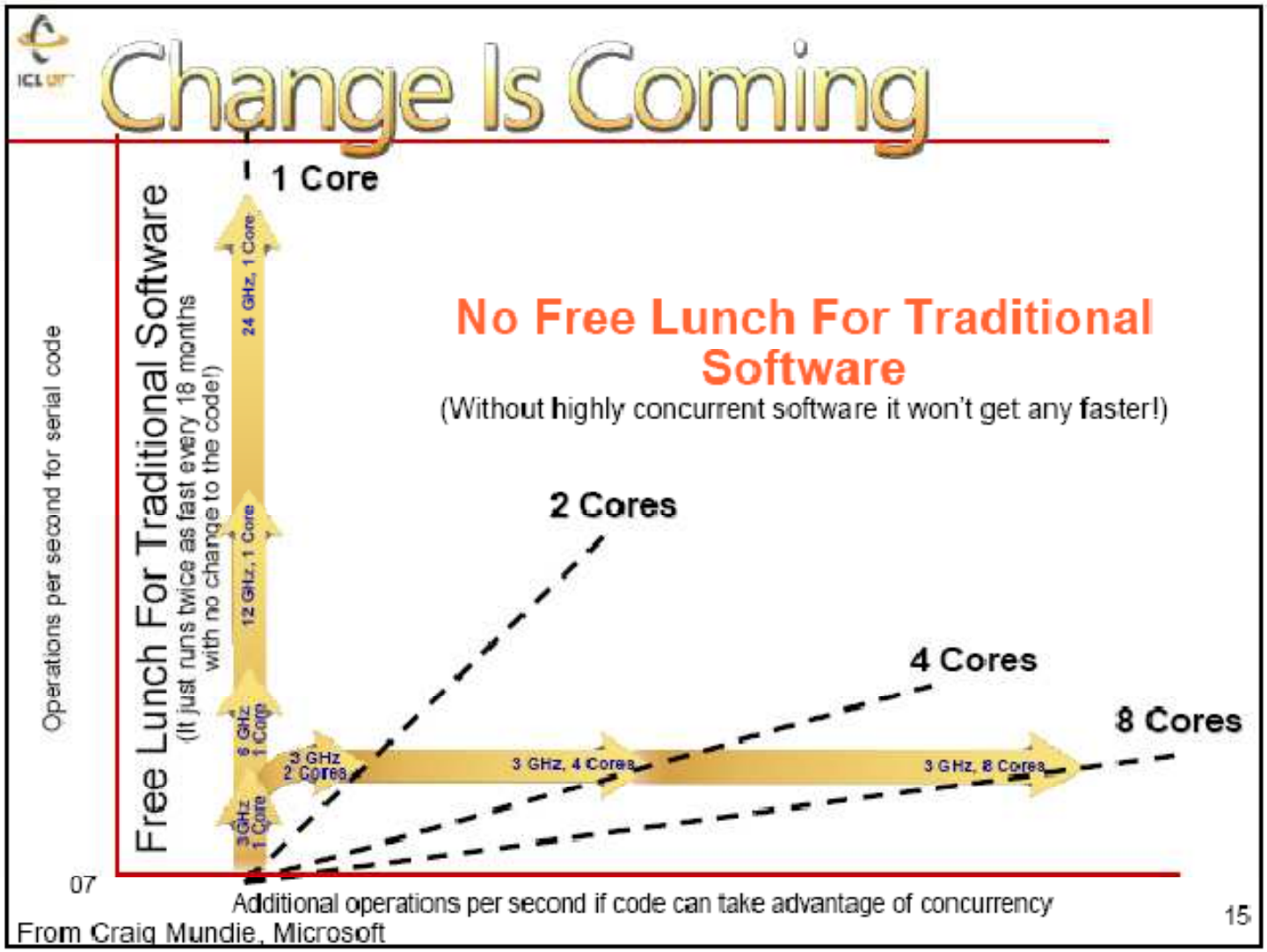
SAN FRANCISCO, Feb. 11 — Intel will demonstrate on Monday an experimental computer chip with 80 separate processing engines, or cores, that company executives say provides a model for commercial chips that will be used widely in standard desktop, laptop and server computers within five years.



The Teraflop Chip has 80 separate processing engines and takes advantage of manufacturing technology that Intel introduced last month.

The new processor, which the company first described as a Teraflop Chip at a conference last year, will be detailed in a technical paper to be presented on the opening day of the International Solid States Circuits Conference, beginning here on Monday.

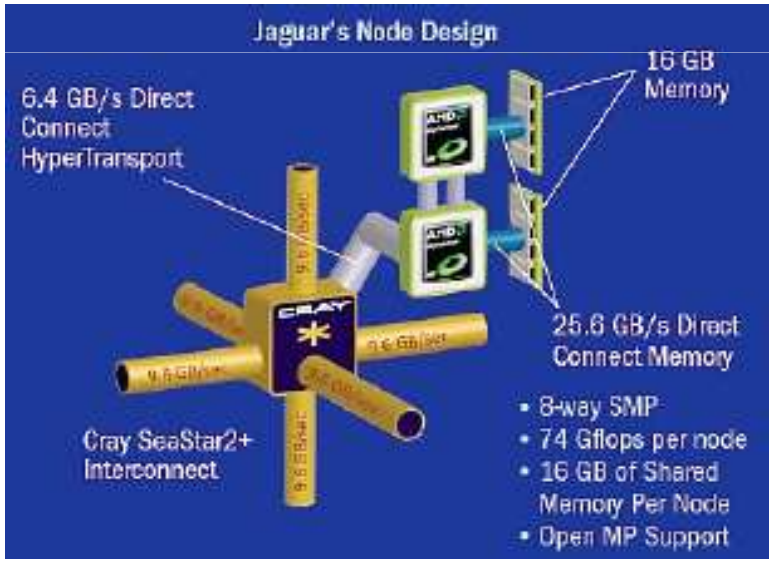
While the chip is not compatible with Intel's current chips, the company said it had already begun design work on a commercial version that would essentially have dozens or even hundreds of Intel-compatible microprocessors laid out in a tiled pattern on a single chip.



Fonte: J. Dongarra, http://www.netlib.org/utk/people/JackDongarra/talks.html

#1 TOP500 ORNL Cray XT5

Jaguar Specifications	XT5	XT4
Peak Teraflops	2,332	263
Six-Core AMD Opterons	37,376	
Quad-Core AMD Opterons		7,832
AMD Opteron Cores	224,256	31,328
Compute Nodes	18,688	7,832
Memory (TB)	299	62
Memory Bandwidth (GB/s)	478	100
Disk Space (TB)	10,000	750
Interconnect Bandwidth	374	157
Floor Space (ft^2)	4,352	1,344
Cooling Technology	Liquid	Air



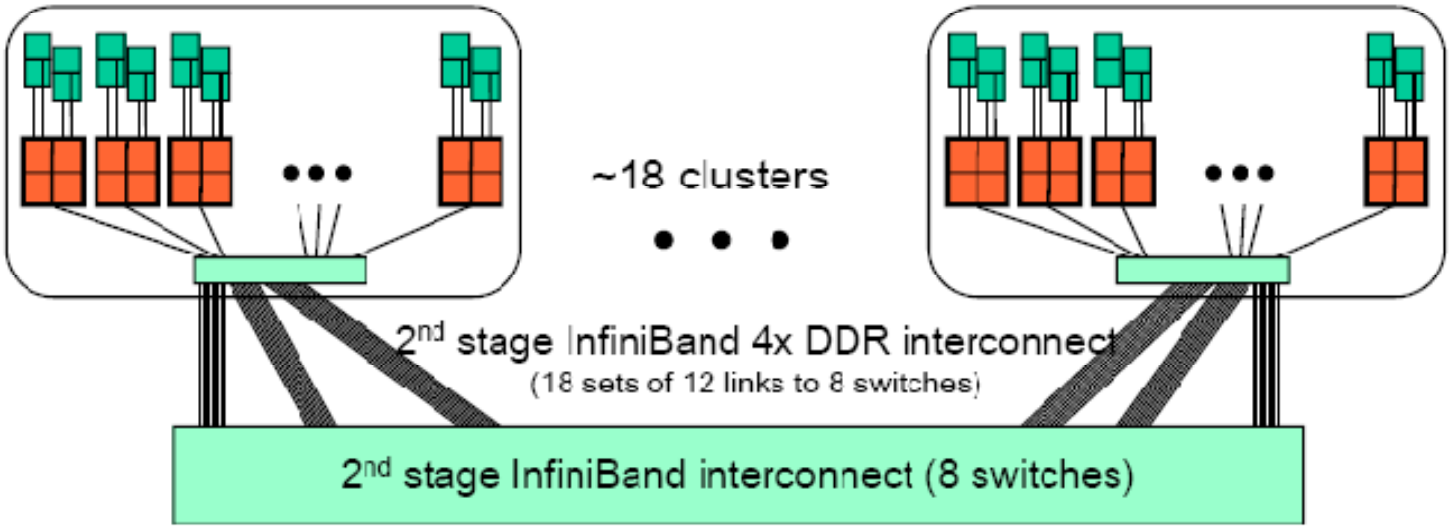
http://www.nccs.gov/jaguar/

Los Alamos Roadrunner
A Petascale System in 2008

"Connected Unit" cluster
192 Opteron nodes
(180 w/ 2 dual-Cell blades
connected w/ 4 PCIe x8 links)

≈ 13,000 Cell HPC chips
- ≈ 1.33 PetaFlop/s (from Cell)

≈ 7,000 dual-core Opterons



Based on the 100 Gflop/s (DP) Cell chip

Approval by DOE 12/07
First CU being built today
Expect a May Pflop/s run
Full system to LANL in December 2008

http://www.lanl.gov/roadrunner/

TACC Ranger

- **Compute power - 504 Teraflops**
  - 3,936 Sun four-socket blades
  - 15,744 AMD "Barcelona" processors
  - Quad-core, four flops/cycle (dual pipelines)
- **Memory - 123 Terabytes**
  - 2 GB/core, 32 GB/node
  - 132 GB/s aggregate bandwidth
- **Disk subsystem - 1.7 Petabytes**
  - 72 Sun x4500 "Thumper" I/O servers, 24TB each
  - 40 GB/sec total aggregate I/O bandwidth
  - 1 PB raw capacity in largest filesystem
  - Interconnect - 10 Gbps / 3.0 µsec latency
  - Sun InfiniBand-based switches (2), up to 3456 4x ports each
  - Full non-blocking 7-stage Clos fabric
  - Mellanox ConnectX InfiniBand

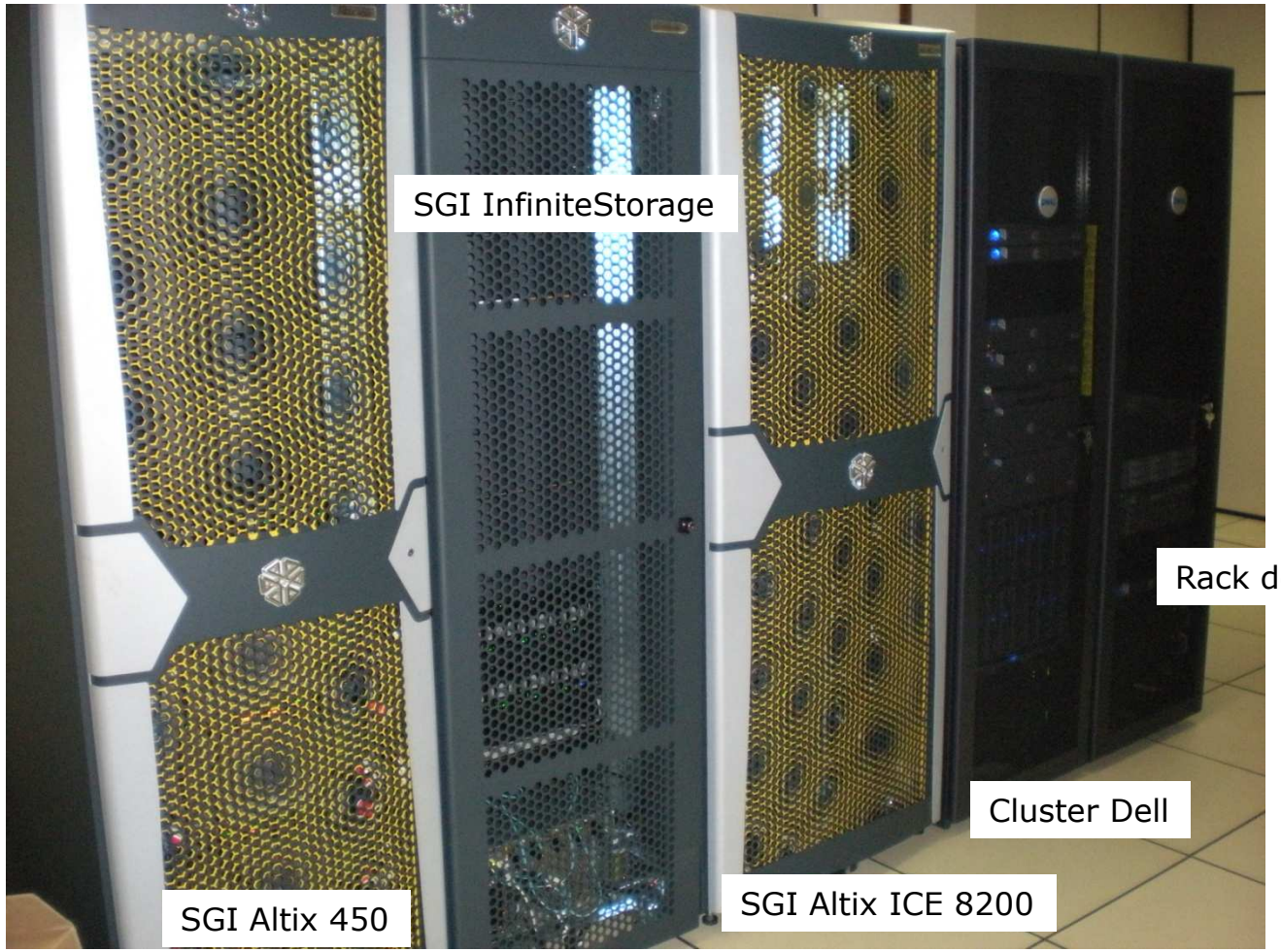
NACAD's Galileu System
#76 in TOP500, 1st in LA

SUN Cluster 6464 Intel Nehalem cores, IB

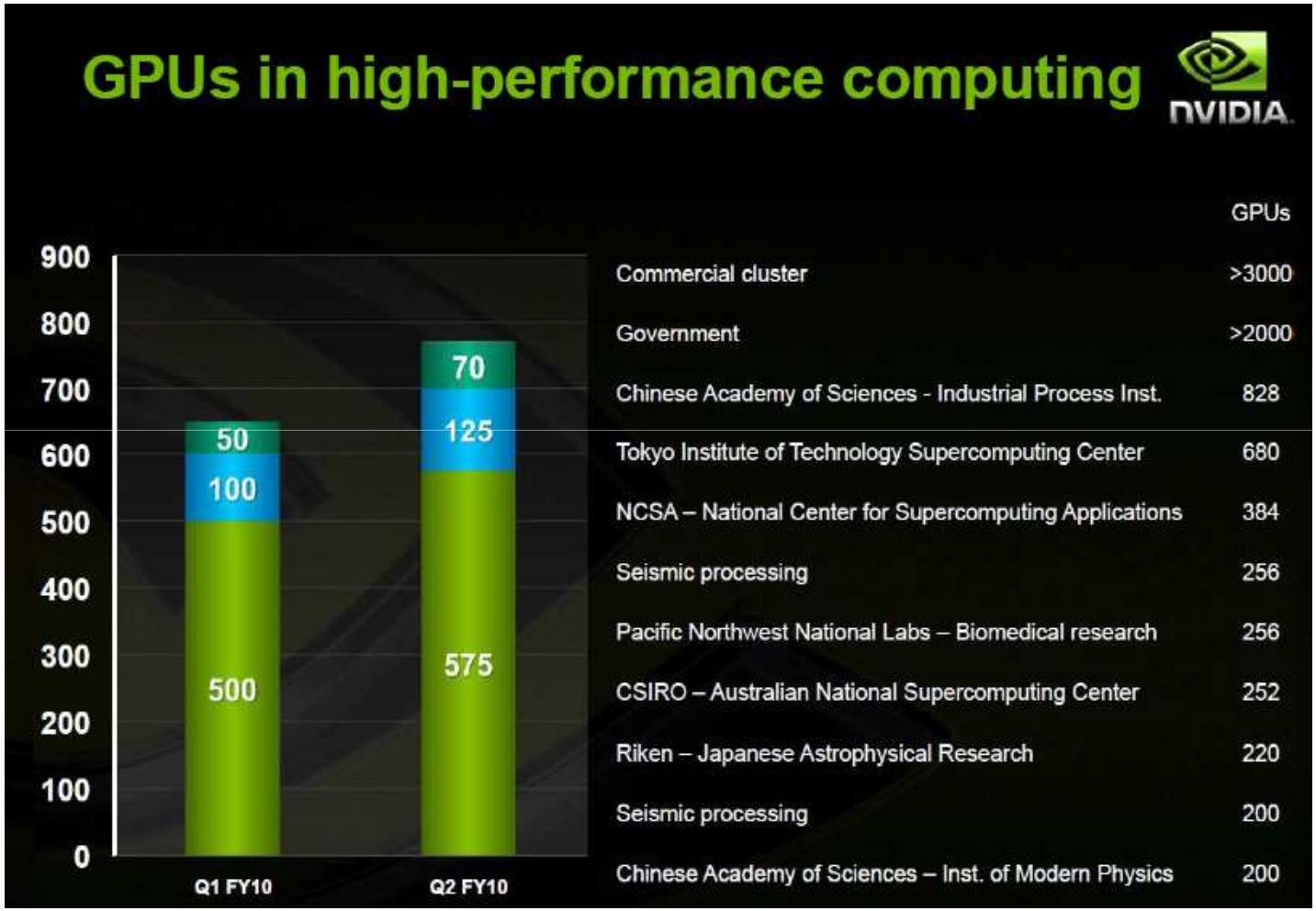
Preliminary tests: 65Tflops Final configuration: 7200 cores


TOP500
CERTIFICATE
Sun Blade x6048, Xeon X5560 2.8 Ghz, Infiniband QDR
NACAD/COPPE/UFRJ, Brazil
Is ranked
No. 1 in Latin America
among the world's TOP500 Supercomputers
with 64.63 Tflop/s Linpack Performance
On the TOP500 List published at the SC09 conference, November 17th, 2009
Congratulations from The TOP500 Editors
Hans Meuer
University of Mannheim
Erich Strohmaier
NERSC/Berkeley Lab
Jack Dongarra
University of Tennessee
Horst Simon
NERSC/Berkeley Lab

NACAD's Research Systems

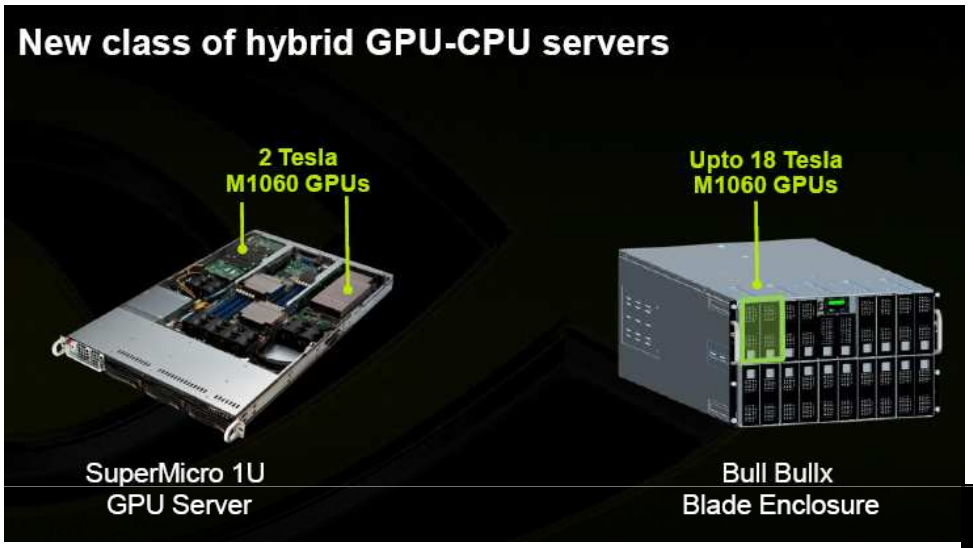


Hybrid Architectures GPGPUs



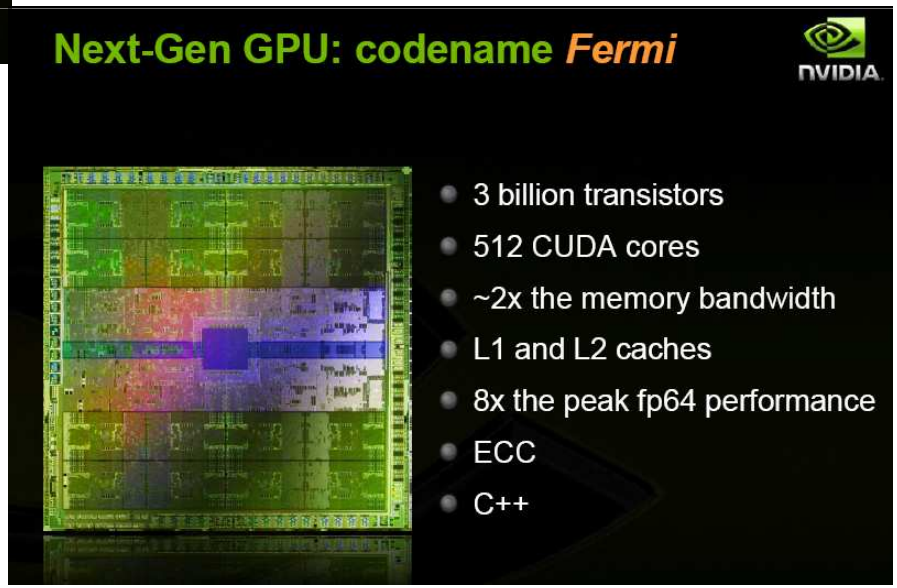
source: http://gpgpu.org/sc2009

New Products Coming



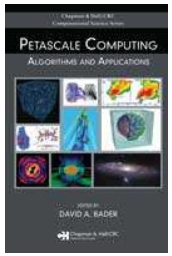
More integration into existing hardware

Requires specific tools for programming



More reading

- T.H. Dunning et al, Science and Engineering in the Petascale Era, IEEE Computing in Science & Engineering, September/October 2009
- Peter M. Kogge, The Challenges of Petascale Architectures, IEEE Computing in Science & Engineering, September/October 2009
- William D. Gropp, Software for Petascale Computing Systems, IEEE Computing in Science & Engineering, September/October 2009
- D. Bader, Petascale Computing



The Grid

- Motivation: When communication is close to free we should not be restricted to local resources when solving problems.

- Infrastructure that builds on the Internet and the Web
- Enable and exploit large scale sharing of resources
- Virtual organization
  - Loosely coordinated groups
- Provides for remote access of resources
  - Scalable
  - Secure
  - Reliable mechanisms for discovery and access

In some ideal setting:

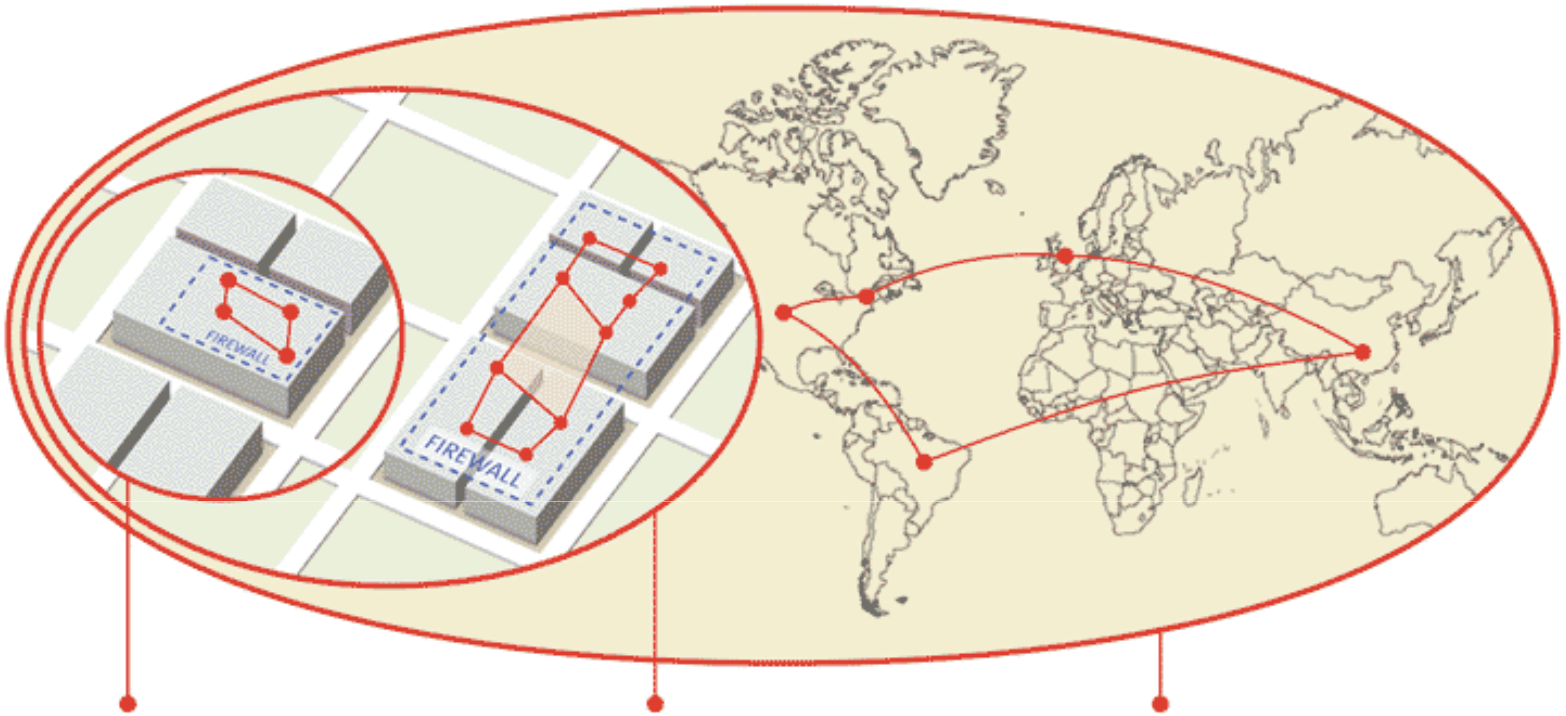
User submits work, infrastructure finds an execution target

Ideally you don't care where.

07

30

Fonte: J. Dongarra, http://www.netlib.org/utk/people/JackDongarra/talks.html



CLUSTER GRID
Departmental Computing

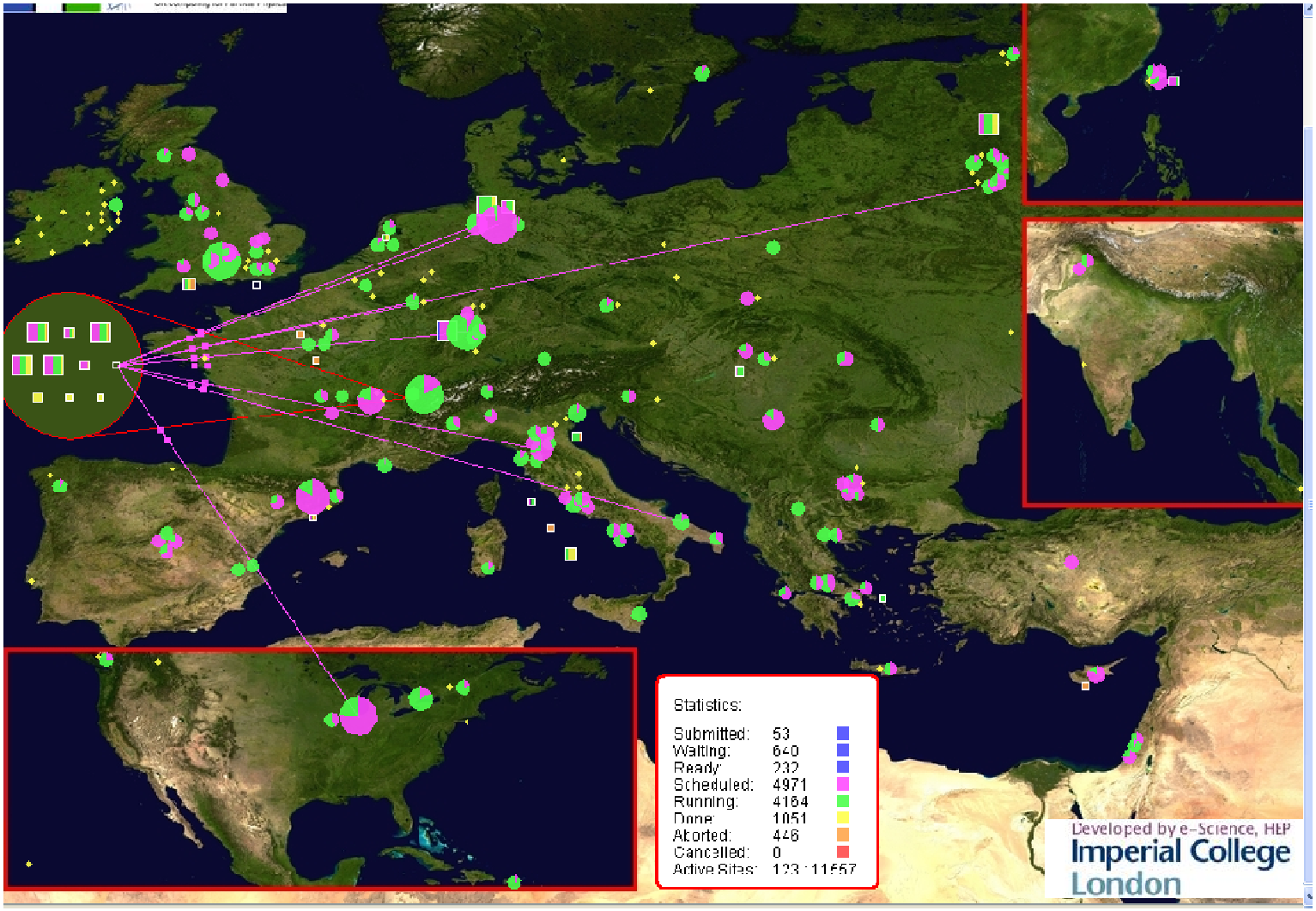
- Simplest Grid deployment
- Maximum utilization of departmental resources
- Resources allocated based on priorities

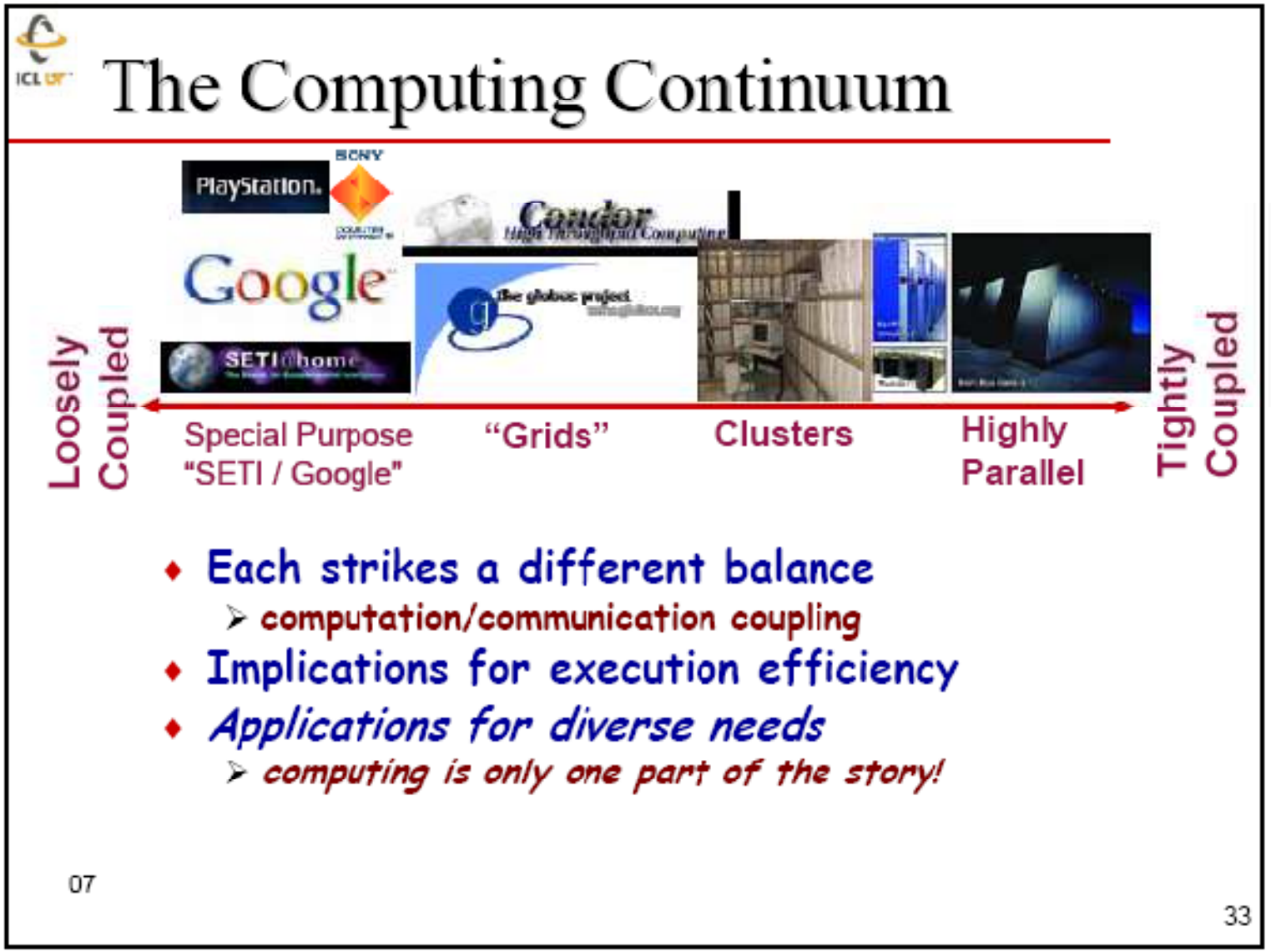
CAMPUS GRID
Enterprise Computing

- Resources shared within the enterprise
- Policies ensure computing on demand
- Gives multiple groups seamless access to enterprise resources

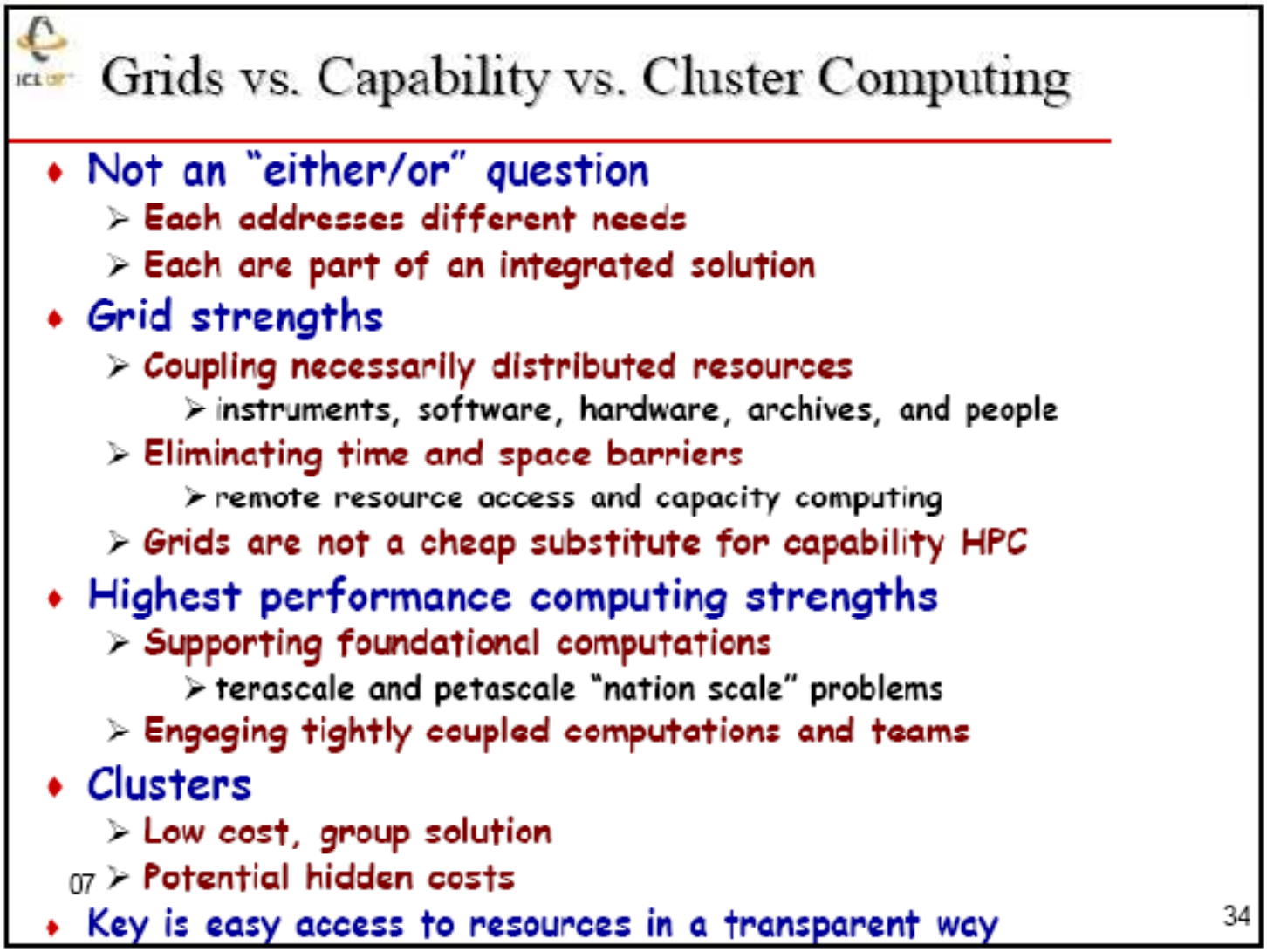
GLOBAL GRID
Internet Computing

- Resources shared over the Internet
- Global view of distributed datasets
- Growth path for enterprise Campus Grids


Statistics:
Submitted: 53
Waiting: 640
Ready: 232
Scheduled: 4971
Running: 4164
Done: 1051
Aborted: 446
Cancelled: 0
Active Sites: 133 : 11557
Developed by e-Science, HEP
Imperial College
London



from: J. Dongarra, http://www.netlib.org/utk/people/JackDongarra/talks.html



from: J. Dongarra, http://www.netlib.org/utk/people/JackDongarra/talks.html

Future HPC Systems

Exascale Computing

- Exascale systems are likely feasible by 2017±2
- 10-100 Million processing elements (cores or mini-cores) with chips perhaps as dense as 1,000 cores per socket, clock rates will grow more slowly
- 3D packaging likely
- Large-scale optics based interconnects
- 10-100 PB of aggregate memory
- Hardware and software based fault management
- Heterogeneous cores
- Performance per watt — stretch goal 100 GF/watt of sustained performance ⇒ >> 10 - 100 MW Exascale system
- Power, area and capital costs will be significantly higher than for today's fastest systems


ExaScale Computing Study:
Technology Challenges in
Achieving Exascale Systems

Peter Kogge, Editor & Study Lead
Keren Bergman
Shekhar Borkar
Dan Campbell
William Carlson
William Dally
Monty Denneau
Paul Franzon
William Harrod
Kerry Hill
Jon Hiller
Sherman Karp
Stephen Keckler
Dean Klein
Robert Lucas
Mark Richards
Al Scarpelli
Steven Scott
Allan Snavely
Thomas Sterling
R. Stanley Williams
Katherine Yelick

September 28, 2008

NOTICE

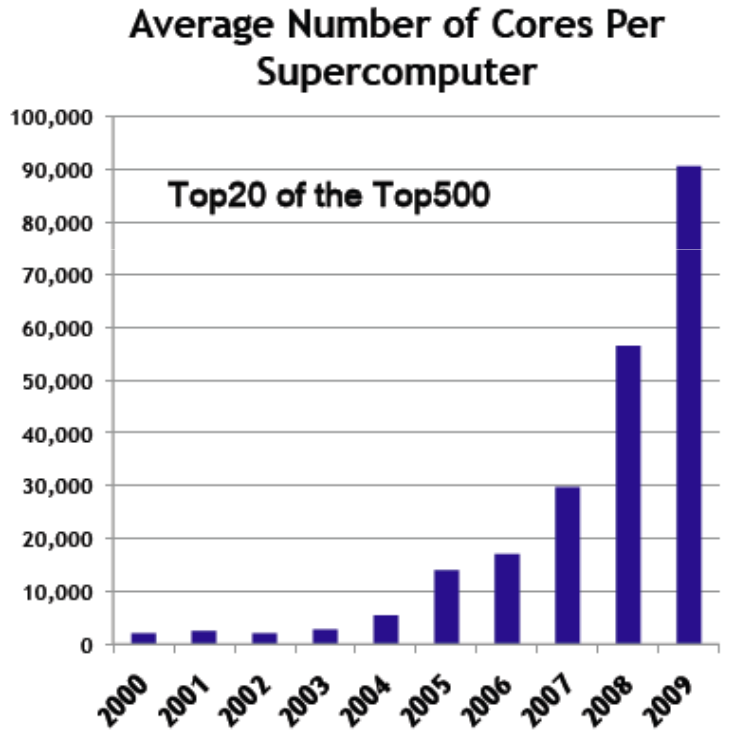
APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED

from: J. Dongarra, http://www.netlib.org/utk/people/JackDongarra/talks.html

Future HPC Systems

Hardware and System Software Scalability

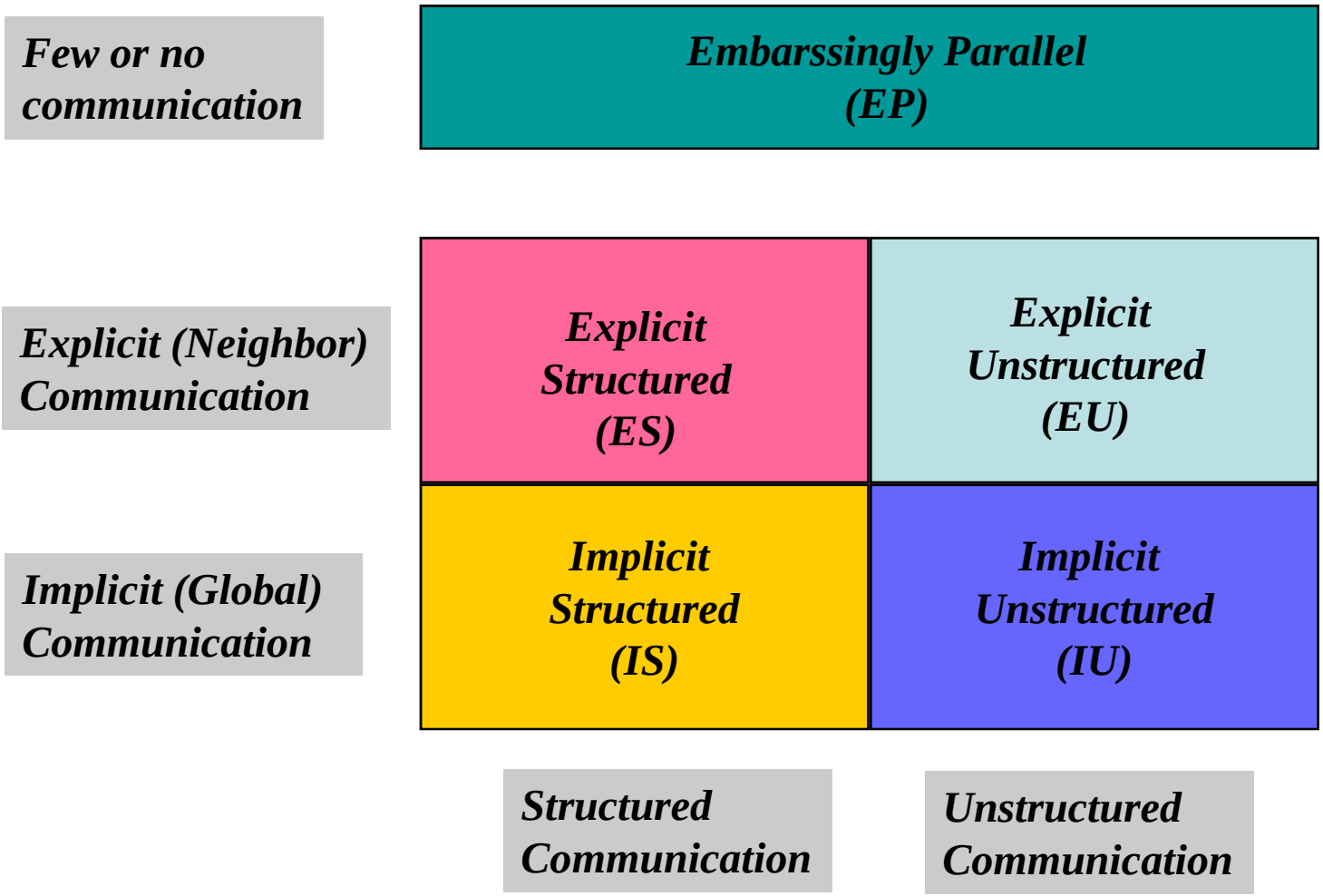
- **Barriers**
  - Fundamental assumptions of system software architecture did not anticipate exponential growth in parallelism
  - Number of components and MTBF changes the game
- **Technical Focus Areas**
  - System Hardware Scalability
  - System Software Scalability
  - Applications Scalability
- **Technical Gap**
  - 1000x improvement in system software scaling
  - 100x improvement in system software reliability



from: J. Dongarra, http://www.netlib.org/utk/people/JackDongarra/talks.html

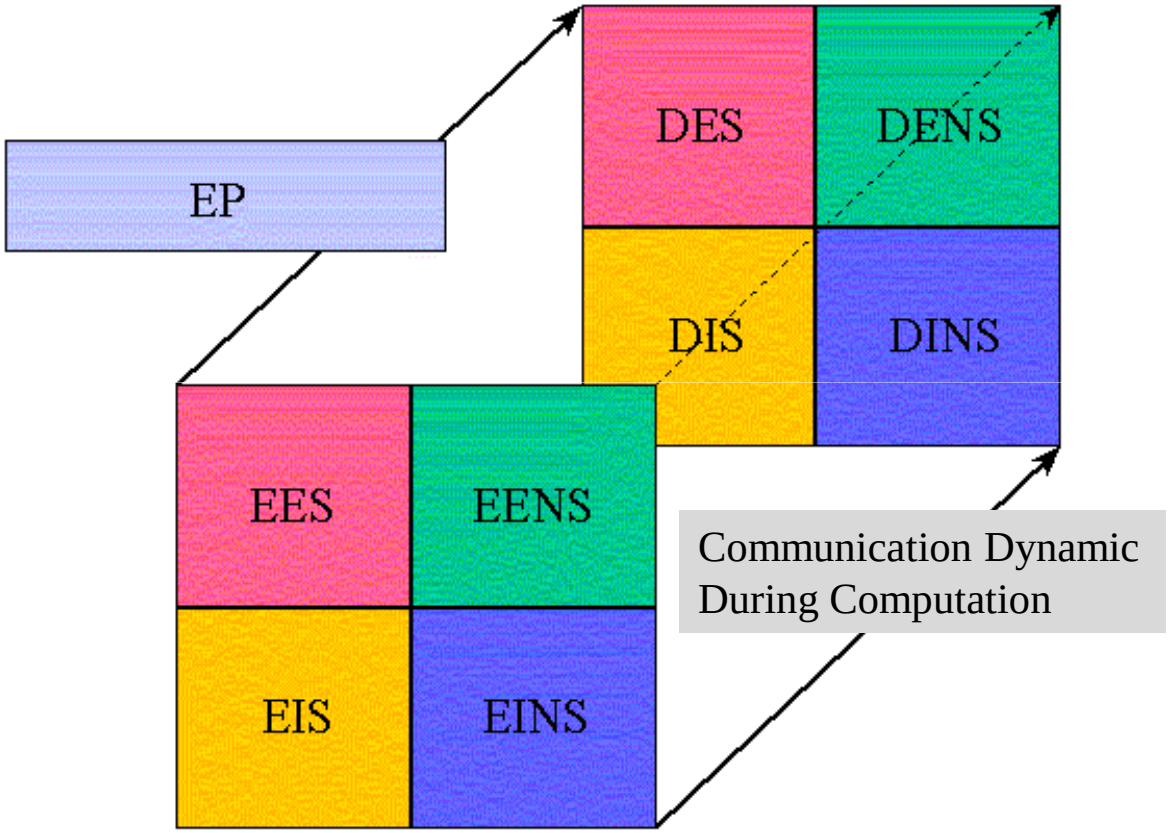
HIGH PERFORMANCE APPLICATIONS

Taxonomy of Parallel Applications

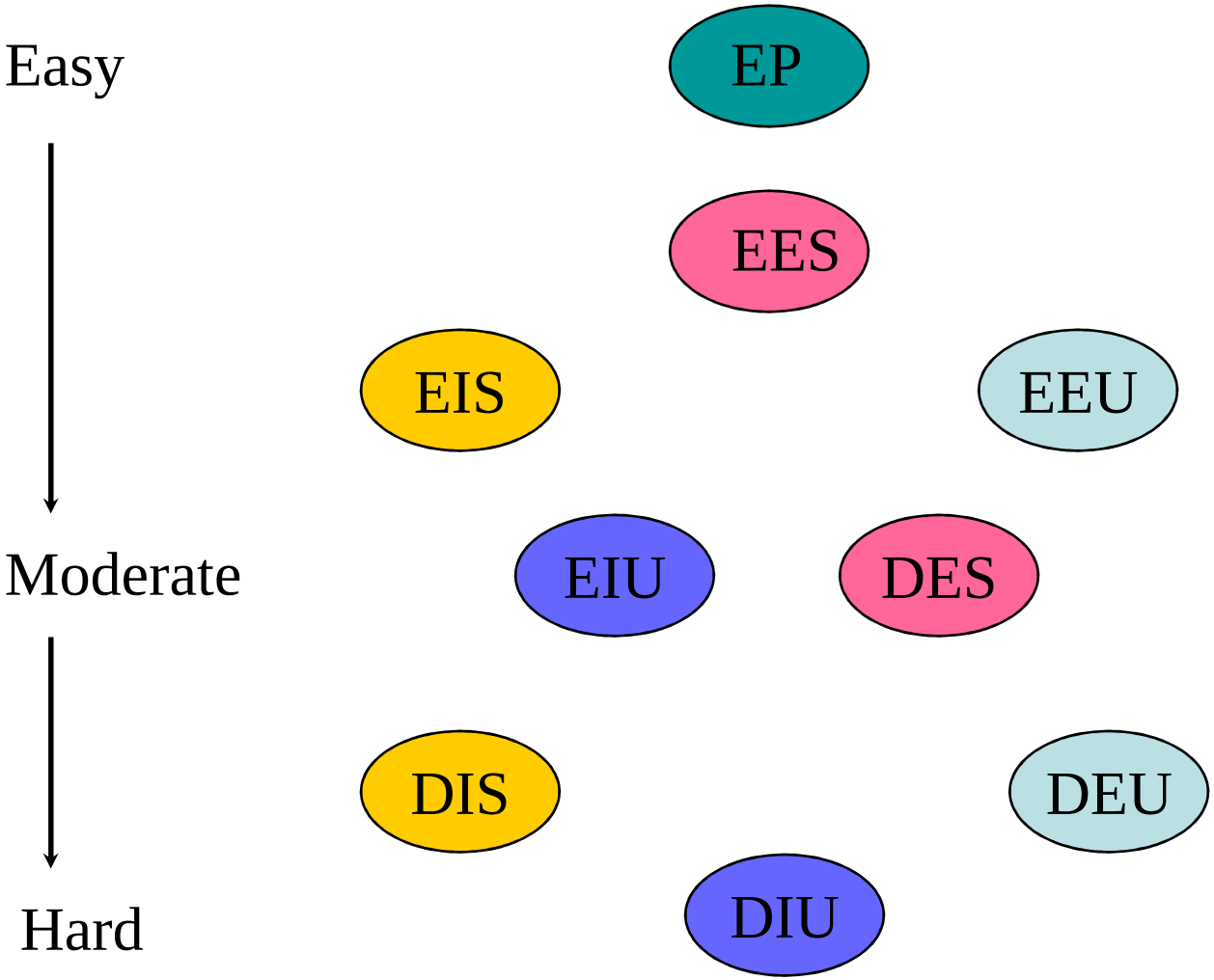


H. D. Simon. High performance computing: Architecture, software, algorithms.
Technical Report RNR-93-018, NASA Ames Research Center, Moffett Field, CA 94035, December 1993

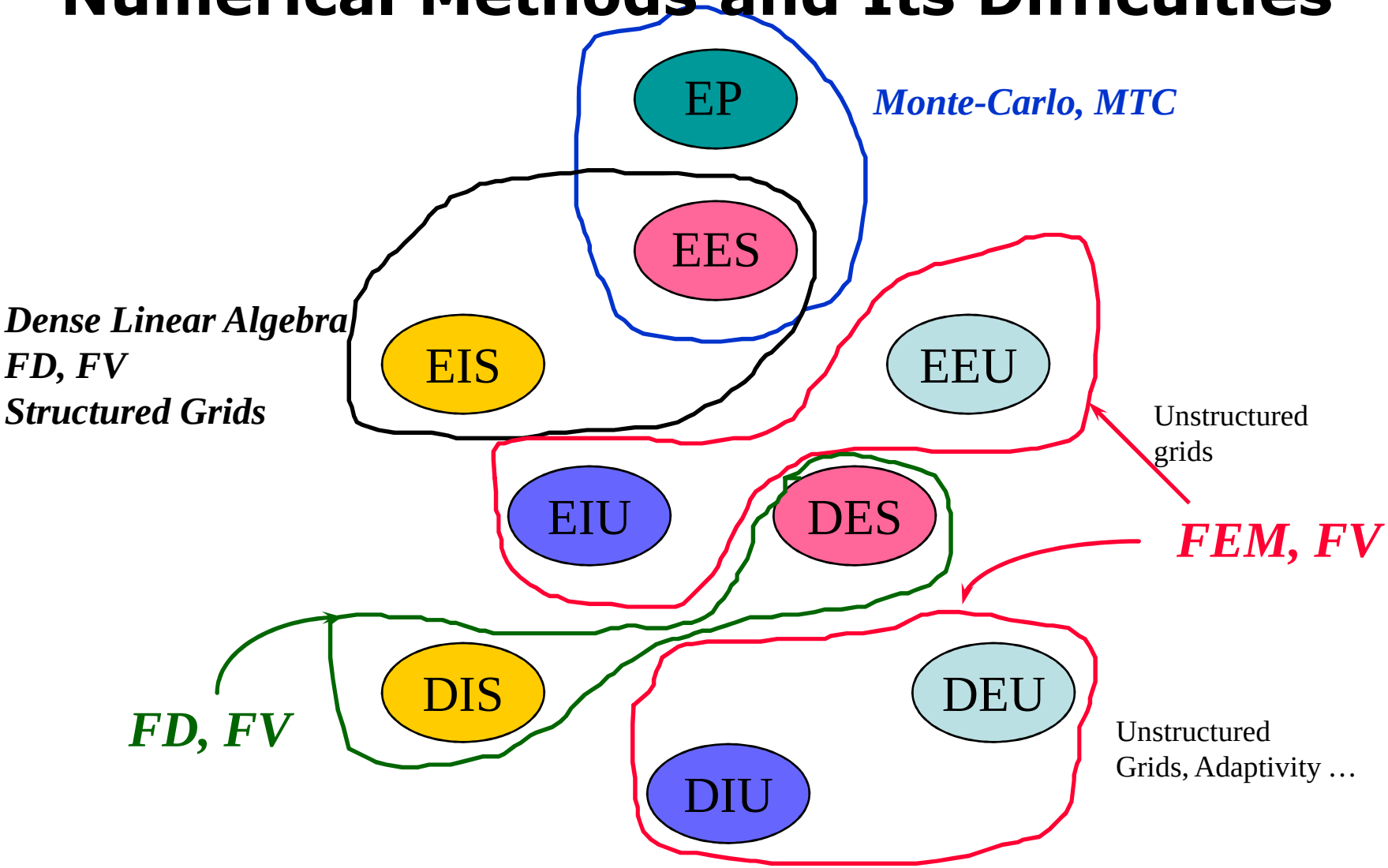
Taxonomy of Parallel Applications – Computations Changing Dynamically During Execution



Parallel Implementation Difficulties



Numerical Methods and Its Difficulties



Measuring Parallel Performance

- T_1 – serial execution time (1 processor)
- T_P – parallel execution time in p processors
- Speed-up → $S_P = T_1/T_P$
- Efficiency → $E_P = T_1/(pT_P)$
- Therefore:

$$E_P = S_P/p \qquad S_P \leq p \quad E_P \leq 1$$

- Note: anomalies may happen due to other resources (cache) as p increases– superlinear speed-up

Amdahl's Law (1967)

- **Serial fraction:** s, $0 \leq s \leq 1$
- **Parallel fraction in p processors:** $1-s$
- **Then:**

$$T_p = sT_1 + (1-s)T_1/p,$$

$$S_p = p/(sp + (1-s)),$$

$$E_p = 1/(sp + (1-s)).$$

- **Corollary:** $S_p \to 1/s$ and $E_p \to 0$ as $p \to \infty$.

Scalability

- **Scalability refers on how an given algorithm can use efficiently additional processors;**
- **An algorithm is scalable when *p* increases if its efficiency is constant as problem size increases**
- **Those ideas goes back to Gustafson (1988)**
  - http://www.scl.ameslab.gov/Publications/Gus/AmdahlsLaw/Amdahls.html
  - Benner, R.E., Gustafson, J.L., and Montry, G.R., Development and analysis of scientific application programs on a 1024-processor hypercube," SAND 88-0317, Sandia National Laboratories, Feb. 1988.

Problem Scaling

- **Amdahl's law is relevant only for a fixed size problem, or when the serial fraction is independent of problem size, a situation hard to find in practice.**
- **Often parallel fraction increases with problem size;**
- **Parallel fraction growth rate can be characterized keeping a fixed quantity as *p* varies:**
  - Size *n* (Amdahl), computational work, execution time, memory per processor, efficiency, etc.
- **Today it's hard to have serial time available**
  - Strong scaling – *p* increases, *n* fixed
  - Weak scaling – *p* increases, *n* increases

Performance Measurement: NPB

NAS Parallel Benchmarks (NPB) are a small set of programs designed to help evaluate the performance of parallel supercomputers.

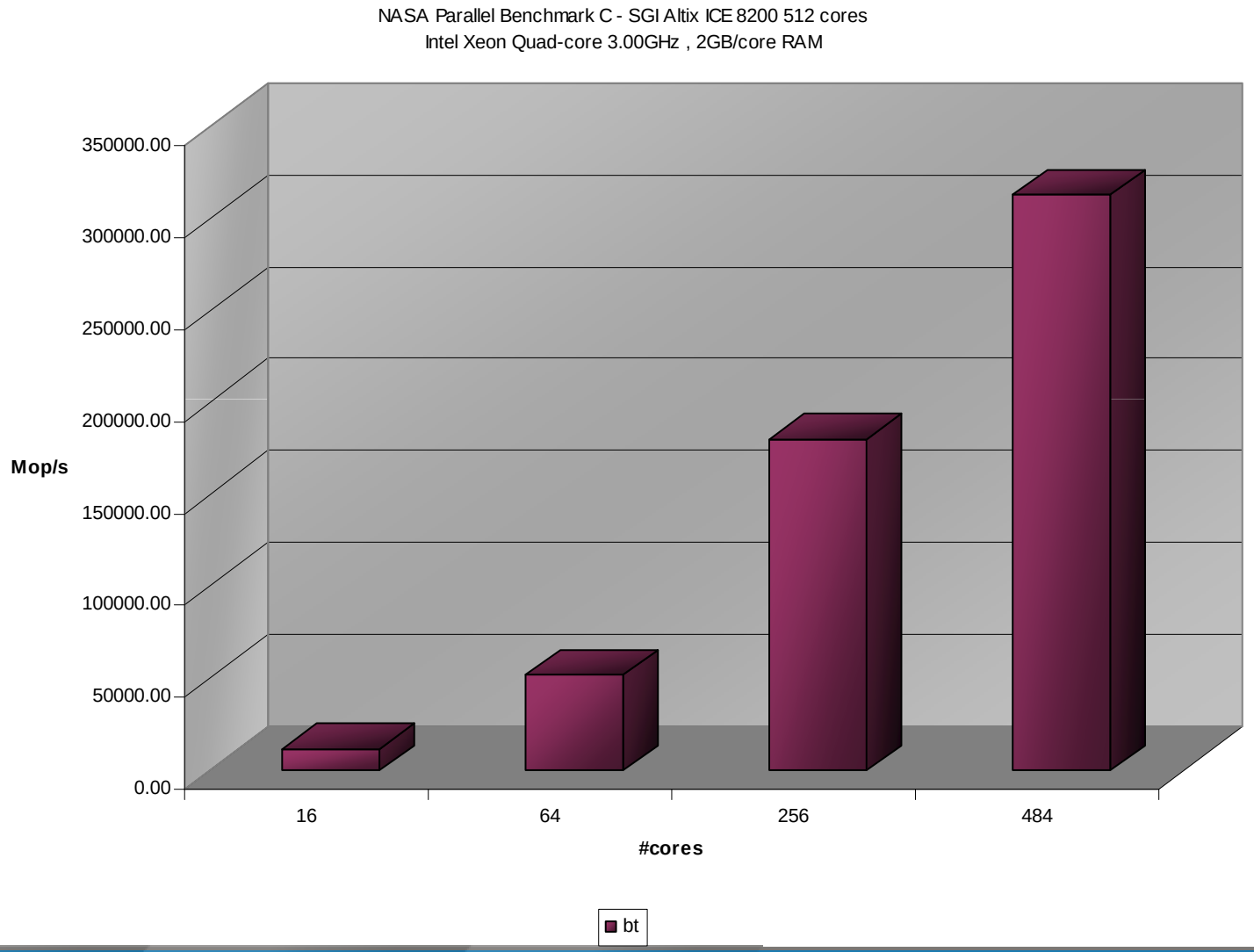
5 kernels (EP, MG, CG, 3D-FFT, IS)

3 CFD pseudo-applications (LU, SP, BT)

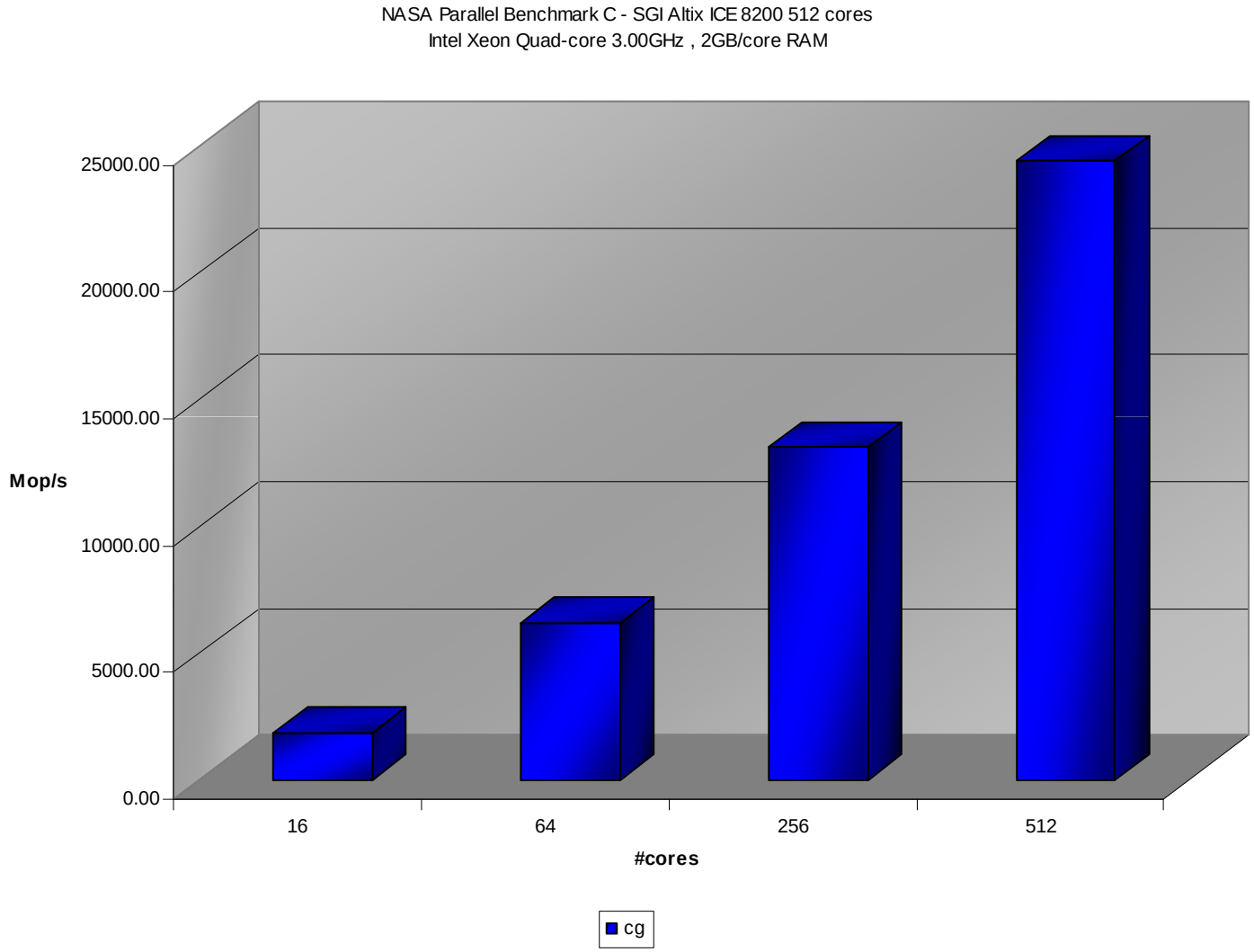
http://www.nas.nasa.gov/Resources/Software/npb.html

see: Subhash Saini, Dale Talcott, Dennis Jespersen, Jahed Djomehri, Haoqiang Jin, and Rupak Biswas, Scientific Application-Based Performance Comparison of SGI Altix 4700, IBM Power5+, and SGI ICE 8200 Supercomputers, NAS Technical Report NAS-09-001, February 2009

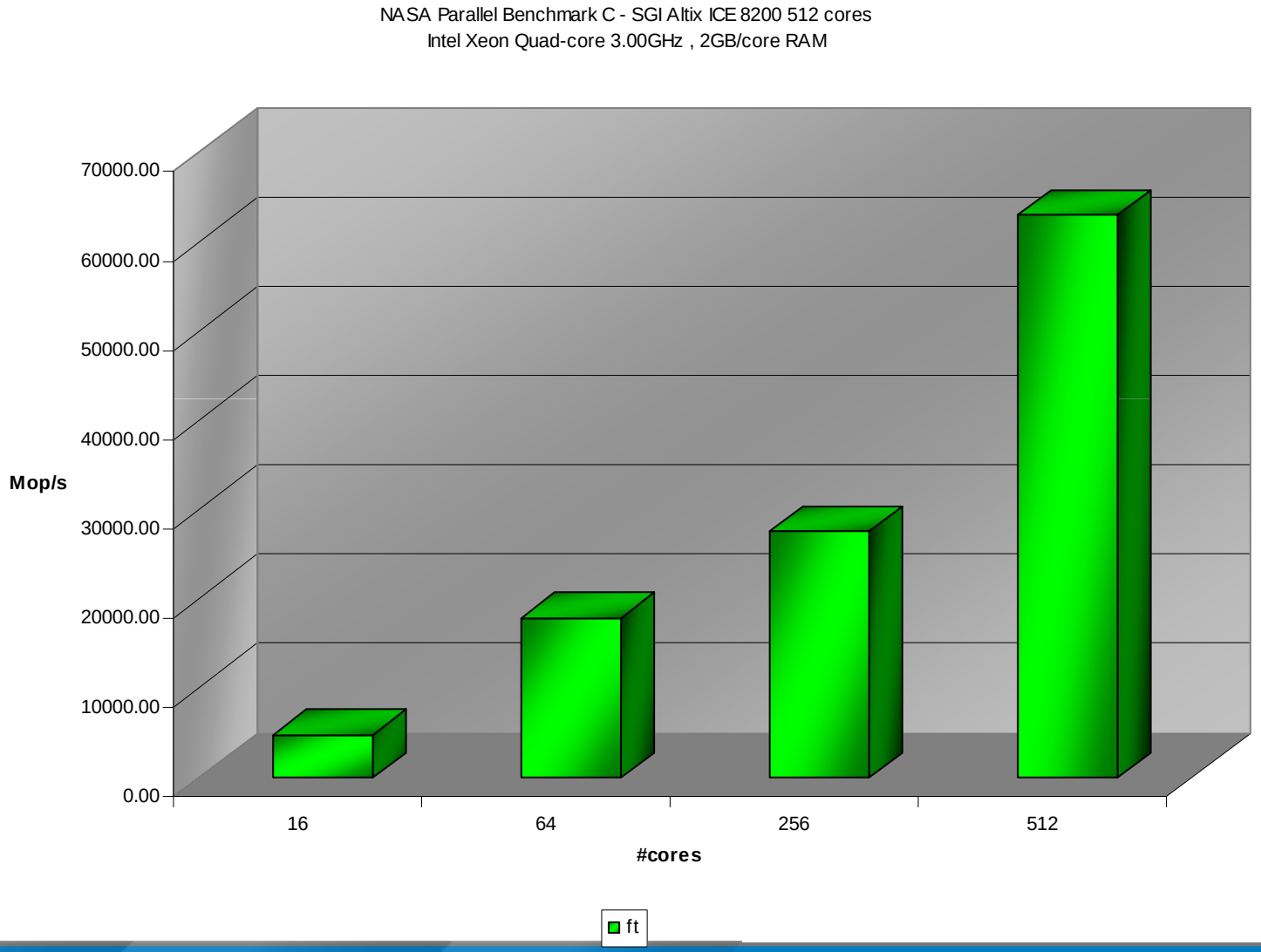
Benchmarks – NASA Parallel Benchmark bt



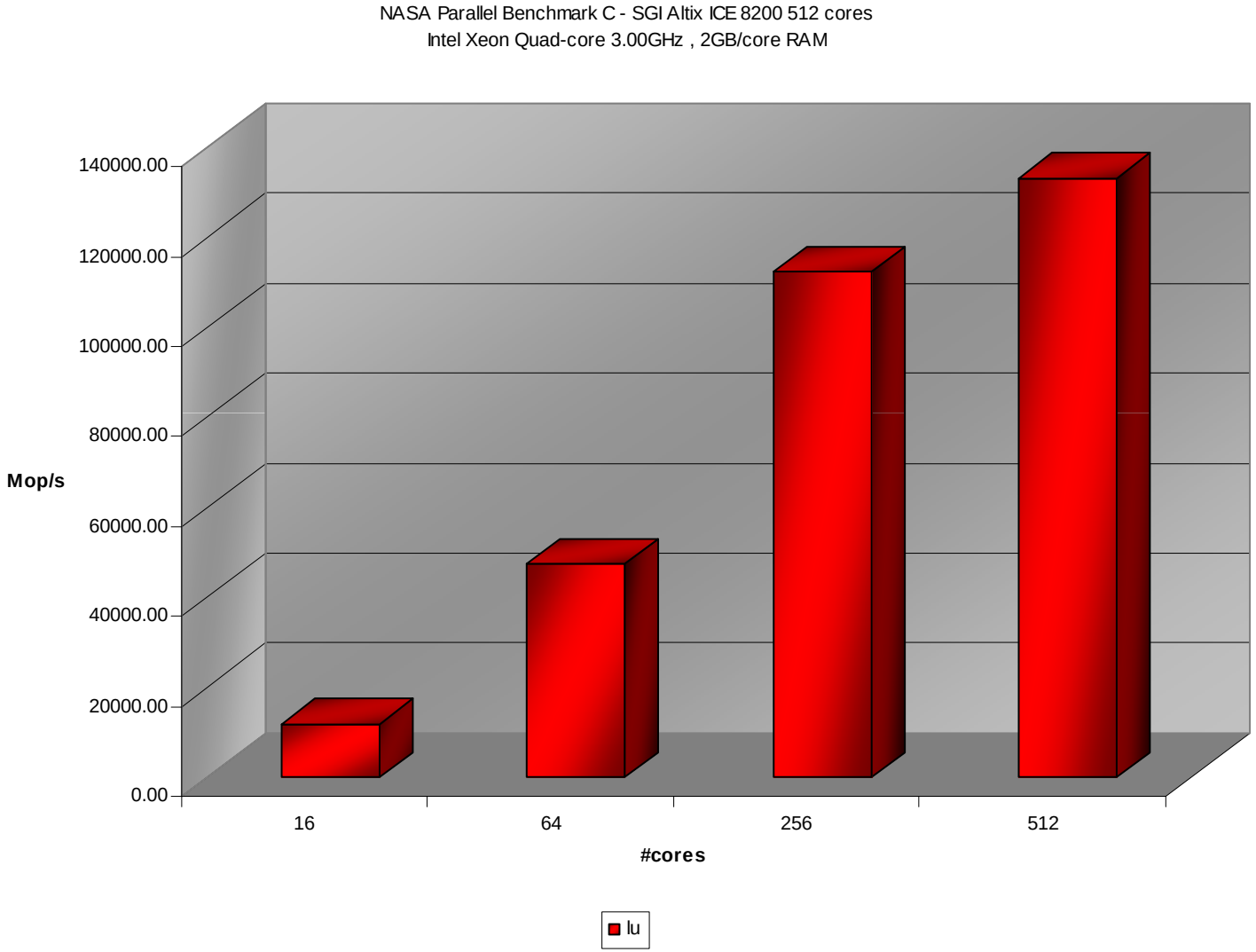
Benchmarks – NASA Parallel Benchmark cg



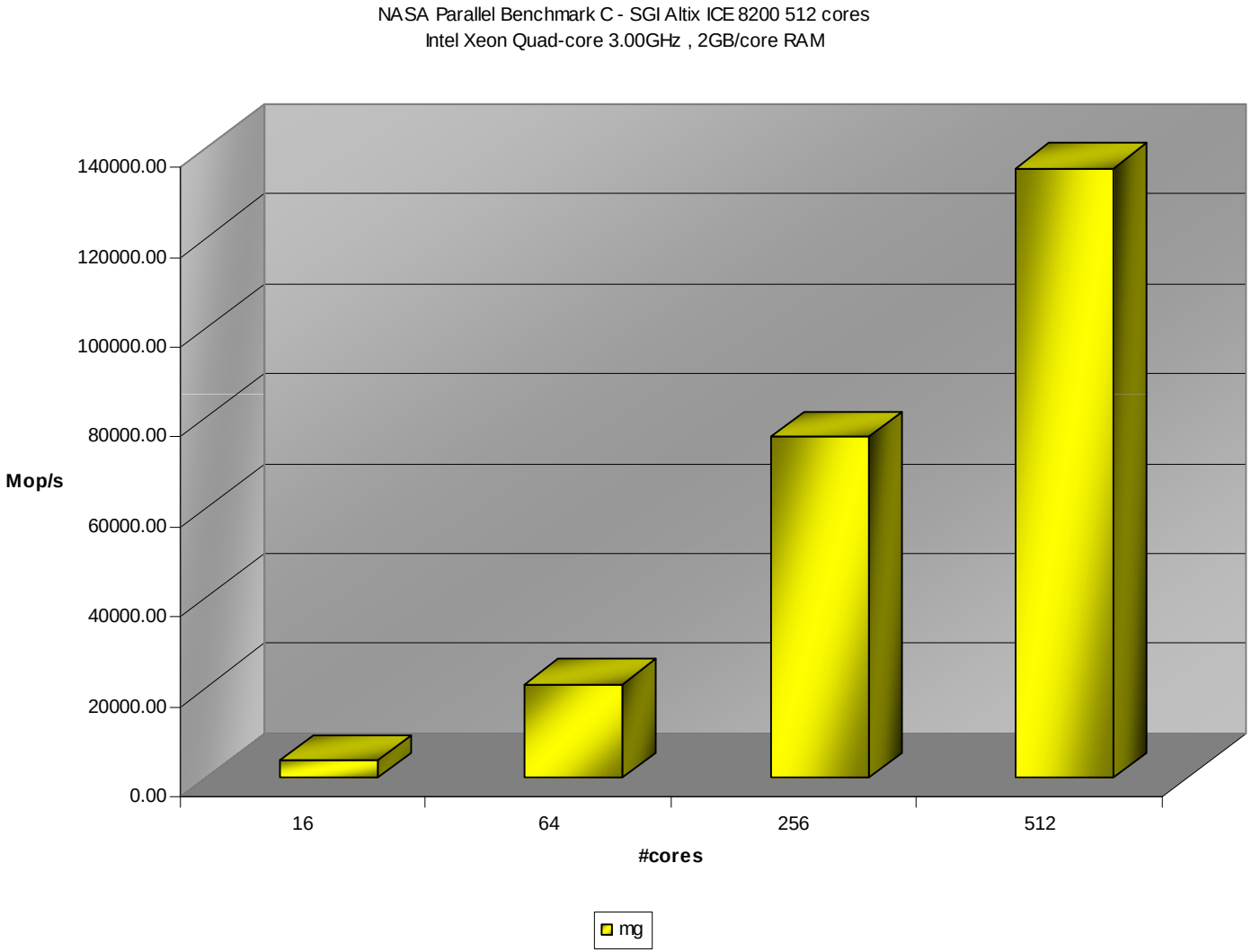
Benchmarks – NASA Parallel Benchmark ft



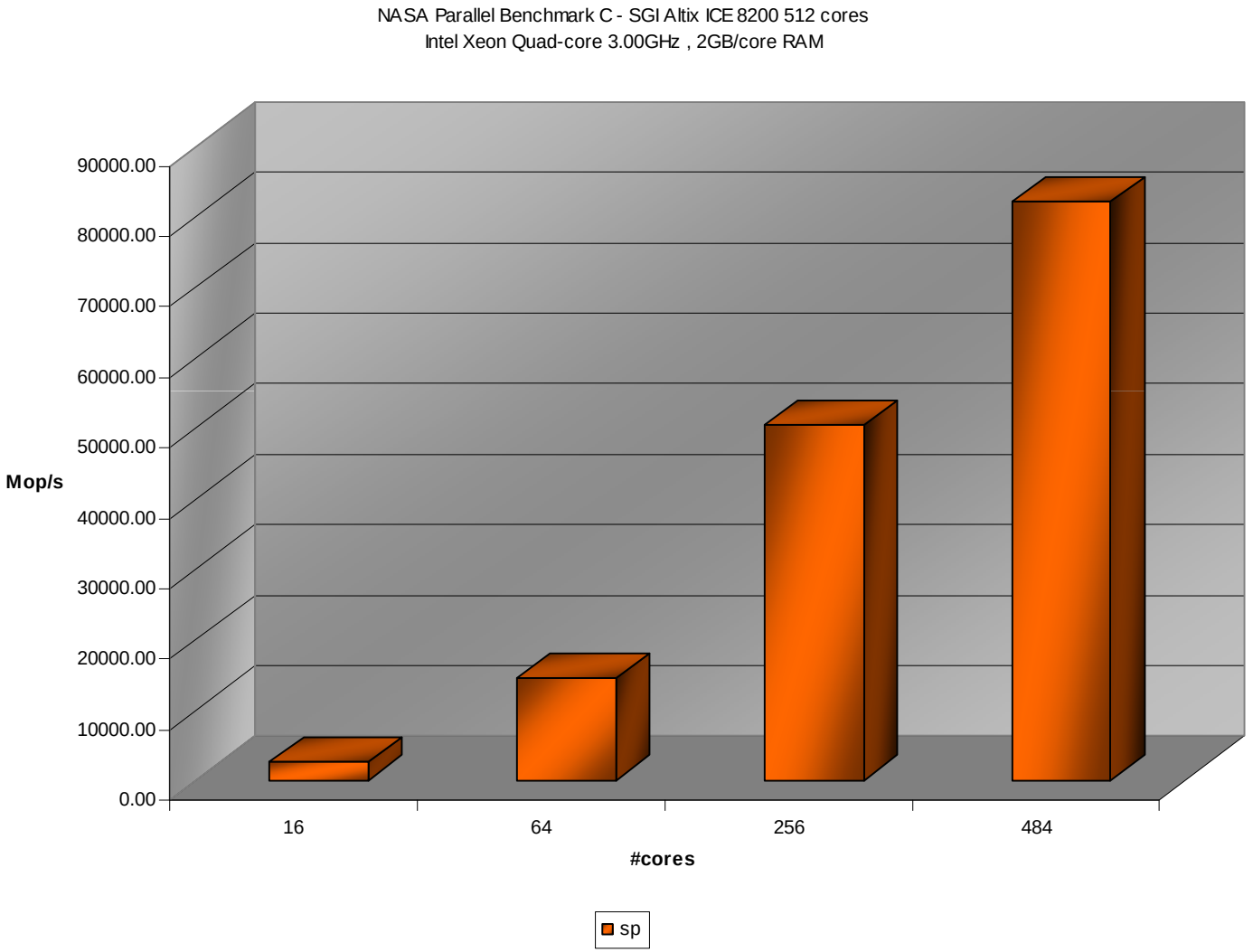
Benchmarks – NASA Parallel Benchmark lu



Benchmarks – NASA Parallel Benchmark mg



Benchmarks – NASA Parallel Benchmark sp



Code Optimization and Programming

Before even thinking in parallelizing a code:

- **Optimize your code for a given class of processors**
  - This is what reduces CPU time
- **Use all optimization TOOLS existing in the compiler and in the system;**
- **Always verify if the code is working properly;**
- **Always use standard libraries such as BLAS, LAPACK, etc.**

Basic Parallel Programming Models

- **Distributed Memory Machines**

  Message Passing: `send/receive`

- **Message Passing Library**

  Message Passing Interface http://www.mpi.com

```
call MPI_Send( ... )
call MPI_Recv( ... )
```

Basic Parallel Programming Models

- **Threaded machines: compiler directives**

OpenMP

```
!$OMP PARALLEL DO PRIVATE (J)
DO J=1,M
...
ENDDO
```

http://www.openmp.org

OpenCL

http://www.khronos.org/opencl/

CUDA:

```
// send data from host to device: a_h to a_d
cudaMemcpy(a_d, a_h, sizeof(float)*N,
cudaMemcpyHostToDevice);
```

http://www.nvidia.com/object/cuda_home.html#/

Development Tools

Numerical Libraries

- Netlib: http://www.netlib.org
- ACTS (Advanced Computational Testing and Simulation) Toolkit http://acts.nersc.gov/
  - PETSc (Portable, Extensible Toolkit for Scientific Computation)
  - ScaLAPACK library extends LAPACK's high-performance linear algebra software to distributed memory

Why computer simulation?

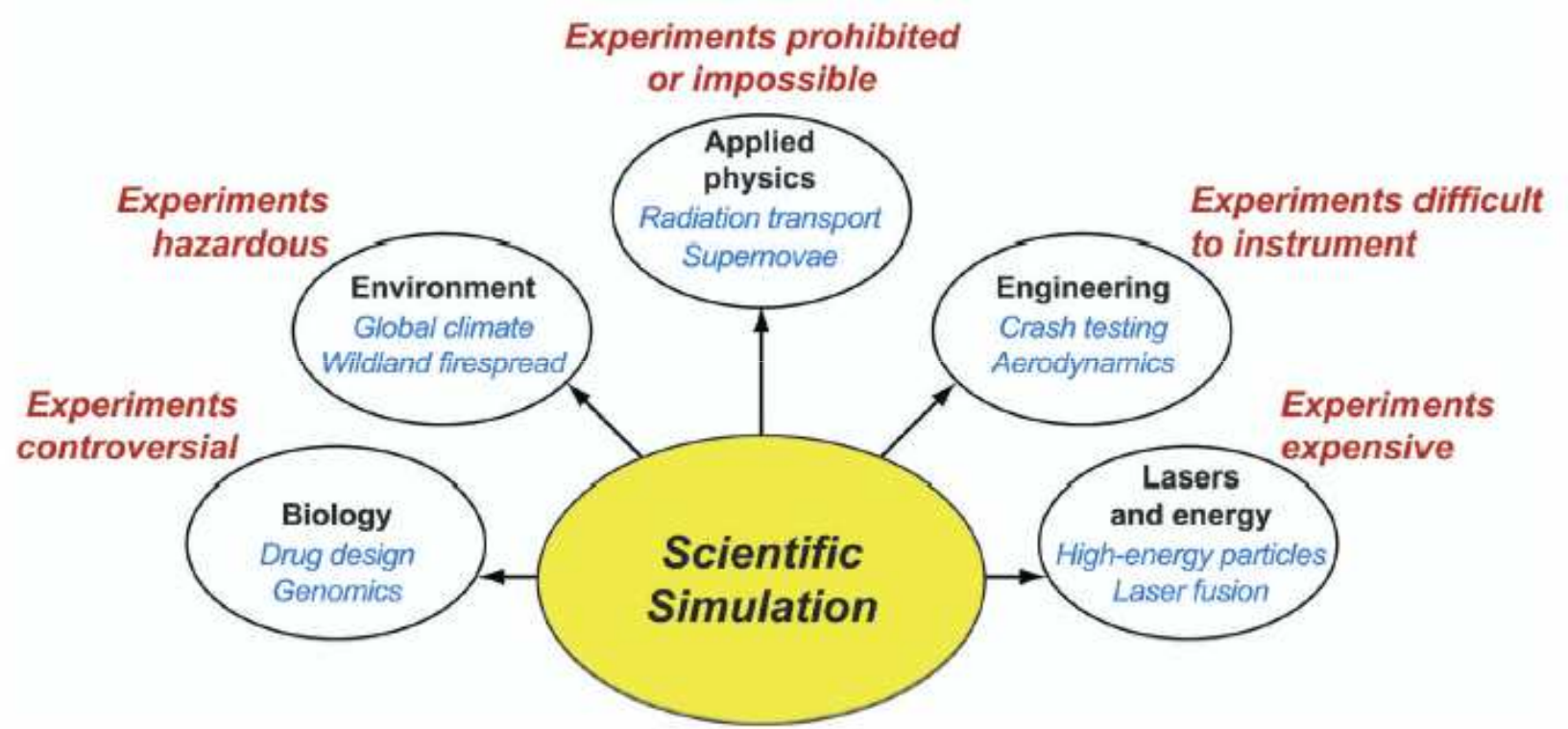


Figure 2. Practical strains on experimentation that invite simulation as a peer modality of scientific investigation.

From: A SCIENCE-BASED CASE FOR LARGE-SCALE SIMULATION, DOE, 2003

The process of scientific simulation

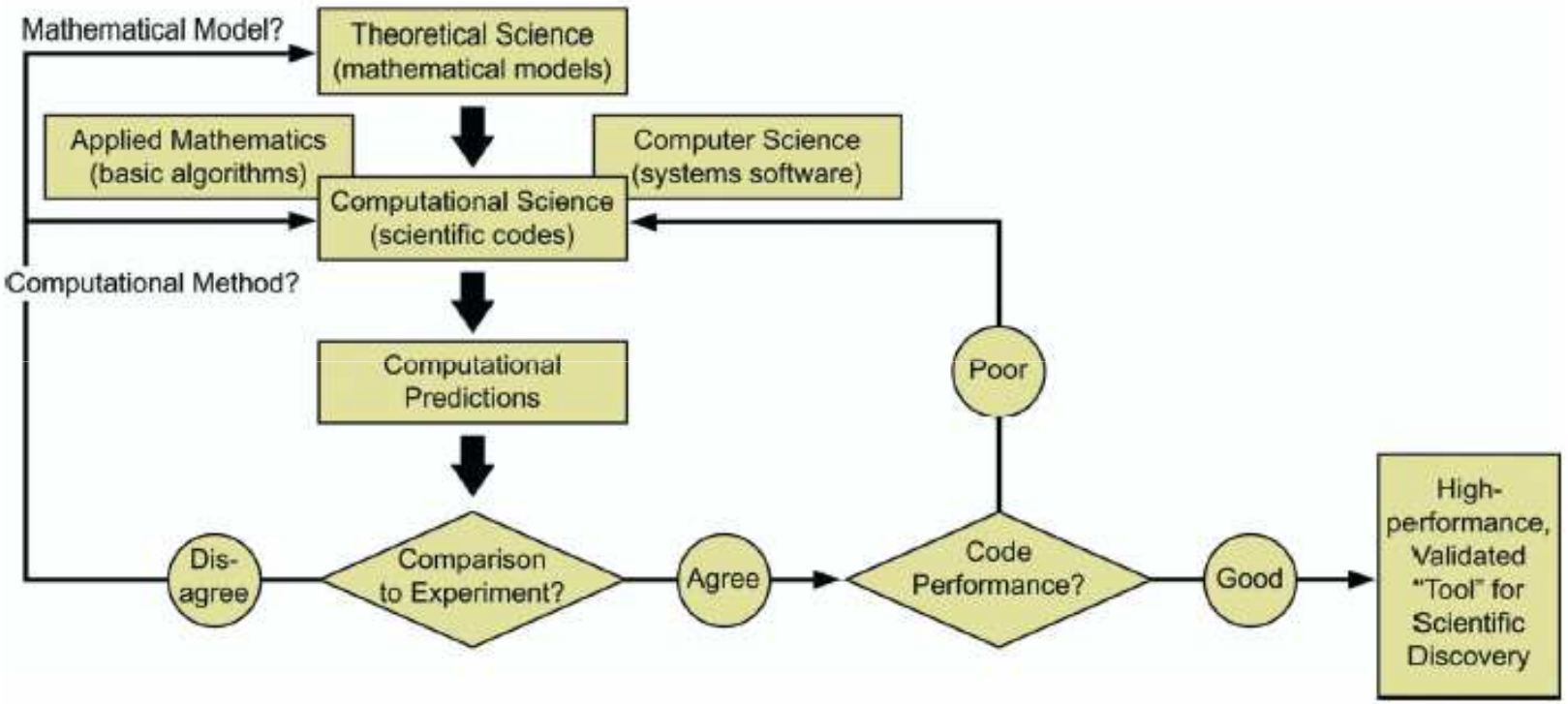


Figure 4. The process of scientific simulation, showing the validation and verification loop (*left*) and algorithm and performance tuning loop (*right*).

From: *A SCIENCE-BASED CASE FOR LARGE-SCALE SIMULATION*, DOE, 2003

HPC IN COMPUTATIONAL MECHANICS

The world made discrete: from PDE's to computer programs

- **General Form of PDE's for Engineering Systems**

Given $f : \Omega \rightarrow \mathbb{R}^{n_{sd}}$ e $r : \Gamma \rightarrow \mathbb{R}^{n_{sd}}$, $n_{sd} = 1, 2,$ or 3 and $t \in I = [0, T]$ find $u = u(x, t), x \in \mathbb{R}^{n_{sd}}$, such as:

$$\mathcal{L}(u) + f = O \quad \text{in} \quad \Omega \times I \ , \ \Omega \subset \mathbb{R}^{n_{sd}} \quad (1)$$

$$\mathcal{M}(u) + r = O \quad \text{in} \quad \Gamma \times I \quad (2)$$

$$u(x, 0) = u_o(x) \quad x \in \Omega \quad (3)$$

Governing Equations in Eulerian Framework

Navier-Stokes Equations

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u} - \nu\,\Delta\mathbf{u} + \frac{1}{\rho}\nabla p = \mathrm{q} + \mathrm{f}(T,\mathbf{c}) \qquad in\ \Omega$$

$$\nabla\cdot\mathbf{u} = 0 \qquad in\ \Omega$$

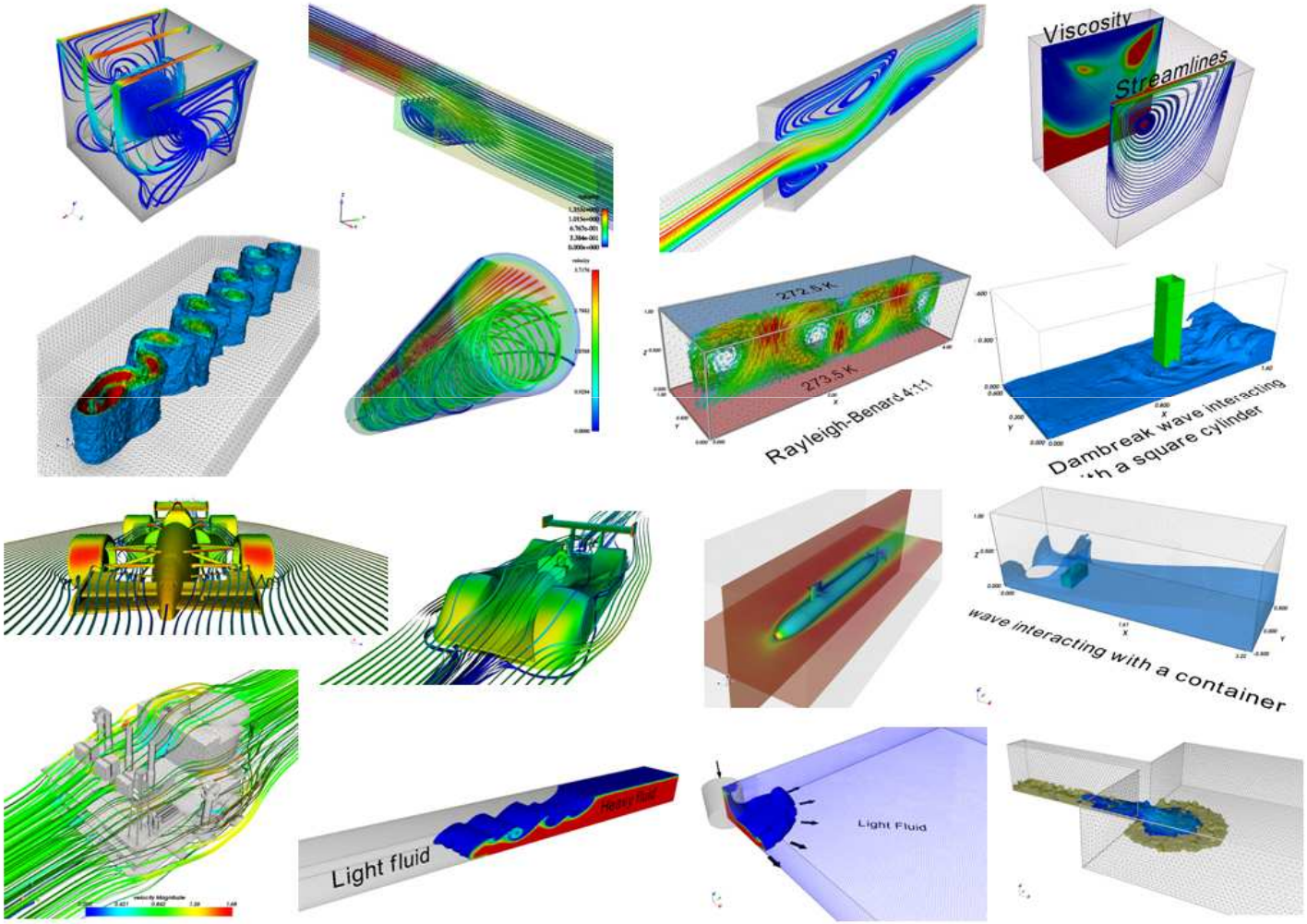
Energy Transport Equation

$$\rho c_p \frac{\partial T}{\partial t} + \rho c_p\,\mathbf{u}\cdot\nabla T - \nabla\cdot(k\nabla T) = h_1(T,\mathbf{c}) \qquad in\ \Omega$$

Mass Transfer Equations

$$\frac{\partial \mathbf{c}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{c} - \nabla\cdot(\mathrm{K}\nabla\mathbf{c}) = \mathrm{h}_2(T,\mathbf{c}) \qquad in\ \Omega$$

Parallel FEM Solver for Coupled Viscous Flow and Transport



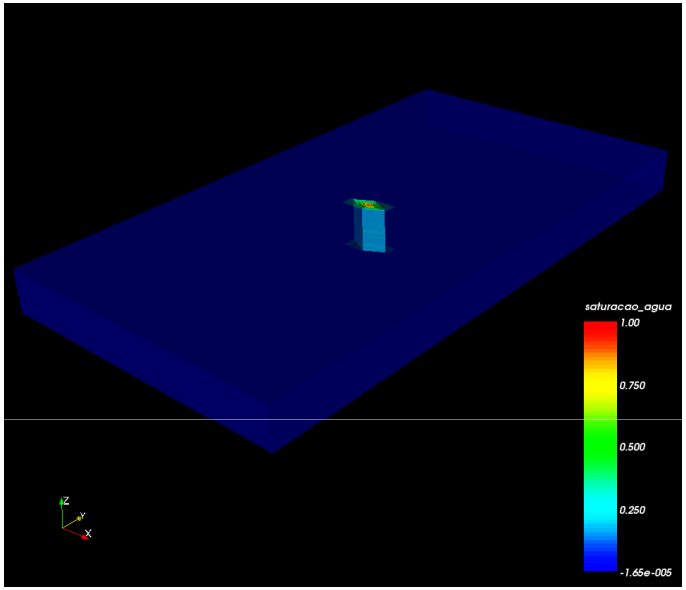
Eulerian Governing Equations

- **Multi-phase Darcy-flow in Porous Media:**

$$\mathbf{u}_\pi = -\frac{\kappa \mathbf{K}_{ij}}{\mu_\pi}\frac{\partial \Phi_\pi}{\partial x_j}$$

$$\Phi_\pi = p_\pi - \rho_\pi \mathbf{g}\cdot\mathbf{z}$$

$$\frac{\partial (S_\pi \rho_\pi \phi)}{\partial t} = \nabla\cdot\left(\frac{\mathbf{K}_{ij}}{\mu_\pi}\frac{\partial \Phi_\pi}{\partial x_j}\rho_\pi\right) + \rho_\pi q_\pi$$



$\pi = 1, 2, \ldots, n_{phases}$

Governing Equations in Lagrangian Framework

- **Equation of Motion for Solids and Structures:**

$$\rho u_{i,tt} - \sigma_{ij,j} + f_i = 0 \text{ in } \Omega \times I$$
$$u_i = g_i \quad \text{in} \quad \Gamma_g \times I$$
$$\sigma_{ij} n_j = h_i \quad \text{in} \quad \Gamma_h \times I$$
$$u_i(x,0) = u_{o_i}(x) \qquad x \in \Omega$$
$$u_{i,t}(x,0) = \dot{u}_{o_i}(x) \qquad x \in \Omega$$

Lagrangian Governing Equations

- **Remarks:**

1 Material Law

$$\sigma_{ij} = C_{ijkl}\,\epsilon_{kl}$$

2 Material Non-linearity

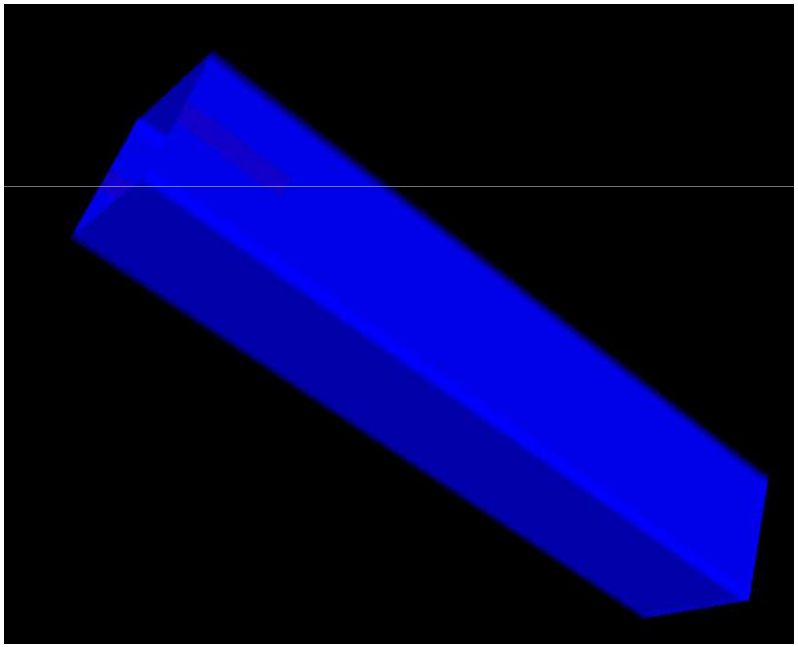
$$\epsilon = \epsilon^e + \epsilon^p$$

$$\epsilon = \text{plastic strains}$$

3 Large Displacements

$$\epsilon_{ij} \neq \frac{1}{2}(u_{i,j} + u_{j,i})$$

4 Contact



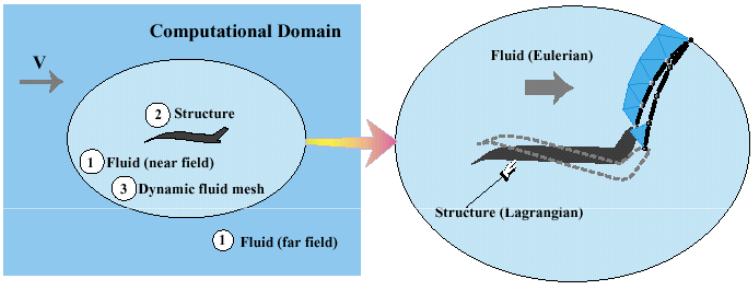
courtesy of J. Alves

Arbitrary Eulerian Lagrangian Governing Equations

- **Incompressible NS equations in ALE frame moving with velocity** w**:**

$$\rho\left[\frac{du_a}{dt}+(u_b-w_b)\frac{\partial u_a}{\partial x_b}\right]-\frac{\partial \tau_{ab}}{\partial x_b}+\frac{\partial p}{\partial x_a}=0$$

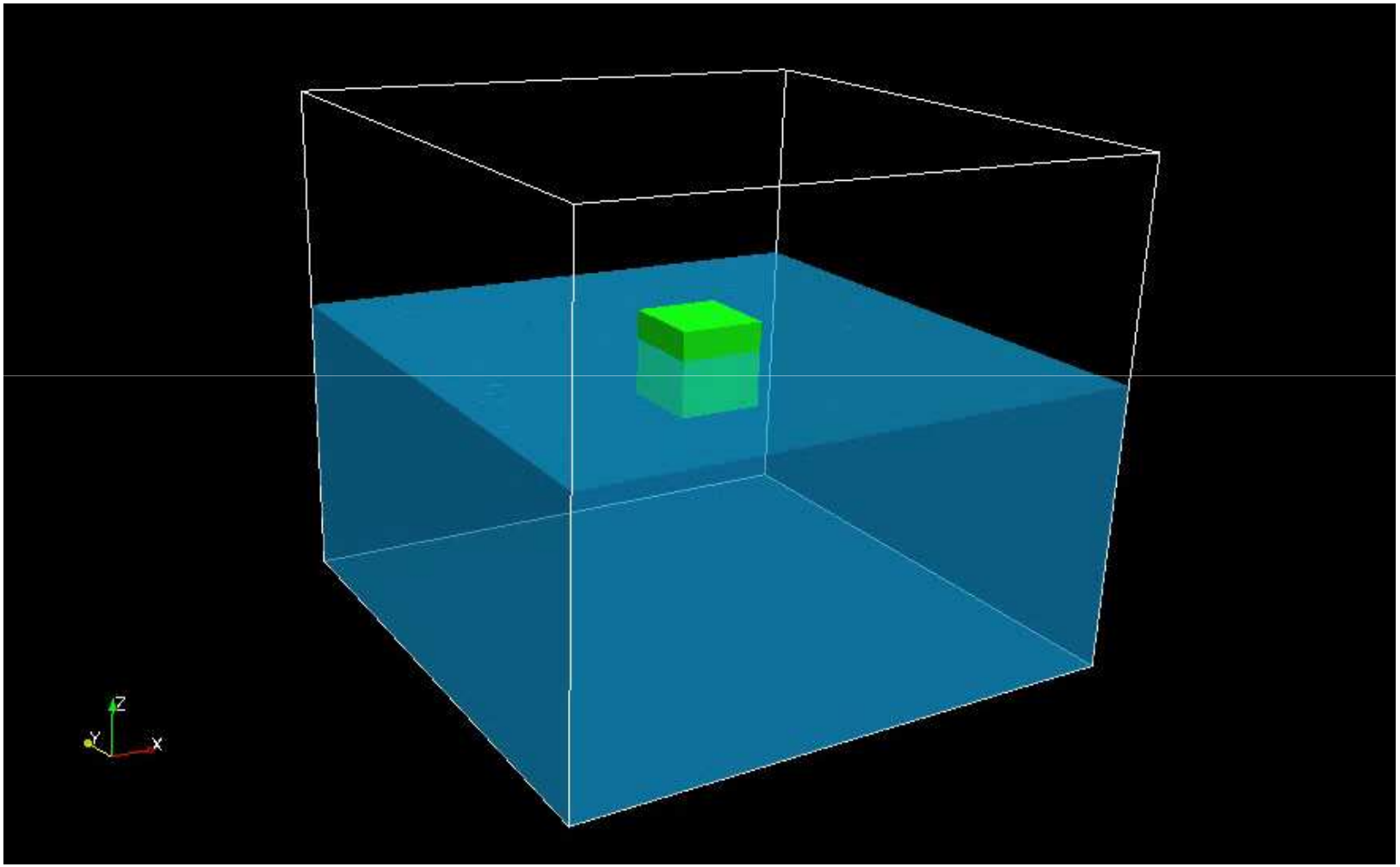
$$\frac{\partial u_a}{\partial x_a}=0$$



From Felippa, Park and Farhat (*CMAME*, 2001)

- **Velocity w is conveniently adjusted to Eulerian (**w**=0), far from moving object to Lagrangian (**w**=**u**) on the fluid-structure interface.**
- **Fluid is considered attached to the body.**
- **Need to solve extra-field equation to define mesh movement: our choice is to solve the Laplacian.**

Fluid-structure interaction with free-surface



FEM Discretization

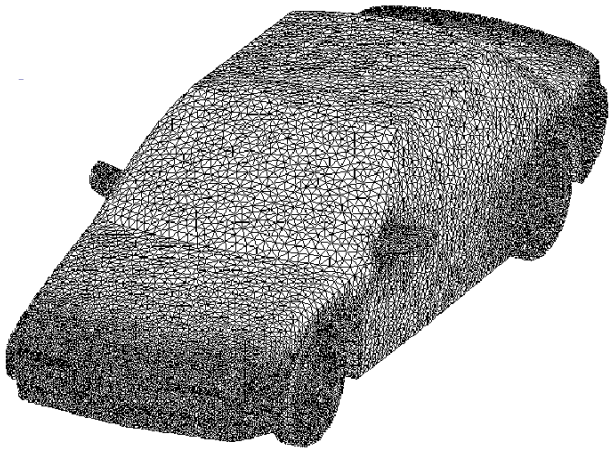
- **Good mathematical background and ability to handle complex geometries by using unstructured grids**

Let $\mathcal{E}$ be the set of all elements in the discretization of Ω in sub-domains Ω^e, $e = 1, 2, \ldots, n_{el}$, such as,

$$\overline{\Omega} = \cup_{e=1}^{n_{el}} \overline{\Omega}^e \quad ; \quad \cap_{e=1}^{n_{el}} \Omega^e = \emptyset$$

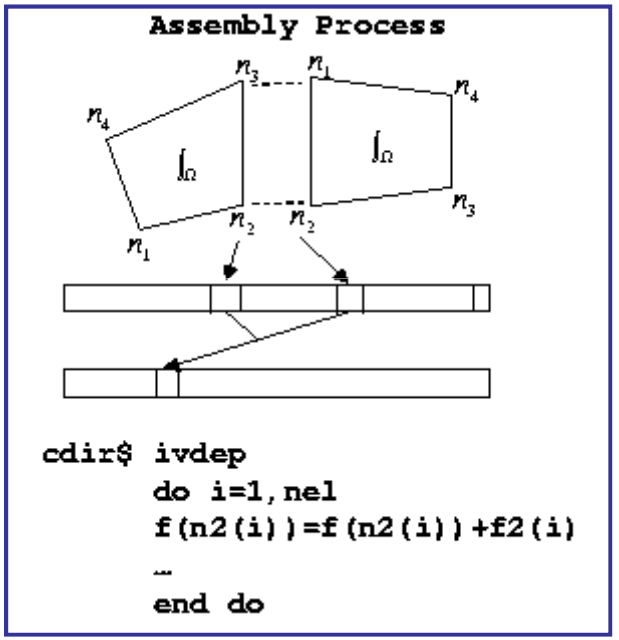
We associate to $\mathcal{E}$ finite dimension spaces

$$H^{kh} = \left\{ \phi^h / \phi^h \in C^o(\overline{\Omega}),\ \phi^h / \Omega^e \in P^k \right\} \quad k = 1, 2, \ldots$$



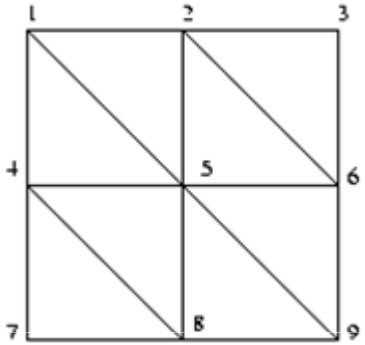
FEM Computing Issues

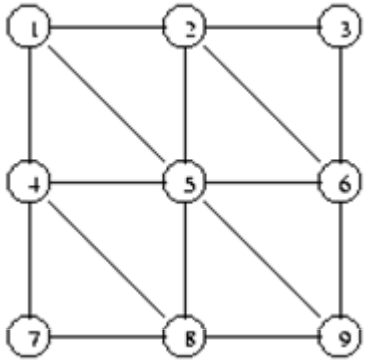
- **FEM is a unstructured grid method characterized by:**
  - Discontinuous data – no i-j-k addressing
  - Gather-scatter operations
  - Random memory access patterns
  - Data dependence
  - Minimize indirect addressing is a must
  - Memory complexity O(mesh parameters)



Mesh, Graphs and Sparse Matrices

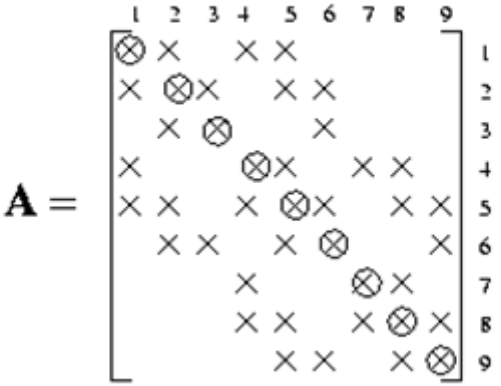
Mesh



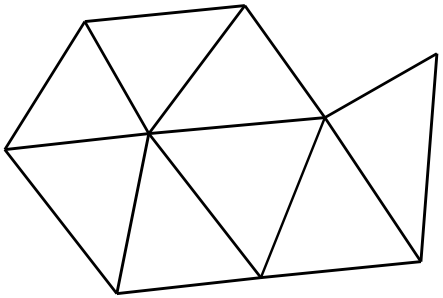


Graph

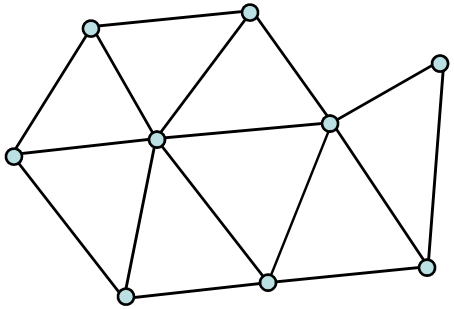
Sparse Matrix



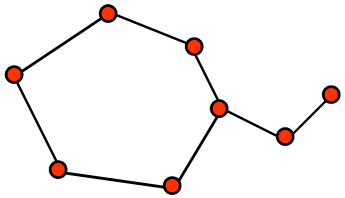
Graph Types Associated to Meshes



2D Mesh



Nodal Graph



Element Graph,
Adjacency Graph
or Dual Graph

Where to place the data: graph partitioning

- **NP-hard problem**
- **Type of partition depends on particular architecture**
  - Distributed memory: minimize edge-cuts → minimize communication
  - Shared memory → avoid data dependencies
- **Many problems we need to repartition on the fly: adaptivity, for instance**

Graph Partitioning for Distributed Memory Machines

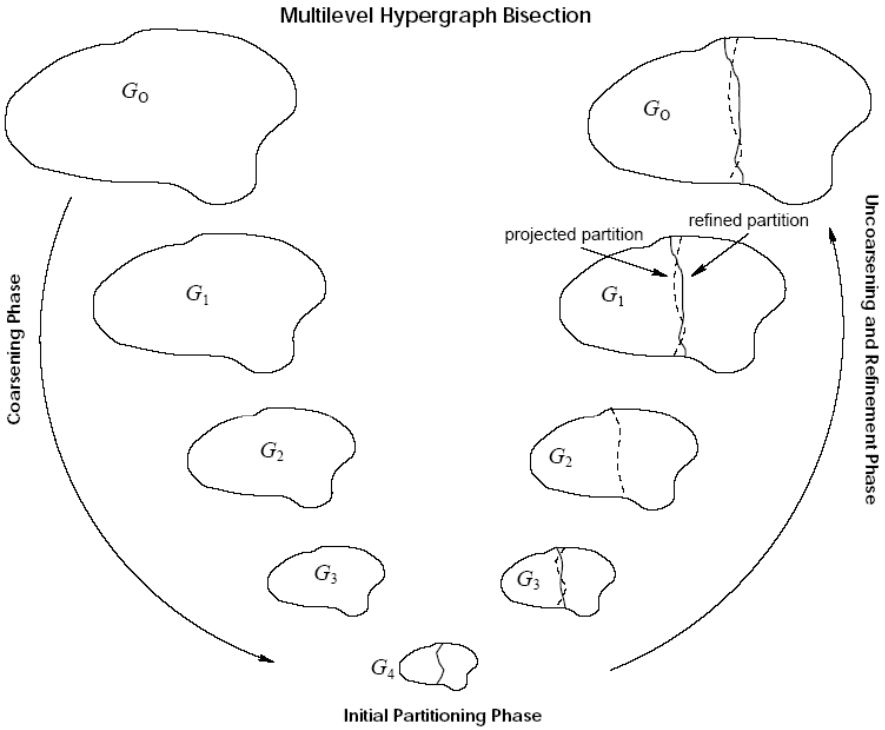
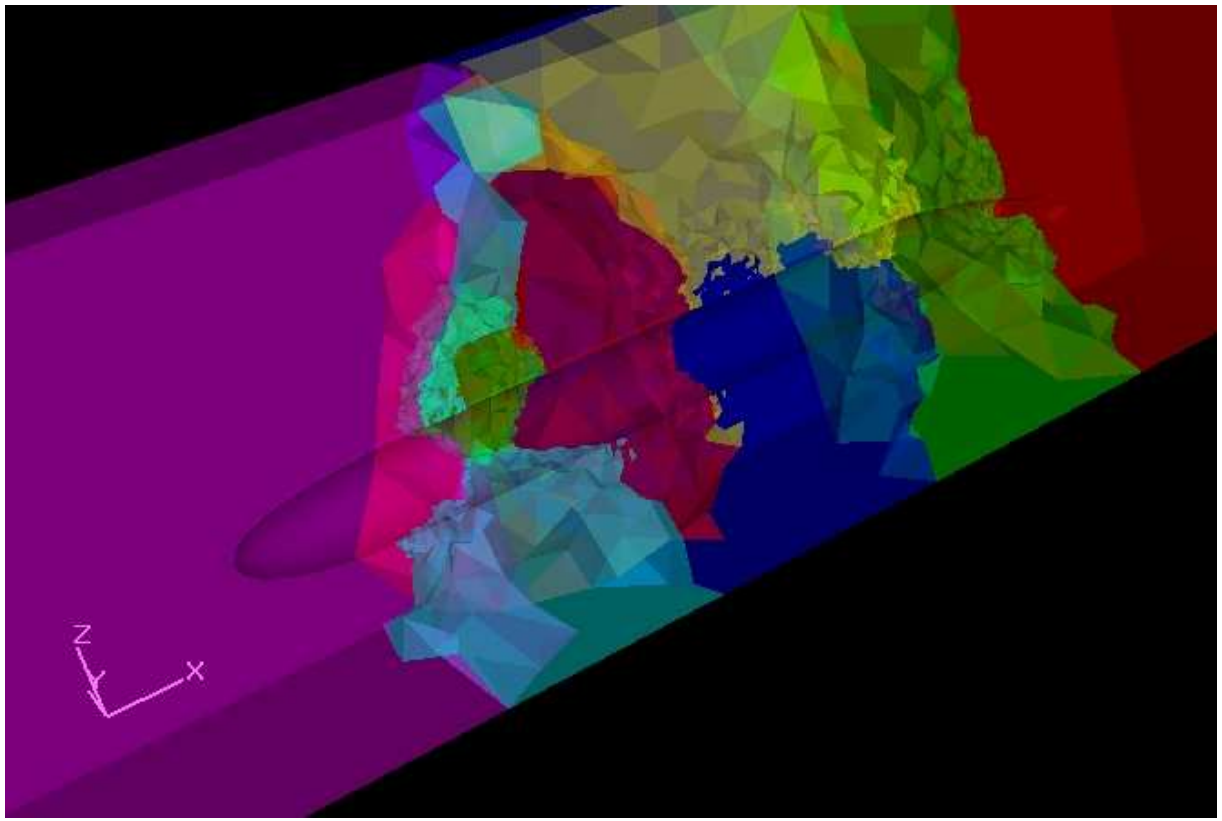


Figure 1.2. The various phases of the multilevel hypergraph bisection. During the coarsening phase, the size of the hypergraph is successively decreased; during the initial partitioning phase, a bisection of the smaller hypergraph is computed; and during the uncoarsening and refinement phase, the bisection is successively refined as it is projected to the larger hypergraphs. During the uncoarsening and refinement phase, the dashed lines indicate projected partitionings and dark solid lines indicate partitionings that were produced after refinement.

METIS: http://www-users.cs.umn.edu/~karypis/metis/index.html

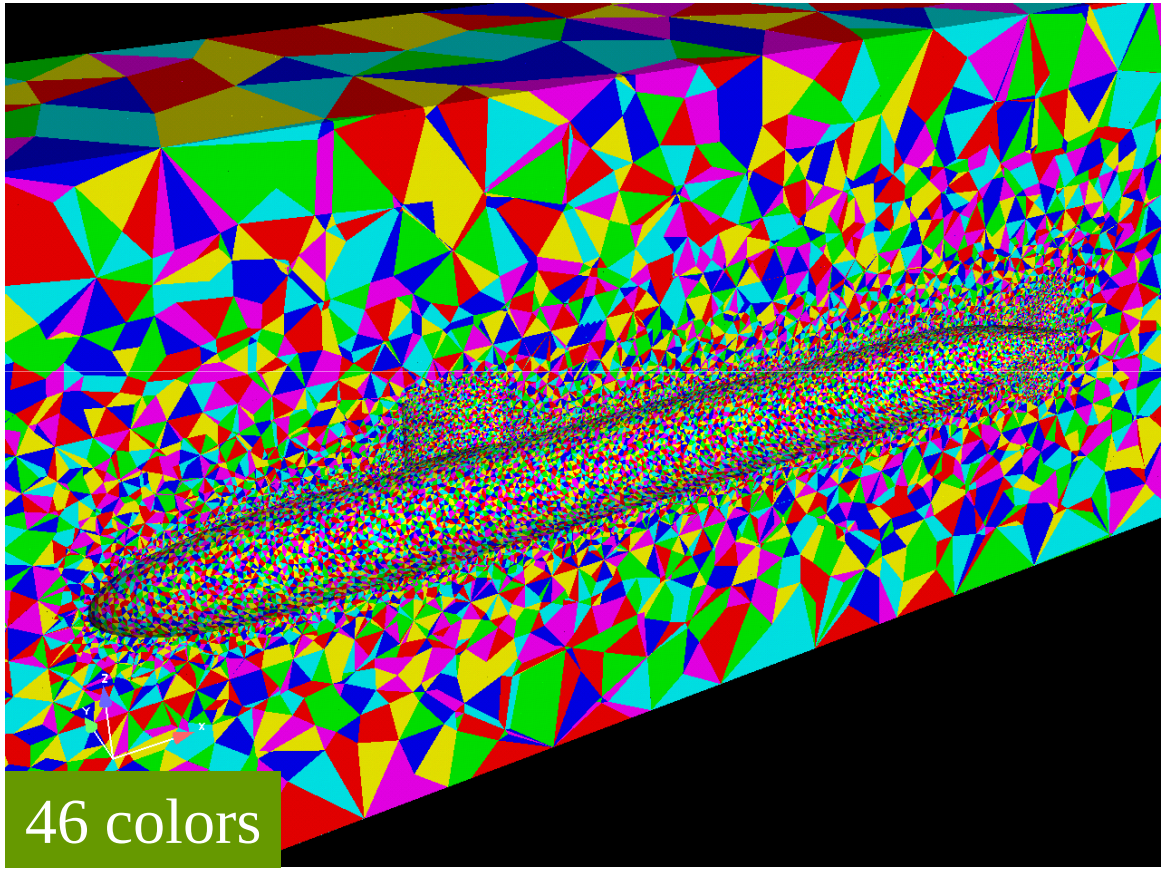
Example: METIS Partitioning for 8 procs



Graph Partitioning for Shared Memory

- Graph coloring or Mesh Coloring
- No adjacent node in the same color
- Applied either for node or element graphs
- Simple and fast greedy algorithm is generally enough

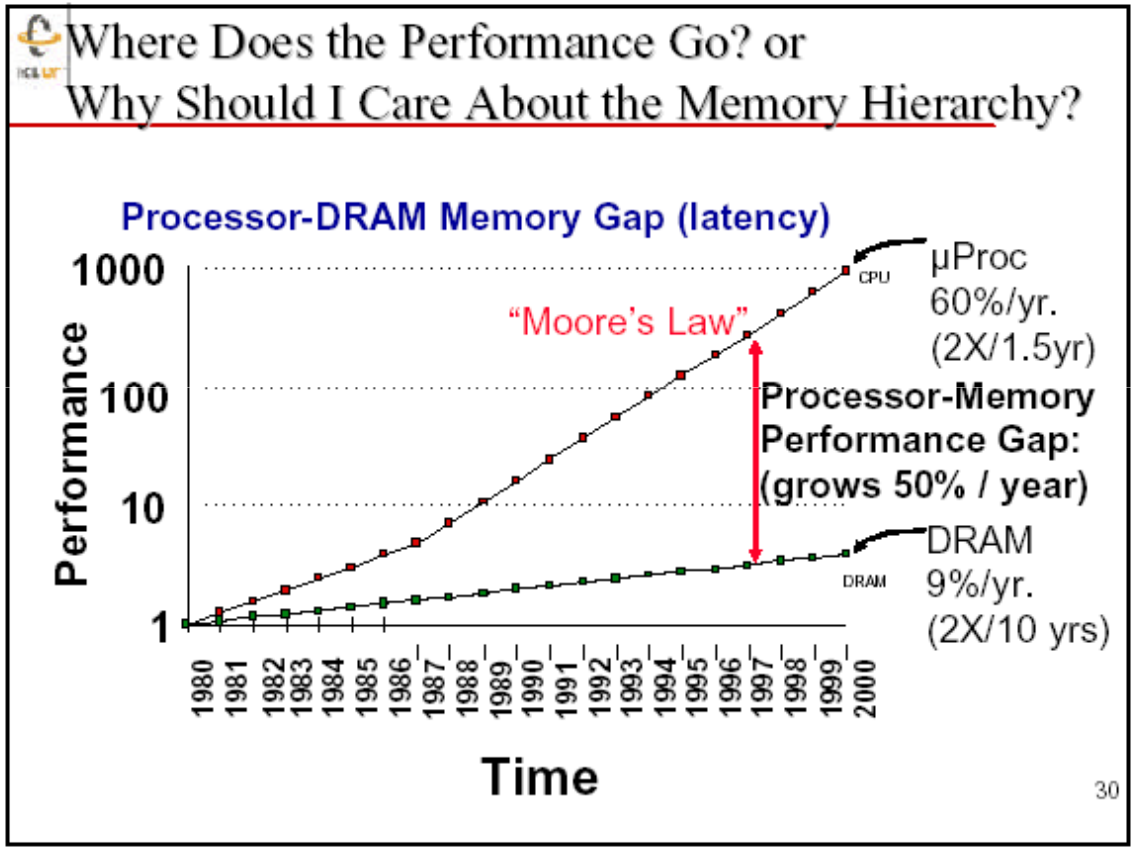
Example: Mesh coloring



Demonstration Problems

- **Where does my performance go? Effects of Memory Speed**
  - Los Angeles Class Submarine
- **Wireless Cluster experiment**
- **Rayleigh-Benard**
  - Algorithmic improvements
  - Pushing our limits
  - Parallel visualization

Effects of Memory Speed



From Jack Dongarra, 2002

Optimizing Computation and Memory Use

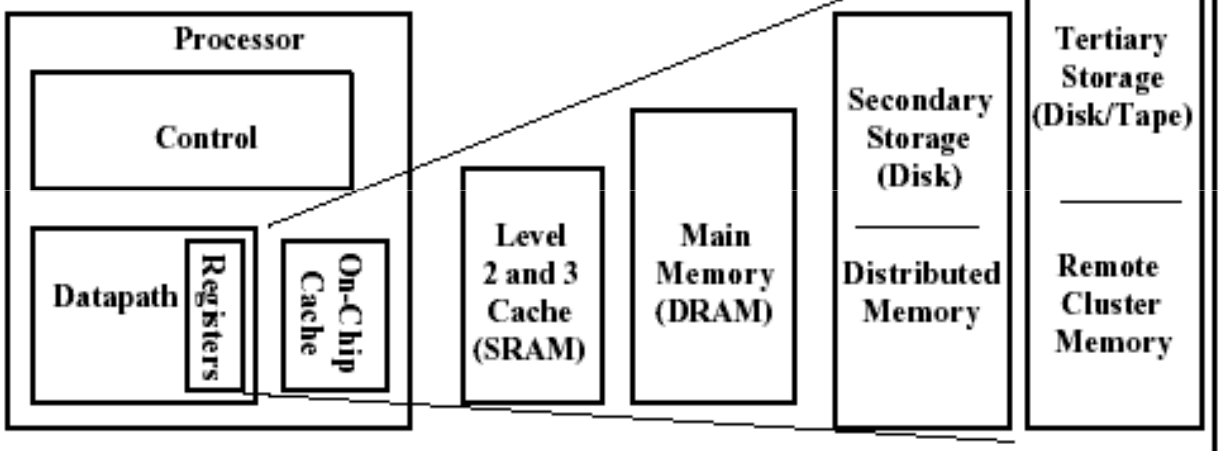
- **Computational optimizations**
  - Theoretical peak:(# fpus)*(flops/cycle) * Mhz
    - Pentium 4: (1 fpu)*(2 flops/cycle)*(2.53 Ghz) = 5060 MFLOP/s
- **Operations like:**
  - $\alpha = x^T y$: 2 operands (16 Bytes) needed for 2 flops; at 5060 Mflop/s will requires 5060 MW/s bandwidth
  - $y = \alpha x + y$: 3 operands (24 Bytes) needed for 2 flops; at 5060 Mflop/s will requires 7590 MW/s bandwidth
- **Memory optimization**
  - Theoretical peak: (bus width) * (bus speed)
    - Pentium 4: (32 bits)*(533 Mhz) = 2132 MB/s = 266 MW/s

31

From Jack Dongarra, 2002

Memory Hierarchy

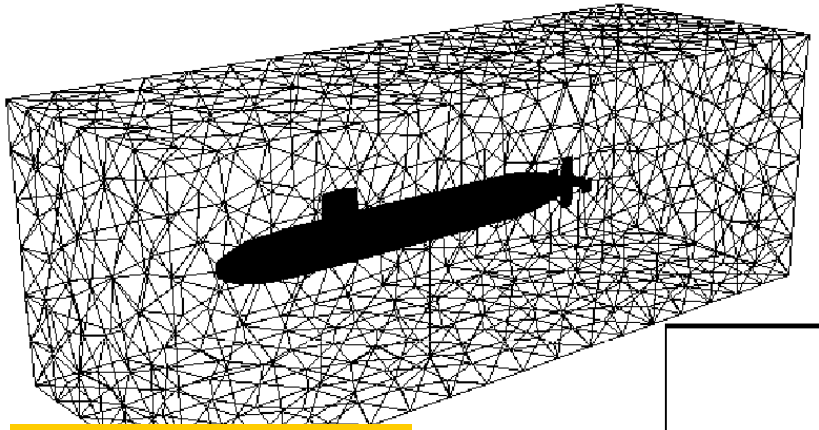
- By taking advantage of the principle of locality:
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology.



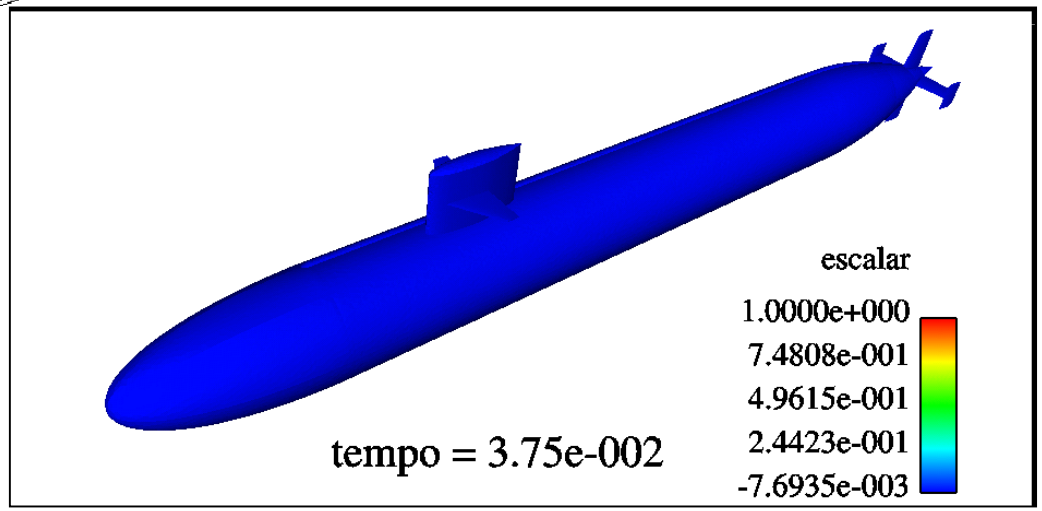
Speed (ns):	1s	10s	100s	10,000,000s (10s ms)	10,000,000,000s (10s sec)
Size (bytes):	100s	Ks	Ms	100,000 s (.1s ms)	10,000,000 s (10s ms)
				Gs	Ts

From Jack Dongarra, 2002

Los Angeles Class Submarine



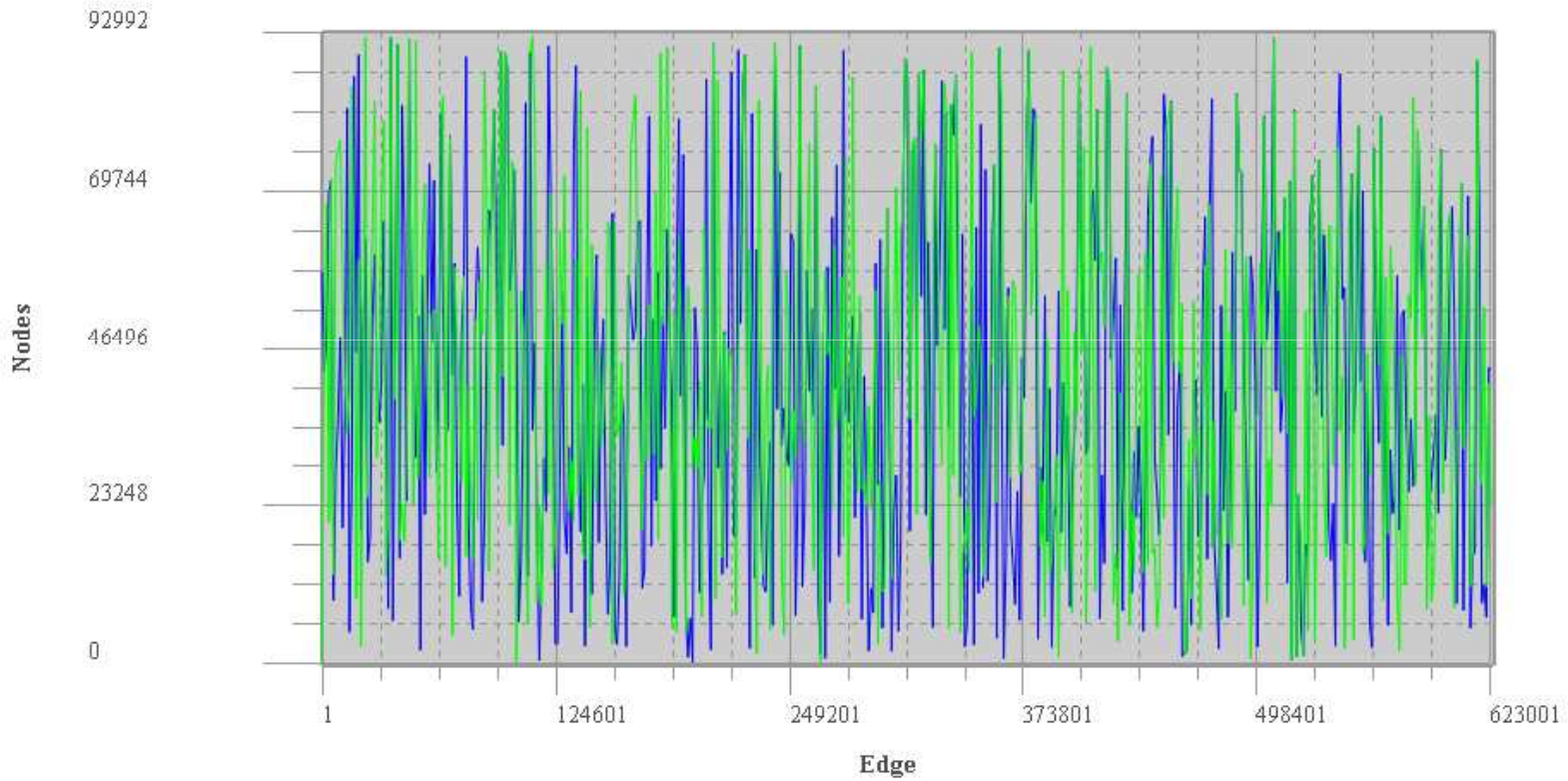
504,947 tetrahedra
92,564 points
623,003 edges



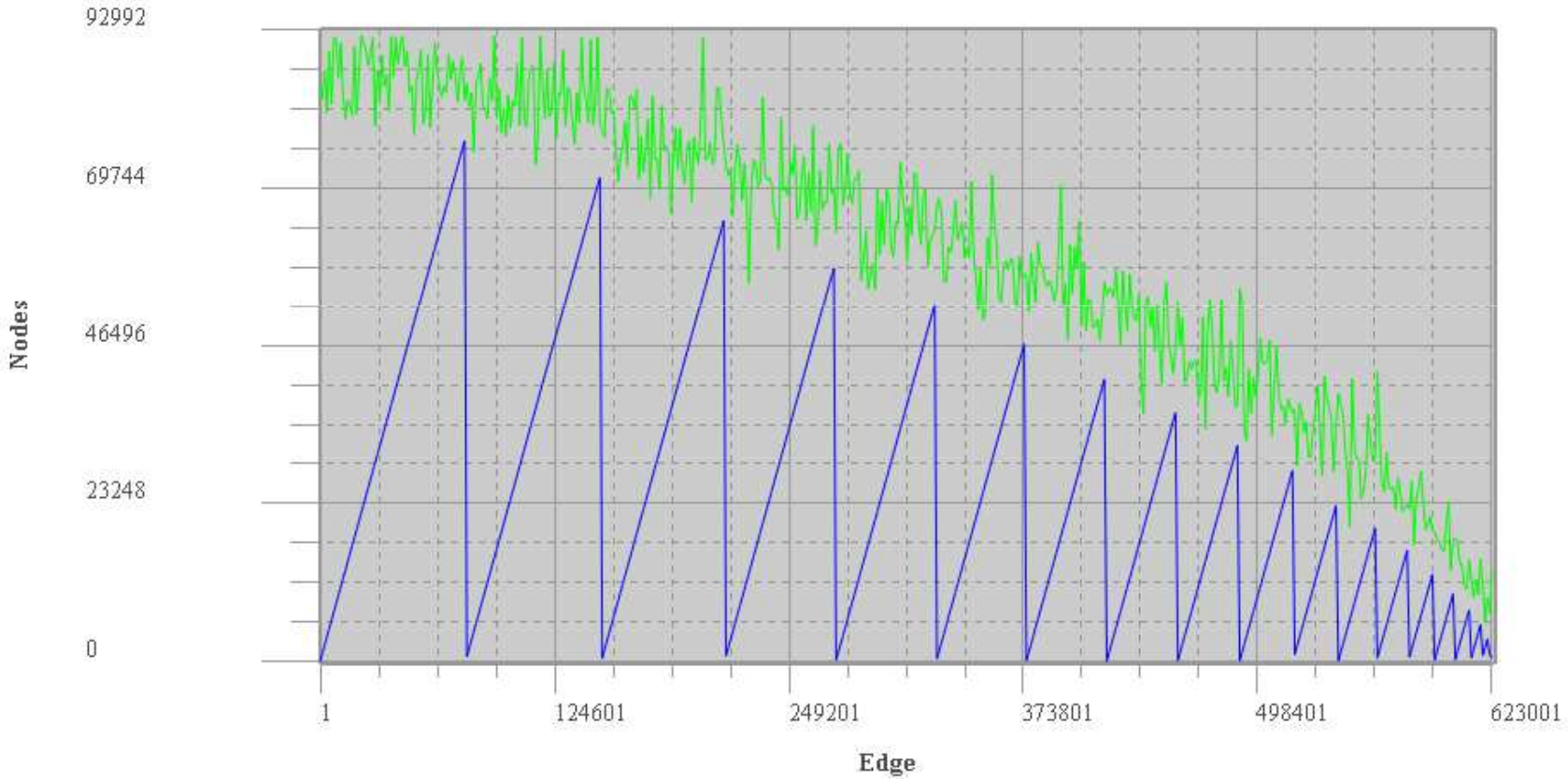
Reordering Graph

- Improve cache utilization
- Minimize data movement in memory hierarchy
- Improve data locality
- Minimize indirect addressing effects
- Reorder graph nodes and edges
- Maximize processor performance

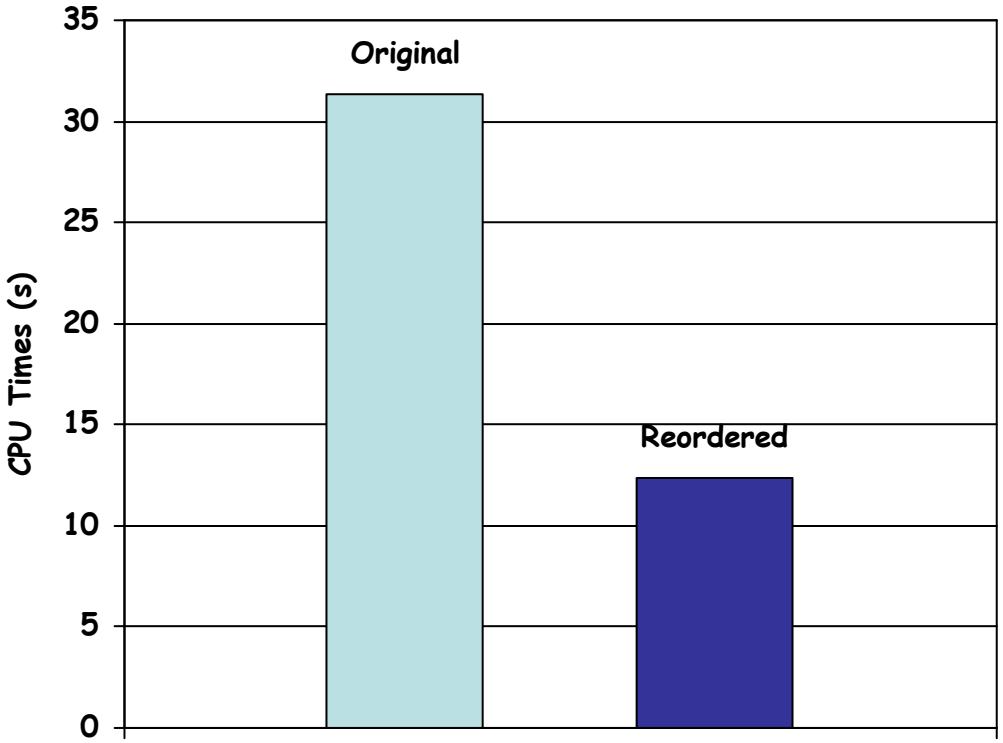
Original Order



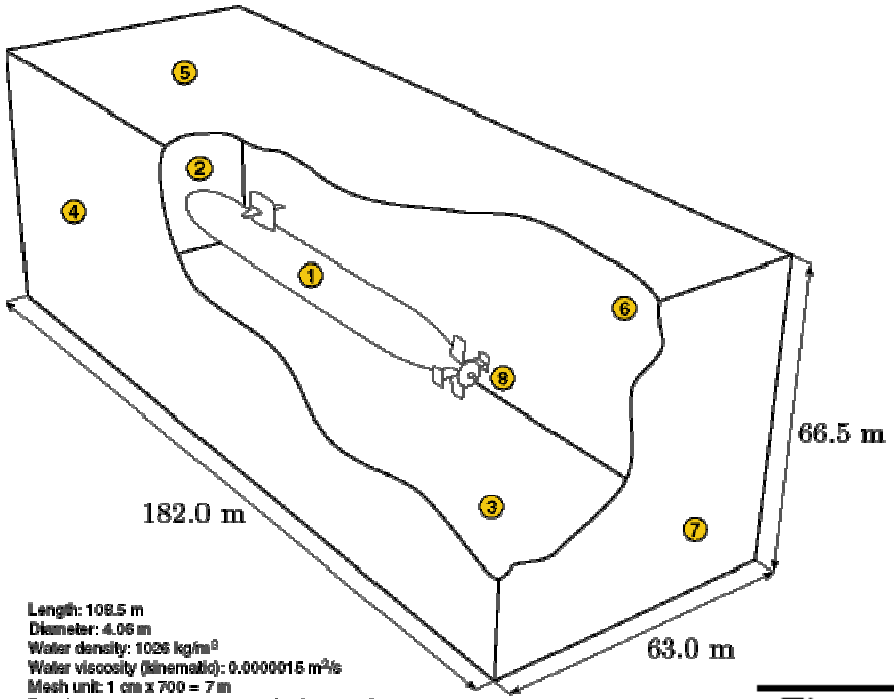
New Order


92992
69744
46496
23248
0
Nodes
1
124601
249201
373801
498401
623001
Edge

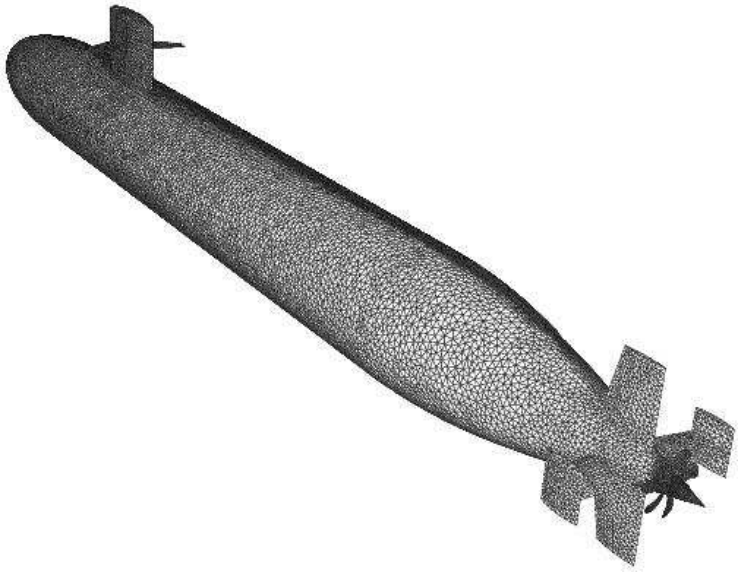
Solution times on PIV 1.8GHz



Los Angeles Class Submarine

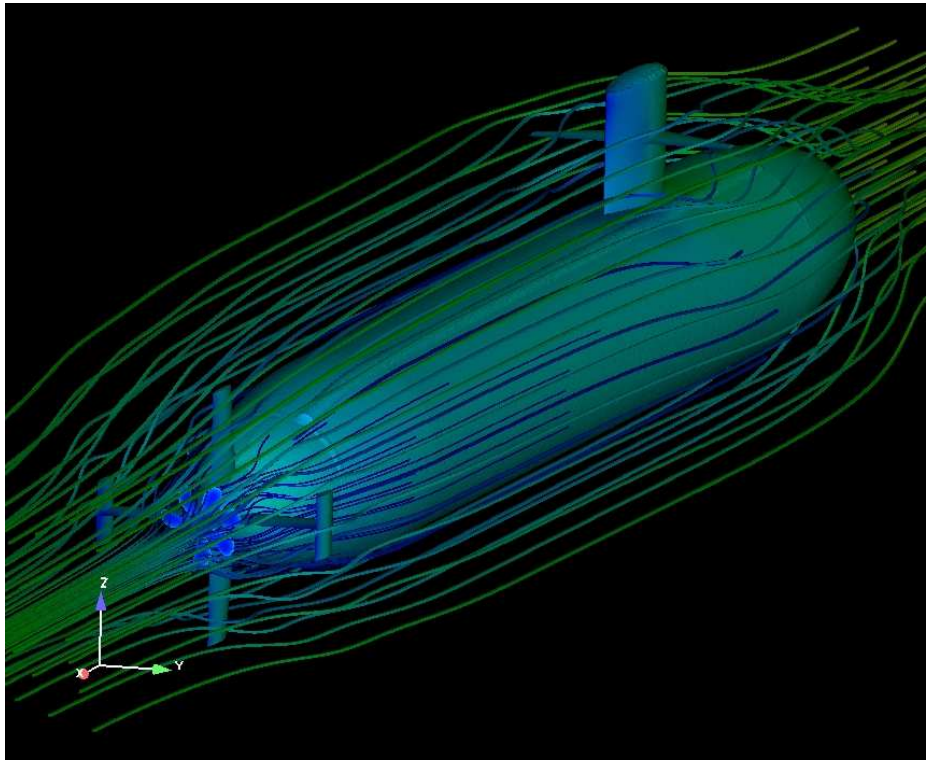


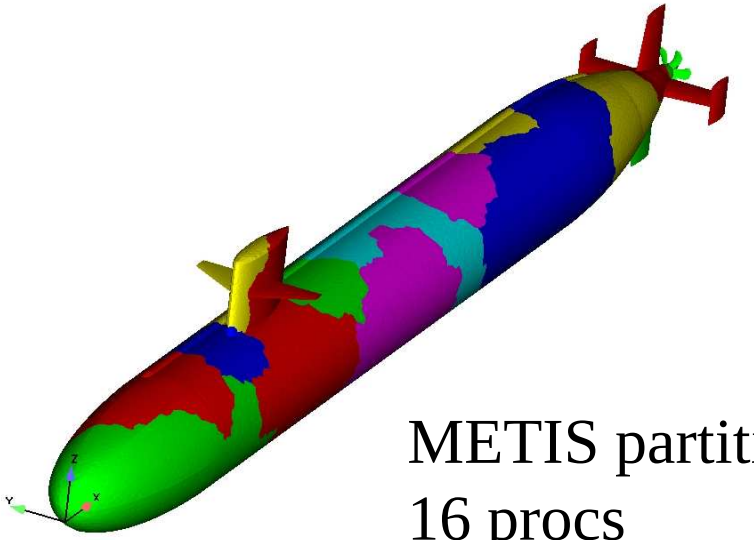
Length: 108.5 m
Diameter: 4.06 m
Water density: 1026 kg/m^3
Water viscosity (kinematic): 0.0000015 m^2/s
Mesh unit: 1 cm x 700 = 7 m
Box: (-4:+22.0, -4.5:+4.5, -4.5:+5.0) = (26, 9, 9.5)
Look at: (-4, -4.5, -4.5)
Look from: (+2 x, -2.5 x, +3.0 x) = (49, 27, 24)



Elements (tetrahedral)	817.608
Nodes	146.492
Edges	998.420
Velocity equations	336.621
Pressure equations	146.491

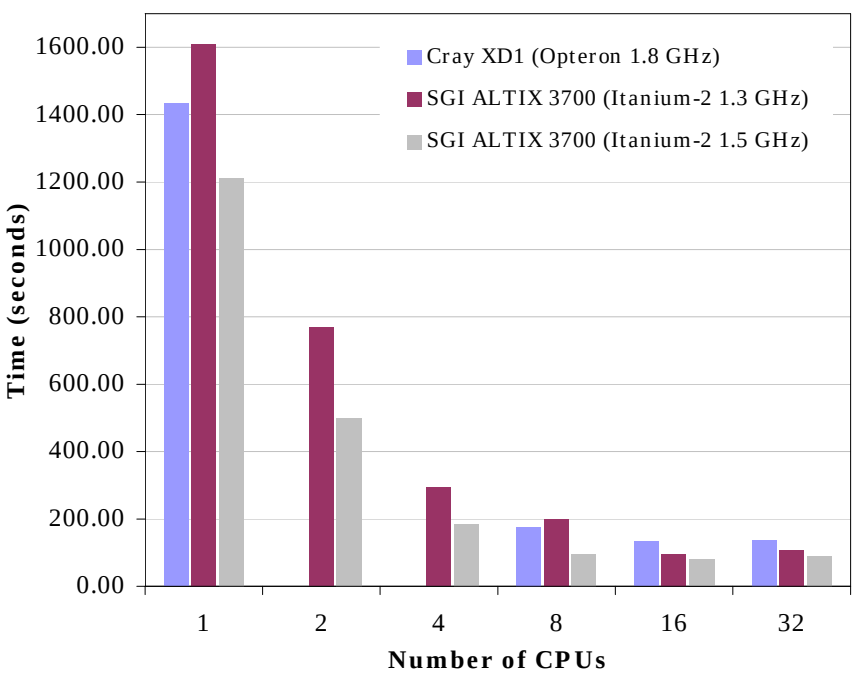
Flow around a Los Angeles Class Submarine

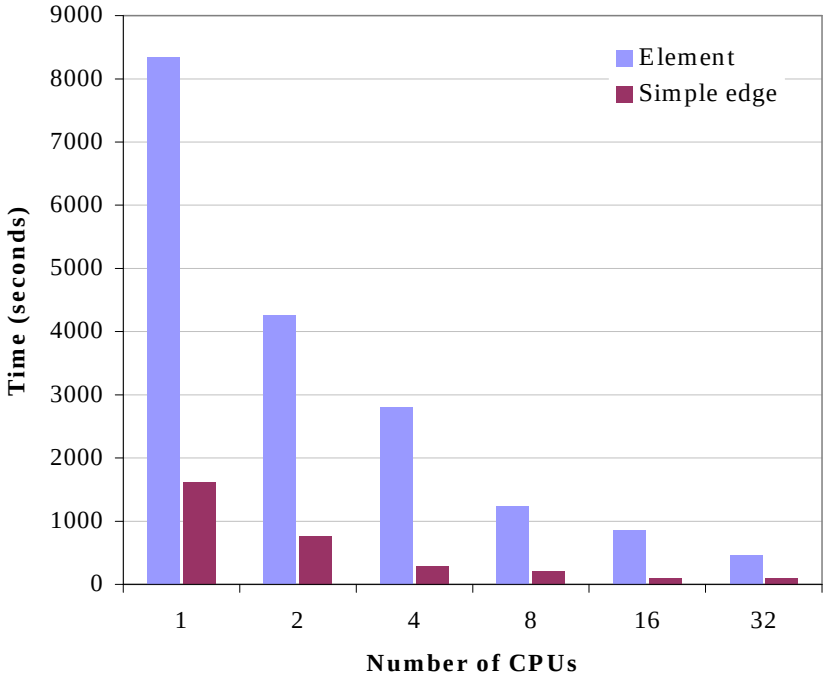




METIS partition
16 procs

Parallel Performance

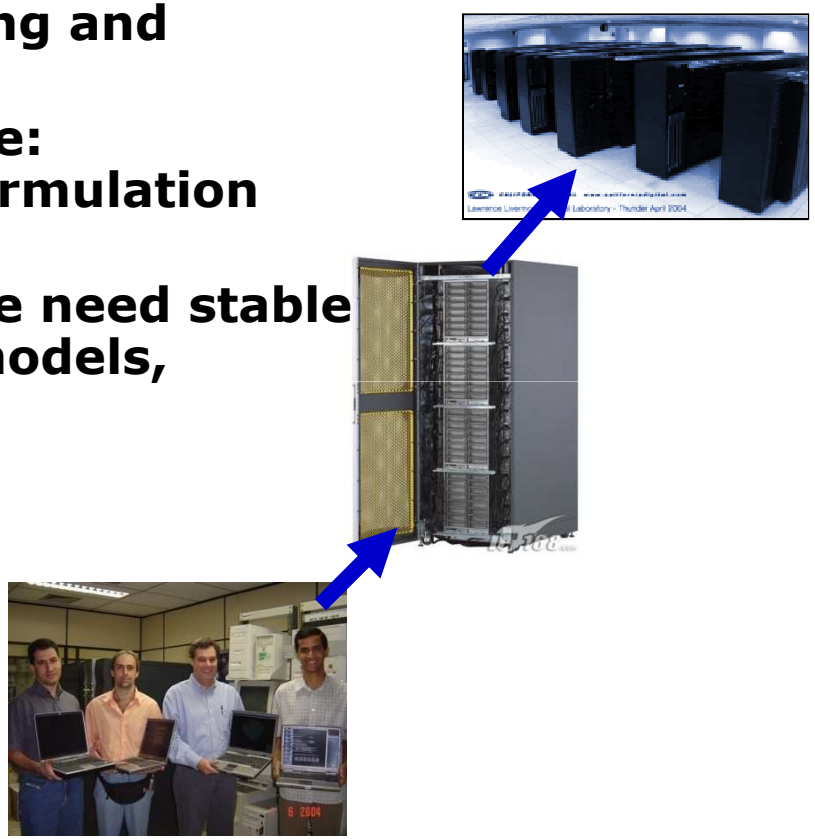




(Left) Message passing performance in SGI Altix and Cray XD1 – edge-based data structure, (Right) Data structure comparisons on SGI Altix (MPI).

What we have learned from the applications

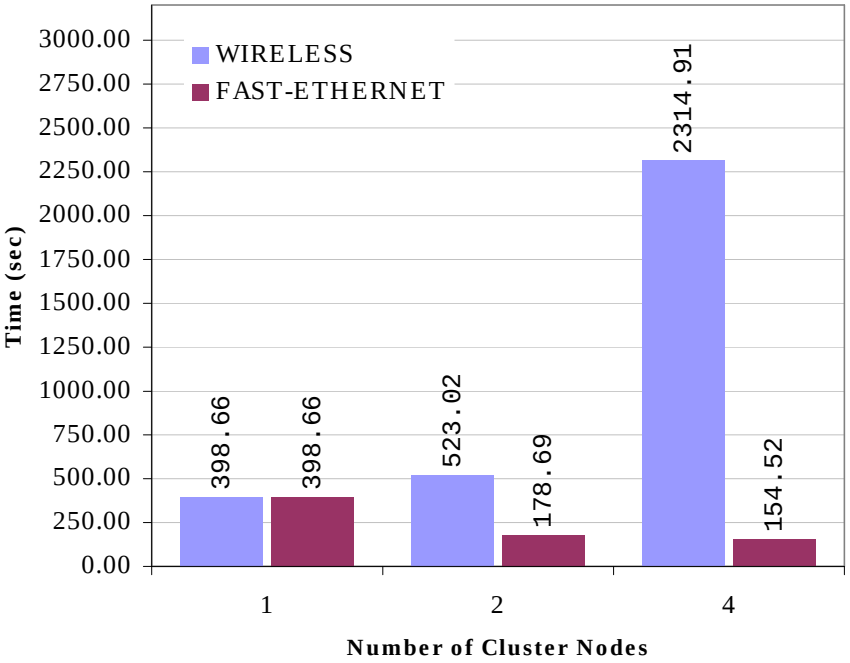
- HPC can transform engineering and science
- Porting a code is not the issue: performance needs code reformulation and new data structures
- Focus is not the hardware: we need stable and effective programming models, scaling upwards



Wireless Cluster Experiment

- mini-cluster formed by 4 laptops and a wireless/fast-ethernet network
- 2 Intel Centrino 1.6 GHz/512Mb, 1 Intel Centrino 1.3 GHz /512Mb and 1 Intel Pentium 4 2.4 GHz/512Mb
- Interconnection: Linksys Wireless-B Hub, IEEE 802.11b/2.4GHz/11Mbps or Fast-Ethernet 10/100Mbps network

Wireless Cluster Experiment



(Left) Minicluster mobile wireless/fast-ethernet,
(Right) Performance comparison between wireless and fast-ethernet networks.

Algorithmic Advances for Coupled Fluid Flow and Transport: Adaptive Time Stepping and Inexact Newton Method

MESH SIZE:

Elements.....: 93,925

Nodes.....: 21,384

Edges.....: 120,306

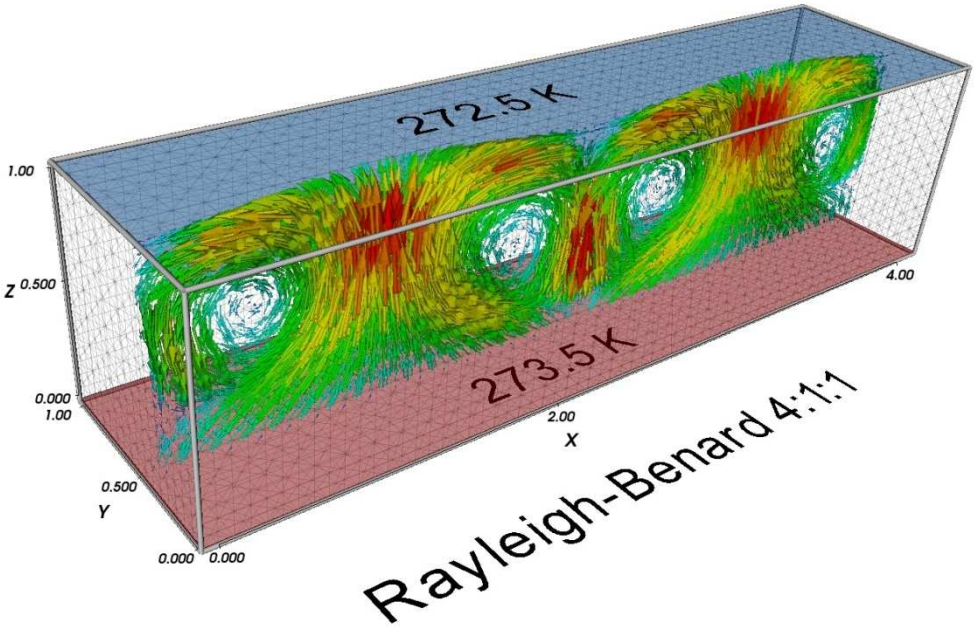
Flow equations.....: 70,536

Transport equations...: 19,008

DIMENSIONLESS NUMBERS:

Prandtl.....: 0.72

Rayleigh.....: 30,000



From Griebel M., Dornseifer T. and Neunhoeffer T, "Numerical Simulation in Fluid Dynamics: A Practical Introduction", SIAM, 97

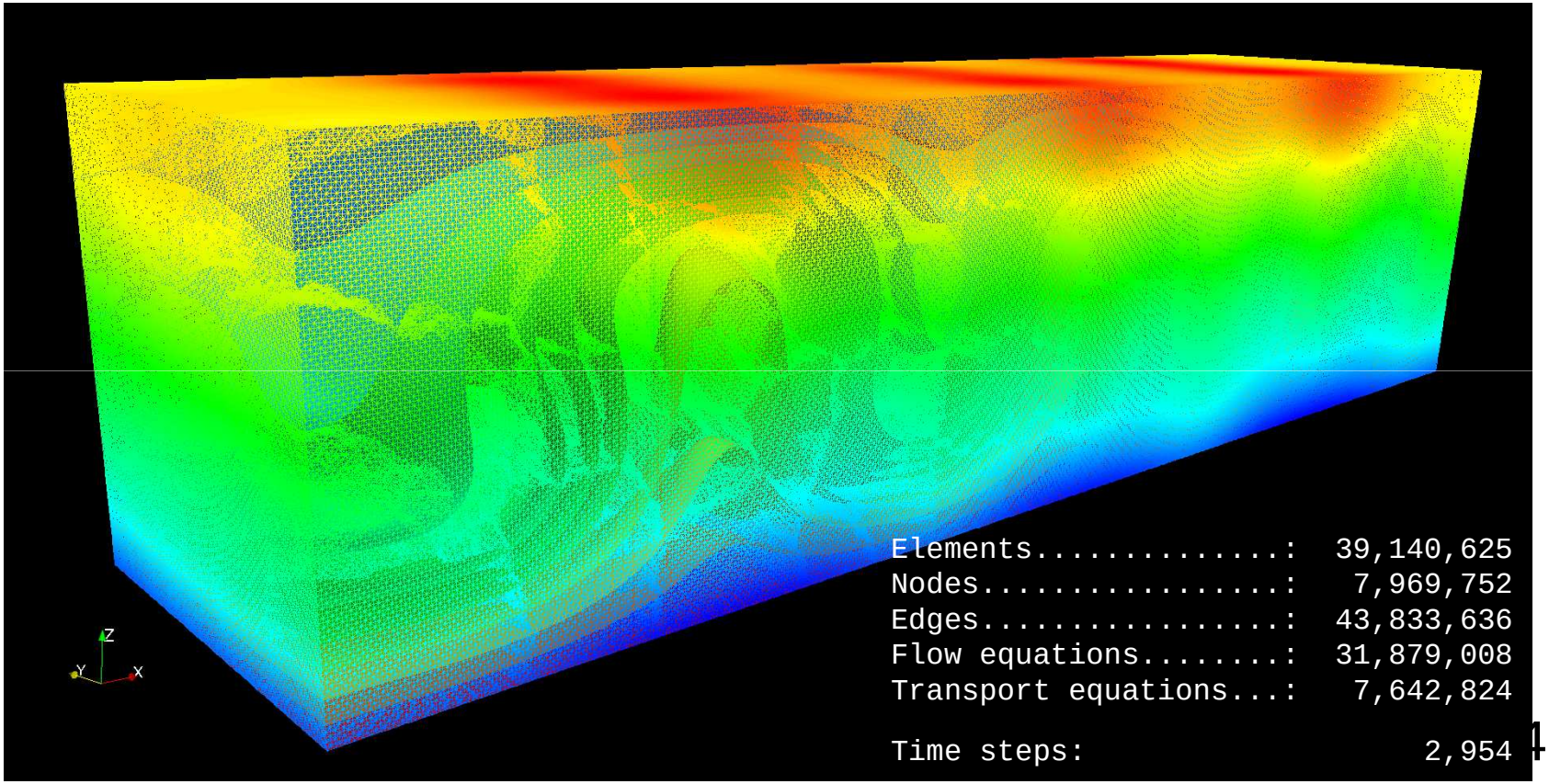
Performance Gains by Adaptive Time Stepping and Inexact Newton

Max IN Tol	CFL min	CFL max	SS time	time steps	wall time
0.001	2	10	0.5686	109	1110.49
		5	0.2410	90	855.42
		2	0.1829	171	1184.81
0.1	2	10	0.5635	108	803.89
		5	0.2418	90	637.71
		2	0.1840	172	1017.02
0.99	2	10	0.7436	142	1652.06
		5	0.2417	90	991.96
		2	0.1839	172	1501.97

SS Tolerance: 10^{-5} Transport P-GMRES Tol: 10^{-3} # Krylov Space Vectors: 25

Valli, Elias, Carey, Coutinho, PID adaptive control of incremental and arclength continuation in nonlinear applications, *Int. J. Numer. Meth. Fluids* 2009; 61:1181–1200

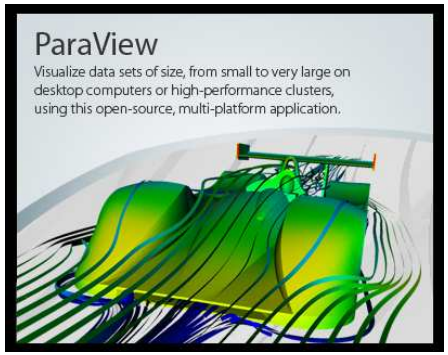
Pushing our limits: Rayleigh-Benard 4:1:1 on a fine mesh



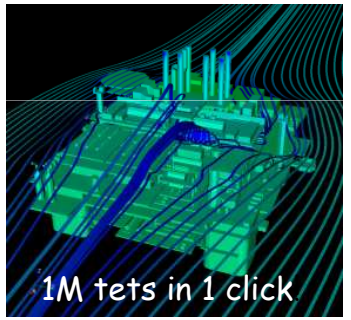
Mesh generation: SGI Altix 450 128GB RAM

Solver: SGI Altix ICE 8200, 128 cores, MPI

Software Tools

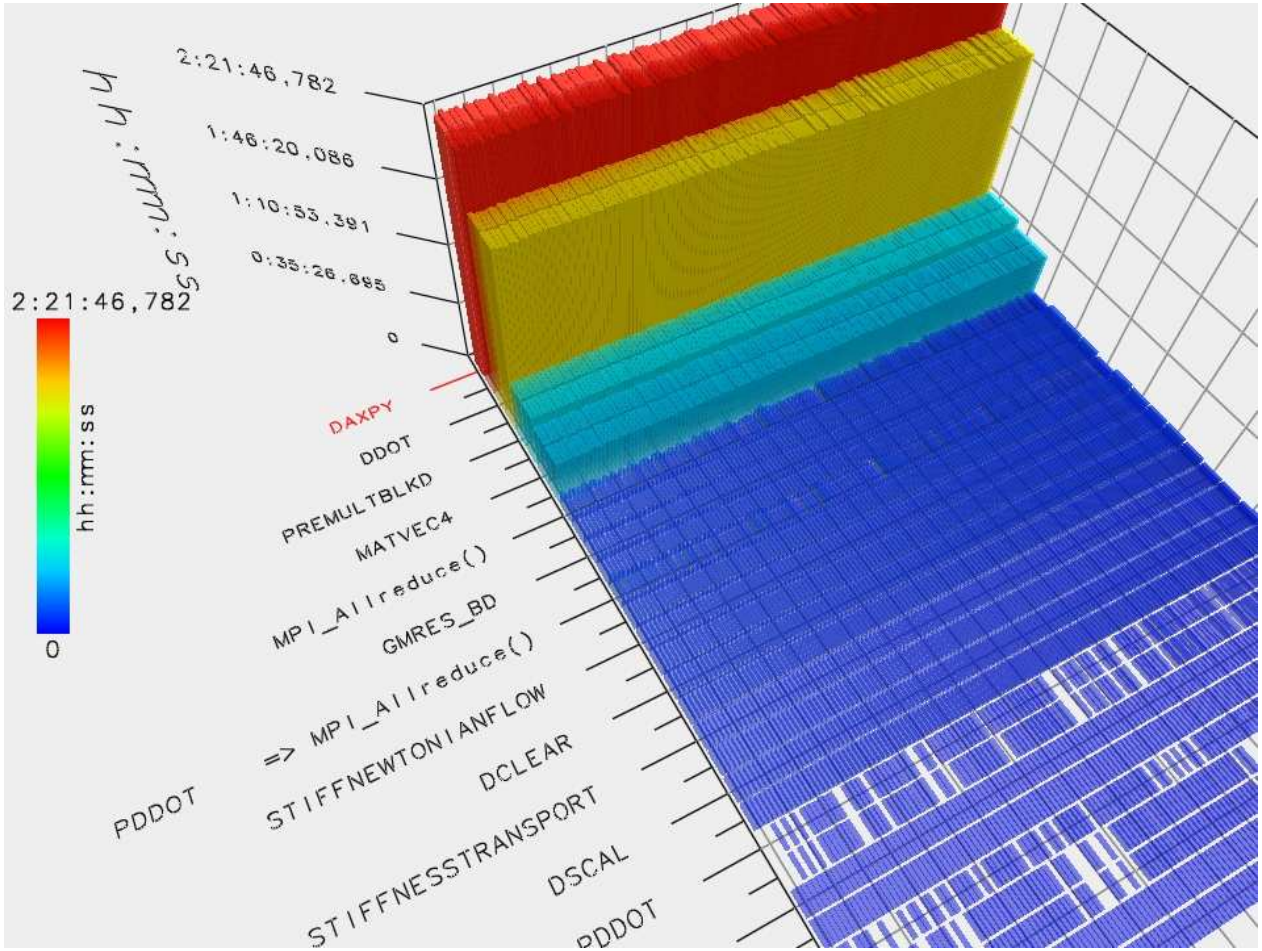


- **Visualization:**
  - http://www.paraview.org/
- **Mesh Generation: NETGEN**
  - http://www.hpfem.jku.at/netgen/



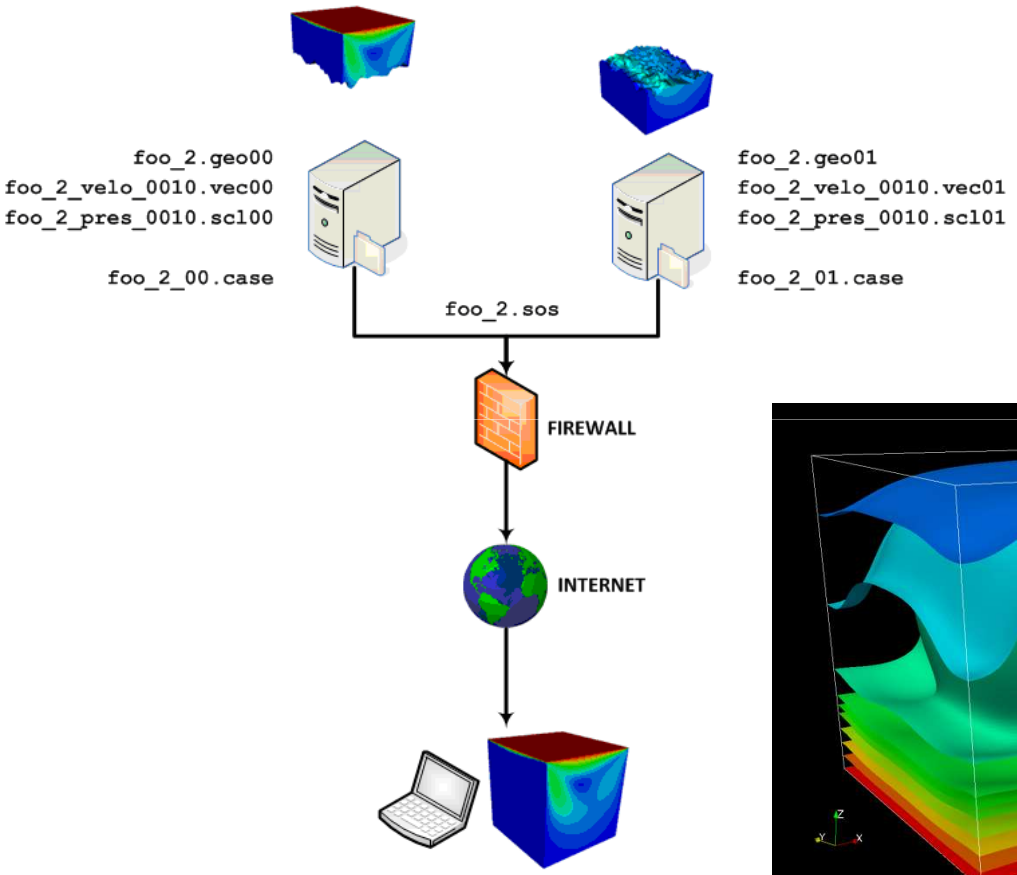
- **Code Tuning and Profiling**
  - TAO: Toolkit for Advanced Optimization
    - http://www-unix.mcs.anl.gov/tao/
  - TAU: Tuning and Analysis Utilities
    - http://www.cs.uoregon.edu/research/tau/home.php

TAU Parallel Profiling

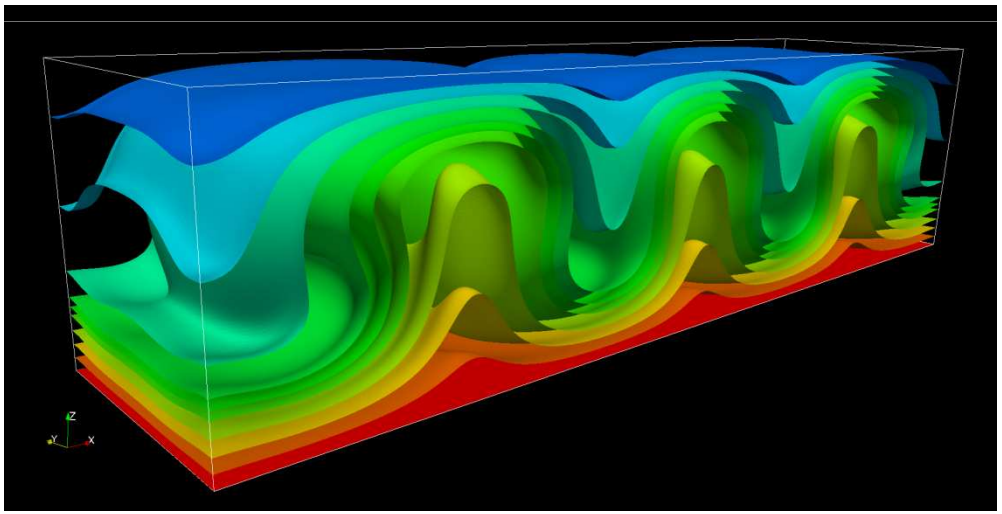


SGI Altix ICE 8200, 128 cores

Parallel Remote Visualization



SGI Altix ICE 8200 152 cores
Paraview: data in xdmf/hdf5 parallel formats
all memory available allocated



Steady-state temperature field

Final Remarks

- **Computational Engineering and Science changed the way we view engineering**
- **There is no general approach**
- **Integrated approach: HPC, Visualization, Storage and Communications**
- **Challenges:**
  - Managing complexity: programming models, data structures and computer architecture → performance
  - Understanding the results of a computation: visualization, data integration, knowledge extraction
  - Collaboration: grid, web, data security

Acknowledgements

- Collaborators: J. Alves, L. Landau, F. Rochinha, A. Loula (LNCC), S. Malta (LNCC), P. Sampaio (IEN), G. Carey (UT-Austin), T. Tezduyar (Rice)
- Students (and ex): M. Martins, M. Cunha, R. Sydenstricker, L. Catabriga, C. Dias, A. Valli, P. Hallak, I. Slobodcicov, P. Antunes, D. Souza, P. Sesini, A. Silva, R. Elias. A. Mendonça, W. Ney, J. Camata, A. Rossa
- Funding: CNPq, CAPES, FINEP/CTPetro, IBM, ANP, Petrobras
- Computational Resources: NACAD, TACC