Parallel Computing

Frank McKenna

UC Berkeley

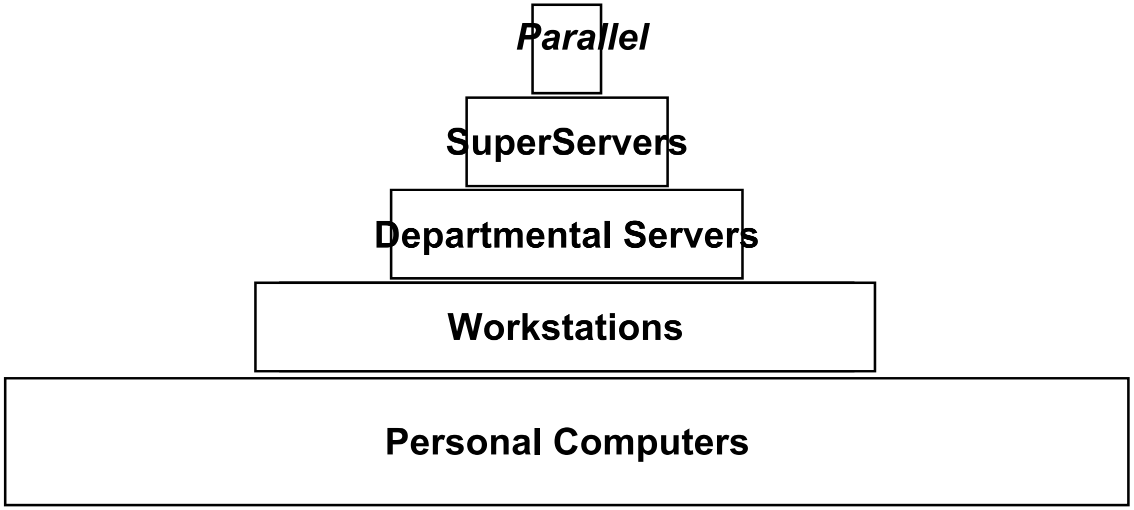
OpenSees Parallel Workshop
Berkeley, CA

Overview

- Introduction to Parallel Computers
- Parallel Programming Models
- Race Conditions and Deadlock Problems
- Performance Limits with Parallel Computing
- Writing Parallel Programs

Why should you care?

- It will save you time
- It will allow you to solver larger problems.
- It's here whether you like it or not!



Parallel

SuperServers

Departmental Servers

Workstations

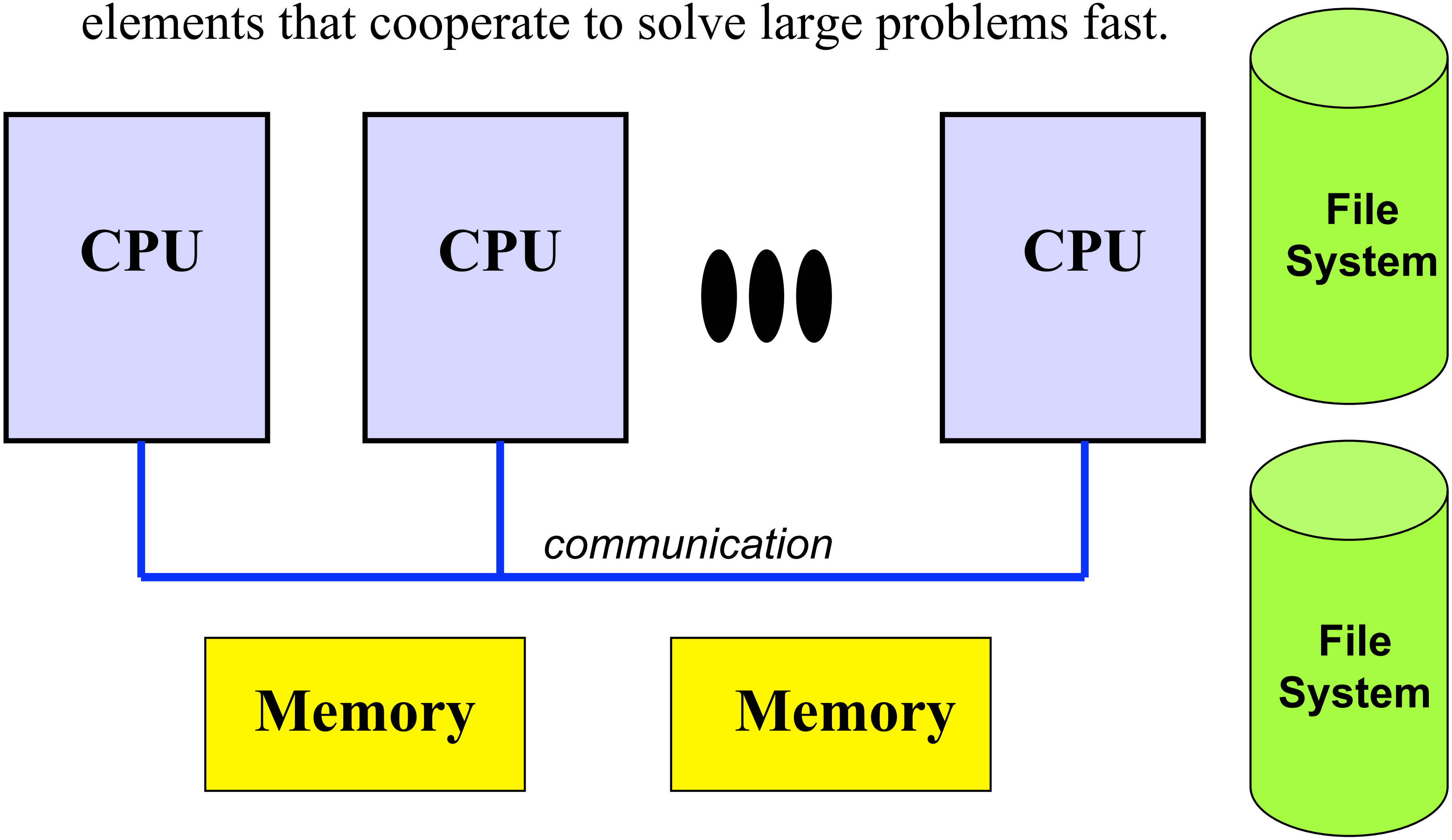
Personal Computers

2000

2008

What is a Parallel Computer?

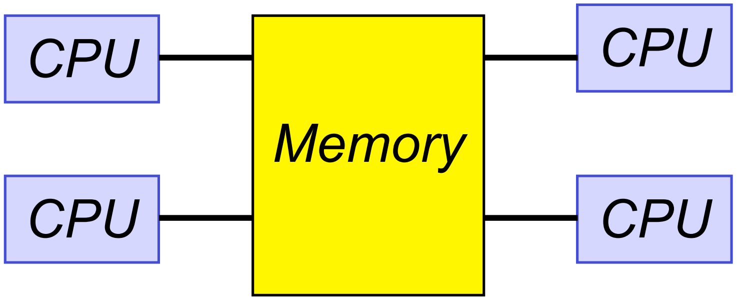
- A *parallel computer* is a collection of processing elements that cooperate to solve large problems fast.



Parallel Machine Classification

- Parallel machines are grouped into a number of types
  1. Scalar Computers (single processor system with pipelining, eg Pentium4)
  2. Parallel Vector Computers (pioneered by Cray)
  3. Shared Memory Multiprocessor
  4. Distributed Memory
     1. Distributed Memory MPPs (Massively Parallel System)
     2. Distributed Memory SMPs - Hybrid Systems
  5. Cluster Systems
  6. Grid

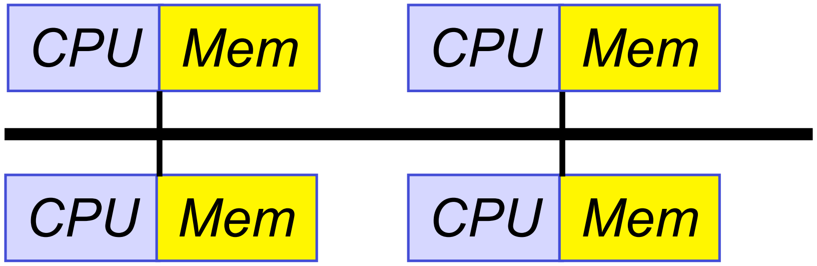
Shared Memory Parallel Computer



- Processors operate independently but all access the same memory.
- Changes by one processor to memory can be seen by all.
- Access to this memory can be uniform (UMA) or non-uniform (NUMA).
- Cache can be either shared or distributed. (multi-core typically shared L2). Cache hit is better if distributed but then the cache must be coherent.

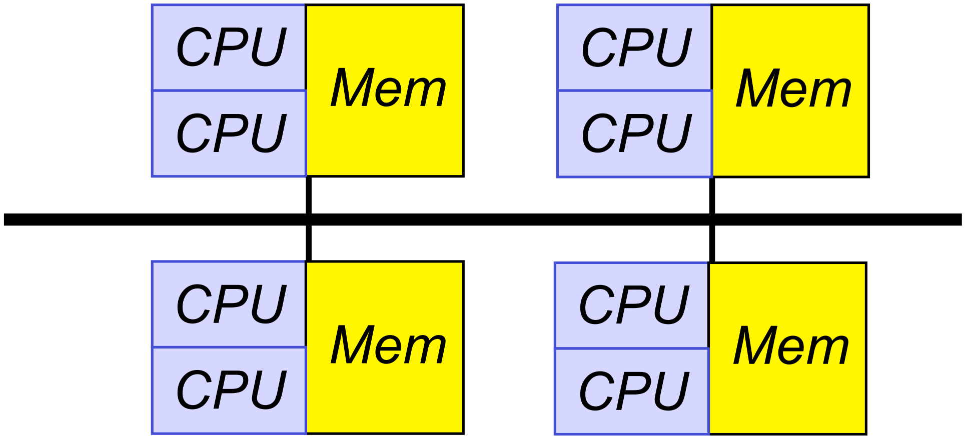
Distributed Memory

1. MPPs



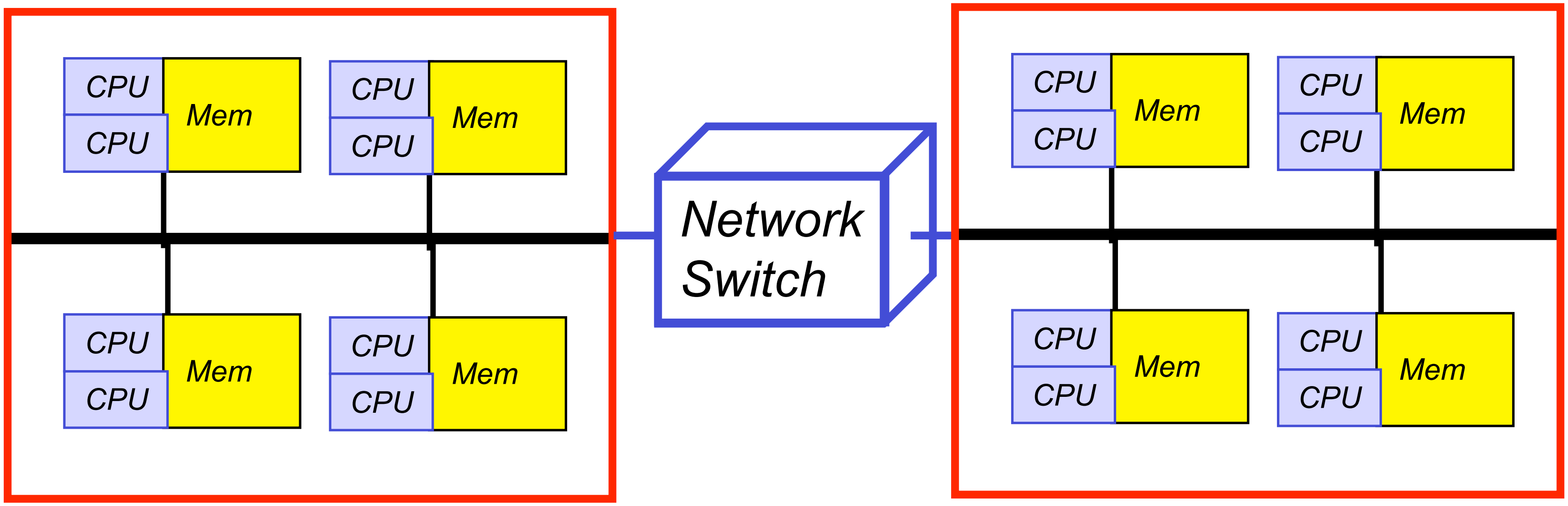
- Processors operate independently, each has it’s own private memory.
- Data must be shared using the communication network.

2. SMPs



•With new multi-core systems distributed memory processors now multi-core

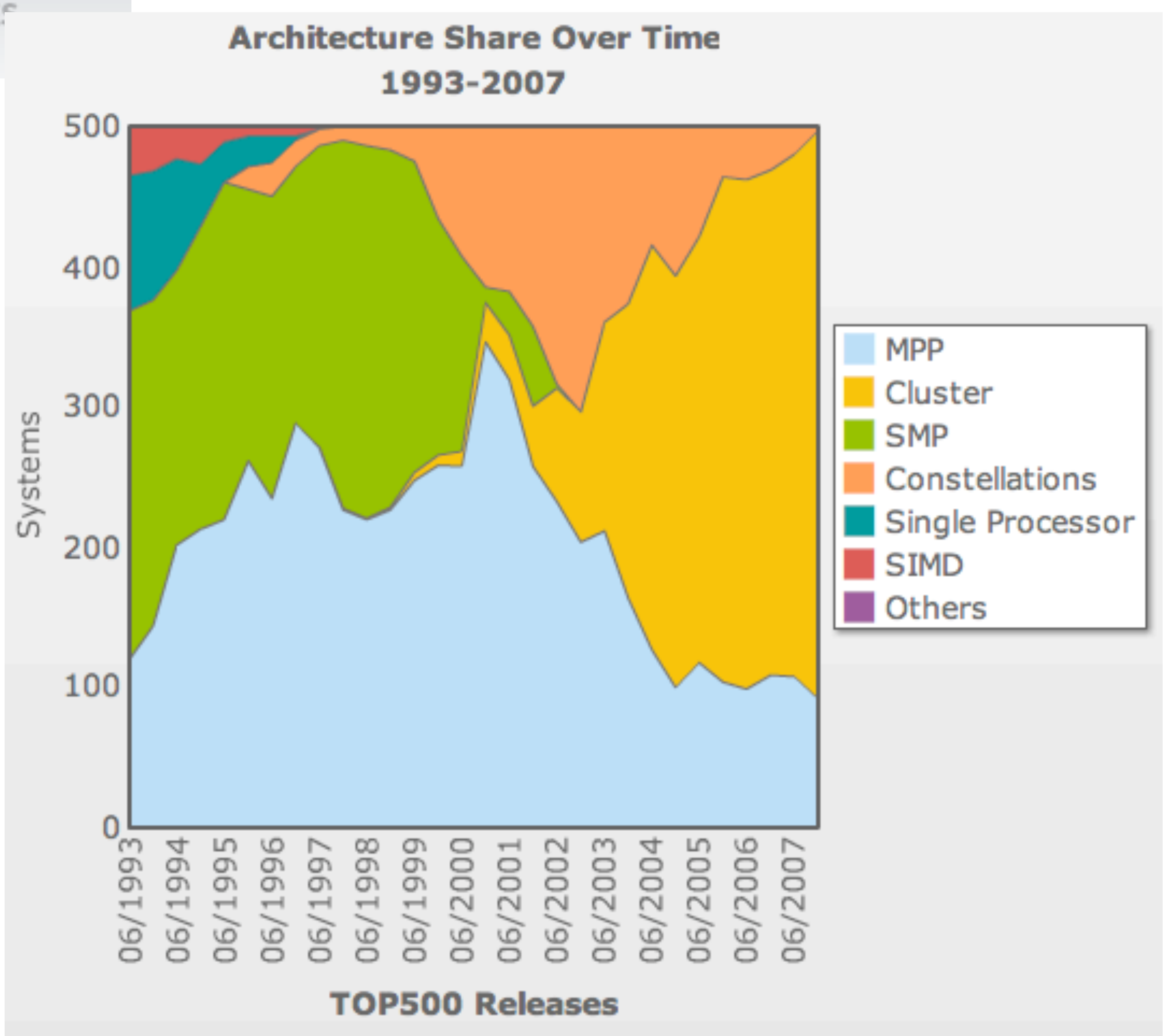
Cluster Systems



- Network Switch on faster machines are Gigabit Ethernet, Myrinet or Infiniband
- 80% of current top 500 (www.top500.org) are designated cluster machines.
- 20% of current Top 500 use intel quad core processors

TOP500 SUPERCOMPUTER SITES

www.Top500.org



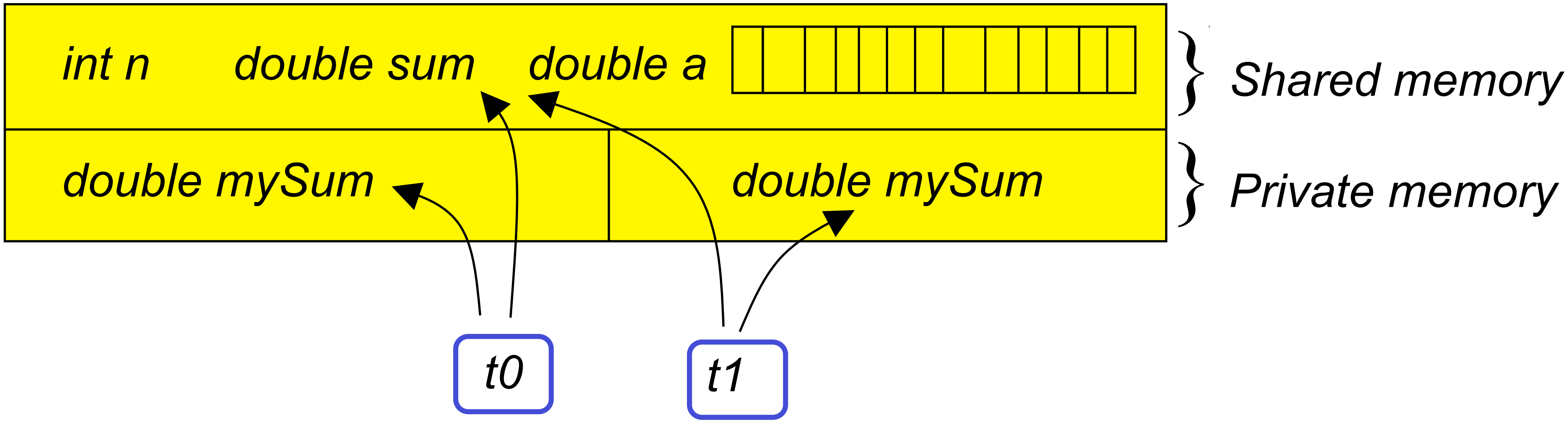
Parallel Programming Models

- A **Programming Model** provides an abstract conceptual view of the structure and operation of a computing system. A computer language and system libraries provide the programmer with this programming model.
- For parallel programming there are currently **2 dominant** models:

1. **Shared Memory**: The running program is viewed as a collection of processes (threads) each sharing it's virtual address space with the other processes. Access to shared data must be synchronized between processes to avoid race conditions.
2. **Message Passing**:The running program is viewed as a collection of independent communicating processes. Each process executes in it's own address space, has it's own unique identifier and is able to communicate with the other processes.

Shared Memory Programming

- •Program is a collection of threads.
- •Each thread has it's own private data, e.g. local stack.
- •Each thread has access to shared variables, e.g.static variables and heap.
- •Threads communicate by writing and reading shared variables.
- •Threads coordinate by **synchronizing** using mutex's (mutual exclusion objects) on shared variables.
- •Examples: Posix Threads (PThreads), OpenMP.



Race Conditions

- A **race condition** is a bug which occurs when two or more threads access the same memory location and the final result depends on the order of execution of the threads. Note: It only occurs when one of the threads is writing to the memory location.

```
static double *a;
static int n;
static double  sum = 0.0;
```

```
Thread 1

   double mySum = 0.0;
   for i = 0; i< n/2-1; i++
        mySum = mySum + a[i]
   lock(lock1)
   sum = sum + mySum;
   release(lock1)
```

```
Thread 2

   double mySum = 0.0;
   for i = n/2, i< n-1; i++
        mySum = mySum + a[i]
   lock(lock1)
   sum = sum + mySum;
   release(lock1)
```

- The use of a mutex’s (mutual exclusion locks)

Deadlock

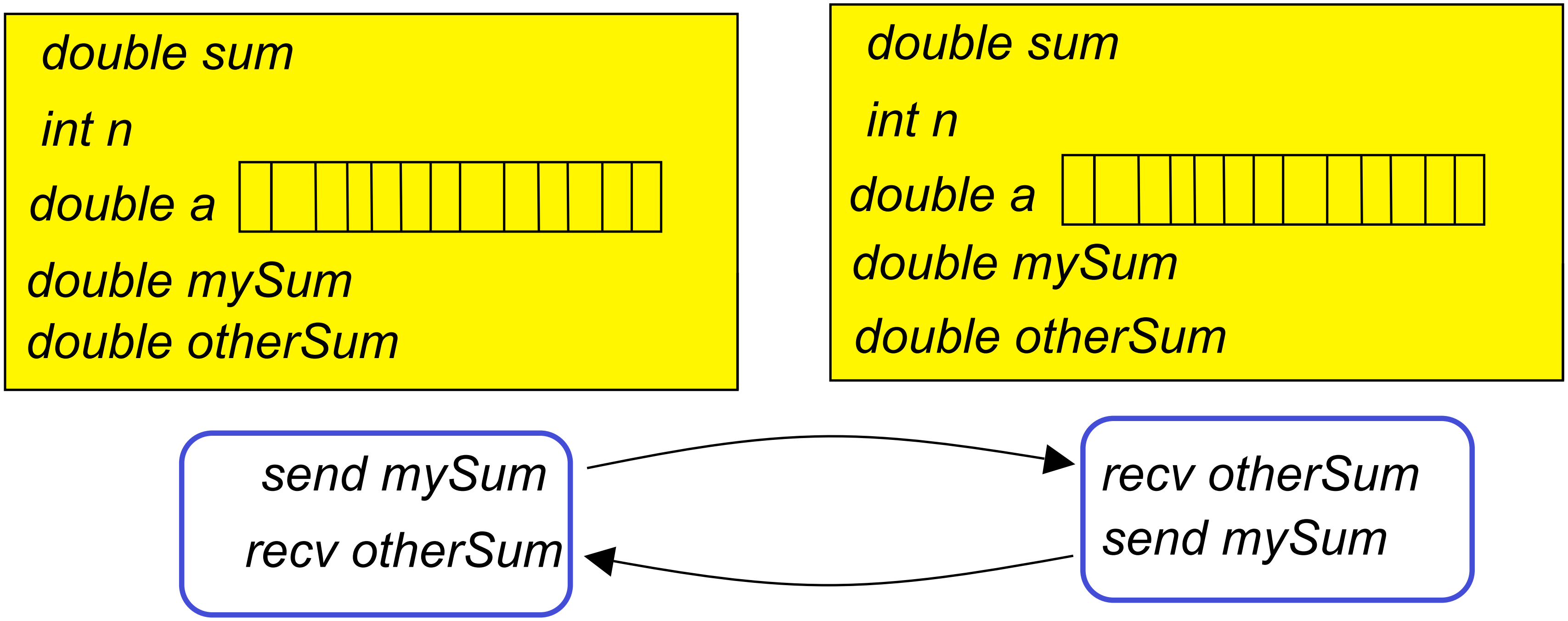
- The use of mutex's can lead to a condition of **deadlock**. Deadlock occurs when 2 or more threads are blocked forever waiting for a mutex locked by the other.

```
Thread 1

  lock (lock1);
  lock(lock2);
     …
  release(lock2);
  release(lock1);
```

```
Thread 2

  lock (lock2);
  lock(lock1);
     …
  release(lock1);
  release(lock2);
```

Message Passing

- Program is a collection of processes.
- Each process only has direct access to it’s own local memory.
- To share data processes must explicitly communicate the data.
- The processes communicate by use of send and recv pairs.
- Examples: MPI, pvm ,cmmd



double sum
int n
double a
double mySum
double otherSum

double sum
int n
double a
double mySum
double otherSum

send mySum
recv otherSum

recv otherSum
send mySum

Watch out for Deadlock

```
Process 1
   double sum, otherSum, mySum = 0;
   for i = 0; I< n/2-1; I++
        mySum = mySum + a[i]
   recv(2, otherSum);
   send(2, mySum);
   sum = mySum + otherSum;
```

```
Process 2
   double sum, otherSum, mySum = 0;
   for i = 0; I< n/2-1; I++
        mySum = mySum + a[i]
   recv(1, otherSum);
   send(1, mySum);
   sum = mySum + otherSum;
```

- send and recv are typically blocking, which can lead to deadlock if amount of data being sent/received is large relative to buffer.

```
Process 1
   double sum, otherSum, mySum = 0;
   for i = 0; i< n/2-1; i++
        mySum = mySum + a[i]
   send(2, mySum);
   recv(2, otherSum);
   sum = mySum + otherSum;
```

```
Process 2
   double sum, otherSum, mySum = 0;
   for i = n/2; I< n-1; i++
        mySum = mySum + a[i]
   recv(1, mySum);
   send(1, otherSum);
   sum = mySum + otherSum;
```

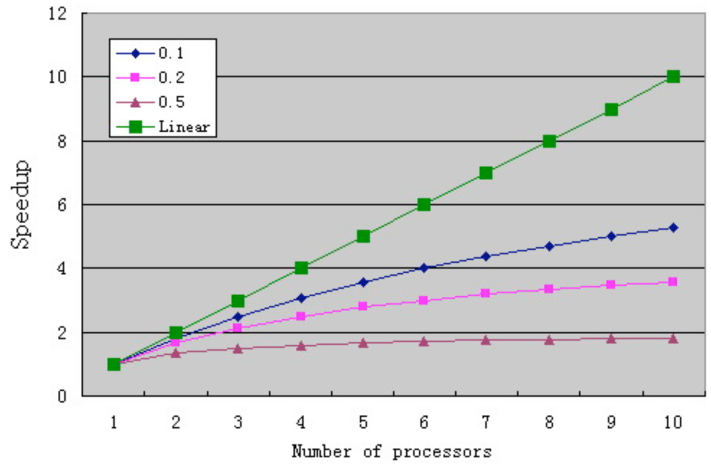
Speedup & Amdahl’s Law

$$speedup_{PC}(p) = \frac{Time(1)}{Time(p)}$$

$$Speedup_{PC} = \frac{T_1}{\alpha T_1 + \frac{(1-\alpha)T_1}{n}} \rightarrow \frac{1}{\alpha} \; as \; n \rightarrow \infty$$

Portion of sequential

# of processors



Writing Parallel Programs: Goal

1. To develop an application that will run **faster** on a parallel machine than it would on a sequential machine.
2. To develop an application that will run on a parallel machine that due to size limitations will not run on a sequential machine.

Writing Parallel Programs: Steps Involved

1. Break the computation to be performed into tasks (which can be based on function, data or both).
2. Assign the tasks to processes, identifying those tasks which can be executed concurrently.
3. Program it.
4. Compile, Test & Debug
5. Optimize
    1. Measure Performance
    2. Locate bottlenecks
    3. Improve them - may require new approach, i.e. back to step 1!

Writing Parallel Programs - Things to Remember

1. The task size is dependent on the parallel machine.
2. The fastest solution on a parallel machine may not be the same as the fastest solution on a sequential machine.
3. A solution that works good on one machine may not work well on another.

Writing parallel programs is a lot harder than writing sequential programs.

Reasons for Poor Performance in Parallel Programs

- Poor Single Processor Performance.
  - Typically due to Memory Performance.
- Too Much Parallel Overhead.
  - Synchronization and Communication.
- Load Imbalance
  - Differing Amounts of Work Assigned to Processors
  - Different Speed of Processors

Solutions for Load Imbalance

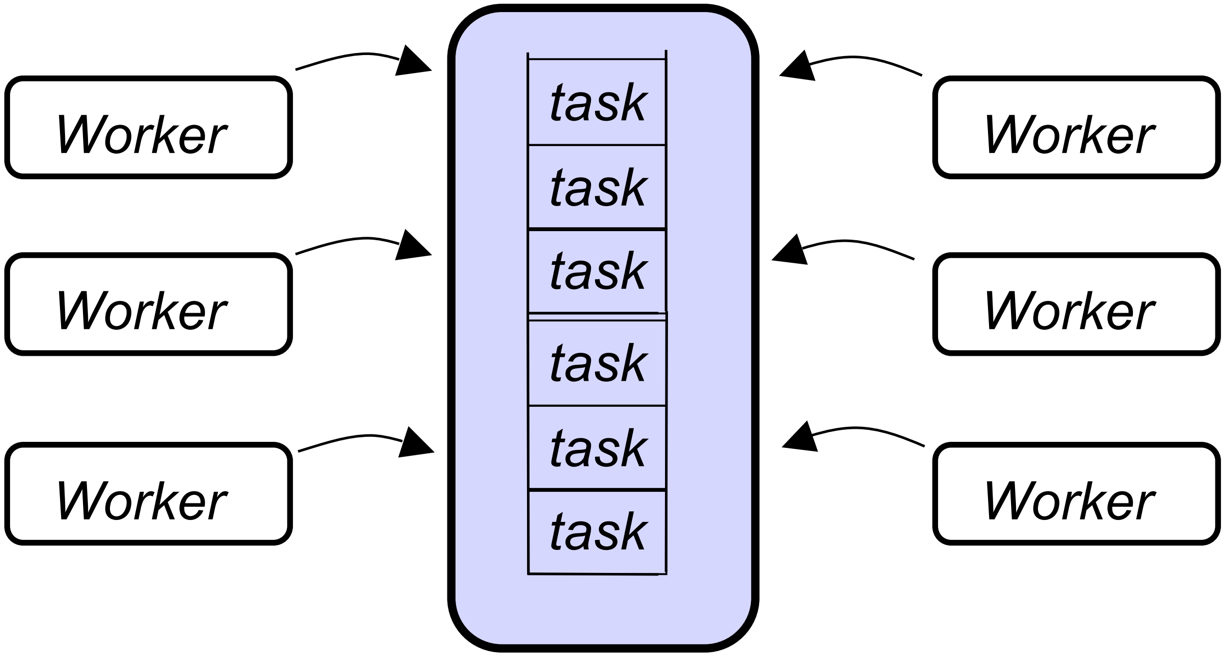
- Better Initial Assignment of Tasks
- Dynamic Load Balancing of Tasks
  1. Centralized Task Queue
  2. Distributed Task Queue

….

Many Others

Centralized Task Queue

- A Centralized queue of tasks waiting to be done
  1. The queue may be held by a shared data structure protected by mutex's.
  2. Or the queue may be held by a process solely responsible for doling out tasks (coordinator)



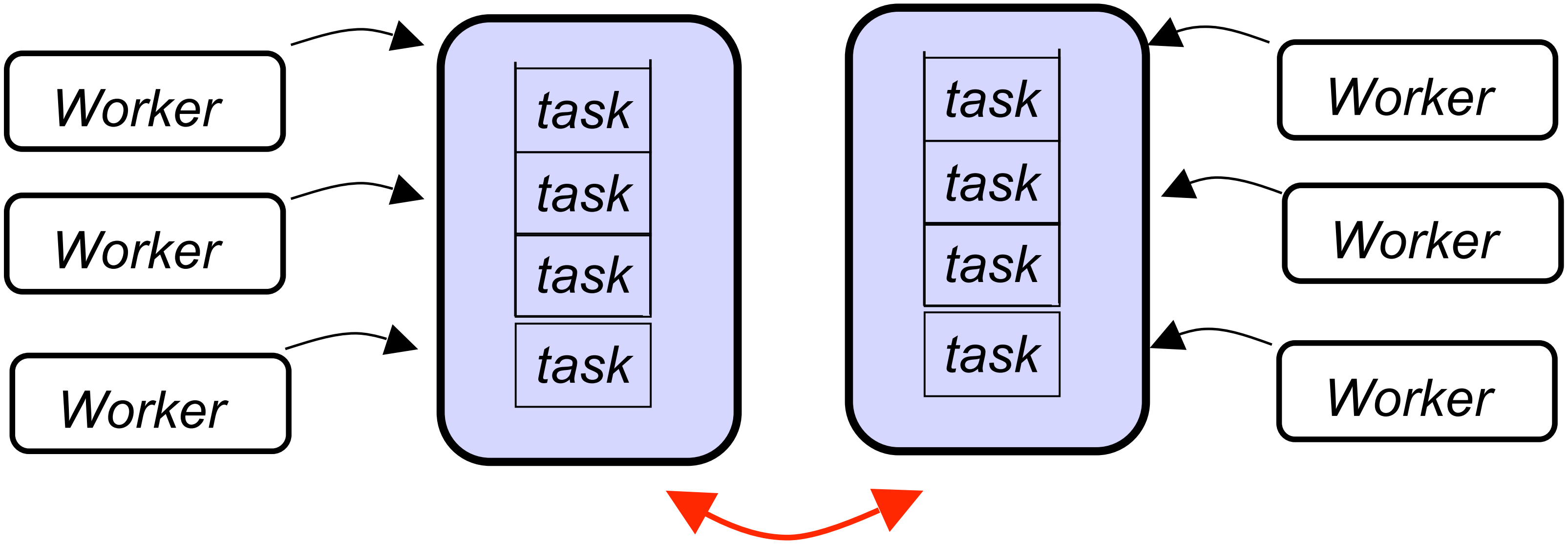
- How to distribute tasks, one at a time?

How to Distribute the Tasks

1. Fixed # of Tasks (K chunk size)
   - If K too large, overhead for accessing tasks is low BUT not all tasks processes may finish at same time
   - If K too small, overhead high but better chance all processes finish at same time.
2. Guided Self Scheduling - use larger chunks at beginning, smaller chunks at end.
3. Tapering - chunk size a function of remaining work and task variance, large variance = smaller chunk size, smaller variance = larger chunk size.

Distributed Task Queue

- Natural Extension when cost of accessing the queue is high due to large # of processors.



- How to distribute tasks?
  - Evenly distributed between the groups.
  - Lightly loaded group PULLS work.

Any Questions?

Extra Slides

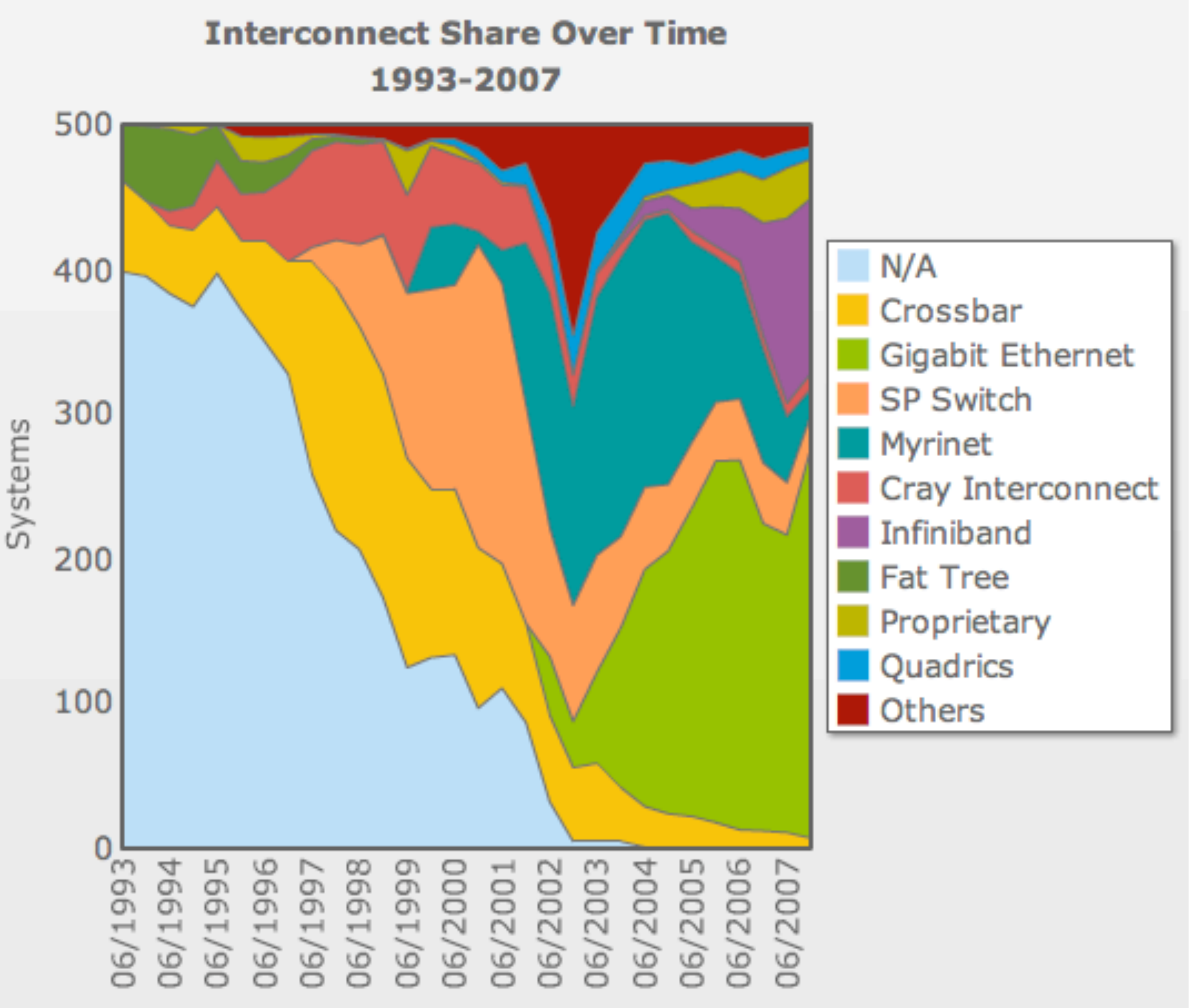
Units of Measure in HPC

- **High Performance Computing (HPC) units are:**
  - **Flop: floating point operation**
  - **Flops/s: floating point operations per second**
  - **Bytes: size of data (a double precision floating point number is 8)**
- **Typical sizes are millions, billions, trillions…**

Mega	**Mflop/s = 10^{6} flop/sec**	**Mbyte = 2^{20} = 1048576 ~ 10^{6} bytes**
Giga	**Gflop/s = 10^{9} flop/sec**	**Gbyte = 2^{30} ~ 10^{9} bytes**
Tera	**Tflop/s = 10^{12} flop/sec**	**Tbyte = 2^{40} ~ 10^{12} bytes**
Peta	**Pflop/s = 10^{15} flop/sec**	**Pbyte = 2^{50} ~ 10^{15} bytes**
Exa	**Eflop/s = 10^{18} flop/sec**	**Ebyte = 2^{60} ~ 10^{18} bytes**
Zetta	**Zflop/s = 10^{21} flop/sec**	**Zbyte = 2^{70} ~ 10^{21} bytes**
Yotta	**Yflop/s = 10^{24} flop/sec**	**Ybyte = 2^{80} ~ 10^{24} bytes**

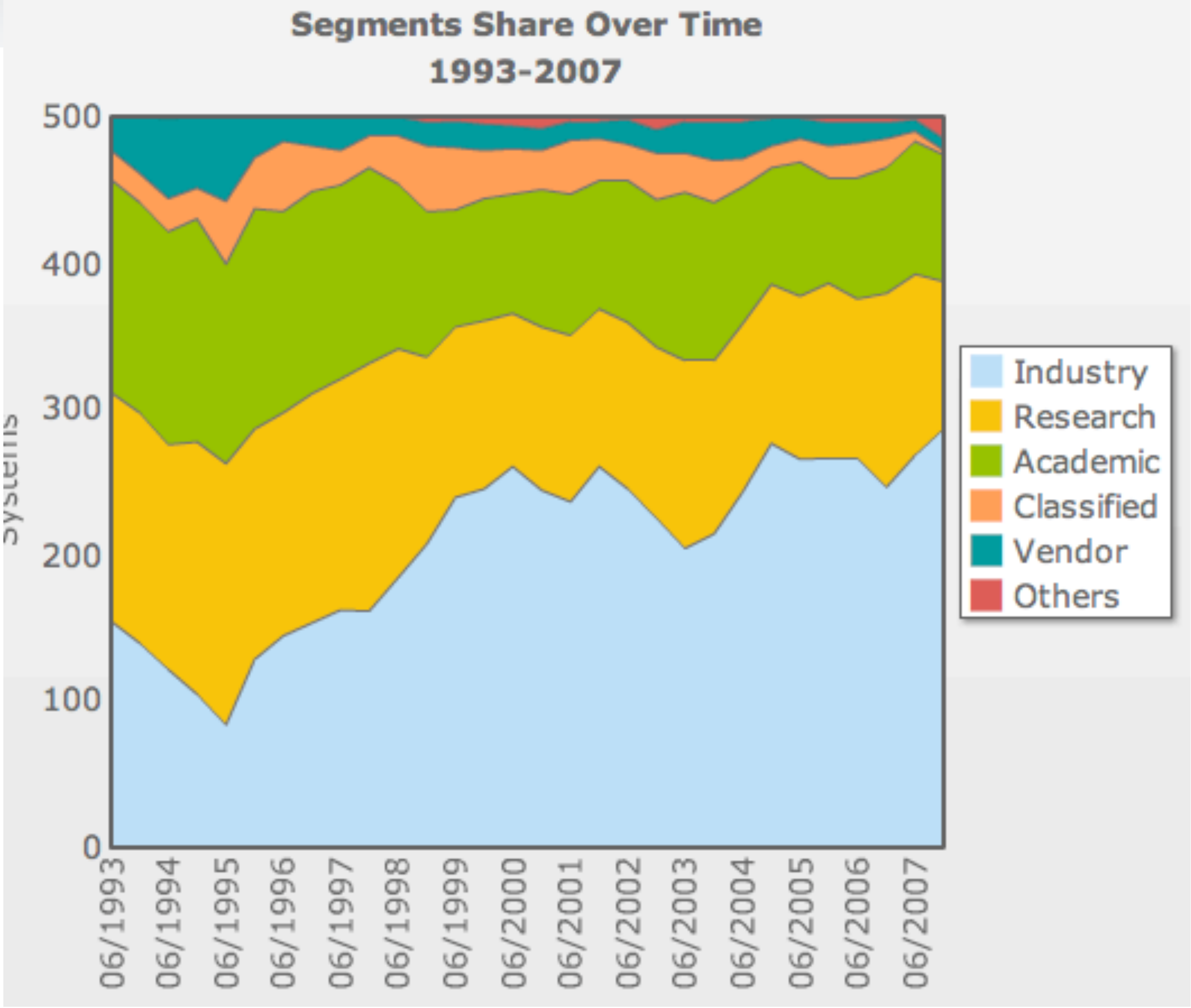
TOP500® SUPERCOMPUTER SITES

www.Top500.org



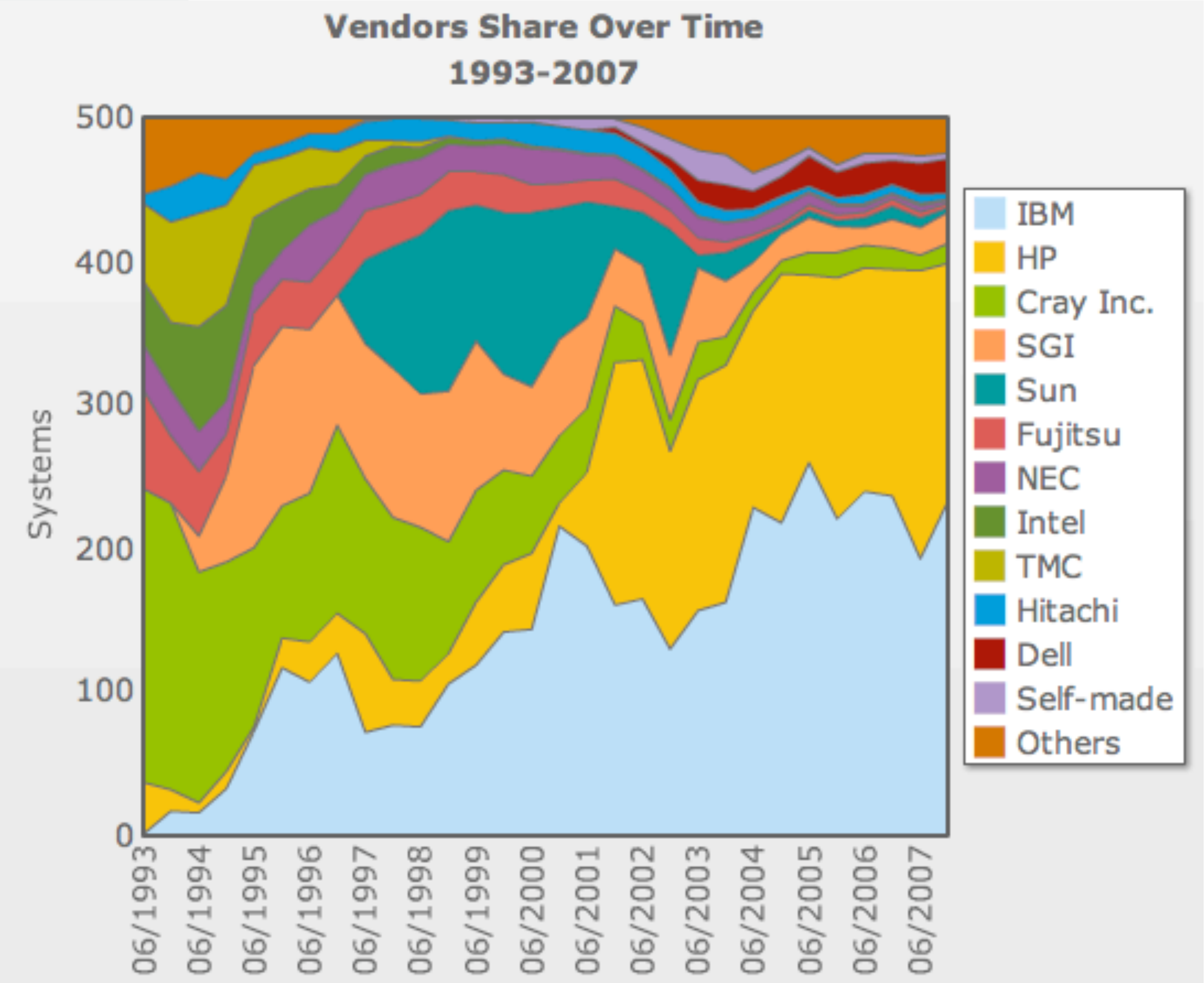
TOP500® SUPERCOMPUTER SITES

www.Top500.org



Segments Share Over Time
1993-2007

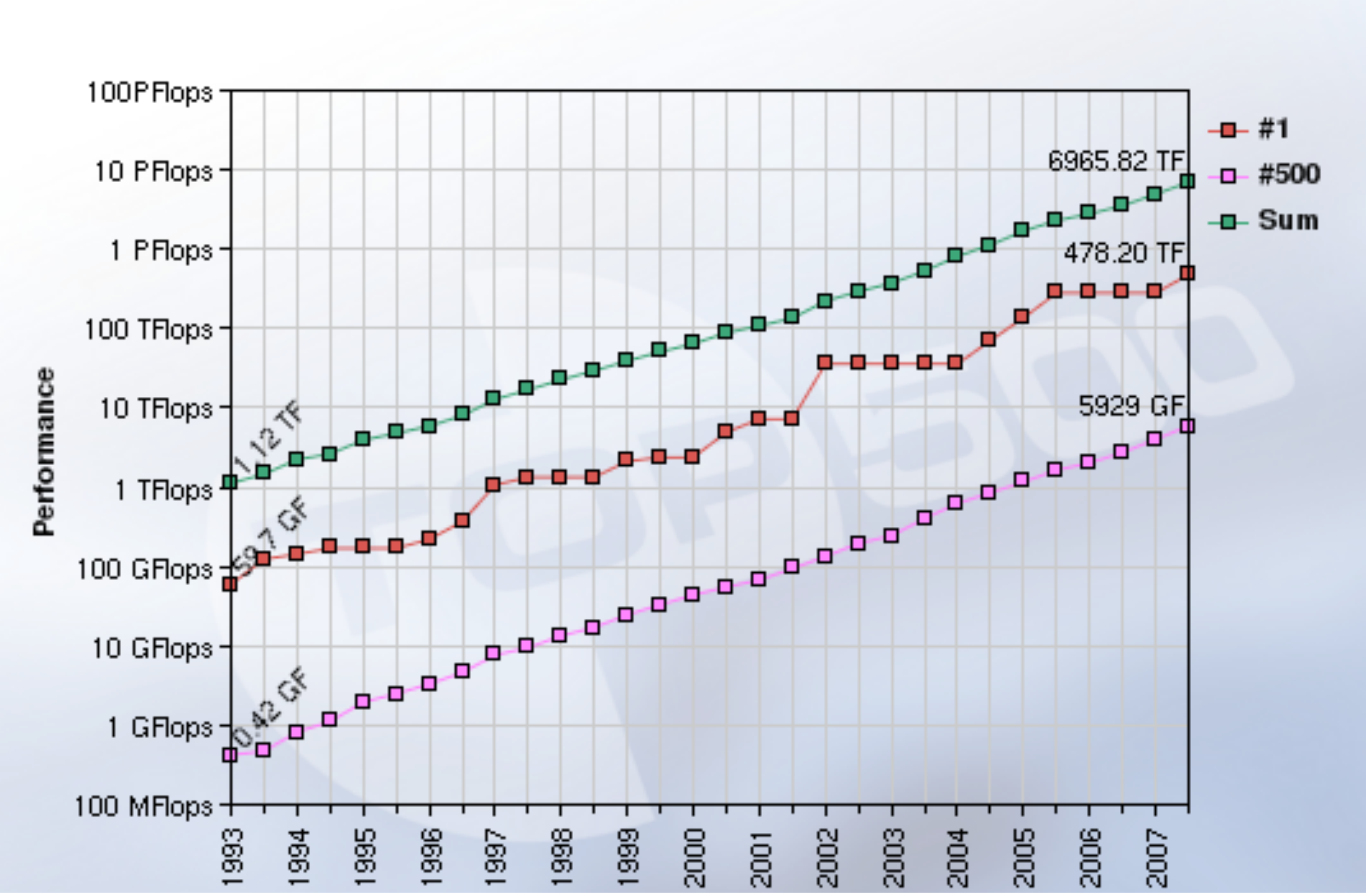
www.Top500.org

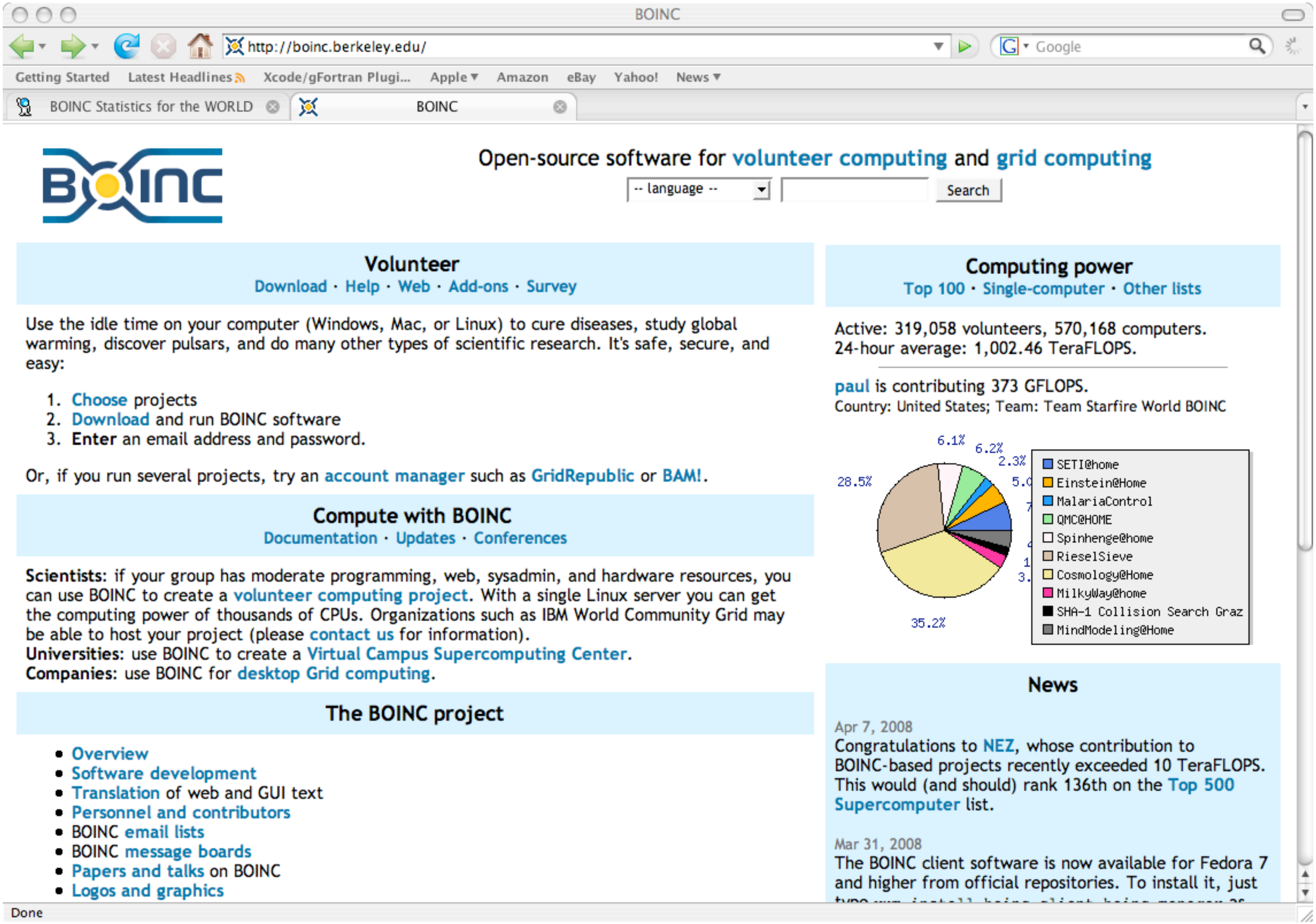


Vendors Share Over Time
1993-2007

TOP500®
SUPERCOMPUTER SITES

www.Top500.org





Open-source software for volunteer computing and grid computing

Volunteer

Download · Help · Web · Add-ons · Survey

Use the idle time on your computer (Windows, Mac, or Linux) to cure diseases, study global warming, discover pulsars, and do many other types of scientific research. It's safe, secure, and easy:

1. Choose projects
2. Download and run BOINC software
3. **Enter** an email address and password.

Or, if you run several projects, try an account manager such as GridRepublic or BAM!.

Compute with BOINC

Documentation · Updates · Conferences

Scientists: if your group has moderate programming, web, sysadmin, and hardware resources, you can use BOINC to create a volunteer computing project. With a single Linux server you can get the computing power of thousands of CPUs. Organizations such as IBM World Community Grid may be able to host your project (please contact us for information).
Universities: use BOINC to create a Virtual Campus Supercomputing Center.
Companies: use BOINC for desktop Grid computing.

The BOINC project

- Overview
- Software development
- Translation of web and GUI text
- Personnel and contributors
- BOINC email lists
- BOINC message boards
- Papers and talks on BOINC
- Logos and graphics

Computing power

Top 100 · Single-computer · Other lists

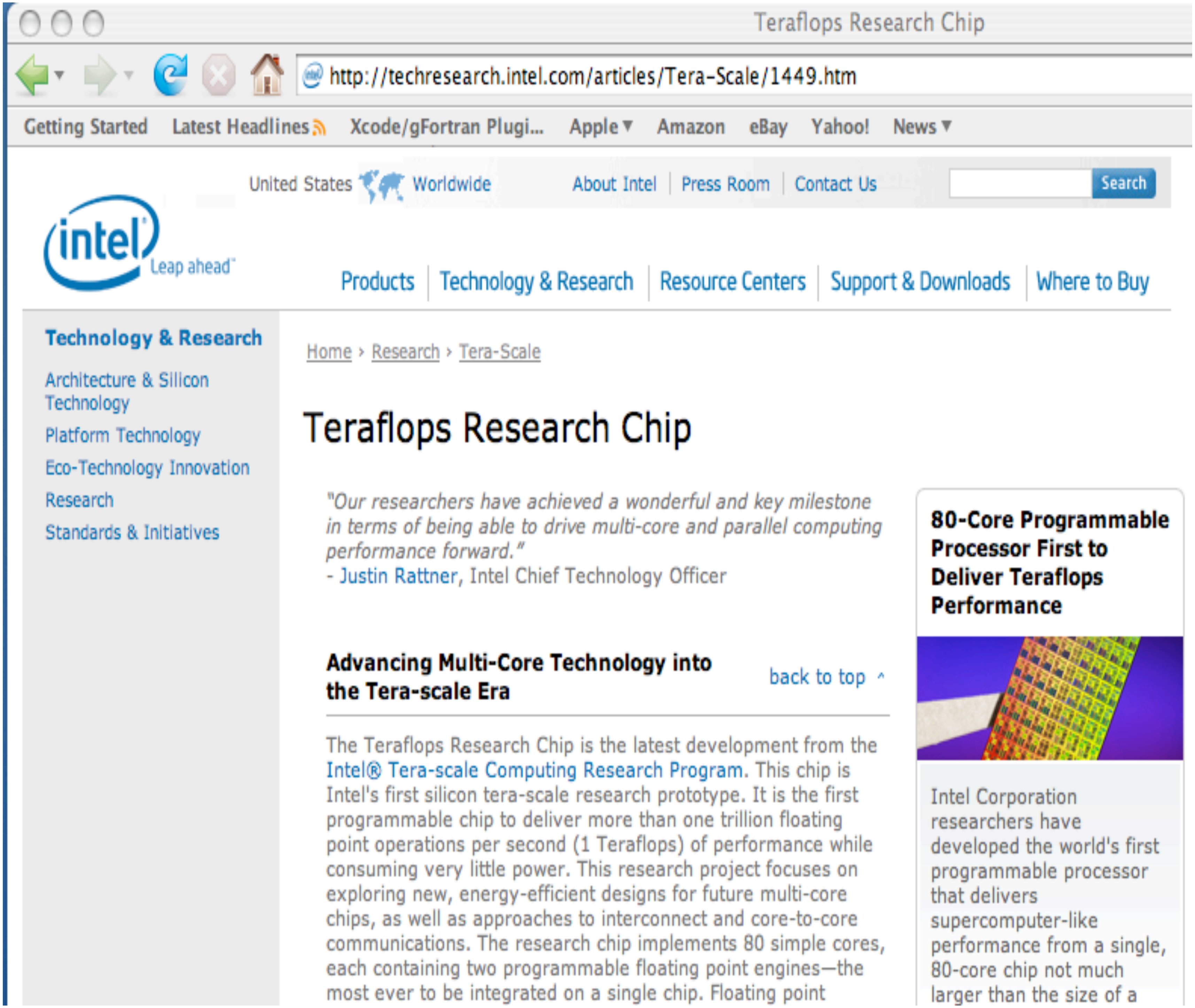
Active: 319,058 volunteers, 570,168 computers.
24-hour average: 1,002.46 TeraFLOPS.

paul is contributing 373 GFLOPS.
Country: United States; Team: Team Starfire World BOINC

News

Apr 7, 2008
Congratulations to NEZ, whose contribution to BOINC-based projects recently exceeded 10 TeraFLOPS. This would (and should) rank 136th on the Top 500 Supercomputer list.

Mar 31, 2008
The BOINC client software is now available for Fedora 7 and higher from official repositories. To install it, just type yum install boinc-client boinc-manager as



http://techresearch.intel.com/articles/Tera-Scale/1449.htm

Technology & Research

- Architecture & Silicon Technology
- Platform Technology
- Eco-Technology Innovation
- Research
- Standards & Initiatives

Home › Research › Tera-Scale

Teraflops Research Chip

> *"Our researchers have achieved a wonderful and key milestone in terms of being able to drive multi-core and parallel computing performance forward."*
> \- Justin Rattner, Intel Chief Technology Officer

Advancing Multi-Core Technology into the Tera-scale Era

back to top ˄

The Teraflops Research Chip is the latest development from the Intel® Tera-scale Computing Research Program. This chip is Intel's first silicon tera-scale research prototype. It is the first programmable chip to deliver more than one trillion floating point operations per second (1 Teraflops) of performance while consuming very little power. This research project focuses on exploring new, energy-efficient designs for future multi-core chips, as well as approaches to interconnect and core-to-core communications. The research chip implements 80 simple cores, each containing two programmable floating point engines—the most ever to be integrated on a single chip. Floating point

80-Core Programmable Processor First to Deliver Teraflops Performance

Intel Corporation researchers have developed the world's first programmable processor that delivers supercomputer-like performance from a single, 80-core chip not much larger than the size of a