OpenSees

# Parallel Computing

Frank McKenna
UC Berkeley

OpenSees Parallel Workshop
Berkeley, CA

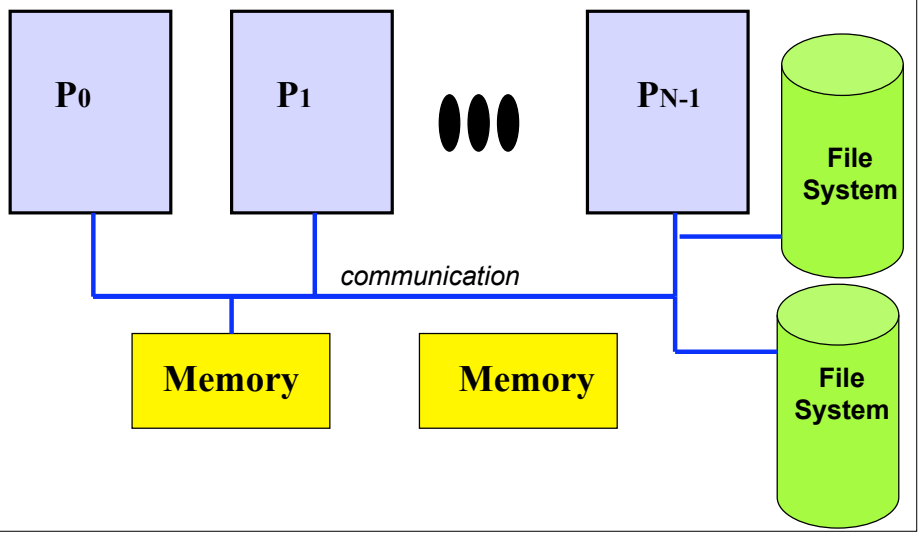
NEES

## Overview

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

# What is a Parallel Computer?

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




# Why should you care?

- They will save you time
- They will allow you to solver larger problems.
- They are here whether you like it or not!

**2009**

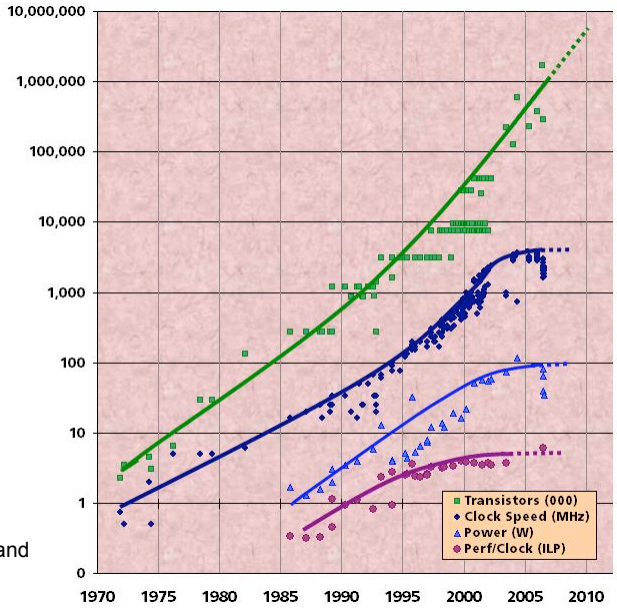
www.Top500.org

| Rank | Site | Computer/Year Vendor | Cores | $R_{max}$ | $R_{peak}$ | Power |
|---|---|---|---|---|---|---|
| 1 | DOE/NNSA/LANL<br>United States | Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz , Voltaire Infiniband / 2008<br>IBM | 129600 | 1105.00 | 1456.70 | 2483.47 |
| 2 | Oak Ridge National Laboratory<br>United States | Jaguar - Cray XT5 QC 2.3 GHz / 2008<br>Cray Inc. | 150152 | 1059.00 | 1381.40 | 6950.60 |
| 3 | NASA/Ames Research Center/NAS<br>United States | Pleiades - SGI Altix ICE 8200EX, Xeon QC 3.0/2.66 GHz / 2008<br>SGI | 51200 | 487.01 | 608.83 | 2090.00 |
| 4 | DOE/NNSA/LLNL<br>United States | BlueGene/L - eServer Blue Gene Solution / 2007<br>IBM | 212992 | 478.20 | 596.38 | 2329.60 |
| 5 | Argonne National Laboratory<br>United States | Blue Gene/P Solution / 2007<br>IBM | 163840 | 450.30 | 557.06 | 1260.00 |
| 6 | Texas Advanced Computing Center/Univ. of Texas<br>United States | Ranger - SunBlade x6420, Opteron QC 2.3 Ghz, Infiniband / 2008<br>Sun Microsystems | 62976 | 433.20 | 579.38 | 2000.00 |
| 7 | NERSC/LBNL<br>United States | Franklin - Cray XT4 QuadCore 2.3 GHz / 2008<br>Cray Inc. | 38642 | 266.30 | 355.51 | 1150.00 |
| 8 | Oak Ridge National Laboratory<br>United States | Jaguar - Cray XT4 QuadCore 2.1 GHz / 2008<br>Cray Inc. | 30976 | 205.00 | 260.20 | 1580.71 |
| 9 | NNSA/Sandia National Laboratories<br>United States | Red Storm - Sandia/ Cray Red Storm, XT3/4, 2.4/2.2 GHz dual/quad core / 2008<br>Cray Inc. | 38208 | 204.20 | 284.00 | 2506.00 |

# Revolution is Happening Now

- Chip density is continuing increase ~2x every 1.5-2 years (Moore's law)
  - Clock speed is not
  - Number of processor cores may double instead
- There is little or no more hidden parallelism (ILP) to be found
- Parallelism must be exposed to and managed by software

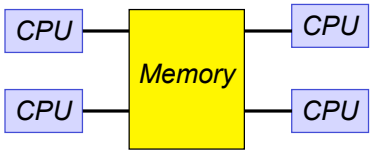
Source: Intel, Microsoft (Sutter) and Stanford (Olukotun, Hammond)




# 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. Symmetric Multiprocessor (Shared Memory)
  4. Distributed Systems (Distributed Memory)
     1. Cluster
     2. MPP (massively parallel processor)
     3. Grid.
  5. Hybrid Systems.

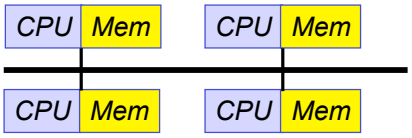
# Shared Memory




• 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.

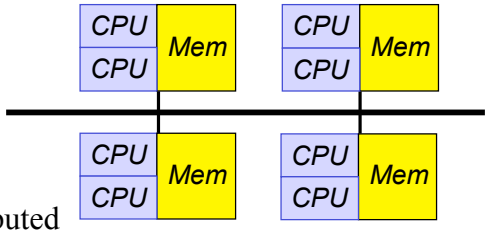
# Distributed Memory

- Processors operate independently, each has it's own private memory.
- Data must be shared using the communication network.
  1. Specialized network - MPP
  2. Commodity network - Cluster
  3. Internet - Grid

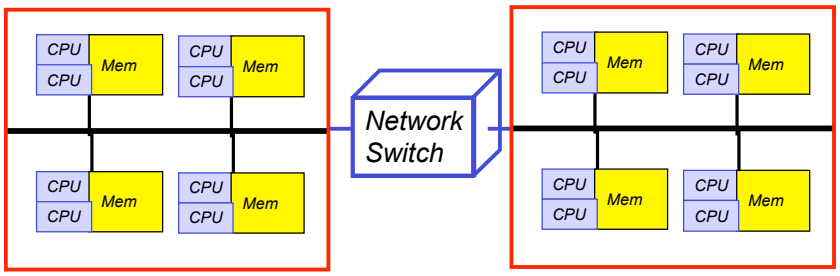



# Hybrid System

•With new multi-core systems distributed memory processors combination of shared and distributed memory.



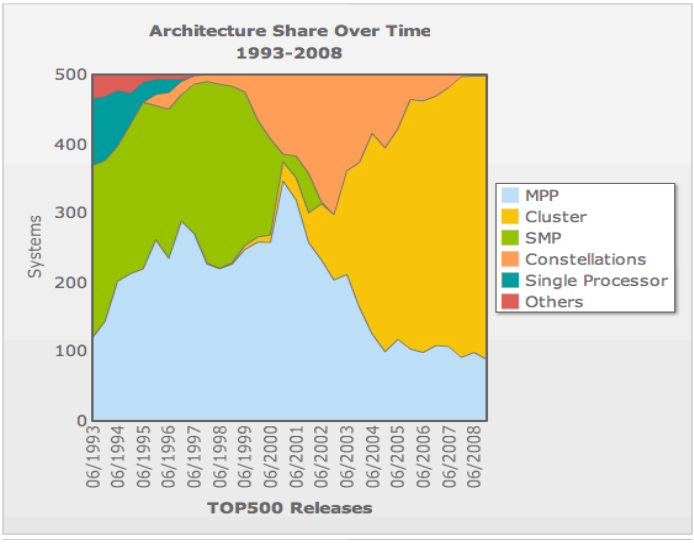
# Cluster Systems Dominate Top500




- Network Switch on faster machines are Gigabit Ethernet, Myrinet or Infiniband
- 82% of current top 500 (www.top500.org) are designated cluster machines.
- 50% 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.

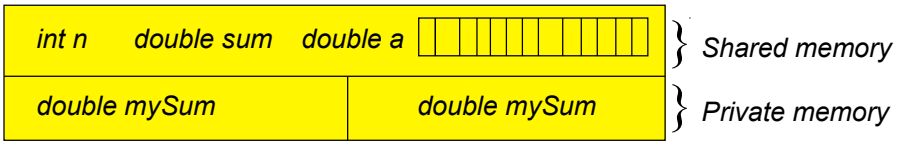
•Each thread has it's own private data, e.g. local stack.

•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

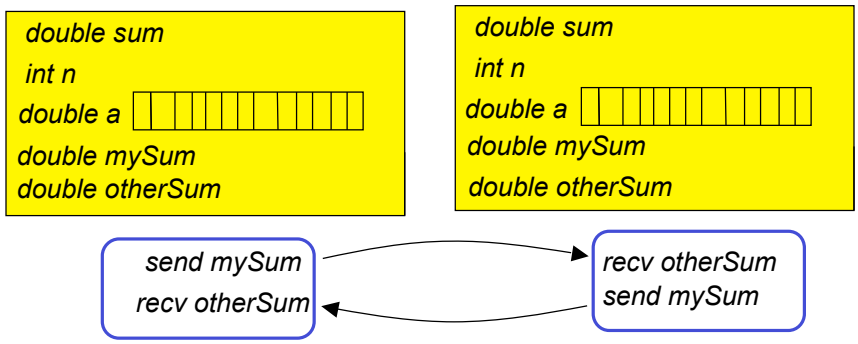
- The use of mutex's can lead to a condition of **deadlock**. Deadlock occurs when 2 or more threads are blocked forever waiting for a mutex locked by the other.

```
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




# 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;
```

# 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.

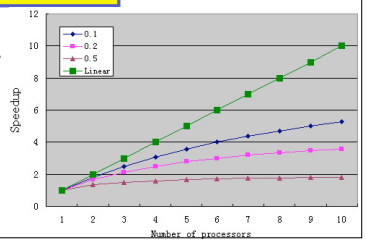
# Speedup & Amdahl's Law

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

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

*Portion of sequential*

*# of processors*



## Writing Parallel Programs: Steps Involved

1. Understand Amdahl's Law
2. Break the computation to be performed into tasks (which can be based on function, data or both).
3. Assign the tasks to processes, identifying those tasks which can be executed concurrently.
4. Program it.
5. Compile, Test & Debug
6. 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.
4. Program needs sufficiently large units of work to run fast in parallel (i.e. large granularity), but not so large that there is not enough parallel work.
5. Cost of starting threads/processes is not insignificant.

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

# Improving Real Performance

**Peak Performance grows exponentially, a la Moore's Law**

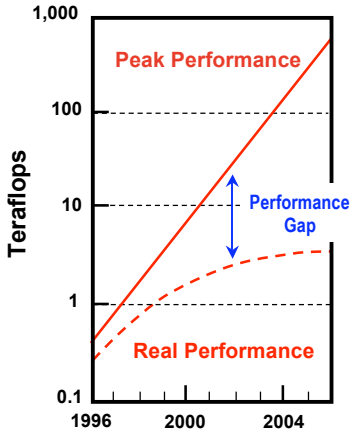
- In 1990's, peak performance increased 100x; in 2000's, it will increase 1000x

**But efficiency (the performance relative to the hardware peak) has declined**

- was 40-50% on the vector supercomputers of 1990s
- now as little as 5-10% on parallel supercomputers of today

**Close the gap through ...**

- Mathematical methods and algorithms that achieve high performance on a single processor and scale to thousands of processors
- More efficient programming models and tools for massively parallel supercomputers




Source: Jim Demmell, CS267 Course Notes

# Performance Levels
## (for example on NERSC-5)

- Peak advertised performance (PAP): 100 Tflop/s
- LINPACK (TPP): 84 Tflop/s
- Best climate application: 14 Tflop/s
  - –WRF code benchmarked in December 2007
- Average sustained applications performance: ? Tflop/s
  - –Probably less than 10% peak!

Source: Jim Demmell, CS267 Course Notes

## Reasons for Poor Performance in Parallel Programs

- Not enough Concurrent Tasks Can Be Identified.
- 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

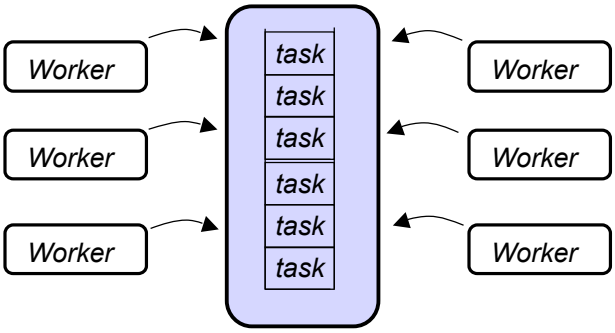
- Better Initial Assignment of Tasks
- Dynamic Load Balancing of Tasks
  1. Centralized Task Queue
  2. Distributed Task Queue

     ....

     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)



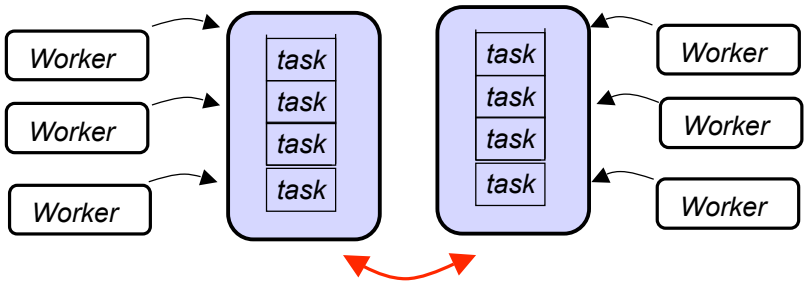

- How to distribute tasks, one at a time?

# 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

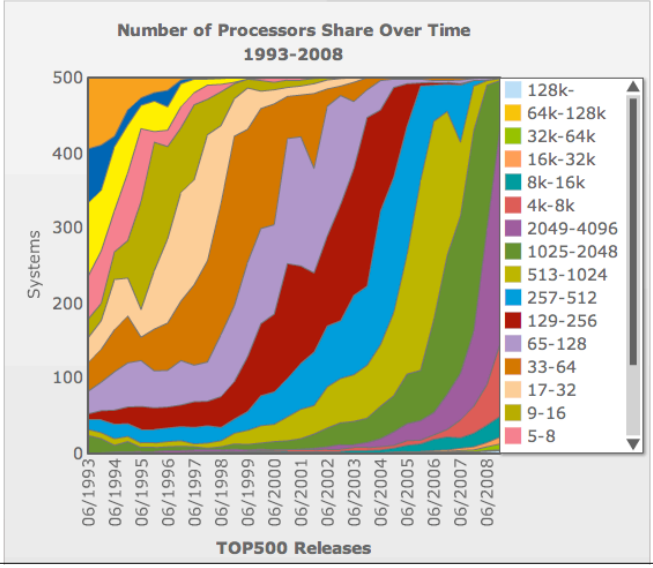
# Units of Measure in HPC

- **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** |

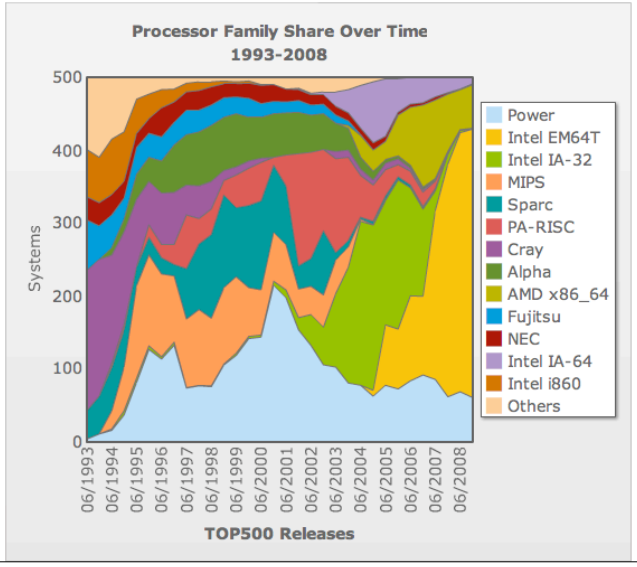
TOP500 SUPERCOMPUTER SITES

www.Top500.org



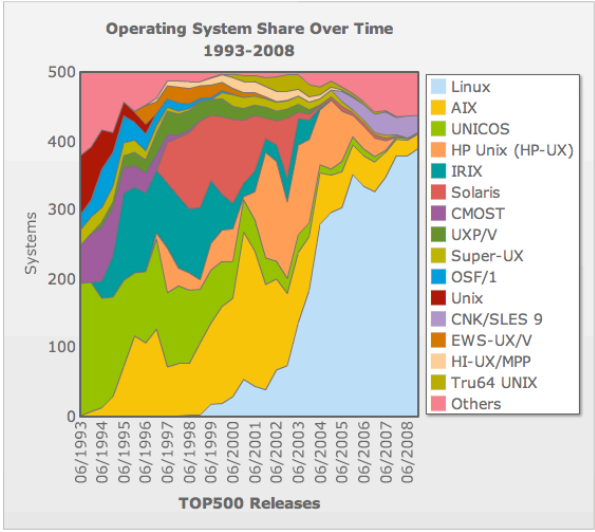

TOP500 SUPERCOMPUTER SITES

www.Top500.org



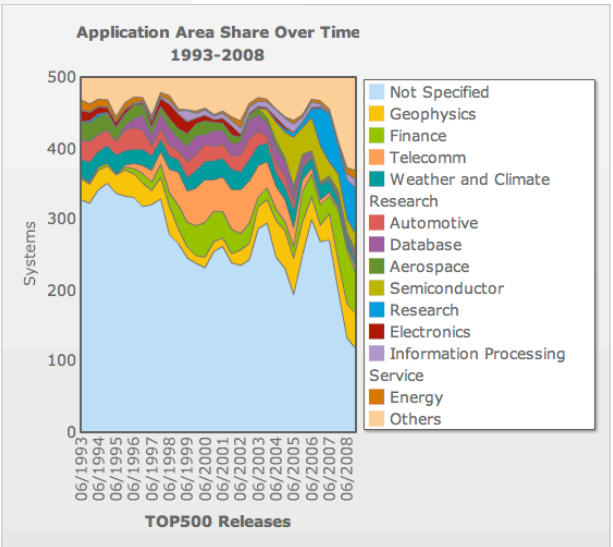
TOP500 SUPERCOMPUTER SITES

www.Top500.org

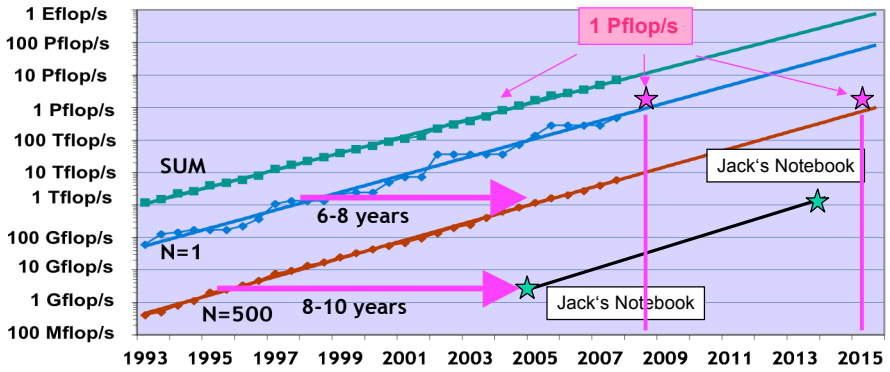



TOP500 SUPERCOMPUTER SITES

www.Top500.org

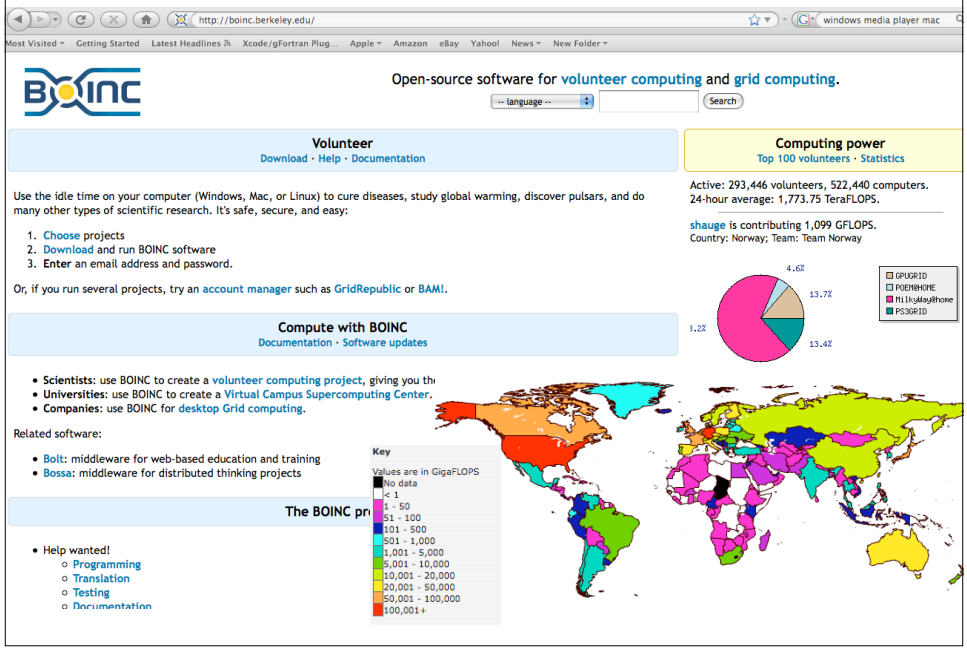


# Performance Projection




Source:Jack Dongarra, Innovate Computing Lab 2009

# Grid Computing



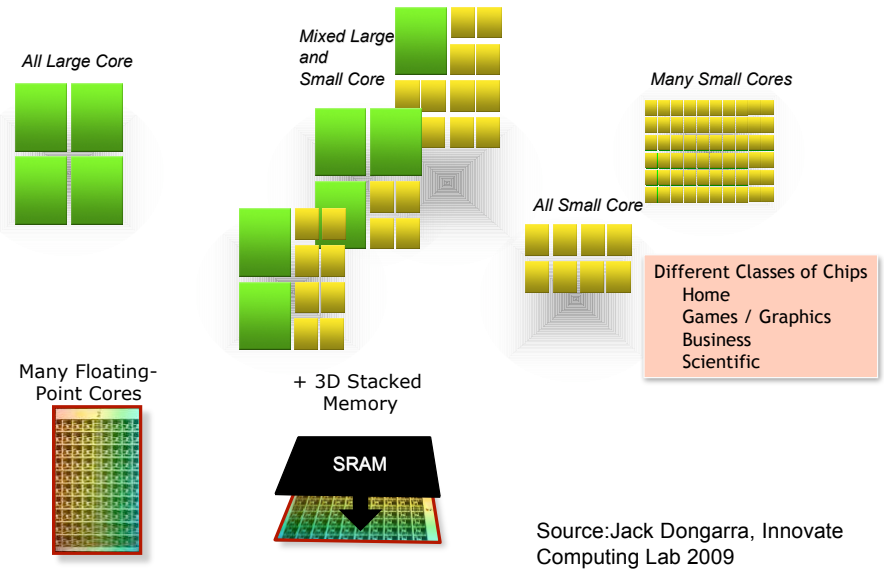
# Bell's Law

**Bell's Law of Computer Class formation**
was discovered about 1972. It states that technology advances in semiconductors, storage, user interface and networking advance every decade enable a new, usually lower priced computing platform to form. Once formed, each class is maintained as a quite independent industry structure. This explains mainframes, minicomputers, workstations and Personal computers, the web, emerging web services, palm and mobile devices, and ubiquitous interconnected networks. We can expect home and body area networks to follow this path.

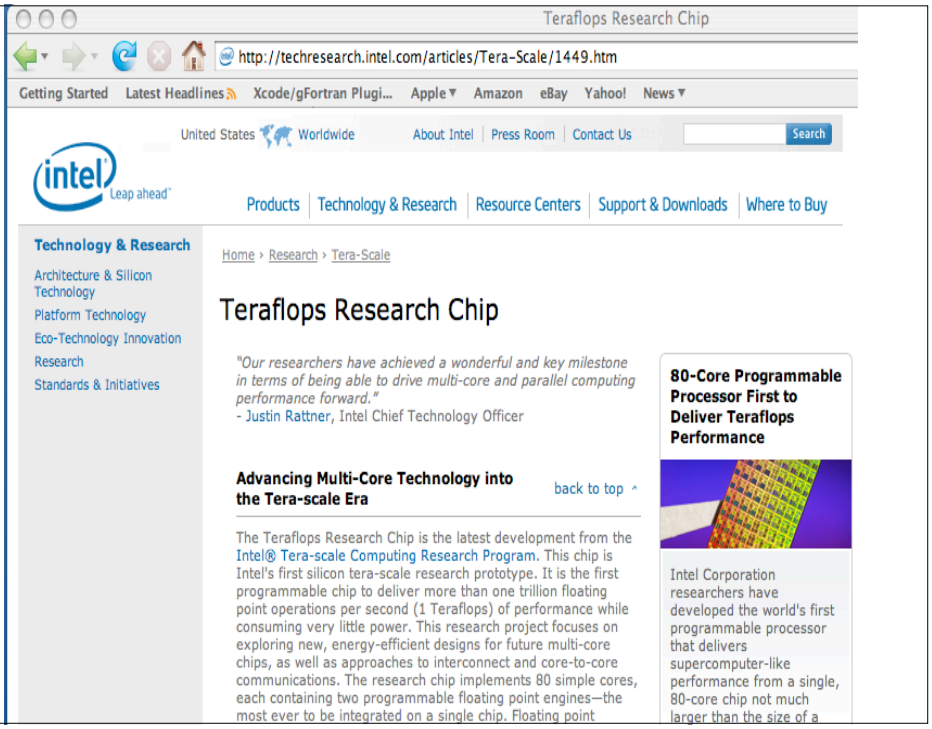
Gordon Bell (2007), http://research.microsoft.com/~GBell/Pubs.htm

Source:Jack Dongarra, Innovate Computing Lab 2009

# What's Next?

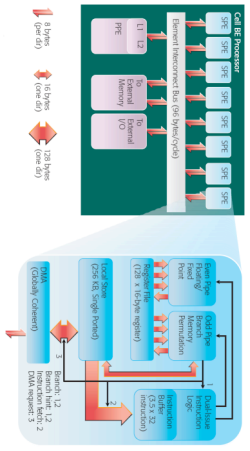



Source:Jack Dongarra, Innovate Computing Lab 2009



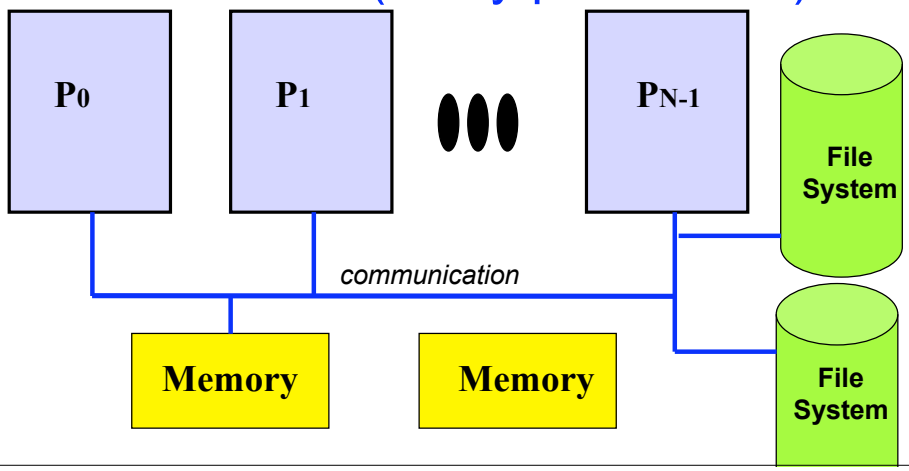

# Cell Processor

- PlayStation 3 based on "Cell" Processor
- Each Cell contains a PowerPC and 8 self contained vector processing units (SPU's).
- Power PC at 3.2 GHz
  - DGEMM at 5 Gflop/s
  - Altivec peak at 25.6 Gflop/s
    - Achieved 10 Gflop/s SGEMM
- 8 SPUs
  - 204.8 Gflop/s peak!
  - The catch is that this is for 32 bit floating point; (Single Precision SP)
  - And 64 bit floating point runs at 14.6 Gflop/s total for all 8 SPEs!!



Source:Jack Dongarra, Innovate Computing Lab 2009

# Whatever happens the computers you will be working on now and in the future will be PARALLEL (many processors)




# Tunnel Vision by Experts

- "I think there is a world market for maybe five computers."
  - Thomas Watson, chairman of IBM, 1943.
- "There is no reason for any individual to have a computer in their home"
  - Ken Olson, president and founder of Digital Equipment Corporation, 1977.
- "640K [of memory] ought to be enough for anybody."
  - Bill Gates, chairman of Microsoft,1981.

Slide source: Warfield et al.