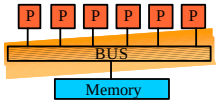
CS 594 Spring 2003
Lecture 3:
Overview of High-Performance Computing

Jack Dongarra
Computer Science Department
University of Tennessee

Types of Parallel Computers

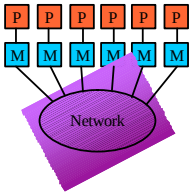
- **The simplest and most useful way to classify modern parallel computers is by their memory model:**
  - **shared memory**
  - **distributed memory**

Shared vs. Distributed Memory

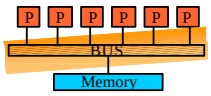


Shared memory - single address space. All processors have access to a pool of shared memory. (Ex: SGI Origin, Sun E10000)

Distributed memory - each processor has it's own local memory. Must do message passing to exchange data between processors. (Ex: CRAY T3E, IBM SP, clusters)

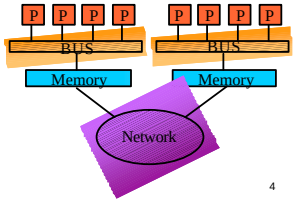


Shared Memory: UMA vs. NUMA

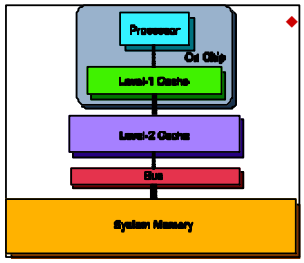


Uniform memory access (UMA): Each processor has uniform access to memory. Also known as symmetric multiprocessors (Sun E10000)

Non-uniform memory access (NUMA): Time for memory access depends on location of data. Local access is faster than non-local access. Easier to scale than SMPs (SGI Origin)



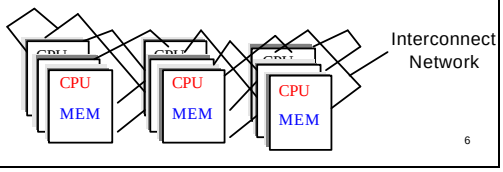
Standard Uniprocessor Memory Hierarchy



- **Intel Pentium III 1.135 GHz processor (Model 11)**
  - **16 Kbytes of 4 way assoc. L1 instruction cache with 32 byte lines.**
  - **16 Kbytes of 4 way assoc. L1 data cache with 32 byte lines.**
  - **512 Kbytes of 8 way assoc. L2 cache 32 byte lines.**

Distributed Memory: MPPs vs. Clusters

- **Processors-memory nodes are connected by some type of interconnect network**
  - **Massively Parallel Processor (MPP): tightly integrated, single system image.**
  - **Cluster: individual computers connected by s/w**



Processors, Memory, & Networks

- Both shared and distributed memory systems have:
  1. processors: now generally commodity RISC processors
  2. memory: now generally commodity DRAM
  3. network/interconnect: between the processors and memory (bus, crossbar, fat tree, torus, hypercube, etc.)

7

Interconnect-Related Terms

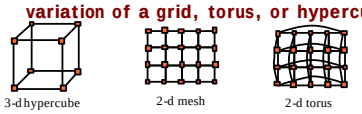
- Latency: How long does it take to start sending a "message"? Measured in microseconds.
  (Also in processors: How long does it take to output results of some operations, such as floating point add, divide etc., which are pipelined?)
- Bandwidth: What data rate can be sustained once the message is started? Measured in Mbytes/sec.

8

Interconnect-Related Terms

Topology: the manner in which the nodes are connected.

- Best choice would be a fully connected network (every processor to every other). Unfeasible for cost and scaling reasons.
- Instead, processors are arranged in some variation of a grid, torus, or hypercube.



3-d hypercube 2-d mesh 2-d torus

9

Shared Memory / Local Memory

- Usually think in terms of the hardware
- What about a software model?
- How about something that works like cache?
- Logically shared memory

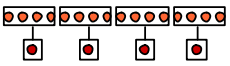
10

Parallel Programming Models

- Control
  - how is parallelism created
  - what orderings exist between operations
  - how do different threads of control synchronize
- Naming
  - what data is private vs. shared
  - how logically shared data is accessed or communicated
- Set of operations
  - what are the basic operations
  - what operations are considered to be atomic
- Cost
  - how do we account for the cost of each of the above

Trivial Example $\sum_{i=0}^{n-1} f(A[i])$

- Parallel Decomposition:
  - Each evaluation and each partial sum is a task
- Assign n/p numbers to each of p procs
  - each computes independent "private" results and partial sum
  - one (or all) collects the p partial sums and computes the global sum

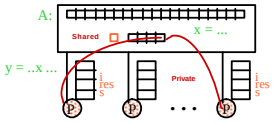


=> Classes of Data

- Logically Shared
  - the original n numbers, the global sum
- Logically Private
  - the individual function evaluations
  - what about the individual partial sums?

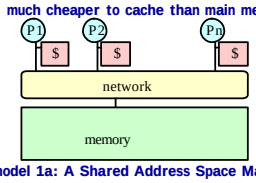
Programming Model 1

- **Shared Address Space**
  - program consists of a collection of threads of control,
  - each with a set of private variables
    - » e.g., local variables on the stack
  - collectively with a set of shared variables
    - » e.g., static variables, shared common blocks, global heap
  - threads communicate implicity by writing and reading shared variables
  - threads coordinate explicitly by synchronization operations on shared variables
    - » writing and reading flags
    - » locks, semaphores
- **Like concurrent programming on uniprocessor**



Model 1

- A shared memory machine
- Processors all connected to a large shared memory
- "Local" memory is not (usually) part of the hardware
  - Sun, DEC, Intel "SMPs" (Symmetric multiprocessors) in Millennium; SGI Origin
- Cost: much cheaper to cache than main memory



- **Machine model 1a: A Shared Address Space Machine**
  - replace caches by local memories (in abstract machine model)
  - this affects the cost model -- repeatedly accessed data should be copied
  - Cray T3E

Shared Memory code for computing a sum

Thread 1

```
[s = 0 initially]
local_s1= 0
for i = 0, n/2-1
   local_s1 = local_s1 + f(A[i])
s = s + local_s1
```

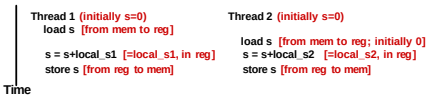
Thread 2

```
[s = 0 initially]
local_s2 = 0
for i = n/2, n-1
   local_s2= local_s2 + f(A[i])
s = s +local_s2
```

What could go wrong?

Pitfall and solution via synchronization

- Pitfall in computing a global sum s = local_s1 + local_s2



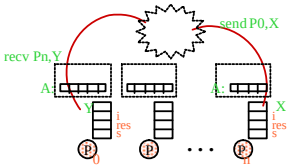
- Instructions from different threads can be interleaved arbitrarily
- What can final result s stored in memory be?
- Race Condition
- Possible solution: Mutual Exclusion with Locks

Thread 1	Thread 2
lock	lock
load s	load s
s = s+local_s1	s = s+local_s2
store s	store s
unlock	unlock

- Locks must be atomic (execute completely without interruption)

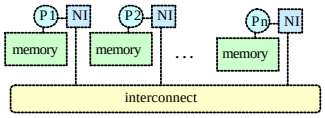
Programming Model 2

- **Message Passing**
  - program consists of a collection of named processes
    - » thread of control plus local address space
    - » local variables, static variables, common blocks, heap
  - processes communicate by explicit data transfers
    - » matching pair of send & receive by source and dest. proc.
  - coordination is implicit in every communication event
  - logically shared data is partitioned over local processes
- **Like distributed programming**
- Program with standard libraries: MPI, PVM



Model 2

- **A distributed memory machine**
  - Cray T3E, IBM SP2, Clusters
- **Processors all connected to own memory (and caches)**
  - cannot directly access another processor's memory
- **Each "node" has a network interface (NI)**
  - all communication and synchronization done through this



Computing s = x(1)+x(2) on each processor

° First possible solution

```
Processor 1                    Processor 2
  send xlocal, proc2             receive xremote, proc1
    [xlocal = x(1)]              send xlocal, proc1
  receive xremote, proc2           [xlocal = x(2)]
  s = xlocal + xremote           s = xlocal + xremote
```

° Second possible solution - what could go wrong?

```
Processor 1                    Processor 2
  send xlocal, proc2             send xlocal, proc1
    [xlocal = x(1)]              [xlocal = x(2)]
  receive xremote, proc2         receive xremote, proc1
  s = xlocal + xremote           s = xlocal + xremote
```

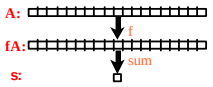
° What if send/receive act like the telephone system? The post office?

19

Programming Model 3

- Data Parallel
  - Single sequential thread of control consisting of parallel operations
  - Parallel operations applied to all (or defined subset) of a data structure
  - Communication is implicit in parallel operators and "shifted" data structures
  - Elegant and easy to understand and reason about
  - Not all problems fit this model
- Like marching in a regiment

A = array of all data
fA = f(A)
s = sum(fA)

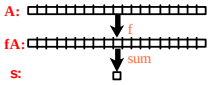


° Think of Matlab

Model 3

- Vector Computing
  - One instruction executed across all the data in a pipelined fashion
  - Parallel operations applied to all (or defined subset) of a data structure
  - Communication is implicit in parallel operators and "shifted" data structures
  - Elegant and easy to understand and reason about
  - Not all problems fit this model
- Like marching in a regiment

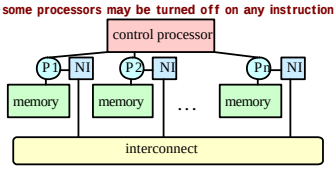
A = array of all data
fA = f(A)
s = sum(fA)



° Think of Matlab

Model 3

- An SIMD (Single Instruction Multiple Data) machine
- A large number of small processors
- A single "control processor" issues each instruction
  - each processor executes the same instruction
  - some processors may be turned off on any instruction



- Machines not popular (CM2), but programming model is
  - implemented by mapping n-fold parallelism to p processors
  - mostly done in the compilers (HPF = High Performance Fortran)

22

Model 4

- Since small shared memory machines (SMPs) are the fastest commodity machine, why not build a larger machine by connecting many of them with a network?
- CLUMP = Cluster of SMPs
- Shared memory within one SMP, message passing outside
- Clusters, ASCI Red (Intel), ...
- Programming model?
  - Treat machine as "flat", always use message passing, even within SMP (simple, but ignore important part of memory hierarchy)
  - Expose two layers: shared memory (OpenMP) and message passing (MPI) higher performance, but ugly to program.

23

Programming Model 5

- Bulk Synchronous Processing (BSP) – L. Valiant
- Used within the message passing or shared memory models as a programming convention
- *Phases* separated by global barriers
  - Compute phases: all operate on local data (in distributed memory)
    - or read access to global data (in shared memory)
  - Communication phases: all participate in rearrangement or reduction of global data
- Generally all doing the "same thing" in a phase
  - all do f, but may all do different things within f
- Simplicity of data parallelism without restrictions

Summary so far

- Historically, each parallel machine was unique, along with its programming model and programming language
- You had to throw away your software and start over with each new kind of machine - ugh
- Now we distinguish the programming model from the underlying machine, so we can write portably correct code, that runs on many machines
  - MPI now the most portable option, but can be tedious
- Writing portably fast code requires tuning for the architecture
  - Algorithm design challenge is to make this process easy
  - Example: picking a blocksize, not rewriting whole algorithm

25

Recap

- Parallel Comp. Architecture driven by familiar technological and economic forces
  - application/platform cycle, but focused on the most demanding applications
  - hardware/software learning curve

Speedup

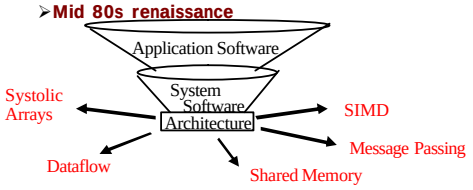
- More attractive than ever because 'best' building block - *the microprocessor* - is also the fastest BB.
- History of microprocessor architecture is parallelism
  - translates area and denisty into performance
- The Future is higher levels of parallelism
  - Parallel Architecture concepts apply at many levels
  - Communication also on exponential curve

=> Quantitative Engineering approach

26

History

- Parallel architectures tied closely to programming models
  - Divergent architectures, with no predictable pattern of growth.
  - Mid 80s renaissance



27

Programming Model

- *Conceptualization of the machine that programmer uses in coding applications*
  - How parts cooperate and coordinate their activities
  - Specifies communication and synchronization operations
- Multiprogramming
  - Independent jobs, no communication or synch. at program level
- *Shared address space*
  - like bulletin board
- *Message passing*
  - like letters or phone calls, explicit point to point
- *Data parallel*:
  - more regimented, global actions on data
  - Implemented with shared address space or message passing

28

Economics

- Commodity microprocessors not only fast but CHEAP
  - Development costs tens of millions of dollars
  - BUT, many more are sold compared to supercomputers
  - Crucial to take advantage of the investment, and use the commodity building block
- Multiprocessors being pushed by software vendors (e.g. database) as well as hardware vendors
- Standardization makes small, bus-based SMPs commodity
- Desktop: few smaller processors versus one larger one?
- Multiprocessor on a chip?

29

Performance Numbers on RISC Processors

- Using Linpack Benchmark

Machine	MHz	Linpack n=100 Mflop/s	Ax=b n=1000 Mflop/s	Peak Mflop/s
Intel P4	2800	1317 (24%)	2444 (47%)	5600
IBM Power 4	1300	1135 (22%)	2899 (56%)	5200
Intel Itanium	1000	1102 (28%)	3534 (88%)	4000
Compaq Alpha	1250	1031 (41%)	1945 (41%)	2500
AMD Athlon	1200	558 (23%)	998 (42%)	2400
HP PA	550	468 (21%)	1583 (71%)	2200
Intel P3	933	234 (25%)	514 (55%)	933
PowerPC G4	533	231 (22%)	478 (45%)	1066
SUN Ultra 80	450	208 (23%)	607 (67%)	900
SGI Origin 2K	300	173 (29%)	553 (92%)	600
Cray T90	454	705 (39%)	1603 (89%)	1800
Cray C90	238	387 (41%)	902 (95%)	952
Cray Y-MP	166	161 (48%)	324 (97%)	333
Cray X-MP	118	121 (51%)	218 (93%)	235
Cray J-90	100	106 (53%)	190 (95%)	200
Cray 1	80	27 (17%)	110 (69%)	160

30

Consider Scientific Supercomputing

- Proving ground and driver for innovative architecture and techniques
  - Market smaller relative to commercial as MPs become mainstream
  - Dominated by vector machines starting in 70s
  - Microprocessors have made huge gains in floating-point performance
    - » high clock rates
    - » pipelined floating point units (e.g., multiply-add every cycle)
    - » instruction-level parallelism
    - » effective use of caches (e.g., automatic blocking)
  - Plus economics
- Large-scale multiprocessors replace vector supercomputers

31

High Performance Computers

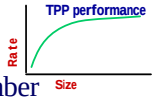
- ~ 20 years ago
  - $1x10^6$ Floating Point Ops/sec (Mflop/s)
    - » Scalar based
- ~ 10 years ago
  - $1x10^9$ Floating Point Ops/sec (Gflop/s)
    - » Vector & Shared memory computing, bandwidth aware
    - » Block partitioned, latency tolerant
- ~ Today
  - $1x10^{12}$ Floating Point Ops/sec (Tflop/s)
    - » Highly parallel, distributed processing, message passing, network based
    - » data decomposition, communication/computation
- ~ 10 years away
  - $1x10^{15}$ Floating Point Ops/sec (Pflop/s)
    - » Many more levels MH, combination/grids&HPC
    - » More adaptive, LT and bandwidth aware, fault tolerant, extended precision, attention to SMP nodes

32

Top 500 Computers

- Listing of the 500 most powerful Computers in the World
- Yardstick: Rmax from LINPACK MPP

$Ax=b$, *dense problem*



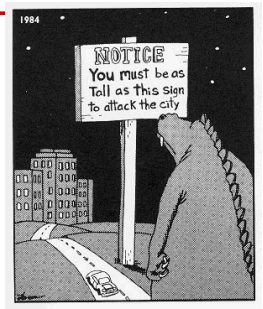
Updated twice a year
SC'xy in the States in November
Meeting in Mannheim, Germany in June

10 Year for Top500 and 25 Year for Linpack Benchmark

33

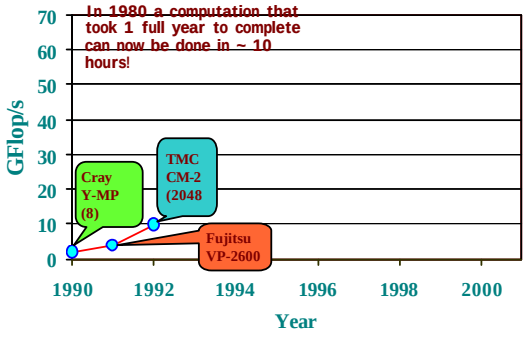
Big Means What?

- Over the last 10 years the range for the Top500 has increased greater than Moore's Law
- 1993:
  - #1 = 59.7 GFlop/s
  - #500 = 422 MFlop/s
- 2002:
  - #1 = 35.8 TFlop/s
  - #500 = 196 GFlop/s

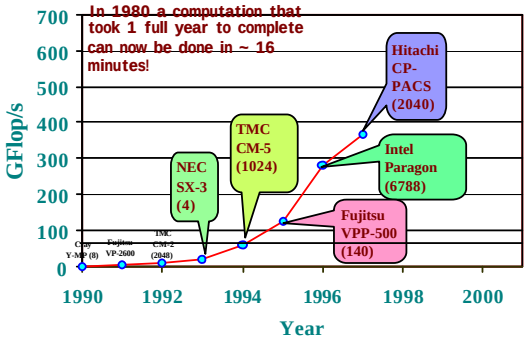


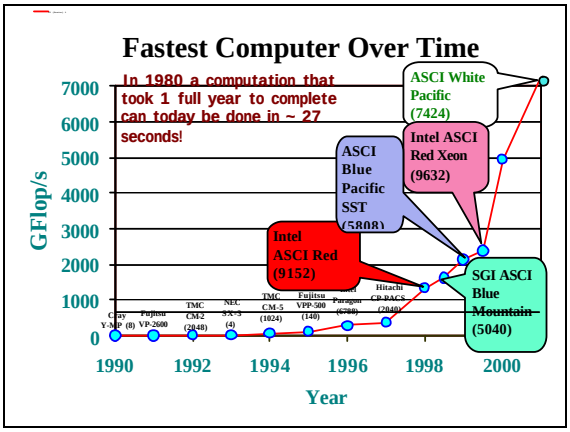
34

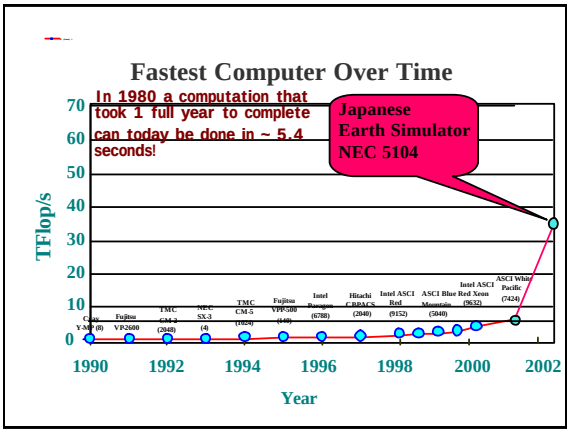
Fastest Computer Over Time



Fastest Computer Over Time







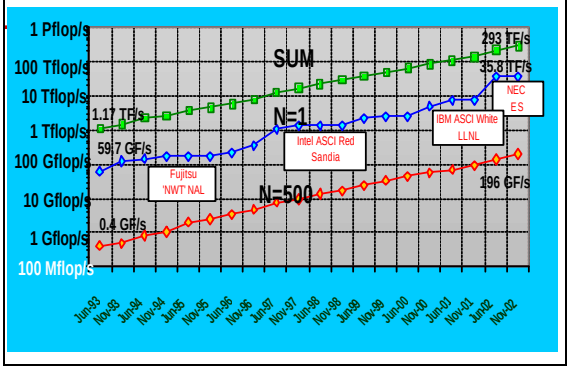
20th List: The TOP10

Rank	Manufacturer	Computer	R_{max} [TF/s]	Installation Site	Country	Year	Area of Installation	# Proc
1	NEC	Earth-Simulator	35.86	Earth Simulator Center	Japan	2002	Research	5120
2	HP	ASCI Q, AlphaServer SC	7.73	Los Alamos National Laboratory	USA	2002	Research	4096
2	HP	ASCI Q, AlphaServer SC	7.73	Los Alamos National Laboratory	USA	2002	Research	4096
4	IBM	ASCI White SP Power3	7.23	Lawrence Livermore National Laboratory	USA	2000	Research	8192
5	Linux NetworX	MCR Cluster	5.69	Lawrence Livermore National Laboratory	USA	2002	Research	8192
6	HP	AlphaServer SC ES45 1 GHz	4.46	Pittsburgh Supercomputing Center	USA	2001	Academic	3016
7	HP	AlphaServer SC ES45 1 GHz	3.98	Commissariat a l'Energie Atomique (CEA)	France	2001	Research	2560
8	HPTi	Xeon Cluster - Myrinet2000	3.34	Forecast Systems Laboratory - NOAA	USA	2002	Research	1536
9	IBM	pSeries 690 Turbo	3.16	HPCx	UK	2002	Academic	1280
10	IBM	pSeries 690 Turbo	3.16	NCAR (National Center for Atmospheric Research)	USA	2002	Research	1216

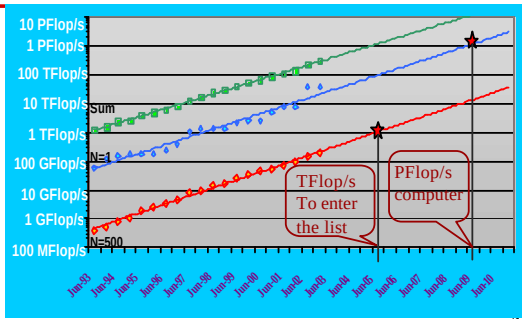
39

182 fell off; 500 was 318 in June

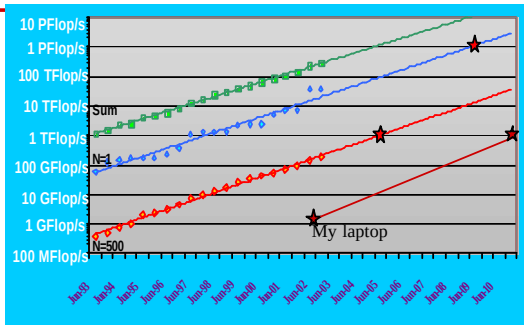
TOP500 - Performance



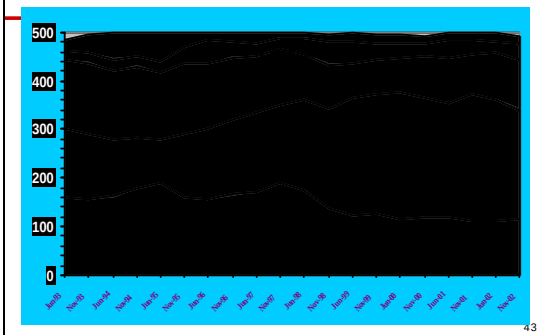
Performance Extrapolation



Performance Extrapolation



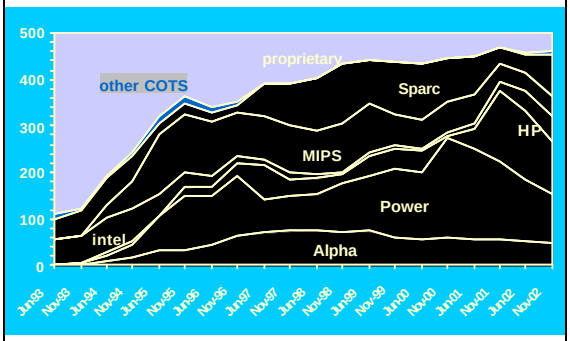
Customer Type



43

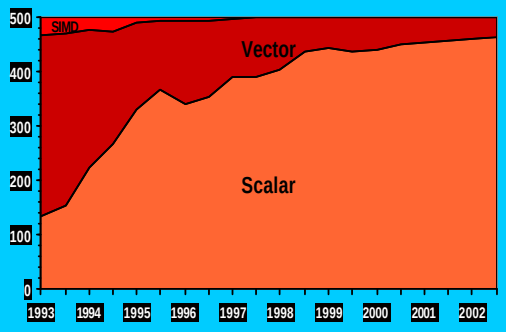
226 Industry, 115 Research, 102 Academic, 32 Classified

Chip Technology

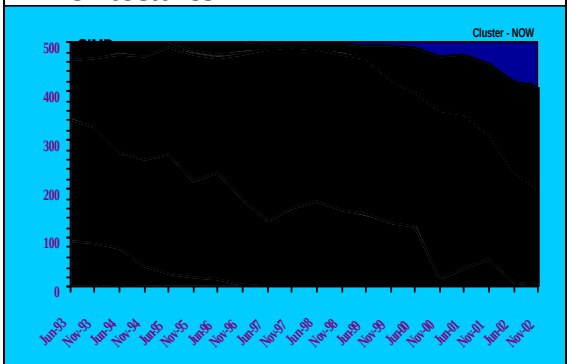


44

Processor Type

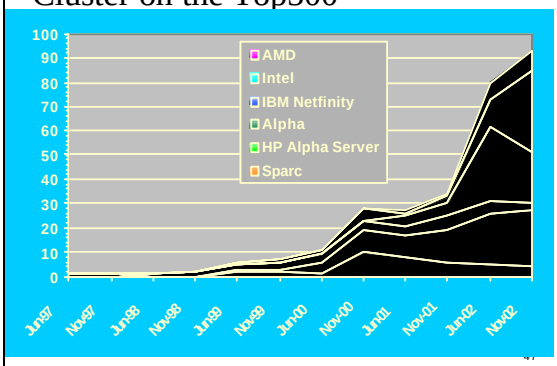


Architectures



92 clusters, Constellation: # of p/n ⩾ n

Cluster on the Top500



Top500 Conclusions

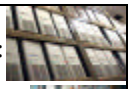
- **Microprocessor based supercomputers have brought a major change in accessibility and affordability.**
- **MPPs continue to account of more than half of all installed high-performance computers worldwide.**

48

Performance Numbers on RISC Processors

Processor	Cycle Time	Linpack n=100	Linpack n=1000	Peak
Intel P4	2540	1190 (23%)	2355 (46%)	5080
Intel/HP Itanium 2	1000	1102 (27%)	3534 (88%)	4000
Compaq Alpha	1000	824 (41%)	1542 (77%)	2000
AMD Athlon	1200	558 (23%)	998 (42%)	2400
HP PA	550	468 (21%)	1583 (71%)	2200
IBM Power 3	375	424 (28%)	1208 (80%)	1500
Intel P3	933	234 (25%)	514 (55%)	933
PowerPC G4	533	231 (22%)	478 (45%)	1066
SUN Ultra 80	450	208 (23%)	607 (67%)	900
SGI Origin 2K	300	173 (29%)	553 (92%)	600
Cray T90	454	705 (39%)	1603 (89%)	1800
Cray C90	238	387 (41%)	902 (95%)	952
Cray Y-MP	166	161 (48%)	324 (97%)	333
Cray X-MP	118	121 (51%)	218 (93%)	235
Cray J-90	100	106 (53%)	190 (95%)	200
Cray 1	80	27 (17%)	110 (69%)	160

High-Performance Computing Directions: Beowulf-class PC Clusters



Definition:

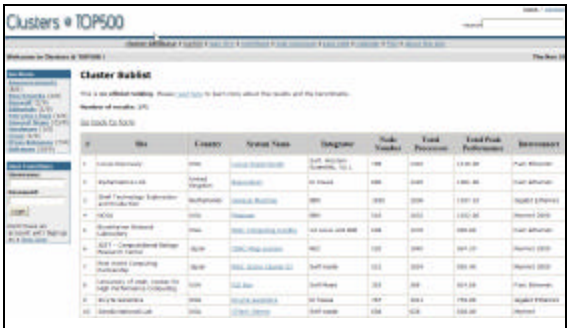
- COTS PC Nodes
  - Pentium, Alpha, PowerPC, SMP
- COTS LAN/SAN Interconnect
  - Ethernet, Myrinet, Giganet, ATM
- Open Source Unix
  - Linux, BSD
- Message Passing Computing
  - MPI, PVM
  - HPF

Advantages:

- Best price-perfo
- Low entry-level cost
- Just-in-place configuration
- Vendor invulnerable
- Scalable
- Rapid technology tracking

Enabled by PC hardware, networks and operating system achieving capabilities of scientific workstations at a fraction of the cost and availability of industry standard message passing libraries. However, much more of a contact sport.

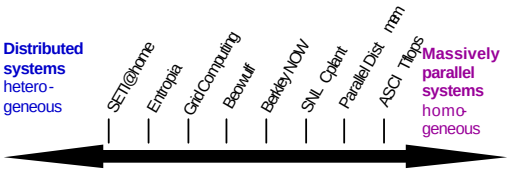
50



- Peak performance
- Interconnection
- http://clusters.top500.org
- Benchmark results to follow in the coming months

51

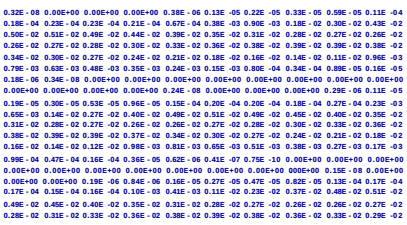
Distributed and Parallel Systems



- Gather (unused) resources
- Steal cycles
- System SW manages resources
- System SW adds value
- 10% - 20% overhead is OK
- Resources drive applications
- Time to completion is not critical
- Time-shared

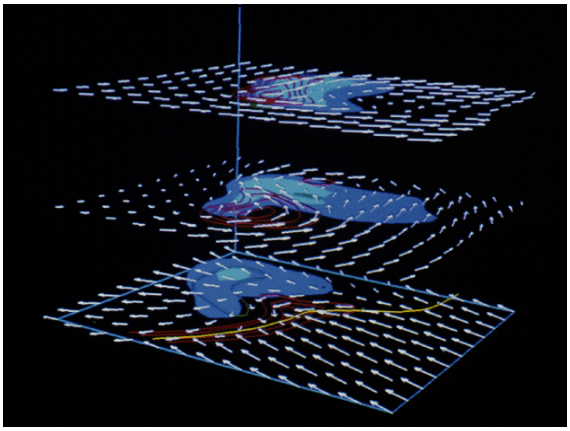
- Bounded set of resources
- Apps grow to consume all cycles
- Application manages resources
- System SW gets in the way
- 5% overhead is maximum
- Apps drive purchase of equipment
- Real-time constraints
- Space-shared

Virtual Environments

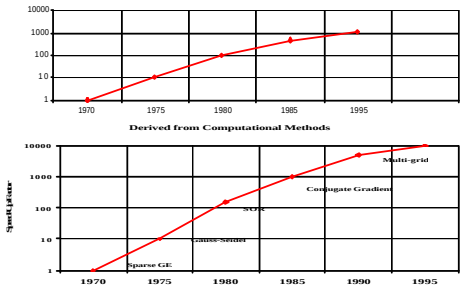


Do they make any sense?

53



Performance Improvements for Scientific Computing Problems



Highly Parallel Supercomputing: Where Are We?

- **Performance:**
  - Sustained performance has dramatically increased during the last year.
  - On most applications, sustained performance per dollar now exceeds that of conventional supercomputers. But...
  - Conventional systems are still faster on some applications.
- **Languages and compilers:**
  - Standardized, portable, high-level languages such as HPF, PVM and MPI are available. But ...
  - Initial HPF releases are not very efficient.
  - Message passing programming is tedious and hard to debug.
  - Programming difficulty remains a major obstacle to usage by mainstream scientist.

Highly Parallel Supercomputing: Where Are We?

- **Operating systems:**
  - Robustness and reliability are improving.
  - New system management tools improve system utilization. But...
  - Reliability still not as good as conventional systems.
- **I/O subsystems:**
  - New RAID disks, HiPPI interfaces, etc. provide substantially improved I/O performance. But...
  - I/O remains a bottleneck on some systems.

The Importance of Standards - Software

- Writing programs for MPP is hard ...
- But ... one-off efforts if written in a standard language
- Past lack of parallel programming standards ...
  - ... has restricted uptake of technology (to "enthusiasts")
  - ... reduced portability (over a range of current architectures and between future generations)
- Now standards exist: (PVM, MPI & HPF), which ...
  - ... allows users & manufacturers to protect software investment
  - ... encourage growth of a "third party" parallel software industry & parallel versions of widely used codes

The Importance of Standards - Hardware

- Processors
  - commodity RISC processors
- Interconnects
  - high bandwidth, low latency communications protocol
  - no de-facto standard yet (ATM, Fibre Channel, HPPI, FDDI)
- Growing demand for total solution:
  - robust hardware + usable software
- HPC systems containing all the programming tools / environments / languages / libraries / applications packages found on desktops

The Future of HPC

- The expense of being different is being replaced by the economics of being the same
- HPC needs to lose its "special purpose" tag
- Still has to bring about the promise of scalable general purpose computing ...
- ... but it is dangerous to ignore this technology
- Final success when MPP technology is embedded in desktop computing
- Yesterday's HPC is today's mainframe is tomorrow's workstation

Achieving TeraFlops

- In 1991, 1 Gflop/s
- 1000 fold increase
  - Architecture
    - » exploiting parallelism
  - Processor, communication, memory
    - » Moore's Law
  - Algorithm improvements
    - » block-partitioned algorithms

61

Future: Petaflops (10^{15} fl pt ops/s)

Today $\approx \sqrt{10^{15}}$ flops for our workstations

- A Pflop for 1 second ≈ a typical workstation computing for 1 year.
- From an algorithmic standpoint
  - concurrency
  - data locality
  - latency & sync
  - floating point accuracy
  - dynamic redistribution of workload
  - new language and constructs
  - role of numerical libraries
  - algorithm adaptation to hardware failure

62

A Petaflops Computer System

- 1 Pflop/s sustained computing
- Between 10,000 and 1,000,000 processors
- Between 10 TB and 1PB main memory
- Commensurate I/O bandwidth, mass store, etc.
- If built today, cost $40 B and consume 1 TWatt.
- May be feasible and "affordable" by the year 2010

63