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 its 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

Interconnect Network
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.)

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.

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.

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

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 \[ \sum_{i=0}^{n} f(A[i]) \]

- **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
  - A 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 implicitly 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
  - Cost: much cheaper to cache than main memory

Shared Memory code for computing a sum

Thread 1

\[
\text{[s = 0 initially]} \\
\text{local}_{s1} = 0 \\
\text{for i = 0, n/2-1} \\
\text{local}_{s1} = \text{local}_{s1} + f(A[i]) \\
\text{s = s + local}_{s1}
\]

Thread 2

\[
\text{[s = 0 initially]} \\
\text{local}_{s2} = 0 \\
\text{for i = n/2, n-1} \\
\text{local}_{s2} = \text{local}_{s2} + f(A[i]) \\
\text{s = s + local}_{s2}
\]

What could go wrong?

Pitfall and solution via synchronization

- Pitfall in computing a global sum \( s = \text{local}_{s1} + \text{local}_{s2} \)

Thread 1 (initially \( s = 0 \))

\[
\text{lock} \\
\text{load } s \text{ [from mem to reg]} \\
\text{s = s + local}_{s1} \\
\text{store } s \text{ [from reg to mem]}
\]

Thread 2 (initially \( s = 0 \))

\[
\text{lock} \\
\text{load } s \text{ [from mem to reg; initially 0]} \\
\text{s = s + local}_{s2} \\
\text{store } s \text{ [from reg to mem]}
\]

- 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

\[
\text{lock} \\
\text{load } s \text{ [from reg to mem]} \\
\text{s = s + local}_{s1} \\
\text{store } s \text{ [from reg to mem]}
\]

Thread 2

\[
\text{lock} \\
\text{load } s \text{ [from reg to mem]} \\
\text{s = s + local}_{s2} \\
\text{store } s \text{ [from reg to mem]}
\]

- Locks must be atomic (execute completely without interruption)

Programming Model 2

- Message Passing
  - A program consists of a collection of named processes
  - Thread of control in local address space
  - Local variables, stack variables, common blocks, heap
  - Processes communicate by explicit data transfers
  - Matching pair of send & receive by source and destination
  - 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
  - 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
- send \( x_{\text{local}}, \text{proc}2 \)
- receive \( x_{\text{remote}}, \text{proc}2 \)
- \( s = x_{\text{local}} + x_{\text{remote}} \)

Processor 2
- receive \( x_{\text{remote}}, \text{proc}1 \)
- send \( x_{\text{local}}, \text{proc}1 \)
- \( x_{\text{local}} = x(1) \)
- \( x_{\text{remote}} = x(2) \)
- \( s = x_{\text{local}} + x_{\text{remote}} \)

Second possible solution - what could go wrong?

Processor 1
- send \( x_{\text{local}}, \text{proc}2 \)
- receive \( x_{\text{remote}}, \text{proc}2 \)
- \( s = x_{\text{local}} + x_{\text{remote}} \)

Processor 2
- receive \( x_{\text{remote}}, \text{proc}1 \)
- send \( x_{\text{local}}, \text{proc}1 \)
- \( x_{\text{local}} = x(2) \)
- \( s = x_{\text{local}} + x_{\text{remote}} \)

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

° First possible solution

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
  - \( fA = f(A) \)
  - \( s = \text{sum}(fA) \)

° Think of Matlab

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 ignores important part of memory hierarchy)
  - Expose two layers: shared memory (OpenMP) and message passing (MPI) higher performance, but ugly to program

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
  - \( fA = f(A) \)
  - \( s = \text{sum}(fA) \)

° Think of Matlab

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
  - Computation 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.
- 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.

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.
- More attractive than ever because 'best' building block - the microprocessor - is also the fastest BB.
- History of microprocessor architecture is parallelism.
- Translates area and density into performance.
- The Future is higher levels of parallelism.
- Parallel Architecture concepts apply at many levels.
- Communication also on exponential curve.
- => Quantitative Engineering approach.

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?
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
    - Effectiveness of caches (e.g., automatic blocking)
    - Plus economics
  - Large-scale multiprocessors replace vector supercomputers

High Performance Computers

- ~ 20 years ago
  - $1 \times 10^9$ Floating Point Ops/sec (Mflop/s)
  - Scalar based
- ~ 10 years ago
  - $1 \times 10^9$ Floating Point Ops/sec (Gflop/s)
  - Vector & Shared memory computing, bandwidth aware
    - Block partitioned, latency tolerant
- ~ Today
  - $1 \times 10^{12}$ Floating Point Ops/sec (Tflop/s)
  - Highly parallel, distributed processing, message passing, network based
    - Data decomposed, communication/computation
  - ~ 10 years away
  - $1 \times 10^{15}$ 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

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

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

Fastest Computer Over Time

In 1980 a computation that took 1 full year to complete can now be done in ~ 16 minutes!
In 1980 a computation that took 1 full year to complete can today be done in ~ 27 seconds!

ASCI White Pacific (7424)
Intel ASCI Red Xeon (9642)
IBM Blue Pacific (ST) (5810)
SGi ASCI Blue Mountain (5840)

Fastest Computer Over Time

20th List: The TOP10

<table>
<thead>
<tr>
<th>Rank</th>
<th>Manufacturer</th>
<th>Computer</th>
<th>$\frac{R_{max}}{1000}$</th>
<th>Institution-Site</th>
<th>Country</th>
<th>Year</th>
<th>Area of Focus</th>
<th>FPrice</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NEC</td>
<td>Earth Simulator</td>
<td>35.84</td>
<td>Earth Simulator Center</td>
<td>Japan</td>
<td>2002</td>
<td>Research</td>
<td>5120</td>
</tr>
<tr>
<td>2</td>
<td>HP</td>
<td>ASCI Alphawave SC</td>
<td>7.71</td>
<td>Los Alamos National Laboratory</td>
<td>USA</td>
<td>2002</td>
<td>Research</td>
<td>4096</td>
</tr>
<tr>
<td>3</td>
<td>HP</td>
<td>ASCI Alphawave SC</td>
<td>7.71</td>
<td>Los Alamos National Laboratory</td>
<td>USA</td>
<td>2002</td>
<td>Research</td>
<td>4096</td>
</tr>
<tr>
<td>4</td>
<td>IBM</td>
<td>ASCI Alphawave SC</td>
<td>7.22</td>
<td>Los Alamos National Laboratory</td>
<td>USA</td>
<td>2000</td>
<td>Research</td>
<td>8192</td>
</tr>
<tr>
<td>5</td>
<td>Linux Network</td>
<td>MCR Cluster</td>
<td>5.60</td>
<td>Los Alamos National Laboratory</td>
<td>USA</td>
<td>2002</td>
<td>Research</td>
<td>8192</td>
</tr>
<tr>
<td>6</td>
<td>HP</td>
<td>Alphawave SC (36MFlop/s)</td>
<td>4.66</td>
<td>Pittsburgh Supercomputing Center</td>
<td>USA</td>
<td>2001</td>
<td>Academia</td>
<td>3016</td>
</tr>
<tr>
<td>7</td>
<td>HP</td>
<td>Alphawave SC (36MFlop/s)</td>
<td>3.18</td>
<td>Los Alamos National Laboratory</td>
<td>USA</td>
<td>2001</td>
<td>Research</td>
<td>2930</td>
</tr>
<tr>
<td>8</td>
<td>IBM</td>
<td>ASCI Red Xeon Cluster (100 MFlop/s)</td>
<td>3.14</td>
<td>Los Alamos National Laboratory</td>
<td>USA</td>
<td>2002</td>
<td>Research</td>
<td>1574</td>
</tr>
<tr>
<td>9</td>
<td>IBM</td>
<td>pSeries 690 Turbo</td>
<td>3.16</td>
<td>CCRC</td>
<td>UK</td>
<td>2002</td>
<td>Academia</td>
<td>1208</td>
</tr>
<tr>
<td>10</td>
<td>IBM</td>
<td>pSeries 690 Turbo</td>
<td>3.16</td>
<td>NOC (National Center for Atmospheric Research)</td>
<td>USA</td>
<td>2002</td>
<td>Research</td>
<td>1208</td>
</tr>
</tbody>
</table>

182 fell off, 500 was 318 in June

Performance Extrapolation

TOP500 - Performance
Customer Type

Chip Technology

Processor Type

Architectures

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.
Advantages:

Virtual Environments

Do they make any sense?

Performance Numbers on RISC Processors

<table>
<thead>
<tr>
<th>Processor</th>
<th>Cycle Time</th>
<th>Linpack n=100</th>
<th>Linpack n=1000</th>
<th>Peak</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel P6</td>
<td>2500</td>
<td>105 (29%)</td>
<td>1255 (15%)</td>
<td>1255</td>
</tr>
<tr>
<td>Intel/HP Station 2</td>
<td>1940</td>
<td>152 (35%)</td>
<td>1556 (20%)</td>
<td>1556</td>
</tr>
<tr>
<td>Compaq Alpha</td>
<td>1804</td>
<td>121 (45%)</td>
<td>1342 (37%)</td>
<td>1342</td>
</tr>
<tr>
<td>AMD Athlon</td>
<td>2000</td>
<td>100 (25%)</td>
<td>1000 (15%)</td>
<td>1000</td>
</tr>
<tr>
<td>MIPS</td>
<td>1510</td>
<td>140 (34%)</td>
<td>1400 (15%)</td>
<td>1400</td>
</tr>
<tr>
<td>IBM Power 3</td>
<td>1700</td>
<td>125 (25%)</td>
<td>1250 (75%)</td>
<td>1250</td>
</tr>
<tr>
<td>Intel P3</td>
<td>1900</td>
<td>125 (25%)</td>
<td>1250 (75%)</td>
<td>1250</td>
</tr>
<tr>
<td>Pentium PC G4</td>
<td>1550</td>
<td>175 (25%)</td>
<td>1750 (75%)</td>
<td>1750</td>
</tr>
<tr>
<td>Sun Ultra 80</td>
<td>1560</td>
<td>150 (25%)</td>
<td>1500 (75%)</td>
<td>1500</td>
</tr>
<tr>
<td>SGI Origin 2K</td>
<td>1570</td>
<td>150 (25%)</td>
<td>1500 (75%)</td>
<td>1500</td>
</tr>
<tr>
<td>Cray T3D</td>
<td>1590</td>
<td>175 (25%)</td>
<td>1750 (75%)</td>
<td>1750</td>
</tr>
<tr>
<td>Cray U90</td>
<td>1600</td>
<td>150 (25%)</td>
<td>1500 (75%)</td>
<td>1500</td>
</tr>
<tr>
<td>Cray XMP</td>
<td>1610</td>
<td>175 (25%)</td>
<td>1750 (75%)</td>
<td>1750</td>
</tr>
<tr>
<td>Cray J90</td>
<td>1620</td>
<td>150 (25%)</td>
<td>1500 (75%)</td>
<td>1500</td>
</tr>
<tr>
<td>Cray Euria</td>
<td>1630</td>
<td>175 (25%)</td>
<td>1750 (75%)</td>
<td>1750</td>
</tr>
</tbody>
</table>

High-Performance Computing Directions:

Beowulf-class PC Clusters

Definition:

Message Passing Computing
Open Source Unix
COTS LAN/SAN
COTS PC Nodes
Interconnect
Ethernet, Myrinet, Gigabit, ATM
COTS LAN/SAN
Interconnect
Ethernet, Myrinet, Gigabit, ATM
Open Source Unix
Linux, BSD
Message Passing Computing
MPC, PVM
HPF
LINPACK
Beowulf-class PC Clusters

Advantages:

- Best price-performance
- Low entry-level
- 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.

Distributed and Parallel Systems

- Gather (unused) resources
- Shared cycles
- System SW manages resources
- System SW gains 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?
Performance Improvements for Scientific Computing Problems

- Derived from Computational Methods

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed-Up Factor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Highly Parallel Supercomputing: Where Are We?

- Performance:
  - Sustained performance has dramatically increased during the last 10 years.
  - 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 scientists.

- 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)
- New 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, HiPPI, 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

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

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

- A Pflop for 1 second $= \approx$ 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

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