CS 267: Distributed Memory Computers

Kathy Yelick

http://www.cs.berkeley.edu/~yelick/cs267
Recap of Last Lectures

• Shared memory multiprocessors
  • Caches in individual processors must be kept coherent -- multiple cached copies of same location must be kept equal.
  • Requires clever hardware (see CS258).
  • Distant memory much more expensive to access.

• Shared memory programming
  • Starting, stopping threads.
  • Synchronization with barriers, locks.

• Distributed memory programming
  • MPI (message passing interface)
  • Lecture Wednesday by Bill Saphir from LBNL
Outline

• Distributed Memory Architectures
  • Topologies
  • Cost models

• Trends in High End Machines
  • Clusters today
  • Top500 data
Historical Perspective

• Early machines were:
  • Collection of microprocessors.
  • Communication was performed using bi-directional queues between nearest neighbors.
• Messages were forwarded by processors on path.
  • “Store and forward” networking
• There was a strong emphasis on topology in algorithms, in order to minimize the number of hops.
Network Analogy

• To have a large number of transfers occurring at once, you need a large number of distinct wires.

• Networks are like streets:
  • Link = street.
  • Switch = intersection.
  • Distances (hops) = number of blocks traveled.
  • Routing algorithm = travel plan.

• Properties:
  • Latency: how long to get between nodes in the network.
  • Bandwidth: how much data can be moved per unit time.
    • Bandwidth is limited by the number of wires and the rate at which each wire can accept data.
Characteristics of a Network

- **Topology** (how things are connected)
  - Crossbar, ring, 2-D and 2-D torus, hypercube, omega network.
- **Routing algorithm**:
  - Example: all east-west then all north-south (avoids deadlock).
- **Switching strategy**:
  - Circuit switching: full path reserved for entire message, like the telephone.
  - Packet switching: message broken into separately-routed packets, like the post office.
- **Flow control** (what if there is congestion):
  - Stall, store data temporarily in buffers, re-route data to other nodes, tell source node to temporarily halt, discard, etc.
Properties of a Network: Latency

- **Diameter**: the maximum (over all pairs of nodes) of the shortest path between a given pair of nodes.
- **Latency**: delay between send and receive times
  - Latency tends to vary widely across architectures
  - Vendors often report hardware latencies (wire time)
  - Application programmers care about software latencies (user program to user program)
- **Observations**:
  - Hardware/software latencies often differ by 1-2 orders of magnitude
  - Maximum hardware latency varies with diameter, but the variation in software latency is usually negligible
- Latency is important for programs with many small messages
Properties of a Network: Bandwidth

• A network is partitioned into two or more disjoint subgraphs if some nodes cannot reach others.
• The bandwidth of a link = \( w \times \frac{1}{t} \)
  - \( w \) is the number of wires
  - \( t \) is the time per bit
• Bandwidth typically in Gigabytes (GB), i.e., \( 8 \times 2^{20} \) bits
• Effective bandwidth is usually lower than physical link bandwidth due to packet overhead.

<table>
<thead>
<tr>
<th>Trailer</th>
<th>Error code</th>
<th>Data payload</th>
<th>Routing and control header</th>
</tr>
</thead>
</table>

• Bandwidth is important for applications with mostly large messages
Properties of a Network: Bisection Bandwidth:

• **Bisection bandwidth**: bandwidth across smallest cut that divides network into two equal halves

• Bandwidth across “narrowest” part of the network

- bisection cut

\[ \text{bisection bw} = \text{link bw} \]

- bisection cut

\[ \text{bisection bw} = \sqrt{n} \times \text{link bw} \]

• Bisection bandwidth is important for algorithms in which all processors need to communicate with all others
Network Topology

• In the past, there was considerable research in network topology and in mapping algorithms to topology.
  • Key cost to be minimized: number of “hops” between nodes (e.g. “store and forward”)
  • Modern networks hide hop cost (i.e., “wormhole routing”), so topology is no longer a major factor in algorithm performance.
• Example: On IBM SP system, hardware latency varies from 0.5 usec to 1.5 usec, but user-level message passing latency is roughly 36 usec.
• Need some background in network topology
  • Algorithms may have a communication topology
  • Topology affects bisection bandwidth.
Linear and Ring Topologies

• Linear array

  • Diameter = n-1; average distance ~n/3.
  • Bisection bandwidth = 1 (units are link bandwidth).

• Torus or Ring

  • Diameter = n/2; average distance ~ n/4.
  • Bisection bandwidth = 2.
  • Natural for algorithms that work with 1D arrays.
Meshes and Tori

Two dimensional mesh

- Diameter = \(2 \times (\sqrt{n} - 1)\)
- Bisection bandwidth = \(\sqrt{n}\)

Two dimensional torus

- Diameter = \(\sqrt{n}\)
- Bisection bandwidth = \(2 \times \sqrt{n}\)

- Generalizes to higher dimensions (Cray T3D used 3D Torus).
- Natural for algorithms that work with 2D and/or 3D arrays.
Hypercubes

- Number of nodes $n = 2^d$ for dimension $d$.
  - Diameter $= d$.
  - Bisection bandwidth $= n/2$.

- Popular in early machines (Intel iPSC, NCUBE).
  - Lots of clever algorithms.
  - See 1996 online 267 notes.

- Greycode addressing:
  - Each node connected to $d$ others with 1 bit different.
Trees

- Diameter = $\log n$.
- Bisection bandwidth = 1.
- Easy layout as planar graph.
- Many tree algorithms (e.g., summation).
- Fat trees avoid bisection bandwidth problem:
  - More (or wider) links near top.
  - Example: Thinking Machines CM-5.
Butterflies

- Diameter = log n.
- Bisection bandwidth = n.
- Cost: lots of wires.
- Used in BBN Butterfly.
- Natural for FFT.

butterfly switch

multistage butterfly network
# Topologies in Real Machines

<table>
<thead>
<tr>
<th>Topology Description</th>
<th>Topology Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red Storm (Opteron + Cray network, future)</td>
<td>3D Mesh</td>
</tr>
<tr>
<td>Blue Gene/L</td>
<td>3D Torus</td>
</tr>
<tr>
<td>SGI Altix</td>
<td>Fat tree</td>
</tr>
<tr>
<td>Cray X1</td>
<td>4D Hypercube*</td>
</tr>
<tr>
<td>Myricom (Millennium)</td>
<td>Arbitrary</td>
</tr>
<tr>
<td>Quadrics (in HP Alpha server clusters)</td>
<td>Fat tree</td>
</tr>
<tr>
<td>IBM SP</td>
<td>Fat tree (approx)</td>
</tr>
<tr>
<td>SGI Origin</td>
<td>Hypercube</td>
</tr>
<tr>
<td>Intel Paragon (old)</td>
<td>2D Mesh</td>
</tr>
<tr>
<td>BBN Butterfly (really old)</td>
<td>Butterfly</td>
</tr>
</tbody>
</table>

Many of these are approximations: E.g., the X1 is really a “quad bristled hypercube” and some of the fat tree are not as fat as they should be at the top.
Evolution of Distributed Memory Machines

• Special queue connections are being replaced by direct memory access (DMA):
  • Processor packs or copies messages.
  • Initiates transfer, goes on computing.

• Message passing libraries provide store-and-forward abstraction:
  • Can send/receive between any pair of nodes, not just along one wire.
  • Time proportional to distance since each processor along path must participate.

• Wormhole routing in hardware:
  • Special message processors do not interrupt main processors along path.
  • Message sends are pipelined.
  • Processors don’t wait for complete message before forwarding.
Performance Models
PRAM

• Parallel Random Access Memory.
• All memory access operations complete in one clock period -- no concept of memory hierarchy (“too good to be true”).
  • OK for understanding whether an algorithm has enough parallelism at all.
  • Parallel algorithm design strategy: first do a PRAM algorithm, then worry about memory/communication time (sometimes works)
• Slightly more realistic: Concurrent Read Exclusive Write (CREW) PRAM.
Latency and Bandwidth Model

• Time to send message of length n is roughly.

\[
\text{Time} = \text{latency} + n \cdot \text{cost\_per\_word} = \text{latency} + n / \text{bandwidth}
\]

• Topology is assumed irrelevant.
• Often called “α–β model” and written

\[
\text{Time} = \alpha + n \cdot \beta
\]

• Usually \( \alpha >> \beta >> \text{time per flop} \).
  • One long message is cheaper than many short ones.

\[
\alpha + n \cdot \beta << n \cdot (\alpha + 1 \cdot \beta)
\]

• Can do hundreds or thousands of flops for cost of one message.
• Lesson: Need large computation-to-communication ratio to be efficient.
Alpha-Beta Parameters on Current Machines

• These numbers were obtained empirically

<table>
<thead>
<tr>
<th>machine</th>
<th>α</th>
<th>β</th>
</tr>
</thead>
<tbody>
<tr>
<td>T3E/Shm</td>
<td>1.2</td>
<td>0.003</td>
</tr>
<tr>
<td>T3E/MPI</td>
<td>6.7</td>
<td>0.003</td>
</tr>
<tr>
<td>IBM/LAPI</td>
<td>9.4</td>
<td>0.003</td>
</tr>
<tr>
<td>IBM/MPI</td>
<td>7.6</td>
<td>0.004</td>
</tr>
<tr>
<td>Quadrics/Get</td>
<td>3.267</td>
<td>0.00498</td>
</tr>
<tr>
<td>Quadrics/Shm</td>
<td>1.3</td>
<td>0.005</td>
</tr>
<tr>
<td>Quadrics/MPI</td>
<td>7.3</td>
<td>0.005</td>
</tr>
<tr>
<td>Myrinet/GM</td>
<td>7.7</td>
<td>0.005</td>
</tr>
<tr>
<td>Myrinet/MPI</td>
<td>7.2</td>
<td>0.006</td>
</tr>
<tr>
<td>Dolphin/MPI</td>
<td>7.767</td>
<td>0.00529</td>
</tr>
<tr>
<td>Giganet/VIPL</td>
<td>3.0</td>
<td>0.010</td>
</tr>
<tr>
<td>GigE/VIPL</td>
<td>4.6</td>
<td>0.008</td>
</tr>
<tr>
<td>GigE/MPI</td>
<td>5.854</td>
<td>0.00872</td>
</tr>
</tbody>
</table>

α is in usecs  
β is usecs per Byte

How well does the model  
\[ \text{Time} = \alpha + n \beta \]
predict actual performance?
Model Time Varying Message Size & Machines

Sum of model size vs. machines

- T3E/Shm
- T3E/MPI
- IBM/LAPI
- IBM/MPI
- Quadrics/Shm
- Quadrics/MPI
- Myrinet/GM
- Myrinet/MPI
- GigE/VIPL
- GigE/MPI

2/11/2004
CS267 Lecture 7
LogP Parameters: Overhead & Latency

- Non-overlapping overhead
  - $o_{\text{send}}$
  - $L$
  - $o_{\text{recv}}$

\[
EEL = o_{\text{send}} + L + o_{\text{recv}}
\]

- Send and recv overhead can overlap
  - $o_{\text{send}}$
  - $L$
  - $o_{\text{recv}}$

\[
EEL = f(o_{\text{send}}, L, o_{\text{recv}})
\]
LogP Parameters: gap

- The Gap is the delay between sending messages
- Gap could be larger than send ovhd
  - NIC may be busy finishing the processing of last message and cannot accept a new one.
  - Flow control or backpressure on the network may prevent the NIC from accepting the next message to send.
- The gap represents the inverse bandwidth of the network for small message sends.
Results: EEL and Overhead

![Graph showing overhead and latency for different systems and operations.](image-url)

- Send Overhead (alone)
- Send & Rec Overhead
- Rec Overhead (alone)
- Added Latency

Data from Mike Welcome, NERSC
Limitations of the LogP Model

- The LogP model has a fixed cost for each messages
  - This is useful in showing how to quick broadcast a single word
  - Other examples also in the LogP papers
- For larger messages, there is a variation LogGP
  - Two gap parameters, one for small and one for large message
  - The large message gap is the $\beta$ in our previous model
- No topology considerations (including no limits for bisection bandwidth)
  - Assumes a fully connected network
  - For some algorithms with nearest neighbor communication, but with “all-to-all” communication we need to refine this further
- This is a flat model, i.e., each processor is connected to the network
  - Clusters of SMPs are not accurately models
Trends in Real Machines
End to End Latency Over Time

- Latency has not improved significantly
  - T3E (shmem) was lowest point
  - Federation in 2003 will not reach that level – 7 years later!

Data from Kathy Yelick, UCB and NERSC
Send Overhead Over Time

- Overhead has not improved significantly; T3D was best
  - Lack of integration; lack of attention in software

Data from Kathy Yelick, UCB and NERSC
- Listing of the 500 most powerful Computers in the World

- Yardstick: $R_{\text{max}}$ from Linpack

\[ Ax = b, \text{ dense problem} \]

- Updated twice a year:
  - ISC‘xy in Germany, June xy
  - SC‘xy in USA, November xy

- All data available from [www.top500.org](http://www.top500.org)
TOP500 list - Data shown

- Manufacturer: Manufacturer or vendor
- Computer Type: indicated by manufacturer or vendor
- Installation Site: Customer
- Location: Location and country
- Year: Year of installation/last major update
- # Processors: Number of processors
- $R_{\text{max}}$: Maxmimal LINPACK performance achieved
- $R_{\text{peak}}$: Theoretical peak performance
- $N_{\text{max}}$: Problem size for achieving $R_{\text{max}}$
- $N_{1/2}$: Problem size for achieving half of $R_{\text{max}}$
- $N_{\text{world}}$: Position within the TOP500 ranking
## 22nd List: The TOP10

<table>
<thead>
<tr>
<th>Rank</th>
<th>Manufacturer</th>
<th>Computer</th>
<th>$R_{\text{max}}$ [TF/s]</th>
<th>Installation Site</th>
<th>Country</th>
<th>Year</th>
<th>Area of Installation</th>
<th># Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NEC</td>
<td>Earth-Simulator</td>
<td>35.86</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 Q AlphaServer SC</td>
<td>13.88</td>
<td>Los Alamos National Laboratory</td>
<td>USA</td>
<td>2002</td>
<td>Research</td>
<td>8192</td>
</tr>
<tr>
<td>3</td>
<td>Self-Made</td>
<td>Apple G5, Mellanox</td>
<td>10.28</td>
<td>Virginia Tech</td>
<td>USA</td>
<td>2003</td>
<td>Academic</td>
<td>2200</td>
</tr>
<tr>
<td>4</td>
<td>Dell</td>
<td>Tungsten PowerEdge, Myrinet</td>
<td>9.82</td>
<td>NCSA</td>
<td>USA</td>
<td>2003</td>
<td>Academic</td>
<td>2500</td>
</tr>
<tr>
<td>5</td>
<td>HP</td>
<td>Mpp2, Integrity rx2600 Itanium2, Qadrics</td>
<td>8.63</td>
<td>Pacific Northwest National Laboratory</td>
<td>USA</td>
<td>2003</td>
<td>Research</td>
<td>1936</td>
</tr>
<tr>
<td>6</td>
<td>Linux Networx</td>
<td>Lightning, Opteron, Myrinet</td>
<td>8.05</td>
<td>Los Alamos National Laboratory</td>
<td>USA</td>
<td>2003</td>
<td>Research</td>
<td>2816</td>
</tr>
<tr>
<td>7</td>
<td>Linux Networx/ Quadrics</td>
<td>MCR Cluster</td>
<td>7.63</td>
<td>Lawrence Livermore National Laboratory</td>
<td>USA</td>
<td>2002</td>
<td>Research</td>
<td>2304</td>
</tr>
<tr>
<td>8</td>
<td>IBM</td>
<td>ASCI White SP Power3</td>
<td>7.3</td>
<td>Lawrence Livermore National Laboratory</td>
<td>USA</td>
<td>2000</td>
<td>Research</td>
<td>8192</td>
</tr>
<tr>
<td>9</td>
<td>IBM</td>
<td>Seaborg SP Power 3</td>
<td>7.3</td>
<td>NERSC Lawrence Berkeley Nat. Lab.</td>
<td>USA</td>
<td>2002</td>
<td>Research</td>
<td>6656</td>
</tr>
<tr>
<td>10</td>
<td>IBM/Quadrics</td>
<td>xSeries Cluster Xeon 2.4 GHz</td>
<td>6.59</td>
<td>Lawrence Livermore National Laboratory</td>
<td>USA</td>
<td>2003</td>
<td>Research</td>
<td>1920</td>
</tr>
</tbody>
</table>
Projected Performance Development

- #1
- #500
- Sum
- #1 Trend Line
- #500 Trend Line
- Sum Trend Line

Performance development over time:
- 1993: 0.42 GF
- 1994: 1.12 TF
- 1995: 59.67 GF
- 1996: 35.86 TF
- 1997: 100 TFlops
- 1998: 529.61 TF
- 1999: 100 PFlops
- 2000: 100 PFlops
- 2001: 100 PFlops
- 2002: 100 PFlops
- 2003: 100 PFlops
- 2004: 100 PFlops
- 2005: 100 PFlops
- 2006: 100 PFlops
- 2007: 100 PFlops
- 2008: 100 PFlops
- 2009: 100 PFlops
- 2010: 100 PFlops
Customer Segment / Systems

- Government
- Vendor
- Classified
- Academic
- Research
- Industry

Systems

Processor Family / Systems

- Others
- Nec
- Fujitsu
- Intel
- Alpha
- HP
- Cray
- Sparc
- MIPS
- Power

Systems

1993 - 2003

22nd List / Nov 2003
http://www.top500.org/
Analysis of TOP500 Data

- Annual performance growth about a factor of 1.82
- Two factors contribute almost equally to the annual total performance growth
  - Processor number grows per year on the average by a factor of 1.30 and the
  - Processor performance grows by 1.40 compared to 1.58 of Moore's Law

Limits to Cluster Based Systems for HPC

- Memory Bandwidth
  - Commodity memory interfaces [SDRAM, RDRAM, DDRAM]
  - Separation of memory and CPU implementations limits performance

- Communications fabric/CPU/Memory Integration
  - Current networks are attached via I/O devices
  - Limits bandwidth and latency and communication semantics

- Node and system packaging density
  - Commodity components and cooling technologies limit densities
  - Blade based servers moving in right direction but are not High Performance

- Ad Hoc Large-scale Systems Architecture
  - Little functionality for RAS
  - Lack of systems software for production environment

- ... but departmental and single applications clusters will be highly successful

After Rick Stevens, Argonne
## Comparison Between Architectures (2001)

<table>
<thead>
<tr>
<th></th>
<th>Alvarez</th>
<th>Seaborg</th>
<th>Mcurie</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Pentium III</td>
<td>Power 3</td>
<td>EV-5</td>
</tr>
<tr>
<td>Clock speed</td>
<td>867</td>
<td>375</td>
<td>450</td>
</tr>
<tr>
<td># nodes</td>
<td>80</td>
<td>184</td>
<td>644</td>
</tr>
<tr>
<td># processors/node</td>
<td>2</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Peak (GF/s)</td>
<td>139</td>
<td>4416</td>
<td>579.6</td>
</tr>
<tr>
<td>Memory (GB/node)</td>
<td>1</td>
<td>16-64</td>
<td>0.256</td>
</tr>
<tr>
<td>Interconnect</td>
<td>Myrinet 2000</td>
<td>Colony</td>
<td>T3E</td>
</tr>
<tr>
<td>Disk (TB)</td>
<td>1.5</td>
<td>20</td>
<td>2.5</td>
</tr>
</tbody>
</table>

Source: Tammy Welcome, NERSC
# Performance Comparison (2)

## Class C NPBs

<table>
<thead>
<tr>
<th></th>
<th>Alvarez</th>
<th>Seaborg</th>
<th>Mcurie</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>64</td>
<td>128</td>
<td>64</td>
</tr>
<tr>
<td>BT</td>
<td>61.0</td>
<td>111.9</td>
<td>55.7</td>
</tr>
<tr>
<td>CG</td>
<td>17.1</td>
<td>34.0</td>
<td>9.3</td>
</tr>
<tr>
<td>EP</td>
<td>3.9</td>
<td>3.9</td>
<td>2.6</td>
</tr>
<tr>
<td>FT</td>
<td>31.3</td>
<td>61.2</td>
<td>30.8</td>
</tr>
<tr>
<td>IS</td>
<td>2.4</td>
<td>2.1</td>
<td>1.1</td>
</tr>
<tr>
<td>LU</td>
<td>26.9</td>
<td>209.0</td>
<td>60.4</td>
</tr>
<tr>
<td>MG</td>
<td>56.6</td>
<td>133.2</td>
<td>93.9</td>
</tr>
<tr>
<td>SP</td>
<td>40.9</td>
<td>100.7</td>
<td>41.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>per processor</th>
<th>SSP (Gflops/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alvarez</td>
<td>39.0</td>
<td>6.2</td>
</tr>
<tr>
<td>Seaborg</td>
<td>108.3</td>
<td>318.9</td>
</tr>
<tr>
<td>Mcurie</td>
<td>48.7</td>
<td>31.3</td>
</tr>
</tbody>
</table>

Source: Tammy Welcome, NERSC
Summary – Wrap-up

• Network structure and concepts
  • Switching, routing, flow control
  • Topology, bandwidth, latency, bisection bandwidth, diameter

• Performance models
  • PRAM, $\alpha - \beta$, and LogP

• Workstation/PC clusters
  • Programming environment, hardware
  • Challenges

• Message passing implementation
Extra Slides
Effectiveness of Commodity PC Clusters

• Dollars/performance based on peak
  • SP and Alvarez are comparable $/TF
• Get lower % of peak on Alvarez than SP
  • Based on SSP, 4.5% versus 7.2% for FP intensive applications
  • Based on sequential NPBs, 5-13.8% versus 6.3-21.6% for FP intensive applications
  • x86 known not to perform well on FP intensive applications
• $/Performance and cost of ownership need to be examined much more closely
  • Above numbers do not take into account differences in system balance or configuration
  • SP was aggressively priced
  • Alvarez was vendor-integrated, not self-integrated
Message Passing Libraries

• Many “message passing libraries” available
  • Chameleon, from ANL.
  • CMMD, from Thinking Machines.
  • Express, commercial.
  • MPL, native library on IBM SP-2.
  • NX, native library on Intel Paragon.
  • Zipcode, from LLL.
  • PVM, Parallel Virtual Machine, public, from ORNL/UTK.
• Others...
  • MPI, Message Passing Interface, now the industry standard.

• Need standards to write portable code.
• Rest of this discussion independent of which library.
• Lecture 7 was detailed MPI lecture
Workstation/PC Clusters

• Reaction to commercial MPPs:
  • build parallel machines out of commodity components
  • Inexpensive workstations or PCs as computing nodes
  • Fast (gigabit) switched network between nodes

• Benefits:
  • 10x - 100x cheaper for comparable performance
  • Standard OS on each node
  • Follow commodity tech trends
  • Incrementally upgradable and scalable
  • Fault tolerance

• Trends:
  • Berkeley NOW (1994): 100 UltraSPARCIs, Myrinet
  • ASCI RED (1997): 4510 dual Pentium II nodes, custom network
  • Millennium (1999): 100+ dual/quad Pentium IIs, Myrinet
  • Google (2001): 8000+ node Linux cluster, ??? network
Implementing Synchronous Message Passing

- Send operations complete after matching receive and source data has been sent.
- Receive operations complete after data transfer is complete from matching send.

1) Initiate send
2) Address translation on $P_{dest}$
3) Send-Ready Request
4) Remote check for posted receive
5) Reply transaction
6) Bulk data transfer
**Example: Permuting Data**

° **Exchanging data between Procs 0 and 1, V.1: What goes wrong?**

  Processor 0
  send(1, item0, 1, tag1)
  recv(1, item1, 1, tag2)

  Processor 1
  send(0, item1, 1, tag2)
  recv(0, item0, 1, tag1)

° **Deadlock**

° **Exchanging data between Proc 0 and 1, V.2:**

  Processor 0
  send(1, item0, 1, tag1)
  recv(1, item1, 1, tag2)

  Processor 1
  recv(0, item0, 1, tag1)
  send(0, item1, 1, tag2)

° **What about a general permutation, where Proc j wants to send to Proc s(j), where s(1), s(2), ..., s(P) is a permutation of 1, 2, ..., P?**
Implementing Asynchronous Message Passing

- Optimistic single-phase protocol assumes the destination can buffer data on demand.

1) Initiate send
2) Address translation on $P_{dest}$
3) Send Data Request

4) Remote check for posted receive
5) Allocate buffer (if check failed)
6) Bulk data transfer
Safe Asynchronous Message Passing

- Use 3-phase protocol
- Buffer on sending side
- Variations on send completion
  - wait until data copied from user to system buffer
  - don’t wait -- let the user beware of modifying data

1) Initiate send
2) Address translation on $P_{\text{dest}}$
3) Send-Ready Request
4) Remote check for posted receive record send-rdy
5) Reply transaction
6) Bulk data transfer
Example Revisited: Permuting Data

° Processor j sends item to Processor $s(j)$, where $s(1),\ldots,s(P)$ is a permutation of $1,\ldots,P$

Processor j

send_asynch(s(j), item, 1, tag)
recv_block( ANY, item, 1, tag)

° What could go wrong?
° Need to understand semantics of send and receive.
° Many flavors available.
Other operations besides send/receive

• “Collective Communication” (more than 2 procs)
  • Broadcast data from one processor to all others.
  • Barrier.
  • Reductions (sum, product, max, min, boolean and, #, …), where # is any “associative” operation.
  • Scatter/Gather.
  • Parallel prefix -- Proc j owns x(j) and computes y(j) = x(1) # x(2) # … # x(j).
  • Can apply to all other processors, or a user-define subset.
  • Cost = O(log P) using a tree.

• Status operations
  • Enquire about/Wait for asynchronous send/receives to complete.
  • How many processors are there?
  • What is my processor number?
Example: Sharks and Fish

• N fish on P procs, N/P fish per processor
  • At each time step, compute forces on fish and move them
• Need to compute gravitational interaction
  • In usual n^2 algorithm, every fish depends on every other fish.
  • Every fish needs to “visit” every processor, even if it “lives” on just one.
• What is the cost?
Two Algorithms for Gravity: What are their costs?

Algorithm 1

Copy local Fish array of length N/P to Tmp array
for j = 1 to N
  for k = 1 to N/P, Compute force of Tmp(k) on Fish(k)
    “Rotate” Tmp by 1
      for k=2 to N/P, Tmp(k) <= Tmp(k-1)
      recv(my_proc - 1,Tmp(1))
      send(my_proc+1,Tmp(N/P))

Algorithm 2

Copy local Fish array of length N/P to Tmp array
for j = 1 to P
  for k=1 to N/P, for m=1 to N/P, Compute force of Tmp(k) on Fish(m)
    “Rotate” Tmp by N/P
      recv(my_proc - 1,Tmp(1:N/P))
      send(my_proc+1,Tmp(1:N/P))

What could go wrong? (be careful of overwriting Tmp)
More Algorithms for Gravity

• Algorithm 3 (in sharks and fish code):
  • All processors send their Fish to Proc 0.
  • Proc 0 broadcasts all Fish to all processors.

• Tree-algorithms:
  • Barnes-Hut, Greengard-Rokhlin, Anderson.
  • $O(N \log N)$ instead of $O(N^2)$.
  • Parallelizable with cleverness.
  • “Just” an approximation, but as accurate as you like (often only a few digits are needed, so why pay for more).
  • Same idea works for other problems where effects of distant objects becomes “smooth” or “compressible”:
    • electrostatics, vorticity, …
    • radiosity in graphics.
    • anything satisfying Poisson equation or something like it.
Reading Assignment

• Reading for today
• Next week: Current high performance architectures
  • MPI
• The following week
  • UPC