Outline

- Announcements
  - one-page project writeups due March 3, 1998
  - groups should schedule a meeting with me

- Last time: Processor-network interfaces for clusters
  - case studies: Myrinet, Hamlyn, Memory Channel, SHRIMP
  - conclusion: good performance, but for very low communication semantics

- This lecture: Low-level messaging layers
  - rationale: problems with traditional messaging implementations
  - interface: Active Messages
  - implementation: Fast Messages on Myrinet [Linda Steinberg]
  - impact: how important is low latency/overhead and high bandwidth? [Leonid Zheleznyak]
Message Passing

- Separation of data transfer (what) from location in memory (where)
  - sender specifies source location
  - receiver specifies destination location
- Integration of data transfer and notification (synchronization)

- Messaging layer: Middleware which bridges the application-hardware gap
  - N-to-1 merge
  - buffer management, in-order delivery, reliable delivery, etc.

High-level Messaging Interfaces: MPI

- Point-to-point
  - Blocking (synchronous)
    int MPI_Send( void *buf, int count, MPI_Datatype datatype, int dest,
                  int tag, MPI_Comm comm );
    int MPI_Recv( void *buf, int count, MPI_Datatype datatype, int source,
                  int tag, MPI_Comm comm, MPI_status *status );
  - Non-blocking (asynchronous)
    int MPI_Isend( void *buf, int count, MPI_Datatype datatype, int dest,
                   int tag, MPI_Comm comm, MPI_Request *request );
    int MPI_Irecv( void *buf, int count, MPI_Datatype datatype, int source,
                   int tag, MPI_Comm comm, MPI_status *status, MPI_Request *request );
    int MPI_Test( MPI_request, int *flag, MPI_Status *status );
- Collective operations (broadcast, scatter, gather, reduce)
- Communicators (application topologies)
- Dynamic processes (MPI-2)

- Primary focus is on providing a portable, standard interface
Problems with direct implementation

- 3-phase protocol
- user-level must ensure good send/recv placement

**Synchronous**

**Proc P**

Send s, Q

Recv Q, v

**Proc Q**

Recv P, t

Send u, P

**Asynchronous**

**Proc P**

Send s, Q

Recv Q, v

**Proc Q**

Recv P, t

Send u, P

• copying
• buffer management
• matching/selection

2/24/98 5

Messaging Costs [ Culler’94 ]

O_{send} \rightarrow xfer \leftarrow network latency \rightarrow O_{recv}

"time to squeeze message onto thinnest wire": n/BW = n^*G

<table>
<thead>
<tr>
<th>Machine</th>
<th>Year</th>
<th>Overhead μs</th>
<th>L μs/hop</th>
<th>G μs/word</th>
<th>Overhead cycles</th>
<th>FLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>nCUBE/2</td>
<td>90</td>
<td>150</td>
<td>2.3</td>
<td>1.8</td>
<td>3000</td>
<td>330</td>
</tr>
<tr>
<td>iPSC/860</td>
<td>91</td>
<td>160</td>
<td>11</td>
<td>1.4</td>
<td>6400</td>
<td>3200</td>
</tr>
<tr>
<td>Delta</td>
<td>91</td>
<td>55</td>
<td>0.5</td>
<td>0.4</td>
<td>2100</td>
<td>1100</td>
</tr>
<tr>
<td>CM5</td>
<td>92</td>
<td>95</td>
<td>0.2</td>
<td>0.4</td>
<td>3200</td>
<td>310</td>
</tr>
<tr>
<td>Paragon</td>
<td>93</td>
<td>250</td>
<td>0.1</td>
<td>0.1</td>
<td>12500</td>
<td>6500</td>
</tr>
<tr>
<td>SP-1</td>
<td>93</td>
<td>45</td>
<td>0.1</td>
<td>0.5</td>
<td>3000</td>
<td>1000</td>
</tr>
</tbody>
</table>

2/24/98 6
Active Messages

- Sender injects the message directly into the network
  - blocks till network accepts message
- Associates a small user-level handler with each message
- Handler executes immediately upon arrival
  - pulls message out of the network, and integrates it into ongoing computation (as required by higher level programming model)

Implementation Costs of Active Messages

<table>
<thead>
<tr>
<th></th>
<th>nCUBE/2 (vs. 150µs)</th>
<th>CM5 (vs. 95µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Send</td>
<td>Send</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1.75</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1.25</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Active Messages: Where are the savings?

- View the network as a pipeline
  - rate determined by communication overhead
  - latency related to message length, network depth

- Asynchronous interface
  - sender launches message into network and continues computing
  - receiver retrieves message from the network
  - *computation continues while message in flight*

- Minimal buffering costs
  - sender waits till network can accept message
  - handler executes immediately on arrival
  - i.e., savings in buffer allocation, management

- Minimal parsing costs (matching/selection)
  - handlers executed in order of message reception

Active Messages: What it does not provide

- High-level guarantees which simplify user code
  - in-order delivery
    - no effort made to reorder messages in adaptive networks
  - reliable delivery
    - sender only waits for network to accept messages
    - higher-level protocols must ensure that message data reaches destination
  - sender-receiver decoupling
    - sender blocks if receiver is not responsive enough in clearing the network

- Building these guarantees in user code results in 2x cost increase
  [ *Software Overheads in Messaging Layers: Where Does the Time Go?* ]

- Fast Messages: Active Messages + above guarantees
  - careful implementation keeps costs close to raw hardware level
Lecture Summary

- Traditional messaging implementations incur high costs
  - buffer management, scheduling, notification
- Active Messages (AM)
  - “RISC” approach to communication
  - integration of communication with computation
- Fast Messages (FM)
  - AM + guarantees (in-order, reliable, decoupling)
  - implementation delivers hardware performance to applications
  - supports higher-level layers without sacrificing performance
- Impact of overhead, latency, and bandwidth
  - low overhead critical for performance
  - latency and bandwidth less so
    - true for explicit message-passing programs
    - what about shared-memory programs?

Next Lecture

- Higher-level messaging layers (protection, fault-tolerance)
  - VMMC-2: Efficient Support for Reliable Connection-Oriented Communication, Dubnicki et al.
  - Incorporating Memory Management into User-Level Network Interfaces, Basu et al.
  - Virtual Interface Architecture Specification, Version 1.0
    [ register online at www.viarch.org; read only pages 11-35 ]