AN O/S PERSPECTIVE ON NETWORKS

Adem Efe Gencer¹

¹Department of Computer Science, Cornell University
Papers


Authors - I

- Thorsten von Eicken
  - Founder of RightScale
  - Ph.D, CS at University of California, Berkeley
  - Assistant Prof. at Cornell (1993-1999)

- David E. Culler
  - Chair, Electrical Engineering and Computer Sciences at Berkeley

- Seth Copen Goldstein
  - Associate Professor at Carnegie Mellon University

- Klaus Erik Schauser
  - Associate professor at UCSB
Authors - II

- Thorsten von Eicken
- Anindya Basu
  - Advised by von Eicken
- Vineet Buch
  - Google
  - MS from Cornell
- Werner Vogels
  - CTO and vice President of Amazon
Active Messages

- Introduction
- Preview: Active Message
  - Traditional programming models
- Active Messages
- Environment
- Split-C
- Message Driven Model vs. Active Messages
- Final Notes
Introduction

- Inefficient use of underlying hardware
  - Poor overlap between communication and computation
  - High communication overhead
Eliminate inefficiency by increasing the overlap between communication & computation!
Solution: A novel asynchronous communication mechanism! **Active messages**

**MESSAGE**

**ACTIVE MESSAGE**

**HEADER**
Address of a userspace handler

**BODY**
Argument of userspace handler
Traditional Programming Models

- Synchronous comm.
  - 3-phase protocol
  - Send / receive are blocking

  - Simple
  - No buffering
  - High comm. latency
  - Underutilized network bandwidth
Traditional Programming Models

- Asynchronous comm.
  - Send / receive are non-blocking
  - Message layer buffers the message

Communication & Computation can overlap
## Traditional Programming Models

<table>
<thead>
<tr>
<th></th>
<th>Synchronous</th>
<th>Asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PROS</strong></td>
<td>No Buffering</td>
<td>Overlap</td>
</tr>
<tr>
<td><strong>CONS</strong></td>
<td>No Overlap</td>
<td>Buffering</td>
</tr>
<tr>
<td></td>
<td>High Comm. Latency</td>
<td></td>
</tr>
</tbody>
</table>

*Active Messages*
Active Messages

- Requires SPMD programming model
- Key optimization: not buffered
  - Except for network transport buffering
  - For large messages buffering is needed on sender side
  - Receiving side pre-allocates structures
- Handlers do not block - deadlock avoidance
Active Messages

- **Sender**
  - Packs the message with a header containing handler’s address
  - Pass the message to the network
  - Continue computing

- **Receiver**
  - Message arrival invokes handler
  - Handler Extracts the data & interrupts computation
  - Passes it to the ongoing computation

- Implementation of active messages on different platforms is not the same (nCube/2 vs. CM-5).
Environment

- Implementation on parallel supercomputers
  - *nCUBE/2*
    - Binary hypercube network
    - NI with 28 DMA channels
  - *CM-5 (connection machine-5)*
    - Hypertree network
- Split-C
  - A parallel extension of C

*CM-5 machine of NSA*
Split-C: PUT and GET

- Nonblocking
  - Message handler
  - Message formatter

- Used in matrix multiplication to show the performance of active message
Split-C: GET

- Matrix multiply achieves 95% performance in nCUBE/2 configuration.

- Xmit: time to inject the message into the network
- Hops: time for the network hops
Split-C: Matrix Multiply

- **m**: # of columns per processor
- # of nodes is constant (N=128)
Message Driven Model vs. Active Messages

- **Message driven model**
  - Computation in message handlers
  - Handler may suspend
  - Memory allocation & scheduling on message arrival

- **Active Messages**
  - Computation in background (handler extracts message)
  - Immediate execution
Final Notes

- Not a new parallel programming paradigm!
  - A primitive communication mechanism
  - Used to implement them efficiently

- The need for userspace handler address leads to programming difficulty

- Requires SPMD model (but not required on some modified versions)
U-Net

- Introduction
- U-Net
  - Remove the Kernel from Critical Path
- Communication in U-Net
- Protection
- Zero Copy
  - True zero copy vs. zero copy
- Environment
- Performance
- Final Notes
Introduction

- Low bandwidth and high communication latency
  - No longer hardware issues!
  - Problem is the message path through the kernel
- Small messages
  - Common in various applications
  - Requires low round-trip latency

\[ Latency = \text{processing overhead} + \text{network latency} \]
Introduction: Objective

Make communication **FASTER!**
U-Net

- Low communication latency
- High bandwidth
  - even with small messages!
- Support for workstations with off-the-shelf network
- Keep providing full protection
  - Kernel controlled channel setup / tear-down
- TCP, UDP, Active Messages, … can be implemented
Remove the kernel from critical path

Solution: Enable user-level access to NI to implement user-level communication protocols

- Does not require OS modification or custom HW
- Provides higher flexibility (app. specific protocols!)

Legend:
- User application
- Operating system kernel
- Network interface
- NI with message multiplex
Communication in U-Net

Each process creates endpoints to access network

- **U-net endpoint**: application handle into the network
- **Communication segment**: regions of memory that hold message data
- **Send/recv/free queues**: hold message descriptors
Communication in U-Net: Send

- Compose the data in the comm. segment
- Push a descriptor to the send queue
  - [Negative feedback]
- [Positive feedback]
  - [NI] Take the message & insert into network
  - [NI] Message injected to network
  - Activate back-pressure mechanism
  - [NI] Send buffer empty flag is set in descriptor
Communication in U-Net: Receive

1. [NI] receive the message
   - identify endpoint
2. Get space from free queue
3. Transfer data into appropriate comm. segment
4. [polling]
   - receive queue has new element
5. [polling]
   - [event driven]
6. Push descriptor to receive queue
7. Using receive queue descriptor read the data
8. Use upcall to signal arrival
Protection

- Owning process protection:
  - Endpoints
  - Communication Segments
  - Send/Receive/Free Queues

- Tag protection:
  - Outgoing message
    - tagged with originating address
  - Incoming message
    - delivered to correct destination point only!
Zero Copy

- **Base level U-Net (zero copy)**
  - Send/receive needs a buffer (not really “zero” copy!)
  - Requires a copy between app. data structures and a buffer in the comm. segment

- **Direct Access U-Net (true zero copy)**
  - Spans on entire address space
  - Has special hardware requirements
Environment

- Implementation on off-the-shelf hardware platform
  - *SPARCStations*
    - Fore SBA-100 NIC
    - Fore SBA-200 NIC
  - SunOS 4.1.3
    - BSD-based Unix OS
      - Ancestor of Solaris (after version 5.0)
  - Split-C
    - A parallel extension of C
Performance: UDP vs. TCP

UDP bandwidth as a function of message size

TCP bandwidth as a function of data generation by application
Performance: UDP vs. TCP

- Fast U-Net roundtrips:
  - Apprx. 7x speedup for TCP
  - Apprx. 5x speedup for UDP

- Fast U-Net TCP roundtrips let use of small window size.
Final Notes

- Low comm. latency
- High bandwidth
- Good flexibility

Cluster of workstations vs. supercomputers

- Comparable performance using U-Net
- Additional system resources (parallel process scheduler, file systems, ...) are needed

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Round-trip latency</th>
<th>Bandwidth 4K packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw AAL5</td>
<td>65μs</td>
<td>120Mbits/s</td>
</tr>
<tr>
<td>Active Msgs</td>
<td>71μs</td>
<td>118Mbits/s</td>
</tr>
<tr>
<td>UDP</td>
<td>138μs</td>
<td>120Mbits/s</td>
</tr>
<tr>
<td>TCP</td>
<td>157μs</td>
<td>115Mbits/s</td>
</tr>
<tr>
<td>Split-C store</td>
<td>72μs</td>
<td>118Mbits/s</td>
</tr>
</tbody>
</table>

Table 3: U-Net latency and bandwidth Summary.