
Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, Klaus Erik Schauser

University of California, Berkeley
Computer Science Division – EECS

Presented by Roberto Starc
Executive Summary

- **Problem** – Communication between processors is slow, and speeding it up sacrifices cost/performance of the system.

- **Goal** - Reduce communication overhead and allow overlapping of communication with computation.

- **Active Messages** - Integrate communication and computation
  - Messages consist of the **address of a user-level handler** at the head, and the **arguments** to be passed as the body.
  - The **handler** gets the message out of the network and into ongoing computation as fast as possible.
  - A simple mechanism close to hardware that can be used to implement existing parallel programming paradigms.

- **Result** – Near order-of-magnitude reduction in per-byte and start-up cost of messages!
Outline

- Problem & Goal
- Background

- Active Messages: Novelty & Mechanism
- Example
- Methodology and Evaluation

- Strengths & Weaknesses
- Takeaways/Beyond the Paper
- Questions & Discussion
Outline

- Problem & Goal
  - Background
- Active Messages: Novelty & Mechanism
  - Example
  - Methodology and Evaluation
- Strengths & Weaknesses
  - Takeaways/Beyond the Paper
  - Questions & Discussion
Communication between processors is slow, and speeding it up sacrifices cost / performance!
Goal

Reduce communication overhead!
Outline

- Problem & Goal
- **Background**
  - Active Messages: Novelty & Mechanism
  - Example
  - Methodology and Evaluation
- Strengths & Weaknesses
- Takeaways/Beyond the Paper
- Questions & Discussion
Algorithmic Communication Model

90% of peak performance: \( T_{\text{compute}} \approx 9T_{\text{communicate}} \)
Algorithmic Communication Model

To achieve high efficiency: $T_{\text{compute}} \gg N_C T_S$

$\rightarrow T \approx T_{\text{compute}}$

dominate or the time to send/receive the message (for large messages)
Shortcomings of Existing Solutions - send/receive

- The simple approach: blocking 3-way send/receive
- **Problem:** Nodes cannot continue computation while waiting for messages!
Shortcomings of Existing Solutions - send/receive

- This can be improved by adding buffering at the message layer
- `send` appears instantaneous to the user
- The message is buffered until it can be sent
- It is then transmitted to the recipient, where it is again buffered until a matching `receive` can be executed
Shortcomings of Existing Solutions

- This allows for the overlap of communication and computation - but it’s still slow. Why?
- **Buffer Management** - Have to make sure that enough space for the whole communication phase is available! This incurs a huge start-up cost

![Graph showing start-up cost, per-byte cost, and cost of a FP operation for different systems](image-url)
Outline

- Problem & Goal
- Background

- Active Messages: Novelty & Mechanism
  - Example
  - Methodology and Evaluation

- Strengths & Weaknesses
- Takeaways/Beyond the Paper
- Questions & Discussion
Novelty

- It aims to integrate communication into ongoing computation instead of separating the two, thereby reducing overhead.

- Active Messages is a primitive, asynchronous communication mechanism
  - Not just a new parallel programming paradigm
  - Can be used to implement a wide variety of models simply and efficiently

- It is close to hardware functionality: Active Messages work like interrupts, which are already supported!
Key Approach and Ideas

Address of a user-level handler  Arguments  Active Message

Processing Node A  Network  Active Message

Interrupt

Handler  Message

Processing Node B

Computation

Memory
Mechanism (in more detail)

- Active Messages are **not buffered** (except as required for network transport)
  - The handler executes immediately upon arrival of the message (like an interrupt!)
- The network is viewed as a pipeline
  - The sender launches the message into the network and continues computation
  - The receiver gets notified or interrupted upon message arrival
- The handler is specified by a user-level address, so traditional protection models apply
- The handler **does not block** – Otherwise deadlocks and network congestion can occur
Outline

- Problem & Goal
- Background

- Active Messages: Novelty & Mechanism
- **Example**
- Methodology and Evaluation

- Strengths & Weaknesses
- Takeaways/Beyond the Paper
- Questions & Discussion
Split-C

- Split-C: provides split phase remote memory operations in C
  - PUT copies a local memory block into a remote memory at an address specified by the sender
  - GET retrieves a block of remote memory and makes a local copy
Matrix Multiplication with Split-C

Example
Matrix Multiplication with Split-C: Processor 1

Example
Matrix Multiplication with Split-C: Processor 1

Example
Matrix Multiplication with Split-C: Processor 1
Matrix Multiplication with Split-C: Processor 1

Example
Matrix Multiplication with Split-C: Master

Example
Matrix Multiplication with Split-C

% utilization

- Result: Performance predicted and measured

90% processor utilization
Outline

- Problem & Goal
- Background

- Active Messages: Novelty & Mechanism
- Example
  - Methodology and Evaluation

- Strengths & Weaknesses
- Takeaways/Beyond the Paper
- Questions & Discussion
Methodology

- nCUBE/2 & CM-5
  - Message passing architectures
  - Each node consists of a simple CPU, DRAM, and a Network Interface
  - Highly Interconnected Network

Figure 3-9: CM-5 fat tree data network topology.
Active Messages on the nCUBE/2

- Sending one word of data: 21 instructions, 11μs
- Receiving such a message: 34 instructions, 15μs
- Reduces buffer management to the minimum required for actual data transport
- Very close to the absolute minimal message layer

<table>
<thead>
<tr>
<th>Task</th>
<th>Instruction count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>send</td>
</tr>
<tr>
<td>Compose/consume message</td>
<td>6</td>
</tr>
<tr>
<td>Trap to kernel</td>
<td>2</td>
</tr>
<tr>
<td>Protection</td>
<td>3</td>
</tr>
<tr>
<td>Buffer management</td>
<td>3</td>
</tr>
<tr>
<td>Address translation</td>
<td>1</td>
</tr>
<tr>
<td>Hardware set-up</td>
<td>6</td>
</tr>
<tr>
<td>Scheduling</td>
<td>–</td>
</tr>
<tr>
<td>Crawl-out to user-level</td>
<td>–</td>
</tr>
<tr>
<td>Total</td>
<td>21</td>
</tr>
</tbody>
</table>
Active Messages on the nCUBE/2

- Sending one word of data: 21 instructions, **11µs**
- Receiving such a message: 34 instructions, **15µs**
- Near order of magnitude reduction in start-up cost
  - $T_c = 30\mu s/msg$, $T_b = 0.45\mu s/byte$
Active Messages on the CM-5

- Sending a single-packet Active Message: 1.6µs
- Blocking send/receive on top of Active Messages: $T_C = 26\mu s$, $T_b = 0.12\mu s$
Executive Summary

- **Problem** – Communication between processors is slow, and speeding it up sacrifices cost/performance of the system.

- **Goal** – Reduce communication overhead and allow overlapping of communication with computation.

- **Active Messages** – Integrate communication and computation
  - Messages consist of the address of a user-level handler at the head, and the arguments to be passed as the body.
  - The handler gets the message out of the network and into ongoing computation as fast as possible.
  - A simple mechanism close to hardware that can be used to implement existing parallel programming paradigms.

- **Result** – Near order-of-magnitude reduction in per-byte and start-up cost of messages!
Outline

- Problem & Goal
- Background

- Active Messages: Novelty & Mechanism
- Example
- Methodology and Evaluation

- **Strengths & Weaknesses**
- Takeaways/Beyond the Paper
- Questions & Discussion
Strengths

- **Simple, novel** Mechanism that solves a very important problem
- **Flexible**: Can be implemented on existing systems and can be used to implement existing models
- **Close to hardware**, which results in low overhead and makes it cheap to implement
- Greatly improves performance
- Well written paper
- Paper highlights several applications of Active Messages
Weaknesses

- **Restricted** to SPMD (Single Program Multiple Data) Model
- Handler code is **restricted**
  - Can't block and has to get the message out of the network as fast as possible
- Performance evaluation is not presented well in the paper
- Possible Hardware Support in the paper is very speculative
Takeaways

- Simple, flexible and effective
- Still very relevant today
- Wide range of possible improvements at software and hardware level
  - A lot of work has already been done
  - But there is a lot more potential here!
- Easy to read paper
Beyond the Paper

- Used in many MPI implementations at the low-level transport layer (e.g. GASNet).

- If you want more detail: Read Thorsten von Eicken's dissertation!
  - "Active Message Applications Programming Interface and Communication Subsystem Organization", David E. Culler, Alan M. Mainwaring, GASNet1996 and
Thoughts and Ideas

● Could be expanded to support other Models like MPMD & many Applications more

● This could be even faster in combination with hardware support!
  ○ “Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages”, M. Besta, T. Hoefler, HPDC 2015
Outline

- Problem & Goal
- Background

- Active Messages: Novelty & Mechanism
- Example
- Methodology and Evaluation

- Strengths & Weaknesses
- Takeaways/Beyond the Paper
- Questions & Discussion
Questions?
Discussion

- Could we somehow make the handler run *arbitrary* code?

- How could we support Active Messages in hardware?

- Is this it? What happens once we get to the minimal required message layer?
Open Discussion
Thanks for watching!

And special thanks to Giray & Geraldo!
Backup Slides
Algorithmic Communication Model

- **Assumption:**
  - The program alternates between computation and communication
  - Communication requires time linear in the size of the message, plus a start-up cost

- **Time to run a program:**
  \[ T = T_{\text{compute}} + T_{\text{communicate}} \]
  \[ T_{\text{communicate}} = N_C(T_S + L_C T_b) \]
  - \( T_S \): start-up-cost, \( T_b \): time per byte, \( L_C \): message length, \( N_C \): number of communications

- To achieve high efficiency, the programmer must tailor the algorithm to achieve a high ratio of computation to communication (i.e. to achieve 90% of peak performance:
  \( T_{\text{compute}} \leq 9T_{\text{communicate}} \))

- If communication is overlapped with communication:
  \[ T = \max(T_{\text{compute}} + N_C T_S, N_C L_C T_b) \]
  To achieve high efficiency:
  \( T_{\text{compute}} \gg N_C T_S \)
# PERFORMANCE CHART

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>iPSC[8]</td>
<td>4100</td>
<td>2.8</td>
<td>25</td>
</tr>
<tr>
<td>nCUBE/10[8]</td>
<td>400</td>
<td>2.6</td>
<td>8.3</td>
</tr>
<tr>
<td>iPSC/2[8]</td>
<td>700</td>
<td>0.36</td>
<td>3.4</td>
</tr>
<tr>
<td></td>
<td>390†</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>nCUBE/2</td>
<td>160</td>
<td>0.45</td>
<td>0.50</td>
</tr>
<tr>
<td>iPSC/860[13]</td>
<td>160</td>
<td>0.36</td>
<td>0.033[7]</td>
</tr>
<tr>
<td></td>
<td>60†</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>CM-5‡</td>
<td>86</td>
<td>0.12</td>
<td>0.33[7]</td>
</tr>
</tbody>
</table>

†: messages up to 100 bytes  
‡: blocking send/receive

![Graph showing performance chart data]
Methodology – CM-5

- **CM-5**
  - Up to a few thousand nodes interconnected in a “hypertree”
  - CPU: 33 Mhz Sparc RISC processor, local DRAM, network interface

![CM-5 Diagram](image_url)

*Figure 3-9: CM-5 fat tree data network topology.*

*Figure 3-7: CM-5 processing node organization.*
Methodology – nCUBE/2

- nCUBE/2
  - Has up to a few thousand nodes interconnected in a binary hypercube network
  - CPU: 64-bit Integer Unit, IEEE floating-point unit, DRAM interface, network interface with 28 channels
    - Runs at 20 Mhz
  - Routers to support routing across a 13 dimensional hypercube

Figure 3-1: NCUBE 6400 processor block diagram.
GET cost model

a) GET cost model

b) Overlapping communication and computation