Active Messages: a Mechanism for Integrated Communication and Computation - ISCA 1992

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

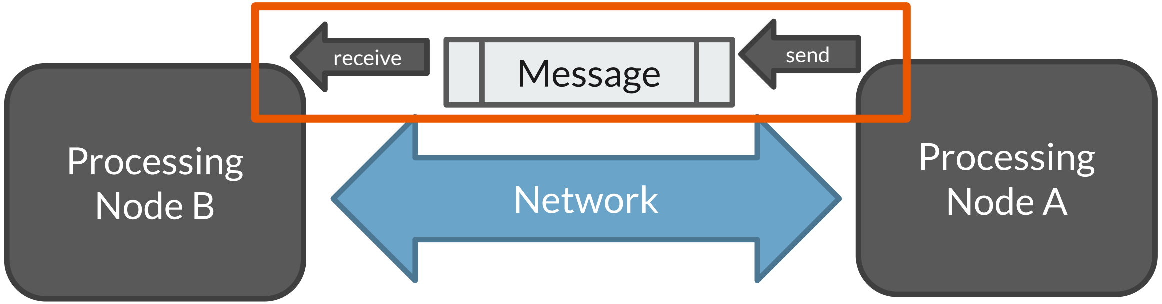
Outline

- **Problem & Goal**
- Background

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

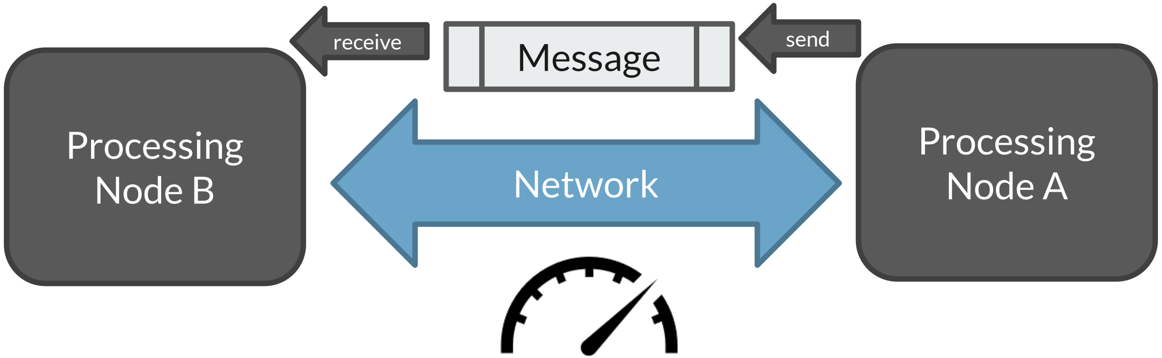
- Strengths & Weaknesses
- Takeaways/Beyond the Paper
- Questions & Discussion

Problem



Communication between processors is slow, and speeding it up sacrifices cost / performance!

Goal



Reduce communication overhead!

Outline

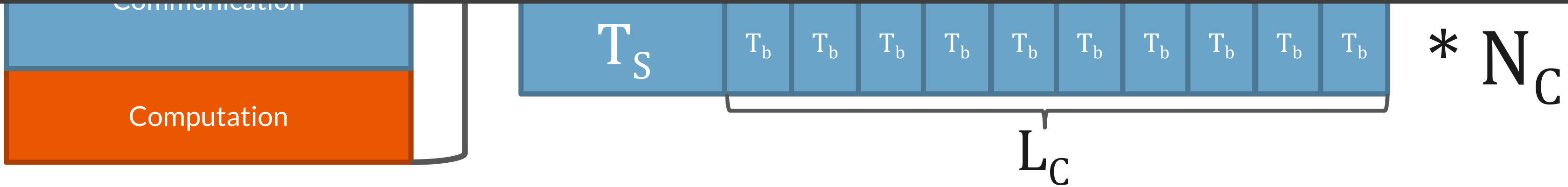
- Problem & Goal
- **Background**

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

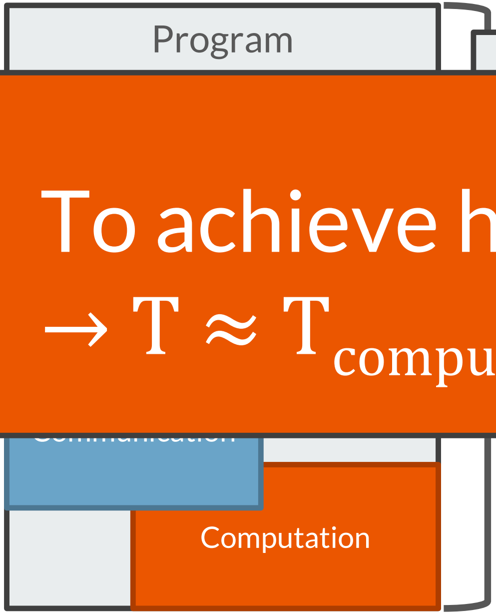
- Strengths & Weaknesses
- Takeaways/Beyond the Paper
- Questions & Discussion

Algorithmic Communication Model

90% of peak performance : $T_{compute} \approx 9T_{communicate}$



Algorithmic Communication Model



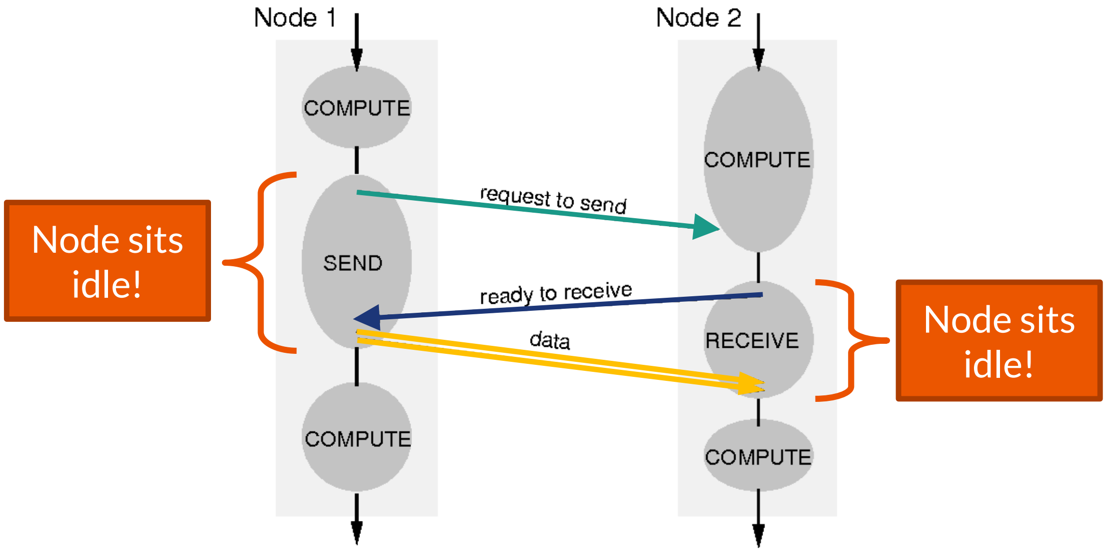
To achieve high efficiency : $T_{compute} \gg N_C T_S$

$\rightarrow T \approx T_{compute}$

dominate or the time to send/receive the message (for large messages)

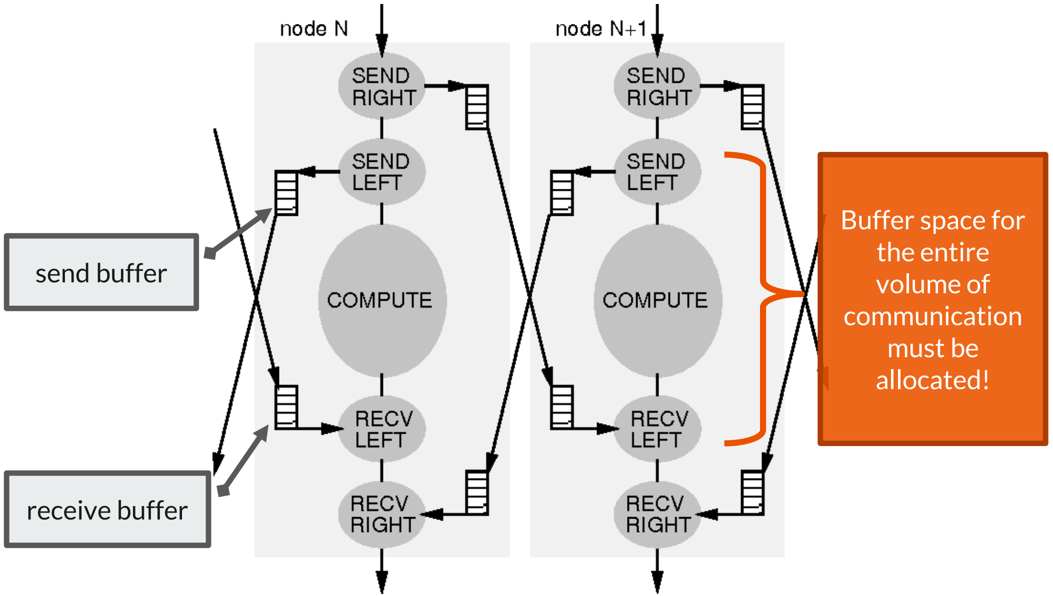
Shortcomings of Existing Solutions - send/receive

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



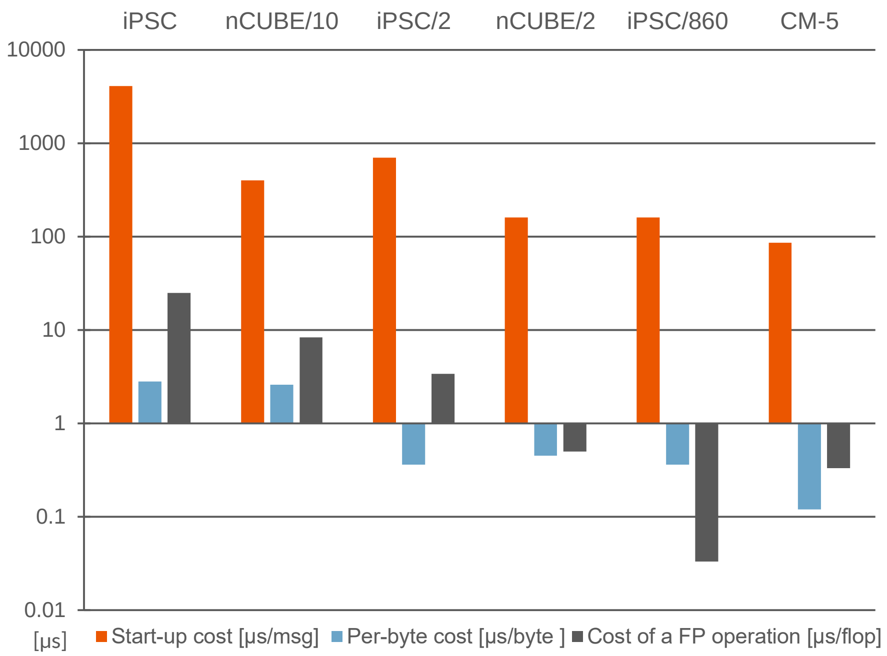
Shortcomings of Existing Solutions - send/receive

- 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**



Outline

- Problem & Goal
- Background

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

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

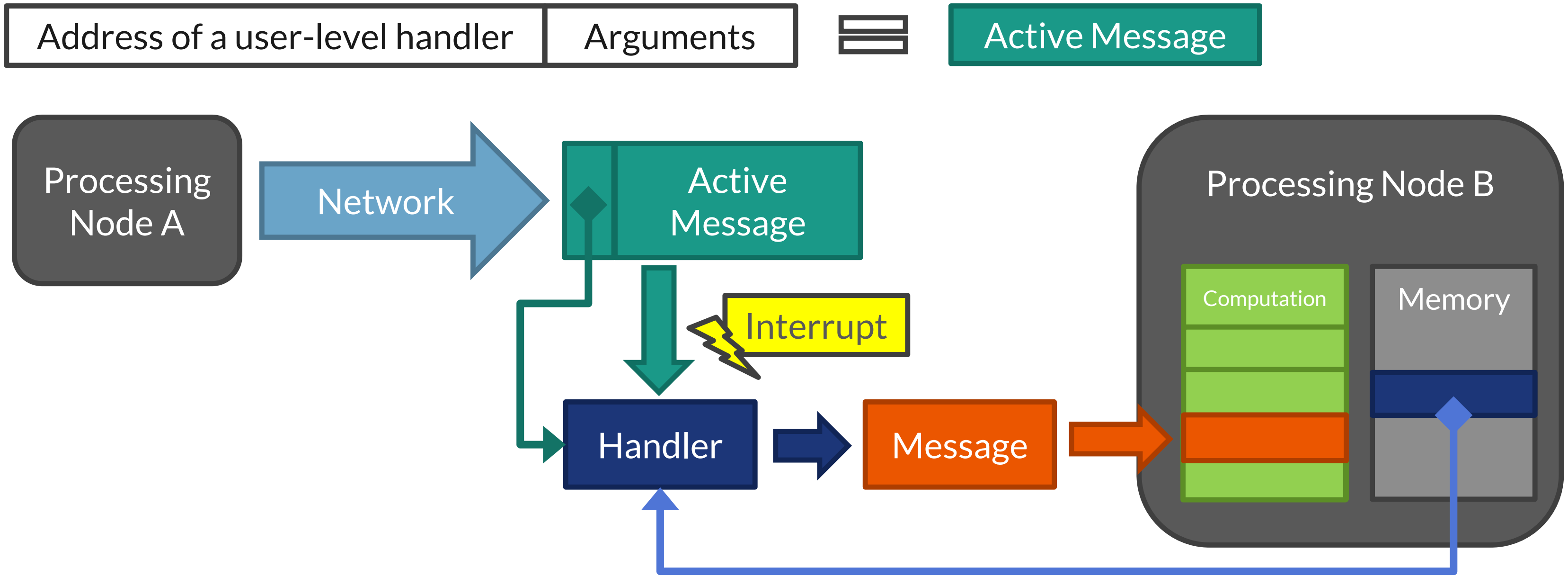
Novelty

- 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



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

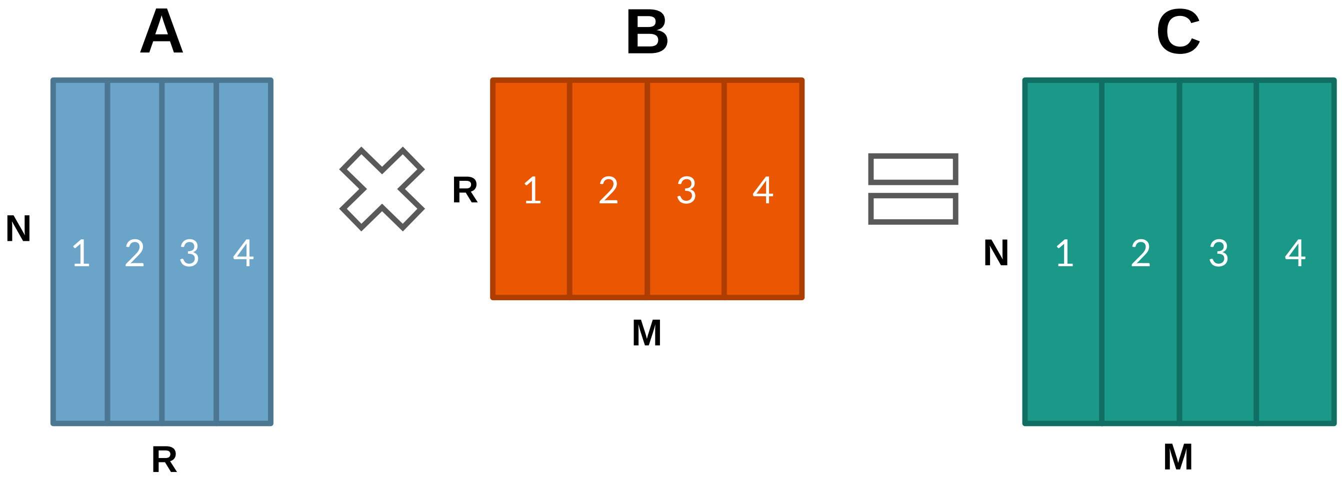
- Active Messages: Novelty & Mechanism
- **Example**
- Methodology and Evaluation

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

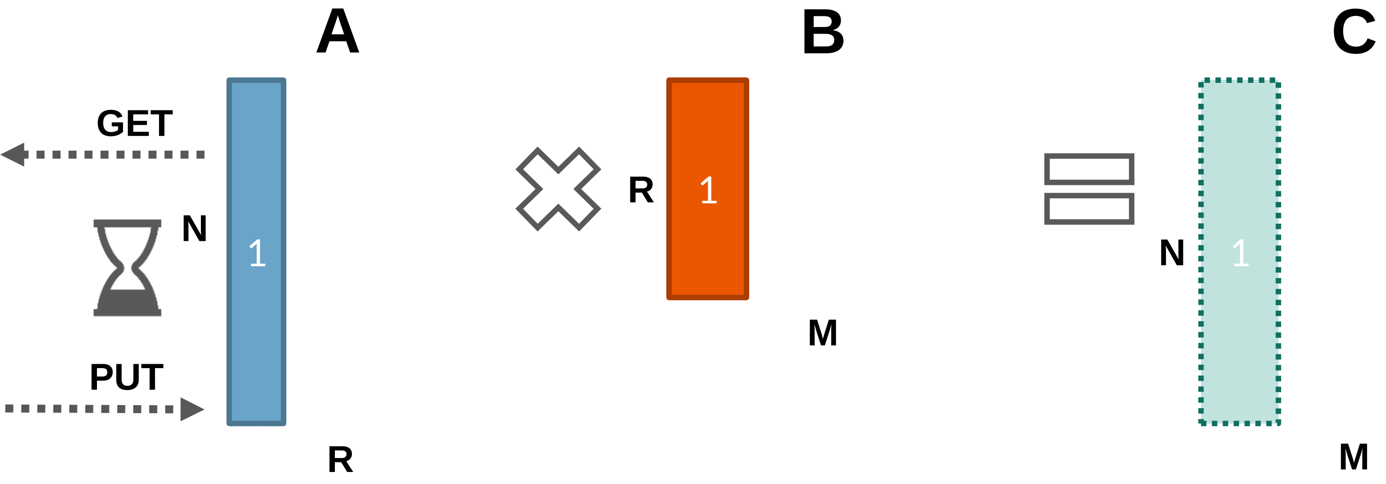
Split-C

- 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

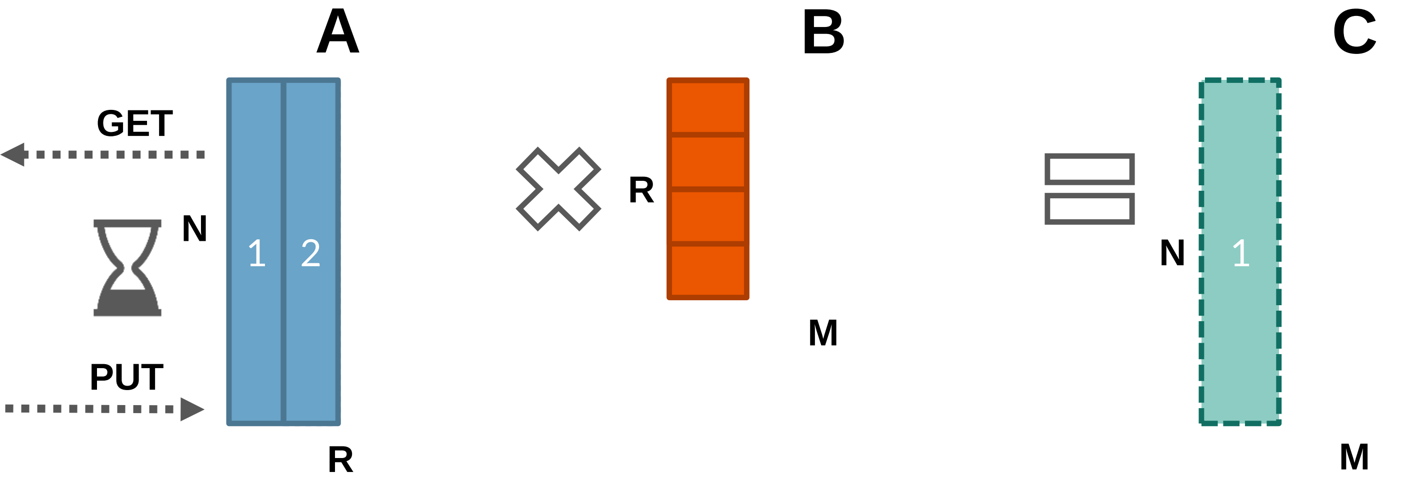
Matrix Multiplication with Split-C



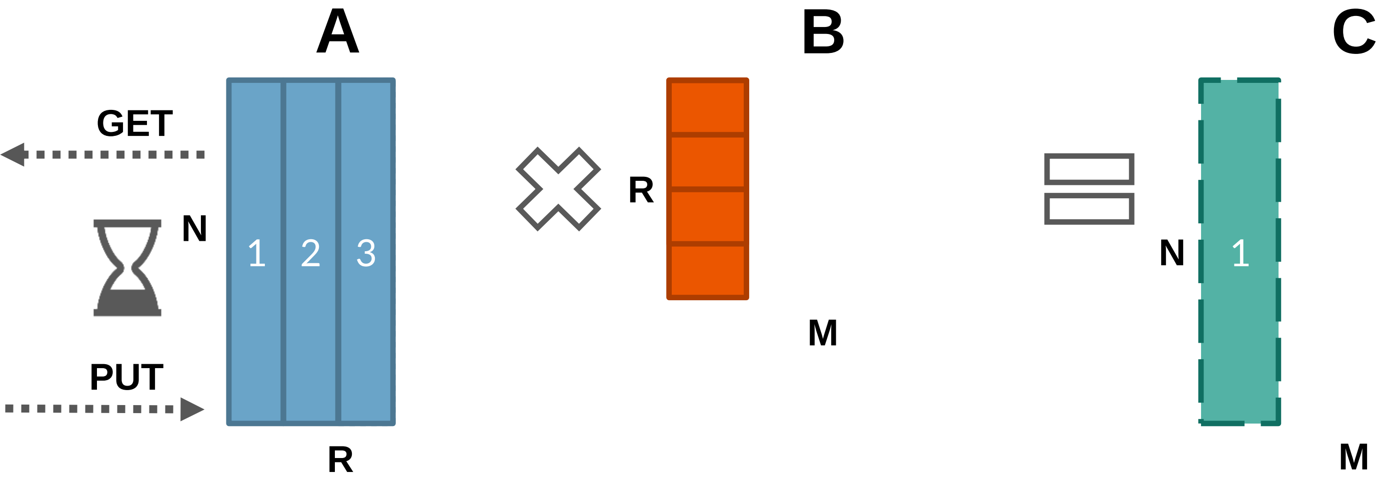
Matrix Multiplication with Split-C: Processor 1



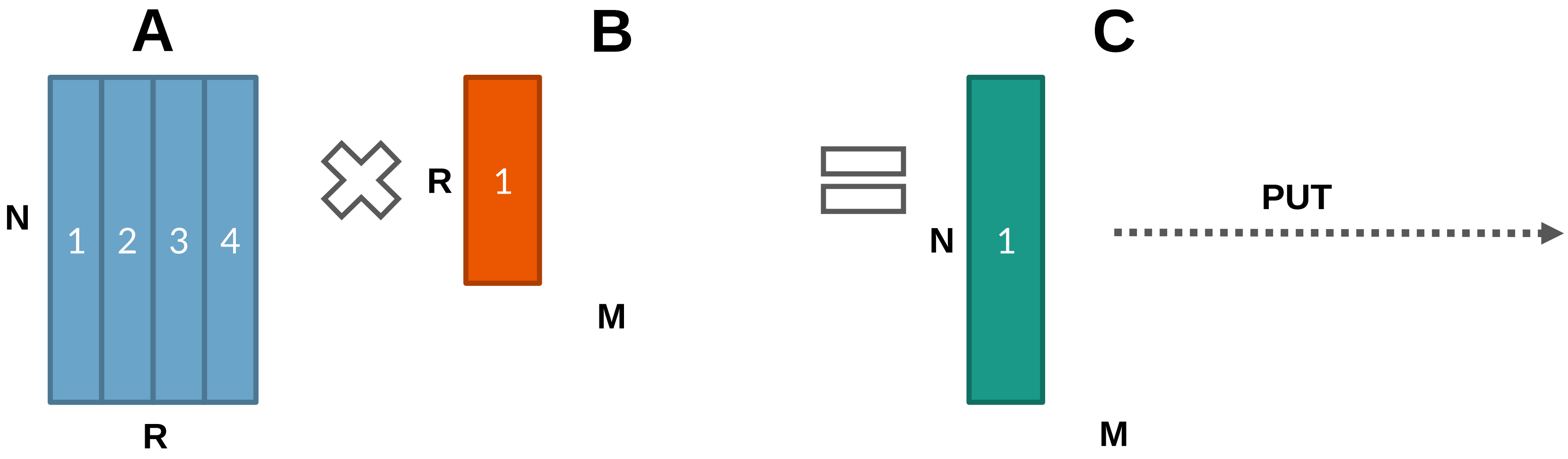
Matrix Multiplication with Split-C: Processor 1



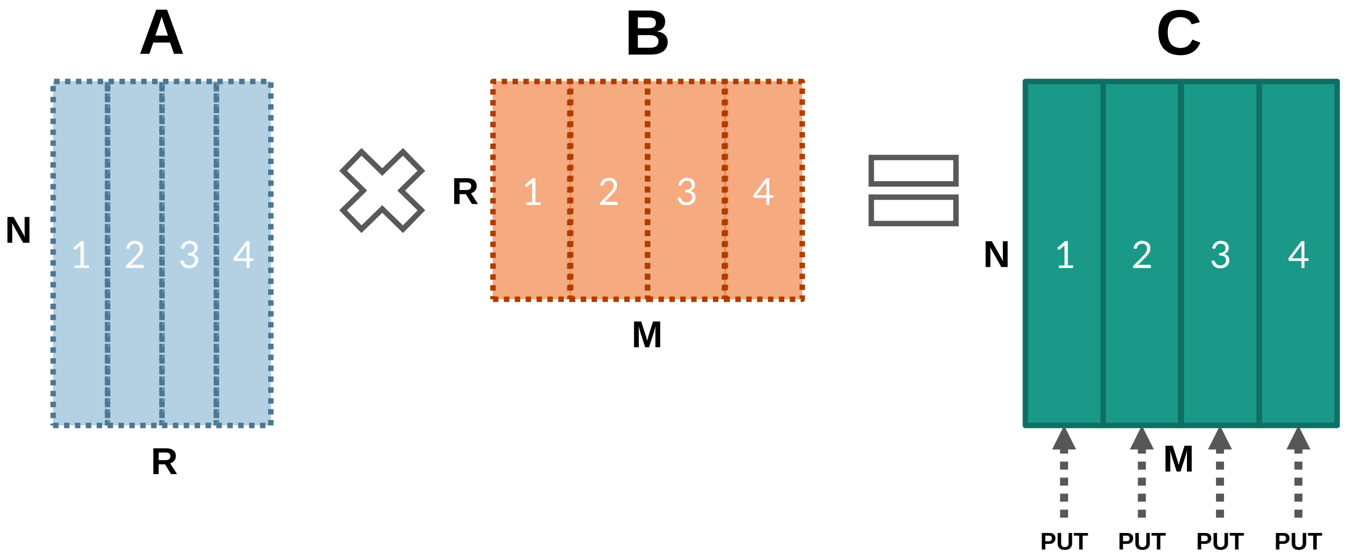
Matrix Multiplication with Split-C: Processor 1



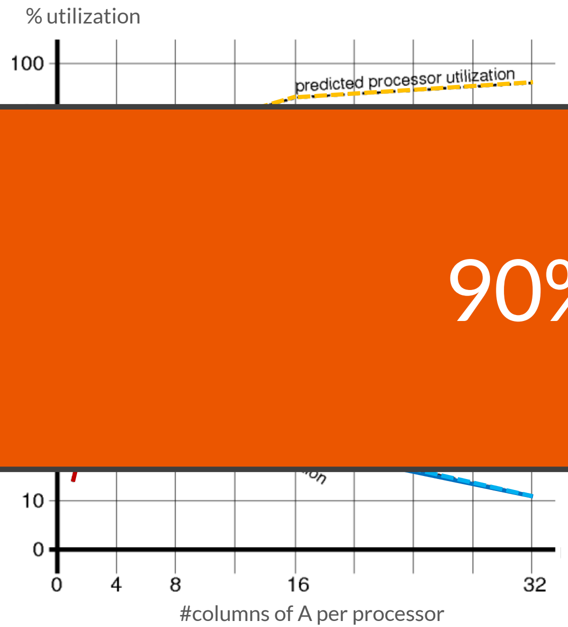
Matrix Multiplication with Split-C: Processor 1



Matrix Multiplication with Split-C : Master



Matrix Multiplication with Split-C



- 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

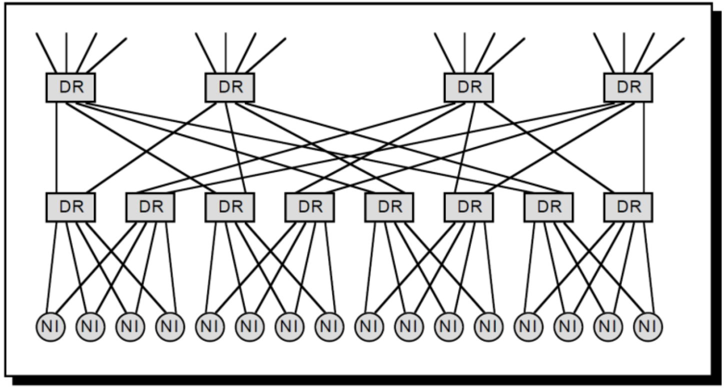


Figure 3-9: CM-5 fat tree data network topology.

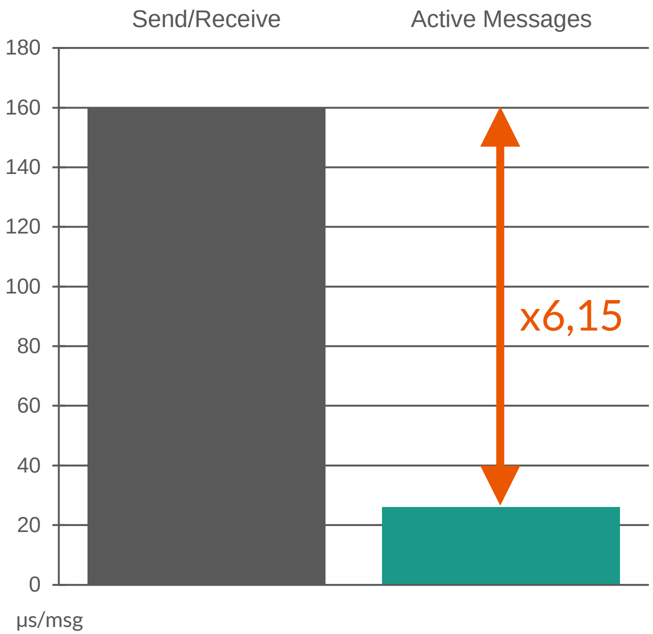
Active Messages on the nCUBE/2

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

	Instruction count	
Task	send	receive
Compose/consume message	6	9
Trap to kernel	2	–
Protection	3	–
Buffer management	3	3
Address translation	1	1
Hardware set-up	6	2
Scheduling	–	7
Crawl-out to user-level	–	12
Total	21	34

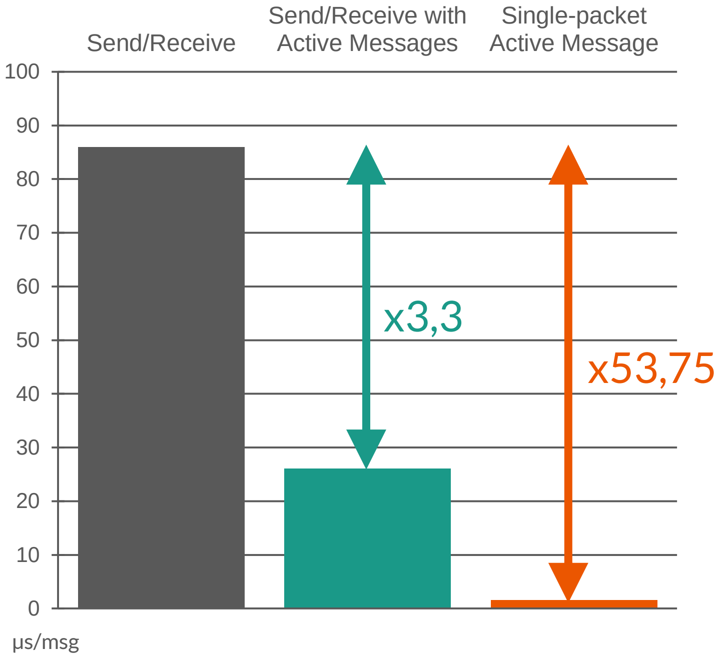
Active Messages on the nCUBE/2

- Sending one word of data: 21 instructions , **11*μs***
- Receiving such a message: 34 instructions, **15*μs***
- Near **order of magnitude reduction** in start-up cost
  - T_C = 30*μs/msg* , T_b = 0.45*μs/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

Outline

- Problem & Goal
- Background

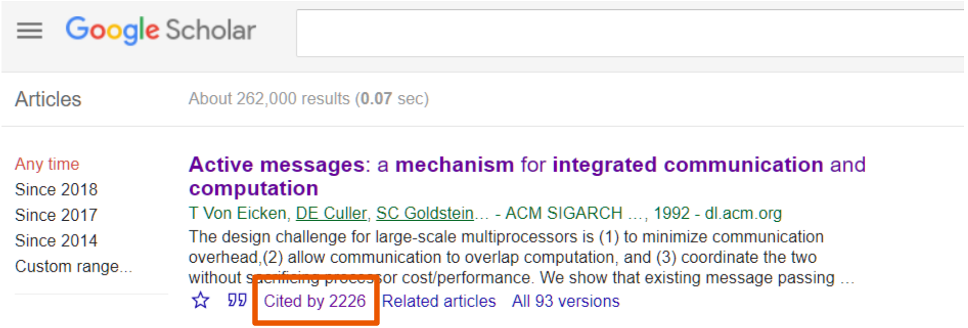
- Active Messages: Novelty & Mechanism
- Example
- Methodology and Evaluation

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

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 Messages: an Efficient Communication for Multiprocessors", Thorsten von Eicken, Cornell 1993 (https://www.cs.cornell.edu/tve/thesis/)
- "Active Message Applications Programming Interface and Communication Subsystem Organization" , David E. Culler, Alan M. Mainwaring, GASNet1996 and
- "AM++: A Generalized Active Message Framework" , T.Hoefler, J.J. Willcock, N.G. Edmonds, A. Lumsdaine, PACT **2010**

Thoughts and Ideas

- Could be expanded to support other Models like MPMD & many Applications more
  - “Active Message Applications Programming Interface and Communication Subsystem Organization” ,D. E. Culler, A. M. Mainwaring, GASNet 1996
  - “AM++: A Generalized Active Message Framework” , T.Hoefler, J.J. Willcock, N.G. Edmonds, A .Lumsdaine, PACT **2010**
- 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?
    - “Optimistic Active Messages: A Mechanism for Scheduling Communication with Computation” , D. A. Wallach, W.C. Hsieh, K.L. Johnson, M.F. Kaashoek, W.E. Weihl , EW SIGOPS 1994
- 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

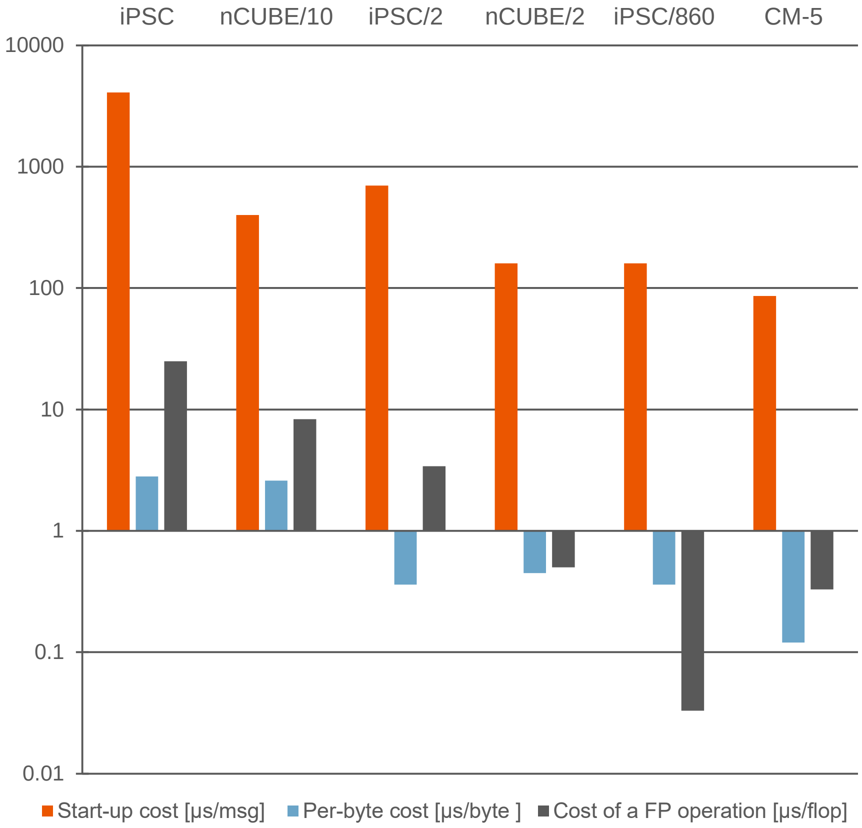
Algorithmic Communication Model

- 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: $T = T_{compute} + T_{communicate}$ and $T_{communicate} = N_C(T_S+L_C T_b)$
  - 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 : $T_{compute} \leq 9T_{communicate}$)
- If communication is overlapped with communication: $T=\max(T_{compute}+N_C T_S, N_C L_C T_b)$
  To achieve high efficiency : $T_{compute} \gg N_C T_S$

PERFORMANCE CHART

Machine	T_s [μs/mesg]	T_b [μs/byte]	T_{fp} [μs/flop]
iPSC[8]	4100	2.8	25
nCUBE/10[8]	400	2.6	8.3
iPSC/2[8]	700	0.36	3.4
	390†	0.2	
nCUBE/2	160	0.45	0.50
iPSC/860[13]	160	0.36	0.033[7]
	60†	0.5	
CM-5‡	86	0.12	0.33[7]

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



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

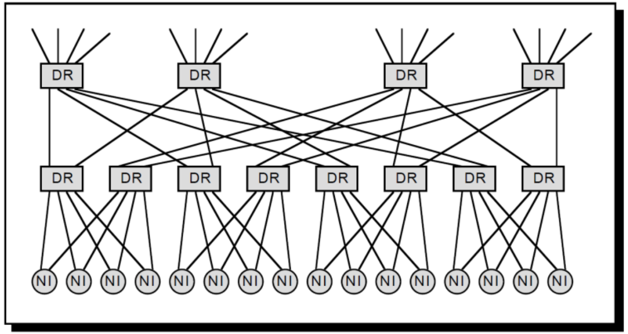


Figure 3-9: CM-5 fat tree data network topology.

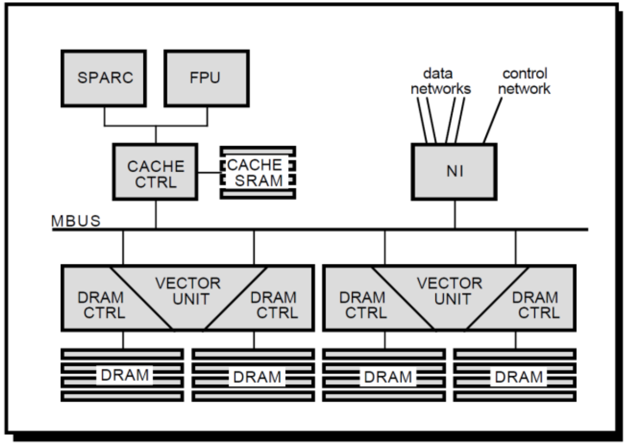


Figure 3-7: CM-5 processing node organization.

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

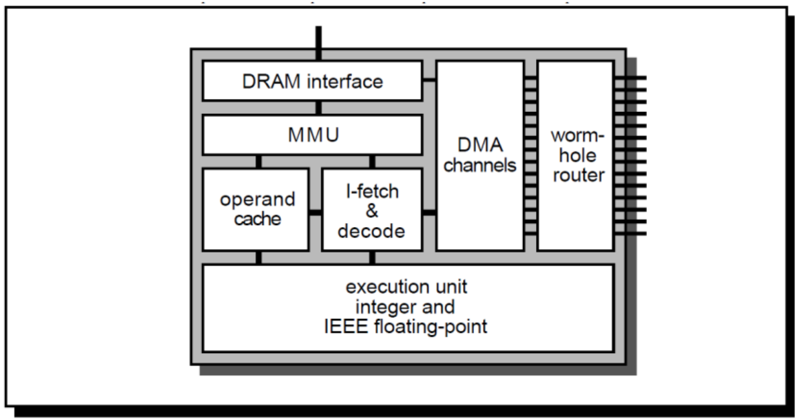


Figure 3-1: NCUBE 6400 processor block diagram.

GET cost model