Active Messages: a Mechanism for Integrated Communication and Computation

von Eicken et. al.

Brian Kazian

CS258 Spring 2008

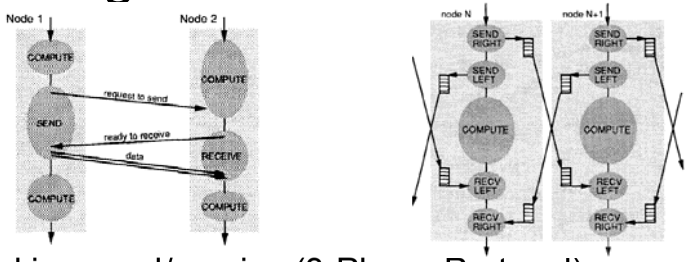
Introduction

- Gap between processor and network utilization
  - Need to maximize overlap to ensure efficiency of program
- High message overhead
  - Requires batching of messages to compensate
- H/W development neglects interaction between processor and network

Active Messages

- Mechanism for sending messages
  - Message header specifies instruction address for integration into computation
  - Handler retrieves message, cannot block
  - No buffering available
- Idea of making a simple interface to match hardware
- Allow for overlap of computation and communication

Existing Send/Receive Models



- Blocking send/receive (3-Phase Protocol)
  - Simple, yet inefficient computationally
  - No buffering needed
- Asynchronous send/receive
  - Communication encapsulates computation
  - Buffer space allocated throughout computation

Active Message Protocol

- Protocol
  - Sender sends a message to a receiver
    - Asynchronous send while still computing
  - Receiver pulls message, integrates into computation through handler
    - Handler executes without blocking
    - Handler provides data to ongoing computation
      - Does not perform any computation itself
    - Handler can only reply to sender, if necessary

Why Active Messages

- Asynchronous communication
  - Non-blocking send/receive for overlap
- No buffering
  - Only buffering needed within network is needed
    - Software handles other necessary buffers
- Improved Performance
  - Close association with network protocol
- Handlers are kept simple
  - Serve as an interface between network and computation
- Concern becomes overhead, not latency

Message Passing Machines

- Computation is via threads
- Discrepancy between H/W and programming models
  - Higher level 3-phase send/recv used
    - Active Messages provide better low-level interaction
- Little overlap of communication/computation
  - Active Messages could allow for this
    - No need for complicated scheduling
    - Large messages may still need to be buffered
- AM provides performance increase solely with software

Message Passing Architectures – nCUBE/2 and CM-5

- Overhead reduction
  - nCUBE/2:
    - 160 us blocking -> 30 us Active Message
  - CM - 5
    - 86 us blocking -> 23 us Active Message
- Deadlock
  - nCUBE/2 uses multiple user buffers to prevent deadlock
  - CM-5 has dual identical networks
    - Split for requests and replies

Message Driven Machines

- Computation is within message handlers
- Network is integrated into the processor
- Developed for fine-grain parallelism
  - Utilizes small messages with low overhead
- May buffer messages upon receipt
  - Buffers can grow to any size depending on amount of excess parallelism
- State of computation is very temporal
  - Small amount of registers, little locality

Hardware Support

- Network Modifications:
  - Data reuse
    - Store pieces of data in network interface for reuse
  - Protection
    - Enforce message restrictions at network level
  - Message Accelerators
    - Frequent messages launched quickly

Processor Support

- Interrupts only way to handle asynchronous events
  - Flushes pipeline, very expensive!
- Can insert polling for messages by compiler
- Use multithreading to switch between PC's
- Use two separate processors
  - Handler and main computation separated

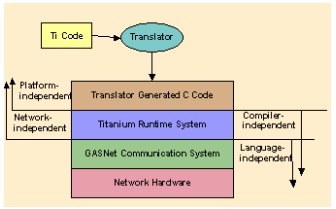
Split-C

- Extension of C for SPMD Programs
  - Global address space is partitioned into local and remote
  - Maps shared memory benefits to distributed memory
    - Dereference of remote pointers
    - Keep events associated with message passing models
  - Split-phase access
    - Enables dereferencing without interruption of processor
- Active Messages serve as interface for Split-C
  - PUT/GET instructions utilized by compiler through prefetching

Active Messaging in its Current Form

- Active Message 2 API
  - Naming updated to allow for models other than SPMD
    - Paper implementation requires uniform code image
  - Support for multi-threaded applications
  - Multiple communication endpoints
    - Controlling communication allows for handling messages that are returned
- Additional robust forms of AM
  - AMMPI, LAPI

Titanium Implementation



- Similar to Split-C, Java-based
  - Utilizes GASNet for network communication
    - GASNet higher level abstraction of core API with AM
  - Global address space allows for portability
  - Skips JVM by compiling translating to C

Image from http://titanium.cs.berkeley.edu/

Conclusion

- Active Messages provide a low-level interface for asynchronous messaging
  - Match hardware well on both message passing/driven machines
- Handlers are simple, keeping complexity low
- Allows for overlap between computation and communication
- Model is the basis for many different communication stacks

Questions?