Design and Implementation of Generic Active Messages on the Intel Paragon

Lok T. Liu

November 30, 1994

Summary

General Goal:

- Evaluate the software and hardware architecture trade-off when full-blown message processor is present
- Tons of issues:
  - what functions should MP perform?
  - communication layer abstraction
  - interaction with VM and scheduling, protection, flow control, message scheduling, error recovery, …

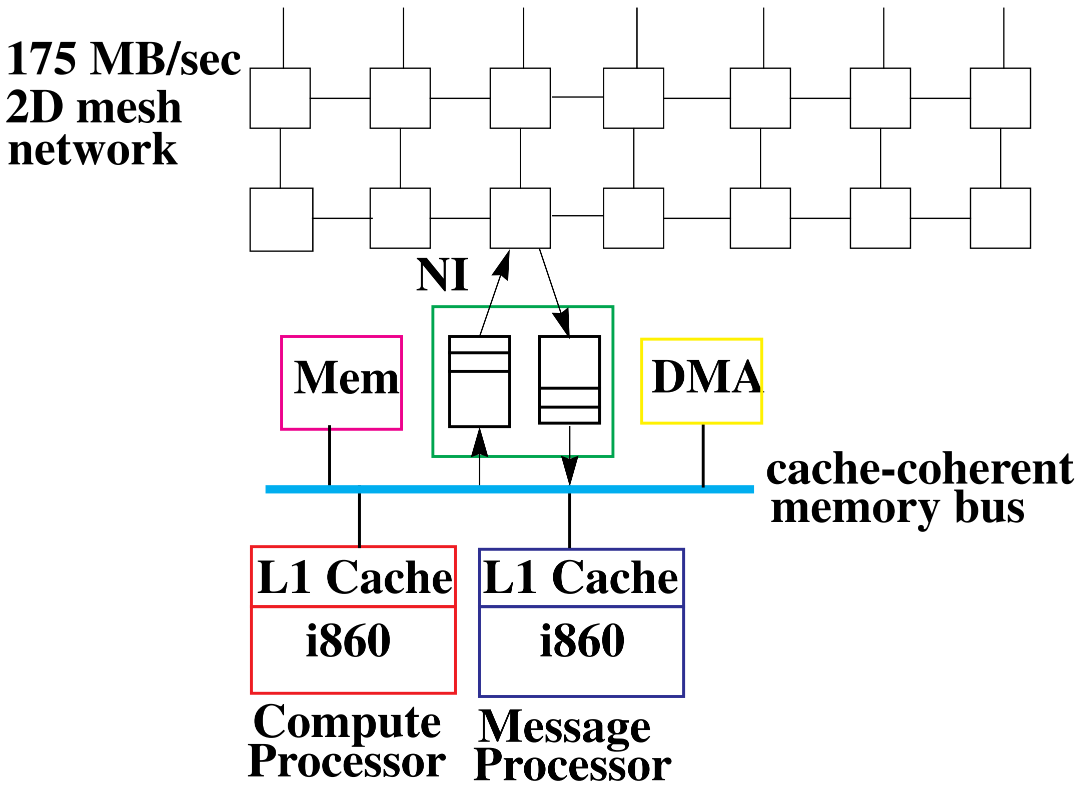
Today's Focus:

- how to build Generic Active Messages with MP?
- what are the costs?

Summary:

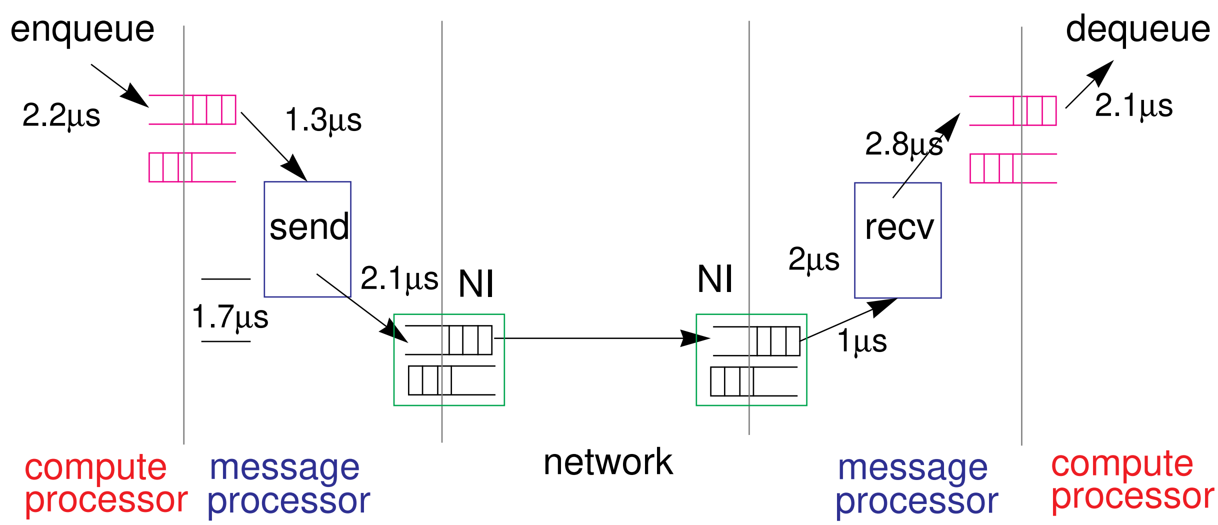
- abstract MP interface as queues and build GAM with enqueue and dequeue
- for small messages, processor-to-processor handoff doubles the round-trip time
- the DMA engine is too simple
  - software needs to worry about alignment, address translation, and fragmentation
  - add to overhead and complexity of software

The Intel Paragon



- Each node is a shared-memory multiprocessor
  - two 50MHz Intel i860 microprocessors and a DMA engine are connected by cache-coherent memory bus
  - one i860 is designated as message processor (MP) to run message handling part of OSF/1 AD kernel
  - memory-mapped NI with 2KB transmit and send FIFO's
- Nodes are connected by a reliabe,175 MB/sec 2D mesh

The Paragon Active Message Architecture



- Abstract the message processor interface as queues
- Build Generic Active Messages on top of enqueue (send) and dequeue (poll)
- Network Contract
  - users specify size of receive queue and manage it
  - message layer promises not to drop anything as long as users do not overflow receive queue
  - message processor is deadlock free and drains network even when user process is not running
- No restrictions on handler invoked on compute processor
  - window flow control with request/reply is a simple example of managing receive queue space
- For short messages, it is a lose in terms of performance
  - 25 µsec round-trip time (12 µsec minimum)
  - 5.1 µsec send overhead and 5.8 µsec receive overhead for pipelined communication (3 µsec minimum)

Bulk Transfer

- Combine DMA transfer with queues
- Message processor(MP) fragments data into packets for DMA transfer and enqueues handler at completion
  - am_store_async() supports asynchronous writes with separate completion handlers at sender and receiver
  - am_get() supports prefetch with completion handler at requester. Remote compute processor is not involved in best case
- Support overlapping computation with communication
- MP needs to handle outstanding DMA send and receive concurrently to avoid fetch deadlock
- Specialized handler, e.g. MP increments Split-C counters
- Performance
  - am_store_async(): 4 µsec send overhead
  - am_store(): 20 µsec send overhead, 140MB/sec peak send bandwidth
  - nearest neighbor exchange: 100 MB/sec send and receive bandwidth
- Peak bandwidth is sender limited by overhead in:
  - sending header
  - virtual address to physical address translation
  - handling alignment, minimum transfer size, and page boundary requirements

Bulk Transfer (cont.)