The Overall SHRIMP Project

The SHRIMP (Scalable High-performance Really Inexpensive Multi-Processor) project goal is to investigate how to construct high-performance servers with a network of commodity PCs, and commodity operating systems that deliver performance competitive with or better than the commercial multicomputer servers.

The research project consists of several components:
- user-level, protected communication
- efficient message-passing
- shared virtual memory
- distributed file system
- performance measurement
- scalable 3-D graphics
- applications.

Two types do systems
- SHRIMP II – Pentium Pro connected with Paragon Switch
- Collection of Intel nodes connected with Myricom

Paper Goals

- The paper is a summary of many aspects of the research project
  - Paper 6/98
- Attempts to cover many areas
  - Analysis of 4 different communication methods
  - Justify building a hardware prototype
  - Explain implementations and models
  - Use applications to assess performance
  - Evaluate Deliberate vs Automatic Update
  - Analyze hardware tradeoffs
- Hence many references

Related Work (from last section)

- Paper Focus – design evaluations
  - DASH, Cedar, Alewife, J machine
  - Claim SHRIMP leverage commodity components to a greater degree
- Network Fabric
  - Already existing commodity networks – Myrinet and Servernet that are similar what they used
  - Therefore using a non-commodity network (from the Paragon) is ok?
- Network Interfaces
  - Several relationships to past research work
- Automatic Update
  - Microchannel,
  - Page based – Memnet, Merlin, SESAME, Plus and Galactica Net
- Hardware-software interface streamlining
  - CM-5
Questions the Paper Poses

- Is it necessary or valuable to build custom hardware or is possible to get comparable results by using off the shelf hardware and software?
- Is the automatic update mechanism in SHRIMP useful and better than simple block transfer mechanism?
- Was user level initiation of outgoing DMA transfers necessary, or is a simple system call based approach as good?
- How necessary is it to avoid receiver-side interrupts?

SHRIMP System Overview

- Sixteen PC nodes
  - 60 MHz Pentium
  - Two level cache that snoops the memory bus
    - Write-back, write-through and no-caching modes
    - Remains consistent – including transactions from the network interface
- Connected with Intel routing backplane
  - Same connection used in the Paragon multicomputer
  - Two dimensional mesh with wormhole routing
  - 200 MB/s transfer rate
- One Network Interface per node
  - Consists of two boards
    - Connects to the Xpress Memory Bus
      - Snoops all main memory writes, and passes
    - Connection to the EISA I/O bus

SHRIMP Test System
Virtual Memory-Mapped Communication - VMMC

- Developed to have extremely low latency and high bandwidth
- Achieved by allowing applications to transfer data directly between two virtual memory address spaces over the network
- Support applications and common communication models such as message passing, shared memory, RPC, and client-server
- Consists of several calls to support user-level buffer management, various data transfer strategies, and transfer of control

Buffers and Import/Export
- Data is directly transferred to receive buffers
  - Variable sizes regions of virtual memory
- Applications export buffers with a set of permissions
  - Any other process with the proper permissions can import the buffer to a proxy receive buffer
- Communication is protected
  - A trusted third party such as the operating system kernel or a trusted process implements import and export operations.
  - The hardware MMU on an importing node makes sure that transferred data cannot overwrite memory outside a receive buffer.
- Deliberate Update
  - To transfer data a process specifies a virtual address in its memory, a virtual address in the proxy receive buffer and a transfer size.
  - Communication subsystem contiguous blocks of data
  - Designed for flexible import-export mappings and for reducing network bandwidth
  - Implemented with two load/store sequences to special I/O mapped addresses.

Automatic Update
- VM is bound to a imported receive buffer
- All writes to the bound memory are automatically transferred to the receive buffer
- Result is all writes performed to the local memory are automatically performed to the remote memory as well, eliminating the need for an explicit send operation.
- Optimized for low latency
- Implemented by network interface maintaining a mapping between physical page numbers and Outgoing Page Table

Notifications
- A process can enable notification for an exported receive buffer
  - Blocking and non-blocking
  - Control transfers to a user level handler

Communication Models

Guarantees the in-order, reliable delivery of all data transfers, provided that the ordinary, blocking version of the deliberate-update send operation is used.
- Guarantees are a bit more complicated in order, reliable delivery for the non-blocking deliberate-update send operation
- Does not include any buffer management since data is transferred directly between user-level address spaces.
  - Gives applications the freedom to utilize as little buffering and copying as needed.
- Directly supports zero-copy protocols when both the send and receive buffers are known at the time of a transfer initiation.
- The VMMC model assumes that receive buffer addresses are specified by the sender, and received data is transferred directly to memory
  - Hence, there is no explicit receive operation.
  - CPU involvement for receiving data can be as little as checking a flag, although a hardware notification mechanism is also supported.
Higher Level Communication

- SHRIMP implementations
  - Native VMMC
  - NX message passing library
  - BSP message passing library
  - Unix stream sockets
  - Sun RPC compatible
  - Specialized RPC library
  - Shared Virtual Memory (SVM)
- The paper has performance studies for
  - Native VMMC
  - NX message passing library
  - Unix stream sockets
  - Shared Virtual Memory (SVM)

Applications

- Shared Virtual Memory
  - Barnes – Barnes-Hut Hierarchical N-body particle simulation
    - Octree of space cells - Rebuild on each iteration
  - Ocean – CFD using partial differential equations
    - Statically load balances work by splitting grid – nearest neighbor communication
  - Radix – Sorts integers
    - Divides keys across processors – induces false sharing
  - VMMC
    - Radix – versions for r deliberate and automatic updates
  - NX
    - Ocean
    - Barnes – Octree limits scaling
- Sockets
  - DFS – a file system
  - Render – a volume render

Questions

- It does it make sense to build hardware
  - Needed in order to study performance with latencies lower than what was commercially available at the time
    - SHRIMP 6 \(\mu\)sec vs Myrinet at 10 \(\mu\)sec
  - Allows comparisons of automatic and deliberate update
    - No other interfaces implement virtual memory mapped automatic update
- Is Automatic Update a good idea
  - Two advantages
    - Very low latency
      - End to end is 3.7 \(\mu\)sec for single word transfer
    - Replaces need for gather scatter
    - Write sparse data in memory and have it automatically exported
    - Three implementations to evaluate automatic updates on SVM
  - Two disadvantages
    - Send and receive buffers must have page boundary alignment
    - Hardware does not guarantee order between deliberate and automatic transfers
      - Not helpful to message passing to socket based applications
      - Large transfers are dominated by DMA performance

Shared Virtual Memory

- In a preliminary evaluation on the Intel Paragon multiprocessor, software home-based protocol (called HLRC) was also found to out perform earlier distributed all-software protocols.
  - HLRC-AU uses HLRC but instead of buffering differences and sending them as messages, it transparently propagate them
    - Idea is to compute diffs as in previous all-software protocols, but to propagate the diffs to the home at a release point and apply them eagerly, keeping the home up to date according to lazy release consistency.
  - Automatic Updates are used to implement Automatic Update Release Consistency (AURC) protocol.
    - Every shared page has a home node, and writes observed to a page are automatically propagated to the home at a fine granularity in hardware.
    - Shared pages are mapped write-through in the caches so that writes appear on the memory bus.
    - When a node incurs a page fault, the page fault handler retrieves the page from the home where it is guaranteed to be up to date.
    - Data are kept consistent according to a page based software consistency protocol such as lazy release consistency.
Questions and Answers

- User Level DMA decreases system calls
  - When using kernel level DMA, application run times increases from 2% to 52.5%. Therefore User Level DMA has a significant benefit

- Interrupt Performance
  - SVM used notification, sockets and VMMC did not.
  - Avoiding receive side interrupts
    - Application execution times increases from .3 to 25.1% when interrupts were used for every message

- Automatic Update Combining
  - Not as effective

- Outgoing FIFO Capacity

- Deliberate Update Queuing
  - Small impact because nodes did not share cycles between CPU and I/O

Conclusions

- VMMC avoids receive side interrupts and calls – increases performance significantly
- Building hardware is hard – but lessons could not be learned without building hardware
- Automatic update very useful for VMMC and SVM. Message passing is better with deliberate update.
- User-level DMA significantly reduces overhead on sending a message – and that provides much better performance.
  - But – there are good ideas on how to improve kernel based implementations
- Automatic Update can significantly reduce network traffic by combining consecutive updates into a single packet.
  - But does not help SVM or Radix using VMMC
- No improvement from queuing multiple deliberate updates
- Small FIFO is adequate