Administrivia

Reader still available

Reading for Monday:

- Blumofe: deadlock avoidance (online)
- O'Malley & Peterson, Protocol layering (reader)

Minor community difference:

- LAN/WAN: "insure good performance"
- MPP: "ensure good performance" (yeah, we win)

Flow Control 3: Outline

Review

One-on-one transfers

Handling unknown traffic patterns

Application-Level Framing

Segments

User-level access to remote address space

Segments are:

- Named by *(node, offset)*
- Managed by the kernel (set up & tear down)
- Read/Written in user level

Excellent block transfer performance:

- No kernel crossings
- Minimal copies
- Supports DMA

Limited flexibility $\Rightarrow$ still need active messages

An Interface

Regular active messages for RPC & other protocols

Kernel-managed segments for user-level access

DMA-based block transfers with optional handlers

- Avoid copying and processor interaction
- Handlers support notification (such as counting)

Simple strided DMA (coprocessor?)

- Reduces overhead for multidimensional arrays
- Optional handler after all blocks (vs. once per block)

What is above the interface?

Many remaining problems:

- Flow/congestion control
- Out-of-order packets
- Deadlock avoidance
- Fault detection (checksums)
- Retransmission
- Encryption
- Format conversion (heterogeneous nodes)

Flow Control

Single transfer $\Rightarrow$ bandwidth matching

What about concurrent transfers?

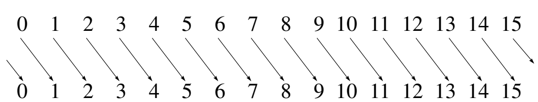
Two-on-one $\Rightarrow$ half as much bandwidth per transfer

- Maximum bandwidth requires all receivers
- An idle receiver increases execution time

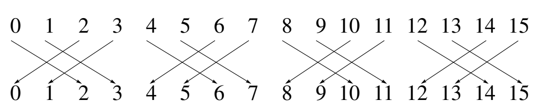
Goal: structure application for even communication load

Easiest solution: permutations

Permutations



Cyclic Shift by One



Hypercube Exchange – Dimension 1 (XOR 2)

Random Traffic?

Random traffic ensures even load,
but only over time...

n balls randomly placed in n bins:

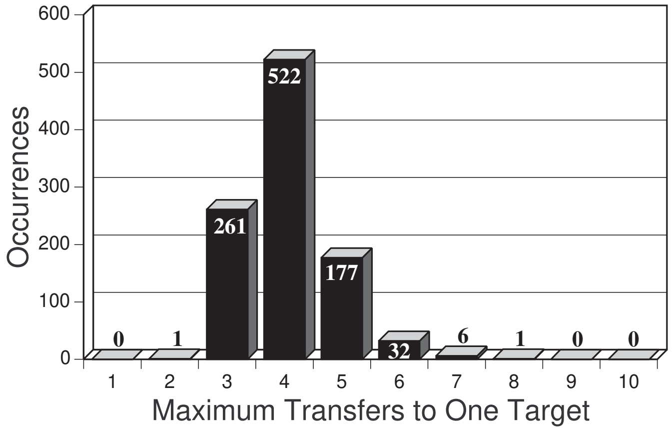
- = n senders picking from among n receivers
- Maximum number of balls in one bin?

$$\Theta\left(\frac{\log n}{\log\log n}\right)$$

Hard to evaluate, so simulate for $n = 64$...

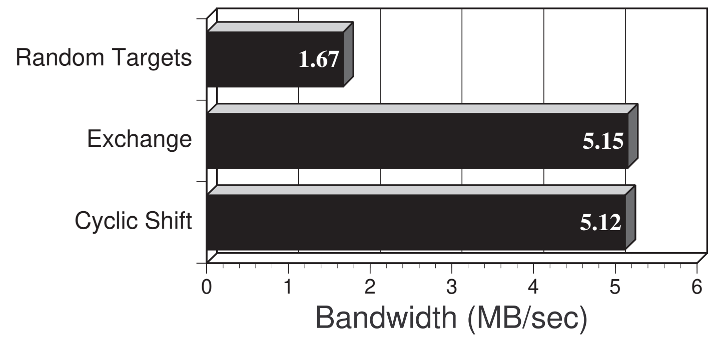
Transfers to one Receiver

64 nodes, random targets, 1000 trials:

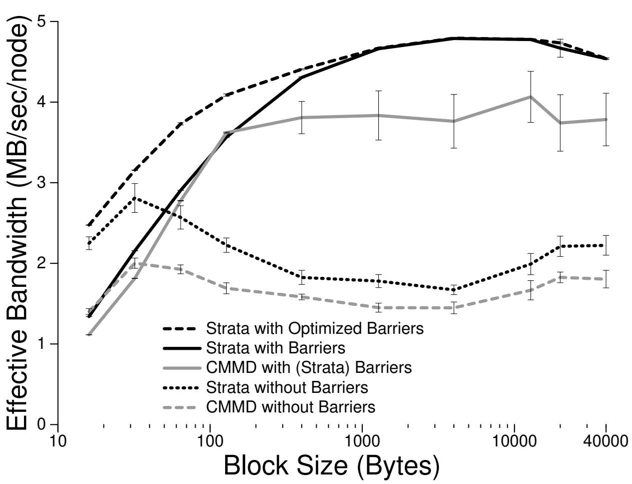


Effect on Bandwidth

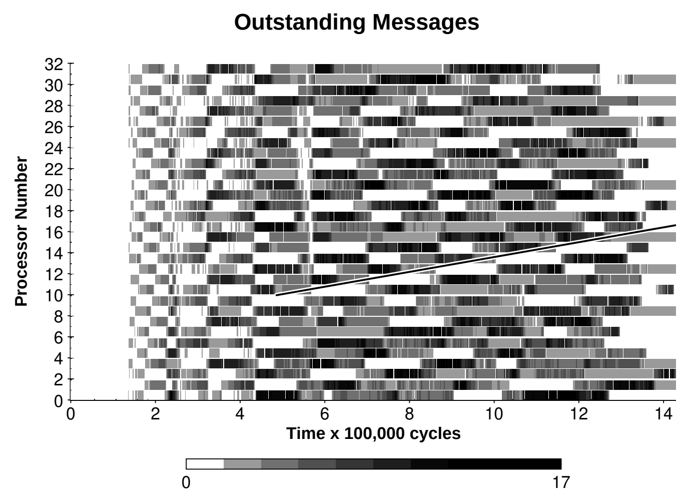
4K block transfers on a 64-node CM-5



Barriers



Why Barriers Help



Optimized Barriers

Problem: barrier overhead too high for small transfers

- Solution: perform several permutations per barrier
- Use k transfers for block size B, where $k = \left\lceil \frac{512}{B} \right\rceil$
- $\Rightarrow$ at least 512 bytes between barriers

Anomaly: upward curve for large transfers

- Misleading: large transfers $\Rightarrow$ fewer rounds
- System never gets out of sync:

Rounds	MB/s	95% CI
32	2.22	±0.122
100	1.67	±0.040

Unknown Distributions

What if we can't exploit permutations?

- Random traffic doesn't work well...

Key idea: small random transfer *do* work well

Solution: interleave packets from several transfers

Requires a new interface:

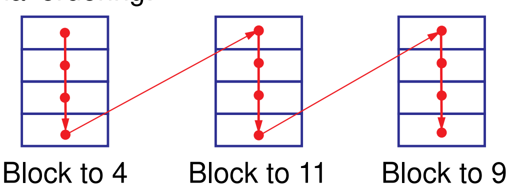
- Hand multiple blocks to the block-transfer engine

One interface: split-phase block transfer

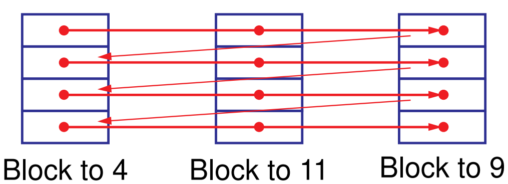
- Start several transfers to different nodes
- Block at the end for all of them
- Engine interleaves packets from all outstanding transfers

Packet Interleaving

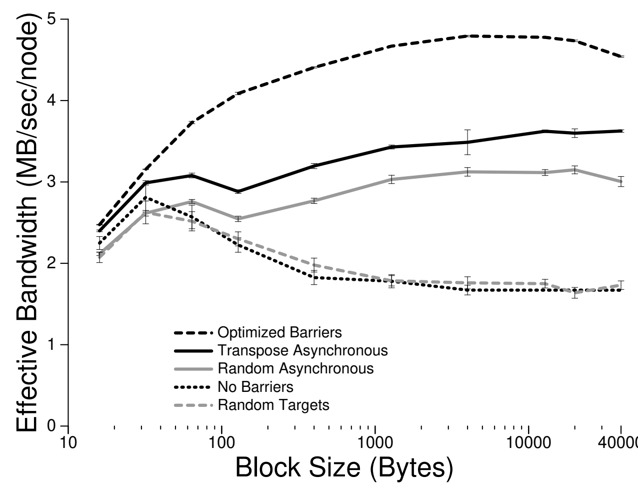
Traditional ordering:



Interleaved order:



Performance Impact



Interleaving and HOL Blocking

Requiring one block at a time is broken
- ◆ Limits performance
- ◆ Acts as head-of-line blocking at the sender

Consider a 3D torus:
- ◆ 6 independent output directions
- ◆ If we only have one transfer & its output is blocked, then we lose...
- ◆ Multiple transfers allows some to make progress

Implications:
- ◆ No impact on CM-5, since there's only one output
- ◆ Shouldn't stall on any one transfer: if the injection fails, go on to the next transfer

Application-Level Flow Control

If possible, use a sequence of permutations
- ◆ Use barriers to keep permutations separate

For unknown patterns, use packet interleaving:
- ◆ Makes even use of the network resources
- ◆ Avoids head-of-line blocking

Bandwidth matching pays off for bulk data movement:
- ◆ Minimizes congestion for a given throughput
- ◆ Claim: can handle bisection limitations cleanly

Keep phases separate with barriers:
- ◆ Can improve performance and rarely hurts
- ◆ Improves predictability and robustness

Data Manipulations

OK, we can transmit data real fast – what about:

Checksums: ignore or place in hardware

Encryption: ignore or place in hardware
- ◆ Key management?

Retransmission: try to avoid...
- ◆ Cray T3D solution: network never drops, but receiver may
  ⇒ sender allocates space for returned packet
  ⇒ no copying
- ◆ Introduces out-of-order packets

Heterogeneous nodes... painful

Heterogeneous Nodes

Problem: reformat data during transfers
- ◆ Ex 1: big vs. little endian (MIPS vs. SPARC)
- ◆ Ex 2: IEEE floating point vs. Cray floating point
- ◆ Ex 3: ASCII data: newlines, special characters

Presentation layer converts to/from canonical form
- ◆ 30-97% of the overhead for TCP
- ◆ Requires nontrivial transformation of all data

Right solution for MPP (TCP?):
- ◆ Avoid canonical form
- ◆ Establish end-to-end transformation at setup
- ◆ *Sender* should transform data

Application-Level Framing

Idea: application-level unit is the basis for all steps

- Unit of error detection, encryption
- Unit of retransmission
- Unit of out-of-order delivery
- Unit of format transformations

Key aspect: ADUs can be handled out of order

- Example: individual block-transfer packets

Application handles errors:

- video/voice packets $\Rightarrow$ just ignore
- AAL5: lose one cell $\Rightarrow$ retransmit entire packet
- If transport layer detects partial drop $\Rightarrow$ drop rest of ADU
- Avoids unnecessary retransmission in lower layers

Application-Level Framing

Format transformations:

- Sender/Receiver share common naming scheme
  - Example: offsets into larger block transfer
- Receiver can transform out-of-order (since destination is known)
- Can move transformation to sender side (sometimes)

Flow control:

- ADUs are also unit of flow control
- Sender-side transformation can exploit delay cycles
- Claim: better performance
  - Perform entire ADU handling together
  - Then delay for flow control
  - Fewer context switches, better cache behavior