Advanced Topics in Computer Systems, CS262b

Brewer/Hellerstein

Parallel Systems: CM-5

Cm-5 Overview
Software Techniques for the CM-5 Data Network

I. Overview of the CM-5

- Made for many different programming models
- Data Parallel: one computation applied in parallel to lots of data => very simple control flow
- SIMD (single instruction, multiple data), ex CM-2
- SPMD (single program, multiple data), ex CM-5 with parallel fortran
- MIMD (multiple instruction, multiple data), subsumes SPMD but very complex in practice to due independent flow control and lack of synchronization
- The bottleneck is memory bandwidth, not CPU or even FPUs
- Commodity processor to ride Moore's law (more easily)
- Control Network simplifies data parallel model: barriers, reductions (ex: sum across all nodes), parallel prefix operations (ex: running sum for n nodes)
- partition into independent (smaller) systems
- Global address space, but physically distributed memory. No caching mechanism: every remote read/write uses the network, which is designed to handle the resulting traffic

II. Review of Active Messages

Approach: Treat message sending as the critical path and get everything off the critical path that you can.

Two primary sources of slow-down:

- generalized buffering & resource allocation,
- allowing for blocking/delayed server activity.

Active message solution:

- No buffering (beyond that needed for data transport).
- Short user-level receive handlers that may not block: either generate a reply message right away or extract the received message from the network somewhere into user space and return.
- Head of a message packet contains the address of the receive handler to run.

Subtle point 1: receiving costs more than sending:

- Send is integrated with the computation, which implies good cache and register usage, while receive is (relatively) unexpected.
- Dispatch overhead occurs only on the receiver

Subtle point 2: deadlock avoidance

- Sender must poll while blocked on injection to avoid deadlock
- Requires two logical networks with independent resources, so that replies can not even indirectly block requests. If we assume infinite resources at the receiver than we could avoid deadlock, since we could avoid deadlock by pulling out messages until our injection succeeded. (If we handle them rather than just pulling them out, then we again could have deadlock if a handler blocked.)

III. Software techniques for the CM-5 Data Network

Basics:

- *Topology*: the shape of the graph representing the network. Topologies aren't too important for this class. The big ones to know are the bus, ring, mesh, and butterfly.
- *Bisection Bandwidth*: the bandwidth of the min-cut of the network that partitions the nodes into two equal-sized groups, i.e., it bisects the machine.
- *Volume/Capacity*: the number of packets that the network can hold
- CM-5 network is packet based with guaranteed delivery and no deadlock

Capacity:

- Result: Capacity is about 10 packets per node
- $\Rightarrow$ Can only inject 10 packets before you need the attention of the receiver $\Rightarrow$ very difficult to overlap communication and computation
- Other reason it is difficult to overlap: very short wire time $\Rightarrow$ not worth much to hide it

Target Collisions:

- Def: two packets arrive at the same node at nearly the same time. One must be delayed while the other is handled. This delay can cause congestion.
- Key Result in Figure 4: communication pattern loses synchronization and becomes increasingly worse. *Sender groups* appear, in which several senders shift together from receiver to receiver.
- Solution: Barriers to maintain synchronization; work through the graph...

Bandwidth Matching:

- Simple idea: artificially limit the sender to the bottleneck rate, thus minimizing congestion. This is essentially static rate-based flow control.
- What does TCP do? (dynamic credit-based flow control) Why is this not so good for MPPs? (too much overhead $\Rightarrow$ reduces bandwidth)

- Option 1: poll once, thus forcing the alternation of sends and receives at each processor; fails to due excess congestion
- Option 2: poll until empty (no more arrivals). Two problems: 1) arrivals accrue faster than we can handle them, 2) fundamentally unfair: receiving decreases your ability to inject...
- Solution: (Bandwidth Matching) poll until empty, but also stall the sender when there are no arrivals. This allows the receiver to clear the network and make progress on its injections.
- Magically eliminates congestion, leading to better performance and 50x lower standard deviation. It is important to understand why this works and is self synchronizing.
- When does it fail?
  1) non-uniform use of receivers, 2) dynamic situations that involve long block transfers (which should use interleaving instead)

Packet Interleaving:

- What if you don't know the global pattern (and thus can't use barriers?)
- Basic idea: interleave packets to different destinations to make better use of receivers and avoid *head-of-line* blocking (HOL). HOL is common problem that affects many systems (under many disguises) — anything that has a queue (FIFO or LIFO) may suffer from this problem. The solution is (always?) to separate the queue into several smaller queues that are independent. CM-5 NI has an internal FIFO, so there's no point in using multiple queues in software.
- Having a single buffer to give messages to the NI is HOL at the interface level...

Three key points:

1. Adding barriers can improve performance by maintaining efficient use of shared resources (global scheduling).
2. Interleaving packets to multiple destinations can avoid HOL and helps to avoid prolonged hot spots at a receiver.
3. Bandwidth Matching can eliminate congestion when there is some knowledge about the global communication pattern.

Criticisms: Only one machine examined, although techniques are probably more general. Bandwidth matching is not very robust, but is a good optimization if you know what's going on.

Lesson: optimizing globally may be quite different than optimizing locally or pair-wise. For example, a single block transfer works best without BM, since congestion is irrelevant.

Questions:

- Why does some overlapping lead to 5% improvement?
  Because we transfer small enough blocks to keep them in the cache; that is, we switch between computation and communication just enough to make good use of the cache.
- What if we know that there are k senders for each receiver? Or more receivers than senders?
- How does DMA and interleaving interact? (hard)