Shared virtual memory and release consistency

So far, we have been talking about multiprocessors that have hardware support for non-local references.

However, shared-memory programming can also be supported in software on a distributed-memory machine, with the aid of an ordinary MMU.

In this case, the unit of sharing and migration must be

Also, because software controls migration, it will be much slower than true shared memory.

Here is how sharing and migration occur.

1. Processor $P_0$ encounters a page fault on a read reference and fetches a read-only copy of the page from secondary memory.

2. Processor $P_1$ encounters a page fault on a read reference and fetches a read-only copy of the page from $P_0$.

3. Now $P_0$ writes the page. What happens?

4. The OS gains control and invokes the SVM library to invalidate $P_1$’s copy of the page.

5. The SVM library gives $P_0$ write-access to the page.

6. Now if $P_1$ tries to read the page again, it needs to fetch a new copy from $P_0$ (through operations of the SVM library).

To find copies of pages, the SVM library uses a directory mechanism as described in Lecture 13.

What are some reasons why an SVM is slower than a hardware-supported approach?

- High overhead of invoking SVM operations. Why is it high?
• High overhead of communication. Why is it high?

• The processor is interrupted by incoming

⇒ Cost of satisfying a remote page fault is < 1 µs. on a hardware-coherent system, but 100s of µs. to > 1 ms. on an SVM system.

How does false sharing come into the picture?

Think of what SC requires. Each write may cause invalidations throughout the system. This is especially serious under false sharing.

Relaxed memory consistency
If we use a model like release consistency, we don’t have to propagate invalidations immediately. Why?

So we delay propagating the invalidations until a synchronization point. How is this different from a hardware-coherent system?

Fortunately, delaying invalidations means fewer invalidations overall.

Now, at which synchronization point should we propagate invalidations?

Under release consistency, a process does not really need to see a write (or an invalidation) until it does an acquire!

• If invalidations are propagated at a release, we have eager release consistency.

• If they are propagated at an acquire, we have lazy release consistency.

However, LRC requires greater attention to inserting synchronization operations correctly.
Here is a piece of code that works under ERC (and ordinary RC), but gets in an infinite loop under LRC:

```c
lock L1;
ptr = non_null_ptr_val;
unlock L1;
while (ptr == null) {};
lock L1;
a = ptr;
unlock L1;
```

What is the solution?

**Interconnection networks**

When more than one processor needs to access a memory structure, interconnection networks are needed to route data—

- from processors to memories (concurrent access to a shared memory structure), or
- from one PE (processor + memory) to another (to provide a message-passing facility).

Inevitably, a large bandwidth is required to match the combined bandwidth of the processing elements.

» One extreme is a *shared bus*. How does the cost scale as the number of processors $N$ increases?

How does the bandwidth scale?

» For concurrent access to shared memory, the ideal structure is a *crossbar switch*, which can simultaneously connect any set of processors to any set of distinct memory modules.
All $N$ processors can access all $M$ memory units with an $N \times M$ crossbar switch.

Since there are usually about as many processors as memories, as processors are added, the complexity of a crossbar switch grows as $N^2$.

How does the bandwidth scale?

For reasonably large values of $N$, the crossbar switch may be more expensive than the processors and memories.

» For message passing, the most general is the complete interconnection network—a path from each processor to every other processor.

Unfortunately, this requires bidirectional links. Cost grows with the square of $N$.

**Measures of interconnection performance**

Several metrics are commonly used to describe the performance of interconnection networks:

- **Connectivity**, or degree, the number of nodes that are immediate neighbors of a node (i.e., the number of nodes that can be reached from it in one hop).

- **Diameter**, the maximum number of nodes through which a message must pass on its way from source to destination. Diameter measures the maximum delay in transmitting a message from one processor to another.

What is the diameter of a crossbar?

- **Average distance**, where the distance between two nodes is defined by the number of hops in the shortest path between those nodes. Average distance is given by
\[
    d_{\text{avg}} = \frac{\sum_{d=1}^{r} (d \cdot N_d)}{N-1}
\]

where \(N\) is the number of nodes, \(N_d\) is the number of nodes at distance \(d\) apart, and \(r\) is the diameter.

- **Bisection width**, the smallest number of wires you have to cut to disconnect the network into two equal halves (+1).
  
  What is the bisection width of a crossbar

**Interconnection topologies**

[§10.4] An idealized interconnection structure—

- takes a set of \(n\) input ports labeled \(0, \ldots, n-1\) and
- sets up connections between them and a set of \(m\) output ports \(0, \ldots, m-1\),
- with the connections determined by control signals.

Usually we will assume that \(m = n\).

Here are some sample topologies.

1. **Ring.**
   
   Processor \(i\) directly connected to processors \(i+1\) (mod \(N\)) and \(i-1\) (mod \(N\)). Data can be moved from any processor to any other by a sequence of cyclic shifts.

   Motivation: Many parallel algorithms include calculations of the form
   
   \[
   X[i] := \frac{X[i-1] + X[j]}{2}
   \]
Usually every item of an array except the first and last is updated in this way.

The data routing can be defined by *routing functions*:

\[
C_{+1}(i) \equiv (i + 1) \mod N \\
C_{-1}(i) \equiv (i - 1) \mod N
\]

The processor interconnections can be diagrammed as a bidirectional ring:

The diameter of a bidirectional ring is ______. Its bisection width is

2. *Mesh interconnection network*

A mesh is similar to having “row & column” cyclic shifts.

One motivation: *Four-point iteration* is common in the solution of partial differential equations. Calculations of the form

\[
X[i, j] := (X[i+1, j] + X[i-1, j] + X[i, j-1] + X[i, j+1]) \div 4
\]

are performed frequently.
Here is an example of a 16-node mesh. Note that the last element in one row is connected to the first element in the next.

If the last element in each row were connected to the first element in the same row, we would have a **torus** instead.

In the Illiac IV, each processor \( i \) was connected to processors:

\[ \{i+1, \ i-1, \ i+8, \ \text{and} \ i-8\} \pmod{64}. \]

Here are the routing functions:

\[
\begin{align*}
R_{+1}(i) &\equiv (i + 1) \pmod{N} \\
R_{-1}(i) &\equiv (i - 1) \pmod{N} \\
R_{+r}(i) &\equiv (i + r) \pmod{N} \\
R_{-r}(i) &\equiv (i - r) \pmod{N} \quad \text{where} \ r = \sqrt{N}
\end{align*}
\]

The diameter of an Illiac IV mesh is \( \sqrt{N} - 1 \). For example, in a 16-node mesh structure, it takes a maximum of 3 steps. To see that, let us look at the mesh interconnection network shown in the form of a **chordal ring**:

![Mesh Diagram]
In a 64-element mesh, any node can be reached from any other in no more than 7 of these shifts.

Without the end-around connections (a “pure” 2D mesh), the diameter is $2(\sqrt{N} - 1)$.

It is also possible to have a multidimensional mesh. The diameter of a $d$-dimensional mesh is $d(N^{1/d}) - 1$ and its bisection width is $N^{(d-1)/d}$.

The average distance is $d \times 2(N^{1/d})/3$ (without end-around connections).

3. Hypercube

A hypercube is a generalized cube. In a hypercube, there are $2^n$ nodes, for some $n$. Each node is connected to all other nodes whose numbers differ from it in only one bit position.

The routing functions have this form:

$$C_i(a_{n-1} \ldots a_{i+1} a_i a_{i-1} \ldots a_0)_2 \equiv (a_{n-1} \ldots a_{i+1} \bar{a}_i a_{i-1} \ldots a_0)_2$$
An interconnection network can be either single stage or multistage.

- If it is single stage, then the individual control boxes must be set up to $n$ times to get data from one node to another. Data may have to pass through several PEs to reach its destination.

- Multistage networks have several sets of switches in parallel, so data only needs to pass through several switches, not several processors.

For a multistage cube network, we can diagram the paths from one cell to another like this:

A multistage cube network is often called an indirect binary $n$-cube.

What is the diameter of a hypercube?

The average distance is $n/2$.

4. Barrel shifter

A barrel shifter is sometimes called a plus-minus- $2^i$ network.

Routing functions:

$$B_{+i}(j) = (j + 2^i) \mod N$$
$$B_{-i}(j) = (j - 2^i) \mod N$$
where $0 \leq j \leq N-1$ and $0 \leq i < \log_2 N$. For example, a 16-node barrel-shifter network would have not just ±1 and ±4 shifts, like a mesh network, but also _______ shifts. What about a 64-node barrel shifter?

Each node in an $N$-node barrel shifter is directly connected to _______ different nodes.

Here is a diagram of a barrel shifter for $N = 16$:

Each node in an $N$-node barrel shifter is directly connected to different nodes.

What is the diameter of this barrel shifter?

In general, the diameter of a barrel shifter is $(\log_2 N)/2$.

5. Perfect-shuffle interconnection

This interconnection network is defined by the routing function

$$S ((a_{n-1} \ldots a_1a_0)_2) = (a_{n-2} \ldots a_1a_0a_{n-1})_2$$
It describes what happens when we divide a card deck of, e.g., 8 cards into two halves and shuffle them “perfectly.”

We can draw the processor interconnections required to obtain this transformation (at near right):

If the links are bidirectional, the inverse perfect shuffle is obtained (above, right).

6. Omega network

By itself, a shuffle network is not a complete interconnection network. This can be seen by looking at what happens as data is recirculated through the network:
An *exchange* permutation can be added to a shuffle network to make it into a complete interconnection structure:

\[ E(a_{n-1} \ldots a_1 a_0)_2 + a_{n-1} \ldots a_1 a_0 \]

A shuffle-exchange network is isomorphic to a cube network, with a suitable renumbering of boxes.

An omega network is a \( \log_2 N \)-stage shuffle-exchange interconnection, with individual cells that can perform four different operations:

- Pass-through
- Exchange
- Broadcast low
- Broadcast high

Sums (or other operations involving all the elements) can be performed in \( \log N \) steps.

In addition, with a shuffle-exchange network, arbitrary cyclic shifts of an \( N \)-element array can be performed in \( \log N \) steps.

Here is a diagram of a multistage omega network for \( N = 8 \).

![Diagram of a multistage omega network for N = 8.](image)

The diameter of an omega network is

The bisection width is \( N \).
7. Butterfly network

Closely related to shuffle-exchange networks.

The butterfly permutation is defined as—

\[ B(a_{n-1} a_{n-2} \ldots a_1 a_0) \equiv a_0 a_{n-2} \ldots a_1 a_{n-1} \]

i.e., the permutation formed by interchanging the most- and least-significant bits in the binary representation of the node number.

This permutation can be diagrammed as shown at the right:

Two variants of the butterfly permutation are the \( k \)th sub-butterfly, performed by interchanging bits 0 and \( k-1 \) in the binary representation—

\[ B_k(a_{n-1} a_{n-2} \ldots a_1 a_0) \equiv a_{n-1}a_{n-2}a_k a_0 \ldots a_1 a_{k-1} \]

and the \( k \)th super-butterfly, performed by interchanging bits \( n-1 \) and \( k-1 \):

\[ B^k(a_{n-1} a_{n-2} \ldots a_1 a_0) \equiv a_{k-1}a_{n-2}a_k a_{n-1}a_{k-2} \ldots a_0 \]

8. Beneš network

As we have seen, a crossbar switch is capable of connecting a set of inputs to any set of distinct outputs simultaneously.

A shuffle-exchange, or multistage cube, network is not capable of doing this. (It is easy to come up with an example.)

Is it possible to achieve an arbitrary permutation of input-output combinations with less than a full crossbar switch?

Yes. The Beneš network substitutes two \( N/2 \times N/2 \) crossbar switches, plus an \( N \)-input exchange switch for a full crossbar switch, as shown below.
The resulting \( N/2 \times N/2 \) crossbar switches can be similarly reduced.

Through this process, a full connection network can be produced from \( 2 \times 2 \) switches at significantly lower cost than a full crossbar:

The stages of a Beneš network are connected by shuffle and inverse-shuffle permutations.

The network is called *rearrangeable*, since the switch settings can always be rearranged to accommodate any input-output mapping.

In some Beneš networks, the switches are capable of performing broadcasts, as well as pass-through or interchange.

Such Beneš networks can achieve all \( N^N \) possible input/output mappings.