Chapter 10

Routing on a 2-D Mesh or Tours

• Learn how to route multiple data items to their respective destinations
  (in PRAM, routing is nonexistent and is the circuit model it is hardwired)
• Become familiar with issues in packet routing and wormhole routing
### 10.1 Types of Data Routing Operations

#### One-to-one communication (point-to-point messages)

- Place sources
- Place destinations
- Propagate

#### Collective communication (per the MPI standard)

- **a.** One to many: broadcast, multicast, scatter
- **b.** Many to one: combine, fan-in, gather
- **c.** Many to many: many-to-many m-cast, all-to-all b-cast, scatter-gather (gossiping), total exchange

#### Some special data routing operations

- **a.** Data compaction or packing

#### Random-access write (RAW):

- Emulating one memory write step of a PRAM with p processors

#### Random-access read (RAR):

- Emulating one memory read step of a PRAM with p processors
Routing within a row or column

![Diagram of a linear array showing routing](image)

**Fig. 10.4.** Example of routing multiple packets on a linear array.
10.3 Data Routing on a 2D Array

Exclusive random-access write on a 2D mesh: MeshRAW

1. Sort packets in column-major order by destination column number; break ties by destination row number

2. Shift packets to the right, so that each item is in the correct column. There will be no conflict since at most one element in each row is headed for a given column

3. Route the packets within each column

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0.2</td>
<td>1.0</td>
<td>3.2</td>
</tr>
<tr>
<td>1</td>
<td>2.3</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3.0</td>
<td>1.1</td>
<td>3.1</td>
</tr>
<tr>
<td>3</td>
<td>1.2</td>
<td>2.2</td>
<td></td>
</tr>
</tbody>
</table>

Initial state
```

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0.0</td>
<td>3.1</td>
<td>3.2</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
<td>0.2</td>
<td>1.3</td>
</tr>
<tr>
<td>2</td>
<td>3.0</td>
<td>1.2</td>
<td>2.3</td>
</tr>
<tr>
<td>3</td>
<td>1.1</td>
<td>2.2</td>
<td></td>
</tr>
</tbody>
</table>

After column-major-order sorting by dest'n column
```

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0.0</td>
<td>3.1</td>
<td>3.2</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
<td>0.2</td>
<td>1.3</td>
</tr>
<tr>
<td>2</td>
<td>3.0</td>
<td>1.2</td>
<td>2.3</td>
</tr>
<tr>
<td>3</td>
<td>1.1</td>
<td>2.2</td>
<td></td>
</tr>
</tbody>
</table>

After row routing
```

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0.0</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
<td>1.1</td>
<td>1.2</td>
</tr>
<tr>
<td>2</td>
<td>3.0</td>
<td>3.1</td>
<td>3.2</td>
</tr>
</tbody>
</table>

After column routing
```

Fig. 10.5. Example of random-access write on a 2D mesh.

Not a shortest-path routing algorithm

e.g., packet headed to (3, 1) first goes to (0, 1)

But fairly efficient

\[
T = 3p^{1/2} + o(p^{1/2}) \quad \{\text{snakelike sorting}\}
+ p^{1/2} \quad \{\text{column reversal}\}
+ 2p^{1/2} - 2 \quad \{\text{row & column routing}\}
= 6p^{1/2} + o(p^{1/2})
\]

Or \(11p^{1/2} + o(p^{1/2})\) with unidirectional communication
10.4 Greedy Routing Algorithms

Greedy: pick a move that causes the most progress toward the destination in each step

Example greedy algorithm: dimension-order (e-cube)

![Diagram of greedy row-first routing on a 2D mesh.]

Fig. 10.6. Example of greedy row-first routing on a 2D mesh.

\[ T = 2p^{1/2} - 2 \text{ but requires large buffers} \]

![Diagram demonstrating the worst-case buffer requirement with row-first routing.]

Fig. 10.7. Demonstrating the worst-case buffer requirement with row-first routing.
Routing algorithms thus far

Slow $6p^{1/2}$, but with no conflict (no additional buffer)
Fast $2p^{1/2}$, but with large node buffers

An algorithm that allows trading off time for buffer space

![Diagram](image-url)

**Fig. 10.8.** Illustrating the structure of the intermediate routing algorithm.

$$T = 4p^{1/2}/q + o(p^{1/2}/q) \quad \text{column-major block sort}$$

$$+ 2p^{1/2} - 2 \quad \text{route}$$

$$= (2 + 4/q)p^{1/2} + o(p^{1/2}/q)$$

Buffer space per node

$$r_k = \text{number of packets in } B_k \text{ headed for column } j$$

$$\sum_{k=0}^{q-1} r_k < \sum_{k=0}^{q-1} \left(1 + \frac{r_k}{p^{1/2}/q}\right) \leq q + (q/p^{1/2})\sum_{k=0}^{q-1} r_k \leq 2q$$
10.5 Other Classes of Routing Algorithms

Row-first greedy routing has very good average-case performance, even if the node buffer size is restricted.

Idea: Convert any routing problem to two random instances by picking a random intermediate node for each message.

Using combining for concurrent writes:

![Combining of write requests headed for the same destination.](image)

**Fig. 10.9.** Combining of write requests headed for the same destination.

Terminology for routing problems or algorithms:

- **Static:** packets to be routed all available at $t = 0$
- **Dynamic:** packets “born” in course of computation
- **Off-line:** routes precomputed, stored in tables
- **On-line:** routing decisions made on the fly
- **Oblivious:** path depends only on source & destination
- **Adaptive:** path may vary by link and node conditions
- **Deflection:** any received packet leaves immediately, even if this means misrouting (via detour path); also known as hot-potato routing
10.6 Wormhole Routing

Fig. 10.10. The notions of worms and deadlock in wormhole routing.

Any routing algorithm can be used to choose the path taken by the worm, but practical choices limited by the need for a quick decision

Example: row-first routing, with 2-byte header for row & column displacements

Fig. 10.11. Various ways of dealing with conflicts in wormhole routing.
The deadlock problem in wormhole routing

Two strategies for dealing with deadlocks:

1. Avoidance
2. Detection and recovery

Checking for deadlock potential via link dependence graph; existence of cycles may lead to deadlock

Fig. 10.12. Use of dependence graph to check for the possibility of deadlock.
Using virtual channels
Several virtual channels time-share one physical channel
Virtual channels serviced in round-robin fashion

Fig. 10.13. Use of virtual channels for avoiding deadlocks.

Figure for Problem 10.14.