6.895 Theory of Parallel Systems	Lecture 19

Squish Routing and Disk-drive Hardware

Lecturer: Bradley C. Kuszmaul *Scribe: Advait D. Karande and Zhang JiaHui*

Lecture Summary

1. *Squish routing*
   We begin by defining what a squish message pattern is.
2. *Squish theorem*
   In this section, we prove that squish routing can always be performed without collisions on a butterfly network.
3. *An application of squish routing*
   This section explains how one-to-one routing can be performed as a sequence of squish routings.
4. *Implementing squish routing*
   This section explains how squish routing can be implemented using parallel prefix (scan) operation.
5. *Analysis of Scan and Routing*
   The cost of scanning in squish-up/squish-down and the bound on the routing time with N processors are analyzed.
6. *Disk Drives-Hardware*
   This section introduces some basic concepts of a disk drive, its operations (read and write), seek time and spin speed.

1 Squish Routing

A **squish** message pattern is defined as follows. Consider a set of k processors out of n available processors ($k < n$). These processors are labeled S_i ($0 \leq i \leq k-1$) such that

$$0 \leq P_{S_0} < P_{S_1} < \ldots < P_{S_{k-1}} < n$$

where P_{S_i} indicates the ID of the processor labeled S_i. P_{S_i} sends a message to processor with ID i. Figure 1 shows a squish pattern, wherein processors 1, 2 and 4 send messages to processors 0, 1 and 2 respectively.

Now consider a butterfly network with $n = 8$ processors. Figure 2 shows a squish involving 5 sources on such a network.

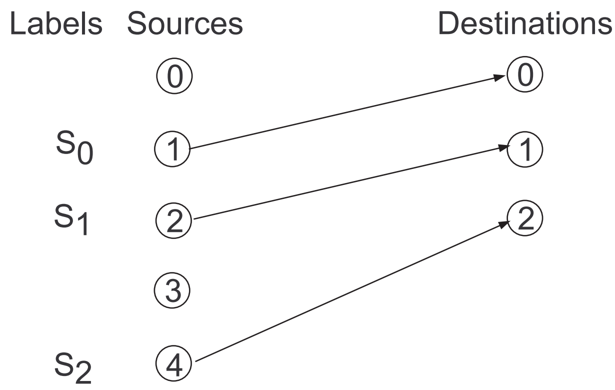

Figure 1: An example of a squish pattern.

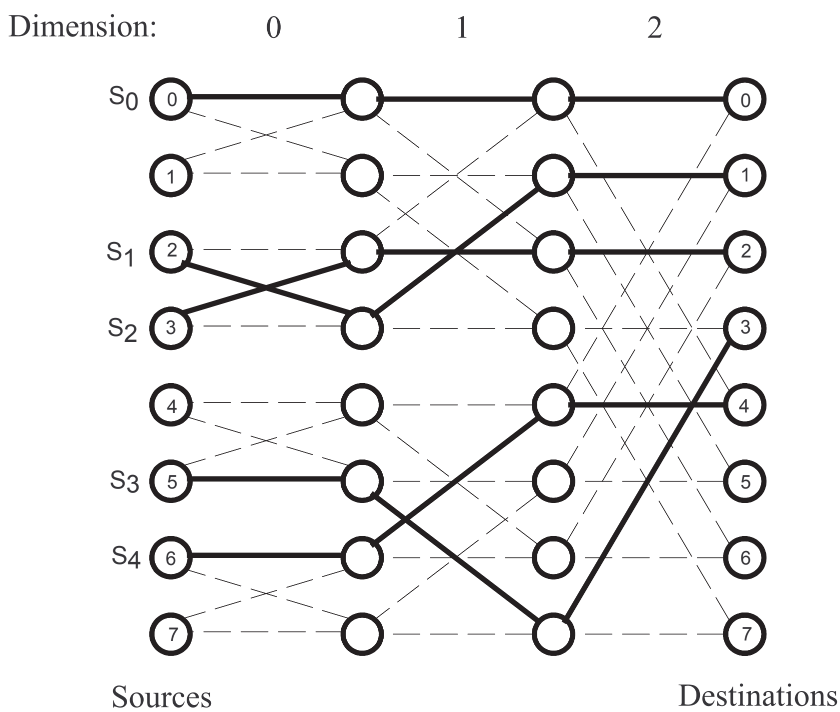

Figure 2: A squish pattern on a butterfly network with $n = 8$ inputs. Nodes 0, 2, 3, 5 and 6 send messages to nodes 0, 1, 2, 3 and 4 respectively. Thin, dashed lines indicate the underlying network connections, while bold lines indicate the message routes.

It is apparent that the messages in the squish have been routed without any collisions. In fact, we will now prove that squish routing can always be performed on a butterfly network without any collisions.

2 Squish theorem

Theorem 1 *Squish routing can always be performed without any edge-conflicts on a butterfly network.*

We begin by defining the terms necessary for the proof.

Definition 2 *A **semi-contraction** is a one-to-one routing from a set of k source processors* $\{s_0, s_1, ..., s_{k-1}\}$ *to k destination processors* $\{d_0, d_1, ..., d_{k-1}\}$ *such that processor* s_i *sends a message to* d_i *and*

$$\forall i, j : |s_i - s_j| \geq |d_i - d_j| \tag{1}$$

Claim 3 *Every squish is a semi-contraction.*

Lemma 4 *Semi-contractions route without an edge-conflict on a butterfly network.*

Proof By Contradiction. We show that if a conflict exists, then the defining equation of a semi-contraction (Equation (1)) no longer holds.

Consider a butterfly network with n processors running from minor to major cycles, similar to the one shown in Figure 2. Assume that each processor ID is given by $r = lg(n)$ bits. Since the network runs from minor to major cycles, a route from source to destination is found by sequentially setting or resetting bits in source processor ID from LSB to MSB. Now we consider a semi-contraction for this network.

Suppose there exists an edge-conflict. This implies that there exist i and j such that the route from s_i to d_i shares at least an edge with the route from s_j to d_j. Let l be the dimension of the shared edge as shown in Figure 3.

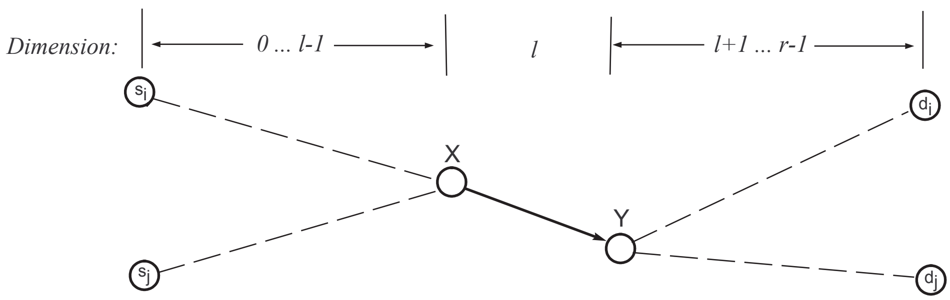

Figure 3: An edge-conflict on an $n = 2^r$ input butterfly network - Routes $s_i \to d_i$ and $s_j \to d_j$ share an edge at dimension l.

Let the bit-pattern of the processor IDs of the two source nodes be as follows:

$$\begin{aligned} s_i &: b_{r-1}b_{r-2}...b_l b_{l-1}...b_0 \\ s_j &: a_{r-1}a_{r-2}...a_l a_{l-1}...a_0 \end{aligned}$$

Since the routes from both s_i and s_j reach point Y as shown in Figure 3, the most significant bits up to dimension l of s_i and s_j must be the same. That is:

$$b_{r-1}b_{r-2}...b_l = a_{r-1}a_{r-2}...a_l$$

Therefore, the distance between the IDs of the two source processors is:

$$\begin{aligned} |s_i - s_j| &= \left| \sum_{m=0}^{r-1} (b_m - a_m) \cdot 2^m \right| \\ &= \left| \sum_{m=0}^{l-1} (b_m - a_m) \cdot 2^m \right| \\ &\leq \left| \sum_{m=0}^{l-1} 2^m \right| \\ &= 2^l - 1 \end{aligned} \tag{2}$$

Similarly, let the bit-patterns of the destination processors be represented as follows.

$$\begin{aligned} d_i &: f_{r-1}f_{r-2}...f_l f_{l-1}...f_0 \\ d_j &: e_{r-1}e_{r-2}...e_l e_{l-1}...e_0 \end{aligned}$$

Since the routes to both d_i and d_j have reached the same point Y (Figure 3), the $l+1$ least significant bits in the IDs of these processors must be the same. That is:

$$f_l f_{l-1}...f_0 = e_l e_{l-1}...e_0$$

Therefore, the distance between the IDs of the two destination processors is:

$$\begin{aligned} |d_i - d_j| &= \left| \sum_{m=0}^{r-1} (f_m - e_m) \cdot 2^m \right| \\ &= \left| \sum_{m=l+1}^{r-1} (f_m - e_m) \cdot 2^m \right| \end{aligned}$$

Replacing m with $i = m - (l+1)$:

$$|d_i - d_j| = \left| \sum_{i=0}^{r-l-2} (f_{i+l+1} - e_{i+l+1}) \cdot 2^i \cdot 2^{l+1} \right| \tag{3}$$

$$= 2^{l+1} \cdot \left| \sum_{i=0}^{r-l-2} (f_{i+l+1} - e_{i+l+1}) \cdot 2^i \right| \tag{4}$$

Given that the two destination IDs must differ by at least 1 bit, the absolute value of summation in the above equation must be at least 1. Hence,

$$|d_i - d_j| \geq 2^{l+1} \tag{5}$$

From Equations (2) and (5),

$$|s_i - s_j| < |d_i - d_j| \tag{6}$$

which contradicts the defining equation of a semi-contraction. □

The proof of Theorem 1 follows from Lemma 4 and Claim 3.

3 An application of squish routing

Squish routing can be used to perform one-to-one routing on a butterfly or a hypercube. The key idea is to decompose one-to-one routing into a sequence of squish-routings.

Consider an r-dimensional hypercube (or a butterfly with 2^r inputs). The ID of each node is given by bit-sequence $b_{r-1}b_{r-2}...b_0$. An algorithm to implement one-to-one routing on such a network is as follows.

For each dimension i = r-1 **downto** 0
A. Squish down all messages whose i^{th} bit in the destination ID is 0.
B. Squish up all messages whose i^{th} bit in the destination ID is 1.

The one-to-one routing, thus, completes after r rounds of squish routings. Figure 4 provides an example.

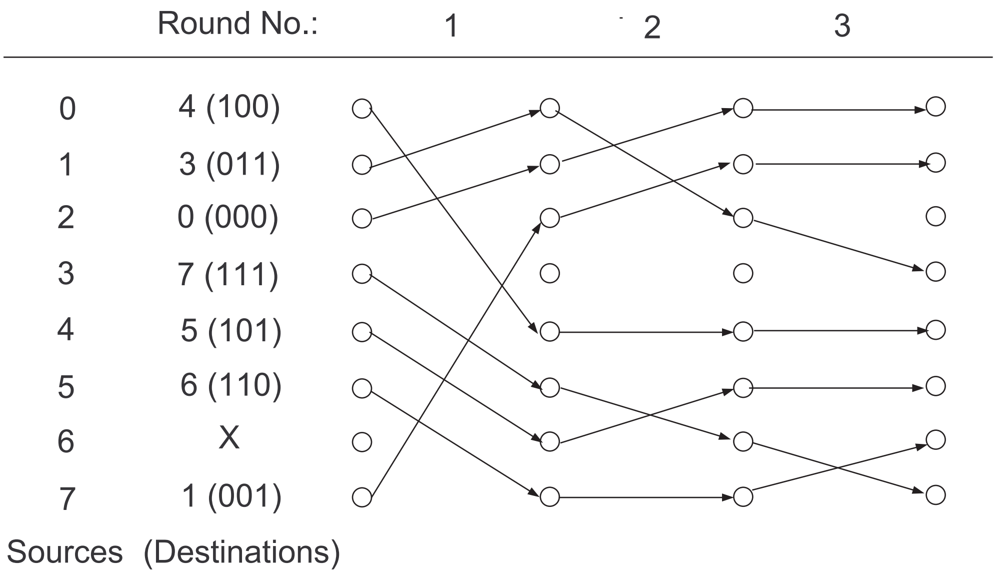

Figure 4: One-to-one routing as a sequence of squish routings. Processors 0, 1, 2, 3, 4, 5 and 7 send messages to processors 4, 3, 0, 7, 5, 6 and 1 respectively. The routing completes after 3 rounds.

4 Implementing squish routing

In order to implement squish routing on a hypercube, for each source processor, a destination processor must be determined in each round. This involes computing an enumeration of the set $S = \{S_i\}$. We define a bit vector v such that:

$$v_i = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

The destination processor for each source processor i ($i \in S$) in a round is the number of 1's in v before i. It is found by computing the prefix sum, ω, of vector v, where:

$$\omega_i = \sum_{j=0}^{i-1} v_i$$

We can compute ω shown in Figure 5. Notice that there is no parallelism in this DAG.

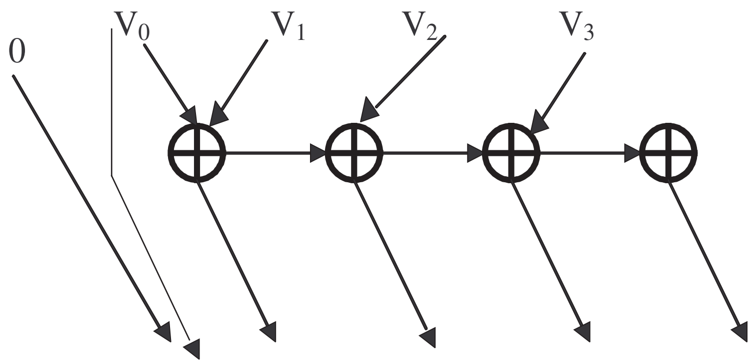

Figure 5: An example of DAG that computes the vector ω.

The work and critical path for this computation are as follows.

$T_1 = O(n)$
$T_\infty = O(n)$

Can we do it faster? Yes, for example, by using a tree structure as shown in Figure 6.
The work and critical path for this tree structure are:

$T_\infty = O(lg(n))$
$T_1 = O(n^2)$

Here the work is proportional to the number of adders. If we reuse the adders/results, we can even achieve

$T_1 = O(nlg(n))$

The parallel prefix operation on a vector is also known as a scan operation.

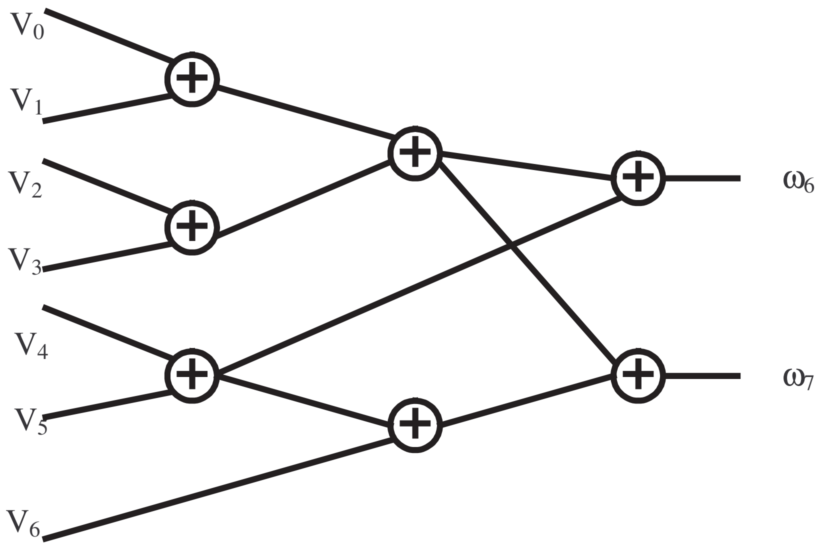

Figure 6: An example of a tree-structure that computes the vector ω.

5 Analysis of Scan and Routing

Time to perform a scan is $O(lg(n))$:

depth of graph = $lg(n)$
The wires carry O(1) bits/unit time
The cost of addition: $lg(n)$ to add 2 n-bit numbers

Now we will show that the time to perform a scan is $O(lg(n))$, if we pipeline the bits through the network. How does the pipeline work? Let's illustrate it with a diagram. In Figure 7,we are trying to add bits 0 1 1 0 1 1 0 1, and the expected result is 101 (5 in decimal).

In Figure 7, we can see that there is no pipeline delay; after $lg(n)$ clock ticks, the first bit comes out. After another $lg(n)$ clock ticks, the last bit comes out.

Now, we will analyze the time required for squish-routing on N processors, $T(N)$. Here M is defined as the message size in bits.

Squish up:
scan — $lg(n)$
squish — $M + lg(n)$

Squish down:
scan — $lg(n)$
squish — $M + lg(n)$

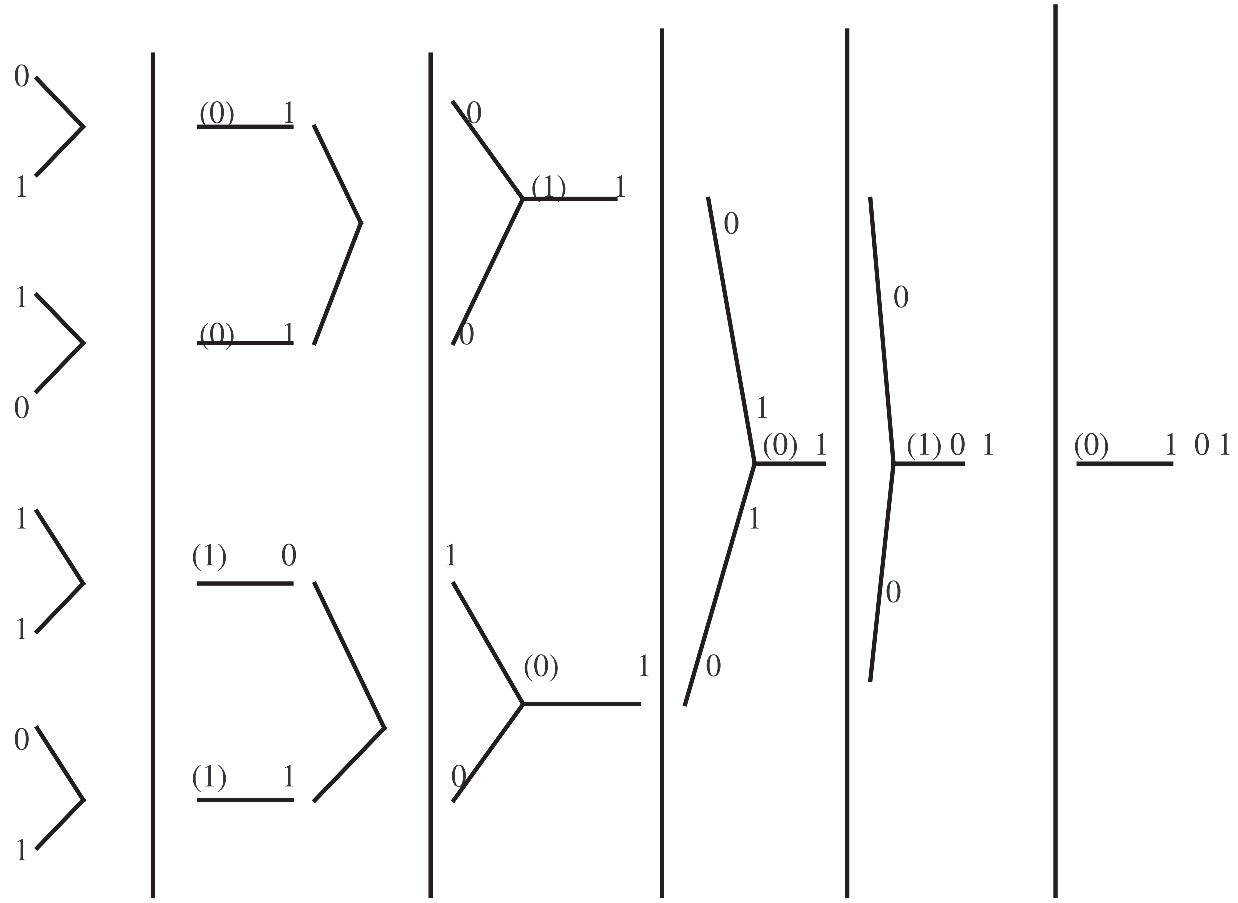

Figure 7: Adding the 8 bits 0 1 1 0 1 1 0 1 by pipelining the bits through the network. Bits in parentheses are carry bit.

So, we obtain the recurrence

$$T(N) = O(M) + O(lg(n)) + T(N/2)$$

Here, M is the message size, which is $\Omega(lg(N))$, since we at least need to encode the destination address in the message. Therefore we have:

$$\begin{aligned}
T(N) &= O(M) + O(lg(n)) + T(N/2) \\
&= O(M) + T(N/2) \\
&= O(M) + O(M) + T(N/4) \\
&= O(M) + O(M) + O(M) + T(N/8) \\
&= ... \\
&= O(Mlg(N))
\end{aligned}$$

The time required for squish-routing on N processors with a message size of M bits is therefore $O(Mlg(N))$.

6 Disk Drives - Hardware

Figure 8 shows a Disk Drive with its head controlled by a motor. How does it work?

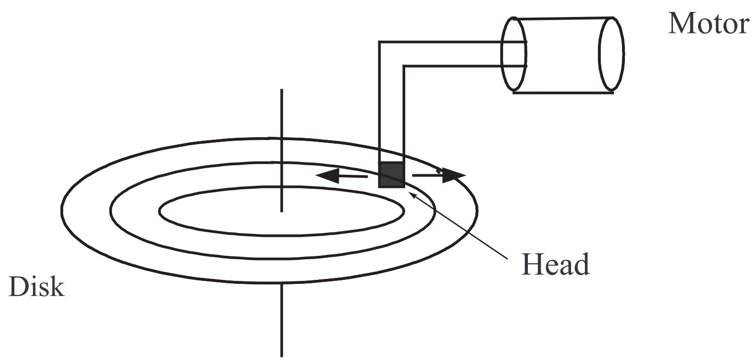

Figure 8: A disk drive with its head controlled by a motor.

In order to read/write a bit from/to a disk, the head is placed above the region of the disk that represents the bit-position.

- Write operation: The head consists of an electromagnet. It writes the bit-value by magnetizing the region with appropriate polarity.
- Read operation: The head also includes a coil. The value of a bit is determined by the direction of the current the bit-region induces in the coil.

In modern hard disks, the read operation is based on the Giant Magneto Resistance effect (GMR). The head includes a read sensor, whose resitance is affected by the magnetic orientation of bit regions. Such read sensor is highly sensitive and supports high-density , high-capacity disks. The GMR effect was discovered in late 1980's. It took about 10 years for it to appear in commodity disks.

As a disk spins, the motor moves the head back and forth (”seek”) radially above the disk, and it can access any spot on disk. The motor is a linear induction motor. Initially, hard disks used to have stepper motors. For such hard disks, the relationship between the seek time, t, and the distance traveled, x, is

$t = \Theta(x)$. With linear induction motors, head accelerates and decelerates to let the seek time be of the order of $\Theta(\sqrt{x})$.

Today, the average seek time is 4.9 ms for a read, and 5.4 ms for a write. The best case seek time is around 1 ms and the worst case seek time is around 9ms. The rotation speed ranges from 5400 rpm to 10000 rpm.

In high performance applications, which access the data very frequently, the data is placed on the outer tracks of the disk to reduce the seek time.