6.895 Theory of Parallel Systems Lecture 18

Hypercube Networks-III

Lecturer: Charles Leiserson *Scribe: Sriram Saroop and Wang Junqing*

Lecture Summary

1. *Review of the previous lecture*
   This section highlights the problem of routing a hard problem on the butterfly network.
2. *Valiant's Algorithm*
   This section describes the idea behind Valiant's algorithm for solving a hard problem on the butterfly network.
3. *Ranade's Algorithm*
   This section briefly introduces Ranade's algorithm.
4. *Shuffle-exchange graph*
   This section explains the properties of the Shuffle-exchange graph.
5. *DeBruijn Network*
   This section introduces the DeBruijn Network, and illustrates the DeBruijn sequence with the aid of a card trick.
6. *Bisection Width for Shuffle-Exchange graph*
   This section provides the proof for deriving the bisection width for the shuffle-exchange graph.

1 Review of the previous lecture

- Routing on an N-node (i.e. $N/\lg N$ input)butterfly takes $O(\lg n)$ time with high probability, assuming all routing problems are equally likely.
- But, some problems are still hard. For example, the Matrix transpose problem is hard. If we consider the routing from $X_1X_2X_3X_40000 \rightarrow 0000X_1X_2X_3X_4$, essentially all inputs have to route through a common intermediate node. This is a subproblem of the matrix transpose problem wherein $(i, j) \rightarrow (j, i)$. The indices i and j are represented by bits and are concatenated to form the input. Therefore, matrix transpose incurs $\sqrt{n}$ congestion. Hence, we find that the butterfly network has a bad worst case ($\sqrt{n}$), even though it has a good average case.

2 Valiant's Algorithm

This is a *permutation routing* algorithm. The idea is to convert one hard problem into two easy ones. The routing is performed in two phases:

- **Phase 1:** Firstly, we route from the sources to random intermediate nodes.
- **Phase 2:** Then, we route from the intermediate nodes to the destination nodes.

Both phases 1 and 2 are *random routing problems* which we had analyzed and found them to be good cases for the butterfly. Hence, there is no hard routing problem. We can however be *unlucky*, but it is quite *unlikely* that we are unlucky. The *unluckiness* does not correspond to the choice of input because we cannot put a bound on the choice of input. However, we can bound probabilistically how likely we could have a bad routing problem.

It is to be noted that an adversary cannot elicit the worst case behavior. In fact, the adversary cannot provide a bad routing problem because every problem is equally bad. In conclusion, by randomizing the chosen intermediate nodes which are destinations in Phase 1 and sources in Phase 2, we have essentially distributed the congestion, thereby making the sub-problems easy ones.

3 Ranade's Algorithm

It is a routing algorithm which runs in $\mathrm{O}(\lg n)$ time with high probability. It has some interesting properties like bounded queues, queues that are of size O(1). The proof is through means of an elegant delay sequence argument.

4 Shuffle-exchange graph

4.1 Description

Definition 1 (Shuffle Exchange Graph) *Let $d \epsilon N$. The d-dimensional shuffle-exchange SE(d) is defined as an undirected graph with node set $V = [2]^d$ and an edge set $E = E_1 \cup E_2$ with*

$E_1 = \{\{(a_{d-1}, ..., a_0), (a_{d-1}, ..., \overline{a_0})\} | (a_{d-1}, ..., a_0) \epsilon [2]^d, \overline{a_0} = 1 - a_0\}$

and

$E_2 = \{\{(a_{d-1}, ..., a_0), (a_0, a_{d-1}, ..., a_1)\} | (a_{d-1}, ..., a_0) \epsilon [2]^d\}$.

It is based on an N node hypercube, where $N = 2^d$. Figure 1 shows an 8-node shuffle-exchange graph.

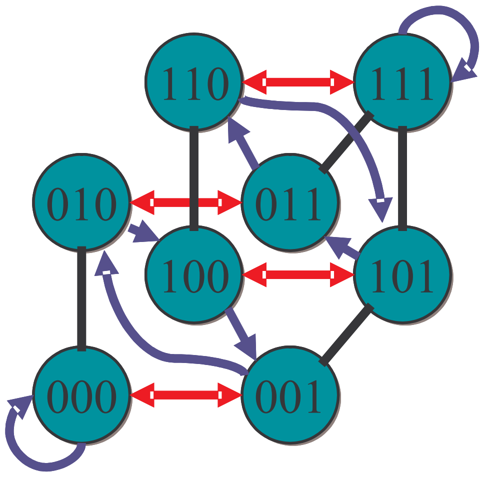


Figure 1: Shuffle-exchange graph

Each node is labelled with a unique $\log N$ bit string. A node labelled a $= a_{\log N-1}, ..., a_0$ is linked to a node labelled b $= b_{\log N-1}, ..., b_0$ by a *shuffle edge* if rotating one position to the left or right yields b, i.e., if either b $= a_0, a_{\log N-1}, a_{\log N-2}, ..., a_1$ or b $= a_{\log N-2}, a_{\log N-3}, ..., a_0, a_{\log N-1}$.Two nodes labelled a and b are linked by an *exchange edge* if a and b differ in only the least significant (rightmost) bit, i.e., b = $a_{\log N-1}, a_{\log N-2}, ..., \overline{a_0}$. In Figure 1, shuffle edges are the blue edges, and exchange edges are the horizontal red edges. Figure 2 shows another view of the same 8 node shuffle-exchange graph. To summarize, if a node

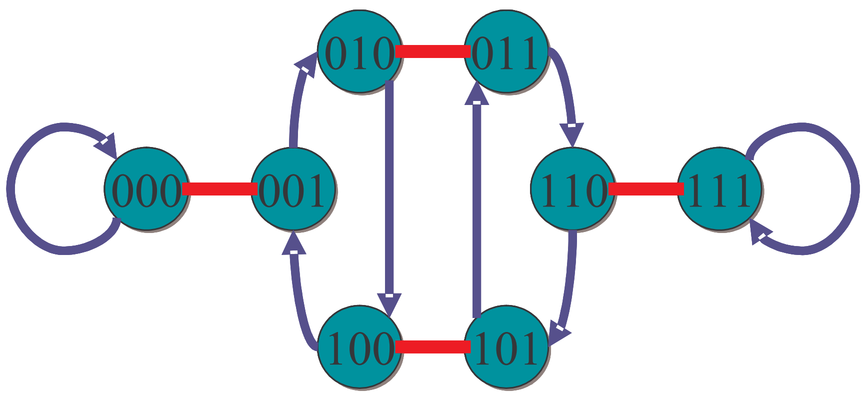


Figure 2: Shuffle-exchange graph: Another view

is a d-bit binary number, *Exchange edges* are between $< b_{d-1}, b_{d-2}, ..., b_1, 0 > \leftrightarrow < b_{d-1}, b_{d-2}, ..., b_1, 1 >$. *Shuffle edges* are from $< b_{d-1}, b_{d-2}, ..., b_1, b_0 > \rightarrow < b_{d-2}, ..., b_0, b_{d-1} >$, which corresponds to a left-cyclic shifting of the bits.

4.2 Properties

- Degree = 3 (Out-degree = 2 , In-degree = 2)
- Number of edges ≤ 2N(in the case of an undirected graph). This is because, the sum of the in-degrees = number of edges , according to the Hand-shaking lemma.
- Diameter ≤ 2 lg N. This corresponds to $\lg N$ shuffle edges and $\lg N$ exchange edges.
- Bisection Width = O($N/\lg N$). This is not obvious , and is a bit hard to show.

It can be seen that every property of the butterfly is also a property of a shuffle-exchange graph. However, the properties were easier to prove in the case of the butterfly because it was more intuitive.

4.3 A Perfect (Out) Shuffle

The connection pattern in Figure 3 is called a *perfect shuffle* because it is like shuffling two halves of a card deck, so that the cards from the two decks interleave perfectly. We notice that the top and bottom cards which correspond to bit patterns 000 and 111 do not change positions, whereas the other positions go a position which is double in value, that also corresponds to a left shift of one bit [1]. For example, 001 goes to 010 that is double of it. Hence, we can conclude that **Shuffle(X) ≡ 2X mod (N-1)**. Moreover we refer to this shuffle, wherein the top and bottom cards retain their positions to be a *Perfect Out-Shuffle*.

4.4 A Perfect (In) Shuffle

In a perfect in-shuffle as shown in Figure 4 the top and the bottom cards are *in* the deck rather than *out*, which means that they do not retain their positions.

A perfect in-shuffle can be viewed as a perfect out-shuffle followed by an exchange, which is illustrated in Figure 5.

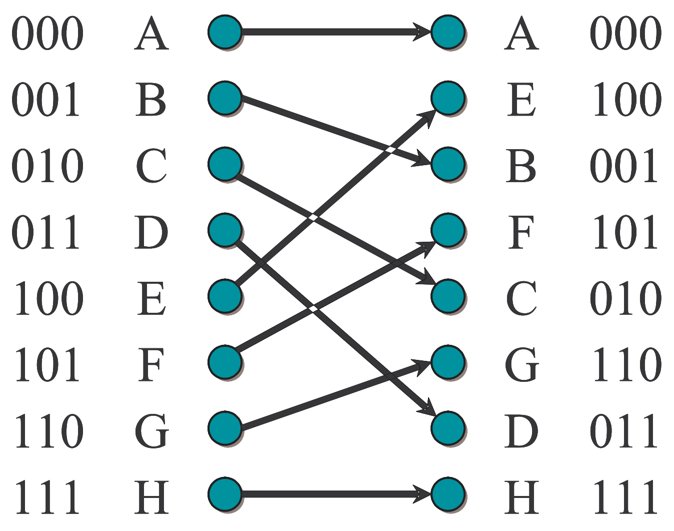


Figure 3: A perfect shuffle

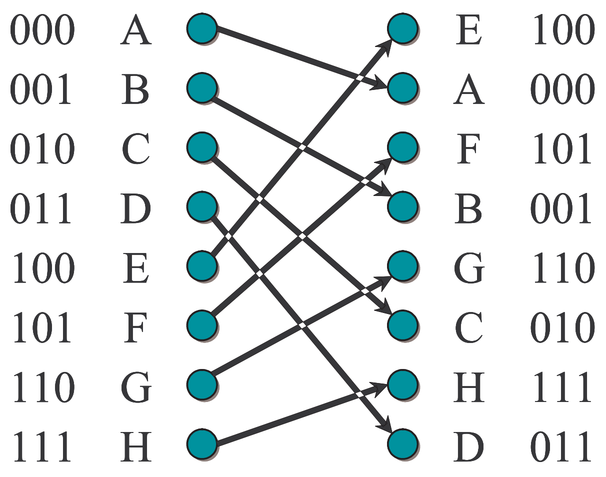


Figure 4: A perfect in-shuffle

4.5 A Card trick

Consider a deck of 8 cards which correspond to the 8 nodes shown earlier in the shuffle-exchange graph. Suppose a card is chosen at random and placed in any location in the deck, given a required destination location, we can perform a certain number of perfect shuffles(in or out) in order that we route the card to the required destination. Let us consider an example. Suppose that the randomly chosen card is placed at the last position (corresponding to 8-1 = 7 = 111), and that the required destination position is 3 (corresponding to 3-1 = 2= 010).

The claim is that 3 perfect shuffles is sufficient to place the last card in the third position. It can be seen from Figure 6 that starting with an in-shuffle, followed by an out-shuffle, and lastly performing an in-shuffle again would route the card at 111 to the position 010. Let us proceed to trace through the sequence of

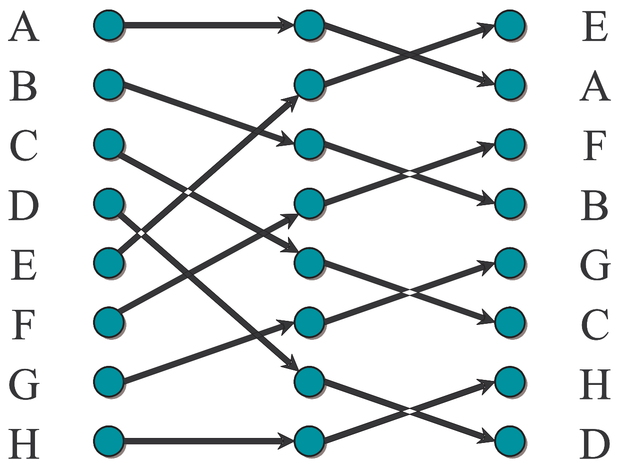


Figure 5: In-shuffle = Out-shuffle + Exchange

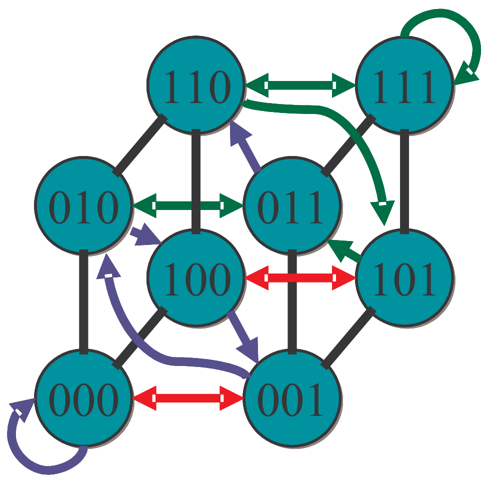


Figure 6: Required path in the shuffle-exchange graph (Shown in green)

operations. The first in-shuffle actually corresponds to a shuffle followed by an exchange. This means that from 111 we go back to 111 and then take the exchange edge to 110. The second operation which is an out-shuffle just corresponds a shuffle from 110 which takes us to 101. The last operation which is an in-shuffle takes us from 101 to 011 following the shuffle edge and then from 011 to the required position 010 following the exchange edge. This sequence is illustrated by the green edges in Figure 6.

In order to obtain the required sequence of operations, we first take the XOR of the source and the destination. The bits which are 1 in the resultant bit string correspond to the bits that have to be inverted by taking exchange edges at appropriate rotations. Hence, we perform an in-shuffle for each 1 in the bit string, and perform an out-shuffle for each 0 in the resultant bit-string.

5 DeBruijn Network

In this section, we look at the ***DeBruijn graph***. This is another network based on the Hypercube. In fact, we may convert the previous Shuffle Exchange graph to the DeBruijn graph. Consider a 3-bit node as an instance. We use the first 2 bits as the prefix, and then combine all nodes with the same prefix. For instance, node $< 1,1,0 >$ and $< 1,1,1 >$ will merge into one node $< 1,1 >$, and so on. We collapse the Shuffle-exchange network along the exchange edges to obtain the DeBruijn Network (as shown in Figure 7).

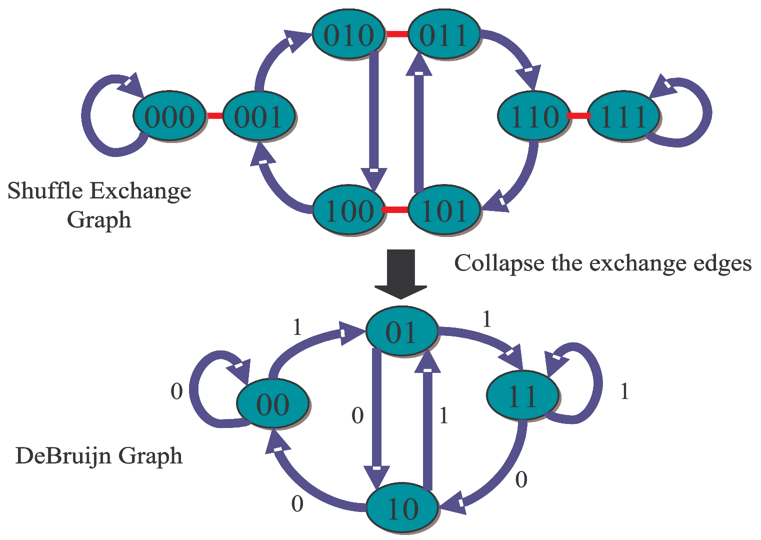


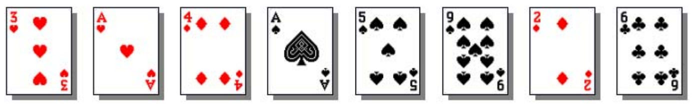
Figure 7: Converting the Shuffle-exchange graph to the DeBruijn graph

Node = d-bit binary number
Out-shuffle edge $< b_{d-1}, b_{d-2}, ...b_0 > \rightarrow < b_{d-2}, ...b_0, b_{d-1} >$
In-shuffle edge $< b_{d-1}, b_{d-2}, ...b_0 > \rightarrow < b_{d-2}, ...b_0, \overline{b_{d-1}} >$
The shuffle edges can also be represented as 0-edges or 1 edges based on whether the higher order bit, b_{d-1} is a 1 or a 0.
0-edge is $< b_{d-2}, ...b_0, 0 >$
1-edge is $< b_{d-2}, ...b_0, 1 >$

Degree = 4 (out-degree = in-degree = 2)
Number of edges $\leq$ 4N
Diameter = lgN
Bisection width = O(N / lgN).
This is because the DeBruijn graph is formed by collapsing the exchange edges of the Shuffle-exchange graph, and hence any upper bound of the latter would be valid for the DeBruijn graph as well.

5.1 Another Card Trick

Consider a deck of 8 cards. Suppose that the deck is cut 3 times, and then 3 cards are chosen consecutively. Now, if the number of black cards chosen is known, then we can find the 3 cards that were chosen ! This is because knowing the number of black cards provides us 3 bits of information, and it is only 3 bits of information that we need in order to get the position of a given card. According to the listing of the 3 card sequences in Figure 8, we find that any sequence of 3 cards corresponds to a different pattern of cards. Any kind of pattern only occurs once there. So 3 bits information is enough here to predicate the cards in hand. For example, if the pattern is 100, then it shows that the cards in hand are 6 clubs, 3 hearts and A hearts. By performing an Eulerian tour of the DeBruijn network in Figure 7, one of the DeBruijn sequences we get is shown in Figure 8.



```
0001110100
000
 001
  011
   111
    110
     101
      010
       100
```

Figure 8: Another Card trick (DeBruijn Sequence)

6 Deriving the Bisection Width for the Shuffle-Exchange Graph

The proof is based on the complex-plane diagram of the Shuffle Exchange network (as shown in Figure 9). The dot edges are the exchange edges and the solid edges correspond to the shuffle edges.
N = 2^d
Let w_d=$e^{2\pi i/d}$ be d-th complex root of unity. Hence, w_d^d=1

Definition 2 $\sigma(b_{d-1}, b_{d-2}, ...1) = \sigma(b_{d-1}, b_{d-2}, ...0) = \sum_{k=0}^{d-1} b_k w_d^k$

A few Lemmas are described below. They eventually lead us to the proof. We find that if we go along the solid edges, the node is always doubled under the constraint mod by 31. Moreover, if we go along the dot edges, the node increases by 1.

Lemma 3 *Exchange edges have unit length.*

Proof This is because in nodes connected by exchange edges, the least significant bits are complemented with respect to each other.

$$\sigma(b_{d-1}, b_{d-2}, ...1) = \sigma(b_{d-1}, b_{d-2}, ...0) + 1w_d^0 \tag{1}$$

□

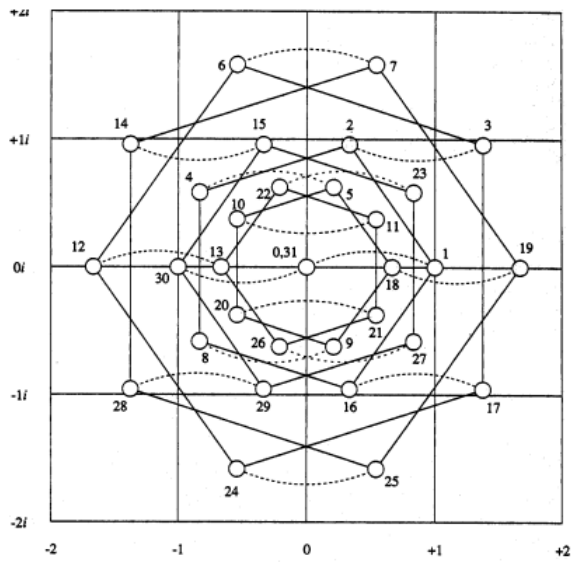


Figure 9: Complex plane diagram of the Shuffle-exchange network

Lemma 4 *Shuffle edges form* necklaces*, all of whose nodes are equidistant from origin:*

Proof

$$w_d \sigma(b_{d-1}, b_{d-2}, ...b_0) = w_d \sum_{k=0}^{d-1} b_k w_d^k \tag{2}$$

$$= \sum_{k=0}^{d-1} b_k w_d^{k+1} \tag{3}$$

$$= \sigma(b_{d-2}, ...b_0, b_{d-1}), since\ w_d^d = 1 \tag{4}$$

Thus, $||\sigma(b_{d-1}, b_{d-2}, ...b_0)|| = ||\sigma(b_{d-2}, ...b_0, b_{d-1})||$.

Most nodes are in *full* necklaces, having d nodes, but some are in *degenerate* necklaces, having lesser number of nodes.

For example:
For N=16 : {0000}, {1111}, {0101,1010} are degenerate necklaces.

□

Lemma 5 *All degenerate necklaces are mapped to origin.*

Proof Suppose: $w_d^k \sigma(x) = \sigma(x)$ with k<d
Each multiplication by w corresponds to a shuffling which in turn rotates the necklace (1/d)th each time.
Then: $w_d^k \neq 1 \Rightarrow \sigma(x)$=0

□

Lemma 6 *O(N/lgN) nodes are mapped to the Origin.*

Proof

$$\sigma(b_{d-1}, b_{d-2}...b_0) = 0 \Rightarrow \sigma(b_{d-1}, b_{d-2}...\overline{b_0}) = \pm 1 \tag{5}$$

Necklace of $< b_{d-1}, b_{d-2}, ..., \overline{b_0} >$ has lgN nodes, at most two of which lie at ± 1

$\therefore \leq$ N/lgN full necklaces $\Rightarrow$ 2N/lgN nodes at $\pm 1 \Rightarrow$ 2N/lgN nodes at 0

□

Lemma 7 *O(N/lgN) nodes mapped to real line*

Proof $\leq$ 2 nodes from each full necklaces + O(N/lgN) of degenerate necklaces at origin.

□

Lemma 8 *O(N/lgN) edges touch real line.*

Proof $\leq$ 4 shuffle edges/full necklaces $\equiv$ O(N/lgN)
$\Rightarrow O(N/lgN) + O(N/lgN)$ exchange edges(horizontal)+ O(N/lgN) shuffle-edges at the origin. □

Lemma 9 *Real line forms bisection*

Proof

$$\sigma(b_{d-1}, b_{d-2}...b_0) + \sigma(\overline{b_{d-1}}, \overline{b_{d-2}}...\overline{b_0}) \tag{6}$$

$$= \sum_{k=0}^{d-1} (b_k + \overline{b_k}) w_d^k \tag{7}$$

$$= \sum_{k=0}^{d-1} w_d^k \tag{8}$$

$$= \frac{w_d^d - 1}{w_d - 1} \tag{9}$$

$$= 0,\ \textit{since}\ w_d^d = 1 \tag{10}$$

Thus, number of nodes above real line = number of nodes below real line. The nodes on the real line are divided in half arbitrarily.

□

References

[1] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays - Trees - Hypercubes.* Morgan Kaufmann, 1992.