Lectures 8/9

1 Overview

We will give a couple of networks that can be used for routing messages. We will illustrate their use in implementing a BSP computer that was described in the previous lecture.

2 Prelude:Routing on the Grid

We illustrate routing and scheduling strategies (which we will be studying over the next few lectures) by showing how to route a permutation (a packet starts and is destined for each node) on a $k \times k$ grid (i.e., implementing the BSP routing for $h = 1$). We route each message first along its starting row to its destination column and then down its destination column. We assume edges are bidirectional, i.e., that a message can traverse left to right at the same time as a message traverse right to left. Furthermore, a node can send out a message out of all its links in any one step. Finally, the queue sizes are infinite; this ensures that a packet only waits if another packet uses the edge that it is waiting for.

Note that at most k messages cross any edge; only k messages can use an row edge as only k messsages start in a row, and only k messages can use any column as k messages are destined for any column.

Now, how long does it take to route? If one always sends a message along a link if one is waiting (in class we sketched a switch design that implements this property); one can only wait k steps at any link, and there are at most $2k$ links in any path, thus we finish in $2k^2$ steps. This is terrible though, as the lower bounds are the maximum congestion and the maximum length or $2k$.

We can achieve $O(k)$ by using the "straight-first" routing strategy. That is for each outgoing row (column) edge, we give priority to the message going along the single incoming row (column) we forward it; delaying any packet waiting to turn into the column.

Thus, we only wait at the turn. Since the congestion is k, we can only wait k steps. Thus the total routing time with this row-column routing and straight first scheduling is at most $3k$. This is $O(\sqrt{N})$ for an N-node mesh.

3 A couple of networks.

Today, we will talk about a particular parallel network; the butterfly and its brother the hypercube. The hypercube in n dimensions consists of a set of nodes labeled with the binary n-bit strings, thus it has 2^n nodes. Each node is connected to any node whose label differs in exactly one bit. (Would you advise drawing your own figure here.)

To route a message from node $a = a_1, \ldots, a_n$ to $b = b_1, \ldots, b_n$ proceeds by "correcting" the bits in some order, say from bit 1 to bit n. Of course, there are many (shortest) paths depending on the order of bit corrections, and of course many many non shortest paths.

A butterfly is the network that embodies fixing the bits in the most significant bit order in the hypercube. Here, the nodes are labeled with a pair (i, x) where i is an integer in $\{0, \ldots n\}$ and x is a n bit string. One can view the first index as the "row" and the second as the "column". The edges basically form a line on the row, i.e., (i, x) is connected to $(i+1, x)$. We also have "hypercube" edges. That is $(i, x_1, \ldots x_n)$ is connected to $(i+1, x_1, \ldots \overline{x_i} \ldots x_n)$. That is, at the ith place you can flip the i bit of your row. (Here a figure would be drawn in class.)

Typically, for the butterfly network, the first node in a row is viewed as the input and the last node is viewed as an output. One can also collapse the rows, and obtain the hypercube. Routing from an input $(0, x)$ to output (n, y) by correcting bits in order corresponds to the canonical shortest path routing algorithm above.

4 What is totally excellent?

Are these good routing networks?

- Small degree. Easy to build. The complete graph is very good in many senses, but has very high complexity. The hypercube has $\log N$ degree where the degree grows moderately with the size of the network. The butterfly has degree four which has the property of being able to be made from the same kind of switch, regardless of the size of the network.

- It has short paths. Any graph with degree at most k, has diameter at least $\log_k N$. The hypercube diameter is close to optimal with respect to degree, i.e., $\log N$ versus $\log_{\log n} N = \log N / \log \log N$. (De Bruijn networks have optimal diameter.) The butterfly has very near optimal diameter, $O(\log N)$.

A binary tree has both properties. What else? Perhaps, you want to have more properties. Hard to cut? No, the binary tree is easy to cut. When you route many messages one would like not to congest any link too much. Is this true? Not for the binary tree, routing messages across the root is very bad.

How about the butterfly. (For now, we leave the hypercube.) Hard to cut? We will address this later.

First, we restrict the routings a bit. Say a single message starts at each input and is destined for each output. That is, no output is the destination of more than one message, and no input is the destination of more than one message.

Let's consider the following pattern? Route a message from each input $(1, x\{0,1\}y)$ to $(n, y\{0,1\}x)$ where x is the first half (actually $(n-1)/2$) of the bits and y is the second (actually $(n-1)/2$) of the bits. This pattern is one to one as required.

THEOREM 1
The transpose pattern gives congestion $\Omega(\sqrt{N})$ on an N-input butterfly, using "greedy routing."

This theorem follows by cutting the butterfly at level $n/2$. This leaves $2^{(n-1)/2}$ separate $2^{(n-1)/2}$ input butterflies on the left and on the right, each of size $2^{(n-1)}/2$. The ith sub butterfly on the left have a common $(n-1)/2$ bit prefix representing i for the row identifier, while the jth sub butterfly on the right has a common suffix j. There is a single edge from the ith sub butterfly on the left to the jth sub butterfly on the right. But, all the messages from the ith sub butterfly on the left are destined for the jth sub butterfly on the right? (Note actually there is a $i, 0$ sub butterfly on the left and a $i, 1$ on the left, and similarly (0,j;1,j...) for the right.)

This causes a congestion of $2^{(n-1)/2}$ on a single edge. Yet, most edges in the middle layer are empty? Can we do better?

5 Benes Routing

A Benes network is a butterfly network connected to a reversed butterfly network. Or one, can view it as a network that goes up the butterfly and back down. This network gives many natural shortest paths from input, one for each of N intermediate outputs.

This choice allows us to overcome the problem from the last section.

THEOREM 2
Any (partial) permutation can be routed on a Benes network, with no congestion.

PROOF:

Notice the following recursive nature of the Benes network. If one removes all the inputs and outputs, that one obtains two separate Benes networks. We note that for a trivial Benes network (i.e., one node) it is easy to route a permutation. Do nothing!!

So, now we reduce the problem of routing a permutation on an N input Benes network to one of routing on a permutation on a $N/2$ input Benes network. What to do?

Well, at the "first" step, we must choose to route message at input i (and destined for say j) to the "top" sub-Benes or to the bottom. This decision then forces us to route the packet to a particular output of the sub-Benes based on its destination j.

We wish to coordinate the decisions so that

- Each input of a sub-Benes is chosen by a single input of the original.
- Each output of a sub-Benes is forced to be used by a single message.

Either is easy to enforce. For example, only two inputs of the original can choose any particular input of a sub-Benes and there are two sub-Benes networks. Same for outputs. But the choice at the inputs *forces* the choice at the outputs.

We encode these dependencies in a bipartite graph each side of which contains $N/2$ nodes. For each message, which starts at $i = i_1 \dots i_n$ and is destined for $j = j_1 \dots j_n$, we add an edge from node $i_2 \dots i_n$ to node $j_1 \dots j_{n-1}$. This encodes the fact, that either original input $0i_2 \dots i_n$ or $1i_2 \dots i_n$ can choose input $i_2 \dots i_n$ of a sub-Benes network. A similar statement hold for the outputs.

The degree of this graph is at most two. If we can partition the edges (messages) into two sets each of which induces degree at most 1 in the graph, we can assign each of the two

sets of messages to the two sub-Benes networks in a manner where the induced message source, destinations is a permutation.

Clearly, since the degree of the graph is two, the graph consists of a set of cycles. Moreover, the length of the cycles is even since the graph is bipartite. Thus, we can partition the cycles into two sets to meet this degree condition.

Note that in class, I did a different proof where we formed a "conflict" graph among messages and showed that it could be two colored. (It is essentially the same argument except that the edges in the conflict graph are the nodes in the graph in this proof.)

□

Question 1: Prove that separating any k of the inputs from any k outputs in the Benes network requires one to cut at least k edges. (That is, it is much "harder to cut" than a binary tree, or for that matter planar graphs. It is a two line argument that follows from the theorem above.)

Question 2: Show that for permutation routing on a N-node mesh ($\sqrt{N} \times \sqrt{N}$ "grid" of nodes), that there exists is a row-column-row routing where only one message turns at any one point. That is, in each row exactly one packet turns into any column from that row, and in each column exactly one packet turns into any row from that column. (Hint. Consider a bipartite graph whose nodes correspond to starting rows and destination rows, put an edge between two nodes r and r' corresponding to a message that starts in row r and ends in row r'. What is the degree of this graph? For any matching of edges (messages), show that they can be routed in a single column. Can the graph be decomposed into matchings? How many?) This has applications in VLSI routing where messages correspond to wires and turns correspond to physical connections. One wishes to limit such connections at each junction.

5.1 Some comments.

This along with much of modern engineering science was invented at Bell Labs in the fifties. It is the basis of routing in the telephone network switches called Clos networks (which are essentially high degree butterflies.)

This particular observation was the basis of the parallel computer built by IBM in the eighties to do quantum chromodynamics computations. This computation proceeded as did the relaxation algorithms of the previous lecture. The pattern of communication in that lecture as well as in QCD computations is fixed in every step. Thus, one could implement the BSP model by precomputing the switch settings in the network.

For example, with very fast switches (say as fast as the processors) and p computers, this network essentially implemented the BSP model with g being one. And L being $\log p$.

One can also view the old-style CRAY supercomputers in this light. They had tremendous processor to memory networks. We will look at this more later.

6 On-line solutions: Valiant/Brebner routing.

Benes routing in some sense says that the hypercube/butterfly are great networks. On the other hand, the algorithm is not very "parallel", i.e., for each pattern of messages one needs to figure out how to route them.

Here we consider a really parallel algorithm for routing permutations in the Benes network. Route to a random intermediate destination in the first butterfly. Then route greedily to the final destination. This method was proposed by Valiant and Brebner for the hypercube. (Valiant has done many truly amazing things. This being merely a comment of wide reaching consequences.)

THEOREM 3
The maximum number of packets that use any edge is $O(\log n)$ with high probability.

What is the expected number of messages that use any edge? Consider the ith level in the first butterfly. The number of inputs that could use it are 2^i.

That is the load can be expressed as the sum of 2^i $0-1$ random variables. The number of outputs it can get to is 2^{n-i} (within a factor of 2). The probability of any of the inputs choosing this output is thus $2^{n-i}/2^n = 2^{-i}$ Thus, the expected number of messages is 1.

7 On-line solutions: Valiant/Brebner routing.

Benes routing in some sense says that the hypercube/butterfly are great networks. On the other hand, the algorithm is not very "parallel", i.e., for each pattern of messages one needs to figure out how to route them.

Here we consider a truly locally controlled and thus parallel algorithm for routing permutations in the Benes network. Route to a random intermediate destination in the first butterfly. Then route greedily to the final destination. This method was proposed by Valiant and Brebner for the hypercube.

By the way, one can view this equivalently, as flipping a coin at each level in the butterfly and deciding whether to route up or down.

Before analysing this method, let's get some basics. What is the highest load on any intermediate destinations? This is just the question of what the maximum load on any of n bins when one throws n balls into them. With high probability, we know it to be $O(\log n/\log\log n)$.

(BTW, Valiant has done several truly amazing things in complexity, learning, algorithms, and parallel computing. This is simple but elegant example of his contributions.)

THEOREM 4
The maximum number of packets that use any edge is $O(\log n)$ with high probability.

What is the expected number of messages that use any edge? Consider the ith level in the first butterfly. The number of inputs that could use it are 2^i.

That is the load can be expressed as the sum of 2^i $0-1$ random variables. The number of outputs it can get to is 2^{n-i} (within a factor of 2). The probability of any of the inputs choosing this output is thus $2^{n-i}/2^n = 2^{-i}$ Thus, the expected number of messages is 1.

We finish by using a Chernoff bound. Here, we have random indicator variables $X_1, \ldots, X_k$, where $k = 2^i$ for a level i edge.

Here the expectation of $X = \sum_i X_i$ is 1 and we wish to show that max load is at most $O(\log n)$, thus we will choose $\delta = 2\log N$. We then use the form that states that

$$P[X > (1+\delta)\mu] \leq 2^{-\mu\delta}.$$

Plugging in, we get that the max load is at more $(2\log n+1)$ with probability than $1/N^2$. This allows us to conclude that any of the $N\log N$ edges is congested with probability at most $\log N/N$ using the union bound. Thus, the theorem follows.

Actually, we can show with high probability that max load is $O(\log n/\log\log n)$. The probability that the load on a level i edge exceeds congestion k is at most

$$\binom{K}{k}1/K^k \leq \left(\frac{e}{k}\right)^k.$$

This is less than $1/N^2$, when one chooses, for example, $k \geq 4\log N/\log\log N$.

Question 3: Show that permutation routing on a mesh where one chooses a random column and then routes the message along the starting row and turns into the random column and then proceeds to the destination row creates $O(\log N)$ congestion at each turn, with high probability. (This ensures that the queuesize needed for straight first routing is of size $O(\log N)$.)