CS273 Foundations of Parallel & Network Computing **Spring 2001**

Lecture 6: 1^{st} February, 2001

Lecturer: Satish B. Rao *Scribe: Ranjit Jhala*

In this lecture (and the next) we will look at some approaches to packet routing.

6.1 Last Lecture

Recall that any packet routing problem can be broken into two subproblems, *path selection*, which is selecting the path from the source to the destination that each packet will take and *scheduling*, which is determining which of several contending packets will be allowed to use an edge at a given time.

Path Selection

- Input: A (directed) network $G = (V, E)$, a set of messages (I/O) pairs *i.e.* pairs of input and output vertices of the graph.
- Output: A set of paths, one for each I/O pair, between the input and output vertices of that pair.
- Quality: The important parameters are Congestion $C = \max \sharp$ paths that use an edge, and Dilation $D = \max$ length of a path.

Scheduling
We saw that scheduling could be done *greedily* such that the packets can be routed in $O(CD)$ time.

Butterfly Network
In the Butterfly network of depth n with $N = 2^n$ nodes, we saw that the worst case permutation (each input is mapped to a unique output) routing can have congestion $C = \Omega(\sqrt{N})$.

However, a random mapping, where each input picks an output uniformly at random, has $C = O(\frac{\log N}{\log \log N})$ with high probability, and $D = \log N$, and so greedy routing can route the packets in time $\Theta(\frac{\log^2 N}{\log \log N})$.

Today
The topic of this lecture is the *Delayed Sequence Argument* using which Ranade [R87] showed that:

1. Routing may be done such that the time taken is $=$ PC $+ D$, where *Path Congestion* or $PC =$ max $\sharp$distinct packets that hit a path. Note that if a packet hits a path and leaves it and hits it again, we count it twice. Using this we will show the Butterfly can route a 1-mapping in $\Theta(\log N)$ time.
2. By randomly assigning priorities to packets, and then scheduling by priority, we can actually route in time $C + D + \Theta(\log N)$. Thus on the butterfly we can route $\log N$ random 1-mappings in $\Theta(\log N)$ time.
3. The above results assume that we have unbounded buffers and that a node can buffer all the packets that come to it. We will show that we can relax this condition to get fast routing even when the buffers are bounded. These results are from [LMRR94].

For the rest of the lecture we will assume that we wish to route on a directed acyclic network of depth D and indegree $d = 2$.

6.2 The Delay Sequence Argument

Definition 6.1 Delay Sequence *A Delay Sequence is a sequence of pairs of packets and edges* $((p_1, e_1), ..., (p_t, e_t))$*, such that:*

1. *Packet p_i uses the edge e_i.*
2. *If $p_i = p_{i+1}$ then $e_i = (x, y)$ and $e_{i+1} = (y, z)$ for some vertices x, y, z.*
3. *If $p_i \neq p_{i+1}$ then $e_i = e_{i+1}$.*
4. *All occurrences of a packet p are consecutive in the sequence.*

We will now bound the maximum length of a Delay Sequence.

Lemma 6.2 *The maximum length of a delay sequence is $PC + D$.*

Proof: Note that a delay sequence defines a path in the network. Everytime we go from pair (p_i, e_i) to (p_{i+1}, e_{i+1}), either the packet is the same and we have used up another edge of the path, or the packet changes and so we have used up another of the distinct packets that use this path. The restriction that the packets appear consecutively ensures that the number of packet changes we see is exactly the same as the PC. Thus the maximum length is equal to the maximum number of packets that use a path plus the length of the path, namely $PC + D$. ■

Now we will relate Delay Sequences to the time taken to route packets.

Lemma 6.3 *In a directed network, if a routing causes a packet p_{late} to arrive at its output at time t then there exists a delay sequence of length t.*

Proof: We will construct the delay sequence as follows: $p_t = p_{late}$ and $e_t =$ edge used by p_{late} at time t, basically at the tth timestep the packet p_{late} hopped across the edge e_t, note that it must have crossed some edge at that time instant. Now consider the time instant $i - 1$. We know that at the end of this instant, the packet p_i was ready to cross edge e_i. Thus either:

1. p_i arrived at e_i at the end of time $i - 1$,in which case $p_{i-1} = p_i$ and $e_{i-1} =$ edge on which p_{i-1} came across at time $i - 1$, or
2. p_i arrived at the edge e_i *earlier than* the end of instant $i - 1$ and was made to wait at that edge by some other packet p' that crossed the edge e_i at time $i - 1$ *instead of* p_i. Then set $p_{i-1} = p'$ and $e_{i-1} = e_i$.

Thus, as described above we can construct *backwards* a delay sequence of length t. All that remains is to show that each packet appears consecutively – intuitively, this is because once a packet is delayed by another at one point, it is never delayed by it again. Formally, in the constructed delay sequence consider for some $i < j < k$ the packets p_i, p_j, p_k. Suppose now that $p_j \neq p_k$, then we can show that $p_i \neq p_k$. We can use the case that $j = k - 1$ without loss of generality. In this case, by our construction it is clear that p_j was sent across the edge $e_j (= e_k)$ *before* p_k. We may generalize this to say that if $j < k$ then p_j is always "ahead" of p_k in the path corresponding to the delay sequence from the point when both of them are on the path. Suppose now that $p_i = p_k$, then as $i < j$ we have that at the point p_j hits the path, p_i is ahead of it - but then as $j < k$ it must be the case that p_j is ahead of p_k – but that we know is not possible as $p_k = p_i$ and thus is ahead of p_j ! Thus, it cannot be that $p_i = p_k$, and so the occurrence of a packet is indeed consecutive. ■

Lemma 6.4 *In a directed network of depth D, it is possible to route such that max delay is $PC + D$.*

Proof: We combine the above lemmas and argue that t must be at most $PC + D$ get the following result, as if it is greater we would not be able to construct a delay sequence as no delay sequence of greater length exists. Thus, the routing may be done such that the max delay is $PC + D$. ■

To show that it is possible to route on the butterfly in time $\Theta(\log N)$, we must show that the Path Congestion on the butterfly is small.

Lemma 6.5 *For a butterfly network of depth n, the Path Congestion for a random 1-mapping is $\Theta(\log N)$ where $N = 2^n$.*

Proof: We will use a weak Chernoff bound: Given $0 - 1$ random variables $X_1, ..., X_n$, and $X = \sum X_i$, if $E(X) = \mu$ then $\Pr[X > m] \leq \left(\frac{e\mu}{m}\right)^m$.

Now, for a particular path p, define $X_i = 1$ if message i uses p (o.w. $X_i = 0$). Now, recall that every path in the butterfly is of length n and so say $p = e_1, ..., e_n$.

Now we will partition the set of input ports into groups G_i where G_i is the set of input ports that hit path p *first* at edge e_i. It is easy to check that the $|g_i| = 2^{i-2}$. The probability that a port in group i actually hits this path is $2^{-(i-1)}$, as at each of $i - 1$ levels before this one, the packet would have had a probability of $1/2$ of taking the edge that lead to e_i. Note also that the number of these groups is $n = \log N$.

Now:

$$\begin{aligned}
&E(X) \\
&\quad = E(\sum X_i) \\
&\qquad \text{by the partitioning and linearity of expectation we get ...} \\
&\quad = \sum_{G_j} \sum_{i \in G_j} E(X_i) \\
&\quad = \sum_{G_j} (2^{i-2} \times 2^{-(i-1)}) \\
&\quad = \sum_{G_j} \frac{1}{2} \\
&\quad = \frac{\log N}{2}
\end{aligned}$$

Thus $\mu = \frac{\log N}{2}$. Now plugging this into the Chernoff bound we get:

$$\begin{aligned}
&\Pr[X > e \log N] \\
&\quad \leq \left(\frac{e \log N}{2e \log N}\right)^{e \log N} \\
&\quad \leq 2^{-e \log N} \\
&\quad \leq \frac{1}{N^e}
\end{aligned}$$

This is for a particular path. By using a union bound and the fact that there are at most N^2 paths, we may conclude that the probability that all paths have less than $e \log N$ packets is at most N^{e-2}.

Thus with very high probability every path p will have less than $e \log N$ packets hitting it. Thus whp, the path congestion is $\Theta(\log N)$. ∎

We put the above two lemmas together to get the following theorem.

Theorem 6.6 *In a butterfly of depth n it is possible to route a random 1-mapping in time* $\Theta(\log N)$.

We now move onto the second application of delay sequences – where we will assign priorities to packets and show an even more startling result.

6.3 Routing with Random Priorities

Algorithm

The approach is the following, we first preassign to each packet a priority, randomly – that is we randomly rank the packets. Say R is the rank function. Now, whenever there are two packets that would like to use an edge, we pick the one with the lower rank and send it across first. Thus it is the same as the greedy algorithm, but when there is contention, the packet with the lower rank is sent across.

We now analyze the performance of this algorithm.

Definition 6.7 Sorted Delay Sequence *A Delay Sequence sorted with respect to a rank function R is a delay sequence where for all i, we have $R(p_i) \leq R(p_{i+1})$.*

Whenever a packet p_{late} arrives at its output port at time t, we can construct a sorted delay sequence, in a manner identical to the construction detailed earlier.

Recall that when the delay sequence was being constructed, we "changed" packets only when another packet used the edge e_{i+1} before the packet p_{i+1} did. In that case, it must be that the rank of this other packet (which is p_i) must be better than that of p_{i+1}. Thus, the delay sequence constructed is sorted.

Given a delay sequence of length t, and a randomly chosen R, the probability that the delay sequence is sorted wrt R is $\leq 1/(t-D)!$. The number of distinct packets is $(t-D)$, and only one permutation out of $(t-D)!$ permutations is good *i.e.* results in a sorted delay sequence.

Theorem 6.8 *In a levelled network of indegree 2 and depth D, the priority algorithm delivers all the packets in time $O(C + D + \log N)$. Where $N = \sharp$packets.*

Proof:

- Step 1: Given a packet arriving at time t we can construct a sorted delay sequence of length t.
- Step 2: $\sharp$ Delay Sequences of length $t \leq (\sharp\text{outputs})(2^D)C^{t-D}\binom{t}{D}$. Here C = Congestion, the max number of packets using an edge. In the product, the first two terms fix the number of distinct paths through the network, the third term to the number of sequences of packets and the last to the D places on the sequence where we "move back an edge".

- Step 3: The probability that a given delay sequence of length t is sorted is $\leq 1/(t-D)!$. Thus

$$\Pr[\exists \text{ a sorted DS of length } t] \leq N(2^D)C^{t-D}\binom{t}{D}\frac{1}{(t-D)!}$$

It can be shown that plugging $t = 2eC+2D+D\log\frac{t}{D}+c\log N$ makes the probability very small $\leq N^{-c}$, by slightly tighter analysis it can be shown that whp, the time taken is bounded by $O(C+D+\log N)$.

■

Priority Routing on the Butterfly

We can show that on the Butterfly, it is possibly to route a random $\log N$-mapping in time $\Theta(\log N)$, by using the priority algorithm.

- Step 1: In the butterfly, $D = \log N$. It can be shown using Chernoff bounds, that when there is a random $\log N$-mapping, the congestion $C = \Theta(\log N)$.
- Step 2: Applying the theorem above directly yields the result.

This shows that in fact it is possible to simulate an $N\log N$ processor PRAM with a butterfly, with $\Theta(\log N)$ time per memory access.

We put $\log N$ processors at each of the input ports of the butterfly. Now there are N memory banks which are laid out such that the memory access patterns are random and so the entire step is exactly a random $\log N$-mapping, as each of the $\log N$ processors at an input node picks a memory bank (output node), and thus we have our PRAM simulated in $\Theta(\log N)$ time. This is an EREW simulation.

6.4 Routing with Bounded Buffers

Upto this point we assume that when some number of packets are waiting to use an edge, the edge has some sort of buffer in which all the messages are stored till they are sent across. Thus, our greedy algorithms assumed that the buffers were unbounded. We showed that a simple greedy scheme could route in $O(CD)$ time, however with bounded buffers, it can be shown that greedy takes time $\Omega(2^D)$. The reason for this is that non-intersecting paths may affect each other as shown in the figure below. The solid lines indicate two non-intersecting paths. However each of the buffers are full and until the top buffer has space the one below it cannot send a packet - and so stays full, so the buffer below it is choked and so on - and the buffer on the lower path stays blocket till the top path has finished pumping packets into the top buffer, thus packets on the lower path stay frozen till the top path finishes.

Let us say we have queues on the edges of size q. A simple extension of the earlier algorithm would be to rank the packets as before and directly apply the algorithm of the previous section. Thus when there are several packets waiting to use the edge, we take the one that has the best rank and let it cross, so long as the buffer of next edge for that packet is not full.

This runs into the problem that a packet with a good rank may be blocked because the buffer of its next edge is already is already filled by packets with a worse rank !

The way to get around this is to enforce sorted order on the sequence of packets, *i.e* we will enforce that the packets that go across an edge are in strictly increasing order. The problem we face is: how do we know

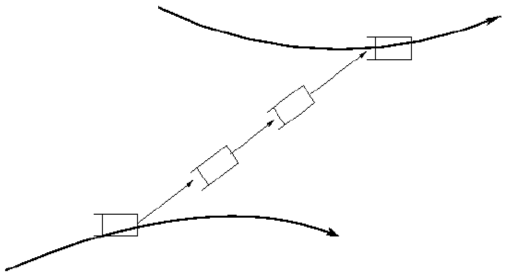

Figure 6.1: Delays due to non-intersecting paths

when to send a packet across the edge - given that we must ensure that once a packet with rank r_i has crossed it, all subsequent packets must have rank greater (worse) than r_i ?

We solve this by introducing *ghost* packets into the system. Each ghost packet has a rank. The purpose of the ghost packets is to provide the receiving node a lower bound on the ranks of packets that it will receive on that edge in the future. The figure illustrates the use of ghost packets: As the packet of rank 35 crosses the over the dotted edge from node A it sends a ghost across the other edge - thus node B knows that it can safely send the packet ranked 25 across. The algorithm is as follows:

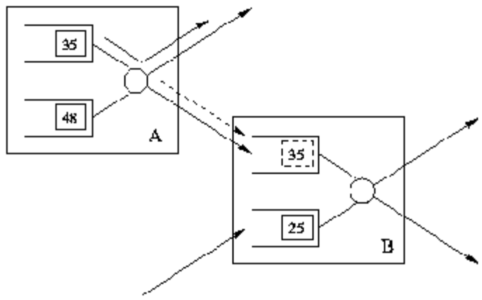

Figure 6.2: A ghost with rank 35 speeds up the delivery of the packet of rank 25

Algorithm:

1. Randomly rank all packets. At the start edges, buffer the packets by their ranks, and at the end of the buffer (the last element), put a ghost packet of rank ∞.

2. At each time step, a node looks at the edges that are incident on it and picks the lowest (best) ranked packet that wishes to use one of the edges that goes out from the node and sends that packet across it.

3. If there is no packet that needs to use one of the edges out of a node, then a *ghost* packet with the same rank across the other edge.

4. Note that in a buffer we can only have *one* ghost packet as the packet that immediately follows it will

have rank greater than the ghost packet's and thus provides a better lower bound and so the ghost packet can be removed from the buffer.

5. Note also that ghost packets are otherwise exactly like normal packets in that they may be selected for transmission if they are at the head of their respective queue and have rank smaller than than those of packets in the other queues – in fact we cannot let any of the other packets (with greater rank) use that edge as there may be a packet of smaller rank that will come across the edge that presently contains the low-ranked ghost packet.

6. Once a node sees a rank ∞ packet it will try to send rank ∞ ghost packets across both its out edges - to indicate to both that it will send no more packets subsequently.

The invariant we maintain is that all the packets (including ghost packets) that cross an edge are in strictly increasing order of rank.

Delay Sequences

Once again, our approach will be to say that whenever a packet p_{late} arrives at its output at time t, we can construct a delay sequence of length t. The delay sequence here is slightly different and is discussed below. We then argue that the probability of getting delay sequences of length greater than T is very small, and hence that with high probability all the packets get routed in less than T time.

To motivate the definition of delay sequences in this setting, we note that delays can be caused by the following:

1. There is a packet of better rank that wishes to use the same edge at that time instant. Note that this packet may also be a ghost packet; a packet will not cross an edge if there is a ghost packet of better rank in one of the other queues into the node.

2. The buffer for the next edge of the packet is full.

Definition 6.9 Bounded Queue Delay Sequence *A Bounded Queue Delay Sequence is a sequence of pairs of the form (p_i, e_i) and (p_i, e_i, S_i) such that:*

1. *If the i-th element is a pair, then the i-th and $i+1$-th tuples are related exactly as in Delay Sequences.*

2. *If the i-th element is a triple, then S_i is a q-sequence of packets sorted by rank such that $\forall p \in S_i.R(p) < R(p_i)$.*

Lemma 6.10 *In a directed network, if a routing causes a packet p_{late} to arrive at its output at time t then there exists a delay sequence of length t.*

Proof: The proof is similar to the earlier proof, and we construct the delay sequence backwards in time. The only difference comes from the second kind of delay - due to the buffer for the next edge of the packet p_i being full. In that case we note that as at the next timestep, the packet went across, it must have been the case that a packet left the full buffer in the present ($i-1$-th) timestep. Thus, we set p_{i-1} to be this packet and now follow its path backwards in time. Note that in this case, unlike going back an edge, we have actually moved *forward* an edge (to the edge who's buffer was full). ■

Theorem 6.11 *In a levelled network of indegree 2 and depth D, with buffers on the edges of constant size q, the above algorithm delivers all the packets in time $O(C + D + \log N)$. Where $N = \sharp packets$.*

Proof: Similar to the proof of the unbounded buffer setting.

- Step 1: Given a packet arriving at time t we can construct a sorted delay sequence of length t.
- Step 2: $\sharp$ Delay Sequences of length t where (as we walk back from the end of the delay sequence) there are f forward edges is at most $\leq (\sharp\text{outputs})(2^f)(2^{D+f})\binom{t}{f}\binom{t}{D+f}C^{t-D+qf}$. Here C = Congestion, the max number of packets using an edge. In the product, the first three terms fix the number of distinct paths through the network -note there are f forward edges and thus $D+f$ backward edges (as we walk back from the end of the delay sequence) and at each step we have two choices. The fourth term is choosing the places where we move forwards, the fifth term the places we move backwards and the final term the different sets of packets that a delay sequence could have, recall that with each forward edge is associated a full sorted buffer which corresponds to the qf in the exponent.

  Note that we use C to count the number of packets of interest. One may think that the ghost packets increase C but this is not so - rather the C that we use corresponds to the congestion due to real packets. Intuitively, this is because everytime we are held up by a ghost packet - we may go back along its path and really say that we are being held up by the real packet corresponding to a ghost packet. Formally, every time there is a ghost packet in the delay sequence, as we walk back along the edges to it, we will hit the real packet corresponding to the ghost packet, and till then we will only see the ghost packet. Another way to think of this is that as we read a delay sequence from start to finish, at each step we either hit a new packet (and are stuck at the same edge) or move across an edge with the same packet – with ghost packets, we are guaranteed to be moving across an edge everytime – and so we are really interested in counting the number of "changes" of packets at a particular edge which is what C is supposed to give an upper bound of. These changes come only from real packets and so the ghost packets do not affect this upper bound.
- Step 3: The probability that a given delay sequence of length t is sorted is $\leq 1/(t-D+qf)!$, as now the packets in the full buffers are also sorted and the entire set of $(t-D+qf)$ packets that we see is sorted. Thus as before we can bound the $\Pr[\exists$ a sorted DS of length $t]$ by taking the product of the above terms. It can be shown that when $t > O(C+D+\log N)$ the probability becomes very small $\leq N^{-c}$.

■

Combining Packets

The above line of reasoning - namely making packets *wait* till it is certain that no packet of smaller rank will pass by - might seem to be counter-intuitive (if not actually counter-productive !).

However, it is likely that it was motivated by the following application: Say we wish to emulate an $N \log N$ node CRCW system using the butterfly. We have seen already how to emulate EREW.

For CRCW, the setting is the same, namely we put $\log N$ processors at each of the N inputs of the butterfly. The memory is divided into N modules each of which sits at an output node. The memory locations are mapped randomly to the nodes so that the requests seem like random $\log N$-mappings.

The difference is that we can do the following optimization: Ensure that if several packets are for a particular location then they reach the module at the same time - by actually combining the requests into one request.

Thus the first node through which the two packets pass can actually wait for both to show up and then combine them and send them along. Thus we actually want packets to wait for all other "relevant" packets to come by. To do this, we rank the packets by ranking the memory locations (randomly) and then assigning to each packet the rank corresponding to its destination memory location. The rest of the algorithm is identical to the one above. At the first node where two packets destined for the same location meet - the one that reached first will wait for the other (as it must wait till all packets with better rank pass by) - and so it will wait till the packet with the same rank - and thus for the same address comes and at the node they will get combined. So with the above solution, we get fast routing as well as a reduction in the congestion as we combine all the packets as soon as we can and different packets to the same location do not occupy buffer space.

In the next few lectures we will look at how to do do packet routing on arbitrary networks - so long as the paths are edge-simple.

References

[R87] A.G. Ranade,"How to Emulate Shared Memory",*Proceedings of the 28th IEEE Foundations of Computer Science*, 1987 pp.185–194.

[LMRR94] F.T. Leighton and B. Maggs and A. Ranade and S.B. Rao, "Randomized Routing and Sorting on Fixed Connection Networks", *Journal of Algorithms*, Vol 17, Number 1, July 1994, pp.157–205.