4.2. Routing in a Parallel Computer

Our first application of the Chernoff bound is another case where a randomized algorithm yields a performance that is *provably superior* to any deterministic algorithm. This application concerns a communication problem in a network of parallel processors.

We model a network of parallel processors by a directed graph on N nodes, each of which is a processing element. Edges in the graph represent communication links between processing elements. All communication between processors proceeds in a sequence of synchronous *steps*. Each link can carry a unit message, or *packet*, in a step. During a step, a processor can send at most one packet to each of its neighbors. Each processor has a unique identifying number, between 1 and N.

We consider the *permutation routing problem* on such a network. Each processor initially contains one packet destined for some processor in the network. Let v_i denote the packet originating at processor i; we denote its destination by $d(i)$. We consider the case when the $d(i)$'s, for $1 \leq i \leq N$, form a permutation of $\{1, \ldots, N\}$, i.e., every processor is the destination of exactly one packet. How many steps are necessary and sufficient to route an arbitrary permutation request $d(1), \ldots, d(N)$? This special case is important in realizing abstract models of parallel computation (such as the PRAM model described in Chapter 12) by means of more feasible models.

A *route* for a packet is a sequence of edges it can follow from its source to its destination. An algorithm for the permutation routing problem must specify a route for each packet. In following a route, a packet may occasionally have to wait at an intermediate node because the next edge on its route is "busy" transmitting another packet. We assume that each node contains one queue for each edge leaving the node; the queue holds packets waiting to leave via that edge. A routing algorithm must also specify a *queueing discipline* for resolving conflicts between packets that simultaneously wish to follow the same edge out of a node.

We focus on a class of algorithms that are especially simple to implement in parallel computer hardware. An *oblivious algorithm* for the permutation routing problem satisfies the following property: the route followed by v_i depends on $d(i)$ alone, and not on $d(j)$ for any $j \neq i$. An oblivious algorithm specifies, for each pair $(i, d(i))$, a route between node i and node $d(i)$. Oblivious routing algorithms are attractive for their simplicity of implementation: the communication hardware at each node in the network can determine the next link on its route, simply by looking at the source and destination information carried by a packet. Often, the topology of the network makes this operation very simple. The communication hardware at a node does not have to compare the sources and destinations of different packets in its queues.

The following theorem gives a limit on the performance of deterministic oblivious algorithms; its proof is beyond the scope of this book (see the Notes section).

Theorem 4.4: *For any deterministic oblivious permutation routing algorithm on a network of N nodes each of out-degree d, there is an instance of permutation routing requiring $\Omega(\sqrt{N/d})$ steps.*

Consider the implications of this theorem for the case when the network is the *Boolean hypercube*, a popular network for parallel processing. The Boolean hypercube has $N = 2^n$ nodes connected in the following manner. Let $(i_0, \ldots, i_{n-1}) \in \{0,1\}^n$ be the (ordered) binary representation of i, i.e., $i = \sum_{j=0}^{n-1} i_j 2^j$. There is a link (a directed edge) from node i to node j if and only if $(i_0, \ldots, i_{n-1})$ and $(j_0, \ldots, j_{n-1})$ differ in exactly one position. Every node in the hypercube has $n = \log_2 N$ directed edges leaving it. Each edge incident on a node is associated with a distinct bit position in the node label, and traversing an edge corresponding to the position j will lead to a node whose label differs in exactly that bit position. Theorem 4.4 then tells us that for any deterministic oblivious routing algorithm on the hypercube, there is a permutation requiring $\Omega(\sqrt{N/n})$ steps.

We now establish a special case of the lower bound of Theorem 4.4 for the hypercube, showing that for a natural algorithm there is a natural permutation that results in poor performance. Given that the source and destination addresses are n-bit vectors, consider the following simple choice of route to send v_i from i to the node $\sigma(i)$: scan the bits of $\sigma(i)$ from left to right, and compare them with the address of the current location of v_i. Send v_i out of the current node along the edge corresponding to the left-most bit in which the current position and $\sigma(i)$ differ. Thus, in going from (1011) to (0000) in a 4-dimensional hypercube, the packet would pass through (0011) and then (0001) en route. This is referred to as the *bit-fixing* routing strategy for obvious reasons.

Exercise 4.2: Suppose that n is even. Consider the *transpose* permutation: writing i as the concatenation of two binary strings a_i and b_i each of length $n/2$, the destination of v_i is the concatenation of b_i and a_i. Show that the transpose permutation causes the bit-fixing strategy to take $\Omega(\sqrt{N/n})$ steps. Why is this permutation called a *transpose*?

We now study a randomized oblivious routing algorithm and show that its expected number of steps is considerably smaller than $\sqrt{N/n}$. This algorithm uses a simple two-phase scheme for permutation routing. Under this scheme, packet v_i executes the following two phases independently of all the other packets.

Phase 1: Pick a random intermediate destination $\sigma(i)$ from $\{1, \ldots, n\}$. Packet v_i travels to node $\sigma(i)$.

Phase 2: Packet v_i travels from $\sigma(i)$ on to its destination $d(i)$.

In each phase, each packet uses the bit-fixing strategy to determine its route.

Since each packet chooses its intermediate destination (in Phase 1) independently of the remaining packets, the scheme is oblivious. Because the $\sigma(i)$ are chosen independently at random, it may be that $\sigma(i) = \sigma(j)$ for $i \neq j$; thus σ is *not* a permutation. The choice of routes is now clear; it remains to specify the queueing discipline. For the above choice of routes, any of several queueing disciplines will in fact yield a result similar to Theorem 4.7 below. All that is required is that if at least one packet is ready to follow an edge e on a step, some packet follows e on that step. For concreteness, we adopt the following queueing discipline: each node maintains a queue for each outgoing edge, with packets leaving in FIFO (first in, first out) order. Ties occur only when two packets simultaneously arrive at a node and wish to leave by the same edge; these ties are broken arbitrarily. The reader should verify that any pair of packets may engage in such a tie at most once.

How many steps elapse before packet v_i reaches its destination? Let us first consider this question for Phase 1. Let ρ_i denote the route for v_i in Phase 1. The number of steps taken by v_i is equal to the length of ρ_i, which is at most n, plus the number of steps for which it is queued (delayed) at intermediate nodes in ρ_i. What is the delay encountered by packet v_i? To tackle this problem we require two additional facts; the first is a simple exercise.

Exercise 4.3: View each route in Phase 1 as a directed path in the hypercube from the source to the intermediate destination. Prove that once two routes separate, they do not rejoin.

We now establish an important step in the analysis. Like the statement in Exercise 4.3 above, it is a deterministic assertion that is independent of the randomization in our routing algorithm. In preparation for this step, the reader should first attempt the following exercise.

Exercise 4.4: Does the statement in Exercise 4.3 imply that for any two packets v_i and v_j, there is at most one queue q such that v_i and v_j are in the queue q at the same step?

Lemma 4.5: *Let the route of v_i follow the sequence of edges $\rho_i = (e_1, e_2, \ldots, e_k)$. Let S be the set of packets (other than v_i) whose routes pass through at least one of $\{e_1, e_2, \ldots, e_k\}$. Then, the delay incurred by v_i is at most $|S|$.*

PROOF: A packet in S is said to *leave* ρ_i at that time step at which it traverses an edge of ρ_i for the last time. If a packet is ready to follow edge e_j at time t, we define its *lag* at time t to be $t - j$. The lag of v_i is initially zero, and the delay incurred by v_i is its lag when it traverses e_k. We will show that each step at which the lag of v_i increases by one can be charged to a distinct member of S.

We argue that if the lag of v_i reaches $\ell + 1$, some packet in S leaves ρ_i with lag ℓ. When the lag of v_i increases from ℓ to $\ell + 1$, there must be at least one packet (from S) that wishes to traverse the same edge as v_i at that time step, since otherwise v_i would be permitted to traverse this edge and its lag would not increase. Thus, S contains at least one packet whose lag reaches the value ℓ.

Let t' be the last time step at which any packet in S has lag ℓ. Thus there is a packet v ready to follow edge $e_{j'}$ at t', such that $t' - j' = \ell$. We argue that some packet of S leaves ρ_i at t'; this establishes the lemma since by the result of Exercise 4.3, a packet that has left ρ_i will never again delay v_i.

Since v is ready to follow $e_{j'}$ at t', some packet ω (which may be v itself) in S follows $e_{j'}$ at t'. Now ω leaves ρ_i at t'; if not, some packet will follow $e_{j'+1}$ at step $t' + 1$ with lag still at ℓ, violating the maximality of t'. We charge to ω the increase in the lag of v_i from ℓ to $\ell + 1$; since ω leaves ρ_i, it will never be charged again. Thus, each member of S whose route intersects ρ_i is charged for at most one delay, establishing the lemma. □

Let the random variable $H_{ij} = 1$ if ρ_i and ρ_j share at least one edge, and 0 otherwise. It follows that the total delay incurred by v_i is at most $\sum_{j=1}^{N} H_{ij}$. Since the routes of the various packets are chosen independently at random, the H_{ij}'s are independent Poisson trials for $j \neq i$. Thus, to bound the delay of packet v_i from above using the Chernoff bound, it suffices to obtain an upper bound on $\sum_{j=1}^{N} H_{ij}$. To do this, we first bound $\mathbf{E}[\sum_{j=1}^{N} H_{ij}]$.

For an edge e in the hypercube, let the random variable $T(e)$ denote the number of routes that pass through e. Fix any route $\rho_i = (e_1, e_2, \ldots, e_k)$, with $k \leq n$. Then,

$$\sum_{j=1}^{N} H_{ij} \leq \sum_{l=1}^{k} T(e_l),$$

and therefore

$$\mathbf{E}[\sum_{j=1}^{N} H_{ij}] \leq \sum_{l=1}^{k} \mathbf{E}[T(e_l)]. \tag{4.13}$$

The following is an easy consequence of symmetry.

Exercise 4.5: Let e_l and e_m be any two edges in the hypercube. Prove that $\mathbf{E}[T(e_l)] = \mathbf{E}[T(e_m)]$. In other words, the expected number of routes passing through an edge is the same for all edges in the hypercube.

The expected length of ρ_j (number of edges traversed by v_j) is $n/2$ for all j, so that the expectation of the total route length summed over all the packets is $Nn/2$. The number of edges in the hypercube is Nn; by the result of Exercise 4.5, it follows that $\mathbf{E}[T(e)] = 1/2$ for all edges e. Using this in (4.13) gives

$$\mathbf{E}[\sum_{j=1}^{N} H_{ij}] \leq \frac{k}{2} \leq \frac{n}{2}.$$

By the Chernoff bound (the form in Exercise 4.1 is most convenient), the probability that $\sum_{j=1}^{N} H_{ij}$ exceeds $6n$ is less than 2^{-6n}. An important point: we apply the Chernoff bound to $\sum_{j=1}^{N} H_{ij}$ and *not* to $\sum_{l=1}^{k} T(e_l)$. We cannot apply the Chernoff bound to $\sum_{l=1}^{k} T(e_l)$ because the random variables $T(e_l)$ are *not* independent (and in fact are not Poisson trials). We use the quantity $\sum_{l=1}^{k} T(e_l)$ only to obtain an upper bound on $\mathbf{E}[\sum_{j=1}^{N} H_{ij}]$, and *then* apply the Chernoff bound to $\sum_{j=1}^{N} H_{ij}$, which is the sum of independent random variables.

Now $\sum_{j=1}^{N} H_{ij}$ is an upper bound on the delay incurred by v_i, so this delay exceeds $6n$ with probability less than 2^{-6n}. Since the total number of packets is $N = 2^n$, the probability that any of the N packets experiences a delay exceeding $6n$ is less than $2^n \times 2^{-6n} = 2^{-5n}$. Adding the length of the route to the delay gives $7n$ as the number of steps taken by v_i in Phase 1.

Theorem 4.6: *With probability at least $1 - 2^{-5n}$, every packet reaches its intermediate destination in Phase 1 in $7n$ or fewer steps.*

What happens to the packets in Phase 2? Observe that the routing scheme for Phase 2 can be viewed as the scheme for Phase 1 "run backwards." The same analysis then shows that with probability at least $1 - (1/32)^n$, every packet reaches its destination in $7n$ or fewer steps. The probability that any packet fails to reach its target in either phase is less than $2(1/32)^n$, which is less than $1/N$ for $n \geq 1$. Combining these facts, we have:

Theorem 4.7: *With probability at least $1 - (1/N)$, every packet reaches its destination in $14n$ or fewer steps.*

Note that we have bounded the delay of a packet in each phase by assuming it is delayed only by packets executing that phase. To avoid packets in Phase 1 delaying packets in Phase 2 and vice versa, rather than allow Phases 1 and 2 to proceed unchecked for the various packets, we make packets wait at their intermediate destinations until $7n$ steps have elapsed before beginning their Phase 2 travel.

An interesting feature of this scheme is that the distribution of the number of steps to completion is insensitive to the instance to be routed. Indeed, it is likely to take as long to route the identity permutation as any other "hard" permutation!

Exercise 4.6: Show that the expected number of steps within which all packets are delivered is less than $15n$.

Comparing the performance of the randomized algorithm with the negative result of Theorem 4.4, we find that our randomized oblivious algorithm is provably better in that it achieves an expected running time that no deterministic

oblivious algorithm can achieve. In fact, any deterministic oblivious algorithm must have performance *exponentially* worse than that of our randomized oblivious algorithm.

4.3. A Wiring Problem

We now consider another application of the Chernoff bound. The problem is that of *global wiring in gate-arrays.* A gate-array is a two-dimensional $\sqrt{n} \times \sqrt{n}$ array of gates abutting each other, arranged at regularly spaced points in the plane. The gates are numbered from 1 through n. A logic circuit is implemented on such an array by connecting together some of the gates using wires. A *net* is a set of gates to be connected by a wire. Wires run over the array in "Manhattan" form, i.e., they run parallel to the axes of orientation of the gate-array. In Figure 4.1, n is 9, and we have 4 wires each of which connects a pair of gates. Each gate is represented as a square with thin lines defining the boundaries. Each net connects a pair of gates, and has the same number marking its end-points (i.e., the thick lines 1-1, 2-2, 3-3, and 4-4). Note that in some cases a gate contains the end-point of more than one net.

The wiring problem is the following: we are given a set of *nets*, each of which is a set of gates to be connected together (to form one electrical connection). Here we consider only the simplest case, where each net consists of two gates to be joined by a wire. We wish to specify for each net a physical path between the two gates in the net, subject to space constraints.

In practice, the wiring problem is usually accomplished in two sequential phases: *global wiring* and *detailed wiring*. In the global wiring phase, we only specify which gates a wire will pass over in connecting its end-points. Thus, in Figure 4.1, the global route for net 4-4 passes through the three gates in the right-most column of the array. This is followed by the detailed wiring phase, in which the exact positions of the wires along their routes are specified – in our example, we would specify that the wire for net 4-4 lies to the right of the wire for net 3-3 as it leaves the top-right gate, and so on. Here we only concern ourselves with the global wiring phase.

The boundary between adjacent gates in an array has a fixed physical dimension and can therefore accommodate only a prescribed maximum number of wires, say w. We wish to find an assignment of global routes to all the nets in the wiring problem, such that no more than w nets pass through any boundary. In Figure 4.1, the set of routes we have indicated is a feasible solution provided w is at least 2. It is not hard to see that in this instance, we cannot find a feasible global wiring of the wires if w were only 1 – four wires must leave the top row of gates, and we have only three boundaries through which they must all pass.

We will solve a somewhat harder optimization problem instead of the feasibility problem – for a boundary b between two gates in the array, let $w_S(b)$ denote the number of wires that pass through b in a solution S to the global wiring problem. Let $w_S = \max_b w_S(b)$ be the maximum number of wires through