Tail Inequalities

Wafi AlBalawi and Ashraf Osman
Department of Computer Science and Electrical Engineering,
West Virginia University,
Morgantown, WV
{wafi,osman@csee.wvu.edu}

1 Routing in a Parallel Computer

A network of parallel processors is typically modeled by a directed graph $G = (N, E)$. The nodes N represent the processing elements and the edges E model the communication links between the processing elements. All communication between processors occurs in synchronous steps. Each link can carry at most one unit message (packet) in one step. During a step, a processor can send at most one packet to each of its neighbors. Each processor is uniquely identified by a number between 1 and N.

1.1 Permutation Routing Problem

The problem we address is known as the *permutation routing problem* .

Each processor initially contains a packet destined for some processor in the network. We denote the packet originating at node i destined for node $d(i)$ by v_i. Also, we assume that each node is the destination of exactly one packet i.e. the $d(i)$'s, for $1 \leq i \leq N$, form a permutation of $\{1, \ldots, N\}$.

The route followed by a packet v_i from its source i to destination $d(i)$ forms a sequence of communication links/edges. During its transmission, a packet may have to wait at an intermediate node because the node is "busy" transmitting another packet. We assume that each node contains a separate queue for each of its links and it follows FIFO (first in first out) queuing discipline to route its packets.

We focus on a class of algorithms that are especially simple to implement in parallel computer hardware called oblivious algorithms. An *oblivious algorithm* for the permutation routing routing problem satisfies the following property: the route followed by v_i depends on $d(i)$ alone, and not on $d(j)$ for any $j \neq i$. Oblivious routing algorithms are attractive for thier 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.

Theorem: 1.1 *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 $\Omega\left(\sqrt{\frac{N}{d}}\right)$ steps [1].*

We shall focus on *Boolean Hypercube* networks which is a popular network for parallel processing. The hypercube network contains $N = 2^n$ processing elements/nodes and is connected in the following manner. If $(i_0, \ldots, i_{n-1})$ and $(j_0, \ldots, j_{n-1})$ are the (ordered) binary representations of node i and node j respectively, then there exists a directed edge $e_{ij} \in E$ and a directed edge $e_{ji} \in E$ if and only if $(i_0, \ldots, i_{n-1})$ and $(j_0, \ldots, j_{n-1})$ differ in exactly one position. Note that the maximum number of transitions is bounded by $d = log\,(2^n) = n$

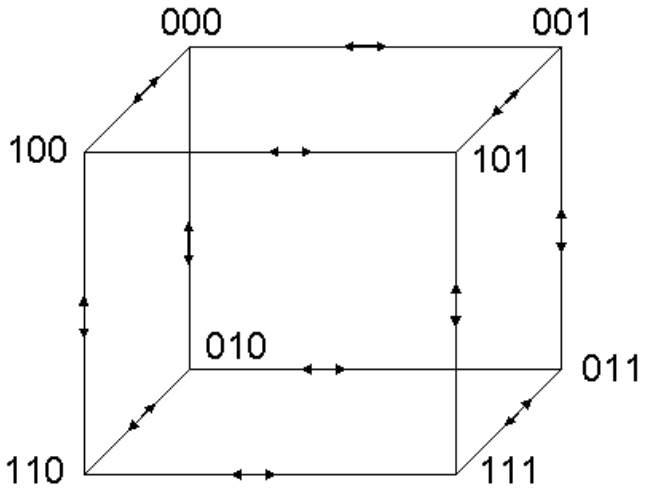

Figure 1: An example of a hypercube

1: 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 $\sigma(i)$:
2: Scan the bits of $\sigma(i)$ from left to right, and compare them with the address of the current location of v_i.
3: 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.

Algorithm 1.1: Bit Fixing Algorithm

1.2 Bit Fixing Strategy

We describe a simple routing strategy where each packet takes a route depending only on its source and destination nodes. We call this the *bit fixing* strategy as shown in algorithm 1.1.

The strategy is best explained through an example. In order to go from $(1,1,0,1)$ to $(0,0,0,0)$, the packet takes the following path:

$$(1,1,0,1) \to (0,1,0,1) \to (0,0,0,1) \to (0,0,0,0).$$

Note that the number of transitions required by the bit-fixing algorithm to route between arbitrary nodes of the hypercube is bounded from above by $n = \log(N)$.

1.3 Randomized Oblivious Routing

We now study a randomized oblivious routing algorithm and show that it takes considerably fewer number of steps (expected) than $\Omega(\sqrt{\frac{N}{n}})$. This algorithm is a two-phase process as shown below in algorithm 1.2.

1: Pick a random intermediate destination $\sigma(i)$ from $\{1, \ldots, N\}$.
2: Packet v_i travels to node $\sigma(i)$.
3: Packet v_i travels from $\sigma(i)$ on to its destination $d(i)$.

Algorithm 1.2: Randomized Oblivious Routing Algorithm

Both phases use the bit fixing algorithm to determine the routes. Since each packet chooses its intermediate destination independently of the remaining packets, the scheme is oblivious. Note that since $\sigma(i)$ come from independent distributions, it is possible that $\sigma(i) = \sigma(j)$ for $i \neq j$; thus σ, is *not* a permutation.

Lemma: 1.1 *Let the route of v_i follow the sequence $p_i = (e_1, \ldots, e_k)$. Let S be the set of packets (other than v_i) whose routes pass through at least one of $\{e_1, \ldots, e_k\}$. Then, the delay incurred by v_i is at most $|S|$.*

To prove this lemma, we need to prove the following points:

- If the bit fixing algorithm is used to route a packet v_i from i to $d(i)$ and v_j from j to $d(j)$, then their routes do not rejoin after they separate.
- The lag of packet v_i is at most $|S|$.

Proof: *Let k be the node at which the two paths separate and l be the node at which they rejoin. Note that the route determined by the bit fixing scheme for v_i and v_j from k to l depends only on the bit representations of k and l, therefore v_i and v_j must follow the same route. A contradiction to the assumption that k is the last node before the paths separate.*

Let $p_i = (e_1, \ldots, e_k)$ be the path followed by v_i. We define the lag of any packet v relative to p_i as $t - j$ if it is ready to follow edge e_j at time t. 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 v_i increases by one can be charged to a distinct member of S.

We shall charge the increase of lag of v_i from l to $l+1$ to a packet leaving p_i with a lag of l. There exists a packet with the lag of l preceding v_i in the queue else the lag of v_i would not have increased.

Let t' be the last time step at which any packet in S has a lag l. Note that there are only a finite number of time steps. There is a packet v demanding $e_{j'}$ at t' such that $t' - j' = l$. Since v demands $e_{j'}$, a packet ω actually travels on $e_{j'}$ at time t'. ω must leave p_i at time t' or else it would demand $e_{j'+1}$ at time $t'+1$ implying some packet must actually follow $e_{j'+1}$ at time $t'+1$ violating the maximality of l. Now, ω never returns to p_i. Therefore, we charge every increase in lag to a packet in S at most once. □

Now we carry out the analysis of the expected time to transfer all the packets. Define a random variable $H_{ij} = 1$ if p_i and p_j share at least one edge, and 0 otherwise. It follows that the total delay incurred by v_i is at most $\sum_{j=1}^{n} H_{ij}$. Therefore,

$$\text{Upper Bound} \leq \textstyle\sum_{j=1}^{n} H_{ij}.$$

Since, the routes of various packets are chosen independently at random, the H_{ij}'s are independent Poisson trials for $j \neq i$. Thus, to bound the dealy of packet v_i from above using the Chernoff bound, it is enough to obtain an upper bound on the $\sum_{j=1}^{n} H_{ij}$. To do this we first bound $E\left[\sum_{j=1}^{n} H_{ij}\right]$.

For an edge e of the hypercube, let the random variable $T(e)$ be the number of routes that pass through e. If $p_i = (e_1, \ldots, e_k)$, then

$$\sum_{j=1}^{n} H_{ij} \leq \sum_{i=1}^{k} T(e_i). \tag{1}$$

This follows from the fact that the total delay is bound by the total number of routes that pass through this route, and therefore

$$E\left[\sum_{j=1}^{n} H_{ij}\right] \leq E\left[\sum_{i=1}^{k} T(e_i)\right]$$

Since, all edges in the hypercube are symmetric, $E[T(e_l)] = E[T(e_m)]$ for any two edges e_l and e_m. The total number of edges in the *directed* graph is Nn counting n edges per node. The expected length of each route is $\frac{n}{2}$ (See Appendix §A.) and hence the expected length of the total route length summed over all packets is $\frac{Nn}{2}$. In other words,

$$\begin{aligned} E[T(e)] &= \frac{N\frac{n}{2}}{Nn} \\ &= \frac{1}{2} \end{aligned}$$

for all $e \in E$. Then, it follows from (1) that

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

Note that the routes of the packets are chosen independently at random. Therefore, for a fixed i and $j \neq i$, the H_{ij}'s are independent Poisson trials. We may then apply the Chernoff bound to obtain the following probabilistic bound on $\sum_{j=1}^{n} H_{ij}$ with $\mu = \frac{n}{2}$:

$$F^{+}(\mu, \delta) < 2^{-(1+\delta)\mu}$$

$$Pr\left[\sum_{j=1}^{n} H_{ij} > 6n\right] < 2^{-6n}.$$

It must be noted that the Chernoff bound may not be applied to $\sum_{l=1}^{k} T(e_l)$, since $T(e_l)$ are not independent variables (and in fact are not Possion trials). We use the quantity $\sum_{l=1}^{k} T(e_l)$ only to obtain an upper bound on $E\left[\sum_{j=1}^{n} H_{ij}\right]$, and then apply Chernoff bound to $\sum_{j=1}^{n} H_{ij}$, which is the sum of independent variables.
Since, the total number of packets is $N = 2^n$, the probability that any of them 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. It follows that:

Theorem: 1.2 *With probability at least $1 - 2^{-5n}$, the packet v_i reaches t_i in $7n$ or fewer steps.*

The Phase 2 of this scheme is identical to Phase 1, if the roles of the destination and source are interchanged. The above analysis is hence valid for the second phase also. Therefore, the probability that any packet fails to reach its target in either phase is less than 2×2^{-5n} which is less than $2^{-n} = \frac{1}{N}$. Combining these facts, we have:

Theorem: 1.3 *With a probability at least $1 - (\frac{1}{N})$, every packet reaches its destination in $14n$ or fewer steps.*

2 Wiring Problem

In this section, we consider an application of the Chernoff bound to the *global wiring* problem 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. The maximum number of wires crossing any gate boundary is minimized, under the following restrictions:

1. Manhattan routing — two cells can be connected if and only if they are adjacent horizontally or vertically; in other words, diagonal connections are not allowed.
2. Every wire connecting a pair of cells can have at most one 90-degree bend.

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. For instance, in Figure 2, 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. Note that in some cases a gate contains the end-point of more than one net.

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

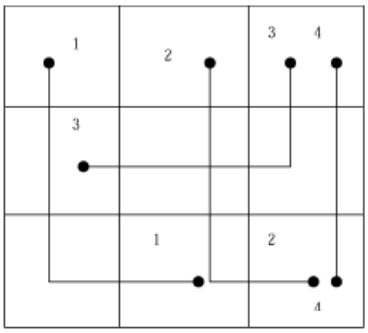

Figure 2: A $gate - array$ with 9 gates

Figure 2, 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 accomodate only a prescribed maximum number of wires, say ω. We want to find an assignment of global routes to all the nets in the wiring problem, such that no more than ω nets pass through any boundary. Let ω_S denote the maximum number of wires through any boundary b in solution S. Therefore, $\omega_S = max_b\ \omega_S(b)$. Our goal is to minimize ω_S over all solutions S. We can phrase the problem as an integer programming problem as follows:

2.1 Integer Programming

For each wire (net) i, define two indicator variables X_{i0} and X_{i1} as follows:

$$\begin{aligned} X_{i0} &= 1, \text{ and} \\ X_{i1} &= 0, \text{ if wire } i \text{ is routed first horizontally and then vertically} \end{aligned}$$

$$\begin{aligned} X_{i0} &= 0, \text{ and} \\ X_{i1} &= 1, \text{ if wire } i \text{ is routed first vertically and then horizontally} \end{aligned}$$

Recall that only one bend is allowed in a wire.

Let T_{b0} denote the set of wires i that cross the gate boundary b if i is routed horizontally first. Similarly, let T_{b1} denote the set of wires that cross b if the wires are routed vertically first. This is defined as follows:

$$T_{b0} = (i | i \text{ crosses } b, \text{ if } X_{i0} = 1)$$

$$T_{b1} = (i | i \text{ crosses } b, \text{ if } X_{i1} = 1)$$

Then, we formulate the integer programming of the problem as shown in algorithm 2.1.

It can be shown that the global wiring problem is `NP-complete`. Consequently, the integer programming formulation does not admit an efficient solution (unless `P=NP`). Our approach to the problem is to relax the integrality constraints using linear programming and solve the resultant linear program.

The objective is to minimize ω under the following conditions:

$$\begin{aligned} \forall i : X_{i0}, X_{i1} &\in \{0,1\} \\ \forall i : X_{i0} + X_{i1} &= 1 \\ \forall b : \sum_{i \in T_{b0}} X_{i0} + \sum_{i \in T_{b1}} X_{i1} &\leq \omega \end{aligned}$$

Algorithm 2.1: Integer Program for Wiring Problem

2.2 Linear Programming

This is done by relaxing the integrality constraints $X_{i0} \in \{0,1\}$ to $X_{i0} \in [0,1]$. Therefore,

$$\forall i : X_{i0}, X_{i1} \in \{0,1\}$$

is replaced with the pairs of inequalities

$$\forall i : 0 \leq X_{i0} \leq 1$$

$$\forall i : 0 \leq X_{i1} \leq 1$$

The other constraints remain the same. We then use the solution of the linear program to compute an *approximate solution* to the integer program. Let us denote the variables corresponding to the optimal solution of the linear program by $\hat{X}_{i0}$ and $\hat{X}_{i1}$. Note that the $\hat{X}_{i0}$'s and $\hat{X}_{i1}$'s may be fractional values and therefore are not directly usable. In order to use them, we need a rounding procedure that converts fractional values to $\{0,1\}$ values. In the next section, we shall study one such technique called *Randomized Rounding*.

2.3 Randomized Rounding

Randomized rounding technique is simply rounding up the real answer with a probability equal to the value of the real variable.Thus,

$$\Pr[X_{i0} = 1] = \hat{X}_{i0}$$

Similarly,

$$\Pr[X_{i1} = 1] = \hat{X}_{i1} = 1 - \hat{X}_{i0}$$

Let $\overline{X}_{i0}, \overline{X}_{i1}$ denote the corresponding rounded values. Then we express the randomized rounding algorithm as shown in algorithm 2.2.

1: Independently, for each i, set $\overline{X}_{i0}$ to 1 and $\overline{X}_{i1}$ to 0 with probability $\hat{X}_{i0}$
2: Otherwise, set X_{i0} to 0 and X_{i1} to 1.

Algorithm 2.2: Randomized Rounding Algorithm

While the above procedure maintains net feasibility, it could result in a solution that does not minimize ω. Let ω_0 denote the optimum solution of ω for the integer program, Theorem (2.1) will prove that the randomized rounding will obtain a close solution to the optimum solution with high probability.

Theorem: 2.1 *Let ϵ be a real number, $0 < \epsilon < 1$, then with probability $1 - \epsilon$, the global wiring S produced by randomized rounding satisfies*

$$\omega_s \leq \hat{\omega}(1 + \Delta^+(\hat{\omega}, \frac{\epsilon}{2n})) \leq \omega_0(1 + \Delta^+(\hat{\omega}, \frac{\epsilon}{2n}))$$

Proof: *We establish that following the rounding process, with probability at least $1-\epsilon$, no boundary in the array has more than $\hat{\omega}(1+\Delta^+(\hat{\omega},\frac{\epsilon}{2n}))$ wires passing through it. We will do so by showing that for any particular boundary b, the probability that $\omega_s(b) > \hat{\omega}(1+\Delta^+(\hat{\omega},\frac{\epsilon}{2n})$ is at most $\frac{\epsilon}{2n}$; then, since a $\sqrt{n}\times\sqrt{n}$ array contains fewer than $2n$ boundaries, we can sum this probability of failure over all the boundaries b to get an upper bound of ϵ on the failure probability.*
Consider a boundary b; since the solutions of the linear program satisfy its constraints, we have

$$\sum_{i\in T_{b0}} \hat{X}_{i0} + \sum_{i\in T_{b1}} \hat{X}_{i1} \leq \hat{\omega}. \tag{2}$$

The number of wires passing through b in the solution S is

$$\omega_S(b) = \sum_{i\in T_{b0}} \overline{X}_{i0} + \sum_{i\in T_{b1}} \overline{X}_{i1}. \tag{3}$$

But $\overline{X}_{i0}$ and $\overline{X}_{i1}$ are Poisson trials with probabilities $\hat{X}_{i0}$ and $\hat{X}_{i1}$, respectively. Further, $\overline{X}_{i0}$ and $\overline{X}_{i0}$ are each independent of $\overline{X}_{j0}$ and $\overline{X}_{j1}$ for $i\neq j$. Therefore, $\omega_S(b)$ is the sum of independent Poisson trials and, by (2) and (3),

$$\begin{aligned} E[\omega_S(b)] &= \sum_{i\in T_{b0}} E[\overline{X}_{i0}] + \sum_{i\in T_{b1}} E[\overline{X}_{i1}] \\ &= \sum_{i\in T_{b0}} \hat{X}_{i0} + \sum_{i\in T_{b1}} \hat{X}_{i1} \leq \hat{\omega} \end{aligned}$$

Now by applying Chernoff Bound, which states that for any positive μ and ϵ, a deviation of $\delta = \Delta^+(\mu,\epsilon)$ suffices to keep $Pr[X > (1+\delta)\mu]$ below ϵ. Therefore,

$$Pr[\omega_S(b) > \hat{\omega}(1+\Delta^+(\hat{\omega},\frac{\epsilon}{2n})] \leq \frac{\epsilon}{2n}$$

Since, for any particular boundary b, the probability that $\omega_s(b) > \hat{\omega}(1+\Delta^+(\hat{\omega},\frac{\epsilon}{2n})$ is at most $\frac{\epsilon}{2n}$; and there are fewer than $2n$ boundaries, we can sum this probability of failure over all the boundaries b to get an upper bound on the failure probability.

The probability of union of events is less than or equal to the summation of probabilities, then, the upper bound of failure is $\frac{\epsilon}{2n}\times 2n = \epsilon$. Therefore, with probability $1-\epsilon$, the global wiring S produced by randomized rounding satisfies

$$\omega_s \leq \hat{\omega}(1+\Delta^+(\hat{\omega},\frac{\epsilon}{2n})) \leq \omega_0(1+\Delta^+(\hat{\omega},\frac{\epsilon}{2n}))$$

□

A Expected Route Length of The Bit-Fixing Strategy

Given: A hypercube network which contains $N = 2^n$ processing elements. Required: To calculate the expected number of edges in a route using bit fixing strategy.

Each packet starts from a fixed node and could have as its destination any of the $N = 2^n$ nodes of the hypercube, chosen uniformly at random. The crucial observation is that the length of the route from a source node to its randomly chosen destination is equal to the number of bit flips in the vector representation of these two nodes!

Accordingly, if the number of bit changes is 1, then there will be n different routes to choose from (depending on the bit location). if the number of bit changes is 2, then there will be $\binom{n}{2}$ different routes to choose from and so on. This is illustrated in the following table .

Number of bit changes	Number of different routes	Number of edges traversed
0	0	0
1	n	1
2	$\binom{n}{2}$	2
3	$\binom{n}{3}$	3
...	...	...
n	$\binom{n}{n}$	n

Let X be a random variable that indicates the number of edges traversed in a route, i.e. the length of the route. $Pr[X = i] = \frac{\text{Number of routes with } i \text{ edges}}{\text{Number of all routes}}$

Observe that the total number of all routes originating from a given vertex is given by:

$$\sum_{i=0}^{n} \binom{n}{i}$$

where $\binom{n}{i}$, represente the number of routes of length i. Using binomial identities, we conclude that the total number of routes $= 2^n$.

Therefore, the expected number of edges traversed is given by:

$$\begin{aligned} E[X] &= \sum_{i=1}^{n} i \times Pr[X = i] \\ &= \frac{0.\binom{n}{0} + 1.\binom{n}{1} + 2.\binom{n}{2} + \ldots + n.\binom{n}{n}}{2^n} \\ &= \frac{n}{2} \end{aligned}$$

In the above proof, we used the binomial identity:

$$\sum_{i=0}^{n} i \times \binom{n}{i} = n.2^{n-1}$$

This follows from the expansion of the binomial as follows:

$$(a+b)^n = a^n + \binom{n}{1} a^{n-1}b + \binom{n}{2} a^{n-2}b^2 + \ldots + \binom{n}{n-1} ab^{n-1} + b^n$$

Differentiate both sides with respect to b,

$$n(a+b)^{n-1} = \binom{n}{1} a^{n-1} + 2\binom{n}{2} a^{n-2}b + \ldots + (n-1)\binom{n}{n-1} ab^{n-2} + nb^{n-1}$$

Substitute $a = 1$ and $b = 1$

$$n(1+1)^{n-1} = 1\binom{n}{1} + 2\binom{n}{2} + \ldots + (n-1)\binom{n}{n-1} + n$$

Therefore,

$$\sum_{i=0}^{n} i \times \binom{n}{i} = n.2^{n-1}$$

References

[1] Christos Kaklamanis, Danny Krizanc, and Thanasis Tsantilas. *Tight bounds for oblivious routing in the hypercube*. Mathematical Systems Theory, 24(4):223-232, 1991.