Load Balancing on the Hypercube and Shuffle-Exchange

C. Greg Plaxton*

Department of Computer Science

Stanford University

Stanford, CA 94305

Maintaining a balanced load is of fundamental importance on any parallel computer, since a strongly imbalanced load often leads to low processor utilization. This paper considers two load balancing operations: Balance and MultiBalance. The Balance operation corresponds to the token distribution problem considered by Peleg and Upfal [9] for certain expander networks. The MultiBalance operation balances several populations of distinct token types simultaneously. Efficient implementations of these operations will be given for the hypercube and shuffle-exchange, along with tight or near-tight lower bounds.

*This work was supported in part by a grant from the AT&T Foundation, NSF grant DCR-8351757 and ONR grant N00014-88-K-0166. The author is primarily supported by a 1967 Science and Engineering Scholarship from the Natural Sciences and Engineering Research Council of Canada.

1 Notation and Terminology

A p processor fixed interconnection network may be viewed as an undirected graph, where vertices correspond to processors and edges correspond to bidirectional communication channels. Each processor has an infinite local memory, and a unique integer ID. There is no global memory; processors communicate with one another by sending and receiving data over the channels provided by the network. In order to discuss the time complexity of an algorithm it is necessary to define exactly what operations can be performed in a single unit of time, or *time step*. For establishing asymptotic upper bounds, it is realistic to assume that:

1. Memory is configured in $O(\log p)$ bit words.
2. In a single time step, a processor can send and/or receive a single word of data and perform $O(1)$ CPU operations on word-sized operands.

The algorithms described in this paper are designed to run on the hypercube and shuffle-exchange network families. A dimension d hypercube has 2^d processors with IDs ranging from 0 to $2^d - 1$. Processor i is adjacent to processor j if and only if the binary representations of i and j differ in a single bit position. The shuffle-exchange was introduced by Stone [11]. Like the hypercube, a shuffle-exchange of dimension d has 2^d processors with IDs ranging from 0 to 2^{d-1}. Processor $i = (i_{d-1} \cdots i_0)_2$ is connected to processors $\mathit{Exchange}(i)$, $\mathit{Shuffle}(i)$ and $\mathit{Unshuffle}(i)$, where

$$\begin{aligned} \mathit{Exchange}(i) &= (i_{d-1} \cdots i_1 (i_0 \oplus 1))_2, \\ \mathit{Shuffle}(i) &= (i_{d-2} \cdots i_0 i_{d-1})_2, \text{ and} \\ \mathit{Unshuffle}(i) &= (i_0 i_{d-1} \cdots i_1)_2, \end{aligned}$$

$0 \leq i < d$.

Some important properties of the hypercube and shuffle-exchange network families are summarized in Table 1. Note that the degree of the shuffle-exchange is constant, while that of the hypercube is unbounded. Furthermore, the optimal VLSI layout area of the shuffle-exchange is somewhat smaller.

A more powerful model of the hypercube will also be considered, one which does not adhere to the 1-port restriction on communication imposed above. This is the *pipelined hypercube* model of Varman and Doshi [12]. The pipelined hypercube remains a realistic model of computation by providing only a very restrictive form of d-port communication.

Network	*Processors*	*Degree*	*Diameter*	*Layout Area*
hypercube	p	$\log p$	$\log p$	$\Theta(p^2)$
shuffle-exchange	p	3	$2 \log p$	$\Theta(p^2 / \log^2 p)$

Table 1: Important properties of the hypercube and shuffle-exchange.

Communication on the pipelined hypercube is via word-sized packets, routed according to the following simple scheme. Address bits are successively corrected in either ascending or descending (as determined by the sender) order of significance, with no collisions permitted. A collision occurs when two packets attempt to traverse the same edge in the same direction at the same time.

In routing a packet, one time step is expended for each bit in the smallest contiguous block of address bits that contains all of the bits to be corrected. For example, a packet sent from processor 10110101_2 to processor 10010011_2 must pass through dimensions 1, 2, and 5. Assuming that the sending processor elects to have address bits corrected in descending order of significance, the packet would be routed according to the schedule given by the following list of (time, processor) pairs: $(0, 10\mathbf{1}10101_2)$, $(1, 100\mathbf{1}0101_2)$, $(2, 1001\mathbf{0}101_2)$, $(3, 10010\mathbf{1}01_2)$, $(4, 100100\mathbf{0}1_2)$, $(5, 10010011_2)$. This packet is sent by processor 10110101_2, received and sent by processors 10010101_2 and 10010001, and received by 10110011_2. Let the first sender (10110101_2, in this example) be called the *originator* of the packet, and let the last receiver (10010011_2, in this example) be called the *acceptor* of the packet. The pipelined hypercube imposes the following pair of restrictions on communication:

1. Each processor is allowed to originate and/or accept at most one packet per time step.

2. Each edge can transmit at most one packet in each direction per time step.

The pipelined hypercube model is not realistic in a strict sense, since a single RAM cannot hope to examine $O(\log p)$ packets in $O(1)$ time. However, it may be a useful model in practice since the only additional hardware required at each hypercube processor is a ring of $O(\log p)$ trivial coprocessors to handle the packet routing scheme described above. Viewing this as an enhancement to the $O(\log p)$ I/O channel hardware already required by a hypercube processor, one would expect to suffer only a small constant factor increase in the VLSI area needed to implement a processor.

Several comments should be made with regard to mathematical notation. First, all logarithms are to be taken base 2, that is, $\log x$ denotes $\log_2 x$. Second, it will sometimes be convenient to make use of the function løg, defined as

$$\text{løg}\, x = \max\{\log x, 1\}.$$

Finally, expressions such as $[4, 8)$ will be used to denote a set of contiguous integers, in this case the set $\{4, 5, 6, 7\}$ (the interval from 4 to 8 with 4 included and 8 excluded).

2 Problem Definition: Balance

The first load balancing problem to be considered, Balance, is defined as follows. Let n tokens be distributed over p processors, with no more than m tokens assigned to any single processor, $\lceil n/p \rceil \leq m \leq n$. It will be assumed that $n = O(p^c)$ for some constant c in order that calculations involving token counts can be performed with a constant number of CPU operations. The problem is to redistribute the tokens so that each processor has either $\lfloor n/p \rfloor$ or $\lceil n/p \rceil$ tokens, that is, so that the load is distributed as evenly as possible. Peleg and Upfal have exhibited tight bounds for this operation on a certain class of expander networks [9]. In many applications, it is not necessary to balance the population of tokens exactly. If the difference between the maximum number of tokens at any processor and the minimum number of tokens at any processor is ξ, it will be said that the tokens have been *balanced with error* ξ. The corresponding operation will be referred to as Balance *with error* ξ.

2.1 Tight Bounds for the Pipelined Hypercube

This section describes an algorithm for the Balance operation with minimum error that runs in $O(m \log p)$ time on the hypercube or shuffle-exchange, and in $O(m + \log p)$ time on the pipelined hypercube. This algorithm is due to Tom Leighton [6]. Assuming that m exceeds $\lceil n/p \rceil$ by at least some constant factor, there is a trivial $\Omega(m + \log p)$ lower bound for the pipelined hypercube. The $\Omega(m)$ term in the lower bound holds since at least $m - \lceil n/p \rceil = \Omega(m)$ time steps are necessary for a processor initially holding m tokens to send away sufficiently many tokens to reach $\lceil n/p \rceil$, the maximum allowable number in any balanced configuration. The $\log p$ term in the lower bound holds because, as will be proven rigorously in Section 2.2, it is possible to configure the tokens in such a way that no token is placed within $\Omega(\log p)$ hops of a particular processor. Thus, Leighton's algorithm provides tight bounds for the pipelined hypercube when m exceeds $\lceil n/p \rceil$ by some constant factor. Note that if m does not exceed $\lceil n/p \rceil$ by a factor of 2 (say), then load balancing is probably not necessary anyway.

The 1-port version of Leighton's algorithm will now be described. The algorithm runs in m phases, and each phase takes care of one token from every processor for which the supply of tokens has not yet been exhausted. In a phase, the designated tokens are routed to a contiguous block (with respect to processor ID modulo p) of processors. Each token is routed in exactly one phase. The first block begins at processor 0 (say), and each subsequent block begins at the processor following the end of the previous block. In this manner, the population of tokens gets distributed as evenly as possible.

A single phase of Leighton's algorithm is implemented by performing a prefix sum over the designated tokens followed by a concentration route as defined by Nassimi and Sahni [8]. The prefix sum gives the offset of each token within the contiguous block of destination processors. The size of the block is broadcast so that all processors can compute the start of the next block. All of these operations can be performed in $O(\log p)$ time , so the over-all running time of Leighton's algorithm is $O(m \log p)$. Note that this time bound is valid for the shuffle-exchange as well as the hypercube.

As indicated above, Leighton's algorithm is best suited for the pipelined hypercube. The prefix sum and broadcast operations can be pipelined on the hypercube but the concentration routes cannot (provably). On the other hand, the pipelined hypercube allows concentration routes to be pipelined [12]. Hence, the Balance operation can be implemented to run in $O(m + \log p)$ time on the pipelined hypercube.

2.2 A Lower Bound for the Hypercube

A lower bound for the running time of Balance on the hypercube can be obtained by concentrating all of the tokens in a small number of processors and then bounding the time required to eliminate the excess tokens from this set of processors. In the following, recall that $d = \log p$ denotes the dimension of the given hypercube.

Definition 2.1 Let $\Gamma^r(i)$ denote the set of $\binom{d}{r}$ processors at Hamming distance r from processor i, $0 \le r \le d$.

Definition 2.2 Let $B(i, r)$ denote the *complete Hamming ball* of radius r centered at processor i. More formally, this is the set of processors given by $B(i, r) = \cup_{0 \le l \le r} \Gamma^l(i)$.

Note that $|B(i, r)| = \sum_{0 \le l \le r} \binom{d}{l}$. It will also be necessary to consider "incomplete" Hamming balls, that is, Hamming balls with only a partially filled outer layer.

Definition 2.3 Given a positive integer x, let r and y be the unique pair of nonnegative integers such that $x = y + \sum_{0 \le l \le r-1} \binom{d}{l}$, $1 \le y \le \binom{d}{r}$. A set of processors B is a *Hamming ball of size x and radius r* if there is some processor i and some set of processors S such that $B = B(i, r-1) \cup S$, $S \subseteq \Gamma^r(i)$ and $|S| = y$. Let $\mathcal{B}(x)$ denote the set of all Hamming balls of size x.

Definition 2.4 Let a graph G with vertex set $V(G)$ and edge set $E(G)$ be given. For every $U \subseteq V(G)$, the *fringe of U with respect to G*, $\mathcal{F}(G, U)$, is defined as the set

$$\{u \in U \mid (u, v) \in E(G) \text{ for some } v \in V(G) \setminus U\}.$$

Finally, let the function $f(G, x)$ be defined as

$$f(G, x) = \min_{\substack{U \subseteq V(G) \\ |U| = x}} |\mathcal{F}(G, U)|,$$

where G is a graph and x is an integer, $0 \le x \le |V(G)|$.

Lemma 2.1 The Balance operation requires $\Omega((n - \lceil n/m \rceil \lceil n/p \rceil)/f(G, \lceil n/m \rceil))$ time.

Proof: The n tokens can be concentrated in a set S of $\lceil n/m \rceil$ processors. Under a balanced load, S contains at most $\lceil n/m \rceil \lceil n/p \rceil$ tokens. Hence, at least $n - \lceil n/m \rceil \lceil n/p \rceil$ tokens must leave the set S. Only processors located at the fringe of S, those in $\mathcal{F}(G, S)$, can send tokens out of S, and these can only transmit one token per time step. Therefore, the running time of Balance must be at least $(n - \lceil n/m \rceil \lceil n/p \rceil)/(f(G, \lceil n/m \rceil))$. □

Having established Lemma 2.1, the desired lower bound can now be obtained by computing $f(H_d, \lceil n/m \rceil)$, where H_d denotes the undirected graph corresponding to a hypercube of dimension d. Intuitively, one might expect that the value of $f(H_d, x)$ is determined by a Hamming ball configuration. The correctness of this intuition is borne out by the following theorem due to Harper [4]. Note that Frankl and Füredi have given a simpler proof of the same result [2].

Theorem 2.1 For every integer x, $0 \leq x \leq p$, there exists a ball $B \in \mathcal{B}(x)$ such that $f(H_d, x) = |\mathcal{F}(H_d, B)|$. □

The following estimate of the "volume-to-surface" ratio of a Hamming ball is proven in Appendix A.

Lemma 2.2 For positive integers d and $r = r(d)$, $0 \leq r \leq d/2$, let $S = \binom{d}{r}$ and let $V = \sum_{0 \leq l \leq r} \binom{d}{l}$. Then $V = 2^{d(1-1/k)}$ implies that $V/S = \Theta(k^{1/2})$, $1 \leq k \leq d$. □

Theorem 2.2 The Balance operation requires $\Omega(k^{1/2}m)$ time on the hypercube if $m = \Theta(p^{1/k}(n/p))$ and $m \geq 2n/p$.

Proof: Note that $k \geq 1$ since $m \leq n$, and $k \leq \log p$ since $m \geq 2n/p$. Theorem 2.1 and Lemma 2.2 together imply that

$$f(H_d, p^{1-1/k}) = \Theta(k^{-1/2}p^{1-1/k}),$$

$1 \leq k \leq \log p$. Now consider the lower bound given by substituting this equation into the statement of Lemma 2.1. The numerator, $n - \lceil n/m \rceil \lceil n/p \rceil$ is at least $n - \lceil p/2 \rceil \lceil n/p \rceil = \Omega(n)$. The denominator, $f(G, \lceil n/m \rceil)$, reduces to $f(H_d, p^{1-1/k}) = \Theta(k^{-1/2}p^{1-1/k})$. Hence, Balance requires $\Omega(k^{1/2}(n/p)p^{1/k}) = \Omega(k^{1/2}m)$ time on the hypercube. □

2.3 Upper Bounds for the Hypercube

Now consider the task of obtaining an efficient implementation of the Balance operation on the hypercube. Let a subcube of dimension 1 be given, with a tokens at processor 0 and b tokens at processor 1. Further assume that each processor knows the number of tokens that it is holding, that is, the value a (b) is stored in the local memory of processor 0 (1). In this case, it is easy to see that the Balance operation can be performed over the given subcube in $|a - b|/2 + O(1)$ time. This observation motivates the following definition.

Definition 2.5 Let a set of n tokens be arbitrarily distributed over the processors of a p processor hypercube, $p \geq 2$. Let the Balance operation be applied to each of the $p/2$ subcubes of dimension 1 induced by the set of $p/2$ edges across dimension i. Then the hypercube has been *balanced across dimension* i.

Clearly the amount of time required to balance across dimension i is $\xi/2 + O(1)$, where ξ is the maximum discrepancy between the number of tokens at a given processor and its neighbor in dimension i.

Lemma 2.3 Let the low and high subcubes with respect to dimension i of a given hypercube be balanced with error ξ. Then after balancing across dimension i, the entire hypercube is balanced with error $\xi + 1$.

Proof: Initially, each processor in the low subcube contains a number of tokens in the range $[a, a+\xi]$ for some integer a. Similarly, each processor in the high subcube contains a number of tokens in the range $[b, b+\xi]$ for some integer b. Thus, after balancing across dimension i every processor contains a number of tokens in the range $\left[\left\lfloor\frac{a+b}{2}\right\rfloor, \left\lceil\frac{a+b+2\xi}{2}\right\rceil\right]$, and

$$\left\lceil\frac{a+b+2\xi}{2}\right\rceil = \left\lceil\frac{a+b}{2}\right\rceil + \xi \leq \left\lfloor\frac{a+b}{2}\right\rfloor + \xi + 1,$$

which completes the proof. □

By repeated application of Lemma 2.3, one finds that successively balancing across every dimension of the hypercube yields an implementation of Balance with error $\log p$. The running time of this algorithm is $O(m \log p)$, since no processor will ever contain more than m tokens and hence each balancing step requires at most $m/2 + O(1)$ time. Furthermore, in the important case $m = O(n/p)$, it is possible to distribute the tokens so that this performance is achieved to within a constant factor. In other words, the worst case running time of such a balancing algorithm is $\Theta(m \log p)$ when $m = O(n/p)$.

The following algorithm improves on this time bound by making a more careful decomposition of the hypercube. One additional definition is needed in order to present the algorithm.

Definition 2.6 Let the *discrepancy across dimension* i, denoted δ_i, represent the absolute value of the difference between the total number of tokens in the high and low subcubes with respect to dimension i.

The efficiency of the following recursive procedure for Balance with error $\log p$ relies upon the fact (shown below) that there is always some dimension with a small associated discrepancy.

Algorithm Balance

1. Each processor counts how many tokens it has and stores the result. This takes $O(m)$ time.

2. Let l denote the dimension of the subcube being balanced ($l = d$ initially). If $l = 0$, return.

3. Compute δ_i, $0 \leq i < l$. This involves performing l independent sums over the entire subcube. These sum operations can be pipelined to run in a total of $O(l)$ time [5].

4. Determine the dimension i^* with least associated discrepancy δ_{i^*}. This takes $O(l)$ time.

5. Recursively balance the high and low subcubes with respect to dimension i^*, using Steps 2 to 6.

6. Balance across dimension i^*, adjusting the token counts appropriately. The running time of this step is analyzed below.

In order to establish the correctness of the preceding algorithm, it is necessary to prove that the output hypercube is balanced with error $\log p$. This follows easily by induction using Lemma 2.3.

To analyze the time complexity, only the cost of Step 6 remains to be determined. When this step is executed, the $(l-1)$-dimensional high and low subcubes with respect to dimension i^* are each balanced with error $l-1$. Hence, there are integers a and b such that each processor in the high subcube contains a number of tokens that is in the range $[a, a+l-1]$, and each processor in the low subcube contains a number of tokens in the range $[b, b+l-1]$. Letting $\delta = \delta_{i^*}$ gives

$$2^{l-1}(b - (a + l - 1)) \leq \delta,$$

and so $(b + l - 1) - a \leq \delta 2^{1-l} + 2l - 2$. Similarly, $(a + l - 1) - b$ can be bounded by the same quantity. Therefore, the cost of the balancing step is $O(\delta 2^{-l} + l)$. It remains to bound the minimum discrepancy δ. In the following sequence of lemmas, let Δ denote the sum of the discrepancies in the given hypercube of dimension d, $\sum_{0 \leq i < d} \delta_i$.

Lemma 2.4 The value of Δ is maximized by packing all of the tokens into a Hamming ball B of size $\lceil n/m \rceil$.

Proof: Given an arbitrary distribution of the tokens, it will be shown that the corresponding value of Δ is at most that achieved by a particular Hamming ball configuration. First, transform the processor IDs of the given hypercube by toggling each bit that corresponds to a dimension for which there are more tokens in the low subcube than in the high subcube. Note that the transformed hypercube yields precisely the same value of Δ as the given hypercube. It has the additional property that for every dimension, the high subcube contains at least as many tokens as the low subcube. Let $w(i)$ denote the number of 1's in the d-bit processor ID i, and let $n(i)$ denote the number of tokens at processor i. Now observe that Δ may be expressed as

$$\begin{aligned} \Delta &= \sum_{0 \le i < p} w(i)n(i) - \sum_{0 \le i < p} (d - w(i))n(i) \\ &= 2 \sum_{0 \le i < p} w(i)n(i) - nd, \end{aligned}$$

where $p = 2^d$. Thus, maximizing Δ is equivalent to maximizing $\sum_{0 \le i < p} w(i)n(i)$. This new sum is clearly maximized by distributing tokens according to the following algorithm: While there are tokens left to distribute, put m tokens (or the number of tokens remaining, if less than m) into an empty processor with largest $w(i)$ in the set of empty processors. The result follows since this algorithm fills a Hamming ball centered at processor $2^d - 1$. $\square$

Lemma 2.5 Let an instance of Balance be given for which the tokens are packed into a Hamming ball of size $\lceil n/m \rceil$. Then $\Delta = O(md^{1/2}p)$. Furthermore, if $m \ge 2n/p$ and $\lceil n/m \rceil = p^{1-1/k}$, then $\Delta = \Theta(mdk^{-1/2}p^{1-1/k})$.

Proof: Assume the tokens are packed into a Hamming ball B of radius r. The total discrepancy Δ is bounded by m times the number of edges between B and $H_d \setminus B$. The number of such edges is maximized when $r = \lfloor d/2 \rfloor$, so $\Delta = O(md\binom{d}{\lfloor d/2 \rfloor}) = O(md^{1/2}p)$. For the sharper bound, Lemma 2.2 tells us that the cardinality of the fringe of B is $f(H_d, p^{1-1/k}) = \Theta(k^{-1/2}p^{1-1/k})$. The number of edges between B and $H_d \setminus B$ is $d - r = \Theta(d)$ (r is at most $\lfloor d/2 \rfloor$ since $m \ge 2n/p$) times the size of the fringe of B. Hence, $\Delta = \Theta(mdk^{-1/2}p^{1-1/k})$, as claimed. $\square$

This section will only make use of the $O(md^{1/2}p)$ bound of Lemma 2.5. The more detailed bound involving k will be needed in the next section in order to analyze a load balancing algorithm involving multiple token types. The following theorem is an immediate consequence of the preceding two lemmas.

Theorem 2.3 For any instance of Balance, the average discrepancy across a dimension, Δ/d, is $O(md^{-1/2}p)$. $\square$

Recall that the cost of the balancing step in algorithm Balance was shown above to be $O(\delta 2^{-l} + l)$, where $\delta = \delta_{i^*}$ is the minimum discrepancy. Now the minimum discrepancy is certainly no larger than the average discrepancy, so δ must be $O(ml^{-1/2}2^l)$ by Theorem 2.3. Hence, the cost of the balancing step is $O(ml^{-1/2} + l)$, and the total running time of algorithm Balance is $O(\sum_{1 \le l \le d} ml^{-1/2} + l) = O(m \log^{1/2} p + \log^2 p)$.

Of course, this algorithm performs balancing with error $\log p$. This should be good enough for most applications, but if a minimum error (0 if $p|n$, 1 otherwise) balancing is desired, some post-processing is needed. Note that the post-processing task can be viewed as an instance of Balance with $m = \log p$, which can be solved in $O(\log^2 p)$ time using the 1-port version of Leighton's algorithm described in Section 2.1.

Theorem 2.4 The Balance operation, with minimum error, runs in $O(m \log^{1/2} p + \log^2 p)$ time on the hypercube.

Proof: First apply algorithm Balance described and analyzed earlier in this section to balance the load with error $\log p$. This takes $O(m \log^{1/2} p + \log^2 p)$ time. Compute the minimum number of tokens, a, at any processor and broadcast this value to all processors. This takes $O(\log p)$ time. Every processor then sets aside a tokens, and the remaining tokens (of which there are at most $\log p$ at any single processor) are balanced using Leighton's algorithm in $O(\log^2 p)$ time. □

2.4 Load Balancing on the Shuffle-Exchange

As noted in Section 2.1, the 1-port version of Leighton's algorithm runs on the shuffle-exchange. Hence, the Balance operation, with minimum error, runs in $O(m \log p)$ time on the shuffle-exchange. Now consider the following lower bound.

Theorem 2.5 Assuming $m \geq 2n/p$, the Balance operation requires $\Omega(m \operatorname{l\o g}(n/m))$ time on the shuffle-exchange.

Proof: Using techniques due to Leighton [7], Cypher has proven that the p processors of a shuffle-exchange can be partitioned onto c chips in such a way that fewer than $p/2$ processors are assigned to any single chip, and the number of pins per chip is $O(p/(c \operatorname{l\o g}(p/c)))$ [1]. The pin count for a particular chip C is determined by the total number of edges joining a processor assigned to C to a processor assigned to some other chip. Letting SE_d denote the graph corresponding to the shuffle-exchange of dimension d, Cypher's bound implies (by an averaging argument) that for every integer x, $0 \leq x < p/2$, there exists an integer $g(x)$, $x \leq g(x) < p/2$, such that

$$f(SE_d, g(x)) = O(x / \operatorname{l\o g} x).$$

Now consider the lower bound for Balance obtained by packing the n tokens into a set S of $g(\lceil n/m \rceil)$ processors with $O(\lceil n/m \rceil / \operatorname{l\o g} \lceil n/m \rceil)$ neighbors. At least $n/2 = \Omega(n)$ tokens need to be moved to processors outside of the set S, and each edge leaving S can carry at most one token per time step. Hence, Balance requires $\Omega(m \operatorname{l\o g}(n/m))$ time on the shuffle-exchange if $m \geq 2n/p$. □

The upper and lower bounds are tight for $2n/p \leq m \leq n^{1-\epsilon}$, where ϵ denotes an arbitrarily small positive constant.

3 Problem Definition: MultiBalance

In this section, a slightly more complicated load balancing problem than Balance will be considered. Let n tokens be evenly distributed over p processors, that is, each processor contains either $\lfloor n/p \rfloor$ or $\lceil n/p \rceil$ tokens. Each token has an associated *type*. There are $g \geq 2$ different types of tokens, and nothing is known about the distribution or proportion of the tokens of any particular type. The set of tokens of a particular type will be called a *group*, and it will be assumed that the g group types are given by the integers $\{0, \ldots, g-1\}$. The problem is to execute g Balance (with error ξ) operations, one over each group of tokens. This operation will be referred to as MultiBalance (with error ξ).

3.1 Upper Bounds for the Hypercube

An implementation of MultiBalance that runs in $O((n/p)(\log g \log p)^{1/2} + g \log^2 p)$ time on the hypercube will now be presented. The following definitions, which build on Definitions 2.5 and 2.6, are required.

Definition 3.1 Given an instance of MultiBalance, the n tokens have been *multi-balanced across dimension* i if and only if each group j has been balanced across dimension i, $0 \leq j < g$.

Definition 3.2 Let δ_i^j denote the discrepancy across dimension i with respect to group j, $0 \leq j < g$. Define the *multi-discrepancy across dimension* i, denoted δ_i^M, as the sum $\sum_{0 \leq j < g} \delta_i^j$.

Algorithm MultiBalance

1. Each processor partitions its set of tokens into g subsets, one subset corresponding to each of the g token types. Each of the subsets is counted and the results are stored. This takes $O(n/p)$ time.

2. Let l denote the dimension of the subcube being multi-balanced ($l = d$ initially). If $l = 0$, return.

3. Compute δ_i^M, $0 \leq i < l$. This involves performing l independent sums over the entire subcube for each of the g groups. Each set of l sum operations can be pipelined to run in $O(l)$ time [5], so the total running time is $O(gl)$.

4. Determine the dimension i^* with least associated multi-discrepancy $\delta_{i^*}^M$. This takes $O(l)$ time.

5. Recursively multi-balance the high and low subcubes with respect to dimension i^*, using Steps 2 to 6.

6. Multi-balance across dimension i^*, adjusting the token counts appropriately. The running time of this step is analyzed below.

The correctness of the preceding implementation of MultiBalance with error $\log p$ follows by induction using Lemma 2.3. If a minimum error multi-balancing is desired, $O(\log^2 p)$ post-processing per group can be performed as described in Section 2. The total cost of post-processing is thus $O(g \log^2 p)$ time.

In order to complete the analysis of the running time of algorithm MultiBalance, it is necessary to consider the cost of Step 6. Let Δ_j denote the sum of the discrepancies with respect to group j in the given hypercube of dimension d, $\sum_{0 \le i < d} \delta_i^j$. Let Δ^M denote the sum of the multi-discrepancies $\sum_{0 \le i < d} \delta_i^M = \sum_{0 \le j < g} \Delta_j$.

Lemma 3.1 For any instance of MultiBalance, $\Delta^M = O(dk^{-1/2}n)$, where k satisfies $g = p^{1/k}$.

Proof: Let the number of tokens in group j be denoted x_j, $0 \le j < g$, and consider the contribution of Δ_j to Δ^M. Since $\sum_{0 \le j < g} x_j = n$, there is at most one x_j that exceeds $n/2$. Suppose that $x_l \ge n/2$ for some group l. Then $\Delta_l = O((n/p)d^{1/2}p) = O(d^{1/2}n)$ by Theorem 2.3. Now $k \le \log p = d$ (since $g \ge 2$), so $d^{1/2} \le dk^{-1/2}$ and $\Delta_l = O(dk^{-1/2}n)$.

Hence, it may be assumed that $x_j \le n/2$, $0 \le j < g$. Let k_j satisfy $x_j/n = p^{-1/k_j}$, $0 \le j < g$. The tokens of group j can be packed into p^{1-1/k_j} processors, and by Lemmas 2.4 and 2.5, $\Delta_j = O((n/p)dk_j^{-1/2}p^{1-1/k_j}) = O(dx_j k_j^{-1/2}) = O(d^{1/2}x_j \log^{1/2}(n/x_j))$. Consider the function $f(x) = x \log^{1/2}(n/x)$, where x is a real value in the range $[1, n/2]$. One may easily verify that $f''(x) < 0$ in this range. In other words, $f(x)$ is a concave function. Therefore, the sum $\sum_{0 \le j < g} f(x_j)$, subject to the constraint $\sum_{0 \le j < g} x_j = n$, is maximized when all of the x_j's are equal, that is, when $x_j = n/g$. Forcing the x_j's to be integers can only decrease this sum, so $\Delta^M = O(d^{1/2}gf(n/g)) = O(d^{1/2}g(n/g)\log^{1/2} g) = O(dk^{-1/2}n)$, as required. □

Lemma 3.2 For any instance of MultiBalance, the average multi-discrepancy Δ^M/d across a dimension is $O(n(\log g/\log p)^{1/2})$.

Proof: Immediate from Lemma 3.1, since $g = p^{1/k}$ and $k = \log p/\log g$. □

Theorem 3.1 The MultiBalance operation runs in $O((n/p)(\log g \log p)^{1/2} + g \log^2 p)$ time on the hypercube.

Proof: By a simple extension of the argument given in Section 2, the time required to perform the multi-balancing step is $O(\delta_{i*}^M 2^{-l} + gl)$. Now δ_{i*}^M is certainly no more than Δ^M/d, and the number of tokens in a subcube of dimension l at depth $d-l$ of the recursion is $O(n2^{l-d} + g2^l(d-l))$, where the latter term bounds the cumulative effect of odd parity in the balancing operations. Thus, Lemma 3.2 implies that the total time expended in Step 6 is

$$O\left(\sum_{1 \le l \le d} 2^{-l}(n2^{l-d} + g2^l(d-l))(\log g/l)^{1/2} + gl\right),$$

which reduces to $O((n/p)(\log g \log p)^{1/2} + g \log^2 p)$. This dominates the time required by all other steps of the algorithm, including the post-processing. □

3.2 A Lower Bound for the Hypercube

It is easy to see that $\frac{1}{2}(\Delta^M - dgp)/p$ is a lower bound on the running time of MultiBalance, since Δ^M can only decrease by $2p$ each time step and it is certainly less than dgp after the MultiBalance operation has been performed. Hence, exhibiting a particular input instance with a high value of Δ^M gives a good lower bound on the worst case running time of MultiBalance. Consider the input instance given by the following construction.

Assume for convenience that $g = 2^r$, $1 \leq r \leq d$, and let $q = \lfloor d/r \rfloor - 1$ or $\lfloor d/r \rfloor$, whichever is odd. Divide the first qr bits of each processor ID into r *fields* of q contiguous bits. The ith field determines the ith bit of an r-bit *condensed ID* according to the following rule, $0 \leq i < r$. If the majority of the q bits in the ith field are 0, then the ith bit of the condensed ID is 0; otherwise, it is a 1. Note that since q is odd there will always be a strict majority of either 0's or 1's. Furthermore, symmetry implies that exactly 2^{d-r} processors share any particular condensed ID. In the following lemma, let U denote such a subset of 2^{d-r} processors.

Lemma 3.3 The number of hypercube edges from processors in U to processors outside of U is $\Theta(rq^{1/2}|U|)$.

Proof: By symmetry, it is sufficient to prove that the number of such edges associated with the first (say) field is $O(q^{1/2}|U|)$. Also, it may be assumed without loss of generality that the first bit of the condensed ID associated with U is a 0. Let U' denote the subset of the processors of U with $\lceil q/2 \rceil$ 0 bits and $\lfloor q/2 \rfloor$ 1 bits in the first field. Lemma B.2 implies that $|U'| = \Theta(q^{-1/2}|U|)$. It should be clear that only the processors in U' contribute to the desired edge count, and each of these contributes exactly $\lceil q/2 \rceil$. Thus, the number of edges leaving U that are associated with the first field is $O(q^{1/2}|U|)$, as required. □

Theorem 3.2 The MultiBalance operation requires $\Omega((n/p)(\log g \log p)^{1/2} - g \log p)$ time on the hypercube.

Proof: Consider the input configuration obtained by filling each of the processors having condensed ID i with n/p tokens from the ith group, $0 \leq i < 2^l$. Lemma 3.3 implies that each group contributes $\Theta((n/g)(\log g \log p)^{1/2})$ to Δ^M, so $\Delta^M = \Theta(n(\log g \log p)^{1/2})$. As argued above, this fact immediately implies the desired lower bound on the running time of the MultiBalance operation. □

3.3 Average Case Analysis

Given n distinct tokens and p processors, there are

$$\frac{n!}{[(n/p)!]^p}$$

different ways of assigning n/p tokens to each processor, assuming that n is an integer multiple of p. This section analyzes the average case running time of MultiBalance over all of these possible input configurations when there are g distinct groups of tokens. The upper bound to be derived will be interesting for sufficiently small values of g. In the following discussion, the phrase "with high probability" means with probability $1-O(p^{-c})$ for an arbitrary positive constant c.

Let the ith group contain n_i tokens, and consider the expected contribution of group i to the total running time of this version of MultiBalance, $0 \le i < g$. Letting $p = 2^d$, there are $\binom{d}{d'}$ distinct subcubes of dimension d'. Focus attention on one such subcube C, and let the random variable X denote the number of tokens from group i initially assigned to some processor in C. Let Y_j denote the random variable that is 1 if the jth token of group i contributes to X, and 0 otherwise, $0 \le j < n_i$. Letting $\theta = 1/2^{d-d'}$, the expected value of Y_j is θ, and the expected value of X is $n_i\theta$. The variance of Y_j is bounded above by the variance of the binomial distribution with probability θ. Thus, the variance of X is at most $n_i\theta(1-\theta) \le n_i\theta$. A standard Chernoff bound implies that with high probability, X is within $O((n_i\theta \log p)^{1/2})$ of its expected value. Since there are only $p = \sum_{0 \le d' < d} \binom{d}{d'}$ choices for C, *every* subcube of dimension d' contains

$$\frac{n_i}{2^{d-d'}} \pm O\left(\sqrt{\frac{n_i \log p}{2^{d-d'}}}\right)$$

group i tokens with high probability, $0 \le d' < d$. Thus, after balancing the group i tokens over subcubes of dimension d', *every* processor contains

$$\frac{n_i}{p} \pm O\left(\sqrt{\frac{n_i \log p}{p2^{d'}}} + d'\right)$$

group i tokens with high probability. The additive d' bounds the worst case error in the balancing, as given by Lemma 2.3. Note that this is a pessimistic estimate to apply to the average case behavior of MultiBalance, and could certainly be improved. Continuing the analysis, the preceding bound implies that the total cost of the jth multi-balancing operation performed by algorithm MultiBalance is

$$O\left(\sum_{0 \le i < g} \sqrt{\frac{n_i \log p}{p2^j}} + j\right),$$

$0 \le j < d$, with high probability. Summing over j and interchanging the order of summation, the cost of all of the multi-balancing operations is

$$O\left(\sum_{0 \le i < g} \sqrt{(n_i/p)\log p} + g \log^2 p\right)$$

with high probability since the sum over j is dominated by the $j = 0$ term. The remaining sum is maximized by setting $n_i = n/g$, $0 \le i < g$, which leads to a total multi-balancing cost of

$$O\left(\sqrt{(n/p) g \log p} + g \log^2 p\right)$$

with high probability. Note that the $g \log^2 p$ term matches the cost of post-processing and other computations performed by MultiBalance.

The preceding analysis can also be applied to the straightforward implementation of MultiBalance that multi-balances across each of the dimensions in ascending order. Furthermore, this version of MultiBalance can be made to run as efficiently on the shuffle-exchange as it does on the hypercube. For the shuffle-exchange version, shuffles are not performed by moving entire sets of n/p tokens, but rather by moving appropriate subsets of the tokens to make the composition of the set of tokens at each processor (that is, the number belonging to each group) the same as it would have been if a true shuffle had been performed. The total cost of simulating the shuffle operations in this manner is easily seen to be on the same order as that of the multi-balancing operations. This algorithm will be referred to as the shuffle-exchange version of algorithm MultiBalance. The following theorem summarizes the two main results of this section.

Theorem 3.3 The average running time of algorithm MultiBalance, as well as that of the shuffle-exchange version of algorithm MultiBalance, is

$$O\left(\sqrt{(n/p) g \log p} + g \log^2 p\right).$$

4 Summary

This paper has described hypercube and shuffle-exchange algorithms for performing two load balancing operations: Balance and MultiBalance. For the Balance operation, lower bounds were derived by considering the particular input configuration obtained by packing the tokens into a smallest possible set of processors with low expansion. For the hypercube, an algorithm was given that matches the lower bound to within a multiplicative constant if $m \geq \max\{2n/p, \log^{3/2} p\}$ and $m = O(n/p)$. The lower bound for the shuffle-exchange is higher because the hypercube has better expansion properties than the shuffle-exchange. Tight upper and lower bounds were obtained for the shuffle-exchange for m in the range $2n/p \leq m \leq n^{1-\epsilon}$, where ϵ denotes an arbitrarily small positive constant.

Upper and lower bounds were given for the MultiBalance operation on the hypercube. These bounds are tight for $(n/p)(\log g \log p)^{1/2} = \Omega(g \log^2 p)$. A straightforward implementation of MultiBalance on the shuffle-exchange was also described. Finally, the average case complexity of the hypercube and shuffle-exchange implementations of MultiBalance was considered. Not surprisingly, these algorithms behave much better on average than they do in the worst case.

Perhaps the most important application of the Balance and MultiBalance operations is to the problem of sorting. Fast, practical sorting algorithms based on these load balancing primitives are described in [10].

A Expansion Properties of the Hypercube

The calculations in this appendix analyze the volume-to-surface ratio of a Hamming ball of radius $r = r(d)$ lying in a hypercube of dimension d. Theorem B.1, which is used in Section 2.2, characterizes the asymptotic behavior of this ratio for r in the range 0 to $d/2$. The results could easily be extended to handle higher values of r (that is, $d/2 \leq r \leq d$) by taking advantage of symmetry.

B Asymptotic Analysis

Definition B.1 Let $R_{d,r}$ denote $\sum_{0\leq l\leq r}\binom{d}{l}/\binom{d}{r}$.

Lemma B.1 Let d and r be positive integers, $1 \leq r \leq d$. Then $R_{d,r} > R_{d,r-1}$.

Proof: Observe that $R_{d,0} = 1$ and for $1 \leq r \leq d$

$$\begin{aligned}
R_{d,r}/R_{d,r-1} &= \frac{r}{d-r+1}\left(\sum_{0\leq l\leq r}\binom{d}{l}\bigg/\sum_{0\leq l\leq r-1}\binom{d}{l}\right) \\
&> \frac{r}{d-r+1}\min_{1\leq l\leq r}\binom{d}{l}\bigg/\binom{d}{l-1} \\
&= \frac{r}{d-r+1}\min_{1\leq l\leq r}\frac{d-l+1}{l} \\
&= 1.
\end{aligned}$$

□

Lemma B.2 For positive integers d and r, $r \leq d/2$,

$$R_{d,r} = \Theta\left(\sqrt{\frac{d}{z+1}}\right),$$

where $r = d/2 - \sqrt{dz}$.

Proof: The following three cases will be considered separately: $z = \Omega(d)$ and $z \leq d/4$; $z = o(d)$ and $z \geq 1$; $0 \leq z \leq 1$.

Case 1: $z = \Omega(d)$ and $z \leq d/4$. Exercise (9.42) of Graham, Knuth and Patashnik [3] establishes that $R_{d,r} = \Theta(1)$ in this range.

Case 2: $z = o(d)$ and $z \geq 1$. It is sufficient to prove that $R_{d,r} = \Theta\left(\sqrt{d/z}\right)$ since $z = \Omega(1)$. For the lower bound, consider the inequality

$$\sum_{0\leq l\leq r}\binom{d}{l} \geq \left\lfloor\sqrt{d/z}\right\rfloor\binom{d}{r-\left\lfloor\sqrt{d/z}\right\rfloor},$$

and observe that for sufficiently large values of d

$$\begin{aligned}
\binom{d}{r-\lfloor\sqrt{d/z}\rfloor}\Big/\binom{d}{r} &\geq \left(\frac{r-\lfloor\sqrt{d/z}\rfloor}{d-r+\lfloor\sqrt{d/z}\rfloor}\right)^{\lfloor\sqrt{d/z}\rfloor} \\
&\geq \left(\frac{d/2-\sqrt{dz}-\sqrt{d/z}}{d/2+\sqrt{dz}+\sqrt{d/z}}\right)^{\sqrt{d/z}} \\
&\geq \left(\frac{d/2-2\sqrt{dz}}{d/2+2\sqrt{dz}}\right)^{\sqrt{d/z}} \\
&\geq \left(1-\frac{8}{\sqrt{d/z}}\right)^{\sqrt{d/z}},
\end{aligned}$$

which converges to $e^{-8}=\Omega(1)$. Hence, $R_{d,r}=\Omega(\sqrt{d/z})$.

For the upper bound, note that

$$\binom{d}{l-1}\Big/\binom{d}{l} \leq \frac{r}{d-r+1},$$

for $1 \leq l \leq r$. Hence, the sum $\sum_{0\leq l\leq r}\binom{d}{l}$ is dominated by the infinite geometric progression with initial value $\binom{d}{r}$ and ratio $r/(d-r+1)$ between successive terms. Thus,

$$\begin{aligned}
R_{d,r} &\leq \frac{d-r+1}{d-2r+1} \\
&= O\left(\sqrt{d/z}\right).
\end{aligned}$$

Case 3: $0\leq z\leq 1$. It is sufficient to prove that $R_{d,r}=\Theta\left(\sqrt{d}\right)$ since $z=O(1)$. A lower bound on $R_{d,r}$ can be obtained as follows:

$$\begin{aligned}
\sum_{0\leq l\leq r}\binom{d}{l} &\geq \sum_{r-\lfloor\sqrt{d}\rfloor\leq l\leq r}\binom{d}{l} \\
&\geq \lfloor\sqrt{d}\rfloor\binom{d}{r-\lfloor\sqrt{d}\rfloor}
\end{aligned}$$

Furthermore, for sufficiently large values of d

$$\binom{d}{r-\lfloor\sqrt{d}\rfloor}\Big/\binom{d}{r} \geq \left(\frac{r-\lfloor\sqrt{d}\rfloor}{d-r+\lfloor\sqrt{d}\rfloor}\right)^{\lfloor\sqrt{d}\rfloor}$$

$$
\begin{aligned}
&\geq \left(\frac{d/2 - 2\sqrt{d}}{d/2 + 2\sqrt{d}}\right)^{\sqrt{d}} \\
&\geq \left(1 - \frac{8}{\sqrt{d}}\right)^{\sqrt{d}},
\end{aligned}
$$

which converges to $e^{-8} = \Omega(1)$. Hence, $R_{d,r} = \Omega(\sqrt{d})$.

For the upper bound, Stirling's approximation can be used to show that $R_{d,\lfloor d/2 \rfloor} = \Theta(\sqrt{d})$. It follows from Lemma B.1 that $R_{d,r} = O(\sqrt{d})$. $\square$

Theorem B.1 Let d be a given integer and let $r = r(d)$ be an integer between 0 and $d/2$. If $\sum_{0 \leq l \leq r} \binom{d}{l} = 2^{d(1-1/k)}$ then $R_{d,r} = \Theta(\sqrt{k})$.

Proof: The following four cases will be considered separately: $r = d/2 - \Omega(d)$ and $r \geq 0$; $r = d/2 - o(d)$ and $r \leq d/2 - d^{2/3}$; $d/2 - d^{2/3} \leq r \leq d/2 - \sqrt{d}$; $d/2 - \sqrt{d} \leq r \leq d/2$.

Case 1: $r = d/2 - \Omega(d)$ and $r \geq 0$. In this range, $R_{d,r} = \Theta(1)$ by Lemma B.2, and $k = \Theta(1)$ by Exercise (9.42) of [3]. Hence, $R_{d,r} = \Theta(\sqrt{k})$.

Case 2: $r = d/2 - o(d)$ and $r \leq d/2 - d^{2/3}$. Let $r = d/2 - d^{1/2+\delta}$, hence $\delta = 1/2 - \omega(1/\log d)$ and $\delta \geq 1/6$. As in the preceding case, $R_{d,r} = \Theta(d^{1/2-\delta})$ by Lemma B.2. The logarithmic form of Stirling's approximation implies that $\ln n! = n \ln n - n + O(\ln n)$. Hence,

$$
\begin{aligned}
\ln \binom{d}{r} &= d \ln d - r \ln r - (d-r)\ln(d-r) + O(\ln d) \\
&= d \ln d - (d/2 - d^{\delta+1/2})\ln(d/2 - d^{\delta+1/2}) \\
&\quad -(d/2 + d^{\delta+1/2})\ln(d/2 + d^{\delta+1/2}) + O(\ln d) \\
&= d \ln 2 - (d/2 - d^{\delta+1/2})\ln(1 - 2d^{\delta-1/2}) \\
&\quad -(d/2 + d^{\delta+1/2})\ln(1 + 2d^{\delta-1/2}) + O(\ln d).
\end{aligned}
$$

The following pair of inequalities may be easily derived from the Taylor's series expansion of $\ln(1+x)$:

$$x - x^2/2 \leq \ln(1+x) \leq x, \;\; x \geq 0, \;\text{ and}$$

$$-x - x^2 \leq \ln(1-x) \leq -x - x^2/2, \;\; 0 \leq x \leq \frac{1}{2}.$$

These inequalities imply that for sufficiently large values of d,

$$
\begin{aligned}
\ln \binom{d}{r} &\geq d \ln 2 - (d/2 - d^{\delta+1/2})(-2d^{\delta-1/2} - 2d^{2\delta-1}) \\
&\quad -(d/2 + d^{\delta+1/2})(2d^{\delta-1/2}) + O(\ln d) \\
&= d \ln 2 - 3d^{2\delta} - 2d^{3\delta-1/2} + O(\ln d) \\
&\geq d \ln 2 - 5d^{2\delta} + O(\ln d),
\end{aligned}
$$

and

$$\begin{aligned}
\ln \binom{d}{r} &\leq d \ln 2 - (d/2 - d^{\delta+1/2})(-2d^{\delta-1/2} - 4d^{2\delta-1}) \\
&\quad -(d/2 + d^{\delta+1/2})(2d^{\delta-1/2} - 2d^{2\delta-1}) + O(\ln d) \\
&= d \ln 2 - d^{2\delta} - 2d^{3\delta-1/2} + O(\ln d) \\
&\leq d \ln 2 - d^{2\delta} + O(\ln d).
\end{aligned}$$

Thus, $\sum_{0 \leq l \leq r} \binom{d}{l} = R_{d,r} \binom{d}{r} = 2^{d-\Theta(d^{2\delta})}$ where the $O(\ln d)$ term has been absorbed into the $\Theta(d^{2\delta})$ term (using the fact that $\delta \geq 1/6$). Hence, $k = \Theta(d^{1-2\delta})$ and $R_{d,r} = \Theta(\sqrt{k})$.

Case 3: $d/2 - d^{2/3} \leq r \leq d/2 - \sqrt{d}$. Let $r = d/2 - d^{1/2+\delta}$, $0 \leq \delta \leq 1/6$. In this case, $R_{d,r} = \Theta(d^{1/2-\delta})$ by Lemma B.2. Equation (9.98) of [3] implies that

$$\binom{d}{r} = \Theta\left(\frac{2^d}{\sqrt{d}} e^{-2d^{2\delta}}\right),$$

so multiplying by $R_{d,r}$ gives

$$\sum_{0 \leq l \leq r} \binom{d}{l} = \Theta\left(\frac{2^d}{d^{\delta}} e^{-2d^{2\delta}}\right) = \Theta(2^{d(1-1/k)}),$$

for $k = d \ln 2/(\delta \ln d + 2d^{2\delta})$. Observe that $k = \Theta(d^{1-2\delta})$ since $d^{2\delta} = \Omega(\delta \ln d)$, $\delta \geq 0$. Hence $R_{d,r} = \Theta(\sqrt{k})$.

Case 4: $d/2 - \sqrt{d} \leq r \leq d/2$. In this case, $R_{d,r} = \Theta(\sqrt{d})$ by Lemma B.2. Together with Equation (9.98) of [3], this implies that $\sum_{0 \leq l \leq r} \binom{d}{l} = \Theta(2^d)$. Hence, $k = \Theta(d)$ and $R_{d,r} = \Theta(\sqrt{k})$. □

References

[1] R. E. Cypher. Theoretical aspects of VLSI pin limitations. *SIAM Journal on Computing*, 22:58–63, 1993.

[2] P. Frankl and Z. Füredi. A short proof for a theorem of Harper about Hamming spheres. *Discrete Mathematics*, 34:311–313, 1981.

[3] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, Reading, MA, 1989.

[4] L. Harper. Optimal numberings and isoperimetric problems on graphs. *J. Combinatorial Theory*, 1:385–393, 1966.

[5] C.-T. Ho and S. L. Johnsson. Distributed routing algorithms for broadcasting and personalized communication in hypercubes. In *Proceedings of the 1986 IEEE International Conference on Parallel Processing*, pages 640–648, 1986.

[6] F. T. Leighton. Personal communication.

[7] F. T. Leighton. *Complexity Issues in VLSI: Optimal Layouts for the Shuffle-Exchange Graph and Other Networks.* MIT Press, Cambridge, MA, 1983.

[8] D. Nassimi and S. Sahni. Parallel permutation and sorting algorithms and a new generalized connection network. *JACM*, 29:642–667, 1982.

[9] D. Peleg and E. Upfal. The token distribution problem. *SIAM Journal on Computing*, 18:229–243, 1989.

[10] C. G. Plaxton. *Efficient Computation on Sparse Interconnection Networks.* PhD thesis, Department of Computer Science, Stanford University, September 1989.

[11] H. S. Stone. Parallel processing with the perfect shuffle. *IEEE Transactions on Computers*, C–20:153–161, 1971.

[12] P. Varman and K. Doshi. Sorting with linear speedup on a pipelined hypercube. Technical Report TR–8802, Department of Electrical and Computer Engineering, Rice University, February 1988.