VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

Department of Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 471 5328
Fax: +64 4 495 5232
Internet: Tech.Reports@comp.vuw.ac.nz

Dimension Exchange Token Distribution on the Linear Array and Ring

Gavin Turner

Technical Report CS-TR-97/1
March 1997

Abstract

We propose and analyse an algorithm to optimally solve the *Token Distribution* problem, a static variant of the load balancing problem, on the ring and linear-array. The algorithm has the added benefit that it is a *dimension exchange* algorithm. Dimension exchange algorithms use a colouring of the interconnection network to pair procesors for data exchange, are extremely simple, and totally scalable.

Publishing Information

Author Information

Gavin Turner received a Ph.D. in Computer Science at the University of Newcastle, Australia in 1996, and is currently a lecturer in Computer Science at Victoria University of Wellington, Wellington New Zealand. His main research interests lie in the area of design and analysis of algorithms, in particular algorithms and architectures for special-purpose parallel computing. In addition, he is interested in a wide variety of problems from image processing, load balancing and combinatorics.

Dimension Exchange Token Distribution on the Linear Array and Ring

Gavin Turner
gavin@comp.vuw.ac.nz
Department of Computer Science,
Victoria University of Wellington, New Zealand

Abstract

We propose and analyse an algorithm to optimally solve the *Token Distribution* problem, a static variant of the load balancing problem, on the ring and linear-array. The algorithm has the added benefit that it is a *dimension exchange* algorithm. Dimension exchange algorithms use a colouring of the interconnection network to pair procesors for data exchange, are extremely simple, and totally scalable.

Keywords : Token Distribution, Dimension-Exchange, Ring, Linear Array

1 Introduction

The **Token Distribution** problem is a static variant of the load balancing problem. Given that each processor possesses some initial load, the goal of a token distribution algorithm is to distribute the tokens (tokens) so that the final load is close to the average load. A token distribution algorithm is judged by two parameters: the time T that the algorithm needs to produce the final (desired) distribution, and the maximal deviation δ from the average load which it guarantees for all processors.

Unlike many data distribution methods which require the gathering of a certain amount of global information [7, 3, 14], the so-called *dimension-exchange* method requires nothing more than the exchange of data elements between directly neighbouring PEs. The method is based on the repetitive application of an extremely simple, local exchange protocol, and all dimension-exchange algorithms are based on the same simple structure. To implement such an algorithm on any topology, the communication edges of the topology must be *partitioned* (or *coloured*) into sets whereby no two edges of the same set are incident on the same processor. For networks which have a uniform structure, (for example mesh-connected topologies), such a partition is trivial. For other networks, partitions may be based on a set of *matchings*.

Recall that a matching F over graph $G = (V, E)$, $F \subseteq E$, is a set of graph edges such that no two edges share a vertex. A k-colouring of a graph, $G = (V, E)$, partitions E into sets $E_1, ..., E_k$ such that each E_h is a matching. Typically, each edge in set E_h is thought to receive *colour* h, and is said to exist along *dimension* h.

Dimension-exchange algorithms use the *dimensions* of a network to pair processors for data exchange and are identically based on the following structure:

```
Dimension-Exchange Algorithm
LOOP
    FOR i = colour 1 to d (* d is the number of dimensions *)
        Apply some local Comparison-Exchange step over connections
        with colour i;
END
```

Dimension-Exchange algorithms have been used sucessfully in the implementation of algorthms for sorting, (for example the well known algorithm of Batcher [1]), and form an integral part of many of the so-called *Hot-Potato* routing algorithms [13, 12, 10]. Due to their simple structure and scalability, many researchers have studied the applicability of dimension-exchange algorithms to the problem of token distribution; the first being Cybenko [2] in 1987.

Based on the assumption that the load in each processor could be treated as a real number (that is, the load is *infinitely-divisible*), Cybenko [2] analysed the performance of a dimension-exchange algorithm for the hypercube using a matrix-iterative approach. The result of Cybenko's analysis was to show that for an algorithm in which the processors connected along dimension i *exactly equalised* their local loads in step i, a totally balanced load would exist after d steps on a d-dimensional hypercube.

This original work prompted a steady stream of research in the analysis of dimension-exchange algorithms for load balancing and token distribution. In 1988 Ranka *et al.* [11] studied the operation of Cybenko's algorithm empirically for finitely-divisible (integer-based) loads on the d-dimensional hypercube. Ranka *et al.* observed a resulting global discrepancy, (being the difference between the maximum number and the minimum number of tokens in any PE), of approximately d for the d-dimensional hypercube. In 1989 and 1990 Hosseini *et al.* [6] and Plaxton [9] confirmed this observation. For finitely-divisible loads, a dimension-exchange algorithm on the hypercube which in each time step reduced the discrepancy of neighbouring processors to at most 1, was proven to lower the global discrepancy to at most d in d such iterations. Their results were completed using an extension of the matrix-iterative approach of Cybenko. In addition, Hosseini *et al.* demonstrated that, for infinitely-divisible loads, the dimension exchange method could be generalised for arbitrary networks by introducing a k-colouring to the network (as described above). Assuming that the loads of each processor were infinitely-divisible, and that an appropriate k-colouring could be found, Hosseini *et al.* were able to show how Cybenko's matrix-iterative approach could be successfully applied to networks of arbitrary topology.

In 1992 Xu and Lau [15] extended the work of Hosseini *et al.* by showing that for some topologies the *rate* at which the global discrepancy converged to zero could be optimized by altering the *ratio* with which infinitely-divisible loads were locally balanced. Later that year the same authors [16] computed optimal ratios for the linear array, ring, 2-dimensional mesh and 2-dimensional torus.

In 1996, Houle and Turner [4] provided a dimension-exchange token-distribution algorithm for the linear array and ring. This algorithm, based on the repeated reduction of the load dicrepancy by 1, was proven to reduce the differnece between the maximum and minimum processor loads in at most n steps on the ring and linear array. In this paper, a dimension-exchange solution to the token-distribution problem is proposed and formally analysed for the ring and linear array. The work presented here improves on that of Houle and Turner, in that it provably reduces the difference between the maximal processor load and minimal processor load by c in at most $\lceil \frac{c}{2} \rceil \cdot n$ steps on the linear array and $\lceil \frac{c}{2} \rceil \cdot \frac{n}{2}$ steps on the ring, as compared to $c \cdot n$ steps on both the linear array and ring for the original algorithm.

2 Model of Computation

The *Ring* and the *Linear Array*, static interconnection networks of length N may be defined as follows:

Definition 1 *A Ring R_N consists of N nodes arranged in a length N chain. Each node represents a processor and each edge a communication link. The node at position i, $(0 \leq i \leq N-1)$, of the ring is connected through direct communication links to 2 neighbours : $(i \pm 1) \bmod N$. An array for which boundary processors (those for which $i = 0$ or $i = n-1$) have one less neighbour, is denoted a Linear Array, L_N.*

The ring interconnection network is illustrated in Figure 1. In terms of load balancing, a task (token) may be of significant size. However, since the token distribution problem is in itself an *abstraction* of load balancing, we must assume that a token can be as small as 1 word. In keeping with the timing models for problems such as routing and sorting, we assume that one time step represents the time required to transfer one token between neighbouring processing elements.

3 The Token Distribution Problem

The token distribution problem $TD(M; L, k, \delta)$ was first defined by Peleg and Upfal in [8] as follows: Given parallel architecture M containing N processors $P_1, ..., P_N$, with each processor P_i containing, at some time t, a stack of $l(P_i, t) \leq k$ tokens (for all $1 \leq i \leq N$) and for a total number of tokens $L(t) = \sum_{i=1}^{N} l(P_i, t)$, distribute the tokens such that at the end all processors contain $\frac{L(t)}{N} \pm \delta$.

If the maximum height of all stacks is $k = max_s$ and the minimum stack height is min_s then we define the *global discrepancy* $\Delta = max_s - min_s$ to be the maximal load difference between processors. If $L(t) \in \Theta(k \cdot n^d)$ then it is easily concluded from a simple bisection argument that the following lower bound is true.

Lemma 1 *To solve problem $TD(R_N; L(t), k, \delta)$, $\Omega((\Delta - 2 * \delta) \cdot N \cdot T)$ steps are required by any algorithm on the ring of length N and for token size T.*

4 Integer Dimension Exchange Approach

4.1 Network Colouring

To implement a dimension-exchange algorithm, two properties must be determined; the network colouring over which the algorithm will be applied, and a suitable comparison-exchange step. The algorithm in this paper will be based on the so-called odd-even colouring illustrated in Figure 1, in which all *odd* connections are given colour 1 and all *even* connections are given colour 2. The choice of an elementary comparison-exchange step is discussed in the next section.

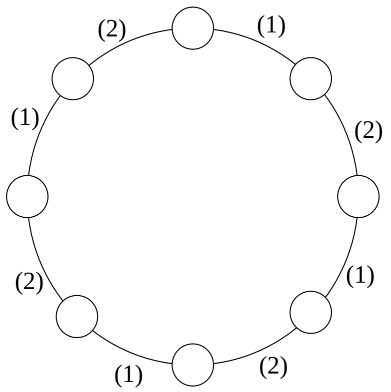

Figure 1: Odd Even Colouring of the Ring

4.2 Elementary Steps

Many different comparison-exchange balancing steps may be used to construct a dimension-exchange based token-distribution algorithm. In the analysis presented by Plaxton [9] and Hosseini *et al.* [6], the elementary balancing step shown in the first row of Table 1, where a and b are the initial numbers of tokens in the cooperating PEs, and a', b' are the resulting numbers of tokens in the respective PEs, was used as a basis for dimension-exchange balancing on the hypercube.

The elementary step of row 1 (Table 1) seems intuitively correct since the local loads of PEs a and b will only be altered when the distance of the global distribution from an even distribution will result. In both [9] and [6] it was shown that such a dimension-exchange algorithm could, on the hypercube, reduce the discrepancy Δ to a level proportional to the *diameter* of the network, (that is $\mathcal{O}(\log N)$), but *no further* in $\mathcal{O}(\log N)$ iterations. Note that on the linear array and ring the same is true; a dimension-exchange algorithm based on this elementary step cannot affect the step distribution[1] shown in Figure 2, and therefore, in the the worst case, is unable to reduce $\Delta \in \mathcal{O}(N)$. This observation was also made by Horton [5] who failed, however, to realize that a simple alteration in the comparison-exchange step (to that given in the second row of Table 1) negates this shortcoming; the algorithm so produced is able to reduce Δ to 2 on the ring and to 1 on the linear array[2].

However, when implementing the elementary step of row 2 on SIMD machines, inter-PE synchronisation is required at the end of each balancing step, given that a different number of

[1]Note that in all diagrams, the load of a PE will be represented by a solid column of blocks, each block representing a token, and the height of the column being equivalent to the number of tokens a PE contains.

[2]This will be proven in subsequent sections.

	Conditions	Rules
1.	if $l(a,t) > l(b,t)$	$l(a,t+1) = \lceil \frac{l(a,t)+l(b,t)}{2} \rceil$; $l(b,t+1) = \lfloor \frac{l(a,t)+l(b,t)}{2} \rfloor$
	else if $l(a,t) < l(b,t)$	$l(a,t+1) = \lfloor \frac{l(a,t)+l(b,t)}{2} \rfloor$; $l(b,t+1) = \lceil \frac{l(a,t)+l(b,t)}{2} \rceil$
	else	$l(a,t+1) = l(a,t)$; $l(b,t+1) = l(b,t)$
	Conditions	**Rules**
2.	if $l(a,t) > l(b,t)$	$l(a,t+1) = \lfloor \frac{l(a,t)+l(b,t)}{2} \rfloor$; $l(b,t+1) = \lceil \frac{l(a,t)+l(b,t)}{2} \rceil$
	else if $l(a,t) < l(b,t)$	$l(a,t+1) = \lceil \frac{l(a,t)+l(b,t)}{2} \rceil$; $l(b,t+1) = \lfloor \frac{l(a,t)+l(b,t)}{2} \rfloor$
	else	$l(a,t+1) = l(a,t)$; $l(b,t+1) = l(b,t)$
	Conditions	**Rules**
3.	if $l(a,t) > l(b,t)$	$l(a,t+1) = l(a,t) - 1$; $l(b,t+1) = l(b,t) + 1$
	else if $l(a,t) < l(b,t)$	$l(a,t+1) = l(a,t) + 1$; $l(b,t+1) = l(b,t) - 1$
	else	$l(a,t+1) = l(a,t)$; $l(b,t+1) = l(b,t)$

Table 1: Some Possible Elementary Balancing Steps

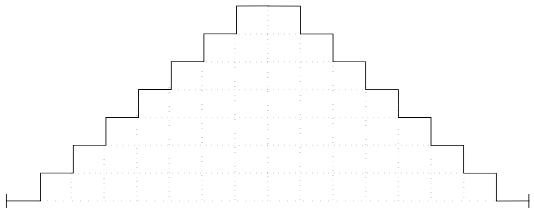

Figure 2: Step Token Distribution: Bad Case for Some Elementary Balancing Steps

tokens may need to be transferred between each pair of communicating processors. If hardware does not exist for some form of efficient global OR function (or a similar operation) this can take worst-case time in $\mathcal{O}(N)$.

For SIMD machines, a more appropriate choice is the elementary step given in row 3 of Table 1 which shares step 2's ability to reduce Δ to 1 on the linear array and 2 on the ring. In this case the dimension-exchange algorithm is extremely simple, (all data transfers are of size equivalent to the size of a single token), and no inter-PE synchronisation is necessary. Furthermore, in section 4.4 it is argued that on SIMD architectures where the communication of i tokens must require time $\in \mathcal{O}(i)$, (assuming the communication of a single token takes time $\in \mathcal{O}(1)$), an algorithm based on this simple step performs better, or at worst equal, to its competitor (the step of row 2). This comparison is made in terms of the time complexity required to reduce Δ by some value c. For these reasons step 3 becomes the basis for the dimension-exchange algorithms explored in this paper.

4.3 Solving Arbitrary Token Distribution Problems

Based on the elementary balancing step chosen in the previous section, we propose a dimension-exchange algorithm which solves all token distribution problems on the linear array and ring in an optimal way[3]. Results which describe the time complexity of integer-based dimension-exchange methods have, to date, been reported only for the hypercube. The results presented here are important as they demonstrate that dimension-exchange methods are useful even on low bisection-width architectures.

[3]That is, optimal in terms of the bisection bound imposed by the constant bisection width of the linear array and ring.

The algorithm, 1D-DEB (1 dimensional, dimension exchange balancing), which decreases $\Delta = max_s - min_s$ by c, $1 \leq c \leq \Delta - 1$ is as follows:

Algorithm 1D-DEB(c)
(* *Reduces Δ by c on the Ring or Linear Array (Note that c=2 is special case)* *)
LOOP $\lceil \frac{c}{2} \rceil \frac{N}{4}$ times on the ring or $\lceil \frac{c}{2} \rceil \frac{N}{2}$ on the linear array
    Apply Elementary Comparison-Exchange step (#3) over
        All *Odd* pairs of PEs
        All *Even* pairs of PEs
END

It is immediately obvious that no algorithm is able to decrease $\Delta = 1$. It is straightforward however, to show that a dimension-exchange algorithm such as (1D-DEB) which uses elementary comparison-exchange step 3 from Table 1 will eventually reduce discrepancy Δ by 1 when $\Delta > 1$ on the linear array, and when $\Delta > 2$ on the ring. That is to say, the discrepancy *converges*. Lemma 2 states this result.

Lemma 2 (Convergence) *1D-DEB eventually reduces all $\Delta > 1$ by 1 on the linear array, and all $\Delta > 2$ by 1 on the ring.*

Proof : Let $l(i,t)$ denote the numbers of tokens stored in processor i at time t. Let us assume that at time t_0 before balancing begins, the number of processors with $l(i,t_0) = min_s$ tokens is $\leq$ the number of processors with $l(j,t_0) = max_s$ tokens. At all processors i with $l(i,t_0) > min_s + 1$ tokens, let us colour the token at height $l(i,t_0)$ **black** (see Figure 3(a)). At all processors i with $l(i,t_0) = min_s$, let us say that a *hole* exists at height min_s (also see Figure 3(a)).

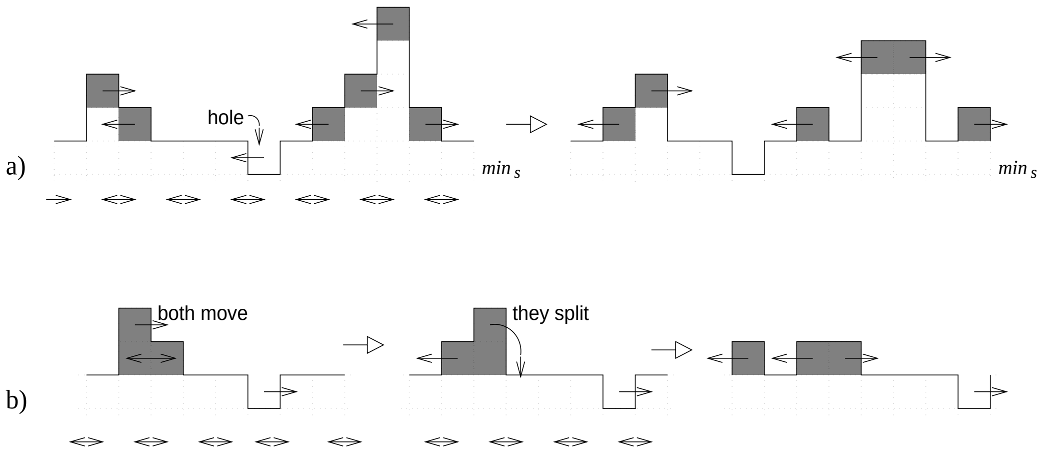

Figure 3: Illustrating the Arguments for Lemma 19

With each balancing step, we assume that black tokens and holes move from processor to processor across the communication links. At each comparison step, the following cases may occur:

1. If a comparison occurs at time t between 2 processors i and $i+1$ such that both $l(i,t)$ and $l(i+1,t) > min_s + 1$, then the black tokens (are assumed to) swap positions[4] (see Figure 3(a)).
2. Similarly if a comparison occurs at time t between two processors i and $i+1$, both with $l(i,t) = l(i+1,t) = min_s$, then the two holes (are assumed to) swap positions.
3. If $l(i,t) = min_s$ and $l(i+1,t) = min_s + 1$ (or vise-versa), OR, $l(i,t) > min_s + 1$ and $l(i+1,t) = min_s + 1$ (or vise-versa), then the hole and black token (respectively) simply move forward one position (see Figure 3(a)).

[4] Note that the loads of the comparing processors (and therefore the heights of the black tokens) may not stay the same as a result of the comparison; we simply observe that the new loads of both PEs must again be $> min + 1$, and therefore each has a black token. Therefore, it is convenient for us to imagine that a *swap* of black tokens has taken place.

4. Finally, if a comparison occurs at time t between 2 processors i and $i+1$ such that $l(i,t) = min_s$ and $l(i+1,t) > min_s + 1$ (or vise-versa), then the black token and hole are said to *collide* and destroy each other.

Note that both holes and black tokens move across the linear array or ring, one position per communication step, until they collide with a black token or hole respectively. Neither a hole nor a black token will be delayed since black tokens move through each other, as do holes. When either all holes or all black tokens have been destroyed, then Δ has been decreased by at least 1.

On the Linear Array : a hole and a black token can move at most N steps without colliding due to the fact that both holes and black tokens reverse directions at processors 0 and N-1. Therefore for $\Delta > 1$, all holes will be destroyed (since the number of processors initially with min_s is less than the number of processors initially with max_s) in at most N steps.

On the Ring : since there are no places at which a hole or black token may change directions, black tokens and holes which move in the same direction may never meet. Therefore, if $\Delta = 2$, distributions exist for which 1D-DEB will never reduce Δ. However, if we extend our analogy slightly, we can show that this is not the case for $\Delta \geq 3$.

Now consider each processor with $l(i,t_0) = max_s$, $max_s \geq min_s + 3$, as the originator of two black tokens; one at height max_s and one at height $max_s - 1$ (see Figure 3(b)). In each comparison step at time t, the following *additional cases* may occur:

1. if $l(i,t) = l(i+1,t) = max_s$, then assume that both the black tokens from i move to $i+1$, and both the black tokens from $i+1$ move to i.
2. if $l(i,t) = max_s$ and $l(i+1,t) > min_s + 1$, then assume that both the black tokens move from i to $i+1$ and the single black token at $i+1$ moves to i (see Figure 3(b)).
3. if $l(i,t) = max_s$ and $l(i+1,t) = min_s + 1$, then one black token moves from i to $i+1$ and one stays behind; that is, the black tokens *split* and begin to move in opposite directions (see Figure 3(b)).
4. if $l(i,t) = max_s$ and $l(i+1,t) = min_s$, then one black token from i and the hole from $i+1$ are destroyed, leaving a single black token at i which has reversed directions.

Now we note that if the number of processors with $l(i,t_0) = max_s$ is $\geq$ the number of processors with $l(j,t_0) = min_s$ (initially), then no hole can survive for more than N steps since at least the same number of black tokens as holes will be moving in both directions after $\frac{N}{2}$ steps (counting those black tokens which may have already destroyed a hole).

These results may be replicated for the case where the number of processors with min_s tokens is $>$ the number of processors with max_s tokens since the system is symmetric with respect to maxima and minima (substitute black holes for black tokens). Therefore we have shown that the lemma holds. □

For the case $\Delta = 2$ the algorithm, when implemented on the ring, is forced to simulate a linear array. In the remainder of this section we determine the running time (number of iterations) required of 1D-DEB to achieve an arbitrary reduction $c \geq 1$ in Δ. Let a *block distribution* be defined as follows.

Definition 2 *Let $(m : N - m)$ denote a token distribution in which the first m processors have max_s tokens and the remaining $N - m$ processors have min_s tokens OR one in which the first $N - m$ processors have min_s tokens and the remaining m processors have max_s tokens. Such a token distribution is denoted a* **block distribution**.

The following lemma recognises that, in terms of time required by 1D-DEB to reduce Δ on the ring or linear array, a *block token distribution* represents the worst-case token distribution for the algorithm.

Lemma 3 *The worst-case "Block" token distribution (in terms of the number of steps required to decrease $\Delta \geq 3$ by some value $c \leq \Delta - 2$) is the worst-case distribution for algorithm 1D-DEB.*

Proof : Such a statement seems intuitively correct since, in reducing Δ by c, a block distribution maximises the number of tokens which must be moved across the cut separating the processors with max_s tokens and those with min_s tokens.

The worst-case time to reduce Δ is dependent on two factors; the maximum distance required of any token to move before meeting a hole and the number of tokens which must be moved across a single edge to reduce c. A block distribution maximises the distance between holes and tokens, and also maximises the number of tokens required to cross any edge - specifically, the edges separating the minimal and maximal blocks. Therefore, any attempt to deviate from the worst-case block distribution will result in a distribution requiring an at-worst *equal* and at best *fewer* number of steps to reduce Δ by c. □

Based on Lemma 3, a discussion is now presented which recognises the worst-case running time of 1D-DEB required to reduce the discrepancy Δ by an arbitrary amount c.

Observation 1 *After $\lceil \frac{c}{2} \rceil N$ steps on the linear array or $\lceil \frac{c}{2} \rceil \frac{N}{2}$ steps on the ring, ($1 \leq c \leq \Delta - 2$), 1D-DEB will produce a decrease in Δ of at least c for block distribution ($\frac{N}{2} : \frac{N}{2}$).*

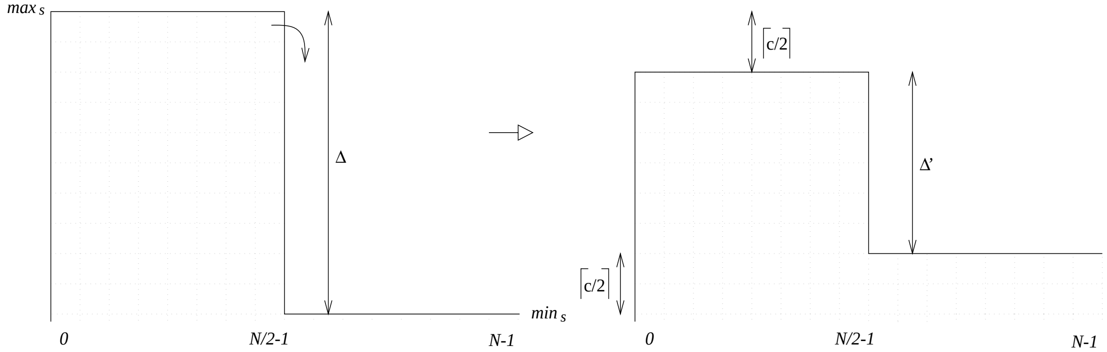

Figure 4: Reducing the Block Distribution ($\frac{N}{2} : \frac{N}{2}$)

Referring to Figure 4, it is easy to see that to reduce Δ by c for this block distribution, $\lceil \frac{c}{2} \rceil \cdot \frac{N}{2}$ tokens must be moved from the maximal block to the minimal block. As a result of odd-even transposition, after the first step a token will move from the maximal block to the minimal block *at least every 2 steps* on the linear array, and *two tokens at least every 2 steps* on the ring. Therefore, the time required to reduce Δ by c for this block distribution is

$$\begin{array}{ll} \lceil \frac{c}{2} \rceil N & \text{on the linear array and} \\ \lceil \frac{c}{2} \rceil \frac{N}{2} & \text{on the ring.} \end{array}$$

The following lemma gives us that for all $c \neq 2$, the worst-case time to reduce Δ by c is the time required for block distribution $(N/2 : N/2)$.

Lemma 4 *Block distribution ($\frac{N}{2} : \frac{N}{2}$) is the worst-case distribution for all reductions $c \neq 2$ in Δ, requiring $\lceil \frac{c}{2} \rceil \frac{N}{2}$ steps on the ring and $\lceil \frac{c}{2} \rceil N$ on the linear array. The exception, $c = 2$, has worst-case distribution ($\frac{2N}{3} : \frac{N}{3}$) and requires $\frac{2N}{3}$ steps on the ring and $\frac{4N}{3}$ steps on the linear array.*

Proof : Let us consider an arbitrary block distribution $(m : N - m)$, as shown in Figure 5, in which the first $m > \frac{N}{2}$ processors contain max_s tokens and the remaining $N - m$ processors contain min_s tokens.

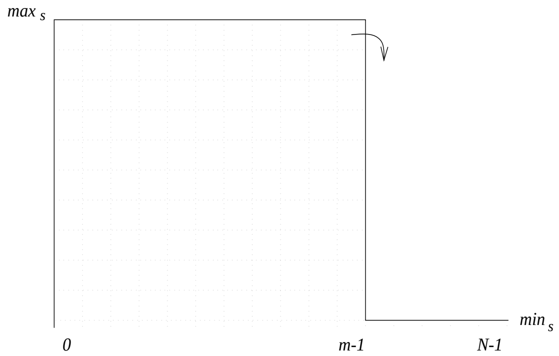

Figure 5: An Arbitrary Block Distribution

The time to reduce Δ by some arbitrary value c is dependent on the ratio of m to N. Let us consider the following cases in which the number of tokens to be moved across the cut varies with this ratio:

1. $m \geq c(N-m)$ (see Figure 6 (a)). The number of tokens we are required to move from the maximal block onto the minimal block before Δ is reduced by c are:

$$c(N-m).$$

Therefore, since

$$\begin{aligned} m &\geq c(N-m) \\ &\geq \frac{c}{c+1}N, \end{aligned}$$

the number of tokens we are required to move is $\leq c(\frac{N}{c+1}) \leq \frac{c}{c+1}N$. This is $< \lceil \frac{c}{2} \rceil \frac{N}{2}$, (the number of tokens needed to be moved for distribution $(\frac{N}{2} : \frac{N}{2})$), for all $c \geq 3$. *For c=2, the maximum number of tokens required to move is $(\frac{2}{3}N)$ corresponding to $m = \frac{2}{3}N$.*

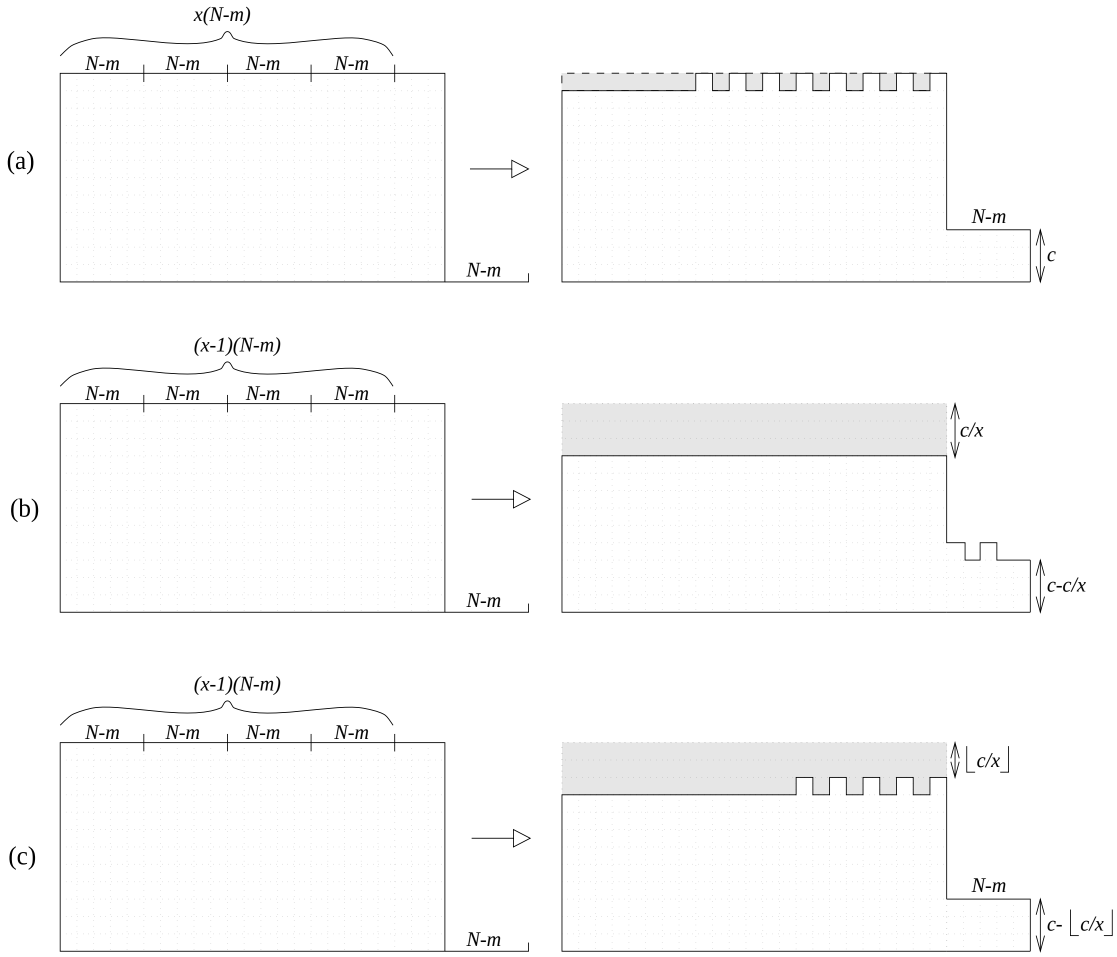

Figure 6: Cases for the Proof of Lemma 21

2. $x(N-m) > m \geq (x-1)(N-m)$ with $c \geq x > 2$ where x divides c evenly (see Figure 6 (b)). The number of tokens we are required to move from the maximal block to the minimal block before Δ is reduced by c are at most:

$$\frac{c}{x} \cdot m.$$

Therefore, since

$$\begin{aligned} m &< x(N-m) \\ &< \frac{x}{x+1}N \end{aligned}$$

the number of tokens we are required to move is $< \frac{c}{x} \cdot \frac{x}{x+1}N < \frac{c}{x+1}N$. Since $(x+1) \geq 4$, the number of tokens to be moved is $\leq \lceil \frac{c}{2} \rceil \cdot \frac{N}{2}$ for all c.

3. $\underline{x(N-m) > m \geq (x-1)(N-m)}$ with $c \geq x > 2$ where x does not divide c evenly (see Figure 6 (c)). The number of tokens we are required to move from the maximal block to the minimal block before Δ is reduced by c are at most:

$$c \cdot (N-m) - \lfloor \frac{c}{x} \rfloor \cdot (N-m).$$

Therefore, since

$$\begin{aligned} m &\geq (x-1)(N-m) \\ &\geq \frac{x-1}{x}N \end{aligned}$$

the number of tokens we are required to move is $\leq (c - \lfloor \frac{c}{x} \rfloor) \cdot \frac{1}{x}N$. For $x > 2$, the number of tokens to be moved is $\leq (\lceil \frac{(x-1)c}{x} \rceil) \cdot \frac{N}{x}$ which is $\leq \lceil \frac{c}{2} \rceil \cdot \frac{N}{2}$ for all c.

Note that the case $m = (N-m)$ is exactly equivalent to block distribution $(\frac{N}{2} : \frac{N}{2})$. Therefore, we have shown that for all $c \neq 2$ there does not exist a block distribution $(m : N-m)$, $(m > \frac{N}{2})$, such that the number of steps required to reduce Δ by c is greater than that required for $(\frac{N}{2} : \frac{N}{2})$. For $c = 2$, distribution $(\frac{2}{3}N : \frac{1}{3}N)$ gives that the worst-case number of steps to reduce Δ by 2 is $\frac{2}{3}N$ on the ring and $\frac{4}{3}N$ on the linear array. □

In summary, it has been shown that the time required by 1D-DEB to reduce Δ by some c is :

$$T_1(c) = \begin{cases} \lceil \frac{c}{2} \rceil \cdot N & \text{on the Linear Array, } c \neq 2, \\ \frac{4}{3} \cdot N & \text{on the Linear Array, } c = 2, \\ \lceil \frac{c}{2} \rceil \cdot \frac{N}{2} & \text{on the Ring, } c \neq 2, \\ \frac{2}{3} \cdot N & \text{on the Ring, } c = 2. \end{cases}$$

The analysis of 1D-DEB makes a significant contribution to the body of known dimension-exchange solutions to the token distribution problem. In addition, 1D-DEB is *flexible*; it can solve token distribution problems to within an arbitrary error δ of the average load λ. To date, all solutions to the token distribution problem on mesh connected architectures [9, 7, 3] have been proposed and analysed simply to solve the the problem $\delta = 1$.

4.4 Alternate Elementary Step

Let us now contrast the performance of 1D-DEB to a similar algorithm constructed using the alternate elementary step, step 2 from Table 1. An algorithm constructed using this step can be considered a *greedy* dimension-exchange algorithm since at every step, neighbouring processors attempt to evenly balance their local loads. Let us denote this algorithm Greedy1D-DEB.

From the proof of Lemma 4, it follows that Greedy1D-DEB can reduce any $\Delta > 1$ on the linear array or $\Delta > 2$ on the ring since the balancing step of 1D-DEB is simply a degenerate case of Greedy1D-DEB. Based on the proof, we can also argue that algorithm Greedy1D-DEB can at best equal the performance of 1D-DEB when reducing Δ by c, if we assume the cost of transferring k tokens between neighbours is k times the cost of transferring one token (which may indeed be the case when distributing equally sized bit patterns over massively parallel, SIMD architectures). If this communication cost can be assumed, then the running time of Greedy1D-DEB is equal to the number of tokens moved from the maximal block to the minimal block (as is the cost of 1D-DEB). To reduce the height of the maximal block by $c = 1$, many more tokens must be moved by Greedy1D-DEB than are moved by 1D-DEB, as shown in Figure 7. As c increases, the difference between the number of tokens moved by the two algorithms is reduced. When $c \in \Theta(\Delta)$, the two algorithms become comparable given that 1D-DEB must also move a significant proportion of tokens from the maximal block to the minimal block.

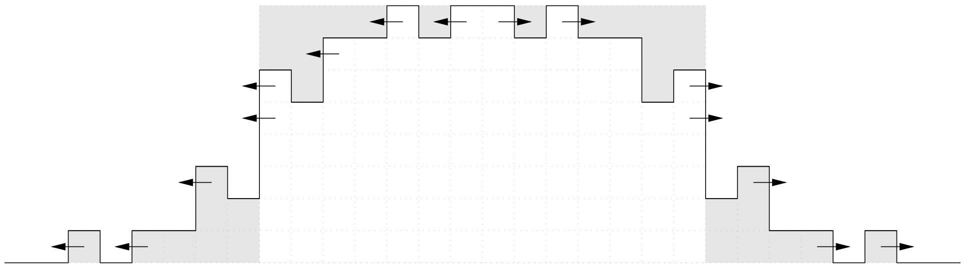

Figure 7: Reducing the Height of a Block using Greedy1D-DEB

5 Conclusion

In this paper we have presented a dimension-exchange algorithm which optimally solves the token distribution problem on the ting and linear array. Specifically, it was proven that the algorithm reduces the difference between the maximal processor load and minimal processor load by c in at most $\lceil \frac{c}{2} \rceil \cdot n$ on the linear array and $\lceil \frac{c}{2} \rceil \cdot \frac{n}{2}$ on the ring. This represents an improvement of a factor of 2 on the only other solution to be previously proposed for this problem.

References

[1] K. Batcher. Sorting networks and their applications. In *Proceedings AFIPS Spring Joint Conference*, volume 32, pages 307–314, 1968.

[2] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 7:279–301, 1989.

[3] D. Diderich and H. Gengler. An efficient algorithm for solving the token distribution problem on k-ary d-cube networks. Technical Report LITH-97, Swiss Federal Institute of Technology, Lausanne, Switzerland, March 1994.

[4] Houle H. and Turner G. Dimension exchange token distribution on the mesh and the torus. In *Proceedings of the International Symposium on Algorithms and Computation*, 1996.

[5] G. Horton. A multi-level diffusion method for dynamic load balancing. *Parallel Computing*, 19:209–218, 1993.

[6] S.H. Hosseini, B. Litow, M. Malkawi, J. McPherson, and K. Vairavan. Analysis of graph coloring based distributed load balancing algorithm. *Journal of Parallel and Distributed Computing*, 10:160–166, 1990.

[7] F. Meyer auf der Heide, B. Oesterdiekhoff, and R. Wanka. Strongly adaptive token distribution. In *Proceedings of the 20th ICALP*, pages 398–409, 1993.

[8] D. Peleg and E. Upfal. The generalised packet routing problem. *Theoretical Computer Science*, 53:218–293, 1987.

[9] C. G. Plaxton. Load balancing, selection and sorting on the hypercube. In *Proceedings of the ACM-SPAA*, pages 64–73, 1989.

[10] R. T. Plunkett and A. Symvonis. On the hot-potato permutation routing algorithm of borodin, rabani and schieber. In *Proceedings of Computing: The Australasian Theory Symposium (CATS '96)*, Jan 1996.

[11] S. Ranka, Y. Won, and S. Sahni. Programming a hypercube multicomputer. *IEEE Software*, pages 69–77, 1988.

[12] A. Roberts and A. Symvonis. On deflection worm routing on meshes. In *Proceedings of the First IEEE International Conference on Algorithms and Architecture for Parallel Processing (ICA3PP)*, pages 375–378, Apr 1995.

[13] A. Roberts, A. Symvonis, and L. Zhang. Routing on trees via matchings. *Lecture Notes in Computer Science*, (955):251–262, Aug 1995.

[14] G. Turner and H. Schröder. Token distribution on reconfigurable d-dimensional meshes. In *Proceedings of the 1st IEEE International Conference on Algorithms and Architectures for Parallel Processing*, volume 1, pages 335–344, 1995.

[15] C.Z. Xu and F.C.M. Lau. Analysis of the generalized dimension exchange method for dynamic load balancing. *Journal of Parallel and Distributed Computing*, 16:385–393, 1992.

[16] C.Z. Xu and F.C.M. Lau. The generalized dimension exchange method on some specific structures. Technical Report TR-92-02, University of Hong Kong, 1992.