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 Complete Binary Trees

**Ewan Tempero** and **Gavin Turner**

Technical Report CS-TR-96/22
November 1996

**Abstract**

Load balancing on a multi-processor systems involves shifting the work around the system so that each processor has about the same amount of work to do. The *token-distribution* problem is a static variant of load balancing for the case when the workloads in the system cannot be divided arbitrarily, where each token represents an atomic element of work. A simple method for distributing tokens is the so-called *dimension-exchange* approach, which is based on the repetitive application of an extremely simple and scalable local exchange protocol. The behaviour of this approach depends on the topology of the interconnection network.

This paper presents an analysis of the convergence properties of a dimension-exchange algorithm for token distribution on the complete binary tree. We show that for the height $H$ complete binary tree and any initial distribution for which the discrepancy in workloads is greater than $H$ tokens, the dimension-exchange approach leads to the eventual convergence of the distribution such that the discrepancy is at most $H$. These results are the first to establish that dimension-exchange techniques lead to useful algorithms for finitely-divisible load balancing on a tree-connected network.

**Publishing Information**

To appear: Proceedings of CATS '97 Sydney Australia.

## Author Information

**Ewan Tempero** received a Ph.D. in Computer Science from the University of Washington in 1990, and is currently is a lecturer in the Department of Computer Science at Victoria University of Wellington, New Zealand. His main research interest lies in examining how programming language features affect the production of code. His current focus is on the impact that programming languages, particularly object-oriented languages, have on the reusability of the resulting code. He is also interested in various aspects of distributed systems.

**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 Complete Binary Trees

*Ewan Tempero*
Department of Computer Science
Victoria University of Wellington
PO Box 600, Wellington, NZ
*ewan@comp.vuw.ac.nz*

*Gavin Turner*
Department of Computer Science
Victoria University of Wellington
PO Box 600, Wellington, NZ
*gavin@comp.vuw.ac.nz*

## Abstract

*Load balancing on a multi-processor systems involves shifting the work around the system so that each processor has about the same amount of work to do. The* token-distribution *problem is a static variant of load balancing for the case when the workloads in the system cannot be divided arbitrarily, where each token represents an atomic element of work. A simple method for distributing tokens is the so-called* dimension-exchange *approach, which is based on the repetitive application of an extremely simple and scalable local exchange protocol. The behaviour of this approach depends on the topology of the interconnection network.*

*This paper presents work in progress on the analysis of dimension-exchange algorithms for token distribution on the complete binary tree. We show that for the height $H$ complete binary tree and any initial distribution for which the discrepancy in workloads is greater than $H$ tokens, the dimension-exchange approach leads to the eventual convergence of the distribution such that the discrepancy is at most $H$. These results are the first to establish that dimension-exchange techniques lead to useful algorithms for finitely-divisible load balancing on a tree-connected network.*

**Keywords**
parallel architectures, load balancing, token distribution, dimension-exchange, binary tree network topology.

## 1 Introduction

One of the fundamental data distribution problems on parallel architectures is that of *token distribution*, a static variant of the well-studied load balancing problem. Each processing element (PE) of the parallel architecture possesses an initial set of *tokens*, each of which represents a task to be performed; the number of tokens stored at a particular PE is called the *load* of that PE. Ideally, one would prefer that the distribution of the tokens over the set of PEs be as even as possible, as imbalances would result in a delay in the time needed to perform all the tasks. The goal of a token distribution algorithm is to redistribute the tokens in such a way that the final loads of the PEs differ as little as possible. Here it is assumed that each token requires only a constant amount of time to send from one PE to an adjacent PE, and that no tokens are created or destroyed before the redistribution is complete.

There are many data distribution methods that achieve a balanced token distribution by gathering and making use of a certain amount of global information [2, 5, 9]. Such methods are often unsatisfactory, in that they do not take into account the practical limitations of the parallel architecture, or result in algorithms that are unnecessarily complex. One method that requires no such global information is the so-called *dimension-exchange* method, which is based on the repetitive application of an extremely simple and scalable local exchange protocol. To be able to implement a dimension-exchange algorithm on a particular parallel architecture, the communication edges of the underlying topology must be *partitionable* (or *colourable*) into sets whereby no two edges of the same set are incident on the same processor. For networks having hypercube or mesh-connected topologies, the edges can be partitioned in a natural fashion, according to the *dimension* of the network along which the edge is oriented. For other networks, partitions may be based on sets of *matchings* [3].

Dimension-exchange algorithms use the edge-colouring of a network to pair processors

**Proceedings of CATS'97 (Computing: The Australasian Theory Symposium), Sydney, Australia, February 5–7 1997.**

for data exchange, and are invariably of the following general form:

```
Dimension-Exchange Algorithm
LOOP
  FOR i = colour 1 to κ  (*κ colours *)
    Over all pairs of PEs connected by edges
    of colour i, compare and exchange values;
END
```

Due to their simplicity and scalability, many researchers have studied the applicability of dimension-exchange techniques to load balancing problems; the first being Cybenko [1] in 1987, who proposed an algorithm for the $d$-dimensional hypercube under the assumption that the load in each PE was *infinitely-divisible* — that is, a real-valued quantity able to be split among processors in an arbitrary fashion. Cybenko showed that if every exchange results in an equal sharing of the load between the two PEs involved, then after $d$ iterations the PE loads would be perfectly balanced.

This original work prompted a steady stream of research into the analysis of dimension-exchange algorithms. In 1990, Hosseini *et al.* demonstrated that, for infinitely-divisible loads, Cybenko's analysis could be generalised to arbitrary $k$-colourable networks [3]. In 1992 Xu and Lau [10] 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 optimised by altering the ratio with which infinitely-divisible loads were locally balanced. Later that year the same authors [11] computed optimal ratios for the linear array, ring, 2-dimensional mesh and 2-dimensional torus.

To date, a large body of results exist detailing the performance of the dimension-exchange approach over infinitely-divisible loads; on the other hand, little has been known concerning dimension-exchange under the more realistic assumption of *finitely-divisible* loads, that is, loads representable as a set of tokens.

In 1988 Ranka, Won, and Sahni [8] studied the operation of Cybenko's algorithm empirically for the $d$-dimensional hypercube assuming finitely-divisible loads. They observed that the difference between the maximum number and minimum number of tasks over all PEs of the network (called the *discrepancy*) would eventually fall to at most $d$. Hosseini *et al.* [3] and Plaxton [7] confirmed this observation by providing an algorithm that after $d$ steps reduced the discrepancy to at most $d$.

In this paper, we present work in progress that aims at analysing the performance of dimension-exchange algorithms for token distribution on the complete binary tree. To date, we have obtained a good understanding of the convergence properties of the algorithm presented herein. Further work is required to establish the rate of convergence, and therefore the potential optimality of the algorithm. For the height $H$ complete binary tree, we show that for any initial distribution for which the discrepancy is greater than $H$, the dimension-exchange approach leads to the eventual convergence of the distribution such that the discrepancy is at most $H$. These results are the first to establish that dimension-exchange techniques lead to useful algorithms for finitely-divisible load balancing on a tree-connected network.

The organisation of the paper is as follows: in the next section, we describe the model of computation. In Section 3, we propose a dimension-exchange algorithm for the complete binary tree. The notation and preliminary concepts that we use in the analysis of the algorithm is introduced in Section 4. The analysis of the convergence properties of the algorithm appears in Section 5, and a discussion of the rate of convergence is given in Section 6. Finally, concluding remarks are made in Section 7.

## 2 Model of Computation

We consider a parallel architecture in which the PEs are connected via an interconnection network based on the *complete binary tree* [4]. A complete binary tree of height $H$, $\mathcal{T}_H$, is a binary tree in which every node has either zero or two children, and all leaves are at distance $H$ from the root. The breadth-first numbering for such a tree numbers the root with 0, and for every node $i$, $0 \leq i < 2^H - 1$, its left subchild is numbered $2 \cdot i + 1$, and its right subchild is numbered $2 \cdot i + 2$. For every node $i$, $0 < i < 2^{H+1} - 1$, its parent is numbered $\lceil \frac{i}{2} \rceil - 1$. Node $i$ is said to exist at depth $\lfloor \log_2 (i+1) \rfloor$. Figure 1 shows a complete binary tree of height 3.

We assume that each PE has facilities for one-port communication. Under this assumption, the PEs are connected to their neighbours by uni-directional communication links, and may send or receive at most one message at any one time. This model is considerably weaker than the MIMD-model, where bi-directional links are assumed and concurrent communication to all the neighbours is allowed.

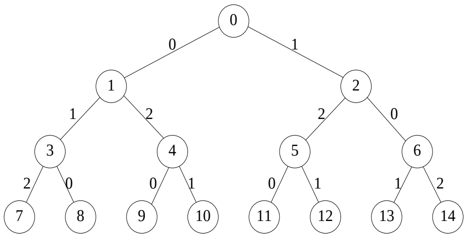


Figure 1: Depth-First Colouring of edges of the Height 3 Complete Binary Tree.

# 3 Token Distribution Problem and Algorithm

The token distribution problem was first posed by Peleg and Upfal ([6]) and can be stated as follows: $TD(A; \Delta, M, \delta)$ is the problem that, given parallel architecture $A$ containing $N$ processors $P_1, ..., P_N$, with each processor $P_i$ containing a stack of $\mu \leq l(P_i) \leq M$ tokens (for all $1 \leq i \leq N$) and for a global discrepancy between loads equal to $\Delta = M - \mu$, distribute the tokens such that at the end the global discrepancy has been reduced to at most $\delta$.

In this paper, we show that the dimension-exchange technique may be used to solve $TD(\mathcal{T}_H; \Delta, M, \delta)$ for $\delta, \Delta \geq H$; that is, under the algorithm we propose, the global discrepancy $\Delta$ in PE loads converges to $\delta$.

## 3.1 Dimension-Exchange Token Distribution Algorithm

To implement any dimension-exchange algorithm, two choices must be made: the network colouring over which the algorithm will be applied, and a suitable comparison-exchange step.

The algorithm presented in this paper, TreeBalance, is based on the so-called *Depth-First* colouring illustrated in Figure 1, in which the connections are consecutively coloured (mod 3) based on a depth-first traversal of the binary tree. All edges are coloured via the following recursive rule, where ColourTree($\mathcal{T}_H$,0) begins the process.

```
Colour Tree(Root, c);
BEGIN
   Root.left_child_edge.Colour = c;
   Root.right_child_edge.Colour = (c+1) mod 3;
   ColourTree(Root.left_subtree, (c+1) mod 3);
   ColourTree(Root.right_subtree, (c+2) mod 3);
END;
```

The local comparison-exchange step used by the algorithm can be informally stated as follows:

[Plus-Minus1]: Neighbouring PEs compare their loads and, if the loads differ, a *single token* is sent from the heavily-loaded PE to the lightly-loaded PE.

Depending on the architecture, implementing such an elementary operation may require several communication steps. Based on this elementary operation, algorithm TreeBalance is given below:

```
Algorithm TreeBalance
i = 0
REPEAT
BEGIN
        Apply elementary step [Plus-Minus1]
        over all network connections having
        colour i mod 3;
        i++;
END
```

For our analysis, it is convenient to assume that one iteration of this algorithm takes place in one unit of time.

In the remainder of the paper, we shall prove that under TreeBalance any initial distribution on the complete binary tree of height $H$ for which $\Delta > H$ converges to a distribution for which $\Delta \leq H$.

# 4 Notation and Preliminaries

In this section, we present notation and observations needed to analyse the convergence properties of TreeBalance over $\mathcal{T}_H$.

## 4.1 Tree Traversal

The operation of the algorithm over a complete binary tree of height 2 is illustrated in Figure 2, and can be described as follows. Consider the situation where the root node of tree $\mathcal{T}_H$ initially contains one token, as does the first leaf of the second subtree (node 5 in Figure 2), and all other nodes contain zero tokens. The algorithm begins by comparing and exchanging across edges of colour 0. In the example, the only such edge that has an imbalanced load is that between nodes 0 and 1 (there is no 0 coloured edge incident on node 5). So, on the first step, node 0 will reduce its

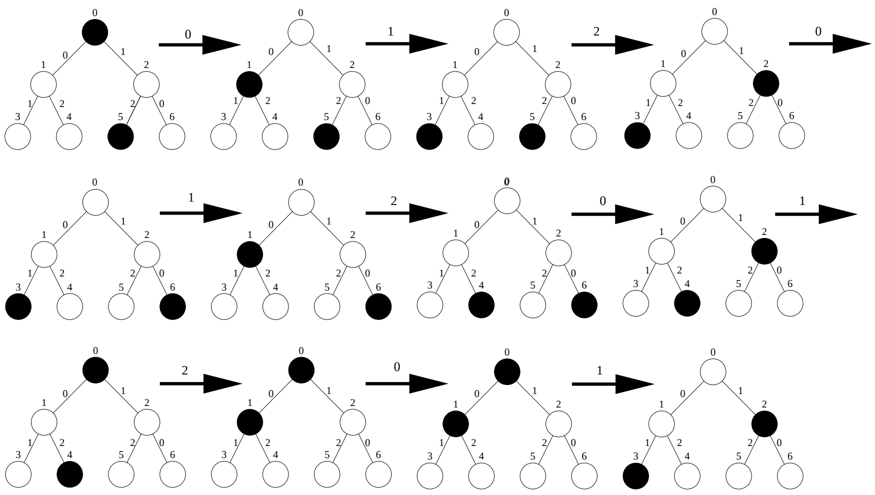

Figure 2: Algorithm TreeBalance Operating on a tree of Height 2.

load by 1 while node 1 will increase its load, resulting in the token *moving from node* 0 *to node* 1. On the second step (edges coloured 1), the token will move to node 3. On the third step, the token at node 3 has nowhere to go, since there is no edge coloured 2 incident on that node, however now the token at node 5 can move to node 2.

The token from the root will continue to move, traversing the tree in a depth-first fashion, until it has visited all nodes of the tree (at which point it arrives back at the root). Furthermore, each node in the tree (including the root) will have been visited exactly 3 times by the token[1]. We denote such a path as a *cycle* or *traversal* of the tree, and represents a path of exactly $3 \cdot N$ ($N = 2^{H+1} - 1$) steps.

So too, the token that begins at node 5 will traverse the tree, using exactly the same path as the root token, but *exactly 12 steps ahead* of it. This means that the token originating at the root will arrive at a particular node on a particular edge exactly 12 steps *after* the token originating from node 5. Conversely, this token may also be considered *exactly 9 steps behind* the root token (since the traversal is a cycle). In addition, the 10th step of the algorithm sees the current node containing the token from the root, being compared to the node containing the token from node 5. Now the outcome of this comparison is such that the loads of two nodes remain the same - no tokens are exchanged. However, if the two tokens of interest are to finish their traversal, it is necessary for us to imagine that they have *swapped position* - in essence we imagine that the tokens have moved through each other!

In general, two such tokens that begin at different nodes remain a constant distance apart with respect to the traversal. This observation is more obvious if we represent the complete tree traversal by a token as a cyclic graph with $3 \cdot (2^{H+1} - 1)$ edges and $3 \cdot (2^{H+1} - 1)$ nodes. Such a graph is shown in Figure 3 for a tree of height 3. Note that each node appears exactly 3 times in the traversal, and each edge appears exactly twice and dummy edges exist corresponding to delays at the leaves and root. The positions at which a token may start (those which are incident on colour 0) have been darkly shaded. Note that the distance between tokens is always a positive multiple of 3. This is because a token arrives at a particular node on a particular edge. Since this edge has some colour $c$, tokens may only arrive at this node, on this edge, every 3rd step.

In general, the number of steps required to traverse a height $h$ subtree (from the root of the subtree and back again) is given by Lemma 1 below.

**Lemma 1** *Consider a subtree $\mathcal{S}$ of height $h$ at time $t$ for which the colour $c$ of the root's left child edge corresponds to the colour to be used in step $t+1$ of algorithm* TreeBalance*. A token that exists at the root node of $\mathcal{S}$ at time $t$ will return to the root after exactly $3 \cdot 2^{h+1} - 4$ steps.*

**Proof** We know from the traversal that each node in the subtree will be visited exactly three times by the token. Since there are $2^{h+1} - 1$ nodes in a height $h$ complete binary tree, this requires exactly $3 \cdot 2^{h+1} - 3$ steps.

[1]Note that at the root and the leaves tokens are delayed 1 and 2 steps respectively. In this context, each step that a token is delayed at a node is identified as another visit to that node.

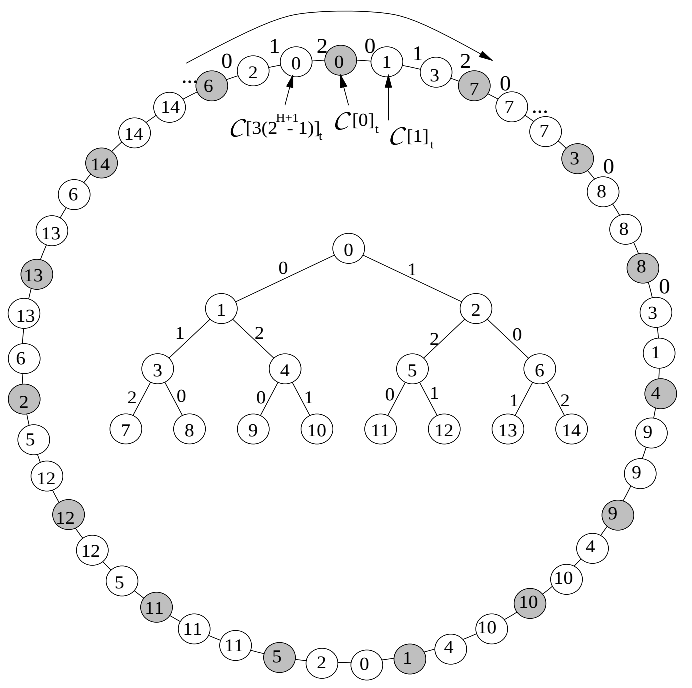

Figure 3: Traversal illustrated for a Complete Binary Tree of Height 3.

However, since we began at the root of the subtree, the root is only *re-visited* once and so the actual number of steps is actually $3 \cdot (2^{h+1} - 1) - 1 \ = \ 3 \cdot 2^{h+1} - 4$. $\square$

Instead of viewing the positions of tokens as being tied to *absolute* (fixed) locations of $\mathcal{T}_H$, it will often be convenient to view their positions *relative* to a position in a *traversal* of the tree. The next subsection establishes the relationship between the absolute (referring to tree nodes) and the relative (referring to cycle locations) interpretations of Algorithm TreeBalance, and the following subsection examines the behaviour of the tokens under the relative interpretation.

### 4.2 Cycle Traversal

As stated above, we can represent the path taken by a token that traverses the tree as a cyclic graph as shown in Figure 3. Under the relative interpretation of algorithm TreeBalance, we imagine a set of $N$ *piles* of tokens, (each a group of zero or more), which circulate around the tree in a manner similar to the single tokens we tracked in the preceding section. At any time instant each node $i$, $0 \leq i < N$, of tree $\mathcal{T}_H$ is the *location* of a particular pile $\alpha$, denoted $i = \mathcal{L}[\alpha]_t$. At each time step, the pile moves to the next node in the cyclic traversal, and the size of the pile changes depending on the outcome of the associated comparison-exchange operation.

The number of tokens in pile $\alpha$ after $t$ steps of TreeBalance will be called its *size*, and will be denoted by $\mathcal{S}[\alpha]_t$. The number of tokens at node $j$ at time $t$ (that is, under the absolute interpretation) is the node's *height*, denoted $\mathcal{H}[j]_t$.

Initially we have, for any pile $\alpha$, $\mathcal{S}[\alpha]_0 = \mathcal{H}[\mathcal{L}[\alpha]_0]_0$. If a pile has size that is minimum or maximum over all positions of $\mathcal{T}_H$, we simply refer to it as a *minimum* or *maximum* respectively. Let $min(t)$ and $max(t)$ be the set of minima and maxima at time $t$.

In addition to being associated with a node of the tree at a particular time, (its location), a pile $\alpha$ is associated with a node in the traversal of the tree. This node is denoted the *cycle position* of the pile $\alpha$, and is denoted $\mathcal{C}[\alpha]_t$.

Of interest to our analysis will be the relative distance between piles with respect to the traversal. This distance is denoted the *cycle distance* and represents the number of steps separating piles in the traversal of the tree. Note that for each node in the tree, there are 3 possible cycle positions at which a pile may exist. However, only one such cycle position will be occupied by a pile at any one time.

With respect to relative locations, the comparison-exchange rule of TreeBalance can be reinterpreted. Consider nodes $i$ and $j$ joined by colour $c$ at time $t$. If $t \bmod 3 = c$ , then the loads of nodes $i$ and $j$ are compared and tokens are possibly exchanged. Whenever the piles $\alpha$ and $\beta$ at locations $i$ and $j$ respectively are compared at time $t$, the new sizes $\mathcal{S}[\alpha]_t$ and $\mathcal{S}[\beta]_t$ are recalculated as follows:

if $\mathcal{S}[\alpha]_{t-1} > \mathcal{S}[\beta]_{t-1} + 1$ then
  $\mathcal{S}[\alpha]_t = \mathcal{S}[\beta]_{t-1} + 1;$
  $\mathcal{S}[\beta]_t = \mathcal{S}[\alpha]_{t-1} - 1;$
else if $\mathcal{S}[\alpha]_{t-1} + 1 < \mathcal{S}[\beta]_{t-1}$ then
  $\mathcal{S}[\alpha]_t = \mathcal{S}[\beta]_{t-1} - 1;$
  $\mathcal{S}[\beta]_t = \mathcal{S}[\alpha]_{t-1} + 1;$
else
  $\mathcal{S}[\beta]_t = \mathcal{S}[\alpha]_{t-1};$
  $\mathcal{S}[\alpha]_t = \mathcal{S}[\beta]_{t-1};$
endif

Note also that the locations of piles $\alpha$ and $\beta$ at time $t$ are $\mathcal{L}[\beta]_t = \mathcal{L}[\alpha]_{t-1}$ and $\mathcal{L}[\alpha]_t = \mathcal{L}[\beta]_{t-1}$ respectively – i.e., piles $\alpha$ and $\beta$ have swapped locations! An example of this operation is illustrated in Figure 4.

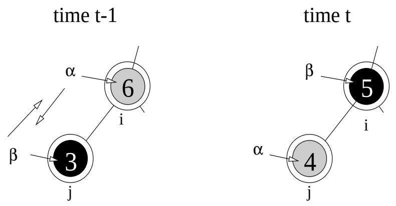


Figure 4: Comparison Exchange operation between neighbouring nodes leads to the relocation of respective piles.

### 4.3 Maxima and Minima

Now consider the situation in which, for some pile $\alpha$, $\mathcal{S}[\alpha]_{t-1} = M$, the maximum possible size. If at step $t$, pile $\alpha$ swaps with pile $\beta$, we distinguish two outcomes. If $\mathcal{S}[\beta]_{t-1} \geq M-1$, then the rules outlined above indicate that after the swap, $\mathcal{S}[\beta]_t = \mathcal{S}[\beta]_{t-1} = M$ and $\mathcal{S}[\alpha]_t = \mathcal{S}[\alpha]_{t-1}$. Otherwise, if $\mathcal{S}[\beta]_{t-1} < M-1$, the greatest value possible for $\mathcal{S}[\alpha]_t$ and $\mathcal{S}[\beta]_t$ is $M-1$. Similarly, if $\mathcal{S}[\alpha]_{t-1} = \mu$, the minimum possible size, we have two cases: if $\mathcal{S}[\beta]_{t-1} \leq \mu+1$ then $\mathcal{S}[\alpha]_t = \mathcal{S}[\alpha]_{t-1} = \mu$ and $\mathcal{S}[\beta]_t = \mathcal{S}[\beta]_{t-1}$; if $\mathcal{S}[\beta]_{t-1} > \mu+1$ then $\mathcal{S}[\alpha]_t$ and $\mathcal{S}[\beta]_t$ must be at least $\mu+1$. Based on these swaps, we are led to the following observation:

**Observation 2** *Let $M$ and $\mu$ be the maximum and minimum size of the piles of $\mathcal{T}_H$ at time $t_0$. For any pile $\alpha$, if $\mathcal{S}[\alpha]_{t_0} < M$, then $\mathcal{S}[\alpha]_t < M$ for all $t > t_0$. Also, if $\mathcal{S}[\alpha]_{t_0} > \mu$, then $\mathcal{S}[\alpha]_t > \mu$ for all $t > t_0$.*

In other words, at any particular time instant, maxima and minima can be destroyed but never created. Maxima and minima are destroyed when they are swapped with piles of height $< M-1$ and $> \mu+1$ respectively, as observed above. This phenomena is illustrated in Figure 5. For each maxima and minima therefore, there exists a set of piles that pose it a *threat*; these piles *threaten* the maxima or the minima respectively.

In the analysis to follow, we characterise the set of piles that threaten a particular maxima or minima with respect to relative cycle positions. This characterisation leads us to formally establish the maximum $\Delta$ for which neither elements of $max(t)$ nor elements of $min(t)$ are threatened. This value of $\Delta$ represents a guaranteeable limit of convergence for the algorithm.

## 5 Analysis of TreeBalance

In this section, Algorithm TreeBalance is proven to converge to $\Delta \leq H$ for complete binary trees of height $H$. The convergence of Algorithm TreeBalance is stated and proven in Theorem 5 at the end of this section. We now present several lemmas that are required to complete the proof.

**Lemma 3** *Let $\mathcal{T}_H$ be a height $H$ complete binary tree whose elements are piles of non-negative size. After $3 \cdot (2^{H+1} - 1)$ steps of Algorithm* TreeBalance *on $\mathcal{T}_H$, a pile $\alpha$, $0 \leq \alpha < N$, will be swapped and only swapped, with piles $\beta$, $0 \leq \beta < N$, such that $\mathcal{C}[\alpha]_{t_0} - \mathcal{C}[\beta]_{t_0} = \pm(3 \cdot 2^k - 1)$, $1 \leq k < H+1$.*

**Proof** Assume that pile $\alpha$ is at a node $i = \mathcal{L}[\alpha]_t$ at height $h$ in $\mathcal{T}_H$ at some time $t$. There are three cases to consider: in the next step, (step $t+1$), $\alpha$ will be either compared with the pile at the node's left child (at height $h-1$), be compared with the pile at the node's right child (at height $h-1$), or be compared with the pile at the node's parent (at height $h+1$).

Let $\beta$ be the pile that $\alpha$ is compared with at step $t+1$. In the first case, since $\alpha$ is being compared with $i$'s left child, $\beta$ must have completed a traversal of the left subtree. That is, $\beta$ was at node $i$ at some time in the past, and since that time, it has travelled to $i$'s left child, has gone around the subtree rooted at $i$'s left child, and is now back at $i$'s left child. Since the left subtree has height $h-1$, Lemma 1 gives that this must have taken $1+3\cdot 2^h - 4 = 3 \cdot (2^h - 1)$ steps, meaning that $\beta$ is $3 \cdot (2^h - 1)$ steps ahead of $\alpha$.

In the second case, $\alpha$ must have arrived at $i$ from $i$'s left child, and $\beta$ has just completed a traversal of $i$'s right subtree. By the same argument as above applied to the right

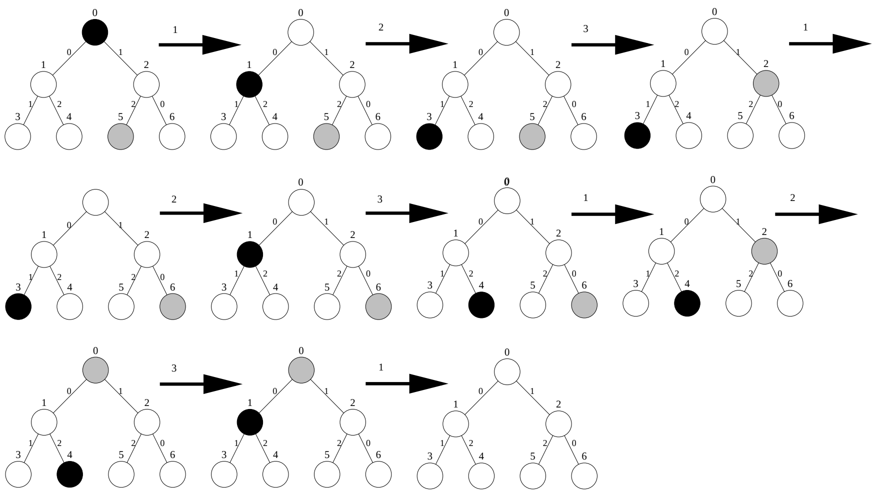

Figure 5: A *Threat* (Lightly Shaded) of cyclic distance 12 ahead of a *Maximum* (Darkly Shaded) is swapped with the Maximum and destroys it.

subtree of $i$, $\beta$ must be $3 \cdot (2^h - 1)$ steps ahead of $\alpha$.

In the third case, $\alpha$ has completed a traversal of the tree rooted at $i$ and is returning to its parent. This is equivalent to the first case, except that $\beta$ is located at $i$'s parent node at height $h+1$ and is being compared with $\alpha$ at $i$'s left child. By the same argument, $\beta$ must be $3 \cdot (2^{h+1} - 1)$ steps *behind* $\alpha$.

Thus, by these arguments, $\alpha$ and $\beta$ may be distances $3 \cdot (2^h - 1)$ or $3 \cdot (2^{h+1} - 1)$ steps apart, where $0 \le h \le H$. However note that if $i$ is a leaf (height 0) then it will only be compared with its parent (height 1) and if $i$ is the root (height H) it will only be compared with its children (height H-1) so in fact $\alpha$ and $\beta$ may only be distances $3 \cdot (2^h - 1)$ apart, where $1 \le h < H+1$, which proves the claim. □

Lemma 3 identifies the piles with which a particular pile $\alpha$ at cycle location $\mathcal{C}[\alpha]_t$ will be compared in the next $3 \cdot (2^{H+1} - 1)$ steps of the algorithm. These piles are all candidate *threats* for pile $\alpha$. We denote such a set of threats as *enemies* and denote the set of enemies for $\alpha$ as $\mathcal{E}(\alpha)$. It follows from lemma 3 that the size of $\mathcal{E}(\alpha)$ is $2H$.

The following lemma will be used in the theorem to show how threats are propagated around the cycle.

**Lemma 4** *Let $H$ be any positive integer. Then for any $k$ such that $1 \le k < 2^{H+1} - 1$ there exists $c_i$ where $1 \le i \le H$ and $\sum_{i=1}^{H} |c_i| \le H$, such that either*

$$k = \sum_{i=1}^{H} c_i(2^i - 1) \text{ or } k = 2^{H+1} - 1 - \sum_{i=1}^{H} c_i(2^i - 1).$$

**Proof** The lemma is similar to one that says that $k$ can be represented as a sum of powers of some radix. Its proof is complicated by the fact that it is not quite sums of powers. The proof is by induction on $H$.

**Basis:** Consider the case when $H = 1$. Then since $1 \le k < 2^2 - 1$ it must be that $k \in \{1, 2\}$. For $k = 1$ the lemma is satisfied for $c_1 = 1$, giving $\sum_{i=1}^{1} c_i(2^i - 1) = 2^1 - 1 = k$. For $k = 2$, the lemma is satisfied for $c_1 = 1$, giving $2^{H+1} - 1 - \sum_{i=1}^{1} c_i(2^i - 1) = (2^2 - 1) - (2^1 - 1) = k$.

**Inductive Hypothesis:** Assume the claim is true for $H = l - 1$, that is, for $k$, $1 \le k < 2^l - 1$, there exist $d_i$, $1 \le i \le l-1$ and $\sum_{i=1}^{l-1} |d_i| \le l - 1$ such that either $k = \sum_{i=1}^{l-1} d_i(2^i - 1)$ or $k = 2^l - 1 - \sum_{i=1}^{l-1} d_i(2^i - 1)$.

**Induction Step:** Consider the case when $H = l$ and so consider any $k$, $1 \le k < 2^{l+1} - 1$. Then there are the following cases:

1. $1 \le k < 2^l - 1$: Then by the induction hypothesis, either $k = \sum_{i=1}^{l-1} d_i(2^i - 1)$ or $k = 2^l - 1 - \sum_{i=1}^{l-1} d_i(2^i - 1)$.

   (a) $k = \sum_{i=1}^{l-1} d_i(2^i - 1)$: Then let $c_l = 0$ and $c_i = d_i$ where $1 \le i \le l - 1$, in which case $\sum_{i=1}^{l} |c_i| = \sum_{i=1}^{l-1} |d_i| + 0 \le l - 1 \le l$ and $\sum_{i=1}^{l} c_i(2^i - 1) = \sum_{i=1}^{l-1} d_i(2^i - 1) = k$.

   (b) $k = 2^l - 1 - \sum_{i=1}^{l-1} d_i(2^i - 1)$: Then let $c_l = 1$ and $c_i = -d_i$ where $1 \le i < l$, in which case $\sum_{i=1}^{l} |c_i| = \sum_{i=1}^{l-1} |d_i| + 1 \le (l-1) + 1 \le l$ and $\sum_{i=1}^{l} c_i(2^i - 1) = c_l(2^l -$

$1) + \sum_{i=1}^{l-1} c_i(2^i - 1) = 2^l - 1 - \sum_{i=1}^{l-1} d_i(2^i - 1) = k$.

2. $k = 2^l - 1$: Then let $c_l = 1$ and $c_i = 0$ where $1 \le i < l$, in which case $\sum_{i=1}^{l} |c_i| = 1 \le l$ and $\sum_{i=1}^{l} c_i(2^i - 1) = 2^l - 1 = k$.

3. $2^l - 1 < k < 2^{l+1} - 2$: Let $j = k - (2^l - 1)$, then $1 \le j < 2^l - 1$, and so by the induction hypothesis either $j = \sum_{i=1}^{l-1} d_i(2^i - 1)$ or $j = 2^l - 1 - \sum_{i=1}^{l-1} d_i(2^i - 1)$.

   (a) $j = \sum_{i=1}^{l-1} d_i(2^i - 1)$: Then let $c_l = 1$ and $c_i = d_i$ where $1 \le i < l$, in which case $\sum_{i=1}^{l} |c_i| = \sum_{i=1}^{l-1} |d_i| + 1 \le (l-1) + 1 \le l$ and $\sum_{i=1}^{l} c_i(2^i - 1) = c_l(2^l - 1) + \sum_{i=1}^{l-1} c_i(2^i - 1) = 2^l - 1 + \sum_{i=1}^{l-1} d_i(2^i - 1) = 2^l - 1 + j = k$.

   (b) $j = 2^l - 1 - \sum_{i=1}^{l-1} d_i(2^i - 1)$: Then let $c_i = d_i, 2 \le i \le l-1$ and $c_1 = d_1 + 1$, in which case $\sum_{i=1}^{l} |c_i| \le \sum_{i=1}^{l-1} |d_i| + 1 \le (l-1) + 1 \le l$ and $2^{l+1} - 1 - \sum_{i=1}^{l} c_i(2^i - 1) = 2^{l+1} - 1 - (\sum_{i=2}^{l-1} d_i(2^i - 1) + (d_1 + 1)(2^1 - 1)) = 2^{l+1} - 1 - (\sum_{i=1}^{l-1} d_i(2^i - 1) + 1) = 2^l - 1 + 2^l - 1 - \sum_{i=1}^{l-1} d_i(2^i - 1) = 2^l - 1 + j = k$

4. $k = 2^{l+1} - 2$: Then let $c_1 = 1$ and $c_i = 0$, where $2 \le i \le l$, in which case $\sum_{i=1}^{l} |c_i| = 1 \le l$ and $2^{l+1} - 1 - \sum_{i=1}^{l} c_i(2^i - 1) = 2^{l+1} - 1 - (2^1 - 1) = 2^{l+1} - 1 - 1 = k$.

□

Finally we are in a position to prove the convergent property of Algorithm TreeBalance. This is accomplished through Theorem 5 below.

**Theorem 5** *Under algorithm* TreeBalance, *any initial distribution on a binary tree $\mathcal{T}_H$ of height $H$ for which $\Delta > H$ will eventually converge to a distribution for which $\Delta \le H$.*

**Proof** To establish a convergent token distribution, it must be the case that for a complete traversal of the tree, no pile changes size. Once this state has been reached, no pile will *ever* change size after that time.

For a pile's size to remain unchanged through a complete traversal, its size and the sizes of its enemies must differ by at most $\pm 1$. If at some time $t$ this condition is not satisfied for some maxima then either the maxima will be destroyed (will have height $< M$, or *another* pile will have its size reduced. In either case, the net effect is to drive the distribution towards the average.

Consider an arbitrary pile $\alpha$ at time $t$. To maintain a steady-state distribution, the members $\beta \in \mathcal{E}(\alpha)$ must have height $\mathcal{S}[\alpha]_t - 1 \le \mathcal{S}[\beta]_t \le \mathcal{S}[\alpha]_t + 1$. Similarly, for any of the enemies $\eta$ of $\beta$, $\gamma \in (\bigcup_{\eta \in \mathcal{E}(\alpha)} \mathcal{E}(\eta))$ a similar condition applies: $\mathcal{S}[\beta]_t - 1 \le \mathcal{S}[\gamma]_t \le \mathcal{S}[\beta]_t + 1$. This implies that $\mathcal{S}[\alpha]_t - 2 \le \mathcal{S}[\gamma]_t \le \mathcal{S}[\alpha]_t + 2$.

Informally, this proof determines how large the union of enemies of enemies of enemies etc can be before it must be the case that $\alpha$ is a member of the union. This determines by how much $\alpha$ may vary from the other piles of the tree whilst maintaining a steady-state distribution.

For any $\beta \in \mathcal{E}(\alpha)$ and $\gamma \in \mathcal{E}(\beta)$, consider how far apart $\alpha$ and $\gamma$ are, that is, consider the size of $\mathcal{C}[\gamma]_t - \mathcal{C}[\alpha]_t$.

Lemma 3 says that for some $l_\alpha$, $1 \le l_\alpha < H + 1$, $\mathcal{C}[\beta]_t - \mathcal{C}[\alpha]_t = \pm 3 \cdot (2^{l_\alpha} - 1)$. If $\mathcal{C}[\beta]_t - \mathcal{C}[\alpha]_t = 3 \cdot (2^{l_\alpha} - 1)$ then $\beta$ is $3 \cdot (2^{l_\alpha} - 1)$ steps away from $\alpha$ in a *clockwise* direction. If $\mathcal{C}[\beta]_t - \mathcal{C}[\alpha]_t = -3 \cdot (2^{l_\alpha} - 1)$ then $\beta$ is $3 \cdot (2^{l_\alpha} - 1)$ steps in an *anti-clockwise* direction, or, since the size of the cycle is $3 \cdot (2^{H+1} - 1)$, $\beta$ is $3 \cdot (2^{H+1} - 1) - 3 \cdot (2^l - 1)$ steps in a clockwise direction away from $\alpha$.

Since it is also the case that, for some $l_\beta$, $1 \le l_\beta < H + 1$, $\mathcal{C}[\gamma]_t - \mathcal{C}[\beta]_t = \pm 3 \cdot (2^{l_\beta} - 1)$, it follows that $\mathcal{C}[\gamma]_t - \mathcal{C}[\alpha]_t = \pm 3 \cdot (2^{l_\alpha} - 1) \pm 3 \cdot (2^{l_\beta} - 1)$. This gives four possibilities, corresponding to whether the piles are clockwise or anti-clockwise from each other. If in fact $\mathcal{C}[\gamma]_t - \mathcal{C}[\alpha]_t = -3 \cdot (2^{l_\alpha} - 1) - 3 \cdot (2^{l_\beta} - 1)$ (that is, the anti-clockwise direction in both cases) then we can rewrite this as $\mathcal{C}[\gamma]_t - \mathcal{C}[\alpha]_t = 3 \cdot (2^{H+1} - 1) - 3 \cdot (2^{l_\alpha} - 1) + 3 \cdot (2^{H+1} - 1) - 3 \cdot (2^{l_\beta} - 1) = 2^{H+1} - 1 - (3 \cdot (2^{l_\alpha} - 1) + 3 \cdot (2^{l_\beta} - 1))$, since $2 \cdot 3 \cdot (2^{H+1} - 1)$ is really just twice around the cycle, and so is no different from once around the cycle. Furthermore, we can rewrite $3 \cdot (2^{l_\alpha} - 1) + 3 \cdot (2^{l_\beta} - 1)$ as $3 \cdot \sum_{i=1}^{H} c_i(2^i - 1)$ where $\sum_{i=1}^{H} |c_i| \le 2$ ($c_i = 2$ in the case where $l_\alpha = l_\beta$). The other two cases are similar, and thus, we can represent $\mathcal{C}[\gamma]_t - \mathcal{C}[\alpha]_t$ as either $3 \cdot \sum_{i=1}^{H} c_i(2^i - 1)$ or $3 \cdot (2^{H+1} - 1 - \sum_{i=1}^{H} c_i(2^i - 1))$ where $\sum_{i=1}^{H} |c_i| \le 2$.

Now consider any $\zeta \in \mathcal{E}(\gamma)$, that is, $\zeta$ is an enemy "thrice-removed" from $\alpha$ (an enemy of an enemy of an enemy of $\alpha$). By an argument similar to that above, we can represent $\mathcal{C}[\zeta]_t -$

$\mathcal{C}[\alpha]_t$ as either $3 \cdot \sum_{i=1}^{H} c_i(2^i - 1)$ or $3 \cdot (2^{H+1} - 1 - \sum_{i=1}^{H} c_i(2^i - 1))$ where $\sum_{i=1}^{H} |c_i| \leq 3$. Thus in general, for an enemy "R-removed" from $\alpha$, say $\epsilon$, $\mathcal{C}[\epsilon]_t - \mathcal{C}[\alpha]_t$ is either $3 \cdot \sum_{i=1}^{H} c_i(2^i - 1)$ or $3 \cdot (2^{H+1} - 1 - \sum_{i=1}^{2} c_i(2^i - 1))$ where $\sum_{i=1}^{H} |c_i| \leq R$.

As noted above, we need to determine an upper bound on $R$ that ensures that $\alpha$ must appear again in one of the enemy sets. In fact, what we will show is an upper bound on $R$ that ensures that *every* pile appears in one of the enemy sets. The theorem claims that in fact $R = H$ is sufficient.

The cyclic distance between enemies is always a multiple of 3 since, as noted in section 4.1, the distance between any pair of piles is a multiple of 3. Thus, assuming a particular pile $\alpha$ is at cycle position 0 at some time $t$ ($\mathcal{C}[\alpha]_t = 0$), all the other piles will be at cycle positions $3 \cdot k, 1 \leq k \leq 2^{H+1} - 1$. Thus, to ensure that every pile appears in one of the enemy sets in a union of $H$ enemy sets, it is sufficient that every cycle position $k$, $1 \leq k \leq 2^{H+1} - 1$ can be represented by either $\sum_{i=1}^{H} c_i(2^i - 1)$ or $2^{H+1} - 1 - \sum_{i=1}^{2} c_i(2^i - 1)$ where $\sum_{i=1}^{H} |c_i| \leq H$. Since this is exactly what lemma 4 says, this completes the proof. $\square$

## 6 Convergence of TreeBalance

At this stage, the *rate of convergence* is yet to be determined. For this reason, a lower bound on the time required to solve problem *TD*$(\mathcal{T}_H; \Delta, M, \delta)$ is not strictly required for our analysis. However, we include it below for the sake of completeness.

For a complete binary tree $\mathcal{T}_H$ of height $H$, a lower bound on the time required to solve token distribution problem *TD*$(\mathcal{T}_H; \Delta, M, \delta)$ is easily derived. The result is stated in Lemma 6.

**Lemma 6** *Let $\mathcal{T}_H$ be a complete binary tree of height $H$, and $N = 2^{H+1} - 1$ nodes. Any solution to problem* $\mathrm{TD}(\mathcal{T}_H; \Delta, M, \delta)$ *requires time in* $\Omega((\Delta - \delta) \cdot N)$.

**Proof** The proof is a simple bisection-width argument. Assume a worst-case load in which the $(N-1)/2$ PEs in the left subtree of $\mathcal{T}_H$ contain $M$ tokens, and the remaining $(N-1)/2$ PEs in the right subtree contain $\mu$ tokens. To reduce $\Delta$ to $\delta$, $\frac{\Delta - \delta}{2} \cdot \frac{(N-1)}{2}$ tokens must be moved from the PEs with $M$ tokens to the PEs with $\mu$ tokens. Given that the bisection width of $\mathcal{T}_H$ is $\mathcal{O}(1)$, this requires at least $\Omega((\Delta - \delta) \cdot N)$ steps. $\square$

An upper bound on the rate of convergence is less clear. At present, we are working on the following conjecture.

**Conjecture 7** *Let $\mathcal{T}_H$ be a complete binary tree of height $H$, and $N = 2^{H+1} - 1$ nodes. Then there exists a solution to problem* $\mathrm{TD}(\mathcal{T}_H; \Delta, M, \Delta - 1)$ *for $\Delta > H$, that takes time at most $3 \cdot (2^{H+1} - 1) \cdot H$ steps.*

**Proof Overview** The idea is that in this situation a maxima $\alpha$ will eventually be compared to all other values. In fact, the proof for lemma 3 shows it will be compared with all of its enemies after a "phase" of $3 \cdot (2^{H+1} - 1)$ steps. If there is a threat in the maxima's set of enemies, then the maxima will be destroyed. Now consider an enemy of $\alpha$, $\epsilon \in \mathcal{E}(\alpha)$. If one of $\epsilon$'s enemies is a threat to the maxima, then after a phase $\epsilon$ can become the threat, and so after the next phase, the maxima will be destroyed. Lemma 4 essentially says that a threat at any start position will eventually move through the sets of enemies until it is reaches the maxima's set of enemies.

By lemma 3, whenever a threat moves from one set of enemies to another, it must move a distance of $\pm 3 \cdot (2^j - 1)$, for $1 \leq j < H + 1$. Thus the distance that a threat has moved after $p$ moves is $d = 3 \cdot \sum_{i=1}^{H} c_i(2^i - 1)$, where $\sum_{i=1}^{H} |c_i| \leq p$. If the threat starts at distance $3k$ from the maxima, and there is a $p$ and a vector $[c_i]$, $\sum_{i=1}^{H} |c_i| \leq p$ such that $k = \sum_{i=1}^{H} c_i(2^i - 1)$, then the threat will eventually reach the maxima. Lemma 4 says that such a vector will always exist when $p \geq H$, that is, provided the number of sets of enemies the threat has to go through is at least $H$.

Thus the argument is that the threat will go through at most $H$ sets, with each phase taking $3 \cdot (2^{H+1} - 1)$ steps, giving the claimed result. $\square$

The problem with this argument is it does not deal with the case where a threat may be destroyed by one of its own threats before it reaches the maxima. Thus a threat may not always make progress on each phase. However we have developed simulations of the algorithm that seem to support this conjecture.

## 7 Conclusion

In this paper we have introduced a new, dimension-exchange algorithm for token distribution on complete binary trees. The benefits of the dimension-exchange approach, in that it is extremely simple, uses only locally-available information and is completely scalable, cannot be overstated. We have presented an analysis of the algorithm which indicates that it does indeed provide a solution to $TD(\mathcal{T}_H; \Delta, M, \delta)$ for $\Delta, \delta \geq H$. This gives an upper bound on possible values of $\delta$.

We know of limits of convergence for particular values of $H$ in which $\delta < H$. We conjecture that in general, there is a solution if and only if $\Delta, \delta \geq \lfloor \frac{H}{2} \rfloor$. Proving this will require proving a version of Lemma 4 with a bound lower than $H$.

We also gave a lower bound on the time for convergence of $TD(\mathcal{T}_H; \Delta, M, \delta)$ in $\Omega((\Delta - \delta) \cdot N)$ steps and a conjecture as to the upper bound on the rate of convergence. It remains for us to prove our conjecture. This will require further research into the fundamental relationships between enemy sets and the way in which piles interact in the traversal.

## References


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

[2] 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.

[3] 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*, Volume 10, pages 160–166, 1990.

[4] F. Thompson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, San Mateo, California, 1992.

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

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

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

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

[9] 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.

[10] 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*, Volume 16, pages 385–393, 1992.

[11] 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.