DATA REDISTRIBUTION ALGORITHMS FOR HETEROGENEOUS PROCESSOR RINGS

Hélène Renard
Yves Robert
Frédéric Vivien

LIP, UMR CNRS–INRIA–UCBL 5668, ENS LYON, FRANCE (YVES.ROBERT@ENS-LYON.FR)

Abstract

We consider the problem of redistributing data on homogeneous and heterogeneous rings of processors. The problem arises in several applications, after each invocation of a load-balancing mechanism (but we do not discuss the load-balancing mechanism itself). We provide algorithms that aim at optimizing the data redistribution, both for unidirectional and bidirectional rings. One major contribution of the paper is that we are able to prove the optimality of the proposed algorithms in all cases except that of a bidirectional heterogeneous ring, for which the problem remains open.

Key words: heterogeneous rings, data redistribution algorithms, load balancing

The International Journal of High Performance Computing Applications,
Volume 20, No. 1, Spring 2006, pp. 31–43
DOI: 10.1177/1094342006061887
© 2006 SAGE Publications

1 Introduction

In this paper, we consider the problem of redistributing data on a heterogeneous ring of processors. The problem typically arises when a load-balancing phase must be initiated. Because of variations either in the resource performances (CPU speed, communication bandwidth) or in the system/application requirements (completed tasks, new tasks, migrated tasks, etc.), data must be redistributed between participating processors so that the current (estimated) load is better balanced. We do not discuss the load-balancing mechanism itself (we take it as external, be it a system, an algorithm, an oracle, or whatever). Rather we aim at optimizing the data redistribution induced by the load-balancing mechanism.

We adopt the following abstract view of the problem. There are n participating processors $P_1, P_2, \ldots, P_n$. Each processor P_k initially holds L_k atomic data items. The load-balancing system/algorithm/oracle has decided that the new load of P_k should be $L_k - \delta_k$. If $\delta_k > 0$, this means that P_k now is overloaded and should send δ_k data items to other processors; if $\delta_k < 0$, P_k is underloaded and should receive $-\delta_k$ data items from other processors. Of course there is a conservation law: $\sum_{k=1}^{n} \delta_k = 0$. The goal is to determine the required communications and to organize them (what we call the data redistribution) in minimal time.

We assume that the participating processors are arranged along a ring, either unidirectional or bidirectional, and either with homogeneous or heterogeneous link bandwidths; hence a total of four different frameworks to deal with. There are two main contexts in which processor rings are useful. The first context is that of many applications which operate on ordered data, and where the order needs to be preserved. Think of a large matrix whose columns are distributed among the processors, but with the condition that each processor operates on a slice of consecutive columns. An overloaded processor P_i can send its first columns to the processor P_j that is assigned the slice preceding its own slice (and P_j would append these columns to the end of its slice). Similarly, P_i can send its last columns to the processor which is assigned the next slice. Obviously, these are the only possibilities. In other words, the ordered unidimensional data distribution calls for a unidimensional arrangement of the processors, i.e. along a ring.

The second context that may call for a ring is the simplicity of the programming. Using a ring, either unidirectional or bidirectional, allows for a simpler management of the data to be redistributed. Data intervals can be maintained and updated to characterize each processor load. Finally, we observe that parallel machines with a rich but fixed interconnection topology (hypercubes, fat trees, grids, to quote a few) are on the decline. Heterogeneous cluster architectures, which we target in this paper, have a largely unknown interconnection graph, including gateways, backbones, and switches, and modeling the communication graph as a ring is a reasonable, if conservative, choice.

As stated above, we discuss four cases for the redistribution algorithms. We delay the formal statement of the redistribution problems until Section 2, but we summarize the main results as follows. In the simplest case, that of a unidirectional homogeneous ring, we derive an optimal algorithm, and we prove its correctness in full detail. Because the target architecture is quite simple, we are able to provide explicit (analytical) formulae for the number of data sent/received by each processor. The same holds true for the case of a bidirectional homogeneous ring, but the algorithm becomes more complicated. When assuming heterogeneous communication links, we still derive an optimal algorithm for the unidirectional case, but we have to use an asynchronous formulation. However, we are only able to solve the bidirectional case in the special case of light redistributions. We point out that one major contribution of the paper is the design of optimal algorithms, together with their formal proof of correctness; to the best of our knowledge, this is the first time that optimal algorithms have been introduced.

The rest of the paper is organized as follows. In Section 2 we formally state the optimization problem. For homogeneous networks (all links have same capacity), the optimal algorithms are described in Section 3 (unidirectional ring) and in Section 5 (bidirectional ring). For heterogeneous networks, the optimal asynchronous unidirectional algorithm is presented in Section 4, and the linear-programming based optimal algorithm for light redistributions on bidirectional links is explained in Section 6. Section 7 is devoted to a survey of related work. In Section 8, we overview some simulation results that confirm the usefulness of data redistributions. Finally, in Section 9 we conclude the paper and highlight future work directions.

Due to page limits, we were not able to include all the proofs in this paper. The missing proofs can be found in Renard, Robert, and Vivien (2004a).

2 Framework

We consider a set of n processors $P_1, P_2, \ldots, P_n$ arranged along a ring. The successor of P_i in the ring is P_{i+1}, and its predecessor is P_{i-1}, where all indices are taken modulo n. For $1 \le k, l \le n$, $C_{k,l}$ denotes the slice of consecutive processors $C_{k,l} = P_k, P_{k+1}, \ldots, P_{l-1}, P_l$.

We denote by $c_{i,i+1}$ the capacity of the communication link from P_i to P_{i+1}. In other words, it takes $c_{i,i+1}$ time units to send an atomic data item from processor P_i to processor P_{i+1}. In the case of a bidirectional ring, $c_{i,i-1}$ is the capacity of the link from P_i to P_{i-1}. We use the one-port model for communications; at any given time, there are at most two communications involving a given processor, one sent and the other received. A given processor can simultaneously send and receive data, so there is no restriction in the unidirectional case. However, in the bidirectional case, a given processor cannot simultaneously send data to its successor and its predecessor; neither can it receive data from both sides. This is the only restriction induced by the model: any pair of communications that does not violate the one-port constraint can take place in parallel.

Each processor P_k initially holds L_k atomic data items. After redistribution, P_k will hold $L_k - \delta_k$ atomic data items. We call δ_k the "unbalance" of P_k. We denote by $\delta_{k,l}$ the total unbalance of the processor slice $C_{k,l}$: $\delta_{k,l} = \delta_k + \delta_{k+1} + \ldots + \delta_{l-1} + \delta_l$.

Because of the conservation law of atomic data items, $\sum_{k=1}^{n} \delta_k = 0$. Obviously the unbalance cannot be larger than the initial load: $L_k \ge \delta_k$. In fact, we suppose that any processor holds at least one data item, both initially ($L_k \ge 1$) and after the redistribution ($L_k \ge 1 + \delta_k$); otherwise we would have to build a new ring from the subset of resources still involved in the computation.

3 Homogeneous Unidirectional Ring

In this section, we consider a homogeneous unidirectional ring. Any processor P_i can only send data items to its successor P_{i+1}, and $c_{i,i+1} = c$ for all $i \in [1, n]$. We first derive a lower bound on the running time of any redistribution algorithm. Then, we present an algorithm achieving this bound (hence optimal), and we prove its correctness.

3.1 Lower Bound

We have the following bound on the optimal redistribution time:

Lemma 1. *Let τ be the optimal redistribution time. Then*

$$\tau \ge \left(\max_{\substack{1 \le k \le n,\\ 0 \le l \le n-1}} |\delta_{k,k+1}| \right) c. \tag{1}$$

Proof. The processor slice $C_{k,k+l} = P_k, P_{k+1}, \ldots, P_{k+l-1}, P_{k+l}$ has a total unbalance of $\delta_{k,k+l} = \delta_k + \delta_{k+1} + \ldots + \delta_{k+l-1} + \delta_{k+l}$. If $\delta_{k,k+l} > 0$, $\delta_{k,k+l}$ data items must be sent from $C_{k,k+l}$ to the other processors. The ring is unidirectional, so P_{k+l} is the only processor in $C_{k,k+l}$ with an outgoing link. Furthermore, P_{k+l} needs a time equal to $\delta_{k,k+l}c$ to send $\delta_{k,k+l}$ data items. Therefore, in any case, a redistribution scheme cannot take less than $\delta_{k,k+l}c$ to redistribute all data items. We have the same type of reasoning for the case $\delta_{k,k+l} < 0$. □

```
1: Let $\delta_{\max} = (\max_{1 \le k \le n, 0 \le l \le n-1} |\delta_{k,k+l}|)$
2: Let start and end be two indices such that the
   slice $C_{\text{start,end}}$ is of maximal unbalance:
   $\delta_{\text{start,end}} = \delta_{\max}$.
3: for $s = 1$ to $\delta_{\max}$ do
4:   for all $l = 0$ to $n - 1$ do
5:     if $\delta_{\text{start,start}+l} \ge s$ then
6:       $P_{\text{start}+l}$ sends to $P_{\text{start}+l+1}$ a data item
         during the time interval $[(s-1)c, sc]$
```

Algorithm 1. Redistribution algorithm for homogeneous unidirectional rings.

3.2 An Optimal Algorithm

Algorithm 1 is an optimal solution to our problem. We first prove its correctness (Lemma 3). Secondly, we prove its optimality (Lemma 4). Intuitively, if step 6 of this algorithm is always feasible, then each execution of step 3 has exactly a length of c, and the algorithm will meet the time bound of Lemma 1.

First, we point out that the slice $C_{\text{start, end}}$ is well defined in step 2 of the algorithm: for any slice with an unbalance δ, the slice made up from the remaining processors has the opposite unbalance $-\delta$. Next, we state the particular role of the processor P_{start}:

Lemma 2. *Processor P_{start} receives no data items during the execution of Algorithm 1.*

Proof. We prove the result by contradiction. Suppose that at a given iteration s processor P_{start} receives some data items. Then the predecessor of P_{start} in the ring, $P_{\text{start}-1}$, sends a data item at this iteration. Thus, $P_{\text{start}-1}$ being a sender, by the condition at step 5 of Algorithm 1, $\delta_{\text{start,start}-1} = \sum_{j=0}^{n-1} \delta_{\text{start}+j} \ge s$. However, due to the conservation law, $\sum_{i=1}^{n} \delta_i = 0$. Hence, $0 \ge s$, the desired contradiction.

To prove that Algorithm 1 is correct, we must show that during each iteration, any processor required to send a data item in step 6 actually holds at least one data item at this iteration. In other words, we must prove that no processor is asked to send a data item that it does not currently own. Let L_i^s be the load of P_i at the end of iteration s of Algorithm 1:

Lemma 3. *During iteration s of loop 3, if P_i sends a data item, then $L_i^{s-1} \ge 1$.*

Proof. We prove Lemma 3 by induction. By definition of unbalances (see Section 2), we know that each processor P_i in the ring initially holds an amount of $L_i^0 = L_i \ge 1$ data items. Thus, the result holds for $s = 1$.

Now we suppose that the result holds until a certain iteration s (included), and we focus on iteration $s + 1$. There are two cases to consider depending whether processor P_i is supposed to receive a data item during iteration $s + 1$ or not:

1. If processor P_i is both a sender and a receiver during iteration $s + 1$, then P_i is both a sender and a receiver during iteration s by the condition at step 5 of Algorithm 1. Then the load of P_i after iteration s was the same as before that iteration and $L_i^s = L_i^{s-1}$ We conclude using the induction hypothesis.
2. If processor P_i is a sender but not a receiver during iteration $s + 1$, we must verify that P_i does not send a data item that it does not hold. Because P_i is a sender we have by the condition at Step 5 of Algorithm 1:

$$\delta_{\text{start},i} \ge s + 1. \tag{2}$$

Furthermore, P_i has sent a data item during each of the previous iterations.

During iteration $s + 1$, P_i is not a receiver. Thus, P_{i-1} is not a sender during this iteration, and, by the condition at step 5 of Algorithm 1, we have $\delta_{\text{start}, i-1} < s + 1$. During each iteration from 1 to $\delta_{\text{start}, i-1}$, P_{i-1} has sent a data item (see below for the proof that $\delta_{\text{start, start}+j} \ge 0$ for all $j \in [0, n-1]$). Hence, during each of these iterations, P_i was both a sender and a receiver, and neither its load nor its unbalance changed.

During each iteration from $1 + \delta_{\text{start}, i-1}$ to s, processor P_i was a sender but not a receiver. So both its load and its unbalance decrease by one during each of these iterations. Hence

$$L_i^s = L_i - (s - \delta_{\text{start},i-1}). \tag{3}$$

However, $\delta_i + \delta_{\text{start}, i-1} = \delta_{\text{start}, i}$. So equation 3 is equivalent to $L_i^s = L_i - \delta_i + \delta_{\text{start}, i} - s$. From equation (2) we know that $\delta_{\text{start}, i} - s \ge 1$. In Section 2, we assumed that $L_i \ge 1 + \delta_i$. So, $L_i^s \ge 2$.

The above proof relies on the property that, for any value of $j \in [0, n-1]$, $\delta_{\text{start, start}+j} \ge 0$. We now prove this result by contradiction. Hence we suppose that there exists a value j such that $\delta_{\text{start, start}+j} < 0$. We have two cases to consider, as follows.

1. $j + \text{start} \in [\text{start}, \text{end}]$. Then $\delta_{\text{start, end}} = \delta_{\text{start, start}+j} + \delta_{\text{start}+j+1, \text{end}}$ and $\delta_{\text{start, end}} < \delta_{\text{start}+j+1, \text{end}}$ which contradicts the maximality of $C_{\text{start, end}}$.
2. $j + \text{start} \notin [\text{start}, \text{end}]$. Then $\delta_{\text{start}, j+\text{start}} = \delta_{\text{start, end}} + \delta_{1+\text{end}, j+\text{start}}$. So $\delta_{\text{start, end}} < -\delta_{1+\text{end}, j+\text{start}}$. However, as the sum of unbalances is null by definition, the sum of unbalances of $C_{1+\text{end}, j+\text{start}}$ is equal to the opposite of the sum of unbalances of $C_{j+1+\text{start, end}}$. Hence, $\delta_{\text{start, end}} < \delta_{j+1+\text{start, end}}$, which contradicts the maximality of $C_{\text{start, end}}$. □

We have proved the correction of Algorithm 1. We still have to prove that when it terminates, the entire redistribution has actually been performed.

Lemma 4. W*hen Algorithm 1 terminates after iteration* $\delta_{\max}$, *i.e. at time* τ, *the load of any processor* P_i *is equal to* $L_i - \delta_i$.

Proof. We prove by induction on the processor indices, starting at processor P_{start}, that any processor P_j has the desired load of $L_j - \delta_j$ at any iteration $s \geq \max_{0 \leq i \leq j} \delta_{\text{start, start}+i}$.

As stated by Lemma 2, processor P_{start} never receives a data item during the algorithm execution. So, after $\delta_{\text{start, start}} = \delta_{\text{start}}$ iterations of loop 3, P_{start} is never the receiver or the sender of a data item. As required, P_{start} exactly holds $L_{\text{start}} - \delta_{\text{start}}$ data items, i.e. its initial load minus the amount of data items sent.

We suppose the result proved up to a processor $P_{\text{start}+l}$ (with $l \geq 0$) included. We focus on processor $P_{\text{start}+l+1}$. Using the induction hypothesis, we know that at any iteration $s \geq \max_{0 \leq i \leq l} \delta_{\text{start, start}+i}$, the total load of the slice $C_{\text{start, start}+l}$ is equal to $\sum_{0 \leq i \leq l} L_i - \sum_{0 \leq i \leq l} \delta_i$.

During the execution of the whole algorithm, processor $P_{\text{start}+l+1}$ has sent exactly $\delta_{\text{start, start}+l+1}$ data items (remember that for any $j \in [0, n-1]$, $\delta_{\text{start, start}+j} \geq 0$). All these send operations took place before or during iteration $\delta_{\text{start, start}+l+1}$. Furthermore, Lemma 2 states that processor P_{start} never receives a data item during the execution. So, the total load of the slice $C_{\text{start, start}+l+1}$ does not change after iteration $\delta_{\text{start, start}+l+1}$, and its total load is equal to its initial total load minus the data items sent by processor $P_{\text{start}+l+1}$: $(\sum_{0 \leq i \leq l+1} L_i) - \delta_{\text{start, start}+l+1}$. Therefore, after any iteration s, where $s \geq \max(\max_{0 \leq i \leq l} \delta_{\text{start,start}+i}\, \delta_{\text{start,start}+l+1}) = \max_{0 \leq i \leq l+1} \delta_{\text{start, start}+i}$, we know the total load of the slices $C_{\text{start, start}+l}$ and $C_{\text{start, start}+l+1}$. Therefore, we know the load of processor $P_{\text{start}+l+1}$ at any step $t \geq s$:

$$L^t_{\text{start}+l+1} = \left[\left(\sum_{0 \leq i \leq l+1} L_{\text{start}+i}\right) - \delta_{\text{start, start}+l+1}\right]$$

$$= \left(\sum_{0 \leq i \leq l} L_{\text{start}+i} - \sum_{0 \leq i \leq l} \delta_{\text{start}+i}\right)$$

$$= L_{\text{start}+l+1} - \delta_{L_{\text{start}+l+1}}. \quad (4)$$

To conclude, we just need to remark that $\delta_{\max} = \max_{0 \leq i \leq n-1} \delta_{\text{start, start}+i}$. □

The optimality of Algorithm 1 is a direct consequence of the previous lemmas:

Theorem 1. *Algorithm 1 is optimal.*

4 Heterogeneous Unidirectional Ring

In this section we still suppose that the ring is unidirectional but we no longer assume the communication paths to have the same capacities. We build on the results of the previous section to design an optimal algorithm (Algorithm 2). In this algorithm, the amount of data items sent by any processor P_i is exactly the same as in Algorithm 1 (namely $\delta_{\text{start}, i}$). However, as the communication links have different capacities, we no longer have a synchronous behavior. A processor P_i sends its $\delta_{\text{start}, i}$ data items as soon as possible, but we cannot express its completion time with a simple formula. Indeed, if P_i initially holds more data items than it has to send, we have the same behavior as previously: P_i can send its data items during the time interval $[0, \delta_{\text{start}, i}\, c_{i, i+1}]$. In contrast, if P_i holds fewer data items than it has to send ($L_i < \delta_{\text{start}, i}$), P_i still starts to send some data items at time 0 but may have to wait to have received some other data items from P_{i-1} to be able to forward them to P_{i+1}.

The asynchronousness of Algorithm 2 implies that it is correct by construction. Furthermore, when the algorithm terminates, the redistribution is complete (the proof is the

```
1: Let $\delta_{\max} = (\max_{1 \leq k \leq n, 0 \leq l \leq n-1} |\delta_{k,k+l}|)$
2: Let start and end be two indices such that
   the slice $C_{\text{start,end}}$ is of maximal unbalance:
   $\delta_{\text{start,end}} = \delta_{\max}$.
3: **for all** $l = 0$ to $n-1$ **do**
4:     $P_{\text{start}+l}$ sends $\delta_{\text{start,start}+l}$ data items one by
       one and as soon as possible to processor
       $P_{\text{start}+l+1}$
```

Algorithm 2. Redistribution algorithm for heterogeneous unidirectional rings.

same as in Lemma 4). It remains to prove that the running time of Algorithm 2 is optimal. We first compute this running time.

Lemma 5. *The running time of Algorithm 2 is* $\max_{0 \le l \le n-1} \delta_{\text{start, start}+l}\, c_{\text{start}+l,\, \text{start}+l+1}$.

The result of Lemma 5 is surprising. Intuitively, it says that the running time of Algorithm 2 is equal to the maximum of the communication times of all the processors, if each of them initially stored locally all the data items it will have to send throughout the execution of the algorithm. In other words, there is no forwarding delay, whatever the initial distribution. The proof of Lemma 5 is technical and can be omitted at first reading.

Proof. We prove the result by contradiction, assuming that the running time of Algorithm 2, denoted as $t_{\max}$, is strictly greater than $\max_{0 \le l \le n-1} \delta_{\text{start, start}+l}\, c_{\text{start}+l,\, \text{start}+l+1}$ (we assume that the algorithm starts running at time 0). Let P_i be any processor whose running time is $t_{\max}$, i.e. let P_i be any processor which terminates the emission of its last data item at time $t_{\max}$. By hypothesis, $t_{\max} > \delta_{\text{start},\, i} c_{i,\, i+1}$. Therefore, there is some time during the running time of the algorithm at which processor P_i is not sending any data items to processor P_{i+1}. Let t_i denote the latest time at which P_i is not sending any data items. Then, by definition of t_i, from time t_i until the completion of the algorithm, processor P_i is continuously sending data items to P_{i+1}. Let n_i denote the number of data items that P_i sends during that interval. Note that we have $t_{\max} = t_i + n_i c_{i,\, i+1}$. We now prove by induction that for any value of $j \ge 1$.

1. Processor P_{i-j} sends a data item to processor P_{i-j+1} during the time interval $[t_i - \sum_{k=1}^{j} c_{i-k,\, i-k+1},\ t_i - \sum_{k=1}^{j-1} c_{i-k,\, i-k+1}]$.
2. Between time $t_i - \sum_{k=1}^{j} c_{i-k,\, i-k+1}$ and the completion of the algorithm, processor P_{i-j} sends at least $j + n_i$ data items to processor P_{i-j+1}.
3. $c_{i-j,\, i-j+1} \le c_{i,\, i+1}$.
4. Right before time $t_i - \sum_{k=1}^{j} c_{i-k,\, i-k+1}$, processor P_{i-j} is not sending any data items to processor P_{i-j+1} (it is idle in sending).

Once we have proved these properties, the contradiction follows from considering processor P_{start}. Processor P_{start} only sends data items that it initially holds ($\delta_{\text{start}} = \delta_{\text{start, start}} \le L_{\text{start}}$), and receives no data items from its predecessor in the ring. However, using the above properties, there is a value of $j \ge 0$ such that $\text{start} = i - j$, and between time $t_i - \sum_{k=1}^{j+1} c_{i-k,\, i-k+1}$ and the completion of the algorithm, processor P_{i-j-1} sends at least $j + 1 + n_i$ data items to processor $P_{i-j} = P_{\text{start}}$. Hence the contradiction.

The construction used in the proof is illustrated by Figure 1. We start by proving the above properties for $j = 1$.

1. By definition of t_i, processor P_i is not sending any data items to processor P_{i+1} right before time t_i. Because of the "as-soon-as" nature of the algorithm, processor P_i is not holding a single data item right before time t_i and is waiting for processor P_{i-1} to send it one. Furthermore, the data item that processor P_i started to send at time t_i is sent to it by processor P_{i-1} during the time interval $[t_i - c_{i-1,\, i},\ t_i]$.

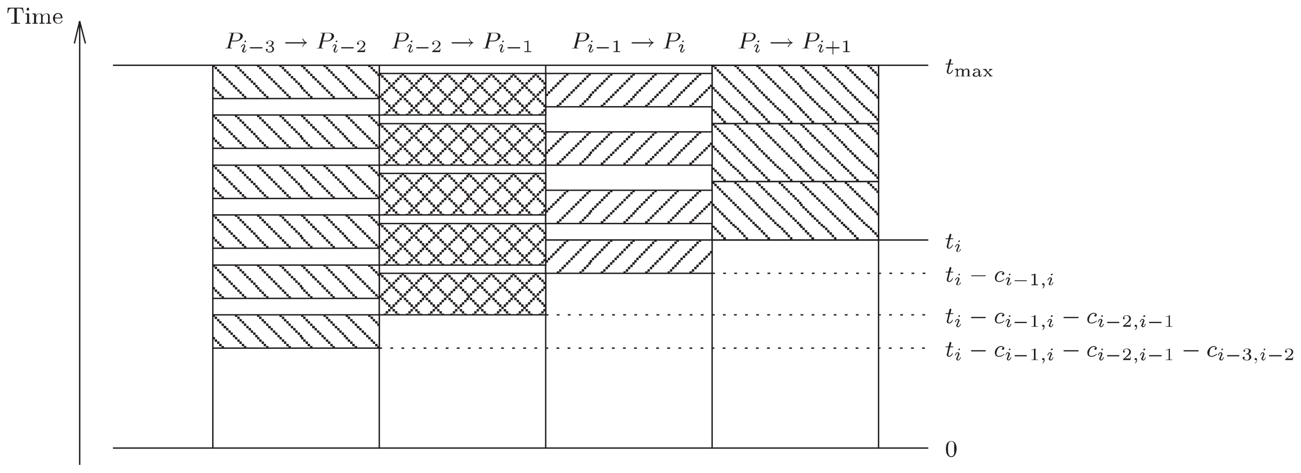

Fig. 1 The construction used in the proof of Lemma 5.

2. Between time t_i and the completion of the algorithm, processor P_i sends n_i data items to processor P_{i+1}. By hypothesis, processor P_i holds at least one data item after the completion of the algorithm. As P_i holds no data item right before time t_i, then between the times $t_i - c_{i-1,i}$ and $t_{\max}$, P_{i-1} sends at least $1 + n_i$ data items to P_i.
3. From what just precedes, and using the relationship between t_i, n_i, and $t_{\max}$, we have $t_i + n_i c_{i,i+1} = t_{\max}$ and $t_{\max} \geq (t_i - c_{i-1,i}) + (1 + n_i)c_{i-1,i}$, which imply $c_{i,i+1} \geq c_{i-1,i}$, as n_i is non-zero by definition.
4. Suppose that processor P_{i-1} is sending a data item to processor P_i right before the time $t_i - c_{i-1,i}$. Then, at the earliest, this data item is received by processor P_i at time $t_i - c_{i-1,i}$. Due to the "as-soon-as" nature of the algorithm, P_i forwards this data item to processor P_{i+1} (as it forwards data items received later). P_i finishes to forward this data item at time $t_i - c_{i-1,i} + c_{i,i+1} \geq t_i$ at the earliest. Therefore, processor P_i has no reason not to be sending any data item at time t_i, which contradicts the definition of t_i.

We now proceed to the general case of the induction. We suppose that the property is proved up to a processor P_{i-j} included (with $j \geq 1$).

1. By induction hypothesis, processor P_{i-j} is not sending any data items to processor P_{i-j+1} right before time $t_i - \sum_{k=1}^{j} c_{i-k,i-k+1}$. Because of the "as-soon-as" nature of the algorithm, processor P_{i-j} is not holding a single data item right before this time and is waiting for processor P_{i-j-1} to send one. Furthermore, the data item that processor P_{i-j} started to send at time $t_i - \sum_{k=1}^{j} c_{i-k,i-k+1}$ is sent to it by processor P_{i-j-1} during the time interval $[t_i - \sum_{k=1}^{j+1} c_{i-k,i-k+1}, t_i - \sum_{k=1}^{j} c_{i-k,i-k+1}]$.
2. Between time $t_i - \sum_{k=1}^{j} c_{i-k,i-k+1}$ and the completion of the algorithm, processor P_{i-j} sends $j + n_i$ data items to processor P_{i-j+1}, by induction hypothesis. By hypothesis, processor P_{i-j} holds at least one data item after the completion of the algorithm. As P_{i-j} holds no data item right before time $t_i - \sum_{k=1}^{j} c_{i-k,i-k+1}$, then between the times $t_i - \sum_{k=1}^{j+1} c_{i-k,i-k+1}$ and $t_{\max}$, P_{i-j-1} sends at least $1 + j + n_i$ data items to P_{i-j}.
3. From what just precedes, and using the relationship between t_i, n_i, and $t_{\max}$, we have

$$t_i + n_i c_{i,i+1} = t_{\max} \geq$$
$$\left(t_i - \sum_{k=1}^{j+1} c_{i-k,i-k+1}\right) + (1 + j + n_i)c_{i-j-1,i-j}$$

Therefore,

$$n_i c_{i,i+1} + \sum_{k=1}^{j} c_{i-k,i-k+1} \geq (j + n_i)c_{i-j-1,i-j}$$

and thus: $c_{i,i+1} \geq c_{i-j-1,i-j}$ as, by induction hypothesis, for any $k \in [1, j]$, $c_{i,i+1} \geq c_{i-k,i-k+1}$.
4. Suppose that processor P_{i-j-1} is sending a data item to processor P_{i-j} right before the time $t_i - \sum_{k=1}^{j+1} c_{i-k,i-k+1}$. Then, at the earliest, this data item is received by processor P_{i-j} at time $t_i - \sum_{k=1}^{j+1} c_{i-k,i-k+1}$. Due to the "as-soon-as" nature of the algorithm, P_{i-j} forwards this data item to processor P_{i-j+1} (as it forwards data items received later). P_{i-j} finishes to forward this data item at time $t_i - c_{i-j-1,i-j} - \sum_{k=1}^{j-1} c_{i-k,i-k+1}$ at the earliest. Then, following the same line of reasoning, processor P_{i-j+1} forwards it to P_{i-j+2}, which receives it at the earliest at time $t_i - c_{i-j-1,i-j} - \sum_{k=1}^{j-2} c_{i-k,i-k+1}$, and so on. So, processor P_i receives this data item at the earliest at time $t_i - c_{i-j-1,i-j}$, and forwards it. Then, it finishes to send it at the earliest at time $t_i - c_{i-j-1,i-j} + c_{i,i+1} \geq t_i$, as we have seen that $c_{i,i+1} \geq c_{i-j-1,i-j}$. Therefore, processor P_i has no reason not to be sending any data items at time t_i, which contradicts the definition of t_i. Hence, processor P_{i-j-1} is not sending any data item to processor P_{i-j} right before the time $t_i - \sum_{k=1}^{j+1} c_{i-k,i-k+1}$. □

Theorem 2. *Algorithm 2 is optimal.*

Proof. Let τ denote the optimal redistribution time. Following the arguments used in the proof of Lemma 1 for the homogeneous case in Section 3.1, we obtain the lower bound:

$$\tau \geq \max_{1 \leq k \leq n,\, 0 \leq l \leq n-1} |\delta_{k,k+l}| c_{k+l,k+l+1}.$$

We conclude using Lemma 5. □

5 Homogeneous Bidirectional Ring

In this section, we consider a homogeneous bidirectional ring. All links have the same capacity but a processor can send data items to its two neighbors in the ring; there exists a constant c such that, for all $i \in [1, n]$, $c_{i,i+1} = c_{i,i-1} = c$. We proceed as for the homogeneous unidirectional case. We first derive a lower bound on the running time of any redistribution algorithm, and then we present an algorithm attaining this bound.

5.1 Lower Bound

We have the following bound on the optimal redistribution time.

Lemma 6. *Let τ be the optimal redistribution time. Then*

$$\tau \geq \max\left\{\max_{1 \leq i \leq n} |\delta_i|, \qquad \max_{\substack{1 \leq i \leq n,\\ 1 \leq l \leq n-1}} \left\lceil \frac{|\delta_{i,\,i+l}|}{2} \right\rceil \right\} c. \qquad (5)$$

Proof. Consider any processor P_i with positive unbalance ($\delta_i > 0$). Even if processor P_i can send data items to both of its neighbors, because of the one-port model, it cannot send data items to both of them simultaneously. So, it requires processor P_i at least a time of $\delta_i c$ to send δ_i data items, whatever the destinations of these data items. We have a symmetric result for the case $\delta_i < 0$. Hence a first lower bound on the optimal redistribution time τ:

$$\tau \geq \left(\max_{1 \leq i \leq n} |\delta_i|\right) c.$$

Now, consider any non-trivial slice of consecutive processors $C_{k,\,l}$. By "non-trivial" we mean that the slice is not reduced to a single processor (we have already considered that case) and that it does not contain all processors. We suppose that $\delta_{k,\,l} > 0$. So, in any redistribution scheme, at least $\delta_{k,\,l}$ data items must be sent by $C_{k,\,l}$. As this slice is not reduced to a single processor, the two processors at the extremities of the slice, P_k and P_l, can simultaneously send data items to their neighbors outside the slice, P_{k-1} and P_{l+1}, respectively. Therefore, during any time interval of length c, at most two data items can be sent from the slice. So, it takes at least a time of $\left\lceil \frac{\delta_{k,l}}{2} \right\rceil c$ for the slice $C_{k,\,l}$ to send $\delta_{k,\,l}$ data items. Once again, the reasoning is similar when receiving data items if $\delta_{k,\,l} < 0$. Hence a second lower bound on τ:

$$\tau \leq \left(\max_{1 \leq k \leq n,\, 1 \leq l \leq n-1} \left\lceil \frac{|\delta_{i,\,i+l}|}{2} \right\rceil \right) c.$$

We just gather the previous two lower-bounds to obtain the desired bound. □

5.2 An Optimal Algorithm

Algorithm 3 is a recursive algorithm which defines communication patterns designed so as to decrease the value of $\delta_{\max}$ (computed at step 1) by one from one recursive call to another. The intuition behind Algorithm 3 is the following.

1. Any non-trivial slice $C_{k,\,l}$ such that $\lceil |\delta_{k,l}|/2 \rceil = \delta_{\max}$ and $\delta_{k,\,l} \geq 0$ must send two data items per recursive call, one through each of its extremities.
2. Any non-trivial slice $C_{k,\,l}$ such that $\lceil |\delta_{k,l}|/2 \rceil = \delta_{\max}$ and $\delta_{k,\,l} \leq 0$ must Greceive two data items per recursive call, one through each of its extremities.
3. Once the mandatory communications specified by the two previous cases are defined, we take care of any processor P_i such that $|\delta_i| = \delta_{\max}$. If P_i is already involved in a communication due to the previous cases, everything is settled. Otherwise, we have the freedom to choose whom P_i will send a data item to (case $\delta_i > 0$) or whom P_i will receive a data item from (case $\delta_i < 0$). To simplify the algorithm we decide that all these communications will take place in the direction from P_i to P_{i+1}.

Algorithm 3 is initially called with the parameter $s = 1$. For any call to Algorithm 3, all the communications take place in parallel and exactly at the same time, because the communication paths are homogeneous by hypothesis. One very important point about Algorithm 3 is that this algorithm is a set of rules which only specify which processor P_i must send a data item to which processor P_j, one of its immediate neighbors. Therefore, whatever the number of rules deciding that there must be some data item sent from a processor P_i to one of its immediate neighbor P_j, only one data item is sent from P_i to P_j to satisfy all these rules.

To prove that Algorithm 3 is optimal, we show that the set of rules is consistent, i.e. that it respects the one-port model, and that the value $\delta_{\max}$ (computed at step 1) decreases by one at each recursive call.

Lemma 7. *Algorithm 3 satisfies to all the one-port constraints.*

Lemma 8. *Algorithm 3 terminates in exactly*

$$\max\left\{\max_{1 \leq i \leq n} |\delta_i|, \qquad \max_{1 \leq i \leq n,\, 1 \leq l \leq n-1} \left\lceil \frac{|\delta_{i,\,i+l}|}{2} \right\rceil \right\}$$

recursive calls.

The optimality of Algorithm 3 is then a simple corollary of Lemma 8 and of the lower bound defined by equation (5) (the missing proofs can be found in Renard, Rober, and Vivien 2004a).

Theorem 3. *Algorithm 3 is optimal.*

6 Heterogeneous Bidirectional Ring

In this section, we consider the most general case, that of a heterogeneous bidirectional ring. We do not know any optimal redistribution algorithm in this case. However, if we assume that each processor initially holds more data than it needs to send during the whole execution of the algorithm (what we call a light redistribution), then we succeed in deriving an optimal solution.

```
1: Let $\delta_{\max} = \max\{\max_{1 \le i \le n} |\delta_i|, \max_{1 \le i \le n, 1 \le l \le n-1} \left\lceil \frac{|\delta_{i,i+l}|}{2} \right\rceil\}$
2: if $\delta_{\max} \ge 1$ then
3:   if $\delta_{\max} \ne 2$ then
4:     for all slice $C_{k,l}$ such that $\delta_{k,l} > 1$ and $\left\lceil \frac{|\delta_{k,l}|}{2} \right\rceil = \delta_{\max}$ do
5:       $P_k$ sends a data item to $P_{k-1}$ during the time interval $[(s-1)c, sc]$.
6:       $P_l$ sends a data item to $P_{l+1}$ during the time interval $[(s-1)c, sc]$.
7:     for all slice $C_{k,l}$ such that $\delta_{k,l} < -1$ and $\left\lceil \frac{|\delta_{k,l}|}{2} \right\rceil = \delta_{\max}$ do
8:       $P_{k-1}$ sends a data item to $P_k$ during the time interval $[(s-1)c, sc]$.
9:       $P_{l+1}$ sends a data item to $P_l$ during the time interval $[(s-1)c, sc]$.
10:  else if $\delta_{\max} = 2$ then
11:    for all slice $C_{k,l}$ such that $\delta_{k,l} \ge 3$ do
12:      $P_l$ sends a data item to $P_{l+1}$ during the time interval $[(s-1)c, sc]$.
13:    for all slice $C_{k,l}$ such that $\delta_{k,l} = 4$ do
14:      $P_k$ sends a data item to $P_{k-1}$ during the time interval $[(s-1)c, sc]$.
15:    for all slice $C_{k,l}$ such that $\delta_{k,l} \le -3$ do
16:      $P_{k-1}$ sends a data item to $P_k$ during the time interval $[(s-1)c, sc]$.
17:    for all slice $C_{k,l}$ such that $\delta_{k,l} = -4$ do
18:      $P_{l+1}$ sends a data item to $P_l$ during the time interval $[(s-1)c, sc]$.
19:  for all processor $P_i$ such that $\delta_i = \delta_{\max}$ do
20:    if $P_i$ is not already sending, due to one of the previous steps, a data item during the time interval
       $[(s-1)c, sc]$ then
21:      $P_i$ sends a data item to $P_{i+1}$ during the time interval $[(s-1)c, sc]$.
22:  for all processor $P_i$ such that $\delta_i = -(\delta_{\max})$ do
23:    if $P_i$ is not already receiving, due to one of the previous steps, a data item during the time interval
       $[(s-1)c, sc]$ then
24:      $P_i$ receives a data item from $P_{i-1}$ during the time interval $[(s-1)c, sc]$.
25:  if $\delta_{\max} = 1$ then
26:    for all processor $P_i$ such that $\delta_i = 0$ do
27:      if $P_{i-1}$ sends a data item to $P_i$ during the time interval $[(s-1)c, sc]$ then
28:        $P_i$ sends a data item to $P_{i+1}$ during the time interval $[(s-1)c, sc]$.
29:      if $P_{i+1}$ sends a data item to $P_i$ during the time interval $[(s-1)c, sc]$ then
30:        $P_i$ sends a data item to $P_{i-1}$ during the time interval $[(s-1)c, sc]$.
31:  Recursive call to Algorithm 3 $(s+1)$
```

Algorithm 3. Redistribution algorithm for homogeneous bidirectional rings (for step *s*).

6.1 Light Redistribution

Throughout this section, we suppose that we have a light redistribution: we assume that the number of data items sent by any processor throughout the redistribution algorithm is less than or equal to its original load. There are two reasons for a processor P_i to send data: (i) because it is overloaded ($\delta_i > 0$); (ii) because it has to forward some data to another processor located further in the ring. If P_i initially holds at least as many data items as it will send during the whole execution, then P_i can send at once all these data items. Otherwise, in the general case, some processors may wait to have received data items from a neighbor before being able to forward them to another neighbor.

6.1.1 Solution by Integer Linear Programming Under the "light redistribution" assumption, we can build an integer linear program to solve our problem (see system 6). Let $\mathcal{S}$ be one of its solutions, and denote by $\mathcal{S}_{i,\,i+1}$ the number of data items that processor P_i sends to processor P_{i+1}. Similarly, $\mathcal{S}_{i,\,i-1}$ is the number of data items that P_i

sends to processor P_{i-1}. In order to ease the writing of the equations, we impose in the first two equations of system 6 that $\mathcal{S}_{i,i+1}$ and $\mathcal{S}_{i,i-1}$ are non-negative for all i, which imposes the use of other variables $\mathcal{S}_{i+1,i}$ and $\mathcal{S}_{i-1,i}$ for the symmetric communications. The third equation states that after the redistribution, there is no more unbalance. We denote by τ the execution time of the redistribution. For any processor P_i, due to the one-port constraints, τ must be greater than the time spent by P_i to send data items (fourth equation) or spent by P_i to receive data items (fifth equation). Our aim is to minimize τ; hence the system:

$$\text{MINIMIZE } \tau \text{, SUBJECT TO}\quad \begin{cases} \forall i, \mathcal{S}_{i,i+1} \geq 0 \\ \forall i, \mathcal{S}_{i,i-1} \geq 0 \\ \forall i, \mathcal{S}_{i,i+1} + \mathcal{S}_{i,i-1} - \mathcal{S}_{i+1,i} - \mathcal{S}_{i-1,i} \;=\; \delta_i \\ \forall i, \mathcal{S}_{i,i+1}c_{i,i+1} + \mathcal{S}_{i,i-1}c_{i,i-1} \leq \tau \\ \forall i, S_{i+1,i}c_{i,i+1} + S_{i-1,i}c_{i,i-1} \leq \tau \end{cases} \tag{6}$$

Lemma 9. A*ny optimal solution of system 6 is feasible, for example using the following schedule: for any $i \in [1, n]$, P_i starts sending data items to P_{i+1} at time 0 and, after the completion of this communication, starts sending data items to P_{i-1} as soon as possible under the one-port model.*

Proof. We have to show that we are able to schedule the communications defined by any optimal solution $(\mathcal{S}, \tau)$ of system 6 so that the redistribution takes a time no greater than τ. For any $i \in [1, n]$, we schedule at time 0 all emissions from P_i to P_{i+1}. This communication is done in time $\mathcal{S}_{i,i+1}c_{i,i+1}$: because of the "light redistribution" hypothesis, P_i already holds all the data items that it must send. Because of the fourth equation of system 6, this communication ends before the time τ.

For any value of $i \in [1, n]$, we still have to schedule the sending of data items from P_i to P_{i-1}. We schedule this communication as soon as possible; therefore, at time $\max\{\mathcal{S}_{i,i+1}c_{i,i+1}, \mathcal{S}_{i-2,i-1}c_{i-2,i-1}\}$, i.e. at the earliest time when (i) P_i has ended sending data items to P_{i+1}, and (ii) P_{i-1} has stopped receiving data items from P_{i-2}. Therefore, the communication from P_i to P_{i-1} ends at the date:

$$\begin{aligned} &\max\{\mathcal{S}_{i,i+1}c_{i,i+1}, \mathcal{S}_{i-2,i-1}c_{i-2,i-1}\} + \mathcal{S}_{i,i-1}c_{i,i-1} \\ &\quad = \max\{\mathcal{S}_{i,i+1}c_{i,i+1} + \mathcal{S}_{i,i-1}c_{i,i-1}, \\ &\qquad \mathcal{S}_{i-2,i-1}c_{i-2,i-1} + \mathcal{S}_{i,i-1}c_{i,i-1} \end{aligned} \tag{7}$$

Once again, this is true owing to the "light redistribution" hypothesis: no processor needs to wait to have received some data items before being able to send them to one of its neighbors.

The first term of the "max" expression is the time needed by P_i to send data items to both P_{i+1} and P_{i-1}. This term is less than or equal to τ because of the fourth equation of system 6. The second term of the "max" expression is the time needed by P_{i-1} to receive data items from both P_{i-2} and P_i. This term is less than or equal to τ because of the fifth equation of system 6. □

So far, we have not mathematically defined a condition for the "light redistribution" hypothesis to hold. In fact, this is not mandatory: we use system 6 to find an optimal solution to the problem. If, in this optimal solution, for any processor P_i, the total number of data items sent is less than or equal to the initial load ($\mathcal{S}_{i,i+1} + \mathcal{S}_{i,i-1} \leq L_i$), we are under the "light redistribution" hypothesis and we can use the solution of system 6 safely.

6.1.2 Solution Through Rational Linear Programming Even if the "light redistribution" hypothesis holds, we may wish to solve the redistribution problem with a technique less expensive than integer linear programming (which is potentially exponential). An idea would be to first solve system 6 to find an optimal rational solution, which can always be done in polynomial time, and then to round up the obtained solution to find a "good" integer solution. In fact, the following theorem shows that one of the two natural ways of rounding always lead to an optimal (integer) solution. The complexity of the light redistribution problem is therefore polynomial.

Theorem 4. *Let $\mathcal{R}$ be an optimal rational solution to the redistribution problem. For any j in $[1, n]$, $\mathcal{R}_j$ denotes the number of data items that processor P_j sends to processor P_{j+1} (using the notations of system 6, $\mathcal{R}_j = \mathcal{S}_{j,j+1} - \mathcal{S}_{j+1,j}$). Let $\mathcal{F}$ be the integer solution defined by $\mathcal{F}_1 = \lfloor \mathcal{R}_1 \rfloor$. Let $\mathcal{G}$ be the integer solution defined by $\mathcal{G}_1 = \lceil \mathcal{R}_1 \rceil$.* Then:
(i) $\mathcal{F}$ and $\mathcal{G}$ are well defined by the single condition above;
(ii) *either $\mathcal{F}$ or $\mathcal{G}$ is an optimal integer solution.*

Proof. Lemma 10 states that $\mathcal{F}$ and $\mathcal{G}$ are both fully defined. Lemma 11 states that there exists at least one optimal integer solution $\mathcal{E}$ such that $|\mathcal{E}_1 - \mathcal{R}_1| < 1$. The only two solutions satisfying these constraints are $\mathcal{F}$ and $\mathcal{G}$. Hence the result. □

Lemma 10. *To fully define the number of data items sent between processors in any redistribution scheme, we only need to define, for a single given value of $j \in [1, n]$, the number of data items that processor P_j sends to processor P_{j+1}.*

Lemma 11. *Let $\mathcal{R}$ be an optimal rational solution to the redistribution problem: for any j in $[1, n]$, $\mathcal{R}_j$ denotes the number of data items processor P_j sends to processor P_{j+1}.*

Then, there exists an optimal integer solution $\mathcal{E}$ to the solution problem such that: $|\mathcal{E}_1 - \mathcal{R}_1| < 1$.

The missing proofs can be found in Renard, Robert, and Vivien (2004a).

6.2 General Case

6.2.1 Lower Bound We have the following bound on the optimal redistribution time.

Lemma 12. *Let τ be the optimal redistribution time. Then:*

$$\tau \geq \max_{l \leq k \leq n,\, \delta_k > 0} \delta_k \cdot \min\{c_{k,k-1}, c_{k,k+1}\}$$

$$\tau \geq \max_{l \leq k \leq n,\, \delta_k < 0} \delta_k \cdot \min\{c_{k,k-1}, c_{k+1,k}\}$$

$$\tau \geq \max_{\substack{1 \leq k \leq n,\\ 1 \leq l \leq n-2,\\ \delta_{k,k+1} > 0}} \min_{0 \leq i \leq \delta_{k,k+1}} \max\{i \cdot c_{k,k-1}, (\delta_{k,k+l-i}) \cdot c_{k+l,k+l+1}\}$$

$$\tau \geq \max_{\substack{1 \leq k \leq n,\\ 1 \leq l \leq n-2,\\ \delta_{k,k+1} > 0}} \min_{0 \leq i \leq -\delta_{k,k+1}} \max\{i \cdot c_{k-1,k}, (-\delta_{k,k+l-i}) \cdot c_{k+l+1,k+l}\}.$$

Proof. Consider any processor P_i with positive unbalance ($\delta_i > 0$). Even if processor P_i can send data items to both of its neighbors, because of the one-port model, it cannot send data items to both of them simultaneously. The best way for processor P_i to send δ_i data items is then to send them using the fastest of its outgoing links. So, it requires processor P_i at least a time of $\delta_i \min\{c_{i,i-1}, c_{i,i+1}\}$ to send δ_i data items, whatever the destinations of these data items. We have a symmetric result for the case $\delta_i < 0$. Hence the first two inequations on τ.

Now, consider any non-trivial slice of consecutive processors $C_{k,l}$. By "non-trivial" we mean that the slice is not reduced to a single processor (we have already considered that case) and that it does not contain all processors. We suppose that $\delta_{k,l} > 0$. So, in any redistribution scheme, at least $\delta_{k,l}$ data items must be sent by $C_{k,l}$. As this slice is not reduced to a single processor, the two processors at the extremities of the slice, P_k and P_l, can simultaneously send data items to their neighbors outside the slice, P_{k-1} and P_{l+1} respectively. Therefore, during the redistribution, processor P_k sends a certain amount $i \in [0, \delta_{k,l}]$ of data items to processor P_{k-1}, while processor P_l sends the remaining data items to P_{l+1}, which takes a time max $\{i \leq c_{k,k-1}, (\delta_{k,l} - i) \leq c_{l,l+1}\}$. Then we choose for i a value which minimizes this time. We have a symmetric result for the case $\delta_{k,l} < 0$. Hence the last two inequations on τ. □

6.2.2 Heuristic Approaches We do not know whether the bound given by Lemma 12 can always be reached, but we have no counter-example proving that the bound is not tight.

When the solution found by system 6 does not satisfy the "light redistribution" hypothesis, there is the possibility to modify the system to enforce it: we obtain system 6 which finds a solution which satisfies the "light redistribution" hypothesis, if one exists. However, there is no reason a priori for the solution of system 6 to be optimal.

$$\text{MINIMIZE } \tau, \text{ SUBJECT TO}$$
$$\begin{cases} \forall i, \mathcal{S}_{i,i+1} \geq 0 \\ \forall i, \mathcal{S}_{i,i-1} \geq 0 \\ \forall i, \mathcal{S}_{i,i+1} + \mathcal{S}_{i,i-1} - \mathcal{S}_{i+1,i} - \mathcal{S}_{i-1,i} = \delta_i \\ \forall i, \mathcal{S}_{i,i+1} c_{i,i+1} + \mathcal{S}_{i,i-1} c_{i,i-1} \leq \tau \\ \forall i, S_{i+1,i} c_{i,i+1} + S_{i-1,i} c_{i,i-1} \leq \tau \\ \forall i, \mathcal{S}_{i,i+1} + \mathcal{S}_{i,i-1} \leq L_i \end{cases} \quad (8)$$

To conclude this section, we point out that the design of an optimal algorithm in the most general case remains open. Given the complexity of the lower bound, the problem looks very difficult to solve.

7 Related Work

Redistribution algorithms have been the focus of an abundant literature. On the theoretical side, in the framework of the high performance Fortran (Koelbel et al. 1994) compilation, Kremer (1993) showed the NP-completeness of a simple redistribution problem. This negative result shows that optimal algorithms can be designed only for particular cases, such as the ring architecture in this paper. To the best of our knowledge, no other redistribution algorithms have been proven optimal, but several efficient algorithms have been designed for rings (Hamdi and Lee 1995; Lee and Hamdi 1995; Deelman and Szymanski 1998), trees or hypercubes (Wu 1997). The elastic load balancing algorithm designed in Miguet and Robert (1992) and Bevilacqua (1999) has led to data redistribution software used for query processing (Brunie, Flory, and Kosch 1995) and medical image analysis (Sarrut and Miguet 1999).

The block-cyclic distribution of data arrays plays a very important role in scientific libraries (Blackford et al. 1997). In a CYCLIC(r) distribution over p processors, blocks of r consecutive elements of the array are distributed to the processors in a wraparound fashion, and the parameter r is chosen to optimize the granularity, i.e. the computation-to-communication ratio. Because this granularity changes from one computational kernel to the other, moving from a CYCLIC(r) distribution over p processors to a CYCLIC(s) distribution over q processors is a very useful redistribution procedure, which has been implemented using a caterpillar algorithm in ScaLAPACK (Prylli and Tourancheau 1997). Several papers, including Kalns and Ni (1995), Thakur, Choudhary, and Ramanujam (1996), Desprez et al. (1998), Park, Prasanna, and Raghavendra (1999), Garcia, Ayguadé, and Labarta (2001), Hsu et al. (2001), and Knoop and Mehofer (2002), have dealt with various optimizations of this redistribution procedure. Along this line of research, automatic data redistribution tools are presented in Garcia, Ayguadé, and Labarta (2001).

Even though we have not dealt with load-balancing algorithms in this paper, we quote some key references on the subject. For homogeneous platforms, see the collection of papers (Shirazi, Hurson, and Kavi 1995), and for heterogeneous clusters see chapter 25 in Buyya (1999). Several authors (Nicol and Saltz 1988; Nicol and Reynolds 1990; Flaherty et al. 1997a; Watts and Taylor 1998; Hu and Blake 1999) have proposed a mapping policy which dynamically minimizes system degradation (including the cost of remapping) for each computation step. Static strategies aiming at distributing independent chunks of work to two-dimensional processor grids are studied in Barbosa, Tavares, and Padilha (2000) and Beaumont et al. (2001a). Relaxing the geometrical constraints induced by two-dimensional grids leads to irregular partitionings (Crandall and Quinn 1993; Kaddoura, Ranka, and Wang 1996; Beaumont et al. 2001b) that allow for good load balancing but are much more difficult to implement. This approach has been extended to three-dimensional problems (Flaherty et al. 1997b).

Finally, we briefly mention three sample applications whose implementation can directly benefit from the redistribution strategies designed in this paper. The analysis of pulses propagating in a nonlinear medium calls for adaptive computational windows, and redistribution must occur frequently as the computation progresses (Bourgeade and Nkonga 2004). A two-level redistribution procedure is advocated in Lan, Taylor, and Bryan (2001) for structured adaptive mesh refinement. A multilevel diffusion re-partitioner is presented in Schloegel, Karypis, and Kumar (1997, 2000) for irregular grid computations and has been incorporated into the ParMetis library. Of course this short list could be extended dramatically.

8 Simulation Results

Due to lack of space, we refer the reader to Renard, Robert, and Vivien (2004a, 2004b) for the details. As expected, when the computation-to-communication ratio is high, the best strategy is to use no redistribution, as the cost is prohibitive. Conversely, when the computation-to-communication ratio is low, it pays off to use many redistributions, but not too many. As the ratio decreases, all trade-offs can be found.

9 Conclusion

We have considered the problem of redistributing data on rings of processors. For homogeneous rings, the problem has been completely solved. Indeed, we have designed optimal algorithms, and provided formal proofs of correctness, for both unidirectional and bidirectional rings. The bidirectional algorithm turned out to be quite complex, and requires a lengthy proof.

For heterogeneous rings, there remains further research to be conducted. The unidirectional case was easily solved, but the bidirectional case remains open. Still, we have derived an optimal solution for light redistributions, an important case in practice. The complexity of the bound for the general case shows that designing an optimal algorithm is likely to be a difficult task.

All our algorithms have been implemented and extensively tested. As expected, the cost of data redistributions may not pay off a little unbalance of the work in some cases. Further work will aim at investigating how frequently redistributions must occur in real-life applications.

Author Biographies

Hélène Renard is currently a Ph.D. student in the Computer Science Laboratory LIP at ENS Lyon. She is mainly interested in parallel algorithm design for heterogeneous platforms and in load-balancing techniques.

Yves Robert received a Ph.D. from Institut National Polytechnique de Grenoble in 1986. He is currently a full professor in the Computer Science Laboratory LIP at ENS Lyon. He is the author of four books, 85 papers published in international journals, and 110 papers published in international conferences. His main research interests are scheduling techniques and parallel algorithms for clusters and grids. He is a senior member of IEEE, and serves as an associate editor of IEEE Transactions on Parallel and Distributed Systems.

Frédéric Vivien received a Ph.D. from École normale supérieure de Lyon in 1997. From 1998 to 2002, he was an associate professor at Louis Pasteur University of Strasbourg. He spent the year 2000 working in the Computer Architecture Group of the MIT Laboratory for Computer Science. He is currently a full researcher from INRIA. His main research interests are scheduling techniques, parallel algorithms for clusters and grids, and automatic compilation/parallelization techniques.

References

Barbosa, J., Tavares, J., and Padilha, A. J. 2000. Linear algebra algorithms in a heterogeneous cluster of personal computers. *Proceedings of the 9th Heterogeneous Computing Workshop (HCW 2000)*, Cancun, Mexico, May 1, IEEE Computer Society Press, Los Alamitos, CA, pp. 147–159.

Beaumont, O., Boudet, V., Petitet, A., Rastello, F., and Robert, Y. 2001a. A proposal for a heterogeneous cluster ScaLAPACK (dense linear solvers). *IEEE Transactions on Computers* 50(10):1052–1070.

Beaumont, O., Boudet, V., Rastello, F., and Robert, Y. 2001b. Matrix multiplication on heterogeneous platforms. *IEEE Transactions on Parallel and Distributed Systems* 12(10): 1033–1051.

Bevilacqua, A. 1999. A dynamic load balancing method on a heterogeneous cluster of workstations. *Informatica* 23(1):49–56.

Blackford, L. S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., and Whaley, R. C. 1997. *ScaLAPACK Users' Guide*, SIAM, Philadelphia.

Bourgeade, A. and Nkonga, B. 2004. Dynamic load balancing computation of pulses propagating in a nonlinear medium. *Journal of Supercomputing* 28(3):279–294.

Brunie, L., Flory, A., and Kosch, H. 1995. New static scheduling and elastic load balancing methods for parallel query processing. *of the Basque International Workshop on Information Technology (BIWIT)*, San Sebastian, Spain, July, IEEE Computer Society Press, Los Alamitos, CA.

Buyya, R. 1999. *High Performance Cluster Computing. Volume 1: Architecture and Systems*, Prentice-Hall, Upper Saddle River, NJ.

Crandall, P. E. and Quinn, M. J. 1993. Block data decomposition for data-parallel programming on a heterogeneous workstation network. *Proceedings of the IEEE International Symposium on High Performance Distributed Computing (HPDC)*, Spokane, WA, July 20–23, IEEE Computer Society Press, Los Alamitos, CA, pp. 42–49.

Deelman, E. and Szymanski, B. 1998. Dynamic load balancing in parallel discrete event simulation for spatially explicit problems. *Proceedings of the Workshop on Parallel and Distributed Simulation (PADS)*, Banff, Alberta, Canada, May 26–29, IEEE Computer Society Press, Los Alamitos, CA, pp. 46–53.

Desprez, F., Dongarra, J., Petitet, A., Randriamaro, C., and Robert, Y. 1998. Scheduling block-cyclic array redistribution. *IEEE Transactions on Parallel and Distributed Systems* 9(2):192–205.

Flaherty, J. E., Loy, R. M., Özturan, C., Shephard, M. S., Szymanski, B. K., Teresco, J. D., and Ziantz, L. H. 1997a. Parallel structures and dynamic load balancing for adaptive finite element computation. *Applied Numerical Mathematics* 26(1–2):241–263.

Flaherty, J. E., Loy, R. M., Shephard, M. S., Szymanski, B. K., Teresco, J. D., and Ziantz, L. H. 1997b. Adaptive local refinement with octree load balancing for the parallel solution of three-dimensional conservation laws. *Journal of Parallel and Distributed Computing* 47(2):139–152.

Garcia, J., Ayguadé, E., and Labarta, J. 2001. A framework for integrating data alignment, distribution, and redistribution in distributed memory multiprocessors. *IEEE Transactions on Parallel and Distributed Systems* 12(4):416–431.

Hamdi, M. and Lee, C. 1995. Dynamic load balancing of data parallel applications on a distributed network. *Proceedings of ICS'95*, ACM, New York, pp. 170–179.

Hsu, C., Chung, Y., Yang, D., and Dow, C. 2001. A generalized processor mapping technique for array redistribution. *IEEE Transactions on Parallel and Distributed Systems* 12(7):743–757.

Hu, Y. and Blake, R. 1999. Load balancing for unstructured mesh applications. *Parallel and Distributed Computing Practices* 2(3).

Kaddoura, M., Ranka, S., and Wang, A. 1996. Array decomposition for nonuniform computational environments. *Journal of Parallel and Distributed Computing* 36:91–105.

Kalns, E. T. and Ni, L. M. 1995. Processor mapping techniques towards efficient data redistribution. *IEEE Transactions on Parallel and Distributed Systems* 6(12):1234–1247.

Knoop, J. and Mehofer, E. 2002. Distribution assignment placement: effective optimization of redistribution costs. *IEEE Transactions on Parallel and Distributed Systems* 13(6):628–647.

Koelbel, C. H., Loveman, D. B., Schreiber, R. S., Steele, G. L. Jr., and Zosel, M. E. 1994. *The High Performance Fortran Handbook*, MIT Press, Cambridge, MA.

Kremer, U. 1993. NP-completeness of dynamic remapping. *Proceedings of the 4th Workshop on Compilers for Parallel Computers*, Delft, the Netherlands. Also available as Rice Technical Report CRPC-TR93330-S.

Lan, Z., Taylor, V., and Bryan, G. 2001. Dynamic load balancing of samr applications on distributed systems. *Proceedings of the ACM/IEEE conference on Supercomputing*, Denver, CO, November 10–16, IEEE Computer Society Press, Los Alamitos, CA.

Lee, C. and Hamdi, M. 1995. Parallel image processing applications on a network of workstations. *Parallel Computing* 21:137–160.

Miguet, S. and Robert, Y. 1992. Elastic load balancing for image processing algorithms. *Parallel Computation, Lecture Notes in Computer Science* Vol. 591, H. Zima, editor, Springer-Verlag, Berlin, pp. 438–451.

Nicol, D. and Reynolds, J. P. F. 1990. Optimal dynamic remapping of data parallel computations. *IEEE Transactions on Computers* 39(2):206–219.

Nicol, D. and Saltz, J. 1988. Dynamic remapping of parallel computations with varying resource demands. *IEEE Transactions on Computers* 37(9):1073–1087.

Park, N., Prasanna, V., and Raghavendra, C. 1999. A framework for integrating data alignment, distribution, and redistribution in distributed memory multiprocessors. *IEEE Transactions on Parallel and Distributed Systems* 10(12):1217–1240.

Prylli, L. and Tourancheau, B. 1997. Fast runtime block-cyclic data redistribution on multiprocessors. *Journal of Parallel Distributed Computing* 45:63–72.

Renard, H., Robert, Y., and Vivien, F. 2004a. Data redistribution algorithms for homogeneous and heterogeneous processor rings. Research Report 5207, INRIA, May.

Renard, H., Robert, Y., and Vivien, F. 2004b. Data redistribution algorithms for homogeneous and heterogeneous processor rings. *Proceedings of High Performance Computing (HiPC 2004)*, Bangalore, India, December 19–22, *Lecture Notes in Computer Science* Vol. 3296, Springer-Verlag, Berlin, pp. 123–132.

Sarrut, D. and Miguet, S. 1999. ARAMIS: a remote access medical imaging system. *Proceedings of the International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE 99)*, San Francisco, CA, December 8–10, *Lecture Notes in Computer Science* Vol. 1732, Springer-Verlag, Berlin.

Schloegel, K., Karypis, G., and Kumar, V. 1997. Multilevel diffusion schemes for repartitioning of adaptive meshes. *Journal of Parallel and Distributed Computing* 47:109–124.

Schloegel, K., Karypis, G., and Kumar, V. 2000. A unified algorithm for load-balancing adaptive scientific simulations. *Proceedings of Supercomputing 2000*, Dallas, TX, November 4–10, IEEE Computer Society Press, Los Alamitos, CA.

Shirazi, B. A., Hurson, A. R., and Kavi, K. M. 1995. *Scheduling and Load Balancing in Parallel and Distributed Systems*, IEEE Computer Society Press, Los Alamitos, CA.

Thakur, R., Choudhary, A., and Ramanujam, J. 1996. Efficient algorithms for array redistribution. *IEEE Transactions on Parallel and Distributed Systems* 7(6):587–594.

Watts, J. and Taylor, S. 1998. A practical approach to dynamic load balancing. *IEEE Transactions on Parallel and Distributed Systems* 9(93):235–248.

Wu, M-Y. 1997. On runtime parallel scheduling for processor load balancing. *IEEE Transactions on Parallel and Distributed Systems* 8(2):173–186.