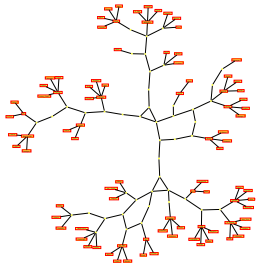

Fig. 1. The platform is composed of 90 machine nodes, connected through 192 communication links.

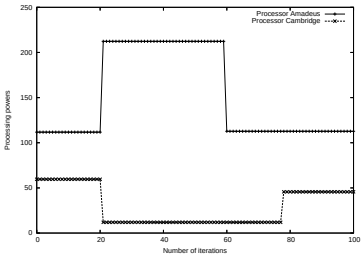

Fig. 2. Processing power of 2 sample machine nodes.

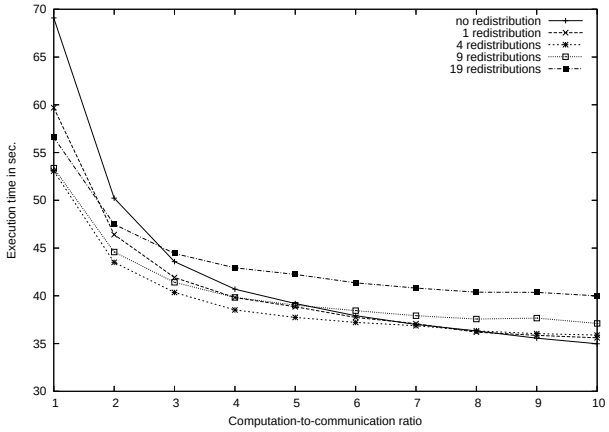

Fig. 3. Normalized execution time as a function of the computation-to-communication ratio, for a ring of 8 processors.

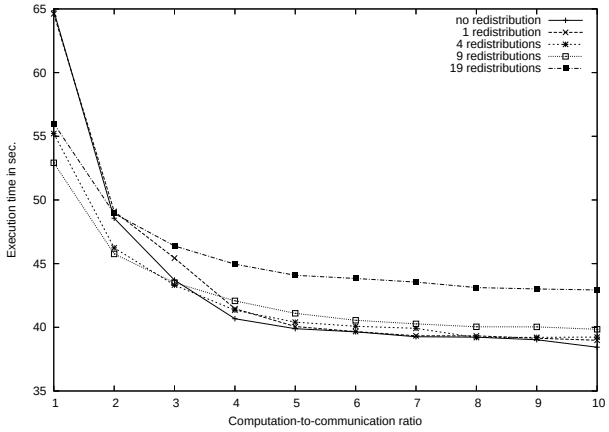

Fig. 4. Normalized execution time as a function of the ratio computation-to-communication, for a ring of 32 processors.

the cost of an iteration. When this parameter increases, iterations take more time, and the usefulness of a redistribution becomes more important.

In Figures 3 and 4, we plot the execution time of different computation schemes. Both figures report the same comparisons, but for different ring sizes: we use 8 processors in Figures 3, and 32 in Figures 4. As expected, when the processing power is high (ratio = 10 in the figures), the best strategy is to use no redistribution, as their cost is prohibitive. Conversely, when the processing power is low (ratio = 1 in the figures), it pays off to uses many redistributions, but not too many! As the ratio increases, all tradeoffs can be found.

8 Conclusion

In this paper, 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 [6] of correctness, both for unidirectional and bidirectional rings. The bidirectional algorithm turned out to be quite complex, and requires a lengthy proof [6].

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 provided 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. We have reported some simulation results for the most difficult combination, that of heterogeneous bi-directional rings. As expected, the cost of data redistributions may not pay off a little imbalance of the work in some cases. Further work will aim at investigating how frequently redistributions must occur in real-life applications.

References

1. R. P. Brent. The LINPACK Benchmark on the AP1000: Preliminary Report. In *CAP Workshop 91*. Australian National University, 1991. Website `http://www.netlib.org/linpack/`.
2. K. L. Calvert, M. B. Doar, and E. W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 35(6):160–163, June 1997. Available at `http://citeseer.nj.nec.com/calvert97modeling.html`.
3. M. Doar. A better model for generating test networks. In *Proceedings of Globecom '96*, Nov. 1996. Available at `http://citeseer.nj.nec.com/doar96better.html`.
4. A. B. Downey. Using pathchar to estimate internet link characteristics. In *Measurement and Modeling of Computer Systems*, pages 222–223, 1999. Available at http://citeseer.nj.nec.com/downey99using.html.
5. A. Legrand, L. Marchal, and H. Casanova. Scheduling Distributed Applications: The SimGrid Simulation Framework. In *Proceedings of the Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)*, May 2003.
6. H. Renard, Y. Robert, and F. Vivien. Data redistribution algorithms for heterogeneous processor rings. Research Report RR-2004-28, LIP, ENS Lyon, France, May 2004. Also available as INRIA Research Report RR-5207.

If we no longer assume the light redistribution hypothesis, we can still derive a lower bound, and use some heuristics [6]. However, 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 Experimental results

To evaluate the impact of the redistributions, we used the SimGrid [5] simulator to model an iterative application, implemented on a platform generated with the Tiers network generator [2,3]. We use the platform represented in Figure 1. The capacities of the edges are assigned using the classification of the Tiers generator (local LAN link, LAN/MAN link, MAN/WAN link,...). For each link type, we use values measured using `pathchar` [4] between some machines in ENS Lyon and some other machines scattered in France (Strasbourg, Lille, Grenoble, and Orsay), in the USA (Knoxville, San Diego, and Argonne), and in Japan (Nagoya and Tokyo).

We randomly select p processors in the platform to build the execution ring. The communication speed is given by the slowest link in the route from a processor to its successor (or predecessor) in the ring. The processing powers (CPU speeds) of the nodes are first randomly chosen in a list of values corresponding to the processing powers (expressed in MFlops and evaluated thanks to a benchmark taken from LINPACK [1]) of a wide variety of machines. But we make these speeds vary during the execution of the application.

We model an iterative application which executes during 100 iterations. At each iteration, independent data are updated by the processors. We may think of a $m \times n$ data matrix whose columns are distributed to the processors (we use $n = m = 1000$ in the experiment). Ideally, each processor should be allocated a number of columns proportional to its CPU speed. This is how the distribution of columns to processors is initialized. To motivate the need for redistributions, we create an imbalance by letting the CPU speeds vary during the execution. The speed of each processor changes two times, first at some iteration randomly chosen between iterations number 20 and 40, and then at some iteration randomly chosen between iterations number 60 and 80 for each node (see Figure 2 for an illustration). We record the values of each CPU speed in a SimGrid trace.

In the simulations, we use the heterogeneous bidirectional algorithm for light redistributions, and we test five different schemes, each with a given number of redistributions within the 100 iterations. The first scheme has no redistribution at all. The second scheme implements a redistribution after iteration number 50. The third scheme uses four redistributions, after iterations 20, 40, 60 and 80. The fourth scheme uses 9 redistributions, implemented every 10 iterations, and the last one uses 19 redistributions, implemented every 5 iterations. Given the shape of the CPU traces, some redistributions are likely to be beneficial during the execution. The last parameter to set is the computation-to-communication ratio, which amounts to set the relative (average) cost of a redistribution versus

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.

Under the "light redistribution" assumption, we can build an integer linear program to solve our problem (see System 2). Let $\mathcal{S}$ be a solution, 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 2 that $\mathcal{S}_{i,i+1}$ and $\mathcal{S}_{i,i-1}$ are nonnegative for all i, which imposes to use 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 imbalance. 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:

$$\text{Minimize } \tau, \text{ subject to}$$
$$\begin{cases} \mathcal{S}_{i,i+1} \geq 0 & 1 \leq i \leq n \\ \mathcal{S}_{i,i-1} \geq 0 & 1 \leq i \leq n \\ \mathcal{S}_{i,i+1} + \mathcal{S}_{i,i-1} - \mathcal{S}_{i+1,i} - \mathcal{S}_{i-1,i} = \delta_i & 1 \leq i \leq n \\ \mathcal{S}_{i,i+1} c_{i,i+1} + \mathcal{S}_{i,i-1} c_{i,i-1} \leq \tau & 1 \leq i \leq n \\ \mathcal{S}_{i+1,i} c_{i+1,i} + \mathcal{S}_{i-1,i} c_{i-1,i} \leq \tau & 1 \leq i \leq n \end{cases} \tag{2}$$

Lemma 4. *Any optimal solution of System 2 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.*

We use System 2 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 2 safely. But even if the "light redistribution" hypothesis holds, one 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 2 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, it turns out that one of the two natural ways of rounding always lead to an optimal (integer) solution [6]. The complexity of the light redistribution problem is therefore polynomial.

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

1: Let $\delta_{\max} = \max\{\max_{1\leq i\leq n} |\delta_i|, \max_{1\leq i\leq n, 1\leq l\leq n-1} \lceil \frac{|\delta_{i,i+l}|}{2} \rceil\}$
2: **if** $\delta_{\max} \geq 1$ **then**
3: **if** $\delta_{\max} \neq 2$ **then**
4: **for all** slice $C_{k,l}$ such that $\delta_{k,l} > 1$ and $\lceil \frac{|\delta_{k,l}|}{2} \rceil = \delta_{\max}$ **do**
5: P_k sends a data item to P_{k-1} during the time interval $[(s-1)\times c, s\times c[$.
6: P_l sends a data item to P_{l+1} during the time interval $[(s-1)\times c, s\times c[$.
7: **for all** slice $C_{k,l}$ such that $\delta_{k,l} < -1$ and $\lceil \frac{|\delta_{k,l}|}{2} \rceil = \delta_{\max}$ **do**
8: P_{k-1} sends a data item to P_k during the time interval $[(s-1)\times c, s\times c[$.
9: P_{l+1} sends a data item to P_l during the time interval $[(s-1)\times c, s\times c[$.
10: **else if** $\delta_{\max} = 2$ **then**
11: **for all** slice $C_{k,l}$ such that $\delta_{k,l} \geq 3$ **do**
12: P_l sends a data item to P_{l+1} during the time interval $[(s-1)\times c, s\times c[$.
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 $[(s-1)\times c, s\times c[$.
15: **for all** slice $C_{k,l}$ such that $\delta_{k,l} \leq -3$ **do**
16: P_{k-1} sends a data item to P_k during the time interval $[(s-1)\times c, s\times c[$.
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 $[(s-1)\times c, s\times c[$.
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 $[(s-1)\times c, s\times c[$ **then**
21: P_i sends a data item to P_{i+1} during the time interval $[(s-1)\times c, s\times c[$.
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 $[(s-1)\times c, s\times c[$ **then**
24: P_i receives a data item from P_{i-1} during the time interval $[(s-1)\times c, s\times c[$.
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 $[(s-1)\times c, s\times c[$ **then**
28: P_i sends a data item to P_{i+1} during the time interval $[(s-1)\times c, s\times c[$.
29: **if** P_{i+1} sends a data item to P_i during the time interval $[(s-1)\times c, s\times c[$ **then**
30: P_i sends a data item to P_{i-1} during the time interval $[(s-1)\times c, s\times c[$.
31: Recursive call to Algorithm 3 $(s+1)$

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 redistribution (what we call a *light* redistribution), then we succeed in deriving an optimal solution.

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 achieving this bound.

Lemma 3. *Let τ be the optimal redistribution time. Then:*

$$\tau \geq \max \left\{ \max_{1 \leq i \leq n} |\delta_i|, \max_{1 \leq i \leq n, 1 \leq l \leq n-1} \left\lceil \frac{|\delta_{i,i+l}|}{2} \right\rceil \right\} \times c. \tag{1}$$

The new (rightmost) term in this lower bound just states that a slice of processor can send (or receive) simultaneously at most two data items. 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 $\lceil \frac{|\delta_{k,l}|}{2} \rceil = \delta_{\max}$ 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 $\lceil \frac{|\delta_{k,l}|}{2} \rceil = \delta_{\max}$ and $\delta_{k,l} \leq 0$ must receive 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.

Theorem 3. *Algorithm 3 is optimal.*

Algorithm 1 Redistribution algorithm for homogeneous unidirectional rings

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 $C_{\texttt{start,end}}$ is of maximal imbalance: $\delta_{\texttt{start,end}} = \delta_{\max}$.
3: **for** $s = 1$ to $\delta_{\max}$ **do**
4: **for all** $l = 0$ to $n - 1$ **do**
5: **if** $\delta_{\texttt{start,start}+l} \geq s$ **then**
6: $P_{\texttt{start}+l}$ sends to $P_{\texttt{start}+l+1}$ a data item during the time interval $[(s-1) \times c, s \times c[$

exactly the same as in Algorithm 1 (namely $\delta_{\texttt{start},i}$). However, as the communication links have different capabilities, we no longer have a synchronous behavior. A processor P_i sends its $\delta_{\texttt{start},i}$ 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 than previously: P_i can send its data items during the time interval $[0, \delta_{\texttt{start},i} \times c_{i,i+1}[$. On the contrary, if P_i holds less data items than it has to send ($L_i < \delta_{\texttt{start},i}$), 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}.

Algorithm 2 Redistribution algorithm for heterogeneous unidirectional rings

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 $C_{\texttt{start,end}}$ is of maximal imbalance: $\delta_{\texttt{start,end}} = \delta_{\max}$.
3: **for all** $l = 0$ to $n - 1$ **do**
4: $P_{\texttt{start}+l}$ sends $\delta_{\texttt{start,start}+l}$ data items one by one and as soon as possible to processor $P_{\texttt{start}+l+1}$

The asynchronousness of Algorithm 2 implies that it is correct by construction: we wait for receiving a data item before sending. Furthermore, when the algorithm terminates, the redistribution is complete.

Lemma 2. *The running time of Algorithm 2 is*

$$\max_{0 \leq l \leq n-1} \delta_{start,start+l} \times c_{start+l,start+l+1}.$$

The result of Lemma 2 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.

Theorem 2. *Algorithm 2 is optimal.*

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 *imbalance* of P_k. We denote by $\delta_{k,l}$ the total imbalance 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 imbalance cannot be larger than the initial load: $L_k \geq \delta_k$. In fact, we suppose that any processor holds at least one data, both initially ($L_k \geq 1$) and after the redistribution ($L_k \geq 1 + \delta_k$): 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.

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

$$\tau \geq \left(\max_{1 \leq k \leq n,\ 0 \leq l \leq n-1} |\delta_{k,k+l}| \right) \times c.$$

Proof. The processor slice $C_{k,k+l} = P_k, P_{k+1}, \ldots, P_{k+l-1}, P_{k+l}$ has a total imbalance 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 $\delta_{k,k+l} \times c$ to send $\delta_{k,k+l}$ data items. Therefore, in any case, a redistribution scheme cannot take less than $\delta_{k,k+l} \times c$ to redistribute all data items. We have the same type of reasoning for the case $\delta_{k,k+l} < 0$.

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 below). In this algorithm, the amount of data items sent by any processor P_i is

those 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; 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 uni-dimensional data distribution calls for a uni-dimensional 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 uni- or bi-directional, 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, which includes 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. In the simplest case, that of a unidirectional homogeneous ring, we derive an optimal algorithm. Because the target architecture is quite simple, we are able to provide explicit (analytical) formulas 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 have to resort to heuristics based upon linear programming relaxation for the bidirectional case. 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 are introduced.

Due to the lack of space, the detailed proofs of correctness and optimality of the algorithms are not provided: please see the extended version [6]. Similarly, please refer to [6] for a survey of related work.

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 $1 \leq k, l \leq n$, $C_{k,l}$ 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 a 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,

Data redistribution algorithms for homogeneous and heterogeneous processor rings

Hélène Renard, Yves Robert and Frédéric Vivien

LIP, UMR CNRS-INRIA-UCBL 5668
ENS Lyon, France
{Helene.Renard | Yves.Robert | Frederic.Vivien}@ens-lyon.fr

Abstract. We consider the problem of redistributing data on homogeneous and heterogeneous processor rings. The problem arises in several applications, each time after a load-balancing mechanism is invoked (but we do not discuss the load-balancing mechanism itself). We provide algorithms that aim at optimizing the data redistribution, both for unidirectional and bi-directional 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 bi-directional heterogeneous ring, for which the problem remains open.

1 Introduction

In this paper, we consider the problem of redistributing data on homogeneous and heterogeneous rings of processors. The problem typically arises when a load balancing phase must be initiated. Because either of variations 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, ..., 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 under-loaded and should receive $-\delta_k$ 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