

Determining the optimal redistribution

Thomas Hérault, Julien Herrmann, Loris Marchal, Yves Robert

▶ **To cite this version:**

Thomas Hérault, Julien Herrmann, Loris Marchal, Yves Robert. Determining the optimal redistribution. [Research Report] RR-8499, INRIA. 2014. hal-00960452v2

HAL Id: hal-00960452
https://inria.hal.science/hal-00960452v2

Submitted on 19 Mar 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

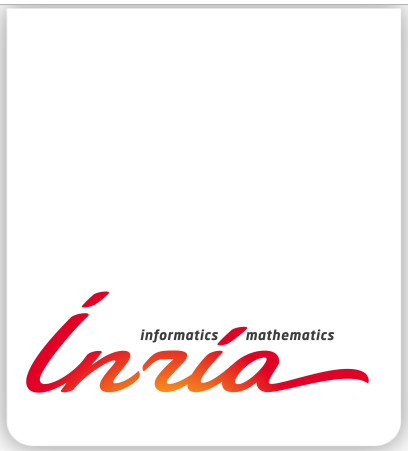


Determining the optimal redistribution

Thomas Hérault, Julien Herrmann, Loris Marchal, Yves Robert

RESEARCH REPORT
N° 8499
Mars 2014
Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--8499--FR+ENG



Determining the optimal redistribution

Thomas Hérault[†], Julien Herrmann[*], Loris Marchal[*], Yves Robert[*†]

Project-Team ROMA

Research Report n° 8499 — Mars 2014 — 24 pages

Abstract: The classical redistribution problem aims at optimally scheduling communications when moving from an initial data distribution $\mathcal{D}_{ini}$ to a target distribution $\mathcal{D}_{tar}$ where each processor P_i will host a subset $P(i)$ of data items. However, modern computing platforms are equipped with a powerful interconnection switch, and the cost of a given communication is (almost) independent of the location of its sender and receiver. This leads to generalizing the redistribution problem as follows: find the optimal permutation σ of processors such that P_i will host the set $P(\sigma(i))$, and for which the cost of the redistribution is minimal. This report studies the complexity of this generalized problem. We provide optimal algorithms and evaluate their gain over classical redistribution through simulations. We also show the NP-hardness of the problem to find the optimal data partition and processor permutation (defined by new subsets $P(\sigma(i))$) that minimize the cost of redistribution followed by a simple computation kernel.

Key-words: Scheduling, Redistribution, Heterogeneous resources, Stencil, Data distribution

[*] Ecole Normale Supérieure de Lyon, CNRS & INRIA, France
[†] University of Tennessee Knoxville, USA

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Déterminer la redistribution optimale

Résumé : Le problème de redistribution classique consiste à ordonnancer les communications de manière optimale lorsque l'on passe une distribution de données initiale $\mathcal{D}_{ini}$ à une distribution cible $\mathcal{D}_{tar}$ où chaque processeur P_i héberge un sous-ensemble $P(i)$ des données. Cependant, les plates-formes de calcul modernes sont équipées de puissants réseaux d'interconnexion programmables, et le coût d'une communication donnée est (presque) indépendant de l'emplacement de l'expéditeur et du récepteur. Cela conduit à généraliser le problème de redistribution comme suit: trouver la permutation optimale σ de processeurs telle que P_i héberge l'ensemble $P(\sigma(i))$, et telle que le coût de redistribution soit minimal. Ce rapport étudie la complexité de ce problème généralisé. Nous proposons des algorithmes optimaux et évaluons leur gain par rapport à la redistribution classique, via quelques simulations. Nous montrons aussi la NP-completude du problème consistant à trouver la partition de données optimale et la permutation des processeurs (définie par les nouveaux sous-ensembles $P(\sigma(i))$) qui minimise le coût de la redistribution suivie d'un noyau de calcul simple.

Mots-clés : Ordonnancement, Redistribution, Ressources hétérogènes, Stencil, Distribution de données

1 Introduction

In parallel computing systems, data locality has a strong impact on application performance. To achieve a good locality, a redistribution of the data may be needed between two different phases of the application, or even at the beginning of the execution, if the initial data layout is not suitable for performance. Data redistribution algorithms are critical to many applications, and therefore have received considerable attention. The data redistribution problem can be stated informally as follows: given N data items that are currently distributed across P processors, re-distribute them according to a different layout. Consider for instance a dense square matrix $A = (a_{ij})_{0 \leq i,j < n}$ of size n whose initial distribution is random and that must re-distributed into square blocks across along a $p \times p$ 2D-grid layout. A scenario for this problem is that the matrix has been generated by a Monte-Carlo method and is now needed for some matrix product $C \leftarrow C + AB$. Assume for simplicity that p divides n, and let $r = n/p$. In this example, $N = n^2$, $P = p^2$, and the redistribution will gather a block of $r \times r$ data elements on each processor. More precisely, all the elements of block $B_{i,j} = (a_{k,\ell})$, where $ri \leq k < (r+1)i$ and $rj \leq \ell < (r+1)j$, must be sent to processor $P_{i,j}$. This example illustrates the classical redistribution problem. Depending upon the cost model for communications, various optimization objectives have been considered, such as the total volume of data that is moved from one processor to another, or the total time for the redistribution, if several communications can take place simultaneously. We detail classical cost models in Section 2, which is devoted to related work.

Modern computing platforms are equipped with a powerful interconnection switch which permits to map the most usual interconnection graphs onto the physical network with reduced (or even negligible) dilation and contention. Continuing with the example, the $p \times p$ 2D-grid will be a virtual grid, meaning that the interconnection switch will emulate a 2D-grid. But the layout of the processors in the grid is completely flexible. For instance, the processors labeled $P_{1,1}$, $P_{1,2}$ and $P_{2,1}$ can be *any* processors in the platform, and we have the freedom to choose which three processors will indeed be labeled as the top-left corner processors of the virtual grid. Now, to describe the matrix product on the 2D-grid, we say that data will be sent *horizontally* between $P_{1,1}$ and $P_{1,2}$, and vertically between $P_{1,1}$ and $P_{2,1}$, but this actually means that these messages will be routed by the actual network, regardless of the physical position of the three processors in the platform.

This leads to revisit the redistribution problem, adding up the flexibility to select the *best* assignment of data to the processors (according to the cost model). The problem can be formulated as mapping a *partition* of the initial data onto the resources: there are P data subsets (the blocks in the example) to be assembled onto P processors, with a huge (exponential) number, namely $P!$, of possible mappings. An intuitive view of the problem is to assign the same color to all data items in a given subset (block), and to look for a coloring of the processors that will minimize the redistribution cost. For instance, if the data items in the first block $B_{0,0}$ are colored red, we may want to select the processor that initially holds the most red items as the target 'red' processor, i.e. the processor where elements of block $B_{0,0}$ are to be redistributed.

One major goal of this report is to assess the complexity of the problem of finding the best processor mapping for a given data partition, given an initial dis-

tribution of this data. This amounts to determine the processor assignment that minimizes the cost of redistributing the data according to the partition. There are $P!$ possible redistributions, and we aim at finding the one with minimal cost. In this report, we use the two most widely-used criteria in the literature to compute the cost of a redistribution:

- **Total volume.** In this model, the platform is not dedicated, and the objective is to minimize the total communication volume, i.e., the total number of data items that are sent from one processor to another. Minimizing this volume is likely to least disrupt the other applications that are running on the platform. Conceptually, this is equivalent to assuming that the network is a bus, globally shared by all resources.
- **Number of parallel steps.** In this model, the platform is dedicated to the application, and several communications can take place in parallel, provided that they involve different processor pairs. This is the one-port bi-directional model used in [1, 2]. The quantity to minimize is the number of parallel steps, where a step is a collection of unit-size messages that involve different processor pairs.

One major contribution of this report is the design of an optimal algorithm to solve this optimization problem for either criterion. We also provide various experiments to quantify the gain that results from choosing the optimal mapping rather than the *canonical* mapping where processors are labeled arbitrarily, and independently of the original data distribution.

As mentioned earlier, a redistribution is usually motivated by the need to efficiently execute a subsequent computational kernel. In most cases, there may well be many data partitions that are suitable to the efficient execution of this kernel. The optimal partition also depends upon the initial data redistribution. Coming back to the introductory example, where the redistribution is followed by a matrix product, we may ask whether a full block partition is absolutely needed? If the original data is distributed along a suitable, well-balanced distribution, a simple solution is to compute the product in place, using the owner-compute rule, that is, we let the processor holding $C_{i,j}$ compute all $A_{i,k}B_{k,j}$ products. This means that elements of A and B will be communicated during the computation, when needed. On the contrary, if the original distribution has a severe imbalance, with some processors holding many more data than others, a redistribution is very likely needed. But in this latter case, do we really need a perfect full block partition? In fact, the optimization problem is the following: given an initial data distribution, what is the best data partition, and the best mapping of this partition onto the processors, to minimize total execution time, defined as the sum of the redistribution time and of the execution of the kernel. Another major contribution of this report is to assess the complexity of this intricate problem. Finding the optimal partition mapping becomes NP-complete when coupling the redistribution with a simple computational kernel such as an iterative 1D-stencil kernel. Here the optimization objective is the sum of the redistribution time (computed using either of the two criteria above, with all communications serialized or with communications organized in parallel steps), and of the parallel execution time of a few steps of the stencil. Intuitively this confirms that determining the optimal data partition and its mapping is a difficult task. Stencil computations naturally favor block distributions, in order to

communicate only block frontiers at each iteration. But this has to be traded-off with the cost of moving the data from the initial distribution, with the number of iterations, and with the possible imbalance of the final redistribution that is chosen (whose own impact depend upon the communication-to-computation ratio of the machine). Altogether, it is no surprise that all these possibilities lead to a truly combinatorial problem.

The rest of the report is organized as follows. We survey related work in Section 2. We detail the model and formally state the optimization problems in Section 3. We deal with the problem of finding the best redistribution for a given data partition in Section 4. Sections 4.1 and 4.2 provide optimal algorithms, while Section 4.3 reports simulation results showing the gain over redistributing to an arbitrary compatible distribution. In Section 5, we couple the redistribution with a stencil kernel, and show that finding the optimal data partition, together with the corresponding redistribution, is NP-complete. We provide final remarks and directions for future work in Section 6.

2 Related work

2.1 Communication model

The *macro-dataflow model* has been widely used in the scheduling literature (see the survey papers [3, 4, 5, 6] and the references therein). In this model, the cost to communicate L bytes is $\alpha + L\beta$, where α is a start-up cost and β is the inverse of the bandwidth. In this report, we consider large, same-sized data items, so we can safely restrict to *unit* communications that involves a single data item; we integrate the start-up cost into the cost of a unit communication.

In the macro-dataflow model, communication delays from one task to its successor are taken into account, but communication resources are not limited. First, a processor can send (or receive) any number of messages in parallel, hence an unlimited number of communication ports is assumed (this explains the name *macro-dataflow* for the model). Second, the number of messages that can simultaneously circulate between processors is not bounded, hence an unlimited number of communications can simultaneously occur on a given link. In other words, the communication network is assumed to be contention-free, which of course is not realistic as soon as the processor number exceeds a few units.

A much more realistic communication model is the *one-port bidirectional model* where at a given time-step, any processor can communicate with at most one other processor in both directions: sending to and receiving from another processor. Several communications can occur in parallel, provided that they involve disjoint pairs of sending/receiving processors. The one-port model was introduced by Hollermann et al. [1], and Hsu et al. [2]. It has been widely used since both for homogeneous and heterogeneous platforms [7, 8].

2.2 Redistribution

The complexity of scheduling data redistribution in distributed architecture strongly depends on the network model. When the network has a general graph topology, achieving the minimal completion time for a set of communication is NP-complete, even when the time required to move any file along any link

is constant [9]. A common assumption is to consider a direct bidirectional link between each pair of devices. Most papers use the one-port bidirectional model, but several variants have also been considered. The first variant is a unidirectional one-port model, where a processor can participate in only one communication at a time (as a sender or a receiver); with this variant, the redistribution problem becomes NP-complete [10]. A second variant consists in assuming that each processor p has a number of ports $v(p)$ representing the maximum number of simultaneous file transfers that it can be participate to [11]. Finally, in a third variant [12], processors have memory constraints that must be enforced during the redistribution process.

2.3 Array redistribution

A specific class of redistribution problems has received a considerable attention, namely the redistribution of arrays that are distributed in a block-cyclic fashion over a multidimensional processor grid. This interest was originally motivated by the HPF [13] programming style, in which scientific applications are decomposed into phases. At each phase, there is an optimal distribution of the data arrays onto the processor grid. Typically, arrays are distributed according to a `CYCLIC(r)` pattern[1] along one or several dimensions of the grid. The best value of the distribution parameter r depends on the characteristics of the algorithmic kernel as well as on the communication-to-computation ratio of the target machine [14]. Because the optimal value of r changes from phase to phase and from one machine to another (think of a heterogeneous environment), runtime redistribution turns out to be a critical operation, as stated in [15, 16, 17] (among others). Communication are scheduled into parallel steps, which involve different processor pairs. The model comes in two variants, synchronous or asynchronous. In the synchronous variant, the cost of a parallel step is the maximal size of a message and the objective is to minimize the sum of the cost of the steps [16, 18]. In the asynchronous model, some overlap is allowed between communication steps [19]. Finally, the ScaLAPACK library provides a set of routines to perform array redistribution [20]. A total exchange is organized between processors, which are arranged as a (virtual) caterpillar. The total exchange is implemented as a succession of synchronous steps.

3 Model and framework

This section details the framework and formally states the optimization problems. We start with a few definitions.

3.1 Definitions

Consider a set of N data items (numbered from 0 to $N-1$) distributed onto P processors (numbered from 0 to $P-1$).

[1] The definition is the following: let an array $X[0...M-1]$ be distributed according to a block-cyclic distribution `CYCLIC(r)` onto a linear grid of P processors. Then element $X[i]$ is mapped onto processor $p = \lfloor i/r \rfloor \bmod P$, $0 \leq p \leq P-1$.

Definition 1 (Data distribution)**.** A *data distribution* $\mathcal{D}$ defines the mapping of the elements onto the processors: for each data item i, $\mathcal{D}(i)$ is the processor holding it.

Definition 2 (Data partition)**.** A *data partition* $\mathcal{P}$ associates to each data item i a partition $\mathcal{P}(i)$ ($0 \leq \mathcal{P}(i) \leq P-1$) so that, for a given index j, all data items i with $\mathcal{P}(i) = j$ reside on the same processor (not necessarily processor j).

It is straightforward to see that a data distribution $\mathcal{D}$ defines a single corresponding data partition (defined by $\mathcal{P} = \mathcal{D}$). However, a given data partition does not define a unique data distribution. On the contrary, any of the $P!$ permutations of $0, \ldots P-1$ can be used to map a data partition to the processors.

Definition 3 (Compatible distribution)**.** We say that a data distribution $\mathcal{D}$ is *compatible* with a data partition $\mathcal{P}$ if and only if there exists a permutation σ of $0, \ldots, P-1$ such that for all $0 \leq i \leq P-1$, $\mathcal{P}(i) = \sigma(\mathcal{D}(i))$.

3.2 Cost of a redistribution

In this section, we formally state the two metrics for the cost of a redistribution, namely the total volume and the number of parallel steps. Both metrics assume that the communication of one data item from one processor to another takes the same amount of time, regardless of the item and of the location of the source and target processors. Indeed, data items can be anything from single elements to matrix tiles, columns or rows, so that our approach is agnostic of the granularity of the redistribution. As already mentioned, modern interconnection networks are fully-connected switches, and they can implement any (same-length) communication in the same amount of time. Note that with asymmetric networks, it is always possible to use the worst-case communication time between any processor pair as the unit time for a communication.

3.2.1 Total volume

For this metric, we simply count the number of data items that are sent from one processor to another. This metric may be pessimistic if some parallelism is possible, but it provides an interesting measure of the overhead of the redistribution, especially if the platform is not dedicated.

Given an initial data distribution $\mathcal{D}_{ini}$ and a target distribution $\mathcal{D}_{tar}$, for $0 \leq i, j \leq P-1$, let $q_{i,j}$ be the number of data items that processor i must send to processor j: $q_{i,j}$ is the number of data items d such that $\mathcal{D}_{ini}(d) = i$ and $\mathcal{D}_{tar}(d) = j$. For a given processor i, let s_i (respectively r_i) be the total number of data items that processor i must send sent (respectively receive) during the redistribution. We have $s_i = \sum_{j \neq i} q_{i,j}$ and $r_i = \sum_{j \neq i} q_{j,i}$. The total communication volume of the redistribution is defined as

$$RedistVol(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) = \sum_i s_i = \sum_i r_i.$$

3.2.2 Number of parallel steps

With this metric, some communications can take place in parallel, provided that each of them involves a different processor pair (sender and receiver). This

communication model is the bidirectional one-port model introduced in [1, 2] and nicely accounts for contention when several communications take place simultaneously.

We define a parallel step as a set of unit-size communications (one data item each) such that all senders are different, and all receivers are different. Given an initial data distribution $\mathcal{D}_{ini}$ and a target distribution $\mathcal{D}_{tar}$, we define $\mathit{RedistSteps}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar})$ as the minimal number of parallel steps that are needed to perform the redistribution.

3.3 Optimization problems

We formally introduce the optimization problems that we study in Sections 4 and 5.

3.3.1 Best redistribution compatible with a given partition

In the optimization problems of Section 4, the data partition is given, and we aim at finding the best compatible target distribution (among $P!$ ones). More precisely, given an initial data distribution $\mathcal{D}_{ini}$ and a target data partition $\mathcal{P}_{tar}$, we aim at finding a data distribution $\mathcal{D}_{tar}$ that is compatible with $\mathcal{P}_{tar}$ and such that the redistribution cost from $\mathcal{D}_{ini}$ to $\mathcal{D}_{tar}$ is minimal. Since we have two cost metrics, we define two problems:

Definition 4 (VolumeRedistrib)**.** Given $\mathcal{D}_{ini}$ and $\mathcal{P}_{tar}$, find $\mathcal{D}_{tar}$ compatible with $\mathcal{P}_{tar}$ such that $\mathit{RedistVol}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar})$ is minimized.

Definition 5 (StepRedistrib)**.** Given $\mathcal{D}_{ini}$ and $\mathcal{P}_{tar}$, find $\mathcal{D}_{tar}$ compatible with $\mathcal{P}_{tar}$ such that $\mathit{RedistSteps}\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar})$ is minimized.

We show in Section 4 that both problems have polynomial complexity.

3.3.2 Best partition, and best compatible redistribution

In the optimization problems of Section 5, the data partition is no longer fixed. Given an initial data distribution $\mathcal{D}_{ini}$, we aim at executing some computational kernel whose cost $T_{comp}(\mathcal{P}_{tar})$ depends upon the data partition $\mathcal{P}_{tar}$ that will be selected. Note that this computational kernel will have the same execution cost for any distribution $\mathcal{D}_{tar}$ compatible with $\mathcal{P}_{tar}$, because of the symmetry of the target platform. However, the redistribution cost from $\mathcal{D}_{ini}$ to $\mathcal{D}_{tar}$ will itself depend upon $\mathcal{D}_{tar}$. We model the total cost as the sum of the time of the redistribution and of the computation. Letting τ_{comm} denote the time to perform a communication, the time to execute the redistribution is either $\mathit{RedistVol}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) \times \tau_{comm}$ or $\mathit{RedistSteps}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) \times \tau_{comm}$, depending upon the communication model. This leads us to the following two problems:

Definition 6 (VolPart&Redistrib)**.** Given $\mathcal{D}_{ini}$, find $\mathcal{P}_{tar}$, and $\mathcal{D}_{tar}$ compatible with $\mathcal{P}_{tar}$, such that $T_{total} = \mathit{RedistVol}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) \times \tau_{comm} + T_{comp}(\mathcal{P}_{tar})$ is minimized.

Definition 7 (StepPart&Redistrib)**.** Given $\mathcal{D}_{ini}$, find $\mathcal{P}_{tar}$, and $\mathcal{D}_{tar}$ compatible with $\mathcal{P}_{tar}$, such that $T_{total} = \mathit{RedistSteps}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) \times \tau_{comm} + T_{comp}(\mathcal{P}_{tar})$ is minimized.

Note that both problems require that we are able to compute $T_{comp}(\mathcal{P}_{tar})$ for any target data partition $\mathcal{P}_{tar}$. This is realistic only for very simple computational kernels. In Section 5, we consider such a kernel, namely the 1D-stencil. We show the NP-completeness of both VolPart&Redistrib and StepPart&Redistrib for this kernel, thereby assessing the difficulty to couple redistribution and computations.

4 Redistribution

This section deals with the VolumeRedistrib and StepRedistrib problems: given a data partition $\mathcal{P}_{tar}$ and an initial data distribution $\mathcal{D}_{ini}$, find one target distribution $\mathcal{D}_{tar}$ among all possible $P!$ compatible target distributions that minimizes the cost of the redistribution, either expressed in total volume or number of parallel steps. We show that both problems have polynomial complexity.

4.1 Total volume of communication

Theorem 1. *Given an initial data distribution $\mathcal{D}_{ini}$ and target data partition $\mathcal{P}_{tar}$, Algorithm 2 computes a data distribution $\mathcal{D}_{tar}$ compatible with $\mathcal{P}_{tar}$ such that $RedistVol(\mathcal{D}_{ini} \to \mathcal{D}_{tar})$ is minimized, and its complexity is $O(NP + P^3)$.*

Proof. The total volume of communication during the redistribution phase from the initial distribution to the target distribution is

$$RedistVol(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) = \sum_{0 \leq i \leq P-1} s_i = \sum_{0 \leq i \leq P-1} r_i$$

Solving VolumeRedistrib amounts to find a one-to-one perfect matching between each component of the target data partition and the processors, so that the total volume of communications is minimized. Algorithm 1 builds the complete bipartite graph where the two sets of vertices represents the P processors and the P components of the target data partition. Each edge (i, j) of this graph is weighted with the amount of data that processor P_i would have to receive if matched to component j of the data partition.

Computing the weight of the edges can be done with complexity $O(NP)$. The complexity of finding a minimum-weight perfect matching in a bipartite graph with n vertices and m edges is $O(n(m + n \log n)$ (see Corollary 17.4a in [21]). Here n=P and m=P^2, hence the overall complexity of Algorithm 1 is $O(NP + P^3)$. □

4.2 Number of parallel communication steps

The second metric is the number of parallel communications steps in the bidirectional one-port model. Note that this objective is quite different from the total communication volume: consider for instance a processor which has to sent and/or receive much more data than the others; all the communications involving this processor will have to be performed sequentially, creating a bottleneck.

Algorithm 1: BESTDISTRIBFORVOLUME

Data: Initial data distribution $\mathcal{D}_{ini}$ and target data partition $\mathcal{P}_{tar}$
Result: a data distribution $\mathcal{D}_{tar}$ compatible with the given data partition so that $\mathit{RedistVol}(\mathcal{D}_{ini} \to \mathcal{D}_{tar})$ is minimized
$A \leftarrow \{1, \dots P\}$ (set of processors)
$B \leftarrow \{1, \dots P\}$ (set of data partition components)
$G \leftarrow$ complete bipartite graph (V, E) where $V = A \cup B$
for *edge* (i, j) *in* E **do**
  $\mathit{weight}(i, j) \leftarrow |\{d \in \mathcal{P}_{tar}(j) \text{ s.t. } \mathcal{D}_{ini}(d) \neq i\}|$
$\mathcal{M} \leftarrow$ minimum-weight perfect matching of G
for $(i, j) \in \mathcal{M}$ **do**
  for $d \in \mathcal{P}_{tar}(j)$ **do** $\mathcal{D}_{tar}(d) \leftarrow i$
return $\mathcal{D}_{tar}$

Theorem 2. *Given an initial data distribution $\mathcal{D}_{ini}$ and target data partition $\mathcal{P}_{tar}$, Algorithm 2 computes a data distribution $\mathcal{D}_{tar}$ compatible with $\mathcal{P}_{tar}$ such that $\mathit{RedistSteps}(\mathcal{D}_{ini} \to \mathcal{D}_{tar})$ is minimized, and its complexity is $O(NP + P^{\frac{9}{2}})$.*

Proof. First, given an initial data distribution $\mathcal{D}_{ini}$ and a target distribution $\mathcal{D}_{tar}$, we can compute $\mathit{RedistSteps}(\mathcal{D}_{ini} \to \mathcal{D}_{tar})$ as

$$\mathit{RedistSteps}(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) = \max_{0 \leq i \leq P-1} max(s_i, r_i)$$

This well-known result [18] is a direct consequence of König's theorem (see Theorem 20.1 in [21]) stating that the edge-coloring number of a bipartite multigraph is equal to its maximum degree.

Algorithm 2 builds the complete bipartite graph G where the two sets of vertices represents the P processors and the P components of $\mathcal{P}_{tar}$. Each edge (i, j) of the complete bipartite graph is weighted with the maximum between the amount $r_{i,j}$ of data that processor i would have to receive if matched to component j of the data partition, and the amount of data that it would have to send in the same scenario. A one-to-one matching between the two sets of vertices whose maximal edge weight is minimal represents an optimal solution to STEPREDISTRIB. We denote by $\mathcal{M}_{opt}$ such a matching and m_{opt} its maximal edge weight. Since there are P processors and P components in $\mathcal{P}_{tar}$, the one-to-one matching $\mathcal{M}_{opt}$ is a matching of size P.

Algorithm 2 prunes an edge with maximum weight from G until it is not possible to find a matching of size P, and it returns the last matching of size P found. We denote by $\mathcal{M}_{ret}$ this matching and m_{ret} its maximal edge weight. Let us assume by contradiction that $m_{ret} > m_{opt}$. Then matching $\mathcal{M}_{opt}$ only contains edges with weight strictly smaller than m_{ret}. Since Algorithm 2 prunes edges starting from the heaviest ones, these edges are still in G when Algorithm 2 returns $\mathcal{M}_{ret}$. Thus we can remove the edges with maximal weight m_{ret} in $\mathcal{M}_{ret}$ and still have a matching of size P. This contradicts the stop condition of Algorithm 2. Thus $m_{ret} = m_{opt}$ and the matching returned by Algorithm 2 is a solution to STEPREDISTRIB.

Again, computing the edge weights can be done with complexity $O(NP)$. Algorithm 2 uses the Hopcroft–Karp Algorithm [22] to find the maximum cardinality matching of a bipartite graph $G = (V, E)$ in time $O(|E|\sqrt{|V|})$. There are

Algorithm 2: BESTDISTRIBFORSTEPS

Data: Initial data distribution $\mathcal{D}_{ini}$ and target data partition $\mathcal{P}_{tar}$
Result: A data distribution $\mathcal{D}_{tar}$ compatible with the given data partition so that $RedistSteps(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar})$ is minimized
$A \leftarrow \{1, \dots P\}$ (set of processors)
$B \leftarrow \{1, \dots P\}$ (set of data partition components)
$G \leftarrow$ complete bipartite graph (V, E) where $V = A \cup B$, **for** *edge* (i, j) *in* E **do**
  $r_{i,j} \leftarrow |\{d \in \mathcal{P}_{tar}(j) \text{ s.t. } \mathcal{D}_{ini}(d) \neq i\}|$
  $s_{i,j} \leftarrow |\{d \in \bigcup_{k \neq j} \mathcal{P}_{tar}(k) \text{ s.t. } \mathcal{D}_{ini}(d) = i\}|$
  $weight(i, j) \leftarrow \max(r_{i,j}, s_{i,j})$
$\mathcal{M} \leftarrow$ maximum cardinality matching of G (using the Hopcroft–Karp Algorithm)
while $|\mathcal{M}| \geq P$ **do**
  $\mathcal{M}_{save} \leftarrow \mathcal{M}$
  Suppress all edges of G with maximum weight
  $\mathcal{M} \leftarrow$ maximum cardinality matching of G (using the Hopcroft–Karp Algorithm)
return $\mathcal{M}_{save}$

no more than P^2 iterations in the while loop, and Algorithm 2 has a worst-case complexity of $O(NP + P^{\frac{9}{2}})$. □

4.3 Evaluation of optimal vs. arbitrary redistributions

In this section, we conduct several simulations to illustrate the interest of the two algorithms introduced above. In particular, we want to show that in many cases, it is important to optimize the mapping rather than resorting to an arbitrary mapping which could induce many more communications. Source code for the algorithms and simulations is publicly available at `http://perso.ens-lyon.fr/julien.herrmann/`.

4.3.1 Random balanced initial data distribution

First we consider a random balanced initial data distribution $\mathcal{D}_{ini}$ where each processor initially host D data items, and each data item has the same probability to reside on any processor. Most parallel applications require perfect load-balancing to achieve good performance, and thus a balanced data partition. Therefore, we consider here a balanced target data partition $\mathcal{P}_{tar}$ (each of the P components $\mathcal{P}_{tar}(j)$ includes D data items). We denote by $\mathcal{D}_{can}$ the canonical data distribution (compatible with partition $\mathcal{P}_{tar}$) which maps component $\mathcal{P}_{tar}(j)$ onto processor j.

As seen in Section 3, the volume of communication involved during the redistribution from $\mathcal{D}_{ini}$ to $\mathcal{D}_{can}$ is:

$$RedistVol(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{can}) = \sum_{0 \leq j \leq P-1} |\{d \in \mathcal{P}_{tar}(j) \text{ s.t. } \mathcal{D}_{ini}(d) \neq j\}|.$$

Since $|\mathcal{P}_{tar}(j)| = D$ for any processor j and $\mathcal{D}_{ini}(d)$ is equal to j with a probability $\frac{1}{P}$ for any processor j and any data item d, we can compute the average volume of communication:

$$E(RedistVol(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{can})) = D(P-1).$$

Thus, picking an arbitrary target distribution leads to a volume of communication linear in P.

Each processor hosts D data items at the beginning and at the end of the redistribution phase. Thus, according to Section 4.2, the number of steps required to schedule the redistribution phase is equal to D if and only if one of the P processors has to send its complete initial data set during the redistribution phase. This happens with probability

$$p = 1 - \left(1 - \left(\frac{P-1}{P}\right)^D\right)^P$$

This probability is equal to 0.986 for $P = 10$ and $D = 10$, and is non-decreasing with P, which means that the worst number of steps is reached in almost all cases for average values of D. This shows that idea most of the time, picking an arbitrary data distribution $\mathcal{D}_{can}$ is a bad. Instead, we can use Algorithm 1 to find the data distribution $\mathcal{D}_{vol}$ that minimizes the volume of communications involved in the redistribution phase and Algorithm 2 to find the data distribution $\mathcal{D}_{steps}$ that minimizes the number of steps of the redistribution phase. Figure 1 depicts the relative volume of communication and the relative number of redistribution steps when using target data distributions $\mathcal{D}_{vol}$ and $\mathcal{D}_{steps}$. The results are normalized with the performance of the arbitrary target distribution $\mathcal{D}_{can}$. The simulations have been conducted with $P = 32$ processors and up to $D = 20$ data items on each of them. For these values, the arbitrary target distribution $\mathcal{D}_{can}$ requires in average 620 communications and involves 20 parallel steps with a probability larger than $1 - 3.3 \times 10^{-11}$. Each point in Figure 1 represents the average results and the standard deviation on a set of 50 random initial distributions. We can see that the best data distributions for the communication volume and for the communication step represents a 10% improvement compared to an arbitrary target distribution when $D \geq 10$, and a larger improvement for smaller values of D. The results for these two data distributions are really close and present a small standard deviation.

4.3.2 Skewed balanced initial data distribution

Real world data distributions are usually not random. Some data are more likely to be initially hosted by some particular processor. In this section, we show the possible gain of using the proposed algorithms for skewed initial distributions. We consider a balanced target data partition $\mathcal{P}_{tar}$ where each of the P components $\mathcal{P}_{tar}(j)$ includes D elements of data. For $0 \leq \alpha \leq 1$, we note $\mathcal{D}^{\alpha}_{ini}$ the initial data distribution which maps $\lfloor \alpha D \rfloor$ data items in $\mathcal{P}_{tar}(j)$ on processor $(j+1) \mod P$, and which randomly maps the other $D - \lfloor \alpha D \rfloor$ data items to all P processors. Note that $\mathcal{D}^{0}_{ini}$ represents a random balanced data distribution as studied in previous section.

We still use $\mathcal{D}_{can}$, the arbitrary target distribution which maps component $\mathcal{P}_{tar}(j)$ onto processor j, as a comparison basis. During the redistribution phase

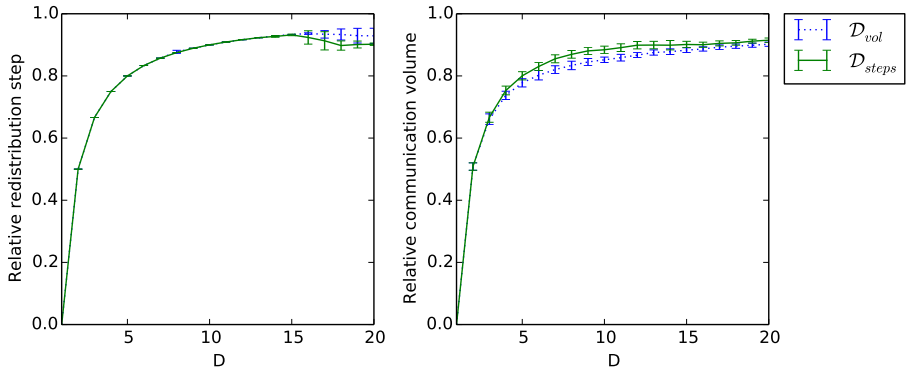


Figure 1: Performance of Algorithm 1 and 2 compared to the canonical distribution for a random initial distribution.

from $\mathcal{D}^{\alpha}_{ini}$ to $\mathcal{D}_{can}$, each processor sends at least $\lfloor \alpha D \rfloor$ of its elements. With the skewed distribution, we can compute the average volume of communication of $\mathcal{D}_{can}$:

$$E(RedistVol(\mathcal{D}^{\alpha}_{ini} \rightarrow \mathcal{D}_{can})) = D(P-1) + \lfloor \alpha D \rfloor.$$

The number of steps required to schedule the redistribution phase from $\mathcal{D}^{\alpha}_{ini}$ to $\mathcal{D}_{can}$ is equal to D with probability:

$$p_{\alpha} = 1 - \left(1 - \left(\frac{P-1}{P}\right)^{D-\lfloor \alpha D \rfloor}\right)^{P}.$$

Figure 2 depicts the relative volume of communication and the relative number of redistribution steps for the target distributions $\mathcal{D}_{vol}$ (obtained with Algorithm 1) and $\mathcal{D}_{steps}$ (obtained with Algorithm 2), normalized with the performance of the arbitrary target distribution $\mathcal{D}_{can}$. The simulations have been conducted with $P = 32$ processors, $D = 20$ elements of data on each of them and α varying from 0 to 1. When α is close to 0, $\mathcal{D}^{\alpha}_{ini}$ is close to a random balanced data distribution and we retrieve the results of the previous section. When α is larger than 0.2, for every component $\mathcal{P}_{tar}(j)$, the proportion of data in $\mathcal{P}_{tar}(j)$ that are initially hosted by processor $(j+1) \mod P$ is significant. Thus, mapping $\mathcal{P}_{tar}(j)$ onto processor $(j+1) \mod P$ becomes the best solution to reduce both the volume of communication and the number of communication steps. We can see that, in this case, Algorithm 1 and Algorithm 2 provide the same target data distribution. Both objectives decrease linearly with α since the proportion of data that are initially mapped onto the correct processor increases linearly with α.

5 Coupling redistribution and stencil computations

In this section, we focus on a simple, yet realistic, application to assess the complexity of redistribution when coupled to a computational kernel. We consider a 1D-stencil iterative algorithm, which updates in parallel each element of an

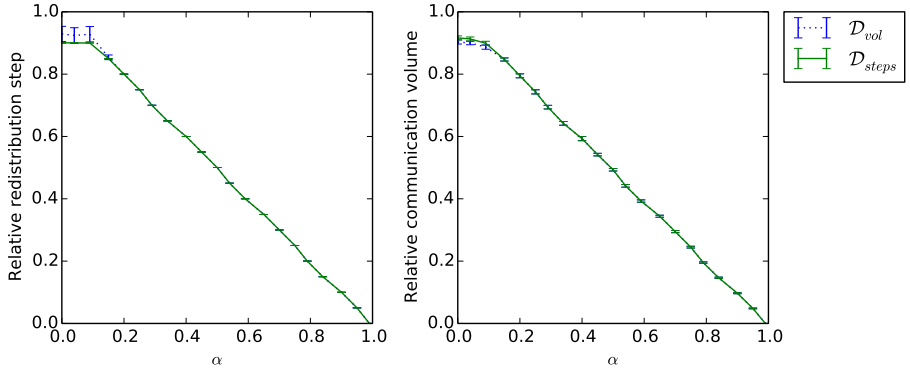


Figure 2: Performance of Algorithm 1 and 2 compared to the canonical distribution for a skewed initial distribution.

Algorithm 3: One iteration of the unidimensional stencil algorithm

Result: N data items numbered from 0 to $N-1$ and their distribution $\mathcal{D}$ on P processors

for $0 \leq d \leq N-1$ *in parallel* **do**
  $\ell_d \leftarrow (d-1) \mod N$;
  $r_d \leftarrow (d+1) \mod N$;
  if $\mathcal{D}(\ell_d) \neq \mathcal{D}(d)$ **then**
    Processor $\mathcal{D}(d)$ receives data item ℓ_d from processor $\mathcal{D}(\ell_d)$;
  if $\mathcal{D}(r_d) \neq \mathcal{D}(d)$ **then**
    Processor $\mathcal{D}(d)$ receives data item r_d from processor $\mathcal{D}(r_d)$;

for $0 \leq d \leq N-1$ *in parallel* **do**
  Processor $\mathcal{D}(d)$ updates data item d using ℓ_d and r_d;

array, according to the value of its direct neighbors. Stencil computations are widely used to numerically solve partial differential equations [23].

5.1 Application model

We consider here a three-point stencil with circular arrangement of the data. More precisely, to compute the value $x(i,t)$ of the data at position i at step t, we need its value and those of its left and right neighbors at the previous step, namely $x(i,t-1)$, $x(i-1 \mod N, t-1)$, and $x(i+1 \mod N, t-1)$. If the neighbors are not stored on the same processor, their value has to be received from the processors hosting them. Thus, each iteration of the stencil algorithm consists in two phases, the *communication phase* when the value of each data item is sent to the processors hosting its neighbors, and the *computation phase*, when each data item is updated according to a given kernel using these values (see Algorithm 3). The update kernel depends on the application.

Given a data partition $\mathcal{P}_{tar}$, let $N_{i,j}$ be the number of data items sent by the processor hosting subset $\mathcal{P}_{tar}(i)$ to the processor hosting subset $\mathcal{P}_{tar}(j)$ during one communication phase of the stencil algorithm: $N_{i,j}$ is the number of left or

right neighbors in $\mathcal{P}_{tar}(i)$ of data items in $\mathcal{P}_{tar}(j)$), and:

$$N_{i,j} = |\{0 \leq d \leq N-1 \text{ s.t. } \mathcal{P}_{tar}(d-1) = j \text{ or } \mathcal{P}_{tar}(d+1) = j\}|.$$

The workload ℓ_i of the processor hosting subset $\mathcal{P}_{tar}(i)$ is:

$$\ell_i = |\{0 \leq d \leq N-1 \text{ s.t. } \mathcal{D}(d) = i\}|.$$

Given a data partition $\mathcal{P}_{tar}$, the running time of the stencil algorithm depends on the communication model, but not on the actual data distribution, provided that it is compatible with $\mathcal{P}_{tar}$. Let τ_{comm} be the time needed to perform one communication (see Section 3.3), and let τ_{calc} be the time needed to perform one data update for the considered stencil application. The processing time for K iterations of the stencil with the two communication models is the following (using the notations of Section 3.3):

- **Total volume:** For problem VolPart&Redistrib, $T_{comp}(\mathcal{P}_{tar}) = KT^{iter}_{vol}(\mathcal{P}_{tar})$, where

$$T^{iter}_{vol}(\mathcal{P}_{tar}) = \tau_{comm} \times \sum_{0 \leq i \leq P-1} \sum_{j} N_{ij} + \tau_{calc} \times \max_{0 \leq i \leq P-1} \ell_i$$

  The first term corresponds to the serialization of all communications, and the second one to the parallel processing of the updates.

- **Number of parallel steps:** For problem StepPart&Redistrib, $T_{comp}(\mathcal{P}_{tar}) = KT^{iter}_{steps}(\mathcal{P}_{tar})$, where

$$\begin{aligned} T^{iter}_{steps}(\mathcal{P}_{tar}) = \quad & \tau_{comm} \times \max_{0 \leq i \leq P-1} \left(\sum_{j} N_{ij}, \sum_{j} N_{ji} \right) \\ & + \tau_{calc} \times \max_{0 \leq i \leq P-1} \ell_i \end{aligned}$$

5.2 Complexity

Assume without loss of generality that N is a multiple of P. There is a well-known optimal data partition for the 1D-stencil kernel, namely the full block partition (data item i is assigned to subset $\lfloor iP/N \rfloor$). This partition minimizes the duration of the communication phase (only two items are sent:received) and the computation phase is perfectly balanced.

Starting from an initial data distribution $\mathcal{D}_{ini}$, we can use either Algorithm 1 or 2 to find a target distribution $\mathcal{D}_{tar}$ which is compatible with the full-block partition and whose redistribution cost is minimal. However, redistributing from $\mathcal{D}_{ini}$ to $\mathcal{D}_{tar}$ may induce a large overhead on the total execution time, and it fully justified only when the number of iterations K is large enough. It may be useful to avoid a costly redistribution for small values of K and to find a target redistribution which is a trade-off between minimizing redistribution time and processing time. Actually, finding such a trade-off distribution is an NP-complete problem for both communication models:

Theorem 3. VolPart&Redistrib *problem with the 1D-stencil kernel is strongly NP complete.*

Proof. The problem clearly belongs to NP: given a new distribution $\mathcal{D}_{tar}$ of data, it is possible to compute the redistribution time and the cost of the K iteration of the stencil algorithm.

To establish the completeness, we use a reduction from the 3-Partition problem [24], which is known to be NP-complete in the strong sense. We consider the following instance $Inst_1$ of the 3-Partition problem: let a_i be $3m$ integers and B an integer such that $\sum a_i = mB$. We consider the variant of the problem, also NP-complete, where $\forall i, B/4 < a_i < B/2$. To solve $Inst_1$, we need to solve the following question: does there exist a partition of the a_i's in m subsets $S_1, ..., S_m$, each containing exactly 3 elements, such that, $\forall S_k, \sum_{i \in S_k} a_i = B$.

We build the following instance $Inst_2$ of the VolPart&Redistrib problem, illustrated on Figure 3. Figure 3 represents the initial data distribution $\mathcal{D}_{ini}$ of $96mB$ elements on $12m$ different processors. To clarify the proof we split the $12m$ processors into 4 different types. There are $3m$ processors of type 1, m processors of type 2, $4m$ processors of type 2 and $4m$ processors of type 2. Processors of type k are denoted by $P_i^{(k)}$. As depicted on Figure 3, we can see that, for example, the $2B$ first consecutive elements are stored on $P_1^{(1)}$, the first processor of type 1. The next $2B$ elements are stored on $P_1^{(4)}$, the first processor of type 4. We set $K = 1$, $\alpha = B^2$, and $T_{\text{MAX}} = 8 + 5mB + 8B^3$ for the cost of the unidimensional stencil algorithm. The construction of $Inst_2$ is polynomial in the size of $Inst_1$. Let show that $Inst_2$ has a solution if and only if $Inst_1$ has a solution.

Assume first that $Inst_2$ has a solution and let $\mathcal{D}_{tar}$ be the final distribution of data. Let C_p be the number of maximal connected components on processor p for this distribution. Thus:

$$T_{vol}^{stencil}(\mathcal{D}_{ini}, \mathcal{D}_{tar}) = \tau_{comm} \times RedistVol(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) + 2 \times \max_p C_p + B^2 \times \max_p |\mathcal{D}_{tar}(p)| \leq 8 + 5mB + 8B^3 \quad (1)$$

We first show that $\forall \mathbf{p},\ |\mathcal{D}_{tar}(\mathbf{p})| = \mathbf{8B}$:

- $\max_p |\mathcal{D}_{tar}(p)| \leq 8B$ because otherwise:
$$T_{vol}^{stencil}(\mathcal{D}_{ini}) \geq B^2 \times (8B + 1) > T_{\text{MAX}}.$$
- There are a total of $96mB$ elements of data and $12m$ processors, thus $\forall p,\ |\mathcal{D}_{tar}(p)| = 8B$.

Thus $\max_p |\mathcal{D}_{tar}(p)| = 8B$ and Equation 1 becomes:

$$RedistVol(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) + 8 \max_p C_p \leq 8 + 5mB \quad (2)$$

Then we show that $RedistVol(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) = 5mB$:

- Initially, in $\mathcal{D}_{ini}$, the type-2 and type-3 processors hosts $7B$ elements of data and since $\forall p,\ |\mathcal{D}_{tar}(p)| = 8B$ in $\mathcal{D}_{tar}$, the type-2 and type-3 processors each have to received at least B elements of data during the redistribution phase. There are $5m$ of them. Thus at least $5mB$ elements of data have to be communicated during the redistribution phase: $RedistVol(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) \geq 5mB$.

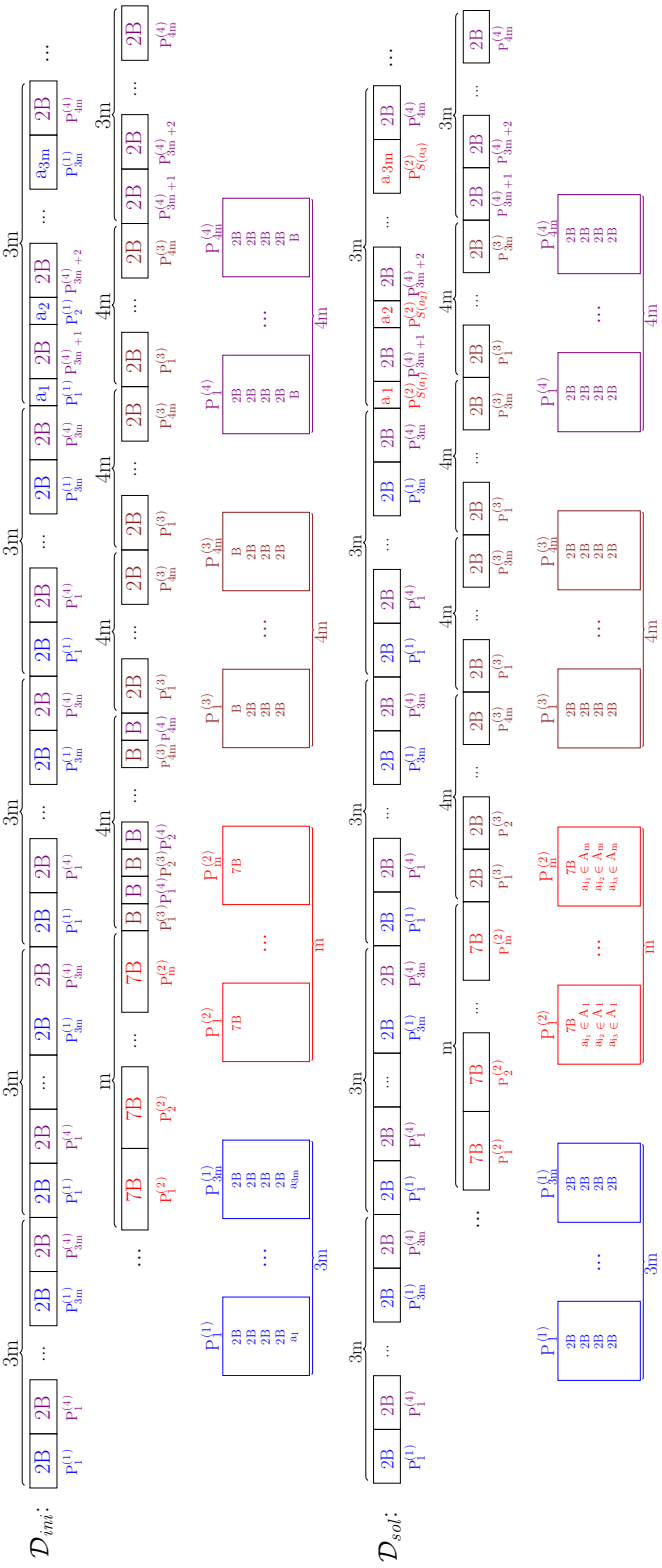


Figure 3: $\mathcal{D}_{ini}$ and $\mathcal{D}_{sol}$ in the proof of Theorems 3 and 4

- If $5mB + 1$ elements of data are communicated during the redistribution phase, we would have $\mathit{RedistVol}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) \geq 5mB + 1$ and $2 \times \max_p C_p \leq 7$, so $\max_p C_p \leq 3$. Initially, in $\mathcal{D}_{ini}$, type-1 processors host 5 maximal connected components and could have at most 3 connected components in $\mathcal{D}_{tar}$. There are only two different ways to decrease the number of maximal connected components in a processor: sending one entire maximal connected components to another processor or connecting two existing components by receiving all the data between them. Both options are impossible in this case, because type-1 processors each would have to send or receive more than $2B$ elements during the redistribution phase ($6mB$ elements in total), which is impossible according to Equation 2.

Thus $\mathit{RedistVol}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) = 5mB$ and Equation 2 becomes:

$$\max_p C_p \leq 4 \tag{3}$$

We now bound the number of elements sent and received by processors of type 2, 3 and 4. For each processor $P_i^{(k)}$, we note $S_i^{(k)}$ (respectively $R_i^{(k)}$) the amount of elements the processor $P_i^{(k)}$ sends (respectively receives) during the redistribution phase. We naturally have

$$\sum_{k,i} S_i^{(k)} = \sum_{k,i} R_i^{(k)} = \mathit{RedistVol}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) = 5mB.$$

- Each type-2 processor $P_i^{(2)}$ and each type-3 processor $P_i^{(3)}$ hosts $7B$ elements in the initial distribution $\mathcal{D}_{ini}$, and $8B$ elements in the final distribution $\mathcal{D}_{tar}$. Thus, they each have to receive at least B elements of data. There are $5m$ of them so they can receive only B elements of data each and no other processors can receive any data. We have $\forall i$: $R_i^{(1)} = 0$, $R_i^{(2)} = B$, $R_i^{(3)} = B$ and $R_i^{(4)} = 0$.

- Each type-1 processor $P_i^{(1)}$ hosts $8B+a_i$ elements in $\mathcal{D}_{ini}$, and $8B$ elements in $\mathcal{D}_{tar}$. Each type-4 processor $P_i^{(4)}$ hosts $9B$ elements in $\mathcal{D}_{ini}$, and $8B$ elements in $\mathcal{D}_{tar}$. Again, this means that each type-1 processor $P_i^{(1)}$ can send only a_i elements of data, each type-4 processor $P_i^{(4)}$ can send only B elements of data and no other processors can send any data. We have $\forall i$: $S_i^{(1)} = a_i$, $S_i^{(2)} = 0$, $S_i^{(3)} = 0$ and $S_i^{(4)} = B$.

Initially, in $\mathcal{D}_{ini}$, each type-3 processor $P_i^{(3)}$ hosts 4 maximal connected components and can have at most 4 maximal connected components in $\mathcal{D}_{tar}$, and it has to receive B elements. There are only two different ways to decrease the number of maximal connected components in a processor: sending one connected components to another processor or connecting two existing components by receiving all the data between them. The first option is impossible since we have shown that processor $P_i^{(3)}$ can not send any data during the redistribution phase ($S^{(3)} = 0$). The second option appears to be impossible too because each maximal connected components of $P_i^{(3)}$ are separated by strictly more than B

elements and we have shown that processor $P_i^{(3)}$ can only receive B elements during the redistribution phase ($R_i^{(3)} = B$). Thus, during the redistribution phase, each processor $P_i^{(3)}$ has to receive B elements without increasing its number of connected components. The only way to do so is to receive the data from its direct neighbors in the distribution $\mathcal{D}_{ini}$. The only neighbors of $P_i^{(3)}$ in $\mathcal{D}_{tar}$ are some processors of type 2 (that can not send any data), some other processors of type 3 (that can not send any data) and some processors of type 4. So only processors of type 4 can send data to processors of type 3. Since each of the $4m$ type-3 processors has to receive exactly B elements and each of the $4m$ type-4 processors has to send exactly B elements, we know that during the redistribution phase, type-4 processors only sends data to type-3 processors (which only receive from type-4 processors). In particular, **type-1 and type-2 processors do not send or receive any data to or from a type-3 or type-4 processor**.

Gathering all the results shown above, we can state that type-1 processors can only send their a_i elements to type-2 processors during the redistribution phase. If a processor $P_i^{(1)}$ splits its a_i consecutive elements and send them to two different type-2 processors, this would create an extra maximal connected component on the type-2 processors. Since each of the m type-2 processors hosts one maximal connected component initially and has to host less than 4 maximal connected components in $\mathcal{D}_{tar}$, they can only receive 3 maximal connected components each, meaning that **each type-1 processor has to send its $\mathbf{a_i}$ elements to the same type-2 processor**.

Let A_k be the set of the size of the maximal connected components received by $P_k^{(2)}$ during the redistribution phase. The A_k sets represent a partition of the a_i's and the cardinality of each set A_k is exactly 3. Finally, $\forall k, \sum_{a_i \in A_k} a_i = R_k^{(2)} = B$, which means that the A_ks are a solution of $Inst_1$.

Suppose now that $Inst_1$ has a solution. Let A_k be the 3-Partition of the integers a_i and consider the distribution $\mathcal{D}_{sol}$ described in Figure 3. To perform the redistribution from $\mathcal{D}_{ini}$ to $\mathcal{D}_{sol}$, each type-2 and type-3 processors have to send receive B elements of data, which means that $RedistVol(\mathcal{D}_{ini} \to \mathcal{D}_{sol}) = 5mB$. In addition, in $\mathcal{D}_{sol}$ each processor hosts $8B$ elements divided in 4 maximal connected components. Thus, $T_{vol}^{iter}(\mathcal{D}_{sol}) = 2 \times 4 + B \times 8B^2 = 8 + 8B^3$ and $T_{vol}^{stencil}(\mathcal{D}_{sol}) = 8 + 5mB + 8B^3$, which means that $Inst_2$ has a solution and concludes the proof. □

Theorem 4. StepPart&Redistrib *problem with the 1D-stencil kernel is strongly NP complete.*

Proof. The problem clearly belongs to NP, and the certificate is the new distribution $\mathcal{D}_{tar}$ of data; it is easy to compute the minimum time $T_{\text{redist}}^{V}(\mathcal{D}_{ini} \to \mathcal{D}_{tar})$ needed to go from the distribution $\mathcal{D}_{tar}$ to $\mathcal{D}_{ini}$ and to compute the cost of K steps of the 1D Stencil.

To establish the completeness, we use a reduction from the 3-Partition problem [24], which is known to be NP-complete in the strong sense. We consider the following instance $Inst_1$ of the 3-Partition problem: let a_i be $3m$ integers and B an integer such that $\sum a_i = mB$. We consider the variant of the problem, also NP-complete, where $\forall i, B/4 < a_i < B/2$. To solve $Inst_1$, we need to

solve the following question: does there exist a partition of the a_i's in m subsets $S_1, ..., S_m$, each containing exactly 3 elements, such that, $\forall S_k, \sum_{i \in S_k} a_i = B$.

We build the following instance $Inst_2$ of the StepPart&Redistrib problem, illustrated on Figure 3. Figure 3 represents the initial data distribution $\mathcal{D}_{ini}$ of $96mB$ elements on $12m$ different processors. To clarify the proof we split the $12m$ processors into 4 different types. There are $3m$ processors of type 1, m processors of type 2, $4m$ processors of type 2 and $4m$ processors of type 2. Processors of type k are denoted by $P_1^{(k)}, P_2^{(k)}, \ldots$. As depicted on Figure 3, we can see that, for example, the $2B$ first consecutive elements are stored on the first processor of type 1: $P_1^{(1)}$ and the next $2B$ elements are stored on the first processor of type 4: $P_1^{(4)}$. We set $K = 1$, $\tau_{comm} = 1$, $\tau_{calc} = B^2$, and $T_{\text{MAX}} = 8 + B + 8B^3$ for the cost of the 1D Stencil Algorithm. The construction of $Inst_2$ is polynomial in the size of $Inst_1$. Let show that $Inst_2$ has a solution if and only if $Inst_1$ has a solution.

Assume first that $Inst_2$ has a solution and let $\mathcal{D}_{tar}$ be the final distribution of data. We note $\mathcal{D}_{tar}(p)$ the set of elements stored on processor p for the distribution $\mathcal{D}_{tar}$ and C_p the number of maximal connected components on processor p. Thus:

$$T_{steps}^{stencil}(\mathcal{D}_{ini}, \mathcal{D}_{tar}) = \mathit{RedistSteps}(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) + 2 \times \max_p C_p + B^2 \times \max_p |\mathcal{D}_{tar}(p)| \leq 8 + B + 8B^3 \quad (4)$$

We first show that $\forall \mathbf{p},\ |\mathcal{D}_{tar}(\mathbf{p})| = \mathbf{8B}$:

- $\max_p |\mathcal{D}_{tar}(p)| \leq 8B$ because otherwise:

$$T_{steps}^{stencil}(\mathcal{D}_{ini}) \geq B^2 \times (8B + 1) > T_{\text{MAX}}.$$

- There are a total of $96mB$ elements of data and $12m$ processors, thus $\forall p,\ |\mathcal{D}_{tar}(p)| = 8B$.

Thus $\max_p |\mathcal{D}_{tar}(p)| = 8B$ and Equation 4 becomes:

$$\mathit{RedistSteps}(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) + 8 \max_p C_p \leq 8 + B \quad (5)$$

Then we show that $\mathit{RedistSteps}(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) = B$:

- Initially, in $\mathcal{D}_{ini}$, the processor $P_1^{(4)}$ hosts $9B$ elements of data and since $\max_p |\mathcal{D}_{tar}(p)| = 8B$ in $\mathcal{D}_{tar}$, the processor $P_1^{(4)}$ has to send at least B elements of data during the redistribution phase. So $\mathit{RedistSteps}(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) \geq B$.
- If one processor send $B + 1$ elements of data during the redistribution phase, we would have $\mathit{RedistSteps}(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) \geq B+1$ and $2 \times \max_p C_p \leq 7$, so $\max_p C_p \leq 3$. Initially, in $\mathcal{D}_{ini}$, processor $P_1^{(1)}$ hosts 5 maximal connected components and could have at most 3 maximal connected components in $\mathcal{D}_{tar}$. There are only two different ways to decrease the number of

maximal connected components in a processor: sending one entire maximal connected components to another processor or connecting two existing components by receiving all the data between them. Both options are impossible in this case, because processor $P_1^{(1)}$ would have to send or receive more than $2B$ elements during the redistribution phase, which is impossible according to Equation 5.

Thus $\mathit{RedistSteps}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) = B$ and Equation 5 becomes:

$$\max_p C_p \leq 4 \tag{6}$$

We now bound the number of elements sent and received by processors of type 2, 3 and 4. For each processor $P_i^{(k)}$, we note $S_i^{(k)}$ (respectively $R_i^{(k)}$) the amount of elements the processor $P_i^{(k)}$ sends (respectively receives) during the redistribution phase. We naturally have

$$\forall P_i^{(k)},\ max(S_i^{(k)}, R_i^{(k)}) \leq \mathit{RedistSteps}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) = B.$$

- Each type-2 processor $P_i^{(2)}$ hosts $7B$ elements in the initial distribution $\mathcal{D}_{ini}$, and $8B$ elements in the final distribution $\mathcal{D}_{tar}$. This means that $R_i^{(2)} - S_i^{(2)} = B$. Since $max(S_i^{(2)}, R_i^{(2)}) \leq B$, we have: $\mathbf{R_i^{(2)}} = \mathbf{B}$ and $\mathbf{S_i^{(2)}} = \mathbf{0}$.

- Each type-3 processor $P_i^{(3)}$ hosts $7B$ elements in $\mathcal{D}_{ini}$, and $8B$ elements in $\mathcal{D}_{tar}$. Again, this means that $R_i^{(3)} - S_i^{(3)} = B$ and since $max(S_i^{(3)}, R_i^{(3)}) \leq B$, we necessarily have: $\mathbf{R_i^{(3)}} = \mathbf{B}$ and $\mathbf{S_i^{(3)}} = \mathbf{0}$.

- Each type-4 processor $P_i^{(4)}$ hosts $9B$ elements in $\mathcal{D}_{ini}$, and $8B$ elements in $\mathcal{D}_{tar}$. Again, $S_i^{(4)} - R_i^{(4)} = B$ and we have $\mathbf{R_i^{(4)}} = \mathbf{0}$ and $\mathbf{S_i^{(4)}} = \mathbf{B}$.

Initially, in $\mathcal{D}_{ini}$, each type-3 processor $P_i^{(3)}$ hosts 4 maximal connected components and can have at most 4 maximal connected components in $\mathcal{D}_{tar}$, and it has to receive B elements. There are only two different ways to decrease the number of maximal connected components in a processor: sending one connected components to another processor or connecting two existing components by receiving all the data between them. The first option is impossible since we have shown that processor $P_i^{(3)}$ can not send any data during the redistribution phase ($S^{(3)} = 0$). The second option appears to be impossible too because each maximal connected components of $P_i^{(3)}$ are separated by strictly more than B elements and we have shown that processor $P_i^{(3)}$ can only receive B elements during the redistribution phase ($R_i^{(3)} = B$). Thus, during the redistribution phase, each processor $P_i^{(3)}$ has to receive B elements without increasing its number of connected components. The only way to do so is to receive the data from its direct neighbors in the distribution $\mathcal{D}_{ini}$. The only neighbors of $P_i^{(3)}$ in $\mathcal{D}_{tar}$ are some processors of type 2 (that can not send any data), some other processors of type 3 (that can not send any data) and some processors of type 4. So only processors of type 4 can send data to processors of type 3. Since

each of the $4m$ type-3 processors has to receive exactly B elements and each of the $4m$ type-4 processors has to send exactly B elements, we know that during the redistribution phase, type-4 processors only sends data to type-3 processors (which only receive from type-4 processors). In particular, **type-1 and type-2 processors do not send or receive any data to or from a type-3 or type-4 processor**.

Let assume that there exists some type-1 processor $P_i^{(1)}$ which hosts some data in $\mathcal{D}_{tar}$ which it didn't host in $\mathcal{D}_{ini}$. Since the only neighbors of processor $P_i^{(1)}$ are type-4 processors (which cannot send any data), this new data constitutes a new maximal connected component for $P_i^{(1)}$. According to Equation 3, $C_{P_i^{(1)}} \leq 4$, which means that processor $P_i^{(1)}$ has to get rid of at least two of its initial maximal connected components. Using the same reasoning than above, we claim that, to perform this, $P_i^{(1)}$ would have to send or receive strictly more than B elements, which is impossible. Thus, each type-1 processor do not keep any data it received during the redistribution phase.

Gathering all the results shown above, we can state that, at the end of the redistribution phase, the a_i elements of type-1 processors can only be hosted by type-2 processors If a processor $P_i^{(1)}$ splits its a_i consecutive elements and send them to two different type-2 processors, this would create an extra maximal connected component on the type-2 processors. Since each of the m type-2 processors hosts one maximal connected component initially and has to host less than 4 maximal connected components in $\mathcal{D}_{tar}$, they can only receive 3 maximal connected components each, meaning that **each type-1 processor has to send its $\mathbf{a_i}$ elements to the same type-2 processor**.

Let A_k be the set of the size of the maximal connected components received by $P_k^{(2)}$ during the redistribution phase. The A_k sets represent a partition of the a_i's and the cardinality of each set A_k is exactly 3. Finally, $\forall k, \sum_{a_i \in A_k} a_i = R_k^{(2)} = B$, which means that the A_ks are a solution of $Inst_1$.

Suppose now that $Inst_1$ has a solution. Let A_k be the 3-Partition of the integers a_i and consider the distribution $\mathcal{D}_{sol}$ described in Figure 3. To perform the redistribution from $\mathcal{D}_{ini}$ to $\mathcal{D}_{sol}$, each processor has to send and/or receive less than B elements of data, which means that $RedistSteps(\mathcal{D}_{ini} \to \mathcal{D}_{sol}) = B$. In addition, in $\mathcal{D}_{sol}$ each processor hosts $8B$ elements divided in 4 maximal connected components. Thus, $T_{stencil}(\mathcal{D}_{sol}) = 2 \times 4 + B \times 8B^2 = 8 + 8B^3$ and $T_{\text{total}}(\mathcal{D}_{ini} \to \mathcal{D}_{sol}) = 8 + B + 8B^3$, which means that $Inst_2$ has a solution and concludes the proof. □

6 Conclusion

In this report, we have studied the problem of finding the best data redistribution, given a target data partition. We have used two cost metrics, the total volume of communication and the number of parallel redistribution steps. We have provided optimal algorithms for both metrics, and shown through simulations that they achieve significant gain over redistributing to an arbitrary fixed distribution. We have also proved that finding the optimal data partition that minimizes the completion time of the redistribution followed by a 1D-stencil

kernel is NP-complete. Altogether, these results lay the theoretical foundations of the data partition problem on modern computers.

Future work will be devoted to an experimental validation of the approach on a multicore cluster. Admittedly, the platform model used in this report will only be a coarse approximation of actual parallel performance, because state-of-the-art runtimes use intensive prefetching and overlap communications and computations. Still, we expect that the optimal algorithms presented in this report will lead to better performance, even for compute-intensive kernels such as dense linear algebra routines.

References

[1] L. Hollermann, T. S. Hsu, D. R. Lopez, and K. Vertanen, "Scheduling problems in a practical allocation model," *J. Combinatorial Optimization*, vol. 1, no. 2, pp. 129–149, 1997.

[2] T. S. Hsu, J. C. Lee, D. R. Lopez, and W. A. Royce, "Task allocation on a network of processors," *IEEE Trans. Computers*, vol. 49, no. 12, pp. 1339–1353, 2000.

[3] M. G. Norman and P. Thanisch, "Models of machines and computation for mapping in multicomputers," *ACM Computing Surveys*, vol. 25, no. 3, pp. 103–117, 1993.

[4] B. A. Shirazi, A. R. Hurson, and K. M. Kavi, *Scheduling and load balancing in parallel and distributed systems*. IEEE CS Press, 1995.

[5] P. Chrétienne, E. G. Coffman Jr., J. K. Lenstra, and Z. Liu, Eds., *Scheduling Theory and its Applications*. John Wiley and Sons, 1995.

[6] H. El-Rewini, H. H. Ali, and T. G. Lewis, "Task scheduling in multiprocessing systems," *Computer*, vol. 28, no. 12, pp. 27–37, 1995.

[7] O. Beaumont, V. Boudet, and Y. Robert, "A realistic model and an efficient heuristic for scheduling with heterogeneous processors," in *HCW'2002, the 11th Heterogeneous Computing Workshop*. IEEE Computer Society Press, 2002.

[8] P. Bhat, C. Raghavendra, and V. Prasanna, "Efficient collective communication in distributed heterogeneous systems," *Journal of Parallel and Distributed Computing*, vol. 63, pp. 251–263, 2003.

[9] P. Rivera-Vega, R. Varadarajan, and S. Navathe, "Scheduling data redistribution in distributed databases," in *Proc. Sixth Int. Conf. Data Engineering*, 1990, pp. 166–173.

[10] Y.-A. Kim, "Data migration to minimize the total completion time," *J. Algorithms*, vol. 55, no. 1, pp. 42–57, 2005.

[11] E. G. Coffman, M. R. Garey, D. S. Johnson, and A. S. LaPaugh, "Scheduling file transfers," *SIAM J. Comput.*, vol. 14, pp. 744–780, 1985.

[12] E. Anderson, J. Hall, J. Hartline, M. Hobbes, A. Karlin, J. Saia, R. Swaminathan, and J. Wilkes, "Algorithms for data migration," *Algorithmica*, vol. 57, no. 2, pp. 349–380, 2010.

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

[14] J. J. Dongarra and D. W. Walker, "Software libraries for linear algebra computations on high performance computers," *SIAM Review*, vol. 37, no. 2, pp. 151–180, 1995.

[15] E. T. Kalns and L. M. Ni, "Processor mapping techniques towards efficient data redistribution," *IEEE Trans. Parallel Distributed Systems*, vol. 6, no. 12, pp. 1234–1247, 1995.

[16] D. W. Walker and S. W. Otto, "Redistribution of block-cyclic data distributions using MPI," *Concurrency: Practice and Experience*, vol. 8, no. 9, pp. 707–728, 1996.

[17] L. Wang, J. M. Stichnoth, and S. Chatterjee, "Runtime performance of parallel array assignment: an empirical study," in *1996 ACM/IEEE Supercomputing Conference*, 1996.

[18] F. Desprez, J. Dongarra, A. Petitet, C. Randriamaro, and Y. Robert, "Scheduling block-cyclic array redistribution," *IEEE Trans. Parallel Distributed Systems*, vol. 9, no. 2, pp. 192–205, 1998.

[19] M. Guo and Y. Pan, "Improving communication scheduling for array redistribution," *J. Parallel Distrib. Comput.*, vol. 65, no. 5, pp. 553–563, 2005.

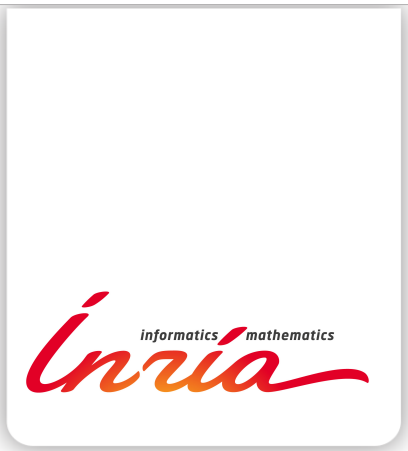
[20] L. Prylli and B. Tourancheau, "Efficient block-cyclic data redistribution," in *EuroPar'96*, ser. Lectures Notes in Computer Science, vol. 1123. Springer Verlag, 1996, pp. 155–164.

[21] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, ser. Algorithms and Combinatorics. Springer-Verlag, 2003, vol. 24.

[22] J. E. Hopcroft and R. M. Karp, "An n^5/2 algorithm for maximum matchings in bipartite graphs," *SIAM Journal on computing*, vol. 2, no. 4, pp. 225–231, 1973.

[23] G. Smith, *Numerical Solutions of Partial Differential Equations: Finite Difference Methods*. Clarendon Press, Oxford, 1985.

[24] M. R. Garey and D. S. Johnson, *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399