Determining the optimal redistribution for a given data partition

Thomas Herault*, Julien Herrmann†, Loris Marchal† and Yves Robert*
*University of Tennessee Knoxville, USA
† École Normale Supérieure de Lyon, CNRS, INRIA & University of Lyon, France

***Abstract*—The classical redistribution problem aims at optimally scheduling communications when moving from an initial data distribution to a target distribution where each processor will host a subset 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 one-to-one mapping of the subsets of data items onto the processors for which the cost of the redistribution is minimal. This paper 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) that minimize the cost of redistribution followed by a simple computation kernel.**

I. 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 be re-distributed into square blocks across 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 items 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 II, which is devoted to related work.

Modern computing platforms are equipped with powerful interconnection switches, that permit 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 items of block $B_{0,0}$ are to be redistributed.

One major goal of this paper is to assess the complexity of the problem of finding the best processor mapping for a given data partition and initial distribution 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 paper, 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 bidirectional 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 paper 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 items 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 paper 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 distribution 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 paper is organized as follows. We survey related work in Section II. We detail the model and formally state the optimization problems in Section III. We deal with the problem of finding the best redistribution for a given data partition in Section IV. Sections IV-A and IV-B provide optimal algorithms, while Section IV-C reports simulation results showing the gain over redistributing to an arbitrary compatible distribution. In Section V, 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 VI.

II. Related work

A. 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 paper, we consider large, same-sized data items, so we can safely restrict to *unit* communications that involves a single block of data; 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].

B. 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 can be progressed simultaneously [11]. Finally, in a third variant [12], processors have memory constraints that must be enforced during the redistribution process.

C. 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), run-time redistribution turns out to be a critical operation, as stated in [15], [16], [17] (among others). Communications 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.

III. Model and framework

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

A. Definitions

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

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 $0, \dots P-1$ 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 $0, \dots, P-1$ such that for all $0 \leq i \leq P-1$, $\mathcal{P}(i) = \sigma(\mathcal{D}(i))$.

B. 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, when considering synchronous communication steps as in ScaLAPACK, it is always possible to use the worst-case communication time between any processor pair as the unit time for a communication.

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

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.

C. Optimization problems

We formally introduce the optimization problems that we study in Sections IV and V.

[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 $p = \lfloor i/r \rfloor \bmod P$, $0 \leq p \leq P-1$.

1) Best redistribution compatible with a given partition: In the optimization problems of Section IV, 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 $RedistVol(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar})$ is minimized.

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

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

2) Best partition, and best compatible redistribution: In the optimization problems of Section V, 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 $RedistVol(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) \times \tau_{comm}$ or $RedistSteps(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) \times \tau_{comm}$, 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 $T_{total} = RedistVol(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) \times \tau_{comm} + T_{comp}(\mathcal{P}_{tar})$ 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 $T_{total} = RedistSteps(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) \times \tau_{comm} + T_{comp}(\mathcal{P}_{tar})$ 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 V, 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.

IV. 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.

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 $RedistVol(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar})$ 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
    $weight(i, j) \leftarrow |\{d \in \mathcal{P}_{tar}(j)$ 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 $\mathcal{D}_{tar}$
```

A. Total volume of communication

Theorem 1. *Given an initial data distribution $\mathcal{D}_{ini}$ and target data partition $\mathcal{P}_{tar}$, Algorithm 1 computes a data distribution $\mathcal{D}_{tar}$ compatible with $\mathcal{P}_{tar}$ such that $RedistVol(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar})$ 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

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

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 $O(NP + P^3)$. ■

B. 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 send and/or receive much more data than the others; all the communications involving this processor will have to be performed sequentially, creating a bottleneck.

Theorem 2. *Given an initial data distribution $\mathcal{D}_{ini}$ and target data partition $\mathcal{P}_{tar}$, Algorithm 2 computes a data distribution*

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
    $r_{i,j} \leftarrow |\{d \in \mathcal{P}_{tar}(j)$ s.t. $\mathcal{D}_{ini}(d) \neq i\}|$
    $s_{i,j} \leftarrow |\{d \in \bigcup_{k \neq j} \mathcal{P}_{tar}(k)$ 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 $|\mathcal{M}| == P$ 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 $\mathcal{M}_{save}$
```

$\mathcal{D}_{tar}$ compatible with $\mathcal{P}_{tar}$ such that $RedistSteps(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar})$ 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 $RedistSteps(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar})$ as

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

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. We denote by $\mathcal{M}_{ret}$ this matching and m_{ret} its maximum 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 no more than P^2 iterations in the while loop, and Algorithm 2 has a worst-case complexity of $O(NP + P^{\frac{9}{2}})$. ■

C. 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/.

1) Random balanced initial data distribution: First we consider a random balanced initial data distribution $\mathcal{D}_{ini}$ where each processor initially hosts 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 III, the volume of communication involved during the redistribution from $\mathcal{D}_{ini}$ to $\mathcal{D}_{can}$ is $RedistVol(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{can}) = \sum_{0 \leq j \leq P-1} |\{d \in \mathcal{P}_{tar}(j)$ 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 expected 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 IV-B, 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 most of the time, picking an arbitrary data distribution $\mathcal{D}_{can}$ will lead to poor performance. 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

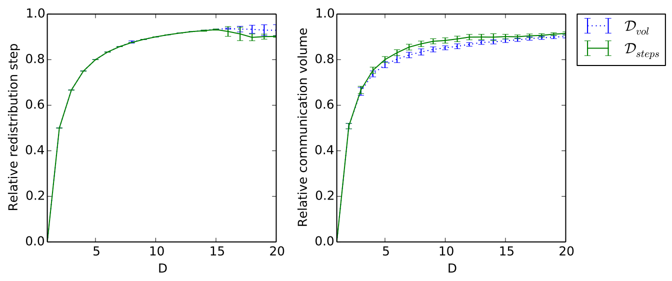

Figure 1: Performance of Algorithm 1 and 2 compared to the canonical distribution for a random initial distribution.

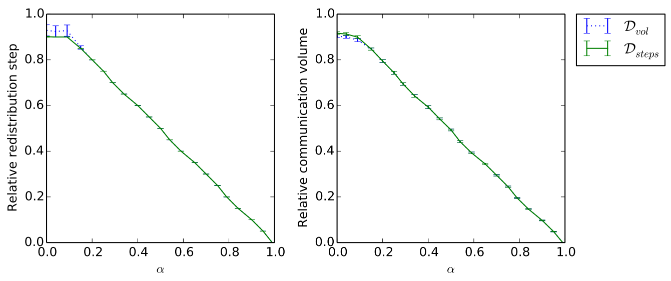

Figure 2: Performance of Algorithm 1 and 2 compared to the canonical distribution for a skewed initial distribution.

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.

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 $\mathcal{D}_{ini}^{\alpha}$ 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 $\mathcal{D}_{ini}^{0}$ 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 from $\mathcal{D}_{ini}^{\alpha}$ 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 $\mathcal{D}_{can}$: $E(RedistVol(\mathcal{D}_{ini}^{\alpha} \rightarrow \mathcal{D}_{can})) = D(P-1) + \lfloor \alpha D \rfloor$. The number of steps required to schedule the redistribution phase from $\mathcal{D}_{ini}^{\alpha}$ to $\mathcal{D}_{can}$ is equal to D with probability $1 - \left(1 - \left(\frac{P-1}{P}\right)^{D - \lfloor \alpha D \rfloor}\right)^{P}$.

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, $\mathcal{D}_{ini}^{\alpha}$ 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. Altogether, we observe significant gains over standard redistribution for a wide range of values of α.

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

A. 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. 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 III-C), 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 III-C):

- **Total volume:** For problem VolPart&Redistrib, $T_{comp}(\mathcal{P}_{tar}) = K \times T_{vol}^{iter}(\mathcal{P}_{tar})$, where
$$T_{vol}^{iter}(\mathcal{P}_{tar}) = \tau_{comm} \times \left(\sum_{0 \leq i \leq P-1} \sum_{j} N_{ij} \right) + \tau_{calc} \times \max_{0 \leq i \leq P-1} \ell_i$$
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, $T_{comp}(\mathcal{P}_{tar}) = K \times T_{steps}^{iter}(\mathcal{P}_{tar})$, where
$$T_{steps}^{iter}(\mathcal{P}_{tar}) = \tau_{comm} \times \max_{0 \leq i \leq P-1} \left(\sum_{j} N_{ij}, \sum_{j} N_{ji} \right) + \tau_{calc} \times \max_{0 \leq i \leq P-1} \ell_i$$

B. 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 is 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. *Both the* VolPart&Redistrib *and* StepPart&Redistrib *problems with the 1D-stencil kernel are strongly NP-complete.*

We only provide a sketch of proof for VolPart&Redistrib (the proof for StepPart&Redistrib is similar), as the complete proof is long, involved and technical. We refer the interested reader to the companion research report [24] for full details.

Proof sketch: VolPart&Redistrib clearly belongs to NP, and the certificate is the new distribution $\mathcal{D}_{tar}$ of data. To establish the completeness, we use a reduction from the 3-Partition problem [25], which is known to be strongly NP-complete. We consider the following instance $Inst_1$ of the 3-Partition problem: let a_i be $3m$ integers and B an integer such that $\sum a_i = mB$. To solve $Inst_1$, we need to solve the following question: is there 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 $96mB$ data items on $12m$ different processors. We set $K = 1$, $\tau_{comm} = 1$, $\tau_{calc} = B^2$, and $T_{total} = 8 + 5mB + 8B^3$ for the total cost of the 1D-stencil algorithm. The construction of $Inst_2$ is polynomial in the size of $Inst_1$. We 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 prove that $\mathcal{D}_{tar}$ is similar to data distribution $\mathcal{D}_{sol}$ illustrated on Figure 3. A first step is to prove that all the processors host $8B$ data items in $\mathcal{D}_{tar}$. In a second step, we prove that the volume of communication involved in the redistribution from $\mathcal{D}_{ini}$ to $\mathcal{D}_{tar}$ is not larger than $5mB$. Finally, we prove that each processor cannot host more that 4 maximal connected components in $\mathcal{D}_{tar}$. Gathering all these results, we know that each processor $P_k^{(2)}$ hosts a set A_k of the a_i's consecutive elements such that $|A_k| = 3$ and $\sum_{a_i \in A_k} a_i = B$, which means that the A_ks are a solution of $Inst_1$. Conversely, if we suppose that $Inst_1$ has a solution, the data distribution $\mathcal{D}_{sol}$ illustrated on Figure 3 is a solution to $Inst_2$. ■

VI. Conclusion

In this paper, 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 paper 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 paper will lead to better performance, even for compute-intensive kernels such as dense linear algebra routines.

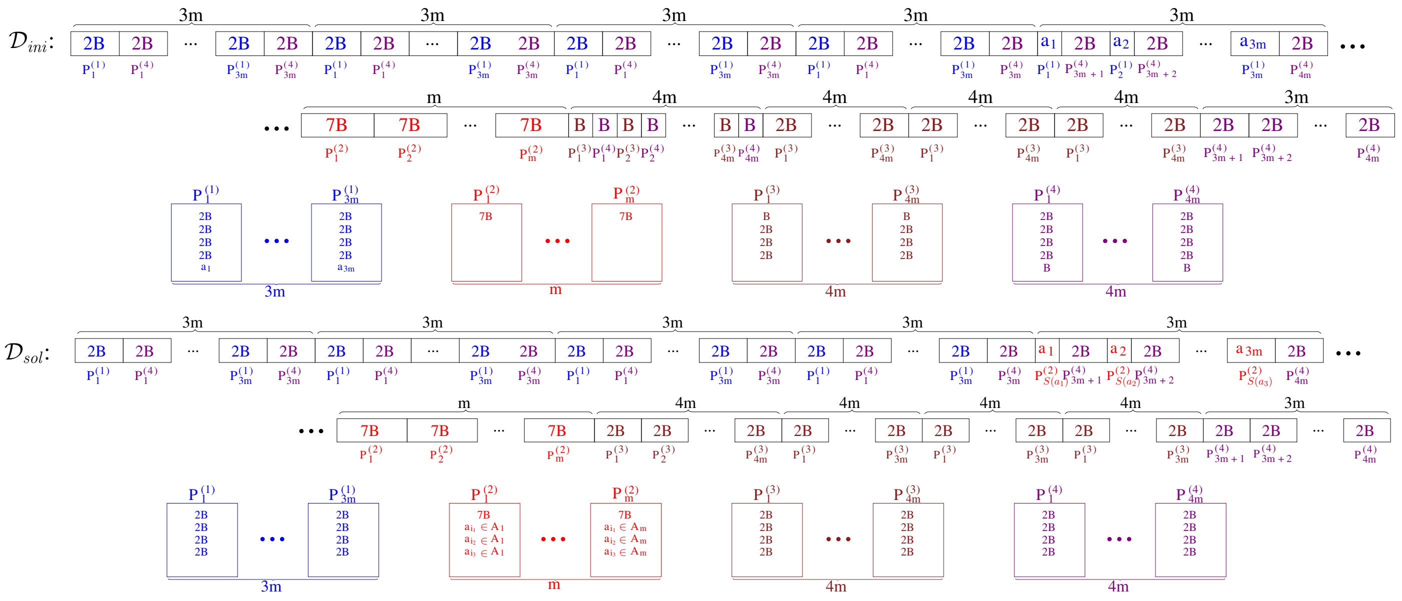

Figure 3: $\mathcal{D}_{ini}$ and $\mathcal{D}_{sol}$ in the proof of Theorem 3

Acknowledgments. This work was supported in part by the ANR *RESCUE* project. Y. Robert is with the Institut Universitaire de France.

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 and al., *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, 2005.

[20] L. Prylli and B. Tourancheau, "Fast runtime block cyclic data redistribution on multiprocessors," *J. Parallel Dist. Computing*, vol. 45, no. 1, pp. 63–72, 1997.

[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 nˆ5/2 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] T. Hérault, J. Herrmann, L. Marchal, and Y. Robert, "Determining the optimal redistribution," INRIA, Research Report 8499, 2014.

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