Parallel Computing 52 (2016) 22–41

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Assessing the cost of redistribution followed by a computational kernel: Complexity and performance results

Julien Herrmann [a,*], George Bosilca [b], Thomas Hérault [b], Loris Marchal [a], Yves Robert [a,b], Jack Dongarra [b]

[a] Ecole Normale Supérieure de Lyon, CNRS & INRIA, Lyon, France
[b] University of Tennessee, Knoxville, TN 37996, USA

ARTICLE INFO

Article history:
Received 13 May 2014
Revised 24 September 2015
Accepted 30 September 2015
Available online 14 November 2015

Keywords:
Redistribution
Linear algebra
QR factorization
Stencil
Data partition
Parsec

ABSTRACT

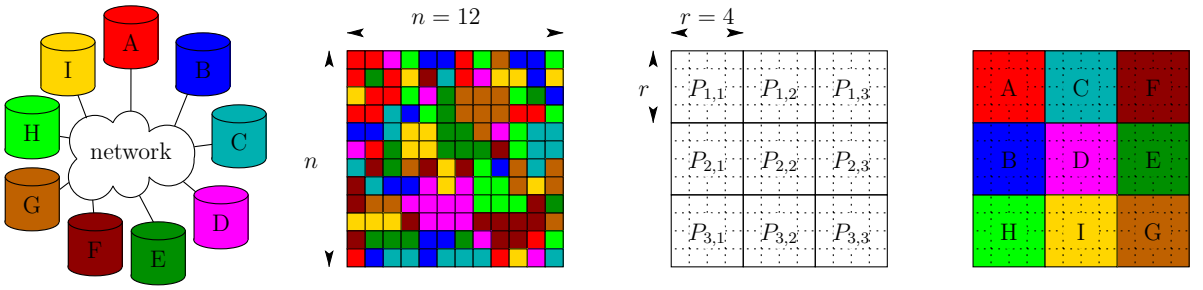
The classical redistribution problem aims at optimally scheduling communications when reshuffling from an initial data distribution to a target data distribution. This target data distribution is usually chosen to optimize some objective for the algorithmic kernel under study (good computational balance or low communication volume or cost), and therefore to provide high efficiency for that kernel. However, the choice of a distribution minimizing the target objective is not unique. This leads to generalizing the redistribution problem as follows: find a re-mapping of data items onto processors such that the data redistribution cost is minimal, and the operation remains as efficient. This paper studies the complexity of this generalized problem. We compute optimal solutions and evaluate, through simulations, their gain over classical redistribution. 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 computational kernel. Finally, experimental validation of the new redistribution algorithms are conducted on a multicore cluster, for both a 1D-stencil kernel and a more compute-intensive dense linear algebra routine.

1. Introduction

In parallel computing systems, data locality has a strong impact on application performance. To achieve 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, redistribute them according to a different target layout. Consider for instance a dense square matrix $A = (a_{ij})_{0 \le i,j < n}$ of size n, whose initial distribution is random, and that must be redistributed 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 elements of A on each processor, as illustrated in Fig. 1. More precisely, all the elements of block $A_{i,j} = (a_{k,\ell})$, where $ri \le k < (r+1)i$ and $rj \le \ell < (r+1)j$, must be sent to processor

* Corresponding author. Tel.: +33 437287644.
E-mail addresses: julien.herrmann@ens-lyon.fr (J. Herrmann), bosilca@icl.utk.edu (G. Bosilca), herault@icl.utk.edu (T. Hérault), loris.marchal@ens-lyon.fr (L. Marchal), yves.robert@ens-lyon.fr (Y. Robert), dongarra@icl.utk.edu (J. Dongarra).

http://dx.doi.org/10.1016/j.parco.2015.09.005
0167-8191/



(a) processors holding the data (b) intial data distribution (c) target data partition (d) final data distribution

Fig. 1. Example of matrix redistribution with $N = 12^2$ data blocks and $P = 3^2$ processors. Each color in the data distributions corresponds to a processor, e.g., all red data items reside on processor A. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$P_{i,j}$. 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 interconnection switches and routing mechanisms mapping 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 virtual, i.e., an overlay topology mapped into the physical topology, forcing the interconnection switch to emulate a 2D-grid. Notwithstanding, the layout of the processors in the grid remains 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 us to revisit the redistribution problem, adding the flexibility to select the *best* assignment of data on 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 that initially reside on the same processor, and to look for a coloring of the virtual grid that will minimize the redistribution cost. For instance, in Fig. 1, most data items of the block allocated to the virtual processor $P_{1,1}$ are initially colored red (they reside on the red processor A), so we decided to map $P_{1,1}$ on processor A to avoid moving these items.

One major goal of this paper is to assess the complexity of the problem of finding the best processor mapping for a given initial data distribution and a target data partition. This amounts to determining 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 minimizing a predefined cost-function. 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 amount of data sent from one processor to another. Minimizing this volume makes it less likely to disrupt the other applications running on the platform, and is expected to decrease network contention, hence redistribution time. Conceptually, this is equivalent to assuming that the network is a bus, globally shared by all computing 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 paper is the design of an algorithm solving this optimization problem for either criterion. We also provide various experiments to quantify the gain that results from choosing the optimal mapping rather than a *canonical* mapping where processors are labeled arbitrarily, and independently of the initial data distribution.

As mentioned earlier, a redistribution is usually motivated by the need to efficiently execute in parallel a subsequent computational kernel. In most cases, there may well be several data partitions that are suitable for the efficient execution of this kernel. The optimal partition also depends upon the initial data distribution. 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 computes* 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 significantly 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,