More on Scheduling Block-Cyclic Array Redistribution*

Frédéric Desprez[1], Stéphane Domas[1], Jack Dongarra[2,3],
Antoine Petitet[2], Cyril Randriamaro[1] and Yves Robert[1]

[1] LIP, Ecole Normale Supérieure de Lyon, 69364 Lyon Cedex 07, France
[2] Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301, USA
[3] Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
e-mail: [desprez, sdomas, crandria, yrobert]@ens-lyon.fr
e-mail: [dongarra,petitet]@cs.utk.edu

February 1998

Abstract

This article is devoted to the run-time redistribution of one-dimensional arrays that are distributed in a block-cyclic fashion over a processor grid. In a previous paper [2], we have reported how to derive optimal schedules made up of successive communication-steps. In this paper we assume that successive steps may overlap. We show how to obtain an optimal scheduling for the most general case, namely, moving from a `CYCLIC(r)` distribution on a P-processor grid to a `CYCLIC(s)` distribution on a Q-processor grid, for *arbitrary* values of the redistribution parameters P, Q, r, and s. We use graph-theoretic algorithms, and modular algebra techniques to derive these optimal schedulings.

Key-words: distributed arrays, block-cyclic distribution, redistribution, asynchronous communications, scheduling, HPF.

Corresponding author: Frédéric Desprez
LIP, Ecole Normale Supérieure de Lyon, 69364 Lyon Cedex 07, France
Phone: + 33 4 72 72 80 37, Fax: + 33 4 72 72 80 80
E-mail: Frederic.Desprez@ens-lyon.fr

*This work was supported in part by the National Science Foundation Grant No. ASC-9005933; by the Defense Advanced Research Projects Agency under contract DAAH04-95-1-0077, administered by the Army Research Office; by the Department of Energy Office of Computational and Technology Research, Mathematical, Information, and Computational Sciences Division under Contract DE-AC05-84OR21400; by the National Science Foundation Science and Technology Center Cooperative Agreement No. CCR-8809615; by the CNRS–ENS Lyon–INRIA project *ReMaP*; and by the Eureka Project *EuroTOPS*.

1 Introduction

Run-time redistribution of arrays that are distributed in a block-cyclic fashion over a multidimensional processor grid is a difficult problem that has recently received considerable attention. Rather than providing a detailed motivation and a survey of the existing literature, we refer the reader to [3, 4, 2].

The redistribution problem is to redistribute an array X with a `CYCLIC(r)` distribution on a P-processor grid to a same-size array Y with a `CYCLIC(s)` distribution on a Q-processor grid. This amounts to perform the HPF assignment $Y = X$.

Without loss of generality, we focus on one-dimensional redistribution problems in this article. Although we usually deal with multidimensional arrays in high-performance computing, the problem reduces to the "tensor product" of the individual dimensions. This is because HPF does not allow more than one loop variable in an `ALIGN` directive. Therefore, multidimensional assignments and redistributions are treated as several independent one-dimensional problem instances.

In a previous paper [2], we have reported how to derive optimal schedules made up of successive communication-steps, assuming a synchronization at the end of each step. The goal was to minimize either the number of steps, or the total cost of the redistribution computed as follows: for each step, the cost is (proportional to) the length of the longest message; the total cost is the sum of the cost of all steps.

In this paper we assume that successive steps may overlap. We show how to obtain an optimal scheduling using this new hypothesis that models more adequately state-of-the-art distributed-memory machines. Now the meaning of a "communication step" is simply that at any time-step, each processor sends/receives at most one message, thereby optimizing the amount of buffering and minimizing contention on communication ports. The construction of our optimal schedules relies on graph-theoretic algorithms, and modular algebra techniques.

Consider an array $X[0...M-1]$ of size M that is distributed according to a block cyclic distribution `CYCLIC(r)` onto a linear grid of P processors (numbered from $p = 0$ to $p = P - 1$). Our goal is to redistribute X using a `CYCLIC(s)` distribution on Q processors (numbered from $q = 0$ to $q = Q - 1$). For simplicity, we assume that the size M of X is a multiple of $L = lcm(Pr, Qs)$, the least common multiple of Pr and Qs: this is because the redistribution pattern repeats after each slice of L elements. Therefore, assuming an even number of slices in X will enable us (without loss of generality) to avoid discussing side effects. Let $m = M \div L$ be the number of slices.

Definition 1 *We let $(P, r) \to (Q, s)$ denote the redistribution problem from an original grid of P processors with distribution* `CYCLIC(r)` *to a target grid of Q processors with distribution* `CYCLIC(s)`*, and assuming a single-slice vector to be redistributed. Any indicated message length must be understood as a unit length (to be multiplied by the number of slices for actual vectors). Finally, we assume that r and s are relatively prime, that is,* $\gcd(r, s) = 1$ *(this handy simplification causes no loss of generality [2]).*

2 Motivating example

Example 1

Consider an example with $P = Q = 6$ processors, $r = 2$, and $s = 3$. Note that the new grid of Q processors can be identical to, or disjoint of, the original grid of P processors. All communications are summarized in Table 1, which we refer to as a *communication grid*. Note that we view the source and target processor grids as disjoint in Table 1 (even if it may not actually be the case).

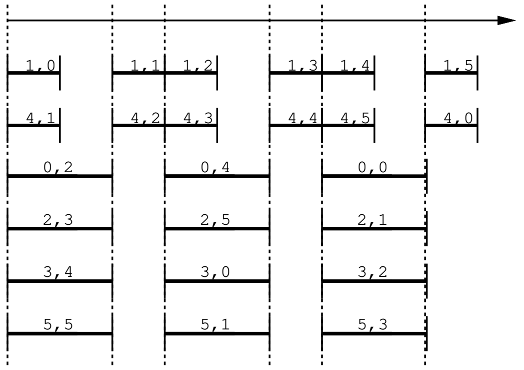

Figure 1: Communication Scheduling for Example 1 (without overlap of time steps).

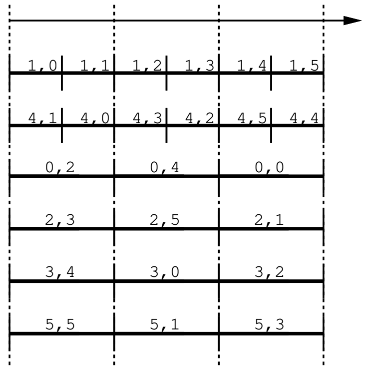

Figure 2: Communication Scheduling for Example 1 (with overlap of time steps).

We see that each source processor $p \in \{0, 2, 3, 5\} \subset \mathcal{P} = \{0, 1, \ldots, P-1\}$ sends 3 messages and that each processor $q \in \mathcal{Q} = \{0, 1, \ldots, Q-1\}$ receives 4 messages. But each source processor $p \in \{1, 4\} \subset \mathcal{P}$ sends 6 messages. Hence there is no need to use a full all-to-all communication scheme that would require 6 steps, with a total of 6 messages to be sent per processor (or more precisely, 5 messages and a local copy). Rather, we should try to schedule the communication more efficiently.

S/R	0	1	2	3	4	5	N.M.
0	2	-	2	-	2	-	3
1	1	1	1	1	1	1	6
2	-	2	-	2	-	2	3
3	2	-	2	-	2	-	3
4	1	1	1	1	1	1	6
5	-	2	-	2	-	2	3
N.M.	4	4	4	4	4	4	

Table 1: Communication grid for $P = Q = 6$, $r = 2$, and $s = 3$. Message lengths are indicated for a vector X of size $L = 36$.

Figure 1 gives the optimal communication schedule for Example 1 when communication steps do not overlap. We obtain 9 time-steps which is optimal with the non-overlapping assumption but can be further reduced if we suppress this assumption. We can see in Table 1 that each processor completes its own communications in 6 time steps. If we allow communication steps to be overlapped, we can pack the smaller communications to fill the "holes" in the schedule. This is what we do in Figure 2: we obtain an optimal schedule with 6 time steps. The communication links of the processors are always used and we do not have to split any message. It can be checked in Figure 2 that each processor sends/receives at most one message at any time-step.

3 The overlapped redistribution problem

According to the previous discussion, we concentrate on "overlapped schedules". The rules of the game are the following:

- At any time-step, each sending processor is sending at most one message.
- At any time-step, each receiving processor is receiving at most one message.

This model is oriented to one-port communication machines, but is well-suited to any architectural platform because it minimizes the amount of buffering that has to be managed by the hardware or operating system.

To compute the cost of a redistribution, we simply take the delay between the initiation of the first message and the termination of the last one. The cost of each message is simply modeled by its length. For Example 1, the total cost is 6.

As for startups, we do not take them into account in the cost (because of the asynchronism, there is no way of determining the cost of a schedule including start-ups). This is not a limitation for long messages or for new-generation machines which exhibit a reduced communication start-up, but it may induce a problem for redistributing short messages on machines with a high communication start-up. Hence, another objective in the redistribution problem is to minimize the number of start-ups, which amounts to minimizing the number of messages that are sent/received. Therefore, a secondary goal when designing a good schedule is to "decompose" or "split" as few messages as possible. Of course this is an algorithmic issue, not an implementation issue (the system may well split all messages into packets).

The obvious bound is the following:

Lemma 1 *For a redistribution problem $(P, r) \to (Q, s)$, the length of any schedule is at least*

$$T_{opt} = \max\{\frac{L}{P}, \frac{L}{Q}\}.$$

Surprisingly, this bound cannot be met in all instances of the problem, unless we split some messages into smaller ones (note that the bound was met in Example 1). We use two main approaches to solve the redistribution problem: one uses graph-theoretic algorithms, and the other relies on modular algebra techniques.

4 Graph-theoretic algorithms

4.1 Complexity

Our first result is the following:

Theorem 1 *For a redistribution problem $(P, r) \to (Q, s)$, assume that we split each message of length x into x messages of unit length. Then the bound T_{opt} can be reached.*

Proof We view the communication grid as a bipartite multigraph and use matching algorithms, see [1]. ∎

The main drawback of the previous approach is that all messages are split into unit-length messages, therefore leading to a high number of start-ups. A natural question is the following: is it possible to design a schedule whose execution time is T_{opt} and for which no message is split into pieces? The answer is negative, as shown in the next section.

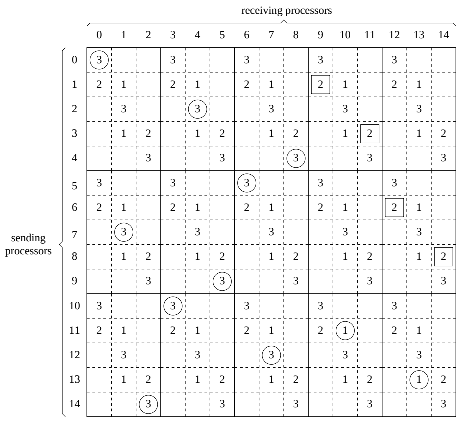

Figure 3: Communication grid for $P = Q = 15$, $r = 3$ and $s = 5$.

4.2 A counter-example

In [1], we use the example with $P = Q = 15$, $r = 3$, $s = 5$ and $L = 225$ (see Figure 3) to prove that any schedule whose execution time is $T_{opt} = 15$ will necessarily split at least one message into several smaller ones.

4.3 An efficient heuristic

We have implemented a heuristic to design a schedule that runs in T_{opt} time-steps and split as few messages as possible (if any). We use a greedy strategy to fill up the schedule table. The program has the ability to exchange some communications within the schedule in order to find a place to possibly remaining communications. As these exchanges are combinatorial, we have set a parameter that fix the maximum number of exchanges. However, the heuristic is guaranteed to run within T_{opt} time-steps if we allow for a high number of exchanges. However, it converges very fast in practice. In Table 2, we give some cases in which our algorithm does not produce an optimal schedule. The T_{opt} column gives the optimal schedule time. The T_{algo} gives the schedule time given by our algorithm without any exchange of remaining packets. At last, the T_{algo}^{ech} gives the schedule time with at least $min(P, Q)$ remaining packets that are exchanged. It is important to point out that most cases are dealt without any exchange, as in the last example in Table 2.

5 Modular algebra techniques

Using modular algebra techniques as in [2], the following result is proved in the extended version of the paper [1]:

Proposition 1

- *if* $\gcd(r, Q) = \gcd(s, P) = 1$, *we are always able to derive a schedule whose execution time is* T_{opt} *and which splits no message at all*
- *otherwise, let* $\gcd(s, P) = s'$ *and* $\gcd(r, Q) = r'$. *We are able to derive a schedule whose execution time is* T_{opt} *and which splits no message at all under the condition*

$$\left(r' < s' \textit{ and } \frac{Q}{r'} < \frac{P}{s'}\right) \quad or \quad \left(r' > s' \textit{ and } \frac{Q}{r'} > \frac{P}{s'}\right)$$

Parameters					T_{opt}	T_{algo}	T_{algo}^{ech}
P	r	Q	s	L			
15	4	12	3	180	15	18	15
15	4	16	3	240	16	23	17
15	2	14	3	210	15	19	16
15	2	16	3	240	16	22	17
16	9	18	5	720	45	49	46
15	9	18	5	270	18	20	18
15	16	18	32	2800	192	208	192
15	16	9	32	1440	160	174	160
14	17	19	33	149226	10659	10659	-

Table 2: Schedule time for different parameters P, Q, r and s.

Note that Proposition 1 covers a wide range of cases. And because the proof is constructive, we can easily implement a schedule based upon modular techniques.

6 Conclusion

In this article, we have extended our previous work devoted to general redistribution problem, that is, moving from a `CYCLIC(r)` distribution on a P-processor grid to a `CYCLIC(s)` distribution on a Q-processor grid.

In our previous paper [2], we have constructed a schedule that is optimal both in terms of number of steps and the total cost, but with a synchronization between each communication step. In this article, we have presented several results to schedule the messages with the only rule that each processor can send/receive at most one message per time-step. An implementation of our algorithm is currently under development for a future release of ScaLAPACK.

References

[1] Frédéric Desprez, Stéphane Domas, Jack Dongarra, Antoine Petitet, Cyril Randriamaro, and Yves Robert. More on scheduling block-cyclic array redistribution. Technical report, LIP, ENS Lyon, France, February 1998. Available on the WEB at http://www.ens-lyon.fr/~yrobert.

[2] Frédéric Desprez, Jack Dongarra, Antoine Petitet, Cyril Randriamaro, and Yves Robert. Scheduling block-cyclic array redistribution. *IEEE Trans. Parallel Distributed Systems*. To appear. Available as Technical Report CS-97-349, The University of Tennessee - Knoxville. **http://www.netlib.org/lapack/lawns/** (LAWN #120).

[3] David W. Walker and Steve W. Otto. Redistribution of block-cyclic data distributions using MPI. *Concurrency: Practice and Experience*, 8(9):707–728, 1996.

[4] Lei Wang, James M. Stichnoth, and Siddhartha Chatterjee. Runtime performance of parallel array assignment: an empirical study. In *1996 ACM/IEEE Supercomputing Conference*. `http://www.supercomp.org/sc96/proceedings`, 1996.