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
[desprez, sdomas, crandria, yrobert]@ens-lyon.fr,
[2] Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301, USA
[dongarra,petitet]@cs.utk.edu
[3] Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA

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, 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.

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. Solving this redistribution problem requires first to generate the messages to be exchanged and second to schedule these messages so that communication overhead is minimized. Rather than providing a detailed motivation and a survey of the existing literature, we refer the reader to [4, 5, 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

*This work was supported in part by the CNRS–ENS Lyon–INRIA project *ReMaP*; and by the Eureka Project *EuroTOPS*.

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.

Notations

The main variables used in the next sections are listed in Table 1. The abbreviations "N.M." and "S/R" are used in a few communication tables. They respectively mean "Number of Messages" and "Sender/Receiver".

Table 1. Main notations in the paper.

variable	definition
P	The number of processors in the source grid
Q	The number of processors in the target grid
r	The block factor of the source distribution
s	The block factor of the target distribution
X	The array to be redistributed
M	The (global) size of X
L	The least common multiple of Pr and Qs
m	The number of slices of L data elements in array X

Consider an array $X[0...M-1]$ of size M that is distributed according to the 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 into an array Y distributed according to the block-cyclic distribution `CYCLIC(s)` onto 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* $(P, r) \rightarrow (Q, s)$ *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 of length* $L = lcm(Pr, Qs)$ *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 2, which we refer to as a *communication grid*. Note that we view the source and target processor grids as disjoint in Table 2 (even if it may not actually be the case).

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. Rather, we should try to schedule the communication more efficiently.

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

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	

Figure 1 gives the optimal communication schedule for Example 1 when communication steps do not overlap. The horizontal axis represents time, and any pair "p, q" means that processor p in $\mathcal{P}$ sends a message to processor q in $\mathcal{Q}$. For example, during the third time-step, processor 1 in $\mathcal{P}$ sends a message to

processor 1 in $\mathcal{Q}$ and processor 4 in $\mathcal{P}$ sends a message to processor 2 in $\mathcal{Q}$. We obtain there 9 time-steps which is optimal with the non-overlapping assumption but can be further reduced if we suppress this assumption. Table 2 shows that each processor completes its own communications in 6 time steps. If we allow the communication steps to overlap, 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.

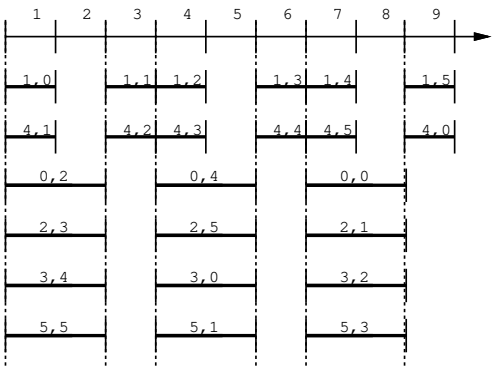

Fig. 1. Communication Scheduling for Example 1 (without overlap of time steps).

3 The overlapped redistribution problem

3.1 Communication Model

According to the previous discussion, we concentrate on "overlapped schedules". The rules 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 shown in Figure 2.

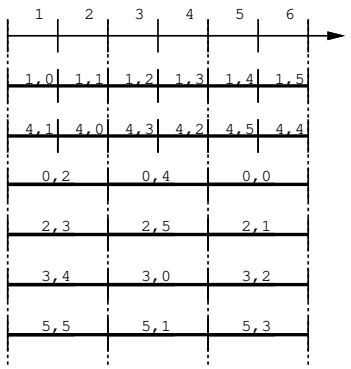

Fig. 2. Communication Scheduling for Example 1 (with overlap of time steps).

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* $(P, r) \rightarrow (Q, s)$*, 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 achieved.*

Proof. We give a constructive proof that this bound is tight. To do so, we borrow some material from graph theory. We view the communication grid as a multigraph $G = (V, E)$, where

- $V = \mathcal{P} \cup \mathcal{Q}$, where $\mathcal{P} = \{0, 1, \ldots, p-1\}$ is the set of sending processors, and $\mathcal{Q} = \{0, 1, \ldots, q-1\}$ is the set of receiving processors; and
- if the entry (p, q) in the communication grid is a nonzero integer x, we add x identical edges from p to q.

G is a bipartite multigraph (all edges link a vertex in $\mathcal{P}$ to a vertex in $\mathcal{Q}$). The degree of G, defined as the maximum degree of its vertices, is $d_G = T_{opt}$. According to König's edge coloring theorem, the edge coloring number of a bipartite graph is equal to its degree (see [3, vol. 2, p.1666] or Berge [1, p. 238]). This means that the edges of a bipartite graph can be partitioned in d_G disjoint edge matchings. A constructive proof is as follows: repeatedly extract from E a maximum matching that saturates all maximum degree nodes. At each iteration, the existence of such a maximum matching is guaranteed (see Berge [1, p. 130]). To define the schedule, we simply let the matchings at each iteration represent the communication steps.

Remark 1. The proof of Theorem 1 gives a bound for the complexity of determining the optimal schedule. The best known algorithm for weighted, bipartite matching has cost $O(|V|^3)$ (Hungarian method, [3, vol. 1, p.206]). Since there are at most $T_{opt} \leq rs \max\{P, Q\}$ iterations to construct the schedule, we have a procedure in $O(rs(|P| + |Q|)^4)$ to construct an optimal schedule.

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.

4.2 A counter-example

As a proof to the impossibility to always reach the optimal schedule without cutting messages, we take the following example: $P = Q = 15$, $r = 3$, $s = 5$ and $L = 225$. Generating the communication grid, we see that 15 unit-length data are sent by all the processors in $\mathcal{P}$, and received by all the processors in $\mathcal{Q}$. This implies that $T_{opt} = 15$ and that $15 = L/T_{opt}$ communications must be simultaneously in progress at any time-step $t \in [0, 15[$. Thus each processor must be sending a message at any time-step. We proceed by contradiction: assume for a while that there exists an optimal schedule in T_{opt} without cutting any message.

First, we reduce the problem to $t \in [0, 3[$. Indeed, nine processors out of fifteen must send five messages of size three. Thus, we want to determine a combination

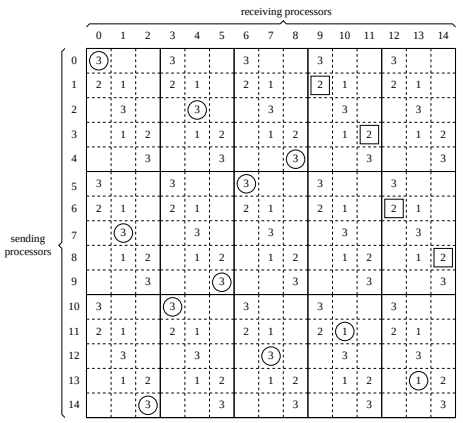

Fig. 3. Communication grid for $P = Q = 15$, $r = 3$ and $s = 5$.

of communications that strictly fit into 3 time-steps. Hence, we must choose 9 communications of size 3. These are circled in Figure 3. The 6 remaining sending processors must be kept busy. We have the choice to send 6 messages of size 2 and 6 messages of size 1, or 18 messages of size 1.

- In the first case, it implies that 6 processors must be sending a message of size 2 at the same time. If we look at the communication grid in Figure 3, we can only select 4 processors.
- In the second case, it implies that 6 processors must be sending a message of size 1 at the same time. If we look at the communication grid in Figure 3, we can only select 4 processors.

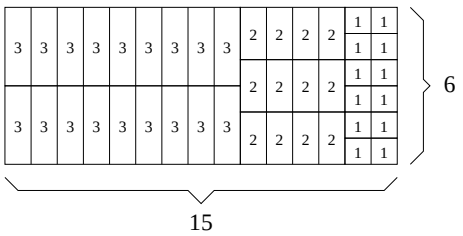

Fig. 4. A solution for the first 6 time-steps.

Thus, it is impossible to find a solution for $t \in [0, 3[$. But, for $t \in [0, 6[$, we can find a combination of communications that strictly fits in 6 time-steps. The

solution is shown in Figure 4. Unfortunately, $15 \bmod 6 = 3$ and we have proved that we cannot find a solution in 3 time-steps. Finally, there are no solutions for $t \in [0, 9[$ since $6 = lcm(3, 2, 1)$. Thus, it is impossible to find a schedule in 15 time-steps.

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 algorithm has the ability to exchange some communications within the schedule in order to place possibly remaining communications. As the number of these exchanges is combinatorial, we have set a parameter that fix the maximum number of exchanges. The heuristic is guaranteed if we allow for a high number of exchanges. However, very fast convergence has been observed in practice. In Table 3, 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 3.

Table 3. Schedule time for different parameters P, Q, r and s.

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	-

5 Modular algebra techniques

Using modular algebra techniques as in [2], we have the following result:

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*

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

Proof. – Since that $\gcd(r,s) = 1$, we introduce integers u and v such that

$$ru - sv = 1.$$

In our previous paper [2], we defined classes by

$$class(k) = \{\begin{pmatrix} p \\ q \end{pmatrix} = \lambda \begin{pmatrix} s \\ r \end{pmatrix} + k \begin{pmatrix} u \\ v \end{pmatrix} \bmod \begin{pmatrix} P \\ Q \end{pmatrix};\ 0 \leq \lambda < \frac{PQ}{g}\},$$

for $0 \leq k < g$.
In [2] we showed that a schedule based upon classes is optimal when $\gcd(r,Q) = \gcd(s,P) = 1$.

– Otherwise, assume that $g > r + s - 1$. Consider $r < s$, $s' = \gcd(s,P)$, $r' = \gcd(r,Q)$, $P = s'P'$, $Q = r'Q'$, and $g_0 = \gcd(P',Q')$. Then $g = \gcd(Pr,Qs) = r's'g_0$. The communication table is divided into subtables of size $s' \times r'$.

Now, we propose to find an optimal scheduling inside such subtables.

Lemma 2. *We can assume that* $g_0 = 1$.

We define a new class by $class(k)$ with the elements of a subtable:

$$class'(k) = \{(p,q) \in class(k); 0 \leq p < s'; 0 \leq q < r'\}$$

Lemma 3. *With such conditions, for* $1 - r \leq k < s - 1$, *each communication class* $class'(k)$ *contains one element only:* $class'(k) = (p_k,\ q_k)$.

Proof. By definition, if $(p,q) \in class'(k)$ then $(p,q) \in class(k)$, so

$$\begin{pmatrix} p \\ q \end{pmatrix} = \lambda \begin{pmatrix} s \\ r \end{pmatrix} + k \begin{pmatrix} u \\ v \end{pmatrix} \bmod \begin{pmatrix} P \\ Q \end{pmatrix};\ 0 \leq \lambda < \frac{PQ}{g};$$

While $s' = \gcd(s,P)$ and $r' = \gcd(r,Q)$, then

$$\begin{pmatrix} p \\ q \end{pmatrix} = \lambda \begin{pmatrix} s' \\ r' \end{pmatrix} + k \begin{pmatrix} u \\ v \end{pmatrix} \bmod \begin{pmatrix} P \\ Q \end{pmatrix}; 0 \leq \lambda$$

As $0 \leq p < s'$ and $0 \leq q < r'$, then $\lambda = 0$. In addition, by definition of s' and r', $\gcd(s',P) = s'$ and $\gcd(r',Q) = r'$, hence

$$\begin{pmatrix} p \\ q \end{pmatrix} = k \begin{pmatrix} u \\ v \end{pmatrix} \begin{matrix} \bmod s' \\ \bmod r' \end{matrix}$$

Now consider $(p', q') \in class'(k)$, so

$$\begin{pmatrix} p' \\ q' \end{pmatrix} = k' \begin{pmatrix} u \\ v \end{pmatrix} \begin{matrix} \mathrm{mod} s' \\ \mathrm{mod} r' \end{matrix}$$

it implies

$$\begin{matrix} ku = k'u \bmod s' \\ kv = k'v \bmod r' \end{matrix} \Rightarrow \begin{matrix} (k-k')u = \alpha s' \\ (k-k')v = \beta r' \end{matrix} \Rightarrow \begin{matrix} s' | (k-k') \ because \ \gcd(s', u) = 1 \\ r' | (k-k') \ because \ \gcd(r', v) = 1 \end{matrix}$$

then $r's' | (k-k')$. $1 - r \leq k, k' < s - 1$, then $0 < |k - k'| < g$. $g = r's'$ so $k = k'$.

Lemma 4. *For any $k \in [1-r, s-1]$, there is no contention (i.e. same sender or same receiver) between class'(k) and class'$(k+x)$, where $x \in [1, r'-1]$*

Proof. Consider a contention between senders of *class'*(k) and *class'*$(k+x)$, then, if $(p, q) = class'(k)$ then $class'(k+x) = (p', q') = (p, q')$. Hence $p = ku \bmod s' = (k+x)u \bmod s'$ so $xu \bmod s' = 0$. In the same way, there is a contention between receivers of *class'*(k) and *class'*$(k+x)$ iff $xv \bmod r' = 0$. Assuming that $r' < s'$, if $x > 0$ a contention implies $x \geq r'$.

Lemma 5. *Sending messages by classes from class'$(1-r)$ to class'$(s-1)$ generates no contention.*

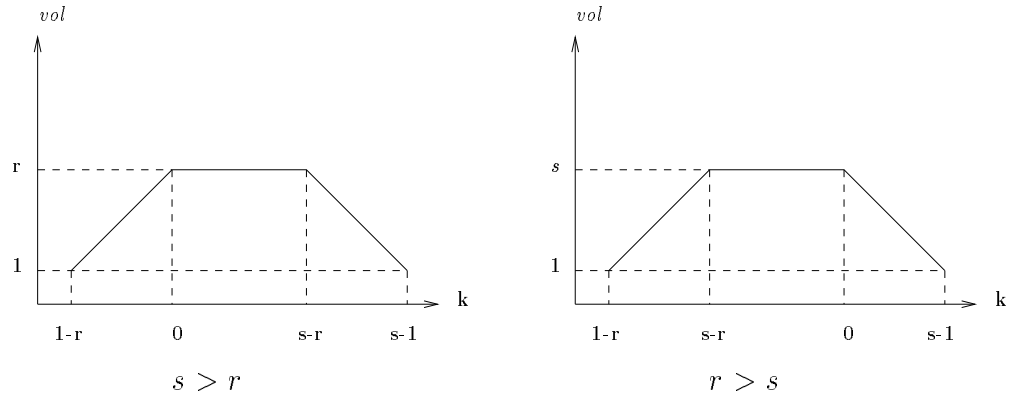

Fig. 5. The piecewise linear function *vol*.

Proof. In paper [2], it is proved that function *vol*(k) given by Figure 5 for $k \in [1-r, s-1]$ represents the size of each communication in *class*(k), for a vector of length $L = lcm(Pr, Qs)$. While *class'*$(k) \subset$ *class*(k) this result is still correct for *class'*(k).

During the first step of that schedule, the r' communications in *class'*(k) with $k \in [1-r, -r+r']$ can be performed in the same time (lemma 4) shows that there is no contention).

$vol(1-r) = 1$
$vol(-r+2) = 2$
$vol(-r+3) = 3$
$\vdots$
$vol(-r+r'-1) = r'-1$
$vol(-r+r') = r'$

The next step is composed by the r' next communications in *class'*(k) (i.e. $k \in [1-r+r', -r+2r']$) during that step due to the same lemma, for any $k \in [1-r+r', -r+2r']$, *class'*(k) generates no contention with any *class'*(k') where $k' \in [k+1, -r+2r']$ or $k' \in [k-r'+1, k-1]$. Because sizes are sorted, communications in *class'*$(k' \leq k-r')$ are already performed when the communication of *class'*(k) begins. Hence, there is no contention at all during the execution of the first two steps. In the same way, we can prove that there in no contention during the next steps up to messages of size r, and then when the message size decreases:

$vol(1-r+r') = r'+1$	(r)	$vol(s-2r') = r$	$vol(s-r') = r'$
$\vdots$	(r)	$\vdots$	$vol(s-r'-1) = r'-1$
$vol(-1) = r-1$	(r)	$vol(r-s) = r$	$\vdots$
$vol(0) = r$	(r)	$vol(r-s+1) = r-1$	$vol(s-3) = 3$
$\vdots$	(r)	$\vdots$	$vol(s-2) = 2$
$vol(-r+2r') = r$	(r)	$vol(s-r'-1) = r'$	$vol(s-1) = 1$

Lemma 6. *If an optimal schedule is found in a $s' \times r'$ grid, an optimal schedule can be found for the $P \times Q$ grid if $s' > r'$ and $\frac{P}{s'} \geq \frac{Q}{r'}$.*

Proof. The idea is to perform schedule in boxes by diagonals.

Example 2. Consider the following example where $P = 15$, $Q = 6$, $r = 3$, and $s = 5$, $r' = \gcd(3,6) = 3$, and $s' = \gcd(5,15) = 5$. As shown in Table 4, the communication table is divided into boxes of size 5×3. In that example, the communication in two diagonal boxes can be performed in the same time with no contention: there is no intersection between the set of senders (or receivers) of the boxes. While the number of sender is greater than the number of receiver, it is the maximum number of boxes that can be taken: when communications are performed in a box, all receivers are always receiving. Then, performing communication by boxes in the same diagonal is optimal. Hence, it is easy to prove that the algorithm by boxes in diagonals is optimal under the condition

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

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.

Table 4. Communication grid for $P = 15$, $Q = 6$, $r = 3$, and $s = 5$. Message lengths are indicated for a vector X of size $L = 225$.

Send/Recv.	0	1	2	3	4	5	Nbr of msg.
0	3	-	-	3	-	-	5
1	2	1	-	2	1	-	10
2	-	3	-	-	3	-	5
3	-	1	2	-	1	2	10
4	-	-	3	-	-	3	5
5	3	-	-	3	-	-	10
6	2	1	-	2	1	-	5
7	-	3	-	-	3	-	10
8	-	1	2	-	1	2	5
9	-	-	3	-	-	3	10
10	3	-	-	3	-	-	5
11	2	1	-	2	1	-	10
12	-	3	-	-	3	-	5
13	-	1	2	-	1	2	10
14	-	-	3	-	-	3	5
Nbr of msg.	6	9	6	6	9	6	

Also note that it is a sufficient but not necessary condition to derive an optimal schedule. For example, with $P = 12$, $Q = 8$, $r = 4$, and $s = 3$ (see [2]), $r' = \gcd(4, 8) = 4$, and $s' = \gcd(3, 12) = 3$. The condition is not true but there exists an optimal schedule in time $T_{opt} = 6$.

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. Claude Berge. *Graphes et hypergraphes.* Dunod, 1970. English translation by Elsevier, Amsterdam (1985).

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. R.L. Graham, M. Grötschel, and L. Lovász. *Handbook of combinatorics*. Elsevier, 1995.
4. David W. Walker and Steve W. Otto. Redistribution of block-cyclic data distributions using MPI. *Concurrency: Practice and Experience*, 8(9):707–728, 1996.
5. 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.