Scheduling Block-Cyclic Array Redistribution*

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

[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, crandria]@lip.ens-lyon.fr`
`e-mail: [dongarra, petitet, yrobert]@cs.utk.edu`

February 1997

Abstract

This article is devoted to the run-time redistribution of arrays that are distributed in a block-cyclic fashion over a multidimensional processor grid. While previous studies have concentrated on efficiently generating the communication messages to be exchanged by the processors involved in the redistribution, we focus on the *scheduling* of those messages: how to organize the message exchanges into "structured" communication steps that minimize contention. We build upon results of Walker and Otto, who solved a particular instance of the problem, and we derive 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.

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

*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*. Yves Robert is on leave from Ecole Normale Supérieure de Lyon and is partly supported by DRET/DGA under contract ERE 96-1104/A000/DRET/DS/SR. The authors acknowledge the use of the Intel Paragon XP/S 5 computer, located in the Oak Ridge National Laboratory Center for Computational Sciences, funded by the Department of Energy's Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research.

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. This interest is motivated largely 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 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 [5]. 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 [10, 21, 22] (among others).

Basically, we can decompose the redistribution problem into the following two subproblems:

Message generation The array to be redistributed should be efficiently scanned or processed in order to build up all the messages that are to be exchanged between processors.

Communication scheduling All the messages must be efficiently scheduled so as to minimize communication overhead. A given processor typically has several messages to send, to all other processors or to a subset of these. In terms of MPI collective operations [16], we must schedule something similar to an `MPI_ALLTOALL` communication, except that each processor may send messages only to a particular subset of receivers (the subset depending on the sender).

Previous work has concentrated mainly on the first subproblem, message generation. Message generation makes it possible to build a different message for each pair of processors that must communicate, thereby guaranteeing a volume-minimal communication phase (each processor sends or receives no more data than needed). However, the question of how to efficiently schedule the messages has received little attention. One exception is an interesting paper by Walker and Otto [21] on how to schedule messages in order to change the array distribution from `CYCLIC(r)` on a P-processor linear grid to `CYCLIC(Kr)` on the same grid. Our aim here is to extend Walker and Otto's work in order to solve the **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.

The general instance of the redistribution problem turns out to be much more complicated than the particular case considered by Walker and Otto. However, we provide efficient algorithms and heuristics to optimize the scheduling of the communications induced by the redistribution operation. Our main result is the following: For **any** values of the redistribution parameters P, Q, r and s, we construct an **optimal** schedule, that is, a schedule whose number of communication steps is minimal. A communication step is defined so that each processor sends/receives at most one message, thereby optimizing the amount of buffering and minimizing contention on communication ports. The construction of such an optimal schedule relies on graph-theoretic techniques such as the edge coloring number of bipartite graphs. We delay the precise (mathematical) formulation of our results until Section 4 because we need several definitions beforehand.

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.

The rest of this article is organized as follows. In Section 2 we provide some examples of redistribution operations to expose the difficulties in scheduling the communications. In Section 3 we briefly survey the literature on the redistribution problem, with particular emphasis given to the Walker and Otto paper [21]. In Section 4 we present our main results. In Section 5 we report on some MPI experiments that demonstrate the usefulness of our results. Finally, in Section 6, we state some conclusions and future work directions.

2 Motivating Examples

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, 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.

Example 1

Consider a first example with $P = Q = 16$ processors, $r = 3$, and $s = 5$. Note that the new grid of Q processors can be identical to, or disjoint of, the original grid of P processors. The actual total number of processors in use is an unknown value between 16 and 32. 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). We see that each source processor $p \in \mathcal{P} = \{0, 1, \ldots, P-1\}$ sends 7 messages and that each processor $q \in \mathcal{Q} = \{0, 1, \ldots, Q-1\}$ receives 7 messages, too. Hence there is no need to use a full all-to-all communication scheme that would require 16 steps, with a total of 16 messages to be sent per processor (or more precisely, 15 messages and a local copy). Rather, we should try to schedule the communication more efficiently. Ideally, we could think of organizing the redistribution in 7 steps, or communication phases. At each step, 16 messages would be exchanged, involving 16 disjoint pairs of processors. This would be perfect for one-port communication machines, where each processor can send and/or receive at most one message at a time.

Note that we may ask something more: we can try to organize the steps in such a way that at each step, the 8 involved pairs of processors exchange a message of the same length. This approach is of interest because the cost of a step is likely to be dictated by the length of the longest message exchanged during the step. Note that message lengths may or may not vary significantly. The numbers in Table 1 vary from 1 to 3, but they are for a single slice vector. For a vector X of length $M = 240000$, say, $m = 1000$ and message lengths vary from 1000 to 3000 (times the number of bytes needed to represent one data-type element).

A schedule that meets all these requirements, namely, 7 steps of 16 disjoint processor pairs exchanging messages of the same length, will be provided in Section 4.3.2. We report the solution schedule in Table 2. Entry in position (p, q) in this table denotes the step (numbered from a to g for clarity) at which processor p sends its message to processor q.

In Table 3, we compute the cost of each communication step as (being proportional to) the length of the longest message involved in this step. The total cost of the redistribution is then the sum of the cost of all the steps. We further elaborate on how to model communication costs in Section 4.3.1.

Table 1: Communication grid for $P = Q = 16$, $r = 3$, and $s = 5$. Message lengths are indicated for a vector X of size $L = 240$.

Communication grid for $\boldsymbol{P = Q = 16}$, $\boldsymbol{r = 3}$, $\boldsymbol{s = 5}$, and $\boldsymbol{L = 240}$

Sender/Recv.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Nbr of msg.
0	3	-	-	3	-	-	3	-	-	2	1	-	1	2	-	-	7
1	2	1	-	1	2	-	-	3	-	-	3	-	-	3	-	-	7
2	-	3	-	-	3	-	-	2	1	-	1	2	-	-	3	-	7
3	-	1	2	-	-	3	-	-	3	-	-	3	-	-	2	1	7
4	-	-	3	-	-	2	1	-	1	2	-	-	3	-	-	3	7
5	2	-	-	3	-	-	3	-	-	3	-	-	2	1	-	1	7
6	3	-	-	2	1	-	1	2	-	-	3	-	-	3	-	-	7
7	-	3	-	-	3	-	-	3	-	-	2	1	-	1	2	-	7
8	-	2	1	-	1	2	-	-	3	-	-	3	-	-	3	-	7
9	-	-	3	-	-	3	-	-	2	1	-	1	2	-	-	3	7
10	1	-	1	2	-	-	3	-	-	3	-	-	3	-	-	2	7
11	3	-	-	3	-	-	2	1	-	1	2	-	-	3	-	-	7
12	1	2	-	-	3	-	-	3	-	-	3	-	-	2	1	-	7
13	-	3	-	-	2	1	-	1	2	-	-	3	-	-	3	-	7
14	-	-	3	-	-	3	-	-	3	-	-	2	1	-	1	2	7
15	-	-	2	1	-	1	2	-	-	3	-	-	3	-	-	3	7
Nbr of msg.	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	

Example 2

The second example, with $P = Q = 16$, $r = 7$, and $s = 11$, shows the usefulness of an efficient schedule even when each processor communicates with every other processor. As illustrated in Table 4, message lengths vary with a ratio from 2 to 7, and we need to organize the all-to-all exchange steps in such a way that messages of the same length are communicated at each step. Again, we are able to achieve such a goal (see Section 4.3.2). The solution schedule is given in Table 5 (where steps are numbered from a to p), and its cost is given in Table 6. (We do check that each of the 16 steps is composed of messages of the same length.)

Example 3

Our third motivating example is with $P = Q = 15$, $r = 3$, and $s = 5$. As shown in Table 7, the communication scheme is severely unbalanced, in that processors may have a different number of messages to send and/or to receive. Our technique is able to handle such complicated situations. We provide in Section 4.4 a schedule composed of 10 steps. It is no longer possible to have messages of the same length at each step (for instance, processor $p = 0$ has messages only of length 3 to send, while processor $p = 1$ has messages only of length 1 or 2), but we do achieve a redistribution in 10 communication steps, where each processor sends/receives at most one message per step. The number of communication steps in Table 8 is clearly optimal, as processor $p = 1$ has 10 messages to send. The cost of the schedule is given in Table 9.

Table 2: Communication steps for $P = Q = 16$, $r = 3$, and $s = 5$.

Communication steps for $\boldsymbol{P = Q = 16}$, $\boldsymbol{r = 3}$, and $\boldsymbol{s = 5}$

Sender/Recv.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	b	-	-	a	-	-	c	-	-	e	g	-	f	d	-	-
1	d	f	-	g	e	-	-	c	-	-	b	-	-	a	-	-
2	-	b	-	-	c	-	-	e	g	-	f	d	-	-	a	-
3	-	g	e	-	-	a	-	-	b	-	-	c	-	-	d	f
4	-	-	c	-	-	e	g	-	f	d	-	-	b	-	-	a
5	e	-	-	c	-	-	a	-	-	b	-	-	d	f	-	g
6	c	-	-	e	g	-	f	d	-	-	a	-	-	b	-	-
7	-	c	-	-	b	-	-	a	-	-	d	f	-	g	e	-
8	-	e	g	-	f	d	-	-	c	-	-	a	-	-	b	-
9	-	-	a	-	-	b	-	-	d	f	-	g	e	-	-	c
10	g	-	f	d	-	-	b	-	-	c	-	-	a	-	-	e
11	a	-	-	b	-	-	d	f	-	g	e	-	-	c	-	-
12	f	d	-	-	a	-	-	b	-	-	c	-	-	e	g	-
13	-	a	-	-	d	f	-	g	e	-	-	b	-	-	c	-
14	-	-	b	-	-	c	-	-	a	-	-	e	g	-	f	d
15	-	-	d	f	-	g	e	-	-	a	-	-	c	-	-	b

Table 3: Communication costs for $P = Q = 16$, $r = 3$, and $s = 5$.

Communication costs for $\boldsymbol{P = Q = 16}$, $\boldsymbol{r = 3}$, and $\boldsymbol{s = 5}$

Step	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**Total**
Cost	3	3	3	2	2	1	1	**15**

Example 4

Our final example is with $P \neq Q$, just to show that the size of the two processor grids need not be the same. See Table 10 for the communication grid, which is unbalanced. The solution schedule (see Section 4.4) is composed of 4 communication steps, and this number is optimal, since processor $q = 1$ has 4 messages to receive. Note that the total cost is equal to the sum of the message lengths that processor $q = 1$ must receive; hence, it too is optimal.

3 Literature overview

We briefly survey the literature on the redistribution problem, with particular emphasis given to the work of Walker and Otto [21].

Table 4: Communication grid for $P = Q = 16$, $r = 7$, and $s = 11$. Message lengths are indicated for a vector X of size $L = 1232$.

$\boldsymbol{P = Q = 16}$, $\boldsymbol{r = 7}$, $\boldsymbol{s = 11}$, and $\boldsymbol{L = 1232}$

Sender/Recv.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Nbr of msg.
0	7	6	2	6	7	2	5	7	3	4	7	4	3	7	5	2	16
1	4	3	7	5	2	7	6	2	6	7	2	5	7	3	4	7	16
2	5	7	3	4	7	4	3	7	5	2	7	6	2	6	7	2	16
3	6	2	6	7	2	5	7	3	4	7	4	3	7	5	2	7	16
4	3	7	5	2	7	6	2	6	7	2	5	7	3	4	7	4	16
5	7	3	4	7	4	3	7	5	2	7	6	2	6	7	2	5	16
6	2	6	7	2	5	7	3	4	7	4	3	7	5	2	7	6	16
7	7	5	2	7	6	2	6	7	2	5	7	3	4	7	4	3	16
8	3	4	7	4	3	7	5	2	7	6	2	6	7	2	5	7	16
9	6	7	2	5	7	3	4	7	4	3	7	5	2	7	6	2	16
10	5	2	7	6	2	6	7	2	5	7	3	4	7	4	3	7	16
11	4	7	4	3	7	5	2	7	6	2	6	7	2	5	7	3	16
12	7	2	5	7	3	4	7	4	3	7	5	2	7	6	2	6	16
13	2	7	6	2	6	7	2	5	7	3	4	7	4	3	7	5	16
14	7	4	3	7	5	2	7	6	2	6	7	2	5	7	3	4	16
15	2	5	7	3	4	7	4	3	7	5	2	7	6	2	6	7	16
Nbr of msg.	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	

3.1 Message Generation

Several papers have dealt with the problem of efficient code generation for an HPF array assignment statement like

$$A[l_1 : u_1 : s_1] = B[l_2 : u_2 : s_2],$$

where both arrays A and B are distributed in a block-cyclic fashion on a linear processor grid. Some researchers (see Stichnoth et al.[17], van Reeuwijk et al.[19], and Wakatani and Wolfe [20]) have dealt principally with arrays distributed by using either a purely scattered or cyclic distribution (`CYCLIC(1)` in HPF) or a full block distribution (`CYCLIC(`$\lceil \frac{n}{p} \rceil$`)`, where n is the array size and p the number of processors).

Recently, however, several algorithms have been published that handle general block-cyclic `CYCLIC(k)` distributions. Sophisticated techniques involve finite-state machines (see Chatterjee et al. [3]), set-theoretic methods (see Gupta et al. [8]), Diophantine equations (see Kennedy et al. [11, 12]), Hermite forms and lattices (see Thirumalai and Ramanujam [18]), or linear programming (see Ancourt et al. [1]). A comparative survey of these algorithms can be found in Wang et al. [22], where it is reported that the most powerful algorithms can handle block-cyclic distributions as efficiently as the simpler case of pure cyclic or full-block mapping.

At the end of the message generation phase, each processor has computed several different messages (usually stored in temporary buffers). These messages must be sent to a set of receiving processors, as the examples of Section 2 illustrate. Symmetrically, each processor computes the number and length of the messages it has to receive and therefore can allocate the corresponding memory space. To summarize, when the message generation phase is completed, each processor

Table 5: Communication steps for $P = Q = 16$, $r = 7$, and $s = 11$.

Communication steps for $\boldsymbol{P = Q = 16}$, $\boldsymbol{r = 7}$, and $\boldsymbol{s = 11}$

Sender/Recv.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	c	f	n	g	d	p	i	e	m	k	a	j	l	b	h	o
1	k	m	d	h	o	c	f	n	g	e	p	i	b	l	j	a
2	h	b	l	k	a	j	m	d	i	n	e	f	o	g	c	p
3	f	n	g	e	p	i	d	l	j	c	k	m	a	h	o	b
4	l	e	i	n	c	f	o	g	d	p	h	a	m	k	b	j
5	b	l	j	c	k	m	e	i	n	d	f	o	g	a	p	h
6	o	g	c	p	i	d	l	k	b	j	m	e	h	n	a	f
7	a	i	p	d	f	n	g	b	o	h	c	l	j	e	k	m
8	m	k	b	j	l	a	h	p	e	f	n	g	d	o	i	c
9	g	a	o	i	e	l	j	c	k	m	b	h	p	d	f	n
10	i	p	e	f	n	g	a	o	h	b	l	k	c	j	m	d
11	j	d	k	m	b	h	p	a	f	o	g	c	n	i	e	l
12	d	o	h	b	m	k	c	j	l	a	i	p	e	f	n	g
13	p	c	f	o	g	e	n	h	a	l	j	b	k	m	d	i
14	e	j	m	a	h	o	b	f	p	g	d	n	i	c	l	k
15	n	h	a	l	j	b	k	m	c	i	o	d	f	p	g	e

Table 6: Communication costs for $P = Q = 16$, $r = 7$, and $s = 11$.

Communication costs for $\boldsymbol{P = Q = 16}$, $\boldsymbol{r = 7}$, and $\boldsymbol{s = 11}$

Step	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**p**	**Total**
Cost	7	7	7	7	7	6	6	5	5	4	4	3	3	2	2	2	**77**

has prepared a message for all those processors to which it must send data, and each processor possesses all the information regarding the messages it will receive (number, length, and origin).

3.2 Communication Scheduling

Little attention has been paid to the scheduling of the communications induced by the redistribution operation. Simple strategies have been advocated. For instance, Kalns and Ni [10] view the communications as a total exchange between all processors and do not further specify the operation. In their comparative survey, Wang et al. [22] use the following template for executing an array assignment statement:

1. Generate message tables, and post all receives in advance to minimize operating systems overhead
2. Pack all communication buffers
3. Carry out barrier synchronization

Table 7: Communication grid for $P = Q = 15$, $r = 3$, and $s = 5$. Message lengths are indicated for a vector X of size $L = 225$.

$\boldsymbol{P = Q = 15,\ r = 3,\ s = 5,\ L = 225}$

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

4. Send all buffers
5. Wait for all messages to arrive
6. Unpack all buffers

Although the communication phase is described more precisely, note that there is no explicit scheduling: all messages are sent simultaneously by using an asynchronous communication protocol. This approach induces a tremendous requirement in terms of buffering space, and deadlock may well happen when redistributing large arrays.

The ScaLAPACK library [4] provides a set of routines to perform array redistribution. As described by Prylli and Tourancheau [15], a total exchange is organized between processors, which are arranged as a (virtual) caterpillar. The total exchange is implemented as a succession of steps. At each step, processors are arranged into pairs that perform a send/receive operation. Then the caterpillar is shifted so that new exchange pairs are formed. Again, even though special care is taken in implementing the total exchange, no attempt is made to exploit the fact that some processor pairs may not need to communicate.

The first paper devoted to *scheduling* the communications induced by a redistribution is that of Walker and Otto [21]. They review two main possibilities for implementing the communications induced by a redistribution operation:

Wildcarded nonblocking receives Similar to the strategy of Wang et al. described above, this asynchronous strategy is simple to implement but requires buffering for all the messages to be received (hence, the total amount of buffering is as high as the total volume of data to be redistributed).

Table 8: Communication steps for $P = Q = 15$, $r = 3$, and $s = 5$.

Communication steps for $\boldsymbol{P = Q = 15}$, $\boldsymbol{r = 3}$, and $\boldsymbol{s = 5}$

Sender/Recv.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	c	-	-	d	-	-	e	-	-	a	-	-	b	-	-
1	f	h	-	b	i	-	j	d	-	g	c	-	e	a	-
2	-	e	-	-	d	-	-	c	-	-	a	-	-	b	-
3	-	i	f	-	j	c	-	g	e	-	h	b	-	d	a
4	-	-	c	-	-	e	-	-	d	-	-	a	-	-	b
5	d	-	-	e	-	-	a	-	-	b	-	-	c	-	-
6	e	j	-	a	h	-	b	i	-	c	f	-	d	g	-
7	-	d	-	-	e	-	-	a	-	-	b	-	-	c	-
8	-	g	d	-	b	f	-	j	a	-	i	e	-	h	c
9	-	-	e	-	-	a	-	-	b	-	-	c	-	-	d
10	b	-	-	f	-	-	c	-	-	d	-	-	a	-	-
11	a	b	-	c	g	-	d	h	-	e	j	-	f	i	-
12	-	c	-	-	a	-	-	b	-	-	d	-	-	e	-
13	-	a	b	-	c	d	-	e	f	-	g	h	-	j	i
14	-	-	a	-	-	b	-	-	c	-	-	d	-	-	e

Table 9: Communication costs for $P = Q = 15$, $r = 3$, and $s = 5$.

Communication costs for $\boldsymbol{P = Q = 15}$, $\boldsymbol{r = 3}$, and $\boldsymbol{s = 5}$

Step	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**Total**
Cost	3	3	3	3	3	3	2	2	2	2	**26**

Synchronous schedules A synchronized algorithm involves communication phases or steps. At each step, each participating processor posts a receive, sends data, and then waits for the completion of the receive. But several factors can lead to performance degradation. For instance, some processors may have to wait for others before they can receive any data. Or hot spots can arise if several processors attempt to send messages to the same processor at the same step. To avoid these drawbacks, Walker and Otto propose to *schedule* messages so that, at each step, each processor sends no more than one message and receives no more than one message. This strategy leads to a synchronized algorithm that is as efficient as the asynchronous version, as demonstrated by experiments (written in MPI [16]) on the IBM SP-1 and Intel Paragon, while requiring much less buffering space.

Walker and Otto [21] provide synchronous schedules only for some special instances of the redistribution problem, namely, to change the array distribution from `CYCLIC(r)` on a P-processor linear grid to `CYCLIC(Kr)` on a grid of same size. Their main result is to provide a schedule composed of K steps. At each step, all processors send and receive exactly one message. If K is smaller than P, the size of the grid, there is a dramatic improvement over a traditional all-to-all implementation.

Table 10: Communication grid for $P = 12$, $Q = 8$, $r = 4$, and $s = 3$. Message lengths are indicated for a vector X of size $L = 48$.

$\boldsymbol{P = 12,\ Q = 8,\ r = 4,\ s = 3,\ L = 48}$

Sender/Recv.	0	1	2	3	4	5	6	7	Nbr of msg.
0	3	1	-	-	-	-	-	-	2
1	-	2	2	-	-	-	-	-	2
2	-	-	1	3	-	-	-	-	2
3	-	-	-	-	3	1	-	-	2
4	-	-	-	-	-	2	2	-	2
5	-	-	-	-	-	-	1	3	2
6	3	1	-	-	-	-	-	-	2
7	-	2	2	-	-	-	-	-	2
8	-	-	1	3	-	-	-	-	2
9	-	-	-	-	3	1	-	-	2
10	-	-	-	-	-	2	2	-	2
11	-	-	-	-	-	-	1	3	2
Nbr of msg.	2	4	4	2	2	4	4	4	

Our aim in this article is to extend Walker and Otto's work in order to solve the 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. We retain their original idea: schedule the communications into steps. At each step, each participating processor neither sends nor receives more than one message, to avoid hot spots and resource contentions. As explained in [21], this strategy is well suited to current parallel architectures. In Section 4.3.1, we give a precise framework to model the cost of a redistribution.

4 Main Results

4.1 Problem Formulation

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 by using a `CYCLIC(s)` distribution on Q processors (numbered from $q = 0$ to $q = Q - 1$). Equivalently, we perform the HPF assignment $Y = X$, where X is `CYCLIC(r)` on a P-processor grid, while Y is `CYCLIC(s)` on a Q-processor grid[1].

The block-cyclic data distribution maps the global index i of vector X (i.e., element $X[i]$) onto a processor index p, a block index l, and an item index x, local to the block (with all indices starting at 0). The mapping $i \longrightarrow (p, l, x)$ may be written as

$$i \longrightarrow \left(p = \lfloor i/r \rfloor \bmod P,\ l = \frac{\lfloor i/r \rfloor}{P},\ x = i \bmod r\right). \tag{1}$$

We derive the relation

$$i = (P \times l + p) \times r + x. \tag{2}$$

[1]The more general assignment $Y[a : ..] = X[b : ..]$ can be dealt with similarly.

Table 11: Communication steps for $P = 12$, $Q = 8$, $r = 4$, and $s = 3$.

Communication steps for $\boldsymbol{P = 12}$, $\boldsymbol{Q = 8}$, $\boldsymbol{r = 4}$, and $\boldsymbol{s = 3}$

Sender/Recv.	0	1	2	3	4	5	6	7
0	a	c	-	-	-	-	-	-
1	-	b	a	-	-	-	-	-
2	-	-	c	a	-	-	-	-
3	-	-	-	-	a	c	-	-
4	-	-	-	-	-	b	a	-
5	-	-	-	-	-	-	c	a
6	b	d	-	-	-	-	-	-
7	-	a	b	-	-	-	-	-
8	-	-	d	b	-	-	-	-
9	-	-	-	-	b	d	-	-
10	-	-	-	-	-	a	b	-
11	-	-	-	-	-	-	d	b

Table 12: Communication costs for $P = 12$, $Q = 8$, $r = 4$, and $s = 3$.

Communication costs for $\boldsymbol{P = 12}$, $\boldsymbol{Q = 8}$, $\boldsymbol{r = 4}$, and $\boldsymbol{s = 3}$

Step	**a**	**b**	**c**	**d**	**Total**
Cost	3	3	1	1	**8**

Similarly, since Y is distributed `CYCLIC(s)` on a Q-processor grid, its global index j is mapped as $j \longrightarrow (q, m, y)$, where $j = (Q \times m + q) \times s + y$. We then get the redistribution equation

$$i = (P \times l + p) \times r + x = (Q \times m + q) \times s + y. \tag{3}$$

Let $L = lcm(Pr, Qs)$ be the least common multiple of Pr and Qs. Elements i and $L+i$ of X are initially distributed onto the same processor $p = \lfloor i/r \rfloor \bmod P$ (because L is a multiple of Pr, hence r divides L, and P divides $L \div r$). For a similar reason, these two elements will be redistributed onto the same processor $q = \lfloor i/s \rfloor \bmod Q$. In other words, the redistribution pattern repeats after each slice of L elements. Therefore, we restrict the discussion to a vector X of length L in the following. Let $g = \gcd(Pr, Qs)$ (of course $Lg = PrQs$). The bounds in equation (3) become

$$\begin{cases} 0 \le p < P & 0 \le q < Q \\ 0 \le l < \frac{L}{Pr} = \frac{Qs}{g} & 0 \le m < \frac{L}{Qs} = \frac{Pr}{g} \\ 0 \le x < r & 0 \le y < s. \end{cases} \tag{4}$$

Definition 1 *Given the distribution parameters r and s, and the grid parameters P and Q, the* **redistribution problem** *is to determine all the messages to be exchanged, that is, to find all values of p and q such that the redistribution equation (3) has a solution in the unknowns l, m, x, and y, subject to the bounds in Equation (4). Computing the number of solutions for a given processor pair (p, q) will give the* length *of the message.*

We start with a simple lemma that leads to a handy simplification:

Lemma 1 *We can assume that r and s are relatively prime, that is,* $\gcd(r, s) = 1$.

Proof The redistribution equation (3) can be expressed as

$$pr - qs = z + (Pr.l - Qs.m), \tag{5}$$

where $z = y - x \in [1 - r, s - 1]$. Let $\Delta = \gcd(r, s)$, $r = \Delta r'$ and $s = \Delta s'$. Equation (3) can be expressed as

$$\Delta(pr' - qs') = z + \Delta(Pr'.l - Qs'.m).$$

If it has a solution for a given processor pair (p, q), then Δ divides z, $z = \Delta z'$, and we deduce a solution for the redistribution problem with r', s', P, and Q. ■

Let us illustrate this simplification on one of our motivating examples:

Back to Example 3

Note that we need to scale message lengths to move from a redistribution operation where r and s are relatively prime to one where they are not. Let us return to Example 3 and assume for a while that we know how to build the communication grid in Table 7. To deduce the communication grid for $r = 12$ and $s = 20$, say, we keep the same messages, but we scale all lengths by $\Delta = \gcd(r, s) = 4$. This process makes sense because the new size of a vector slice is ΔL rather than L. See Table 13 for the resulting communication grid. Of course, the scheduling of the communications will remain the same as with $r = 3$ and $s = 5$, while the cost in Table 9 will be multiplied by Δ.

4.2 Communication Pattern

Lemma 2 *Consider a redistribution with parameters r, s, P, and Q, and assume that* $\gcd(r, s) = 1$. *Let* $g = \gcd(Pr, Qs)$. *The communication pattern induced by the redistribution operation is a complete all-to-all operation if and only if*

$$g \leq r + s - 1.$$

Proof We rewrite Equation (5) as $ps - qr = z + \lambda \times g$ because $Pr.l - Qs.m$ is an arbitrary multiple of g. Since z lies in the interval $[1 - r, s - 1]$ whose length is $r + s - 1$, it is guaranteed that a multiple of g can be found within this interval if $g \leq r + s - 1$. Conversely, assume that $g \geq r + s$: we will exhibit a processor pair (p, q) exchanging no message. Indeed, $p = P - 1$ and $q = 0$ is the desired processor pair. To see this, note that $pr - qs = -r \bmod g$ (because g divides Pr); hence, no multiple of g can be added to $pr - qs$ so that it lies in the interval $[1 - r, s - 1]$, Therefore, no message will be sent from p to q during the redistribution.[2] ■

In the following, our aim is to characterize the pairs of processors that need to communicate during the redistribution operation (in the case $g \geq r + s$). Consider the following function f:

$$\left\{ \begin{array}{ccc} [0..P-1] \times [0..Q-1] & \longrightarrow & \mathbb{Z}_g \\ (p, q) & \longrightarrow & f(p, q) = pr - qs \bmod g \end{array} \right. \tag{6}$$

[2]For another proof, see Petitet [14].

Table 13: Communications for $P = Q = 15$, $r = 12$, and $s = 20$. Message lengths are indicated for a vector X of size $L = 900$.

$\boldsymbol{P = Q = 15,\ r = 12,\ s = 20,\ L = 900}$

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

Function f maps each processor pair (p, q) onto the congruence class of $pr - qs$ modulo g. According to the proof of Lemma 2, p sends a message to q if and only if $f(p, q) \in [1 - r, s - 1] \pmod{g}$. Let us illustrate this process by using one of our motivating examples.

Back to Example 4

In this example, $P = 12$, $Q = 8$, $r = 4$ and $s = 3$. We have $g = 24$. Take $p = 11$ (as in the proof of Lemma 2). If $q = 0$, $f(p, q) = -4 \notin [-3, 2]$, and q receives no message from p. But if $q = 6$, $f(p, q) = 2 \in [-3, 2]$, and q does receive a message (see Table 10 to check this).

Definition 2 *For $0 \leq k < g$, let class$(k) = f^{-1}(k)$, that is,*
$$f^{-1}(k) = \{(p, q) \in [0..P-1] \times [0..Q-1];\ \ f(p, q) = k\}.$$

To characterize classes, we introduce integers u and v such that

$$r \times u - s \times v = 1$$

(the extended Euclid algorithm provides such numbers for relatively prime r and s). We have the following result.

Proposition 1 *Assume that* $\gcd(r, s) = 1$. *For $0 \leq k < g$,*

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

Proof First, to see that $\frac{PQ}{g}$ indeed is an integer, note that $PQ = PQ(ru - sv) = Pr.Qu - Qs.Pv$. Since g divides both Pr and Qs, it divides PQ.

Two different classes are disjoint (by definition). It turns out that all classes have the same number of elements. To see this, note that for all $k \in [0, g-1]$,

$$(p, q) \in \mathit{class}(0) \iff (p + ku \bmod P,\ q + kv \bmod Q) \in \mathit{class}(k).$$

Indeed, $p + ku \bmod P = p + ku + dP$ for some integer d, $q + kv \bmod Q = q + kv + d'Q$ for some integer d', and

$$\begin{aligned} f(p + ku \bmod P, q + kv \bmod Q) &= (p + ku + dP)r - (q + kv + d'Q)s \bmod g \\ &= pr - qs + k + dPr + d'Qs \bmod g \\ &= f(p, q) + k \bmod g. \end{aligned}$$

Since there are g classes, we deduce that the number of elements in each class is $\frac{PQ}{g}$.

Next, we see that $(p_\lambda, q_\lambda) = (\lambda s \bmod p,\ \lambda r \bmod Q) \in \mathit{class}(0)$ for $0 \leq \lambda < \frac{PQ}{g}$ (because $p_\lambda r - q_\lambda s = 0 \bmod g$).

Finally, $(p_\lambda, q_\lambda) = (p_{\lambda'}, q_{\lambda'})$ implies that P divides $(\lambda - \lambda')s$ and Q divides $(\lambda - \lambda')r$. Therefore, both Pr and Qs divide $(\lambda - \lambda')rs$; hence, $L = lcm(Pr, Qs) = \frac{Pr.Qs}{g}$ divides $(\lambda - \lambda')rs$. We deduce that $\frac{PQ}{g}$ divides $(\lambda - \lambda')$; hence all the processors pairs (p_λ, q_λ) for $0 \leq \lambda < \frac{PQ}{g}$ are distinct. We have thus enumerated $\mathit{class}(0)$. ■

Definition 3 *Consider a redistribution with parameters r, s, P, and Q, and assume that $\gcd(r, s) = 1$. Let $\mathrm{length}(p, q)$ be the length of the message sent by processor p to processor q to redistribute a single slice vector X of size $L = lcm(Pr, Qs)$.*

As we said earlier, the communication pattern repeats for each slice, and the value reported in the communication grid tables of Section 2 are for a single slice; that is, they are equal to $\mathit{length}(p, q)$. Classes are interesting because they represent homogeneous communications: all processor pairs in a given class exchange a message of same length.

Proposition 2 *Assume that $\gcd(r, s) = 1$, and let $L = lcm(Pr, Qs)$ be the length of the vector X to be redistributed. Let $\mathrm{vol}(k)$ be the piecewise function given by Figure 1 for $k \in [1 - r, s - 1]$.*

- *If $r + s - 1 \leq g$, then for $k \in [1 - r, s - 1]$,*

  $$(p, q) \in \mathit{class}(k) \Rightarrow \mathit{length}(p, q) = \mathit{vol}(k)$$

  (recall that if $(p, q) \in \mathrm{class}(k)$ where $k \notin [1 - r, s - 1]$, then p sends no message to q).

- *If $g \leq r + s$, then for $k \in [0, g - 1]$,*

  $$(p, q) \in \mathit{class}(k) \Rightarrow \mathit{length}(p, q) = \sum_{k' \in [1-r, s-1];\ k' \bmod g = k} \mathit{vol}(k').$$

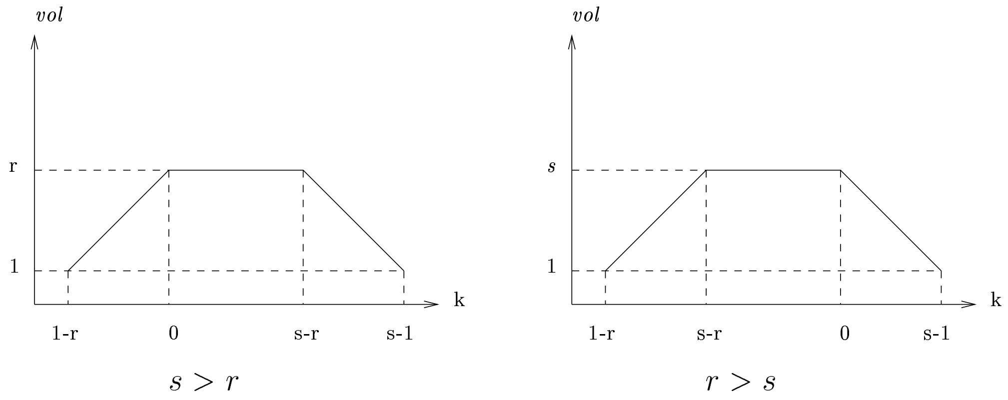

Figure 1: The piecewise linear function *vol*.

Proof We simply count the number of solutions to the redistribution equation $pr - qs = y - x \bmod g$, where $0 \leq x < r$ and $0 \leq y < s$. We easily derive the piecewise linear *vol* function represented in Figure 1. ■

We now know how to build the communication tables in Section 2. We still have to derive a schedule, that is, a way to organize the communications as efficiently as possible.

4.3 Communication Schedule

4.3.1 Communication Model

According to the previous discussion, we concentrate on schedules that are composed of several successive steps. At each step, each sender should send no more than one message; symmetrically, each receiver should receive no more than one message. We give a formal definition of a schedule as follows.

Definition 4 *Consider a redistribution with parameters r, s, P, and Q.*

- *The* **communication grid** *is a $P \times Q$ table with a nonzero entry $length(p, q)$ in position (p, q) if and only if p has to send a message to q.*
- *A* **communication step** *is a collection of pairs $\{(p_1, q_1), (p_2, q_2), \ldots, (p_t, q_t)\}$ such that $p_i \neq p_j$ for $1 \leq i < j \leq t$, $q_i \neq q_j$ for $1 \leq i < j \leq t$, and $length(p_i, q_i) > 0$ for $1 \leq i \leq t$. A communication step is* **complete** *if $t = \min(P, Q)$ (either all senders or all receivers are active) and is incomplete otherwise. The cost of a communication step is the maximum value of its entries, in other words, $\max\{length(p_i, q_i);\ 1 \leq i \leq t\}$*
- *A schedule is a succession of communication steps such that each nonzero entry in the communication grid appears in one and only one of the steps. The cost of a schedule may be evaluated in two ways:*
  1. *the* **number of steps** *NS, which is simply the number of communication steps in the schedule; or*
  2. *the* **total cost** *TC, which is the sum of the cost of each communication step (as defined above).*

The communication grid, as illustrated in the tables of Section 2, summarizes the length of the required communications for a single slice vector, that is, a vector of size $L = lcm(Pr, Qs)$. The motivation for evaluating schedules via their number of steps or via their total cost is as follows:

- The number of steps NS is the number of synchronizations required to implement the schedule. If we roughly estimate each communication step involving all processors (a permutation) as a measure unit, the number of steps is the good evaluation of the cost of the redistribution.
- We may try to be more precise. At each step, several messages of different lengths are exchanged. The duration of a step is likely to be related to the longest length of these messages. A simple model would state that the cost of a step is $\alpha + \max\{length(p_i, q_i)\} \times \tau$, where α is a start-up time and τ the inverse of the bandwidth on a physical communication link. Although this expression does not take hot spots and link contentions into account, it has proven useful on a variety of machines [4, 6]. The cost of a redistribution, according to this formula, is the affine expression

$$\alpha \times NS + \beta \times TC$$

with motivates our interest in both the number of steps and the total cost.

4.3.2 A Simple Case

There is a very simple characterization of processor pairs in each class, in the special case where r and Q, as well as s and P, are relatively prime.

Proposition 3 *Assume that* $\gcd(r, s) = 1$. *If* $\gcd(r, Q) = \gcd(s, P) = 1$, *then for* $0 \le k < g$,

$$(p, q) \in class(k) \iff q = s^{-1}(pr - k) \bmod g \iff p = r^{-1}(qs + k) \bmod g$$

(s^{-1} and r^{-1} respectively denote the inverses of s and r modulo g).

Proof Since $\gcd(r, s) = \gcd(r, Q) = 1$, r is relatively prime with Qs, hence with g. Therefore the inverse of r modulo g is well defined (and can be computed by using the extended Euclid algorithm applied to r and g). Similarly, the inverse of s modulo g is well defined, too. The condition $pr - qs = k \bmod g$ easily translates into the conditions of the proposition. ■

In this simple case, we have a very nice solution to our scheduling problem. Assume first that $g \ge r + s - 1$. Then we simply schedule communications class by class. Each class is composed of $\frac{PQ}{g}$ processor pairs that are equally distributed on each row and column of the communication grid: in each class, there are exactly $\frac{Q}{g}$ sending processors per row, and $\frac{P}{g}$ receiving processors per column. This is a direct consequence of Proposition 3. Note that g does divide P and Q: under the hypothesis $\gcd(r, Q) = \gcd(s, P) = 1$, $g = \gcd(Pr, Qs) = \gcd(P, Qs) = \gcd(P, Q)$.

To schedule a class, we want each processor $p = \alpha g + p'$, where $0 \le \alpha < \frac{P}{g}$, $0 \le p' < g$, to send a message to each processor $q = \beta g + q'$, where $0 \le \beta < \frac{Q}{g}$, $0 \le q' < g$, and $q' = s^{-1}(p'r - k) \bmod g$ (or equivalently, $p' = r^{-1}(q's + k) \bmod g$ if we look at the receiving side). In other words, the processor in position p' within each block of g elements must send a message to the processor in position q' within each block of g elements. This can be done in $\frac{\max(P,Q)}{g}$ complete steps of $\min(P, Q)$ messages. For instance, if there are five blocks of senders ($P = 5g$) and three blocks of receivers ($Q = 3g$), we have 5 steps where 3 blocks of senders send messages to 3 blocks of receivers. We can use any algorithm for generating the block permutation; the ordering of the communications between blocks is irrelevant.

If $g = r + s - 1$, we have an all-to-all communication scheme, as illustrated in Example 2, but our scheduling by classes leads to an algorithm where all messages have the same length at a given step. If $g < r + s - 1$, we have fewer classes than $r + s - 1$. In this case we simply regroup classes that are equivalent modulo g and proceed as before.

We summarize the discussion by the following result

Proposition 4 *Assume that* $\gcd(r, s) = 1$. *If* $\gcd(r, Q) = \gcd(s, P) = 1$, *then scheduling each class successively leads to an optimal communication scheme, in terms of both the number of steps and the total cost.*

Proof Assume without loss of generality that $P \geq Q$. According to the previous discussion, if $g \geq r+s-1$, we have $r+s-1$ (the number of classes) times $\frac{P}{g}$ (the number of steps for each class) communication steps. At each step we schedule messages of the same class k, hence of same length $vol(k)$. If $g < r+s-1$, we have g times $\frac{P}{g}$ communication steps, each composed of messages of the same length (namely, $\sum_{k' \in [1-r, s-1];\ k' \bmod g = k} vol(k')$ when processing a given class $k \in [0, g-1]$. ■

Remark 1 Walker and Otto [21] deal with a redistribution with $P = Q$ and $s = Kr$. We have shown that going from r to Kr can be simplified to going from $r = 1$ to $s = K$. If $\gcd(K, P) = 1$, the technique described in this section enables us to retrieve the results of [21].

4.4 The General Case

When $\gcd(s, P) = s' > 1$, entries of the communication grid may not be evenly distributed on the rows (senders). Similarly, when $\gcd(r, Q) = r' > 1$, entries of the communication grid may not be evenly distributed on the columns (receivers).

Back to Example 3

We have $P = 15$ and $s = 5$; hence $s' = 5$. We see in Table 7 that some rows of the communication grid have 5 nonzero entries (messages), while other rows have 10. Similarly, $Q = 15$ and $r = 3$; hence $r' = 3$. Some columns of the communication grid have 6 nonzero entries, while other columns have 10.

Our first goal is to determine the maximum number of nonzero entries in a row or a column of the communication grid. We start by analyzing the distribution of each class.

Lemma 3 *Let* $\gcd(s, P) = s'$ *and* $\gcd(r, Q) = r'$. *Let* $P = P's'$ *and* $Q = Q's'$, *and* $g_0 = \gcd(P', Q')$. *Then* $g = r's'g_0$, *and in any class* $\mathsf{class}(k)$, $k \in [0, g-1]$, *the processors pairs are distributed as follows:*

- *There are* $\frac{P'}{g_0}$ *entries per column in* Q' *columns of the grid, and none in the remaining columns.*
- *There are* $\frac{Q'}{g_0}$ *entries per row in* P' *rows of the grid, and none in the remaining rows.*

Proof First let us check that $g = r's'g_0$. We write $r = r'r"$ and $s = s's"$. We have $Pr = (P's').(r'r") = r's'.(P'r")$. Similarly, $Qs = r's'.(Q's")$. Thus $g = \gcd(Pr, Qs) = r's' \gcd(P'r", Q's")$. Since $r"$ is relatively prime with Q' (by definition of r') and with $s"$ (because $\gcd(r, s) = 1$), we have $\gcd(P'r", Q's") = \gcd(P', Q's")$. Similarly, $\gcd(P', Q's") = \gcd(P', Q') = g_0$.

There are $\frac{PQ}{g}$ elements per class. Since all classes are obtained by a translation of $class(0)$, we can restrict ourselves to discussing the distribution of elements in this class. The formula in Lemma 1 states that $class(0) = \{\begin{pmatrix} p \\ q \end{pmatrix} = \lambda \begin{pmatrix} s \\ r \end{pmatrix} \bmod \begin{pmatrix} P \\ Q \end{pmatrix}\}$ for $0 \leq \lambda < \frac{PQ}{g}$. But $\lambda s \bmod P$ can take only those values that are multiple of s' and $\lambda r \bmod Q$ can take only those values that are multiple of r', hence the result. To check the total number of elements, note that $\frac{PQ}{g} = \frac{P's'.Q'r'}{r's'g_0} = \frac{P'Q'}{g_0}$. ■

Let us illustrate Lemma 3 with one of our motivating examples.

Back to Example 3

Elements of each class should be located on $\frac{P'}{g_0} = \frac{3}{1} = 3$ rows and $\frac{Q'}{g_0} = \frac{5}{1} = 5$ columns of the processor grid. Let us check $class(1)$ for instance. Indeed we have the following.

$$\begin{aligned} class(1) = \{ \quad & (2,1), (7,4), (12,7), (2,10), (7,13), (12,1), \\ & (2,4), (7,7), (12,10), (2,13), (7,1), (12,4), (2,7), (7,10), (12,13) \ \} \end{aligned}$$

Lemma 3 shows that we cannot use a schedule based on classes: considering each class separately would lead to incomplete communication steps. Rather, we should build up communication steps by mixing elements of several classes, in order to use all available processors. The maximum number of elements in a row or column of the communication grid is an obvious lower bound for the number of steps of any schedule, because each processor cannot send (or receive) more than one message at any communication step.

Proposition 5 *Assume that* $\gcd(r, s) = 1$ *and that* $r + s - 1 \leq g$ *(otherwise the communication grid is full). If we use the notation of Lemma 3,*

1. *the maximum number* m_R *of elements in a row of the communication grid is* $m_R = \frac{Q'}{g_0}\lceil \frac{r+s-1}{s'} \rceil$; *and*

2. *the maximum number* m_C *of elements in a column of the communication grid is* $m_C = \frac{P'}{g_0}\lceil \frac{r+s-1}{r'} \rceil$.

Proof According to Lemma 1, two elements of $class(k)$ and $class(k')$ are on the same row of the communication grid if $\lambda s + ku = \lambda' s + k'u \bmod P$ for some λ and λ' in the interval $[0, \frac{PQ}{g} - 1]$. Necessarily, s', which divides P and $(\lambda - \lambda')s$, divides $(k - k')u$. But we have $ru - sv = 1$, and s is relatively prime with u. A fortiori s' is relatively prime with u. Therefore s' divides $k - k'$.

Classes share the same rows of the processor grid if they are congruent modulo s'. This induces a partition on classes. Since there are exactly $\frac{Q'}{g_0}$ elements per row in each class, and since the number of classes congruent to the same value modulo s' is either $\lfloor \frac{r+s-1}{s'} \rfloor$ or $\lceil \frac{r+s-1}{s'} \rceil$, we deduce the value of m_R. The value of m_C is obtained similarly. ■

It turns out that the lower bound for the number of steps given by Lemma 5 can indeed be achieved.

Theorem 1 *Assume that* $\gcd(r, s) = 1$ *and that* $r + s - 1 \leq g$ *(otherwise the communication grid is full), and use the notation of Lemma 3 and Lemma 5. The optimal number of steps* NS_{opt} *for any schedule is*

$$NS_{opt} = \max\{m_R, m_C\}.$$

Proof We already know that the number of steps NS of any schedule is greater than or equal to $\max\{m_R, m_C\}$. We give a constructive proof that this bound is tight: we derive a schedule whose number of steps is $\max\{m_R, m_C\}$. To do so, we borrow some material from graph theory. We view the communication grid as a graph $G = (V, E)$, 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
- $e = (p, q) \in E$ if and only if the entry (p, q) in the communication grid is nonzero.

G is a bipartite graph (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 $d_G = \max\{m_R, m_C\}$. According to König's edge coloring theorem, the edge coloring number of a bipartite graph is equal to its degree (see [7, vol. 2, p.1666] or Berge [2, 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 [2, p. 130]). To define the schedule, we simply let the matchings at each iteration represent the communication steps. ■

Remark 2 The proof of Theorem 1 gives a bound for the complexity of determining the optimal number of steps. The best known maximum matching algorithm for bipartite graphs is due to Hopcroft and Karp [9] and has cost $O(|V|^{\frac{5}{2}})$. Since there are at most $\max(P, Q)$ iterations to construct the schedule, we have a procedure in $O((|P| + |Q|)^{\frac{7}{2}}$ to construct a schedule whose number of steps is minimal.

4.5 Schedule Implementation

Our goal is twofold when designing a schedule:

- minimize the number of steps of the schedule, and
- minimize the total cost of the schedule.

We have already explained how to view the communication grid as a bipartite graph $G = (V, E)$. More accurately, we view it as an edge-weighted bipartite graph: the edge of each edge (p, q) is the length $length(p, q)$ of the message sent by processor p to processor q.

We adopt the following two strategies:

stepwise If we specify the number of steps, we have to choose at each iteration a maximum matching that saturates all nodes of maximum degree. Since we are free to select any of such matchings, a natural idea is to select among all such matchings one of maximum weight (the weight of a matching is defined as the sum of the weight of its edges).

greedy If we specify the total cost, we can adopt a greedy heuristic that selects a maximum weighted matching at each step. We might end up with a schedule having more than NS_{opt} steps but whose total cost is less.

To implement both approaches, we rely on a linear programming framework (see [7, chapter 30]). Let A be the $|V| \times |E|$ incidence matrix of G, where

$$a_{ij} = \begin{cases} 1 \text{ if edge } j \text{ is incident to vertex } i \\ 0 \text{ otherwise} \end{cases}$$

Since G is bipartite, A is totally unimodular (each square submatrix of A has determinant 0, 1 or -1). The matching polytope of G is the set of all vectors $x \in \mathbb{Q}^{|E|}$ such that

$$\begin{cases} x(e) \geq 0 & \forall e \in E \\ \sum_{e \ni v} x(e) \leq 1 & \forall v \in V \end{cases} \tag{7}$$

(intuitively, $x(e) = 1$ iff edge e is selected in the matching). Because the polyhedron determined by Equation 7 is integral, we can rewrite it as the set of all vectors $x \in \mathbb{Q}^{|E|}$ such that

$$x \geq 0, \; Ax \leq b \text{ where } b = \begin{pmatrix} 1 \\ 1 \\ \dots \\ 1 \end{pmatrix} \in \mathbb{Q}^{|V|}. \tag{8}$$

To find a maximum weighted matching, we look for x such that

$$\max\{c^t x; \; x \geq 0, \; Ax \leq b\}, \tag{9}$$

where $c \in \mathbb{N}^{|E|}$ is the weight vector.

If we choose the *greedy* strategy, we simply repeat the search for a maximum weighted matching until all communications are done. If we choose the *stepwise* strategy, we have to ensure that, at each iteration, all vertices of maximum degree are saturated. This task is not difficult: for each vertex v of maximum degree in position i, we replace the constraint $(Ax)_i \leq 1$ by $(Ax)_i = 1$. This translates into $Y^t Ax = k$, where k is the number of maximum degree vertices and $Y \in \{0,1\}^{|V|}$ whose entry in position i is 1 iff the ith vertex is of maximum degree. We note that in either case we have a polynomial method. Because the matching polyhedron is integral, we solve a rational linear problem but are guaranteed to find integer solutions.

To see the fact that the greedy strategy can be better than the stepwise strategy in terms of total cost, consider the following example.

Example 5

Consider a redistribution problem with $P = 15$, $Q = 6$, $r = 2$, and $s = 3$. The communication grid is given in Table 14. The stepwise strategy is illustrated in Table 15: the number of steps is equal to 10, which is optimal, but the total cost is 20 (see Table 16). The greedy strategy requires more steps, namely, 12 (see Table 17), but its total cost is 18 only (see Table 18).

Table 14: Communication grid for $P = 15$, $Q = 6$, $r = 2$, and $s = 3$. Message lengths are indicated for a vector X of size $L = 90$.

$\boldsymbol{P = 15}$, $\boldsymbol{Q = 6}$, $\boldsymbol{r = 2}$, $\boldsymbol{s = 3}$, $\boldsymbol{L = 90}$

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

4.5.1 Comparison with Walker and Otto's Strategy

Walker and Otto [21] deal with a redistribution where $P = Q$ and $s = Kr$. We know that going from r to Kr can be simplified to going from $r = 1$ to $s = K$. If $\gcd(K, P) = 1$, we apply the results of Section 4.3.2 (see Remark 1). In the general case ($s' = \gcd(K, P) \geq 1$), classes are evenly distributed among the columns of the communication grid (because $r' = r = 1$), but not necessarily among the rows. However, all rows have the same total number of nonzero elements because s' divides $r + s - 1 = K$. In other words, the bipartite graph is regular. And since $P = Q$, any maximum matching is a perfect matching.

Because $r = 1$, all messages have the same length: $length(p, q) = 1$ for every nonzero entry (p, q) in the communication grid. As a consequence, the stepwise strategy will lead to an optimal schedule, in terms of both the number of steps and the total cost. Note that $NS_{opt} = K$ under the hypotheses of Walker and Otto: using the notation of Lemma 5, we have $g = P = Q$. Since $r = r' = 1$, $Q' = Q$; $s' = \gcd(K, P)$, $P = s'P'$, and $g_0 = P'$. We have

$$m_R = \frac{Q'}{g_0} \lceil \frac{r + s - 1}{s'} \rceil = 1.s = K,$$

$$m_C = \frac{P'}{g_0} \lceil \frac{r + s - 1}{r'} \rceil = \frac{P}{P'} \frac{s}{s'} = s = K.$$

Note that the same result applies when $r = 1$ and $P \neq Q$. Because the graph is regular and all entries in the communication grid are equal, we have the following theorem, which extends Walker and Otto main result [21].

Table 15: Communication steps (stepwise strategy) for $P = 15$, $Q = 6$, $r = 2$, and $s = 3$.

Stepwise strategy for $\boldsymbol{P = 15}$, $\boldsymbol{Q = 6}$, $\boldsymbol{r = 2}$, and $\boldsymbol{s = 3}$

Sender/Recv.	0	1	2	3	4	5
0	a	-	b	-	c	-
1	h	i	j	e	g	f
2	-	b	-	c	-	a
3	b	-	c	-	a	-
4	j	e	i	h	f	g
5	-	c	-	a	-	b
6	c	-	a	-	b	-
7	i	h	f	g	j	d
8	-	a	-	b	-	c
9	f	-	e	-	d	-
10	g	j	d	f	i	h
11	-	g	-	d	-	e
12	d	-	h	-	e	-
13	e	f	g	j	h	i
14	-	d	-	i	-	j

Table 16: Communication costs (stepwise strategy) for $P = 15$, $Q = 6$, $r = 2$, and $s = 3$.

Stepwise strategy for $\boldsymbol{P = 15}$, $\boldsymbol{Q = 6}$, $\boldsymbol{r = 2}$, and $\boldsymbol{s = 3}$

Step	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**Total**
Cost	2	2	2	2	2	2	2	2	2	2	**20**

Proposition 6 *Consider a redistribution problem with $r = 1$ (and arbitrary P, Q and s). The schedule generated by the stepwise strategy is optimal, in terms of both the number of steps and the total cost.*

The strategy presented in this article makes it possible to directly handle a redistribution from an arbitrary `CYCLIC(r)` to an arbitrary `CYCLIC(s)`. In contrast, the strategy advocated by Walker and Otto requires two redistributions: one from `CYCLIC(r)` to `CYCLIC(lcm(r,s))` and a second one from `CYCLIC(lcm(r,s))` to `CYCLIC(s)`.

5 MPI Experiments

This section presents results for runs on the Intel Paragon for the redistribution algorithm described in Section 4.

Table 17: Communication steps (greedy strategy) for $P = 15$, $Q = 6$, $r = 2$, and $s = 3$.

Greedy strategy for $\boldsymbol{P = 15}$, $\boldsymbol{Q = 6}$, $\boldsymbol{r = 2}$, and $\boldsymbol{s = 3}$

Sender/Recv.	0	1	2	3	4	5
0	a	-	b	-	c	-
1	j	k	l	h	g	i
2	-	b	-	c	-	a
3	b	-	c	-	a	-
4	i	g	h	f	e	j
5	-	c	-	a	-	b
6	c	-	a	-	b	-
7	h	e	g	i	j	d
8	-	a	-	b	-	c
9	e	-	f	-	d	-
10	f	i	d	g	h	k
11	-	f	-	d	-	e
12	d	-	e	-	f	-
13	g	h	i	j	k	l
14	-	d	-	e	-	f

Table 18: Communication costs (greedy strategy) for $P = 15$, $Q = 6$, $r = 2$, and $s = 3$.

Greedy strategy for $\boldsymbol{P = 15}$, $\boldsymbol{Q = 6}$, $\boldsymbol{r = 2}$, and $\boldsymbol{s = 3}$

Step	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**Total**
Cost	2	2	2	2	2	2	1	1	1	1	1	1	**18**

5.1 Description

Experiments have been executed on the Intel Paragon XP/S 5 computer with a C program calling routines from the MPI library. MPI is chosen for portability and reusability reasons. Schedules are composed of steps, and each step generates at most one send and/or one receive per processor. Hence we used only one-to-one communication primitives from MPI.

Our main objective was a comparison of our new scheduling strategy against the current redistribution algorithm of ScaLAPACK [15], namely, the "caterpillar" algorithm that was briefly summarized in Section 3.2. To run our scheduling algorithm, we proceed as follows:

1. Compute schedule steps using the results of Section 4.

2. Pack all the communication buffers.

3. Carry out barrier synchronization.

4. Start the timer.

5. Execute communications using our redistribution algorithm (resp. the caterpillar algorithm).

6. Stop the timer.

7. Unpack all buffers.

The maximum of the timers is taken over all processors. We emphasize that we do not take the cost of message generation into account: we compare communication costs only.

Instead of the caterpillar algorithm, we could have used the `MPI_ALLTOALLV` communication primitive. It turns out that the caterpillar algorithm leads to better performance than the `MPI_ALLTOALLV` for all our experiments (the difference is roughly 20% for short vectors and 5% for long vectors).

We use the same physical processors for the input and the output processor grid. Results are not very sensitive to having the same grid or disjoint grids for senders and receivers.

5.2 Results

Three experiments are presented below. The first two experiments use the schedule presented in Section 4.3.2, which is optimal in terms of both the number of steps NS and the total cost TC. The third experiment uses the schedule presented in Section 4.4, which is optimal only in terms of NS.

Back to Example 1

The first experiment corresponds to Example 1, with $P = Q = 16$, $r = 3$, and $s = 5$. The redistribution schedule requires 7 steps (see Table 3). Since all messages have same length, the theoretical improvement over the caterpillar algorithm, which as 16 steps, is $7/16 \approx 0.44$. Figure 2 shows that there is a significant difference between the two execution times. The theoretical ratio is obtained for very small vectors (e.g., of size 1200 double-precision reals). This result is not surprising because start-up times dominate the cost for small vectors. For larger vectors the ratio varies between 0.56 and 0.64. This is due to contention problems: our scheduler needs only 7 step, but each step generates 16 communications, whereas each of the 16 steps of the caterpillar algorithm generates fewer communications (between 6 and 8 per step), thereby generating less contention.

Back to Example 2

The second experiment corresponds to Example 2, with $P = Q = 16$ processors, $r = 7$, and $s = 11$. Our redistribution schedule requires 16 steps, and its total cost is $TC = 77$ (see Table 6). The caterpillar algorithm requires 16 steps, too, but at each step at least one processor sends a message of length (proportional to) 7, hence a total cost of 112. The theoretical gain $77/112 \approx 0.69$ is to be expected for very long vectors only (because of start-up times). We do not obtain anything better than 0.86, because of contentions. Experiments on an IBM SP2 or on a Network of Workstations would most likely lead to more favorable ratios.

Back to Example 4

The third experiment corresponds to Example 4, with $P = 12$, $Q = 8$, $r = 4$, and $s = 3$. This experiment is similar to the first one in that our redistribution schedule requires much fewer steps (4) than does the caterpillar (12). There are two differences, however: $P \neq Q$, and our algorithm is not guaranteed to be optimal in terms of total cost. Instead of obtaining the theoretical ratio of $4/12 \approx 0.33$, we obtain results close to 0.6. To explain this, we need to take a closer look at the caterpillar algorithm. As shown in Table 19, 6 of the 12 steps of the caterpillar algorithm are indeed empty steps, and the theoretical ratio rather is $4/6 \approx 0.66$.

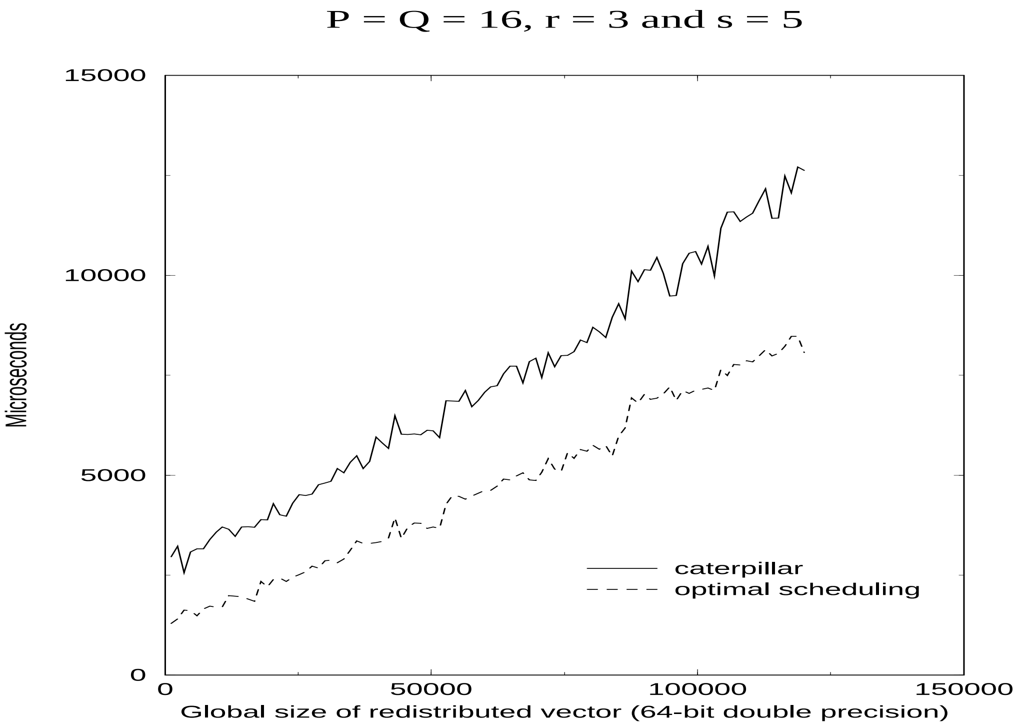

Figure 2: Comparing redistribution times on the Intel Paragon for $P = Q = 16$, $r = 3$ and $s = 5$.

Table 19: Communication costs for $P = 12$, $Q = 8$, $r = 4$, and $s = 3$ with the caterpillar schedule.

Caterpillar for $\boldsymbol{P = 12}$, $\boldsymbol{Q = 8}$, $\boldsymbol{r = 4}$, and $\boldsymbol{s = 3}$

Step	a	b	c	d	e	f	g	h	i	j	k	l	**Total**
Cost	3	0	0	0	3	3	3	0	0	0	3	3	**18**

6 Conclusion

In this article, we have extended Walker and Otto's work in order to solve the **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. For **any** values of the redistribution parameters P, Q, r, and s, we have constructed a schedule whose number of steps is optimal. Such a schedule has been shown optimal in terms of total cost for some particular instances of the redistribution problem (that include Walker and Otto's work). Future work will be devoted to finding a schedule that is optimal in terms of both the number of steps and the total cost for arbitrary values of the redistribution problem. Since this problem seems very difficult (it may prove NP-complete), another perspective is to further explore the use of heuristics like the greedy algorithm that we have introduced, and to assess their performances.

We have run a few experiments, and these generated optimistic results. One of the next releases of the ScaLAPACK library may well include the redistribution algorithm presented in this article.

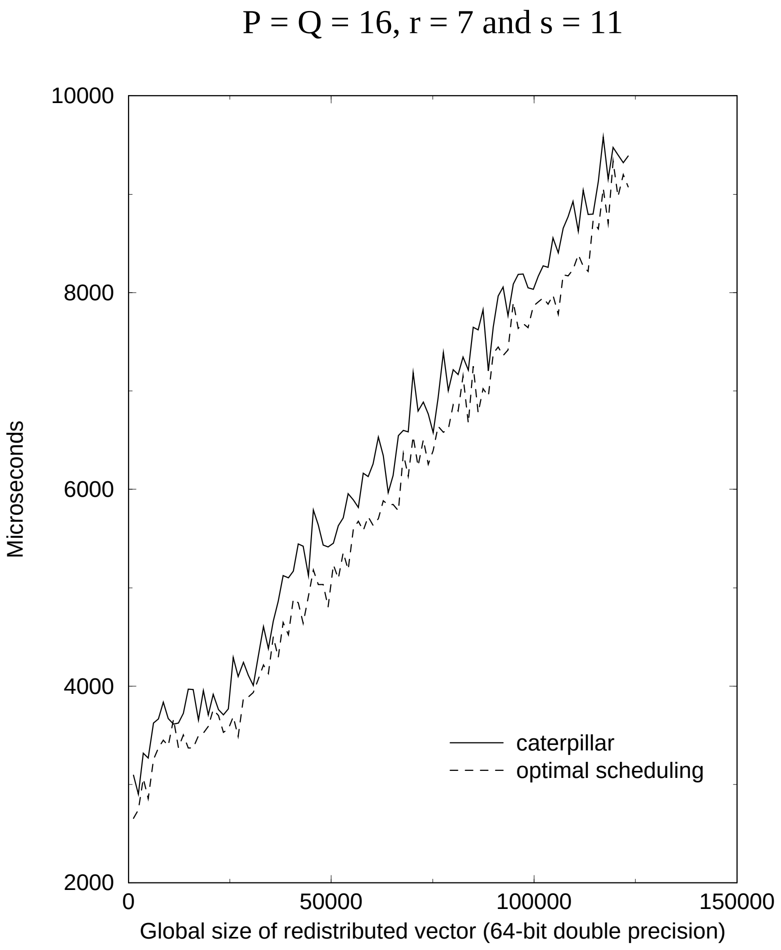

Figure 3: Time measurement for caterpillar and greedy schedule for different vector sizes, redistributed from $P = 16$, $r = 7$ to $Q = 16$, $s = 11$.

References

[1] C. Ancourt, F. Coelho, F. Irigoin, and R. Keryell. A linear algebra framework for static HPF code distribution. *Scientific programming*, to appear. Avalaible as CRI–Ecole des Mines Technical Report A-278-CRI, and at `http://www.cri.ensmp.fr`.

[2] Claude Berge. *Graphes et hypergraphes.* Dunod, 1970.

[3] S. Chatterjee, J. R. Gilbert, F. J. E. Long, R. Schreiber, and S.-H. Teng. Generating local addresses and communication sets for data-parallel programs. *Journal of Parallel and Distributed Computing*, 26(1):72–84, 1995.

[4] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. ScaLAPACK: A portable linear algebra library for distributed memory computers - design issues and performance. *Computer Physics Communications*, 97:1–15, 1996. (also LAPACK Working Note #95).

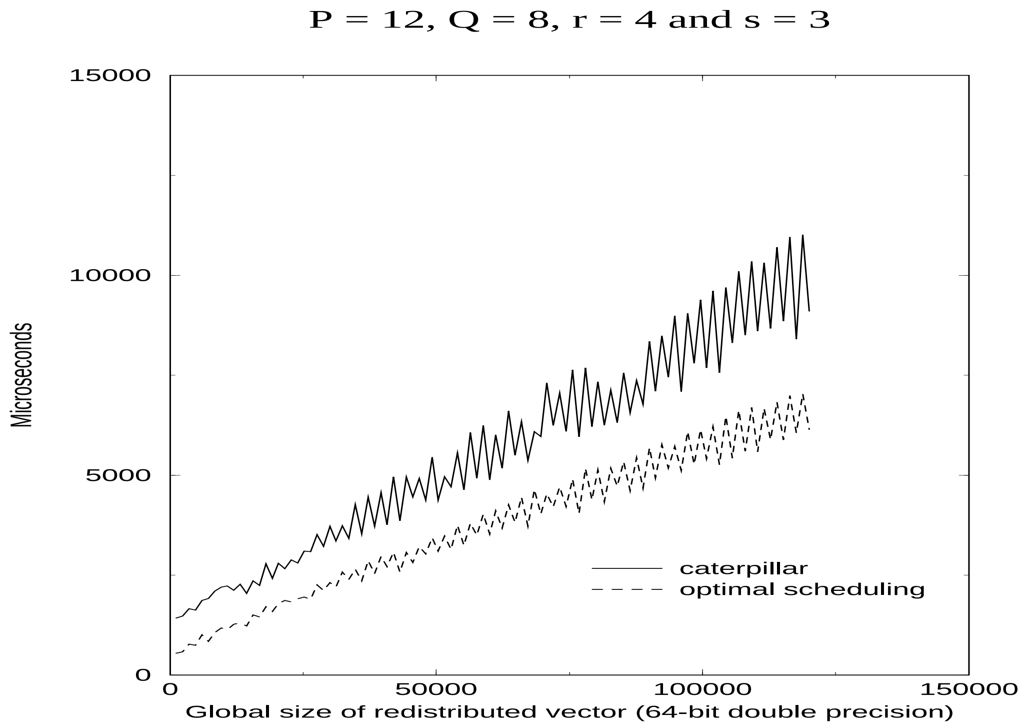

Figure 4: Time measurement for caterpillar and greedy schedule for different vector sizes, redistributed from $P = 15$, $r = 4$ to $Q = 6$, $s = 3$.

[5] J. J. Dongarra and D. W. Walker. Software libraries for linear algebra computations on high performance computers. *SIAM Review*, 37(2):151–180, 1995.

[6] Gene H. Golub and Charles F. Van Loan. *Matrix computations.* Johns Hopkins, 2 edition, 1989.

[7] R.L. Graham, M. Grötschel, and L. Lovász. *Handbook of combinatorics.* Elsevier, 1995.

[8] S. K. S. Gupta, S. D. Kaushik, C.-H. Huang, and P. Sadayappan. Compiling array expressions for efficient execution on distributed-memory machines. *Journal of Parallel and Distributed Computing*, 32(2):155–172, 1996.

[9] J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM J. Computing*, 2(4):225–231, 1973.

[10] E. T. Kalns and L. M. Ni. Processor mapping techniques towards efficient data redistribution. *IEEE Trans. Parallel Distributed Systems*, 6(12):1234–1247, 1995.

[11] K. Kennedy, N. Nedeljkovic, and A. Sethi. Efficient address generation for block-cyclic distributions. In *1995 ACM/IEEE Supercomputing Conference.* `http://www.supercomp.org/sc95/proceedings`, 1995.

[12] K. Kennedy, N. Nedeljkovic, and A. Sethi. A linear-time algorithm for computing the memory access sequence in data-parallel programs. In *Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 102–111. ACM Press, 1995.

[13] Charles H. Koelbel, David B. Loveman, Robert S. Schreiber, Guy L. Steele Jr., and Mary E. Zosel. *The High Performance Fortran Handbook.* The MIT Press, 1994.

[14] Antoine Petitet. *Algorithmic redistribution methods for block cyclic decompositions.* PhD thesis, University of Tennessee at Knoxville, December 1996.

[15] L. Prylli and B. Tourancheau. Efficient block-cyclic data redistribution. In *EuroPar'96*, volume 1123 of *Lectures Notes in Computer Science*, pages 155–164. Springer Verlag, 1996.

[16] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI the complete reference.* The MIT Press, 1996.

[17] J. M. Stichnoth, D. O'Hallaron, and T. R. Gross. Generating communication for array statements: design, implementation, and evaluation. *Journal of Parallel and Distributed Computing*, 21(1):150–159, 1994.

[18] A. Thirumalai and J. Ramanujam. Fast address sequence generation for data-parallel programs using integer lattices. In C.-H. Huang, P. Sadayappan, U. Banerjee, D. Gelernter, A. Nicolau, and D. Padua, editors, *Languages and Compilers for Parallel Computing*, volume 1033 of *Lectures Notes in Computer Science*, pages 191–208. Springer Verlag, 1995.

[19] K. van Reeuwijk, W. Denissen, H. J. Sips, and E. M.R.M. Paalvast. An implementation framework for HPF distributed arrays on message-passing parallel computer systems. *IEEE Trans. Parallel Distributed Systems*, 7(9):897–914, 1996.

[20] A. Wakatani and M. Wolfe. Redistribution of block-cyclic data distributions using MPI. *Parallel Computing*, 21(9):1485–1490, 1995.

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

[22] 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.