International Journal of Pure and Applied Mathematics

Volume 2 No. 3 2002, 335-355

MATHEMATICAL MODELLING OF RUNTIME REDISTRIBUTION OF FIXED EXCHANGED MESSAGES OVER A MULTIPROCESSOR GRID

Stavros Souravlas[1], Manos Roumeliotis[2] §

University of Macedonia
Applied Informatics Dept.
156 Egnatia Str., P.O. Box 1591
540 06 Thessaloniki, GREECE
[1]e-mail: sourstav@uom.gr
[2]e-mail:manos@uom.gr

Abstract: This paper presents a mathematical model used to solve an instance of the problem of redistributing data over a parallel processor grid during run-time. This particular instance is moving from a *cyclic(r)* redistribution on a P-processor grid to *cyclic(s)* on a P-processor grid, where s is a multiple of r. In this paper, we introduce mathematical definitions and propositions concerning important factors of the data redistribution problem between parallel processors. These are the total communication cost, the communications pattern and the communication scheduling, that is, the messages to be exchanged between processors. The total cost factor is critical when redistributing data, since the redistribution is performed during run-time. The communication pattern is directly induced by the redistribution parameters. When scheduling the communication processor pairs, it is important to know which processors can communicate. The communication scheduling organizes the redistribution in communicating steps, such that each processor sends and receives one message

Received: April 23, 2002

§Correspondence author

at each time. Scheduling is based on a number of matrix transformations, which correspond to local memory operations of the processors involved.

AMS Subject Classification: 68M07
Key Words: redistribution, block-cyclic redistribution, communication scheduling, cost analysis, communication pattern

1. Introduction

Many complicated parallel computing applications are composed of several stages. As the program proceeds from one stage to another, it may require different distribution of data between several processor sets. Examples of such applications include the alternate direction implicit method [5] and the multidimensional Fast Fourier Transform [11, 14, 15]. Data redistribution is, therefore, required each time the distribution of data to a processor is improper to complete a certain execution phase [20]. Since the redistribution must be completed during runtime, an efficient redistribution algorithm in terms of both time and startup costs must be adopted.

The interest in runtime redistribution was initially motivated by High Performance Fortran (HPF), parallel programming language [1, 10]. Programs developed in HPF use the ALIGN, DISTRIBUTE and REDISTRIBUTE directives to specify data redistributions [9]. The array redistribution in High Performance Fortran has three types, *Block*, *Cyclic*, and *Block Cyclic* [6, 17]. Thakur et al [17] have developed algorithms for these redistribution types, that can be implemented in the library of HPF compiler. The block-cyclic array redistribution is the most regular among the three types of redistribution [2]. The block cyclic array redistribution with block size x is referred to as *cyclic(x)*. Each block contains x successive elements.

An instance of the block cyclic redistribution problem is to redistribute data form *cyclic(r)* on a P-processor grid to *cyclic(s)* on the same processor grid. Figure 1 shows an example of moving 12 elements from *cyclic(2)* to *cyclic(3)* on a grid of three processors.

In this paper, we present a simple mathematical model of this instance of redistribution. The model facilitates the implementation of an efficient algorithm that has the following features:

• It maximizes the utility of each processor since it minimizes the time the processors remain idle.

• It eliminates contention on communication ports.

• It minimizes the total cost of redistribution.

• It minimizes the number of communication steps required to perform redistribution, therefore reduces *software overheads* (procedure call overheads, indexing calculations, error checking) [16].

• It can be applied to all communication patterns, that is, *all-to-all communication* or *non-all-to-all communication*

Element	1	2	3	4	5	6	7	8	9	10	11	12
Block-Cyclic (2)	P0	P0	P1	P1	P2	P2	P0	P0	P1	P1	P2	P2
Block-Cyclic (2)	P0	P0	P0	P1	P1	P1	P2	P2	P2	P0	P0	P0

Figure 1: Moving from cyclic(2) to cyclic(3) on a 3-procesor grid

The rest of the paper is organized as follows: In Section 2, a brief description of related work appears. In Section 3, we present the preliminaries about block cyclic redistribution. In Section 4, based on the preliminaries of Section 3, we analyze the mathematical model we used to describe the redistribution problem. The model is divided into three parts:

a. the *cost analysis*,

b. the *communication pattern* and

c. the *communication scheduling*.

Finally, the conclusions and further work are presented in Section 5.

2. Related Work

A lot of effort has been focused on the problem of runtime redistribution between several processors. The techniques for implementing the redistribution can be divided into two categories: In the first category, the attention is mainly focused on the problem of building the messages to be exchanged between processor sets with little or no attention paid to the problem of reducing communication overheads [2, 4, 18]. In the second category, attention has also been focused on the problem of *communication scheduling*, that is, to organize the messages in such a way that the number of communication steps is reduced, in order to reduce start-up costs [3, 8, 11, 19].

In [4], Kalns et al provided a processor mapping technique for dynamic data redistribution. This technique minimized the amount of data to be moved among processor memories, by mapping logical processor IDs to data elements that are redistributed, without violating the redistribution patterns.

Chung et al [2] provided a basic-cycle calculation technique for dynamic data redistribution. Their method was based on the idea of computing the source and destination processors of the elements in the basic cycle. The number of those elements equals *lcm(r,s)/gcd(r,s).* Array elements that are in the same local position but belong to different redistribution cycles have the same source and destination processor.

Thakur et al [18] provided algorithms for runtime redistribution. Their work is divided into two cases: the general case of *Cyclic(x)* to *Cyclic(y)* redistribution, where there is no relation between x and y and a special case where x is a multiple of y or y is a multiple of x. For the special case, they developed the KY-TO-Y algorithm. Each processor p calculates the destination processor p_d of it's first element as p_d= *mod(kp, P).* Then, the first y elements are sent to p_d, the next y to $p_d + 1$ and so on, until the end of the first block. Then, the other blocks are moved in the same pattern. For the general case, they implemented the GCD (greatest common divisor) and the LCM (least common multiplier) algorithms. The main idea is to redistribute from *cyclic(x)* to *cyclic(m)*, where m is the LCM or GCD of x, y using the KY-TO-Y algorithm which solves the special case. Kennedy et al [7] provided algorithms for calculating the local memory access sequence involving references to arrays redistributed in a *cyclic(k)* pattern.

The problem of *message scheduling* was successfully dealt with, in a very interesting paper by Desprez et al [3]. Their effort was focused on solving the general redistribution problem, moving from *cyclic(r)* on a P-processor grid, to *cyclic(s)* on a Q-processor grid. The main idea behind their algorithm was to create homogeneous communication patterns which they called *classes.* Processor pairs in a certain class, exchanged messages of the same size. Having created the classes, they arranged the message scheduling by mixing elements that belonged to different classes in such a manner that the elements of the most costly classes are distributed in the fewer steps possible. They also introduced the *stepwise* and *greedy* strategy for the redistribution of array elements. The *greedy* strategy aimed at using intermediate processors between a processor pair that is to communicate, to reduce the total communication cost. Therefore, it required more steps than the *stepwise* strategy that aimed at minimizing the communication steps.

Park et al [11] also provided algorithms for minimizing the communication steps and eliminate node contentions in a special redistribution case, moving from *cyclic(x)* on P processors to *cyclic(Kx)* on Q processors. Their methods applied to *non-all-to-all* and *all-to-all* communication patterns. Walker and Otto [19] dealt with the same redistribution problem as in [11], but the number of processors-senders and processors-receivers was equal. They provided *synchronized* and *unsynchronized* schemes that were free of conflicts. In the *synchronized* scheme however, performance was reduced by the fact that some processes had to wait for others before receiving data, while the main problem of the *unsynchronized* algorithm was the necessity for buffering space equal to the data redistributed. Furthermore, the number of steps required for the implementation of those schemes was not minimal.

Lim et al [8] provided conflict-free direct, indirect and hybrid algorithms, for moving from *cyclic(r)* to *cyclic(Kr)* on a P-processor grid. The direct algorithms sent the message directly from the source to the destination processor. In the indirect algorithm, messages having the same destination were gathered in an intermediate relay processor that started an exclusive session with the destination processor. The hybrid algorithm was a combination of the direct and indirect approach, where the first d steps performed indirectly and the remaining directly.

3. Block Cyclic Redistribution Preliminaries

Data redistribution in a parallel processor system is the mapping of data in the processes of that system. Each processor executes several processes, thus we can consider a system of parallel processors as a set of executing processes rather than a set of processors. If we consider a network of $P \times Q$ processes, then Γ can be defined as the set of coordinates for each couple of processes (p, q) [12, 13].

$$\Gamma = (p, q) \,\epsilon\, (0..P-1) \times \ (0..Q-1) \tag{1}$$

Thus, the process grid can be represented by a two-dimensional table with rows being the senders p, and columns being the receivers q. Data may be represented as a two-dimensional table $M \times N$. Since the data redistribution must be accomplished in a round robin fashion, to maintain load balance between processors, the total amount of information must be divided into pieces that will be distributed. These pieces are called *blocks.* Figure 2 shows an array of 15×20 elements partitioned into blocks of size 4×3. If each block contains r rows and s columns, then data will be divided into an $M_b \times N_b$ array of blocks where:

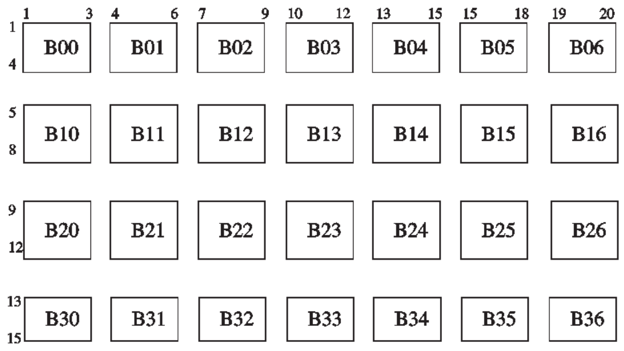


Figure 2: A 15×20 elements divided into blocks of size 4×3 ($M = 15$, $N = 20$, $r = 4$, $s = 3$)

$$M_b = \frac{M}{r},\ N_b = \frac{N}{s},\ M_b = \frac{M}{r} + 1,\ N_b = \frac{N}{s} + 1\,. \tag{2}$$

The second set of relationships for M_b, N_b holds if M does not divide r or N does not divide s. In the example of Figure 2, $M_b = \frac{15}{4} + 1 = 4$, $N_b = \frac{20}{3} + 1 = 7$.

We may use a set Λ, to relate data blocks to a process, or coordinates of blocks to coordinates of processes. If l relates block lines to rows of the process grid *(p)* and m relates block columns to the columns of the process grid *(q)*, then l, m may be found in the interval $[0..\frac{M_b-1}{P}]$, $[0..\frac{N_b-1}{Q}]$, respectively. Thus, set Λ can be defined as:

$$\Lambda = (l,m)\,\epsilon\,[0..\frac{M_b-1}{P}) \times\ [0..\frac{N_b-1}{Q}) \qquad \text{or}$$

$$\Lambda = (l,m)\,\epsilon\,[0..\frac{M_b}{P}) \times\ [0..\frac{N_b}{Q}) \tag{3}$$

The second relationship for l and m stands if M_b divides P or N_b divides Q. Figure 3 shows how the blocks of Figure 2 can be distributed over a 2x3 process grid in a round robin fashion. The corresponding values of l and m appear in parentheses. Indeed, l ranges from $[0..\frac{4}{2})$=[0..2), while m ranges from $[0..\frac{7-1}{2}) = [0..3)$ Suppose that we consider the movement of a block i during the redistribution. It will be the l^{th} block in the local memory of process p. Thus, $l = \frac{i}{P}$, $p = i\,mod\,P$. From these 2 relationships we derive that:

$$i = lP + p\,. \tag{4}$$

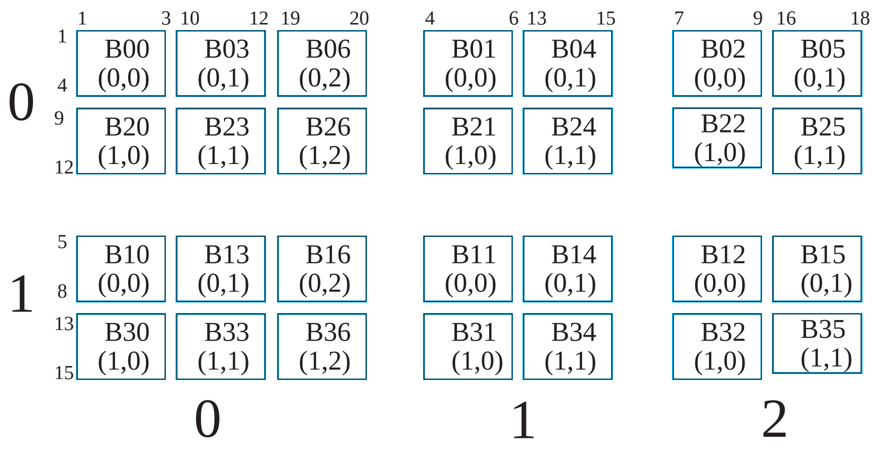


Fig 3: Distributing a block partitioned array over a 2x3 processor grid

In the same way, we derive that a block indexed j will be the m^{th} in the local memory of a processor q. Thus:

$$j = mQ + q\,. \tag{5}$$

Each block contains r rows and s columns. Set Θ is used to define the position of the elements locally, inside the block

$$\Theta = (x, y)\,\epsilon\,[0..r-1)\;\times\;[0..s-1)\,. \tag{6}$$

Thus, the local position p_i, q_i of each element redistributed to p and q will respectively be:

$$p_i = (lP + p)r + x, \qquad x\,\epsilon\,[0..r-1]\,, \tag{7}$$

$$q_i = (mQ + q)s + y, \qquad y\epsilon\,[0..s-1]\,. \tag{8}$$

Since it can be proven [12] that the redistribution in the way defined is a periodical procedure, with it's periodicity being equal to the least common multiplier of the quantities Pr and Qs, that is, L=LCM(Pr,Qs) elements, Λ can be redefined as:

$$\Lambda = (l, m) \,\epsilon\, [0..\frac{L}{Pr}) \;\times\; [0..\frac{L}{Qs}) \,. \tag{9}$$

Relationship (9) is obtained after replacing M_b and N_b in equation (3) with their equals $\frac{M}{r}$ $\frac{N}{s}$, respectively.

4. The Mathematical Model

In this section, we will deal separately with the three most important factors of redistribution : *a. Cost Analysis, b. Communication Patterns , c. Communication Scheduling.*

4.1. The Cost Analysis

The modelling of redistribution cost is very important for easily checking the performance and efficiency of our algorithm. The cost model is derived directly from the redistribution preliminaries of the previous paragraph. If we consider the movement of a block indexed i during a redistribution stage, from a known number of processors P to a known number of processors Q, then according to equation (7) and equation (8) this movement can be described by the following equation:

$$(lP + p)r + x = (mQ + q)s + y \,. \tag{10}$$

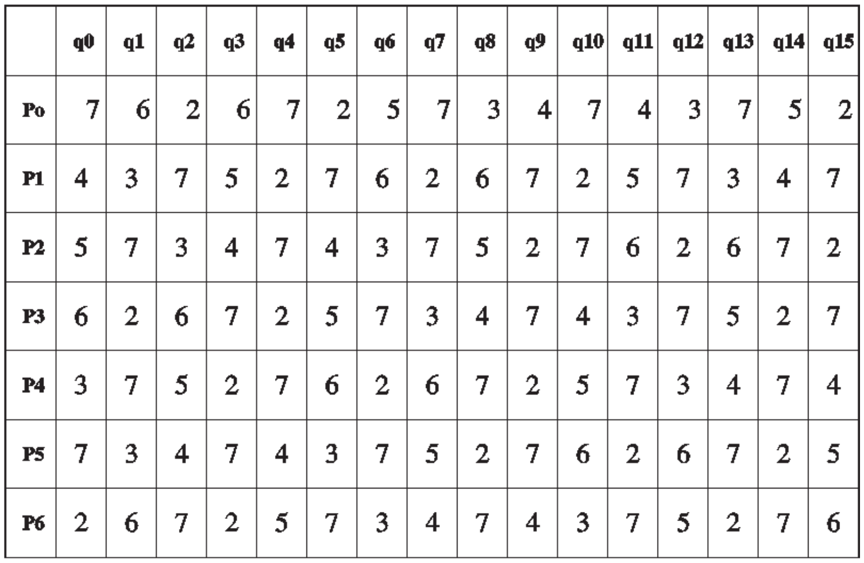
This equation is subject to the following restrictions:

$$p\epsilon[0..P), \quad q\epsilon[0..Q) \,, \tag{11}$$

$$x\epsilon[0..r), \quad y\epsilon[0..s) \,, \tag{12}$$

$$l\epsilon[0..\frac{L}{Pr}), \quad m\epsilon[0..\frac{L}{Qs}) \,. \tag{13}$$

The redistribution problem turns out to be a problem of defining, not only the block in which each element will move to, but also it's precise position in the block or in the local memory of the processor. If we consider the total



	q0	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14	q15
Po	7	6	2	6	7	2	5	7	3	4	7	4	3	7	5	2
P1	4	3	7	5	2	7	6	2	6	7	2	5	7	3	4	7
P2	5	7	3	4	7	4	3	7	5	2	7	6	2	6	7	2
P3	6	2	6	7	2	5	7	3	4	7	4	3	7	5	2	7
P4	3	7	5	2	7	6	2	6	7	2	5	7	3	4	7	4
P5	7	3	4	7	4	3	7	5	2	7	6	2	6	7	2	5
P6	2	6	7	2	5	7	3	4	7	4	3	7	5	2	7	6

Figure 3: Communication costs for $P = 16$, $r = 7$ and $s = 11$

redistribution cost as the information about the block each element will move to, and the set of all possible local memory positions it may be located, then that cost may be defined as the number of solutions [3] for equation (10), given the number of the process coordinates *(p,q)* and the total number of senders P and receivers Q, and with unknowns *l, m, y, x* which define the coordinates of the block number and the local memory position. The redistribution equation, can be solved using Euclid's theorem for solving linear Diophantine equations. It must be mentioned, that if for a given pair of sender-receiver *p,q*, there is no quadruple *(l, m, y, x)* to satisfy equation (10), then there is no communication between p and q. Figure 4 shows the redistribution cost of communication between senders from P_0 to P_6 and all possible receivers, in a redistribution from *cyclic(7)* to *cyclic(11)* on a 16-processor grid.

The algorithm derived by the mathematical model, treats redistribution as a series of communication steps. At each step, a processor can only send/receive one message of a particular *length*. This *length* is determined by the number of solutions of the redistribution equation, as described above. The total cost of a certain step, is the maximum time it takes for a pair of communicating processors to complete the message distribution. If we name C_s the total cost per step, then C_s will be [3]:

$$C_s = \max[length(p_i,\ q_i)] + a, \tag{14}$$

where a is the startup cost which includes procedure call overheads, indexing calculations and error controls. If the number of steps required to complete the redistribution is N then the total redistribution cost is:

$$T_C = NC_s \tag{15}$$

However, in cases where s is a multiple of r, the total redistribution cost depends *only* on the number of the communication steps, since the exchanged messages are fixed in terms of cost and more specifically, their length is one time unit as Proposition 1 states.

Proposition 1. *Message lengths in a redistribution R, from P senders to P receivers, with a block size $r \times s$ such that s mod r =0 are always equal between each other and their cost is one time unit.*

Proof. Consider a redistribution R with the parameters given in Proposition 2. Redistribution is a periodical procedure, and it's periodicity equals to L, $L = LCM(Pr, Ps)$. If $s \, mod \, r = 0$, then $s = kr$, where k is an integer value. Therefore $L = LCM(Pr, Pkr)$. Since the least common multiplier of Pr and Pkr equals Pkr, we derive that $L = Ps$. Suppose that $r = 1$. Then, a new redistribution R' with periodicity L' can be defined:

$$R'(P, Ps),\ L' = LCM(P, Ps) = Ps\,. \tag{16}$$

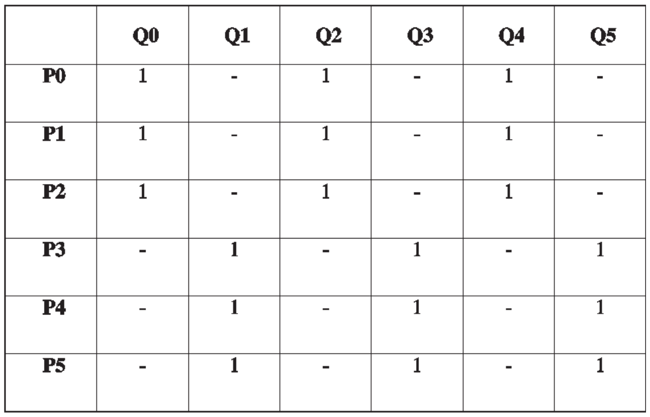
Thus, R' is similar to R and can replace it. However, for redistribution R', we have $r = 1$. If we replace this in the redistribution equation (10), we will get: $(lP + p) + x = (mQ + q)s + y$. Due to the restrictions (equation (12)), the redistribution equation is subject to, $x = 0$. Therefore, the equation will turn to:

$$lP - msQ = qs - p + y\,. \tag{17}$$

Name g the greatest common divisor of Pr and Qs. Suppose that we calculate a solution $(l*, m*)$ for the equation $mQs - lP = g$. According to Euclid's theorem, if the redistribution equation (10) has a solution, then the quantity $g - (pr - qs)$ must be in the interval [1-s..r-1] or [1-s..0] since $r = 0$. If this holds, the solution $(l*, m*, x, y)$ is unique since x is always zero. Therefore, the number of solutions for the redistribution equation is either one or zero. In case it is zero, there is no communication between p and q. Otherwise there is, and it's cost is always equal to one time unit.

According to Proposition 1 and equation (15), the total redistribution cost when s is a multiple of r is:

$$T_C = N \tag{18}$$



	Q0	Q1	Q2	Q3	Q4	Q5
P0	1	-	1	-	1	-
P1	1	-	1	-	1	-
P2	1	-	1	-	1	-
P3	-	1	-	1	-	1
P4	-	1	-	1	-	1
P5	-	1	-	1	-	1

Figure 4: Communication grid for $P = 6$, $r = 1$ and $s = 3$

4.2. Communication Pattern

As Figure 4 indicates, there can be redistributions where all senders can communicate with all receivers (and this is also true for senders P_7 to P_{15} which are not depicted in Figure 4). However, there may be redistribution problems, where there are processor pairs that do not communicate. Proposition 2 offers a criterion for the existence or not of *all-to-all communication* between processor pairs.

Proposition 2. *Consider a redistribution problem from cyclic(r) to cyclic(s) over a P-processor grid. The communication pattern is an all-to-all communication if and only if the value of the greater common divisor of Pr and Qs, name it g, is such that:* $g \leq (r + s - 1)$.

Proof. Equation (10) can be rewritten as:

$$lPr - mQs = qs - pr + (y - x)\,. \tag{19}$$

We can rewrite $lPr - mQs$ as λg, since $lPr - mQs$ is a multiple of g. If we also rewrite $z = y - x$, then equation (19) is rewritten as:

$$\lambda g + z = \; qs - pr\,. \tag{20}$$

According to the restriction described in equation (12), z lies in the interval [1-r..s-1] the integer length of which, is *s+r-1.* equation (20) has a solution for unknowns *x, y, l, m* if there is an integer multiple of g in the interval [1-r..s-1] and this holds only if $g \leq (r + s - 1)$. If we assume that $g > (r + s - 1)$

then we must find a processor pair that will not exchange messages during redistribution. Indeed, if we set *p=P-1* and *q=0* then equation (20) is rewritten as: $Pr - r = z + \lambda g$. If this equation has a solution, there must be a multiple of g added to *Pr-r* and produce a value in the interval [1-r..s-1]. However, $(Pr - r)\ mod\ g = g - r$ since g divides *Pr*. Therefore, if $\frac{Pr-r}{g} = a$, then $(Pr-r)-ga = g-r$ which is out of the interval [1-r..s-1] because $g > r+s-1$, therefore the minimum integer value of $g - r$ is $r + s - r = s$. If we add the next multiple of g to *Pr-r*, the minimum value we will get will be g-r+r+s=s+g (since $g > r+s-1$). Obviously s is out of the interval [1-r..s-1]. Therefore, there is no quadruple *(x, y, l, m)* to satisfy the redistribution equation (10) for p=P-1, q=0. This means that the processor pair *(p,q)=(P-1,0)* will not exchange any message during redistribution and the communication pattern is *non-all-to-all communication* (for other proof see also [3], [11]).

Consider a redistribution from *r=2* to *s=6* over a 6-processor grid. According to Proposition 1, we can go to a redistribution from *r=1* to *s=3*, instead. Figure 5 shows the communication grid derived by the solution of equation (10). It is obvious that the communication pattern induced by the redistribution parameters is a *non-all-to-all communication.* The 1's indicate the existence of communication between the corresponding processors (p, q), the cost of which is one time unit.

4.3. Communication Scheduling

Communication scheduling is the act of arranging the communication steps of redistribution. Steps are arranged in such a way, that at each step, a processor cannot send/receive more than one message to/from another processor. To achieve this, we first have to define two tables for our communication grid. The block indices table T_{bit} that contains the indices of the blocks that reside in the memory of the local processor p_i and the destination processor table T_{dpt} that contains the indices of the processors, where the blocks will move to, after redistribution. Figure 6 shows the two tables, for a redistribution for $P = 6$, $r = 1$, $s = 4$. Our purpose is to make a series of column transformations in T_{dpt} to end up in a matrix, where all elements of each row are different. This means, that each processor sends/receives only one message at a time, which is our initial goal. The column transformations correspond to operations in the local memory of each processor and they are not considered as part of the cost, for simplicity. We will prove that there is indeed a series of matrix transformations which guarantee that we will end in a free of conflicts communication scheme.

As Figure 6a indicates, the entries of T_{bit} and T_{dpt} are satisfying the equa-

P0	**P1**	**P2**	**P3**	**P4**	**P5**
0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23

(a)

P0	**P1**	**P2**	**P3**	**P4**	**P5**
0	0	0	0	1	1
1	1	2	2	2	2
3	3	3	3	4	4
4	4	5	5	5	5

(b)

Fig. 6. (a) Block indices for P=6, r=1, s=4. (b) Destination Processor indices for P=6, r=1, s=4.

tion:

$$T_{bit}(i,j) = iP + j,\ T_{dpt} = \frac{T_{bit}(i,j)}{s}\,. \tag{21}$$

If we set G as the greatest common divisor of s, P that is $G = gcd(s,P)$ then there must be integers s' and P' such that: $s = s'G$ and $P = P'G$. For the example of Figure 6. we have $G = 2$, $s' = 2$ and $P' = 3$. If we consider the rows that their indices differ by a multiple of s' then those line indices will produce the same modulo when divided by s'. Indeed, if we consider two rows of T_{bit} with indices i and i' then there is an integer μ such that $i' = i + \mu s$. We already have (see equation (21)) : $T_{bit}(i,j) = iP + j$, thus: $T_{bit}(i',j) = i'P + j = (i + \mu s')P + j$. Therefore, if we subtract the two lines we will have: $T_{bit}(i',j) - T_{bit}(i,j) = (i + \mu s')P + j - (iP + j) = \mu s'P = \mu s'P'G$. Since the quantity $\mu s'P'G$ divided by s' will produce a modulo 0, the elements of T_{bit} whose row indices differ by s' have the same modulo when divided by s'. In each column, there are G elements that their row indices produce the same modulo when divided by s'. These elements can appear in contiguous row positions, if row i is moved to row i' in such a way that:

$$i' = (i\ mod\ s')G + \frac{i}{s'}\,. \tag{22}$$

These transformations result in table T'_{bit} as shown in Figure 7 The transformed

P0	**P1**	**P2**	**P3**	**P4**	**P5**
0	1	2	3	4	5
12	13	14	15	16	17
6	7	8	9	10	11
18	19	20	21	22	23

Fig. 7. Block indices for P=6, r=1, s=4 after transformation (T'bit)

table T'_{bit} consists of $s'xP'$ sub-tables with dimension GxG. We declare those sub-tables as $D_{w,z}$, where $0 \leq w < s'$ and $0 \leq z < P'$. Figure 8 shows these sub-tables. An element of T'_{bit} will be in the sub-table defined by w, z in the position defined by w_1, z_1, where $0 \leq w_1, z1 < G$. Obviously, an element that belongs to row i of T'_{bit} will be in the w row of sub-tables, where $w = \frac{i}{G}$.

The specific row position of an element *inside* a sub-table will be given by w_1, where $w_1 = i \, mod \, G$. Suppose that we want to find the value of an element of T'_{bit}. From equation (22) we derive that $T'_{bit}((i \; mod \; s')G + \frac{i}{s'}, \; j) = T_{bit}(i, j)$. If we invert this relationship, we will have:

$$T'_{bit}(i, j) = T_{bit}((i \; mod \; G)s' + \tfrac{i}{G}, \; j)$$

$$= T_{bit}(w_1 s' + w, \; j)$$

$$= (w_1 s' + w)P + j \text{ (from equation (21))}$$

$$= w_1 s' P + wP + j$$

$$= w_1 s' P + wP + Gz + z_1 \text{ (because } j = Gz + z_1\text{)}$$

$$= w_1 s' P + wP'G + Gz + z_1 \text{ (because } P = P'G\text{)}$$

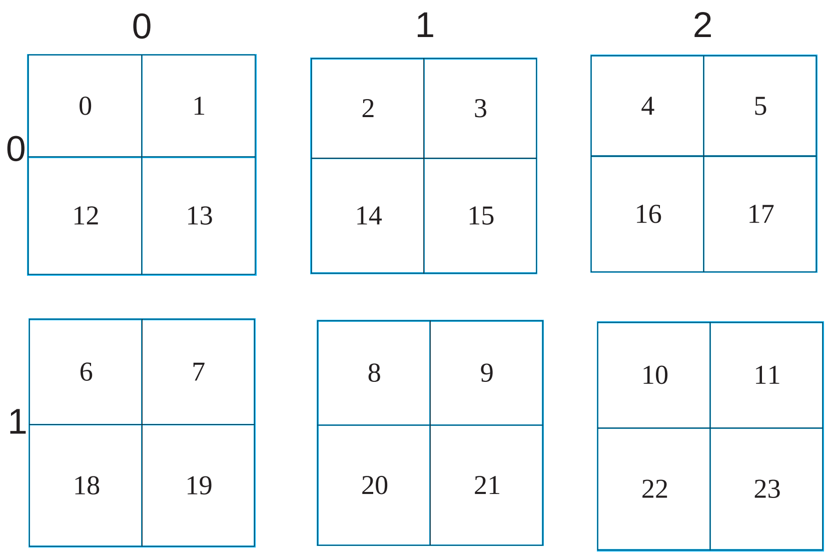


Fig. 8. s' x P' subtables created after transformation for P=6, r=1, s=4

$$=(wP'+z)G+(w_1s'P+z_1)$$

The value which we found for $T'_{bit}(i,j)$ is a sum of two parts, of which the leftmost is constant for all elements because it is independent of the local indices w_1, z_1 of the sub-tables. We can express T'_{bit} as a set of $s' \times P'$ sub-tables of size $G \times G$.

$$T'_{bit} = \left| \begin{array}{cccc} D_{0,0} & D_{0,1} & .... & D_{0,P'-1} \\ D_{1,0} & D_{1,1} & .... & D_{1,P'-1} \\ .... & .... & .... & .... \\ .... & .... & .... & .... \\ D_{s'-1,0} & D_{s'-1,1} & .... & D_{s'-1,P'-1} \end{array} \right|, \tag{23}$$

where each of the sub-tables will be:

$$D_{w,z} = (wP'+z)G + \left| \begin{array}{cccc} 0 & 1 & .... & G-1 \\ s'P & s'P+1 & .... & s'P+G-1 \\ .... & .... & .... & .... \\ .... & .... & .... & .... \\ (G-1)s'P & (G-1)s'P+1 & .... & (G-1)(s'P+1) \end{array} \right| . \tag{24}$$

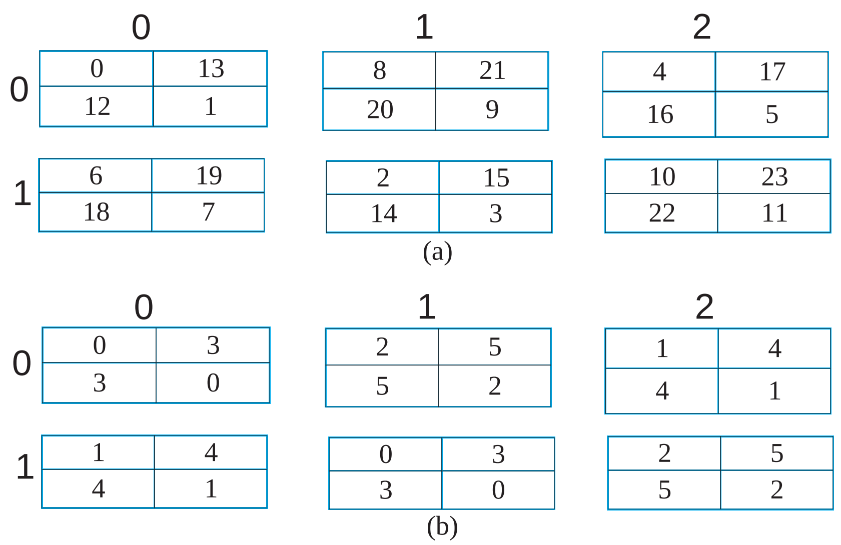


Fig. 9: (a) D' sub-tables after transformations, (b) D' subtables after division of each element by s=4

For example, consider $D_{1,1}$. According to equation (24), we have (see also Figure 8):

$$D_{1,1} = \begin{vmatrix} 8 & 8 \\ 8 & 8 \end{vmatrix} + \begin{vmatrix} 0 & 1 \\ 12 & 13 \end{vmatrix} = \begin{vmatrix} 8 & 9 \\ 20 & 21 \end{vmatrix} .$$

To obtain the final destination processor table containing a free of contention communication scheme, we have to move each element of T'_{bit} from the previous row of sub-tables w' to the new row of sub-tables w and from the previous row position *inside* the sub-table w'_1 to the new row position *inside* the sub-table, w_1. The column references z and z_1 will remain unchanged. Then each element of T'_{bit} has to be divided by s since $T_{dpt} = \frac{T_{bit}(i,j)}{s}$ (see equation (21)).

The movement of each element to it's new position will be completed in such a way that:

$$w = (w'P' + z) \; mod \; s', \; w_1 = (z_1 + w'_1) \; mod \; G \, . \tag{25}$$

Figure 9a shows the sub-tables created after the transformations while figure 9b shows the sub-tables after the division of each element by s.

After the transformations and the division of each element by s, and due to the relationships: $s = s'G \Longrightarrow \frac{G}{S} = \frac{1}{s'}$ and $P = P'G$, we will obtain a new set of sub-tables D', which will be (see equation (24)):

$$D'_{w,z} = \frac{(w'P'+z)G}{s'} + \left| \begin{array}{cccc} 0 & (G-1)P' & .... & P' \\ P' & 0 & .... & 2P' \\ .... & .... & .... & .... \\ .... & .... & .... & .... \\ (G-1)P' & (G-2)P' & .... & 0 \end{array} \right| . \qquad (26)$$

The final destination processor table T'_{dpt} will be composed of those $D'_{w,z}$ sub-tables. Therefore:

$$T'_{dpt} = \left| \begin{array}{cccc} D'_{0,0} & D'_{0,1} & .... & D'_{0,P'-1} \\ D'_{1,0} & D'_{1,1} & .... & D'_{1,P'-1} \\ .... & .... & .... & .... \\ .... & .... & .... & .... \\ D'_{s'-1,0} & D'_{s'-1,1} & .... & D'_{s'-1,P'-1} \end{array} \right| . \qquad (27)$$

From equation 26 we derive that each sub-table consists of two parts. The leftmost part, depends only on the values of w' and z and it is independent of local row and column indices w_1 and z_1. Furthermore, the P' sub-tables that constitute each sub-table row, contain *different* elements between them due to the first transformation of equation (25). Indeed, if we divide each element of the constant part (leftmost part) of equation (24) by s, there will be successive sub-tables in each row that their elements have the same integer result when divided by s'. These sub-tables are driven to different rows by the transformation $w = (w'P' + z) \ mod \ s'$. This is achieved by the presence of the column reference z in the transformation relationship. (Recall that z is different for successive sub-tables that belong to the same row.)

The rightmost part of each sub-table D' depends only on the values of w_1 and z_1 which means that all the sub-tables of D' are equal between them. From the above we conclude that the row elements of T'_{dpt} are *different between them and they lead to a free of contention scheme* (see Figure 10), Figure 10 indicates that the message exchange between processors is going to be implemented in a series of communication steps. In each step, each processor will send/receive only one message to/from any other processors. For example, in step 0, processors (0, 1, 2, 3, 4, 5) will send to (0, 3, 2, 5, 1, 4), respectively.

	P0	**P1**	**P2**	**P3**	**P4**	**P5**
step 0	0	3	2	5	1	4
step 1	3	0	5	2	4	1
step 2	1	4	0	3	2	5
step 3	4	1	3	0	5	2

Fig. 10. T'_{dpt} for P=6, r=1, s=4. Each row contains different elements

5. Conclusions

In this paper, we have presented a mathematical way of modelling the problem of redistributed data between processor sets during run-time. The mathematical model was applied in an instance of the redistribution problem, that is, going from a *Cyclic(r)* to *Cyclic(s)* on a P-processor grid, under the condition that s is a multiple of r.

The purpose of the model is to facilitate the implementation of an algorithm for run-time data redistribution. Three aspects of the redistribution problem were covered in the model:

1. *The total communication cost* which is an important factor for the evaluation of the algorithm. The model offers an easy way of computing the total cost,

2. *The communication patterns*, which is a crucial parameter when scheduling the redistribution. The knowledge of the existence of all-to-all or non-all-to-all communication between processors is obtained easily with a single comparison as Proposition 2 suggested,

3. *The communication scheduling* which is implemented by a series of array transformations proven to lead to a free of contention communication scheme.

Our further work includes the development of a model for solving the general redistribution problem independently of the redistribution parameters, or for transforming the general problem to one similar to the one described by the present model. Another important instance of the problem would be to model a redistribution case where the algorithm changes from static to dynamic and from dynamic to static. The problem in this case, however, is that in static parallel algorithms, the distribution parameters like communication cost between processor pairs, size of transferred messages and communication patterns are arranged a-priori, unlike the dynamic algorithms that calculate these parameters before each redistribution phase. This causes difficulties in effectively modelling such a situation.

References

[1] B. Chapman, P. Mehrotra, H. Moritsch, H. Zima, *Dynamic Data Distribution in Vienna Fortran*, *Proc. Supercomputing '93* (Nov. 1993), 284-293.

[2] Yeh-Ching Chung, Ching-Hsien Hsu, Sheng-Wen Bai, A basic-cycle calculation technique for efficient dynamic data redistribution, *IEEE Transactions on Parallel and Distributed Systems*, **9**, No. 4 (April 1998), 359-377.

[3] Frederic Desprez, Jack Dongarra, Antoine Petitet, Cyril Randriamaro, Yves Robert, Scheduling block-cyclic array redistribution, *IEEE Transactions on Parallel and Distributed Systems*, **9**, No. 2 (February 1998), 192-205.

[4] E.T. Kalns, Lionel M. Ni, Processor mapping techniques toward efficient data redistribution, *IEEE Transactions on Parallel and Distributed Systems*, **6**, No. 12 (Dec. 1995), 1234-1247.

[5] S.D. Kaushik, C.H. Huang, R.W. Johnson, P. Sadayappan An approach to communication-efficient data redistribution, In: *Proceedings of the 8th ACM International Conference on Supercomputing*, Manchester, England (July 1994).

[6] S.D. Kaushik, C.H. Huang, Sadayappan, J. Ramanujam, P. Sadayappan, Multi-phase redistribution: a communication-efficient approach to array redistribution, *Technical Report OSU-CISRC-9/94-52*, Ohio State University (1994).

[7] K. Kennedy, N. Nedeljkovic, A. Sethi, Efficient address generation for block-cyclic distributions, In: *Proc. 1995 ACM/IEEE Supercomputing Conf.*, http://www.supercomp.org/sc95/proceedings (July 1995).

[8] Y.W. Lim, P.B. Bhat, V.K. Prasanna, Efficient algorithms for block cyclic redistribution of arrays, *Algorithmica*, No 24 (1998), 298-330.

[9] Y.W. Lim, Neungsoo Park, V.K. Prasanna, Efficient algorithms for multidimensional block cyclic redistribution of arrays, In: *Proc. International conference on Parallel Processing* (Aug. 1997), 234-241.

[10] Erin M. Miller, *Beginner's Guide to HPF*, Joint Institute For Computational Science, http://www.-jics.cs.utk.edu/HPF/HPFguide.html (Aug. 1998)

[11] Neungsoo Park, V.K Prassana, Cauligi S. Raghavendra, Efficient algorithms for block-cyclic array redistribution between processor sets, *IEEE Transactions on Parallel and Distributed Systems*, **10**, No. 12 (December 1999), 1217-1240.

[12] Antoine Petitet, *Algorithmic Redistribution Methods for Block Cyclic Decompositions*, PhD Thesis, University of Tennessee, Knoxville (1996).

[13] Antoine Petitet, Jack Dongarra, Algorithmic redistribution methods for block cyclic decompositions, *IEEE Transactions on Parallel and Distributed Systems*, **10**, No. 12 (December 1999), 1201-1216.

[14] L. Prylli, B. Touranchean, Fast runtime block cyclic data redistribution on multiprocessors, *Parallel and Distributed Computing*, **45** (Aug. 1997), 63-72.

[15] S. Ramaswamy, P. Benerjee, Automatic generation of efficient array redistribution routines for distributed memory multicomputers, In: *Proc. Fifth Symp. Frontiers of Massively Parallel Computation* (Feb. 1995), 342-349.

[16] Peter Strazdins, Reducing software overheads in parallel linear algebra libraries, *Technical Report TR-CS-97-15*, Australian National University (July 1997).

[17] Rajeev Thakur, Alok Choudhary, Geoffrey Fox, *Runtime Array Redistribution in HPF Programs*, SHPCC 94, Northeast Parallel Architectures Center (1994).

[18] Rajeev Thakur, Alok Choudhary, J. Ramanujam, Efficient algorithms for array redistribution, *IEEE Transactions on Parallel and Distributed Systems*, **7**, No. 6 (June 1996), 587-594.

[19] D.W Walker, S.W. Otto, Redistribution of block-cyclic data distributions using MPI, *Concurrency: Practice and Experience*, **8**, No. 9 (1996), 707-728.

[20] L. Wang, J.M. Stichnoth, S. Chatterjee, Runtime performance of parallel array assignment: An empirical study, In: *Proc. 1996 ACM/IEEE Supercomputing Conf.*, http://www.supercomp.org/sc96/proceedings (1996).