HEWLETT PACKARD
PISA SCIENCE CENTER
HEWLETT-PACKARD LABORATORIES

HPL–PSC–93–56

September 22, 1993

Compiling Data-Parallel Paradigms through Basic Communication Mechanisms

Salvatore Orlando Marco Vanneschi

Dipartimento di Informatica,
Università di Pisa,
Corso Italia, 40 – 56125 Pisa - Italy,
e-mail: {orlando,vannesch}@di.unipi.it

This paper presents a compiling technique to generate parallel code with explicit local communications for a mesh-connected distributed memory, MIMD architecture. Our compiling technique works for the *geometric* paradigm of parallel computation, i.e. a *data-parallel* paradigm where array data structures are partitioned and assigned to a set of processing nodes, which, to perform their identical tasks, need to exchange some of the data allocated to them. This means that some data dependencies exist between computations mapped onto different nodes. Our technique solves them statically, by inserting explicit communications in the parallel code running on each node. We also demonstrate that the technique produces an optimal code as far as the exploitation of the communication media bandwidth is concerned.

Keywords. Compilers, Parallel Processing, Data-parallel Paradigms, Methodologies for Parallel Programming, MIMD.

Contents

1 Introduction

Distributed memory, MIMD systems (DM-MIMD), with regular direct networks and low dimensional degree nodes [5,1], have been recognized as the parallel architectures that are able to efficiently exploit large-scale parallelism. No general agreement has been reached, however, about the corresponding programming models, nor about the kind of mechanisms to be exported by these architectures to run-time support (RTS) and operating system layers (OS).

The power of the communication mechanisms supplied, and the correct balance between communication and computing bandwidth are two of the main features that a parallel system should have. In any case, these architectural features alone are not enough to achieve high performance. The programming model, the hierarchy of abstract machines, and the compiling tools are all crucial to effectively exploit all the power of a system, so that no bottlenecks occur at any level. As an example of the problems that may arise, let us consider commercial DM-MIMDs equipped with *routing units*. These units efficiently implement *global* communication mechanisms, i.e. communications which involve any pairs of nodes in the system. Global forms of communication make both the development of language RTS and the OS functionalities easier, but, depending on the specific programming model, the performance of these forms of communications is not always predictable, and large degradations in the communication bandwidth may be observed. Collisions (also called conflicts) on communication links are the main source of this phenomenon.

Some researchers believe that using more restricted forms of communication, and exploiting suitable programming models and/or compiling techniques may considerably improve performance. In [8], for instance, structured communication primitives are introduced, and in [12], these structured primitives are employed by novel compiling techniques.

In this article, we restrict the forms of communications allowed even more, i.e., only *local* communications between directly connected nodes (or *basic*) communications, since more complex forms can be obtained as combination of them). We show that, for some *data-parallel paradigms*, local forms of communication, along with a suitable *compiling techniques*, make it possible to efficiently exploit the features of a DM-MIMD machine. More specifically, we present a compiling technique that analyzes some information about the dependencies between the data-parallel tasks, and translates these dependencies in terms of basic communication mechanisms. The dependencies can be obtained from programs written in a structured high-level language that has explicit data-parallel constructs, e.g. `forall`, or from sequential programs by means of a parallelizing compiler [2,4,12,14]. In other works we also showed that paradigms other than the *geometric* one can be efficiently implemented simply by using basic communication mechanisms [7,3].

Below, we will refer to the parallel machine which provides such a basic set of communication mechanisms as a Reduced Mechanism Set Parallel Machine (RMS-PM). The rationale, the advantages, and the drawbacks of RMS-PMs are similar to those of the RISC machines. A guiding reason comes from the experience of programmers of parallel applications. In fact, the paradigms recognized as those most frequently encountered in parallel applications [9,11] can be efficiently implemented on an RMS-PM. The main advantage in using reduced sets of mechanisms is their simplicity, which make it possible to devise more effective and optimized implementations of the mechanisms. On the other hand, the drawback of RMS-PMs is the need for compiling tools which free the programmer from dealing with these low-level mechanisms. This paper deals with the compiling technique needed to implement a *data-parallel* paradigm on top of an RMS-PM.

2 The `geometric` paradigm

In this paper we deal with a particular form of *data-parallel* computations, usually known as the `geometric` paradigm. In this paradigm, an array is first partitioned and then assigned to distinct

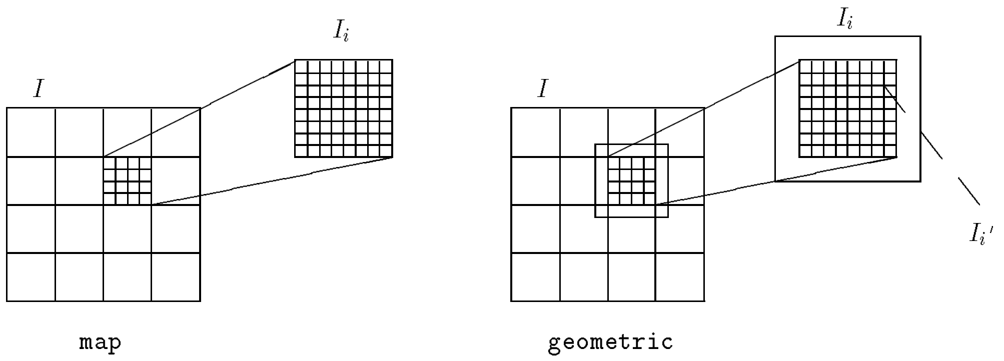


Figure 1: Examples of domain partitions for the `map` and the `geometric` paradigm.

processing nodes, which cooperate to compute identical tasks on the partitions assigned to them. Each processing node produces a partition of the whole output data, which is another array, so that a collecting phase is then needed to recompose the output data.

More formally, the `geometric` paradigm takes:

- an *array* data structure $\mathcal{I}$ as input (*domain*),
- an *array* data structure $\mathcal{O}$ as output (*codomain*).

Both $\mathcal{I}$ and $\mathcal{O}$ need to be partitioned into many partitions (subsets), and a data-parallel function $\mathcal{F}$ has to be applied to each partition of $\mathcal{I}$ to produce a corresponding partition of $\mathcal{O}$.

Let $\mathcal{I}_i$ and $\mathcal{O}_i$, $i \in [1, \ldots, N]$, be the partitions of $\mathcal{I}$ and $\mathcal{O}$, respectively. The following properties have to hold:

$$\mathcal{I} = \bigcup_{i=1}^{N} \mathcal{I}_i \qquad \mathcal{O} = \bigcup_{i=1}^{N} \mathcal{O}_i$$

$$\forall i, j \in [1, \ldots, N], i \neq j : \quad \mathcal{O}_i \cap \mathcal{O}_j = \emptyset$$

and

$$\forall i \in [1, \ldots, N] : \quad \mathcal{F}(\mathcal{I}_i) = \mathcal{O}_i$$

Note that, since it is also possible that $\mathcal{I}_i \cap \mathcal{I}_j \neq \emptyset$ for some values of i and j, the $\mathcal{I}_i$'s can overlap. If there is no overlapping between the various $\mathcal{I}_i$'s, the `geometric` paradigm becomes equivalent to the `map` one, which may be considered as its limiting case.

3 Implementation issues

The `map` paradigm, i.e. the limiting case of the `geometric` one, is the easy form of *data-parallelism*, since implementing it is straightforward. The domain is partitioned to a set of processing nodes, so that each node receives all the data needed to compute its own task.

The naive implementation suggested for the `map` would produce data replication if used for the `geometric` paradigm, since overlapping partitions $\mathcal{I}_i$ would have to be assigned to distinct processing nodes. To avoid data replication, the usual implementation for the `geometric` paradigm is thus based on a different decomposition of the domain, so that the data partition assigned to each node is smaller than the partition needed for its execution. More formally, if $\mathcal{I}'_i$ is the data partition of the domain $\mathcal{I}$ assigned to each node i, then $\forall i, j \in [1, \ldots, N],\ i \neq j$,

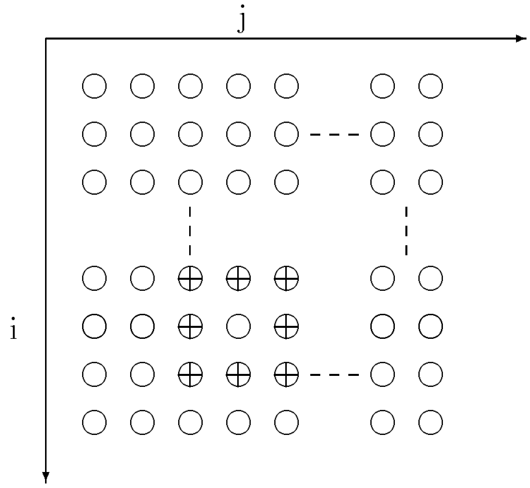


Figure 2: Eight neighbors of the node i, j.

$$\mathcal{I}'_i \subseteq \mathcal{I}_i, \quad \text{and} \quad \mathcal{I}'_i \cap \mathcal{I}'_j = \emptyset$$

The rest of the domain partition, i.e. $\mathcal{I}_i - \mathcal{I}'_i$, is allocated to other processing nodes. All the processing nodes thus have to communicate with each other to gather all the data items that are needed for the execution of the function $\mathcal{F}$.

Figure 1 shows two examples of partitions assigned to the various processing nodes implementing either a `map` or a `geometric` paradigm. In both the examples a two-dimensional array, i.e. the input domain, is distributed to a two-dimensional mesh of processing nodes.

4 Compiling dependencies on a mesh-based RMS-PM

In this paper we deal with the implementation of `geometric` paradigms on an RMS-PM whose nodes are arranged in a *mesh* network topology, and whose links are bidirectional. Suppose that the input domain $\mathcal{I}$ of our paradigm is thus a two-dimensional array `a[][]`, and that the data partition assigned to each node, namely $\mathcal{I}'_{i,j}$, is the single element of the array `a[i][j]`[1]. Thus, each processing node, i.e. $\mathcal{N}_{i,j}$, and the associated domain partitions, i.e. $\mathcal{I}_{i,j}$, $\mathcal{I}'_{i,j}$, and $\mathcal{O}_{i,j}$, are identified by the same pair of indexes (i, j).

Suppose that the data items that $\mathcal{N}_{i,j}$ needs to access, i.e. $\mathcal{I}_{i,j} - \mathcal{I}'_{i,j}$, are those corresponding to the following indexes: $(i-1,\ j)$, $(i+1,\ j)$, $(i,\ j-1)$, $(i,\ j+1)$, $(i-1,\ j-1)$, $(i-1,\ j+1)$, $(i+1,\ j-1)$, $(i+1,\ j+1)$. The positions of these data items with respect to $\mathcal{I}'_{i,j}$ are illustrated in Figure 2. For each of these data items, we know the *distance* between the $N_{i,j}$ and the other node to which such a data item has been assigned. In fact, suppose that $N_{i,j}$ needs the data item assigned to the $N_{i+d_i,\ j+d_j}$, where d_i and d_j are integers (positive or negative). The distance between two such nodes is

$$\mid d_i \mid \ + \ \mid d_j \mid$$

where $\mid d_i \mid$ and $\mid d_j \mid$ are the numbers of *routing hops* in each dimension that the messages have to cross to arrive at the destination node. As regards the example in Figure 2, the data items for which only one routing hop is needed are:

$$(i+1,\ j), \quad (i-1,\ j), \quad (i,\ j+1), \quad (i,\ j-1)$$

[1] Since we will refer to a two-dimensional mesh, hereafter we will use a pair of indexes to denote the various data partitions, i.e., $\mathcal{I}'_{i,j}$, $\mathcal{I}_{i,j}$, and $\mathcal{O}_{i,j}$.

while the data items for which two routing hops are needed are:

$$(i-1,\ j-1),\quad (i-1,\ j+1),\quad (i+1,\ j-1),\quad (i+1,\ j+1)$$

The problem is how these routing hops can be arranged so that

- no conflicts arise between messages to be transmitted over the same link;
- all the $N_{i,j}$ know all the data items of $\mathcal{I}_{i,j}$ within a given amount of time.

4.1 Implementation on a mesh of virtual nodes

In this section we present the implementation of the `geometric` paradigm on a *virtual*, mesh-connected, RMS-PM. The RMS-PM is regarded as a virtual one, since the number of processing nodes is unbounded. In fact, the first step of our compiling technique produces an implementation for the virtual machine, where the data parallel tasks refer to data partitions with the finest granularity. The next step in the technique consists in augmenting the granularity of the data partitions and their actual allocation, as well as in optimizing the interprocessor communications.

The ability of the compiler to decide the optimal granularity of the tasks encourages programming styles that do not cope with optimizations which are only possible by knowing the low-level architectural features of the target parallel machine, such as the actual parallelism degree of the machine, or the best granularity of the partitions to assign to each processing node to exploit the computation/communication ratio better.

To present our compiling technique, we will refer to the following definition of the `geometric` paradigm, seen as a function that transforms an array into another array:

$$\texttt{geometric}:\quad Type_{in}[N][M] \longrightarrow Type_{out}[N][M]$$

where $\mathcal{T}[\,][\,]$ identifies the type of a 2-dimensional array, whose elements have type $\mathcal{T}$.

As previously stated, the first step in our technique implements this paradigm on a virtual mesh of $N \times M$ nodes, by allocating a fine-grain process $W_{i,j}$, called *worker*, to each node $N_{i,j}$ of the virtual mesh. Suppose that the input domain is an array `a[N][M]`, which has type $Type_{in}[\,][\,]$, while the codomain is another array `b[N][M]`, with type $Type_{out}[\,][\,]$. The corresponding partitions with minimum granularity are:

- $\mathcal{I}'_{i,j} = \texttt{a[i][j]}$,
- $\mathcal{O}_{i,j} = \texttt{b[i][j]}$,

where $i \in [0, \ldots, N-1]$, and $j \in [0, \ldots, M-1]$. For any i, j, $\mathcal{I}'_{i,j}$ and $\mathcal{O}_{i,j}$ are assigned to the worker $W_{i,j}$, which, in turn, is mapped onto the node $N_{i,j}$ of the mesh. Note that the various nodes $N_{i,j}$ and the associated workers $W_{i,j}$ are identified by the same pair of indexes, so that we will refer to both indifferently.

Our compiling technique derives which messages the workers have to exchange with each other from the knowledge of both $\mathcal{I}_{i,j}$ and $\mathcal{I}'_{i,j}$. Our technique, however, only works if the shape of the partitions $\mathcal{I}_{i,j}$ have particular features. In order to formally specify the shapes allowed for these partitions, we need the definitions given below.

4.1.1 Preliminary definitions

Definition 1 *Let $\mathcal{M}$ be a mesh. A pair of workers $W_{i,j}, W_{h,k} \in \mathcal{M}$ are defined as neighbors iff one of the four following conditions holds*

- $i = h \pm 1 \ \wedge \ j = k$;
- $i = h \ \wedge \ j = k \pm 1$

□

Definition 2 *Given a mesh $\mathcal{M}$, and a workers $W_{i,j} \in \mathcal{M}$, for each pair (h,k) s.t. $W_{i+h,j+k} \in \mathcal{M}$, the following workers are* symmetrical *w.r.t. $W_{i,j}$*

$$\mathcal{S} = \ \{W_{i+h,j+k}, \ W_{i+k,j-h}, \ W_{i-h,j-k}, \ W_{i-k,j+h}\}$$

The cardinality of $\mathcal{S}$ is 4, the same as the interconnection degree of each node of mesh. The worker $W_{i,j}$ is defined as the center of symmetry *of $\mathcal{S}$, while the value $\mid h \mid + \mid k \mid$ is referred to as the* distance *of any worker in $\mathcal{S}$ from $W_{i,j}$.*

Note that the four neighbors *of the worker $W_{i,j}$ (see Def. 1) correspond to a particular set of symmetrical workers w.r.t. $W_{i,j}$, and whose* distance *from $W_{i,j}$ is 1.*

□

Definition 3 *Given a mesh $\mathcal{M}$, and a set of 4 symmetrical workers $\{W_0, W_1, W_2, W_3\}$ w.r.t. $W_{i,j} \in \mathcal{M}$ (see Def. 2), the ordering in which we find the 4 workers when moving* clockwise *w.r.t. the center of symmetry $W_{i,j}$ is defined as the* circular clockwise ordering.

□

The following definition of *symmetrical region* is concerned with the workers of the mesh to which $\mathcal{I}_{i,j}$ is allocated. Thus, because of the correspondence between the indexes of the workers and those of the associated partitions, the following definition also characterizes the shapes of $\mathcal{I}_{i,j}$ for which our compiling technique works.

Definition 4 *Given a mesh $\mathcal{M}$, and a subset $\mathcal{R}^{(d)}$ of workers in $\mathcal{M}$, $d \geq 0$, $|\mathcal{R}^{(d)}| = 4d+1$, we define $\mathcal{R}^{(d)}$ as a* symmetrical region *iff $\mathcal{R}^{(d)}$ can be built by using the following recursive definition*

1. *$\exists W_{i,j} \in \mathcal{M}$ s.t. $\mathcal{R}^{(0)} = \{W_{i,j}\}$;*
2. *$\mathcal{R}^{(n)} = \mathcal{R}^{(n-1)} \cup \{W^0, W^1, W^2, W^3\}$, where*
   - *$\mathcal{S} = \{W^0, W^1, W^2, W^3\}$ is a set of symmetrical workers w.r.t. $W_{i,j} \in \mathcal{R}^{(0)}$ (see Def. 2);*
   - *$\mathcal{S} \cap \mathcal{R}^{(n-1)} = \ \emptyset$;*
   - *for every $W \in \{W^0, W^1, W^2, W^3\}$, there exists a distinct worker $W' \in R^{(n-1)}$ s.t. W and W' are neighboring (see Def. 1);*
   - *if d is the* distance *of any worker in $\mathcal{S}$ from $W_{i,j} \in \mathcal{R}^{(0)}$ (see Def. 2), all the other worker $W \in \mathcal{R}^{(n-1)}$ are characterized by distances $\leq d$.*

Note that the recursive construction of $\mathcal{R}^{(d)}$ induces an enumeration of the workers in $\mathcal{R}^{(d)}$.

The worker $W_{i,j} \in \mathcal{R}^{(0)}$ is referred to as the center of symmetry *of $\mathcal{R}^{(d)}$. Moreover, by constructive hypothesis, the cardinality of $\mathcal{R}^{(d)}$ is $|\mathcal{R}^{(d)}| = 4d+1$.*

Below we refer to $\mathcal{R}^{(d)}_{i,j}$ as the symmetrical region $\mathcal{R}^{(d)}$ that has $W_{i,j}$ as its center of symmetry.

□

4.1.2 Soundness of the technique

In the previous section we gave the definition of *symmetrical region*. Our compiling technique works for those instances of the `geometric` paradigm where the input partition $\mathcal{I}_{i,j}$ has been allocated to a set of workers included in a particular symmetrical region. Only in this case can our technique arrange *local communications* between the various workers. These communications allow all the workers to receive all the data needed for their computations, i.e. the whole data partition $\mathcal{I}_{i,j}$, in a given amount of time.

Before showing the soundness of the technique, we need the following definition of *communication phase*.

Definition 5 *Let $\mathcal{M}$ be a mesh, whose physical links are bidirectional. We define a* communication phase *as a particular computational phase of all workers mapped onto the nodes of $\mathcal{M}$. During a communication phase, every $W_{i,j}$ exchanges messages with its neighbors. Since each $W_{i,j}$ has 4 neighbors (see Def. 1), during a* communication phase *$W_{i,j}$ can send 4 messages to its neighbors, and can receive 4 messages from the same neighbors.*

□

Finally, we can show the following Theorem, which states an *upper bound* on the number of *communication phases* needed by every worker to receive the data allocated to the other workers of a given *symmetrical region*.

Theorem 1 *Let $\mathcal{M}$ be a mesh, whose physical links are bidirectional.*

Suppose that, for its computation, every worker $W_{i,j} \in \mathcal{M}$ needs the data items allocated to the workers belonging to the symmetrical region *$\mathcal{R}^{(d)}_{i,j}$ (see Def. 4).*

Every worker $W_{i,j}$ can know all the data items allocated to the workers in $\mathcal{R}^{(d)}_{i,j}$ within, at most, $\frac{|\mathcal{R}^{(d)}|-1}{4} = \frac{(4d+1)-1}{d} = d$ communication phases *(see Def. 5).*

Proof by induction:
d=1:
The only symmetrical region that has $W_{i,j}$ as its center of symmetry, and has cardinality $4d+1 = 5$ is:

$$\mathcal{R}^{(1)}_{i,j} \quad = \quad \{W_{i,j}\} \cup \{W_{i\pm1,j},\ W_{i,j\pm1}\}$$

Since the worker $W_{i,j}$ is a *neighbor* of the other four workers in $\mathcal{R}^{(1)}_{i,j}$, it can receive, within one *communication phase*, all the data items allocated to its four neighboring workers $\{W_{i\pm1,j},\ W_{i,j\pm1}\}$. Note that this is true for all the workers $W_{i,j}$, since, during a communication phase, each pair of neighboring workers can exchange messages in both the directions.

d>1:
Suppose that the Theorem holds for $d-1$. This means that, for every pair (i,j), there exists a region $\mathcal{R}^{(d-1)}_{i,j}$, with $W_{i,j}$ as its center of symmetry, and whose cardinality is $4(d-1)+1$, s.t. $W_{i,j}$ received all the data items allocated to the workers in $\mathcal{R}^{(d-1)}_{i,j}$ within $d-1$ communication phases.

By hypothesis, $\mathcal{R}^{(d-1)}_{i,j} \subset \mathcal{R}^{(d)}_{i,j}$, and $\mathcal{R}^{(d)}_{i,j} = \mathcal{R}^{(d-1)}_{i,j} \cup \mathcal{S}$, where $\mathcal{S} = \{W^0, W^1, W^2, W^3\}$ is a set of symmetrical workers.

The Theorem holds if the worker $W_{i,j}$ can receive all the data items allocated to $\mathcal{S}$ in only one communication phase. This is only possible if each of the data items allocated to $\mathcal{S}$ is known by a distinct neighbor of $W_{i,j}$.

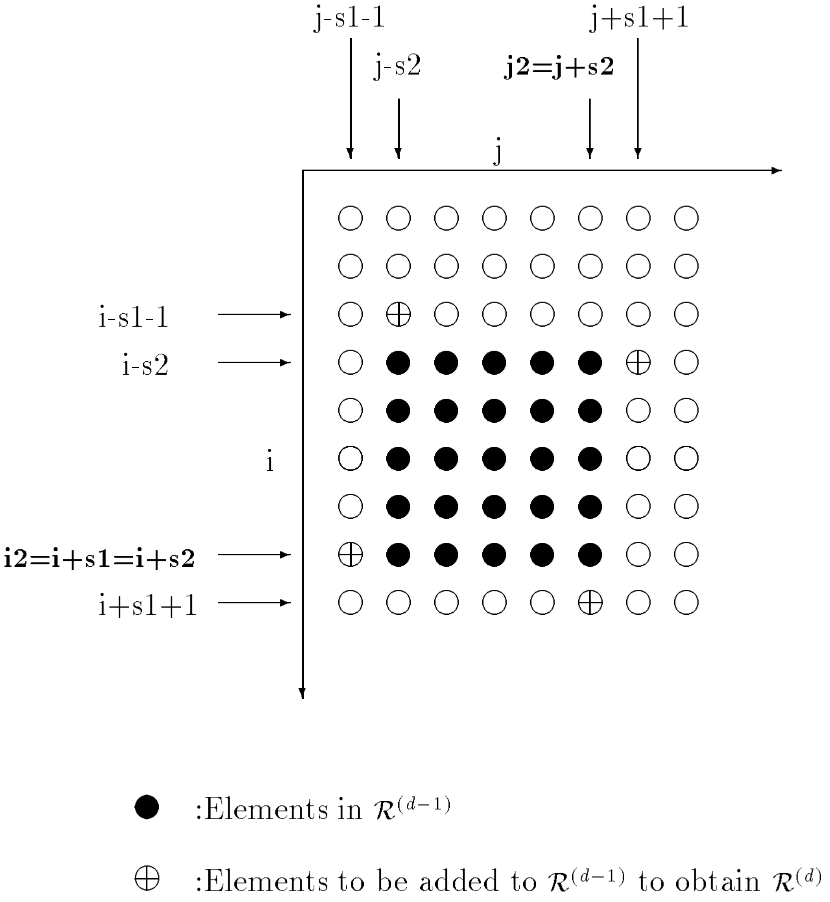


Figure 3: A symmetrical region $\mathcal{R}^{(d-1)}$, and the four workers to be added to obtain the region $\mathcal{R}^{(d)}$.

By hypothesis, each worker in $\mathcal{S}$ is a neighbor of a worker in $\mathcal{R}_{i,j}^{(d-1)}$. Suppose that there exists a worker $W^s \in \mathcal{S}$, $s \in \{0, \ldots, 3\}$, s.t. W^s is a neighbor of $W_{h,k} \in \mathcal{R}_{i,j}^{(d-1)}$, where $h \geq i, k \geq j$. The hypothesis on h and k is very general. In fact, because of symmetry reasons, there always exists a $W^s \in \mathcal{S}$ s.t. its neighbor $W_{h,k} \in \mathcal{R}_{i,j}^{(d-1)}$ falls in the first quadrant of that Cartesian plane whose origin is placed in $W_{i,j}$.

Assume that $h = i+s_1$ and $k = j+s_2$, where $s_1, s_2 \geq 0$ by hypothesis. Since W^s and $W_{h,k}$ are neighboring, some relations have to exist between the indexes (h, k), and the indexes associated with W^s. Moreover, since the *distance* of W^s from $W_{i,j}$ must be greater than (or equal to) the *distance* of $W_{h,k}$ from $W_{i,j}$ (see Def. 4), the only possible relations are:

$$W^s = W_{h+1,k} = W_{i+s_1+1,j+s_2} \quad or \quad W^s = W_{h,k+1} = W_{i+s_1,j+s_2+1}$$

We prove the Theorem for the former case, i.e., $W^s = W_{h+1,k}$, by showing that the neighbor $W_{i+1,j}$ of $W_{i,j}$ also knows the data item allocated to $W^s = W_{h+1,k} = W_{i+s_1+1,j+s_2}$.

By inductive hypothesis, every worker $W_{i,j}$ knows the data allocated to a region $\mathcal{R}_{i,j}^{(d-1)}$, which has itself as the center of symmetry. Thus, if $W_{i,j}$ knows the data item allocated to $W_{i+s_1,j+s_2} = W_{h,k}$, then $W_{i+1,j}$ knows the data item allocated to $W_{(i+1)+s_1,j+s_2} = W_{h+1,k} = W^s$.

The same proof holds for the other three workers, symmetrical of W^s with respect to $W_{i,j}$ (see Figure 3):

$$W_{i+s_2,j-s_1-1},\ W_{i-s_1-1,j-s_2},\ W_{i-s_2,j+s_1+1}$$

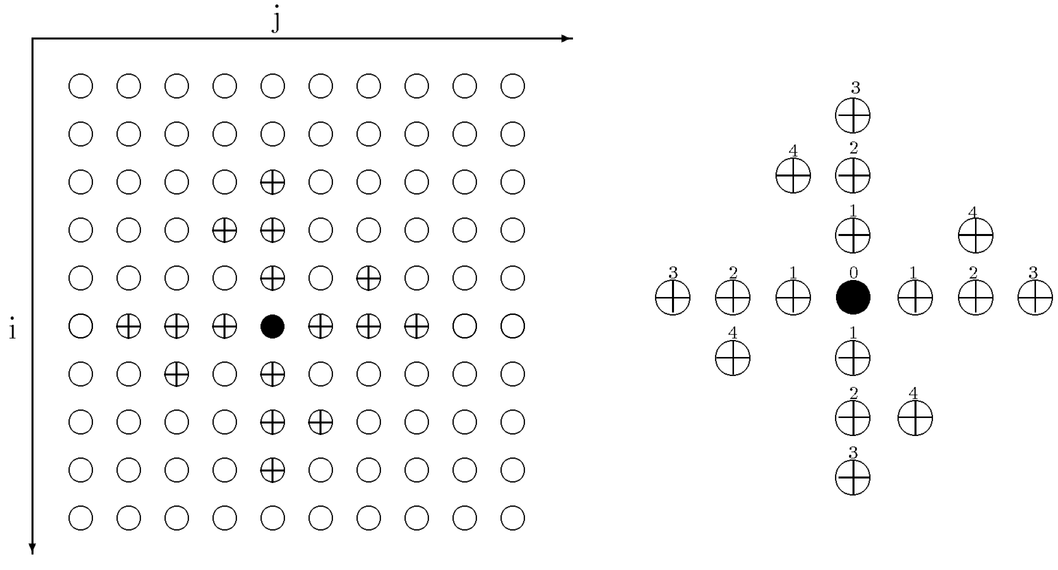


Figure 4: (a) A *symmetrical region* $\mathcal{R}^{(4)}_{i,j}$, which has $W_{i,j}$ as its center of symmetry; (b) the enumeration of the workers in $\mathcal{R}^{(4)}_{i,j}$.

For all the workers in $\mathcal{S}$, the following table shows the neighbors of $W_{i,j}$ that know the data items allocated to them:

Ord.	New worker	Neighbor of $W_{i,j}$
0	$W^s = W_{i+s_1+1,j+s_2}$	$W_{i+1,j}$
1	$W^{(s+1)\mathrm{mod}4} = W_{i+s_2,j-s_1-1}$	$W_{i,j-1}$
2	$W^{(s+2)\mathrm{mod}4} = W_{i-s_1-1,j-s_2}$	$W_{i-1,j}$
3	$W^{(s+3)\mathrm{mod}4} = W_{i-s_2,j+s_1+1}$	$W_{i,j+1}$

(1)

A further property, which will be used to generate the routing code between the workers, concerns the ordering of the entries in the table. In fact, the enumeration of these entries corresponds exactly to the *circular clockwise ordering* of the symmetrical regions appearing in the two columns of the table (see Def. 3).

By the same reasoning, we can prove the Theorem if the latter relation holds, i.e. $W^s = W_{h,k+1} = W_{i+s_1,j+s_2+1}$. The following table shows the neighbors of $W_{i,j}$ that know, in this case, the data items allocated to the various workers in $\mathcal{S}$:

Ord.	New worker	Neighbor of $W_{i,j}$
0	$W^s = W_{i+s_1,j+s_2+1}$	$W_{i,j+1}$
1	$W^{(s+1)\mathrm{mod}4} = W_{i+s_2+1,j-s_1}$	$W_{i+1,j}$
2	$W^{(s+2)\mathrm{mod}4} = W_{i-s_1,j-s_2-1}$	$W_{i,j-1}$
3	$W^{(s+3)\mathrm{mod}4} = W_{i-s_2-1,j+s_1}$	$W_{i-1,j}$

(2)

The property on the ordering of the rows in the table holds in this case as well.

Note that this proof holds for every worker $W_{i,j}$, since, during a communication phase, each pair of neighboring workers can exchange messages in both the directions.

□

Thus, Theorem 1 states the minimum number of communication phases that a generic worker $W_{i,j}$ takes to receive all the data items allocated to $\mathcal{R}_{i,j}^{(d)}$. To obtain this, the workers $W_{i,j}$ exchange messages with their neighbors, thus receiving data items originally allocated to the other workers in $\mathcal{R}_{i,j}^{(d)}$. The following Corollary gives some properties on the data items that every $W_{i,j}$ can receive from its neighbors.

Corollary 1 *If Theorem 1 holds, the following relations also hold:*

1. *the data items that $W_{i,j}$ receives from $W_{i+1,j}$ were originally allocated to $W_{h,k} \in \mathcal{R}_{i,j}^{(d)}$, $h > i$;*
2. *the data items that $W_{i,j}$ receives from $W_{i,j-1}$ were originally allocated to $W_{h,k} \in \mathcal{R}_{i,j}^{(d)}$, $k < j$;*
3. *the data items that $W_{i,j}$ receives from $W_{i-1,j}$ were originally allocated to $W_{h,k} \in \mathcal{R}_{i,j}^{(d)}$, $h < i$;*
4. *the data items that $W_{i,j}$ receives from $W_{i,j+1}$ were originally allocated to $W_{h,k} \in \mathcal{R}_{i,j}^{(d)}$, $k > j$.*

Proof by induction:
d=1:
For $d = 1$, the proof is trivial.

d>1:
Assume that the Corollary holds for $d - 1$. Let $\mathcal{S} = \{W^0, W^1, W^2, W^3\}$ be the set of symmetric workers to be added to $\mathcal{R}_{i,j}^{(d-1)}$ to obtain $\mathcal{R}_{i,j}^{(d)}$.

From the proof of Theorem 1, we know that the neighbors of $W_{i,j}$ which know the data items allocated to the various $W^s \in \mathcal{S}$, $s > \{0, \ldots, 3\}$, are described either by Table (1) or Table (2). It is trivial to show that the Corollary holds in both the cases.

As regards the first entry of the former table, we can see that $W_{i,j}$ receives the data item allocated to $W_{h,k} = W_{i+s_1,j+s_2+1}$ from $W_{i,j+1}$. Since $s_1, s_2 \geq 0$, then $k = j + s_2 + 1 > j$, as $s_2 \geq 0$.

The same proof can be applied to the other entries in both tables.

□

4.2 An algorithm to generate the routing code

Figure 4.(a) shows a *symmetrical region* $\mathcal{R}_{i,j}^{(4)}$. Theorem 1 states that the data items allocated to the workers in $\mathcal{R}_{i,j}^{(4)}$ can be transmitted to the worker $W_{i,j}$ within, at most, four *communication phases*. In this section we present the algorithm that generates these communication phases.

To generate the input/output commands between a generic worker $W_{i,j}$ and its neighbors, i.e. to generate its communication phases, we need an enumeration like the one shown in Figure 4.(b). This enumeration is induced from the recursive construction of the region $\mathcal{R}_{i,j}^{(4)}$ (see Def. 4). In fact, the element labeled with 0 belongs to $\mathcal{R}_{i,j}^{(0)}$, the nodes labeled with 1 are those added to $\mathcal{R}_{i,j}^{(0)}$ to obtain $\mathcal{R}_{i,j}^{(1)}$, and so on. The meaning of this enumeration is:

1. at the beginning, the worker $W_{i,j}$ knows the data item whose associated number is 0;
2. after d *communication phases* $W_{i,j}$ knows all the data items allocated to the workers whose associated numbers are less than (or equal) to d.

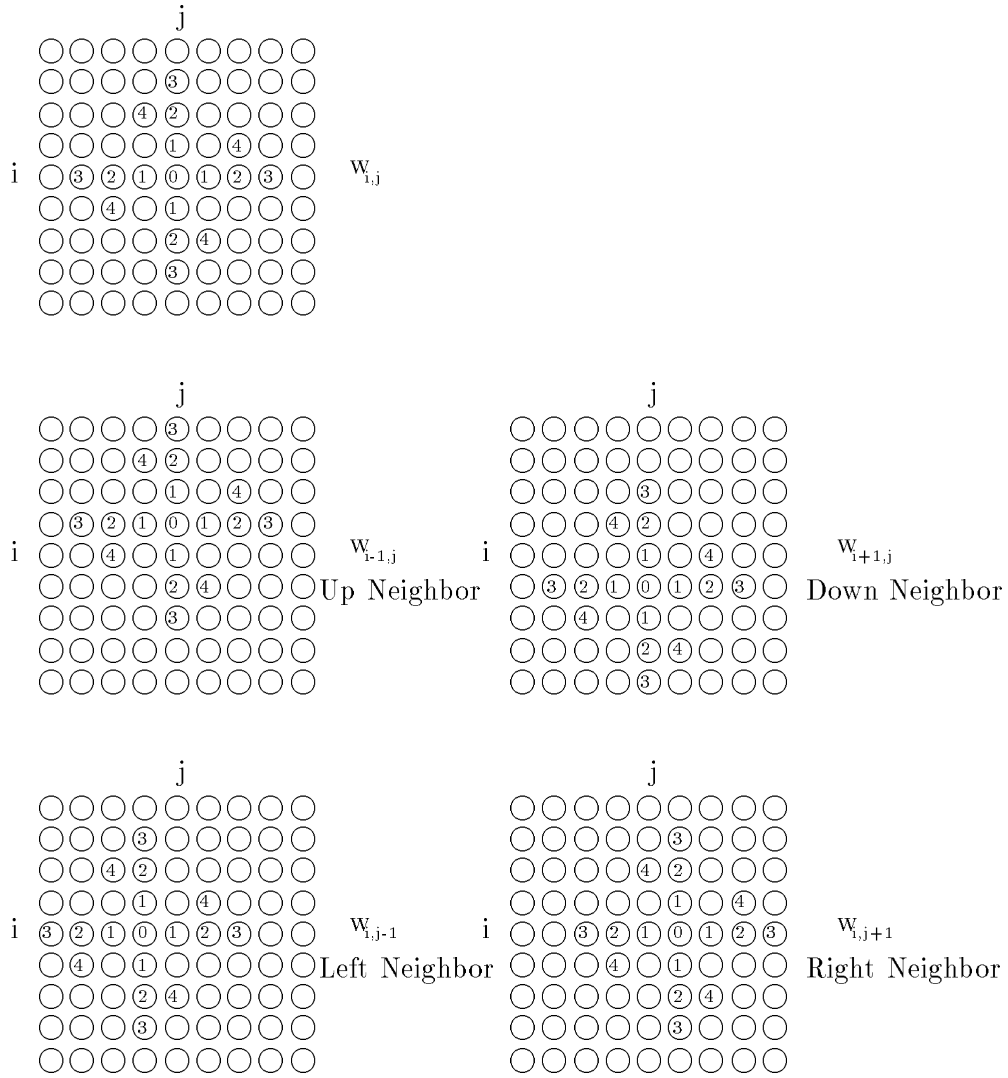


Figure 5: An enumerated symmetrical region with respect to distinct centers of symmetry.

Figure 5 shows the same region and the corresponding enumeration if the center of symmetry is either the worker $W_{i,j}$, or one of its four neighbors.

The algorithm shown in Figure 6 exploits an enumeration like the one in Figure 5 to generate the communication phases of a generic worker $W_{i,j}$. It considers a generic array `a[][]`, which is partitioned and assigned to the various workers mapped onto the virtual mesh. Each worker $W_{i,j}$ receives the element `a[i][j]`, and thus $\mathcal{I}'_{i,j} =$ `a[i][j]`. The data partition $\mathcal{I}_{i,j}$, which each worker $W_{i,j}$ needs for its computation, corresponds to the data items allocated to the workers in $\mathcal{R}^{(d)}_{i,j}$.

Note that the algorithm only generates the *input* communication phases of a generic worker $W_{i,j}$. The output commands, which complete the communication phases of each worker, are easily derived from the input phases generated by the algorithm, as shown by the following example.

An example By using the algorithm in Figure 6, the four input communication phases of the example in Figure 4 are built as follows:

```
procedure create_input_communication_phase(i,j)
{
      for (id_ph = 1  to  d)
            Input_Phase_{id_ph} = ∅
      for (id_ph = 1  to  d)
            {
```

1. Let $S = \{W_{h_0,k_0}, W_{h_1,k_1}, W_{h_2,k_2}, W_{h_3,k_3}\}$ be the symmetrical set whose elements are labeled with id_{ph}. The indexing of this set corresponds to the circular clockwise ordering (see Def. 3).

2. Let $N = \{N_0, N_1, N_2, N_3\}$ be the symmetrical set of the neighbors of $W_{i,j}$. The indexing of this set corresponds to the circular clockwise ordering (see Def. 3).

3. Find an element $W_{h_i,k_i} \in S$ s.t. W_{h_i,k_i} also belongs to the region $\mathcal{R}^{(id_{ph}\ -\ 1)}$, which has N_0 as its center of symmetry. By hypothesis, W_{h_i,k_i} must be labeled with a number n, $n \leq id_{ph} - 1$.

4. for (ind = 0 to 3)

   $\texttt{Input\_Phase}_{id_{ph}}$ = $\texttt{Input\_Phase}_{id_{ph}}$ $\cup$

   { $\texttt{N}_{\texttt{ind}}$? a[$h_{(i+\texttt{ind})\bmod 4}$] [$k_{(i+\texttt{ind})\bmod 4}$] } [a]

[a] We have used a CSP notation for synchronous input/output commands, i.e., $P!msg$ to send a message to process P, and $P?vtg$ to receive a message from P.

```
            }
}
```

Figure 6: An algorithm to generate the *input communication phases* of $W_{i,j}$.

```
Input_Phase1   = { Up    ? a[i-1][j],       Input_Phase3   = { Up    ? a[i-3][j],
                   Right ? a[i][j+1],                          Right ? a[i][j+3],
                   Down  ? a[i+1][j],                          Down  ? a[i+3][j],
                   Left  ? a[i][j-1]                           Left  ? a[i][j-3]
                 }                                           }

Input_Phase2   = { Up    ? a[i-2][j],       Input_Phase4   = { Up    ? a[i-1][j+2],
                   Right ? a[i][j+2],                          Right ? a[i+2][j+1],
                   Down  ? a[i+2][j],                          Down  ? a[i+1][j-2],
                   Left  ? a[i][j-2]                           Left  ? a[i-2][j-1]
                 }                                           }
```

where `Up` $= W_{i-1,j}$, `Right` $= W_{i,j+1}$, `Down` $= W_{i+1,j}$, and `Left` $= W_{i,j-1}$.

To generate the corresponding output commands, we need to generate first the input commands for the four neighbors of $W_{i,j}$. This can easily be done by translating the array indexes used in each communication phase.

Suppose that we want to generate the input phases of the neighbor `Up` $= W_{i-1,j}$. This can be done by simply subtracting one from the first index of the array. The new input phases of the worker `Up` are thus:

```
Input_Phase1  =  { Up    ? a[i-2][j],        Input_Phase3  =  { Up    ? a[i-2][j],
                   Right ? a[i-1][j+1],                         Right ? a[i-1][j+3],
                   Down  ? a[i][j],                             Down  ? a[i+2][j],
                   Left  ? a[i-1][j-1]                          Left  ? a[i-1][j-3]
                 }                                            }

Input_Phase2  =  { Up    ? a[i-1][j],        Input_Phase4  =  { Up    ? a[i-2][j+2],
                   Right ? a[i-1][j+2],                         Right ? a[i+1][j+1],
                   Down  ? a[i+1][j],                           Down  ? a[i][j-2],
                   Left  ? a[i-1][j-2]                          Left  ? a[i-3][j-1]
                 }                                            }
```

Of the input commands that make up these four phases, we are interested in the input commands from `Down`= $W_{i,j}$.

```
Input_Phase1  =  { Down ? a[i][j]   }      Input_Phase3  =  { Down ? a[i+2][j] }
Input_Phase2  =  { Down ? a[i+1][j] }      Input_Phase4  =  { Down ? a[i][j-2] }
```

The corresponding output commands of $W_{i,j}$ are thus:

```
Output_Phase1  =  { Up ! a[i][j]   }       Output_Phase3  =  { Up ! a[i+2][j] }
Output_Phase2  =  { Up ! a[i+1][j] }       Output_Phase4  =  { Up ! a[i][j-2] }
```

The same procedure can be applied to all the other neighbors of $W_{i,j}$, in order to generate all the output commands of each phase, and, thus, all the communication phases of a generic $W_{i,j}$:

```
Phase1  =  { Up    ! a[i][j],              Phase3  =  { Up    ! a[i+2][j],
             Right ! a[i][j],                           Right ! a[i][j-2],
             Down  ! a[i][j],                           Down  ! a[i-2][j],
             Left  ! a[i][j],                           Left  ! a[i][j+2],

             Up    ? a[i-1][j],                         Up    ? a[i-3][j],
             Right ? a[i][j+1],                         Right ? a[i][j+3],
             Down  ? a[i+1][j],                         Down  ? a[i+3][j],
             Left  ? a[i][j-1]                          Left  ? a[i][j-3]
           }                                          }

Phase2  =  { Up    ! a[i+1][j],            Phase4  =  { Up    ! a[i][j-2],
             Right ! a[i][j-1],                         Right ! a[i-2][j],
             Down  ! a[i-1][j],                         Down  ! a[i][j+2],
             Left  ! a[i][j+1],                         Left  ! a[i+2][j],

             Up    ? a[i-2][j],                         Up    ? a[i-1][j+2],
             Right ? a[i][j+2],                         Right ? a[i+2][j+1],
             Down  ? a[i+2][j],                         Down  ? a[i+1][j-2],
             Left  ? a[i][j-2]                          Left  ? a[i-2][j-1]
           }                                          }
```

Once the various communication phases $\{\texttt{Phase}_\texttt{i} \mid \texttt{i} \in [1, \ldots, d]\}$ are generated, the code of the generic workers $W_{i,j}$ becomes:

virtual_worker(i,j)
{
   Allocate memory for $\mathcal{I}_{i,j}$ *and* $\mathcal{O}_{i,j}$

Receive the domain partition $\mathcal{I}'_{i,j}$

```
for (ind = 1 to d)
      do the communication phase Phase_ind
```

$\mathcal{O}_{i,j} = \mathcal{F}(\mathcal{I}_{i,j})$

}

4.3 Optimization of communication phases

One of the most useful optimizations in message-passing programs concerns the reduction of synchronization points between processes. The concatenation of multiple short messages to be sent to the same process is one of these optimizations.

In the example above we built some communication phases, but some of these phases can be merged together. In fact, the data items sent during `Phase4` are received during `Phase2`. Thus, the communications of `Phase4` can be brought forward, and be included in `Phase3`:

```
Phase3  =  { Up    ! { a[i+2][j], a[i][j-2] },
             Right ! { a[i][j-2], a[i-2][j] },
             Down  ! { a[i-2][j], a[i][j+2] },
             Left  ! { a[i][j+2], a[i+2][j] },

             Up    ? { a[i-3][j], a[i-1][j+2] },
             Right ? { a[i][j+3], a[i+2][j+1] },
             Down  ? { a[i+3][j], a[i+1][j-2] },
             Left  ? { a[i][j-3], a[i-2][j-1] }
           }
```

This transformation can be done by slightly changing the algorithm in Figure 6. In fact, the only part that needs to be changed is the fourth step of the main `for`, i.e.,

```
3.   ...

4.   for (ind=0 to 3)
            Input_Phase_{n+1} = Input_Phase_{n+1} ∪
                 { N_ind ?  a[h_{(i+ind) mod 4}] [k_{(i+ind) mod 4}] }
```

The new input commands are now added to $\texttt{Input\_Phase}_{n+1}$ instead of $\texttt{Input\_Phase}_{id_{ph}}$. Note that, on the one hand, several input commands in the same phase and from the same worker can be combined together, but, on the other hand, some communication phases may remain empty.

4.4 Boundary conditions

So far we have given definitions, theorems and algorithms with respect to symmetrical regions whose center of symmetry is a generic worker $W_{i,j}$. Unfortunately, the definition of *symmetrical region* does not take into account the case in which the elements of the region fall out of the $N \times M$ mesh of virtual workers. Consider, for example, the region $\mathcal{R}^{(2)}_{i,j}$, which is defined as follows:

1. $\mathcal{R}^{(0)}_{i,j} = \{W_{i,j}\}$
2. $\mathcal{R}^{(1)}_{i,j} = \mathcal{R}^{(0)}_{i,j} \cup \{W_{i\pm1,j}, W_{i,j\pm1}\}$
3. $\mathcal{R}^{(2)}_{i,j} = \mathcal{R}^{(1)}_{i,j} \cup \{W_{i\pm1,j\pm1}\}$

Two of these regions, whose center of symmetry is denoted by a black circle, are shown in Figure 7.

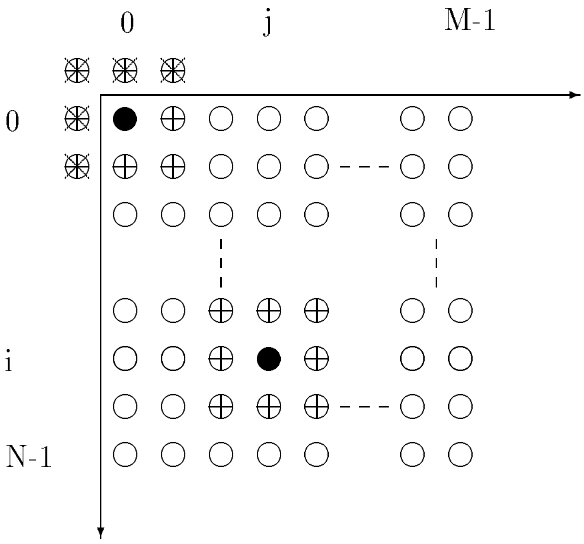


Figure 7: Symmetrical regions falling out of the boundaries.

```
procedure remove_communication_commands(i,j)
{
    for (id_ph = 1  to  d)
        {
          for each W_{t,s}!a[h][k] ∈ Phase_{id_ph} do
               if (not (0 ≤ t < N and 0 ≤ s < M)) then
                    Remove W_{t,s}!a[h][k] from Phase_{id_ph}
               else if (not (0 ≤ h < N and 0 ≤ k < M)) then
                    Remove W_{t,s}!a[h][k] from Phase_{id_ph}
               endif

          for each W_{t,s}?a[h][k] ∈ Phase_{id_ph} do
               if (not (0 ≤ t < N and 0 ≤ s < M)) then
                    Remove W_{t,s}?a[h][k] from Phase_{id_ph}
               else if (not (0 ≤ h < N and 0 ≤ k < M)) then
                    Remove W_{t,s}?a[h][k] from Phase_{id_ph}
               endif
        }
}
```

Figure 8: An algorithm that removes incorrect input/output commands.

Problems arise when this center of symmetry is close to the boundaries of the $N \times M$ mesh. For example, Figure 7 shows the region $\mathcal{R}^{(2)}_{0,0}$. Some of the workers in $\mathcal{R}^{(2)}_{0,0}$ do not belong to the virtual mesh, so that the computation phases obtained by the algorithm in Figure 6 for $W_{0,0}$ are not correct.

To solve this problem, once the communication phases of a generic worker $W_{i,j}$ is built, some of the input/output commands that make up the communication phases of some workers can be removed. This can be done by exploiting the algorithm in Figure 8. Note that this algorithm removes input/output commands from the communication phases for two reasons:

1. the neighbor $W_{t,s}$ does not exist;

2. the data item sent (received) to (from) $W_{t,s}$ is characterized by subscripts that are outside the range allowed.

As regards the the input commands that the algorithm removes, it is important to show that, if Condition 1 holds, 2 holds as well. If this did not hold, in fact, $W_{i,j}$ would not receive some of the data items needed for its computation.

Corollary 1 gives some properties about the data items that $W_{i,j}$ can receive from $W_{t,s}$. These properties can be used to show that, if $W_{t,s}$ does not exist (i.e., Condition 1 holds), then the data item received would be characterized by subscripts that are outside the range allowed (i.e., Condition 2 holds too). Therefore, suppose that $W_{t,s}$ does not exist, so that one of the following conditions holds:

$$\begin{array}{c} (t=-1 \quad \vee \quad t=N) \quad \wedge \quad (0 \le s \le M-1) \\ or \\ (s=-1 \quad \vee \quad s=M) \quad \wedge \quad (0 \le t \le N-1) \end{array} \tag{1}$$

Consider the case $t=-1$. Since $W_{t,s}$ is a neighbor of an existing $W_{i,j}$, then $t=i-1$, $s=j$, and thus $i=0$. As Corollary 1 states, $W_{i,j}$ should receive from $W_{t,s}=W_{i-1,j}$ data items originally allocated to $W_{h,k}$ (namely, the data items `a[h][k]`), where $h<i$. But, since $i=0$, all the subscripts `a[h][k]`, `h`$<i$, are out of range.

Thus, for $t=-1$, we proved that all the data items that $W_{i,j}$ should receive from $W_{t,s}$ are also outside the range allowed. The same proof can be applied in all the other cases described by (1).

As regards the example in Figure 7, where $\mathcal{I}_{i,j} = \mathtt{a}[(\mathtt{i}-1)\div(\mathtt{i}+1)][(\mathtt{j}-1)\div(\mathtt{j}+1)]$, the communications phases of a generic $W_{i,j}$ are:

```
Phase1  = { Up    ! a[i][j],
            Right ! a[i][j],
            Down  ! a[i][j],
            Left  ! a[i][j],

            Up    ? a[i-1][j],
            Right ? a[i][j+1],
            Down  ? a[i+1][j],
            Left  ? a[i][j-1]
          }

Phase2  = { Up    ! a[i][j+1],
            Right ! a[i+1][j],
            Down  ! a[i][j-1],
            Left  ! a[i-1][j],

            Up    ? a[i-1][j-1],
            Right ? a[i-1][j+1],
            Down  ? a[i+1][j+1],
            Left  ? a[i+1][j-1]
          }
```

Since $W_{0,0}$ is close to the boundaries, it has to access a smaller $\mathcal{I}_{i,j}$, i.e. $\mathcal{I}_{0,0} = \mathtt{a}[0\div 1][0\div 1]$. The algorithm shown in Figure 8 modifies the communication phases of $W_{0,0}$, so that the new phases become:

```
Phase1  = { Right ! a[i][j],
            Down  ! a[i][j],

            Right ? a[i][j+1],
            Down  ? a[i+1][j]
          }

Phase2  = { Right ! a[i+1][j],

            Down  ? a[i+1][j+1]
          }
```

4.5 Symmetrical regions covering input partitions

In the previous section we saw examples in which a generic worker $W_{i,j}$ needs a partition $\mathcal{I}_{i,j}$ that, except for the boundary effects, exactly corresponds to a symmetrical region $\mathcal{R}_{i,j}^{(d)}$. In any case, the same technique also works if this is not true.

Suppose that the partition each worker $W_{i,j}$ needs for its computation is:

$$\mathcal{I}_{i,j} \;= \{\texttt{a[i][0} \div \texttt{(M}-\texttt{1)]},\;\; \texttt{a[0} \div \texttt{(N}-\texttt{1)][j]}\},$$

i.e. the i^{th} row, and the j^{th} column of the array. This is the same kind of dependency as that which comes from the `geometric` implementation of the *matrix multiplication*. The smallest symmetrical region $\mathcal{R}_{i,j}^{(d)}$ that includes this $\mathcal{I}_{i,j}$, i.e. the region that is characterized by the smallest index d, and that covers the whole $\mathcal{I}_{i,j}$, is defined by the following recursive definition:

1. $\mathcal{R}_{i,j}^{(0)} = \{W_{i,j}\}$.
2. $\mathcal{R}_{i,j}^{(n)} = \mathcal{R}_{i,j}^{(n-1)} \cup \{W_{i+n,j}, W_{i,j-n}, W_{i-n,j}, W_{i,j+n}\}$.

The region $\mathcal{R}_{i,j}^{(max(N-1,M-1))}$, which is built by using the definition above, includes workers whose distance from $W_{i,j}$ is $max(N-1, M-1)$. Although some of the nodes in $\mathcal{R}_{i,j}^{(max(N-1,M-1))}$ fall out of the $N \times M$ mesh, the communication phases of a generic worker $W_{i,j}$ can be generated as follows:

- first, the communication phases of a generic virtual worker $W_{i,j}$ is generated by using the algorithm in Figure 6;
- then, unnecessary input/output commands on the boundaries are removed by using the algorithm illustrated in Figure 8.

4.6 Grain optimization

We have discussed the implementation of a `geometric` constructor on a mesh of virtual workers. If this mesh of virtual workers is mapped onto an actual mesh of processing nodes, the granularity of the data-parallel tasks may need to be adjusted to match the features of the target architecture.

If each worker is mapped onto a different node, the resulting grain is the minimum possible. Otherwise, several workers along with their data partitions can be mapped onto the same processing node, so that the granularity of the task and the data partition allocated to each node is increased. Hereafter, the sets of workers mapped onto the same processing node $N_{h,k}$ will be called *computational units*, and will be denoted by $\mathcal{CU}_{h,k}$. A generic $\mathcal{CU}_{h,k}$ is defined as follows:

$$\mathcal{CU}_{h,k} = \{W_{i,j} \;\;|\;\; h \cdot H \leq i < (h+1) \cdot H, \quad k \cdot B \leq j < (k+1) \cdot B\} \tag{2}$$

where $H, B > 0$, $N \bmod H = 0$, and $M \bmod B = 0$, i.e. each $\mathcal{CU}$ corresponds to a $B \times H$ box. Figure 9 shows an example of grain optimization with respect to a virtual mesh of 9×9 workers, mapped onto a actual mesh of 3×3 nodes. In this case, a distinct $\mathcal{CU}_{h,k}$ of 3×3 workers is allocated to each processing node of the actual mesh.

Our grain optimization technique is similar to the data distribution phases of [10], and the data-layout strategies of [12], though it does not cover the case of cyclic (also called interleaving) layout strategy.

To perform a grain optimization like the one illustrated in Figure 9, two different approaches can be adopted:

1. An approach that only relies on the multiprocessing features of RMS-PM. In this case the grain optimization might be done by allocating several virtual workers to the same node as distinct processes. Using this approach, the workers hold their own space of memory, and communicate by exchanging messages.

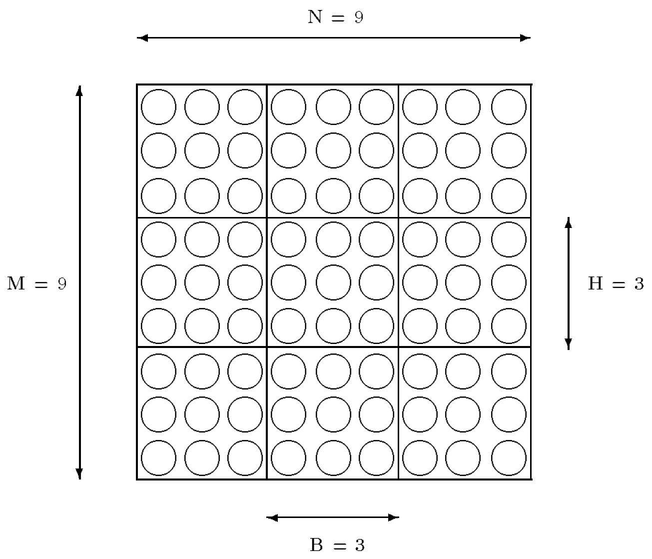


Figure 9: An example of grain optimization.

2. Conversely, since the former approach introduces overheads, the compiler might derive a single process to be mapped onto each node. From the functional point of view, this single process must be equivalent to multiprocessing each node with the various virtual workers, but it should be more efficient.

We will discuss the latter approach to grain optimization. The single process that must be mapped onto $N_{h,k}$ and that corresponds to $\mathcal{CU}_{h,k}$ is thus:

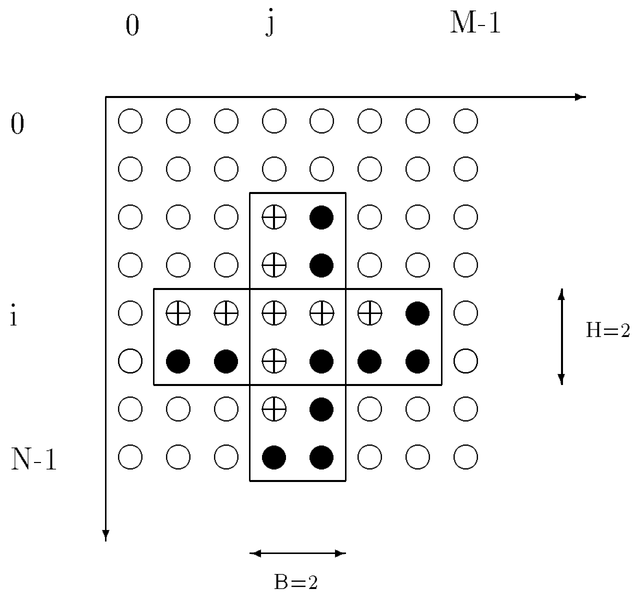
computational_unit(h,k)
{
  Allocate memory for $\bigcup \mathcal{I}_{i,j}$ *and* $\bigcup \mathcal{O}_{i,j}$
  Receive the domain partition $\bigcup \mathcal{I}'_{i,j}$, *where*
    $h \cdot H \leq i < (h+1) \cdot H$ and $k \cdot B \leq j < (k+1) \cdot B$

  for (ind = 1 to d)
    do *the communication phase* New_Phase$_{ind}$

  for each i,j s.t. $h \cdot H \leq \mathtt{i} < (h+1) \cdot H$ and $k \cdot B \leq \mathtt{j} < (k+1) \cdot B$
    $\mathcal{O}_{i,j} = \mathcal{F}(\mathcal{I}_{i,j})$
}

To generate this new process, is necessary to provide the new communication phases New_Phase$_{ind}$. Each of these phases can be obtained by means of the following steps:

1. Make the union of all the communication phases of the virtual workers in $\mathcal{CU}_{h,k}$, i.e. New_Phase$_{ind}$ $= \bigcup$ Phase$_{ind}$.
2. Remove from New_Phase$_{ind}$ all the input/output commands between virtual workers which belong to $\mathcal{CU}_{h,k}$.
3. Combine all the input/output commands from/to virtual workers which are allocated to the same processing nodes. Note that these nodes can only be the neighbors of $N_{h,k}$, i.e. $\{N_{h\pm1,k}, N_{h,k\pm1}\}$.



$\oplus$: Region $\mathcal{R}^{(2)}_{i,j}$

$\oplus$ $\bullet$: Union of the four regions $\{\mathcal{R}^{(2)}_{i,j}, \mathcal{R}^{(2)}_{i+1,j}, \mathcal{R}^{(2)}_{i,j+1}, \mathcal{R}^{(2)}_{i+1,j+1}\}$

Figure 10: A computational unit $\mathcal{CU}_{h,k}$, and the symmetrical regions corresponding to each $\mathcal{I}_{i,j}$.

At the end of these steps, the only input/output commands that are not removed (and that are combined into input/output commands from/to the the neighbors of $N_{h,k}$) are those which refer to virtual workers which are not included in $\mathcal{CU}_{h,k}$. The number of these workers is $2(B+H)$, i.e. the number of "virtual links" that connect $\mathcal{CU}_{h,k}$ with the other $\mathcal{CU}$s.

A further optimization may be applied to the new communication phases $\texttt{New\_Phase}_{ind}$ thus obtained. In fact, some of them can be merged by concatenating multiple short messages into a longer one. These optimizations belong to the same class as those discussed in Sec. 4.3. Consider the example in Figure 10. The symmetrical region $\mathcal{R}^{(2)}_{i,j}$, which is composed of the workers in $\{W_{i,(j-2 \div j+2)}, W_{(i-2 \div i+2),j}\}$, exactly corresponds to the partition $\mathcal{I}_{i,j}$, and the 2×2 computational unit that includes $W_{i,j}$ consists of the following workers:

$$\mathcal{CU}_{h,k} = \{W_{i,j}, W_{i,j+1}, W_{i+1,j}, W_{i+1,j+1}\}$$

After applying the three steps illustrated above, we obtain the following communication phases, related to a generic computational unit $\mathcal{CU}_{h,k}$ which is mapped onto $N_{h,k}$, and exchanges messages with its neighbors $\{\mathcal{CU}_{h\pm1,k}, \mathcal{CU}_{h,k\pm1}\}$:

$\texttt{New\_Phase}_1 =$

$$
\begin{array}{llll}
\{ & \mathcal{C}_{h-1,k} & ! & \{\mathtt{a[i][j]}, \quad \mathtt{a[i][j+1]}\}, \\
 & \mathcal{C}_{h,k+1} & ! & \{\mathtt{a[i+1][j]}, \quad \mathtt{a[i+1][j+1]}\}, \\
 & \mathcal{C}_{h,k-1} & ! & \{\mathtt{a[i][j]}, \quad \mathtt{a[i+1][j]}\}, \\
 & \mathcal{C}_{h+1,k} & ! & \{\mathtt{a[i][j+1]}, \quad \mathtt{a[i+1][j+1]}\}, \\
 & & & \\
 & \mathcal{C}_{h-1,k} & ? & \{\mathtt{a[i-1][j]}, \quad \mathtt{a[i-1][j+1]}\}, \\
 & \mathcal{C}_{h,k+1} & ? & \{\mathtt{a[i+2][j]}, \quad \mathtt{a[i+2][j+1]}\}, \\
 & \mathcal{C}_{h,k-1} & ? & \{\mathtt{a[i][j-1]}, \quad \mathtt{a[i+1][j-1]}\}, \\
 & \mathcal{C}_{h+1,k} & ? & \{\mathtt{a[i][j+2]}, \quad \mathtt{a[i+1][j+2]}\} \\
\} & & &
\end{array}
$$

$$
\begin{array}{llll}
\multicolumn{4}{l}{\mathtt{New\_Phase}_2 =} \\
\{ & \mathcal{C}_{h-1,k} & ! & \{\mathtt{a[i+1][j]}, \quad \mathtt{a[i+1][j+1]}\}, \\
 & \mathcal{C}_{h,k+1} & ! & \{\mathtt{a[i][j]}, \quad \mathtt{a[i][j+1]}\}, \\
 & \mathcal{C}_{h+1,k} & ! & \{\mathtt{a[i][j]}, \quad \mathtt{a[i+1][j]}\}, \\
 & \mathcal{C}_{h,k-1} & ! & \{\mathtt{a[i][j+1]}, \quad \mathtt{a[i+1][j+1]}\}, \\
 & & & \\
 & \mathcal{C}_{h-1,k} & ? & \{\mathtt{a[i-2][j]}, \quad \mathtt{a[i-2][j+1]}\}, \\
 & \mathcal{C}_{h,k+1} & ? & \{\mathtt{a[i+3][j]}, \quad \mathtt{a[i+3][j+1]}\}, \\
 & \mathcal{C}_{h+1,k} & ? & \{\mathtt{a[i][j+3]}, \quad \mathtt{a[i+1][j+3]}\}, \\
 & \mathcal{C}_{h,k-1} & ? & \{\mathtt{a[i][j-2]}, \quad \mathtt{a[i+1][j-2]}\} \\
\} & & &
\end{array}
$$

Note that the data items sent during $\mathtt{New\_Phase}_2$ are not received during $\mathtt{New\_Phase}_1$. Thus, all the input/output communications commands in $\mathtt{New\_Phase}_2$ can be combined with the commands of a previous phase ($\mathtt{New\_Phase}_1$, in this case). The following is the simple algorithm that performs this kind of optimization:

```
optimize_comp_unit(h,k)
{
     for (ind = d to 1 step -1)
          if (The data sent in New_Phase_ind are received
              during New_Phase_t, t < (ind - 1)) then
                         Combine New_Phase_ind and New_Phase_t+1
          else if (The data sent in New_Phase_ind are not
                    received in any phase) then
                         Combine New_Phase_ind and New_Phase_1
          endif
}
```

Thus, applying this algorithm to the two communication phases above, we remove $\mathtt{New\_Phase}_2$, and obtain the following $\mathtt{New\_Phase}_1$:

$$
\begin{array}{llllll}
\multicolumn{6}{l}{\mathtt{New\_Phase}_1 =} \\
\{ & \mathcal{C}_{h-1,k} & ! & \{\mathtt{a[i][j]}, & \mathtt{a[i][j+1]}, & \mathtt{a[i+1][j]}, \quad \mathtt{a[i+1][j+1]}\}, \\
 & \mathcal{C}_{h,k+1} & ! & \{\mathtt{a[i+1][j]}, & \mathtt{a[i+1][j+1]}, & \mathtt{a[i][j]}, \quad \mathtt{a[i][j+1]}\}, \\
 & \mathcal{C}_{h+1,k} & ! & \{\mathtt{a[i][j+1]}, & \mathtt{a[i][j]}, & \mathtt{a[i+1][j+1]}, \quad \mathtt{a[i+1][j]}\}, \\
 & \mathcal{C}_{h,k-1} & ! & \{\mathtt{a[i][j]}, & \mathtt{a[i][j+1]}, & \mathtt{a[i+1][j]}, \quad \mathtt{a[i+1][j+1]}\}, \\
 & & & & & \\
 & \mathcal{C}_{h-1,k} & ? & \{\mathtt{a[i-1][j]}, & \mathtt{a[i-1][j+1]}, & \mathtt{a[i-2][j]}, \quad \mathtt{a[i-2][j+1]}\}, \\
 & \mathcal{C}_{h,k+1} & ? & \{\mathtt{a[i+2][j]}, & \mathtt{a[i+2][j+1]}, & \mathtt{a[i+3][j]}, \quad \mathtt{a[i+3][j+1]}\}, \\
 & \mathcal{C}_{h+1,k} & ? & \{\mathtt{a[i][j+2]}, & \mathtt{a[i][j+3]}, & \mathtt{a[i+1][j+2]}, \quad \mathtt{a[i+1][j+3]}\}, \\
 & \mathcal{C}_{h,k-1} & ? & \{\mathtt{a[i][j-1]}, & \mathtt{a[i][j-2]}, & \mathtt{a[i+1][j-1]}, \quad \mathtt{a[i+1][j-2]}\} \\
\} & & & & &
\end{array}
$$

5 Analytic evaluation of performance

In this section we deal with the performance prediction of our implementation for the `geometric` paradigm. For the sake of brevity, we only consider the case of minimum granularity, i.e. the case in which each virtual worker is mapped onto a distinct node. The analytic formula used to predict the performance for coarse-grain implementations can easily be derived from that of the fine-grain implementation.

The analytic formula that models the execution time of our implementation is thus based upon the knowledge of

- the time taken to exchange data packets of a given size between neighboring nodes, and
- the time taken for the execution of the sequential code which computes the function $\mathcal{F}$.

Therefore, without considering the I/O time, spent for loading/unloading the various processing nodes with the domain/codomain data, the total execution time is:

$$d \cdot (Dt) + t_w \tag{3}$$

where

- d is the index of the symmetrical region $\mathcal{R}^{(d)}$ covering $\mathcal{I}_{i,j}$;
- D is the size of each packet to be exchanged between neighbors. The unit of measurement of D is the *flit* [6], i.e., the smallest unit of information that can be transmitted over each link;
- t is the time to transmit each flit. If $\mathcal{B}$ is the total bandwidth of any link in each direction, then $t = 1/\mathcal{B}$;
- t_w is the time to execute the sequential code to compute $\mathcal{F}()$.

Note that Formula (3) only holds if all the links are full-duplex, i.e. if communications on distinct links and directions can proceed independently of each other. A slightly different formula must be used if the links are assumed to be half-duplex.

Note that the overheads for startup communications do not appear in (3), as they are regarded as being included in t_w. In any case, in an RMS-PM these overheads are negligible. In fact, since messages are only exchanged between neighbors, no information on routing must be provided along with packets, and the communication subsystem does not have to perform complex messages handling.

Comparing mesh RMS-PMs with other PMs with routing units The implementation of the `geometric` paradigm on a mesh-connected RMS-PM is optimal, even if compared with the time taken by architectures which provide more powerful communication mechanisms. To show this, we have to derive that $d\frac{D}{\mathcal{B}} = d(Dt)$ is both a lower and an upper bound for T_{comm}, where T_{comm} is the time taken by each $W_{i,j}$ to receive all the data included in $\mathcal{I}_{i,j}$. In fact, since each packet is composed of D flits, and each node needs to receive d packets per incoming link, the time taken by each $W_{i,j}$ to receive all the data included in $\mathcal{I}_{i,j}$ is

$$\Theta(d\frac{D}{\mathcal{B}})$$

Note that Formula (3), which models our implementation for the `geometric` paradigm, states that the time to receive the same d packets is $o(d \cdot Dt) = o(\frac{dD}{\mathcal{B}})$, so both the upper and the lower bounds are identical.

Let us now consider a PM with routing units, which is able to communicate a packet as a set of flits transmitted in pipeline on multiple links. We know that, in general, non-local communications on this PM have reduced latency than the same kind of communications on an RMS-PM. In fact, to perform a non-local communication, which has to cross n physical links, an RMS-PMs has to exploit store-and-forward routing mechanisms. Thus, if a packet composed of D flits must be transmitted over n links, this communication takes $d\frac{D}{\mathcal{B}}$. Conversely, the same non-local communication on a PM with routing units, which adopt *virtual cut-through* or *wormhole* policy, takes a time $\frac{D}{\mathcal{B}} + n$, provided that no conflicts occur.

Thus, suppose that n links is the greatest distance of any worker in $\mathcal{R}_{i,j}^{(d)}$ from $W_{i,j}$, where, by constructive hypothesis of $\mathcal{R}_{i,j}^{(d)}$, $n \leq d$. By using a routing unit, and in the absence of conflicts, the time taken by this communication is $\frac{D}{\mathcal{B}} + n$. In any case, we cannot generalize, and deduce that the same time is enough to communicate all the data items allocated to the workers in $\mathcal{R}_{i,j}^{(d)}$ to $W_{i,j}$. In fact, the lower bound is still $\Theta(d\frac{D}{\mathcal{B}})$. The reasons for this loss of performance derives from the network conflicts, even when messages are delivered by routing units. In fact, for this kind of paradigm, all processing nodes, simultaneously, try to send and receive messages on all the physical links of the architecture. Also the adoption of *one-to-many* mechanisms for communication, which would seem to be more suitable for solving the dependencies of the `geometric` paradigm, cannot improve the lower bound.

6 Conclusion

We have presented a general algorithm to statically compile communications into the parallel code that implements a `geometric` paradigm on top of a mesh-connected RMS-PMs. These kinds of *data-parallel* paradigms are used, above all, to program image-processing problems, and, in general, problems where data, structured as a two-dimensional array, can be partitioned and allocated to the nodes of a mesh, but some dependencies exist between the corresponding computations.

The compiling algorithm presented in this paper only works if the dependencies of each node from data allocated elsewhere are *regular*. The meaning of regular dependencies is related to the topological location of dependent nodes on the mesh. In fact, our algorithm works if dependent nodes are arranged as particular *symmetrical regions*, which are *regular* denotations of some nodes of the mesh with respect to a generic node $\mathcal{N}_{i,j}$.

By using the terminology of [12], the forms of non-local communications corresponding to a symmetrical region can be classified as *uniform* and *aggregate*. In fact, the relationships between senders and receivers are uniform because they do not vary in dependence of their locations on the mesh, while each node needs to receive a collection of data items grouped in a given symmetrical region. With respect to the communication routines presented in [12], the algorithm presented in this paper can be considered as an optimal technique to implement communication routines for particular reference patterns, i.e. patterns that match, or are strictly included in, a given symmetrical region.

Our technique is aimed at obtaining *compile-time* resolution of dependencies, in the same way as [13] does. Moreover, our compiling technique takes into account non-local dependencies, by transforming them into sequences of local communications. Since our technique tries to solve dependencies at compile-time, it is more effective than the Single Program Multiple Data (SPMD) technique [12,10], where, at run-time, processes decide whether to send or receive messages in order to perform every step of their programs.

We have also shown the optimality of the message-passing program obtained by our compiling technique, even when compared with implementations which rely on systems equipped with routing units.

Acknowledgments

We would like to thank M. Danelutto, R. Di Meglio, M. Mackey, and S. Pelagatti for the helpful discussions and hints. Finally, we acknowledge the Hewlett Packard Laboratories - Pisa Science Center, which partially supported this research.

References

[1] A. Agarwal. Limits on Interconnection Network Performance. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):398–412, October 1991.

[2] R. Allen and K. Kennedy. Automatic Translation of FORTRAN Programs to Vector Form. *ACM Transactions on Programming Languages and Systems*, 9(4):491–542, October 1987.

[3] B. Bacci, M. Danelutto, S. Orlando, S. Pelagatti, and M. Vanneschi. P^3L: a Structured High-level Parallel Language and its Structured Support. Technical Report HPL-PSC-93-55, Hewlett Packard Laboratories, Pisa Science Center (Italy), 1993. Submitted for publication.

[4] G. Callagan and K. Kennedy. Compiling Programs for Distributed-Memory Multiprocessors. *The Journal of Supercomputing*, 2(2):151–170, October 1988.

[5] W. Dally. Performance Analisys of k-ary n-cube Interconnection Networks. *IEEE Transactions on Computers*, 39(6):775–785, June 1990.

[6] W. Dally. Network and Processors Architecture for Message-Driven Computers. In R. Suaya and G. Bithwistle, editors, *VLSI and Parallel Computation*, chapter 3, pages 140–222. Morgan Kaufmann Publisher, Inc. - San Mateo, California, 1991.

[7] M. Danelutto, R. Di Meglio, S. Orlando, S. Pelagatti, and M. Vanneschi. A Methodology for the Development and the Support of Massively Parallel Programs. *Future Generation Computer Systems J.*, 8:205–220, 1992.

[8] G.C. Fox et al. *Solving Problems on Concurrent Processors*. Prentice-Hall International, 1988.

[9] A.J.G. Hey. Experiments in MIMD Parallelism. In *Proc. of Int. Conf. PARLE '89*, pages 28–42, Eindhoven, The Netherlands, June 1989. LNCS 366 Spinger-Verlag.

[10] S. Hiranandani, K. Kennedy, and C. Tseng. Compiling Fortran D for MIMD Distributed-Memory Machines. *Communications of the ACM*, 35(8):67–80, August 1992.

[11] H.T. Kung. Computational Models for Parallel Computers. In C.A.R. Hoare Series editor, editor, *Scientific applications of multiprocessors*, pages 1–17. Prentice-Hall International, 1988.

[12] J. Li and M. Chen. Compiling Communication-Efficient Programs for Massively Parallel Machines. *IEEE Transactions on Parallel and Distributed Systems*, 2(3):361–383, July 1991.

[13] A. Rogers and K. Pingali. Process Decomposition through Locality of Reference. In *Proc. of SIGPLAN '89 Conf. on Prog. Lang. Design and Impl.*, pages 69–80, Portland, Oregon, 1989.

[14] M. Wolfe. Multiprocessor Synchronization for Concurrent Loops. *IEEE Software*, pages 34–42, January 1988.