Compiling Geometric Paradigms through Local Communications

Salvatore Orlando and 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.

Keyword Codes: C.1.2; D.1.3; D.2.2
Keywords: Processor architectures, Multiple Data Stream Architectures (Multiprocessors); Programming techniques, Concurrent Programming; Software Engineering, Tools and Techniques

1. INTRODUCTION

Distributed memory, MIMD systems (DM-MIMD), with regular direct networks and low dimensional degree nodes [1, 3, 5], have been recognized as the parallel architectures that can effectively exploit large-scale parallelism. However, 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 harness such powerful systems with scalable applications. In particular, the scalability of applications is strictly related to the way in which communications are employed.

Some researchers have proposed to put some restricted/structured forms of communication into a *communication support*, and to adopt programming models and/or compiling techniques that use this support. In [7], for example, structured communication primitives are introduced, and in [9], these structured primitives are employed by novel compiling techniques. In this article, we present a static technique that does not use this support, but directly produce an optimized code that only exploits *local* communications between directly connected nodes (also called *basic* communications, since more complex forms can be obtained as combination of them). This technique works for a specific *data-parallel paradigm*, namely the `geometric` one. It analyzes some information about the dependencies between the data-parallel tasks, and, given a data distribution and a computation rule [12], translates these dependencies in terms of local communication mechanisms. The dependencies can be obtained from programs written in a structured high-level language with explicit data-parallel constructs, e.g. `forall`, or from sequential programs by means of a parallelizing compiler [4, 9, 11]. The advantage in using our approach is a conflict-free exploitation of the communication media, so that we can statically derive the exact cost of the communication part of each implementation. Therefore, we can derive an *analytical performance model* that can be exploited by a compiler to decide, for each application and for a given target architecture, the best implementation in terms of resource allocation, parallelism degree, and the task granularity. In other papers we have showed that paradigms other than the *geometric* one can be efficiently implemented by using the same approach [6, 2].

2. THE GEOMETRIC PARADIGM

In this paper we refer to the following definition of the `geometric` paradigm, seen as a function that transforms between two *bi-dimensional* arrays:

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

where $Type_{in}[N][M]$ is the type of the input *domain* $\mathcal{I}$, and $Type_{out}[N][M]$ is the type of the output *codomain* $\mathcal{O}$. The *data-parallel* computation associated with this paradigm concerns the partitions $\mathcal{I}_{i,j}$ and $\mathcal{O}_{i,j}$ of the $\mathcal{I}$ and $\mathcal{O}$, respectively, i.e.

$$\forall i \in [0, \ldots, N-1], \quad j \in [0, \ldots, M-1]: \quad \mathcal{F}(\mathcal{I}_{i,j}) = \mathcal{O}_{i,j}$$

where $\mathcal{F}$ is the data-parallel function. Moreover, the following properties must hold:

$$\mathcal{I} = \bigcup_{\forall i,j} \mathcal{I}_{i,j} \qquad \mathcal{O} = \bigcup_{\forall i,j} \mathcal{O}_{i,j} \qquad \mathcal{O}_{i,j} \bigcap_{\substack{\forall i,i',j,j' \\ (i,j) \neq (i',j')}} \mathcal{O}_{i',j'} = \emptyset$$

Note that, since the various $\mathcal{I}_{i,j}$'s may overlap, the naive implementation in which, for each (i,j), we allocate a "virtual" processing node $N_{i,j}$ that receives the whole partition $\mathcal{I}_{i,j}$, would produce data replication. To avoid this, the partition $\mathcal{I}'_{i,j}$ to be assigned to $N_{i,j}$ must be smaller than $\mathcal{I}_{i,j}$, i.e. $\mathcal{I}'_{i,j} \subseteq \mathcal{I}_{i,j}$, and, for $(i,j) \neq (i',j')$, $\mathcal{I}'_{i,j} \cap \mathcal{I}'_{i',j'} = \emptyset$. In this case, since the rest of the domain partition, i.e. $\mathcal{I}_{i,j} - \mathcal{I}'_{i,j}$, is allocated to other processing nodes, these nodes have to communicate with each other to gather all the data items that are needed for the execution of the function $\mathcal{F}$.

For example, the following program, whose syntax resembles the HPF one, can be classified as a `geometric` paradigm that takes the array `a(1000,1000)`, and produces the array `b(1000,1000)`.

```
INTEGER a(1000,1000), b(1000, 1000)
...
FORALL ( i=3:987, j=3:987)
    b(i) = F( a(i,j), a(i-1,j), a(i-2,j),
              a(i+1,j), a(i+2,j), a(i,j-1), a(i,j-2),
              a(i,j+1), a(i,j+2) )
END FORALL
```

In this example, `F()` corresponds to our data-parallel function $\mathcal{F}$, $\mathcal{I}_{i,j} = \{\mathtt{a}(\mathtt{h},\mathtt{g}) \mid \mathtt{h} = \mathtt{i}-2 : \mathtt{i}+2, \mathtt{g} = \mathtt{j}-2 : \mathtt{j}+2\}$, $\mathcal{O}_{i,j} = \{\mathtt{b(i,j)}\}$, and $\mathcal{I}'_{i,j} = \{\mathtt{a(i,j)}\}$. For the sake of simplicity, we have not considered the computation of `b(1000,1000)` in correspondence of the borders, i.e. for `i,j=1:2` and `i,j=988:1000`.

3. COMPILING DEPENDENCIES ON A MESH-CONNECTED MACHINE

The first step of our technique produces an implementation for a virtual machine, namely a $N \times M$ mesh, where the data parallel computations refer to data partitions with the finest granularity. Thus, if $\mathcal{I} = \mathtt{a[N][M]}$ and $\mathcal{O} = \mathtt{b[N][M]}$, we have $\mathcal{I}'_{i,j} = \mathtt{a[i][j]}$ and $\mathcal{O}_{i,j} = \mathtt{b[i][j]}$. The technique has to arrange the (local) communications between the *fine-grain processes* mapped onto the various nodes $N_{i,j}$ of the mesh in such a way that (1) each process can receive its domain partition $\mathcal{I}_{i,j}$ within a *predictable time interval*, and (2) *no conflicts* arise between messages to be transmitted over the same link. Once the code for these processes is generated, the further 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.

3.1. Implementation on a mesh of virtual nodes

In this section we illustrate the first step of our technique, which produces the fine-grain processes $W_{i,j}$, called *workers*, to be mapped onto distinct nodes $N_{i,j}$ of a virtual mesh. The technique assumes that the communication links in the mesh are *bidirectional*, i.e. distinct messages can be transmitted over a link in both the directions.

The computation performed by each $W_{i,j}$ can be thought as subdivided into *computational phases*, during which only local computations are carried out, and *communication phases*, during which every $W_{i,j}$ exchanges messages with its neighbours, i.e. the workers mapped onto neighbouring nodes. Since each $W_{i,j}$ has 4 neighbours, during a *communication phase* $W_{i,j}$ can send 4 messages to its neighbours, and can receive 4 messages from the same neighbours. It is worth noticing that these communication phases synchronize the execution of the parallel workers.

Our technique derives the communication phases of each $W_{i,j}$ from the knowledge of both $\mathcal{I}_{i,j}$ and $\mathcal{I}'_{i,j}$, provided that the shape of every partition $\mathcal{I}_{i,j}$ has particular features. In order to formally specify the shapes allowed for the partitions, we need the following definition.

Definition 1 *For each pair (h,k) 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 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 $W_{i,j}$. The ordering of the four workers in $\mathcal{S}$ is defined as* circular clockwise*, because this is the order in which we find the workers when moving* clockwise *w.r.t. the center of symmetry $W_{i,j}$. Note that the four* neighbours *of the worker $W_{i,j}$ correspond to a particular set of symmetrical workers w.r.t. $W_{i,j}$, and whose* distance *from $W_{i,j}$ is 1.* □

Below we give the definition of *symmetrical region*, which characterizes a subset of workers. Our compiling technique works for those `geometric` paradigms where the input partition $\mathcal{I}_{i,j}$ has been allocated to a set of workers included in a particular symmetrical region.

Definition 2 *Given a subset $\mathcal{R}^{(d)}$ of workers, $d \geq 0$, $|\mathcal{R}^{(d)}| = 4d+1$, we define $\mathcal{R}^{(d)}$ as a* symmetrical region *iff $\mathcal{R}^{(d)}$ can be built by using the following recursive definition*

1. $\exists W_{i,j} s.t.\ \mathcal{R}^{(0)} = \{W_{i,j}\}$;

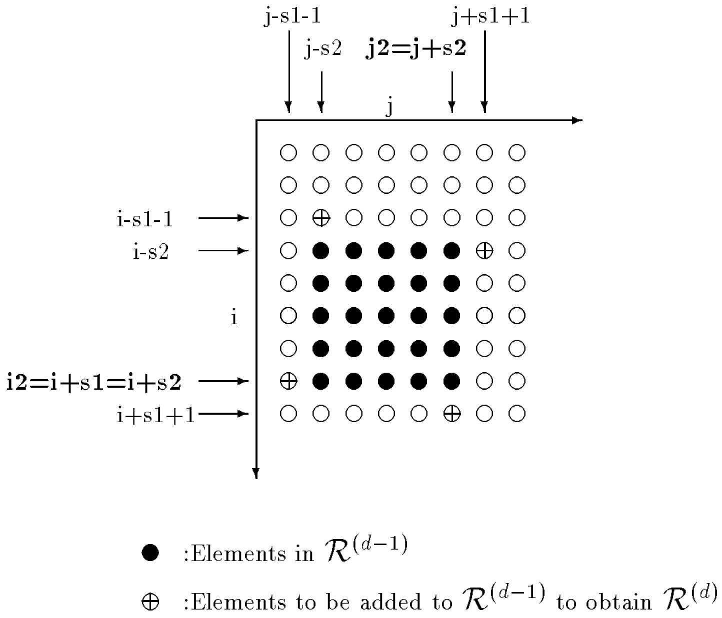

Figure 1. The symmetrical regions $\mathcal{R}^{(d-1)}$ and $\mathcal{R}^{(d)}$.

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. 1), s.t.* $\mathcal{S} \cap \mathcal{R}^{(n-1)} = \emptyset$;
- *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 neighbouring;*
- *if* d *is the* distance *of any worker in* $\mathcal{S}$ *from* $W_{i,j} \in \mathcal{R}^{(0)}$ *(see Def. 1), all the other workers* $W \in \mathcal{R}^{(n-1)}$ *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 $\mathcal{R}_{i,j}^{(d)}$ *as the symmetrical region* $\mathcal{R}^{(d)}$ *that has* $W_{i,j}$ *as its center of symmetry.* □

Hence, suppose that a generic $\mathcal{I}_{i,j}$ is allocated to a given $\mathcal{R}_{i,j}^{(d)}$. In this case our technique can generate the *local communication phases* between the various workers. The following Theorem shows the soundness of our technique, stating an *upper bound* on the number of *communication phases* that each worker needs to receive the corresponding partition $\mathcal{I}_{i,j}$.

Theorem 1 *Suppose that, for its computation, every worker* $W_{i,j}$ *needs the data items allocated to the workers belonging to* $\mathcal{R}_{i,j}^{(d)}$. *Thus, every worker* $W_{i,j}$ *can know all these data items within,* at most, $\frac{|\mathcal{R}^{(d)}|-1}{4} = \frac{(4d+1)-1}{d} = d$ communication phases

Proof by induction:
d=1:

Table 1
Ordering of two related sets of symmetrical workers

Ordering	New worker	Neighbour 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	$W^{(s+3)\mathrm{mod}4} = W_{i-s_2,j+s_1+1}$	$W_{i,j+1}$

The only region that has $W_{i,j}$ as its center of symmetry, and has cardinality $4d+1=5$ is $\mathcal{R}^{(1)}_{i,j} = \{W_{i,j}\} \cup \{W_{i\pm1,j}, W_{i,j\pm1}\}$. Since the worker $W_{i,j}$ is a *neighbour* of the other four workers in $\mathcal{R}^{(1)}_{i,j}$, it can receive, within one *communication phase*, all the data items allocated to them. Note that this is true for all the workers $W_{i,j}$, since, during a communication phase, each pair of neighbouring workers can exchange messages in both the directions.

d>1:
Suppose that the Theorem holds for $d-1$. 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 neighbour of $W_{i,j}$.

By hypothesis, each worker in $\mathcal{S}$ is a neighbour of a worker in $\mathcal{R}^{(d-1)}_{i,j}$ (see Def. 2). For symmetry, there must exist a worker $W^s \in \mathcal{S}$, $s \in \{0,\ldots,3\}$, s.t. W^s is a neighbour of $W_{h,k} \in \mathcal{R}^{(d-1)}_{i,j}$, where $h \geq i, k \geq j$. Thus, assume that $h = i + s_1$ and $k = j + s_2$, where $s_1, s_2 \geq 0$. Since W^s and $W_{h,k}$ are neighbouring, and the *distance* between W^s and $W_{i,j}$ must be greater than (or equal to) the *distance* of $W_{h,k}$ from $W_{i,j}$ (see Def. 2), we have that either $W^s = W_{h+1,k} = W_{i+s_1+1,j+s_2}$, or $W^s = W_{h,k+1} = W_{i+s_1,j+s_2+1}$.

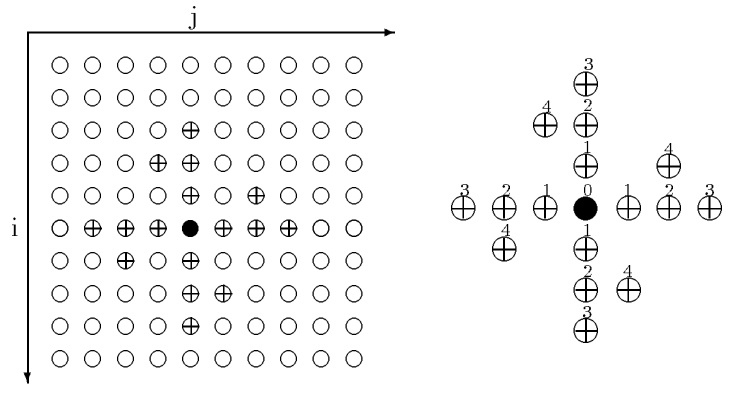

Figure 2. (a) A *symmetrical region* $\mathcal{R}^{(4)}_{i,j}$; (b) the enumeration of the workers in $\mathcal{R}^{(4)}_{i,j}$.

Now we prove the Theorem for the former case, i.e. $W^s = W_{h+1,k}$. By the same reasoning, you can prove the Theorem for the latter case. We have to show that $W_{i+1,j}$ knows the data item allocated to $W^s = W_{h+1,k} = W_{i+s_1+1,j+s_2}$. By inductive hypoth-

```
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_{h0,k0}, W_{h1,k1}, W_{h2,k2}, W_{h3,k3}} be the symmetrical set whose elements
                           are labelled with id_ph.  The indexing of this set corresponds to the circular
                           clockwise ordering.

                        2. Let N  =  {N0, N1, N2, N3} be the symmetrical set of the neighbours of W_{i,j}.
                           The indexing of this set corresponds to the circular clockwise ordering.

                        3. Find an element   W_{hi,ki}  ∈  S   s.t.  W_{hi,ki} also belongs to the region
                           R^(id_ph - 1), which has N0 as its center of symmetry.

                        4. for (ind = 0 to 3)
                                Input_Phase_{id_ph} = Input_Phase_{id_ph} ∪
                                        { N_ind ?  a[h_{(i+ind)mod4}] [k_{(i+ind)mod4}] } ^a
```

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

```
                }
}
```

Figure 3. An algorithm to generate the *input communication phases* of $W_{i,j}$.

esis, every worker $W_{i,j}$ knows the data allocated to $\mathcal{R}_{i,j}^{(d-1)}$. 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 in $\mathcal{S}$. Table 1 shows the neighbours of $W_{i,j}$ that know the data items allocated to the workers in $\mathcal{S}$. A further property, which will be used to generate the communication phases between the workers, concerns the ordering of the entries in Table 1, which corresponds to the *circular clockwise* one. □

3.2. An algorithm to generate the communication phases

Figure 2.(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 $W_{i,j}$ within, at most, four *communication phases*. The algorithm in Figure 3 generates these communication phase, using the enumeration induced by the recursive construction of $\mathcal{R}_{i,j}^{(4)}$ (see Figure 2.(b)). More precisely, the algorithm only generates the *input* communication phases of a generic $W_{i,j}$. The output commands, which complete the communication phases of each worker, are easily derived from them. For example, the communication phases relative to the region $\mathcal{R}_{i,j}^{(4)}$ in Figure 2.(b) are:

```
Phase1 ={ U ! a[i][j],   Phase2 ={ U ! a[i+1][j],   Phase3 ={ U ! a[i+2][j],   Phase4 ={ U ! a[i][j-2],
          R ! a[i][j],             R ! a[i][j-1],             R ! a[i][j-2],             R ! a[i-2][j],
          D ! a[i][j],             D ! a[i-1][j],             D ! a[i-2][j],             D ! a[i][j+2],
          L ! a[i][j],             L ! a[i][j+1],             L ! a[i][j+2],             L ! a[i+2][j],

          U ? a[i-1][j],           U ? a[i-2][j],             U ? a[i-3][j],             U ? a[i-1][j+2],
          R ? a[i][j+1],           R ? a[i][j+2],             R ? a[i][j+3],             R ? a[i+2][j+1],
          D ? a[i+1][j],           D ? a[i+2][j],             D ? a[i+3][j],             D ? a[i+1][j-2],
          L ? a[i][j-1]            L ? a[i][j-2]              L ? a[i][j-3]              L ? a[i-2][j-1]
        }                        }                          }                          }
```

where $\mathtt{U} = W_{i-1,j}$, $\mathtt{R} = W_{i,j+1}$, $\mathtt{D} = W_{i+1,j}$, and $\mathtt{L} = W_{i,j-1}$.

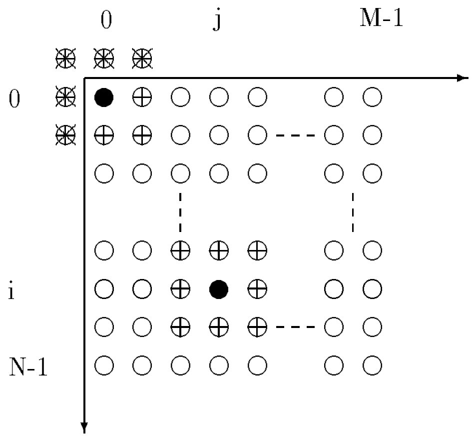

Figure 4. Symmetrical regions falling out of the boundaries.

In [10] we illustrate some optimization techniques that reduce the number of communication phases produced by concatenating multiple short messages to be sent to the same worker. For example, this kind of optimization can be also applied to the four phases shown above. 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`.

Once the various communication phases $\{\texttt{Phase}_\texttt{i} \mid \texttt{i} \in [1, \ldots, d]\}$ are generated, the compiler produces the following code for a generic worker $W_{i,j}$:

```
virtual_worker(i,j)
{
    Allocate memory for I_{i,j} and O_{i,j}
    Receive the domain partition I'_{i,j}
    for (ind = 1 to d)
        do the communication phase Phase_{ind}
    O_{i,j} = F(I_{i,j})
}
```

3.3. Boundary conditions, and regions fitting input partitions

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 generic region $\mathcal{R}^{(2)}_{i,j} = \{W_{i,j}, W_{i\pm1,j}, W_{i,j\pm1}, W_{i\pm1,j\pm1}\}$. Two instances of this region are shown in Figure 4, with the center of symmetry denoted by black circles. A problem arises when the center is close to the boundaries of the $N \times M$ mesh. For example, Figure 4 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 3 for $W_{0,0}$ are not correct. To solve this problem, once the communication phases of a generic worker $W_{i,j}$ are built, some of the input/output commands that make up the communication phases of some workers must be removed. The algorithm to remove these input/output commands is simple, and is presented in [10]. Informally, the input/output commands that must be removed are either those whose partner process would be identified by a pair of indexes outside the range allowed, or those in which the data item sent to (or received from) the partner process would be characterized by subscripts that are outside the range allowed.

Another similar problem arises from partitions $\mathcal{I}_{i,j}$ that do not exactly correspond to any symmetrical region. Suppose, for example, that $\mathcal{I}_{i,j}$ is the i^{th} row, and the j^{th} column of the array `a[N][M]`. The `geometric` implementation of the *matrix multiplication* exploits, for example, this kind of dependency. The smallest symmetrical region $\mathcal{R}_{i,j}^{(d)}$, $d = max(N-1, M-1)$, which covers the whole $\mathcal{I}_{i,j}$ is defined by:

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

Note that, for any pair i, j, some of the nodes in $\mathcal{R}_{i,j}^{(max(N-1,M-1))}$ fall out of the $N \times M$ mesh. Hence, once the communication phases of a generic $W_{i,j}$ are generated by means of the algorithm in Figure 3, then the unnecessary input/output commands must be removed from the the communication phases of each specific worker.

3.4. Grain optimization

We have discussed the first step of our compiling technique, which generates a mesh-connected network of fine-grain workers. The step of the technique presented in this section is concerned with the grain optimization strategy, according to which these workers are mapped onto a real mesh-connected machine. In particular, the technique determines how the granularity of the task assigned to each processing node can be augmented. The grain optimization, or distribution, strategy employed is *blocking* [8, 9], according to which a block of $H \times B$ contiguous workers are mapped onto a node $N_{h,k}$. Hereafter, these blocks will be called *computational units*, and will be denoted by $\mathcal{CU}_{h,k} = \{W_{i,j} \mid h \cdot H \leq i < (h+1) \cdot H, \quad k \cdot B \leq j < (k+1) \cdot B\}$, where $H, B > 0$, $N \bmod H = 0$, and $M \bmod B = 0$.

Our technique derives a single process for each $\mathcal{CU}_{h,k}$. From the functional point of view, this single process is equivalent to multiprogramming each node with the workers included in $\mathcal{CU}_{h,k}$, but is more efficient. The single process will be:

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 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`$_{ind}$, and remove from `New_Phase`$_{ind}$ all the input/output commands between virtual workers which belong to $\mathcal{CU}_{h,k}$.

2. Combine all the input/output commands from/to virtual workers which are allocated to the same computational unit. These new input/output command will have to refer to computational units in $\{\mathcal{C}_{h\pm1,k}, \mathcal{C}_{h,k\pm1}\}$.

3. Bring forward some of the new communication phases obtained by the previous steps. This must be done if some of the data items sent during $\texttt{New\_Phase}_t$ are not received during $\texttt{New\_Phase}_{t-1}$.

4. ANALYTIC EVALUATION OF PERFORMANCE

This section deals with the *performance models* thart can be associated with the implementations of a `geometric` paradigm. The compiler can exploit these models to predict the performance of the different implementations that it can generate, i.e. implementations characterized by different granularity. 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 coarser-grain cases can be easily derived from it.

The analytical models are based based upon the knowledge of the time taken to exchange data packets of a given size between neighbouring nodes, and the time taken for the execution of the sequential code which computes the function $\mathcal{F}$.

Hence, consider a `geometric` paradigm, for which we have determined the smallest symmetrical region $\mathcal{R}^{(d)}$ which covers $\mathcal{I}_{i,j}$. Our algorithm generates a worker executing d communication phases, during which packets of size D are exchanged between neighbouring workers. The unit of measurement of D is the *flit*, i.e., the smallest unit of information that can be transmitted over each link. Without considering the I/O time, spent for loading/unloading the various processing nodes with the domain/codomain data, the total execution time is summarized by the following formula:

$$d \cdot (Dt) + t_w \tag{1}$$

where t is the time to transmit each flit,and t_w is the time to execute the sequential code to compute $\mathcal{F}()$. If $\mathcal{B}$ is the total bandwidth of any link in each direction, then $t = 1/\mathcal{B}$; Note that (1) 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. Moreover, note that the overheads for startup communications do not appear in (1), as they are regarded as being included in t_w. It is worth noticing that the parameters d, D, t, and t_w depends on both the target machine and the specific application.

A final remark concerns the optimality of our technique. The implementations generated, even if use local communications only, exploit the whole bandwidth of the communication network. In other words, $d(Dt) = d\frac{D}{\mathcal{B}}$ 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 $\mathcal{I}_{i,j} - \mathcal{I}'_{i,j}$, whose size is $4Dd$.

5. CONCLUSIONS

We have presented a general algorithm to statically compile local communications into the parallel code that implements a `geometric` paradigm on top of a mesh-connected DM-MIMD. This kind of *data-parallel* paradigm is used to program problems where data, structured as a two-dimensional array, can be statically partitioned and allocated to the processes mapped onto a mesh, but some dependencies exist between these processes. The compiling algorithm presented in this paper only works if the dependencies of each process from data allocated elsewhere are *regular*. The meaning of regular dependencies is related to the topological location of dependent processes on the mesh.

By using the terminology of [9], the forms of non-local communications used to solve these regular dependencies 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*.

We have also shown the optimality of the message-passing program obtained by our compiling technique, even when compared with implementations which rely on more complex hardware/software supports for communications.

Acknowledgements

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 staff at the Pisa Science Center.

REFERENCES

1. A. Agarwal. Limits on Interconnection Network Performance. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):398–412, October 1991.
2. 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 to "Concurrency: Practice and Experience" – In revision.
3. G. Bilardi and F.P. Preparata. Horizons of Parallel Computation. Technical Report CS-93-20, Dept. of Computer Science - Brown University, May 1993.
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. 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.
7. G.C. Fox et al. *Solving Problems on Concurrent Processors*. Prentice-Hall International, 1988.
8. 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.
9. 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.
10. S. Orlando and M. Vanneschi. Compiling Data-Parallel Paradigms through Basic Communication Mechanisms. Technical Report HPL-PSC-93-56, Hewlett Packard Laboratories, Pisa Science Center (Italy), 1993.
11. M. Wolfe. Multiprocessor Synchronization for Concurrent Loops. *IEEE Software*, pages 34–42, January 1988.
12. H.S. Zima and B.M. Chapman. Compiling for Distributed-Memory Systsssems. *Proceeding of the IEEE*, pages 264–287, February 1993.