Abstract—Parallelization methodologies allow to automate the process of designing optimal processor arrays based on mathematical representations of the algorithm to be implemented. In this work, an optimized multiprojection approach based on the Polytope model is proposed as well as an automated way for getting the scheduler and the allocator vectors. Using a recurrence equations representation, three key criteria for choosing the characteristics of the final implementation are also proposed. As a case of study, the methodology is applied on a matrix-vector multiplication example. Results and relevance of the proposed methodology are finally discussed.

Index Terms—Data flow computing, Parallel architectures, Parallel machines, Parallel processing, Systolic arrays.

I. INTRODUCTION

Implementation of algorithms on hardware platforms presents multiple interesting points because the inherent capacity of hardware devices (FPGA, ASIC, etc) for parallel processing. One of the most important challenges when designing hardware architectures is to identify and to exploit the parallelism in the algorithm to be implemented. Such parallelism is limited by the data dependencies which have to be preserved in order to maintain the original behavior of the algorithm. Traditionally, design decisions are based on the experience of the designer, however, using a modeling technique it makes possible to explore the design space before the final implementation to improve some specific characteristic in design time, for instance: implementation area or throughput.

A modeling technique allows generating a processor array [1] from an algorithmic representation. Such techniques are in particular useful when the algorithm to be parallelized includes loops since loop elements are represented as points inside polyhedra [2-5]. It is possible to apply multiple transformations to the polyhedral representation in order to obtain an optimal execution time for each loop and even more, an optimal mapping between loops and processor elements to perform the original algorithm by using a design methodology.

A design methodology provides a direction vector called scheduler vector which indicates when each calculation will be executed [8-9], [11]. This second vector is called allocator vector; it provides a direction which is used as base in order to project points inside the polyhedron on physical processor elements generating a relationship between points and processor elements. The allocator vector is carefully selected since a bad decision could generate a final processor array extremely complex or to be prejudicial to the final performance. Despite the importance of the allocator vector, it is traditionally proposed by hand as in [12-13].

II. METHODOLOGY

The Polytope model [2], [14] is a mathematical tool that allows to generate efficient processor arrays for a specific algorithm. In the Polytope model, a program is modeled by a polytope, which is a finite convex set of some dimensionality with flat surfaces where every iteration in the original loop code is represented by one point (node) inside the polytope. The iterations set is called Iteration Space (IS). Figure 1 shows a characteristic polytope which represents an IS of two dimensions (i and j) including a data dependence between iterations [15-17]. Such dependence indicates that there exists some specific result in iteration 1,1 that will be required in iteration 2,1 and so on. This vector generates execution hyperplanes which are orthogonal to it and indicates that all nodes in the same hyperplane can be executed in parallel without affecting the normal behavior of the algorithm. Figure 1 shows the best possible scheduler (1,0) for this example.
The full set of data dependencies is represented by a dependence matrix $D$ as in (1).

$$D = \begin{bmatrix} \vec{d}_1, \vec{d}_2, \ldots, \vec{d}_i \end{bmatrix}$$

(1)

Where, $\vec{d}_i$ is the $i$-dependency in the dependence set. In case of Fig. 1, matrix $D$ is composed by only one vector. Using data dependencies and $IS$ characteristics, it is possible to find an optimum execution time for each node $I$ in the $IS$, i.e., an optimum execution order by using a linear integer program approach [6], [12], [18]. The result of the linear integer programming approach is a linear scheduler vector $\Lambda \in \mathbb{Z}^{1 \times n}$ or equivalently expressed as a function $\phi(I)$, which assigns an execution time to every point in the $IS$. Always it is possible to express the scheduler vector as a function $\phi(I)$ as shown in (2), the scheduler is called linear.

$$\phi(I) = \Lambda I \quad \forall I \in IS$$

(2)

In (2), the linear scheduler vector $\Lambda \in \mathbb{Z}^{1 \times n}$ is such that $\Lambda d_i \geq 1$ for all $d_i \in D$. This condition [19] is enough to ensure that all data dependencies are preserved and could be expressed as in (3).

$$\phi(I_2^{di}) - \phi(I_1^{di}) \geq 1$$

(3)

where $I_2^{di}$ and $I_1^{di}$ are the destiny and the source nodes in $IS$ of the data dependence $d_i$. In other words, if there exists a data dependence between nodes $I_2^{di}$ and $I_1^{di}$, the condition in (3) ensures that the node $I_2^{di}$ will be not executed before the node $I_1^{di}$ using the scheduler $\Lambda$.

After getting the scheduler vector we have to decide where iterations will be computed (allocation) [12], [18]. The first choice for allocating iterations to physical processors is to perform a projection [20-21]. Such projection should be carefully selected since a bad choice could be unfavorable to the final throughput. A bad decision could generate a final processor array requiring more communication lines of the processor array since it could be as expensive as the processors itself. Different vectors used as allocator or scheduler will require different number of communication lines. Since processor arrays present a high regularity level, it is enough to analyze one node and its relations with the neighbors to extrapolate the findings to all elements in the array.

1. Communication resources.

Communication between processors is an important aspect of the hardware architecture since it could be as expensive as the processors itself. Different vectors used as allocator or scheduler will require different number of communication lines. Since processor arrays present a high regularity level, it is enough to analyze one node and its relations with the neighbors to extrapolate the findings to all elements in the array.

2. Number of processors and total execution time.

The most important premises when designing hardware architectures are the total execution time and number of processors. As in point number 1, each vector used as scheduler or allocator will produce different number of processors and different execution times.

Details for points 1 and 2 are defined from the proposed methodology next.

- Communication resources

As mentioned in Section II, (3) could be used to determine if one specific data dependence is fulfilled using a scheduler vector. Additionally, (3) represents the time ($w_i$) when a datum will be transmitted from the execution point $I_1$ to the destiny $I_2$ [22] as shown in (5).

$$\phi(I_2^{di}) - \phi(I_1^{di}) = w_i$$

(5)

Where super index $d_i$ indicates the transmitted datum fulfills the data dependence $i$. Eq. (5) is important because in case of multiple data dependencies between nodes, for example $i$ and $j$, the best approach for improving the use of communication resources is to use...
the same communication channel for transmitting all the required information. This is only possible if all the information have to be transmitted in different times which are defined by the scheduler vector.

As mentioned in Section II, we propose to modify the original data dependence set in order to generate a second vector as new candidate to scheduler vector. Multiple options for selecting the scheduler vector allow selecting the best option in terms of use of communication resources.

The vector that allows sending datum for each data dependence at different moments allows simplifying the communication network between processors by reusing the communication channels. Such vector could improve the use of pipeline stages in the final hardware implementation.

In case of Fig. 1, applying (5), using the vector (0,1) as scheduler vector and the only data dependence, the result is 1, which indicates that the datum will be required in the execution of the next hyper plane so, it has to be sent at the end of the actual hyper plane execution.

- **Total execution time**

Scheduling function (2) indicates when every node in the IS should be processed by assigning an execution time to each node. By calculating the execution time of the first and the last nodes it is possible to know the total execution time under the proposed scheduler in 4 time units.

\[
\text{Execution time} = \phi(I_{last}) - \phi(I_{first}) + 1 \quad (6)
\]

With reference to Fig. 1, the last hyper plane will be executed at time 4 and the first hyper plane will be executed at time 1, using (6) it is possible to calculate the total execution time under the proposed scheduler in 4 time units.

- **Number of processor elements**

The size of the processor array is a key point to be considered since it has a direct relation with power consumption. For mapping some nodes of the IS point to the corresponding processor element under the allocator vector it is possible to use (6). However, in this case results are interpreted as processor IDs instead of execution times. In case of Fig. 1, the higher index for an PE is 4 and the lowest index for a PE is 1 then, the total number of required processors according to (6) is 4. It is important to note that the scheduler and the allocator could be different.

**III. Example. Matrix-vector multiplication**

In order to show the proposed methodology, we present an example using the matrix-vector multiplication. For representing the algorithm we propose to use a set of recurrence equations which allows a cleaner representation avoiding the overhead of a C representation as in [23][24]. Additionally, a C representation imposes an \textit{a priori} execution order which is avoided by using the recurrence equations representation. Equations set (7) shows the proposed representation using a set of recurrence equations [25] corresponding to the matrix-vector multiplication example.

1. \( a(i, j) = A_{ij} \quad 1 \leq i, j \leq N; \)
2. \( b(i, j) = \begin{cases} b & i = 1, 1 \leq j \leq N; \\ b(i-1, j) & 1 < i \leq N, 1 \leq j \leq N; \end{cases} \)
3. \( z(i, j) = a(i, j) \cdot b(i, j) \quad 1 \leq i, j \leq N; \)
4. \( c(i, j) = c(i, j-1) + z(i, j) \quad 1 \leq i, j \leq N; \)
5. \( c(i) = c(i, j) \quad j = N; \)

Where \( a(i, j) \) is the input variable for values of the matrix \( A \) and \( b(i, j) \) is used as the input variable for values of the vector \( B \) and as temporal variable to propagate the values of the vector \( B \). The temporal variable \( z(i, j) \) stores the result of intermediate multiplications, \( c(i, j) \) is used as accumulator and finally, \( c(i) \) is the vector result of the multiplication. Fig. 2 shows how all of these variables are related using \( N=4 \). Fig. 2 shows the data dependencies as connections between nodes too.

Equations set (7) includes two data dependencies, the first in line 2 and the second in line 4 which correspond to the two column components of the dependence matrix \( D \), shown in (8).

\[
D = \begin{bmatrix} \vec{d}_x & \vec{d}_y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (8)
\]

From (7) we can get the dimensionality of 2 \((i \text{ and } j)\) of the IS. Finally the size of each dimension corresponds to the interval where every dimension \((i \text{ and } j)\) is defined, in this case \(N\).

Linear programming problem requires conditions for the objective function. In this case the only condition is to respect data dependencies [3]. Such condition is represented by (3) which indicate that for any possible solution to the linear programming problem is not possible to execute calculations in node \((1,2)\) before or even in parallel with calculations in node \((1,1)\) if there is a data dependence from node \((1,1)\) to node \((1,2)\).

Solving the linear programming approach on the dependence matrix and the IS information we get the first proposed vector \((1, 1)\). As previously mentioned, the next step is to include one new data dependence to the original data dependence set. The new data dependence has to be orthogonal to the first obtained vector. Eq. (2) is used to get the \((-1, 1)\) vector which is included in the original data dependence set. Basically we transform the original IS shown in Fig. 2 into the shown in Fig. 3.

The linear programming approach is solved on the new data dependence set to get the vector \((2, 1)\). Now, we have two different vectors \((1, 1)\) and \((2, 1)\). Both vectors fulfill the original data dependencies and are congruent with the data flow. This means that any of both vectors could be used as scheduler vector or as allocator vector without affecting the normal behavior of the algorithm.
The second vector was generated using an extra component orthogonal to the first vector and finally they present different characteristics. We propose that the final decision about which one will be used as scheduler and which one as allocator depends on the following three aspects: Communication resources, number of processor elements and total execution time that are discussed next.

From Table I, values in column 2 indicate that using the vector (1, 1) as scheduler requires sending both data at the same time however, from column 2 using the vector (2, 1) as scheduler vector, data could be sent at different moments. At this point, it is better to select the vector (2, 1) as scheduler since such vector allows saving communication resources by reusing communication channels or even implementing pipeline stages. However, it is important to consider other aspects such as area and execution time which are revised next.

- **Number of processor elements**

Area resources are, in general, a key target when implementing some algorithms in hardware platforms because the available resources are limited. Fig. 4 shows the number of required processors by using as allocator the vector (1, 1) and the vector (2, 1) on different IS sizes. Using the vector (1, 1) as allocator requires less processors by a factor close to 1/3. According to Fig. 4 this factor is a constant, independently of the IS size in this specific problem.

- **Total execution time**

As mentioned in Section II, (2) provides a practical way for knowing the execution time of each node in the IS as well as the number of required physical processors. Since the IS of the proposed example is square, it is possible to reinterpret Fig. 4 as a time graphic which means that using vector (1, 1) as scheduler vector implies a faster processing time than using vector (2, 1) as scheduler vector. In this case, the difference between total execution times between both vectors is the same constant factor close to 1/3. Fig. 5 shows the final projection vector (1,1) and the scheduler vector (2,1). In this case, a compact implementation is preferred even at the expense of the required processing time which as before mentioned is greater by a constant factor close to 1/3. Using the vector (1, 1) as scheduler vector, the number of processor elements is minimal. Fig. 5 shows a graphical representation of the scheduler vector and the generated hyperplanes. As mentioned in Section I, all the nodes in the same hyperplane are executed in parallel. The time used in each hyperplane is the required time to complete the calculations in a single node.

<table>
<thead>
<tr>
<th>Data dependencies</th>
<th>Sending time using vector (1,1)</th>
<th>Sending time using vector (2,1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0,1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
If the interest is focused on reducing the execution time, it is possible to select the vector (2,1) as projection vector and the vector (1,1) as scheduler vector. Such approach provides a higher throughput at the expense of increased number of physical processors. Final processor array is shown in Fig. 6 Here, execution time is the smallest possible execution time limited only by data dependences however the communication is more complex when compared with using the vector (2,1) as scheduler vector. More complex communication network requires a more complex control system.

No matter the case, the proposed methodology prevents the manual selection of the projection vector as in [12][20] generating an optimized second vector which is congruent with the data flow. Having multiple choices for selecting the scheduler vector provides flexibility in defining the best projection vectors under certain criteria. Proposed aspects for selecting scheduler and allocator vectors are easily evaluated using (2). Methodology starts with a recurrence equations representation which avoids the overhead and the a priori order of a C representation.

IV. CONCLUSIONS

In this work, a parallelization methodology based on a multiprojection approach is proposed. The methodology provides two vectors and three criteria for deciding which vector is used as scheduler vector and which is used as allocator vector. The allocator vector is obtained according to an optimization process with a linear programming approach instead of using manual selection as in previous works. In this work, the original algorithm is represented as recurrence equations which avoids the overhead of using a C representation. The entire process could be fully automated since user intervention is not required. In order to demonstrate the proposed methodology, in the second part of the paper, the methodology was applied on the matrix vector multiplication example. Results on the proposed example demonstrate that the proposed methodology is successfully implemented allowing flexibility in design time and optimizing the execution time or the number of processor elements in the processor array.

REFERENCES


