Automatic Generation of Communications for Redundant Multi-dimensional Data Parallel Redistributions

Corinne Ancourt*, Teodora Petrisor§, Francois Irigoin*, and Eric Lenormand§
*MINES ParisTech
CRI, 35 rue Saint-Honoré, 77305 Fontainebleau, France
Email: {corinne.ancourt,francois.irigoin}@mines-paristech.fr
§THALES
Thales Research and Technology, 1 avenue Augustin Fresnel, 91767 Palaiseau Cedex
Email: {claudia-teodora.petrisor, eric.lenormand}@thalesgroup.com

***Abstract*—In this paper we concentrate on embedded parallel architectures with heterogeneous memory management systems combining shared and local memories, and more precisely we focus on efficient data communications between the various architecture parts. We formulate explicit data transfers in a polyhedral context and give several strategies for managing efficient communications for redundantly stored/read data. This allows automatic DMA-style code generation for a variety of data mappings onto parallel processing elements. Our approach is validated on a wide series of data redistribution examples linked with a domain-specific parallelisation framework developed in Thales, SpearDE. We give the solution for efficient data transfers mathematically as well as under the form of generated C code.**

I. Introduction

Since the power and frequency walls risen by transistor miniaturization and Moore's law have limited the performance one can get from computing architectures, parallelism has become again the obvious choice. This holds for very high performance computing over huge amounts of distributed data but also for embedded systems. Yet this did not quite solve the problem of getting the maximum performance effortlessly, but rather brought with it a huge exploration space for architecture and application design and adequacy.

The nature of applications in terms of intrinsic parallelism, data locality, computation type is equally diverse. There is arguably no one-size-fits-all solution, and therefore architectural heterogeneity is privileged for getting the most in terms of performance and/or energy. Some common examples of such embedded or high-performance architectures are general purpose processors (GPPs) coupled with various accelerators such as GPUs [1], FPGAs [2], DSPs [3], many-core platforms such as CELL-BE [4], ST STHORM [5] etc. The multi-processor components of these platforms will efficiently handle parallelisable parts of an application, assuming they were transformed at the right granularity (task, data etc.) for the platform.

In addition to architecture heterogeneity in terms of processing elements, there is also memory heterogeneity, usually with large but slow shared memories combined with small but very fast local memories close to computations. This usually calls for explicit data transfers between all these memories and thus efficient communication algorithms. Transfer efficiency can be sought according to various criteria such as load balancing, redundancy removal, loop fusion, etc.

In this paper, we explicitly concentrate on optimizing data communications for multi-dimensional arrays already mapped onto heterogeneous parallel parts of a given architecture. The mapping operation may result in data replication inside different local memories either for load balancing purposes or for reducing the communication overhead between local and shared memories, whenever possible. But, while this redundancy is imposed by the explicit mapping, transferring the same element several times is inefficient or may even result in data coherency problems. Our goal is to generate consistent communication code with appropriate redundancy handling both at sender and receiver levels in the most general case, without prior assumptions on a given architecture topology. This approach is motivated by the need of a generic framework for fine-grain parallelisation of dataflow applications (mainly in the signal and image processing domains) onto a large variety of configurations. The optimization of application placement onto parallel cores is out of the scope of this paper. We concentrate on the automatic generation of the data redistributions between different, potentially hierarchical, parallel parts of a platform, in a multi-dimensional DMA (Direct Memory Access) style.

Signal/image processing applications offer some specificities that make them very suitable candidates for fine-grain parallelisation, due to the high amount of potential data parallelism. This is because operations are usually performed in a systematic manner on tiles of the multi-dimensional signals and thus the computations are easily partitioned. Moreover, recurrent operations, like filtering, involve neighbourhood processing (e.g. convolutions) and mapping these operations onto several distinct processing elements usually introduces a certain amount of data overlap (of the size of the convolution kernel for instance); border handling in image processing is another example where partitioned parts of an image will overlap, data being thus found in different memories, in a distributed architecture.

An efficient transfer of overlapping data in a source distribution involves a pre-processing step of redundancy elimination which can be also dependent on the desired data

redistribution at the target. We have defined two strategies for communication balancing between a source-distributed memory and a destination one (potentially having a different distribution). Via a polyhedral formulation we ensure correct automatic generation of data transfers between a source distribution and a destination one. In this paper we will give the solution for the efficient data transfer mathematically as well as under the form of generated C code.

This paper is organized as follows. The next section presents our motivations stemming from the requirements of an industrial design environment for parallelisable dataflow applications, as well as the positioning of our work with respect to the state of the art. Then we will give the polyhedral formulation for our problem and its linear algebra solution in Sections III, IV and V. In Sections VI and VII we present the different implemented optimizations. Section IX concludes the paper.

II. Context

As data volumes grow, so does the need for more efficient processing. As explained in the previous section, this is particularly true for the signal processing field where information from various sensors needs to be put together and the potential for parallelism is extremely high. Many applications such as beam-forming, RADAR adaptive filtering, SONAR processing, image analysis via different transforms, video coding, space-time block coding, etc. can often be modelled as dataflow graphs. Then different parts of these graphs can be mapped onto various accelerators, this leading to an explicit need for data communication in distributed memory systems. Our industrial framework relies on a generic approach allowing to address many different (usually embedded) architectures with streaming processing capabilities. The main difficulty consists in devising code generators that can efficiently handle data communications in a correct-by-construction manner for most scenarios of data-parallel mappings of multi-dimensional arrays onto locally SPMD [6] segments of heterogeneous platforms (both COTS and in-house ones).

A. Multidimensional arrays

In the general case, a signal can be seen as a multi-dimensional array with usually independently processed dimensions. For instance, an antenna array in RADAR can be described by a $3D$ array having as dimensions the antenna axis, the range axis and the pulse axis. A video signal is usually viewed as a $3D$ signal also, where the first two dimensions are the rows and the columns of each video frame and the third one is the temporal axis of the video. The possibility of processing signal dimensions independently is quite valuable for data parallelisation. Indeed the multi-dimensional array can be split onto one of the directions and the resulting blocks can be distributed between different processing elements in their local memories. Moreover, a signal can be subsampled during its different processing stages, like for instance when performing the Fast Fourier Transform or the fast wavelet transform, where filtering and subsampling stages are cascaded. Also, redundancy can occur, for instance, with oversampled filter bank transforms and in a data parallel context parts of the signal might be replicated into different memories for local processing purposes.

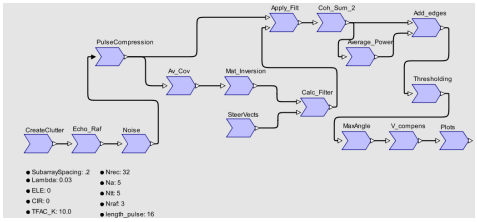

Fig. 1. Radar application modelling in SpearDE as a dataflow graph. The tasks (nodes in the graph) can be individually mapped onto parallel segments in the architecture.

In our industrial framework SpearDE [7], data flow applications are represented as directed acyclic graphs, as in Figure 1, in which the nodes are elementary operations iterated into nested loops allowing to consume the multi-dimensional input data distributions represented as the edges in the graph. Data handling in the application graph follows very similar rules to the ArrayOL formalism [8], especially in terms of data parallelism inside a node where a pattern (indivisible portion of the input array) is repeated until the entire array has been parsed. In this manner the iteration space can easily be partitioned at the mapping stage.

In order to perform co-design stages like application placement, here called mapping, code generation and performance evaluation, SpearDE simultaneously provides an abstract model of the target architecture, as the one given in Figure 2. Here, we highlighted the different levels of encapsulation of a multi-processor architecture on FPGA [9] that can be addressed via our parallelisation framework. In this example, the platform is composed of 4 clusters each having 3 tiles. Every tile consists of a CPU coupled with an accelerator. Different memory levels are available and potentially need to be accessed during a communication operation: the clusters communicate via a shared memory, the tiles inside the cluster equally access a cluster shared memory, while inside each tile there are local memories for the computing elements. One strength of this parallelisation framework is the potential of modelling very heterogeneous architectures via this abstract model, including non-COTS ones, as is the case in Figure 2. One limitation however is the fact that data transfers only include rectangular distributions at the moment and the need to extend this to generic (polyhedral) distributions motivate part of the work presented here. Note that the architecture topology in the example given in Figure 2 amounts to a multi-dimensional memory structure where the local memory in a processing element will be indexed inside a 4×3 array.

In the general case, when two linked nodes in the application are mapped onto different parts of an embedded many-core architecture, memory transfers are often explicit. In our framework this operation introduces supplementary nodes in the graph, as can be seen in Figure 3.

Inside the (mapped) graph nodes, data distributions have explicit dimensions, the mapping operation adding the information related to data parallelism, as in Figure 4. In this example we considered the frontier between the "orange" and the "green" segments in Figure 3. More specifically, we generate a communication between the 5 distributed memories - column "Arch0" in Figure 4 (corresponding to indexes $(0, 1)$ to $(1, 2)$

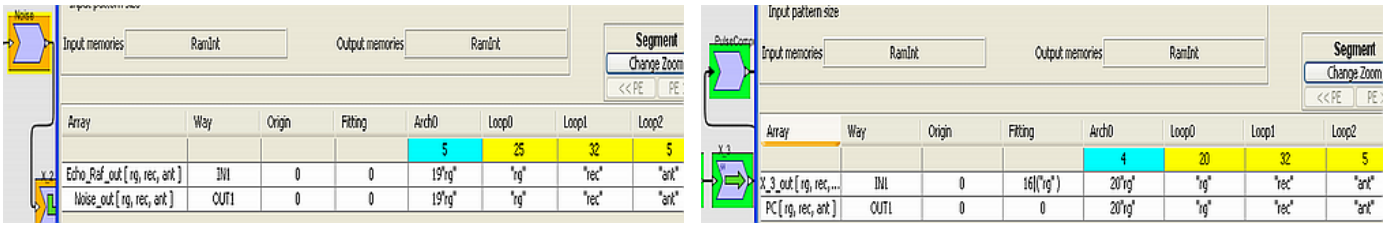

Fig. 4. The two ends of a communication: multi-dimensional signals inside source (left) and destination (right) nodes in the mapped dataflow graph. The source distribution is a 3D array split into 5 cubes of $25 \times 32 \times 5$ points each, on the dimensions named **rg**, **rec** and **ant**, respectively. The number of data cubes is given by the number of inter-cluster tiles associated with this task in the example architecture. The index stride on the **rg** axis in the 5 considered memories is of 19, there is thus an overlap of 6 points on the **rg** axis. On the right-hand side, the destination distribution results from a different mapping on 4 inter-cluster tiles in this architecture with different striding between the 4 associated memories, 20 in this case, and different data tiling: $36 \times 32 \times 5$ points on **rg**, **rec** and **ant** respectively.

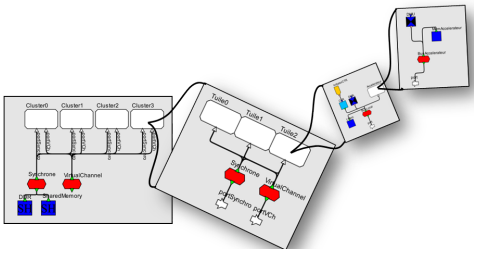

Fig. 2. Example of hierarchical parallel architecture on FPGA - abstract model in SpearDE. The model highlights the architecture topology, i.e. the number of processing elements and their different levels of encapsulation, the associated memories forming a multi-dimensional memory structure that needs to be addressed by specific data communications and the different buses. These main elements can be parametrised in terms of bandwidth, speed, capacity etc. and will later serve for performance simulations in SpearDE.

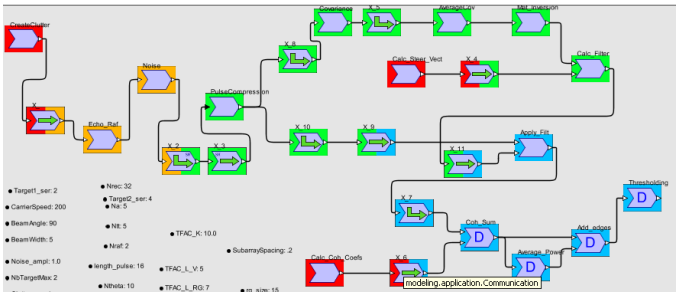

Fig. 3. Mapping the example application onto the example architecture. The application graph is partitioned into different coloured segments that work in a Kahn Process Network-like manner. Each segment is mapped onto a part of the architecture containing one or more processing elements intra/inter tile.

in the 4×3 memory topology of the example architecture) and 4 distributed memories (indexes $(2,0)$ to $(3,1)$). Note that data redundancy on one of the signal dimensions (**rg** in this example) occurs both at source and destination levels, and different handling strategies are needed since its semantics is different according to the transfer end.

A code generator will then need to integrate the corresponding loop nests implementing these data transfers. The difficulty of the communication task is enhanced by the fact that the source and destination tasks may be mapped onto different multi-dimensional architecture topologies as well. In the same time, many applications require precise dimension ordering for the multi-dimensional signals in order to ensure the correctness of the often occurring transposition operations. When data is scattered among different dimensions of multi-processor architecture parts, a communication task must in addition preserve dimension semantics, distinguishing architecture dimensions from data (signal) ones, in order to obtain both correctness and efficiency.

Assume a mapping on a p-dimensional processing architecture of a one-dimensional array `T`, giving an n-dimensional distribution, with $n = p + 1$ (architectural + data dimensions). The access function for this n-dimensional distribution can be expressed in pseudo-code using the following nested loops.

$$
\begin{aligned}
&\text{forall } arch_0 = 0, m_0 \\
&\quad \ldots \\
&\quad \text{forall } arch_p = 0, m_p \\
&\qquad \text{forall } data = 0, m_{p+1} \\
&\qquad\quad T(s_0 \cdot arch_0 + \ldots + s_p \cdot arch_p + s_{p+1} \cdot data) = f(\ldots);
\end{aligned}
$$

where $m_{i=0,p-1}$ are the number of local memories available on each dimension i of the processing architecture and m_{p+1} is the number of data points per dimension in the signal. Note equally the sampling steps $s_{i=0,p+1}$ that describe this mapped distribution in the most general case. It is important to retain the fact that there is a semantic difference between the sampling steps of the data distributions which is functional, application dependent, and the sampling steps of the data distributions mapped onto the p-dimensional computing architecture. The latter is imposed by the chosen mapping and has no functional signification for the application. It only represents the way data is distributed for parallel computing on the considered platform.

Now, in order to express data transfers, equivalent access functions for the target distribution into the source distribution can be generated and we address this issue through a polyhedral formulation that is explained in the next sections.

Possible overlaps of data points onto a given dimension k will be expressed using multiple variables in the k^{th} dimension of the access function. In this case, the access function is $T(s_0 \cdot arch_0 + \ldots + \mathbf{s_k} \cdot \mathbf{arch_k} + \mathbf{s_{o_k}} \cdot \mathbf{o_k} + \ldots + s_p \cdot arch_p + s_{p+1} \cdot data)$ where o_k varies from 0 to m_{o_k} and serves at accessing $m_{o_k} + 1$ possible redundant points (with stride s_{o_k}) either replicated in a single memory or between different memory dimensions.

Handling transfers in the context of data replication in distributed memories is one key point of our communication framework. As mentioned above, duplication can be used to share data between multi-processor parts of a heterogeneous

embedded platform, when shared memories are too inefficient or even unavailable.

B. DMA Communication Generation

Fine grain optimization through data parallelism must be combined with coarser grain optimizations such as task parallelism in order to fully exploit heterogeneous parallel architectures. Therefore, our objective is to express communications in DMA style, allowing further optimizations such as software pipelining.

DMA engines allow some hardware subsystems to access system memory directly. The DMA transfers do not block the system and the computations can continue in parallel to communications. Usual DMA engines have multiple channels to process multiple transfer requests. Therefore they are very efficient for coarse grain data transfers. They can transfer blocks of either contiguous or strided data. Our work is in line with the assumption that multidimensional DMAs being able to transfer strided blocks exist in the architecture, which is a common scenario in current platforms. To initiate a data transfer, the DMA controller needs the address of the data d, the offset ω of the first element, the number ν of elements to transfer and the stride σ between the elements in the block, for both the source and target destinations.

Our objective in this paper is to provide an algorithm that generates communications in a *DMA style* from the source and target distributions of data provided after the mapping phase of an application. That means we have to generate calls to DMA controllers for the input and output channels (respectively at the target and source architectures). DMA_SEND($d_{source}, \omega_{source}, \nu_{source}, \sigma_{source}$) and DMA_RECEIVE($d_{target}, \omega_{target}, \nu_{target}, \sigma_{target}$) are examples of simplified DMA calls that we aim to generate.

The problem of finding an optimal transfer granularity to balance computation and communication [10] is outside the scope of this article.

C. Related Work

Many compiler techniques have already been proposed to generate the communication codes for applications mapped onto distributed memory machines [11], [12], especially in the context of HPF applications [13], [14], [15], [16], [17], but, to our knowledge, none of them deals with multidimensional, potential replication on several processors, and *DMA style* code (Section II-B).

In [11], Amarasinghe and Lam propose algorithms to optimize data and computation decompositions and communications for SPMD programs. Their communication code generation algorithm uses linear algebra plus some techniques to optimize the communications at one level mapping. [12] presents methods for array redistributions using *Mathematica* for block/cyclic distributions in the PARADIGM compiler. In [15], Lee and Chen address the problem of determining the data distribution (block/cyclic/both) and generating the communication sets on distributed memory multicomputers. Generic solutions for *send* and *receive* transfers are proposed but only for strict block or/and cyclic redistribution without any overlapping. In [16], Ramanujam proposes code generation techniques for scanning sequences of local memory addresses accessed by processors based on integer lattices for HPF and Fortran D, but only for two-level mapping. In [17], Adve and Mellor-Crummey present many communication generation techniques developed in the Rice dHPF compiler for message passing systems, to optimize computation and commnications at the message level. Their techniques use the Omega library based on Presburger arithmetic. In [14] HPF remappings for message passing parallel architectures are handled. Useless remappings are removed at the global application level. The only available redundant scenario concerns the replication of an entire dimension. We do not restrain to this, any number of points being possibly replicated onto different dimensions.

One important HPF distribution characteristic is that data can be distributed over the processors in bloc or cyclic way. Potentially replicated elements are duplicated on entire architectural dimensions.

Our redistribution hypotheses differ from the previous related work because: 1) data can be replicated on a dimension (e.g. an overlap of the size of a convolution kernel); 2) data can be partially duplicated onto several local memories associated to processors (e.g. for border handling in an overlapping block-transform approach); 3) any number of points can be possibly replicated onto one or different (architecture) dimensions.

Given two distributions mixing architecture and data dimensions, that is, a source and a target distribution resulting from the mapping of an application task onto a multidimensional parallel architecture, we address the following issues: partial data redistribution between processing steps, multidimensional DMA communications, minimisation of the number of transferred elements, improvement of memory access locality and load balancing, as well as dynamic and parametric redistributions.

III. Modeling with linear algebra

We use linear algebra as in [13] to encode the mapping of array elements at the source and target destinations and to express the redistribution relation.

As explained in the previous section, we consider multidimensional data arrays with independent dimensions. Therefore, without loss of generality, to simplify the presentation and the mathematical formulae we first introduce the mapping of a one-dimensional array onto m architectural dimensions.

The array elements are linearly distributed over the architectural elements. Computations are distributed in the same way. In an m-dimensional memory architecture including local and shared memories, an array element t is identified by the affine equation

$$t = \sum_{i=1}^{m_a} \alpha_i \cdot a_i + \alpha_0 \tag{1}$$

where a_i represents the i^{th} of the architecture element onto which the data is distributed, α_i is an integer stride between two successive elements on a_i and α_0 is a constant origin. Note that the m_a architectural dimensions involved in the distribution, may be a subset of the m hierarchical parallel architectures.

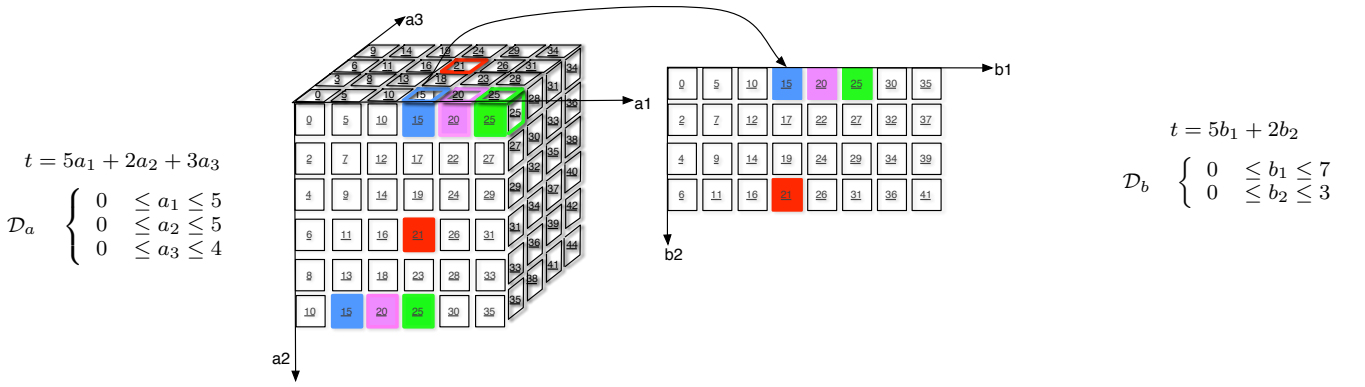

Fig. 5. Polyhedral representation of the source A and destination B distributions for a mono-dimensional array. For instance, the blue element t=15, of coordinates (3,0,0), on the first row of A is mapped to t=15, of coordinates (3,0) in B.

Among the a_i variables some represent *architectural* dimensions while other simply encode *data* overlaps onto a given dimension. The difference between pure architectural and data variables is that the data ones can be collapsed/compacted (as explained in Section VI) and all the array elements belonging to data memories $a_l (l \geq k)$ are on the same architectural component $a_l (l < k)$.

Note that the strides are not necessarily multiple of each other such as it is the case in general block and/or cyclic distributions. For example, in HPF, the general distribution of template elements `t` can be written as the linear relation $t = C \cdot P \cdot c + C \cdot p + l$ [13] between the processor coordinate `p`, the number `c` of cycles and the local block coordinate `l` (in respectively cyclic and block distribution). Strides of c, p and l are respectively $C \cdot P, C, 1$ multiples of each other. In our context, we have to deal with arbitrary strides.

A distribution $M_{t,a}$ of an 1-dimensional array `t` onto the m-dimensional architecture `a` can be represented by a $\mathbb{Z}$-polyhedron having the lattice points defined by the hyperplane

$$\mathcal{E}_{t,a} \quad \{ \; t = \sum_{i=1}^{m_a} \alpha_i \cdot a_i + \alpha_0$$

and the domain $\mathcal{D}_a$ characterizing `a` by the constraints:

$$\mathcal{D}_a \quad \begin{cases} L_{a_1} \leq a_1 \leq U_{a_1} \\ \cdots \\ L_{a_{m_a}} \leq a_{m_a} \leq U_{a_{m_a}} \end{cases}$$

L_{a_i} and U_{a_i} are the lower and upper bounds of the a_i memory. These bounds are integer, but not necessarilly explicit constants. They can be symbolic coefficients or affine expressions of the other variables a_i.

In Figure 5 we present a numerical example highlighting the equivalence between the data distribution and a constraint system for a communication scenario in which B (i.e. the destination) has to be recovered from A (i.e. the source).

Since stride coefficients can be arbitrary and because the domain bounds are not limited to the block and/or cyclic distribution coordinates, some redundant array elements may appear, even on one architectural dimension. In the example of Figure 5, many elements are replicated at least once. Some of them have been highlighted here in blue, red, green and purple.

The redistribution relation that links data from the source `a` to the target `b` can be expressed with affine equations linking the distributions dimension by dimension as expressed in Equation (2) for one dimension.

$$\sum_{i=1}^{m_a} \alpha_i \cdot a_i + \alpha_0 = \sum_{i=1}^{m_b} \beta_i \cdot b_i + \beta_0 \tag{2}$$

Thus, a communication aiming to find one distribution $M_{t,a}$ defined by the equation system $\mathcal{E}_{t,a}$ and the domain $\mathcal{D}_a$ into another, $M_{t,b}$, described by $\mathcal{E}_{t,b}$ and $\mathcal{D}_b$ will then amount to solving the following system:

$$\begin{array}{ll} \mathcal{E}_{a,b} & \left\{ \begin{array}{c} \sum_{i=1}^{m_a} \alpha_i \cdot a_i + \alpha_0 = \sum_{i=1}^{m_b} \beta_i \cdot b_i + \beta_0 \\ \\ L_{a_1} \leq a_1 \leq U_{a_1} \\ \cdots \\ L_{a_{m_a}} \leq a_{m_a} \leq U_{a_{m_a}} \\ L_{b_1} \leq b_1 \leq U_{b_1} \\ \cdots \\ L_{b_{m_b}} \leq b_{m_b} \leq U_{b_{m_b}} \end{array} \right. \\ \mathcal{D}_a & \\ \mathcal{D}_b & \end{array} \tag{3}$$

In fact, the redistribution problem comes down to enumerating the integer elements that are solutions of this constraint system. Several classical algorithms of scanning polyhedra can be used [18], [19] but they only give generic solutions that need further processing in order to derive DMA-style codes. Especially, the stride on each dimension must be clearly identified (Section V-D).

From Equation (2), it is possible to derive the lattice basis of $\begin{pmatrix} a_1 \\ .. \\ a_{m_a} \\ b_1 \\ .. \\ b_{m_b} \end{pmatrix}$ that is necessary to generate transfer codes in a *DMA style*. This resulting lattice can be found via the resolution of the diophantine equation $\sum_{i=1}^{m_a} \alpha_i \cdot a_i + \alpha_0 = \sum_{i=1}^{m_b} \beta_i \cdot b_i + \beta_0$, which is of course equivalent to solving $\sum_{i=1}^{m_a+m_b} \delta_i \cdot x_i = \delta_0$ by separating the constant part representing the origin shift between the source and the destination, and choosing appropriate δ_i-s. Section IV details this solution.

Furthermore, general algorithms for scanning polyhedra do not address the problem of redundant items. They enumerate all the elements that satisfy a constraint system. Section VII presents different solutions for enumerating only once data having to be transfered.

Note that this polyhedral formalization only characterizes all the source array elements that have to be transferred to the

destination, that is the elements in A that are required in B ($A \cap B$). Since all array elements defined by Distribution B have to be communicated, an additional test of the existence of these elements in Distribution A should be added ($B \backslash A = \emptyset$?)

To generalize to the n-dimensional array t, Equation (1) for Element t_j corresponding to the j^{th}dimension of a signal, for instance, is given by the linear equation

$$t_j = \sum_{i=1}^{m_j} \alpha_{j,i} \cdot a_{j,i} + \alpha_{j,0} \tag{4}$$

where $a_{j,i}$ represents the i^{th} dimension of the architecture onto which the j^{th} dimension of the data is distributed, $\alpha_{j,i}$ is an integer stride between two successive elements on $a_{j,i}$ and $\alpha_{j,0}$ is a constant origin.

IV. Diophantine equations and inequations

This section briefly presents some of the pre-existent mathematical results that are needed later in the paper. The first part expresses the general solution of a linear diophantine equation with any number of variables. This formulation using a lower diagonal matrix is particularly suited to the loop nest generation of communication codes. The second subsection provides references to the detection of particular integer solutions in a $\mathbb{Z}$-polyhedron, especially interesting to check the existence of the source distribution elements.

This section is technical and can be glanced through temporarily before returning studying the details.

A. General Solution of the linear diophantine equation $\sum_{i=1}^{m} \delta_i \cdot x_i = \delta_0$

If δ_0 is a multiple of the greatest common divisor of the integer coefficients δ_i this equation has an infinite number of solutions. We use the *extended Euclidean algorithm* to compute the solution for an arbitrary dimension m. The general solution [20] of the linear diophantine equation $\sum_{i=1}^{m} \delta_i \cdot x_i = \delta_0$ can be expressed as:

$$\vec{x} = \vec{x_0} + \begin{pmatrix} d_1 & 0 & 0 & 0 & 0 \\ . & . & 0 & 0 & 0 \\ -\gamma_1 x_{i,1} & . & d_i & 0 & 0 \\ . & . & . & . & 0 \\ -\gamma_1 x_{m-1,1} & . & -\gamma_i x_{m-1,i} & . & \gamma_m \\ -\gamma_1 x_{m,1} & . & -\gamma_i x_{m,i} & . & -\gamma_{m-1} \end{pmatrix} \cdot \vec{k} \tag{5}$$

where $d_j = pgcd_{j+1 \le i \le m}\left(\frac{\delta_i}{\Delta_{j-1}}\right) = \frac{pgcd_{j+1 \le i \le m}(\delta_i)}{\Delta_{j-1}}$ and $\gamma_i = \frac{\delta_i}{\Delta_{i-1}}$, $\Delta_n = \prod_{i=1}^{n} d_i$, $\Delta_0 = 1$

$\vec{k}$ are free variables. When the dimension of x is m only $m-1$ free variables are necessary to express the general solution.

The $x_{*,k}$ are the components of a particular solution of Equation $\sum_{i=k+1}^{m} \delta_i \cdot x_{i,k} = 1$ and $\vec{x_0}$ is a particular solution of Equation $\sum_{i=1}^{m} \delta_i \cdot x_i = \delta_0$. These particular solutions are computed using techniques derived from the Bezout algorithm.

This expression of the general solution is very interesting because the matrix is lower triangular. For each dimension i only one new free variable k_i is added. The expression of the general solution for x_i is

$$x_i = x_{0_i} + \sum_{j=1}^{i-1} -\gamma_j \cdot x_{i,j} \cdot k_j + d_i \cdot k_i \tag{6}$$

x_{0_i}, γ_j, $x_{i,j}$ and d_i are integer constants.

In our context, x_i represent the hierarchical memories. Note that the variable order is very important as it encodes the logical access order for the source memories.

The expression of the solution can be seen as an offset part $x_{0_i} + \sum_{j=1}^{i-1} \gamma_j \cdot x_{i,j} \cdot k_j$ plus a variable part $d_i \cdot k_i$. In our context d_i is the transfer stride between two consecutive data elements along the i-th dimension. For the last two dimensions:

$$x_{m-1} = x_{0_{m-1}} + \sum_{j=1}^{m-2} -\gamma_j \cdot x_{m-1,j} \cdot k_j + \gamma_m \cdot k_{m-1} \tag{7}$$

$$x_m = x_{0_m} + \sum_{j=1}^{m-1} -\gamma_j \cdot x_{m,j} \cdot k_j - \gamma_{m-1} \cdot k_{m-1} \tag{8}$$

They share the same free variable k_{m-1} which implies that the scanning strides for these dimensions are linked. This is the reason why it seems appropriate to choose the latest hierarchical memory dimension respectively in the source and destination architectures for the last two dimensions of x in the expression of the diophantine equation. In this case, the source and destination memory controllers will access regularly addressed components in the same way and direction.

B. Positive Integer Solution of the diophantine equation $\alpha x + \beta y = \delta$

To test the existence of an element in a particular distribution, we need to test whether equation $\alpha x + \beta y = \delta$ has *positive integer* solutions i.e. $x, y \in \mathbb{N}$ (architectural memories are positive integer variables). The following Number Theory theorems have been used to reduce the search space and to obtain accurate results.

Theorem 1: (Paoli [21]) If q is the quotient of the division of δ by $\alpha\beta$ and r is the remainder, the number of non-zero integer positive solutions of the equation $\alpha x + \beta y = \delta$ is q or $q+1$ depending on whether the equation $\alpha x + \beta y = r$ admits zero or one solution.

We deduce that it always exists a positive integer solution to $\alpha x + \beta y = r$ for x and y when $r \ge \alpha\beta$.

The number of integers r between 0 and $\alpha\beta - 1$ for which the equation $\alpha x + \beta y = r$ has a solution is given by Cesaro's theorem:

Theorem 2: (Cesaro [22]) There are exactly $\alpha\beta - \frac{1}{2}(\alpha - 1).(\beta - 1)$ natural numbers r between 0 and $\alpha\beta - 1$ for which the equation $\alpha x + \beta y = r$ has a solution.

When the integer r is between 1 and $\alpha\beta - 1$, to determine if the equation $\alpha x + \beta y = r$ has a positive integer solution we must compute the *minimal integer point* described as follows.

The point of $\alpha x+\beta y = r$ that is the closest to the origin has positive rational coordinates. The only positive integer solution (if it exists) corresponds to one point having the nearest integer coordinates of this point. If (x_1, y_1) is a positive or negative integer solution of the equation $\alpha x + \beta y = r$, the unique positive solution (if it exists) of the equation is to be found in the pairs $(x_1 + \beta k_1, y_1 - \alpha k_1)$ or $(x_1 + \beta k_2, y_1 - \alpha k_2)$ where k_1 and k_2 are the two nearest integers of $\frac{\alpha y_1 - \beta x_1}{\alpha^2+\beta^2}$. If neither of these couples is in $\mathbb{N}^2$ the equation admits no positive solution.

V. Algorithms for scanning elements to be transferred

Based on the mathematical results presented in Section IV we now give the basic algorithms for automatic code generation of data redistributions.

A. The Isolve Operator

The `Isolve` operator takes as input an equation `eq` and a variable list $\mathcal{L}$ defining the enumeration order of `a` components for the code generation.

Let `eq` be the redistribution equation: $\sum_{i=1}^{m_a} \alpha_i \cdot a_i + \alpha_0 = \sum_{i=1}^{m_b} \beta_i \cdot b_i + \beta_0$ and $\mathcal{L}$ be the list of its $m_a + m_b$ variables (m_a variables a_i and m_b variables b_i) which are subsequently renamed x_l with $1 \leq l \leq m_a + m_b$, for readability.

`Isolve` gives as output the m_a+m_b new equations on x_l as described in the general solution of diophantine equations (Section IV Equation 5). The expression of the general solution for each x_l has the form

$$x_l = x_{0_l} + \sum_{j=1}^{l-1} -\gamma_j \cdot x_{l,j} \cdot k_j + d_l \cdot k_l$$

Note that this operation introduces $m_a + m_b - 1$ new free variables k_l in the system.

B. Scanning Polyhedra using the New_bounds Algorithm

Algorithm *New_bounds* is used to enumerate the data having to be transfered and characterised by a polyhedron. It has already been presented in [18] and developed in PIPS [23]. The equations represent the lattice of the regular points and the inequations are domain constraints.

The algorithm takes as input a data set defined by a system of linear equations and inequations and an ordered set of variables. It gives as output the same polyhedron defined by a new system of inequalities such that each variable is bounded by `min` and `max` expressions containing only higher-ranked variables. It is described as follows.

The basic idea here is to use a projection algorithm to find loop bounds for each dimension. Fourier pairwise elimination cannot be used without care because it is only valid for rational and real polyhedra and provides a simple inclusion instead of a strict equality for integer points.

The first step of the algorithm consists in projecting as many useless variables as possible using the pair-wise elimination method for constraints satisfying conditions that maintain exact integer projections. At least one coefficient of the variable to eliminate in the pairwise inequations should be equal to 1 or -1 [18].

In the second step, redundant constraints are eliminated. All redundant constraints on a useless variable can be eliminated if the variable does not appear in a superior rank constraint. At least two constraints on useful variables must be kept to generate loop bounds.

Finally, the remaining useless variables are eliminated by combining pairs of constraints and by introducing integer divisions, if the variable does not appear in a constraint of superior rank.

After these steps, the final system may still contain some useless variables. Occurrences of these variables in the constraints express, like integer divisions, the non convexity of a polyhedron affine image.

C. Ordered set of variables in $\mathcal{L}$

In our context, we use a reception viewpoint for code generation. Assume a three-level (dimensions) hierarchy for the memories in a given platform for both the target and source distributions: i.e. several clusters of processors communicating through a shared cluster memory, local memories for each processor in a cluster communicating only with the shared cluster memory, and a linear address space inside each local memory for data storage. From a target cluster processor `PR`, from a local memory `MR` of `PR`, on which source processor `PS` is the data to be communicated to `PR`? On which memory `MS` of `PS` are they? How many elements `NS` have to be transfered from `MS` to `MR`? What is the offset `OS` (resp. `OR`) of the first referenced local cell `CS` on `MS` (resp. `CR` on `MR`)? What is the stride `SR` (resp. `SS`) between two successive elements on `MR` (resp. `MS`)? These are the questions that must be answered in order to generate the DMA-like redistribution.

The following example illustrates the variable enumeration order of the generated code expressed via $\mathcal{L}$. In our context we would generate the following kind of code:

```
for each target processor PR
 for each memory MR of PR
  for each sender processor PS
   for each memory MS of PS
     DMA_SEND(T,OS,NS,SS)
     DMA_RECEIVE(T,OR,NS,SR)
```

Fig. 6. Generated code with $\mathcal{L}$={PR,MR,PS,MS,CS,CR}

This enumeration order depends on the kind of code to be generated and does not affect the techniques presented in the paper.

D. Generating DMA style code

Our algorithm of code generation in *DMA style* uses the previous techniques. It is the basic algorithm of redistribution code generation used by the different optimizations detailed in the following sections.

The algorithm takes as Input a **source distribution** $M_{t,s}$ of array elements t_s characterised by a set of equations $\mathcal{E}_{t,s}$ and its domain $\mathcal{D}_s$, a **target distribution** $M_{t,r}$ of array elements t_r characterised by a set of equations $\mathcal{E}_{t,r}$ and its domain

$\mathcal{D}_r$, a **variable list** $\mathcal{L}$ defining the enumeration order of s, r components for code generation.

The Output code is in *DMA style*. It transfers all the data that respects the $M_{t,s}$ and $M_{t,r}$ constraints.

The algorithm proceeds as follows:

1) The system $\mathcal{E}_{s,r}$ of equalities is generated from $\mathcal{E}_{t,s}$ and $\mathcal{E}_{t,r}$. For each array dimension i, the equation $t_{s_i} = t_{r_i}$ is added to $\mathcal{E}_{t,s} \cup \mathcal{E}_{t,r}$, then t's variables are eliminated by projection.
2) Each equation *eq* in $\mathcal{E}_{s,r}$ is replaced by the list of equations $leq = Isolve(eq, \mathcal{L})$. This operation introduces new free variables k in the system. Each new equation has the form:

$$x_l = x_{0_l} + \sum_{j=1}^{l-1} -\gamma_j \cdot x_{l,j} \cdot k_j + d_l \cdot k_l$$

We know that $x_{0_l}, \gamma_j, x_{l,j}$ and d_l are constant integers. Only x_l, k_j and k_l are variables.
3) The scanning polyhedra algorithm is used on the $\mathbb{Z}$-polyhedron defined by $\mathcal{E}_{s,r}$ and $\mathcal{D}$ to enumerate the free variables k introduced in the previous step. All the other variables are eliminated.
4) The equations of *leq* are used to generate DMA information. For memory x_l, the offset part is $x_{0_l} + \sum_{j=1}^{l-1} -\gamma_j \cdot x_{l,j} \cdot k_j$, the stride between two successive elements is d_l and the number of elements to transfer equals to $U_{k_l} - L_{k_l} + 1$ where L_{k_l} and U_{k_l} are the lower and upper bounds of k_l computed previously.

The following example illustrates the different steps of the algorithm. From the 3-dimensional source and target distributions.

$$\mathcal{E}_{t,s} \quad \{t_s = 200ps + 50ms + 5cs$$
$$\mathcal{D}_s \quad \begin{cases} 0 \le ps \le 1 \\ 0 \le ms \le 3 \\ 0 \le cs \le 9 \end{cases}$$

$$\mathcal{E}_{t,r} \quad \{t_r = 200pr + 40mr + 10cr$$
$$\mathcal{D}_r \quad \begin{cases} 0 \le pr \le 1 \\ 0 \le mr \le 4 \\ 0 \le cr \le 3 \end{cases}$$

The redistribution problem is characterised by the following polyhedron:

$$\mathcal{E}_{s,r} \quad \{200ps + 50ms + 5cs = 200pr + 40mr + 10cr$$
$$\mathcal{D} \quad \begin{cases} 0 \le pr \le 1 & 0 \le ps \le 1 \\ 0 \le mr \le 4 & 0 \le ms \le 3 \\ 0 \le cs \le 9 & 0 \le cr \le 3 \end{cases}$$

The first step of the algorithm replaces the unique equation with the `Isolve` operator result.

$$\mathcal{E}_{s,r} \quad \begin{cases} pr = k_1 \\ mr = k_2 \\ ps = k_3 \\ ms = k_4 \\ cs = 40k_1 + 8k_2 - 40k_3 - 10k_4 + 2k_5 \\ cr = k_5 \end{cases} \qquad \mathcal{D} \quad \begin{cases} 0 \le pr \le 1 \\ 0 \le mr \le 4 \\ 0 \le ps \le 1 \\ 0 \le ms \le 3 \\ 0 \le cs \le 9 \\ 0 \le cr \le 3 \end{cases}$$

In the second step, the scanning polyhedra algorithm gives the following bounds for each free variables.

$$\begin{gathered} 0 \le k_1 \le 1 \\ 0 \le k_2 \le 4 \\ 0 \le k_3 \le 1 \\ 0 \le k_4 \le 3 \\ MAX(5 * k_4 + 20 * k_3 - 4 * k_2 - 20 * k_1, 0) \le k_5 \\ k_5 \le MIN(4 + 5 * k_4 + 20 * k_3 - 4 * k_2 - 20 * k_1, 3) \end{gathered}$$

Finally Figure 7 illustrates the redistribution code generated from these values. The offsets and strides for each memory are directly deduced from the set of equations. Note that the elements of cs are accessed with a stride 2 while elements on cr are accessed contiguously. The fastest target and source memories are accessed simultaneously and regularly.

```
int inf_k_1 = 0;
int sup_k_1 = 1;
int off_k_1 = 0;
int str_k_1 = 1;
for (int k1 = 0; k1 < sup_k_1 - inf_k_1 +1; k1++) {
 int pr = off_k_1 + k1 * str_k_1;
 int k_1 = inf_k_1 + k1;
 int inf_k_2 = 0;
 int sup_k_2 = 4;
 int off_k_2 = 0;
 int str_k_2 = 1;
 for (int k2 = 0; k2 < sup_k_2 - inf_k_2 +1; k2++) {
  int mr = off_k_2 + k2 * str_k_2;
  int k_2 = inf_k_2 + k2;
  int inf_k_3 = 0;
  int sup_k_3 = 1;
  int off_k_3 = 0;
  int str_k_3 = 1;
  for (int k3 = 0; k3 < sup_k_3 - inf_k_3 +1; k3++) {
   int ps = off_k_3 + k3 * str_k_3;
   int k_3 = inf_k_3 + k3;
   int inf_k_4 = 0;
   int sup_k_4 = 3;
   int off_k_4 = 0;
   int str_k_4 = 1;
   for (int k4 = 0; k4 < sup_k_4 - inf_k_4 +1; k4++) {
    int ms = off_k_4 + k4 * str_k_4;
    int k_4 = inf_k_4 + k4;
    int inf_k_5 = MAX(5*k_4+20*k_3-4*k_2-20*k_1,0);
    int sup_k_5 = MIN(4+5*k_4+20*k_3-4*k_2-20*k_1,3);
    int off_k_5 = 40*k_1+8*k_2-40*k_3-10*k_4+2*inf_k_5;
    int str_k_5 = 2;
    int off_last = inf_k_5;
    int str_last = 1;
     DMA_SEND(ts,off_k_5,sup_k_5-inf_k_5+1,str_k_5)
     DMA_RECEIVE(tr,off_last,sup_k_5-inf_k_5+1,str_lst)
}}}}}
```

Fig. 7. Redistribution code with DMA calls

VI. Optimization: Compacting dimensions

The goal of the *Compact* operator is to pack the elements distributed on the *data* variables encoding the distribution of the same array dimension, i.e. encoding data overlap. For example, this situation can occur when performing a convolution. Assume a convolution kernel of 10 points applied to a signal of 50 points with a stride of 2 at each iteration. If this data distribution is encoded in the described polyhedral formalism, we obtain two variables, for instance d_0 and d_1, with loop bounds between 0 and 9 or 49, respectively. Parsing this entire distribution with the corresponding loop indexes would then lead to 500 iterations involving repeated storage of many elements of the original signal, while in fact storing signal indexes between 0 and 107 $(= 2 * 49 + 9)$ suffice to address the entire distribution. These artificial data dimensions can sometimes be projected without increasing the cardinality of the domain $\mathcal{D}$ and therefore we apply an algorithm to eliminate them. This reduces the number of duplicated elements. In the above example, we can thus replace d_0 and d_1 by a new variable, d, indexed by $\mathcal{E}_{t_s,a}$: $\{t_s = d\}$ (note the new stride of 1) and the domain $\mathcal{D}_a$: $\{0 \le d \le 107\}$.

Let $M_{t,a}$ be a distribution of Array `t` onto the memories `a` defined by the equation system $\mathcal{E}_{t,a}$ and the domain $\mathcal{D}_a$. The *Compact* operator takes as input $M_{t,a}$ and gives as output a new distribution $M_{t,a'}$ such that $M_{t,a} \subset M_{t,a'}$ and $|\mathcal{D}_{a'}| \le |\mathcal{D}_a|$ and $dim(\mathcal{D}_{a'})$ is minimal.

Let be the set of *reduced* distributions:

$$\mathcal{RD}(M_{t,a}) = \{M_{t,a'}/M_{t,a} \subset M_{t,a'}, |\mathcal{D}_{a'}| \leq |\mathcal{D}_a|, dim(\mathcal{D}_{a'}) \leq dim(\mathcal{D}_a)\}$$

Let $\mathcal{CD}(M_{t,a})$ be the subset of $\mathcal{RD}(M_{t,a})$ with the minimal cardinal domain.

$$\mathcal{CD}(M_{t,a}) = \left\{M_{t,a'} \in \mathcal{RD}(M_{t,a})/|\mathcal{D}_{a'}| = min_{M_{t,a''} \in \mathcal{RD}(M_{t,a})}(|\mathcal{D}_{a''}|)\right\}$$

Compact is defined as follows:

$$Compact(M_{t,a}) = \{M_{t,a'} \in \mathcal{CD}(M_{t,a})/ \\ dim(\mathcal{D}_{a'}) = min_{M_{t,a''} \in \mathcal{CD}(M_{t,a})}(dim(\mathcal{D}_{a''}))\}$$

Figure 8 illustrates the different steps of the compaction of the distribution $M_{t,a}$. First, a_1 and a_2 are collapsed because $|\mathcal{D}_{a'}| = 180 < |\mathcal{D}_a| = 360$. Second, ${a'}_1$ and ${a'}_2$ are compacted because this reduces the dimension of $\mathcal{D}_{a''}$. No other dimension can be compacted because the constraint $M_{t,a} \subset M_{t,a'}$ must be preserved.

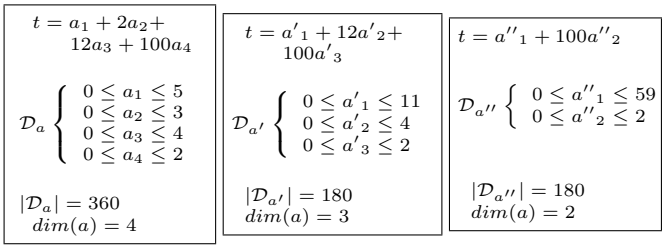

Fig. 8. Example of Compaction of distribution $M_{t,a}$

In our context only source distributions will be compacted because if redundancy exists at the target distribution that means the same element has to be transferred into different memories at destination, but only one copy of the source element has to be communicated.

VII. Optimization: Elimination of redundant transfers

This section presents the additional steps in the algorithm of redistribution code generation necessary to avoid redundant transfers. We start with the case of a two-dimensional distribution before extending the technique to an arbitrary number of dimensions.

Let us first define what we call a redundant transfer. A redundant transfer is the communication of a given data several times unnecessarily. This occurs when a piece of data is present on different *source* processors of the distribution. This element has to be communicated but it is not necessary to transfer all its occurrences; only one is enough. Quite the opposite, if an element appears more than once in the *target* distribution, it must be repeatedly communicated to the different target processors when in a distributed memory scenario.

Therefore it is useful to detect the different occurrences of an array element in the source distribution.

Let $t = \alpha x + \beta y$ be a source distribution equation. Several occurrences of t exist if $\{(x,y) \in \mathcal{D}/\exists (x',y') \in \mathcal{D}, (x',y') \neq (x,y), \alpha x + \beta y = \alpha x' + \beta y'\}$ is not empty. This system admits the solutions $(\beta\gamma, -\alpha\gamma), \gamma \in \mathbb{Z}$.

Let $\mathcal{D}$ be the domain $\{L_x \leq x \leq U_x, L_y \leq y \leq U_y\}$. Thus there exist some duplicated data if $(U_x - L_x \geq \beta)$ and $(U_y - L_y \geq \alpha)$ are verified.

Moreover each block (β, α) of data is repeated at least once if $min(\frac{U_x - L_x + 1}{\beta}, \frac{U_y - L_y + 1}{\alpha}) \geq 2$.

A. Cutting Planes

To eliminate the redundant elements it is sufficient to cut the redundant part of the domain $\mathcal{D}$. According to the previous *redundancy distance vector* $(\beta\gamma, -\alpha\gamma), \gamma \in \mathbb{Z}$, two possibilities occur:

Case A: cut the $\{x > L_x + \beta, y < U_y - \alpha + 1\}$ part or

Case B: cut the $\{x < U_x - \beta + 1, y > L_y + \alpha\}$ part of domain $\mathcal{D}$.

Figure 9 illustrates these two possible cuts. The initial source distribution is defined by $t = 5x + 2y$ and $\{0 \leq x \leq 7, 0 \leq y \leq 12\}$.

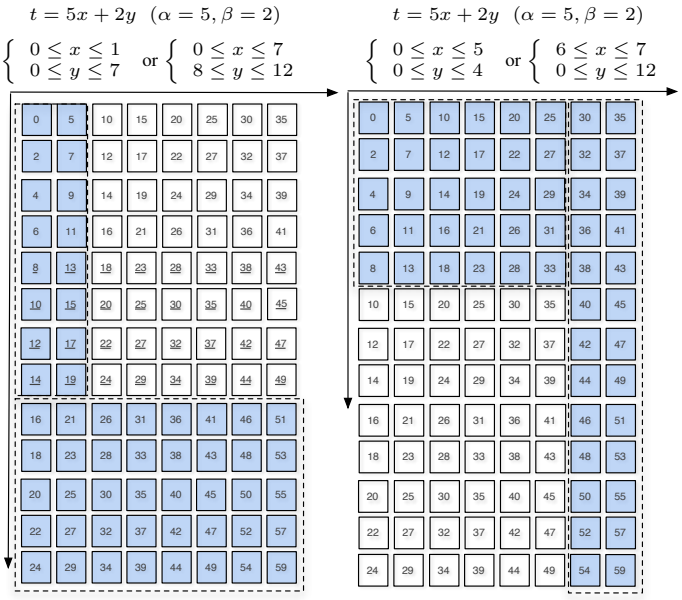

Fig. 9. Cuts for redundancy elimination - Case A (left) and Case B (right)

We can observe that cutting the redundant part leads to a non-convex domain (in blue color). So we translate the redistribution problem into two new redistribution problems to match two convex domains (surrounded by gray hashed lines).

To sum up, let $M_{t,s}$ be the initial source distribution defined by the equation system $\mathcal{E}_{t,s} = \{t_s = \alpha x + \beta y\}$ and the domain $\mathcal{D}_s = \{L_x \leq x \leq U_x, L_y \leq y \leq U_y\}$. In case of redundant data in this source distribution, we apply instead the Algorithm presented in Section V-D on the two new source distributions M_{t,s_1} and M_{t,s_2} defined such as:

Case A:

1) M_{t,s_1} is characterized by the equation system $\mathcal{E}_{t,s_1} = \{t_{s_1} = \alpha x + \beta y\}$ and the domain $\mathcal{D}_{s_1} = \{L_x \leq x \leq \beta - 1, L_y \leq y \leq U_y - \alpha\}$;
2) M_{t,s_2} is characterized by the equation system $\mathcal{E}_{t,s_2} = \{t_{s_2} = \alpha x + \beta y\}$ and the domain $\mathcal{D}_{s_2} = \{L_x \leq x \leq U_x, U_y - \alpha + 1 \leq y \leq U_y\}$.

Case B:

1) M_{t,s_1} is characterized by the equation system $\mathcal{E}_{t,s_1} = \{t_{s_1} = \alpha x + \beta y\}$ and the domain $\mathcal{D}_{s_1} = \{L_x \leq x \leq U_x - \beta, L_y \leq y \leq L_y + \alpha - 1\}$;
2) M_{t,s_2} is characterized by the equation system $\mathcal{E}_{t,s_2} = \{t_{s_2} = \alpha x + \beta y\}$ and the domain $\mathcal{D}_{s_2} = \{U_x - \beta + 1 \leq x \leq U_x, L_y \leq y \leq U_y\}$.

B. Sliding windows

The cutting plane strategies presented in Section VII-A imply generating a number of transfers that is not always load-balanced, especially when the strides α and β are small and the domain is large. This section suggests another strategy to divide the domain in a more balanced way.

As for the cutting plane strategy, the constraints can be added on any dimension.

Case A (load balancing on x): Data are duplicated every β items on x. h is the number of blocks on x.

$$h = \frac{x}{\beta} \tag{9}$$

First α elements are assigned to each block. We compute the block size BS of additional data y that can be fairly distributed in each block h.

$$BS = \frac{U_y + 1 - \alpha}{\frac{U_x - L_x + 1}{\beta}} \tag{10}$$

Once data are equitably distributed, some constraints on y should be added to avoid the enumeration of duplicates belonging to a previously scanned block.

$$BS.h \leq y \leq BS.h + \alpha + BS - 1 \tag{11}$$

Case B (load balancing on y) is symmetric of the previous one on x.

Figure 10 illustrates the *sliding windows* strategy - case A - that avoids redundant transfers from the source distribution in the code generation algorithm and load balances the non-redundant transfers on x.

Theorem 3: The condition (Eq 11) is sufficient to avoid redundancy.

Proof 1: The *redundancy distance vector* on (x, y) is $(\beta, -\alpha)$. Assume it exists $t = \alpha x + \beta y$ and $t' = \alpha x' + \beta y'$ two redundant elements; we want to prove by contradiction that y, y' cannot both verify the constraints

$$BS.h \leq y \leq BS.h + \alpha + BS - 1 \tag{12}$$
$$BS.h' \leq y' \leq BS.h' + \alpha + BS - 1 \tag{13}$$

t and t' are redundant implies $x' - x \geq \beta$ and $y - y' \geq \alpha$. From Equations (12) and (13), the following constraint on $y - y'$ holds:

$$BS.(h - h') - \alpha - BS + 1 \leq y - y' \leq BS.(h - h') + \alpha + BS - 1$$

Since $h = \frac{x}{\beta} < h' = \frac{x'}{\beta}$ then $y - y' \leq \alpha - 1$. This is in contradiction with $y - y' \geq \alpha$. ■

To generate code with the *sliding windows* strategy, Constraints (9) and (11) are added to the polyhedra describing the source distribution. Constraint (9) is translated into inequations using the integer division rules because integer division cannot be directly introduced in the polyhedral system. The values of h should be computed dynamically because it depends on x.

Constraint (10) can be computed at compile time because α, β and the domain bounds are known. The upper bound of Constraint (11) is relaxed for the last value of h.

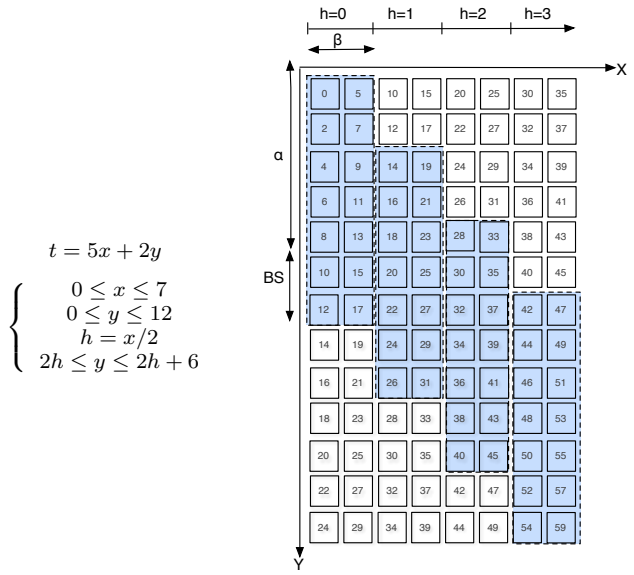

Fig. 10. Load-balanced cuts for redundancy elimination

C. Generalization to arbitrary dimensions in the redistribution equation

The previous two strategies to avoid redundant transfers can be generalized to an arbitrary number of dimensions in the redistribution. Constraints characterizing the cuts on the different dimensions of the source distribution can be added recursively in the systems. Then the code generation algorithm is applied onto these different non-redundant convex parts of the domain. If the source distribution is n-dimensional and in case of redundancy, there will be at most 2^{n-1} polyhedral subsystems.

However, because the coefficients in the redistribution equation are arbitrary, some elements may be missed. This section presents the cases of *missing points*, the techniques to detect them and how to generate their relative communications.

Missing points in a lattice: To illustrate the case of *missing points* when adding the non-redundant constraints, we choose the following 3-dimensional distribution equation

$$t = \alpha x + \beta y + \gamma z$$

and the domain $\{0 \leq x \leq U_x, 0 \leq y \leq U_y, 0 \leq z \leq U_z\}$. First, constraints to eliminate redundancy on the 2-dimensional domain of (x, y) are added. Secondly, the following change of basis $v = \alpha' x + \beta' y$ where $d = gcd(\alpha, \beta), \alpha' = \frac{\alpha}{d}, \beta' = \frac{\beta}{d}$ is applied and the following new distribution equation is considered:

$$t' = dv + \gamma z \tag{14}$$

Finally the second cuts to eliminate redundancy relative to (v, z) components are added.

Figure 11 shows the elements of the distribution on the example $t = 3x + 5y + 19z$ with the domain

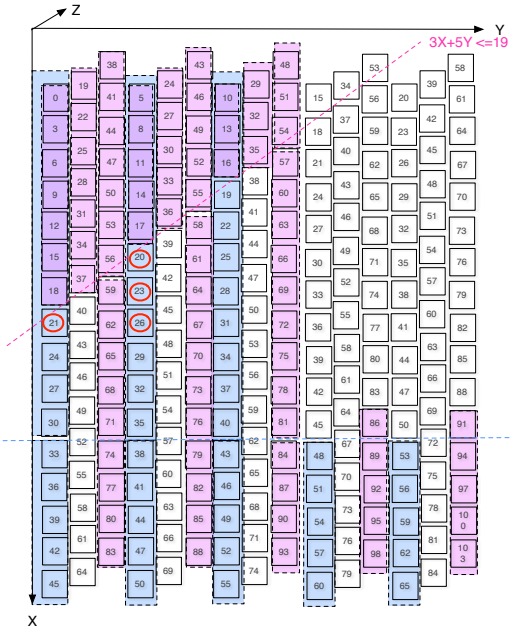

Fig. 11. Cuts for a 3-dimensional distribution

$\mathcal{D} = \{0 \leq x \leq 15, 0 \leq y \leq 4, 0 \leq z \leq 2\}$. We use the *cutting planes* strategy to eliminate the redundant elements. The non-redundant part for Domain (x, y) is represented in blue, and the non-redundant part for Domain (x, y, z) is coloured in purple.

We can observe that the data indexed by 20, 21, 23 and 26 are eliminated by the second cuts and thus are not present in the final domain. In fact, this problem occurs because the lattice of the linear application $3x + 5y$ has a unit base ($gcd(3, 5) = 1$) but, on the selected domain $\mathcal{D} \subset \mathbb{N}^2$, Elements indexed by 1, 2, 4 and 7 are not referenced [1]. Cesaro's theorem (Section IV-B) confirms that 4 elements have no solutions for $x, y \in \mathbb{N}, 0 \leq 3x + 5y \leq 15$. Note that because our domain is finite, four other symmetric elements will be missing at the upper bounds of the domain. But on the interval $[\alpha\beta, \alpha.U_x + \beta.U_y - \alpha\beta]$ all the elements are referenced:

$$\forall g \in [\alpha\beta, \alpha.U_x + \beta.U_y - \alpha\beta]$$
$$\exists x, y \in \mathbb{N}, 0 \leq x \leq U_x, 0 \leq y \leq U_y / g = \alpha x + \beta y$$

Before adding the cutting planes relative to the redundant elements on z, the change of basis is applied: $v = 3x + 5y$ and, from Equation $t' = 1v + 19z$, the second cuts are added, but this assumes the domain of v is dense. Since the points indexed by 1, 2, 4 and 7 are missing, $20(1 + 19)$, $21(2 + 19)$, $23(4 + 19)$, $26(7 + 19)$ are not referenced anymore.

We use the Paoli's theorem and the computation of the *minimal integer point* (Section IV-B) to develop an algorithm testing the existence of missing points in the intervals $[1, \alpha\beta - 1]$ and $[\alpha.U_x + \beta.U_y - \alpha\beta + 1, \alpha.U_x + \beta.U_y]$ and giving their list $\mathcal{G}$.

The cuts of Sections VII-A, VII-B are designed to add additional constraints to remove redundant data. The second cuts relative to redundant elements on z intend to eliminate elements belonging to the two lattices $t = \alpha x + \beta y$ and $t' = dv + \gamma z$ (Eq. 14).

The goal now is to find the elements $\mathcal{P}$ that have been unintentionally eliminated by these cuts because of *missing points*, then to characterize them to enable their transfer if they belong to the target distribution. For each element g of $\mathcal{G}$, these points are characterized by the following constraints:

Case A (refers to Section VII-A-Case A-1)

When $g \leq \gamma - 1$ then $\mathcal{P}$ satisfies the additional constraints $\{g = \alpha' x + \beta' y, z <= U_z - z_p\}$

Case B (refers to Section VII-A-Case A-2)

When $g \geq \gamma$ then $\mathcal{P}$ satisfies the additional constraints $\{g = \alpha' x + \beta' y, z <= U_z - z_p, U_z - z_p - d + 1 <= z\}$

where z_p is the smallest value in the set $\{z_p / \exists x, y \in \mathcal{D}, 0 \leq z_p \leq U_z, \alpha' x + \beta' y = g + \gamma z_p\}$.

These constraints are added to the redistribution systems after the first cuts. The two resulting systems represent additional redistribution polyhedra, used as input of our code generation algorithm to transfer *missing points* (when they exist, i.e. when the polyhedron is not empty).

In real applications and mappings to architecture, the number of distribution architectural dimensions are small. Missing elements therefore are very limited, and will be even more often non-existent because the problem occurs only when $gcd(\alpha, \beta) \neq min(\alpha, \beta)$. In particular, it does not appear in general block/cyclic distributions without sampling.

VIII. Testbench and Experiments

The proposed techniques have been successfully implemented using the LinearC3 library which is a robust, open source and free license (LGPL) library of the PIPS Project [23] developed since 1988.

To validate our approach we have gathered a test bench of about 50 significant redistribution examples covering the different criteria we want to optimize. First, for a given multi-dimensional architecture scenario, we have tested exhaustive combinations of mappings of a one-dimensional signal: block distribution at the source versus block distribution at the destination, block versus cyclic, and so on and so forth, with and without redundancy both at the source and the destination, with or without offset on one dimension, etc. We have developed a validation code to test the correctness of the generated redistributions for the entire test bench and for any randomly generated case. In addition, for some specific distributions, where a simple particular solution to each stage of the diophantine equation solution process can be given, we also provide parametric and generic communication code with our techniques.

Note that our industrial environment aims above all to reduce the development costs linked to parallelising both legacy and emerging applications onto a wide variety of architectures without any assumptions on the mapping type. We mostly privilege the man-in-the-loop approach for mapping domain-specific applications especially when having to deal with non-COTS architectures for which mapping optimisation engines do not exist. Our validation framework relies on the

[1] This problem is well-known and has been encountered in several compilation domains [24], [25]

generality of the approach in terms of worst-case redundant mapping scenarios for multi-dimensional architectures which are covered by our extended example suite. Generating correct and also simpler than manual code is a highly sought for feature in the industrial world.

The potential gains of our techniques are numerous. DMA engines accelerate memory transfers and computations can be executed in parallel with communications. Our DMA-style code provides obvious gains for this kind of architectures, but not only, because the generated codes promote regular and contiguous communications which cause gains comparable to those observed when restructuring memory accesses. Our experiments to compare the costs of non-redundant and redundant communications show that the gain in terms of time is proportional to the number of unnecessary transfers eliminated. Moreover, the non-redundancy strategy leads to simpler communication codes: more regular, with fewer calls to functions such as min, max, and integer divisions. On our testbench the gains in number of cycles are systematic when compared to original communication code without redundancy handling. Pushing exploration, we observe that, by explicitly evaluating architectural dimensions, the generated code is more efficient than the generic polyhedral codes. Additional experiments should be performed to evaluate, for a particular architecture, the trade-off between the numbers of DMA transfer calls and the DMA initialization costs depending on the machine and its run-time.

IX. Conclusion

In this paper, we have presented algorithms for automatically generating correct-by-construction communication code that is redundancy-optimized for multi-dimensional redistributions. Our developments stem from the goals of an industrial parallelisation framework, SpearDE, in terms of code generators for a very wide range of multi-dimensional communications in redundant mappings scenarios. The main advantage of this approach is the relaxation of parallel programming constraints for application developers, while considering efficient tradeoffs between communications and computations at the mapping stage. Then, the very error-prone task of explicit data transfers within heterogeneous memory systems is performed automatically and efficiently, while preserving the functional semantics of the application. We have considered the somewhat worst-case scenario in which a communication can be any linear multi-dimensional distribution between the source and the target, not only block and/or cyclic distributions, and we have used a polyhedral formulation to find the solution for the desired transfers. The generated code follows DMA principles and several optimisations where performed for redundancy handling, either at data level or at architecture level. The proposed optimisations ensure a minimization of transfer time through load balancing performed on the architecture dimensions. Note also that these optimisations stand for different criteria as well, as they are linked to our considered variable semantics. By changing this semantics (cuts on data dimensions for instance instead of the architecture ones), other types of communications can be obtained.

Acknowledgements*: We are thankful to Pierre Jouvelot, Ali Koudri and Fabien Coelho for their careful reading and many suggestions which helped us to improve the paper.*

References

[1] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007.

[2] P. Bonnot, F. Lemonnier, G. Edelin, G. Gaillat, O. Ruch, and P. Gauget, "Definition and SIMD implementation of a multi-processing architecture approach on FPGA," in *DATE*, 2008.

[3] "6ak2e05/02 Multicore DSP+ARM keystone II SoC," Nov. 2012.

[4] M. Gschwind, H. Hofstee, B. Flachs, M. Hopkin, Y. Watanabe, and T. Yamazaki, "Synergistic processing in cell's multicore architecture," *Micro, IEEE*, vol. 26, no. 2, pp. 10–24, 2006.

[5] L. Benini, E. Flamand, D. Fuin, and D. Melpignano, "P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator," in *DATE*, 2012, pp. 983–987.

[6] D. A. Patterson and J. L. Hennessy, *Computer organization and design (2nd ed.): the hardware/software interface*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998.

[7] E. Lenormand and G. Edelin, "An industrial perspective: a pragmatic high-end signal processing design environment at Thales," in *SAMOS*, 2003, pp. 52–57.

[8] P. Boulet, "Array-OL Revisited, Multidimensional Intensive Signal Processing Specification," INRIA, Research Report RR-6113, 2007.

[9] F. Lemonnier, P. Millet, G. Marchesan Almeida, M. Huebner, J. Becker, S. Pillement, O. Sentieys, M. Koedam, S. Sinha, K. Goossens, C. Piguet, M. Morgan, and R. Lemaire, "Towards Future Adaptive Multiprocessor Systems-On-Chip: an Innovative Approach for Flexible Architectures," in *SAMOS*, Greece, Jul. 2012.

[10] S. Saidi, "Optimizing dma data transfers for embedded multi-cores," Ph.D. dissertation, Universite de Grenoble, october 2012.

[11] S. P. Amarasinghe and M. S. Lam, "Communication optimization and code generation for distributed memory machines," *SIGPLAN Not.*, vol. 28, no. 6, pp. 126–138, Jun. 1993.

[12] E. Su, A. Lain, S. Ramaswamy, D. J. Palermo, E. W. Hodges, IV, and P. Banerjee, "Advanced compilation techniques in the paradigm compiler for distributed-memory multicomputers," ser. ICS '95.

[13] C. Ancourt, F. Coelho, F. Irigoin, and R. Keryell, "A linear algebra framework for static high performance fortran code distribution," *Sci. Program.*, vol. 6, no. 1, pp. 3–27, Jan. 1997.

[14] F. Coelho and C. Ancourt, "Optimal compilation of hpf remappings," *J. Parallel Distrib. Comput.*, vol. 38, no. 2, pp. 229–236, Nov. 1996.

[15] P. Lee and W.-Y. Chen, "Compiler techniques for determining data distribution and generating communication sets on distributed-memory machines," ser. HICSS '96, pp. 537–546.

[16] J. Ramanujam, "Compiler optimizations for scalable parallel systems," 2001, ch. Integer lattice based methods for local address generation for block-cyclic distributions, pp. 597–645.

[17] V. Adve and J. Mellor-Crummey, "Compiler optimizations for scalable parallel systems," 2001, ch. Advanced code generation for high performance Fortran, pp. 553–596.

[18] C. Ancourt and F. Irigoin, "Scanning polyhedra with do loops," in *in ACM SIGPLAN*, ser. PPOPP '91, 1991, pp. 39–50.

[19] C. Bastoul, "Code generation in the polyhedral model is easier than you think," in *PACT'13 IEEE*, 2004, pp. 7–16.

[20] J. Bond, "Calculating the general solution of a linear diophantine equation," in *American Math Monthly*, 1967, vol. 74, no. 8.

[21] *History of the theory of numbers*. Washington, Canergie institution of Washington, 1874.

[22] E. Cesaro, "Sur diverses questions d'arithmetique," 1883.

[23] MINES-ParisTech, "LinearC3 - PIPS project," http://pips4u.org, 1989, open source, under LGPL.

[24] W. Pugh, "The omega test: a fast and practical integer programming algorithm for dependence analysis," *Comm. of the ACM*, vol. 8, pp. 4–13, 1992.

[25] B. Meister, "Projecting periodic polyhedra for loop nest analysis," in *IN CPC*, 2004, pp. 13–24.