Optimization of the Background Memory Utilization by Partitioning

Uwe Eckhardt

IEE, TU Dresden
eckhardt@iee.et.tu-dresden.de

Renate Merker

IEE, TU Dresden
merker@iee1.et.tu-dresden.de

Abstract

The skilful utilization of the memory structure of a processor and of its background memory may crucially affect the system performance. We propose a restructuring of for-loop programs by hierarchical partitioning which improves the properties of the algorithm with respect to the memory utilization. We consider the problem for regularly connected processor arrays (where single processors are a special case) and for a memory structure which is subdivided into local foreground memory (register) and background memory with up to three levels (cache, RAM, mass storage). The extension of the lifetime of a variable on an inner memory level, i.e. the decrease of the number of read accesses to more outer memory levels is the object of the proposed method.

1 Introduction

The efficient processing of for-loop programs on single processors and processor arrays is a well studied problem, which incorporates program restructuring techniques and high level synthesis methods. There exist compiler systems for the mapping of for-loop programs, where indexing functions of the variables are affine functions of the loop indices, onto (piecewise) regularly connected processor arrays. An essential step in these compilers is the transformation of global data dependencies into local data dependencies by localization procedures [16]. The resulting algorithms, the so-called (conditional) uniform recurrence equations (URE's) [8], are considered in this paper.
The loop-transformations splitting and interchange have been discussed in [18, 17] for the memory synthesis for these algorithms. We apply these ideas in a systematic partitioning framework. We extend the improvement of the foreground memory utilization by an improvement of the utilization of different levels of the background memory. The background memory is supposed to be structured in a cache level, a RAM level and a mass storage level (see Fig.(1)). The extension of the lifetime of a variable on an inner memory level , i.e. the decrease of the number of read accesses to more outer memory levels is the object of the proposed method. The foreground memory is supposed to be local memory (register) associated to each processor and the background memory belongs to the peripheral system.
We propose a hierarchical application of the well studied locally parallel, globally sequential (LPGS)- and locally sequential, globally parallel (LSGP)-partitioning schemes [7, 9, 11, 10, 12, 6, 1, 14, 15] for the systematic solution of the memory utilization problem.
The remainder of the paper is organized as follows: Section 2 presents the key-concept of the optimization of the background memory utilization by hierarchical partitioning for the example of the matrix multiplication. Section 3 is devoted to the mathematical model of the proposed method. Finally, a conclusion is given.

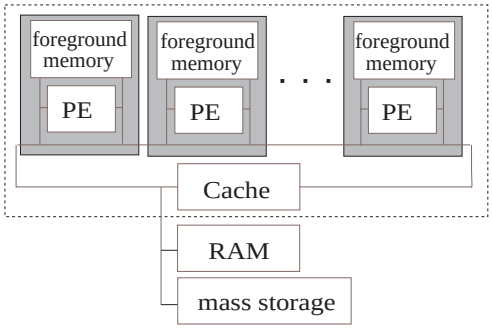

Figure 1: *Memory structure*

2 The key-concept of hierachical partitioning

In this section we illustrate the key-concept of the hierarchical partitioning for the example of the matrix multiplication $\mathbf{C} = \mathbf{QW}$ of two square matrices of

order N.

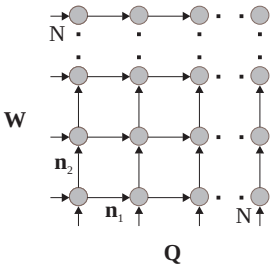

Figure 2: *Example: matrix muliplication array*

2.1 Foreground memory utilization and I/O access behaviour to the background memory

The first step of the mapping of the matrix multiplication onto the processor array results in a so-called full size array (see section 3.1). Here, the extend (i.e. the number of processing elements) of the array depends on the order of the matrices. The full size array for the considered matrix multiplication is shown in Fig.(2). The variables w and q are propagated along the directions $\mathbf{n}_1$ and $\mathbf{n}_2$ respectively. The variables c are fixed on the processing elements.
The goal is the processing of the algorithm on a structure with a defined number of processing elements and a defined I/O-behaviour between the processing elements and between the whole array and the peripheral system. Hence, partitioning techniques have to be applied. Suppose as target architecture a processor array with two times two processing elements and with given I/O-capacities of the interconnections between the processing elements of the array and between the array and the peripheral system, which contains the background memory. The co-partitioning method [4] which represents a simultaneous LPGS- and LSGP-partitioning (see section 3.2) has to be applied to adjust the full size array Fig.(2) to the given constraints. Fig.(3) depicts the co-partitioning of the full size array. The small rectangles symbolize the LSGP-partitions. All tasks belonging to an LSGP-partition are performed sequentially on one processing element of the partition. Thus, there remains only one processing element for each LSGP-partition. The remaining processing elements are drawn black. The doted lines within the LSGP-partitions in Fig.(3) represent the introduced sequential scheduling. All LSGP-partitions are active in parallel. Hence, the $N \times N$ full size array

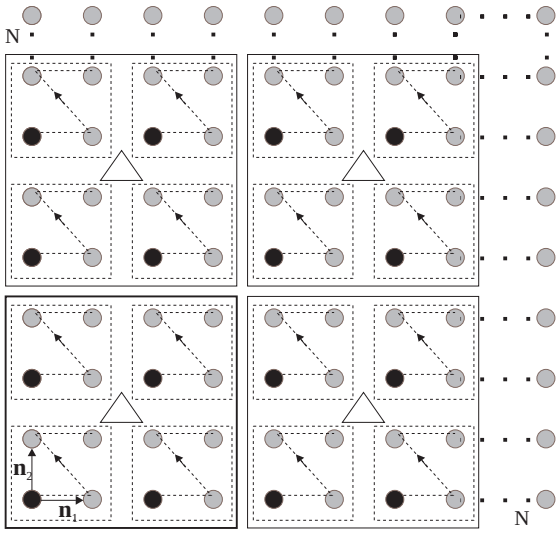

Figure 3: *Foreground memory allocation by two-fold partitioning*

has been transformed into an LSGP-partitioned array of $\frac{N}{2} \times \frac{N}{2}$ processing elements. The large rectangles in Fig.(3) symbolize the LPGS-partitions. The processors of an LPGS-partition are active in parallel and the partitions are scheduled sequentially. Each partition consists of two times two processing elements.
The LSGP-part of the co-partitioning defines the size of the local foreground memory, which has to be added to each processing element. The size of this local memory depends directly on the number of processing elements, which belong to the LSGP-partition. The LSGP-partitioning and hence, the addition of foreground memory leads to a decrease of the I/O-demand between the processing elements of the array and between the whole array and the peripheral system. A partition of two times two processing elements has to receive two variables w and two variables q and has to put out also two variables w and two variables q for each processing cycle. Hence, a partition of two times two processing elements has an I/O-demand with the peripheral system of eight data items in each processing cycle. Each LPGS-partition in the co-partitioned array shown in Fig.(3) has an I/O-demand of 16 data items. However, this I/O-demand is distributed over the four sequential cycles introduced by the LSGP-partitioning. Hence, we achieve a mean value of the I/O-demand of four data items per clock cycle.
Therefore, we can realize a balancing of the size of the foreground memory and the I/O-access behaviour of the array to the background memory by

co-partitioning. The co-partitioning represents a partitioning with a hierarchy of two levels (see section 3.2).

2.2 Improvement of the cache utilization

Suppose, that a co-partitioning has been applied to achieve an array of two times two processors with an I/O-behaviour which is adjusted to the I/O-constraints imposed by the peripheral system. The resulting LPGS-partitions (co-partitions) have to be scheduled sequentially. The elements symbolized by the triangles in Fig.(4) represent the LPGS-partitions (co-partitions) achieved in the previous foreground memory allocation process. The sequential scheduling of the LPGS-partitions can be given by two nested for-loops, where one loop is associated with direction $\mathbf{n}_1$ and the other with $\mathbf{n}_2$. The loop nest can be arbitrarily selected.

We assume, that we have a cache of $4 * 10^3$ words

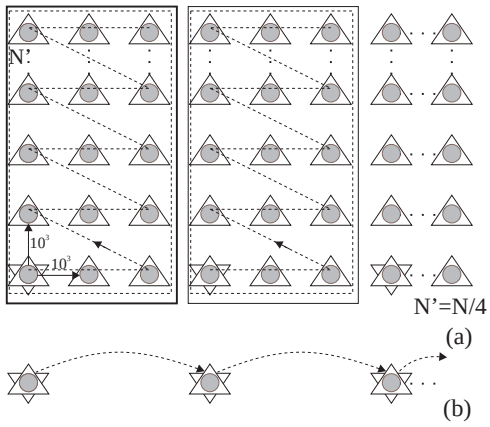

Figure 4: *Improved cache level utilization by four-fold partitioning*

and an I/O-demand of 10^3 words in direction $\mathbf{n}_1$ and $\mathbf{n}_2$ (i.e. 10^3 words have to be transferred to the array in each direction $\mathbf{n}_{1,2}$ for the processing of all tasks which are allocated to an LPGS-partition (co-partition)). We could store the first $3 * 10^3$ words of variables q and the 10^3 words of variable w in the cache in the case of a scheduling of the LPGS-partitions first along direction $\mathbf{n}_1$ (inner loop) and then along direction $\mathbf{n}_2$ (outer loop). Hence, all variables w and the first $3 * 10^3$ words of variables q have to be read once and the remaining $(N' - 3) * 10^3$ words of variables q have to be read N' times from the RAM or mass storage level. We obtain for this scheduling a read access of $(N'^2 - 2N' + 3) * 10^3$ words to the RAM and mass storage level. The partitioning and scheduling like shown in Fig.(4) leads to a decrease of this access to $(\frac{N'^2}{3} + N') * 10^3$ words. Hence, we obtain for large N' a decrease of factor three. The doted lines in Fig.(4) represent the sequential scheduling within the partitions and of the partitions respectively.
Obviously, this partitioning represents also a co-partitioning, where the LPGS-partitions consist of one processing element, which is symbolized by the white star in Fig.(4). Fig.(4b) illustrates the sequential scheduling of the LPGS-partitions of the second co-partitioning. Hence, we can apply the same formal framework like for the foreground memory allocation for the representation of this partitioning. Therefore, we have to apply a partitioning with a hierarchy of four levels for foreground memory allocation and improvement of the cache utilization (see section 3.4). The access to the RAM and mass storage occurs on the boundary of the partitions. Hence, the optimization of the cache utilization is given by the determination of LSGP-partitions (in the second co-partitioning) of maximal size, which can realize the internal communication (data transfers within the partition) via the cache, i.e. without access to the RAM or mass storage.

read access to level	improved cache	improved cache & RAM
RAM & MS	$\frac{N'^2}{3} + N'$	$\frac{N'^2}{3} + 2N'$
MS	$\frac{N'^2}{6} + 1.5N'$	$3N'$

Table 1: Access to RAM and mass storage (MS) level

2.3 Improvement of the RAM utilization

Suppose a RAM of $N'/2 * 10^3$ words in the memory structure, which is attached to the array. If we apply the partitioning as shown in Fig.(4), then we can store $N'/2 * 10^3$ words w which have to be transferred from the left to right side of the array (i.e. along $\mathbf{n}_1$) in the RAM for the reuse in the next partition. The remaining $N'/2 * 10^3$ words which have to be transferred along $\mathbf{n}_1$ have to be read from the mass storage each time a new partition is scheduled. Therefore, the total amount of read accesses to the mass storage during the processing of the matrix multiplication is $(\frac{N'^2}{6} + 1.5N') * 10^3$. A decrease of the amount of read accesses to the mass storage can be achieved as follows: We modify the extend of the partitions shown in Fig.(4) in direction $\mathbf{n}_2$ from N' to one. Thereafter, we perform a third co-partitioning analogous to the sec-

ond one. Fig.(5a) depicts the second co-partitioning after the third one has been performed, where the white stars represent the LPGS-partitions. The extend of the partitions in direction $\mathbf{n}_2$ has been changed from one to $N'/2$. Fig.(5b) illustrates the third co-partitioning. Thereby, we achieved a decrease of the amount of read accesses to the mass storage from $(\frac{N'^2}{6}+1.5N')*10^3$ to $3N'*10^3$. However, the amount of read access to the RAM and mass storage has been increased from $(\frac{N'^2}{3}+N')*10^3$ to $(\frac{N'^2}{3}+2N')*10^3$ in comparison with the previous result shown in Fig.(4). The optimization of the RAM utilization is given by

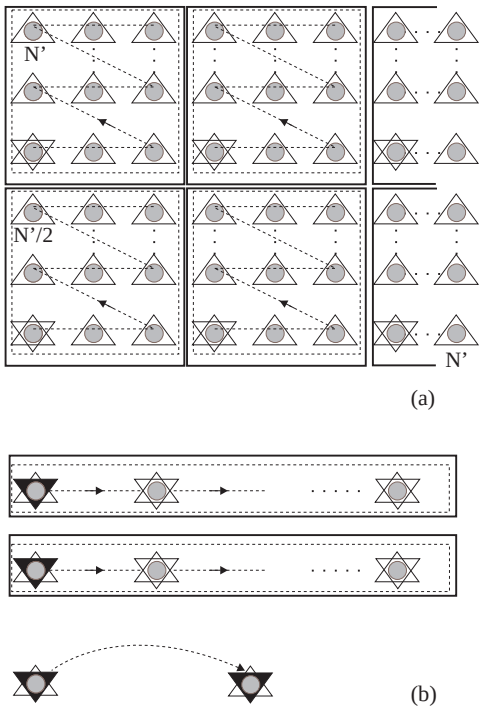

Figure 5: *Improved RAM level utilization by six-fold partitioning*

the determination of LSGP-partitions (in the third co-partitioning) of maximal size, which can realize the internal communication (data transfers within the partition) via the RAM, i.e. without access to the mass storage. Hence, we obtain a partitioning with a hierarchy of six levels (see section 3.5).
The achieved results are recapitulated in Tab.(1).

3 Formalized partitioning

3.1 Basic definitions

We introduce the definitions of the considered system of uniform recurrence equations (URE's), of the mapping onto systolic arrays and of the partitioning.

Definition 1 (System of URE's) *A system of uniform recurrence equations (URE's) on a linearly bounded lattice is a set of equations*

$$y_j[\mathbf{i}] = F(x_1[\mathbf{i}-\mathbf{d}_{x,1}],\ldots,x_l[\mathbf{i}-\mathbf{d}_{x,l}],$$
$$y_1[\mathbf{i}-\mathbf{d}_{y,1}],\ldots,y_m[\mathbf{i}-\mathbf{d}_{y,m}]), j=1,\ldots,m \qquad (1)$$

where the index space $\mathcal{I}$ of these equations is a linearly bounded lattice [1]

$$\mathcal{I} = \{\mathbf{i} \mid \mathbf{i} = \mathbf{B}\mathbf{z}+\mathbf{b}_0,\ \mathbf{A}\mathbf{i} \geq \mathbf{a}_0\}, \qquad (2)$$

$\mathbf{i} \in \mathcal{I}$ are index points and $\mathbf{i}, \mathbf{z}, \mathbf{b_0} \in \mathcal{Z}^n, \mathbf{a_0} \in \mathcal{Q}^m, \mathbf{B} \in \mathcal{Z}^{n\times n}, rank\mathbf{B} = n, \mathbf{A} \in \mathcal{Q}^{m\times n}$. The data dependence vectors $\mathbf{d}_{...} \in \mathcal{D}$ are constant, where $\mathcal{D}$ is the set of all data dependence vectors in (1).

Example: The multiplication $\mathbf{C} = \mathbf{QW}$ of two square matrices of order N is given by:

$$c[\mathbf{i}] = c[\mathbf{i}-(0\ 0\ 1)^t] + q[\mathbf{i}-(0\ 1\ 0)^t] * w[\mathbf{i}-(1\ 0\ 0)^t]$$
$$\mathbf{i} \in \mathcal{I} = \{\mathbf{i} = (i\ j\ k)^t \mid \mathbf{i} = \mathbf{I}\mathbf{z},$$
$$\begin{pmatrix} \mathbf{I} \\ -\mathbf{I} \end{pmatrix} \mathbf{i} \geq (0\ 0\ 0\ (1-N)\ (1-N)\ (1-N))^t\},$$

where $\mathbf{I}$ is the identity matrix of order three. The data dependence vectors are $\mathbf{d}_c = (0\ 0\ 1)^t$, $\mathbf{d}_q = (0\ 1\ 0)^t$, and $\mathbf{d}_w = (1\ 0\ 0)^t$. •

The scheduling of the computation tasks $\mathbf{i} \in \mathcal{I}$ in an array is defined by scalar product with the scheduling vector $\boldsymbol{\tau}$. The allocation of the computation tasks $\mathbf{i} \in \mathcal{I}$ to processors of the array is defined by a linear projection with a projection vector $\mathbf{u}$.

Definition 2 (Mapping) *The mapping M of the algorithm onto the systolic array is of the following form:*

$$M : \mathcal{I} \rightarrow \mathcal{J} \subseteq \mathcal{T} \times \mathcal{P},\ M : \mathcal{D} \rightarrow \mathcal{C} \subseteq \mathcal{T}_v \times \mathcal{V}, \qquad (3)$$

where $\mathcal{J}$ is the time processor space, $\mathcal{T} = \{t|t = \boldsymbol{\tau}\mathbf{i}\}$ is the set of time instances at which the iterations $\mathbf{i}$ are processed, and $\mathcal{P} = \{\mathbf{p}|\mathbf{p} = \mathbf{S}\mathbf{i}\}$ is the processor space, where $\mathbf{S}\mathbf{u} = \mathbf{0}, \mathbf{S} \in \mathcal{Z}^{n-1\times n}, rank\ \mathbf{S} = n-1$. $\mathcal{C}$ is the set of interconnection primitives, where $\mathcal{V} = \{\mathbf{v}|\mathbf{v} = \mathbf{S}\mathbf{d}\}$ is the set of interconnections which have to be realized in the array, and $\mathcal{T}_v = \{t_v|t_v = \boldsymbol{\tau}\mathbf{d}\}$ is the set of time delays associated with the interconnections. The constraints $\forall \mathbf{d} \in \mathcal{D} : \boldsymbol{\tau}\mathbf{d} > 0$ (causality) and $\boldsymbol{\tau}\mathbf{u} \neq 0$ (conflict free) have to be satisfied. The resulting array architecture is called full size array.

[1] This definition represents a special case of the more general definition of linearly bounded lattices in [13] for a full rank matrix B.

We claim, that $|\boldsymbol{\tau}\mathbf{u}|/gcd(\boldsymbol{\tau}\mathbf{B}) = 1$ holds, which is equivalent to the fact that a processing element is active in each clock cycle. The processor space is a linearly bounded lattice.

Example: We achieve the matrix multiplication array shown in Fig.(2), if we apply the scheduling vector $\boldsymbol{\tau} = (1\ 1\ 1)^t$ and a projection vector $\mathbf{u} = (0\ 0\ 1)^t$. The processor space $\mathcal{P}$ is given by

$$\mathbf{p} \in \mathcal{P} = \{\mathbf{p} = (i\ j)^t \mid \mathbf{p} = \mathbf{I}\mathbf{z},$$

$$\begin{pmatrix} \mathbf{I} \\ -\mathbf{I} \end{pmatrix} \mathbf{p} \geq (0\ 0\ (1-N)\ (1-N))^t\},$$

where $\mathbf{I}$ is the identity matrix of order two. The resulting interconnections in the array are $\mathbf{v}_c = (0\ 0)^t$, $\mathbf{v}_q = (0\ 1)^t$, and $\mathbf{v}_w = (1\ 0)^t$. •

In general, constraints in the structure of the array (e.g. number of processing elements) and I/O-demand cannot be taken into account with this mapping procedure. Hence, partitioning techniques have to be applied. In the following we define the subdivision of the algorithm and the array into congruent partitions.

Definition 3 (Partitioning) *The partitioning of the index space $\mathcal{I}$ of a system of URE's is given by the partitions $\mathcal{I}_\pi(\mathbf{i}')$ in the index space together with the set $\mathcal{I}'_\pi$ as follows:*

$$\mathcal{I}_\pi(\mathbf{i}') = \{\mathbf{i} | \mathbf{i} = \mathbf{i}' + \mathbf{G}^*\boldsymbol{\kappa}^*,\ \mathbf{i} \in \mathcal{I},\ 0 \leq \kappa_z < \vartheta_z,$$

$$z = 1, \ldots, n-1\}, \tag{4}$$

where $\kappa_z, \vartheta_z \in \mathcal{Z}, \boldsymbol{\kappa}^* \in \mathcal{Z}^n, \mathbf{G}^* = [\mathbf{G}, \mathbf{u}] = [\mathbf{g}_1 \ldots \mathbf{g}_{n-1}\ \mathbf{u}]$. $\mathbf{G}$ is an integral basis of the hyperplane $\boldsymbol{\tau}\mathbf{i} = 0$, and $\mathbf{i}' = \mathbf{G}\boldsymbol{\kappa}, \boldsymbol{\kappa} \in \mathcal{Z}^{n-1}$. The set $\mathcal{I}'_\pi$ of (reference) vectors $\mathbf{i}'$ is given by

$$\mathcal{I}'_\pi = \{\mathbf{i}' | \mathbf{i}' = \mathbf{i}_{zero} + \mathbf{G}\boldsymbol{\Theta}\mathbf{c},\ \mathbf{L}\mathbf{c} \geq \mathbf{l},\ \mathbf{c} \in \mathcal{Z}^{n-1},$$

$$\mathbf{i}_{zero} = \mathbf{G}\boldsymbol{\kappa},\ \boldsymbol{\kappa} \in \mathcal{Z}^{n-1}\} \tag{5}$$

with $\boldsymbol{\Theta} = diag(\vartheta_i)$, and the polyhedron $\mathbf{L}\mathbf{c} \geq \mathbf{l}$ is given by the successive projection of the convex hull of

$$\begin{pmatrix} \mathbf{A}\mathbf{G}^* & \mathbf{A}\mathbf{G}\boldsymbol{\Theta} \\ \mathbf{I} & \mathbf{0} \\ -\mathbf{I} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \kappa_1 \\ \vdots \\ \kappa_n \\ \mathbf{c} \end{pmatrix} \geq \begin{pmatrix} \mathbf{a}_0 - \mathbf{A}\mathbf{i}_{zero} \\ \mathbf{0} \\ -(\boldsymbol{\vartheta} - \mathbf{1}) \end{pmatrix} \tag{6}$$

along the κ_j-axes, $j = 1, \ldots, n$, i.e. by Fourier-Motzkin elimination, where $\boldsymbol{\vartheta} - \mathbf{1} = (\vartheta_1 - 1 \ldots \vartheta_{n-1} - 1)^t$, and $\mathbf{I} \in \mathcal{Z}^{(n-1)\times(n-1)}$ is the identity matrix.

Hence, the processor space $\mathcal{P}$ is subdivided into partitions

$$\mathcal{P}_\pi(\mathbf{p}') = \{\mathbf{p} | \mathbf{p} = \mathbf{p}' + \mathbf{N}\boldsymbol{\kappa},\ \mathbf{p} \in \mathcal{P},\ 0 \leq \kappa_z < \vartheta_z,$$

$$z = 1, \ldots, n-1\} \tag{7}$$

with $\mathbf{N} = [\mathbf{n}_1 \ldots \mathbf{n}_{n-1}]$, $\mathbf{n}_j = \mathbf{S}\mathbf{g}_j$, and $\mathbf{p}' \in \mathcal{P}'_\pi$,

$$\mathcal{P}'_\pi = \{\mathbf{p}' | \mathbf{p}' = \mathbf{S}\mathbf{i}_{zero} + \mathbf{N}\boldsymbol{\Theta}\mathbf{c},\ \mathbf{L}\mathbf{c} \geq \mathbf{l}\}. \tag{8}$$

Example: Let us consider the partitioning of the processor space symbolized by the small rectangles in Fig.(3). The parameters are $\mathbf{N} = (\mathbf{n}_1\ \mathbf{n}_2)$ and $\vartheta_1 = \vartheta_2 = 2$, $\boldsymbol{\Theta} = diag(\vartheta_i)$. •

3.2 Foreground memory utilization

A two-fold partitioning, called co-partitioning [4], is necessary for the scaling of the structure of the array and for the balancing of the size of the foreground memory and of the I/O-access behaviour of the array to the peripheral system (background memory). The co-partitioning is a simultaneous LPGS- and LSGP-partitioning. The LPGS-partitioning leads to an array of fixed size, which is independent of the algorithm's size. However, for a fixed number of processing elements exists a minimum for the volume of data which has to be exchanged in one clock cycle with the peripheral system. If this minimum exceeds the I/O-capacity, then wait cycles have to be introduced in the case of a pure LPGS-partitioning. An LSGP-partitioning of the LPGS-partitioned array serves in the co-partitioning for the elimination of this effect. The interesting property of the LSGP-scheme is that it allows a tuning of the I/O-demand by tuning the size of the partitions. The co-partitioning is defined as follows:

Definition 4 (Co-partitioning)
The co-partitioning of the algorithm's index space $\mathcal{I}$ of an system of URE's is given by the following co-partitions in the index space:

$$\mathcal{I}_\pi(\mathbf{i}') = \{\mathbf{i} | \mathbf{i} = \mathbf{i}' + \kappa_n\mathbf{u} + \mathbf{G}(\boldsymbol{\Theta}^{LS}\boldsymbol{\kappa}^{LP} + \boldsymbol{\kappa}^{LS}),$$

$$\mathbf{i} \in \mathcal{I},\ \mathbf{i}' \in \mathcal{I}'_\pi\}, \tag{9}$$

where $\boldsymbol{\Theta}^{LS} = diag(\vartheta_j^{LS})$, $0 \leq \kappa_j^{LS} < \vartheta_j^{LS}$, $0 \leq \kappa_j^{LP} < \vartheta_j^{LP}, j = 1, \ldots, n-1$, and by the set $\mathcal{I}'_\pi = \{\mathbf{i}'\}$, which is defined according to equations (5) and (6), where $\boldsymbol{\Theta} = \boldsymbol{\Theta}^{LP}\boldsymbol{\Theta}^{LS}$.
The co-partitioning of the processor space $\mathcal{P}$ is given by the co-partitions in the processor space as follows:

$$\mathcal{P}_\pi(\mathbf{p}') = \{\mathbf{p} | \mathbf{p} = \mathbf{p}' + \mathbf{N}(\boldsymbol{\Theta}^{LS}\boldsymbol{\kappa}^{LP} + \boldsymbol{\kappa}^{LS}),$$

$$\mathbf{p} \in \mathcal{P},\ \mathbf{p}' \in \mathcal{P}'_\pi\}, \tag{10}$$

and by the set $\mathcal{P}'_\pi = \{\mathbf{p}'\}$ *, which is defined according to equation (8), where* $\mathbf{\Theta} = \mathbf{\Theta}^{LP}\mathbf{\Theta}^{LS}$*. We apply here the abbreviations LP instead of LPGS and LS instead of LSGP.*

The structure of the co-partitioned array is given by the LPGS-parameters. The transformation of the internal interconnection network of the co-partitioned array is omitted here for the sake of brevity (see [5]).

Example: Let us consider the partitioning of the processor space in Fig.(3). The small rectangles symbolize the LSGP-partitions. The parameters are $\mathbf{N} = (\mathbf{n}_1\ \mathbf{n}_2)$ and $\vartheta_1^{LS} = \vartheta_2^{LS} = 2$, $\mathbf{\Theta}^{LS} = diag(\vartheta_i^{LS})$. The large rectangles symbolize the LPGS-partitions. The parameters are $\vartheta_1^{LP} = \vartheta_2^{LP} = 2$, $\mathbf{\Theta}^{LP} = diag(\vartheta_i^{LP})$. The tasks of the four processing elements of the LSGP-partitions are scheduled sequentially on the black drawn processing element. The shaded processing elements are not implemented. The LPGS-partitions (i.e. large rectangles) are scheduled sequentially. Hence, the resulting array consists of two times two processing elements. •

The size L_{loc} of the local memory (i.e. the number of registers) of each processing element of the array imposed by the co-partitioning is directly proportional to the size of the LSGP-partitions. The size is given by $L_{loc} = |\mathcal{D}| \det \mathbf{\Theta}^{LS}$ (if $\boldsymbol{\tau}\mathbf{d}$=1). The I/O-behaviour between the array and the peripheral system has been considered in [4]. The number of variables which have to be transferred between the array and the background memory in the time interval required for the processing of a computation $\mathbf{i} \in \mathcal{I}$ is indirectly proportional to the size $\det \mathbf{\Theta}^{LS}$ of the LSGP-partitions. Hence, we have to search a trade-off between the implemented size L_{loc} of the local foreground memory and the access behaviour of the array and to the background memory.

3.3 Nested partitioning

As we have seen in the example given in section 2, a three-fold co-partitioning is necessary to scale the utilization of the cache and RAM level. We introduce a generalized hierarchical partitioning, called nested-partitioning, to formalize the proposed method. Each system level j is constructed of two sub-levels according to the co-partitioning policy. Hence, a mapping of an algorithm onto a system with a hierarchy of h levels requires a partitioning into $q_{max} = 2h$ sub-levels.

Definition 5 (Nested-Partitioning Scheme)
The nested-partitioning scheme is defined by the partitions $\mathcal{I}_\pi^{(q)}$ *at each sublevel and the set* $\mathcal{I}_\pi^{(q_{max}+1)}$*. A partition at sublevel q is defined by:*

$$\mathcal{I}_\pi^{(q)} = \{\mathbf{i}^{(q)} | \mathbf{i}^{(q)} = \mathbf{i}^{(q+1)} + \kappa_n \mathbf{u} + \mathbf{G} \sum_{j=1}^{q} \left(\prod_{l=0}^{j-1} \mathbf{\Theta}^{(l)} \right) \boldsymbol{\kappa}^{(j)}, \mathbf{i}^{(q)} \in \mathcal{I}\}, \tag{11}$$

with $0 \leq \kappa_z^{(j)} < \vartheta_z^{(j)}, z = 1, ..., n-1$, $\mathbf{\Theta}^{(j)} = diag(\vartheta_z^{(j)})$, $\mathbf{\Theta}^{(0)} = \mathbf{I}$*. The set of (reference) points* $\mathcal{I}_\pi^{(q_{max}+1)} = \{\mathbf{i}^{(q_{max}+1)}\}$ *is given according to (5), where* $\boldsymbol{\vartheta}$ *in the vector on the right hand side in (6) has to be replaced by* $\boldsymbol{\vartheta}^{(q_{max})} = \left(\prod_{l=1}^{q_{max}} \vartheta_1^{(l)} \ldots \prod_{l=0}^{q_{max}} \vartheta_{n-1}^{(l)} \right)^t$*.*
We define that all LSGP-paramaters in (11) are represented by the odd indices q and the LPGS-parameters by the even indices q .

Now, we can turn to the consideration of the cache and RAM level utilization.

3.4 Cache utilization

As we have seen in the example given in section 2, we have to apply a two-fold co-partitioning, i.e. a nested-partitioning according to (11) with $q = 4$, for an improvement of the cache utilization. The parameter set is given by $\mathbf{\Theta}^{(1)} = \mathbf{\Theta}^{LS}, \mathbf{\Theta}^{(2)} = \mathbf{\Theta}^{LP}, \mathbf{\Theta}^{(3)} = diag(\vartheta_i^{(3)})$ and $\mathbf{\Theta}^{(4)} = \mathbf{I}$. We obtain $\vartheta_1^{(3)} = 3$ and $\vartheta_2^{(3)} = N$ for the example shown in Fig.(4). Generally, there exist $(n-1)!$ feasible loop nests (i.e. interchanges) for the sequential scheduling within the LSGP-partitions. For the example shown in Fig.(4), we have the feasible loop nests $(1,2)$ and $(2,1)$, where the left number represents the index of the outer loop. We chose the loop nest $(2,1)$, i.e. the loop associated with $\mathbf{n}_1$ is the inner loop and the loop associated with $\mathbf{n}_2$ is the outer loop. With a load $L(\mathbf{n}_j) = 10^3$ for a step in direction $\mathbf{n}_j, j = 1,2$, we obtain for the loop nest $(2,1)$ a load $\hat{L} = L(\mathbf{n}_1) + L(\mathbf{n}_2)\vartheta_1^{(3)}$. If this load has to be exchanged only via a associated cache of the size $L_{cache} = 4 * 10^3$, then we have to ensure, that $L_{cache} \geq \hat{L}$. Hence, we achieve the condition $\vartheta_1^{(3)} \leq 3$. The parameter $\vartheta_2^{(3)}$ is only bounded by the extend of the processor space [3]. For the considered example follows $\vartheta_2^{(3)} = N$.
We associate to each direction $\mathbf{n}_j$ a weighting function w_j, which depends on the selected loop nest and on the size of the LSGP-partitions of the second co-partitioning (see Fig.(4)). Hence, we obtain the condition

$$L_{cache} \geq \sum_{j=1}^{n-1} w_j L(\mathbf{n}_j) \tag{12}$$

for the selection of the parameters $\vartheta_i^{(3)}$ for a given cache size L_{cache}. The term on the right hand side of (12) represents the maximum number of variables reused during the processing of a partition in the second co-partitioning. Condition (12) ensures that the internal communication (data transfers within the partition) can be realized via the cache, i.e. without access to the RAM or mass storage.
The order of the $m = n-1$ loops in a loop nest is defined by the sequence of numbers $o(i), i = 1,\ldots,m$, where $o(1)$ represents the outermost loop and $o(j)$ represents a more outer loop than $o(j+1)$. Furthermore holds {o(i), i=1,...,m}={1,...,m}. The weighting functions w_j are given by:

$$w_{o(i)} = \prod_{j=i+1}^{m+1} \vartheta_{o(j)}^{(3)}, \quad \vartheta_{o(m+1)}^{(3)} := 1 \tag{13}$$

Example: For a loop order $o(1) = 2, o(2) = 1$, $o(3) = 3$, the weighting functions

$$w_{o(1)=2} = \vartheta_{o(2)}^{(3)} * \vartheta_{o(3)}^{(3)}, \; w_{o(2)=1} = \vartheta_{o(3)}^{(3)}, \; w_{o(3)=3} = 1$$

result. It follows

$$\hat{L} = L(\mathbf{n}_1)\vartheta_3^{(3)} + L(\mathbf{n}_2)\vartheta_1^{(3)} * \vartheta_3^{(3)} + L(\mathbf{n}_3)$$

for the load. •

The numerical determination of the loads L has been given in [3]. Obviously, there may exist interconnections which are a linear combination of the $\mathbf{n}_j$. The derivation of the weighting functions w for such an interconnection has been given in [3].
Furthermore, we have to consider the case, that for all loads L hold $L > L_{cache}$. Then, there exist no solution of (12) for the case of a partitioning in the processor space. We can replace the condition (12) by

$$|L_{cache} - \sum_{j=1}^{n-1} w_j L(\mathbf{n}_j)| \to \min \tag{14}$$

or we include the additional partitioning of the time axis, i.e. of the $\mathbf{u}$-axis.
The objective function in the improvement of the cache utilization is the minimization of the access to the RAM and mass storage. Hence, we have to claim that the size of the LSGP-partitions in the second co-partitioning (Fig.(4)) becomes maximum, i.e. $\prod_{i=1}^{n-1} \vartheta_i^{(3)} \to \max$. That means, that we are looking for the largest for-loop programs, which satisfy condition (12) or (14). If we include the foreground memory in the consideration, then we obtain the objective function $\prod_{j=1}^{3}\prod_{i=1}^{n-1} \vartheta_i^{(j)} \to \max$.

3.5 RAM utilization

We have to apply a three-fold co-partitioning, i.e. a nested-partitioning according to (11) with $q = 6$, for an improvement of the RAM utilization. The additional parameter set is given by $\mathbf{\Theta}^{(5)} = diag(\vartheta_i^{(5)})$ and $\mathbf{\Theta}^{(6)} = \mathbf{I}$. The parameter $\vartheta_2^{(3)}$, which was only bounded by the extend of the algorithm's index space in the above discussed improvement of the cache utilization becomes now a variable. This variable reflects the increase of the data traffic between the cache and the RAM level, which occurs with the improvement of the RAM utilization. We obtain $\vartheta_2^{(3)} = N'/2$, $\vartheta_1^{(5)} = N'$ and $\vartheta_2^{(5)} = 1$ for the example shown in Fig.(5). The determination of the parameters occurs analogous to the cache level. We obtain the conditions

$$L_{RAM} \geq \sum_{j=1}^{n-1} w_j^{(R)} L^{(R)}(\mathbf{n}_j), \tag{15}$$

$$|L_{RAM} - \sum_{j=1}^{n-1} w_j^{(R)} L^{(R)}(\mathbf{n}_j)| \to \min \tag{16}$$

analogous to (12) and (14), where L_{RAM} represents the available size of the RAM. The objective function in the improvement of the RAM utilization is the minimization of the access to the mass storage. Hence, we have to claim that the size of the LSGP-partitions in the third co-partitioning (Fig.(5)) becomes maximum, i.e. $\prod_{i=1}^{n-1} \vartheta_i^{(5)} \to \max$. That means, that we are looking for the largest for-loop programs, which satisfy condition (15) or (16). If we include the foreground memory and cache in the consideration, then we obtain the objective function $\prod_{j=1}^{5}\prod_{i=1}^{n-1} \vartheta_i^{(j)} \to \max$. Since we intend to improve the utilization of the cache and RAM, we have to include here as second objective function $\prod_{i=1}^{n-1} \vartheta_i^{(3)} \to \max$. This represents the cache utilization as discussed above.

Obviously, the example given in section 2 can also be solved by a two-fold co-partitioning, where the optimization of the cache and RAM utilization has to be realized by the parameters $\vartheta_i^{(3)}$. However, the optimization of the RAM utilization can be realized without an influence on the cache utilization for algorithms with an index space of dimension $n > 3$. One parameter $\vartheta_i^{(3)}$ is bounded by the extend of the algorithms index space. Therefore, the optimization of the cache utilization causes a reduction of the dimension of the for-loop program, which represents the scheduling of the co-partitions, by one. (cf. Fig.(4b) symbolizes the one-dimensional scheduling of the two-dimensional co-partitions Fig.(4a).) We obtain a for-loop program,

which represents the scheduling of the co-partitions after the second co-partitioning, of dimension $n \geq 2$ for an algorithm with index space of dimension $n > 3$. Such a program of dimension $n \geq 2$ can be partitioned for the improvement of the RAM utilization analogous to the cache problem without an influence onto the parameters $\vartheta_i^{(3)}$. Hence, the application of a six-fold partitioning for the RAM utilization establishes the more general model.

4 Concluding Remarks

The proposed memory utilization procedure can be stated as follows: The size of the implemented foreground memory (number of registers) defines the number of accesses of the array (or single processor) to the background memory in given time interval. This dependency has been discussed in [4]. The foreground memory allocation is realized by a co-partitioning. A repeated co-partitioning of the resulting program is performed to minimize the number of accesses to the RAM and mass storage level during the processing of the algorithm. This number of accesses depends directly on the size of the implemented cache memory. A third co-partitioning can be performed to minimize the number of accesses to the mass storage level during the processing of the algorithm. This number of accesses depends directly on the size of the implemented RAM memory.

The numerical values for the number of accesses to the different memory levels during the processing of the algorithm can be achieved by means of the formulas given in [4, 5] (see [3]). The computation of the I/O access pattern of the hierarchical partitioned for-loop program has been given in [2]. The program transformation of the initially given for-loop program by co-partitioning has been given in [4].

We proposed a mathemtical framework for the hierarchical partitioning of systems of URE's. However, the most algorithms represents conditional URE's. The index space of conditional URE's is the union of all conditional spaces. The parameters of the partitioning have to be adjusted to the worst case over all conditional spaces to achieve a homogeneous partitioning for the whole algorithm.

References

[1] J. Bu. *Systematic design of regular VLSI processor arrays*. PhD thesis, Delft University of Technology, 1990.

[2] U. Eckhardt. I/O management for hierarchically structured array architectures. In *Proc. 5th Euromicro Workshop on Parallel and Distributed Processing*, pages 205 – 210, 1997.

[3] U. Eckhardt. Memory management by partitioning. Technical Report SFB 358-A1-1/97, Deutsche Forschungsgemeinschaft, TU Dresden, 1997.

[4] U. Eckhardt and R. Merker. Co-partitioning- a method for hardware/software codesign for scalable systolic arrays. In R. Hartenstein and V. Prasanna, editors, *Reconfigurable Architectures*, pages 131 – 138. ITpress, 1997.

[5] U. Eckhardt and R. Merker. Scheduling in co-partitioned array architectures. In *Proc. Int. Conf. on Application-Specific Array Processors (ASAP)*, 1997.

[6] Y.-T. Hwang and Y. Hu. Novel scheduling scheme for systolic array partitioning problem. In *Proc. IEEE Workshop on VLSI Signal Processing*, pages 355 – 364, 1992.

[7] K. Jainandunsing. Optimal partitioning scheme for wavefront/systolic array processors. In *Proc. IEEE Symp. on Circuits and Systems*, 1986.

[8] R. Karp, R. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *Journal of the ACM*, 14(3):563 – 590, July 1967.

[9] D. Moldovan and J. Fortes. Partitioning and mapping of algorithms into fixed-size systolic arrays. *IEEE Trans. Computers*, C35(1):1 – 12, Jan. 1986.

[10] J. Moreno and T. Lang. Matrix computations on systolic-type meshes. *IEEE Computer*, 23(4):32 – 51, April 1990.

[11] J. Navarro, J. Llaberia, and M. Valero. Partitioning: an essential step in mapping algorithms into systolic array processors. *IEEE Computer*, 20(7):77 – 89, July 1987.

[12] H. Nelis and E. Deprettere. Automatic design and partitioning of systolic/wavefront arrays for VLSI. *Circuits systems signal processing*, 7(2):235 – 252, 1988.

[13] J. Teich. *A compiler for application specific processor arrays*. PhD thesis, Verlag Shaker, 1993.

[14] J. Teich and L. Thiele. Partitioning of processor arrays: a piecewise regular approach. *INTEGRATION, the VLSI journal*, 14(3):297 – 332, 1993.

[15] J. Teich, L. Thiele, and L. Zhang. Scheduling of partitioned regular algorithms on processor arrays with constrained resources. In *Proc. Int. Conf. on Application-Specific Array Processors (ASAP)*, 1996.

[16] L. Thiele and V. Roychowdhury. Systematic design of local processor arrays for numerical algorithms. In E. Deprettere and A.-J. van der Veen, editors, *Algorithms and parallel VLSI architectures*, volume A, pages 329–339. Elsevier Science Publisher B.V., 1991.

[17] I. Verbauwhede, F. Catthoor, J. Vandevalle, and D. H. Background memory synthesis for algebraic algorithms on multiprocessor DSP chips. In *VLSI 89*, pages 209–218. Elsevier Science Publisher B.V.

[18] I. Verbauwhede, F. Catthoor, J. Vandevalle, and D. H. In-place memory management of algebraic algorithms on application specific ics. In *Algorithms and Parallel VLSI Architectures*, pages 353–362. Elsevier Science Publisher B.V.