

# Systematic Data Reuse Exploration Methodology for Irregular Access Patterns

Tanja Van Achteren, Rudy Lauwereins  
K.U.Leuven ESAT-ACCA  
Kard. Mercierlaan 94, B-3000 Leuven, Belgium  
Tanja.VanAchteren@esat.kuleuven.ac.be  
Rudy.Lauwereins@esat.kuleuven.ac.be

Francky Catthoor  
IMEC  
Kapeldreef 75, B-3000 Leuven, Belgium  
catthoor@imec.be

## Abstract

*Efficient use of an optimized custom memory hierarchy to exploit temporal locality in the memory accesses on array signals can have a very large impact on the power consumption in embedded data dominated applications. Only recently effective formalized techniques to deal with this specific task have been addressed. They work well for homogeneous signal access patterns but cannot handle other cases. In this paper we will extend and parameterize the design space and establish heuristics for an efficient exploration, such that better results in terms of area and power can be achieved for applications where holes are present in the signal access pattern. The extended methodology will be illustrated for several real-life image processing algorithms.*

## 1. Introduction

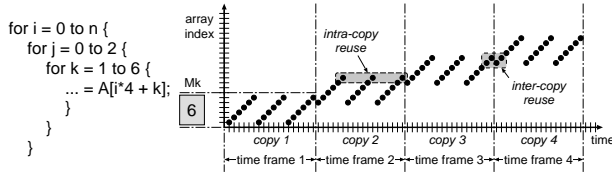
The idea of using memory hierarchy to minimize the power consumption, is based on the fact that memory power consumption depends primarily on the access frequency and the size of the memory. Power savings can be obtained by accessing heavily used data from smaller memories instead of from large background memories. Such an optimization requires architectural transformations that consist of adding layers of smaller and smaller memories to which frequently used data will be copied [15]. So, memory hierarchy optimization has to introduce copies of data from larger to smaller memories in the Data Flow Graph (DFG). This means that a trade-off exists here: on the one hand, power consumption is decreased because data is now read mostly from smaller memories, while on the other hand, power consumption is increased because extra memory transfers are introduced. Moreover, adding another layer of hierarchy can also have a negative effect on the area and interconnect cost, and as a consequence also on the power. It is the task of the data reuse decision step to find the best solution for this trade-off. This is required for a custom hierarchy, but also for efficiently using a predefined memory hierarchy with SW cache control, where we have the design freedom to copy data at several moments with different copy sizes.

The main related work to this problem lies in the parallel compiler area, especially related to the cache hierarchy [1]. This, however is not resulting yet in any formal-

izable method to guide the memory organization issues. In most work on parallel MIMD processors, the emphasis in terms of storage hierarchy has been on hardware mechanisms based on cache coherence protocols [8]. Partitioning or blocking strategies for loops to optimize the use of caches has been studied in several contexts [5]. The main focus is on CPU performance improvement though and not on memory related energy and area cost. Also in a system synthesis context, applying transformations to improve the cache usage has been addressed [4][13]. None of these approaches determine the best memory hierarchy organization for a given (set of) applications and they do not address the power cost. Only an analysis of memory hierarchy choices based on statistical information to reach a given throughput expectation has been discussed [3]. In the hardware realization context, much less work has been performed, mainly oriented to memory allocation [12][10][6]. The impact of this step turns out to be very large in the entire methodology for power reduction. This has been demonstrated for a H.263 video decoder [9], a motion estimation application [16] and a 3D reconstruction algorithm [17].

At IMEC, a formalized methodology for data reuse exploration [14] has been developed as part of the ATOMIUM script [7] for data transfer and storage exploration. In this methodology assumptions were made to reduce the search space of possible memory hierarchies. However, some of these assumptions can lead to less optimal results for the power and area cost, when applied on an important class of applications where a “hole” is present in the access pattern of a multi-dimensional signal. Examples of real-life image processing algorithms where such holes exist are the “Binary Tree Predictive Coder” and “Cavity Detection” discussed further on. In this paper we will introduce an example to illustrate the problem, and deduce the required parameters to describe a more complete search space. The relationships between these parameters and the relevant cost variables are explored, which help us to establish heuristics to steer the search for a good solution. This is our main contribution.

The rest of this paper is organized as follows. In Section 2 we will give an overview of the current methodology for data reuse exploration. Section 3 defines a detailed power cost function for a memory hierarchy. Section 4 will give a simple example of an application with a hole in the access pattern. We will extend the search space definition to deal with this problem. How to steer our exploration of the huge resulting search space is explained in Section 5. In



**Figure 1. Exploiting data reuse local in time to save power.**

Section 6 we will give results on the application of these extensions on some real-life applications. Section 7 concludes the paper.

## 2. Current methodology for data reuse exploration

In this Section we give a short summary of the current methodology for data reuse exploration [14][7].

### 2.1. Exploiting temporal locality

Memory hierarchy design exploits data reuse local in time to save power. This happens by copying data that is reused often in a short amount of cycles to a smaller memory, from which the data can then be accessed. This is shown in Fig. 1 for a simple example for all read operations to a given array. Note that loops have been represented here as groups of dots. Every dot represents a memory read operation accessing a given array element.

The horizontal axis is the time axis. It shows how the data accesses are ordered relatively to each other in time. This is not necessarily an absolute time assignment. The vertical axis shows the index of the data element (scalar) that is accessed from the array. It can be seen that, in this example, most values are read multiple times. Over a very large time-frame all data values of the array are read from memory, and therefore they have to be stored in a large background memory. However, when we look at smaller time-frames (indicated by the vertical dashed lines), we see that only part of the data is needed in each time-frame, so it would fit in a smaller, less power consuming memory. If there is sufficient reuse of the data in that time-frame, it can be advantageous to copy the data that is used frequently in this time-frame to a smaller memory, such that from the second usage on, a data element can be read from the smaller memory instead of the larger memory.

### 2.2. Definition of data reuse factor

The usefulness of a memory hierarchy, especially if it is customized to the application as proposed in [14], is strongly related to the signal reusability, because this is what determines the ratio between the number of read operations from a copy of a signal in a smaller memory, and the number of read operations from the signal in the larger memory on the next hierarchical level.

The reuse factor of a group of data  $D$  stored in a memory on layer  $i$  relative to layer  $i - 1$ , where layer 0 is the memory furthest from the data paths, is defined as:

$$F_R(i - 1, i, D) = \frac{N_R(M_i, D)}{N_R(M_{i-1}, D)} \quad (1)$$

Here  $N_R(M_i, D)$  is the total number of times an element from  $D$  is read from a memory at layer  $i$ .

A reuse factor larger than 1 is the result of **intra-copy reuse** and **inter-copy reuse** (cfr. Figure 1). *Intra-Copy reuse* means that each data element is read several times from memory during one time-frame. *Inter-Copy reuse* means that advantage is taken from the fact that part of the data needed in the next time-frame could already be available in the memory from the previous time frame, and therefore does not have to be copied again. If  $F_R(i - 1, i, D) = 1$ , this sublevel is useless and would even lead to an increase of area and power, because the number of read operations from  $M_{i-1}$  would remain unchanged while the data also has to be stored and read from  $M_i$ .

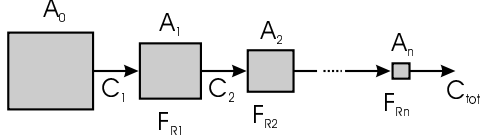
### 2.3. Assumptions

1. Up to now we assumed single assignment code, where every array value can only be written once but read several times. This makes the analysis much easier.
  2. We assume that the order and direction of loop iterators of nested loops is fixed. The ordering of sequential loops relative to each other is still free. This is compatible with the position of the data reuse decision step in our complete ATOMIUM memory management methodology [7].
  3. It is assumed that the optimal time-frame boundaries are likely to coincide with the loop boundaries of loop nests.
  4. A copy-candidate contains only the data which is accessed during its time-frame. At the end of a time-frame, the data copied into the intermediate memory is replaced by the data needed in the next time-frame.
  5. Inter-copy reuse is fully exploited. Data already present in the smaller memory from the previous copy will not be copied again. This affects the number of writes to each of the copy-candidates.
- In Section 4 it will be shown that the third and fourth assumption can lead to less optimal results in terms of area and power cost for a certain kind of applications, where there is a "hole" in the access pattern of the signal.

### 2.4. Current Methodology

The current data reuse methodology is based on the assumptions listed in the previous subsection.

For each read instruction inside a loop nest, a copy-candidate chain can be determined. The first copy-candidate in the chain contains all data elements accessed by the given read instruction during the execution of the complete loop nest. The time-frame of the second copy-candidate corresponds to the iterations of the outer loop. The data elements accessed during one iteration of the outer loop are stored in the copy-candidate. This can be repeated for the remaining loop nest levels. Together these copy-candidates form *the copy-candidate chain* for the considered read instruction. An example of a copy-candidate chain for  $n+1$  nested loops is given in Fig. 2.



**Figure 2. Copy-candidate chain with area sizes  $A_j$ , data reuse factors  $F_{Rj}$  and number of writes  $C_j$**

An optimal subset of this copy-candidate chain can be selected, based on the possible gain acquired by introducing an intermediate copy-candidate, trading off the lower cost (speed & power) of accesses to a smaller memory and the higher cost of the extra memory area.

## 2.5. Cost function

Let  $P_j(N_{bits}, N_{words}, F_{access})$  be the power function for read and write operations to a level  $j$  in a copy-candidate chain, which depends on the estimated size and bitwidth of the final memory, as well as on the real access frequency  $F_{access}$  corresponding to the array signals, which is obtained by multiplying the number of memory accesses per frame for a given signal with the frame rate  $F_{frame}$  of the application (this is **not** the clock frequency). The total cost function for a copy-candidate chain with  $n$  sublevels is defined as:

$$F_c = \sum_{j=1}^n [\alpha P_j(N_{bits}, N_{words}, F_{access}) + \beta \cdot A_j(N_{bits}, N_{words})] \quad (2)$$

Here  $\alpha$  and  $\beta$  are weighting factors for Area/Power trade-offs.

## 3. Power cost function

In this Section we explore the power cost function in more detail. The power cost function  $P_j(N_{bits}, N_{words}, F_{access})$  for one level  $j$  can be written as:

$$P_j = \frac{\#reads}{frame} * P_j^r + \frac{\#writes}{frame} * P_j^w \quad (3)$$

where  $P_j^{r(w)}$  is defined as

$$P_j^{r(w)} = F_{frame} * \frac{Energy}{1 \text{ read(write)}} \quad (4)$$

In Fig. 2 a generic copy-candidate chain is shown where for each level  $j$ , we can determine an area size  $A_j$ , a number of writes to the level  $C_j$  (equal to the number of reads from level  $(j-1)$ ) and a data reuse factor  $F_{Rj}$  defined as

$$F_{Rj} = \frac{C_{tot}}{C_j} \quad (5)$$

$C_{tot}$  is the total number of reads from the signal in the last level in the hierarchy. This is a constant independent of

the introduced copy candidate chain. The number of writes  $C_j$  is a constant for level  $j$ , independent from the presence of other levels in the hierarchy. As a consequence we can determine  $F_{Rj}$  as the fraction of the number of reads from level  $j$  to the number of writes to level  $j$ , as it would be the only sublevel in the hierarchy.

Note that this definition of the data reuse factor is different from the one in the current methodology in Section 2.2, where it is equal to the number of reads from the level itself (instead of the last level) to the number of writes to the level. Since the number of reads from a level depends on the level below, the data reuse factor as defined in the current methodology was not a constant independent from the presence of other levels. As we will see further on, this new definition enables us to define the power cost function in such a way that the cost contribution of each introduced sublevel is clearly visible. Since data reuse factors are independent of the presence of other copy-candidates, they have not to be recomputed for each different hierarchy.

Based on the previous definitions we can deduce the following expression for the total power cost for a complete copy candidate chain of  $n$  sublevels:

$$\begin{aligned} P &= C_1(P_0^r + P_1^w) + C_2(P_1^r + P_2^w) + \dots + C_{tot}(P_n^r) \\ &= C_{tot} \left[ \frac{C_1}{C_{tot}} (P_0^r + P_1^w) + \frac{C_2}{C_{tot}} (P_1^r + P_2^w) + \dots + (P_n^r) \right] \\ &= C_{tot} \left[ \frac{1}{F_{R1}} (P_0^r + P_1^w) + \frac{1}{F_{R2}} (P_1^r + P_2^w) + \dots + (P_n^r) \right] \end{aligned} \quad (6)$$

From equation (6) we learn that smaller area sizes  $A_j$ , which reduce  $P_j^{r(w)}$ , and higher data reuse factors  $F_{Rj}$  reduce the total power cost of a copy candidate chain.

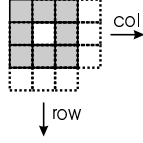
## 4. A hole in the picture

In this Section we will give a simple example of an application where assumptions made in the current methodology on the properties of a copy-candidate lead to non-optimal results for the power and area cost. We will identify the parameters which allow us to explore a larger search space, leading to a re-definition of the reuse factors for a copy-candidate.

### 4.1. A realistic example

In Fig. 3 an example in C code is given, where a square mask is moved over an image in vertical and horizontal direction. A pixel in the output image is based on the *surrounding* pixels of the input image and not the reference pixel itself, i.e. there is a *hole* in the access pattern of the two inner loops, as indicated by the arrow in the code. A data access graph of the input image can be constructed showing the accessed index values as a function of relative time. The graph for three subsequent mask iterations is illustrated in Fig. 4. We define a data reuse dependency (DRD) as the time interval between two accesses to the same index, represented in the graph by grey horizontal lines started and ended by a bullet representing the accesses.

In the current methodology we fix the time-frame boundaries defining the copy-candidates on the iteration boundaries [14], shown in the graph as full vertical lines. One



```

for (col=0; col<(MAXCOL-2); col++)
for (row=0; row<(MAXROW-2); row++)
{
  outpix = 0.0;
  for (i=0; i<MAXMASK; i++)
  for (j=0; j<MAXMASK; j++)
    if ((i!=1) || (j!=1)) ←
      outpix += mask[i][j] * inim[row+i][col+j];
  outim[row][col] = (int)(outpix/2.0) + 127;
}

```

Figure 3. Example C code

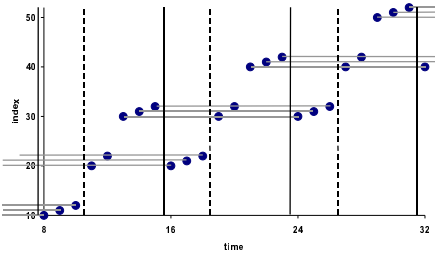


Figure 4. Data access graph for 3 subsequent mask iterations, showing the influence of the time-frame on area size and data reuse factors

copy-candidate fits the data accessed during this time-frame, in this case 8 different indices. Data, not present in the copy-candidate during the previous time-frame has to be copied from a higher level. Data, present in the previous time-frame but not accessed in the current time-frame, is discarded. In this way, the reference pixel (the *hole*) in the current time-frame, which was accessed in the previous time-frame, is discarded. However this index is accessed again in the *next* time-frame and has to be copied again from a higher level. By keeping this index in the copy-candidate over *multiple* time-frames until the next access, we can reduce the global number of writes to the copy-candidate, which results in a higher data reuse factor, but at the same time increases the area size to 9 since more data is kept in the copy-candidate. However, when we move the time-frame with an offset of 3, shown in Fig. 4 by dashed vertical lines, the number of indices kept in the copy-candidate (or the area size) is only 6, while retaining the higher data reuse factor. In general, we also have the freedom to change the size of the time-frame, i.e. the number of reads during a time-frame. In this specific example, no more gain for area and power cost can be achieved though.

As illustrated by this example, some assumptions made in the current methodology can lead to less optimal results for power and area. Namely, it was assumed that time-frames are determined by loop boundaries, and that only the data accessed during the current time-frame is kept in

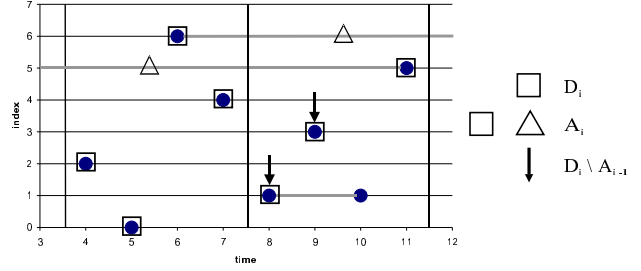


Figure 5. Definition of  $D_i$ ,  $A_i$ ,  $\#(D_i \setminus A_{i-1})$ : graphical representation in a data access graph

the copy-candidate.

By varying the following three parameters, a larger search space is explored:

- the time-frame size  $T_{timeframe}$
- the time-frame *offset*
- the included data reuse dependencies (*DRD*)

The last parameter defines whether and during which time intervals a certain index is kept in the copy-candidate over multiple time-frames, trading off higher data reuse for higher area size.

#### 4.2. Re-definition of the original reuse factors

We will re-define the equations for reuse factors and area size, to take into account the effect of the newly defined included data reuse dependencies.

Let  $D_i$  be defined as the set of different indices accessed during  $timeframe_i$ , and  $A_i$  as the set of different indices actually present in the copy-candidate during  $timeframe_i$ . Then  $A_i$  is equal to the union of  $D_i$  and the set of indices kept in the copy-candidate by including certain data reuse dependencies in  $timeframe_i$ , i.e. data that will be reused in a later time-frame but not in  $timeframe_i$ . The data accessed during  $timeframe_i$  which was not present in the copy-candidate during the previous  $timeframe_{i-1}$ , is copied from a higher level. The number of writes to the copy-candidate at the start of  $timeframe_i$  is thus equal to  $\#(D_i \setminus A_{i-1})$ . A graphical representation of these notions is given in Fig. 5. Note that in the current methodology the set  $A_i$  would always be equal to the set  $D_i$ .

A set of reuse factor definitions per time-frame is given in Fig. 6. The *data reuse factor*  $F_{R_i}$  is equal to the fraction of the number of reads from the last level during this time-frame to the number of writes to the copy-candidate from a higher level at the start of this time-frame. The effect of reuse inside the time-frame, during which the same index can be accessed multiple times, is expressed by the *intra-copy reuse factor*  $NIaC_i$ . The *inter-copy reuse factor*  $NIeC_i$  illustrates the reuse of data already available in the copy-candidates during the previous time-frame and accessed during this time-frame. The overall data reuse factor  $F_{R_i}$  is equal to the product of the intra-copy and inter-copy

$$\begin{aligned}
F_{R_i} &= \frac{T_{timeframe}}{\#(D_i \setminus A_{i-1})} \\
&= NIaC_i * NIeC_i \\
NIaC_i &= \frac{T_{timeframe}}{\#(D_i)} \\
NIeC_i &= \frac{\#(D_i)}{\#(D_i \setminus A_{i-1})}
\end{aligned}$$

**Figure 6. Re-definition of data-reuse factors: per time-frame**

$$\begin{aligned}
F_R &= \frac{\sum_i T_{timeframe}}{\sum_i \#(D_i \setminus A_{i-1})} = \frac{C_{tot}}{C} \\
&= NIaC * NIeC \\
NIaC &= \frac{\sum_i T_{timeframe}}{\sum_i \#(D_i)} = \frac{C_{tot}}{\sum_i \#(D_i)} \\
NIeC &= \frac{\sum_i \#(D_i)}{\sum_i \#(D_i \setminus A_{i-1})} \\
A &= Max_i(A_i)
\end{aligned}$$

**Figure 7. Re-definition of data reuse factors: per copy-candidate**

reuse factor. The global reuse factors over all time-frames and the needed area size of a certain copy-candidate are given in Fig. 7.  $C_{tot}$  is the total number of reads from the last level in the copy-candidate chain,  $C$  is the total number of writes to this copy-candidate.

### 4.3. Application to example

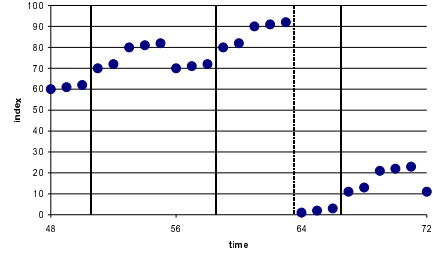
We have applied these new definitions of Fig. 6 for reuse factors and area size on the example given in Section 4.1, which is derived from a real-life image processing application. The results are given in Table 1. The first column shows the outcome for the current methodology. The second column shows a 33% increase of the data reuse factor  $F_{R_i}$  when including data reuse dependencies, but a 12.5% increase of the area size. Finally, the third column shows both an 33% increase of the data reuse factor, as well as a 33% decrease of the area size, when including data reuse dependencies and offsetting the time-frame.

### 4.4. Time-frame splitting at iteration boundaries

A specific problem arises when we use time-frames, which do not coincide with the loop boundaries. In Fig. 8 the data access graph is shown for the example at the end of one iteration of the `col` loop in Fig. 3. At time unit 64, a sudden jump in the access graph occurs, and instead of 6, we need to store 8 different indices in this time-frame. Since the final area size  $A = Max(A_i)$ , we loose the gain

	(A) Original	(B) DRD	(C) DRD, Offset=3
$timeframe_i$	8	8	8
$\#(D_i)$	8	8	6
$\#(D_i \setminus A_{i-1})$	4	3	3
$NIaC_i$	$\frac{8}{8} = 1$	$\frac{8}{3} = 2.67$	$\frac{8}{3} = 2.67$
$NIeC_i$	$\frac{4}{4} = 1$	$\frac{3}{3} = 1$	$\frac{3}{3} = 1$
$F_{R_i}$	$1 * 1 = 1$	$2.67 * 1 = 2.67$	$2.67 * 1 = 2.67$
$\#(A_i)$	8	9	6

**Table 1. Data reuse factors and area values for current methodology (a), including data reuse dependencies (b) and offsetting the time-frame (c)**



**Figure 8. Influence of iteration boundaries on area size illustrated in a data access graph**

in area size cost by offsetting the time-frame. The solution is to temporarily split the time-frame in two parts at the iteration boundaries of the higher loops, shown in Fig. 8 by a dotted vertical line. The price to pay is an increase in code size to implement this copy behaviour at the two iteration boundaries.

### 4.5. Multiple levels

In general a copy-candidate chain consists of copy-candidates of decreasing size, time-frame and data reuse factor. Certain constraints exist on which copy-candidates can be placed in a hierarchy. The time-frame boundaries of a higher level should coincide with the smaller time-frame boundaries of the lower hierarchy. This is to ensure that at the start of a smaller time-frame, all data accessed during this time-frame which should be loaded from the higher level is also available at that time. Thus a start of a higher level time-frame cannot be located in between time-frame boundaries of the lower level.

## 5. Steering the exploration of the search space

As explained in Section 2, the current methodology defines the search space as all possible subsets of a copy-candidate chain with *fixed* copy-candidates determined by the loop boundaries. By adding the freedom of resizing and offsetting time-frames and including data reuse dependencies for *each* copy-candidate, we deal now with a search

space explosion. Heuristics have to be found which steer the search for optimal solutions.

### 5.1. Extension of Belady’s MIN Algorithm to larger time-frames

Belady’s MIN Algorithm [2] [11] is a replacement algorithm for local memory which optimally exploits temporal locality, and assumes knowledge of the future. Given an address trace, and a fixed local memory size  $A_{fixed}$ , at each time instance the algorithm selects which references will be kept in local memory and which not in order to minimize the total number of memory fetches.

The algorithm can be explained by using the definitions of Fig. 6. Consider a time-frame of size 1. Then  $\#(D_i)$  is always equal to 1, and  $\#(A_i)$  depends on which DRD’s are kept in the local memory. When the value for  $\#(A_i)$  exceeds  $A_{fixed}$ , certain DRD’s present in  $timeframe_i$  are cut to meet the area size constraint. This means that the data corresponding to a cut DRD is not kept in the local memory during the time interval between two accesses. Belady’s algorithm now states that those DRD’s are cut which second access is furthest in the future.

For a time-frame size bigger than 1, this algorithm can still be applied. Since it should always be possible to fit the data accessed during one time-frame ( $=D_i$ ) the minimum area size possible for  $A_{fixed}$  is  $Max(\#(D_i))$ , where the minimum area size  $A_{fixed}$  for Belady’s algorithm was 1. The same selection procedure for cutting DRD’s to meet the area size constraints can be applied. Since Belady’s algorithm results in the minimum number of writes to the copy-candidate or the maximum data reuse factor  $F_R$  for a certain fixed area size, larger time-frames will always lead to equal or smaller data reuse factors for the same fixed area size, or if the same data reuse factor is required, the fixed area size should possibly increase.

### 5.2. $DRD, T_{timeframe}, offset$ : influence on relevant cost variables

We will now investigate how the three parameters - time-frame size  $T_{timeframe}$ , time-frame  $offset$ , included data reuse dependencies  $DRD$  - influence the relevant cost variables:

- $F_R = f(DRD)$

For a fixed set of data reuse dependencies included, the data reuse factor is fixed for all possible time-frame sizes and offsets, since changing the time-frame properties only provoke an exchange of intra-copy reuse for inter-copy reuse or vice versa. The product of the corresponding reuse factors  $NIaC$  and  $NIeC$  is a constant for constant  $DRD$ . More  $DRD$  results in a higher  $F_R$ . When all possible data reuse dependencies are included, which means that each data element is kept in the copy-candidate from the first to the last read, then a maximal value for the data reuse factor is reached where  $F_{Rmax} = \frac{C_{tot}}{signal\_size}$ . Then each element of the signal is written to the sublevel only once.

- $A = f(DRD, T_{timeframe}, offset)$

A larger time-frame results in a higher  $D_i$ , more included data reuse dependencies result in a higher  $A_i$ , both leading to a higher area size  $A$ . Changing the offset, changes both

$D_i$  and  $A_i$ , depending on the access pattern. For a fixed data reuse factor  $F_R$ , the minimum needed area size is found with Belady’s algorithm ( $T_{timeframe} = 1$ ).

- **copy behaviour cost** =  $f(T_{timeframe}, offset)$

The time-frame size and offset define how and when copies are made to copy-candidates during the execution of the algorithm. The implementation of this copy behaviour thus affects the *code size* and *address calculation cost* of the final program. Larger time-frames are better for *burst copies*.

### 5.3. Heuristics to speed up the search

Enumerating and computing the cost of all possible combinations of time-frame sizes, offsets, and DRD for each of the copy-candidates, is not feasible for real-life applications. Therefore a reduced search space will be explored, taking into account the constraints given by the implementation platform.

We have seen in Section 5.1 that, for a fixed area size, the solution provided by Belady’s algorithm with  $T_{timeframe} = 1$  gives an equal or better power-area trade-off than a solution with the time-frame size larger than 1. However, larger time-frames and offsets are generally better for the copy behaviour cost.

We propose to search for a good solution in two steps: Firstly, find the copy-candidate with minimum area size  $A_{Max}$  for maximum data reuse  $F_{RMax}$  with Belady’s algorithm. Then also with Belady’s algorithm, for each possible value of  $A < A_{Max}$  one can find a (smaller)  $F_R$ . Each subset of the corresponding copy-candidates for different signals can be combined to a possible hierarchy. Time-frame constraints for multiple levels as discussed in Section 4.5 do not exist here, since the time-frame size is equal to 1 for all levels. From these copy-candidates a set of good combinations in terms of area and power cost is selected, possibly taking into account a fixed hierarchy constraint when the implementation platform is already known.

In a second step we decrease the cost of the copy behaviour by enlarging and offsetting the time-frames. Time-frame sizes and offsets, which can be written as a regular function of the iteration indices of the nested loops, are easier to implement and reduce the code size and address calculation cost. Larger time-frames are better for burst transfers from DRAM. A trade-off with increasing area size and power cost has to be made. Time-frame boundary constraints reduce the number of possibilities for subsequent sublevels in the hierarchy.

The weight assigned to each of the cost functions for area, power, code size and execution time, steer the decisions made during these two steps.

## 6. Experimental results

We have applied these extensions for the data reuse methodology to some real-life applications as the “*Binary Tree Predictive Coder*” and a “*Cavity Detection*” algorithm, where holes in the access pattern are present. We are using a proprietary power model for on-chip memory, therefore only relative values for the power cost are given in Table 2.

	Previous meth.			Extended meth.		
	Tot. WD	# reg.	Power	Tot. WD	# reg.	Power
BTPC	15	0	1.00	71	0	0.77
Cavity	$(3*N)+(8)$	1	0.74	$(2*N+3)$	2	0.51
Det.	$(3*N)+(9)$	0	1.00	$(2*N+3)+(9)$	0	0.75

**Table 2. Comparing results for power and area cost for the previous and extended methodology**

In the Binary Tree Predictive Coder (BTPC) about 15 values of the input image are accessed in the loop body, which are partly reused in the next iteration. However, including data reuse over multiple iterations, reduces the power cost by 23%. The word depth of the extra memory level increased but is still small compared to the image size (typically  $1024*1024$  bytes).

In the Cavity Detection algorithm two dominant signals with a size equal to the size of the input image ( $N*M$  bytes) exhibit a similar access pattern as given in the example in Section 4.1. When register data reuse is possible, The extended methodology comes up with a 1-level hierarchy, the previous methodology with a 2-level hierarchy; without register data reuse, the solution is a 2-level memory hierarchy. For both cases the total word depth of the extra memory levels could be reduced by about 30%. The power cost is reduced by respectively 31.1% and 24.8%. This clearly indicates the potential of our new approach.

## 7. Conclusion

In this paper we have extended the current available methodology for data reuse exploration in a customized embedded memory context [14], to find more cost-effective memory hierarchy solutions for applications with holes in the signal access patterns. By relaxing some of the assumptions of the current methodology, we can achieve much better results in terms of area and power. However, next to the power and area cost, also the cost of code size and address calculation cost has to be taken into account for more complex copy behaviour. We have identified the parameters which define the huge search space and indicated their relationship with the relevant cost variables. A strategy to steer the search for a good solution is based on these relationships and the weight assigned to each of the different costs. Constraints given by knowledge of the final implementation platform should also be included in the future.

## Acknowledgements

This project is partly sponsored by the Belgian Pole of Attraction IUAP-4/24, FWO Project G.0036.99 and IMEC. K.U.Leuven-ESAT and IMEC are members of the DSP Valley network.

## References

- [1] A.Faruque and D.Fong. Performance analysis through memory of a proposed parallel architecture for the efficient use of

- memory in image processing applications. In *SPIE'91, Visual communications and image processing*, pages 865–877, Boston MA, Oct. 1991.
- [2] L. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems J.*, 5(6):78–101, 1966.
- [3] B.Jacob, P.Chen, S.Silverman, and T.Mudge. An analytical model for designing memory hierarchies. *IEEE Trans. on Computers*, C-45(10):1180–1193, 1996.
- [4] D.Kolson, A.Nicolau, and N.Dutt. Elimination of redundant memory traffic in high-level synthesis. *IEEE Trans. on Comp-aided Design*, 15(11):1354–1363, Nov. 1996.
- [5] D.Kulkarni and M.Stumm. Linear loop transformations in optimizing compilers for parallel machines. Technical report, Comp. Systems Res. Inst., Univ. of Toronto, Canada, Oct. 1994.
- [6] F.Balasa, F.Catthoor, and H. De Man. Dataflow-driven allocation for multi-dimensional processing systems. In *IEEE Intl. Conf. Comp. Aided Design*, San Jose CA, Nov. 1994.
- [7] F.Catthoor, S.Wuytack, E. Greef, F.Balasa, L.Nachtergaele, and A.Vandecappelle. *Custom Memory Management Methodology – Exploration of Memory Organisation for Embedded Multimedia System Design*. Number ISBN 0-7923-8288-9. Kluwer Acad. Publ., Boston, 1998.
- [8] L.Liu. Issues in multi-level cache design. In *IEEE Intl. Conf. on Computer Design*, pages 46–52, Cambridge MA, Oct. 1994.
- [9] L.Nachtergaele, F.Catthoor, B.Kapoor, D.Moolenaar, and S.Janssen. Low power storage exploration for h.263 video decoder. In *IEEE workshop on VLSI signal processing*, Monterey CA, Oct. 1996.
- [10] L.Ramachandran, D.Gaiski, and V.Chaiyakul. An algorithm for array variable clustering. In *European Design and Test Conf.*, pages 262–266, Paris, France, March 1994.
- [11] O.Temam. An algorithm for optimally exploiting spatial and temporal locality in upper memory levels. *IEEE Trans. on Computers*, 48(2):150–158, Feb. 1999.
- [12] P.Lippens, J. Van Meerbergen, W.Verhaegh, and A. Van der Werf. Allocation of multiport memories for hierarchical data streams. In *IEEE Intl. Conf. Comp.-Aided Design*, pages 728–735, Santa Clara, Nov. 1993.
- [13] P.Panda, N.Dutt, and A.Nicolau. Memory organization for improved data cache performance in embedded processors. In *1996 Intl. Symp. on System Synthesis*, pages 90–95, La Jolla CA, Nov. 1996.
- [14] S.Wuytack, J. Diguët, F.Catthoor, and H. De Man. Formalized methodology for data reuse exploration for low-power hierarchical memory mappings. *IEEE Trans. on VLSI Systems*, 6(4):529–537, Dec. 1998.
- [15] S.Wuytack, F.Catthoor, F.Franssen, L.Nachtergaele, and H. Man. Global communication and memory optimizing transformations for low power systems. In *IEEE Intl. Workshop on Low Power Design*, pages 203–208, Napa CA, April 1994.
- [16] S.Wuytack, F.Catthoor, L.Nachtergaele, and H. Man. Power exploration for data dominated video applications. In *IEEE Intl. Symp. on Low Power Design*, pages 359–364, Monterey, Aug. 1996.
- [17] T. Van Achteren, M.Adé, R.Lauwereins, M.Proesmans, L. Van Gool, J.Bormans, and F.Catthoor. Transformations of a 3D image reconstruction algorithm for data transfer and storage optimisation. In *10th IEEE Intl. Workshop on Rapid System Prototyping*, Clearwater FL, USA, June 1999.