Data Reuse Exploration Techniques for Loop-dominated Applications

Tanja Van Achteren, Geert Deconinck,
K.U.Leuven ESAT/ACCA
Kasteelpark Arenberg 10
B-3001 Leuven-Heverlee, Belgium
{vanachte,gdec}@esat.kuleuven.ac.be

Francky Catthoor, Rudy Lauwereins
IMEC, part-time professors at K.U.Leuven
Kapeldreef 75
B-3001 Leuven-Heverlee, Belgium
{catthoor,lauwerei}@imec.be

Abstract

Efficient exploitation of temporal locality in the memory accesses on array signals can have a very large impact on the power consumption in embedded data dominated applications. The effective use of an optimized custom memory hierarchy or a customized software controlled mapping on a predefined hierarchy, is crucial for this. Only recently effective systematic techniques to deal with this specific design step have begun to appear. They were still limited in their exploration scope. In this paper we introduce an extended formalized methodology based on an analytical model of the data reuse of a signal. The cost parameters derived from this model define the search space to explore and allow us to exploit the maximum data reuse possible. The result is an automated design technique to find power efficient memory hierarchies and generate the corresponding optimized code.

1. Introduction

Memory hierarchy design has been introduced long ago to improve the data access (bandwidth) to match the increasing performance of the CPU [8][24][23]. The well-known idea of using memory hierarchy to minimise the power consumption, is based on the fact that memory power consumption depends primarily on the access frequency and the size of the memory [30]. Power savings can be obtained by accessing heavily used data from smaller memories instead of from large background memories. This can be controlled by hardware as in a cache controller but then only opportunities for reuse are exploited that are based on ”local” access locality. Instead, also a compile-time analysis and code optimization stage can be introduced to arrive at a ”global” data reuse exploitation across the entire program [6]. Such a power-oriented optimization requires architectural transformations that consist of adding layers of memories of decreasing size to which frequently used data will be copied, 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 A. The dots represent the relative time order of the accesses to the array element. 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 to a smaller memory, such that from the second usage on, a data element can be read from the smaller memory instead

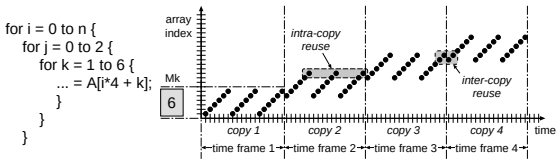


Figure 1. Exploiting data reuse local in time to save power.

of the larger memory, reducing the power consumption. In a hardware controlled cache all data would be copied the first time into the cache and possibly overwrites existing data, based on a replacement policy which only uses knowledge about *previous* accesses. In contrast, in our approach it is checked at compile-time that the new data that will be copied has sufficient *future* reuse potential to improve the overall power cost, compared to leaving existing data or copying other data. This analysis is performed not for entire arrays only but also for uniformly accessed sub-arrays of data. So, efficient memory hierarchy mapping 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 (in a custom memory organisation context) can also have a negative effect on the memory size and interconnect cost, and as a consequence also on the power. It is the task of the data reuse decision step to find a set of good solutions for these trade-offs [5]. This is required for a custom hierarchy, but also for efficiently using a predefined memory hierarchy with software cache control, where one has the design freedom to copy data at several moments with different copy sizes [6]. In the latter case, several of the virtual layers in the global copy-candidate chain of Fig. 2 (see further) can be collapsed to match the available memory layers.

2. Related Work

The main related work to this problem lies in the parallel compiler area, especially related to the cache hierarchy itself. Many papers have focused on introducing more access locality by means of loop transformations (see e.g. [16][2][19][15] [7]). That literature is however complementary to the work presented here: it forms only an enabling step. The techniques proposed there on cache exploitation are not resulting yet in any formalisable method to guide the memory organisation issues that can be ex-

ploited at compile-time. In most work on parallel MIMD processors, the emphasis in terms of storage hierarchy has been on hardware mechanisms based on cache coherence protocols [18]. Some approaches address the hierarchical data organisation in processors for programs with loop nests, e.g. a quantitative approach based on approximate life-time window calculations to determine register allocation and cache usage [7]. The main focus is on CPU performance improvement though and not directly on memory related energy and memory size cost or trade-off analysis. Also in an embedded system synthesis context, applying transformations to improve the cache usage has been addressed [13][22][10]. None of these approaches determine the best memory hierarchy organisation for a given application and they do not directly address the power cost. An analysis of memory hierarchy choices based on statistical information to reach a given throughput expectation has been discussed [9]. More recently, dynamic management of a scratch-pad memory for temporal and spatial reuse has been addressed [11]. In the hardware realization context, much less work has been performed, mainly oriented to memory allocation [17][25][1][26][4]. The impact of the data reuse exploration step turns out to be very large in the entire methodology for power reduction. This has been demonstrated for a H.263 video decoder [21], a motion estimation application [6] and a 3D image reconstruction algorithm [28].
At IMEC, a formalized methodology for data reuse exploration [6] has been developed as part of the ATOMIUM script [5] for data transfer and storage exploration (DTSE). That methodology was systematic and manually applicable but not directly implementable in a fully automated tool. Moreover, it has restrictions on the actual data reuse behaviour that can be handled. In [29] some vital cost parameters were introduced to describe a more complete search space than [6]. The relationship between these parameters and the cost function for power and memory size were explored, and heuristics were proposed to steer the search for a good solution. However, a simulation-based approach was used, which is not yet applicable for applications with large loop nests. In this paper we will formalize this extended search space by introducing an analytical model for the cost parameters as a function of the index expressions and loop bounds. This avoids long simulation times and more importantly, it will now be possible to identify exactly which array elements have to be copied to a sub-level for optimal data reuse. This leads to a fully automatable design technique for all loop-dominated applications to find optimal memory hierarchies and generate the corresponding optimized code. This is novel and our main contribution in this paper. The rest of this paper is organized as follows. In Section 3 we give a short overview of the DTSE steps and a summary of the basic cost functions to evaluate a possible hierarchy. Section 4 gives some results for a motion estimation example when simulating the search space for the different cost parameters. Section 5 then presents a model to describe the data reuse of array elements in an analytical way. The data reuse related cost parameters are derived from this model in Section 6, and results for motion estimation and a more complex test-vehicle are given. Section 7 concludes the paper.

3. Basic concepts

To clarify the function of the data reuse step in the complete methodology, we present a quick overview of the DTSE steps [5] we assume are applied before and after.

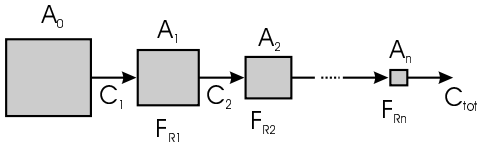


Figure 2. Copy-candidate chain with memory sizes A_j, data reuse factors F_{Rj} and number of writes C_j.

1. For the specific step addressed in this paper, we assume the code has been pre-processed to single assignment code, where every array value can only be written once but read several times. A *pre-processing* step in our overall approach allows to achieve this systematically [5]. This makes the further analysis much easier.
2. We assume the code has been transformed to optimize the data locality during a previous *data flow and loop transformation* step. A certain freedom in loop nest ordering is still available after this step.
3. During the *data reuse* step we determine the optimal memory hierarchy cost for each of the signals and each loop nest ordering separately. A global decision optimizing the total memory hierarchy including all signals, will then be taken in a subsequent *global hierarchy layer assignment* step.
4. In the *storage cycle budget distribution (SCBD)* step, the bandwidth/latency requirements and the balancing of the available cycle budget over the different memory accesses in the algorithmic specification are determined.
5. The goal of the *memory allocation and assignment (MAA)* step is, given the cycle budget, to allocate memory units and ports from a memory library and to assign the data to the best suited memory units.
6. The *inplace mapping* step exploits the limited life-time of signals to further decrease the storage size requirements.

Definition of data reuse factor

The usefulness of a memory hierarchy, especially if it is customized to the application as proposed in [6][29], 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 higher hierarchical level.

In Fig. 2 a generic copy-candidate chain is shown where for each level j, we can determine a memory 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

$$F_{Rj} = \frac{C_{tot}}{C_j}. \quad (1)$$

C_{tot} is the total number of reads from the signal in the lowest 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 sub-level in the hierarchy. If F_{Rj} equals 1 or is too low (dependent on the power models of the memories in the different layers), this sub-level is useless and would even lead to an increase of memory size and power, because the

number of read operations from level (j-1) would remain unchanged while the data also has to be stored and read from level j. So these cases are pruned from the search space.

Cost functions

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 bit-width 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 sub-levels is defined as:

$$F_c = \alpha \sum_{j=0}^{n} P_j(N_{bits}, N_{words}, F_{access}) + \beta \sum_{j=1}^{n} A_j(N_{bits}, N_{words}) \qquad (2)$$

Here α and β are weighing factors for Memory size/Power trade-offs. When we define P_j^r and P_j^w as the power consumption for respectively read and write operations to level j at frame rate F_{frame}, the following expression for the total power cost for a complete copy candidate chain can be deduced:

$$\begin{aligned} \sum_{j=0}^{n} P_j() &= C_1(P_0^r + P_1^w) + C_2(P_1^r + P_2^w) + \cdots + C_{tot}(P_n^r) \\ &= C_{tot}[\sum_{j=1}^{n-1}(\frac{1}{F_{Rj}}(P_{j-1}^r + P_j^w)) + P_n^r] \end{aligned} \qquad (3)$$

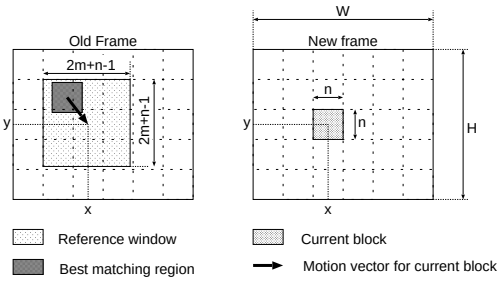
From equation (3) we learn that smaller memory sizes A_j, which reduce $P_j^{r(w)}$, and higher data reuse factors F_{Rj} reduce the total power cost of a copy candidate chain.

4. Simulation of power and memory size cost

In [29] a number of parameters were introduced which define the search space for a certain copy-candidate. First of all the *data reuse dependencies* (DRD) determine during which subsequent accesses a certain data element is kept in a copy-candidate of the hierarchy. This fixes the number of transfers between the different levels, thus also the data reuse factors F_{Rj}. For a fixed memory size A_j the highest possible data reuse factor is reached by applying Belady's optimal replacement strategy [3]. When data is loaded earlier than needed, the memory size of the copy-candidate can increase. So, the *timing of the transfers* is a second parameter which influences the memory size and power cost of a copy-candidate. However, when these timings are chosen according to the iteration time-frames, this generally leads to more regular programs. A clear trade-off is present here. The timing of the copies will be fixed in a subsequent *storage cycle budget distribution (SCBD)* step [5]. For evaluating the cost functions (2) and (3), the memory size cost provided by Belady can be used as a lower bound. More realistic upper and lower bounds on sizes for single assignment code can be produced by a system-level *memory size estimation* tool based on [12].

Motion estimation: Simulation results

With our simulation prototype tool, we explore the search space for accesses to one signal in nested loops [29]. It




```
for (i1=0;i1<H/n; i1++)             /*Vert.  CB counter*/
  for (i2=0; i2<W/n; i2++) {        /*Horz.  CB counter*/
    Δopt[i1][i2] = +∞;
    for (i3=-m; i3<m; i3++)         /*Vert.  in RW*/
      for (i4=-m; i4<m; i4++) {     /*Horz.  in RW*/
        Δ = 0;
        for (i5=0; i5<n; i5++)      /*Horz.  in CB*/
          for (i6=0; i6<n; i6++) { /*Vert.  in CB*/
            Δ += ABS(New[i1*n+i5][i2*n+i6]
                   - Old[i1*n+i3+i5][i2*n+i4+i6]);
          }
        Δopt[i1][i2] = MIN(Δ, Δopt[i1][i2]);
        }
    }
```

Figure 3. Motion estimation kernel.

generates a data reuse factor curve and power-memory size trade-off curve (which is called a Pareto curve in mathematics) for different levels when optimal replacement is assumed. A good solution should be chosen on this Pareto curve because all points above it are suboptimal and below only infeasible points exist. We show here the results for the *Old* frame in a "full-search full-pixel" motion estimation kernel [14] used in moving image compression algorithms. It allows to estimate the motion vector of small blocks of successive image frames. The algorithm is shown in Fig. 3. The simulations were done for the following parameter values: H=144, W=176, n=m=8.

In Fig. 4a the data reuse factor is shown as a function of the copy-candidate size. The maximum (average) reuse factor of 209,5 is obtained at a size of 2745 (A_1) which is about 16 lines of the *Old* frame. This is the maximum reuse for the iterations in the outer loop. We also see several discontinuities in the curve for smaller copy-candidate sizes ($A_2 - A_4$). These are the sizes where maximum reuse is obtained for a subset of inner loops in the total loop nest. Using equation (3) a Pareto curve for power and memory size is obtained by considering all possible hierarchies combining points on the data reuse factor curve (See Fig. 4b). The choice of the final hierarchy depends on the weighing factors assigned to power and memory size cost in equation (2). Since we are using proprietary memory power models, only relative values are given, which are normalised to the cost when all accesses for this signal are external memory accesses. Nonetheless, these results confirm that the power consumption can be drastically reduced by introducing an application-optimized memory hierarchy. They also show that a trade-off exists between on-chip memory size and power.

5. Analytical model

Simulation is very time-consuming when large applications and signal sizes are considered. Also, simulation does not give clear information about which data has to be copied to a smaller copy-candidate to reach the computed power consumption savings. Therefore we have developed an ana-

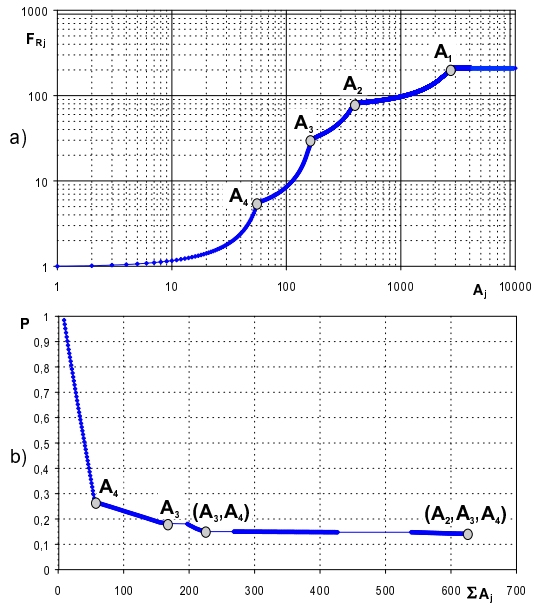


Figure 4. Motion estimation: (a) Data reuse factor for array `Old[][]` as a function of the copy-candidate size. (b) Power - memory size Pareto curve for array `Old[][]`.

```
for (i=iL; i<=iU; i++)
...
  for (j=jL; j<=jU; j++)
    for (k=kL; k<=kU; k++)
      {
       ...  = A[a*i + ...  + b*j + c*k + d];
      }
```

Figure 5. Generic nested loop with an access to signal A.

lytical model of the signal accesses, which is the main contribution of this paper.

5.1. Application domain

A large application domain is covered when considering accesses with *affine index expressions* of the loop iterators. A full mathematical model will be developed for this. For other kind of expressions we will rely on simulation. We present in this paper the results for an access in the most frequently occurring case of an inner double nested loop with fixed loop bounds. The extension to handle more inner loops of a higher-dimensional loop nest and non-rectangular access patterns are the subject of current research and will be discussed in a future paper. The generic loop nest considered throughout the following sections is presented in Fig 5. Analogous formulas can be derived for decremental loops. The theory described in this paper is easily extended to loops with incremental step sizes larger than 1, by (temporarily) transforming the loop nest to a loop nest with a step size equal to 1 and deriving the new formulas as a function of the original step sizes.

5.2. Data reuse in the iteration space

We will analyze the data reuse in the two inner loops (j,k) of Fig. 5 for one iteration of the higher loop levels (i,...) at a

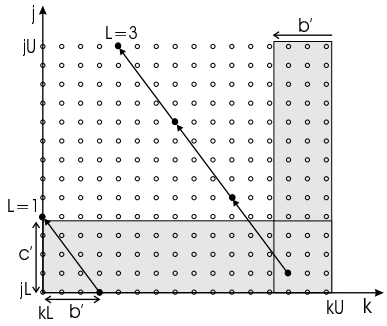


Figure 6. Data Reuse in the iteration space.

time. The index expression for one element of signal A can then be written as follows, since the index expression part related to the higher loop level indices can be regarded as a constant:

$$\begin{aligned} y &= b*j + c*k + constant \\ &= b*j_{tMIN} + c*k_{tMIN} + constant \\ &= b*j_{tMAX} + c*k_{tMAX} + constant \end{aligned}$$

$$\Rightarrow b*(j_{tMAX} - j_{tMIN}) - c*(k_{tMIN} - k_{tMAX}) = 0 \quad (4)$$

where (j_{tMIN}, k_{tMIN}) and (j_{tMAX}, k_{tMAX}) are the values of the iterators respectively the first time and last time the element A[y] is accessed. Assuming now integer values for $b \geq 0$ and $c > 0$ [1](analogous formulas for $b < 0$ and/or $c \leq 0$ can be straightforwardly derived in the same way), the solution to equation (4) is:

$$j_{tMAX} - j_{tMIN} = L * c' \quad (5)$$

$$k_{tMIN} - k_{tMAX} = L * b' \quad (6)$$

$$c' = \frac{c}{\gcd(b,c)}, \; b' = \frac{b}{\gcd(b,c)} \quad (7)$$

$$L = min[(k_{tMIN} - kL)div(b'), (jU - j_{tMIN})div(c')] \quad (8)$$

A graphical interpretation is given in the (j,k) iteration space in Fig. 6. Uniformly generated data reuse dependency vectors [7] $(c', -b')$ connect the iterations which reuse the same data inside the two inner loops. The gray zone corresponds to iterations (j_{tMIN}, k_{tMIN}) where a data element is accessed for the first time. L is the number of separate data reuse dependencies on the line connecting different iterations in the $(c', -b')$ direction. In the special case where both b and c are equal to 0, the index expression is independent of the values of j and k and the same element is accessed in every iteration of the (j,k) iteration space.

5.3. Multiple indices

The formulas given in the previous sections assume a signal of one dimension. Consider a n-dimensional signal $A[y_1][y_2]...[y_n]$ where $y_i = b_i * j + c_i * k + constant_i$. Then, analogous to eq. (4) for one dimension, we have the following system of linear equations:

$$B = \begin{bmatrix} b_1 & -c_1 \\ b_2 & -c_2 \\ \vdots & \vdots \\ b_n & -c_n \end{bmatrix} \quad B * \begin{bmatrix} j_{tMAX} - j_{tMIN} \\ k_{tMIN} - k_{tMAX} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

[1]For $b = 0, c > 0 : gcd(b,c) = c \Rightarrow b' = b/c = 0, \; c' = c/c = 1.$

The condition for a non-trivial (non-zero) solution to this system and consequently the condition for data reuse is

$$rank(B) \leq 1 \tag{9}$$

When rank(B)>1, each element is accessed only once and no gain is possible from data reuse. When rank(B)=1, all non-zero rows of B result in the same (b',c') pair, and the original formulas (5)-(8) for a 1-dimensional signal can be applied. Rank(B)=0 is again a special case where the index expression is independent of the values of j and k and the same element is accessed in every iteration of the (j,k) iteration space.

6. Derivation of data reuse related cost parameters

In this section we will derive formulas for the data reuse factor F_{Rj} and a lower bound for the copy-candidate size A_j. This will allow us to generate points on the data reuse factor curve and the power-memory size Pareto curve (see Section 4 and Fig. 4) in an analytical way. Given

$$jRANGE = jU - jL + 1 \tag{10}$$
$$kRANGE = kU - kL + 1, \tag{11}$$

then reuse is only possible when $(jRANGE > c')$ and $(kRANGE > b')$ (see Fig. 6), the formulas are derived for the cases where these conditions holds. The rest of the section discusses which copies have to be made to obtain these savings, and a generic code implementation template will be given.

6.1. Maximum data reuse

First we derive the formulas for maximum reuse inside the (j,k) iteration space. This corresponds to the first discontinuity on the data reuse factor curve (e.g. A_4 on Fig. 4a). Each data element is written once to the copy-candidate before the first read access and will be reused there for all following read accesses.

Data reuse factor

The domain of iterations (j_{tMIN}, k_{tMIN}) where an element is accessed for the first time (see the gray zone in Fig. 6) is subject to the following condition:

$$(k_{tMIN} \in [kU - b' + 1, kU]) \vee (j_{tMIN} \in [jL, jL + c' - 1])$$

The maximum data reuse factor[2] is then equal to the total number of accesses divided by the size of the domain of first accesses:

$$F_{RMax} = \frac{C_{tot}}{C_{tot} - C_R} \tag{12}$$

$$with \quad C_{tot} = jRANGE * kRANGE \tag{13}$$
$$C_R = (jRANGE - c') * (kRANGE - b') \tag{14}$$

Copy-candidate size

We now have to determine which elements are in the copy-candidate at each time instance. These are all elements which have a read access before and after this time instance. We define a time instance t(j,k) as the time at which the element A[..+b*j+c*k+d] is accessed. An element $A[.. + b * j_p + c * k_p + d] = A[.. + b * j_n + c * k_n + d]$, with two consecutive accesses at time instances $t(j_p, k_p)$ and $t(j_n, k_n)$, is present in the copy-candidate at time instance t(j,k) if

$$t(j_p, k_p) < t(j,k) \leq t(j_n, k_n)$$

Working out this inequality results in four regions: an element $A[.. + b * j_c + c * k_c + d]$ is present in the copy-candidate at time instance t(j,k) if

$$\begin{array}{lll}
I. & j_c \in [max(jL, j - c' + 1), min(jU - c', j - 1)] & \\
 & k_c \in [kL + b', kU] & \\
II. & j_c = j & only\ if\ (j \geq jL + c') \\
 & k_c \in [k + 1, kU - b'] & \\
III. & j_c = j & only\ if\ (j \leq jU - c') \\
 & k_c \in [kL + b', k - 1] & \\
IV. & j_c = j & \\
 & k_c = k &
\end{array}$$

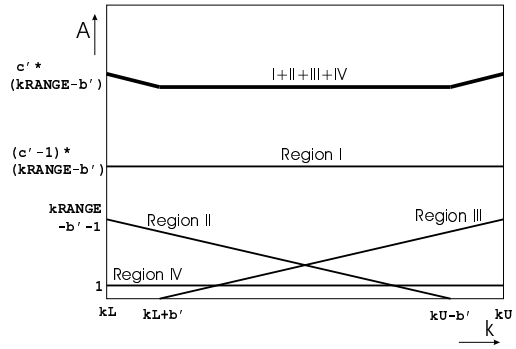


Figure 7. Copy-candidate size variation for steady state when ($kRANGE > 2 * b'$) and ($jRANGE > 2 * c'$).

Region I represents the elements which are accessed in at most $(c' - 1)$ *previous* j-iterations and kept in the copy-candidate for future reuse. Region II consists of the elements used in the *following* k-iterations in the *current* j-iteration and already available in the copy-candidate from iteration $(j - c')$. Region III contains the elements used in the *previous* k-iterations in the *current* j-iteration and kept in the copy-candidate for future reuse. Region IV represents the element accessed in the *current* (j,k)-iteration.

The maximum of the needed copy-candidate size will be reached for values of j in steady state, i.e. when $(j \geq jL + c')$ and $j \leq jU - c')$. The contribution of each of the four regions I-IV to the copy-candidate size as a function of the value of k is shown in Fig. 7. To not overload the paper, we have omitted the formulas for the cases where $(b' < kRANGE < 2 * b')$ and/or $(c' < jRANGE < 2 * c')$, where boundary effects come into play. The maximum copy-candidate size[3] needed for maximum data reuse thus becomes

$$A_{Max} = c' * (kRANGE - b') \tag{15}$$

[2] For b=c=0, F_{RMax}=C_{tot}=jRANGE*kRANGE.

[3] For b=c=0, A_{Max}=1.

```
int A_sub[cp][kRANGE-bp];

for (i=iL; i<=iU; i++)
  {
  ...
  for (j=jL; j<=jL+cp-1; j++)      /* Initialization */
    for (k=kL; k<=kU-bp; k++)
      A_sub[j%cp][k%(kRANGE-bp)] = A[a*i+...+b*j+c*k+d];

  for (j=jL; j<=jU; j++)
    for (k=kL; k<=kU; k++)
      {
      if(k>kU-bp)                  /* Update */
        A_sub[j%cp][((((j-jL)/cp)*bp+k)%(kRANGE-bp)]
             = A[a*i+...+b*j+c*k+d];

      ...  = A_sub[j%cp][((((j-jL)/cp)*bp+k)%(kRANGE-bp)];
      }
  }
```

Figure 8. Generic code for the introduction of a copy-candidate with maximum reuse and optimal timings.

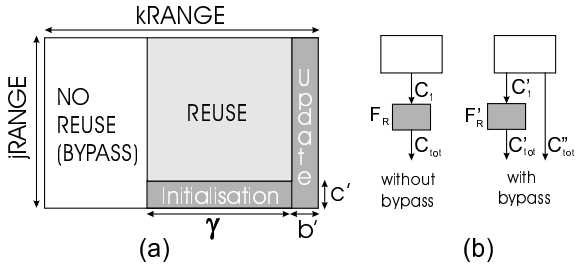


Figure 9. (a) Division of iteration space for partial reuse, (b) Hierarchy with and without bypass for not-reused data.

Code implementation template

Analyzing the formulas derived in section 6.1, we come to the following conclusions. In the copy-candidate, we keep the elements accessed in the previous $(c' - 1)$ j-iterations except for $k \in [kL, kL + b' - 1]$ (region I). In the current j-iteration we access the first b' elements for the last time, so these can be overwritten by the last b' elements which are accessed for the first time (regions II-III-IV). This is also visible in the variation of the copy-candidate size in Fig. 7. From the data reuse dependency $(c', -b')$ we know that the elements accessed in the j and $(j - c')$ iteration are partly the same, translated by $-b'$ in the k direction. This results in the generic code illustrated in Fig. 8 where a copy `A_sub` of size $c' * (kRANGE - b')$ is introduced and a convenient replacement policy is implemented. The addressing looks rather complicated, but can be linearized and greatly simplified by the ADOPT tools [20] for address optimization, a stage following the DTSE stage. So we do not worry yet about this complexity at this time. Also, the conditional update will be moved out of the critical kernel by the SCBD step to allow for software pipelining. In fact, to give the SCBD step the full freedom to schedule the updates at earlier time instances, we will generate a single assignment version of this code by enlarging the dimensions of the copy to `A_sub[cp][((jU-jL)/cp)*bp+kU+1]` and eliminate the `%(kRANGE-bp)` part in the index expressions. The SCBD can then trade off a larger final copy-candidate size with better timings for performance. The final copy-candidate size and implementation is determined by the *In-place mapping step* [5] afterwards.

6.2. Partial data reuse for Pareto trade-offs

Until now we have only defined the point for maximum reuse on the data reuse factor curve (fig. 10). To enable a good power-memory size trade-off, we need to define more points on the Pareto curve which also include partial reuse in the (j,k) iteration space.

Copy-candidate without bypass

An important requirement is that the solution must be relatively easy to implement in the data memory space to not make the code too complex for the subsequent DTSE steps. We propose to divide the iteration space in two parts (see Fig. 9(a)). Other strategies trading off power efficiency and code complexity are possible, but are omitted here due to lack of space. Given an integer γ, $(b' \leq \gamma < kRANGE - b')$, then for the iterations where $(k > (kU - \gamma - b'))$, we assume complete reuse of the accessed elements, for the other there is no reuse. The data reuse factor and copy candidate size then become

$$F_R(\gamma) = \frac{C_{tot}}{C_{tot} - C_R(\gamma)} \tag{16}$$

$$with \qquad C_R(\gamma) = \gamma * (jRANGE - c') \tag{17}$$

$$A(\gamma) = c' * \gamma + 1 \tag{18}$$

Copy-candidate with bypass

Since part of the data is not reused (Fig. 9(a)), it is not useful at all to write this data to the intermediate copy-candidate, but instead it can immediately be bypassed to the next level as illustrated in Fig. 9(b). This information was not available when using simulation, since the actual data elements present in the copy-candidate were not known. Consequently no extra element is needed for the copies of the not-reused elements. The formulas for the data reuse factor and copy-candidate size then become:

$$F'_R(\gamma) = \frac{C'_{tot}}{C'_{tot} - C_R(\gamma)} \tag{19}$$

$$with \qquad C'_{tot} = (\gamma + b') * jRANGE \tag{20}$$

$$C''_{tot} = C_{tot} - C'_{tot} \tag{21}$$

$$A'(\gamma) = c' * \gamma \tag{22}$$

6.3. Test-vehicle: MPEG4 motion estimation

To analyze the test vehicles described in this paper, we developed a basic prototype tool which computes, based on the loop and index expression parameters as input, the data reuse factor and power/memory size Pareto curve points with and without bypass (with graphical output using gnuplot). Also, a template is generated for the original and transformed code for different solutions in the search space. The application code is then easily (manually) transformed for a chosen solution based on these templates. The tool will be extended in the future for automatic input parameter extraction and transformation of the source code.

We now apply the formulas for maximum and partial reuse to the motion estimation kernel in Fig. 3 for the access to the *Old* array. The two inner

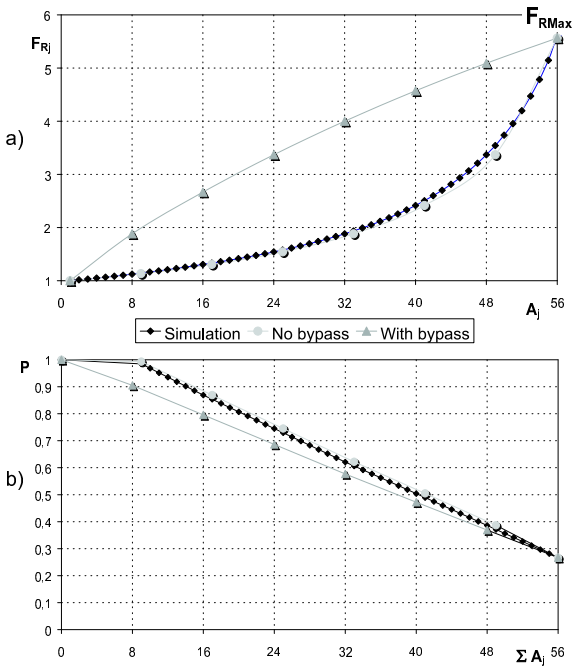


Figure 10. Motion estimation, analytically computed points for the inner (i4-i5-i6) loop nest on: (a) The simulated data reuse factor curve. (b) The simulated power-memory size Pareto curve.

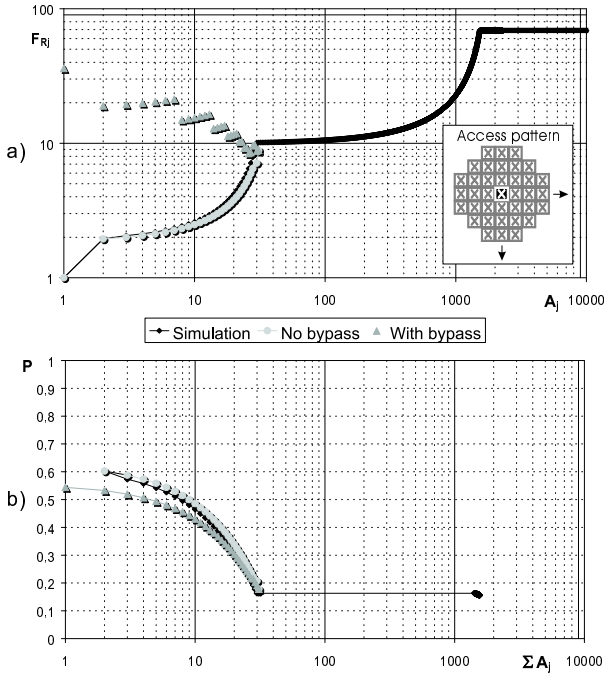


Figure 11. SUSAN principle: (a) Combined data reuse factor curve for image pixel accesses. (b) Combined power - memory size Pareto curve for image pixel accesses.

loops (5)-(6) do not carry reuse since rank(B)=2 (see condition in equation (9)) for the index expression `Old[..+1*i5+0*i6][..+0*i5+1*i6]`. So, each iteration in the (i5,i6) iteration space corresponds to a different element of the *Old* array. However, the (4)-(6) loop with index expression `Old[..+0*i4+1*i5+0*i6][..+1*i4+0*i5+1*i6]` carries reuse with rank(B)=1, b'=1, c'=1. This reuse is repeated for the elements accessed in each iteration of loop (5). To take this into account, the formula for the copy candidate size is adapted with an additional factor equal to the range of loop (5) = n. Filling in equations (12)-(18), this leads to

$$
\begin{aligned}
F_{RMax} &= \frac{(2*m)*(n)}{(2*m)*(n)-(2*m-1)*(n-1)} \\
A_{Max} &= (\mathbf{n})*(1)*(n-1) \\
F_R(\gamma) &= \frac{(2*m)*(n)}{(2*m)*n-\gamma*(2*m-1)} \\
A(\gamma) &= (\mathbf{n})*(1)*\gamma+1
\end{aligned}
$$

Simulation shows that the obtained data reuse is very close to the theoretically optimal data reuse possible with these copy-candidate sizes[4], as illustrated by the light gray bullets that lie nearly on the simulated curve in Fig. 10.

When introducing also the bypass option, the data reuse factor is increased compared to the previous results and the power consumption is reduced, as illustrated by the triangle points in Fig. 10. This makes copy-candidates with partial reuse much more interesting solutions for implementation, when there is not enough memory space available for maximum reuse. The generic code in Fig. 8 is easily adapted to a partial reuse copy-candidate. The copy-candidate is resized to `A_sub[cp][gamma]` and a conditional `(k > (kU-gamma-bp))` is included to discern between copied data and bypassed data.

[4] At the cost of a very irregular access pattern and thus a too huge penalty on code size and I-cache misses.

6.4. Test-vehicle: SUSAN principle.

In this section we show the results for a more complex test-vehicle. The SUSAN principle [27] is a basis for algorithms to perform edge detection, corner detection and structure-preserving image noise reduction. The image is accessed by moving a reference pixel in the horizontal and vertical direction, where each time the difference is computed with the surrounding pixels lying on a circular mask (see Fig. 11a). The original unfolded pointer-based loop body first has been (manually) pre-processed to a series of loops with different accesses to an array *image*. Each of the accesses is handled separately. However, the copy-candidates of accesses with identical index expressions are merged. An approximate solution is found when a conditional is present in the loop body, namely the loop accessing the middle row of the mask is not executed for the position where the reference pixel is located.

Again, the obtained data reuse without bypass is close to the simulated optimal data reuse factor curve and a factor of 1,6 to 6 decrease in power consumption is obtained (gray bullets in Fig. 11). Introducing a bypass for less reused data results in even more power gain for the smaller copy-candidate sizes (gray triangles in Fig. 11). To not overload the graphs, only the points lying on the new Pareto curve in Fig. 11b are plotted.

These results clearly show that the memory access power

consumption of data-dominated applications can be drastically reduced with the proposed technique. The analytical model provides us with a way to identify exactly close to optimal copy-candidates, and even more cost-effective solutions are found by introducing a bypass.

7. Conclusions

In this paper we have introduced an extended formalized data reuse exploration methodology to find optimal memory hierarchies in terms of power consumption, trading off with the memory size cost. An exact analytical model for the data reuse of a signal access is proposed, and we have shown that the cost parameters derived from this model exploit almost the maximum reuse possible, which was verified by comparing them with simulation-based results. Moreover, by introducing a bypass for data, for which there is no reuse in the sub-level, we can further reduce the power consumption. At the same time the solutions found are easily implemented, introducing a small overhead in the code, which can be eliminated by subsequent performance and address optimization steps. The analysis and subsequent code generation are completely automatable. A prototype tool has been developed which supports this technique.

Currently we are extending the model to characterize multiple level hierarchies.

ACKNOWLEDGEMENTS This project is partly sponsored by the IUAP project 4/24, FWO projects G.0036.99 and G.0160.02.

References

[1] F. Balasa, F. Catthoor, and H. De Man. Dataflow-driven allocation for multi-dimensional processing systems. In *IEEE Intnl. Conf. Comp. Aided Design*, San Jose CA, Nov. 1994.

[2] U. Banerjee, R. Eigenmann, A. Nicolau, and D. Padua. Automatic program parallelisation. *Proceedings of the IEEE, invited paper*, 81(2):211–243, Feb. 1993.

[3] L. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems J.*, 5(6):78–101, 1966.

[4] L. Benini, A. Macii, and M. Poncini. A recursive algorithm for low-power memory partitioning. In *IEEE International Symposium on Low Power Design*, pages 78–83, Rapallo, Italy, Aug. 2000.

[5] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, and A. Vandecappelle. *Custom Memory Management Methodology – Exploration of Memory Organisation for Embedded Multimedia System Design*. Kluwer Academic Publishers, 1998.

[6] J. Diguet, S. Wuytack, F. Catthoor, and H. De Man. Formalized methodology for data reuse exploration in hierarchical memory mappings. In *Proceedings of the IEEE International Symposium on Low Power Design*, pages 30–35, Monterey, CA, Aug. 1997.

[7] D. Gannon, W. Jalby, and K. Gallivan. Strategies for cache and local memory management by global program transformation. *Parallel and Distributed Computing*, 5:586–616, 1988.

[8] M. Hill. *Aspects of Cache Memory and Instruction Buffer Performance*. PhD thesis, Univ. of California at Berkeley, Nov 1987.

[9] 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.

[10] M. Kandemir, J. Ramanujam, and A. Choudhary. Improving cache locality by a combination of loop and data transformations. *IEEE Transactions on Computers*, 48(2):159–167, Feb. 1999.

[11] M. Kandemir, J. Ramanujam, M. Irwin, N. Vijaykrishnan, I. Kadayif, and A. Parikh. Dynamic management of scratch-pad memory space. In *Design Automation Conference (DAC)*, Las Vegas, USA, June 2001.

[12] P. Kjeldsberg, F. Catthoor, and E. J. Aas. Automated data dependency size estimation with a partially fixed execution ordering. In *International Conference on Computer Aided Design, ICCAD 2000*, pages pp. 44–50, San Jose, CA, Nov 2000.

[13] 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.

[14] T. Komarek and P. Pirsch. Array architectures for block matching algorithms. *IEEE Trans. on Circuits and Systems*, page 36, Oct. 1989.

[15] 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.

[16] A. W. Lim and M. S. Lam. Maximizing parallelism and minimizing synchronization with affine partitions. *Parallel Computing*, 24(3-4):445–475, May 1998.

[17] P. Lippens, J. Van Meerbergen, W. Verhaegh, and A. Van der Werf. Allocation of multiport memories for hierarchical data streams. In *IEEE Intnl. Conf. Comp.-Aided Design*, pages 728–735, Santa Clara, Nov. 1993.

[18] L. Liu. Issues in multi-level cache design. In *IEEE Intnl. Conf. on Computer Design*, pages 46–52, Cambridge MA, Oct. 1994.

[19] K. McKinley, S. Carr, and C.-W. Tseng. Improving data locality with loop transformations. *ACM Trans. on Programming Languages and Systems*, 18(4):424–453, July 1996.

[20] M. Miranda, F. Catthoor, M. Janssen, and H. De Man. High-level address optimisation and synthesis techniques for data-transfer intensive applications. *IEEE Trans. on VLSI Systems*, 6(6):667–676, Dec 1998.

[21] 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.

[22] P. Panda, N. Dutt, and A. Nicolau. Memory organization for improved data cache performance in embedded processors. In *1996 Intnl. Symp. on System Synthesis*, pages 90–95, La Jolla CA, Nov. 1996.

[23] D. Patterson and J. Hennessy. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, San Mateo, CA, 1990.

[24] S. Przybylski, M. Horowitz, and J. Hennessy. Performance tradeoffs in cache design. In *15th Annual International Symposium on Computer Architecture*, pages 290–298, Honolulu, Hawaii, 1988.

[25] L. Ramachandran, D. Gajski, and V. Chaiyakul. An algorithm for array variable clustering. In *European Design and Test Conf.*, pages 262–266, Paris, France, March 1994.

[26] W.-T. Shiue, S. Tadas, and C. Chakrabarti. Low power multi-module, multi-port memory design for embedded systems. In *IEEE Workshop on Signal Processing Systems (SIPS)*, pages 529–538, Lafayette LA, Nov. 2000.

[27] S. Smith and J. Brady. Susan - a new approach to low level image processing. *International Journal of Computer Vision*, 23(1):45–78, May 1997. Source code: http://www.fmrib.ox.ac.uk/∼steve/susan/susan2l.c.

[28] 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. *Design Automation for Embedded Systems*, 5(3):313–327, 2000.

[29] T. Van Achteren, R. Lauwereins, and F. Catthoor. Systematic data reuse exploration methodology for irregular access patterns. In *IEEE/ACM 13th International Symposium on System Synthesis (ISSS)*, pages 115–121, Madrid, Spain, Sept 2000.

[30] S. Wuytack, F. Catthoor, F. Franssen, L. Nachtergaele, and H. De Man. Global communication and memory optimizing transformations for low power systems. In *IEEE Intnl. Workshop on Low Power Design*, pages 203–208, Napa CA, April 1994.