Formalized methodology for data reuse exploration in hierarchical memory mappings

J.Ph. Diguet*, S. Wuytack, F. Catthoor and H. De Man
IMEC, Kapeldreef 75, B-3001 Leuven, Belgium, *name*@imec.be

ABSTRACT

Efficient use of an optimized memory hierarchy to exploit temporal locality in the memory accesses on array signals can have a very large impact on the power consumption in data dominated applications. In the past, this task has been identified as crucial in a complete low-power memory management methodology. But effective formalized techniques to deal with this specific task haven't been addressed yet. In this paper the design freedom available for the basic problem is explored in-depth and the outline of a systematic solution methodology is proposed. The efficiency of the methodology is illustrated on a real-life motion estimation application.

1 Introduction

The idea of using memory hierarchy (MH) 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. So, by exploiting hierarchy Data-Reuse (DR), 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 [1]. So, MH optimization will introduce copies of data from larger to smaller memories in the Data Flow Graph (DFG). This means that there is a trade-off involved 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. The aim of the MH design task, identified as an important step of the whole methodology, is to find the best solution for this trade-off at an early stage of the design strategy. Note also that the problem is basically different from caching for performance [2, 3, 4, 5] where the question is to find how to fill the cache such that the data needed have been loaded from main memory before. This question doesn't address the number of transfers that can even grow.

The main related work to this problem lies in the parallel compiler area, especially related to the cache hierarchy. Here, several papers have analyzed memory organization issues in processors [6, 7]. This, however, has not resulted yet in any formalizable method to guide the memory organization issues and is mostly oriented to stream data organization (not to array signals). 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, 9, 10, 11, 12]. The relation with the compiler optimization is sometimes incorporated [13], but even then it remains run-time oriented. In this context, also work has been going on optimal policies for page fetching and replacement [14]. Some approaches address the hierarchical data organization in processors for programs with loop nests [15]. Partitioning or blocking strategies for loops to optimize the use of caches have been studied in several flavours and contexts [2, 3, 4]. Recently, also multi-level caches have been investigated [5]. The main focus is on performance improvement though and not on memory cost. Recently, also in a system synthesis context, this issue of transformations to improve the cache usage has been addressed [16, 17]. None of these approaches determine the best MH/DR organization for a given (set of) applications and only few address the power cost. Only an analysis of MH choices based on statistical information to reach a given throughput expectation has been discussed recently [18]. In the hardware realization context, much less work has been performed, mainly oriented to memory allocation [19, 20, 21]. Only few papers with a detailed analysis on MH experiments in real-life applications have been published. The impact of this step turns out to be very large in the entire methodology for power reduction. This has been demonstrated by us for a H.263 video decoder [22] and a motion estimation application [17]. What is not yet solved, however, is how to decide on the optimal use of the MH in a systematic way, i.e. how to solve the DR issue.

In this paper we present a formalized methodology for this particular task. At IMEC this task is embedded in the global methodology of ATOMIUM, but the DR subtask as such can also be viewed as a preprocessing step of other memory organisation approaches, and we give an indication of how large the search space really is. A full description of our complete script may be retrieved from our web site[1]. More details about the DR methodology and the question of its positioning in the global script may be found in [23].

This search space is much larger than conventionally ex-

* Supported by a French Government Fellowship - *LAVOISIER*

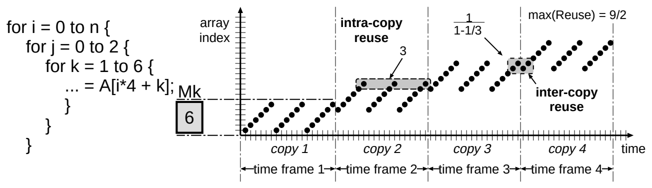
[1] http://www.imec.be/vsdm/domains/designtechno/index.html
ftp://ftp.imec.be/pub/vsdm/reports

ploited in state-of-the-art designs. Given the very large complexity of the entire problem, it is impossible to discuss all the issues in one paper. In this initial paper we discuss a simplified version of the problem. We want especially to motivate the importance of the DR issue and to formalize the main subtasks needed to solve it. The ordering of the array accesses is assumed to be fully fixed here. The search space becomes much larger still when also the available freedom in loop reorganization is incorporated but this issue will be dealt with in another paper. The question of data-reuse based on splitting memories and the CAD techniques, still under development by us, will also not be discussed here. The rest of the paper is organized as follows. Section 2 defines the DR problem. Section 3 explores the problem and points out important issues towards a solution. In Section 4 we present a real-life example, namely a motion estimation kernel, that we use to illustrate the methodology detailed in Section 5. We comment the global results of this interesting experiment in Section 6. Section 7 concludes the paper.

2 Data Reuse Problem Formulation

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 here been represented as groups of dots. Every dot represents a memory read operation accessing a given array element. In general, image and video processing algorithms contain too many loops with too large loop bounds to represent them in this “unrolled” way so in practice, polyhedral techniques [24, 25] will be used further on to model and manipulate the array accesses.



```
for i = 0 to n {
   for j = 0 to 2 {
      for k = 1 to 6 {
         ... = A[i*4 + k];
      }
   }
}
```

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

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 (vertical dashed lines), we see that only part of the data is needed several times 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 for the following usages, a data element can be read from the smaller memory instead of the larger memory.

Taking full advantage of temporal locality for power, usually requires architectural transformations that correspond to adding several layers of memories (each corresponding to its own time frame level) between the main background memories and the small foreground memories (registers in the datapath). If the original loop ordering is not optimally suited to exploit this locality, algorithmic transformations should be applied to increase it. In our general approach, these modifications are integrated in the methodology but in this paper, we will assume that loop ordering is fully fixed to simplify the discussion. Every layer in the MH contains smaller memories than the ones used in the layer above it. A generic MH for all read accesses operating on one array signal is shown in Fig. 2. The relevant parameters for every layer M_i are the size S_i and the number of transfers N_i. In an optimal MH $S_i < S_{i-1}$ for every i.

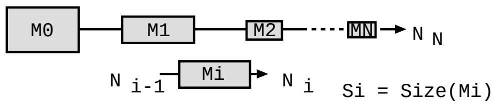


Figure 2: Multi-layer MH for a given signal.

2.2 Data Reuse and In-Place Mapping

Data-reuse factor The usefulness of hierarchy 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 $i-1$ is defined as:

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

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. Fig. 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 of 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 doesn’t have to be copied again. If $F_R(0, 1, D) = 1$, MH is useless and would even lead to an increase of area and power, because the number of read operations from M_0 would remain unchanged while the data also has to be stored and read from M_1. Let’s consider the simple example in Fig. 1 showing a sub-memory M_k with a size equal to 6 words. In this case, we see that each data in a copy is read three times, so the *intra-reuse* factor equals 3. We also note that $1/3$ of the data in a given copy will be reused in the subsequent copy, so the *inter-reuse* factor is $\frac{1}{1-1/3}$. Finally we obtain $9/2$ as the maximal value for the reuse factor. Practically, the computation of these features is much more complex but can be handled by polyhedral techniques.

In-place mapping assumption We assume that at the end of a time frame, the data copied to the intermediate layer corresponding to that time frame level is not needed anymore and can therefore be overwritten by the data needed in the next time frame. Therefore, the size of the memory on a given layer is determined by the amount of data that has to be copied to this memory within one time frame[2]. To exploit *inter-copy* reuse, data that is already in a memory and would be copied again to that memory has to be kept alive, which means that this memory space cannot be used to store other data for the next time frame. But at the same time, less data has to be copied and therefore also stored. In the end, the memory size requirements are equal whether or not *inter-copy* reuse is taken into account. Only the number of transfers is affected.

3 Formalized Problem Exploration

In this section we will explore the DR problem more formally, and point out some elements that can be used to define a systematic methodology for DR exploration.

3.1 Read vs write operations

DR has to focus on read operations only. The reason is that repeated reading of the same data *value* makes sense, whereas writing *the same* data *value* usually doesn't make sense. So there is no need for creating a MH for repeatedly written data. The only thing that has to be decided for write operations is which signals will be written in (temporary) buffer. That important but independent decision is taken earlier in the global ATOMIUM script during the *Loop Transformation* step [17] and won't be further discussed in this paper.

3.2 Divide and conquer

The hierarchy design problem can best be tackled for each array signal separately for three main reasons, discussed hereafter in a kind of growing impact order. The first one is that even then it is already complex enough. Secondly, the different signals usually have distinct sizes and reuse rates, so they need various kind of ad hoc hierarchies. Thirdly, in order to be efficient, hierarchy decisions have to be handled at a high level of abstraction. They must be taken early in the design script[1], before cycle budget distribution [26] when no accurate information is yet available about *Inter-signal In-Place*. This means that every signal has its own DR scheme that will be combined later on into a common *physical MH*.

3.3 Classification of MH opportunities

In this subsection we present a classification of three cases in which memory hierarchy for array signals may be useful (cfr. Fig. 3).

Read once a signal in a (nested) loop This is the basic case (Fig. 3-a) on which our methodology is based. Both *intra-copy* and *inter-copy* reuse are possible here, when the loop nesting (i.e., the order of the different nested loops, and the direction in which the loops are traversed) is fixed. Indeed in this case, the ordering of the different copies is known and every two consecutive copies can be examined for overlap (i.e., *inter-copy* reuse).

[2]or the maximum over all time frames in case this is not constant.

```
For ...
  For ...
    <- F1(A[][])
  End
End
```
(a)

```
...
For ...
  For ...
    <- F1(A[][])
    ...
    <- F2(A[][])
  End
End
...
```
(b)

```
...
For ...
  For ...
    <- F1(A[][])
  End
End
...
For ...
  For ...
    <- F2(A[][])
  End
End
...
```
(c)

Figure 3: Classification of data reuse and loop schemes

Read repeatly a signal in a (nested) loop Here (Fig. 3-b) it is assumed that each of the read operations has a different index expression, because otherwise they can be reduced to the previous case by reading once from background memory and storing the result in a temporary foreground register. When the different read operations are accessing different parts of the signal, a different MH can be constructed for each of them. However, in practice, these memory hierarchies will contain partly the same data, and are therefore best combined. Determining which part of the MH can be shared, can be done with a kind of polyhedral basic set analysis [23].

Read a signal from distinct loop nests In this case (Fig. 3-c), a MH can be derived for each of the loop nests separately. Because they are in different loop nests, these DR schemes can be very different from each other. So, heuristics aiming at the *best-fitting* common hierarchy, reducing the final MH complexity must be used. This more complex case requires a distinct detailed development and will not be further discussed in this paper.

3.4 Conclusions from the classification

A number of conclusions can be drawn from this classification:

1. For every read operation in a perfectly nested loop, we can determine a chain of possible memory candidates independently from other read operations. This is called a *logical DR chain*. Two simple rules can be applied to prune memory candidates from *logical DR chains*, because they will never be used in an optimal physical MH: the size of the memory candidates must decrease from one layer to the next ($S_i < S_{i-1}$), and the reuse ratio $F_R(i-1, i, D)$ of each level must be larger than 1.

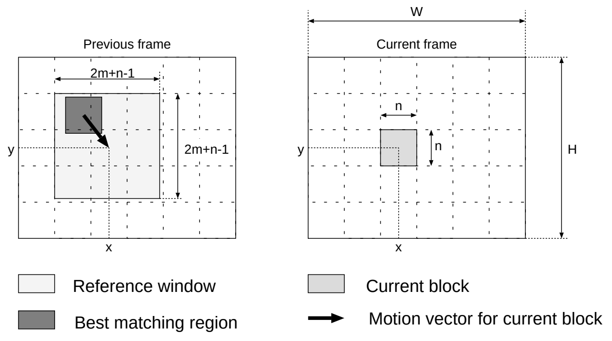
2. A read operation on an indexed signal that is not surrounded by loops can be considered as a degenerate case of a read operation inside a loop with only one iteration. In the area of data-dominated applications, these cases provide negligible gains and can be ignored, otherwise some more traditional register allocation techniques can be used.

3. *Logical DR chains* of different read operations to the same signal, must be examined for sharing opportunities.

This is needed to reduce both the search space and the number of proposed sub-memories, as explained above. The result is a shared *logical DR tree*. Remark that the root node is always common to all read operations accessing the same signal. Therefore this combination into a tree is always possible (see Section 5.2). However, it has to be assumed that temporal locality is feasible in order to exploit in-place savings on the size ($S_i < S_{i-1}$). Therefore, the loop nests ordering will be guided, during the *DFG balancing* task, as much as possible by previous DR decisions.

4 Test vehicle: Motion Estimation Algorithm

In this section we introduce a motion estimation algorithm which will be used as a test-vehicle to illustrate the methodology proposed in the next section.



```
for (g=0; g<H/n; g++)            /* vert. CB counter */
  for (h=0; h<W/n; h++) {        /* horz. CB counter */
    $\Delta_{opt}[g][h] = +\infty;$
    for (i=-m; i<m; i++)         /* horz. searching of RW */
      for (j=-m; j<m; j++) {     /* vert. searching of RW */
        $\Delta = 0;$
        for (k=0; k<n; k++)      /* horz. traversal of CB */
          for (l=0; l<n; l++) {  /* vert. traversal of CB */
            $\Delta$ += abs($New[g.n+k][h.n+l]$
                                 $-Old[g.n+i+k][h.n+j+l]$;)
          }
        $\Delta_{opt}[g][h]$ = min($\Delta$, $\Delta_{opt}[g][h]$);
      }
  }
```

Figure 4: The motion estimation algo. and its parameters

The motion estimation algorithm [27] is used in moving image compression algorithms. It allows to estimate the motion vector of small blocks of successive image frames. The version we consider here is the kernel of what is commonly referred to as the "full-search full-pixel" implementation [28]. The algorithm and its parameters are shown in Fig. 4. The basic operation at the inner loop consists of an accumulation of pixel differences, while the basic operation two levels higher in the loop hierarchy consists of the calculation of the new minimum and its location.

5 Proposed Methodology

In this section we propose a methodology based on a number of assumptions to make it feasible for real-life applications. However, regarding to large power savings, the principle of the global methodology suits to much more complex schemes than the ideal one thanks to heuristics that we can't detail here.

5.1 Assumptions

- Currently we assume single assignment code, where every array value can only be written once but read several times. This makes the analysis much easier.
- We assume that the order and direction of loop iterators of nested loops is fixed. This is compatible with the position of the DR decision in our complete ATOMIUM memory management methodology [17].
- Finding an optimal time-frame hierarchy (see Fig. 1) is a very complex problem. However, we believe that the optimal time-frame boundaries are likely to coincide with the loop boundaries of loop nests. This idea is reinforced by the position of the abstract DR decisions that are far before the final implementation. Therefore, as heuristics we use firstly the loop boundaries as time-frame boundaries, instead of trying to find optimal time-frame boundaries. Secondly, we take the operand spaces associated to the different time-frames to obtain memory sizes. For normal writing styles of the designer this leads to (very) good results.

5.2 Hierarchy tree building

Logical DR chains & trees From Section 3 we can conclude that for every signal in a loop nest, a *logical DR chain* can be determined. Each node in the chain represents a memory candidate[3] based on the loop nest structure. A memory candidate can be characterized by its required memory size that is directly related to the operand space defined by the associated loops.

Regarding the motion estimation application, only the frame signals Old (O) and New (N) will be considered here, because the other signals can easily be stored in a foreground register. Therefore, they can be pruned from the problem. For the signal N there is only one read operation. This read operation is contained in a nested loop. Therefore we can construct a *logical DR chain* for it, made of nodes NF, NF_1 and NF_2 in Fig. 5 with sizes respectively equal to S_N: size of a *frame*, S_{N1}: size of a *row of blocks* and S_{N2}: size of a *block*. Thus, every nested loop corresponds a memory candidate and a time-frame level. The size of memory candidates can be computed by means of a basic set analysis. Because there is only one read operation to New, the *logical DR chain* can't be merged with another chain. The *logical DR tree* of New is thus a chain in this case. This procedure is repeated, in the same way, for signal Old.

Logical DR graphs From the chains other valid DR schemes can be derived because it is allowed to copy data from any ancestor node in the tree, not only the parent node. Therefore, we propose to extend the *logical DR tree* to a *logical DR graph* in the following way: for every node in the tree, we add edges starting from all its ancestor nodes towards the node itself.

The edges between the tree nodes are weighted by the number of write operations to copy data from one node to

[3]This is not an accurate name, because memory allocation doesn't necessarily have to select one of those candidates.

the other, and the total number of read operations to memories on lower layers in the MH. The data-transfers strongly depend on the way data-reuse is exploited. In our methodology, we assume a maximal data-reuse. The main reason is that we will maximize data reuse by improving data locality and reducing data transfers in the following steps. This will be achieved with DFG transformations as *Loop folding* and loop merging. Moreover, at this stage, not enough information is available to do a more accurate estimation.

Let's come back to the application. The *logical DR trees* of signals N and O are now extended to *logical DR graphs* as described in Fig. 5 where some memory candidates have been pruned (see Section 3.4). The arrows represent the number of transfers that are needed to copy data from a larger to a smaller memory in the hierarchy.

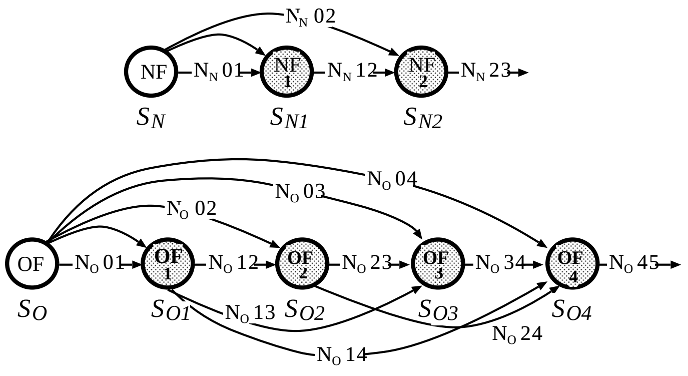


Figure 5: Logical DR graphs

Optimal DR trees Within each *logical DR graph*, all possible trees that can be derived by selecting a single path from the root node to every leave node, represent a valid *logical DR tree*. The *optimal logical DR tree* is the tree obtained this way with lowest cost.

All the valid search trees for finding the *optimal logical DR tree* for both the *Old* and *New* frames are shown in Fig. 6 where M_i represents the level of hierarchy. The grey trees indicate the lowest-cost solutions. The crossed leaves represent the largest pruned candidates that don't provide data-reuse factors greater than one.

5.3 Cost function

Let $P^{r/w}(N_{bits}, N_{words}, F_{access})$ be the power function for read and write operations. It depends on the estimated size and bit-width of the final memory, as well as on the real access frequency F_{access} of the array signals, which is obtained by multiplying the number of memory accesses per frame for a given signal with the frame rate of the application (this is **not** the clock frequency). The cost function we are using is defined as :

$$F_c(M_i) = \sum_{j=1}^{i} \alpha\,[P^r(M_j) + P^w(M_j)] + \beta.A(j) + \gamma.Nmem(j) \quad (2)$$

Here α and β are weighting factors for Area/Power trade-offs, and γ is a weighting factor to take into account the increase of interconnections due to a growing number of memories.

6 Results on power and area impact

The motion estimation example has been analyzed with parameters from the QCIF format (W=176, H=144, m=n=8) with a frame rate of 30 frames/s. We are using an accurate but proprietary model for estimating the power and area of the memory modules from a specific library for which we are not allowed to publish absolute values on area and power. Therefore only relative values are provided.

The search trees for the two relevant signals (*Old* and *New*) with area and power figures are shown in Fig. 6. The dashed lines divide the search trees in a number of layers. Each layer corresponds to a memory candidate (or time-frame level). The solutions can either include this memory candidate (memory is shown in search tree) or skip it (memory is not shown). Every node in the tree represents a solution: the memory candidate on that node is the lowest layer in the hierarchy, and all memory candidates on the path between the root node and the node itself are intermediate layers in the hierarchy. For each solution the area (A) and power (P) of the complete MH for that solution relative to the solution without hierarchy (root node) are indicated.

A surprising result is that the total memory area can $decrease$ by adding extra memory layers. The reason for this is that the maximum access frequency of the memories is taken into account in our estimations. If a certain memory would be accessed above its maximum access frequency, this memory will be split into two memories of half the size to increase the memory bandwidth. This splitting introduces overhead. By adding extra memory layers with small memories, the bandwidth requirements of the large background memories can be reduced, and therefore splitting can be avoided. This area gain can be larger than the area lost by adding a few small memories.

The optimal MH for power (without interconnect) is :

- for the *Old* frame, a 3 level hierarchy that leads to a power saving of 83% compared to the solution without memory hierarchy;
- for the *New* frame, a 2 level hierarchy that leads to a power saving of 87% compared to the solution without memory hierarchy.

If we compare this result with the one we proposed in an ad hoc way, in an earlier paper [17], we note that a different MH with only two levels was selected wich results in a higher power consumption. This clearly shows that by using a more systematic design space exploration methodology which exploits the full search space available, as proposed in this paper, better results can be obtained.

The power savings that can be achieved through DR are relevant enough to justify placing the DR design task early in the memory management script.

7 Conclusion

This paper has pointed out the relevance of data-reuse decisions in hierarchically organized memories for power optimization of data-dominated applications. A new approach has been proposed to tackle the hierarchy decision problem in an applicative context though the discussion in this paper has mostly focussed on the simpler sequential context where is it also useful. The feasibility and the large impact of the proposed techniques has been shown on a real-life video application.

Applying the methodology without care to complex loop schemes, containing distributed read operations, would rapidly lead to an intractable amount of DR opportunities. Moreover the DR decisions must be done very early in the design flow. Hence, it would be inefficient when they impose too strong and accurate constraints on the following steps. Consequently, a preprocessing step has to be added to simplify the general hierarchy trees in order to obtain sub-optimal but yet powerful hierarchy-schemes like those presented in Section 3. Finally, the proposed principle remains valid for more irregular and complex applications.

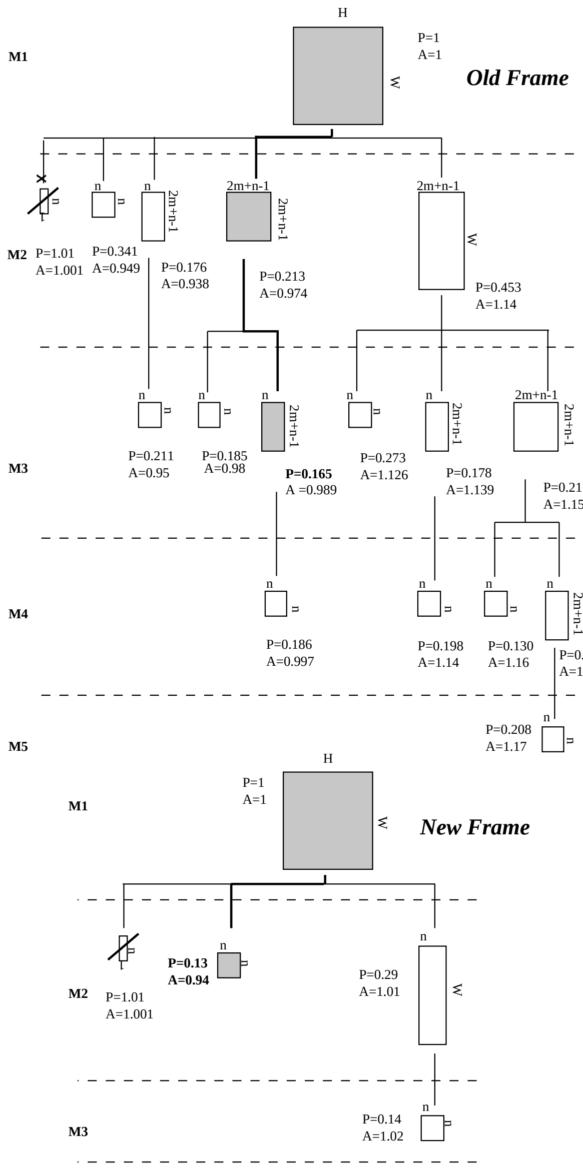


Figure 6: Search trees for the optimal logical DR graphs

REFERENCES

[1] S.Wuytack, F.Catthoor, L.Nachtergaele, and H.De Man, “Global communication and memory optimizing transformations for low power systems,” in *IEEE/ACM Int. Work. on Low Power Design*, Napa Valley, CA, Apr. 1994, pp. 203–208.

[2] J.Z.Fang and M.Lu, “An iteration partition approach for cache or local memory thrashing on parallel processing,” *IEEE Trans. on Computers*, vol. C-42, no. 5, pp. 529–546, May 1993.

[3] D.Kulkarni and M.Stumm, “Linear loop transformations in optimizing compilers for parallel machines,” Tech. Rep., Comp. Systems Res. Inst. Univ. of Toronto, Oct. 1994.

[4] N.Manjiakian and T.Abdelrahman, “Reduction of cache conflicts in loop nests,” Tech. Rep., CSRI-318, Comp. Systems Res. Inst. Univ. of Toronto, Cananda, Mar. 1995.

[5] M.Jimenez, J.Llaberia, A.Fernandez, and E.Morancho, “A unified transformation technique for multi-level blocking,” in *Proc. EuroPar Conf. ”Lecture notes in computer science” series, Springer Verlag*, Lyon, France, Aug. 1996, pp. 402–405.

[6] S-M.Moon and K.Ebcioglu, “A study on the number of memory ports in multiple instruction issue machines,” *Micro’26*, pp. 49–58, Nov. 1993.

[7] 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*, Boston MA, Oct. 1991, pp. 865–877.

[8] M.Dubois and J-C.Wang, “Analytical modeling of data sharing in cache based multiprocessors,” Tech. Rep., CENG 89-18, U.S.C, June 1989.

[9] K.Gharachorloo, A.Gupta, and J.Hennessy, “Performance evaluation of memory consistency models for shared-memory multiprocessors,” in *4th Int. Conf. on Arch. Support for Progr. Lang. and Oper. Systems*, Apr. 1991, pp. 245–257.

[10] L.Liu, “Issues in multi-level cache design,” in *IEEE Int. Conf. on Computer Design*, Cambridge MA, Oct. 1994, pp. 46–52.

[11] P.Stenström, “A survey of cache coherence schemes for multiprocessors,” Computer, vol. 23, no. 6, pp. 12–24, June 1990.

[12] J.Gee and A.Smith, “Analysis of multiprocessor memory reference behavior,” in *IEEE Int. Conf. on Computer Design*, Port Chester NY, Oct. 1994, pp. 53–59.

[13] L.Choi and P-C.Yew, “A compiler-directed cache coherence scheme with improved intertask locality,” in *Supercomputing*, Washington DC, Nov. 1994.

[14] A.Choi and M.Ruschitzka, “Managing locality sets: the model and fixed-size buffers,” *IEEE Trans. on Computers*, vol. 42, no. 2, pp. 190–204, Feb. 1993.

[15] F.Bodin, W.Jalby, D.Windheiser, and C.Eisenbeis, “A quantitative algorithm for data locality optimization,” Tech. Rep., IRISA (INRIA/CNRS), Rennes, France, 1992.

[16] D.Kolson, A.Nicolau, and N.Dutt, “Elimination of redundant memory traffic in high-level synthesis,” *IEEE Trans. on Computer-Aided Design*, vol. 15, no. 11, pp. 1354–1363, Nov. 1996.

[17] S.Wuytack, F.Catthoor, L.Nachtergaele, and H.De Man, “Power exploration for data dominated video applications,” in *IEEE/ACM Int. Symp. on Low Power Design*, Monterey, USA, Aug. 1996, pp. 359–364.

[18] B.Jacob, P.chen, S.Silverman, and T.Mudge, “An analytical model for designing memory hierarchies,” *IEEE Trans. on Computers*, vol. C-45, no. 10, pp. 1180–1193, Oct. 1996.

[19] P.Lippens, J.van Meerbergen, W.Verhaegh, and A.van der Werf, “Allocation of multiport memories for hierarchical data streams,” in *ICCAD*, Santa Clara, CA, Nov 1993.

[20] L.Ramachandrana, D.Gajski, and V.Chaiyakul, “An algorithm for array variable clustering,” in *5th ACM/IEEE Europ. Design and Test Conf.*, Paris, France, Feb 1994, pp. 262–266.

[21] F.Balasa, F.Catthoor, and H.De Man, “Dataflow-driven memory allocation for multi-dimensional,” in *ICCAD*, Santa Jose, CA, Nov 1994.

[22] L.Nachtergaele, F.Catthoor, B.Kapoor, S.Janssens, and D.Moolenaar, “Low power storage for H.263 video decoder,” in *IEEE Work. on VLSI Signal Processing*, Monterey, CA, Oct. 1996, pp. 115–124.

[23] J.Ph. Diguet, S.Wuytack, and F.Catthoor, “Hierarchy exploration in high level memory management,” Tech. Rep., IMEC, Leuven, Belgium, June 1997.

[24] F.P.Preparata and M.I.Shamos, *Computational geometry, an introduction*, Springer-Verlag, New York, 1985.

[25] D.K.Wilde, “A library for doing polyhedral operations,” Tech. Rep. 785, IRISA (INRIA/CNRS), Rennes, France, dec 1993.

[26] S.Wuytack, F.Catthoor, G.De Jong, B.Lin, and H.De Man, “Flow graph balancing for minimizing the required memory bandwidth,” in *9th IEEE/ACM Int. Symp. on System Synthesis*, La Jolla, CA, Nov. 1996, pp. 127–132.

[27] C.Lin and S.Kwatra, “An adaptive algorithm for motion compensated colour image coding,” in *Proc. IEEE Globecom*, 1984, pp. 47.1.1–4.

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