

Dataflow-driven Memory Allocation for Multi-dimensional Signal Processing Systems *

Florin Balasa

Francky Catthoor[†]

Hugo De Man[†]

IMEC, Kapeldreef 75, B-3001 Leuven, Belgium

Abstract

Memory cost is responsible for a large amount of the chip and/or board area of customized video and image processing systems. In this paper, a novel background memory allocation and assignment technique is presented. It is intended for a behavioural algorithm specification, where the procedural ordering of the memory related operations is not yet fully fixed. Instead of the more restricted classical scheduling-based explorations, starting from procedurally interpreted specifications in terms of loops, a novel optimization approach – driven by data-flow analysis – is proposed. Employing the estimated silicon area as a steering cost, this allocation/assignment technique yields one or (optionally) several distributed (multi-port) memory architecture(s) with fully-determined characteristics, complying with a given clock cycle budget for read/write operations. Moreover, our approach can accurately deal with complex multi-dimensional signals by means of a polyhedral data-flow analysis operating with groups of scalars.

1 Introduction

Application studies in the areas of speech, image and video processing indicate that a large part of the area and power cost in (application-specific) architectures for real-time multi-dimensional signal processing (RMSP) is due to *memory units*, i.e. single or multi-port RAMs, pointer-addressed memories, and register files [6, 4]. Therefore, design support to determine the *storage organization* for the *multi-dimensional (M-D) signals* is crucial to reduce the system and architecture design time while maintaining a sufficiently high design quality in terms of area and/or power. Such decisions should be fixed as early as possible in the design trajectory, preferably before data-path allocation and scheduling [4].

Many techniques tackling the storage allocation problem employ a *scheduling-driven scalar-oriented* view [5, 1, 9] where the control steps of production/consumption are assumed to be known for each individual scalar. This strategy is mainly due to the fact that applications targeted in conventional high-level synthesis contain a relatively small number of scalar signals (at most $\sim 10^3$). In that case, (binary) ILP formulations, graph colouring, or clique partitioning techniques have provided satisfactory results for register allocation, signal-to-register assignment, and signal-to-port assignment, under the (usually implicit) mentioned assumptions.

*This research was partly sponsored by the ESPRIT Basic Research Action 6632 (NANA-2) project of the E.C.

[†]Professor at the Katholieke Universiteit Leuven

A behavioural video domain specification like that of Fig. 1 is untractable by means of a scalar-oriented technique. First, this class of examples usually contains large amounts of scalars. At the same time, scalar-oriented approaches entail a loss of the code regularity. According to our experience, this leads to an unacceptable growth of the controller size. In addition, an applicative language like SILAGE is by definition *non-procedural*: besides the natural production-consumption order imposed by the dependence relations existent in the code, there is much freedom left in the execution ordering. This can be exploited by a designer in order to better meet his goals, e.g. to take profit of the parallelism “hidden” in the code.

Most tools do not tolerate this however. E.g. the current memory tool of the CATHEDRAL system cannot handle the example of Fig. 1, as it interpretes the ordering in the source code procedurally [10]. The MESA allocation approach [8] can deal with large indexed signals but it also starts from a procedurally interpreted loop structure. A nice feature of this system is that it tries to take into account a realistic allocation cost function based on a memory layout model and throughput. As no data-flow analysis is accomplished to steer the allocation process, it does not take into account the possibility of storing (parts of) M-D signals in-place, resulting in an important over allocation of memory. In contrast, the allocation tool MEDEA from the PHIDEO system [6] is capable of handling non-procedural applications (in terms of stream ordering). By means of a hierarchical stream model, it is possible to handle M-D signals with complex affine indices, but only in nests of loops with constant boundaries: the PHIDEO stream model is more targeted towards fixed-rate front-end video applications, and not to irregular image or speech processing systems which typically do not exhibit fixed periods and fixed length streams.

In this paper, a novel background memory allocation and assignment technique is presented. The proposed approach is driven by data-flow, which leads to a larger flexibility in the search space exploration compared to the scheduling-driven solutions. In order to handle applications with large amounts of scalar signals (mainly organized in M-D signals), our allocation system incorporates a polyhedral data-flow analysis [2], allowing to operate directly with *groups* of signals, *analytically characterized*, rather than using the *individual* scalar signals. This novel allocation approach can process non-procedural functional descriptions, containing conditions (data-dependent or not), delays, M-D signals with complex affine indices, and nests of loops having as

Permission to copy without fee all or part of this material is granted, provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

```

func main (A: W[G+1][H+1]) opt: fix<16,4>[[]] =
begin (i: -M .. M)::
(j: -N .. N)::
(k: -m .. m)::
(l: -n .. n)::
(g: M+m .. G-M-m)::
(h: N+n .. H-N-n):: begin
D[g][h][g+i][h+j][0] = if (k==m)&(l==n) -> fix<16,4>(0) fi;
optD[g][h][0] = if (i==M)&(j==N)&(k==m)&(l==n) -> fix<16,4>(0) fi;
opt[g][h] = if (i==M)&(j==N)&(k==m)&(l==n) -> optD[g][h][int((2*M+1)*(2*N+1))]fi;
Ad[g][h][g+i][h+j][g+i+k][h+j+1] = A[g+i+k][h+j+1];
Ad[g][h][g+i][h+j][g+i+k][h+j+1]@1 = fix<16,4>(0);
D[g][h][g+i][h+j][int((k+m)*(2*n+1)+l+n+1)] = A[g+i][h+j] -
Ad[g][h][g+i][h+j][g+i+k][h+j+1]@1+D[g][h][g+i][h+j][int((k+m)*(2*n+1)+l+n)];
optD[g][h][int((i+M)*(2*N+1)+j+N+1)] = if (k==m)&(l==n) ->
D[g][h][g+i][h+j][int((2*m+1)*(2*n+1))] + optD[g][h][int((i+M)*(2*N+1)+j+N)]fi;
end;
end;

```

Figure 1: A motion-detection kernel for video coding

loop boundaries affine functions of the surrounding loop iterators. The allocation/assignment technique yields one (or optionally several) distributed (multi-port) memory architecture(s) with fully-determined characteristics, complying with a given clock cycle budget allocated for read/write operations. Our approach currently employs as a steering cost function an accurate estimation of the total silicon area occupied by the memory architecture.

Section 2 presents the mathematical model of the simultaneous memory and port allocation and the optimization technique to solve it. Section 3 introduces the signal-to-memory assignment method. Results obtained so far are substantiated in Section 4, followed by conclusions and our future directions of research in Section 5.

2 Background memory allocation

The subsequent allocation model produces a memory configuration of RAMs and ROMs "optimal" in terms of silicon area requirement, and complying with a given clock cycle budget allocated for R/W operations. The proposed solution approach provides answers to the following practical questions: (1) How many memories are necessary? (2) What is their type (RAM/ROM)? (3) What are their word-lengths? (4) How many ports has each memory? (5) What kind of ports are required for each of the memories? (How many *read* ports, how many *write* ports, and *read/write* ports?) (6) What are their estimated sizes (numbers of locations)?

The cost function employed by our approach is based on an evaluation of the actual silicon area occupied by the background memory, based on the model for on-chip memories proposed by Mulder *et al.* [7]:

$$A = TF \cdot b \cdot (1 + \alpha P) \cdot (N + \beta) \cdot [1 + (P + P_{rw} - 2)/4]$$

where: TF is a technology scaling factor, b - the memory word-length, N - the number of storage locations, P - the total number of ports (*read*, *write*, and *read/write*), P_{rw} - the number of *read/write* ports.

Assuming that the word-lengths of the signals in a RMSP algorithm are $b_1 > \dots > b_m$, and denoting with $RW(i)$ the maximum number of simultaneous R/W operations of signals having as word-length b_i bits, there must be at most $RW(i)$ memories of b_i bits. Consequently, the cost function to be minimized is the total area estimation: $\sum_{i=1}^m \sum_{j=1}^{RW(i)} A_{ij}$, where

$$A_{ij} = \begin{cases} kb_i(1 + \alpha P_{ij})(N_{ij} + \beta)[1 + (P_{ij} + p_{ij}^{rw} - 2)/4] \\ 0, & \text{if } N_{ij} = 0 \\ 0, & \text{if } P_{ij} = 0 \end{cases}$$

A_{ij} denotes the area of memory j from the collection of at most $RW(i)$ memories which can have storage cells of b_i bits wide. (Some of these memories will result to be 0 after the optimization process.) N_{ij} and $P_{ij} = p_{ij}^r + p_{ij}^w + p_{ij}^{rw}$ denote the number of locations respectively the number of ports. The subscripts have the same significance as above and the superscripts of ports denote the *read*, *write*, or *read/write* function.

There are two categories of constraints. The first class refers to port configurations:

$$\sum_{j=1}^{RW(k)} (p_{kj}^r + p_{kj}^{rw}) \leq R(\overline{k, m}) \quad (1)$$

$$\sum_{i=1}^k \sum_{j=1}^{RW(i)} (p_{ij}^r + p_{ij}^{rw}) \geq R(\overline{1, k}) \quad (2)$$

for $k = \overline{1, m}$. Constraint set (1) limits the reading capacity for memories kj with b_k bits to be the maximum number of simultaneous *read* accesses referring to signals of word-lengths (b_k, \dots, b_m) , denoted as $R(\overline{k, m})$. Constraint set (2) ensures that the number of ports for all memories of b_1, \dots, b_k bits is sufficient to support is the maximum number of simultaneous *read* accesses referring to signals of word-lengths (b_1, \dots, b_k) , denoted as $R(\overline{1, k})$. All these R/W values are determined during our data-flow analysis stage [2]. Similar constraint sets refer respectively to *write*, and *read/write* accesses. Another set of port constraints ensures that each memory ij has both *read* and *write* accesses (if RAM assumption is effective): e.g. $p_{ij}^r = p_{ij}^{rw} = 0 \implies p_{ij}^w = 0$. Linear constraints modelling these conditions are:

$$W(\overline{i, m}) (p_{ij}^r + p_{ij}^{rw}) \geq p_{ij}^w \quad (3)$$

Our implementation also allows user interaction. For instance, optionally a predefined set of port configurations can be imposed (e.g. maximum two ports per memory). In this case, only the valid port configurations are retained and analyzed by our program.

The second category of constraints refers to memory locations. The significance of these constraints is the same as for constraints (1) and (2), but now related to the number of locations and signals simultaneously alive, rather than the number of ports and simultaneous signal accesses. The constraints are taking also into account that a signal of b_2 bits can be eventually stored in a memory of b_1 bits ($b_1 \geq b_2$).

$$\sum_{k=1}^{RW(k)} N_{kj} \leq N_{max}(\overline{k}, \overline{m}) \quad (4)$$

$$\sum_{i=1}^k \sum_{j=1}^{RW(i)} N_{ij} \geq N_{max}(\overline{1}, \overline{k}) \quad (5)$$

$N_{max}(\overline{m}, \overline{n})$ represents the maximum number of scalars contained in groups of signals of word-lengths b_m, \dots, b_n which are simultaneously alive. These data are provided also by the polyhedral data-flow analysis [2]. No integrality condition is imposed on the variables N_{ij} when solving the allocation phase, as the memory sizes will be adjusted further, during the assignment stage.

The allocation model presented so far represents the minimization of a non-linear cost function with linear constraints and mixed (integer and real) variables. The method employed to solve it is mainly based on the property that the constraints are naturally decoupled: some constraints are referring to ports and the others are referring to storage locations. By considering the port variables as parameters, and by assigning specific values to these, the cost function becomes linear in the N_{ij} variables (if the discontinuity for $N_{ij} = 0$ is neglected). This observation is the key idea of the proposed optimization strategy:

1. The port constraints (1), (2) and (3) determine a polytope. All the integer lattice points of this polytope represent a possible configuration of ports. The lattice points are enumerated with a Fourier-Motzkin [3] based technique.

2. In each of the lattice points generated in step 1 (for each possible port configuration), an LP optimization - having N_{ij} as variables - subject to constraints (4) is performed (slightly modified in order to comply with the discontinuities of the cost function for $N_{ij} = 0$).

The result of the allocation scheme is a set with the best k (parameter given by the designer) memory configurations, fully determined in terms of number and type of memories, number and type of ports, word-lengths, and estimated number of locations.

The computational effort mainly depends on the "size" of the "port" polytope (constraints (1), (2), (3)), as a modified LP optimization must be done in all its lattice points. In practice, the number of realistic port configurations is relatively small though not feasible to enumerate manually. We have experimented with examples leading to "port" polytopes of up to hundreds of points and the computational effort remained even then of the order of seconds (see Section 4).

3 Signal-to-memory assignment

Several binary ILP formulations have been proposed in the past to solve the signal-to-memory assignment ([1] and references). These assignment models employ as binary variables x_{ij} - which defines the assignment of the scalar i to memory j . If several memories are present, the maximum number of signals that can be assigned is limited to a few hundreds, due to limits of commercial ILP packages. This is unacceptable for realistic applications. In contrast, our assignment model considers *groups* of signals (according to our terminology [2], the *basic sets* of signals - constructed during the data-flow analysis) rather than *individual* signals. It is also not

based on expensive ILP formulations. In this way, the assignment problem for applications with large amounts of signals becomes less expensive and more tractable. The proposed algorithm operates in two phases:

1. a constructive phase (greedy), which generates an initial assignment solution;
2. an iterative improvement phase, which is basically a branch-and-bound algorithm.

The inputs of the assignment procedure are: (1) a configuration of memories with all parameters determined during the allocation phase; (2) the groups of signals to be assigned, and the constraints due to port conflicts, both resulting from the data-flow analysis [2].

The assignment decision is steered by computing a cumulative penalty composed of three weighted terms:

$$P = \lambda_1 P_1 + \lambda_2 P_2 + \lambda_3 P_3 \quad (6)$$

The first term $\lambda_1 P_1$ penalizes the memory area "lost" by assigning a group of signals to a memory having superior word-length. If A_i denotes the area of memory i , N_i - its size, and b_i - its word-length, the area per bit ratio is $A_i/N_i b_i$. If a group of n_k signals of b_k bits ($b_k < b_i$) is assigned to that memory, then $n_k(b_i - b_k)$ bits are "lost", resulting in a cost of:

$$P_1 = n_k \frac{b_i - b_k}{b_i} \frac{A_i}{N_i}$$

The second term of (6) encourages the assignment of *basic sets* common to the same definition/operand domain [2] to the same memory. Intuitively, this term tends to reduce the address logic for each memory as it is likely that these signals can then share the same address equations and thus address logic. The third term penalizes exceeding the estimated memory sizes determined during the allocation phase:

$$P_3 = \left(\text{overflow}_i \cdot \frac{A_i}{N_i} \right)^2$$

Different assignment effects can be obtained by varying the weights in (6). E.g. if $\lambda_2 \gg \lambda_1, \lambda_3$ all scalars belonging to the same M-D signals are grouped in the same memories. If $\lambda_3 \gg \lambda_1, \lambda_2$ the final allocation solution is closer to the "optimal" (in terms of estimated area) solution resulting from the allocation phase.

The initial assignment solution is constructed in a greedy way, obeying the assignment constraints and minimizing the global penalty value in (6). Afterwards, a branch-and-bound process is initiated, intended to further improve (6), while complying with the constraints. In order to improve the practical computational effort, a mechanism is used to avoid the generation of symmetric assignment solutions. Therefore, the algorithm guarantees that every valid solution is produced only once. The final solution consists of the memory configuration(s) found at the allocation phase, but having the memory sizes adjusted by the assignment. A map of the assignment solution allows to determine where each scalar is actually stored.

4 Overview of main results

The implementation of the presented approach was done in C++, under the CATHEDRAL framework. The allocation program has been tested so far on several applications listed in Table 1. The number of signals, and the word-lengths encountered in the application are

No.	Application	Signals	Sets	Cycles	Mem. structure after alloc./after assig.	Area	CPU
1	Updating singular value decomposition n=36	146590 16b	1298	720000	1M.16b/1RW N=3995 1M.16b/1RW N=3995	51.58 51.58	0.18s 0.01s
2				360000	2M.16b/1RW N=2 × 1997 2M.16b/1RW N=2089/2467	51.66 58.89	0.51s 8m05s
3				240000	3M.16b/1RW N=3 × 1331 3M.16b/1RW N=1369/1365/1333	51.74 52.67	1.56s 8m21s
4	n=32	103550	1150	170000	3M.16b/1RW N=3 × 1055 3M.16b/1RW N=1089/1089/1057	41.06 41.94	1.54s 5m38s
5	Motion detection G=H=15 M=N=4 m=n=2	100080 16b	102	120000	1M.16b/1RW N=32800 1M.16b/1RW N=32800	422.9 422.9	0.10s 0.08s
6				200000	2M.16b/1RW N=2 × 16400 2M.16b/1RW N=18016/15376	423.0 430.7	0.61s 1m38s
7	Medical image reconstruction	6784 16,12b	6401	12000	2M.16b/1R+1W N=2; 1RW N=384 2M.16b/1R+1W N=2; 1RW N=384	5.13 5.14	1.93s 1m02s

Table 1: Storage allocation results

listed in column 2. The third column of Table 1 contains the number of basic sets [2] (groups of scalars) extracted by the polyhedral data-flow analysis from the source code: this data suggest the amount of complexity reduction for the subsequent assignment phase. Column 4 indicates the clock cycle budget allocated for R/W operations. The allocation and assignment results obtained are listed in the last three columns. The memory structure indicates the word-length, the port structure, and the number of locations (before/after assignment). The total area (mm^2) is estimated according to Mulder's model [7], assuming a CMOS technology with $1.2\mu m$ minimum geometry. Because the estimated memory sizes N are adjusted at the assignment phase, the resulting memory areas are slightly higher than the values computed at the allocation step. The CPU times for allocation and assignment are measured on an HP-station 715/50. The optimization technique used for the allocation phase is very fast. The heuristic assignment, although more time expensive, is still very acceptable.

The first example represents an updating singular value decomposition algorithm – algebraic kernel used e.g. in antenna beamforming and Kalman filtering. It contains nine loop nests, up to three levels deep. The experiments have been carried out for two values of the parameter n (the matrix order) and for different cycle budgets. The second application is a full motion detection kernel for video coding. The SILAGE code, shown in Fig. 1, is non-procedural. The amount of signal instances (over 10^5) for the chosen parameter set is prohibitive for any scalar-oriented method. While for the first cycle budget (entry 5) 1RW port is sufficient, several ports are necessary in order to achieve the throughput for the second experiment (entry 6). The memory organization corresponding to the minimum estimated area (according to the layout model embedded in the program), and complying with the given throughput constraint, is indicated in column 5. The last test-vehicle is extracted from a medical back-projection vehicle for computer tomography images (entry 7). The code contains data-dependent indices which are correctly modelled. Also here, good results have been obtained, comparable to the manual design. The second memory with 2 locations is actually a small multi-port register-file.

5 Conclusions

In this paper, we have addressed the problem of background memory allocation and M-D signal assignment for RMSP systems. In order to address non-procedural functional specifications with large M-D indexed signals, the proposed methodology is driven by a polyhedral data-flow analysis. The allocation technique yields one or several distributed memory architecture(s) with fully-determined characteristics. Our future work will concentrate on extending this technique to a more general hierarchical model.

References

- [1] I.Ahmad, C.Y.R.Chen, "Post-processor for data path synthesis using multiport memories," *Proc. ICCAD'91*, pp.276-279, Santa Clara CA, Nov. 1991.
- [2] F.Balasa, F.Catthoor, H.De Man, "Exact Evaluation of Memory Area for M-D Processing Systems," *Proc. ICCAD'93*, pp.669-672, Santa Clara CA, Nov. 1993.
- [3] G.Dantzig, B.Eaves, "Fourier-Motzkin Elimination and Its Dual," *J. Combinatorial Theory*, pp.288-297, 1973.
- [4] F.Franssen, L.Nachtergaele, H.Samsom, F.Catthoor, H.De Man, "Control flow optimization for fast system simulation and storage minimization," *Proc. 5th EDAC'94*, pp.20-24, Paris, France, Mar. 1994.
- [5] G.Goossens, J.Rabaey, J.Vandewalle, H.De Man, "An efficient microcode compiler for application-specific DSP processors," *IEEE Trans. CAD*, pp.925-937, Sep. 1990.
- [6] P.Lippens *et al.*, "Allocation of multiport memories for hierarchical data streams," *Proc. ICCAD'93*, pp. 728-735, Santa Clara CA, Nov. 1993.
- [7] J.M.Mulder, N.T.Quach, M.J.Flynn, "An Area Model for On-Chip Memories and its Application," *IEEE J. Solid-state Circ.*, Vol.SC-26, pp.98-105, Feb. 1991.
- [8] L.Ramachandran, D.Gajski, V.Chaiyakul, "An algorithm for array variable clustering," *Proc. 5th EDAC'94*, pp.262-266, Paris, France, Feb. 1994.
- [9] L.Stok, J.Jess, "Foreground memory management in data path synthesis," *Int. J. on Circ. Theory and Appl.*, Vol.20, pp.235-255, 1992.
- [10] J.Vanhooft, K.Van Rompaey, I.Bolsens, G.Goossens, H.De Man, "High-level synthesis for real-time digital signal processing," Kluwer Acad. Publ., 1993.