AUTOMATIC GENERATION OF MAPS OF MEMORY ACCESSES FOR ENERGY-AWARE MEMORY MANAGEMENT

Florin Balasa* Ilie I. Luican† Hongwei Zhu‡ Doru V. Nasui§

* Dept. of Computer Science, Southern Utah University, Cedar City, UT
† Dept. of Computer Science, University of Illinois at Chicago, Chicago, IL
‡ ARM, Inc., Sunnyvale, CA
§ American Int. Radio, Inc., Rolling Meadows, IL

ABSTRACT

Many signal processing systems are synthesized to execute data-dominated applications. Their behavior is described in a high-level programming language, where the code is typically organized in sequences of loop nests and the main data structures are multidimensional arrays. Since data transfer and storage have a significant impact on both the system performance and the major cost parameters – power consumption and chip area, the designer must spend a significant effort during the system development process on the exploration of the memory subsystem in order to achieve a cost-optimized design. This paper focuses on the reduction of the dynamic energy consumption in the hierarchical memory subsystem of multidimensional signal processing systems, starting from the high-level behavioral specification of the application. The paper presents an algorithm which identifies those parts of arrays from a high-level specification that are intensely accessed (for *read* and/or *write* operations), whose storage on-chip yields the highest benefit in terms of dynamic energy consumption. Tested on a two-layer memory hierarchy (scratch-pad and off-chip memories), this algorithm led to significant savings of energy in comparison to previous computation models.

Index terms– Memory allocation, multi-layer memory subsystem, dynamic energy consumption, signal assignment to memory layers

1. INTRODUCTION

Many multidimensional signal processing systems (particularly, in the domains of multimedia and telecommunications), are synthesized to execute data-intensive applications. Since data transfer and storage have a significant impact on both the system performance and the major cost parameters – power consumption and chip area, the designer must spend a significant effort during the system development process on the exploration of the memory subsystem in order to achieve a cost-optimized design.

The memory subsystem is, typically, a major contributor to the overall energy budget of the system [5]. Savings of dynamic energy (which expands only when memory accesses occur) at the level of the whole memory subsystem can be potentially obtained by accessing frequently used data from smaller on-chip memories rather than from large background (off-chip) memories, the problem being how to optimally assign the data to the memory layers.[1] As on-chip storage, the scratch-pad memories (SPMs) – software-controlled static or dynamic random-access memories, more energy-efficient than caches – are widely used in embedded systems, in which the flexibility of caches in terms of workload adaptability is often unnecessary, whereas power consumption and cost play a much more critical role. Different from caches, the SPM occupies one distinct part of the virtual address space with the rest of the address space occupied by the main memory. The consequence is that there is no need to check for the availability of the data in the SPM. Hence, the SPM does not possess a comparator and the miss/hit acknowledging circuitry [3]. This contributes to a significant energy (as well as area) reduction. Another consequence is that in cache memory systems, the mapping of data to the cache is done during the code execution, whereas in SPM-based systems this can be done either manually by the designer, or automatically – by a compiler, using a suitable algorithm.

The energy-efficient assignment of signals to the on- and off-chip memories has been studied since the late nineties. These previous works focused on partitioning the data structures from the application code into so-called *copy candidates*, and on the optimal selection and mapping of these into the memory hierarchy [13]. Their general idea was to identify the most frequently accessed data in each loop nest. For instance, Kandemir and Choudhary analyze and exploit the temporal locality by inserting local copies [10]. Their layer assignment builds a separate hierarchy per loop nest and then combines them into a single hierarchy. However, the approach lacks a global view on the (part of) arrays lifetimes in applications having imperfect nested loops. Brockmeyer *et al.* use the steering heuristic of assigning the arrays having the lowest access number over size ratio to the cheapest memory layer first, followed by incremental reassignments [4]. Hu *et al.* can use *parts* of arrays as copies, but they typically use cuts along the array dimensions [9] (like rows and columns of matrices).

The energy-aware partitioning of an on-chip memory in mul-

[1] Note that this problem is basically different from caching for performance [8], where the question is to find how to fill the cache such that the needed data have been loaded in advance from the main memory.

```
optDelta[0] = 0 ;                          //  int A[81][81]: input;
for ( i=16; i<=64; i++ )
  for ( j=16; j<=64; j++ )
  {  Delta[i][j][0] = 0 ;
     for ( k=i-16; k<=i+16; k++ )
        for ( l=j-16; l<=j+16; l++ )
           Delta[i][j][33*k-33*i+l-j+545] = A[i][j] - A[k][l]
                          +  Delta[i][j][33*k-33*i+l-j+544] ;
     optDelta[49*i+j-799] = Delta[i][j][1089] + optDelta[49*i+j-800];
  }
opt[0] = optDelta[2401];
```

Figure 1: Illustrative example whose structure is similar to a motion detection kernel ($m = n = 16,\ M = N = 64$) [6].

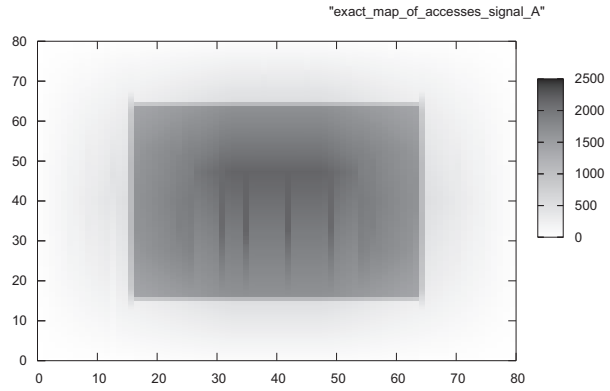

Figure 2: Exact map of memory *read* accesses (obtained by simulation) for the 2D signal A from the illustrative code in Fig. 1.

tiple banks has been studied by several research groups, as well. Techniques of an exploratory nature analyze possible partitions, matching them against the access patterns of the application [7]. Other approaches exploit the properties of the dynamic energy cost and the resulting structure of the partitioning space to come up with algorithms able to derive the optimal partition for a given access pattern [1].

Starting from the behavioral specification of a given application, where the code is organized in sequences of loop nests and the main data structures are multidimensional arrays, this paper presents an algorithm which can automatically build maps of memory accesses to the array space of signals and, consequently, allows to identify with accuracy those parts of arrays that are more intensely accessed for *read* or *write* operations. Storing on-chip these parts of arrays yields the highest reduction of the dynamic energy consumption in a hierarchical memory subsystem. The proposed computation model was tested for the time being assuming two memory layers – on-chip scratch-pad and off-chip memories – focusing on the reduction of the dynamic energy consumption due to memory accesses. Extensions of the model to the exploitation of memory banking, as well as taking also into account the leakage energy consumption, will be addressed in the future.

The rest of the paper is organized as follows. Section 2 presents the algorithm used to detect the parts of the arrays intensely accessed by memory operations during the execution of a loop-organized algorithmic specification. Section 3 discusses implementation aspects and presents experimental results. Finally, Section 4 summarizes the main conclusions of this research.

2. MAPS OF MEMORY ACCESSES FOR ENERGY-AWARE MEMORY MANAGEMENT

The algorithms describing the functionality of real-time multimedia and telecom applications are typically specified in a high-level programming language, where the code is organized in sequences of loop nests having as boundaries linear functions of the outer loop iterators. Conditional instructions are very common as well, and the multidimensional array references have (possibly complex) linear indices (the variables being the loop iterators).

Figure 1 shows an illustrative example whose structure is similar to the kernel of a motion detection algorithm[2] [6]. The problem is to automatically identify those parts of arrays from the given application code that are more intensely accessed, in order to steer their assignment to the energy-efficient data storage layer (the on-chip scratch-pad memory) such that the dynamic energy consumption in the hierarchical memory subsystem be reduced.

The number of storage accesses for each array element can certainly be computed by the simulated execution of the code. The result of such a simulation is displayed in Fig. 2, where the area represents the so-called *index space* of the 2D signal A from the illustrative code in Fig. 1. For each pair of possible indexes (between 0 and 80), the number of accesses was counted and the level of grey depends on the intensity with which the array elements are accessed (the darker the color, the higher the number of accesses). The array elements near the center of the index space are accessed with high intensity (for instance, $A[40][40]$ is accessed 2178 times; $A[16][40]$ is accessed 1650 times), whereas the array elements at the periphery are accessed with a significantly lower intensity (for instance, $A[0][40]$ is accessed 33 times and $A[0][0]$ only once).

The drawbacks of such an approach are twofold. First, the simulated execution may be computationally ineffective when the number of array elements is very significant, or when the application code contains deep loop nests. Second, even if the simulated execution were feasible, such a scalar-oriented technique would not be helpful since the addressing hardware of the data memories would result very complex. In order to obtain a reasonable memory addressing logic, it would be desirable to model the parts of the signals' index space we are mapping into physical memories as (images of) **Z**-*polyhedra* [14], rather than sets of array elements.

Our proposed computation methodology for building approximate maps of memory accesses is described below, after defining a few basic concepts.

Each *array reference* $M[x_1(i_1, \ldots, i_n)] \cdots [x_m(i_1, \ldots, i_n)]$ of an m-dimensional signal M, in the scope of a nest of n loops having the iterators $i_1, \ldots, i_n$, is characterized by an *iterator space* and an *index* (or *array*) *space*. The iterator space signifies

[2] The actual code contains also a *delay* operator that is irrelevant in our context.

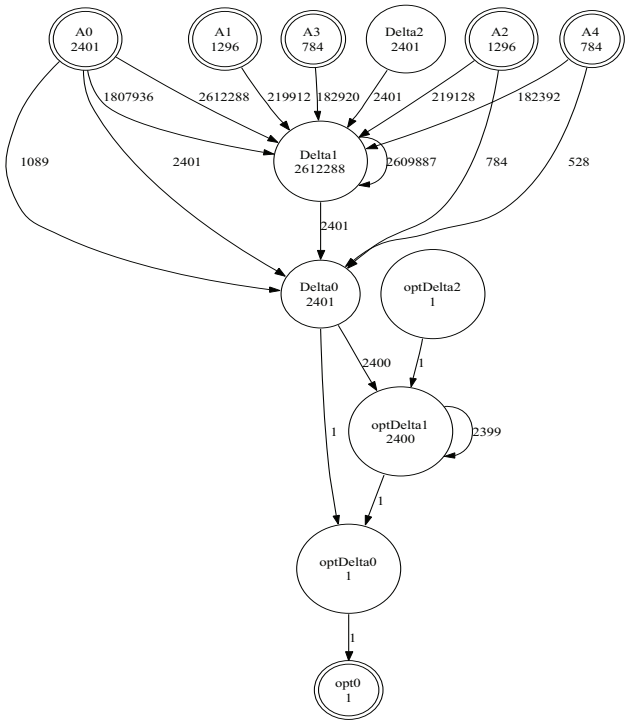

Figure 3: Graph showing the dependence relations between the lattices derived from the illustrative code in Fig. 1.

the set of all iterator vectors $\mathbf{i} = (i_1, \ldots, i_n) \in \mathbf{Z}^n$ in the scope of the array reference, and it can be typically represented by a so-called **Z**-polytope (a polyhedron bounded and closed, restricted to the set $\mathbf{Z}^n$): $\{ \mathbf{i} \in \mathbf{Z}^n \mid \mathbf{A} \cdot \mathbf{i} \geq \mathbf{b} \}$. E.g., for the array reference $A[k][l]$ from the code in Fig. 1, the iterator vector is the column vector $\mathbf{i}=[i \;\; j \;\; k \;\; l]^T$, the matrix **A** has 8 rows (and 4 columns) derived from the lower and upper boundaries of the 4 nested loops, and **b** is a column vector of 8 elements. The index space is the set of all index vectors $\mathbf{x} = (x_1, \ldots, x_m) \in \mathbf{Z}^m$ of the array reference. When the indices of an array reference are linear mappings with integer coefficients of the loop iterators, the index space consists of one or several *linearly bounded lattices*: $\{ \mathbf{x} = \mathbf{T} \cdot \mathbf{i} + \mathbf{u} \in \mathbf{Z}^m \mid \mathbf{A} \cdot \mathbf{i} \geq \mathbf{b} \, , \, \mathbf{i} \in \mathbf{Z}^n \}$. E.g., for the same array reference $A[k][l]$, $\mathbf{T}= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ and $\mathbf{u}= \begin{bmatrix} 0 \\ 0 \end{bmatrix}$.

Step 1 *Extract the array references from the given algorithmic specification and decompose the array references for every indexed signal into* disjoint *lattices.*

The motivation of the decomposition of the array references relies on the following intuitive idea: the disjoint lattices which belong to many array references are actually those parts of the arrays more heavily accessed during the code execution. This decomposition into disjoint lattices – used also in [2] – can be performed analytically, by recursively intersecting the array references of every multidimensional signal in the code.

Step 2 *Build the* polyhedral dependence graph *of the algorithmic specification.*

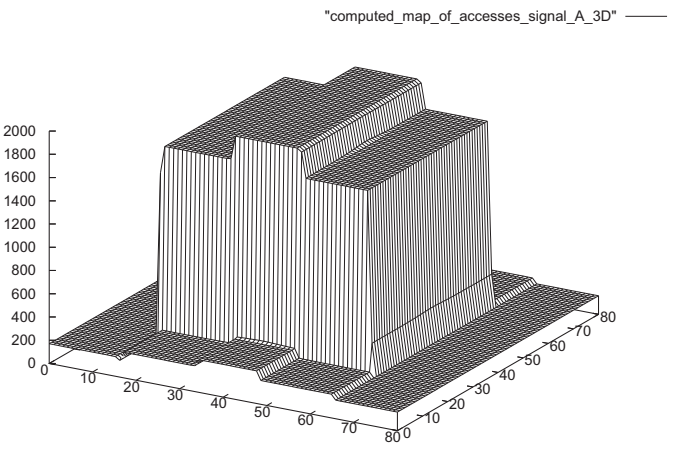

Figure 4: Computed 3D map of memory *read* accesses for the signal A from the illustrative code in Fig. 1.

Figure 3 shows the polyhedral dependence graph derived from the illustrative code in Fig. 1. The nodes in this graph represent the disjoint lattices determined at *Step 1* and the arcs are dependence relations between these lattices. The nodes are labeled with the number of array elements covered by the lattices and the arcs are labeled with the number of dependencies between the lattices.

Step 3 *Compute the average number of memory accesses for each disjoint lattice.*

Notice that the total number of dependencies of a lattice (that is, the sum of the labels of all the arcs starting from the node representing the lattice in the polyhedral graph) yields, actually, the number of *read* operations for the array elements covered by the lattice. For instance, there are 4 arcs starting from the node $A0$ (see Fig. 3) representing the lattice $A0$ covering the central part of the index space of signal A. The total number of *read* accesses to this lattice is the sum of 4 arc labels: 1,089+2,401+1,807,936+2,612,288=4,423,714. Since the number of A-elements covered by this lattice is 2,401 (see the label of node $A0$), the average number of accesses for this lattice is 1,842.45 .

Step 4 *Build the approximate map of memory accesses for the index space of each signal.*

The index space of each signal was partitioned into disjoint lattices (*Step 1*), each lattice having its own average number of memory accesses (computed at *Step 3*). Figure 4 displays such a 3D map for the signal A (see the code in Fig. 1), where A's index space is in the horizontal plane xOy and the average numbers of memory accesses are on the vertical axis Oz.

Step 5 *Select the lattices having the highest access numbers, whose total size does not exceed the maximum SPM size (assumed to be a design constraint), and map them into the SPM* [14].

3. EXPERIMENTAL RESULTS

A hierarchical memory allocation tool has been implemented in C++, incorporating the algorithm described in this paper. For the time being, the tool supports only a two-level memory hierarchy,

Application	#Array refs.	#Array elements	#Memory accesses	Mem. size	Dyn. energy 1-layer [μJ]	SPM size	Energy saved [4]	Energy saved [9]	Energy saved	CPU [sec]
Motion estim.	13	265,633	864,900	2,465	3,088	1,416	38.7%	40.7%	50.7%	23
Durbin alg.	21	252,499	1,004,993	1,249	3,588	764	55.2%	58.5%	73.2%	28
SVD updating	85	3,045,447	6,227,124	34,950	22,231	12,672	35.9%	38.4%	46.0%	37
Vocoder	236	33,619	200,000	11,890	714	3,879	30.8%	32.5%	39.5%	8
Dyn. prog.	3,992	21,082,751	83,834,000	124,751	299,287	27,316	43.3%	46.6%	56.1%	47

Table 1: Experimental results.

where an SPM is used between the main memory and the processor core. The dynamic energy is computed based on the number of accesses to each memory layer. In computing the dynamic energy consumptions for the SPM and the main (off-chip) memory, the CACTI v4.2 power model [12] is used.

Table 1 summarizes the results of our experiments, carried out on a PC with a 1.85 GHz Athlon XP processor and 512 MB memory. The benchmarks used are algebraic kernels (like Durbin's algorithm for solving Toeplitz systems) and algorithms used in multimedia applications (like, for instance, an MPEG4 motion estimation algorithm). The table displays the numbers of array references, total number of array elements, and memory accesses; the data memory size (in storage locations/bytes) and the dynamic energy consumption assuming only one (off-chip) memory layer; the SPM size and the savings of dynamic energy applying, respectively, a previous model steered by the total number of accesses for whole arrays [4], another previous model steered by the most accessed array rows/columns [9], and the current model, versus the single-layer memory scenario; the CPU times. The energy consumptions for the motion estimation benchmark were, respectively, 1894, 1832, and 1522 μJ; the saved energies relative to the energy in column 6 are displayed as percentages in columns 8-10.

Storing on-chip all the signals is, obviously, the most desirable scenario in point of view of dynamic energy consumption. We assumed that the SPM size is constrained to smaller values than the overall storage requirement. In our tests, we computed the ratio between the dynamic energy reduction and the SPM size; the value of the SPM size maximizing these ratio was selected, the idea being to obtain the maximum benefit for the smallest SPM size. The sizes of the main memory and of the SPM were evaluated *after* the mapping of the signals into the physical memories, using a mapping algorithm based on the computation of maximal bounding windows [14]. Our experiments show that the savings of dynamic energy consumption are from 40% to over 70% relative to the energy used in the case of a flat memory design. Although the previous models produce important energy savings as well, our model led to 20%-33% better savings than them.

4. CONCLUSIONS

This paper has presented a methodology for partitioning the index space of the multidimensional signals from data-dominated applications such that those array parts heavily accessed are identified and stored in scratch-pad memories in order to diminish the dynamic energy consumption due to memory accesses.

References

[1] F. Angiolini, L. Benini, and A. Caprara, "An efficient profile-based algorithm for scratchpad memory partitioning," *IEEE Trans. CAD*, vol. 24, no. 11, pp. 1660-1676, Nov. 2005.

[2] F. Balasa, H. Zhu, and I.I. Luican, "Computation of storage requirements for multi-dimensional signal processing applications," *IEEE Trans. VLSI Systems*, vol. 15, no. 4, pp. 447-460, April 2007.

[3] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad memory : A design alternative for cache on-chip memory in embedded systems," in Proc. 10th Int. Workshop on Hardware/Software Codesign, Estes Park CO, May 2002.

[4] E. Brockmeyer, M. Miranda, H. Corporaal, and F. Catthoor, "Layer assignment techniques for low energy in multi-layered memory organisations," in *Proc. ACM/IEEE Design Aut. & Test in Europe*, Munich, Germany, Mar. 2003, pp. 1070-1075.

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

[6] E. Chan and S. Panchanathan, "Motion estimation architecture for video compression," *IEEE Trans. on Consumer Electronics*, vol. 39, pp. 292-297, Aug. 1993.

[7] S. Coumeri and D.E. Thomas, "Memory modeling for system synthesis," *IEEE Trans. VLSI Syst.*, vol. 8, no. 3, pp. 327-334, 2000.

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

[9] Q. Hu, A. Vandecapelle, M. Palkovic, P.G. Kjeldsberg, E. Brockmeyer, and F. Catthoor, "Hierarchical memory size estimation for loop fusion and loop shifting in data-dominated applications," in *Proc. Asia-S. Pacific Design Automation Conf.*, Yokohama, Japan, 2006, pp. 606-611.

[10] M. Kandemir and A. Choudhary, "Compiler-directed scratch-pad memory hierarchy design and management," in *Proc. 39th ACM/IEEE Design Automation Conf.*, Las Vegas NV, June 2002, pp. 690-695.

[11] A. Schrijver, *Theory of Linear and Integer Programming*, New York: John Wiley, 1986.

[12] S. Wilton and N. Jouppi, "CACTI: An enhanced access and cycle time model," *IEEE J. Solid-State Circuits*, May 1996.

[13] S. Wuytack, J.-P. Diguet, F. Catthoor, and H. De Man, "Formalized methodology for data reuse exploration for low-power hierarchical memory mappings," *IEEE Trans. VLSI Syst.*, vol. 6, no. 4, pp. 529-537, Dec. 1998.

[14] H. Zhu, I.I. Luican, and F. Balasa, "Mapping multi-dimensional signals into hierarchical memory organizations," *Proc. ACM/IEEE Design Aut. & Test in Europe*, Nice, France, April 2007, pp. 385-390.