Mesh Partitioning Approach to Energy Efficient Data Layout

Sambuddhi Hettiaratchi Peter Y.K. Cheung

Department of Electrical and Electronic Engineering
Imperial College of Science, Technology and Medicine, London
E-mail: s.hetti@ic.ac.uk, p.cheung@ic.ac.uk

Abstract

Memory access consumes a significant amount of energy in data transfer intensive applications. The selection of a memory location from a CMOS memory cell array involves driving row and column select lines. A switching event on a row select line often consumes significantly more energy in comparison to a switching event on a column select line. In order to exploit this difference in energy consumption of row and column select lines, we propose a novel data layout method that aims to minimize row switching activity by assigning spatially and temporally local data items to the same row. The problem of minimum row switching data layout has been formulated as a multi-way mesh partitioning problem. The constraints imposed on the problem formulation ensure that the complexity of the address generator required to implement the optimized data layout is bounded and that the data layout optimization can be applied to all address generator synthesis methods. Our experiments demonstrate that our method can significantly reduce row transition counts over row major data layout.

1. Introduction

Memory access consumes a significant amount of energy in data transfer intensive applications [11], such as video and image processing. Dynamic power dissipation is significant in CMOS circuits, and therefore, behavioural level energy minimization efforts often attempt to minimize signal transition counts, particularly on high capacitance nodes [8].

In order to minimize switching activity caused by memory access, it is necessary to have some knowledge about the access sequences. For many application-specific integrated circuits (ASICs), the access sequences are usually known *a priori*. ASICs may also contain data dependent access sequences. However, because the application is known, statistical information can be collected about the pattern of access. Information about the access sequences enable the application of energy optimizations to ASICs which would not be applicable to general purpose systems.

CMOS memory cell arrays are usually organized into rectangular blocks of memory cells. The selection of a memory location involves driving row, column and, in large memories, block select signals. Signal transitions on high capacitance select lines such as the row and block select lines consume more energy in comparison to those on column select lines [4].

In this paper, we present a mesh partitioning approach to minimizing the energy consumption of memory access through data layout that minimizes row switching. A multi-way graph partitioning approach to this problem has been recently proposed [5]. The graph partitioning problem formulation does not impose a structure on the optimized data layout. Therefore, it is general and achieves good results over a broad spectrum of access patterns. However, due to the lack of structure of the optimized data layout, multi-way graph partitioning approach suffers from two drawbacks. One drawback is that it does not limit the complexity of the resulting address generator. The other is that graph partitioning approach cannot easily be used with address generator synthesis methods which require address equations expressed in terms of application specific units (ASUs) [7] such as adders, multipliers and multiplexors. Mesh partitioning problem formulation reported here imposes additional constraints on the graph partitioning approach so that the complexity of the address generator is limited and row minimization optimization can more easily be applied to ASU type address generators.

Memory hierarchies that exploit data reuse can be used to minimize memory access energy [12]. Data layout optimizations are a complementary memory access energy reduction technique that can be applied together with other optimizations such as memory hierarchy. In this paper we also examine the interaction of data layout methods with simple memory hierarchies.

The novel contributions of this paper are: (1) formulation of the minimum row switching data layout problem as a mesh partitioning problem, (2) evaluation of the quality

1530-1591/03 $17.00 © 2003 IEEE

of the solution, and (3) study of the interaction of memory hierarchy with data layout methods.

This paper is organized as follows. Section 2 presents some of the previous work in the area of memory access energy minimization and data layout. Section 3 formulates the minimum row switching problem as a mesh partitioning problem. Section 4 describes our mesh partitioning data layout method. Section 5 introduces the interaction of memory hierarchy with mesh partitioning data layout. Section 6 reports and discusses experimental results and Section 7 contains conclusions and indicates some future work.

2. Previous Work

The relevant previous work address the problem of reducing memory access energy in ASICs at the behavioural level. Research on data layout are also briefly discussed.

In-place mapping attempts to reduce required memory size by sharing physical memory locations amongst variables whose lifetimes do not overlap [10]. Smaller memories consume less energy per access compared to larger memories [2]. Therefore, memory size minimization techniques such as in-place mapping usually reduce energy consumption. Minimizing the number of memory accesses through, for instance loop transformations [11], can also result in lower energy consumption.

Most modern memory architectures are based on memory hierarchy [2]. Memory hierarchies consist of several layers of memories. The lower layers consume less energy per access than higher layers. Although, extra transfer are introduced to copy data from higher layers to lower layers, if there is sufficient temporal locality, energy is saved due to the decrease in the number of accesses to higher layers [12].

For a given number of memory accesses energy can be further minimized by address assignment that attempts to minimize signal transition counts, particularly high energy transitions on off-chip address busses [8].

Multimedia applications such as image and video processing often organize data variables into d-dimensional arrays, which are usually accessed through nested loops. d-dimensional data array variables can be laid out in memory in various ways. Row major and column major layouts are two linear mappings which are commonly used. However, non-linear mappings such as tile based mappings are shown to, for example, improve cache reuse [3]. Tile based mapping has also been applied to ASICs to minimize high capacitance off-chip memory address bus activity [8].

Graph based techniques, such as the graph clustering data layout technique reported in [9] for minimizing cache conflicts, are traditionally only applied to *scalar* variables. In [5] a graph partitioning data layout technique has been applied to *array* data variables to exploit the fact that different select lines in a memory cell array consume different amounts of energy. However, as pointed out in Section 1, the technique reported in [5] has some drawbacks.

Our work reported in this paper extends the work in [5], so that the row switching minimization optimization proposed in [5] can be applied to any address generator architecture and that the complexity of the address generator required to implement the optimized data layout is bounded. We also report results of a study on the interaction of data layout techniques with memory hierarchy.

3. Problem Definition

The problem of minimizing activity on the row select lines can be thought of as a problem of clustering the accessed data items such that the number of transitions between the clusters is minimized. The cluster size is no greater than the number of memory columns and the number of clusters is equal to the number of memory rows. We now define mathematically the minimum row switching data layout problem as a mesh partitioning problem where each point of the mesh corresponds to an array data variable, and a weighted edge between two points indicates the number of transitions between a pair of points.

Given an undirected edge-weighted mesh $G(V,E)$, where vertex set $V = \{v_i | i = 0,1,\ldots,n-1\}$ is the set of vertices, E is the set of weighted edges and positive integers p,q,m, and n where $p \times q \geq |V|$ and $m \times n = q$, find p subsets $V_0, V_1, \ldots, V_{p-1}$ of V such that:

1. $\cup_{i=0}^{p-1} V_i = V$ and $V_i \cap V_j = \emptyset$ for $i \neq j$
2. $|V_i| \leq q$ for $i = 0,1,\ldots,p-1$
3. Partitions (subsets) form an $m \times n$ rectangle of points on the mesh surface.
4. The *cut size*, i.e., the sum of weights of edges crossing between subsets, is minimized.

4. Mesh Partitioning Data Layout Method

4.1 Assumptions

Assumptions 1- 6 are necessary for the problem formulation and its solution. It is important to note that Assumptions 7- 11 are made to simplify the experiments and presentation of the paper and can be relaxed. These assumptions hold throughout this paper unless stated otherwise.

1. A signal transition on a row select line consumes more energy than a signal transition on a column select line.
2. Reducing switching activity on high capacitance signals such as row select lines reduces energy consumption.

```
for(i=1;i<S;i++)
  for(j=1;j<S;j++) {
    pred = 2*a[i-1][j-1] + a[i-1][j] + a[i][j-1];
    a[i][j] = a[i][j] - pred; }
```

Figure 1. Code segment from "compress" benchmark

3. Every individual data item has been mapped to a location in a physical memory but is not bound to a physical memory address.
4. There is no preference when switching from one row to another in terms of energy consumption.
5. Address generators have not yet been synthesized.
6. Memory cell arrays are rectangular. p and q represent the number of rows and the number of columns respectively. w represents the width of a memory word. p and q are known and $p \times q$ gives the total number of locations of that cell array. $p \times q \times w$ gives the total capacity of the cell array in bits.
7. Memory arrays have one read/write port.
8. The data access sequences are known.
9. Memory cells are accessed by activating row and column select lines. We ignore block select lines.
10. Each data array is assigned to a separate memory cell array.
11. Data arrays are two dimensional.

4.2. Input

Input sequences contain symbolic addresses. A symbolic address identifies a unique location within the memory cell array but is not bound to any physical address. Symbolic addresses also contain information about the array indices of the data variables they represent. A sequence of these symbolic addresses specify the access sequence (reads and writes) to a single port of a memory cell array. The access sequences can be extracted from an executable specification , such as the one shown in Figure 1, or from a control/data flow graph (CDFG) of the system.

4.3. Transition mesh

A d-dimensional data array can be described as an *integer lattice* of points in d-dimensional Euclidean space. Here points of the lattice correspond to array data variables and integer coordinates are array indices. An integer lattice is often referred to as a *mesh* and so from here on we will refer to it as such[1]. The input symbolic address sequence contains information regarding transitions between symbolic addresses. The mesh can be augmented with the transition information by adding weighted edges between points. We call this mesh which contains both the indices and the access information of the array as the *transition mesh* for that array. Figure 2(a) shows the transition mesh for the array $a[y][x]$ in the "compress" benchmark. The numbers on the edges indicate the number of transitions between data variables.

4.4. Mesh partitioning

In mesh partitioning problem formulation the shape and the size of the partitions is constant throughout the mesh. One of the most regular ways of partitioning a two-dimensional surface is to divide it up into rectangles. Hence, the partitions are restricted to be of rectangular shape. Each partition should fit in a memory row and therefore the size of the rectangle is fixed to q. So if $q = 32$, we have 6 rectangular shapes to choose the one that produces the minimal row transition count. If there are peripheral rectangles which contain less than q mesh points, they are laid out in row major or column major format. Note that the number of search points is independent of the data array size. The selection of the optimal rectangular shape is done with exhaustive search, as the search space is very small and increases very slowly with increasing number of memory columns.

For illustration purposes let us suppose that the array $a[y][x]$ is of size 4×4 and is required to be mapped to a memory with four columns (i.e., $q = 4$). The following rectangular shapes are candidates: 1×4, 2×2 and 4×1. The three corresponding mesh partitionings are shown in Figure 2 (b)-(d). Rectangle 1×4 produces a row transition count (RTC) of 29 and rectangles 2×2 and 4×1 produce RTCs of only 15. The rectangle 4×1 would be chosen in preference to the rectangle 2×2 as the former corresponds to row major mapping and hence potentially simpler address generation.

Once the optimal rectangle size has been chosen we still do not have a mathematical layout function that maps data variables from the array index space to the physical address space. There are a number of ways to layout the data variables within the rectangles and the rectangles themselves. As we aim to keep the layout function relatively simple we restrict ourselves to row major and column major schemes for data variables within the rectangles as well as for rectangles themselves. This restriction produces four layout candidates. This type of layout has been variously referred to

[1] http://mathworld.wolfram.com/topics/DiscreteMathematics.html

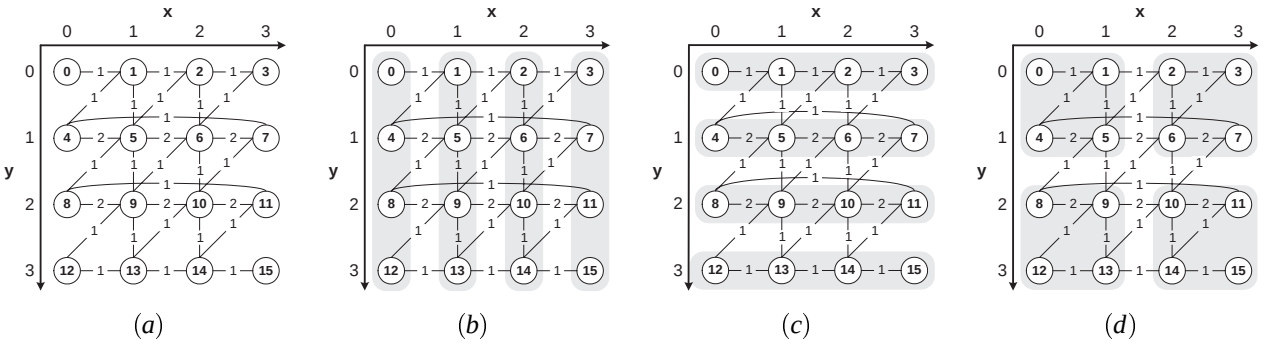

Figure 2. (a) Transition mesh for array a[y][x] in "compress" benchmark. Mesh partitions with a rectangle size of (b) 1×4**, (c)** 4×1**, and (d)** 2×2**.**

as 4D [3] and tile based [8] in the literature.

When the rectangle size equals the number of columns, as is the case here, the four tile based layout candidates all produce the same number of row transitions. Therefore, the data layout optimization needs not decide on the exact tile based scheme. That decision can be left to address generator synthesis.

4.5. Output

The output of the method is a choice of four mathematical mapping functions from the data array index space to the physical address space. Equation 1 defines one such mapping function and represents the case when rectangles are mapped row major and elements within those rectangle are mapped column major:

$$f_{rc}(y,x) = yK + xn - (K-1)(y \bmod n) \qquad (1)$$

Data arrays are of size $K \times K$ and rectangles are of size $m \times n$. Data array indices are represented by y and x. The equation can be used as is shown for the special case when m and n are integer multiples of K. When this is not the case incomplete peripheral rectangles are row major (or column major) mapped.

The address generator synthesis can select which one of the four mapping functions to use. Address generator implementation and optimization for 4D [3] or tile based mapping [8] has been reported in the literature.

5 Interaction of Data Layout with Memory Hierarchy

The array $a[y][x]$ in Figure 1 is accessed through a two level nested loop structure. The access pattern and the data reuse can be described in several ways. We will describe the access pattern by defining a basic shape of access on the index space. In the inner most loop of the "compress" code segment, 5 accesses are made to $a[y][x]$ and the basic shape for these 5 accesses is a 2×2 square on the array index space. The basic shape moves horizontally to the right by one unit for every iteration of the inner loop. The basic shape moves down vertically by one for every iteration of the outer loop. There is significant data reuse in this access pattern when $a[y][x]$ is traversed horizontally as well as vertically. The temporal locality in the horizontal direction is higher than in the vertical direction because the time distance between consecutive accesses to the same data variables is much shorter in the horizontal direction. Of the four data variables accessed by each successive iteration of the inner loop two have just been accessed in the previous iteration. These two data variables can be kept in two registers for use in the following iteration rather than to be fetched again from the memory (The assumption is keeping them in two registers consumes less energy than fetching them from a memory cell array.).

The introduction of this simple memory hierarchy alters the access sequence to the memory which stores the data array $a[y][x]$. Figures 3 (b) and 3 (a) show the transition meshes for $a[y][x]$ access sequence with and without the memory hierarchy respectively. The impact of this transformation of the transition graph/mesh, as a result of memory hierarchy, on data layout schemes will become apparent in Section 6.2.

6 Experimental Results

6.1 Mesh Partitioning Data Layout

We performed experiments on several memory access sequences to evaluate our mesh partitioning data layout method. The total number of row transitions caused by the memory accesses was used as an energy consumption metric. A row transition in a larger memory may consume more

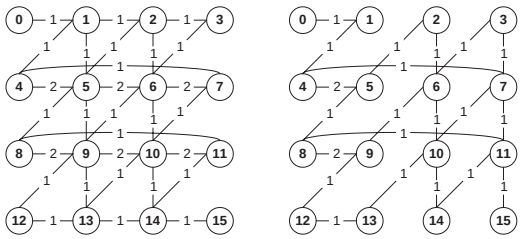

Figure 3. Transition meshes for the memory containing the array a[y][x] in "compress" (a) without (b) with memory hierarchy.

Example	Avg. % reduc. over row major	Avg. % reduc. over graph part.
Compress	58.10	4.63
Conv	19.45	4.17
DCT	74.79	0.02
GSR	43.56	4.66
Lowpass	45.78	4.46
SOR	46.36	-7.01

Table 1. Average reductions in row transition count for mesh partitioning data layout (q=32).

Example	Avg. % reduc. in RTC				
	row major	graph part.		mesh part.	
	mh	none	mh	none	mh
Compress	3.1	56.3	79.0	58.1	74.9
Conv	8.3	15.9	82.0	19.4	77.0
DCT	0.0	74.8	74.8	74.8	74.8
SOR	0.0	59.9	67.0	46.4	46.4

Table 2. Reduction in row transition count (RTC) over row major data layout and no memory hierarchy(q=32 and mh = with memory hierarchy).

energy than a row transition in a smaller memory. However, in our experiments row transition counts are not weighted to take account of the memory size.

Memory access sequences used for our experiments were obtained from the following examples: Compress, Gauss-Seidel formula (GSR), Lowpass, Successive Over Relaxation (SOR)[8] and two dimensional Convolution (Conv), separable Discrete Cosine Transform (DCT) [1]. Some of the examples contain several data arrays, exhibiting different access sequences. From such examples we have manually selected the data array with the most number of accesses for our experiments. For simplicity, two dimensional data arrays in our examples are $K \times K$ square arrays. We varied the data array dimension K from 10 to 1000 at intervals of 10 (16 to 1000 at intervals of 8 for DCT) and studied the effects on the row transition count when the arrays are mapped with different mapping schemes to memory cell arrays with different numbers of columns.

Table 1 shows the average percentage reductions in row switching achieved through mesh partitioning data layout over row major and graph partitioning [5] layouts when the data arrays are mapped to a memory with 32 columns ($q = 32$). Average reductions of 19-74% are achievable over row major mapping. The average reduction over graph partitioning data layout vary between -7% to 5%. The structure of the solution imposed by the mesh partitioning data layout works quite well on most of our examples which exhibit rectangular basic shapes. SOR access pattern has a plus-shaped basic shape, therefore graph partitioning data layout performs better for this example.

6.2. Interaction of Data Layout with Memory Hierarchy

We performed experiments on several memory access sequences to investigate the impact of memory hierarchy on several data layout methods. For each of the access sequences, we manually introduced a simple register level memory hierarchy layer between the computation units and the memory, and compared the resulting RTCs to the original RTCs.

The second column of Table 2 shows the average reduction in RTC as a result of the introduction of memory hierarchy when arrays are row major mapped to memories with 32 columns. The second column of Table 3 shows the same information when memories containing the data have 34 columns. One can observe from these results that memory hierarchy alone can reduce RTC significantly depending on the access pattern (e.g. DCT) and the memory configuration (e.g. q=34) for row major data layout.

Columns 3 to 6 in Table 2 and Table 3 record the percentage reductions in RTC over row major layout and no memory hierarchy, for graph partitioning and mesh partitioning data layouts, in the presence and absence of memory hierarchy. As a general rule transition graphs/meshes which have low connectivity (few edges) and low total number of transitions (low total edge weights) give low RTCs when partitioned. Memory hierarchy usually removes edges from the transition graph/mesh and thereby make the graph/mesh more disjoint. A graph/mesh partitioning heuristic will usually find better (more disjoint) partitions on a graph/mesh which is itself more disjoint.

Example	Avg. % reduc. in RTC				
	row major	graph part.		mesh part.	
	mh	none	mh	none	mh
Compress	2.9	57.6	79.6	44.1	67.5
Conv	8.0	16.6	82.6	20.3	77.1
DCT	68.1	28.8	79.6	-3.0	4.2
SOR	0.0	51.4	67.9	26.0	29.8

Table 3. Same as Table 2 but for q=34 (mh = with memory hierarchy).

The DCT access sequence shows no improvement in RTC when memory hierarchy optimizations are applied together with graph partitioning data layout when q=32. However, when q=34 the improvement is as significant as that for the "Conv" access sequence. DCT naturally has a lowly connected graph. However, the introduction of the memory hierarchy reduces the total number of transition by a factor of about 8. Given a certain access pattern there will be certain sizes of partitions for which it will be necessary to cut edges with large weights. In these cases simply reducing the total number of transitions can reduce RTC.

Mesh partitioning data layout shows similar behaviour to graph partitioning data layout for DCT when $q = 32$, "compress" and "Conv". However they differ significantly for DCT when $q = 34$ and SOR. Unlike, graph partitioning data layout, mesh partitioning data layout has fixed boundary vertices. Therefore, mesh partitioning data layout will only benefit from memory hierarchy if there is a reduction in the number of neighbours and transitions into and out of these vertices.

7. Conclusion and Future Work

In this paper we have presented a new approach for energy efficient data layout through minimization of memory row switching. Row transition counts for many commonly found access sequences in multimedia applications can be significantly reduced over row major mapping with our method.

The mesh partitioning data layout method is directly applicable ASU type address generator synthesis techniques [7] as well as those that require an expanded address sequence [6]. Energy consumption in the address generators was not considered in this work.

Furthermore, the method presented can be extended to data dependent memory access sequences through the use of statistical methods for construction of the transition mesh. Our method can be easily extended to more complex memory organizations [4] that use block select lines in addition to row and column select lines by first assigning data variables to blocks and then to rows.

Our experimental results also show that memory hierarchies usually improve the effectiveness of both graph and mesh partitioning data layout methods at minimizing row transition counts over simple layouts such as row major.

Acknowledgement

This work was funded by LSI Logic, USA and Overseas Research Students Awards Scheme, UK.

References

[1] P. Baglietto, M. Maresca, M. Migliardi, and N.Zingirian. Image processing on high-performance RISC systems. *Proceedings of the IEEE*, 84:917–930, July 1996.

[2] L. Benini and G. de Micheli. System-level power optimization: Techniques and tools. *ACM Trans. on Design Automation of Electronic Systems*, 5(2):115–192, Apr. 2000.

[3] S. Chatterjee, V. V. Jain, A. R. Lebeck, S. Mundhra, and M. Thottethodi. Nonlinear array layouts for hierarchical memory systems. In *Int. Conf. on Supercomputing*, pages 444 – 453, 1999.

[4] R. J. Evans and P. D. Franzon. Energy consumption modeling and optimization for SRAM's. *IEEE J. of Solid-State Circuits*, 30:571 – 579, May 1995.

[5] S. Hettiaratchi, P. Y. Cheung, and T. J. Clarke. Energy efficient address assignment through minimized memory row switching. In *Int. Conf. on Computer Aided Design*, pages 577–581, Nov. 2002.

[6] S. Hettiaratchi, P. Y. Cheung, and T. J. Clarke. Performance-area trade-off of address generators for address decoder-decoupled memory. In *Design Automation and Test in Europe*, pages 902–908, Mar. 2002.

[7] M. Miranda, F. Catthoor, M. Janssen, and H. D. Man. High-level address optimization and synthesis techniques for data-transfer-intensive applications. *IEEE Trans. on VLSI Systems*, 6(4):677 – 686, Dec. 1998.

[8] P. R. Panda and N. Dutt. Low power memory mapping through reducing address bus activity. Technical Report 95-32, University of California, Irvine, Nov. 1995.

[9] P. R. Panda, N. D. Dutt, and A. Nicolau. Memory data organization for improved cache performance in embedded processor applications. *ACM Trans. on Design Automation of Electronic Systems*, 2:384 – 409, Oct. 1997.

[10] I. Verbauwhede, F. Catthoor, J. Vandewalle, and H. D. Man. Background memory management for the synthesis of algebraic algorithms on multi-processor DSP chips. In *Int. Conf. on VLSI*, pages 209–218, Aug. 1989.

[11] S. Wuytack, F. Catthoor, F. Franssen, , L. Nachtergaele, and H. D. Man. Global communication and memory optimizing transformations for low power systems. In *IEEE Int. Workshop on Low Power Design*, pages 203–208, Apr. 1994.

[12] S. Wuytack, Jean-Philippe, F. V. Catthoor, and H. J. D. Man. Formalized methodology for data reuse exploration for low-power hierarchical memory mappings. *IEEE Trans. on VLSI Systems*, 6(4):529–537, Dec. 1998.