Memory Exploration for Low Power, Embedded Systems

Wen-Tsong Shiue
Arizona State University
Department of Electrical Engineering
Tempe, AZ 85287-5706
Ph: 1(602) 965-1319, Fax: 1(602) 965-8325
shiue@imap3.asu.edu

Chaitali Chakrabarti
Arizona State University
Department of Electrical Engineering
Tempe, AZ 85287-5706
Ph: 1(602) 965-9516, Fax: 1(602) 965-8325
chaitali@asu.edu

ABSTRACT
In embedded system design, the designer has to choose an on-chip memory configuration that is suitable for a specific application. To aid in this design choice, we present a memory exploration strategy based on three performance metrics, namely, cache size, the number of processor cycles and the energy consumption. We show how the performance is affected by cache parameters such as cache size, line size, set associativity and tiling, and the off-chip data organization. We show the importance of including energy in the performance metrics, since an increase in the cache line size, cache size, tiling and set associativity reduces the number of cycles but does not necessarily reduce the energy consumption. These performance metrics help us find the minimum energy cache configuration if time is the hard constraint, or the minimum time cache configuration if energy is the hard constraint.

Keywords
Design automation, Low power design, Memory hierarchy, Low power embedded systems, Memory exploration and optimization, Cache simulator, Off-chip data assignment.

1. INTRODUCTION
The increase in the level of abstraction of modern VLSI design is accompanied by a corresponding increase in the complexity of the building blocks that constitute the design library. Modern design libraries frequently consist of predesigned mega-cells such as embedded microprocessor cores and memories. An important feature of embedded processor-based design is that the processor core is decoupled from the on-chip memory, and thus the system designer has to choose an on-chip memory configuration that is suitable for a specific application. To aid in this design choice, a memory exploration strategy is clearly needed. In this paper, we present an exploration strategy for determining an efficient on-chip data memory architecture based on three performance metrics, namely, cache size, the number of processor cycles and the energy consumption. We add energy to the traditional performance metrics, since for low power applications, energy plays a decisive role. We choose the line size, the number of lines, tiling and the degree of set associativity such that the performance requirements are met. We focus only on the data cache. This is because for many embedded software applications, the volume of data being processed far exceeds the number of instructions.

Memory optimization for embedded system has been addressed by Panda, Dutt, and Nicolau [1,2]. The performance metrics of their system are data cache size and number of processor cycles. In addition, they propose an excellent method for off-chip data placement such that the number of conflict misses is reduced. We extend the work of [1,2] to include energy consumption as one of the performance metrics. We also consider the impact of tiling, and the degree of set associativity on the memory performance. Our memory exploration algorithm can be summarized as follows.

Algorithm MemExplore

```
for on-chip memory size,M (in powers of 2)
    for cache size, T (in powers of 2, < M)
        for line size, L (in powers of 2, < T)
            for set associativity, S (in powers of 2,≤8)
                for tiling size, B (in powers of 2, ≤ T/L)
                    Estimate Memory Performance (number
                    of cycles C, and energy consumption E).
    Select (T, L, S, B) that maximizes performance.
```

Here for each candidate cache size T (that is a power of two), for different cache line sizes, tiling sizes and the degree of set associativity, we estimate the number of processor cycles and energy consumption. We then select the cache configuration that best matches the system requirements (time bound, energy bound, time and energy bound). We have applied our procedure to several benchmark examples including Compress, Matrix multiplication, PDE, SOR [9] and Dequant [1]. Our analysis shows that the largest performance enhancement is obtained by organizing the data off-chip in a way to reduce the number of conflict misses. We also show that for memory exploration in low power systems, it is not sufficient to only consider the cache size and miss rate. This is because while the miss rate reduces with increase in cache size, the energy consumption does not always reduce. Thus it is important to study the tradeoffs between size, time and energy. The exploration procedure described here for data caches can be extended to instruction caches by merging the method of Kirovski et al [8] with ours.

2. PERFORMANCE METRICS
In this section we describe the three performance metrics of our

system, namely, cache size, number of processor cycles and energy consumption.

2.1 Cache Size

Choosing a cache size involves balancing the conflicting requirements of area and miss rate. If the cache line size is increased, then the miss rate is reduced. If the number of cache lines is increased, then the miss rate can be reduced if the tiling size or the degree of set associativity is also increased. Since there is a limit to the cache size, performance tradeoffs have to be investigated.

2.2 Number of Processor Cycles

The number of processor cycles that would be required to execute the code is a function of the miss rate. We adopt the model used in [10] and assume that the number of cycles per hit is 1, 1.1, 1.12, and 1.14 for 1, 2, 4, and 8-way set associative cache respectively. We also assume that the number of cycles per miss is 40, 40, 42, 44, 48, 56, and 72 for line sizes of 4, 8, 16, 32, 64, 128, and 256 respectively. Since increasing the line size reduces the miss rate while increasing the miss penalty, greater associativity can come at the cost of increased hit time. The number of processor cycles is shown as follows.

*Number of processor cycles = hit_rate*trip_count*(number of cycles per hit) + miss_rate*trip_count*(tiling size + number of cycles per miss).*

2.3 Energy Consumption

There exist several energy models for caches. Kamble and Ghose [3] have developed an analytic model for power consumption in various cache structures. Their model combines memory traffic, process features such as capacitance, and architectural factors including cache line size, set associativity, and capacity. The process models are based on measurements reported by Wilton and Jouppi [4] for a 0.8 μm process technology. Su and Despain [5] have developed an intuitive and simple model for power consumption due to hits in a cache. Hicks, Walnock and Owens [6] extended this model (albeit erroneously) and considered the energy consumption due to cache misses as well. We have rectified the model in [6] in this paper.

In our model, the total energy is given by *Energy= hit_rate*Energy_hit + miss_rate*Energy_miss*, where the Energy_hit is the sum of the energy in the decoder and the energy in the cell arrays, and Energy_miss is the sum of Energy_hit and the energy required to access data in main memory. We consider only energy due to READ (READ HIT and READ MISS) because reads dominate processor cache accesses [10]. Our energy model is simple and is based on those cache components that dominate overall cache power consumption. For instance, in the address decoding path, the capacitance of the decoding logic is less than that of the address bus, and so we consider only the energy consumption of the address buses, E_dec. Similarly, in cell arrays, we consider E_cell to be the energy consumed by the pre-charged cache word/bit lines. During a miss, the dominant components are the energy consumed in the I/O path of the host processor, E_io and the energy consumed during data access from the main memory E_main. We consider E_io to be the energy consumed in the address and data I/O pads [5]. For most of our experiments, we have used the SRAM CY7C1326-133 from Cypress as our main memory. The SRAM is of size 2M bits, has an access time of 4 ns, voltage of 3.3V, current of 375 mA, and has an energy consumption of 4.95 nJ per access.

In summary,

- ***Energy = Hit rate * Energy_hit + Miss rate * Energy_miss***
  Energy_hit = E_dec+E_cell
  Energy_miss = E_dec+E_cell+E_io+E_main
  = Energy_hit + E_io+E_main
  - *E_dec = α*(Add_bs)*
  - *E_cell = β*(Word_line_size) * (Bit_line_size)*
  - *E_io = γ*(Data_bs* Cache_line_size + Add_bs)*
  - *E_main = γ*(Data_bs* Cache_line_size) + Em * Cache_line_size*

  where
  Add_bs = Number of bit switches on address bus per instruction.
  Data_bs = Number of bit switches on data bus per instruction
  Word_line_size = Number of memory cells in a word line.
  Bit_line_size = Number of memory cells in a bit line.
  Em = Energy consumption of a main memory access
  α=0.001, β=2 and γ=20 are used for 0.8 μm CMOS technology

The above energy model can be used for set associative caches as well. Even though the set associative cache consumes more power in the control logic, tag comparators and address comparators, the amount is not significant [3] and hence can be ignored in our simplified energy model. Finally, in the computation of address bus switching, we have assumed Gray code encoding of the address lines, and for data bus switching, we have assumed a value of 0.25.

3. FINDING THE MINIMUM CACHE SIZE

In this section we describe a procedure for computing the minimum cache size that is required to avoid cache conflicts. The cache conflicts occur when data that could possibly be reused in the near future, is mapped to the same cache line by subsequent accesses in a limited associativity cache.

Two references, a[f(i)] and a[g(i)] are said to be *uniformly* generated if f(i)=Hi+cf and g(i)=Hi+cg, where H is a linear transformation and cf and cg are constant vectors [9]. We partition references in a loop nest into equivalent *classes* of reference or equivalent *cases* of reference. References belong to the same class if they have the same H and operate on the same arrays as described in [9]. Here we introduce the notion of case. We say that the references belong to the same case if they have the same H but belong to different arrays. For each class or case, we find the value of the (distance) mod (cache line size), where distance= floor(abs(difference of constant vector/stride of loop)+1. If the value of the (distance) mod (cache line size) is zero or one, then the number of cache lines is equal to floor(distance/cache line size)+1; otherwise, the number of cache lines is equal to floor(distance/cache line size)+2. The total number of cache lines is the sum of the number of cache lines for each class or case. We explain this procedure with the help of Example Compress.

<u>Example 1. Compress</u>

```
int a[32,32]
for  i=1,31
   for j=1,31
      a[i.j]=a[i,j]-a[i-1,j]-a[i,j-1]-2*a[i-1,j-1];
```

In this example, there are two equivalent classes. Class 1: a[i-1,j-1], a[i-1,j] and class 2: a[i,j-1], a[i,j]. The total number of cache lines is 4 (two cache lines for references in class 1 and two cache lines for references in class 2). The minimum cache size is 4*L, where L is the line size. If the given cache size is larger than the minimum cache size, then the cache line size can be increased to exploit the spatial locality or the number of cache lines can be increased in proportion to the number of classes or cases. While the miss rate reduces with increase in cache size and line size, the energy consumption may or may not reduce. To illustrate the differences in the energy consumption trends, we consider two extremes values of Em, (the energy due to main memory access). On one end of the spectrum is a Cypress 2Mbit SRAM with Em=2.31nJ and on the other end of the spectrum is a 16Mbit SRAM with Em=43.56 nJ. While the energy consumption values reduce with increase in cache size and line size for Em = 43.56 nJ, the energy consumption values increase with increase in cache size and line size for Em=2.31 nJ. Figure 1 shows this variation in energy consumption for Em=43.56nJ and Em=2.31nJ for the example Compress. Figure 2 describes the variation in the miss rate, number of cycles and energy for different cache and line sizes for the benchmark examples (Compress, Matrix Multiplication, PDE, SOR and Dequant). In all these examples, the iteration space is 31*31. Figure 3 plots the variation in the number of cycles for the example Compress. Figure 4 shows the variation in energy consumption for Em =4.95 nJ (corresponding to SRAM CY7C1326-133) for different cache and line sizes for the example Compress.

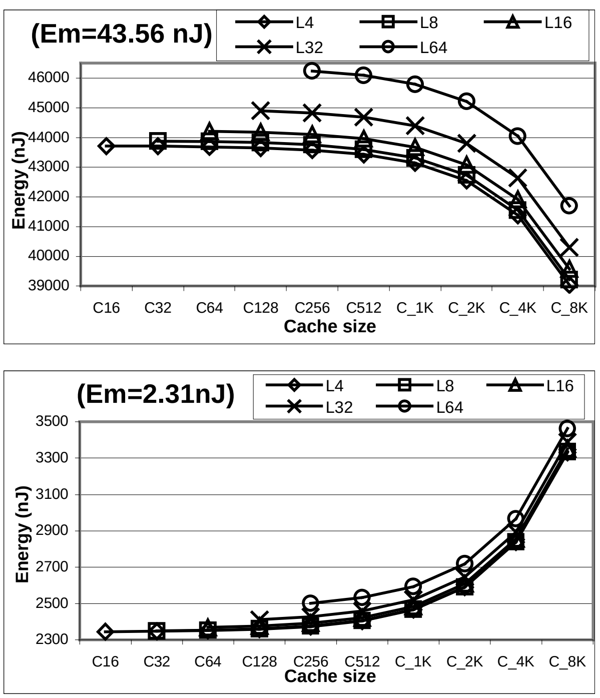
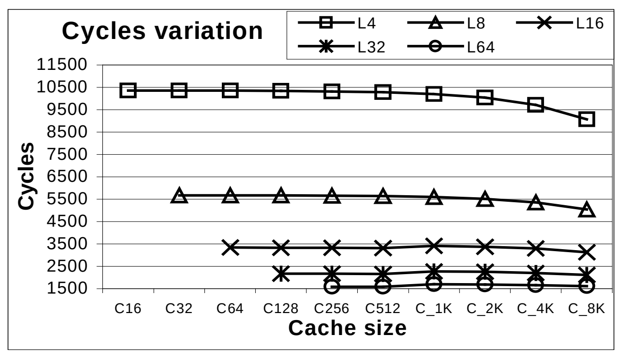


Figure 1. Example Compress. Variation in energy for different cache sizes and line sizes. (Em = 43.56 nJ and Em = 2.31 nJ).

The study of the variation in the number of cycles and energy for different cache sizes and line sizes can be used to obtain the best cache configuration given the energy and/or time bounds. For instance, the minimum energy cache configuration for the example Compress is C16L4 (cache size of 16 bytes and line size of 4 bytes) and the minimum time cache configuration is C512L64. If the number of processor cycles is bound to 5,000, then the minimum energy cache configuration is C64L16. Similarly, if the energy (nJ) is bound to 5,500, then the minimum time cache configuration is C512L64.

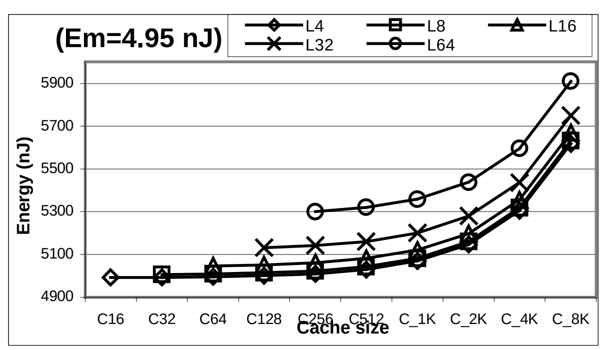
Cache size (C) Cache line size (L)	Em =4.95nJ	Examples				
		Compress	Mat. Multi.	PDE	SOR	Dequant
C=16 L=4	Miss rate	0.251	0.25	0.251	0.25	0.5
	Cycles	10358	320260	10357	320260	1311
	Energy (nJ)	4992	4978	4986	4983	9953
C=32 L=8	Miss rate	0.126	0.125	0.126	0.125	0.375
	Cycles	5678	175020	5676	175020	999
	Energy (nJ)	5006	4973	5002	4975	14896
C=64 L=16	Miss rate	0.0634	0.0625	0.0634	0.0625	0.312
	Cycles	3338	102400	3336	102400	843
	Energy (nJ)	5046	4974	5042	4975	24796
C=128 L=32	Miss rate	0.0322	0.0312	0.0322	0.0312	0.281
	Cycles	2167	66096	2166	66096	765
	Energy (nJ)	5132	4982	5126	4983	44605

Figure 2. Miss rate, # of cycles and energy vs. cache size and cache line size.



	C16	C32	C64	C128	C256	C512	C_1K	C_2K	C_4K	C_8K
L4	10358	10355	10350	10340	10320	10280	10199	10038	9716	9072
L8		5678	5675	5670	5660	5640	5600	5519	5358	5036
L16			3338	3335	3330	3320	3420	3377	3293	3123
L32				2167	2165	2160	2272	2250	2205	2116
L64					1582	1579	1700	1688	1664	1615

Figure 3. Example Compress. Variation in the number of cycles for different cache sizes and line sizes. The number of cache lines is >=4.



	C16	C32	C64	C128	C256	C512	C_1K	C_2K	C_4K	C_8K
L4	4992	4993	4996	5001	5010	5030	5069	5147	5304	5617
L8		5006	5009	5014	5023	5043	5082	5161	5318	5632
L16			5046	5051	5061	5081	5120	5198	5355	5670
L32				5132	5142	5161	5201	5279	5436	5750
L64					5300	5320	5359	5438	5595	5909

Figure 4. Example Compress. Variation in energy for different cache sizes and line sizes. (Em = 4.95 nJ)

4. ENHANCING CACHE PERFORMANCE

We present three techniques to improve cache performance, namely, off-chip memory assignment that significantly reduces the number of conflict misses, tiling size and set associativity.

4.1 Off-chip Memory Assignment

The off-chip memory assignment method is an extension of the method in [1] that is based on utilizing compatible array access patterns. We call two array access patterns compatible [1] if the difference in the accesses is independent of loop index. Thus, two arrays, a[i] and a[i-2] are compatible while two arrays, a[i] and a[b[i]], are incompatible. If all accesses in a loop are compatible for each equivalent class, then we can use a suitable data layout in memory to avoid cache conflicts completely [1]. Conflict misses occur when data that could possibly be used in the near future, is displaced from cache by a subsequently accessed element in a direct-mapped cache or limited associativity cache. Given an embedded application program and a cache line size, we determine the optimal memory assignment to avoid conflict misses not only for references that belong to the same class (as in [1]) but also for references that belong to the same case.

Consider the Compress benchmark example. The cache line size is 2 bytes. A memory assignment that avoids conflicts should ensure that the elements in class 1 and class 2 are never mapped into the same cache line. If the cache size is 8, then the first element in class 1, a[0][0], with memory address 0 is mapped to cache line 0. If the first element in class 2, a[1][0], has memory address 32, then it is mapped to cache line 0, causing a conflict in each iteration. If, instead, a[1][0] has memory address 36, then it is mapped to cache line 2, resulting in no conflict misses. Note that if a[1][0] has memory address 34, then there would be a conflict every second iteration. Thus, even though there is no valid data in locations 32 through 35 in the off-chip memory, the conflict misses have been avoided, thereby enhancing the performance significantly. Figure 5 show how the miss rate is significantly reduced if this memory assignment algorithm used in the Compress example.

Now consider the Matrix addition example [1] shown in example 2. The cache is direct-mapped with cache line size=2. The three different arrays a, b and c, can be assigned to three different cache lines which is the minimum number of cache lines as shown in the figure below.

0	1	2
a00 a01	b00 b01	c00 c01

0	35		38	73		76	111
a00	a55		b00	b55		c00	c55

If array a[] is stored in main memory from locations 0 through 35, then in order that b[0][0] be assigned to cache 1, it has to be stored in location 36+2=38. Thus the array b[] is stored in location 38 through 73. Similarly, in order that c[0][0] be assigned to cache line 2, it has to be stored in location 74+2=76. The array c[] is then stored in location 76 through 111.

Example 2. Matrix Addition

```
int a[6][6], b[6][6], c[6][6]
for  i=0,5
   for j=0,5
       c[i.j]=a[i,j]+b[i,j];
```

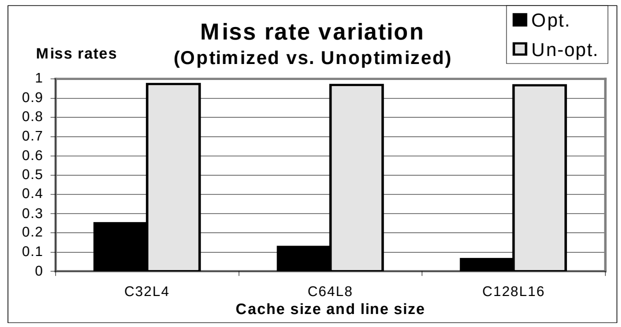


Figure 5. Example Compress. Miss rate reduction due to off chip memory assignment.

We would like to point out that we have developed analytical expressions to calculate the minimum cache line requirement, minimum cache size (in Section 3), off-chip data assignment (in Section 4.1), miss rates, # of cycles and energy (in Section 2). We chose to do this rather than developing a trace driven simulator that could be ported to Dinero ([11]).

4.2 Tiling

Tiling is primarily used to improve cache reuse by dividing the iteration space into tiles and transforming the loop nest to iterate over them [9]. However, it can also used to improve processor, register and cache locality. The need for tiling is illustrated by the loop in Example 3(a). With the j loop innermost, access to array b[] is stride-1, while access to array a[] is stride-n. Interchanging does not help, since it makes access to array b[] stride-n. After tiling, Example 3(b), the miss rate is drastically reduced. For instance, if tiling size is two, then the miss rate is reduced from 0.44 to 0.22.

Example 3(a)

```
for i=1,n
   for j=1,n
      a[i,j]=b[j,i];
```

After tiling →

Example 3(b)

```
for ti=1,n,64
  for tj=1,n,64
     for i=ti,min(ti+63,n)
        for j=tj,min(tj+63,m)
           a[i,j]=b[j,i];
```

Figure 6 describes how the miss rate, number of processor cycles and energy consumption are reduced as the tiling size is increased. Here the cache size is 64 bytes and cache line size is 8 bytes. The energy consumption reduces in all the examples up to tiling size of 8. This is to be expected since the tiling algorithm tries to reduce misses via improved temporal locality. However, if the tiling size is greater than the number of cache lines, the data in the cache gets replaced before being used. Therefore, the number of misses increases which results in an increase in the energy consumption. Our experiments show that for low energy applications, the tiling size should be as large as the number of cache lines. Figure 7 plots energy vs. tiling size for the Compress and Dequant examples.

Tiling size	C64L8	Examples				
		Compress	Mat. Multi.	PDE	SOR	Dequant
1	Miss rate	0.126	0.125	0.126	0.125	0.374
	Cycles	5680	175000	5670	175000	998
	Energy (nJ)	5010	4980	5000	4980	14900
2	Miss rate	0.0634	0.0625	0.0634	0.0625	0.312
	Cycles	3460	106000	3460	106000	883
	Energy (nJ)	2530	2490	2530	2500	12400
8	Miss rate	0.0166	0.0156	0.0166	0.0156	0.265
	Cycles	1710	51700	1710	51700	862
	Energy (nJ)	671	632	670	632	10600
16	Miss rate	0.0322	0.0312	0.0322	0.0312	0.281
	Cycles	2290	69800	2290	69800	837
	Energy (nJ)	1290	1250	1290	1250	11200

Figure 6. Miss rate, # of cycles and energy vs. tiling size. Em=4.95 nJ.

4.3 Set Associativity

The hit rate of the cache can also be improved by increasing its associativity. At the lowest level of associativity is the direct mapped cache, followed by increasing levels of set associativity. Figure 8 illustrates how the miss rate, # of cycles and energy consumption are reduced if the degree of set associativity is increased for our benchmark examples. Here the cache size is 64 bytes and the line size is 8 bytes. However, if the cache size is 1024 bytes and line size is 32 bytes, the number of processor cycles as well as the energy values do not necessarily decrease.

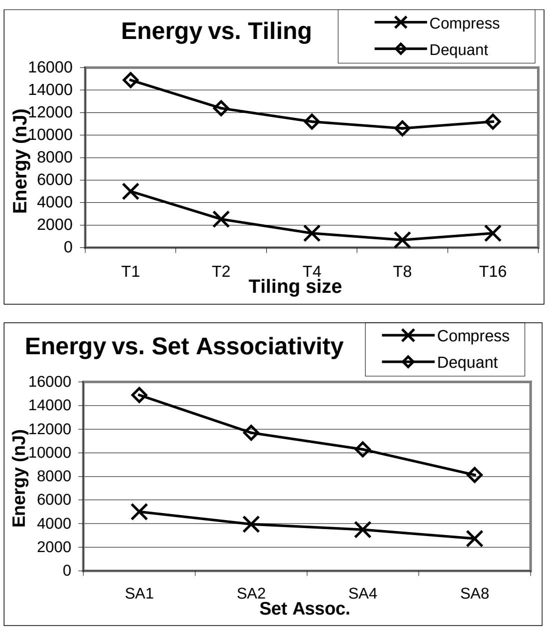


Figure 7. Example Compress and Dequant. Variation in energy with increase in tiling and set associativity.

Set assoc.	C64L8	Examples				
		Compress	Mat. Multi.	PDE	SOR	Dequant
1	Miss rate	0.126	0.125	0.126	0.125	0374
	Cycles	5800	179000	5800	179000	1020
	Energy (nJ)	5010	4980	5000	4980	14900
2	Miss rate	0.0989	0.0982	0.0988	0.0982	0.294
	Cycles	4850	150000	4850	150000	821
	Energy (nJ)	3940	3920	3940	3920	11700
4	Miss rate	0.0874	0.0869	0.0874	0.0869	0.26
	Cycles	4430	137000	4420	137000	735
	Energy (nJ)	3480	3460	3480	3460	10300
8	Miss rate	0.0687	0.0683	0.0687	0.0683	0.204
	Cycles	3730	115000	3730	115000	594
	Energy (nJ)	2740	2720	2740	2720	8130

Figure 8. Miss rate, # of cycles and energy vs. set associativity. Tiling size is 1 and Em=4.95 nJ.

Figure 9 shows the reduction in miss rate, # of cycles and energy when both set associativity and tiling are considered. The values in parentheses are unoptimized values, that is, values for the case when the memory assignment scheme in Section 4.1 has not been incorporated. Note that there is a significant difference between optimized and unoptimized values. The miss rate for the unoptimized case is extremely large, so large that tiling and set associativity have little effect.

Set Assoc. (SA) Tiling Size (TS)	C64L8	Examples				
		Compress	Mat. Multi.	PDE	SOR	Dequant
SA=1 TS=1	Miss rate	0.126 (0.969)	0.125 (0.999)	0.126 (0.968)	0.125 (0.999)	0.374 (0.531)
	Cycles	5680 (37300)	175000 (1190000)	5670 (37200)	175000 (1190000)	998 (1390)
	Energy (nJ)	5010 (38500)	4980 (39700)	5000 (38400)	4980 (39700)	14900 (21100)
SA=2 TS=4	Miss rate	0.0253 (0.761)	0.0246 (0.785)	0.0253 (0.761)	0.0246 (0.785)	0.221 (0.399)
	Cycles	2100 (32400)	64200 (1040000)	2100 (32400)	64200 (1040000)	677 (1170)
	Energy (nJ)	1020 (30300)	986 (31200)	1020 (30200)	987 (31200)	8790 (15900)
SA=8 TS=8	Miss rate	0.00908 (0.529)	0.00854 (0.546)	0.00906 (0.529)	0.00854 (0.546)	0.154 (0.273)
	Cycles	1500 (24900)	45900 (796000)	1500 (24900)	45900 (796000)	508 (892)
	Energy (nJ)	371 (21000)	350 (21700)	370 (21000)	350 (21700)	5770 (10900)

Figure 9. Miss rate, # of cycles and energy vs. set associativity and tiling size. The values in parentheses are unoptimized values.

5. MPEG DECODER: A CASE STUDY

In previous sections, we validated our exploration strategy by performing simulation experiments on small benchmark loop kernels. In this section, we present a case study of our exploration technique on a relatively large program: the MPEG Decoder. The MPEG behavior is characterized by a number of array access patterns, which makes it a good candidate on which to apply our memory exploration techniques. The MPEG decoder consists of many modules including VLD (Variable Length Decoder), Dequant, IDCT, Plus, Display, Store, and Prediction (consists of Addr, Fetch, and Compute) [7]. Figure 10 shows the minimum energy cache configuration for each kernel program in the MPEG decoder.

Kernel Program	Cache size	Line size	Set Assoc.	Tiling size	Energy (nJ)
VLD	64	4	8	16	223
Dequant	64	4	8	16	2900
IDCT	128	4	8	8	546
Plus	64	4	8	16	177
Display	256	16	8	16	167
Store	64	4	8	16	177
Addr	128	4	8	8	366
Fetch	128	4	8	8	367
Compute	64	4	8	16	177

Figure 10. Minimum energy cache configuration for each kernel program in the MPEG decoder.

The procedure to calculate the miss rate, # of processor cycles and energy of the large program is as above. The input to the procedure is a set of records for each kernel program, where a record is defined by (T, L, S, B, mr, C, E). Here T is the cache size, L is the line size, S is the degree of set associativity, B is the tiling size, mr is the miss rate, C is the number of cycles and E is the energy. For each combination of T, L, S, B, we compute the miss rate, the number of cycles and energy for the whole program (T, L, S, B, MISS_R, CYCLES, ENERGY). Let trip(j) be the number of times kernel program j is invoked. The procedure to determine the cache configuration that satisfies the energy bound or the time bound or both the energy and time bound is the same as before.

$$MISS\_R = \frac{\sum_j mr(j) * trip(j)}{\sum_j trip(j)}$$

$$CYCLES = \sum_j C(j) * trip(j)$$

$$ENERGY = \sum_j E(j) * trip(j)$$

The minimum energy cache configuration for the MPEG decoder is cache size of 64 bytes, line size of 4 bytes, 8-way set associativity and tiling size of 16. The energy consumption for this configuration is only 293,000 nJ and the number of processor cycles is 142,000. The cache configuration corresponding to the minimum number of processor cycles (121,000) is cache size of 512 bytes, line size of 16 bytes, 8-way set associativity and tiling size of 8. The corresponding energy is 1,110,000 nJ. Note that the lowest energy cache configuration for the large program is different from the lowest energy configuration any of the kernel programs. Also the cache configuration corresponding to lowest energy is quite different from the configuration corresponding to minimum number of processor cycles.

6. CONCLUSION

We presented a data memory exploration procedure for low power, embedded systems. Our analysis shows that while increasing cache size, cache line size, tiling and set associativity reduces the miss rate and the number of cycles, it does not necessarily reduce the energy. Thus it is important to add energy to the (traditional) performance metrics of size and number of processor cycles. Furthermore, the miss rate can be significantly reduced by suitable off-chip data organization if the arrays are compatible (since the conflict misses can be completely eliminated). Thus off-chip data organization should be an integral part of the design procedure. The energy-time tradeoffs developed in this analysis enable us to find the minimum energy cache configuration if there is a bound on the number of cycles or the minimum time cache configuration if there is a bound on the energy. This exploration procedure was validated by performing simulation experiments on small benchmark loop kernels (Compress, Matrix Multiplication, PDE, SOR, Dequant) as well as the MPEG decoder which consists of several loop kernels. We found that the minimum time cache configuration can be significantly different from the minimum energy cache configuration. Furthermore, the minimum energy (or time) cache configuration for a large program (like the MPEG decoder) can be quite different from the minimum energy (or time) cache configuration of its constituent kernel programs.

7. ACKNOWLEDGEMENTS

This work was carried out at the Center of Low Power Electronics which is supported by the National Science Foundation, the state of Arizona, and 13 member companies from the microelectronics industry. The authors would like to thank Andreas Wild of Motorola for supporting this research, Manapat Rajesh of Cypress for providing energy estimates of off-chip memory, and P. R. Panda of Synopsys for sharing his work on caches for embedded applications.

8. REFERENCES

[1] P. R. Panda, N. D. Dutt, and A. Nicolau. "Data Cache Sizing for Embedded Processor Applications." Technical Report ICS-TR-97-31, University of California, Irvine, June 1997.

[2]P. R. Panda, N. D. Dutt, and A. Nicolau. "Architectural Exploration and Optimization of Local Memory in Embedded Systems." International Symposium on System Synthesis (ISSS 97), Antwerp, Sept. 1997.

[3] M. B. Kamble and K. Ghose, "Analytical Energy Dissipation Models for Low Power Caches", International Symposium on Low Power Electronics and Design, 1997.

[4] S. E. Wilton and N. Jouppi, "An Enhanced Access and Cycle Time Model for On-chip Caches", Digital Equipment Corporation Western Research Lab, Tech. Report 93/5, 1994.

[5] C. Su and A. Despain, "Cache Design Trade-offs for Power and Performance Optimization: A Case Study", International Symposium on Low Power Electronics and Design, pages 63-68, 1995.

[6] P. Hicks, M. Walnock, R. M. Owens, "Analysis of Power Consumption in Memory Hierarchies", International Symposium on Low Power Electronics and Design, pages 239-242, 1997.

[7] A. Thordarson, "Comparison of Manual and Automatic Behavioral Synthesis of MPEG Algorithm", Master's thesis, University of California, Irvine, 1995.

[8] D. Kirovski, C. Lee, M. Potkonjak, and W. Mangione-Smith, "Application –Driven Synthesis of Core-based Systems", In Proceedings of the IEEE/ACM International Conference on Computer Aided Design, pages 104-107, San Jose, CA, November 1997.

[9] M. E. Wolf and M. Lam. "A Data Locality Optimizing Algorithm." In proceedings of the SIGPLAN'9 Conference on Programming Language Design and Implementation, pages 30-44, June 1991.

[10] J. L. Hennessy and D. A. Patterson, "Computer Architecture A Quantitative Approach", 2nd edition Morgan Kaufman Publishers, 1996.

[11] J. Edler and M. D. Hill, " Dinero IV Trace-Driven Uniprocessor Cache Simulator", web site: http://www.neci.nj.nec.com/homepages/edler/d4 or http://www.cs.wisc.edu/~markhill/DineroIV.