

# Memory Design and Exploration for Low Power, Embedded Systems

WEN-TSONG SHIUE AND CHAITALI CHAKRABARTI

*Arizona State University, AZ, USA*

*Received January 31, 2000; Revised October 31, 2000*

**Abstract.** In this paper, we describe a procedure for memory design and exploration for low power embedded systems. Our system consists of an instruction cache and a data cache on-chip, and a large memory off-chip. In the first step, we try to reduce the power consumption due to memory traffic by applying memory-optimizing transformations such as loop transformations. Next we use a memory exploration procedure to choose a cache configuration (cache size and line size) that satisfies the system requirements of area, number of cycles and energy consumption. We include energy in the performance metrics, since for different cache configurations, the variation in energy consumption is quite different from the variation in the number of cycles. The memory exploration procedure is very efficient since it exploits the trends in the cycles and energy characteristics to reduce the search space significantly.

## 1. Introduction

In systems that involve multidimensional streams of signals such as images or video sequences, it has been shown that the majority of the area and power cost is not due to the datapath or controllers but due to the global communication and memory interactions [1][2]. In fact, in embedded applications for real-time signal processing, 50-80% of the power cost is due to memory traffic caused by transfers between the ASIC and the off-chip memories. Even general-purpose processors such as the 21164 DEC Alpha chip or the StrongArm SA-110 processor dissipate 25-30% of the total power in the cache. Clearly, in order to reduce the system-level power consumption, it is very important to focus on design strategies that would reduce the power consumption due to memory traffic.

In this paper, we describe a procedure for memory design and exploration for low power embedded systems. Our system consists of a register file, a data cache and an instruction cache on-chip, and a large memory off-chip.

The first step in our procedure is application of memory optimizing transformations to reduce the memory size and number of accesses. Work on applying loop transformations to reduce power in data

dominated applications has been presented in [2][3] and more recently in [4]. The next step in our procedure is memory exploration. While traditionally the system constraints for memory exploration have been area and number of cycles, in low power embedded system design, the system constraints are area, number of cycles and energy consumption. Energy is added to the performance metrics since the variation in the number of cycles is quite different from the variation in the energy consumption. The proposed exploration procedure for data cache configuration chooses the cache size and line size that best satisfies the system requirements of area, number of cycles and energy consumption. It does so efficiently by exploiting the trends in the cycles and energy characteristics to reduce the search space significantly. For applications where the effect of the instruction cache cannot be ignored, the exploration procedure determines the instruction cache and data cache sizes that satisfies the area and energy bounds. The exploration procedure has been validated by performing simulation experiments on a variety of applications including image and video coders, speech and audio coders, etc. from the MediaBench suite of applications. The cache simulator in the *SimpleScalar* tool suite has been used. The work presented in this paper is an extension of our earlier work [3][5].

The main contributions of this paper are listed below.

- Developed a partial ordering of the memory-optimizing loop transformations.
- Showed the differences in the variation of the number of cycles and energy consumption for different cache sizes and line sizes.
- Showed the effect of set associativity on the number of cycles and energy.
- Developed memory exploration procedures that determine the minimum energy data cache configuration that satisfies the area and cycles bound, and the minimum cycles data cache configuration that satisfies the area and energy bound.
- Developed memory exploration procedures that determine the minimum energy cache configuration (instruction cache and data cache) that satisfy the area bounds.

Memory optimization for embedded systems has been addressed extensively in [6]-[8]. The performance metrics of the system are data cache size and number of processor cycles. In addition, an excellent method for off-chip data placement is proposed that reduces the number of conflict misses significantly. In [9], memory system design for video processors has been studied; the performance metrics are area, cycle time and utilization.

In recent years, the focus has been on system-level based approaches that consider the interdependencies between the different system parts. For instance, in [10], a heuristic procedure to select an area-minimal processor core and cache configuration that satisfies the performance requirements (in terms of the hit rate) has been proposed. Different data and instruction cache sizes for fixed line size and associativity have been considered. In [11], tradeoffs between energy and performance for a system with CPU and caches has been studied. Different sizes and associativities of the instruction and data cache have been considered. In [12], power, performance and area tradeoffs have been studied for a system comprising of not only the CPU and caches but also the bus interfaces. An analysis of the results shows that the CPU-to-cache bus size is the major component in determining the system's power and execution time metrics. Also, with deep submicron technologies, the interface power would be dominant. Finally, cycle-accurate simulation

of energy dissipation in embedded systems have recently been proposed in [13][14]. These simulators will certainly enhance the study and analysis of system-level energy-performance trade-offs.

The rest of the paper is organized as follows. Section 2 describes how loop transformation effects the reduction in number of memory accesses. Section 3 describes the performance metrics used in our system. Section 4 describes the tradeoffs between number of cycles and energy consumption in the data cache. Section 5 describes the data cache exploration procedure. Section 6 describes the effect of the instruction cache. Section 7 describes the search space reduction in data cache – instruction cache exploration. Section 8 concludes the paper.

## 2. Loop Transformations

Loop transformation procedures can be used to significantly reduce the number of accesses as well as reduce the size of the on-chip/off-chip memory. This has been illustrated in several papers [1][15][16]. We demonstrate the power of loop transformations with the example in Figure 1. Here, loop reordering allows array  $c[]$  and array  $w[]$  to share memory space, thereby reducing the size of the off chip memory. Loop interchange helps to reduce the number of memory reads. Loop fusion reduces the number of memory accesses since variables  $b[i]$  and  $a[i]$  can be stored in register files. Before doing loop transformations, we need off-chip memory of size  $6n$ ,  $n+2$  on-chip registers and  $8n$  accesses while after doing loop transformations, we only need off-chip memory of size  $3n$ ,  $5$  on-chip registers and  $4n$  accesses. Clearly, the number of accesses and the size of storage have been significantly reduced.

Determining the order in which loop transformations have to be applied is hard. This is because application of some of the transformations preclude the application of some others. Clearly, a loop transformation framework is required. The framework introduced in [17] supports several transformations including loop interchange, iteration space tiling, loop skewing, and vectorization. However, each transformation has its own special legality test based on the direction vectors and on the nature of loop bound expressions.

A method to unify transformations like loop interchange, loop reversal and loop skewing has been proposed in [18][19]. This unification is possible since these loop transformations can be represented by unimodular matrices. Loop tiling which cannot be

represented by a unimodular matrix is facilitated by this framework. However, other transformations are not supported by this framework, making it restrictive. A more general framework based on transformation template instantiations has been presented in [20].

Use of loop transformation to reduce the power consumption due to memory accesses has been first presented in [1]. The approach consists of applications of loop fusion, loop reordering, and loop interchange to reduce the number of off-chip accesses as well as size of storage.

In this paper, we propose the following loop transformation ordering.

1. *Compiler techniques.*
2. *Memory access transformations..*
3. *Loop reordering.*
4. *Loop fission.*
5. *Loop interchange enabled by loop skewing.*
6. *Loop fusion enabled by loop normalization and loop peeling.*
7. *Loop tiling enabled by loop skewing.*
8. *Loop unrolling.*

In step 1, existing compiler techniques such as (i) constant propagation and elimination techniques to eliminate unreachable code, useless code, and dead variables and (ii) transformation of procedure calls to reduce the amount of register usage, are used. Compilers today are advanced enough to perform the above operations on a given code. In step 2, memory access transformations consisting of (i) array contraction which is used to transform array variables into scalar variables within a loop nest (in contrast to array elements, scalar variables have better cache behavior and can be allocated to registers), (ii) scalar replacement which is used to reduce the number of accesses by replacing invariant arrays within the innermost loop with scalars, and (iii) code colocation which is used to improve memory access behavior by placing related code in close proximity, are used.

In step 3, loop reordering is used to reduce the size of off-chip memory by moving forward loops that read from arrays that are not alive in the rest of the program. Such a movement helps save the off-chip storage since the space occupied by these arrays can now be used to store other arrays. For instance, in Example 1, arrays  $m[]$ ,  $n[]$ , and  $p[]$  in loop 3 are not alive in the rest of the program and thus this loop is moved to the beginning. This allows the space for arrays  $m[]$ ,  $n[]$ , and  $p[]$  to be used to store arrays  $a[]$ ,

$c[]$ , and  $e[]$ . A global dependency graph is required to apply this transformation.

Example 1

Before	After Loop reordering
L1:for i=1,N a[i]=f(b[i])	L3:for i=1, N k[i]=f(m[i],n[i],p[i])
L2:for i=1, N c[i]=f(d[i])	L1:for i=1,N a[i]=f(b[i])
L3:for i=1,N k[i]=f(m[i],n[i],p[i])	L2:for i=1,N c[i]=f(d[i])
L4:for i=1,N e[i]=f(h[i])	L4:for i=1,N e[i]=f(h[i])

In step 4, loop fission is done on loops that have either no data dependencies or support different access patterns. In Example 2, the first statement has temporal locality (stride 1) while the second statement has stride N. Loop interchange alone does not help since it results in the second statement having stride 1 while the first statement has stride N. If, however, loop fission followed by loop interchange is done, then both the loops would have temporal locality.

Example 2

Original code	After loop fission
for i=1,N for j=1,N a[i,j]=f(a[i,j-1]) b[i+1,j]=f(b[i-1,j])	for i=1,N for j=1,N a[i,j]=f(a[i,j-1]) for i=1,N for j=1,N b[i+1,j]=f(b[i-1,j])
After loop fission and loop interchange	
for i=1,N for j=1,N a[i,j]=f(a[i,j-1]) for j=1,N for i=1,N b[i+1,j]=f(b[i-1,j])	

In step 5, loop interchange enabled by loop skewing is done. Loop interchange is one of the most powerful transformations. It is used to increase the reuse of code variables such that they can be temporarily stored in registers instead of memory. This causes reduction in the size of on-chip memory and the number of memory reads.

In step 6, loop fusion enabled by loop normalization and loop peeling is done. Loop normalization not only exposes opportunities for fusion, it also helps to reveal which loops are candidates for peeling followed by fusion. Loop peeling can remove dependences created

by the first or last few loop iterations, thereby enabling parallelization and matching the iteration space of adjacent loops to enable fusion. While loop fusion helps reduce the number of memory accesses and also the size of off-chip memory, it should only be done if there are data dependencies between the two fusing loops. This is because loop fusion causes an increase in the size of the loop body which in turn causes an increase in the minimum cache size (that is required to avoid conflict misses) which in turn causes an increase in the energy consumption. Note that loop fusion is done after loop fission and loop interchange. We explain why this is so with the help of Example 3.

Example 3

Original code	Loop fission
L1: for i=1,N for j=1,N b[i,j]=b[i,j-1]+b[i,j-2] a[i,j]=a[i-1,j]+a[i-2,j] L2: for i=1,N for j=1,N c[i,j]=b[i,j]+b[i,j-1]+k L3: for j=1,N For i=1,N d[i,j]=a[i,j]+d[i-1,j]	L11: for i=1,N for j=1,N b[i,j]=b[i,j-1]+b[i,j-2] L12: for i=1,N for j=1,N a[i,j]=a[i-1,j]+a[i-2,j] L2: for i=1,N for j=1,N c[i,j]=b[i,j]+b[i,j-1]+k L3: for j=1,N For i=1,N d[i,j]=a[i,j]+d[i-1,j]
Loop interchange	Loop fusion
L11: for i=1,N for j=1,N b[i,j]=b[i,j-1]+b[i,j-2] L12: for j=1,N for i=1,N a[i,j]=a[i-1,j]+a[i-2,j] L2: for i=1,N for j=1,N c[i,j]=b[i,j]+b[i,j-1]+k L3: for j=1,N For i=1,N d[i,j]=a[i,j]+d[i-1,j]	L11&L2: for i=1,N for j=1,N b[i,j]=b[i,j-1]+b[i,j-2] c[i,j]=b[i,j]+b[i,j-1]+k L12&L3 for j=1,N for i=1,N a[i,j]=a[i-1,j]+a[i-2,j] d[i,j]=a[i,j]+d[i-1,j]

Here, loop fission is done on loop L1 to separate the first statement which has stride 1 from the second statement which has stride N. Loop interchange of the indices of L12 allows it to be fused with loop L3. At the same time, loop L11 and L2 are fused since there are data dependencies between the statements in each of the loops. Thus, if the original code has loops with arrays of different vector space shapes, we first use loop fission to partition the loops. Then we use loop skewing and loop interchange to reshape the vector

space and finally use loop fusion to save on the number of off-chip reads.

In step 7, loop tiling is done to improve register, cache locality by dividing an iteration space into tiles and transforming the loop nest to iterate over them. Loop skewing may have to be needed to enable loop tiling. In step 8, loop unrolling is used to improve register and cache locality. This is because loop unrolling results in larger number of elements of the same array in the body of the loop.

### 3. Memory Exploration Preliminaries

#### 3.1 Experimental Set Up

Seven different applications from MediaBench [21] are simulated and their characteristics are analyzed. The applications considered are: EPIC, JPEG, and MPEG2, which are image and video coders, GSM, G.721, and ADPCM, which are speech and audio coders, and PEGWIT, which is a public key encryption algorithm. Simulations are performed using the cache simulators available in SimpleScalar tool release 2.0 [22]. Simulators *sim-cache* and *sim-outorder* have been used.

#### 3.2 Performance Metrics

In this section we describe the three performance metrics of our memory exploration system, namely, (i) cache size, (ii) number of cycles and (iii) energy consumed in the memories.

##### 3.2.1 Cache Size

The cache sizes are chosen to be powers of 2. If the cache size,  $C$ , is increased or if the number of cache lines,  $L$ , is increased, the miss rate is reduced. Choosing the largest cache size is the best solution when cycles reduction is the primary goal. However, it is not necessarily the best solution when energy reduction is the primary goal. Thus performance tradeoffs have to be studied before choosing the best cache configuration.

##### 3.2.2 Number of Cycles

The number of cycles is very closely related to the miss rate. In this paper, the number of cycles was obtained using *sim-outorder* simulator from SimpleScalar. Note that *sim-outorder* takes a long time and so it may be preferable to use the average memory access time as the performance metrics instead [23].

### 3.3 Energy Consumed in the Memories

There exist several energy models for caches [24]-[26]. The model that we have used here is based on the analytical model developed by [24]. The process models are based on measurements reported by [25] for a 0.8  $\mu\text{m}$  process technology. We have combined the energy models in [24] and [26] to develop a model that leads to fast estimation as in [26] and yet matches the energy values in [24] quite closely. Furthermore, we not only consider the energy consumed in the host processor but also consider the energy consumed in the off-chip memory.

In our energy model, the total energy is given by  $\text{Energy} = \text{Edec} + \text{Ecell} + \text{Eio} + \text{Emain}$ , where  $\text{Edec}$  is the energy in the decoder block,  $\text{Ecell}$  is the sum of the energy in the cell arrays,  $\text{Eio}$  is the energy consumed in the address pads and data pads of the host processor,  $\text{Emain}$  is the energy consumed in off-chip memory.  $\text{Emain}$  consists of two components – the energy consumed in the address and data pads in the off-chip memory and the energy of accessing the off-chip memory.

The energy model 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  $\text{Edec}$  to be only the energy consumed in the address buses. Similarly, in cell arrays, we consider  $\text{Ecell}$  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 pads of the host processor and off-chip memory and the energy consumed in accessing the off-chip memory. For 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 4ns, voltage of 3.3V, current of 375 mA, and has energy consumption of 4.95 nJ per access. In summary,

$$\begin{aligned} \text{Energy} &= \text{Edec} + \text{Ecell} + \text{Eio} + \text{Emain} \\ - \text{Edec} &= \alpha * (\text{Add\_bus\_bs}) * (C/L) \\ - \text{Ecell} &= \beta * (\text{Word\_line\_size}) * (\text{Bit\_line\_size} + 4.8) \\ &\quad * (\text{Nhit} + \text{Nmiss}) \\ - \text{Eio} &= \gamma * (\text{Data\_pad\_bs} * 8L + \text{Add\_pad\_bs}) \\ - \text{Emain} &= \gamma * (\text{Data\_pad\_bs} * 8L + \text{Add\_pad\_bs}) \\ &\quad + \text{Em} * 8L * \text{Nmiss} \end{aligned}$$

and

$$\begin{aligned} \text{Add\_bus\_bs} &= \text{Pr}_1 * (\text{Nhit} + \text{Nmiss}) * \text{Wadd} \\ \text{Add\_pad\_bs} &= \text{Pr}_1 * (\text{Nmiss}) * \text{Wadd} \end{aligned}$$

$$\begin{aligned} \text{Data\_pad\_bs} &= \text{Pr}_2 * (\text{Nmiss}) \\ \text{Word\_line\_size} &= m * (8L + T + \text{st}) \\ \text{Bit\_line\_size} &= C / (m * L) \\ \alpha &= 7.89\text{e-}17; \quad \beta = 1.44\text{e-}14; \quad \gamma = 5.45\text{e-}11; \\ \text{Em} &= 4.95\text{e-}9 \text{ J} \end{aligned}$$

where  
 $C$  = cache size.  
 $L$  = cache line size.  
 $m$  = m-way set associative cache.  
 $T$  = tag size of  $T$  bits  
 $\text{St}$  = number of status bits per block frame.  
 $\text{Pr}_1, \text{Pr}_2$  = the probability of a 0 to 1 transition  
 $\text{Nhit}$  = number of hits.  
 $\text{Nmiss}$  = number of misses.  
 $\text{Wadd}$  = the width of address bus.  
 $\text{Add\_bus\_bs}$  = number of bit switches on address bus.  
 $\text{Add\_pad\_bs}$  = number of bit switches on address pads.  
 $\text{Data\_pad\_bs}$  = number of bit switches on data pad.  
 $\text{Word\_line\_size}$  = number of cells in a word line.  
 $\text{Bit\_line\_size}$  = number of memory cells in a bit line.  
 $\text{Em}$  = Energy consumption of a main memory access  
 $\alpha, \beta$  and  $\gamma$  are used for 0.8  $\mu\text{m}$  CMOS technology

## 4. Tradeoffs Between Number of Cycles and Energy Consumption in the Data Cache

In this section, we describe the differences between the variation in the energy consumption and the variation in the number of cycles for different data cache configurations. We first describe the miss rate variation since it helps us analyze the variations in the number of cycles and energy consumption.

### 4.1 Variation in the Miss Rate

For a specific line size, as the cache size increases, the miss rate drops. The absolute value of the miss rate varies from program to program. For the MPEG2 encoder example (see Figure 1(a)), for  $L=32$ , the miss rate drops from 0.073 at  $C=1\text{K}$  to 0.016 at  $C=64\text{K}$ . (In other examples that we studied, the drop in the miss rate is a lot more significant). For very large cache sizes, the drop in the miss rate is usually quite small. In the MPEG2 encoder example, there is negligible drop in the miss rate for  $C>32\text{K}$ .

In some examples, for low cache size, the miss rate for low  $L$  is lower than the miss rate for high  $L$ . However, as the cache size increases, the miss rate for low  $L$  becomes higher than the miss rate for high  $L$ . We refer to this phenomenon as cross over.

#### 4.2 Variation in the Number of Cycles

The variation in the number of cycles is very closely related to the variation in the miss rate. For a specific line size, as the data cache size increases, the number of cycles drops. While the drop in the cycles is large for smaller cache sizes, it is very small for larger cache sizes. In fact, the cycles curve is flat for large cache sizes. For instance, for the MPEG2 encoder (see Figure 1(b)), for  $L=16$ , the cycles saturate at  $C=32K$ . An interesting point to note is that the number of cycles saturates for all values of  $L$  at  $C=32K$ .

The variation in the number of cycles with line size is very similar to the variation in the miss rate with line size. If there is a cross over in the miss rate plot, there is a similar cross over in the cycles plot.

#### 4.3 Variation in the Energy Consumption

The variation in the energy consumption is different from the variation in the number of cycles. For a specific line size, as the cache size increases, the energy reduces at first and then increases. In other words, there exists a cache size ( $C_{dip}$ ) for which the energy is minimum. In the MPEG2 example shown in Figure 1(c), the minimum energy cache size for  $L=32$  is  $C=32K$ . The minimum energy cache size varies with the line size. Note that the value of  $C_{dip}$  varies from program to program. Also, the dip in the energy curve may not be apparent if the maximum allowed cache size,  $C_{max}$ , is smaller than  $C_{dip}$ .

The reason for the dip in the energy curve can be analyzed by revisiting the energy equation.  $E=E_{dec}+E_{cell}+E_{io}+E_{main}$  can be simplified to  $E=E_{cell}+E_{main}$  since the  $E_{dec}$  and  $E_{io}$  terms are negligible. For small cache sizes, the miss rate is higher and  $E_{main}$  is the dominant term. As the cache size increases, the effect of  $E_{main}$  reduces since miss\_rate reduces but  $E_{cell}$  increases. Thus for  $C < C_{dip}$ ,  $E$  follows  $E_{main}$  and for  $C > C_{dip}$ ,  $E$  follows  $E_{cell}$ .

The energy consumption increases with the increase in line size for all cache sizes. This is to be expected since a larger line size implies that a larger amount of data has to be fetched from the main memory. In some examples, the energy curves for different values of  $L$  converge after the dip. This is because the  $E_{cell}$  term is almost constant for all  $L$ , and the variation in the  $E_{main}$  term is negligible (due to the variation in the miss rate being negligible).

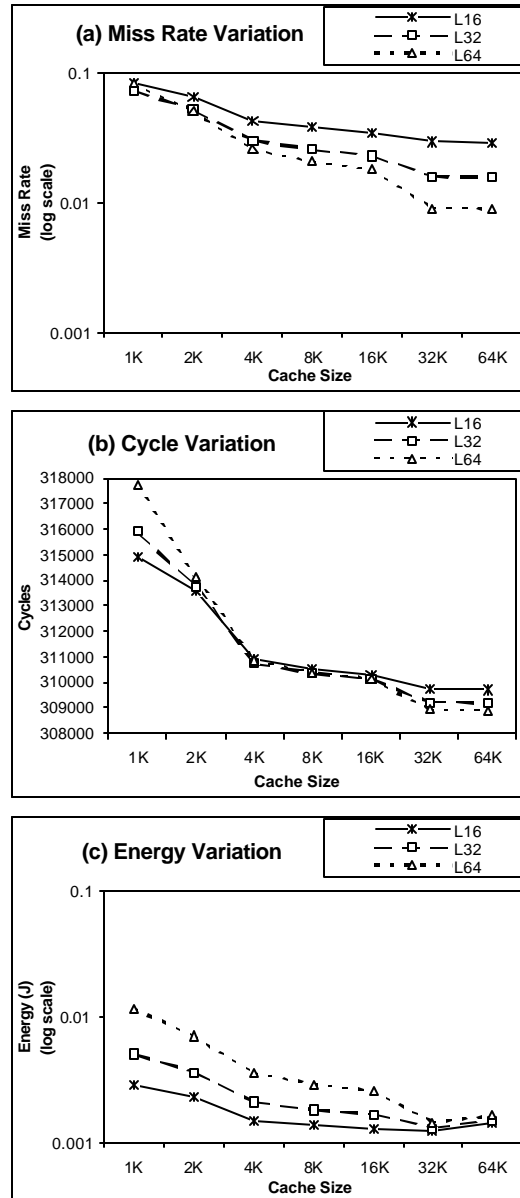


Figure 1. MPEG2 encoder. (a) Miss rate, (b) Number of cycles, and (c) Energy consumption for different data cache sizes and line sizes.

Next, we study the variation in the data cache energy for different cache sizes and line sizes. The data cache energy is a subset of the total energy, since it does not

include the energy of the off chip memory. This study is important to processor vendors who are more concerned about the energy consumption in the processor chip.

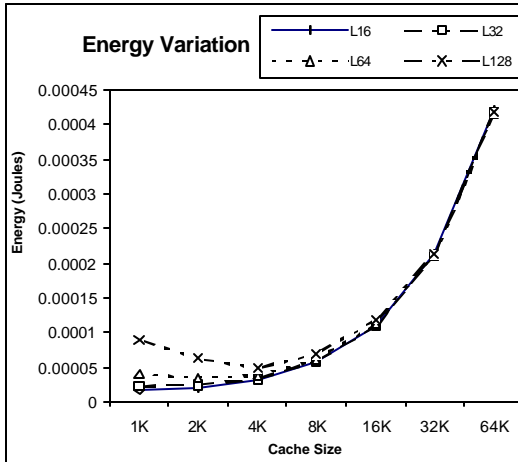


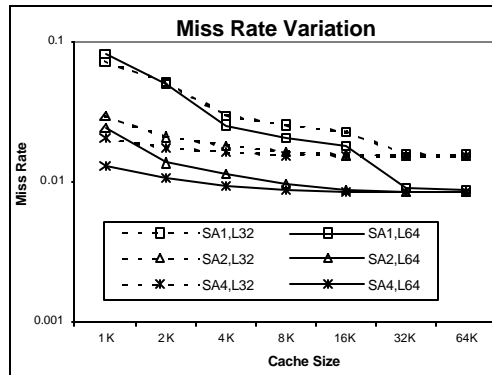
Figure 2. Variation in the data cache energy for MPEG2 encoder.

Figure 2 describes the variation in the data cache memory energy for the MPEG2 encoder example. For small cache sizes, the miss rate is larger, and the effect of the  $E_{io}$  term is not negligible. As the cache size increases, the miss rate reduces and the  $E_{cell}$  term dominates. For large cache size ( $C > 8K$ ), the variation in the miss rate for different values of  $L$  is quite small and as a result, the energy curves for different values of  $L$  converge.

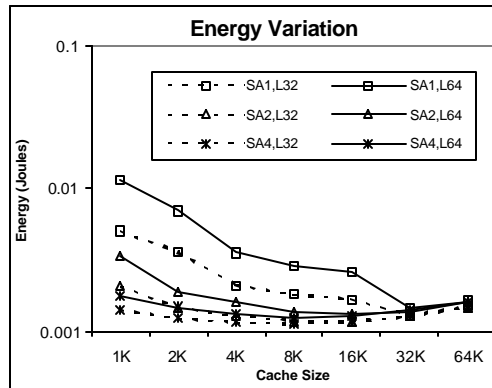
#### 4.4 Effect of Set Associativity

The miss rate of the cache can be improved by increasing its associativity. Figure 3 illustrates how the miss rate and energy consumption vary with data cache size for  $L=32$  and  $64$ , and associativities of  $1, 2,$  and  $4$ . For the same line size (say  $L=64$ ), as the associativity increases, the miss rate drops and the energy consumption reduces. Interestingly enough, as the associativity increases, the energy curve dips at lower value of the cache size. For instance, for the MPEG2 encoder example, for  $L=64$ , the energy curve dips at  $32KB$  for associativity of  $1$ , dips at  $16KB$  for associativity of  $2$  and dips at  $8KB$  for associativity of  $4$ .

For the same cache size and line size, the drop in energy when the associativity increases from  $1$  to  $2$  is larger than when the associativity increases from  $2$  to  $4$ . For the MPEG2 encoder example (see Figure 3), for  $C=2KB$  and  $L=64$ , the drop in energy is  $0.012J$  when the associativity increases from  $1$  to  $2$  and only  $0.004J$  when the associativity increases from  $2$  to  $4$ . As the cache size increases, this drop reduces even further.



(a)



(b)

Figure 3. Variation in (a) miss rate and (b) energy for different line sizes and set associativity for MPEG2 encoder.

## 5. Search Space Reduction in Data Cache Exploration

In this section, we briefly describe efficient algorithms to find the minimum energy data cache configuration given the cache size and cycle time bounds, and the minimum cycle cache configuration given the cache

size and energy bounds. These algorithms do not look at all possible cache configurations but only a small subset of configurations. This reduces the time to search for the “best” cache configuration significantly. For more details, please refer to [3].

### 5.1 Minimum Energy Data Cache Configuration

The algorithm for finding the minimum energy cache configuration given the cache size bound ( $C_{\max}$ ) and the cycle time bound ( $T_{\text{bound}}$ ) first identifies whether  $C_{\max}$  is smaller or larger than the cache size at which the energy curve dips ( $C_{\text{dip}}$ ). This is important since for cache sizes larger than  $C_{\text{dip}}$ , the energy monotonically increases and the number of cycles either monotonically decreases or saturates, while for cache sizes smaller than  $C_{\text{dip}}$ , both the energy and the number of cycles monotonically decreases. If  $C_{\max} < C_{\text{dip}}$ , the algorithm chooses the smallest line size that meets the cycles bound. This is because smaller the line size, smaller the energy. If  $C_{\max} > C_{\text{dip}}$ , the algorithm first searches for  $C_{\text{dip}}$  (since this is not known apriori) corresponding to  $L_{\min}$ , provided the cycles bound is satisfied. If the time bound is not satisfied for  $L_{\min}$ , it searches for higher line sizes. The input parameters are the range of line sizes ( $L_{\min}$  to  $L_{\max}$ ), the range of cache sizes ( $C_{\min}$  to  $C_{\max}$ ) and the cycles bound ( $T_{\text{bound}}$ ).

**Complexity Analysis:** Let  $n_L$  be the number of line sizes and let  $n_C$  be the number of cache sizes considered. We measure the complexity on the basis of the number of simulations that we have to run. The complexity in the worst case, is  $n_L+2$  if  $C_{\max} < C_{\text{dip}}$ , and  $\max\{k, n_L\}+2+2*\min\{k, n_L\}$  if  $C_{\max} > C_{\text{dip}}$ .

### 5.2 Minimum Cycle Data Cache Configuration

The algorithm for finding the minimum cycle cache configuration given the cache size bound and the energy bound ( $E_{\text{bound}}$ ) also begins by identifying whether  $C_{\max}$  is smaller or larger than  $C_{\text{dip}}$ . If  $C_{\max} < C_{\text{dip}}$ , the algorithm chooses  $L_{\min}$  if cross over has not occurred, and chooses the largest line size that satisfies the energy bound if cross over has occurred. Recall that while the minimum energy cache configuration always occurs with  $L_{\min}$ , it is not true for the minimum cycle time configuration. If  $C_{\max} > C_{\text{dip}}$ , and the energy bound is not satisfied at  $C_{\max}$ , the algorithm tries to find a smaller cache size (corresponding to  $L_{\min}$ ) at which the bound is satisfied.

**Complexity Analysis:** The worst case complexity is  $n_L+3$  if  $C_{\max} < C_{\text{dip}}$ ,  $k+2n_L+3$  if  $C_{\max} > C_{\text{dip}}$ .

## 6. Effect of Instruction Cache

In the previous sections, we have considered the variation in the number of cycles and energy consumption in the data cache for a fixed sized instruction cache ( $C=8\text{KB}$ ,  $L=32$ ). In this section, we consider the variation in the number of cycles and energy consumption when both the instruction and data cache sizes are varied. We assume that  $L=32$  for both the instruction and data cache, and that each of the cache sizes vary from 1KB to 64KB. We consider two examples from MediaBench – EPIC and MPEG2. We choose these since the energy consumption in the instruction cache and the data cache are very different for the two cases.

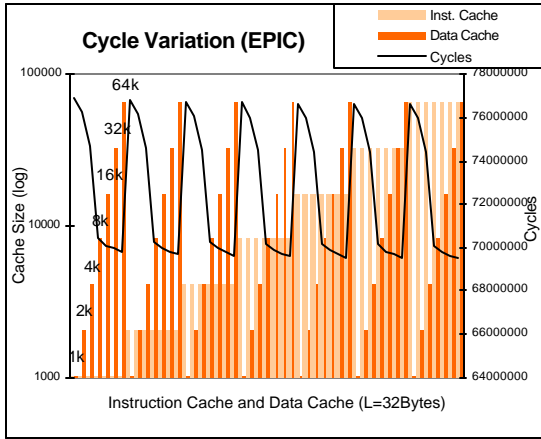
Figure 4 plots the number of cycles and energy for example EPIC (from MediaBench). In this example, we see that for a constant instruction cache size, the number of cycles vary significantly ( $\sim 76.8 - 69.7e6$ ) with increase in data cache size. The variation is almost negligible if the data cache size is kept constant and the instruction cache size is varied. If we consider energy, then for the same data cache size, the instruction cache energy increases from 0.072J to 0.407J. If however, the instruction cache size is kept constant, the data cache energy drops from 2.31J to 0.28J with increase in data cache size. Thus the variation in the data cache energy is a lot more significant. This is not surprising since the instruction cache miss rates are so small ( $1e-3 - 1e-4$ ) that variation due to instruction cache can be almost neglected.

Next we consider the MPEG2 encoder example where the miss rate for instruction cache is significant and its effect cannot be ignored. Figure 5 plots the number of cycles and energy for this example. For constant instruction cache size, the number of cycles vary by  $\sim 5e3$  cycles with variation in data cache size. For instance, if the instruction cache size is 2K, the number of cycles vary from 40.4 to 40e3, while if the instruction cache size is 32K, the number of cycles vary from 31 to 30.4e3 with increase in data cache size.

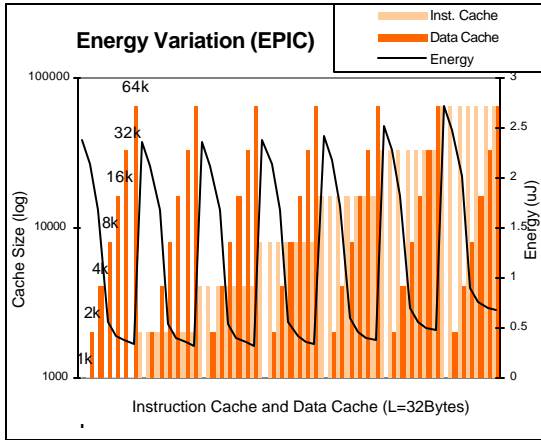
The variation is much more spectacular if the data cache is kept constant and the instruction cache is varied. For instance, if the data cache is 2K, the number of cycles varies from 42.5 to 29.9e3 with increase in instruction cache size.

The energy variation follows a similar pattern. For constant instruction cache size, the variation in energy

due to variation in data cache energy is small (0.005J – 0.001J). In contrast, for constant cache size, the variation in energy due to variation in instruction cache size is significant (0.035J – 0.004J).



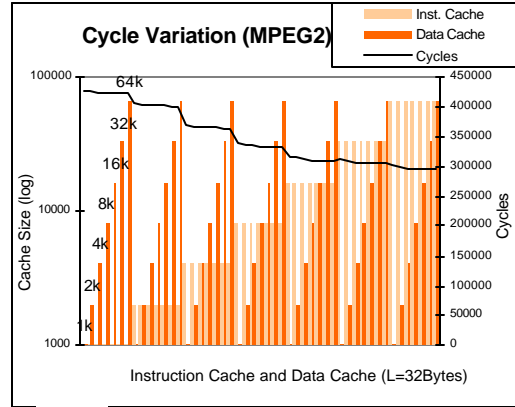
(a)



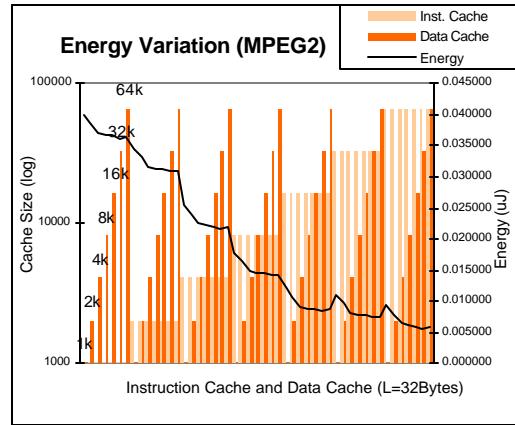
(b)

Figure 4. EPIC encoder. (a) Cycle variation (b) Energy variation for different instruction cache sizes and data cache sizes.

From these two examples, we see that for some cases (as in EPIC), data cache plays a dominant role, while in other cases (as in MPEG2 encoder), instruction cache plays a dominant role. Thus when the total memory available on chip is constant, more memory should be allocated to the cache that is dominant.



(a)



(b)

Figure 5. MPEG2 encoder. (a) Cycle variation (b) Energy variation for different instruction cache sizes and data cache sizes.

## 7. Search Space Reduction in Data Cache – Instruction Cache Exploration

In this section, we briefly describe our procedure to determine the cache configuration – instruction cache size and data cache size – such that the energy consumption is minimum. We assume that the total cache size is given. The procedure first identifies which cache plays the dominant role and assigns the dominant cache the larger size. We assume that the data cache and instruction cache sizes are powers of 2. To show that the dominant cache in different for different programs, we show the energy variation of the instruction and data cache for L=32 for EPIC, MPEG2,

and JPEG examples in Figure 6. Note that the data cache is dominant in EPIC, the instruction cache is dominant in MPEG2, and both instruction cache and data cache are equally important in JPEG. To determine which cache is dominant, we compare the miss rates of the instruction cache and the data cache at  $C_{\min}$  and  $C_{\max}$ . The dominant cache is the one with a comparatively high miss rate. If the miss rates are comparable, then none of the caches can be considered dominant.

If the instruction cache is dominant, then its size  $C_I = \min\{C_{\max}-1, C_{\text{dip}}\}$ , where  $C_{\text{dip}}$  is the cache size at which the energy curve for instruction cache dips. The data cache size  $C_D = \min\{C_{\max}-C_I, C_{\text{Ddip}}\}$ , where  $C_{\text{Ddip}}$  is the cache size at which the energy curve for data cache dips. For instance, for MPEG2 example, if  $C_{\max} = 18\text{K}$ , then  $C_I = \min\{17, 64\} = 17\text{K} \rightarrow 16\text{K}$  and  $C_D = \min\{18-16, 32\} = 2\text{K}$ .

Similarly, if the data cache is dominant, then  $C_D = \min\{C_{\max}-1, C_{\text{Ddip}}\}$  and  $C_I = \min\{C_{\max}-C_D, C_{\text{Idip}}\}$ . If both the instruction cache and the data cache are equally important, then the cache sizes are chosen to be as equal as possible. For instance, for JPEG, if  $C_{\max} = 12\text{K}$ , then we consider (i)  $C_I = 8\text{K}$ ,  $C_D = 4\text{K}$  and (ii)  $C_I = 4\text{K}$ ,  $C_D = 8\text{K}$ , and choose the configuration with minimum energy.

## 8. Conclusion

In this paper, we have described a memory design and exploration procedure for low power embedded systems. Before invoking the memory exploration procedure, we apply high-level transformations such as loop transformations to reduce the number of off-chip memory accesses. The exploration procedure chooses a cache configuration (cache size, line size) that satisfies the area, time and energy constraints. When the instruction cache is assumed to be of constant size, we determine the minimum energy data cache configuration given the area and time bound, and the minimum time data cache configuration given the area and energy bound. When both the instruction cache and the data cache sizes can be varied, we determine the minimum energy cache configuration that satisfies the area bound.

We validate our exploration procedure by performing simulation experiments on the MediaBench examples. We are currently looking at extending our procedure to handle cache performance enhancement procedures such as block buffering,

subbanking, using filter cache, and placement of data in main memory.

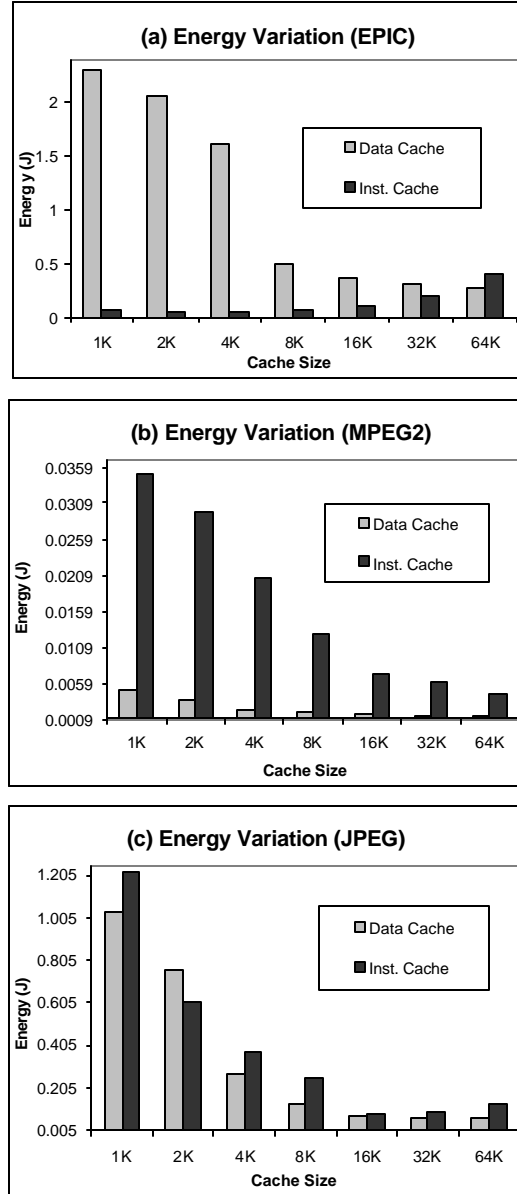


Figure 6. Energy variation for (a) EPIC, (b) MPEG2, and (c) JPEG for different instruction cache sizes and data cache sizes.

## Acknowledgements

The authors would like to thank Shashikiran Tadas of ASU for help with the SimpleScalar experiments. This work was carried out at the National Science Foundation's State/Industry/University Cooperative Research Center for Low Power Electronics (CLPE). CLPE is supported by the NSF, the State of Arizona and the following companies: Burr Brown, Conexant, Gain Technology, Intel Corporation, Medtronic, Microchip, Motorola, Raytheon, Texas Instruments and Western Design Center.

## 9. References

- [1] F. Catthoor, F. Franssen, S. Wuytack, L. Nachtergaele, and H. De Man, "Global Communication and Memory Optimizing Transformations for Low Power Signal Processing Systems", *Workshop on VLSI Signal Processing*, La Jolla, CA, Oct. 1994.
- [2] F. Catthoor, S. Wuytack, E. D. Greef, F. Balasa, L. Nachtergaele, and A. Vandecappelle, *Custom Memory Management Methodology – Exploration of Memory Organisation for Embedded Multimedia System Design*, Kluwer Academic Publishers, June 1998.
- [3] Wen-Tsong Shiue and Chaitali Chakrabarti, "Memory Design and Exploration for Low Power, Embedded Systems," *IEEE Workshop on Signal Processing Systems: Design and Implementation*, Taipei, Taiwan R.O.C., Oct. 1999.
- [4] M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and W. Ye, "Influence of Compiler Optimizations on System Power," *37th IEEE/ACM Design Automation Conference*, 2000, pp. 304-307.
- [5] Wen-Tsong Shiue and Chaitali Charabarti, "Memory Exploration for Low Power, Embedded Systems," *36th IEEE/ACM Design Automation Conference*, New Orleans, LA, June 1999, pp.140-145.
- [6] P. R. Panda, N. D. Dutt, and A. Nicolau, "Architectural Exploration and Optimization of Local Memory in Embedded Systems," *International Symposium on System Synthesis*, Antwerp, Sept. 1997.
- [7] P. R. Panda, N. D. Dutt, and A. Nicolau, "Memory Data Organization for Improved Cache Performance in Embedded Processor Applications," *ACM Transactions on Design Automation of Electronic Systems*, vol. 2, no. 4, Oct. 1997.
- [8] P. R. Panda, N. D. Dutt, and A. Nicolau, "Local Memory Exploration and Optimization in Embedded Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 1, January 1999.
- [9] S. Dutta, W. Wolf, and A. Wolfe, "A Methodology on Evaluate Memory Architecture Design Tradeoffs for Video Signal Processors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.8, no.1, Feb. 1998.
- [10] D. Kirovski, C. Lee, M. Potkonjak, and W. Mangione-Smith, "Application-Driven Synthesis of Core-Based Systems," *IEEE International Conference on Computer Aided Design*, 1997, pp. 104-107.
- [11] Y. Li and J. Henkel, "A Framework for Estimating and Minimizing Energy Dissipation of Embedded HW/SW Systems," *35th IEEE/ACM Design Automation Conference*, 1998, pp. 188-193.
- [12] T. Givargis, J. Henkel, and F. Vahid, "Interface and Cache Power Exploration for Core-based Embedded System Design," *IEEE International Conference on Computer Aided Design*, 1999.
- [13] T. Simunic, L. Benini, and G. De Miceli, "Cycle-accurate Simulation of Energy Consumption in Embedded Systems," *36th IEEE/ACM Design Automation Conference*, 1999, pp. 867-872.
- [14] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. "The Design and Use of SimplePower: a Cycle-accurate Energy Estimation Tool," *37th IEEE/ACM Design Automation Conference*, 2000.
- [15] D. F. Bacon, S. L. Graham, and O. J. Sharp, "Compiler Transformations for High-Performance Computing," *ACM Computing Surveys*, vol. 26, no. 4, Dec. 1994.
- [16] M. E. Wolf and M. S. Lam, "A Loop Transformation Theory and An Algorithm to Maximize Parallelism," *IEEE Transactions on Parallel and Distributed Systems*, Oct. 1991.

- [17] Michael Wolfe, *High Performance Compiler for Parallel Computing*, Addison Wesley, Redwood City, CA 1996.
- [18] M. E. Wolf and M. Lam, "A Data Locality Optimizing Algorithm," *Proceedings of the SIGPLAN'91 Conference on Programming Language Design and Implementation*, June 1991, pp. 30-44.
- [19] Utpal Banerjee, "Unimodular Transformations of Double Loops," *Proceedings of the 3<sup>rd</sup> Workshop on Languages and Compilers for Parallel Computing*, Aug. 1990.
- [20] V. Sarkar and R. Thekkath, "A General Framework for iteration-Reordering Loop Transformations (Technical Summary)," *Proceedings of ACM SIGPLAN'92 Conference on Programming Language Design and Implementation*, San Francisco, CA, June 1992, pp. 175-187.
- [21] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "Mediabench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems," *In Proceedings of the 30th International Symposium on Microarchitecture*, 1997.
- [22] T. Austin, D. Burger, S. Keckler, "SimpleScalar Simulation Tools for Microprocessor and System Evaluation," <http://www.simplescalar.org>, 2000.
- [23] Wen-Tsong Shiue, Sathish Udayanarayanan and Chaitali Chakrabarti, "Data Memory Design and Exploration for Low Power Embedded Systems", revised version submitted to ACM Transactions on Design Automation of Electronic Systems, Aug. 2000.
- [24] M. B. Kamble and K. Ghose, "Analytical Energy Dissipation Models for Low Power Caches," *International Symposium on Low Power Electronics and Design*, 1997.
- [25] 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.
- [26] 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*, 1995, pp. 63-68.