

Algorithmic Techniques for Memory Energy Reduction

Mitali Singh and Viktor K. Prasanna

Department of Computer Science,
University of Southern California,
Los Angeles, CA-90089, USA
{mitalisi, prasanna}@usc.edu

Abstract. Energy dissipation is a critical concern for battery-powered embedded systems. Memory energy contributes significantly to overall energy in data intensive applications. Low power memory systems are being designed that support multiple power states of memory banks. In low power states, energy dissipation is reduced but time to access memory is increased. We abstract an energy model for the memory system and exploit it to develop algorithmic techniques for memory energy reduction. This is achieved by exploring the structure and data access pattern of a given algorithm to devise memory power management schedules. We illustrate our approach through two well-known embedded benchmarks - Matrix Multiplication and Fast Fourier Transform. The optimality of our schemes is discussed using information theoretic lower bounds on memory energy. Simulations demonstrate that significant energy reduction can be achieved by using our approach over state-of-the-art implementations.

1 Introduction

Due to the explosive growth of portable, wireless devices and battery-operated embedded systems, energy efficiency has become a critical concern for designers today. Design technologies at all levels of abstraction are evolving with the common goal of energy reduction. The significance of high level analysis in the design cycle cannot be underestimated. It is rapid, fairly accurate, and platform independent. Moreover, decisions made at higher levels are likely to have a larger impact on energy reduction than those at the lower levels of abstraction. We have thus been motivated to explore energy efficient design and analysis methodologies at the algorithmic level.

Until recently, majority of the research focus has been on optimizing processor energy by exploiting techniques such as dynamic voltage scaling [26] [17], precision management, and IPC management [14]. Advancements in processor technology have led to development of low power processors such as the XScale PXA250 [15] that dissipate with less than 1nJ per instruction. The next challenge lies in reduction of memory energy which accounts for as much as 90% of the overall system energy for CPU systems with peripherals [7]. For several wireless applications implemented on the pico radio [25], more than 50% of the overall energy is dissipated in the memory [27].

Advanced memory technologies such as the Mobile SDRAM [21] support several low power features such as multiple power states, bank/row specific activation, and partial array refresh (PASR). Our goal is to abstract the advanced features of the state-of-the-art memory systems and exploit them to design algorithms that reduce memory energy dissipation. This is achieved by designing algorithms optimized for reduced number of memory accesses, and implementing energy optimal memory power management schedules. We analyzed several benchmark kernels from the EDN Embedded Microprocessor Benchmark Consortium (EEMBC) [20] and the freely available Mibench Benchmarks [19], which are discussed in [34]. In this paper we present our results for Matrix Multiplication and Fast Fourier Transform(FFT). We discuss information theoretic lower bounds for energy dissipation in the memory and use them to guide our algorithm design. Simulation results (see Section 5) demonstrate that using our techniques significant energy reduction can be achieved.

Rest of this paper is organized as follows. In Section 2 we discuss related research. A high-level energy analysis is presented in Section 3 and lower bounds on memory energy dissipation are discussed. An energy efficient memory architectural design is proposed in Section 3.3. Optimal memory activation schedules for some well-known kernels are discussed in Section 4. Our simulation framework and results are presented in Section 5. Finally, we conclude in Section 6.

2 Related Work

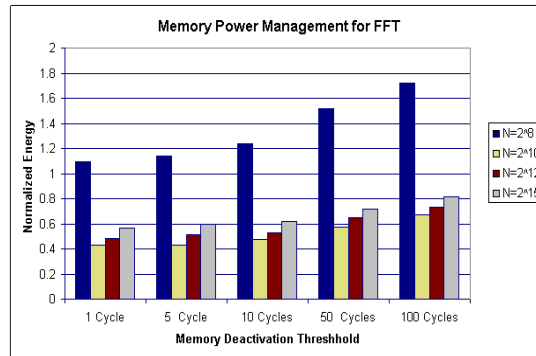


Fig. 1. Implicit Power Management

Memory technologies such as the SDRAM reduce energy dissipation by switching to lower power states based on the time of inactivity in the current state. We call this *implicit power management* as the memory power state switching is defined only by the duration of inactivity (wait time) of the memory bank. Several researchers have exploited implicit power management for memory energy reduction. Energy efficient page allocation techniques for general-purpose processors have been proposed in [16]. In [3], the authors investigate array allocation schemes to minimize memory energy dissipation. A large number of benchmark applications have been analyzed in [9] to find the optimal wait time for memory power state switching. The conclusion drawn suggests that memory

should immediately transition to a lower power state when inactive. However, our simulations show that this is not the optimal policy always. Fig. 1 illustrates normalized values (w.r.t. case with no power management) for memory energy dissipation for FFT with implicit power management for various wait times. For smaller size FFT ($n = 2^8$) all the policies result in increased energy dissipation. Energy reduction is observed for larger size problems with optimal wait-time as one cycle. Thus, we propose algorithm specific power management of the memory. In this paper, we focus on design of optimal power management schemes for the memory based upon the structure and access pattern of the algorithms.

The AMRM project [1] focuses on adaptation of the memory hierarchy to reduce latency and improve power efficiency. Techniques such as off-chip memory assignment, set associativity and tiling to improve cache performance and energy efficiency have been investigated in [29]. The memory segmentation problem has been shown to be NP complete in [10]. Several researchers [24] [18] [22] [6] [4] have explored memory organization and optimization for embedded systems. Their approach has been summarized below.

Memory architecture customization: Memory allocation and memory bank customization problems deal with selection of memory parameters such as type, size, and number of ports. Memory building blocks and organization for application customized memory architectures are design synthesis problems and are not in the scope of this paper. We focus on algorithmic optimizations rather than hardware customizations.

Application specific code and data layout optimizations: Application specific platform-independent code (loop) and data flow transformations have been proposed to optimize the algorithm's storage (memory, cache) and transfer requirements. This is followed by hardware customization (such as selection of a smaller cache) to reduce energy dissipation. Our approach is reverse as we optimize algorithms for a fixed architecture.

Scratch pad memory: On-chip memory is partitioned into data cache and scratch pad. Split spatial-temporal caches have been proposed to improve spatial and temporal data reuse in the cache. In our analysis, we propose use of a small (of the order of cache line size) memory buffer to aid in implementation of power management schemes. A partial bank of a Mobile SDRAM that can be power controlled independently can be used as buffer. Scratch pad memory can act as a buffer but at the expense of smaller cache size.

Our approach aims at optimizing memory energy dissipation by improving the memory access pattern of the algorithms. Higher data reuse reduces the number of memory accesses. Algorithm directed power management schemes are described within the algorithm to dynamically alter the memory power states. Buffering, prefetching and blocking strategies are used to reduce energy and latency overheads for memory power management. Memory energy is analyzed using a simple, high-level, memory energy model that considers memory to be organized as multiple banks that can be power controlled independently. Since we assume, the architecture to be fixed, memory parameters (such as no of ports, banks, interconnect bandwidth) are considered constant.

3 Memory Energy

We define a high level model for memory energy analysis of algorithms and discuss lower bounds on memory energy dissipation of algorithms.

3.1 Our Energy Model

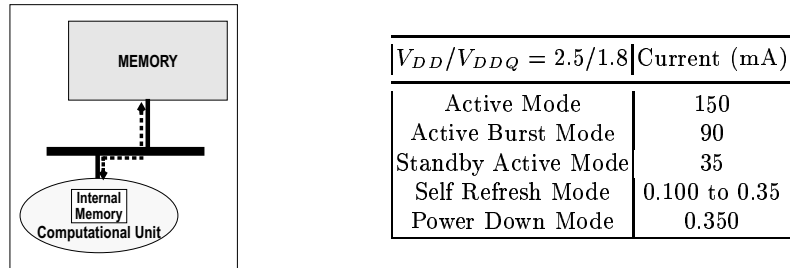


Fig. 2. (a) System Model

(b) 128Mb Mobile SDRAM

We consider our system to comprise of a computational unit with an internal memory (e.g. cache) connected to a memory unit over an interconnect (Fig. 2(a)). We abstract the low power features of the memory systems as discussed below.

- Memory has multiple power modes (states). Fig. 2(b) illustrates the the current drawn in various power modes of the SDRAM [21] memory system.
 - Active:** Memory can be read or written to only in the active mode. Power dissipation is the highest in this mode. The energy dissipation is higher when the memory is being accessed and lower when it is in standby active mode. We do not consider the burst access mode in our current analysis.
 - Refresh (idle/inactive):** Power dissipation in this mode is low (zero for analysis). Data is preserved in this mode. Memory access in this mode results in higher latency as the memory must transit to the Active mode.
 - Power Down:** The power dissipation in this mode is the least but data is not preserved. For our analysis, we do not consider transition to this mode to ensure that the data is not lost.
- Memory is organized as banks. Each bank can be placed in any power mode independent of the other banks. For example, Mobile SDRAM memory supports bank and row specific activation and deactivation (precharge) and partial-array refresh (PASR).
- Transition of memory from one power mode to another incurs energy and time overheads.
- Memory power management can be controlled through software.

The total memory energy $E(N)$ for problem size N is defined as the sum of the memory access energy $E_a(N)$, the data storage energy $E_s(N)$, and state transition overheads $E_p(N)$. The memory access energy is proportional to the memory data traffic. It also depends on parameters such as the load capacitance, frequency and voltage of the memory I/O, but we assume these to be fixed. The data storage energy is a function of the memory size and the time for which it is active. Energy dissipation in the idle state is considered to be negligible. Let K_A

denote the memory access energy cost per unit of data, K_s be the storage energy cost per unit of data per unit time, and K_p represent the energy overheads for each power state transition. The memory energy is defined as follows.

$$E(N) = E_a(N) + E_s(N) = K_a \times C(N) + K_s \times S(N) \times A(N) + K_p \times P(N)$$

Here, $C(N)$ represents the total number of memory accesses and $S(N)$ is the space complexity of the algorithm. We define memory *activation complexity* $A(N)$, as the time for which memory is in the active mode. For an algorithm of time complexity $T(N)$, if memory is active for time fraction α then $A(N) = \alpha T(N)$. For conventional systems that do not support memory power management $A(N) = T(N)$. $P(N)$ denotes the number of power mode transitions of the memory banks.

3.2 Lower Bounds on Memory Energy

Algorithms have been extensively analyzed and optimized in the past using the I/O complexity model [12]. This model measures performance as a function of the number of I/O operations, and abstracts systems where the latencies involved in accessing external memory are much larger as compared to internal processing. Prior results on I/O complexity are of significant interest as they can be used to determine lower bounds for memory energy dissipation.

Every computational unit in an embedded system has an internal memory of fixed size as illustrated in Fig. 2(a). The interaction between the internal memory and the (external to computational unit) on-chip memory, can be captured by using the I/O model for analysis. We defined $C(N)$ as the number of memory accesses which is the asymptotically same as the I/O complexity $T_{I/O}(N)$ [28].

Theorem 1. *A lower bound on memory energy dissipation $E(N)$ for an algorithm with problem size N and I/O complexity $T_{I/O}(N)$ is given by $\Omega(T_{I/O}(N))$.*

Proof: We know $E(N) = K_a \times C(N) + K_s \times S(N) \times A(N) + K_p \times P(N)$ (see Section 3.1). The memory must remain active for at least the time it is accessed. Thus if the access latency is l cycles, $A(N) = \Omega(l \times C(N))$. $S(N)$ represents the space complexity of the algorithm or the size of memory required to store data. $S(N) = k \times M$, where M is the smallest segment (bank incase of multi-banked architecture) that can be power controlled independently. Each memory access requires at least one memory bank to be active. In the best case scenario memory power management overheads $P(N)$ are negligible. Thus,

$$\begin{aligned} E(N) &= K_a \times C(N) + K_s \times S(N) \times A(N) \\ &= \Omega(K_a \times C(N) + K_s \times M \times C(N)) = \Omega(C(N)) = \Omega(T_{I/O}(N)) \quad \square \end{aligned}$$

Note that the above lower bound on memory energy dissipation is independent of how the computation is performed (on RISC, DSP, FPGA). For a given kernel mapped on a single computation unit, it depends only on the size of the input data and the size of the internal memory of the computational unit.

3.3 Memory Architecture Design

Memory energy can be optimized by introduction of a small memory buffer of size B between the computation unit and the memory unit. Data is transferred

from memory to buffer before it is accessed by the processor. The buffer is active all the time, but it permits the **much larger sized** memory module to remain *idle* for a longer time. The buffer is placed near the memory modules with a simple data transfer policy to/from memory. Thus, the latency for data transfer between the memory and the buffer is much lower than a memory access from a computational unit. The latter (for example a PCI) involves complex scheduling and bandwidth allocation. We consider memory to buffer latency to be unity while the buffer to computational unit latency is l .

A simple power management scheme is described as follows. A memory bank is activated only when there is a data transfer required from/to the buffer. The memory access energy and activation overheads are $O(T_{I/O}(N))$, which is optimal. Next, consider the memory storage energy $E_s(N)$. In absence of the buffer, the entire memory must remain active while it is accessed. Hence by definition $E_s(N) = K_s \times S(N) \times A(N)$, where $A(N) = \Omega(l.C(N))$ and $A(N) = O(T(N))$. The power management scheme described above reduces memory activation time to $A(N) = 1 \times C(N) = O(T_{I/O}(N))$. Thus, $E_s(N) = K_s \times (M \times T_{I/O}(N) + B \times T(N))$. Any scheduled data transfer from buffer to computational unit need not be of size larger than the line size of the internal memory of the computational unit. Hence, it is sufficient to have $B = O(L)$, where L is the cache size. Cache size is typically in Kilobytes whereas memory size ranges from Megabytes implying $M \gg B$. Therefore, energy dissipation in the buffer can be ignored as compared to the memory, and $E_s(N) = K_s \times M \times T_{I/O}(N) = \Omega(T_{I/O}(N))$. Thus, $E(N) = O(T_{I/O}(N))$. \square

Remark 1. Since the memory access latency is reduced from l to 1, memory storage energy is reduced by a factor of l by using the buffer even in the best case scenario when $A(N) = O(l.C(N))$ in absence of buffer.

Remark 2. The memory buffer increases memory access latency from l to $l + 1$ as data is fetched to buffer before it is transferred to (from) memory. However, data prefetch can be utilized to schedule transfer of data into the buffer before it is accessed by the computational unit.

4 Memory Energy Optimization

Memory energy optimization involves designing algorithms that reduce memory-processor data transfers and implement energy optimal memory activation schedules. Our approach can be summarized as follows:

- Understanding the memory access behavior of the kernel algorithm.
- Minimizing the number of memory accesses required by optimizing the cache complexity of the algorithm. For example, data layout can be altered.
- Designing power management schedule based on the memory access pattern.
- Reducing the power management overheads.

It is important to note that our analysis holds for all computational units with an internal memory. For our simulations, we consider a RISC computational unit. The data cache (internal memory for RISC) is of size $O(L)$.

4.1 Matrix Multiplication

Matrix Multiplication is an embedded automotive/industrial benchmark [20]. *Baseline (MMS)*: Consider multiplication of two $N \times N$ matrices A and B to produce matrix D. Computation complexity of this algorithm is $O(N^3)$. Computation of each element of D requires the corresponding row from A and column from B to be fetched into the cache. There is no data reuse. The number of memory to cache transfers $C(N)$ is at least $2N \times N^2 = O(N^3)$. $3N^2$ elements need to be stored in memory. The memory is active all the time. The storage energy is $O(A(N) \times S(N))$, which is $O(N^3) \times O(N^2) = O(N^5)$. Thus, memory energy is given by $O(C(N)) + O(A(N) \times S(N)) = O(N^3) + O(N^5) = O(N^5)$.

Blocked (MMB): We investigate an alternative implementation using blocked (tiled) data layout with block size b . The block should be able to fit into the cache and thus, $b^2 = O(L)$. The arithmetic complexity for this implementation remains $O(N^3)$. Data is fetched and operated upon as blocks, resulting in higher data reuse in the cache. To compute b^2 elements of D, we require $2N \times b$ transfers. $C(N)$ for this algorithm is reduced to $O(2N^3/b)$, which is an improvement by a factor of b . Since the memory modules remain active for the entire duration, the memory energy is given by $O(C(N)) + O(A(N) \times S(N)) = O(N^3/b) + O(N^3 \times 3N^2) = O(N^5)$. The memory access energy is decreased with reduced data traffic, but the storage energy remains same.

Memory Power Management (MMBPM):

As the next level of optimization, we reduce the activation time for the memory. This is achieved by explicitly scheduling the memory power state transitions (see Fig. 5) based on the memory access pattern of the algorithm. The data access timing diagram is illustrated in Fig. 3. After each successive fetch of $O(b^2)$ from matrix A and B computation of $O(b^3)$ takes place. Energy can be saved by deactivating the memory modules for this duration. Thus, memory activation time $A(N)$ is reduced to $O(N^3/b)$. The storage energy is reduced to $O(N^3/b) \times O(N^2) = O(N^5/b)$.

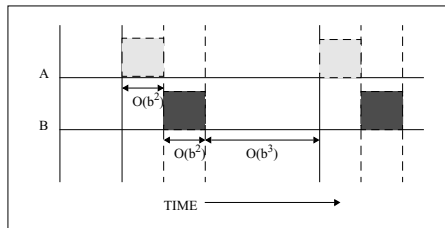


Fig. 3. Memory Access Schedule

Memory Buffer (MMBBUF): Using a memory buffer and blocking, the energy of the system can be decreased to $O(N^3/b)$ as described in Section 3.3. Since we can predict which block will be required for computation next, data prefetch [2] can be used to hide buffer to memory data transfer latencies.

The I/O complexity as analyzed by Hong and Kung [12] is given by :

Lemma 1. *For matrix multiplication of two $N \times N$ matrices on a processor with L words of memory, the I/O complexity bound is given by $\Omega(N^3/\sqrt{L})$.*

Using Theorem 1 and Lemma 1 it follows,

Theorem 2. *Memory energy dissipation for Matrix Multiplication of two $N \times N$ matrices is $\Omega(N^3/\sqrt{L})$, where L is the cache size.*

Remark 3. We discussed an energy optimal implementation of Matrix multiplication (MMBBUF). For $b = \sqrt{L}$, where L represents the cache size, the memory energy are reduced to $\Omega(N^3/\sqrt{L})$. This is the optimal as shown by Theorem 2.

4.2 Fast Fourier Transform

Fourier Transforms are used in digital signal processing in several embedded applications such as the Asynchronous Digital Subscriber Line (ADSL) where data is converted from the time domain to the frequency domain to reduce error rate. The problem size in Mibench is 2^{15} and we use the same for analysis. We examine the Cooley-Tukey algorithm, which involves recursive decomposition of a larger FFT into smaller sub-problems that can be solved efficiently. For computation of an N -point FFT, the computation complexity of this algorithm is $O(N \log_2 N)$. Consider computation of an $N1 \times N2$ -point FFT. The data layout in the memory can be analyzed in terms of data *stride*, which is defined as the distance between data blocks that are accessed successively. The stride of data for the $N2$ -point computation is determined by the $N1$ -point FFT computation. The cache performance is poor if the stride is too large. Dynamic Data Layouts [23] [31] can be used to improve the cache performance. These have been exploited to improve the time performance [23].

Baseline (FFT): As our baseline case, we chose an optimal implementation from the FFTW library and analyze its cache behavior as a function of the data stride. For small strides, there is a large spatial locality in the data. Therefore, there are only compulsory cache misses, and for an N -point FFT, $O(N/b)$ misses are encountered. Here b is the cache block size. However, if the stride is very large, cache performance is very poor due to increased number of conflict misses. A cache miss could incur for every data access required for the FFT computation, which is $O(N \log_2 N)$.

Dynamic Data Layouts (FFTD): Data reorganization could be performed prior to every FFT computation to reduce the data stride. However, this itself dissipates a lot of energy as it could involve $O(N)$ memory accesses. Therefore, prior to each FFT computation, a tradeoff is performed between the data reorganization overheads involved and the acceptable data stride. The FFT decomposition strategy is selected taking this tradeoff into consideration. Significant energy reduction is achieved by improving the cache performance. Details of the algorithm are discussed in [34].

Memory Power Management (FFTDPM): Conventional computation of FFT follows the following sequence of operations. Two elements are fetched from memory, operated on and the result is placed back in the memory. This approach requires the memory to be active all the time. By increasing the interval

between successive data fetches, we can exploit memory power management for energy reduction. This is achieved by block computation. FFT is computed over blocks of size $h = O(L)$, where L is the cache size. h elements are fetched, FFT is computed over this block and the result is placed back. This involves only $O(N \log_h N)$ data fetches. Moreover, it permits us to schedule the memory power state switching. Each time h elements of data are fetched and computed upon for $O(h \log h)$ time before the next data transfer is scheduled. Memory is activated and deactivated based on this schedule. The memory activation time is reduced from $O(N \log_2 N)$ to $O(N \log_2 N)/(\log_2 h)$, which results in improvement of storage energy by $O(\log_2 h)$.

Memory Buffer (FFTDBUF): Next, we utilize the memory buffer along with data prefetch. The storage energy in all the previously discussed implementations is $A(N) \times S(N)$, which is $O(N^2 \log_h N)$. Using a memory buffer, we can reduce this to $O(N \log_h N)$ (see Section 3.3).

Hong and Kung [12] have shown the following result:

Lemma 2. *The I/O time for computing an N -point FFT on a processor with L words of memory is at least $\Omega(N \log_2 N / \log_2 L)$.*

From Theorem 1 and Lemma 2, it follows,

Theorem 3. *For computing an N -point Fast Fourier Transform, the lower bound for memory energy is $\Omega(N \log_2 N / \log_2 L)$, where L is the cache size.*

Remark 4. The energy reduction techniques discussed above achieve an energy optimal implementation of FFT. The optimal bound for memory energy for our implementation is $\Omega(N \log_2 N / \log_2 L)$.

5 Simulations

Energy estimation for the algorithms described above is challenging as currently there is no hardware or middle-ware support, or simulators that support algorithm directed power management. Thus, we designed a high level energy estimation framework for fast and yet fairly accurate energy estimation of algorithms.

5.1 Simulation Framework

The simulator is based upon instruction level analysis, where cost of each instruction depends on the power state of the system. The choice of this level of abstraction is based on several reasons. The foremost being the speed without much loss in accuracy. Moreover, it is a suitable abstraction from an algorithm designer’s perspective. An algorithm description augmented with the power management schedule (see Fig. 5) is supplied as input to the simulator. The SimpleScalar Toolset [5] is configured for the chosen architecture and modified to provide an instruction execution trace with an embedded power schedule. The trace consists of a limited set of instructions (7 currently). The classification is based on their energy costs.

- *State*: The State instruction permits the designer to change the power state of the system by changing the power mode of any component. For example the memory can be placed in a low power inactive mode.
- *Compute*: This represents all processor only instructions that do not require any cache or memory access. Measurements [30] have demonstrated that there is little power variation among these instructions.
- *ReadCache and WriteCache*: Data is accessed from the cache.
- *ReadMem and WriteMem*: A cache miss occurs resulting in a memory access.
- *Prefetch*: Data is prefetched into the buffer. Energy costs are similar to *ReadMem* but there is no latency in fetching the data.

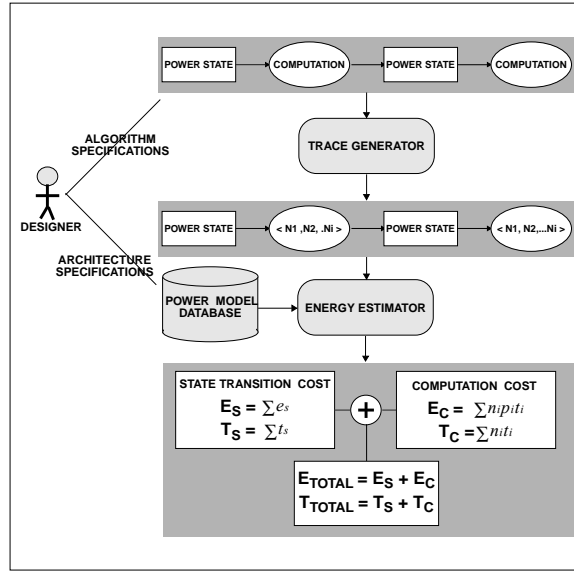


Fig. 4. Simulation framework

Each instruction has an associated power and time cost depending on the power state of the system. The *State* instruction denotes change in the power state of the memory and accounts for the state transition overheads. A profile is maintained for power dissipation in each architecture component to identify hot spots. We have modeled the Itsy Pocket Computer for which detailed measurements are available [35]. Currently we model only power modes of the memory and the switching clock enabled/disable mode of the processor. The Itsy measurements do not profile state transitions. The LART [26] measurements are used to obtain overheads for changing the processor mode. The memory activation/deactivation latency is considered to be one cycle.

The goal of this simulator is to guide algorithm design by identifying the trend. It cannot guarantee high accuracy due to several reasons. It does not simulate the dynamic effects of changing the power state. For example, consider

a scenario where memory is inactive. The trace shows a cache miss. There is a delay associated with memory activation. This could have resulted in more cache misses in a running system. Since our trace is pre-computed, such effects are not accounted.

The accuracy of the simulator depends on the power models incorporated. Some of the values have been approximated when measurements are not available. For example, the power during a state transition is approximated as the average power between previous and next state. Variation in input data may change the switching activity. We assume an average power cost for each instruction. We have used benchmark data when available, and randomly generated data otherwise. The framework is fast, modular and sufficiently accurate for algorithmic analysis. Once some algorithm designs are identified to be energy efficient they can be tested on lower level simulators or implementation boards for higher accuracy.

```

TransformNtoB(A, Ab, k, m, kblk, mblk)
Tr_NtoB(B, Bb, n, k, nblk, kblk);
TransformNtoB(C, Cb, n, m, nblk, mblk);
for(jb=jj=0;jb<nblk;jb++){
  jbs = ((jj+=BS) < n) ? BS: n-jj+BS;
  for(lb=ll=0;lb<kblk;lb++){
    lbs = ((ll+=BS) < k) ? BS: k-ll+BS;
    bpt = &(Bb(jb,lb));
    for(ib=ii=0;ib<mblk;ib++){
      ibs = ((ii+=BS) < m) ? BS: m-ii+BS;
      a = &(Ab(ib,lb)); t = &(Cb(ib,jb));
POWERMG(STATE MEMORY MODE 0)
      for(i=0;i<ibs;i++){ b = bpt;
        for(j=0;j<jbs;j++){ temp_c=0;
          for(l=0;l<lbs;l++){
            temp_c += a[l] * b[l];
            t[j] += temp_c; b += BS;}
          a += BS; t += BS;}
POWERMG(STATE MEMORY MODE 0)}}}
      TransformBtoN(Cb, C, n, m, nblk, mblk);
    }
  }
}

```

Fig. 5. Sample Code

5.2 Simulation Results

Matrix Multiplication: We simulated matrix multiplication for $n = 32$ and $n = 64$ and the results are illustrated in Fig. 6. Note that to improve the clarity of the figure we have scaled down the results for $n = 64$ by a factor of 8. The energy reduction is incremental with problem size. We consider $n = 64$. Our algorithm using blocking (MMB) with block size 16 reduced energy dissipation by 93.6% keeping memory active all the time. This reduction was achieved due to improved cache behavior and thus the execution time also decreased. We examined the effect of implicit power management (MMSP) on the two algorithms. We assumed memory becomes idle if not accessed for 1 cycle. The energy of the conventional algorithm was reduced by 57.4%, but the execution time was increased due to memory activation overheads. The same policy applied to our blocked algorithm (MMBP) reduced energy by 96.4% with no increase in time. Explicit power management (see Fig. 5)) (MMBPM) reduced the number of accesses to the memory when it was inactive. Memory energy was reduced by 96.43%. Memory power management in presence of a memory buffer (MMB-BUF) reduced memory energy by 37% over MMBPM. Thus, an overall memory energy reduction by 97.7% was achieved with no increase in execution time.

Fast Fourier Transform: Simulation results for FFT are illustrated in Fig. 6. We have scaled down the results for large n to improve clarity. Data reorganization (FFTD) and implicit power management (FFTP) are only beneficial for large size problems. We observe an increase in energy for both these techniques

for $n = 2^8$ due to high overheads. For larger problems, energy is reduced and the improvement is proportional to problem size. Implicit power management (FFTP) reduces energy by 57%. For $n = 2^{12}$, we observe energy reduction 5% by using our algorithm (FFTD) without power management, 59.7% using implicit power management (FFTD), 60% using memory power management (FFTDPM) and 70% using memory buffer and power management (FFTDDBUF).

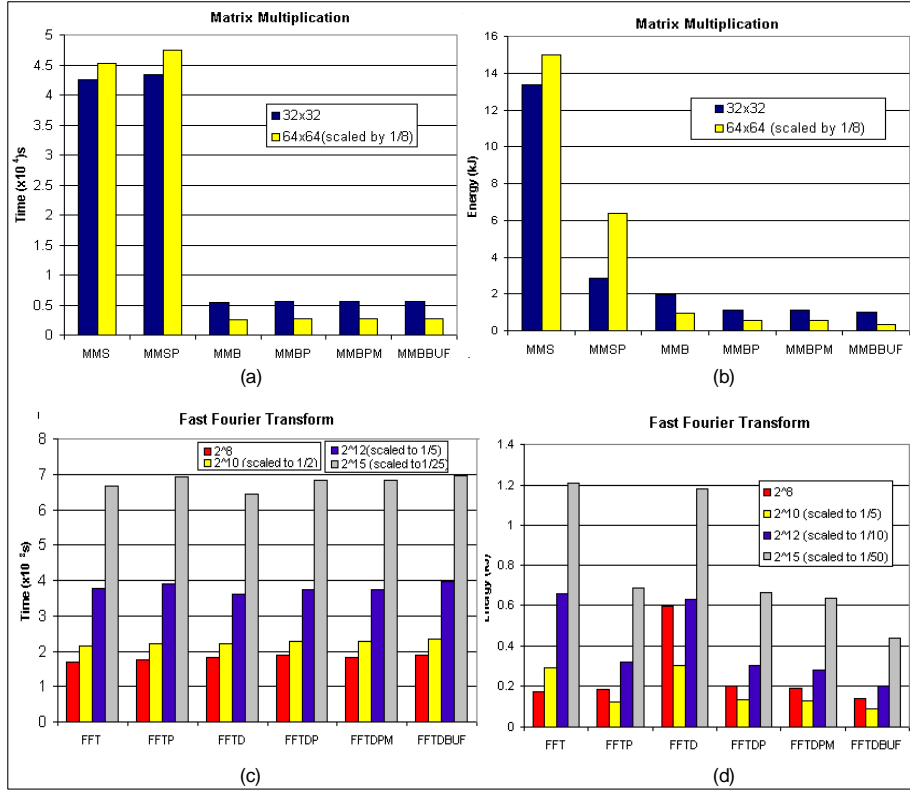


Fig. 6. Simulation results

6 Conclusion

In this paper, we presented algorithmic techniques for memory energy reduction by reducing data traffic and design of efficient power management schedules. Note that reduction in data traffic also decreases energy dissipation over the (high capacitance) interconnect. Currently, we do not exploit all the features of the memory such as lower energy dissipation by accessing data in a burst (see Fig. 2(b)), which will be investigated in our future work. We will also integrate other power management schemes in our analysis such as the DVS for the processor to understand the interactions between the processor and the memory. For example, slowing the processor may reduce processor energy but increase memory latency and energy.

Acknowledgment

This work is supported by the DARPA Power Aware Computing and Communication Program under contract no. F33615-02-2-4005.

References

1. The AMRM project, <http://www1.ics.uci.edu/~amrm/>
2. T. Alexander and G. Kedem, "Distributed Prefetch-buffer/Cache Design for High Performance Memory systems," Symposium on High-Performance Computer Architecture (HPCA), February 1996.
3. R. Athavale, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "Influence of Array Allocation Mechanisms on Memory System Energy," International Parallel and Distributed Processing Symposium (IPDPS), April 2001.
4. L. Benini, L. Macchiarulo, A. Macii, and M. Poncino, "Layout-Driven Memory Synthesis for Embedded Systems-on-Chip," IEEE Transactions on VLSI Systems, Vol. 10(2), April 2002.
5. D. Burger, T. M. Austin, and S. Bennett, "The SimpleScalar Tool Set, Version 2.0," Technical Report, UW-Madison, 1997.
6. F. Catthoor, K. Danckaert, S. Wuytack, and N. D. Dutt, "Code Transformations for Data Transfer and Storage Exploration Preprocessing in Multimedia Processors," IEEE Design & Test of Computers, Vol. 18(3), pp 70-82, May/June 2001.
7. R. Y. Chen and M. J. Irwin, "Architectural-Level Power Estimation and Design Experiments," ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol. 6(1), pp 50-66, January 2001.
8. N. D. Dutt, "Memory Organization and Exploration for Embedded Systems-on-Silicon," International Conference on VLSI and CAD (ICVC), October 1997.
9. X. Fan, C. Ellis, and A. R. Lebeck, "Memory Controller Policies for DRAM Power Management," International Symposium on Low Power Electronics and Design (ISLPED), August 2001.
10. A. H. Farrahi, G. E. Tellez, and M. Sarrafzadeh, "Memory Segmentation to Exploit Sleep Mode Operation," Design Automation Conference (DAC), June 1995.
11. FFTW, <http://www.fftw.org>.
12. J. W. Hong and H. T. Kung, "I/O Complexity: The Red-Blue Pebble Game," Symposium on Theory of Computing (STOC), May 1981.
13. P. Kirschenhofer, P. H. Prodingler, and W. Szpankowski, "On the balance property of patricia tries: External path length view," Theoretical Computer Science, Vol. 68, pp 1-17, 1989.
14. P. M. Kogge, V. W. Freeh, K. Ghose, N. Toomarian, and N. Aranki, "Morph: Adding an Energy Gear to a High Performance Microarchitecture for Embedded Applications," Kool Chips Workshop, MICRO-33, December 2000.
15. Intel PXA250 Processor, <http://www.intel.com/design/pca/prodbref/298620.htm>
16. A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis, "Power Aware Page Allocation," International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), November 2000.
17. J. Luo and N. K. Jha, "Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems," International Conference on VLSI Design, Jan 2002.

18. T. V. Meeuwen, A. V. Zelst, F. Catthoor, "System-level Interconnect Architecture Exploration for Custom Memory Organizations," International Symposium on Systems Synthesis (ISSS), October 2001.
19. Mibench version 1.0, <http://eecs.umich.edu/jringenb/mibench>
20. EEMBC-Embedded Microprocessor Benchmarking Consortium, <http://www.eembc.org>
21. "Mobile SDRAM Power Saving Features," Technical Note TN-48-10, MICRON, <http://www.micron.com>
22. L. Nachtergaele, F. Catthoor, and C. Kulkarni, "Random-Access Data Storage Components in Customized Architectures," IEEE Design & Test of Computers, Vol. 18(3), pp 70-82, May/June 2001.
23. N. Park and V. K. Prasanna, "Cache Conscious Walsh-Hadamard Transform," International Conference on Acoustics, Speech, and Signal Processing (ICASSP), May 2001.
24. P. R. Panda, N. D. Dutt, A. Nicolau, F. Catthoor, A. Vandecappelle, E. Brockmeyer, C. Kulkarni, and E. D. Greef, "Data Memory Organization and Optimizations in Application-Specific Systems," IEEE Design & Test of Computers, Vol. 18(3), pp 56-68, May/June 2001.
25. PICO Radio, <http://bwrc.eecs.berkeley.edu/Research/>
26. J. Pouwelse, K. Langendoen, and H. Sips, "Dynamic Voltage Scaling on a Low-Power Microprocessor," UbiCom-Tech. Report, 2000.
27. Jan Rabaey, "Piconodes for Sensor Networks," DARPA PACC PI Meeting, 2001.
28. S. Sen and S. Chatterjee, "Towards a Theory of Cache-Efficient Algorithms," Symposium on Discrete Algorithms (SODA), January 2000.
29. W. T. Shiue and C. Chakrabarti, "Memory Exploration for Low Power Embedded Systems," Design Automation Conference (DAC), October 1999.
30. A. Sinha and A. P. Chandrakasan, "JouleTrack - A Web Based Tool for Software Energy Profiling," Design Automation Conference (DAC), April 2001.
31. The SPIRAL Project, <http://www.ece.cmu.edu/~spiral/>
32. W. Tang, A. V. Veidenbaum, and R. Gupta, "Architectural Adaptation for Power and Performance," International Conference on ASIC, October 2001.
33. K. V. Palem, R. M. rabbah, V. J. Mooney III, P. Korlmaz, and K. Puttaswamy, "Power Optimization of Embedded Memory Systems via Data Remapping," CREST Technical report, Georgia Institute of Technology, February 2002.
34. M. Singh and V. K. Prasanna, "Application Directed Power Management for Optimizing Memory Energy Dissipation", Technical Report, EEB-Systems, University of Southern California, 2003.
35. M. A. Viredaz and D. A. Wallach, "Power Evaluation of a Handheld Computer: A Case Study," COMPAQ WRL Research Report, January 2001.