Reducing Energy Consumption of Wireless Sensor Networks through Processor Optimizations

Gürhan Küçük and Can Başaran
Department of Computer Engineering,
Yeditepe University, 34755 Istanbul, Turkey
{gkucuk, cbasaran}@cse.yeditepe.edu.tr

***Abstract*- When the environmental conditions are stable, a typical Wireless Sensor Network (WSN) application may sense and process very similar or constant data values for long durations. This is a common behavior of WSN nodes that can be exploited for reducing their power consumption. This study combines two orthogonal techniques to reduce the energy dissipation of the processor component of the sensor nodes. First, we briefly discuss silent-store filtering *MoteCache*. Second, we utilize Content-Aware Data MAnagement (CADMA) on top of *MoteCache* architecture to achieve further energy savings and possible performance improvements. The complexity increase introduced by CADMA is also compensated by further complexity reduction in MoteCache architecture. Our optimal configuration reduces the total node energy, and hence increases the node lifetime, by 19.4% on the average across a wide variety of simulated sensor benchmarks. Our complexity-aware configuration with a minimum MoteCache size with only four entries not only achieves energy savings up to 16.2% but also performance improvements up to 14%, on the average.**

***Index Terms*- Computer Architecture, Sensor Networks**

I. INTRODUCTION

Recent advances in process technologies and the shrinking sizes of radio communication devices and sensors allowed researchers to combine three operations (i.e. sensing, communication and computation) into tiny devices called wireless sensor nodes. Once these devices are scattered through the environment, they can easily construct data-oriented networks known as wireless sensor networks (WSNs). Today, there are a vast number of application scenarios involving WSNs in business, military, medical and science domains.

The lifetime, scalability, response time and effective sampling frequency are among the most critical parameters of WSNs, and they are closely related to one crucial resource constraint that is very hard to satisfy: the power consumption. The WSN nodes are designed to be battery-operated, since they may be utilized in any kind of environment including thick forestry, volcanic mountains and oceanbeds. Consequently, everything must be designed to be power-aware in these networks.

Small-scale operating systems, such as TinyOS [1], ambientRT [2], and computation-/communication-intensive applications significantly increase the energy consumption of the processor component of WSN nodes. Today, new sensor platforms with 16- [3] and 32-bit [4] processor architectures target more and more power-hungry applications. In [5], the researchers show that the processor itself dissipates 35% of the total energy budget of the MICA2 platform while running *Surge*, a TinyOS monitoring application. In [6], the authors claim similar energy values when running a TinyDB query reporting light and accelerometer readings once every minute. In [7], the researchers find that the energy consumption of the processor/memory component for raw data compression is higher than the energy consumption of raw data transmission. Similarly, today most of the WSN applications avoid extensive computations and choose to transfer raw data to server machines to increase the lifetime of the sensor nodes. On the contrary, our proposed design encourages the WSN application developers to design less centralized applications by distributing the computation work and reducing the network traffic among the nodes.

After observations of the results from our departmental testbed and various simulations, we found two important characteristics of WSN data:

1. Temporal and value locality: Sensor network applications have periodic behavior. Especially, monitoring applications, such as *Surge*, may sense and work on constant data and constant memory locations for long durations. In this study, first we show that we considerably reduce the energy dissipation of the WSN processors by caching commonly used data in a small number of latches (*MoteCache*) [8]. Then, we also show that most of the store instructions in WSN applications are silent (these instructions write values that exactly match the values that are already stored at the memory address that is be-

ing written), and propose to filter them by extending the MoteCache architecture.

2. Common data values: Further, we find that there are some common data values flowing through the datapath. In the second part of this study, we propose a technique called Content-Aware Data MAnagement (CADMA) that exploits this behavior and boosts our energy savings by reducing the read/write energy not only in SRAM but also in register file and the MoteCache. When combined with the silent-store filtering MoteCache technique, we show that we can even increase the performance of the processor as much as 14%.

We begin by presenting a brief summary related work and current state of the art of processors used in wireless sensor networks in Section II and III. Following that, in Section IV, we present the design of MoteCache. Next, CADMA design is presented in Section V. The simulation methodology is given in Section VI, and our results are presented in Section VII. Finally, we conclude our study in Section VIII.

II. RELATED WORK

In the literature, there are various studies that target energy-efficient microprocessors for wireless sensor nodes [9, 10, 11]. However, to our best knowledge, this is a unique study that proposes architecture-independent extensions to the existing commercially available, general-purpose microprocessors that are used by sensor nodes.

In [9], the authors present SNAP/LE, an event-driven microprocessor for sensor networks. They avoid the TinyOS overhead by their asynchronous, event-driven approach. In [10], the authors seek to fully leverage the event-driven nature of applications. They study the application domain of sensor networks and they seek to replace the basic functionality of a general-purpose microcontroller with a modularized, event-driven system.

In [12], the authors reveal that significant benefits can be gained by detecting and removing silent store instructions. They propose free silent store squashing and successfully increase the processor performance by 11%, on the average. In this study, on the other hand, we choose to move to a completely different direction, and utilize a similar mechanism to reduce the energy consumption of the microcontrollers that are used in wireless sensor devices.

In [13], the authors study partial value locality, which is defined as occurrence of multiple live value instances identical in a subset of their bits. This study targets integer register files in a 8-way superscalar processor.

III. CURRENT STATE OF THE ART

In this paper, we study the MICA2 platform manufactured by Crossbow Inc. [14]. The platform includes an AVR-RISC processor, ATmega128L. The AVR architecture implements the Harvard architecture with two main memory spaces, 4Kx8 bytes of SRAM data memory and 64Kx16 bytes of flash program memory space. Additionally, the Atmega128 features an EEPROM memory for data storage. All three memory spaces are linear and regular [15]. In this study, we mainly focus on the energy reduction of SRAM storage component.

Figure 1 shows the timing diagram of memory read/write operations for the SRAM structure of the AVR. Each memory read/write operation takes two cycles to complete. In the first clock cycle, T1, the memory address is computed, and in the second clock cycle, T2, the actual memory access takes place.

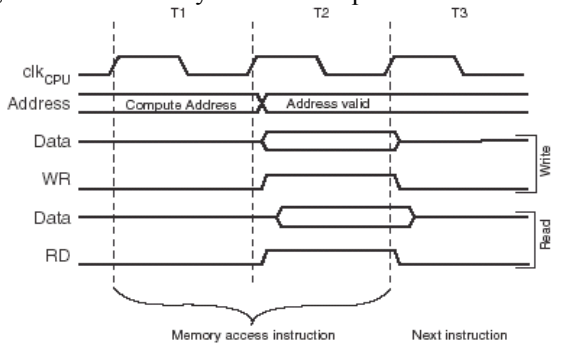


Fig. 1. On-chip Data SRAM access cycles for ATmega128L.

The AVR architecture does not contain a cache structure. The reason behind this cacheless architecture becomes very clear, when we examine the timing diagram in Figure 1, in detail. The main idea to include a cache structure in any design is to increase the processor performance, and it is clear that no cache structure may improve the performance of a 1-cycle-access SRAM. On the other hand, a cache structure may be beneficial from the energy/power point of view. A tiny cache structure may reduce the memory access energy by filtering most of the accesses to the main memory. In the first part of this study, we propose such cache structure, called MoteCache, for the same purpose. Moreover, we show that with a very small, additional complexity we can also use this cache structure to filter silent-store instructions [12].

It is also important to emphasize that our techniques are platform-independent and can be easily applied to other sensor platforms such as moteiv [3], EYES [16] and imote2 [4] sensor nodes. Moreover, these platforms utilize 16- and 32-bit microprocessors (compared to 8-bit microprocessor of the MICA2 platform). Thus, our techniques will achieve much better energy savings in these new platforms.

IV. MOTECACHE

We now explain the design of MoteCache (hereafter MC) which is a tiny buffer structure that exploits temporal locality of data in WSN applications. In MC, each row consists of data buffers and an associative content-addressable memory (CAM) for holding the tags corres-

ponding to the contents of these buffers. The idea is to cache data values of the *N* most recently accessed addresses in an *N*-byte MC [17, 18].

A. The MoteCache Configurations

In this paper, we study three types of MoteCache configuration:

1) Direct-Mapped MoteCache (DMMC): This configuration mimics the behavior of an ordinary direct-mapped cache. Since the SRAM of the MICA2 platform contains a single byte at each line, the direct-mapped cache is organized to contain a single byte at each of its sets. The DMMC configuration has the smallest latency among all, since we can read data in parallel with the tag comparison as soon as the address is available.

2) Set-Associative MoteCache (SAMC): In order to reduce the possible conflict misses of the DMMC configuration; we also decided to try the set-associative cache configuration. This configuration is similar to a standard set-associative cache configuration, and requires the activation of more comparators in a single access.

3) Fully-Associative MoteCache (FAMC): This configuration activates all the tag/address comparators for each memory access. Therefore, it dissipates more power compared to other configurations, and its latency is not better than the DMMC configuration since it has additional multiplexer delay.

B. Access to MoteCache

A read access in the MC structure proceeds as follows:

Cycle 1. MoteCache Tag Comparison: As soon as the address is computed, its tag part is compared associatively with the tag numbers associated with the contents of the MC. If there is a match (MC-hit), the scheduled readout of the memory is cancelled, potentially saving energy wasted in reading out the row from the memory. Again, the tag comparison and data retrieval from the MC may be overlapped to decrease the access latency in the DMMC configuration.

Cycle 2. Data Steering: In case of a MC-hit, data is read from the corresponding MC entry[1]. When the associative match, using the CAM, fails (MC-miss), the memory access continues in this cycle and the data is read out into a MC entry selected as a victim, based on the least recently used (LRU) replacement policy.

Writing to a MC entry has analogous steps, followed by a step that installs the update into the tag and data part of the corresponding MC entry. We study the writeback policy, since it is more suitable for our power-aware design.

Figure 2 shows the modified timing diagram for the proposed architecture. Notice that, at the end of the first clock cycle, T1, there is a comparison latency that compares the tag (or tags, depending on the MC configuration) of the MC set with the computed address[2]. According to the outcome of the tag comparison process, either MC or SRAM is accessed.

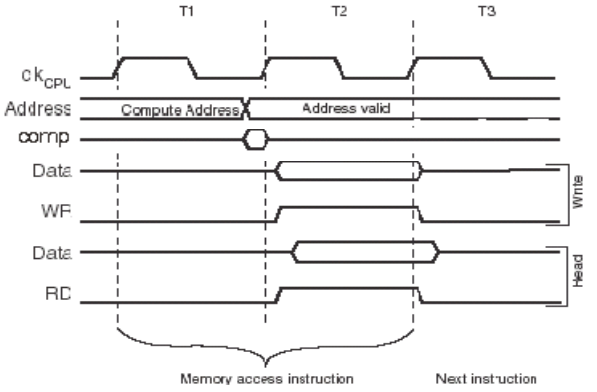


Fig. 2. Modified On-chip Data SRAM and MC access cycles

As locality of reference guarantees that there is a good chance of producing a MC-hit. Figure 3 shows that MC-hit rate is more than 89%, on the average, for the SAMC configuration with maximum size (8x8). The lowest hit rate is observed in the smallest DMMC configuration (4x1) as 18.4%. Please refer to Table 1 in Section VI for the details of the simulated WSN benchmarks.

C. Silent-Store Filtering MoteCache

When a store instruction writes a value that actually matches the value already stored at the memory address being written, that store instruction is called a *silent store* [12]. These instructions have no effect on machine state, and we show that detection and removal of these instructions may considerably improve the lifetime of WSNs. Figure 4 shows the percentages of silent store instructions. Across all WSN benchmarks, the average percentage of silent stores is 81.6%.

We increase the functionality of the *dirty-bit* in our writeback MC for detecting and removing silent store instructions. Therefore, the *dirty-bit* is renamed to *dirty&noisy-bit* (DN). A data value is written back to SRAM only when it is *dirty* (i.e. it is modified) and also *noisy* (i.e. not silent: if the new data value is different than the one stored before.) A write access in the silent-store filtering MC structure proceeds as follows:

[1] To increase processor performance, data can be read from the MC at the end of the first cycle.

[2] Notice that, we assume that the address computation process may be completed a little earlier than the original design. This is possible, since the microprocessors used by the sensor nodes have very low frequency (approx. 7 MHz). Therefore, we assume large clock cycle periods provided by these processors make small tasks such as tag comparison possible in the same clock cycle. However, this is not a strict requirement for the success of our solution. Actually, WSNs may easily tolerate a small increase in processor cycle time for serializing address computation and MoteCache tag comparison in cycle T1, since performance is not their major concern, anyways.

1. **MC-hit and Silent Store Detection:** After a MC-hit, the data value to be written is compared with the data value of the corresponding MC line. If there is no match, the DN-bit of that MC line is set to indicate that the store instruction is *noisy*
2. **MC-miss and Writeback:** After a MC-miss, we select a victim line. If the DN-bit of the victim line is set, the standard MC write-back procedure is followed. Otherwise, we just replace the MC line and cancel the rest of the writeback process, since the MC and SRAM data are already in sync.

V. CONTENT-AWARE DATA MANAGEMENT (CADMA)

In this section, we describe the details of our second technique in this study: Content-Aware Data MAnagement (CADMA.)

During our simulations, we observed that there are some common data values (specifically, 0, 1, 2 and 3) flowing in the datapath. Figure 5 shows that, across all simulated benchmarks, 48.4% of the SRAM data and 56.7% of the register file data are composed of these common data values.

The idea behind CADMA is to exploit this phenomenon to reduce energy dissipation of the SRAM, the MoteCache and the register file. For this purpose, we introduce an additional bit, common data (CD), to each line of these structures. The CD bit is very similar to zero-byte indicator in [19]. Figure 6 depicts the circuitry needed to derive the CD bit for a byte. The CD operations are as follows:

1. Data Write + CD Reset: The CD bit is reset to zero when there is a register/memory write operation trying to store a common data value. In that case, CADMA only writes 3 bits (one CD bit + the two least significant bits of the common data value) for that byte. Note that, for all of the common data values (0, 1, 2 and 3), there is no need for any encoding/decoding process, since they already fit into the least significant 2 bits.

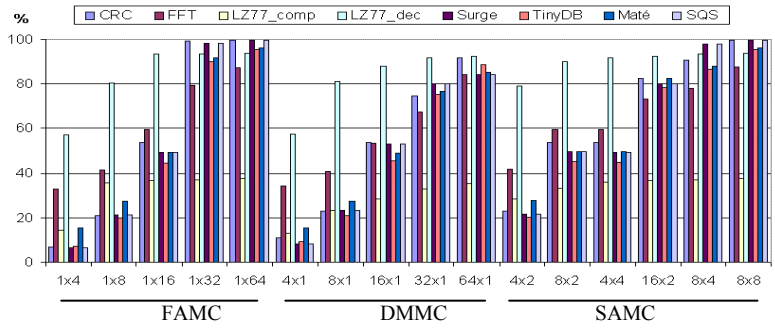


Fig. 3. Hit ratios to various MC configurations (s x a, s: set no, a: associativity)

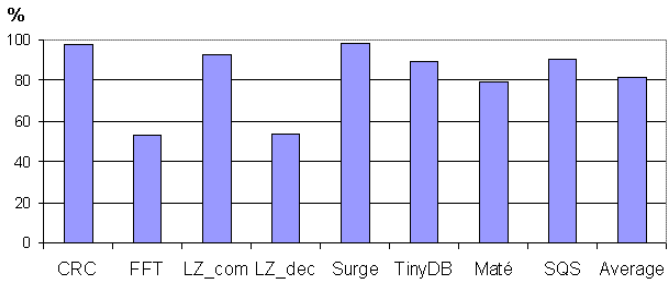
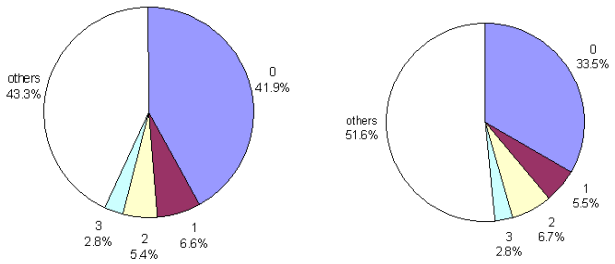


Fig.4. Percentages of silent stores in WSN benchmarks



a) SRAM **b) Register File**

Fig. 5. Average occurrence percentages of data values in SRAM and register file

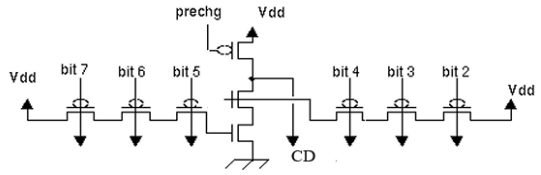


Fig. 6. Encoding logic for the Common Data (CD) bit

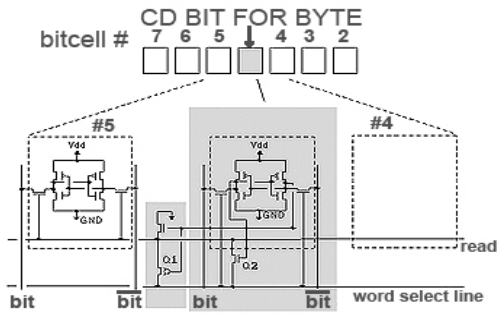


Fig. 7. The CD bit and associated logic common to six most significant bitcells within byte

When the data value is not one of the four common data values, on the other hand, CADMA writes 9 bits (one CD bit + the eight bits of data value) dissipating slightly more power compared to the baseline case.

2. Data Read + CD Probe: A read operation with CADMA starts by probing the CD bit. When it is reset, CADMA only activates the read operation for the two least significant bitlines. By canceling the read from the other six bitlines, CADMA achieves significant power savings in all three structures. In this case, the rest of the byte is constructed by setting the six most significant bits to logic-0. The logic used to disable the readout of six most significant bits of a byte is shown in Figure 7. P-transistor Q1 and n-transistor Q2 are used to connect the read line of the corresponding six most significant bits of a byte to the word select line if the value stored in the CD is one. Otherwise, the local read line is grounded and the six most significant bits of the byte is not read out.

If CD bit is set, slightly more power is dissipated (reading 9 bits: one CD bit + the eight bits of data value) compared to the baseline case.

In this study, we integrated CADMA within MC, SRAM, and register file structures. In our experiments, across all simulated benchmarks, we observed that CADMA boosts our processor energy savings as much as 10% and sensor node's total energy savings by 5% compared to the bare MC.

VI. SIMULATION METHODOLOGY

The most important requirement of this study was using a simulator that accurately models ATmega128L processor of MICA2 platform in a cycle-accurate manner, and simulates sensor networks in various topologies. We found *Avrora*, the AVR Simulation and Analysis Framework [20], quite suitable to our needs. For all simulated benchmarks, a three-node network topology is used.

A. Wireless Sensor Networks Benchmarks

Table 1 gives the details of the benchmarks used in this study. The detailed descriptions of the studied benchmarks are given below:

- CRC implements a CRC32 algorithm continuously running on 448 bytes of data. CRC can be utilized in sensor networks for error detection and is heavy on CPU. We have added packet transfers in between calculations as to simulate a sensor network operation.
- FFT benchmark is a discrete Fast Fourier Transform on 256 bytes of data. This transformation is executed within an infinite loop [21]. Fast Fourier Transform has many uses in sensor networks, mainly for event detection. This is also a CPU intensive benchmark. We have inserted radio transmission every 2 seconds, so it acts like an event is detected and propagated through the network.
- LZ77 compression is again enclosed in an infinite loop and works on 448 bytes of data taken from the header file of an *excel* file.
- LZ77 decompression is the decompression of the compressed data obtained from the LZ77 compression. This data is 330 bytes long[3]. Decompression, along with the compression has also a significant use in sensor networks. This is because the data-send operation requires significantly more energy than compressing or decompressing that data.
- Surge is a TinyOS core application to demonstrate multi-hop routing. This is a radio intensive benchmark and represents a regular sensor network application.
- TinyDB application is used to execute the query "select light from sensors each second". This benchmark is again a radio intensive one. Most applications in sensor networks require a database framework. TinyDB is an important selection in that sense. However, there are many algorithms that reduce the need for radio transmission such as predicting next reading or pruning the data to be propagated. So, TinyDB does not represent an optimal WSN application and is highly communication-intensive.
- Maté (BombillaMica) is used with a script that reads light sensor readings in every 10ms and sends them through RF interface after 10 readings [22]. Maté implements a scripting language with data querying capabilities.

[3] CRC, FFT and LZ77 applications trigger radio communication every 2 seconds to imitate typical WSN applications.

- SeMA/SQS is a data querying framework developed in Yeditepe University. SQS creates a topology tree, which is used both for error handling and optimal radio usage purposes. SeMA/SQS is used to execute the query "select temperature from sensors each second" [23]. The query results are sent to the base station after each reading.

B. Calculation of Energy Dissipation

We modified *Avrora* to record transitions at the level of bits for the processor/memory components. We combined the simulator results with energy per transition measurements obtained from the SPICE simulations of the actual component layouts for 0.35µ CMOS technology. We modified the energy model of Avrora to accurately model very fine-grain, instruction-level energy dissipations of the processor. For accurate energy savings estimations, we also considered detailed energy dissipations of the additional CADMA logic.

VII. RESULTS AND DISCUSSIONS

Figure 8 shows our energy savings for selected silent-store filtering MC configurations. Notice that, even the smallest MC configuration (4x1) saves more than 57% of the memory access energy, on the average. The optimum configuration is found to be the 8x4 configuration, since it gives similar savings compared to the MC configurations twice of its size (80.6% of 8x4 vs. 81.4% of 8x8 – not shown in the Figure.)

When we apply CADMA on top of these MC results, the energy savings gap between the minimum (DMMC 4x1) and the maximum (SAMC 8x4) configurations almost disappears (70% of 4x1 vs. 84% of 8x4). In Figure 9, we also show the percentage of improvement in the energy savings compared to the bare MC case (the rightmost bar.) Again, notice that the integration of CADMA architecture with the previously proposed MC makes the minimum MC configuration quite feasible.

Table 1. WSN benchmarks

Benchmark	Size in ROM (bytes)	Size in RAM (bytes)	Execution Time (mins)
CRC	1072	1492	5
FFT	3120	1332	5
LZ77 Compression	2700	486	5
LZ77 Decompression	1228	356	5
Surge	17120	1928	50
TinyDB	65050	3036	50
Maté	38974	3196	50
SeMA/SQS	22346	2175	50

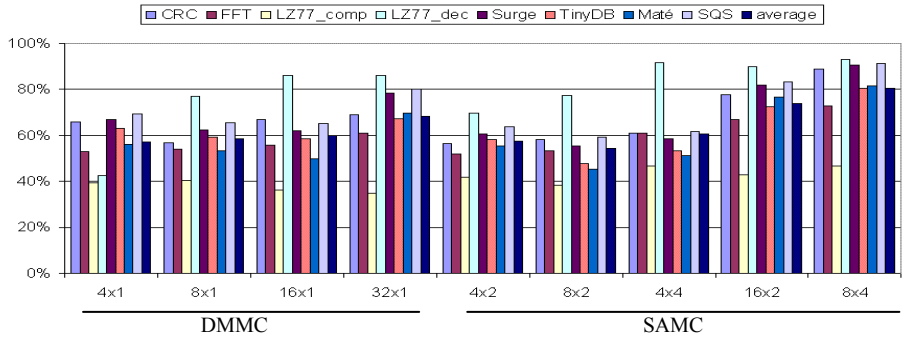


Fig. 8. Average savings on memory access energy dissipation for various MC configurations with the silent-store filter (s x a, s: set no, a: associativity)

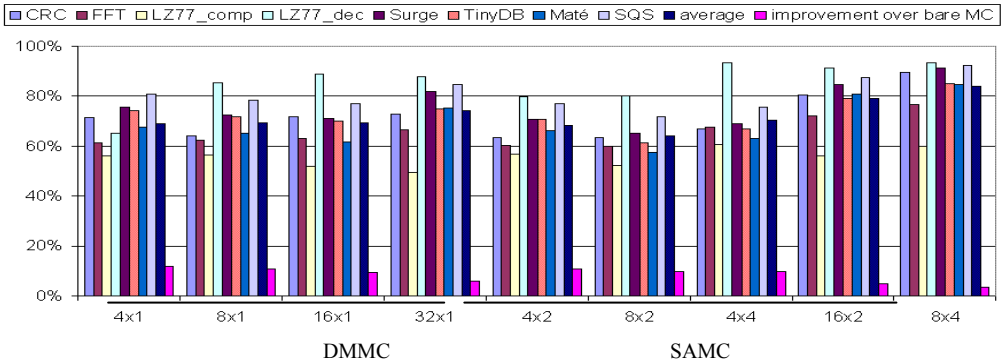


Fig. 9. Average savings on memory access energy dissipation for MC/CADMA

We then compute the effect of CADMA on total processor energy savings. The rightmost bar in Figure 10 shows that 41% of total instructions are memory read/write instructions, on the average. When we apply these figures to our optimal configuration (SAMC 8x4), we find that the total processor energy savings are 51.3% (Figure 10). Note that CADMA extracts additional energy savings from register file (with bare MC, total processor energy savings stay at 46.5%, on the average).

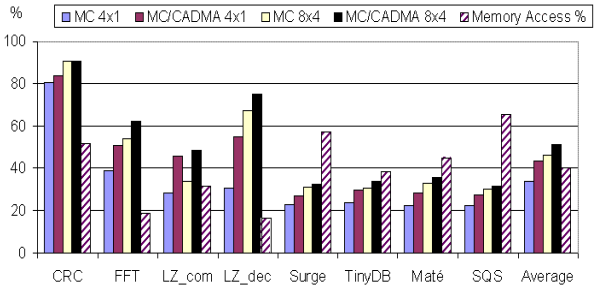


Fig. 10. Processor energy savings and percentages of memory access

When we focus on the minimum MC configuration, energy savings due to CADMA become much clearer. Total processor savings of MC/CADMA combination are 44%, on the average. Notice that, these savings are very close to the savings of bare optimum MC configuration. The energy savings of bare minimum MC configuration are as low as 34%, on the average. These results show that CADMA is a very effective technique that boosts savings of low complexity MC configurations.

Next, we identify the percentage of CPU energy dissipation over total energy dissipation of a sensor node to compute the possible lifetime increase with MC/CADMA architecture. Figure 11 shows that, CPU energy constitutes 49.6% of the total energy dissipation across the simulated benchmarks, on the average. Using the percentages given in this graph, we computed the percentages of total energy savings of individual WSN nodes. Figure 12 shows that the total energy savings are 19.3% for optimum MC/CADMA and 17% for bare optimum MC configuration. Notice that, the gap between the bare minimum MC and minimum MC/CADMA configurations in Figure 10 is still preserved in Figure 12. Again, the energy savings with minimum MC/CADMA configuration are 16.2% (very close to 17% of the bare optimum MC configuration.) The energy savings for the bare minimum MC configuration are 11.7%, on the average. These energy savings are directly related to lifetime improvement of the nodes.

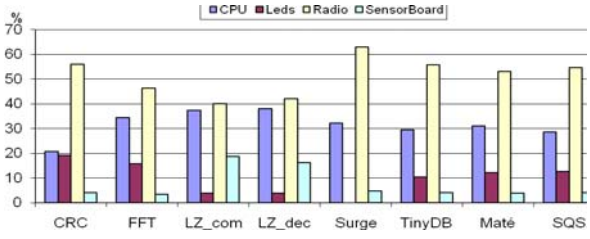


Fig. 11. Energy dissipation of components in MICA2 platform

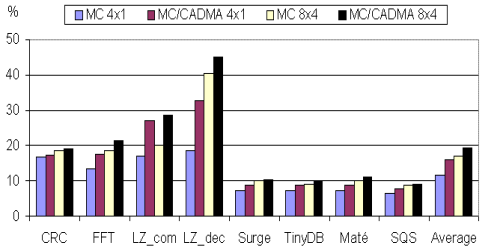


Fig. 12. Node-level energy savings and lifetime improvements

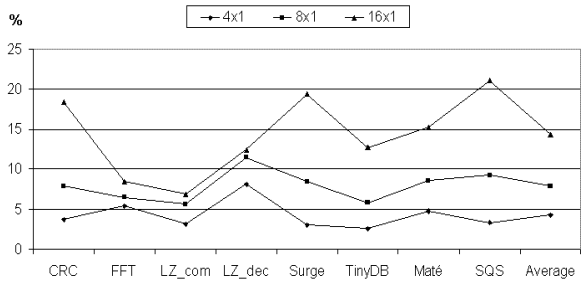


Fig. 13. Possible performance improvements in DMMC configurations

Finally, we also studied the possible performance improvements when we assume that we can access to a MC structure in the first cycle of a memory operation. In Figure 13, we give these results for three DMMC configurations: 4x1, 8x1 and 16x1. The minimum configuration improves the performance by 4.3%, on the average, whereas 8x1 and 16x1 improve it by 8% and 14%, respectively. These figures also indicate that when the latency of our MC/CADMA architecture and address computation cannot be squeezed into a single cycle period, we can still increase the cycle time and tolerate slight performance penalties.

VIII. CONCLUDING REMARKS

In this study, we proposed a platform-independent MC/CADMA architecture for reducing the energy dissipation of the processor component of wireless sensor nodes. We studied various MC configurations and found that 32-byte, 4-way, set-associative, silent-store-filtering configuration shows the best energy/lifetime characteristics. Combined with CADMA, this optimal configuration reduces the node energy by 19.4%, on the average, across a variety of simulated sensor benchmarks. We also found that CADMA integration noticeably improves the results of the minimum MC configuration (4-byte, direct-mapped with silent-store filter.) We showed that this configuration achieves not only energy savings up to 16.2%, but also performance improvements up to 4.3%, on the average. We strongly believe that better results can be observed once these platform-independent techniques are implemented in newer platforms.

Additionally, benchmarks we have selected for this study reflect general requirements of Wireless Sensor Networks. We have included benchmarks, balancing

CPU load and radio load. However, energy consumption of radio is always the leading factor in the context of real life WSN applications. So we believe that results we gathered strongly applicable to any WSN application.

IX. ACKNOWLEDGMENTS

This work is supported by The Scientific and Technical Research Council of Turkey (TUBITAK) under the grant EEEAG/105E0158.

REFERENCES

[1] Hill J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., and Pister, K., "System Architecture Directions for Networked Sensors", *in Proc. of ASPLOS IX*, 2000
[2] AmbientRT: Real-Time Operating System for embedded devices, http://www.ambient-systems.net
[3] Tmote Sky wireless sensor device,http://www.moteiv.com
[4] imote2 sensor, http://www.intel.com/research/sensornets/
[5] Conner, W.S., Chhabra, J., Yarvis, M., Krishnamurthy, L., "Experimental Evaluation of Synchronization and Topology Control for In-Building Sensor Network Applications", *in Proc. of WSNA'03*, Sep. 2003
[6] Madden, S., Franklin, M.J., Hellerstein, J.M., and Hong, W., "TinyDB: An Acquisitional Query Processing System for Sensor Networks", *in ACM TODS*, 2005
[7] Polastre, J.R., "Design and Implementation of Wireless Sensor Networks for Habitat Monitoring", Research Project, University of California, Berkeley, 2003
[8] Kucuk, G., Basaran, C., "Reducing Energy Dissipation of Wireless Sensor Processors Using Silent-Store Filtering MoteCache", *in Proc. of PATMOS'06*, France, 2006
[9] Ekanayake, V. et al., "An Ultra Low-Power Processor for Sensor Networks", *in Proc. ASPLOS*, Oct. 2004
[10] Hempstead, M. et al., "An Ultra Low Power System Architecture for Sensor Network Applications", *in the Proc. of 32nd ISCA'05*, Wisconsin, USA, 2005
[11] Warneke, B.A. and Pister, K.S., "An Ultra-Low Energy Microcontroller for Smart Dust Wireless Sensor Networks", *in Proc. ISSCC*, Jan. 2005
[12] Lepak, K.M., Bell, G.B., and Lipasti, M.H., "Silent Stores and Store Value Locality", *in IEEE Transactions on Computers*, (50)11, Nov. 2001
[13] Gonzalez, R., Cristal, A., Veidenbaum, A., and Valero, M., "A Content Aware Register File Organization", *in Proc. 31st International Symposium on Computer Architecture (ISCA04)*, Munich, Germany, June 2004.
[14] Crossbow Technology, Inc., http://www.xbow.com
[15] Atmel 8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash, http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf
[16] Dulman, S. and Havinga, P., "Operating System Fundamentals for the EYES Distributed Sensor Network", *in Progress 2002*, Utrecht, the Netherlands, October 2002
[17] Ghose, K. et al., "Reducing Power in Superscalar Processor Caches Using Subbanking, Multiple Line Buffers and Bit-Line Segmentation", *in ISLPED'99*, 1999
[18] Kucuk, G. et al., "Energy-Efficient Register Renaming", *in Proc. of PATMOS'03*, Torino, Italy, September 2003. Published as LNCS 2799, pp.219-228
[19] Ponomarev, D., Kucuk, G., Ergin, O., Ghose, K., and Kogge, P. M., "Energy-Efficient Issue Queue Design", *in IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 11, No.5, October 2003, pp.789-800
[20] Titzer, B. et al., "Avrora: Scalable Sensor Network Simulation with Precise Timing", *In 4th Int'l. Conference on Information Processing in Sensor Networks*, 2005
[21] Numerical recipes in C, http://www.library.cornell.edu/nr/cbookcpdf.html
[22] Levis, P., Culler, D. "Maté: A Tiny Virtual Machine for Sensor Networks", *in Proc. of the ASPLOS X*, 2002
[23] Baydere, S., Ergin, M.A., "An Architectural Approach to Service Access in Wireless Ad Hoc Networks", *in Proc. of Wireless and Optical Communications Conf.*, 2002

Gurhan Kucuk Received his BS degree in Computer Engineering from Marmara University, Istanbul, Turkey, in 1995, the MS degree in Computer Engineering from Yeditepe University, Istanbul, Turkey, in 1999, and the Ph.D. degree in Computer Science from the State University of New York at Binghamton in 2004. Since 2004, he has been on the faculty of the Computer Engineering department at the Yeditepe University, where he is an Assistant Professor. His current research interests include wireless sensor networks, energy-efficient/high-performance processor design.

Can Basaran Received his BS degree in Computer Engineering from Yeditepe University in 2005. He is currently an MS student in Yeditepe University. His research interests are Wireless Sensor Networks and Embedded Systems