TOWARDS INCREASED POWER EFFICIENCY IN LOW END EMBEDDED PROCESSORS: CAN CACHE HELP?

Muhammad Yasir Qadri*, Klaus D. McDonald-Maier†

Department of Computing and Electronic Systems
University of Essex, CO4 3SQ, UK
Email: yasirqadri@acm.org *, kdm@essex.ac.uk †

Keywords: Cache overhead analysis, Cache mathematical models.

Abstract

Embedded processors are often characterized by limited resources and are optimized for specific applications. A rising number of battery powered applications has driven a trend towards increased energy efficiency sometimes even traded with performance. Particularly, lower power and low specification embedded processors lack on-chip cache memories. This is mainly in order to avoid the higher energy overhead a cache structure would pose in an embedded processor. This paper proposes energy and throughput models which can be used to analyze energy and time overhead for a particular application due to introduction of a data cache architecture in a previously non-cached system or alternatively can be used in reconfigurable systems for cache overhead analysis.

1 Introduction

With the growing interest in mobile communication devices, battery operated systems and wireless sensor networks in particular, there is an increasing focus on power efficient processor architectures that provide a greater throughput to power consumption ratio. Most low power / low specification 8 and 16 bit embedded processors such as the MSP430 [1], AVR [2], PIC [3], and MAXQ [4], typically lack dedicated on-chip cache memories. This is mainly due to higher energy overhead a cache structure poses on an embedded processor. Related research has shown that caches may consume up to 50% of a microprocessors' total energy [5-7]. This paper presents energy and throughput models which can be used to analyze energy and time overhead due to introduction of a data cache architecture in a previously non-cache system. The model can also be used in reconfigurable processors to appraise the cache overheads. The models analyze the cache overhead based on a particular application running on the system, in that way it is possible to identify if adding a data cache is feasible for a particular application or not. This is particularly important for embedded systems which have to run a single (or a few) application(s) for its life duration. The models are independent of the specific cache configuration used; however they do utilize inputs that may vary according to cache configurations.

The remainder of this paper is divided into five chapters. After the review of related work in the next chapter, the cache energy and throughput models are presented. In chapter 4, experimental setup is detailed and in the remaining two chapters results are discussed and a conclusion is made.

2 Related Work

CACTI [8] is an open-source tool that has been widely used for cache energy and timing analysis. It provides estimates that are within 10% accuracy of the more detailed HSpice results for the circuits chosen, and is computationally simpler and faster than HSpice itself. However, it does not take account of hit rate information as it is not a trace driven simulator. Xipeng et al. [9] provide a framework to predict miss rate based on particular software and cache configurations. This model has shown accuracy within 2 percent of the hit rate for set associative caches on a set of floating-point and integer programs, using array and pointer-based data structures. This model can be utilized at an advanced level to provide accurate hit rate information as an input to the cache overhead models presented in this paper. Based on the same principles a tool named RDVIS [10, 11] was developed to analyze a program by plotting temporal locality or reuse distance information. It utilizes cluster analysis of basic block vectors to hint the user to optimize the code. This tool is particularly useful to profile a piece of code to minimize miss rate for a given size of cache. Chuanjun et al. [5] have presented the idea of a configurable cache. They also have presented an energy model of cache that gives dynamic and static energy consumption in the cache system. Milind et al. [12] have also presented a very detailed model of cache energy consumption. Their analytical models use the inputs such as hit/miss counts, fraction of read/write requests and assume stochastical distributions for signal values. As per their claim, the analytical models for conventional caches were found to be accurate within 2% error. However, their analytical models over–predict the dissipations of low–power caches by as much as 30%. Wen-Tsong et al. [13] present algorithms to find optimal memory configuration based on cache size, the number of processor cycles and the energy consumption.

Most of the cache energy models presented in the work discussed earlier require more accurate information of energy consumed in a hit and miss. However, the cache energy model presented in this paper provides estimates of energy for a cache organization using per cycle energy consumption values on a complete application which is more useful to analyze the overall impact of cache in terms of energy and throughput.

3 Cache energy and throughput models

As discussed above, the models that are presented here estimate the energy consumption and throughput of a complete application; which in fact would be helpful to analyze the overhead caused by a data cache. The overhead can be calculated as a ratio of energy or throughput in a cache based architecture to the one in a non-cache architecture. The two models are presented below.

3.1 Energy Model

The energy model takes into account the total energy consumed by the cache structure for all the cache read and write accesses, and cache to memory accesses along with the energy consumption as cache miss penalty. Here the total energy consumption is calculated as the sum of all the earlier mentioned energy components with the energy consumed in non-memory access instructions.

$$E_{total} = E_{read} + E_{write} + E_{c\to m} + E_{mp} + E_{misc} \qquad (1)$$

where
E_{total} = Total energy consumption of the code [J],
E_{read} = Energy consumed by cache read accesses [J],
E_{write}= Energy consumed by cache write accesses [J],
$E_{c\to m}$ = Energy consumed by cache to memory accesses [J],
E_{mp} = Energy miss penalty [J] and
E_{misc} = Energy consumed by other instructions (which do not require memory access) [J].

The individual components, are further defined as

$$E_{read} = n_{read}.\, E_{dyn.read}.\left[1 + \frac{r_{miss}}{100}\right], \qquad (2)$$

$$E_{write} = n_{write}.\, E_{dyn.write}.\left[1 + \frac{r_{miss}}{100}\right], \qquad (3)$$

$$E_{c\to m} = E_{m}.\,(n_{read} + n_{write}).\left[1 + \frac{r_{miss}}{100}\right], \qquad (4)$$

$$E_{mp} = E_{idle}.\,(n_{read} + n_{write}).\left[P_{miss}.\frac{r_{miss}}{100}\right], \qquad (5)$$

where
n_{read} = Total number of read accesses,
n_{write} = Total number of write accesses,
$E_{dyn.read}$ = Energy consumed per read access [J],
$E_{dyn.write}$ = Energy consumed per write access [J],
E_{m}= Energy consumed per data memory access [J],
E_{idle}= Per cycle idle mode energy consumption of the processor [J],
r_{miss} = Miss ratio (in percentage) and
P_{miss}= Miss penalty (in number of stall cycles).

3.2 Throughput Model

Although throughput is often defined in MIPS; equation (6) can be used to find the amount of time an application will take to execute. Assuming cache to memory access time is overlapping the cache access time (due to concurrent nature of the two processes); it is possible to express that

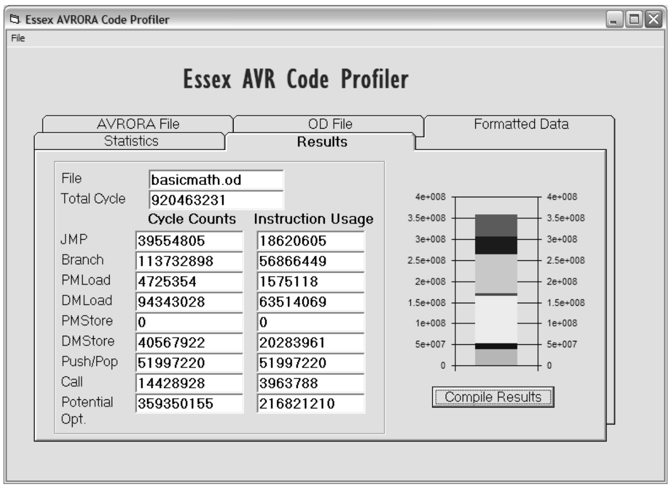
$$T_{total} = t_{cache} - t_{mem} + t_{mp} + t_{misc} \qquad (6)$$

where
T_{total} = Total time taken by an application [Sec],
t_{cache} = Time taken for cache operations [Sec],
t_{mem} = Time saved from memory operations [Sec],
t_{mp} = Time miss penalty [Sec] and
t_{misc} = Time taken while executing other instructions (which do not require memory access) [Sec].
Furthermore

$$t_{cache} = t_{c}.\,(n_{read} + n_{write}).\left[1 + \frac{r_{miss}}{100}\right] \text{ and} \qquad (7)$$

$$t_{mp} = t_{cycle}.\,(n_{read} + n_{write}).\left[P_{miss}.\frac{r_{miss}}{100}\right] \qquad (8)$$

where
t_{c} = Time taken per cache access [Sec] and
t_{cycle}= Cycle time [Sec].



File	basicmath.od
Total Cycle	920463231

	Cycle Counts	Instruction Usage
JMP	39554805	18620605
Branch	113732898	56866449
PMLoad	4725354	1575118
DMLoad	94343028	63514069
PMStore	0	0
DMStore	40567922	20283961
Push/Pop	51997220	51997220
Call	14428928	3963788
Potential Opt.	359350155	216821210

Figure 1: Essex AVR Code Profiler output

4 Experimental Setup

To analyze the proposed models, a simulation environment was created that uses an AVR core (ATmega128 [14]) to implement the models for a non-cache microcontroller, AVRORA [15] was used to get the coarse-grained code profile and cycle counts, and a cache modeler (CACTI 4.2 [8]) was used to obtain data for cache access time and energy per access information. As the proposed models give the information per application basis, a benchmark application (i.e. BasicMath) from MiBench [16] was chosen, which is an open-source embedded system specific benchmark suit. The BasicMath application is a part of automotive and industrial control suite of MiBench. This application performs mathematical calculations such as cubic function, integer square root and angle conversions from degrees to radians. The input data is a fixed set of constants. This application was particularly chosen because of it relevance to a wide range of automotive, industrial and sensor data processing applications.

As discussed earlier, UCLA's AVRORA [15] was used to extract code profile information. However this tool does not give instruction level code profile so that instructions particular to memory access could be separated. To overcome this problem a software named Essex AVR Code Profiler (see Figure 1) was created, that gives an in-depth instruction level profile of a particular application, which is vital for the presented models based cache analysis. The software uses coarse-grained code profile output of the AVRORA simulator along with the object dump file to generate a fine-grained instruction level profile of the code (see Figure 2). The Object Dump utility is a part of AVR-gnu tool chain. Figure 2, shows a detailed distribution of instructions into several classes like Calls and Returns (CR), Push/Pop (PP), data memory store (DMS), program memory store (PMS), data memory load (DML), program memory load (PML), branch (BRN), Jumps (JMP) and miscellaneous instructions (Misc.) based on the MiBench benchmark suite. As per focus of this study on data cache, instructions regarding to data memory access can easily be identified. The Branch and Jump information present in the output data could be used to identify potential stalls, for pipeline and I-cache operations if introduced.

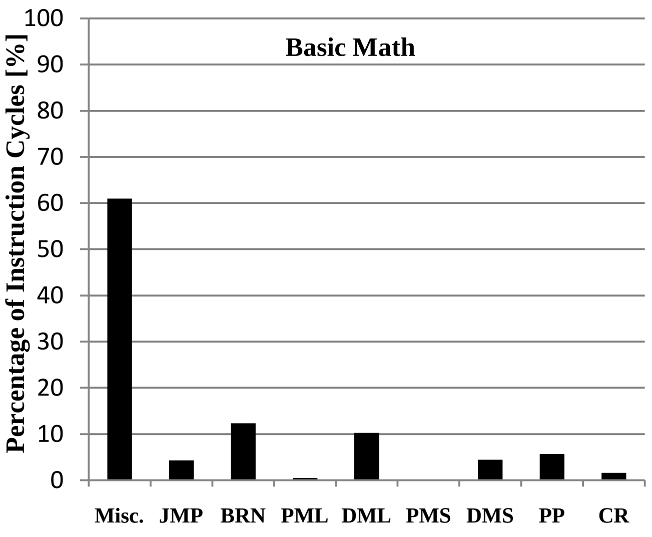


Figure 2: Instruction usage in percentage of cycles for BasicMath application.

Now using CACTI cache modeler [8], information regarding to cache access time and read/write energy consumption per access for a direct mapped cache, was recorded. The parameters given in Table 1 were used to simulate the cache structure.

A simplified single block direct mapped cache was selected for the experiment due to its highest potential throughput and lowest energy consumption per access. However it is important to note that a direct mapped cache has a lowest hit rate compared to other cache configurations [17]. The cache size was chosen 512 bytes to be at realistic ratio with ATmega128 on-chip memory i.e. 4K bytes. The fabrication technology was chosen to be 0.35µm, in line with technology the ATmega128 is manufactured [18].

Cache Parameter	Value
Number of banks	1
Total cache Size(bytes)	512
Size in bytes of a bank	512
Number of sets per bank	64
Associativity	direct mapped
Block Size (bytes)	8
Read/Write Ports	1
Read Ports	0
Write Ports	0
Technology Size	0.35µm
V_{dd}	2.6V

Table 1: CACTI 4.2 Input Parameters

5 Results

The cache model given to CACTI resulted in access time of 2.12nS, a total dynamic read and write energy of 0.0795701nJ, and 0.0246899nJ respectively. Due to the absence of ATmega128 memory access energy consumption data, the cache to memory access energy ($\boldsymbol{E_{c\rightarrow m}}$) was conservatively assumed to be equal to per cycle energy consumption of the controller i.e. 3.75nJ at 3V and 4MHz [14]. The cache miss penalty was assumed to be 10 cycles. The results from Essex AVR Code Profiler, showed that the BasicMath application comprises a total of 91087797 memory read, 46282571 memory write operations; assuming all program memory load (PMLoad) operations as data memory load operations. This assumption is made keeping in view of practical aspect where the constants used in the benchmark application would be replaced by variables in the data memory.

Based on these models of energy and throughput it was found that for miss rates less than 20% a significant increase in throughput with less energy consumption can be observed (see Figure 3). It is interesting to note that the overall power consumption of the device remains higher than the actual power consumption (for a non-cache ATmega128) i.e. 15 mW. The power consumption tends to decrease with the increase in miss rate, and for miss rate beyond 40%, it drops even lower than the actual power consumption (see Figure 4). The power consumption statistics may mislead to a conclusion that higher miss rates result in greater power efficiency. It must be noted that, power consumption is a ratio of energy consumption to the time. If the time overhead is increased with same amount of energy, power consumption tends to decrease. That is why Performance/Watt (e.g.

MIPS/Watt) is typically used as a metric to evaluate power efficiency of a processor. Thus in this case, for analyzing cache feasibility, energy and throughput statistics should be considered alone. The results show viability of the cache for an AVR microcontroller, if the miss rate is kept below 20% for BasicMath application.

It is also important to recognize that improved cache structures, such as those proposed in [19, 20], could result in higher throughput at the cost of even lower energy consumption; making cache a good choice for an optimized code.

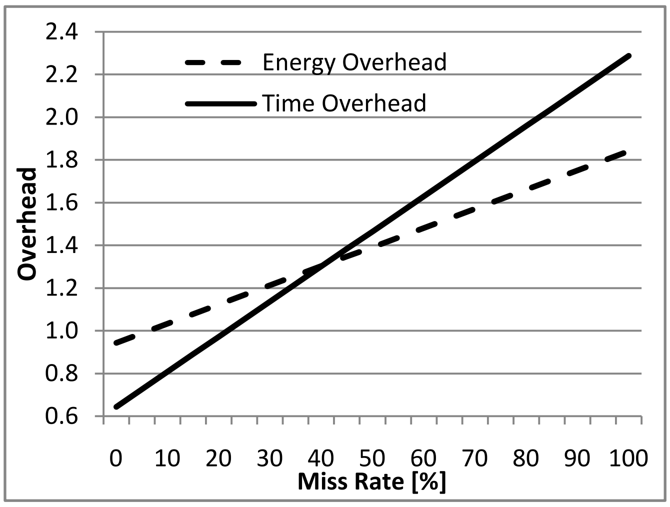


Figure 3: Cache overhead analysis:
Energy and Time overhead vs. miss rate

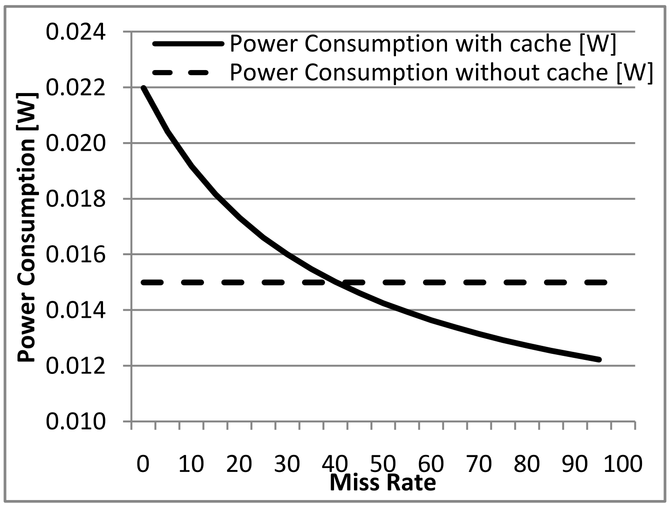


Figure 4: Cache Power consumption analysis

6 Conclusion

The energy model presented here does not take account of static power consumption due to non-availability of leakage data at 0.35µm technology in the used simulator; which is an important factor in memory hierarchies. However these simple models with fewer input parameters can give an earlier estimate of energy and throughput improvements for a reconfigurable or non-cache system. In future the models are to be evaluated for a reconfigurable processor along with some miss rate prediction techniques, so that a fair estimate could be made about their accuracy.

Acknowledgements

This research is part supported by the UK Engineering and Physical Sciences Research Council (EPSRC) under Grants EP/C005686/1, EP/C014790/1 and EP/C54630X/1.

The support of Mr. Hemal S Gujarathi for the review of cache architectures and related technologies is gratefully acknowledged.

References

[1] Buccini, M., *MSP430F21x1 Architecture Summary*. 2004, Texas Instruments.

[2] Turley, J., Atmel AVR Brings RISC to 8-Bit World, in Microprocessor Report. 1997.

[3] *8-bit PIC® Microcontrollers*. [cited 2008 May 1st]; Available from: www.microchip.com.

[4] *MAXQ Microcontrollers*. [cited 2008 May 1st,]; Available from: http://www.maxim-ic.com.

[5] Chuanjun, Z., V. Frank, and N. Walid, *A highly configurable cache architecture for embedded systems.* SIGARCH Comput. Archit. News, 2003. **31**(2): p. 136-146.

[6] Afzal, M., M. Bill, and C. Dan, A low power unified cache architecture providing power and performance flexibility (poster session), in Proceedings of the 2000 international symposium on Low power electronics and design. 2000, ACM: Rapallo, Italy.

[7] Segars, S., Low power design techniques for microprocessors, in Int. Solid-State Circuits Conf. Tutorial,. 2001.

[8] Tarjan, D., S. Thoziyoor, and N.P. Jouppi, *CACTI 4.0*. 2006, HP Laboratories Palo Alto.

[9] Xipeng, S. and S. Ahren, *Miss Rate Prediction Across Program Inputs and Cache Configurations.* IEEE Trans. Comput., 2007. **56**(3): p. 328-343.

[10] Beyls, K. and E. D'Hollander, *Platform-Independent Cache Optimization by Pinpointing Low-Locality Reuse*. Lecture Notes in Computer Science. Vol. 3038/2004. 2004: Springer.

[11] Beyls, K., E. D'Hollander, and F. Vandeputte, *RDVIS: A Tool that Visualizes the Causes of Low Locality and Hints Program Optimizations*. Lecture Notes in Computer Science. Vol. 3515. 2005: Springer.

[12] Milind, B.K. and G. Kanad, Analytical energy dissipation models for low-power caches, in Proceedings of the 1997 international symposium on Low power electronics and design. 1997, ACM: Monterey, California, United States.

[13] Wen-Tsong, S. and C. Chaitali, Memory exploration for low power, embedded systems, in Proceedings of the

36th ACM/IEEE conference on Design automation. 1999, ACM: New Orleans, Louisiana, United States.
[14] *ATmega128(L) Datasheet*. [cited 2008 May 1st,]; Available from: www.atmel.com.
[15] Ben, L.T., K.L. Daniel, and P. Jens, Avrora: scalable sensor network simulation with precise timing, in Proceedings of the 4th international symposium on Information processing in sensor networks. 2005, IEEE Press: Los Angeles, California.
[16] Guthaus, M.R., et al., MiBench: A free, commercially representative embedded benchmark suite, in Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop. 2001, IEEE Computer Society.
[17] Steven J.E. Wilton, N.P.J., An Enhanced Access and Cycle Time Model for On-Chip Caches, in WRL Research Report 93/5. 1994, Western Research Laboratory.
[18] *ATMega128 Reliability Qualification Report*. [cited 2008 May 1st,]; Available from: www.atmel.com.
[19] Koji, I., I. Tohru, and M. Kazuaki, Way-predicting set-associative cache for high performance and low energy consumption, in Proceedings of the 1999 international symposium on Low power electronics and design. 1999, ACM: San Diego, California, United States.
[20] Song, J. and Z. Xiaodong, LIRS: an efficient low inter-reference recency set replacement policy to improve buffer cache performance. SIGMETRICS Perform. Eval. Rev., 2002. **30**(1): p. 31-42.