The Filter Cache: An Energy Efficient Memory Structure

Johnson Kin, Munish Gupta and William H. Mangione-Smith
The Department of Electrical Engineering, UCLA Electrical Engineering
{johnsonk,mupgupta,billms}@icsl.ucla.edu

Abstract

Most modern microprocessors employ one or two levels of on-chip caches in order to improve performance. These caches are typically implemented with static RAM cells and often occupy a large portion of the chip area. Not surprisingly, these caches often consume a significant amount of power. In many applications, such as portable devices, low power is more important than performance. We propose to trade performance for power consumption by filtering cache references through an unusually small L1 cache. An L2 cache, which is similar in size and structure to a typical L1 cache, is positioned behind the filter cache and serves to reduce the performance loss. Experimental results across a wide range of embedded applications show that the filter cache results in improved memory system energy efficiency. For example, a direct mapped 256-byte filter cache achieves a 58% power reduction while reducing performance by 21%, corresponding to a 51% reduction in the energy-delay product over a conventional design.

1 Introduction

In order to mask latency, and thus improve performance, most microprocessors have one or two levels of on-chip caches. For simple single-issue microprocessors, the access time of the first level cache often is on the critical path [1]. Even though these caches consist of densely packed MOS transistors, the area assigned to on-chip caches can be a significant fraction of the entire IC area. Figure 1 shows a die photo of the StrongARM processor from DEC, which is dominated by cache memory. The power dissipated by the on-chip caches is often a significant part of the power dissipated by the entire microprocessor. Table 1 compares key characteristics for two modern embedded RISC microprocessors, the StrongARM 110 [2] and a PowerPC from IBM [3]. For these chips, the cache power consumption is either the largest or second largest power-consuming block. This trend will likely continue as embedded processors become more sophisticated and provide higher performance.

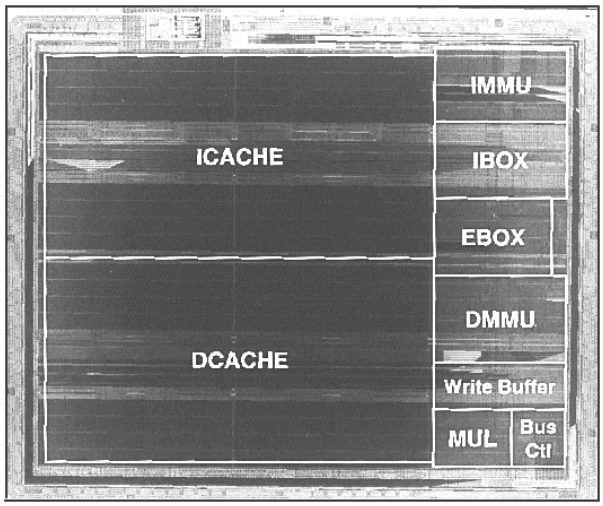

Figure 1: Die of DEC StrongARM

Caches clearly present one of the most attractive targets for power reduction. Power reduction in caches can be achieved through several means: semiconductor process improvements, memory cell redesign, voltage reduction, and optimized cache structures. Our focus here is on cache structures, which is where architects can have the largest impact.

The motivation for our research is the increased application of embedded systems for multimedia and communication applications. Many techniques have been discovered which lead to increased performance as well as reduced power consumption. For example, improvements in cache organizations for high performance commercial processors also serve to reduce traffic on high capacitance buses. This phenomenon clearly helps to save power although the primary goal is improving performance. We believe that low power researchers should be open to sacrificing some performance for power savings. Our experience is that, without this perspective, the best power saving ideas often have been discovered in the chase for high performance. Many approaches can be used to reduce power by sacrificing arbitrary amounts of performance, e.g. avoiding pipelining and fully associative caches. Furthermore, by introducing an arbitrary reduction in the clock rate, we can achieve an arbitrary reduction in the power consumed by CMOS circuits. Clearly, power alone is not a good design

metric, and must be evaluated along with some concern for performance. We have decided to adopt the Energy•Delay metric, which has been used evaluating CAD tools and circuit designs.

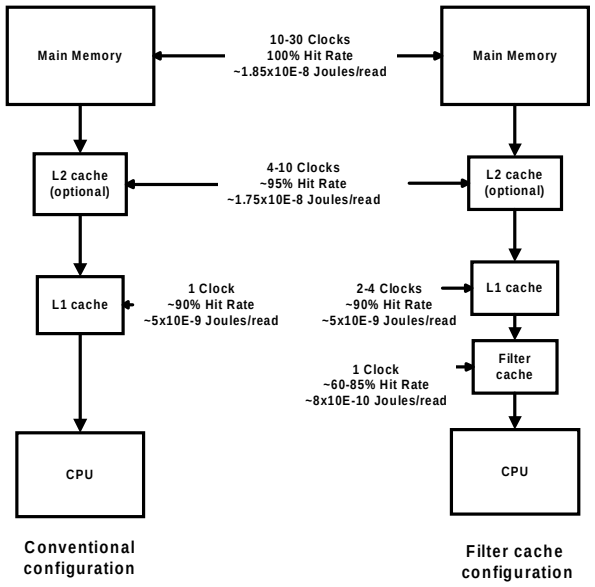

Figure 2: Power and performance characteristics of traditional caches and the filter cache

We propose the use of a first level cache that is very small relative to conventional designs. This cache has reduced power dissipation relative to a traditional cache architecture, albeit at the expense of a decreased hit ratio. Our hypothesis is that the decrease in power consumption will compensate for the loss in performance, resulting in a reduced Energy•Delay product. The small L1 cache is called a *filter cache* in order to distinguish it from traditional caches that are designed solely for performance. However, while the design decisions differ from those of a traditional cache, the basic structure is the same.

The basic filter cache organization is illustrated in Figure 2, where it is compared to a traditional memory organization. Power consumption estimates for the caches are calculated using a power model presented later in the paper. The power consumption of the main memory only accounts for the bus capacitance, and ignores the power consumed in the memory chips. The L1 cache is likely to have the same characteristics for both systems. However, with the new design, the L1 cache is only accessed as a consequence of a miss in the filter cache, otherwise it is not cycled and remains in a standby mode. Thus, although the L1 cache has a similar design in both cases, it will require an additional clock cycle for access with the filter cache. Because the filter cache is smaller than the L1 cache, it will generally have a faster access time. While this phenomenon may present an opportunity to increase the processor clock, it will necessarily result in an increase in the latency for access to the L1 cache. We have not investigated the impact of this option, and will only be comparing systems with equal clock frequencies and two-cycle access for the L1 cache backing the filter cache.

1.1 Overview

We use the following approach to evaluate the energy dissipation in the cache of an embedded processor. The cache power is mainly a function of the capacitance of the memory array and the access transitions. We determined the actual number of transitions through a detailed cycle-level simulation. The capacitance values were obtained from published results for 0.8um technology. The analysis can be repeated, with the same resulting trends, using values from a more modern process if available. These parameters were applied to an analytical model in order to determine the power dissipation for the filter caches and the backing L1 caches.

The remainder of the paper is organized as follows. The next section presents the previous work that is relevant to our problem and approach. Section 3 presents the experimental methods used. The workload used for evaluation is discussed in Section 4. Section 5 presents the experimental results and discusses effective filter cache design. The paper concludes with a discussion and summary.

2 Previous Work

The related previous work can be divided into three major areas: low power processor evaluation, cache modeling, and low power cache structures.

Gonzalez and Horowitz investigated various methods for evaluating low power processor designs [4], and they presented some of the first arguments concerning power and performance to the architecture community. Much of this discussion follows the structure of previous arguments in the CAD community. Based on this work, we have adopted the Energy•Delay metric for system evaluation.

Kamble and Ghose [5, 6] have developed an analytic model for power consumption in various cache structures. Their model combines memory traffic, process features such as capacitance, and architectural factors including line size, associativity, and capacity. The process models are based on measurements reported by DEC for a 0.8um process technology [7]. We use these models and values for power estimation.

Su and Despain evaluated the effectiveness of a number of low power cache structures. Block (i.e. line) buffering involves latching the last cache line, while sub-banking involves only powering portions of the L1 cache [8]. Ko and Balsara have investigated a similar technique that they call Multiple-Divided Modules (MDM) [9]. These approaches are conceptually similar to the filter cache. Block buffering can be viewed as a degenerate case of the filter cache and our results indicated that a significant benefit could be realized by larger structures. Compared to MDM, the filter cache results in better Energy•Delay results because of improved performance. Furthermore, the filter cache can be turned off when higher performance is needed, which is not possible with MDM.

3 Experimental Methods

The base machine model used here is an embedded processor executing the HPPA instruction set. The system is designed to be roughly comparable to the DEC StrongARM 110 in terms of system resources [10]: 16 KB instruction and data caches, single issue processor core, no aggressive branch prediction, no L2 cache. Applications were compiled and executed using the IMPACT toolset [11]. This approach allows us to experiment with various cache structures and generate accurate clock cycle counts for execution time. These counts are used directly for Delay in the Energy•Delay measures. The Energy, Delay, and Energy•Delay measures were determined for the base processor executing each application in the experimental workload, and subsequently used to evaluate alternate filter cache designs.

The cache power models are based on the work by Kamble and Ghose [6] for a 0.8um cache implemented in the CMOS technology. We have assumed a supply voltage of 3.3 Volts. The cache energy is a function of the number of transitions on cache components (bit-lines, word-lines, memory cells and decoders) along with the capacitance of each component. The IMPACT Lsim processor simulator produces application-dependent cache performance statistics. This data is used in the following models to determine transitions within the various cache components.

The power consumption model is developed in Equations 1-3. The terms N_{hit} and N_{miss} represent the raw number of hits and misses to the first level cache. The term N_{addr} counts the average number of address line transitions seen by the first level cache from the CPU, assuming that half of the address lines switch during each memory request. T represents the number of tag bits, while C is the number of control bits stored with each cache line. M is the degree of associativity, and L is the line size measured in bytes. Using these characteristics of the cache organization, it is easy to count the number of precharged bit transitions that will occur, which is designated N_{bp}. Using W for the average number of bits that switch on each write operation, the total number of bit reads (N_{br}) and the total number of bit writes (N_{bw}) can be calculated. A value of 19 is used for W in the studies below. The last three terms in Equation 1 account for the address and data traffic seen on the far side of the cache, as a result of misses. The capacitance values were obtained by substituting the technology and layout specific parameters ($C_{drain,Q1}$, $C_{bitwire}$, $C_{drain,Qp}$, $C_{drain,Qpa}$, $C_{gate,Q1}$, $C_{wordwire}$, $C_{memaddr}$, $C_{cpuaddr}$, $C_{memdata}$, $C_{cpudata}$, C_{decode}, C_{awire}) [7] into Equation 2.

By combining the capacitance values with the raw transition counts, we can determine the energy consumed by each component of the cache (Equation 3). Using these equations, it is possible to calculate the energy consumed in a single read to a 16-KB direct mapped instruction cache with a 32-byte line. The values of C_{bp}, C_{ba}, and C_{word} are 7E-13F, 8.6E-13F, and 2E-13F respectively. Consequently, E_{bit} is 2.56E-9J, E_{word} is 2.45E-12J, E_{output} is 3.74E-10J and E_{ainput} is 2.57E-15J, for a total energy of 2.93E-9J.

4 Benchmarks

There currently exists a significant void with regards to effective benchmarks for embedded systems. While a number of industrial and academic efforts have been proposed, to date there has been little progress towards a suite of representative programs and workloads. One part of the problem is that the field of embedded systems covers a wide range of computing systems. It is difficult to imagine a benchmark suite that would be useful to the designers of fax machines and cellular phones, because of the drastically different uses for these products. This unfortunate state of affairs is best reflected by the use of the Dhrystones benchmark, and the derivative metric Dhrystones per milli-Watt. For the purposes of this paper we have adopted the MediaBench benchmark suite [12]. These benchmarks encompass most of the media applications in use today. Table 2 summarizes the codes, provides a brief description of them, and indicates the number of instructions simulated for each.

5 Experimental Results

For our base case, we have assumed a one level cache hierarchy with split Instruction and Data caches, each with a capacity of 16 KB. These are direct

mapped caches with a line size of 32 bytes each. The filter cache machines have a two level cache hierarchy consisting of instruction and data filter caches and a unified direct mapped 32 KB L1 cache. The L1 used in conjunction with the filter cache has the same size as the combined L1 caches for the base machine. We considered two small filter cache sizes: 128 and 256 bytes. The line size was varied between 8 and 32 bytes, and both direct mapped and fully associative caches were considered. The instruction and data filter caches were simulated with the same configuration in order to reduce the number of design points to be evaluated. The hardware cost of these structures is small relative to the base case L1 cache.

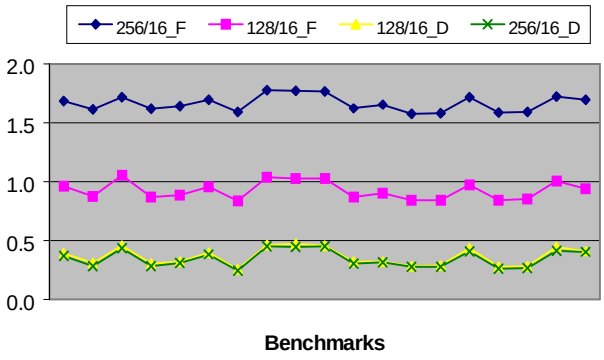

Figure 3 : Energy efficiency vs. size and associativity

The results for each filter cache design are presented relative to the base machine. Power consumption for the MediaBench applications is shown in Table 3 for the 128 byte filter caches and Table 4 for the 256-byte configurations. Figure 3 summarizes the most important aspects. Each line is identified by its cache size, line size and degree of associativity; for example a 128 byte fully associative filter cache with 16 byte lines is identified as 128/16_F. First, associativity tends to have a strong impact on power consumption. For example, the 128 byte fully associative cache with 16byte lines consumes almost as much power as the split 16K byte direct mapped caches used for the base case. Associativity increases the amount of data and control information read out of cache arrays, thus consuming more power. The variation between the 128 byte and 256 byte fully associative caches is due to the fixed line sizes. The 256 byte fully associative cache has twice as many lines as the 128 byte filter cache, thus consuming approximately twice as much power. The improved hit rate does not fully compensate for the increased energy per reference.

Secondly, the power consumption for the 128 byte and 256 byte direct mapped caches is reasonably similar. These caches have similar hit rates; thus they have approximately equal effectiveness at filtering out memory references.

Finally, for the direct mapped cases, increased line size tends to increase power consumption. Again, for these caches, we tend to see hit rates that are fairly close. However, each reference to a 32-byte direct mapped line reads out four times as much data as the 8-byte case, thus discharging four times as many bit-lines.

The performance impact of the filter cache is shown in Table 5 for the 128 byte caches and Table 6 for the 256-byte caches. Somewhat surprisingly, increased associativity often reduces performance for these applications by a small amount. This phenomenon appears to be the result of the combination of uncommonly small caches and the corresponding small number of cached lines, as the impact is generally less for the larger cache size. For the caches considered here the line sizes are still small enough that the system sees a significant benefit from the effect of instruction prefetch. Figure 4 illustrates this point for the 256-byte direct mapped filter caches. While the delay is always greater than the base case (i.e. the performance is lower) the longer line sizes have reduced performance loss and exhibit significantly less variability. Although the shortest line size approaches the performance of the longer lines for several applications, in general, it appears to be a poor choice for performance.

The resultant Energy•Delay measures are shown in Table 7 and Table 8 for the 128 byte and 256 byte cases respectively. We will first consider the impact of full associativity. Full associativity performs worse than the base case for all but three instances. These particular points correspond to the smallest cache size and the largest line size; thus they have the smallest amount of associativity. For these cases (epic, pgpdecode, and pgpencode) the Energy•Delay improvement is still very small relative to the base case. Due to these results, we will no longer consider the fully associative caches.

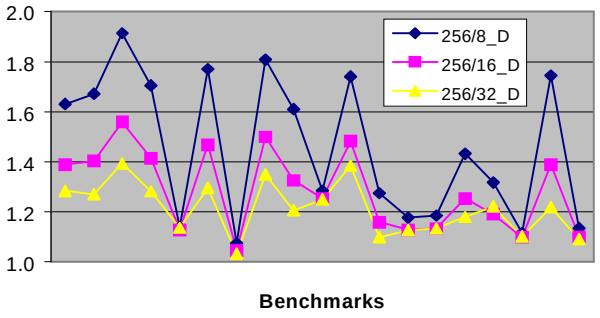

Figure 4: Performance ratio vs. line size

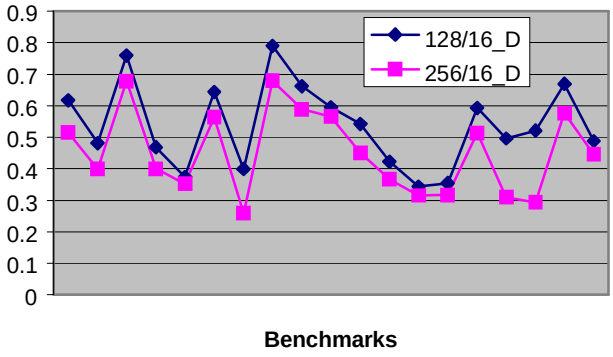

Figure 5: Energy•Delay vs. filter cache size

Figure 5 summarizes the Energy•Delay for the direct mapped caches with 16 byte lines. For many cases, the benefits of the larger cache are relatively small, and on average the 256-byte filter cache has only a 16% benefit. However, the overall effect in reduced Energy•Delay is much flatter for the larger cache, suggesting that the additional size is likely to be worth the investment.

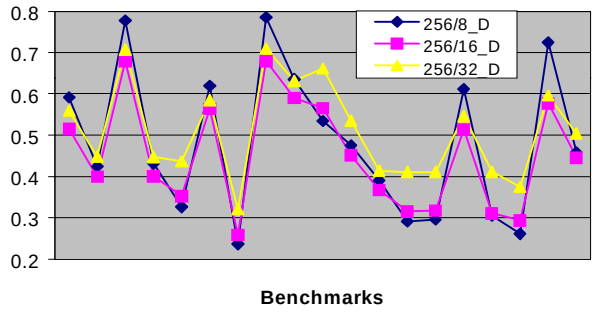

Figure 6: Energy delay product ratio vs. line size

Having selected the 256-byte direct mapped filter caches for closer examination, Figure 6 highlights the impact of varying the line size. The conclusions are less clear here, because the conflicting factors that drive Energy•Delay result in no consistent design advice. As the line size gets longer, the delay tends to decrease while the power consumption tends to increase. Based on the mean Energy•Delay, one would conclude that the 16-byte line size is the best choice. This seems natural, as it is a balance between the competing pulls of energy and delay. Additionally, this line size has the least variance across Energy•Delay, and thus provides the most consistent benefits.

6 Discussion

Several topics remain to discuss which do not neatly fall into any section of this report. While this work introduces the filter cache and establishes its effectiveness, there remains a significant amount of further work required to better analyze and quantify the design space. In particular, we are moving forward with further simulation to explore the effectiveness of a small degree of associativity. A sound analytic model is also needed to capture the basic physical phenomena and allow rapid exploration of the design space without exhaustive simulation.

The filter cache design presented here uses a backing cache that has an access time that is typical of existing embedded L1 caches. Clearly two design opportunities exist: the backing cache can stay with a single cycle access, or it can require multiple cycle access while the processor clock is increased to match the shorter access time of the filter cache. It is interesting to imagine a design that left the backing cache with a single cycle access. The filter cache can then be turned off, moving the backing cache back to the L1 position with single cycle access time. By controlling the filter cache with a processor mode bit, the system can switch between high-speed operation and low Energy•Delay operation as system demands vary.

7 Conclusions

In spite of the increased commercial interest in sophisticated embedded processors, very little work has been conducted to improve the Energy•Delay performance of embedded processor systems through architecture. This paper develops and evaluates the filter cache, a small memory that trades performance for reduced power in order to optimize Energy•Delay. The filter cache has been shown to provide an average Energy•Delay reduction of 51% across a set of 19 multimedia and communications applications. In

particular, simple direct mapped caches with moderate line sizes appear to be extremely good design points.

Acknowledgements: This work was supported by DARPA under contract F04701-97-C-0010. The authors also greatly appreciate the assistance of Professor W. M. Hwu's IMPACT team at UIUC.

References

[1] D. A. Patterson and J. L. Hennessy, "Large and Fast: Exploiting Memory Hierarchy," in *Computer Organization & Design The Hardware/Software Interface*: Morgan Kaufmann, 1994.

[2] J. Montanaro and e. al., "A 160MHz 32b 0.5W CMOS RISC Microprocessor," Proc. of International Solid-State Circuits Conference, 1996.

[3] R. Bechade and e. al., "A 32b 66MHz 1.8W microprocessor," Proc. of International Solid-State Circuits Conference, 1994.

[4] R. Gonzalez and M. Horowitz, "Energy Dissipation in General Purpose Microprocessors," *IEEE Journal of Solid State Circuits*, vol. 31, pp. 1277-1284, 1996.

[5] M. B. Kamble and K. Ghose, "Energy-Efficiency of VLSI Caches: A Comparative Study," Proc. of International Conference on VLSI Design, 1997.

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

[7] S. E. Wilton and N. Jouppi, "An Enhanced Access and Cycle Time Model for On-Chip Caches," DEC WRL, Research Report 93/5, 1994.

[8] C. Su and A. Despain, "Cache Design Tradeoffs for Power and Performance Optimization: A Case Study," Proc. of International Symposium on Low Power Design, 1995.

[9] U. Ko, P. T. Balsara, and A. K. Nanda, "Energy Optimization of Multi-Level Processor Cache Architectures," Proc. of International Symposium on Low Power Design, 1995.

[10] J. Turley, "ARM Grabs Embedded Speed Lead," in *Microprocessor Report*, vol. 10, 1996.

[11] P. P. Chang, S. A. Mahlke, W. Y. Chen, N. J. Warter, and W.-m. W. Hwu, "IMPACT: An Architectural Framework for Multiple-Instruction-Issue Processors," Proc. of International Symposium on Computer Architecture, 1991.

[12] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: A Tool for Evaluating Multimedia and Communications Systems," Proc. of Micro 30, 1997.

$$
\begin{aligned}
N_{Addr} &= 16*(N_{hit} + N_{miss}) \\
N_{bp} &= (N_{hit} + N_{miss})*(T*M + C + 8*L*M) \\
N_{br} &= (N_{hit} + N_{miss})*(T*M + C + 8*L*M) \\
N_{bw} &= N_{readmiss}*(T*M + C + 8*L*M) + N_{writehit}*(C + W) \\
N_{memaddr} &= 16*(N_{readmiss} + N_{writemiss} + N_{writehit}) \\
N_{memdata} &= 0.5*(N_{writemiss} + N_{writehit})*W \\
N_{cpudata} &= 4*L_2*(N_{readmiss} + N_{readhit}) \qquad (\text{L2 cache onwards}) \\
&= 9.5*(N_{readmiss} + N_{readhit}) \qquad (\text{L1 cache})
\end{aligned}
$$

Equations 1

$$
\begin{aligned}
C_{bp} &= N_{rows}*(0.5*C_{drain,Q1} + C_{bitwire}) \\
C_{ba} &= N_{rows}*(0.5*C_{drain,Q1} + C_{bitwire}) + C_{drain,Qp} + C_{drain,Qpa} \\
C_{word} &= N_{columns}*(2*C_{gate,Q1} + C_{wordwire})
\end{aligned}
$$

Equations 2

$$E_{bit} = 0.5 * V_{dd}^{\ 2} * \begin{pmatrix} N_{bp} * C_{bp} + M * (N_{hit} + N_{miss}) * (8 * L + T + C) * (C_{gate,Qpa} + C_{gate,Qpb} + C_{gate,Qp}) + \\ N_{bw} * C_{ba} + N_{br} * C_{ba} \end{pmatrix}$$

$$E_{word} = V_{dd}^{\ 2} * (N_{hit} + N_{miss}) * (8 * L + T + C) * M * (2 * C_{gate,Q1} + C_{word})$$

$$E_{addr} = 0.5 * V_{dd}^{\ 2} * (N_{memaddr} * C_{memaddr} + N_{cpuaddr} * C_{cpuaddr})$$

$$E_{data} = 0.5 * V_{dd}^{\ 2} * (N_{memdata} * C_{memdata} + N_{cpudata} * C_{cpudata})$$

$$E_{output} = E_{addr} + E_{data}$$

$$E_{ainput} = 0.5 * V_{dd}^{\ 2} * N_{addr} * [(m+1) 2 S C_{in,dec} + C_{awire}]$$

$$E_{cache} = E_{bit} + E_{word} + E_{output} + E_{ainput}$$

Equations 3

Processor	Cache	Cache Power	CPU Power	Other	Process	Clock Speed	Die size
PowerPC	16KB unified 4-way	.4W	.29W	1.08W	0.8um	66 MHz	9x7.7 mm
StrongARM 110	16 KB split I/D, 32-way	0.215W	0.04W	0.245W	0.35um	162MHz	7.8x6.4mm

Table 1: Comparison of two modern embedded processor

APPLICATION	DESCRIPTION	Instruction Count
cjpeg	Image compression using DCT algorithm.	13,603,279
Decode	Voice decompression using the G.721 standard.	283,515,107
Djpeg	Image decompression using DCT algorithm.	4,242,741
Encode	Voice compression using the G.721 standard.	287,610,167
Epic	Data compression using wavelet decomposition and Huffman coding.	40,476,502
Gs	Interpreter for the Adobe PostScript language level 1 and 2.	102,608,385
Gsmdecode	Audio and speech decoding for the GSM standard.	88,339,462
Gsmencode	Audio and speech encoding for the GSM standard.	197,619,750
Mipmap	Texture Mapping using MESA 3-D graphics library.	27,174,530
mpeg2dec	Decoding in the standard MPEG digital compressed format.	130,126,836
mpeg2enc	Encoding in the standard MPEG digital compressed format.	1,063,842,098
Osdemo	Draws simple polygons with Z-Buffering.	7,345,397
Pgpdecode	PGP decoding exercising RSA, IDEA and MD5.	532,291,763
Pgpencode	Data encryption and signing using RSA, IDEA and MD5.	547,549,398
Rasta	Speech recognition.	8,238,012
Rawcaudio	Speech compression using the ADPCM algorithms.	7,213,464
Rawdaudio	Speech decompression using the ADPCM algorithms.	7,084,321
Texgen	Draws Texture Mapped Teapot using MESA 3-D graphics library.	70,462,746
Unepic	Epic decoding wavelets and Huffman coding.	6,952,632

Table 2: MediaBench application descriptions

Applications	128bytes					
	Direct Mapped			Fully Associative		
	8B	16B	32B	8B	16B	32B
cjpeg	0.3847	0.3930	0.4622	1.1264	0.9643	0.8945
decode	0.2688	0.3098	0.3849	1.0294	0.8778	0.8156
djpeg	0.4154	0.4579	0.5429	1.1403	1.0556	0.9876
encode	0.2755	0.3021	0.3689	1.0324	0.8710	0.8164
epic	0.2946	0.3218	0.4049	1.0608	0.8885	0.8136
gs	0.3648	0.3991	0.4670	1.1246	0.9578	0.8858
gsmdecode	0.2314	0.2552	0.3184	1.0050	0.8412	0.7641
gsmencode	0.4415	0.4632	0.5393	1.2014	1.0398	0.9753
mipmap	0.4370	0.4752	0.5586	1.2203	1.0293	0.9517
mpeg2dec	0.4144	0.4564	0.5403	1.1840	1.0298	0.9685
mpeg2enc	0.2863	0.3237	0.4239	1.0357	0.8733	0.7945
osdemo	0.3085	0.3221	0.3850	1.0767	0.9026	0.8337
pgpdecode	0.2473	0.2813	0.3687	1.0203	0.8435	0.7590
pgpencode	0.2473	0.2859	0.3760	1.0223	0.8448	0.7643
rasta	0.4414	0.4383	0.5066	1.1850	0.9719	0.8864
rawcaudio	0.2464	0.2776	0.3601	1.0093	0.8424	0.7693
rawdaudio	0.2510	0.2848	0.3658	1.0139	0.8525	0.7821
texgen	0.4379	0.4497	0.5174	1.1964	1.0064	0.9436
unepic	0.4018	0.4115	0.4776	1.1261	0.9412	0.8723
Mean	**0.3366**	**0.3636**	**0.4405**	**1.0953**	**0.9281**	**0.8567**

Table 3 Energy for 128-byte filter caches relative to base machine

Applications	256bytes					
	Direct Mapped			Fully Associative		
	8B	16B	32B	8B	16B	32B
cjpeg	0.3631	0.3708	0.4355	2.1506	1.6865	1.5055
decode	0.2535	0.2845	0.3527	2.0346	1.6156	1.4212
djpeg	0.4064	0.4348	0.5075	2.1727	1.7194	1.5948
encode	0.2533	0.2830	0.3481	2.0517	1.6221	1.4208
epic	0.2863	0.3124	0.3839	2.0517	1.6416	1.4457
gs	0.3500	0.3839	0.4511	2.1403	1.6980	1.5050
gsmdecode	0.2198	0.2469	0.3102	2.0294	1.5907	1.3913
gsmencode	0.4345	0.4532	0.5263	2.2317	1.7803	1.6001
mipmap	0.3947	0.4447	0.5237	2.1387	1.7745	1.5968
mpeg2dec	0.4162	0.4511	0.5289	2.1809	1.7693	1.5986
mpeg2enc	0.2728	0.3041	0.3865	2.0592	1.6237	1.4267
osdemo	0.3063	0.3167	0.3770	2.1157	1.6564	1.4519
pgpdecode	0.2477	0.2794	0.3644	2.0196	1.5762	1.3911
pgpencode	0.2498	0.2799	0.3622	2.0269	1.5825	1.3956
rasta	0.4269	0.4101	0.4630	2.2095	1.7180	1.4932
rawcaudio	0.2316	0.2604	0.3368	2.0218	1.5878	1.3978
rawdaudio	0.2343	0.2669	0.3396	2.0285	1.5944	1.4054
texgen	0.4153	0.4154	0.4884	2.1908	1.7259	1.5347
unepic	0.4037	0.4049	0.4617	2.1658	1.6989	1.4864
Mean	**0.3245**	**0.3475**	**0.4183**	**2.1063**	**1.6664**	**1.4770**

Table 4 Energy for 256-byte filter caches relative to base machineassociativity

Applications	128bytes					
	Direct Mapped			Fully Associative		
	8B	16B	32B	8B	16B	32B
cjpeg	1.8892	1.5706	1.4305	1.8511	1.5313	1.3891
decode	1.9346	1.5533	1.3697	1.9195	1.5348	1.3630
djpeg	2.0294	1.6573	1.4901	1.9936	1.6951	1.4905
encode	1.9689	1.5487	1.3751	1.9535	1.5473	1.3875
epic	1.1771	1.1614	1.1811	1.1820	1.1397	1.1352
gs	2.0095	1.6131	1.3957	2.0955	1.6235	1.3789
gsmdecode	2.0956	1.5658	1.3124	2.2942	1.6643	1.3544
gsmencode	2.1673	1.7063	1.4819	2.2905	1.7511	1.4831
mipmap	1.7432	1.3939	1.2395	1.7443	1.3640	1.1923
mpeg2dec	1.3402	1.3039	1.3044	1.4614	1.3472	1.2993
mpeg2enc	2.0420	1.6740	1.5454	2.0177	1.6019	1.3745
osdemo	1.5195	1.3101	1.1868	1.6615	1.3618	1.2041
pgpdecode	1.2337	1.2224	1.2102	1.2469	1.3134	1.1852
pgpencode	1.2093	1.2405	1.2301	1.2318	1.1958	1.1842
rasta	1.5640	1.3530	1.2803	1.5608	1.3173	1.2136
rawcaudio	2.2365	1.7859	1.5609	2.5169	1.8460	1.5266
rawdaudio	2.2694	1.8282	1.6590	2.5300	1.8511	1.5051
texgen	1.8624	1.4872	1.2927	1.9230	1.4925	1.2873
unepic	1.2503	1.1864	1.1655	1.2807	1.1763	1.1495
Mean	**1.7654**	**1.4822**	**1.3532**	**1.8292**	**1.4923**	**1.3212**

Table 5 Performance for 128-byte filter caches relative to base machine

Applications	256bytes					
	Direct Mapped			Fully Associative		
	8B	16B	32B	8B	16B	32B
cjpeg	1.6303	1.3888	1.2845	1.6145	1.3524	1.2600
decode	1.6719	1.4029	1.2703	1.8569	1.4819	1.2874
djpeg	1.9136	1.5590	1.3940	1.8366	1.4463	1.3948
encode	1.7043	1.4142	1.2827	1.8881	1.4876	1.2969
epic	1.1393	1.1284	1.1360	1.0726	1.1079	1.1097
gs	1.7708	1.4683	1.2953	1.7123	1.3968	1.2453
gsmdecode	1.0742	1.0458	1.0327	1.2129	1.1168	1.0631
gsmencode	1.8082	1.4995	1.3488	1.7939	1.4952	1.3646
mipmap	1.6104	1.3261	1.2057	1.6273	1.3307	1.1884
mpeg2dec	1.2846	1.2533	1.2501	1.2086	1.2460	1.2271
mpeg2enc	1.7416	1.4820	1.3851	1.9324	1.5180	1.3082
osdemo	1.2742	1.1580	1.0985	1.3149	1.1687	1.1047
pgpdecode	1.1765	1.1280	1.1266	1.1472	1.0824	1.0653
pgpencode	1.1846	1.1335	1.1335	1.1400	1.0899	1.0760
rasta	1.4322	1.2527	1.1798	1.4200	1.2225	1.1242
rawcaudio	1.3163	1.1906	1.2211	1.1772	1.2672	1.3882
rawdaudio	1.1148	1.0987	1.1013	1.0066	1.0060	1.2066
texgen	1.7443	1.3888	1.2197	1.7089	1.3490	1.1950
unepic	1.1351	1.1002	1.0914	1.0982	1.0624	1.0438
Mean	**1.4593**	**1.2852**	**1.2135**	**1.4615**	**1.2751**	**1.2079**

Table 6 Performance for 256-byte filter caches relative to base machine

Applications	128bytes					
	Direct Mapped			Fully Associative		
	8B	16B	32B	8B	16B	32B
cjpeg	0.7269	0.6173	0.6611	2.0851	1.4766	1.2425
decode	0.5201	0.4812	0.5272	1.9759	1.3472	1.1117
djpeg	0.8430	0.7589	0.8089	2.2733	1.7894	1.4720
encode	0.5425	0.4679	0.5072	2.0168	1.3477	1.1328
epic	0.3468	0.3737	0.4783	1.2539	1.0127	0.9236
gs	0.7330	0.6438	0.6518	2.3566	1.5550	1.2214
gsmdecode	0.4848	0.3996	0.4179	2.3058	1.3999	1.0350
gsmencode	0.9569	0.7903	0.7992	2.7517	1.8207	1.4466
mipmap	0.7618	0.6624	0.6924	2.1286	1.4040	1.1346
mpeg2dec	0.5554	0.5952	0.7048	1.7302	1.3873	1.2583
mpeg2enc	0.5847	0.5418	0.6551	2.0897	1.3989	1.0921
osdemo	0.4688	0.4221	0.4570	1.7890	1.2292	1.0038
pgpdecode	0.3051	0.3439	0.4462	1.2723	1.1079	0.8996
pgpencode	0.2990	0.3547	0.4625	1.2592	1.0102	0.9051
rasta	0.6904	0.5930	0.6486	1.8495	1.2803	1.0758
rawcaudio	0.5510	0.4959	0.5621	2.5403	1.5551	1.1745
rawdaudio	0.5696	0.5207	0.6068	2.5651	1.5782	1.1772
texgen	0.8154	0.6688	0.6688	2.3007	1.5020	1.2146
unepic	0.5024	0.4882	0.5567	1.4421	1.1071	1.0027
Mean	**0.5924**	**0.5378**	**0.5954**	**1.999**	**1.3847**	**1.1328**

Table 7 Energy•Delay product ratio with 128-byte filter caches

Applications	256bytes					
	Direct Mapped			Fully Associative		
	8B	16B	32B	8B	16B	32B
cjpeg	0.5920	0.5149	0.5594	3.4720	2.2807	1.8969
decode	0.4239	0.3992	0.4480	3.7779	2.3941	1.8297
djpeg	0.7776	0.6779	0.7074	3.9903	2.4868	2.2245
encode	0.4318	0.4003	0.4465	3.8738	2.4131	1.8426
epic	0.3262	0.3525	0.4361	2.2007	1.8187	1.6043
gs	0.6197	0.5637	0.5843	3.6648	2.3717	1.8741
gsmdecode	0.2361	0.2582	0.3204	2.4615	1.7765	1.4790
gsmencode	0.7856	0.6796	0.7098	4.0036	2.6618	2.1836
mipmap	0.6356	0.5898	0.6314	3.4802	2.3613	1.8977
mpeg2dec	0.5347	0.5653	0.6612	2.6358	2.2047	1.9617
mpeg2enc	0.4752	0.4507	0.5353	3.9791	2.4647	1.8665
osdemo	0.3903	0.3668	0.4141	2.7820	1.9358	1.6039
pgpdecode	0.2914	0.3152	0.4105	2.3169	1.7061	1.4820
pgpencode	0.2959	0.3172	0.4106	2.3107	1.7248	1.5016
rasta	0.6114	0.5138	0.5462	3.1374	2.1002	1.6786
rawcaudio	0.3049	0.3100	0.4113	2.3800	2.0120	1.9404
rawdaudio	0.2612	0.2933	0.3740	2.0420	1.6039	1.6958
texgen	0.7244	0.5768	0.5957	3.7439	2.3282	1.8340
unepic	0.4582	0.4455	0.5039	2.3785	1.8050	1.5515
Mean	**0.4829**	**0.4521**	**0.5108**	**3.0858**	**2.1289**	**1.7867**

Table 8 Energy•Delay product ratio with 256-byte filter caches