Dynamic Techniques to Reduce Memory Traffic in Embedded Systems*

Ben Juurlink
B.H.H.Juurlink@ewi.tudelft.nl

Pepijn de Langen
P.J.deLangen@ewi.tudelft.nl

Computer Engineering Laboratory
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology
P.O. Box 5031, 2600 GA Delft
Netherlands

ABSTRACT

Memory transfers, in particular from/to off-chip memories, consume a significant amount of power. In order to reduce the amount of off-chip memory traffic, one or more levels of cache can be employed, located on the same die as the processor core. For performance, energy, and cost reasons, it is expedient that the on-chip cache is small and direct-mapped. Small, direct-mapped caches, however, generally produce much more traffic than needed. The purpose of this paper is two-fold. First, to measure how much traffic is generated by small, direct-mapped caches and what the minimal amount of traffic is. This yields an upper bound on the amount of traffic that can be saved by utilizing the on-chip memory more effectively. Second, we survey some techniques that can be deployed to reduce the amount of traffic produced by direct-mapped caches and present results for some of these techniques.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles—*Cache memories*; C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems

General Terms

Design, measurement, performance.

Keywords

Memory traffic, embedded processors, caches, power consumption.

*This research was supported in part by the Netherlands Organisation for Scientific Research (NWO).

CF'04, April 14–16, 2004, Ischia, Italy.

1. INTRODUCTION

Many new mobile devices such as Personal Digital Assistants and cellular phones with new capabilities are currently being introduced on the market. Since these devices are powered by batteries, energy is very precious and, therefore, reducing the energy consumption of embedded processors employed in such devices has become an important research area.

For many of these embedded devices, off-chip memory accesses consume more energy than the datapaths and the control units. For example, in [20] it is estimated that a 16-bit external I/O access requires approximately 180pJ per operation, which is 10 times more than the energy consumed by a 16-bit add. (A 0.8μm CMOS technology and a supply voltage of 1.5V were assumed.) It is also shown that external memory accesses consume approximately 32% of the total power of a subband decoder chip, internal memory dissipates about 33% of the the total power, the datapath approximately 28%, and the control section only a small fraction (approximately 7%). As another example, in [31] it is stated that "architecture experiments have shown that between 50 and 80% of the power and area cost in (application-specific) architectures for multi-dimensional real-time signal processing is due to *memory units*, i.e., single or multi-port RAMs, pointer-addressed memories, and register files." More recently, Simunic et al. [27] have modeled the energy consumption of a SmartBadge while decoding an MPEG video. A SmartBadge is an embedded system consisting of a StrongARM-1100 processor, FLASH and SRAM memories, sensors, and a modem/audio analog front-end on a PCB board powered by batteries through a DC-DC converter. The simulation results show that when a low-power SRAM is used to implement the data memory, it consumes approximately 30% of the total energy and is responsible for more than 75% of the execution time. Furthermore, using a 4-way set-associative, burst SDRAM L2 cache in addition to the L1 cache improves performance by approximately 10% but almost doubles the total energy consumption.

Given these numbers, a large saving in energy consumption can be attained by making effective use of on-chip memory. A common method to reduce the number of off-chip memory accesses is to employ a large on-chip (L1) cache and/or to increase the cache associativity. However, large and/or associative caches should be avoided if possible be-

cause they are expensive both in terms of area as well as power consumption and may increase the cycle time. Furthermore, in [21] it is calculated that 25% of the cost of an embedded system can be saved by employing a 1MB instead of a 2MB secondary cache.

The purpose of this paper is two-fold. First, we investigate how much traffic is generated by conventional, direct-mapped caches and compare this to the amount of traffic produced by a hypothetical minimal-traffic cache. This gives an upper bound on the amount of traffic that can be saved by utilizing the on-chip cache more effectively. Similar experiments have been performed by Burger et al. [3], but we use a benchmark suite that is more representative of embedded applications, consider request traffic also, and use a different metric. The results show that there are two reasons why direct-mapped caches generate much more traffic than necessary. First, because each word is mapped to exactly one cache block, direct-mapped caches generally incur many conflict misses. Second, employing large blocks in order to exploit spatial locality in combination with zero associativity implies that a lot of data is loaded in the cache that is never used before it is replaced.

The second purpose of this paper is to present a survey of techniques that can be employed to reduce the amount of processor-memory traffic. We focus on dynamic, runtime techniques. Given that (lack of) associativity and the block size are the two factors that contribute the most to the inefficiency of direct-mapped caches, we review techniques that attempt to reduce conflict misses and/or try to improve spatial reuse by varying the block size adaptively. Results are presented for some of these techniques.

The remainder of this paper is organized as follows. Related work is described in Section 2. In Section 3 we measure how much traffic is generated by direct-mapped caches for some benchmarks representative of applications that run on battery-powered, embedded devices. In addition, we measure how much traffic is produced by a hypothetical minimal traffic cache. A survey of dynamic techniques that may be used to reduce off-chip memory traffic is presented in Section 4. Finally, in Section 5, some conclusions are drawn.

2. RELATED WORK

The work on energy-efficient embedded processors can be broadly classified as static (at the compiler level) and dynamic (at the hardware or processor architecture level). In this section we briefly review some of this work.

The Data Transfer and Storage Exploration (DTSE) methodology [4, 5] developed at IMEC focuses on compile-time techniques. One of the steps in this methodology aims at improving locality by applying loop transformations such as loop interchange and loop splitting and merging. In another step the restricted lifetimes of parts of array variables is exploited to overlap them in the address space. The leads to better cache performance and also reduces the total size of the required memories. While very good results have been achieved using the DTSE methodology (speedups of up to a factor of 3 and reductions in bus load by an order of magnitude have been reported), the approach has two limitations. First, the methodology is only partially supported by tools and, hence, requires manual intervention which increases the design complexity. Second, it is unclear if it can be applied to dynamic, pointer-based applications.

Panda et al. [22] investigated how a scratchpad memory can be used to better utilize the cache. A scratchpad memory is a small, on-chip random access memory that is controlled by software. Amongst others, they consider the following problem: given an embedded processor with a scratchpad memory, a cache, and an off-chip DRAM, which array variables should be stored in the scratchpad memory and which should be mapped to the off-chip DRAM and accessed through the cache?

Petrov and Orailoglu [23] proposed a static methodology for application-specific customization of the data cache of embedded processors. They present an algorithm that partitions memory access instructions into groups of instructions that exhibit data reuse amongst them. Each group is mapped to a certain partition of the cache, which allows to isolate them from possibly interfering groups. Furthermore, the tag comparison can be avoided if it can be determined that a reference will invariably hit the cache.

A lot of work has been performed on power-efficient cache designs. Since large caches consume more power per access than small caches, Kin et al. [16] proposed to employ a small *filter cache* between the processor and the primary level-1 (L1) cache. Although this level-0 cache incurs many misses, it filters enough accesses to the more power-consuming L1 cache to decrease the overall power dissipation. Lee and Tyson [18] showed that accesses to the stack, global, and heap segments exhibit different characteristics and, therefore, proposed to partition the cache by these memory regions.

The *static* power dissipation in caches can be reduced by deactivating cache lines. A deactivated cache line loses its data but dissipates less static power. Examples of works in this field are the DRI cache [32], cache line decay [15], and Adaptive Mode Control [33].

We remark that cache structures aiming at reducing conflict misses and techniques that dynamically vary the amount of data fetched into the cache are discussed in Section 4.

3. AMOUNT OF TRAFFIC GENERATED BY DIRECT-MAPPED CACHES

In this section we determine how much traffic is generated by direct-mapped caches and what the minimal amount of traffic is. In addition, we attempt to isolate the factors that contribute to the traffic inefficiency.

3.1 Background

In order to evaluate how much traffic can be saved by making effective use of the cache memory, we measure *traffic inefficiency*, which is defined as the ratio of the amount of traffic generated by the cache considered to the amount of traffic produced by a minimal traffic cache (MTC) with equal capacity. All caches use *write-back*, since this will produce less traffic than *write-through* for all but some anomalous cases (e.g., when writes hardly ever hit the cache). Furthermore, we assume *write-allocate* caches.

The MTC is fully associative, employs Belady's optimal, off-line replacement policy MIN, and has a block size equal to the word size of the processor (four bytes). Furthermore, if all blocks currently cached are accessed nearer in the future than the requested block, the MTC is bypassed. We remark that the MTC is not a true lower bound for two reasons. First, OPT is optimal for write-through caches but not necessarily for write-back caches. As Burger et al. [3]

observed, however, the discrepancy between OPT and the optimal algorithm in the presence of write-backs is small. Second, since request traffic is not ignored and most significant when the block size is small, the MTC with a block size of 4 bytes may actually produce more traffic than a cache with a larger block size.

We assume that the L1 cache is located on-chip and that the lower cache levels (if present) and main memory are situated off-chip. Off-chip traffic is, therefore, defined as the total number of bytes in and out of the L1 cache. Traffic is the sum of data traffic and request traffic. Data traffic is calculated as follows. Cache hits do not produce any data traffic. For every cache miss, an amount of data equal to the cache block size needs to be fetched. Furthermore, if the evicted block is dirty, it needs to be written back to the next memory level. We measured write-back data traffic by counting the number of writes to clean blocks. Note that is possible to associate a dirty bit with every word in a cache block and to write back only the dirty words, but we assumed that entire cache blocks are written back. The amount of request traffic is equal to the amount of data traffic divided by the number of words per cache block.

We implemented a trace-driven cache simulator to measure how much traffic is produced. Our implementation of the optimal replacement algorithm is based on the algorithm described in [28]. The `sim-safe` simulator from the SimpleScalar toolset was modified to produce traces of memory references.

3.2 Benchmarks

For our experiments, we have selected a number of benchmarks representative of applications that run on battery-powered embedded devices. In other words, an embedded system must exist or be in active development, on which similar software is expected to be used, while the corresponding embedded system is battery-powered. Typically, a lot of these applications are from the field of multimedia.

All benchmarks, except one, have been selected from two different sources: the *MediaBench* suite [17] and the *MiBench* suite [9]. The first is mostly aimed at applications from the field of multimedia, while the second consists of typical embedded applications. Both suites share a number of benchmarks.

From *MediaBench*, the following benchmarks have been selected:

ADPCM (*author: Jack Jansen*)
Adaptive Differential Pulse Code Modulation. Speech compression and decompression, using the Intel/DVI ADPCM code. Note that this code differs from the CCITT ADPCM standard. Due to its high speech quality, low delay, and moderate complexity, this is one of the most widely used speech compression techniques.

JPEG (*author: Independent JPEG Group*)
Lossy image compression. Both compression (encoding) as decompression (decoding) benchmarks are used. This application is typically found in digital photo cameras, but also in other devices that use digital images.

MPEG (*author: MPEG Software Simulation Group*)
Encoding/decoding of a MPEG-2 video bitstream. The decoding is representative of portable video players, while the encoding is found on modern digital video cameras.

GSM (*author: The Communications and OS Research Group at the Technische Universität Berlin*)
GSM 06.10 full-rate speech transcoding at 13 kbit/s, using residual pulse excitation/long term prediction coding (RPE/LTP). Both encoding and decoding routines are used. Widely used for cellular telephony.

Ghostscript (*author: Aladdin Software*)
A Postscript™ interpreter. Representative of document formatting applications, often found on Personal Digital Assistants (PDA's).

From *MiBench*, we have selected:

Rijndael (*author: Brian Gladman*)
Rijndael is a block cipher, designed by Joan Daemen and Vincent Rijmen. It was selected by the National Institute of Standards and Technology to be the Advanced Encryption Standard (AES). Therefore, these routines are representative for devices that support block encryption.

And the only benchmark that does not belong to either *MiBench* or *MediaBench*:

libDV (*authors: Erik Walthinsen and Charles Krasic*)
An implementation of the Digital Video (DV) codec, released under the GNU Library General Public License. The latest version (0.98) has been used and the input consists of the same frames as were used for MPEG2 encoding. This application was selected because it implements the most widely used codec on digital camcorders.

Benchmark	suite	input
ADPCM encode	MediaBench	`clinton.pcm`
ADPCM decode	MediaBench	`clinton.adpcm`
GSM encode	MediaBench	`clinton.pcm`
GSM decode	MediaBench	`clinton.pcm.gsm`
JPEG encode	MediaBench	`testimg.ppm`
JPEG decode	MediaBench	`testimg.jpg`
MPEG2 encode	MediaBench	3 frames defined in `options.par`
MPEG2 decode	MediaBench	`meiv16v2.m2v`
AES encode	MiBench	`input_large.asc`
AES decode	MiBench	`output_large.enc`
DV encode	-	Same frames as MPEG2 encode

Table 1: The benchmarks, their origin, and the used inputs.

In Table 1 all benchmarks and the used input datasets are listed, as well as the benchmark suites from which they were taken.

3.3 Experimental Results

In this section the experimental results are presented. First, the traffic inefficiencies of direct-mapped caches of various sizes are shown. After that, we investigate which factors contribute the most to the traffic inefficiency.

Benchmark	256B	512B	1kB	2kB	4kB	8kB	16kB	32kB
adpcm-dec	6.51	3.33	2.72	2.36	56.21	2.45	0.62	0.62
adpcm-enc	7.92	4.93	3.97	3.44	47.05	2.57	0.65	0.65
jpeg-dec	6.22	6.18	5.85	6.54	4.36	3.07	5.20	3.64
jpeg-enc	6.40	6.49	6.28	4.97	3.91	1.37	0.84	0.74
mpeg2-dec	14.66	16.81	26.53	40.81	18.72	13.49	3.61	1.33
mpeg2-enc	11.63	26.58	47.27	50.99	50.65	48.06	22.24	9.72
aes-dec	6.61	7.19	8.10	10.13	11.76	25.46	81.14	76.37
aes-enc	6.08	6.40	7.78	9.81	11.19	15.45	43.77	47.63
gsm-dec	3.31	3.17	4.01	2.77	3.34	9.82	0.78	0.76
gsm-enc	19.97	18.73	2.07	5.55	3.72	8.95	1.76	1.74
dv-enc	2.90	2.82	2.43	2.39	2.43	2.68	1.60	1.14
gs-dec	14.55	13.05	11.35	13.69	18.63	7.05	5.28	3.14
Geometric Mean	7.635	7.457	6.627	7.411	11.008	6.878	3.662	2.809

Table 2: Traffic inefficiencies for direct-mapped caches with a block size of 32 bytes.

3.3.1 Traffic Inefficiency

Table 2 depicts the traffic inefficiencies of direct-mapped caches with a block size of 32 bytes. The cache size varies from 256 bytes to 32 kilobytes. Several observations can be drawn from these results.

First, it can be seen that there is a large variance across the benchmarks. For most benchmarks the inefficiency is typically between 1 and 10. For some benchmarks, however, in particular `mpeg2-dec`, `mpeg2-enc`, and `aes-dec` and `aes-enc` for the largest cache sizes, the inefficiency is typically larger than 10.

Second, when the cache size is 16 or 32kB, the traffic inefficiency is smaller than 1 for some benchmarks. As explained before, since our measurements also include request traffic, a cache with smaller blocks can generate more traffic than a cache with larger blocks. Since the MTC has a block size of four bytes, it can actually produce more traffic than a cache with a larger block size.

Third, it can be observed that for some benchmarks the inefficiency exhibits non-monotonic behavior. In particular for `adpcm-dec` and `adpcm-enc` at caches of 4kB, the inefficiency exhibits large peaks. This can be explained more clearly by looking at Figure 1, which depicts the amount of traffic as a function of the cache size for three benchmarks which are representative of the observed behavior. The results for direct-mapped caches with different block sizes are shown as well as the minimal traffic cache. In these graphs, the curve labelled `DM-n` denotes a direct-mapped cache with a block size of `n` bytes.

It can be seen that for `adpcm-dec` (Figure 1(a)) when moving from the 2kB MTC to the 4kB MTC, the amount of produced traffic decreases enormously (by more than a factor of 8). Furthermore, when the MTC is larger than 4kB, the traffic volume does not decrease any further. This shows that the MTC, because it is fully associative and employs the optimal replacement algorithm, is able to contain the entire data set. The direct-mapped caches, on the other hand, are unable to hold the entire data set until their capacities exceed 16kB. This explains the large jump in traffic inefficiency. Figure 1(a) also shows that eventually, when the cache size is larger than 16kB, the MTC and the 4-byte block, direct-mapped cache produce the same amount of traffic.

Such large peaks in the traffic inefficiency are not encountered for, e.g., `dv-enc` (Figure 1(b)). For this and other benchmarks such as `jpeg-dec`, the different configurations benefit almost equally from an increase in cache size. Because of this, the inefficiency hardly depends on the cache size, as can be seen in Table 2.

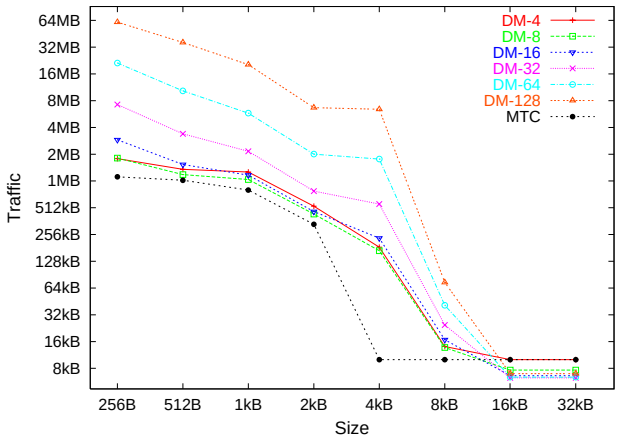
As depicted in Figure 1(c), especially for `mpeg2-enc` there is a huge difference between the amount of traffic produced by direct-mapped caches and the amount of traffic generated by the MTC. Whereas the MTC achieves good results already for a capacity of 2kB, the 32kB direct-mapped caches are still unable to keep a large part of the data set in cache. It can also be observed from Figure 1(c) that `mpeg2-enc` either exhibits little spatial locality or that the spatial locality cannot be exploited because the blocks are replaced before they are re-used, since any increase in block size causes an almost corresponding increase in traffic. Because request traffic is included and most significant when the cache blocks are small, the gaps between the direct-mapped cache with 4-byte blocks and the one with 8-byte blocks are slightly smaller than, for example, the gaps between the 8-byte and 16-byte block, direct-mapped caches.

Overall, the results indicate that direct-mapped caches produce much more traffic than needed. As depicted in the last row of Table 2, the geometric mean of all inefficiencies over all benchmarks is between 2.8 and 10.5. This demonstrates that large traffic reductions can be achieved by better utilizing the on-chip cache.

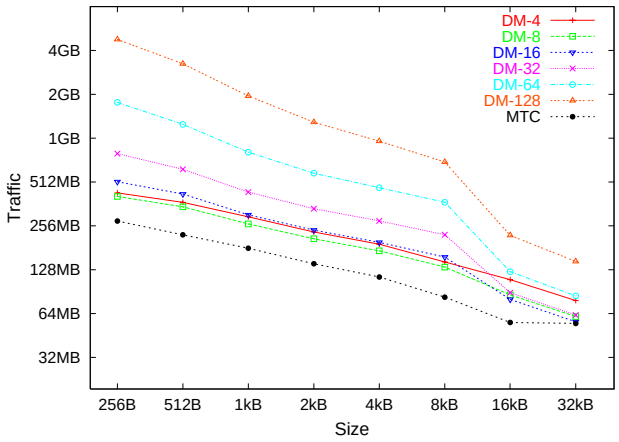
3.3.2 Contribution of Different Factors

The block size, the associativity, and the replacement strategy all contribute to the traffic inefficiency. Although their contributions are not independent (for example, as the results will show, the block size is mainly relevant for direct-mapped caches and hardly for fully associative caches), in this section we attempt to isolate their contribution by comparing cache configurations that differ in only one parameter. The cache configurations compared with each other are described in Table 3.

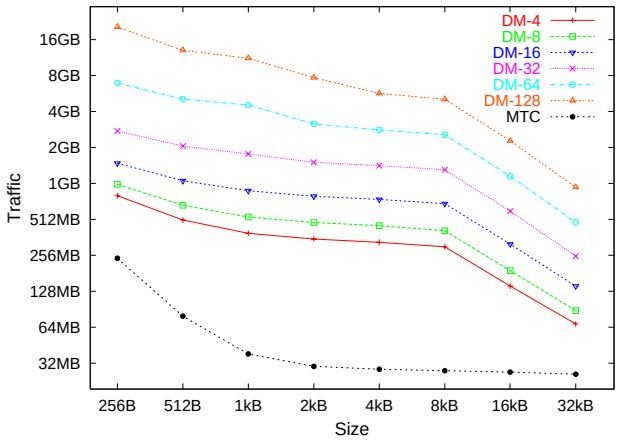
To calculate the contribution of single cache parameter such as the block size, we use a metric that is different from the one that Burger et al. [3] employed. They subtracted two inefficiencies to obtain the *inefficiency gap*. We, on the other hand, use the ratio since this provides more information on



(a) adpcm-dec



(b) dv-enc



(c) mpeg2-enc

Figure 1: Amount of traffic as a function of the cache size for different benchmarks.

	Assoc., repl. policy, block size	
Factor	**Cache 1**	**Cache 2**
I. Associativity	direct-mapped, n/a, 32B	fully associative, LRU, 32B
II. Replacement	fully associative, LRU, 32B	fully associative, MIN, 32B
III. Block size (DM)	direct-mapped, n/a, 32B	direct-mapped, n/a, 4B
IV. Block size (MTC)	fully associative, MIN, 32B	fully associative, MIN, 4B

Table 3: Cache configurations compared to isolate the factors that contribute to the traffic inefficiency.

the influence of each parameter. Assume, for example, that a cache C_1 is compared to a cache C_2, that for one benchmark the traffic inefficiency of C_1 is 2 and that of C_2 1, and for another benchmark the traffic inefficiencies are 100 and 50, respectively. Then Burger et al. define the inefficiency gap for the first benchmark as 1 and for the second as 50. Our metric, on the other hand, will be 2 in both cases, since in both cases cache C_1 produces twice as much traffic as C_2. To avoid confusion, we refer to our metric as the *inefficiency ratio*.

The inefficiency ratios for cache sizes of 2, 4, and 8kB are presented in Table 4. From these results the total traffic inefficiency can be computed as:

$$\textit{traffic inefficiency} = F_{assoc} \times F_{repl} \times F_{blk\_size\_MTC},$$

where F_{assoc}, F_{repl}, and $F_{blk\_size\_MTC}$ denote the factors listed in the columns labelled Associativity, Replacement, and Block Size (MTC), respectively.

The first factor we consider is associativity. To compute the contribution of this factor to the traffic inefficiency, a 32-byte block, direct-mapped cache is compared to a 32-byte block, fully associative cache with LRU replacement. As can be seen in Table 4, associativity makes a significant contribution to the total traffic inefficiency. The geometric mean over all benchmarks is 2.35, 5.35, and 6.38 for caches of 2kB, 4kB, and 8kB, respectively. Furthermore, we again observe that for some benchmarks the inefficiency ratio increases considerably when going to a larger cache size. This is once more due to the fact that the fully associative cache is able to keep a significant part of the working set in cache whereas the direct-mapped cache is not. When both caches can hold the entire data set, the inefficiency ratio due to limited associativity will be 1. We also observe that for `aes-dec` and `aes-enc` with a 4kB cache size the fully associative cache produces more traffic than the direct-mapped cache. It is known that LRU sometimes performs worse than a direct-mapped cache.

The second factor is the replacement strategy. To determine the influence of this factor, a 32-byte block, fully associative cache with LRU replacement is compared to the same cache that employs the optimal replacement algorithm. The results show that, in general, using the MIN replacement algorithm has the smallest effect on the traffic volume. However, when the cache size is 2kB, its contribution is larger than the impact of associativity for half of the benchmarks. Considering that most caches have limited associativity, these results indicate that improvements to the

Benchmark	Associativity			Replacement			Block Size (DM)			Block Size (MTC)		
Cache size	2kB	4kB	8kB	2kB	4kB	8kB	2kB	4kB	8kB	2kB	4kB	8kB
adpcm-dec	1.16	90.03	3.92	2.89	1.00	1.00	1.48	3.06	1.74	0.71	0.62	0.62
adpcm-enc	1.21	72.39	3.95	4.20	1.00	1.00	1.32	4.57	1.80	0.68	0.65	0.65
jpeg-dec	3.00	1.96	2.72	1.59	2.06	1.41	2.59	1.84	1.24	1.37	1.08	0.80
jpeg-enc	3.83	2.96	1.36	1.37	1.47	1.57	2.87	2.47	0.96	0.95	0.90	0.64
mpeg2-dec	9.34	6.07	14.18	2.26	2.64	1.58	5.87	5.21	4.66	1.93	1.17	0.60
mpeg2-enc	24.02	50.15	61.70	1.88	1.50	1.30	4.39	4.38	4.41	1.13	0.67	0.60
aes-dec	1.09	0.80	11.89	2.01	3.75	2.43	3.82	3.55	3.39	4.61	3.90	0.88
aes-enc	1.06	0.78	7.41	2.02	3.80	2.46	3.87	3.81	4.49	4.60	3.78	0.85
gsm-dec	1.02	2.41	11.43	4.24	1.92	1.29	1.16	1.29	1.87	0.64	0.72	0.66
gsm-enc	1.04	4.15	10.79	3.12	1.35	1.23	1.23	1.22	1.28	1.72	0.66	0.67
dv-enc	2.11	2.26	2.34	1.46	1.51	1.60	1.44	1.45	1.54	0.77	0.71	0.72
gs-dec	3.06	3.39	3.57	1.68	2.41	1.98	4.32	4.50	3.04	2.67	2.28	1.00
G. Mean	2.352	5.350	6.380	2.226	1.850	1.506	2.460	2.767	2.203	1.417	1.110	0.714

Table 4: Inefficiency ratios for different factors and cache sizes.

LRU algorithm will probably have a limited effect.

The third parameter we consider is the block size. For both the direct-mapped cache and the MTC, we calculate the ratio of the traffic volume of the 32-byte block variant to the amount of traffic produced by the 4-byte block variant. As can be seen in Table 4, the block size is mainly relevant for the direct-mapped cache. In fact, when the cache capacity is 8kB, the 32-byte block MTC generates less traffic than the 4-byte block variant for 7 of the 12 benchmarks. The reason is that request traffic is included and that the 8kB MTC, being fully associative, is able to keep a large working set in cache. The direct-mapped cache, on the other hand, cannot select where to cache data, so large blocks potentially replace frequently used data.

Overall, associativity is the most important factor. We remark that this observation is different from the work of Burger et al. [3], where the most important factor was found to be the block size. A plausible explanation is that we have included request traffic whereas Burger et al. have not. The block size also makes a significant contribution, at least for direct-mapped caches, whereas using the MIN replacement policy has the smallest effect. This motivates us to study techniques that aim at reducing conflict misses in direct-mapped caches and mechanisms that support variable block sizes in the next section.

4. DYNAMIC TECHNIQUES TO REDUCE TRAFFIC: A SURVEY

The previous section has established that direct-mapped caches produce much more data traffic than needed and, hence, that there is potential for large improvements. It was shown that the (lack of) associativity and the block size are the two factors that contribute the most to the inefficiency. In this section we briefly survey some dynamic, μ-architectural techniques that attempt to resolve the problems associated with these two factors. For some of these cache structures, data traffic measurements are provided.

4.1 Techniques Aiming at Reducing Conflict Misses

One way to reduce the amount of traffic that emanates from conflict misses is to employ a set-associative cache. Set-associative caches, however, are slower and consume significantly more power than direct-mapped caches [25]. Consequently, the energy saved by decreasing the amount of off-chip memory traffic might be lost again because the on-chip cache organization consumes more energy. A more promising approach is the *Pseudo Set-Associative Cache* (PSAC) [1]. A PSAC can be thought of as a set of smaller caches (each corresponding to way) that can be probed independently. Accessing one of these small caches is likely to be faster and less energy consuming than accessing a conventional set-associative cache organization. Some kind of prediction mechanism is used to determine which way to probe first. A common way prediction algorithm is MRU (Most Recently Used). Thus, a PSAC has a fast hit time and a slow hit time. Inoue et al. [11] and Huang et al. [10] analyzed the energy efficiency of several PSAC designs.

Another way to eliminate conflict misses in direct-mapped caches is to employ a small (typically four or eight entries), fully associative *victim cache* [13]. Blocks evicted from the primary cache are not immediately written back to the next memory level but are given a second chance in the victim cache. Albera and Bahar [2] and Memik et al. [19] proposed to use victim cache structures to reduce the number of accesses to more power consuming structures such as the level-2 cache. Originally, the victim cache is placed between the primary cache and the next memory level. Albera and Bahar proposed to access the primary cache and the victim cache in parallel. Memik et al. presented techniques to detect cache misses early and proposed to bypass the victim cache if it is known that an access is going to miss.

One reason why direct-mapped caches sometimes perform poorly is conflicts between data structures (in particular arrays) that are accessed simultaneously with the same stride and whose base addresses differ by a power of two. González et al. [8] proposed to use XOR-based mapping functions to eliminate such conflicts. In an XOR mapping scheme, the cache index is obtained by XORing two fields of the memory address. Gonzalez et al. showed that XOR direct-mapped caches perform slightly better than conventional direct-mapped caches. However, if the XOR mapping scheme is used in conjunction with a two-way set-associative cache, the average miss ratio is improved by more than a factor of 2. In fact, the performance of a two-way set associative

XOR cache approaches the performance of a fully associative cache.

Recently, we proposed two novel cache organizations that reduce processor-memory traffic by detecting and avoiding conflict misses [6]. Both employ a table called the *Conflict Detection Table* (CDT) that contains the tag field of the addresses referenced by recently executed load/store instructions. The idea behind the CDT is the following. If an entry corresponding to a load/store instruction is found in the table and the tag stored in this entry matches the tag of the current address, a (spatial) hit is expected because the requested word was loaded the previous time this instruction was executed. If a hit is expected but a miss occurs, a conflict is detected because the requested word must have been replaced by data loaded by a different instruction.

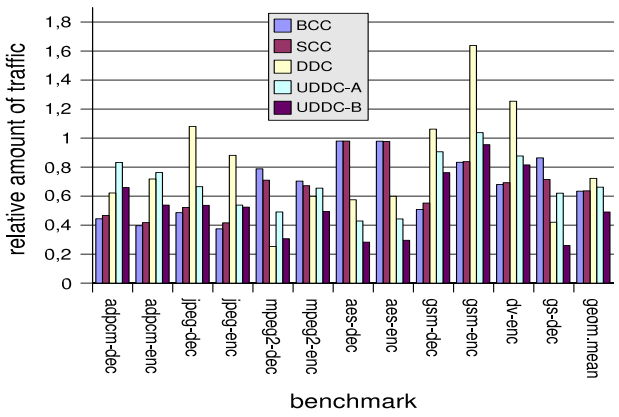
When a conflict is detected, several actions can be taken. The *Bypass in Case of Conflict* (BCC) cache decides to bypass the cache when there is a conflict. So, until an instruction references another cache line, the data accessed by this instruction will not be cached. The *Sub-block in Case of Conflict* (SCC) cache is a sector cache that caches only the missing sector (or sub-block) when a conflict is detected. Both the BCC as well as the SCC are direct-mapped.

Figure 2 depicts the amount of traffic generated by 1kB, 2kB, and 4kB BCC and SCC caches, relative to the amount of traffic produced by a 32-byte block, direct-mapped cache of equal capacity. It can be seen that in general both the BCC as well as the SCC reduce the amount of traffic significantly. For example, when the cache size if 4kB, the SCC achieves a traffic reduction by a factor of between 1.02 and 3.23 compared to the direct-mapped cache, with a geometric mean of 1.56. The geometric mean of the traffic reduction achieved by the BCC is a factor of 1.49, but this cache produces slightly more traffic than the direct-mapped cache for one benchmark (`gs-dec`). Of course, it remains to be investigated if this translates to reduced energy consumption, since the CDT needs to be accessed during every cache access. However, since this table can be small, it need not be very energy consuming and, furthermore, it is indexed by the program counter which is available very early in the pipeline.

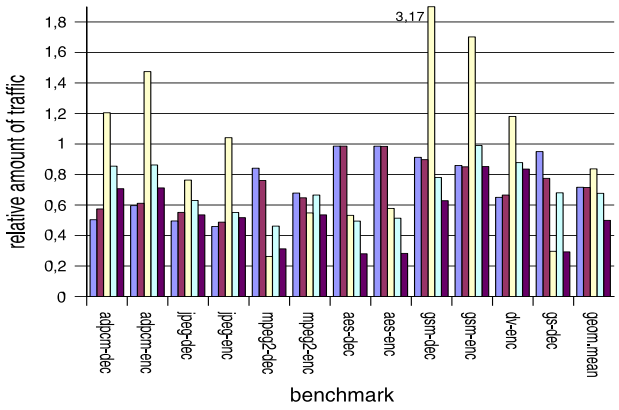
4.2 Techniques With Adaptive Block Size

Besides conflict misses, another reason why conventional caches sometimes generate much more traffic than needed is large blocks. Not considering request traffic, a cache with smaller blocks will always produce less traffic than a cache with larger blocks because data is fetched on demand and not implicitly prefetched without knowing if they are going to be used. This indicates that it might be advantageous to fetch small blocks if data exhibits temporal locality only and large blocks if it (also) exhibits spatial locality. In this section we, therefore, review some cache designs that support multiple line sizes.

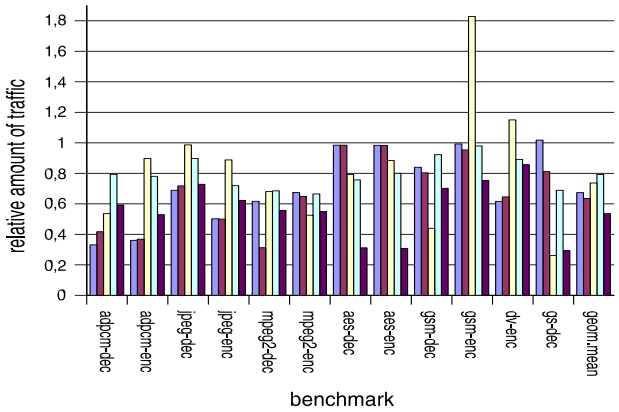
In the Split Temporal/Spatial (STS) cache [24] the cache is split into a temporal subcache (with a block size of one word) and a spatial subcache (with a block size of four words). Two counters are associated with each block in the spatial subcache to determine if only half of the block is used most of the time. The Y-counter is incremented each time a particular cache block is accessed. The X-counter is incremented each time the upper half of a cache block is accessed and decremented on each access to the lower half. When the



(a) Caches of 1kB



(b) Caches of 2kB



(c) Caches of 4kB

Figure 2: Amount of traffic produced by different caches, compared to the conventional direct-mapped cache.

Y-counter saturates and the absolute value of the X-counter exceeds a certain threshold, the cache block is marked temporal and removed from the spatial subcache. The results presented in [24] show that on average the STS cache incurs fewer misses (which generally translates to less data traffic) than a conventional cache. For some applications, however, the performance of the STS cache is worse than that of the conventional cache. As indicated in [24], one reason is that the performance of the STS cache is very dependent on the X- and Y-counter thresholds.

Johnson et al. [12] propose a hardware mechanism that detects spatial locality by counting how many of the smaller blocks in a larger block are valid in the cache. When a line is displaced its spatial locality information is stored in a table called the *Memory Address Table* (MAT) whose entries correspond to memory regions called *macroblocks*. The idea is that the access characteristics of the cache lines contained within each macroblock are relatively uniform. On a memory access the MAT is accessed in parallel with the data cache. If an entry is found that indicates spatial reuse, a larger block is fetched, otherwise the smaller size is fetched. If no entry is found the larger fetch size is chosen. A disadvantage of this approach compared to the STS cache is the tag implementation cost since every smaller block must be tagged. Furthermore, this technique appears rather costly to implement and it is unclear if the decrease in data traffic will translate to a significant reduction in energy dissipation. Other related approaches are the *Adaptive Line Size* (ALS) cache [30] and the work of Van Vleet et al. [29].

The *Dual Data Cache* (DDC) [7] also consists of a temporal and a spatial subcache. It is mainly targeted at numerical, vectorizable applications. The locality detection mechanism tries to detect if a load/store instruction accesses memory at a constant stride. If this is the case and the stride is smaller than the line size of the spatial subcache, the cache controller decides to cache the data in the spatial subcache. If the stride is larger or if the stride varies frequently, data is cached in the temporal subcache. Furthermore, if it is detected that a vector interferes with itself, the cache is bypassed. This happens, for example, when a vector is accessed with a unit stride and the size of the vector is larger than the cache size. Thus, the DDC uses instruction addresses to predict the type of locality, whereas the STS cache and the spatial locality detection technique of Johnson et al. employ data addresses. As we have shown in [14], this means that the DDC is unable to detect spatial reuse when adjacent words in a cache line are accessed by different load/store instruction or by non-consecutive executions of the same instruction. This happens, for example, when different fields of a structure (record) are accessed.

Another drawback of the DDC is the static allocation of cache capacity between the temporal and spatial subcache. If most memory references exhibit the same type of locality, only one of the sub-caches will be used. Moreover, the locality behavior may vary during program execution. While some parts of a program may exhibit mainly temporal locality, other parts may exhibit primarily spatial locality. In an attempt to remedy this problem, we [14] proposed two variants of the *Unified Dual Data Cache* (UDDC) called the UDDC-A and UDDC-B, respectively. Both these cache organizations support two block sizes and employ the same locality prediction mechanism as the DDC. The difference between the two variants is that in the UDDC-A the smaller blocks within a larger block share the tag, while in the UDDC-B each smaller block has its own tag. Specifically, the UDDC-A is a sector cache where each block (sector) is divided into several subblocks and a valid bit is added to every subblock. The UDDC-B is a conventional, direct-mapped cache with the possibility to fetch several consecutive blocks (a super-block).

Figure 2 also depicts the amount of traffic produced the DDC, the UDDC-A, and the UDDC-B normalized with respect to the traffic volume of the 32-byte block, direct-mapped cache. In these experiment the block size of the spatial sub-cache of the DDC was 32 bytes, the block size of the temporal sub-cache was 4 bytes, and the cache capacity was equally divided between the temporal and spatial sub-cache. Correspondingly, the block (super-block) size of the UDDC-A (UDDC-B) was 32 bytes and the sub-block (block) size of the UDDC-B 4 bytes. A locality prediction table with 128 entries was employed.

It can be seen that although the DDC generally reduces the amount of traffic, there are also cases where it generates significantly more traffic than the direct-mapped cache. In particular, for `gsm-dec` at 2kB, it produces a factor of 3.2 more traffic than the direct-mapped cache. In this case the data set fits in the direct-mapped cache as well as in the UDDC-A and UDDC-B, but not in the DDC because it has separate temporal and spatial sub-caches. The best results are obtained with the UDDC-B with an average traffic reduction (using the geometric mean) by a factor of 1.85 for 4kB caches. However, compared to the DDC and the UDDC-A, this organization has the largest tag implementation cost, since every smaller block needs to be tagged. The UDDC-A has the smallest tag implementation cost, but the traffic reduction it achieves is considerably smaller than the decrease achieved by the UDDC-B. It may, therefore, be promising to investigate decoupled sectored caches [26] where each sub-block is dynamically associated with two or more tags.

5. SUMMARY

Traffic between the L1 data cache and (off-chip) memory is very energy consuming. Because large caches consume more power than small caches and because set-associative caches consume more power than direct-mapped caches, a small, direct-mapped cache is preferable to a large, set-associative one. In this paper we have investigated how much traffic is generated by small, direct-mapped caches for a set of benchmarks typical for wearable, battery-powered devices such as mobile phones and digital camcorders. In addition, we have measured the amount of traffic generated by a hypothetical minimal traffic cache.

The results demonstrate that small, direct-mapped caches produce considerably more traffic than necessary, indicating that there is a significant opportunity to reduce traffic. Furthermore, we have identified two factors that make the largest contribution to the traffic inefficiency. First, because in a direct-mapped cache each memory word is mapped to exactly one cache block, they incur many conflict misses. Set-associative caches would reduce this problem but a set-associative cache consumes more power than a direct-mapped one. Second, a direct-mapped cache with large blocks produces considerably more traffic than one with small blocks. Since the block size is less relevant for the MTC, the problem is not so much caused by implicitly

prefetching data of which it is uncertain that it is going to be used, but more by the combination of zero associativity and block size. In a fully associative cache with optimal replacement the data that is implicitly prefetched can stay in the cache long enough to be used (since it is not replaced by other, less critical data), but in a direct-mapped cache it might be replaced before it is used.

Consequently, we have presented a survey of techniques aiming at reducing conflict misses and cache organizations that support variable block sizes and fetch large blocks if there is sufficient spatial locality and small blocks if not. Most of these techniques have been proposed to improve performance but appear promising for reducing the energy consumption as well. For some of these proposals experimental data has been presented. Although these cache organizations generally decrease the amount of traffic, it remains to be investigated if they also reduce the energy consumption, since most techniques employ some kind of history table which needs to be accessed during every cache access.

6. REFERENCES

[1] A. Agarwal and S. D. Pudar. Column-Associative Caches: A Technique for Reducing the Miss Rate of Direct Mapped Caches. In *Proc. Int. Symp. on Computer Architecture*, 1993.

[2] G. Albera and I. Bahar. Power/Performance Advantages of Victim Buffer in High-Performance Processors. In *Proc. Int. Symp. on Low Power Electronics and Design*, 1998.

[3] D. Burger, J.R. Goodman, and A. Kägi. Memory Bandwidth Limitations of Future Microprocessors. In *Proc. Int. Symp. on Computer Architecture*, 1996.

[4] Francky Catthoor. Energy-Delay Efficient Data Storage and Transfer Architectures and Methodologies: Current Solutions and Remaining Problems. *Jnl. of VLSI Signal Processing*, 21(3):219–231, 1999.

[5] Francky Catthoor, Koen Danckaert, Chidamber Kulkarni, Erik Brockmeyer, Per Gunnar Kjeldsberg, Tanja Van Achteren, and Thierry Omnes. *Data Access and Storage Management for Embedded Programmable Processors*. Kluwer Academic Publishers, 2002.

[6] Pepijn de Langen and Ben Juurlink. Reducing Traffic Generated by Conflict Misses in Caches. In *Proc. Computing Frontiers Conf.*, 2004.

[7] A. González, C. Aliagas, and M. Valero. A Data Cache with Multiple Caching Strategies Tuned to Different Types of Locality. In *Proc. Int. Conf. on Supercomputing*, 1995.

[8] A. González, M. Valero, N. Topham, and J. Parcerisa. Eliminating Cache Conflict Misses through XOR-based Placement Functions. In *Proc. Int Conf. on Supercomputing*, 1997.

[9] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. In *Proc. Annual Workshop on Workload Characterization*, 2001.

[10] M. Huang, J. Renau, S-M. Yoo, and J. Torrellas. L1 Data Cache Decomposition for Energy Efficiency. In *Int. Symp. on Low Power Electronics and Design*, pages 10–15, 2001.

[11] Koji Inoue, Tohru Ishihara, and Kazuaki Murakami. Way-Predicting Set-Associative Cache for High Performance and Low Energy Consumption. In *Proc. Int. Symp. on Low Power Electronics and Design*, 1999.

[12] Teresa L. Johnson, Matthew C. Merten, and Wen-mei W. Hwu. Run-Time Spatial Locality Detection and Optimization. In *Proc. Int. Symp. on Microarchitecture*, pages 57–64, 1997.

[13] N.P. Jouppi. Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers. In *Proc. Int. Symp. on Computer Architecture*, pages 364–373, 1990.

[14] Ben Juurlink. Unified Dual Data Caches. In *Proc. Euromicro Symp. on Digital System Design*, 2003.

[15] S. Kaxiras, Z. Hu, G. Narlikar, and R. McLellan. Cache-Line Decay: A Mechanism to Reduce Cache Leakage Power. In *Proc. Workshop on Power-Aware Computer Systems*, 2000.

[16] Johnson Kin, Munish Gupta, and William H. Mangione-Smith. Filtering Memory References to Increase Energy Efficiency. *IEEE Trans. on Computers*, 49(1), 2000.

[17] Chunho Lee, Miodrag Potkonjak, and William H. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems. In *Proc. Int. Symp. on Microarchitecture*, pages 330–335, 1997.

[18] H.-H. S. Lee and G.S. Tyson. Region-Based Caching: An Energy-Delay Efficient Memory Architecture for Embedded Processors. In *Proc. Int. Conf. on Compilers, Architecture, and Synthesis for Embedded Systems*, 2000.

[19] G. Memik, G. Reinman, and W.H. Mangione-Smith. Reducing Energy and Delay Using Efficient Victim Caches. In *Proc. Int. Symp. on Low Power Electronics and Design*, pages 262–265, 2003.

[20] T.H. Meng, B. Gordon, E. Tsern, and A. Hung. Portable Video-on-Demand in Wireless Communication. *Proc. of the IEEE, special issue on "Low Power Electronics"*, 83(4), 1995.

[21] K. Palem, R. Rabbah, V. Mooney III, P. Korkmaz, and K. Puttaswamy. Design Space Optimization of Embedded Memory Systems via Data Remapping. In *Proc. Joint Conf. on Languages, Compilers, and Tools for Embedded Systems and Software and Compilers for Embedded Systems*, 2002.

[22] Preeti Ranjan Panda, Nikil Dutt, and Alexandru Nicolau. *Memory Issues in Embedded Systems-on-Chip*. Kluwer Academic Publishers, Boston/Dordrecht/London, 1999.

[23] P. Petrov and A. Orailoglu. Performance and Power Effectiveness in Embedded Processors - Customizable Partitioned Caches. *IEEE Trans. of Computer-Aided Design of Integrated Circuits and Systems*, 20(11), 2001.

[24] M. Prvulović, D. Marinov, Z. Dimitrijević, and V. Milutinović. Split Temporal/Spatial Cache: A Survey and Reevaluation of Performance. IEEE TCCA Newsletter, July 1999.

[25] G. Reinman and N. Jouppi. An Integrated Cache

Timing and Power Model. Technical Report CACTI 2.0, COMPAQ Western Research Lab, 1999.

[26] Jeffrey B. Rothman and Alan Jay Smith. The Pool of Subsectors Cache Design. In *Proc. Int. Conf. on Supercomputing*, pages 31–42. ACM Press, 1999.

[27] Tajana Simunic, Luca Benini, and Giovanni De Micheli. Energy-Efficient Design of Battery-Powered Embedded Systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(1), 2001.

[28] R. A. Sugumar and S. G. Abraham. Efficient Simulation of Caches Under Optimal Replacement With Applications to Miss Characterization. In *Proc. ACM SIGMETRICS Conf. on Measurement and Modeling Computer Systems*, 1993.

[29] Peter van Vleet, Eric Anderson, Linsay Brown, Jean-Loup Baer, and Anna Karlin. Pursuing the Performance Potential of Dynamic Cache Line Sizes. In *Proc. Int. Conf. on Computer Design*, 1999.

[30] Alexander V. Veidenbaum, Weiyu Tang, Rajesh Gupta, Alexandru Nicolau, and Xiaomei Ji. Adapting Cache Line Size to Application Behavior. In *Proc. Int. Conf. on Supercomputing*, pages 145–154, 1999.

[31] S. Wuytack, F. Catthoor, F. Franssen, L. Nachtergaele, and H. De Man. Global Communication and Memory Optimizing Transformations for Low-Power Signal Processing Systems. In *Proc. IEEE Int. Workshop on Low Power Design*, 1994.

[32] Se-Hyun Yang, Michael D. Powell, Babak Falsafi, Kaushik Roy, and T. N. Vijaykumar. An Integrated Circuit/Architecture Approach to Reducing Leakage in Deep-Submicron High-Performance I-Caches. In *Proc. Int. Symp. on High Performance Computer Architectures*, 2001.

[33] H. Zhou, M.C. Toburen, E. Rotenberg, and T.M. Conte. Adaptive Mode Control: A Static-Power-Efficient Cache Design", Instruction Set. In *Proc. Int. Conf. on Parallel Architectures and Compilation Techniques*, 2001.