Off-Chip Memory Traffic Measurements of Low-Power Embedded Systems

Pepijn de Langen   Ben Juurlink

Computer Engineering Laboratory
Electrical Engineering Department
Delft University of Technology
P.O.Box 5031, 2600 GA Delft, The Netherlands
Phone: +31 15 278 1572, Fax: +31 15 278 4898
E-mail: {pepijn|benj}@cs.et.tudelft.nl

Abstract—**In all processors, power can be saved by making effective use of on-chip memory. For embedded systems this is crucial, since they often drain their power from a pair of batteries. In this paper, we experimentally measure the amount of off-chip traffic produced by several caches. It is shown that large savings can be achieved if size, associativity, and block size are well-chosen. Most examined direct-mapped caches produce five to ten times as much traffic as needed, but sometimes much more. For most benchmarks the minimum cache size to perform well is 8 kB. Overall, the results indicate that cost-effective on-chip caches are highly application-specific.**

Keywords— **memory traffic; embedded systems; caches; power consumption**

I. Introduction

Power consumption is an important aspect of embedded systems. For many embedded applications, off-chip memory accesses consume more power than the data paths and the controllers [2]. Hence, large power savings can be achieved by making effective use of on-chip memory.

In this paper we investigate the reduction of off-chip traffic that can be achieved for embedded applications. This is done by measuring the *traffic inefficiency*, defined as the ratio of the amount of traffic generated by a conventional cache to the amount of traffic produced by the minimal-traffic cache. Similar experiments have been performed by Burger et al. [1] for SPEC92 and SPEC95 benchmarks, but our work differs in the following ways. First, our benchmarks are taken from the MediaBench suite [3], which is more representative of embedded applications. Second, *request traffic*, i.e., the addresses sent from the processor to the off-chip memory, is also incorporated in our calculations. Burger et al. did not consider request traffic and their results are, therefore, biased in favor of smaller blocks. Third, we also consider *traffic* $\times$ *delay* since many traffic reduction techniques (e.g., reducing the block size) can have a negative impact on performance. Given the real-time nature of many embedded applications, performance should not be neglected. The fourth difference is the way how contributions to the traffic inefficiency are calculated. Burger et al. subtracted two inefficiencies to calculate the contribution of a cache parameter, such as block size. In this paper, the ratio between the two is used, since this provides more information on the influence of the various parameters. 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.

This paper is organized as follows. Section II explains how *traffic*, *traffic inefficiency*, and *delay* are defined. This section also describes the cache parameters and the employed benchmarks. Section III presents the results of our experiments. First, the traffic inefficiencies of conventional direct-mapped caches are presented. After that, it is determined how various factors influence the amount of traffic generated by a cache. Finally, we look at *traffic* $\times$ *delay* for various cache sizes. Conclusions are given in Section IV.

II. Background

A. Cache Parameters

Three different cache organizations are considered in this paper: direct-mapped caches, fully-associative caches, and the minimal-traffic cache. 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. A word size of 4 bytes is assumed, equal to the size of a transfer to or from the memory system. The address space is 32-bit.

The minimal traffic cache (MTC) is fully-associative, has a minimal block size, and employs Belady's off-line **min** replacement strategy. This cache produces a minimum amount of data traffic for write-through caches. For write-back caches, however, **min** is not optimal. As it is still an aggressive lower bound, we will use it for the MTC. One should also note that since request traffic is not ignored, the MTC with a block size of 4 bytes may produce more traffic than the MTC with larger blocks. Although the MTC typically uses a block size equal to the transfer size, we also simulate this cache with larger blocks.

All caches use *allocate-on-write,* although the MTC may be *bypassed* in order to keep data of higher priority in the cache. This is beneficial if all blocks currently in the cache are accessed nearer in the future than the requested block. The fully-associative caches considered in this study use the *least-recently-used* replacement strategy.

B. Traffic

Since our goal is to decrease power consumption by reducing the amount of data that has to be transferred to or from the chip, traffic is defined as the total number of bytes in and out of the first-level cache. Loads and stores that hit the cache, therefore, do not produce any traffic. For cache misses, traffic is calculated as follows. For every missed load or store, an amount of data equal to the cache block size needs to be fetched. Furthermore, since write-back is used, every write miss or hit to a clean block increases the total amount of data traffic by another block, since the block needs to be written back when it is replaced.

For each requested block, an address is sent from the processor to the off-chip memory. For a block size of one word this means that the amount of *request traffic* will be equal to the amount of data traffic, for a block size of two words it will be half as much as the data traffic, and so on. The amount of *request traffic* is equal to:

$$\textit{request traffic} = \frac{\textit{data traffic}}{\textit{\# words per block}}.$$

To measure how much more off-chip traffic than necessary is generated by a cache, we use *traffic inefficiency,* which is defined as:

$$\textit{traffic inefficiency} = \frac{\textit{traffic generated by a cache}}{\textit{traffic generated by the MTC}}.$$

C. Delay

Since the benchmarks are largely memory-bound, we only consider memory accesses. For both loads and stores that hit the cache, a delay of one cycle is assumed. For a cache miss, the first word arrives after 40 cycles, the others follow in a rate of one word per cycle. The total miss penalty is, therefore, 39 cycles plus the number of words in a cache block.

D. Benchmarks and Tools

In this paper, benchmarks from the *MediaBench* suite [3] are used, as they represent a large part of applications used in modern embedded systems, such as cell phones, digital cameras, and PDA's.

The `sim-safe` simulator from the SimpleScalar toolset was modified to produce traces of the memory references. These traces are read by our cache simulator which produces statistics such as the number of loads and stores that hit the cache and the number of replaced dirty blocks. From this output the traffic and delay are calculated.

III. 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, the factors that contribute to the traffic inefficiency are isolated. Finally, *traffic*$\times$*delay* is considered.

A. Traffic Inefficiency for Direct-Mapped Caches

In order to calculate traffic inefficiencies, we simulated direct-mapped caches with blocks of 32 bytes as well as the minimal traffic cache (MTC). The cache size varies from 256 bytes to 32 kilobytes, since embedded systems typically have small caches.

The resulting traffic inefficiencies of direct-mapped caches are presented in Table I. From this table, it can be seen that most inefficiencies are between 5 and 10, but sometimes they exceed two orders of magnitude. Overall, the results indicate that large savings can be achieved. Furthermore, it can be observed that for some benchmarks the traffic inefficiency is not monotonically decreasing in the cache size. Sometimes, the traffic inefficiency increases significantly, after which it decreases again.

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 various benchmarks. 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.

In general, if the cache size is increased, the amount of traffic decreases. Furthermore, if the cache is large enough to hold the entire working set, the amount of traffic no

	256B	512B	1kB	2kB	4kB	8kB	16kB	32kB
adpcm-dec	6.510	3.329	2.724	2.362	56.209	2.448	.624	.624
adpcm-enc	7.922	4.934	3.973	3.444	47.048	2.570	.649	.649
jpeg-dec	6.221	6.178	5.853	6.544	4.358	3.072	5.200	3.636
jpeg-enc	6.402	6.490	6.278	4.971	3.911	1.365	.838	.742
mpeg2-dec	14.658	16.806	26.526	40.814	18.718	13.486	3.611	1.334
g721-dec	26.738	131.149	45.529	25.414	56.240	15.972	5.257	5.233
g721-enc	15.186	222.200	39.078	21.022	26.945	17.677	13.059	13.035
pegwit-dec	7.297	6.901	6.756	6.737	5.997	6.236	6.625	7.447
pegwit-enc	7.535	7.036	6.797	6.447	5.970	6.073	7.200	8.159
gsm-dec	3.315	3.188	3.990	2.573	3.518	11.760	3.093	.718
gsm-enc	19.974	18.746	2.095	5.377	3.944	10.256	3.811	1.534
epic	5.128	4.042	3.752	3.728	3.633	1.695	1.308	1.109
unepic	2.799	2.426	2.209	1.760	1.326	1.216	1.216	1.260
mesa-mipmap	8.504	9.020	5.798	5.034	3.365	1.799	1.115	.771
mesa-texgen	4.246	5.572	7.377	7.575	6.149	4.776	3.146	2.150
mesa-osdemo	3.087	2.340	1.795	1.412	1.191	1.080	.833	.796
gs-dec	14.554	13.049	11.345	13.687	18.625	7.050	5.279	3.135

TABLE I
TRAFFIC INEFFICIENCIES FOR 32-BYTE DIRECT-MAPPED CACHES.

longer decreases. An example of this can be seen in Figure 1(a). When the direct-mapped caches are larger than 16kB, the amount of traffic no longer decreases. For the MTC this occurs already at 4kB. As a result, traffic inefficiency will also become a constant when the cache size exceeds a certain limit.

For some benchmarks, like `pegwit-enc` whose results are depicted in Figure 1(c), the different configurations benefit almost equally from an increased cache size. None of the caches is able to hold the entire working set. For these benchmarks the traffic inefficiency hardly depends on the cache size, as can be seen in Table I.

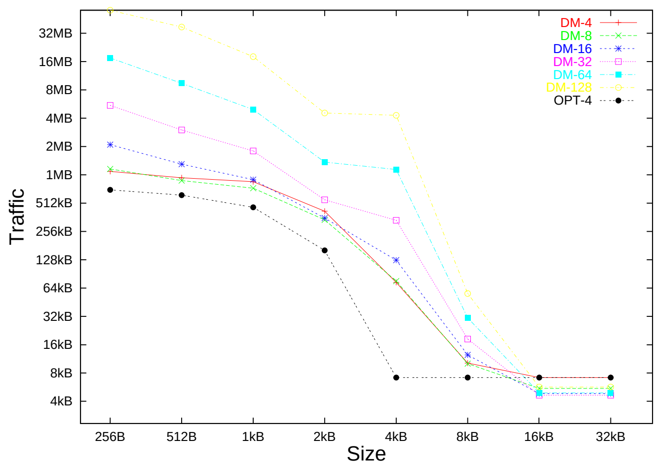
The MTC will always benefit from an increased cache size, unless it is already able to hold the entire working set. Direct-mapped caches, however, may not be able to benefit at all from the increase in size. This can be seen, for example, in Figures 1(a) and 1(b) where in some caches the amount of traffic hardly decreases if the size of the direct-mapped cache is doubled from 2kB to 4kB. This can be explained as follows. If two arrays conflict with each other, then these conflict misses will not disappear unless the cache is larger than the array(s).

So, because the direct-mapped caches and the MTC may not benefit equally from increased size, the traffic inefficiency may increase if cache size increases. Examples of this can also be seen in Figure 1(a) and 1(b), where the distance between the lines corresponding to the direct-mapped cache with 32-byte blocks and the MTC shows sudden increases for certain sizes of the cache. However, if a cache is made large enough, it will eventually reach a constant value. This value will be at most 8, the relative difference in block size.

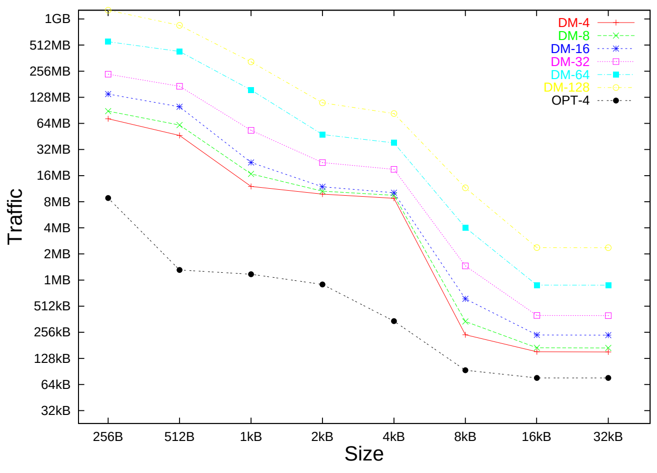
Not considering request traffic, a smaller block size always results in less traffic. Since in our experiments the request traffic is not ignored, a cache with smaller blocks can produce more traffic than a cache with larger blocks. As noted before, the MTC is not really a lower bound, since it may produce more traffic than a cache with blocks larger than 4 bytes. This can be seen, for example, in Figures 1(a) and 1(d), where the amount of request traffic is significant for caches with small blocks. For `adpcm-enc`, even the direct-mapped cache with blocks of 32 bytes produces less traffic than the MTC with 4 bytes per block. This results in traffic inefficiencies smaller than 1, as can be seen in Table I.

B. Isolating the Factors That Contribute to the Inefficiency

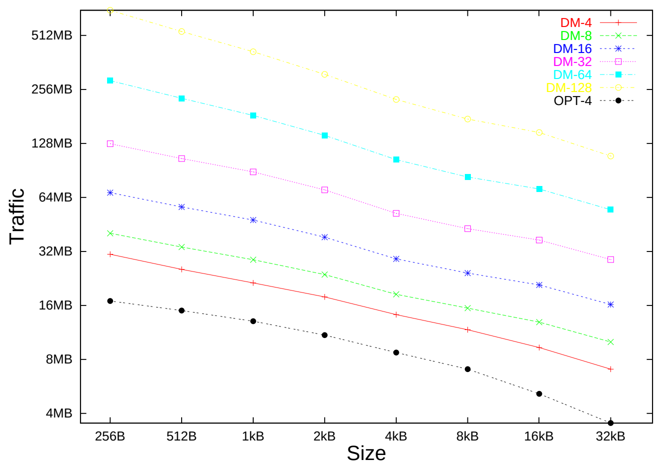
There are several factors that contribute to the traffic inefficiency, such as the block size, the associativity, and the replacement strategy. Although the real contributions to the inefficiency are not independent, in this section we attempt to isolate their contribution by comparing caches that differ only in one parameter. The caches compared in these experiments are listed in Table II. As noted in



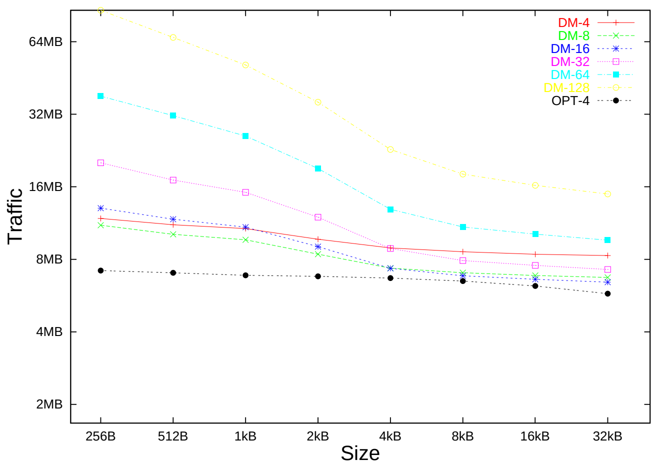
(a) adpcm-enc



(b) g721-dec



(c) pegwit-enc



(d) unepic

Fig. 1
AMOUNT OF TRAFFIC FOR DIFFERENT BENCHMARKS.

the introduction, in the work of Burger et al. [1] the traffic inefficiency of the first cache is subtracted from the inefficiency of the other cache to compute the *inefficiency gap*. Since this metric does not allow different benchmarks and different cache sizes to be compared, we compute the relative difference, by dividing the inefficiency of the first by the latter. To avoid confusion, we will refer to our metric as the *inefficiency ratio*.

The inefficiency ratios are presented in Tables III, IV, and V for caches of 2kB, 4kB, and 8kB, respectively. The most interesting changes in efficiency seem to be in this range, as can be seen in Table I. From these tables the traffic inefficiency can be computed as:

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

where F_{accoc}, 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 associativity, a direct-mapped cache is compared to a fully-associative one that employs LRU replacement. Tables III, IV, and V show that the influence of associativity on the amount of traffic depends on the benchmark as well as on the size of the cache. Note that this factor is 1 when both caches are large enough to hold

Benchmark	Associativity	Replacement	Block Size (DM)	Block Size (OPT)
adpcm-dec	1.156	2.894	1.478	.705
adpcm-enc	1.214	4.197	1.321	.675
jpeg-dec	3.004	1.593	2.592	1.366
jpeg-enc	3.826	1.371	2.867	.947
mpeg2-dec	9.336	2.259	5.874	1.934
g721-dec	29.541	1.377	2.314	.624
g721-enc	27.403	1.267	2.653	.605
pegwit-dec	1.207	1.496	3.912	3.729
pegwit-enc	1.228	1.365	3.946	3.842
gsm-dec	.950	4.238	1.134	.638
gsm-enc	1.006	3.116	1.236	1.715
epic	4.644	1.245	2.895	.644
unepic	1.998	1.449	1.235	.608
mesa-mipmap	6.069	1.212	2.594	.684
mesa-texgen	1.499	2.704	2.930	1.868
mesa-osdemo	1.516	1.282	1.153	.726
gs-dec	3.055	1.678	4.317	2.667

TABLE III
INEFFICIENCY RATIOS OF 2KB CACHES.

Benchmark	Associativity	Replacement	Block Size (DM)	Block Size (OPT)
adpcm-dec	90.033	1.000	3.055	.624
adpcm-enc	72.393	1.000	4.574	.649
jpeg-dec	1.963	2.061	1.842	1.076
jpeg-enc	2.960	1.473	2.468	.896
mpeg2-dec	6.073	2.642	5.206	1.166
g721-dec	24.683	3.143	2.158	.724
g721-enc	21.018	2.034	2.318	.630
pegwit-dec	1.232	1.398	3.559	3.479
pegwit-enc	1.186	1.351	3.667	3.722
gsm-dec	2.541	1.920	1.311	.721
gsm-enc	4.405	1.350	1.270	.663
epic	3.899	1.466	2.674	.635
unepic	1.651	1.347	.994	.596
mesa-mipmap	3.529	1.389	1.845	.686
mesa-texgen	4.806	1.462	2.349	.874
mesa-osdemo	1.446	1.215	.977	.677
gs-dec	3.387	2.407	4.495	2.283

TABLE IV
INEFFICIENCY RATIOS OF 4KB CACHES.

Benchmark	Associativity	Replacement	Block Size (DM)	Block Size (OPT)
adpcm-dec	3.921	1.000	1.742	.624
adpcm-enc	3.954	1.000	1.801	.649
jpeg-dec	2.721	1.405	1.241	.802
jpeg-enc	1.356	1.573	.956	.640
mpeg2-dec	14.178	1.583	4.662	.600
g721-dec	11.842	2.065	6.219	.653
g721-enc	9.939	2.603	5.929	.683
pegwit-dec	1.237	1.449	3.508	3.476
pegwit-enc	1.095	1.456	3.661	3.805
gsm-dec	13.709	1.294	1.997	.662
gsm-enc	12.369	1.234	1.438	.671
epic	1.571	1.745	1.355	.617
unepic	1.523	1.345	.919	.593
mesa-mipmap	1.675	1.614	1.085	.665
mesa-texgen	4.905	1.411	1.993	.689
mesa-osdemo	1.461	1.171	.904	.630
gs-dec	3.565	1.977	3.044	1.000

TABLE V
INEFFICIENCY RATIOS OF 8KB CACHES.

		Assoc., repl. policy, blk. size	
Factor		Exp1	Exp2
I.	Associativity	dm,n/a,32B	fa,LRU,32B
II.	Replacement	fa,LRU,32B	fa,MIN,32B
III.	Blk. size (DM)	dm,n/a,32B	dm,n/a,4B
IV.	Blk. size (MTC)	fa,MIN,32B	fa,MIN,4B

TABLE II
CACHES COMPARED TO ISOLATE THE FACTORS THAT CONTRIBUTE TO THE TRAFFIC INEFFICIENCY

all referenced data. Although in general this factor decreases as the cache size is increased, we again observe that sometimes the inefficiency ratio increases if the cache is enlarged. This is because the direct-mapped caches may still have a lot of conflict misses, in contrast to the fully associative ones. Furthermore, since we are using the LRU replacement strategy, it is possible that the fully associative cache produces more traffic than the direct-mapped cache. This can be seen in Table III, for example, for the `gsm-dec` benchmark.

The second factor is the replacement strategy. To determine the influence of this factor, a fully associative cache with 32-byte blocks is compared to the MTC with the same block size. Again, since there are no conflict nor capacity misses in a cache that is sufficiently large, this ratio will eventually become one. Because both caches need not benefit equally from increased capacity, the ratio can sometimes increase if the cache size is increased. This occurs, for example, in both `pegwit` benchmarks for cache sizes of 4 and 8 kilobytes. Overall, the replacement strategy contributes to the amount of traffic with factors between 1 and 3 for all benchmarks and cache sizes.

The third factor is the block size. For both the direct-mapped cache and the MTC, we calculate the ratios in produced traffic between 32-byte and 4-byte variants. Since request traffic is included in our calculations, this ratio may be smaller than 1. For this comparison, the inefficiency ratios are between 1 and 8 for most direct-mapped caches. For the MTC, the block size is less important. Actually, for most of the the benchmarks and cache sizes, the MTC with a block size of 4 bytes produces more traffic than the MTC with a block size of 32 bytes.

Interesting behavior can be seen for both `pegwit` benchmarks. For these benchmarks, the inefficiency ratios resulting from employing larger blocks are about the same for both the direct-mapped cache and the MTC, independent of the cache size. The reason for this is that these benchmarks exhibit little spatial locality, so caches with larger blocks have about the same number of hits and misses as the caches with smaller ones.

Overall, associativity is the most important factor, especially when when caches are small. We remark that this is different from the work of Burger et al., where the most important factor was found to be the block size. A plausible explanation is that we consider request traffic whereas Burger et al. do not. For caches large enough to hold the entire working set, the amount of spatial locality determines whether or not large blocks are beneficial. In Figure 1, for example, `adpcm-enc` generates about the same amount of data traffic for all direct-mapped caches larger than 16kB, indicating that most cached data was actually used. For `g721-dec`, the cache with larger blocks loads a significant amount of data that is never referenced.

C. Traffic $\times$ Delay

Many traffic reduction techniques (e.g., reducing the block size) can have a negative impact on performance. Given the real-time nature of many embedded applications, this should not be neglected. We consider *traffic*$\times$*delay* as an important measure of quality. In Figure 2, *traffic*$\times$*delay* is depicted as a function of the cache size for a number of benchmarks. In these figures, the MTC is excluded, since it uses an off-line replacement algorithm and thus has no reasonable delay.

Like traffic, delay will also reach a certain constant value when the cache is large enough to hold the whole data-set. The graphs in Figure 2 are similar to the corresponding graphs in Figure 1. The main reason for this is that the extra delay, caused by employing smaller blocks, is already taken into account by also considering request traffic. However, some differences can be noted. For some benchmarks, such as `adpcm-enc` or `unepic`, the cache configuration that is optimal with respect to traffic$\times$delay is different from the cache configuration that is optimal with respect to traffic. Others benchmarks like `pegwit-enc` show no relative change at all.

IV. Conclusions

It is concluded that the amount of traffic generated by a cache depends heavily on the cache size and the benchmark. In many cases, we observed that the amount of traffic was significantly reduced when the size of the direct-mapped cache was increased from 4kB to 8kB. Furthermore, lack of associativity can be disastrous for the amount of produced traffic, especially if the cache size is small.

For caches with 4-byte blocks, the amount of request traffic is equal to the amount of data traffic. Still, this cache produces the least traffic for several configurations. For small caches this is caused by the smaller number of conflicts in comparison to caches with larger blocks. Also, writing back data is cheaper for caches with smaller blocks.

Larger blocks seem to outperform smaller ones only if the cache is large enough and the application exhibits spatial locality. If neither is the case, smaller blocks are preferable.

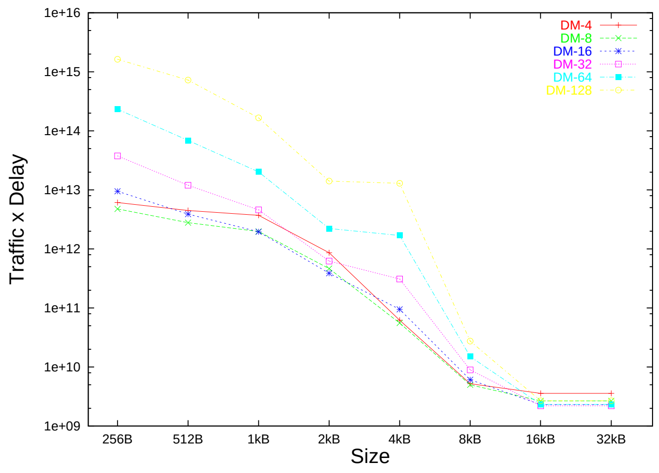
When looking at *traffic*$\times$*delay*, some caches with small blocks show to be less preferable. However, the overall results are the same as when only traffic is considered.

References

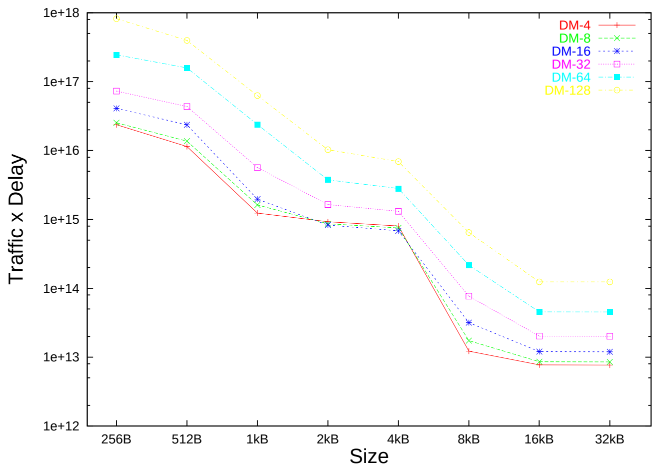
[1] D. Burger, J.R. Goodman, and A. Kägi. Memory Bandwidth Limitations of Future Microprocessors. In *23rd International Symposium on Computer Architecture*, 1996.

[2] F. Catthoor, F. Franssen, S. Wuytack, L. Nachtergaele, and H. De Man. Global Communication and Memory Optimizing Transformations for Low-Power Signal Processing Systems. In *VLSI Signal Processing Workshop*, 1994.

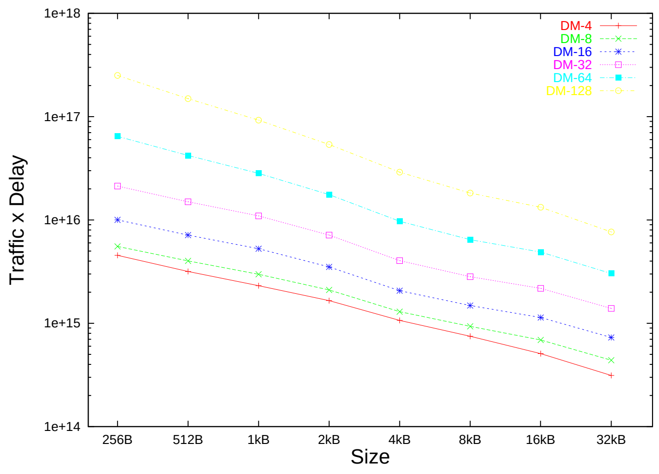
[3] Chunho Lee, Miodrag Potkonjak, and William H. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems. In *International Symposium on Microarchitecture*, pages 330–335, 1997.



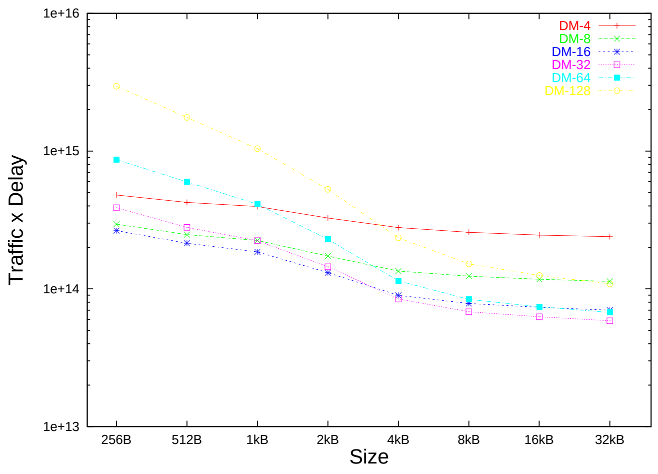
(a) adpcm-enc



(b) g721-dec



(c) pegwit-enc



(d) unepic

Fig. 2

TRAFFIC×DELAY FOR DIFFERENT BENCHMARKS.