HitME: Low Power Hit MEmory Buffer for Embedded Systems

Andhi Janapsatya, Sri Parameswaran† and Aleksandar Ignjatović†
School of Computer Science & Engineering, University of New South Wales, Sydney, NSW 2052, Australia
†NICTA, Sydney, NSW 2052, Australia
{andhij, ignjat, sridevan}@cse.unsw.edu.au

ABSTRACT

In this paper, we present a novel HitME (Hit-MEmory) buffer to reduce the energy consumption of memory hierarchy in embedded processors. The HitME buffer is a small direct-mapped cache memory that is added as additional memory into existing cache memory hierarchies. The HitME buffer is loaded only when there is a hit on L1 cache. Otherwise, L1 cache is updated from the memory and the processor's memory request is served directly from the L1 cache. The strategy works due to the fact that 90% of memory accesses are only accessed once, and these often pollute the cache. Energy reduction is achieved by reducing the number of accesses to the L1 cache memory. Experimental results show that the use of HitME buffer will reduce the L1 cache accesses resulting in a reduction in the energy consumption of the memory hierarchy. This decrease in L1 cache accesses reduces the cache system energy consumption by an average of 60.9% when compared to traditional L1 cache memory architecture and an energy reduction of 6.4% when compared to filter cache architecture for 70nm cache technology.

1. INTRODUCTION

Low energy consumption of embedded systems can lengthen battery life, improve reliability, decrease weight and reduce packaging cost. Memory hierarchies inside an embedded processor contribute as much as 50% of the total microprocessor power [1][2]. However, the performance gap between processor and the memory system necessitates the use of cache memory hierarchies to improve performance and reduce energy consumption.

Many different techniques to reduce energy consumption of memory inside a microprocessor have been proposed in the past. Examples of these techniques include code compression to reduce the memory bus traffic [3][4][5], efficient cache management strategies such as branch alignment [7][8] and code placement [6][9], scratch-pad memories [10] and the addition of tiny memories on top of existing cache memories (loop cache [11][12] and filter cache [13]).

In this paper, we propose the use of a small Hit-MEmory (tiny direct-mapped cache) buffer, which we will call HitME buffer. The HitME buffer is inserted in addition to the existing cache memory structure. Figure 1(b) shows the memory hierarchy with the HitME buffer. Figure 1(a) shows the traditional L1 cache memory hierarchy for comparison. A multiplexer is inserted in the HitME buffer architecture (Figure 1(b)) to allow the CPU to select whether to fetch data from the L1 cache or the HitME buffer. The use of a multiplexer allows the CPU to fetch memory content directly from the L1 cache and bypass the HitME buffer. This is to reduce data pollution in the small HitME buffer.

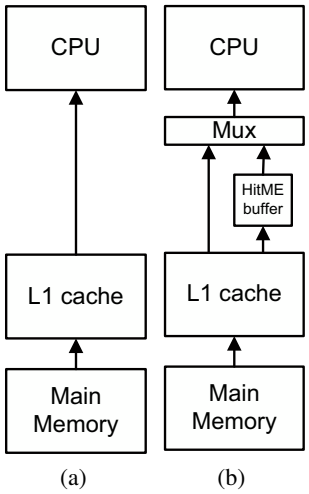

Figure 1: 1(a) a traditional memory hierarchy with L1 cache; 1(b) Hit-ME buffer memory hierarchy

The name HitME buffer is chosen because the content of the HitME buffer is only updated whenever an L1 cache hit event occurs (please refer to Section 4 for a more detailed description). If there is a miss on L1 cache, and L1 cache is updated from the main memory, then the memory request will be directly served to the processor by the L1 cache, and the HitME buffer will not be updated. Only if there is another access to the same cache location (resulting in an L1 cache hit), will the L1 cache update the HitME buffer. Such a buffer update strategy ensures that only memory elements that have been accessed more than once will be loaded into the HitMe buffer.

The motivation for using such a replacement strategy is inspired by the results presented in [14], which stated two important points. The first stated that 90% of memory blocks are accessed only once. The second stated that the probability of re-access increases as the number of accesses increase. For example, they showed that if an instruction is accessed twice, then there was a 40% probability of re-access, but if it was accessed 10 times or more, there was almost a 100% probability of re-access.

The rest of this paper is structured as follows. Section 2 presents a survey of existing memory optimization techniques that use tiny cache memories; Section 3 describes the system architecture includ-

ASP-DAC 2009, January 19–22, 2009, Yokohama, Japan.

978-1-4244-2749-9/09/$25.00 ©2009 IEEE

ing the HitME organization; Section 4 presents the utilization strategy of the HitME buffer; Section 5 describes the experimental setup. Section 6 presents the analysis of the results; and finally Section 7 concludes this paper.

2. RELATED WORK

Cache memory optimization techniques are the subject of a number of researchers due to the large percentage of energy the memory hierarchy consumes, and the promise of reduced energy consumption that such techniques can offer. For comparison with the HitME buffer presented in this paper, we present a brief survey of other techniques which introduce tiny memories to optimize the memory systems inside a microprocessor.

In 1997, Kin et al. [13] introduced the filter-cache. Filter cache is a small cache memory inserted between the L1 cache and the processor. The filter cache operates as a Level-0 cache memory. Due to its smaller size, the filter cache has reduced power dissipation in comparison to traditional L1 cache memory. While the tiny filter cache causes decrease in hit ratio, the decrease in power consumption should compensate for the loss in performance. The filter cache memory hierarchy has been shown to produce an average $energy * delay$ reduction of 51% for the benchmark applications in the MediaBench [15].

The HitME buffer presented in this paper differs from the filter cache because it is not a level 0 cache. Thus, not all memory accesses are fetched from the HitME buffer, and only a selected memory contents are inserted into the HitME buffer.

Scratchpad memory is a memory array with the decoding and column circuitry logic [10]. Scratchpad memories are different to cache memories because they do not have the tag checking mechanism that caches contain. In 1997, Panda [16] presented the scheme for static scratchpad memory for use as data memory. In 2002, Steinke [17] presented a dynamic management scheme to utilize the scratchpad memory as instruction memory. In 2006, Janapsatya [18] presented a statistical method to profile applications and identify code blocks that can be dynamically inserted into the scratchpad memory during program execution.

In comparison to scratchpad memory, the HitME buffer is still a cache memory with tag checking mechanism; thus, its access time will not be as fast as a scratchpad memory, because of cache tag checking on each memory access. Nevertheless, the tag checking mechanism in cache allows the HitME buffer to be transparent to the software, unlike the scratchpad memory where the software needs to know whether to send memory request to the scratchpad or the cache memory.

In 2003, Gordon-Ross et al. [12] introduced the pre-loaded loop-caches. Application profiling is required to identify loops inside the application. Loop inside applications are then selected to be pre-loaded into the loop cache before the program begins execution. The contents of the loop cache do not change once the program begins execution. Their experimental results compared pre-loaded loop cache, dynamic loop cache and filter caches. The results shows that filter cache achieves the best instruction fetch energy reduction of 60-80% but at the cost of 20% performance degradation.

Pre-loaded loop caches require profiling of the software to identify memory content that should be uploaded into the loop cache and it does not allow the modification of the content of the loop cache during run time. This is different compared to the HitME buffer presented in this paper, where the content of the HitME buffer changes according to the memory access pattern of the application.

In 1999, Lee et al. [19][2] presented the concept of dynamic tagless loop caching. In 2002, Gordon-Ross et al. [20] presented a modified dynamic loop cache scheme which they called hybrid loop cache. With hybrid loop cache, complex loops are to be pre-loaded into the loop cache prior to program execution and other loops will be dynamically loaded into the loop cache during program execution.

2.1 Our Contribution

The concept of HitME buffer described in this paper is similar to the idea of dynamic loop cache memory. The HitME buffer allows selective insertion of memory content to reduce HitME buffer pollution and improve the HitME buffer hit rate. Selective HitME buffer update is performed whenever an L1 cache hit occurs. This is different from the dynamic loop cache scheme where a dynamic profiler is needed to identify loops that are to be inserted into the loop cache. In comparison, the HitME buffer idea is cheaper and easier to implement because we do not need custom compilers and expensive dynamic loop profilers. In addition, loop caches only target instruction memory, while HitME buffer can target both instruction and data memory.

The contributions of the work presented in this paper are as follows:

1. A novel HitME buffer design, configuration and organization;
2. HitME buffer utilization strategy.

We also present an energy and performance model to estimate and compare the performance and energy consumption of a system utilizing the HitME buffer. Cache simulations were performed to calculate the cache hits and misses for different cache configurations. The energy model is then applied together with cache hits and misses to evaluate the effectiveness of the HitME buffer configurations.

3. SYSTEM ARCHITECTURE

The architecture of the processor with the HitME buffer is as shown in Figure 1(b). The HitME buffer is a direct-mapped cache and the number of sets of the HitME buffer is designed to be equal to the number of sets in the L1 cache memory. Size of the HitME buffer will be equal to

$$HitME_{size} = L1Cache_{sets} * L1Cache_{block} \quad (1)$$

For example, an L1 cache configuration of size 1024 bytes, block size 16 bytes and associativity of 4, has 16 cache sets ($\frac{size}{asso*block} = \frac{1024}{4*16}$). The HitME buffer will then be a direct-mapped cache with block size 16 bytes and set size of 16, equating to a $HitME_{size} = 256$ bytes.

With the HitME buffer block size and number of sets equal to the L1 cache block size and set size, each cache set in the L1 cache will map to exactly one set in the HitME buffer. Hence, the idea of HitME buffer is to provide each L1 cache set with a hit buffer.

4. HITME BUFFER STRATEGY

The HitME buffer allows frequently accessed memory content to be stored in a small direct-mapped cache memory allowing low power access by the processor. Memory accesses from a small direct-mapped cache memory will consume less energy and allow faster access time than L1 cache alone which are both beneficial to the overall system.

The strategy to utilize and to selectively fill the HitME buffer is as follows. When the memory hierarchy receives a memory request from the CPU, the HitME buffer is the first memory level to be searched. If a HitME buffer hit occurs, the memory request will be serviced from the data inside the HitME buffer. If a HitME buffer miss occurs, the L1 cache memory will be the next memory level to be searched.

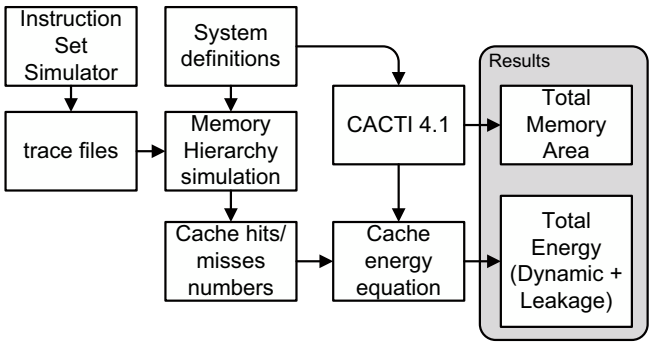

Figure 2: Experimental Setup

L1 cache size (Bytes)	1K - 64K
L1 block size (Bytes)	16
L1 associativity	4 - 16
L1 replacement policy	FIFO
HitME size (Bytes)	128 - 16K
HitME block size (Bytes)	16
HitME associativity	1

Table 1: System configuration design space

If an L1 cache miss occurs, memory request is sent to the main memory and the L1 cache memory is updated according to the L1 replacement policy. The CPU memory request will then be serviced from the L1 cache memory and no update is done to the HitME buffer.

However, when a HitME buffer miss occurs and an L1 cache hit occurs, the content of the HitME buffer is first updated with the data from the L1 cache memory. The HitME buffer will then service the CPU memory request. The decision to allow the HitME buffer to be updated prior to servicing the CPU memory request will ensure the validity of the content of the HitME buffer and allow a single read/write port L1 cache memory to be used with the HitME buffer. Alternative to the strategy presented above is to allow the L1 cache memory to service the CPU request and update the HitME buffer content concurrently. While such a strategy can reduce the memory access time, we do not see such a need due to the L1 cache memory and the HitME buffer operating at a much faster speed compared to the CPU used for embedded processors (see subsection 5.2 for comparison of CPU cycle time and cache access time).

5. EXPERIMENTAL SETUP

The experiments were done via trace based simulations. Traces were obtained from simulating the benchmarks from the Mediabench [15] suite with Tensilica [22] processors. Figure 2 shows the experimental methodology. The experiment starts with trace generation using Tensilica Instruction Set Simulator [22]. The traces are then simulated for various cache configurations using our in-house cache simulator [21] (cache simulator tool was verified against DineroIV [23]). System definition provide the cache configurations and the HitME buffer configurations. Table 1 shows the L1 cache and HitME buffer configurations that are simulated in the experiment. For comparison purposes, filter cache architecture is simulated using DineroIV [23]. Cache memory area, leakage power, cache access time and access energy numbers were obtained using CACTI4.1 [24]. CACTI4.1 provides information for various semiconductor technologies. In this paper, we investigate the effect of 70*nm*, 100*nm*, 130*nm* and 180*nm* cache technologies.

L1 Cache memory			HitME size	percentage
size (Bytes)	assoc.	set size	(Bytes)	increase
1024	4	16	256	25%
2048	4	32	512	25%
4096	4	64	1024	25%
1024	8	8	128	12.5%
2048	8	16	256	12.5%
4096	8	32	512	12.5%
2048	16	8	128	6.25%
4096	16	16	256	6.25%

Table 2: HitME buffer size

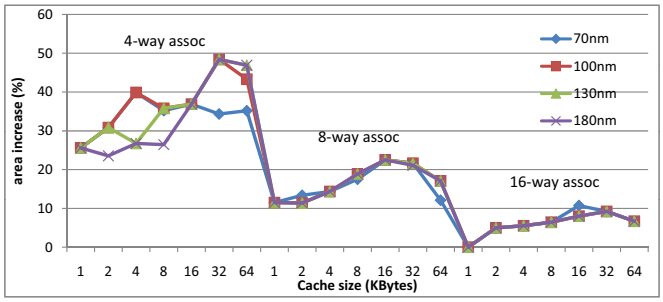

Figure 3: Area increase due to the inclusion of HitME buffer

5.1 Area Estimation

The addition of the HitME buffer will increase the on-chip area of the cache memory hierarchy. Figure 1 shows a comparison between traditional memory hierarchy (Figure 1(a)) and the proposed memory hierarchy with the HitME buffer (Figure 1(b)). In comparing the chip area of the two architectures, the differences are the addition of the HitME buffer, the multiplexer and the extra busses connecting the HitME buffer. To estimate the chip area, we assume the HitME buffer area will be much larger compared to the area for the multiplexer and the extra busses. Thus, the area comparison only considers the size of the HitME buffer in calculating the increase in chip area.

The HitME buffers are equivalent to direct-mapped caches with a total memory size equal to the L1 cache memory set size multiplied by the L1 cache block size. We use CACTI 4.1 to obtain the cache area of direct-mapped caches. Table 2 shows the theoretical increase in memory bytes on chip due to the addition of the HitME buffer. Figure 3 shows the increase of the chip area for differing L1 cache configurations (L1 cache area estimates is obtained from CACTI 4.1). The on-chip area increase shown is the ratio of the area of the HitME buffer over the area of the L1 cache memory. Observing the area increase, it can be seen that for smaller cache sizes (L1 size of 1K to 2K bytes compared to HitME buffer size of 128 to 512 bytes) the addition of HitME buffer increases the area by the theoretical size shown in Table 2. While for larger cache sizes (L1 cache size of 32K to 64K compared to HitME buffer size of 4K to 16K), it is shown that the HitME buffer increases the memory area by almost double the expected area increase. To limit the area increase and for a fair comparison on the performance of HitME architecture with existing cache architecture, we reduce the L1 cache size by the size of the HitME buffer.

5.2 Performance Estimation

System performance depends on the processor, the memory system and the off-chip memory accesses. Table 3 shows the area, clock frequency and power consumption for the ARM9E processor [25]. Table 4 shows the access time, for various cache memory configura-

Technology	$0.18\mu m$	$0.13\mu m$	$0.09\mu m$
Frequency (Mhz)	166	230	440
cycle time (ns)	6.02	4.35	2.27
Area w/o cache(mm^2)	2.00	1.07	0.61
Power w/o cache(mW/Mhz)	0.86	0.37	0.14

Table 3: ARM processor characteristic

cache memory		access time (ns)			
size	assoc.	70nm	100nm	130nm	180nm
128	1	0.40	0.57	0.74	1.03
256	1	0.40	0.57	0.75	1.03
8192	1	0.49	0.71	0.92	1.27
16K	1	0.53	0.78	0.96	1.32
1024	4	0.54	0.76	0.99	1.36
2048	4	0.56	0.79	1.02	1.43
16K	4	0.64	0.88	1.13	1.54
32K	4	0.66	0.91	1.17	1.59
1024	8	0.57	0.80	1.04	1.43
2048	8	0.55	0.82	1.04	1.42
16K	8	0.64	0.89	1.14	1.55
32K	8	0.67	0.92	1.18	1.75
2048	16	0.63	0.87	1.10	1.50
4096	16	0.63	0.87	1.11	1.51
32K	16	0.70	0.95	1.21	1.64
64K	16	0.86	1.17	1.48	1.99

Table 4: cache memory access time

tions (direct-map caches shown in Table 4 represent the HitME buffer and associative caches shown represent L1 cache). Comparison of the cache access time (L1 cache or HitME buffer) versus the processor cycle time shows that the processor clock cycle time is much slower compared to the cache access time. Hence, we assume the addition of the HitME buffer would not reduce performance as HitME buffer miss and L1 cache hit can be serviced within a processor clock cycle (this will not be the case for general purpose processor where processors can operate at clock frequency $> 3GHz$). The only factor that can cause performance degradation is if there exist many more off-chip memory accesses.

5.3 Energy Estimation

The on-chip system energy consumption is contributed to by the processor and the cache memory hierarchy. Energy consumption contributed by the processor is similar for both architectures as the number of instructions executed are the same for both architectures. The memory hierarchy energy components are the only factors that differ between the traditional cache memory hierarchy and the HitME buffer memory hierarchy. To estimate the energy consumption of the memory systems, we use the following energy equations to calculate the energy (dynamic + leakage).

Energy equation for the L1 cache memory hierarchy is given by,

$$Energy_{cache} = L1Cache_{read} * L1Cache_{readEnergy} + (L1Cache_{write} + L1Cache_{miss}) * L1Cache_{writeEnergy} + L1Cache_{leakage}$$

where $L1Cache_{read}$ is the total number of L1 cache read accesses, both instruction and data cache. $L1Cache_{write}$ is the total number of write accesses of the data cache. $L1Cache_{readEnergy}$ is the L1 cache read energy, $L1Cache_{miss}$ is the total number of L1 cache misses, $L1Cache_{writeEnergy}$ is the L1 cache write energy and $L1Cache_{leakage}$ is the total leakage energy of the L1 cache. It should be noted that the L1 cache size in HitME buffer architecture is smaller compared to the L1 cache size in traditional cache architecture, but we choose

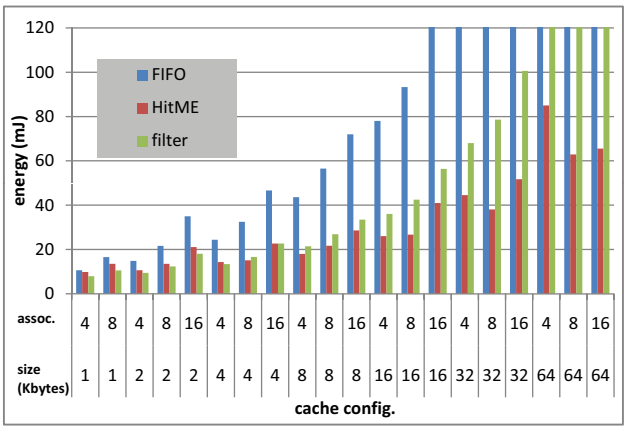

Figure 4: mpeg2dec memory energy - 70nm

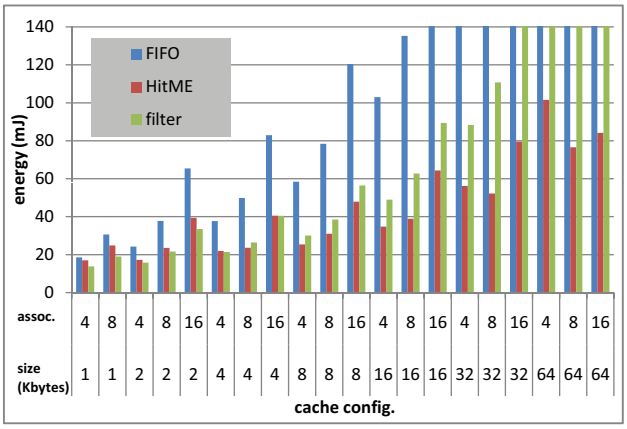

Figure 5: mpeg2dec memory energy - 100nm

to use the same cache access energy number to factor in the energy cost of accessing the multiplexor in HitME buffer architecture.

The energy consumption of the HitME buffer architecture is estimated as

$$Energy_{HitME} = HitME_{read} * HitME_{readEnergy} + HitME_{miss} * L1Cache_{readEnergy} + (HitME_{write} + L1Cache_{hit}) * HitME_{writeEnergy} + L1Cache_{miss} * L1Cache_{writeEnergy} + L1Cache_{leakage} + HitME_{leakage}$$

where $HitME_{read}$ is the total number of HitME buffer read accesses, $HitME_{write}$ is the total number of HitME write accesses, $HitME_{readEnergy}$ is the HitME buffer read energy, $HitME_{miss}$ is the total number of HitME buffer misses, $HitME_{writeEnergy}$ is the HitME buffer write energy and $HitME_{leakage}$ is the total leakage energy of the HitME buffer. All cache energy and HitME buffer energy consumption numbers include cache access energy to both tag-RAM and data-RAM.

6. RESULT ANALYSIS

Simulations were performed to compare traditional L1 cache memory with FIFO replacement policy, HitME buffer and filter cache architecture. From the area estimation analysis(Section 2), we can see that the addition of HitME buffer increases the on-chip memory area. For the purpose of comparison, all three memory architectures have approximately the same number of storage bytes to allow the comparison of different architectures. DineroIV only allows cache sizes

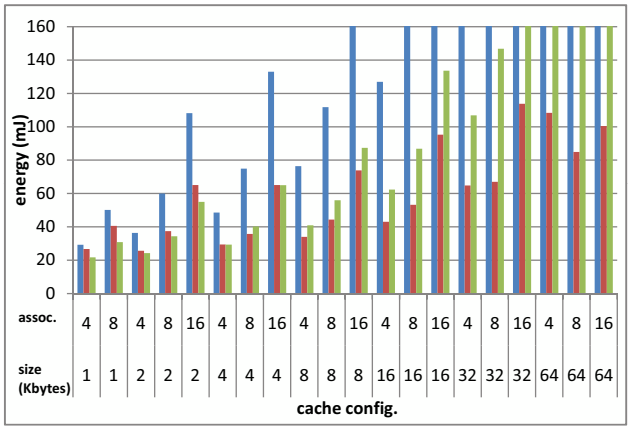

Figure 6: mpeg2dec memory energy - 130nm

to be powers of 2, due to this limitation of DineroIV, the filter cache is fixed at 256 bytes as described in [13].

Figure 4 shows the energy comparison of the three memory systems for mpeg2decode application. The x-axis represents the cache configuration and the y-axis shows the total energy (dynamic + leakage) of each memory systems. The three bars shown are (in order) FIFO, HitME buffer and filter cache. Figure 5, Figure 6 and Figure 7 show a similar graph to Figure 4 for different cache technologies — 100nm, 130nm and 180nm respectively. From the four figures, it can be seen that the HitME system and filter cache system consume less memory energy compared to traditional L1 cache architecture. For small cache sizes (1K and 2K caches), HitME architecture and filter cache consumes marginally smaller energy compared to traditional L1 cache architecture. This indicates that for small caches, the use of small filter cache or HitME buffer is not beneficial due to cache thrashing. While in large caches, significant energy reduction are seen when comparing HitME buffer to traditional L1 cache architecture.

Table 5 shows the average energy reduction results for the mpeg2dec application. Column 1 shows the total cache size (including HitME buffer). Column 2 shows the L1 cache associativity. Column 3 shows the type of the memory architecture (type 1 - traditional cache architecture, 2 - HitME buffer and 3 - filter cache). Column 4, 5 and 6 shows the number of instruction accesses, HitME buffer/filter cache misses and the L1 cache misses, respectively. Column 7, 8 and 9 shows the data accesses, HitME buffer/filter cache misses and L1 cache misses, respectively. Column 10 shows the percentage energy reduction of the HitME architecture compared to traditional L1 cache memory architecture (rows with architecture type - 2) and energy reduction of HitME architecture compared to filter cache architecture (rows with architecture type - 3).

The addition of HitME buffer reduces the L1 cache access, column 5 and 8 (in Table 5) shows the total number of L1 cache accesses (HitME miss) for instruction and data memory respectively. Results shown in Table 5 for mpeg2decode shows that on average the HitME buffer reduces L1 instruction cache accesses by 69.9% and reduces L1 data cache accesses by 74.0%. Calculating the L1 access number for all the other MediaBench application shows that on average the HitME buffer reduces L1 instruction cache accesses by 78.0% and reduces L1 data cache accesses by 75.3%.

Due to space constraints, we are unable to display the table of results for the other benchmarks. Table 6 shows the average energy reduction of the HitME memory system over traditional L1 cache memory architecture and the HitME memory system over filter cache memory architecture for various cache technologies.

Cache		arch.	instruction (millions)			data(millions)			energy
size (Bytes)	asso	type	acc.	HitME miss	L1 miss	acc.	HitME miss	L1 miss	red. (%)
1K	4	1	937.3		219.78	474.1		37.69	
1K	4	2	937.3	656.97	284.50	474.1	150.43	46.33	7.76
1K	4	3	937.3	453.32	219.78	474.1	147.44	37.69	-23.94
1K	8	1	937.3		199.93	474.1		39.64	
1K	8	2	937.3	685.60	241.46	474.1	182.02	43.89	18.42
1K	8	3	937.3	453.32	198.41	474.1	147.44	39.21	-27.98
2K	4	1	937.3		69.53	474.1		13.94	
2K	4	2	937.3	450.35	122.03	474.1	127.14	23.84	28.32
2K	4	3	937.3	453.32	69.53	474.1	147.44	13.94	-12.62
2K	8	1	937.3		56.58	474.1		10.76	
2K	8	2	937.3	513.16	81.82	474.1	147.81	14.99	37.49
2K	8	3	937.3	453.32	56.58	474.1	147.44	10.76	-9.67
2K	16	1	937.3		50.76	474.1		10.48	
2K	16	2	937.3	552.79	60.36	474.1	177.87	12.61	39.74
2K	16	3	937.3	453.32	50.63	474.1	147.44	10.45	-16.57
4K	4	1	937.3		30.06	474.1		3.25	
4K	4	2	937.3	332.49	63.84	474.1	109.62	8.83	41.20
4K	4	3	937.3	453.32	30.06	474.1	147.44	3.25	-6.76
4K	8	1	937.3		2.47	474.1		1.88	
4K	8	2	937.3	365.12	4.61	474.1	126.53	2.77	53.48
4K	8	3	937.3	453.32	2.47	474.1	147.44	1.88	9.41
4K	16	1	937.3		0.84	474.1		1.74	
4K	16	2	937.3	454.03	0.99	474.1	147.70	2.01	51.25
4K	16	3	937.3	453.32	0.84	474.1	147.44	1.74	-0.18
8K	4	1	937.3		1.32	474.1		2.55	
8K	4	2	937.3	154.91	6.09	474.1	108.47	8.13	58.64
8K	4	3	937.3	453.32	1.32	474.1	147.44	2.55	15.83
8K	8	1	937.3		0.28	474.1		0.61	
8K	8	2	937.3	285.16	0.69	474.1	109.91	0.77	61.65
8K	8	3	937.3	453.32	0.28	474.1	147.44	0.61	19.39
8K	16	1	937.3		0.18	474.1		0.62	
8K	16	2	937.3	362.20	0.25	474.1	126.41	0.69	60.20
8K	16	3	937.3	453.32	0.18	474.1	147.44	0.62	14.41
16K	4	1	937.3		0.04	474.1		0.43	
16K	4	2	937.3	66.17	0.11	474.1	105.47	1.83	66.60
16K	4	3	937.3	453.32	0.04	474.1	147.44	0.43	27.70
16K	8	1	937.3		0.01	474.1		0.33	
16K	8	2	937.3	150.10	0.02	474.1	108.71	0.44	71.38
16K	8	3	937.3	453.32	0.01	474.1	147.44	0.33	37.23
16K	16	1	937.3		0.00	474.1		0.24	
16K	16	2	937.3	284.64	0.00	474.1	109.76	0.25	67.42
16K	16	3	937.3	453.32	0.00	474.1	147.44	0.24	27.29
32K	4	1	937.3		0.01	474.1		0.19	
32K	4	2	937.3	22.69	0.02	474.1	103.03	0.26	70.90
32K	4	3	937.3	453.32	0.01	474.1	147.44	0.19	34.48
32K	8	1	937.3		0.00	474.1		0.18	
32K	8	2	937.3	66.08	0.00	474.1	105.11	0.19	78.61
32K	8	3	937.3	453.32	0.00	474.1	147.44	0.18	51.54
32K	16	1	937.3		0.00	474.1		0.19	
32K	16	2	937.3	150.08	0.00	474.1	108.65	0.19	77.49
32K	16	3	937.3	453.32	0.00	474.1	147.44	0.19	48.61
64K	4	1	937.3		0.00	474.1		0.16	
64K	4	2	937.3	0.20	0.00	474.1	101.57	0.19	74.80
64K	4	3	937.3	453.32	0.00	474.1	147.44	0.16	41.93
64K	8	1	937.3		0.00	474.1		0.16	
64K	8	2	937.3	22.67	0.00	474.1	103.00	0.16	82.48
64K	8	3	937.3	453.32	0.00	474.1	147.44	0.16	59.59
64K	16	1	937.3		0.00	474.1		0.16	
64K	16	2	937.3	66.08	0.00	474.1	105.09	0.16	83.78
64K	16	3	937.3	453.32	0.00	474.1	147.44	0.16	62.52

Table 5: mpeg2encode cache hits/misses results for 70nm technology (architecture type: 1-FIFO, 2-HitME, 3-filter cache)

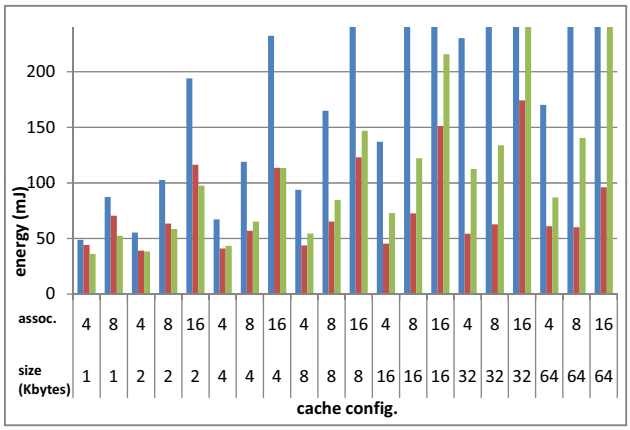

Figure 7: mpeg2dec memory energy - 180nm

appl. name	70nm tech.		100nm tech.		130nm tech.		180nm tech.	
	HitMe	HitME /filter	HitMe	HitME /filter	HitMe	HitME /filter	HitMe	HitME /filter
cjpeg	70.4%	-2.4%	70.1%	-1.2%	69.8%	3.4%	69.3%	4.3%
djpeg	60.5%	6.4%	60.4%	7.3%	60.2%	8.3%	59.8%	11.0%
g721enc	58.1%	-1.2%	58.0%	-0.3%	57.9%	0.8%	57.6%	3.5%
g721dec	61.4%	2.4%	61.3%	3.2%	61.1%	4.2%	60.9%	6.5%
mpeg2enc	58.6%	15.4%	58.4%	15.9%	58.2%	16.6%	57.8%	18.2%
mpeg2dec	56.6%	17.6%	56.4%	18.1%	56.2%	18.8%	55.9%	20.2%
average	60.9%	6.4%	60.7%	7.2%	60.6%	8.7%	60.2%	10.6%

Table 6: Average energy reduction of HitME memory systems over traditional L1 cache memory hierarchy

HitME buffer causes large reduction in L1 cache accesses. In energy terms, this translates to significant energy reductions, with an average of 58.6% less energy compared to traditional cache hierarchy for mpeg2encode application with 70nm technology. For very large L1 caches (32K and 64K L1 cache size in Table 5), we can see that energy reduction of up to 80% is observed. This can be explained by the fact that the energy costs are much higher to access a large 64K-16way associative cache compared to accessing a 4K direct-mapped cache (HitME buffer). By reducing the number of L1 cache accesses with the HitME buffer, we are able to reduce a large portion of memory energy consumption. Comparing HitME architecture with traditional FIFO cache memory for the MediaBench applications, we observed an average energy reduction of 60.9% for 70nm technology, 60.7% for 100nm technology, 60.6% for 130nm technology and 60.2% for 180nm technology in the memory system. Compared to filter cache architecture, HitME buffer reduces energy consumption by an average of 6.4% (70nm technology).

7. CONCLUSIONS

This paper presents a novel HitME buffer memory hierarchy to reduce energy consumption of on-chip memory in embedded processors. HitME buffer is a direct-mapped cache that is inserted in addition to existing traditional cache memory hierarchy.

Our experiments investigated the effect of HitME buffer for various configurations and technologies. The experimental results show that compared to traditional L1 cache memory architecture, HitME buffer on average reduces L1 cache accesses that leads to an overall energy reduction by the memory hierarchy. On average the HitME buffer reduces memory energy consumption by 60.9% when compared to traditional L1 cache memory and 6.4% when compared to filter cache architecture for 70nm cache technology.

8. REFERENCES

[1] S. Segars, "Low Power Design Techniques for Microprocessors," *ISSCC,* 2001.

[2] L. Lee, B. Moyer and J. Arends, "Low-Cost Embedded Program Loop Caching - Revisited," *Technical Report CSE-TR-411-99,* University of Michigan, 1999.

[3] J. Henkel and H. Lekatsas, "A2BC: Adaptive Address Bus Coding for Low Power Deep Sub-Micron Designs," *Proceedings of 38th Design Automation Conference,* pp.744-749, 2001

[4] H. Lekatsas, J. Henkel, and W. Wolf, "Code compression for low power embedded system design," *Design Automation Conference,* pages 294-299, 2000.

[5] S. Debray and W. Evans, "Profile-Guided Code Compression," *Proceedings of PLDI,* Berlin, Germany, June 2002.

[6] N. Gloy and M. D. Smith, "Procedure placement using temporal-ordering information," *ACM Transactions on Programming Languages and Systems,* vol. 21, no. 5, pp. 977-1027, 1999.

[7] B. Calder and D. Grunwald, "Reducing Branch Cost via Branch Alignment," *ASPLOS,* pp. 242-252, 1994.

[8] D. A. Jimenez, "Code placement for improving dynamic branch prediction accuracy," *PLDI*, 2005.

[9] A. Janapsatya, S. Parameswaran and J. Henkel, "REMcode: relocating embedded code for improving system efficiency," *Computers and Digital Techniques, IEE Proceedings,* vol. 151, no. 6, pp. 457-465, 2004.

[10] R. Banakar et al., "Scratchpad memory: design alternative for cache on-chip memory in embedded systems," *Proceedings of the tenth international symposium on Hardware/software codesign,* 2002.

[11] N. Bellas et al., "Energy and Performance Improvements in Microprocessor Design Using a Loop Cache," *ICCD,* 1999.

[12] A. Gordon-Ross, S. Cotterell and F. Vahid, "Tiny Instruction Caches for Low Power Embedded Systems," *ACM Transactions on Embedded Computing Systems,* vol. 2, no. 4, 2003.

[13] J. Kin, M. Gupta and W. H. Mangione-Smith, "The Filter Cache: An Energy Efficient Memory Structure," *IEEE Micro,* 1997.

[14] J. T. Robinson and M. V. Devarakonda, "Data Cache Management using Frequency Based Replacement," *SIGMETRICS,* pp.134-142, 1990.

[15] C. Lee et.al., "MediaBench: A Tool for Evaluating Multimedia and Communications Systems," *IEEE MICRO 30,* 1997.

[16] P. R. Panda, "Efficient Utilization of Scratch-pad Memory in Embedded Processor Applications," *European Design and Test Conference,* 1997.

[17] S. Steinke et al., "Reducing Energy Consumption by Dynamic Copying of Instructions onto Onchip Memory," *ISSS,* 2002.

[18] A. Janapsatya, A. Ignjatovic and S. Parameswaran, "Exploiting Statistical Information for Implementation of Instruction Scratchpad Memory in Embedded Systems," *Very Large Scale Integration Systems, IEEE Transactions on,* vol. 14, no. 8, August 2006.

[19] L. Lee, B. Moyer and J. Arends, "Instruction Fetch Energy Reduction using Loop Caches for Embedded Applications with Small Tight Loops," *ISLPED,* 1999.

[20] A. Gordon-Ross and F. Vahid, "Dynamic Loop Caching Meets Preloaded Loop Caching - A Hybrid Approach," *ICCD,* 2002.

[21] A. Janapsatya, A. Ignjatovic, and S. Parameswaran, "Finding Optimal L1 Cache Configuration for Embedded Systems," *The 11th Asia and South Pacific Design Automation Conference (ASP-DAC),* Yokohama, Japan, 2006.

[22] Xtensa Processor, *(http://www.tensilica.com)*

[23] J. Edler and M. D. Hill, "Dinero IV Trace-Driven Uniprocessor Cache Simulator," *http://www.cs.wisc.edu/~markhill/DineroIV/.*

[24] D. Tarjan, S. Thoziyoor and N. P. Jouppi, "CACTI 4.0," *Technical Report HPL-2006-86,* HP Laboratories Palo Alto, June 2, 2006.

[25] ARM Inc, *http://www.arm.com/products/CPUs/ARM946E-S.html*