"Reconfigurable Split Data Caches: A Novel Scheme for Embedded Systems"

Afrin Naz[1] Krishna Kavi[1] Juan Oh[1]

[1]Department of Computer Science and Engineering
University of North Texas,
Denton, TX 76203, USA
940-565-2767
email: {afrin, kavi, Juan}@cse.unt.edu

Pierfrancesco Foglia[2]

[2] Dept. Ingegneria della Informazione
University of Pisa,
Diotisalvi I-56126PIS, Italy
050-221-7530
email: foglia@iet.unipi.it

ABSTRACT
This paper shows that even very small reconfigurable data caches, when split to serve data streams exhibiting temporal and spatial localities, can improve performance of embedded applications without consuming excessive silicon real estate or power. It also shows that neither higher set-associativities nor large block sizes are necessary with reconfigurable split cache organizations. We use benchmark programs from the MiBench suite to show that our cache organization outperforms an 8k unified data cache in terms of miss rates, access times, energy consumption and silicon area. Finally we show how the saved area can be utilized for supporting techniques for improving performance of embedded systems. Our design enables the cache to be divided into multiple partitions that can be used for different processor activities other than conventional caching. In this paper we have evaluated one of those options to support "prefetching".

Categories and Subject Descriptors
C.1.1. [Processor Architectures]: Single Data Stream Architecture --- cache memories

General Terms
Performance, experimentation, measurement and Design.

Keywords
Embedded systems, Split cache, reconfigurability, locality, cache.

1. INTRODUCTION
In today's microprocessors, cache has become a vital element in improving performance over a wide range of applications. Studies have found that the on-chip cache is responsible for 50% of an embedded processor's total power dissipation [3,5,16]. For that reason we feel that it is worthwhile investigating new reconfigurable cache organizations to address both performance and the power requirements. The performance of a given cache architecture is largely determined by the behavior of the applications. Unfortunately the manufacturer typically sets the cache architecture as a compromise across several applications. This leads to conflicts in deciding on total cache size, line size and associativity. For embedded systems where everything needs to be cost effective, this "one-size-fits-all" design philosophy is not adequate. In this paper we apply reconfigurability to the design of caches that address these conflicting requirements and explore how to design caches that achieve high performance for embedded applications while remaining both energy and area efficient.

The key contributions of this work are the following. First, we introduce a novel cache architecture for embedded microprocessor platforms. This proposed cache will detect program access patterns and fine-tune cache policies to improve both data localities and the overall cache performance for embedded applications. Second, our design enables the cache (as we save area) to be divided into multiple partitions that can be used for purposes other than conventional caching. In this paper we evaluate our cache architecture that uses reconfigurability coupled with split data caches (separate array and scalar data caches) complemented by a very small victim cache. Our goal is to reduce (silicon) area, access time, and dynamic power consumed by cache memories while retaining performance gains. In our design, we address the problem of improving cache performance in embedded systems through the use of separate array and scalar data caches. Then we further extend our architecture by augmenting the scalar cache with a victim cache [19]. Victim caches are based on the fact that reducing cache misses due to line conflicts for data exhibiting temporal locality is an effective way of improving cache performance, without increasing the overall cache associativity. In this paper we also study how our cache organizations can be reconfigured based on an application's behavior. By setting a few bits in a configuration register, the cache can be configured by software for optimum sizes for each of our three structures (array cache, scalar cache, victim cache) and use the rest of the unused area for other processor activities. The cache system can also be configured to shutdown certain regions in order to effectively reduce energy consumption. For both cases, the reconfiguration leads to only a small overhead in terms of time, power, silicon area and hardware complexity. In this paper, we provide the details of our configurable cache. When using our augmented split caches for embedded applications, our results show excellent reductions in both memory size and memory access time, translating into reduced power consumption. Our cache architecture reduces the cache area by as much as 78%, execution time by as much as 55%, and energy consumption by as much as 67%, when compared with an 8k byte direct-mapped unified data cache with a 32k byte level-2 cache. If we consider tradeoffs in performance improvement, we can achieve as much as a 83% reduction in area consumption (without any increase in execution time) and by as much as a 61% decrease in execution cycles (without any increase in silicon area). These reductions can be profound when working with small L-1 caches often found in embedded systems.

The space savings resulting from our cache structures may be used for many architectural features to further improve the performance of embedded systems. Techniques such as hardware prefetching, instruction reuse, branch predictions have been used effectively in desktop applications. However these techniques require additional hardware for implementing look-up tables, which lead to increased size and power requirements. Since reductions in cache sizes are acquired in our designs (while not sacrificing performance or increasing power consumptions), the look-up tables for these optimizations could be implemented in a partition of the reconfigurable cache instead of using other valuable chip area. In this paper we study one such technique, prefetching, with reconfigurable caches. With prefetching our cache architecture reduces execution time by as much as 67% when compared with an 8kbyte direct-mapped unified data cache with a 32k byte level-2 cache. Even with the additional power consumed by the prefetching, our studies show significant reductions in energy consumption.

The rest of the paper is organized as follows. Section 2 provides a survey of related work, while section 3 describes the architectural design of our reconfigurable cache. In section 4, we describe the benchmarks and experimental set up used in our evaluation. In section 5 we evaluate our reconfigurable cache and in section 6 we evaluate our prefetching option. Finally we present our conclusions in section 7.

2. PREVIOUS WORK

Ranganathan *et al* [22] proposed a reconfigurable unified data cache architecture for general purpose processors. They proposed dividing cache into different partitions that can be used for different processor activities. Ranganathan *et al* did not provide an analysis of silicon area involved in the reconfigurable cache, but explored different design alternatives, focusing on one option that of using the saved silicon area for "instruction reuse". We provide a detailed analysis of silicon area involved in our cache organizations. We also perform detailed analyses of execution cycles and energy consumed using our cache structures to demonstrate the gains achieved by our cache. We concentrate on embedded benchmarks for our evaluation. Albonesi *et al* [11] proposed "selective cache ways" to selectively disable portions of unified data cache, trading off performance with power. In our analyses, in addition to trading off performance with power, we also explore how unused cache portions can be used for other purposes (such as prefetch buffers), providing further options in design trade-offs.

Work by Vahid *et al* [4] is closely related to our research, as they evaluate reconfigurable unified data caches for embedded applications. Unlike the work by this research team, we do not see associativity as an important reconfigurable design parameter. This is because, both our array and scalar caches are designed as direct mapped caches, and we use victim caches to solve associativity for scalar data. In addition to showing performance gains and power reductions, we also analyze silicon area savings obtained from our caches. Instead of shutting down cache area to save power, we also explore how the unused portion of cache area can be used for other architectural features that can improve applications' performance. To summarize, the most important significance of our work is our comprehensive analysis of execution cycles and area reductions achieved by our caches, which is not done by anyone before. Several studies have been reported on split data caches [8, 12, 18, 20, 25] but there has been no work reported on reconfigurable split data caches.

3. ARCHITECTURAL DESIGN

Figure 1 shows our proposed Reconfigurable Split cache architecture, with array and scalar data caches, victim cache with scalar data cache and the instruction cache augmented by a small prefetching buffer. In order to speedup access, current cache implementations partition caches into multiple sub-arrays [7,10]. For example, the SA-110 embedded microprocessor [7] uses 32-way associative 16KB L1 instruction and Data caches, each of which is divided into 16 fully associative sub-arrays. With this

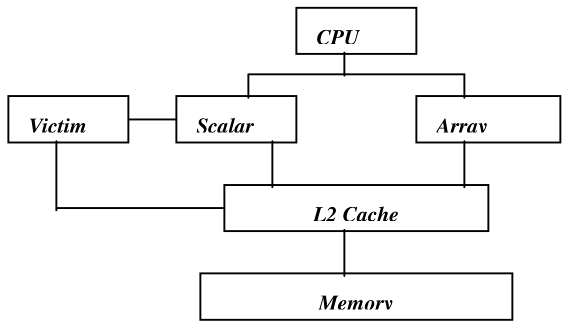

Figure 1. Reconfigurable split cache organization

partitioning in place, our reconfigurable caches can easily be implemented if there are at least as many sub-arrays as the maximum number of partitions (because for a reconfigurable cache different partitions must be implemented in physically different sub-arrays indexed by different addresses). In order to implement reconfigurable caches, only a small amount of additional logic is required. Additional wiring is also necessary from the cache to the processor for directing data to/from the various partitions. The most challenging part in designing a reconfigurable cache is the implementation of a mechanism to divide the cache into different (variable sized) partitions and designing an addressing scheme that can address any partition. Ranganathan *et al* [22] have already proposed two partitioning and addressing schemes: "Associativity based partitioning" and "Overlapped wide-tag partitioning". In our design we use "Overlapped wide-tag partitioning" scheme. In this scheme, the key challenge is to devise a mechanism so that the size of the array tag can be dynamically changed with the size of partitions (since the number of bits in a tag and index fields of the address will vary based on the size of the partition). We restrict the size of each partition to a power of 2 and support a limited number of possible configurations (usually two or three). A reconfigurable cache with N partitions must accept N addresses and generate N hit/miss signals. In order to track the number and sizes of the partitions and control hit/miss signals, a special hardware register is needed. This register will be a part of the processor state.

The additional logic will add to silicon area, access time and power consumed. Ranganathan *et al* [22] have studied the impact of reconfigurable cache organizations on cache access times and showed that for a small number of partitions, reconfigurable caches increase the cache access time by less than 5%. In a different study, Vahid *et al* have shown that a reconfigurable cache does not consume significantly additional power over traditional cache structures [4]. In this paper we have used the CACTI timing model [23] to obtain values for these overheads of our reconfigurability.

A reconfigurable cache can be used in different ways. The best configuration for an application can be determined by extensive simulations (or actual executions). Software profiling tools, used to identify portions of code that exhibit different cache behaviors, can also be used. Reconfiguration can also be implemented dynamically with appropriate hardware profiling and an automatic cache tuner.

4. EXPERIMENTAL METHODS

In this section we describe the experimental framework and the benchmarks used for this study. We also define performance metrics and power models used in our studies.

4.1 Benchmarks

We use benchmark programs from the MiBench suite[17]. MiBench includes benchmarks from several embedded application

domains. In order to cover a wide range of applications in, in our study we included benchmarks from (1) Automotive (2) Office Automation, (3) Networking, (4) Security, and (5) Telecommunications groups. The descriptions of the benchmarks used in our studies are listed in Table 1. Since the performance of our system depends on the percentage of memory references caused by an application, we also include the load/store percentage for each benchmark.

4.2 Simulation

Our experimental environment builds on the SimpleScalar (version 3.0d) simulation tool set [6] modeling an out-of-order speculative processor with a two-level cache hierarchy. We rely on default parameters defined by SimpleScalar.

The base cache system, which is the cache with which we will compare our configurable cache, uses an 8k byte L1 instruction cache, an 8k byte L1 data cache and a 32 k byte unified L2 cache. Our performance evaluation includes silicon area needed for cache structures, because embedded systems designers are interested not only in performance but also in better use of silicon area. We use CACTI[23] for computing silicon areas need by caches. We use a modified CACTI timing model to obtain overheads of our reconfigurability. We include energy consumed due to misses and off-chip accesses. Our analysis use the following general equations to compute the dynamic power consumption of a cache.

power = Hit * power_hit + Miss * power_miss

power_miss = OPC + PCW + FTM

We obtained values for *hits* and *misses* for our array and scalar caches by executing the selected benchmarks on the Simplescalar simulator. Here it should be mentioned that

Table 1: Descriptions of benchmarks

Name	Description	% of L/S	Name in fig
bit count	Test bit manipulation	11	bc
qsort	Computational Chemistry	52	qs
dijkstra	Shortest path problem	34.8	dj
blowfish	Encription/decription	29	bf
sha	Secure Hash Algorithm	19	sh
ri	Encryption Standard	34	ri
string search	Search mechanism	25	ss
adpcm	Variation of PCM	7	ad
CRC	Redundency check	3 6	cr
FFT	Fast Fourier Transform	23	ff

different cache structures have different *power_hit* values based on the cache type, size and hit type of each access. The *PCW* is the power consumed to write an entire line to the cache. OPC is the power needed for off-chip access and calculated as $0.5 * Vdd^2 * (0.5 * W_{data} + W_{addr})) * 20pF$ [15, 16, 21, 23, 24], where W_{data} and W_{addr} are the number of bits for both the data sent/returned and the address sent to the next level of memory on a miss request. The last term is the load capacitance for off-chip destinations. At any miss the overhead for searching in cache is also included as FTM (First Time Miss).

5. EVALUATION

In this section we present the results from our evaluation, comparing our cache organization with the base cache architecture. The standard way to evaluate the impact of several parameters is to vary one of the parameters while keeping the others fixed, which we follow in following sections.

5.1 Area

For some embedded applications size reduction is far more important than being faster or less power consuming. As a side-benefit, in many of these applications, reducing the foot-print of processing resources can also lead to reduced power consumption. In this subsection we show how our cache design leads to substantial reduction in size (in terms of silicon area needed). For this purpose we compare the areas consumed by our cache while achieving equal or fewer execution cycles as compared to an 8k base cache. In other words, we fix the number of cycles to show how our design requires a smaller foot-print. The first series in figure 2 shows the percentage reduction in area needed by using our system instead of the base cache, while requiring no more execution cycles than the base case of 8KB unified data cache. From Figure 2 we can see that for half of the benchmarks, our system offers more than 80% reduction in silicon area.

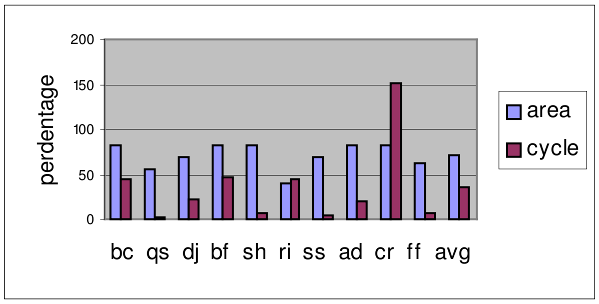

Figure 2: Percentage of area and cycle reduction

In Figure 2 we also compare the number of cycles (total execution time) needed when using our cache system with that using unified data cache of equal size. By this we mean, if there is 75% reduction in area for our cache leading to a total L-1 size for scalar, victim and array portions, we allocate the same cache capacity for the unified data cache – thus keeping cache sizes equal in both designs. For some benchmarks (cr, bf, ri) we can see that there is a large increase in execution time for a unified data cache with smaller overall cache capacity (as compared to our split data caches). For these benchmarks separation of data into array and scalar data significantly reduces the number of conflict cache miss. For other benchmarks (ss, ff) we can see that our cache does not show any reduction in execution time. This is easy to see (also check Table 1) since these applications have very small number of load/store instructions and thus any optimization to cache substructures has very minimal impact on the program execution. For these cases our cache structure can be reconfigured to gain other benefits including shutting off portions of caches to save energy, or utilizing unused portions of caches for purposes other than caching. We will explore these options in a later section.

5.2 Performance

Most modern embedded applications are demanding higher performance and added features. In such applications, faster execution of programs may be more important than reducing the foot-print of the computing system. Such systems may afford larger caches, say 8KB or larger L-1 data caches. Here we will show how our design reduces the execution times while using equal (or smaller) amounts of area for caches. Note that our system uses 3 cache structures --(scalar cache, victim cache and array cache). This presents more design choices in terms of selecting a size for each of these structures.

Optimal sizes for each of the cache structures are selected in order to obtain overall reduction in execution cycles while maintaining the same overall silicon area needed (as that of the base case using an 8KB unified data cache). Sometimes the optimal selections of sizes for the different structures may lead to overall cache areas that are somewhat different from our target sizes. For example if we use 4KB scalar, 512-byte victim cache and 2KB array cache,

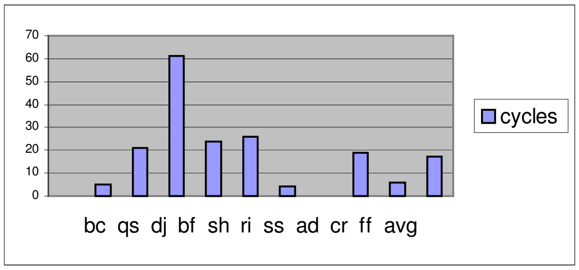

Figure 3: Percentage of cycle reduction without increased area

the total size is less than 8KB, but rounded to the nearest power of 2 (which is 8KB). For most cases the total area needed by our cache is smaller than the size of the base cache. In a few cases, we needed less than 512 additional bytes when compared to the 8KB base cache. Figure 3 shows the percentage improvement in execution times of applications assuming (approximately) equal numbers of bytes of cache for our designs and those for the base case. Those benchmarks that showed significant improvement in terms of silicon areas (Figure 3), also show reductions in execution cycles (Figure 3). The benchmarks for which our design did not show reductions in area, do not show significant performance gains with our designs. Once again this should be expected since these benchmarks do not involve many memory accesses. For two benchmarks sh and ff, although the percentages of memory references are small (19 and 23%) large improvements are achieved by our cache. For these applications, the separation of data types into scalar and stream (or array) is the main source of the performance gains. In Figure 3 we are also showing the average execution time across all the benchmarks used in our experiments.

5.3 Power consumption

The most important concern for the designers of any embedded system is power consumption. Our overall goal is the reduction of

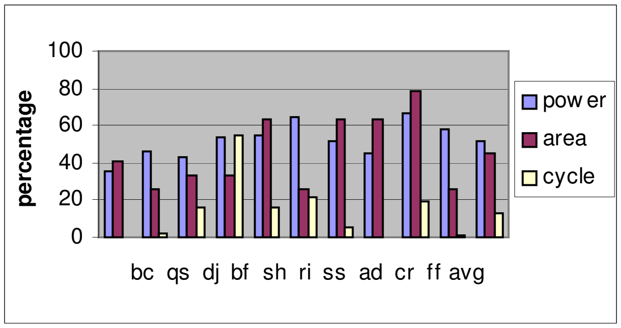

Figure 4: Percentage reduction of power, area and cycle

energy consumption by capitalizing on both the split cache organization and the reconfigurability of our cache structures.

Moreover the concomitant reductions in execution cycles and silicon areas, can further contribute to energy savings. In this subsection we will show the overall improvement in energy consumption by combining the efforts described in previous results.

The three series in Figure 4 represent percentage reductions in power, area and cycles respectively. As we can see, on average we show more than 50% reduction in power. Each of the benchmarks also provides reduction in cycles (around 1% for bc and ad) and significant reduction in area consumption.

6. UTILIZATION OF ADDITIONAL AREA

From the results shown in section 5 we can make two observations. First our cache design will result in silicon area savings, as high as 84% and an average savings of 60% across the selected benchmarks (see Figure 3). Second, our designs also consume less power than conventional unified data caches. We have shown (Figure 4) that we can achieve as high as 63% reduction in energy consumption, with an average of 50% reduction across all the benchmarks used.

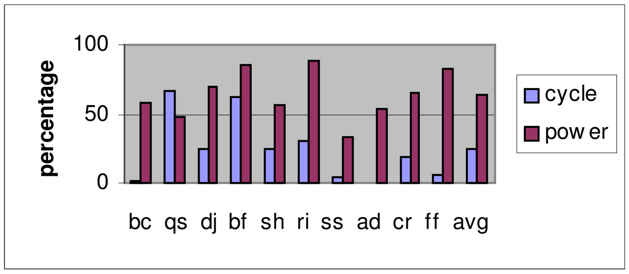

Figure 5: Percentage of cycle reduction with prefetching

When provided with larger caches, we can either disable unused sub-arrays of cache to save energy or use the sub-arrays for purposes other than traditional caching, so that execution performance can be further improved. We propose our reconfigurable cache to enable its dynamic partitions to be assigned to processor activities other than conventional caching. Techniques such as hardware prefetching, instruction reuse, value prediction and branch prediction have been used effectively in desktop applications. However, these techniques require additional space for implementing look-up tables or buffers (viz trace caches, branch prediction buffers) and the achievable performance gains increase with the size of these tables [26]. Because of additional tables, these techniques are often viewed as inappropriate for embedded systems [5]. Since we show reductions in cache sizes needed for our designs (while not sacrificing performance or increasing power consumptions), these savings may be used to implement look-up tables or buffers to implement elaborate branch prediction or instruction reuse ideas. To provide evidence of the benefits of reconfigurable caches, in this paper we study one such technique, hardware prefetching.

Prefetching or exploiting the overlap of processor computations with data access has proven to be effective in tolerating large memory latencies [2, 13]. Prefetching can be either hardware [13] or software based, [2]. Successful prefetching can reduce miss rates, but scheduling the prefetching requests is still a challenge Prefetching too far ahead not only wastes the embedded system's valuable power but may also cause cache pollution, since the prefetched data may displace data that will be used before the prefetched data. This in turn leads to additional misses and wasted energy. On the other hand prefetching too late will not hide the latency. In our reconfigurable cache we can use separate partitions for prefetched data and avoid cache pollution. The prefetching areas can be implemented in cache arrays with minor hardware and software changes.

Figure 5 shows the percentage improvement in execution times and reductions in power consumptions of applications using prefetching (along with our scalar, victim and array caches) when compared to the base 8KB unified data cache. As can be seen, some benchmarks (qs, bf) show more than 60% reductions in execution cycles with prefetching. The benchmarks with very few load/store instructions did not show significant improvement. For two benchmarks, ad and bc, as the percentage of memory references are very low (7 and 11 % respectively) prefetching did not show further improvements in execution cycles over those shown in Figure 4. However for all of benchmarks there is a significant reduction in power consumption. While access to memory is hidden with prefetching, additional energy is consumed by prefetching. The data in Figure 5 accounts for the added energy for prefetching. Thus our data shows that our split data cache augmented by prefetching (and victim cache) can improve performance and reduce power consumption of embedded benchmarks when compared to a unified data cache. The average power savings in Figure 5 is 64% and the average performance improvement is 23%.

7. CONCLUSIONS

In this paper we introduce a novel cache architecture for embedded microprocessor platforms. Our proposed cache architecture uses reconfigurability coupled with split data caches (separate array and scalar data caches) containing a very small victim cache to reduce (silicon) area and dynamic power consumed by cache memories while retaining performance gains. Our cache architecture reduces the cache size by as much as 78% (average 45%), execution time by as much as 55% (average 13%), and energy consumption by as much as 67% (average 52%) when compared with an 8KB direct-mapped L-1 unified data cache (in addition to 8KB L-1 instruction cache) with a 32KB level-2 cache (for both data and instructions). If we consider tradeoffs in performance we can achieve as much as 83% reduction in area consumption (without any increase in execution time) and as much as 61% in cycles (without any increase in silicon area).

Our design enables the cache (as we save area) to be divided into multiple partitions that can be used for other processor's activities (such as hardware prefetching, instruction reuse, branch predictions) or the cache system can also be configured to shutdown certain regions. Since our reconfigurable approach leverages the subarray partitioning that is already present in modern caches, only minor changes to conventional caches are required. The reconfiguration only requires a small overhead in terms of silicon area, power and execution times. In this paper we evaluated how the unused cache subarrays can be used for prefetching. We show that such a use will lead to as much as 67% reduction in execution times when compared with the base case (8KB direct-mapped unified data cache with a 32k level-2 cache). Even accounting for additional power consumed by prefetching, our structures show an average power reduction for embedded applications of 64% over traditional unified data caches. Since our goal is to find the impact of reconfigurability on split data cache, we have worked only with data caches. In future we will explore a combined instruction and (our split) data caches in reconfiguring choices. We will also explore how unused cache portions can be used for instruction reuse, value prediction and branch predictions.

8. REFERENCES

[1] A. Gordon-Ross, F. Vahid and N. Dutt, Automatic tuning of two-level caches to embedded applications, Design Automation and Test in Europe Conference (DATE), February 2004, pp. 208-213.

[2] C.K. Luk and T. Mowry, Compiler based prefetching for recursive data structures, in Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems, Oct. 1996, pp. 222-233.

[3] C. Zhang, F. Vahid and W. Najjar, Energy benefits of a configurable line size cache for embedded systems, IEEE International Symposium on VLSI Design, Tampa, Florida, February 2003.

[4] C. Zhang, F.Vahid and W.Najjar, A highly configurable cache architecture for embedded systems, in Proceedings of 30th Annual International Symposium on Computer Architecture, June. 2003, pp.136 -146.

[5] C. Zhang and F. Vahid, Using a victim buffer in an application-specific memory hierarchy**,** Design Automation and Test in Europe Conference (DATE), February 2004, pp. 220-225.

[6] D. Burger and T. M. Austin. "The SimpleScalar Tool Set, Version 2.0", *Tech. Rep. CS-1342*, University of Wisconsin-Madison, June 1997.

[7] E. McLellan. The Alpha AXP architecture and 21064 processor. *IEEE Micro*, 13(4):36–47, June 1993.

[8] F. J. Sanchez, A. Gonzalez, and M. Valero, "Software Management of Selective and Dual Data Caches*", IEEE TCCA NEWSLETTERS*, March 97, pp. 3-10.

[9] G. Ammons, T. Ball, and J. Larus. Exploiting hardware performance counters with flow and context sensitive profiling. *Proceedings of the ACM SIGPLAN Conference on rogramming Language Design and Implementation*, June 1997.

[10] G. Lesartre and D. Hunt. PA-8500: The continuing evolution of the PA-8000 family. *Proceedings of Compcon*, 1997.

[11] H. Albonesi, "Selective Cache Ways: On-Demand Cache Resource Allocation," Journal of Instruction Level Parallelism, May 2000.

[12] J. A. Rivers and E. S. Davidson, "Reducing Conflicts in Direct-Mapped Caches with a Temporality based Design, *Proc. 1996 International Conference.*

[13] J. L. Baer and T. F. Chen, "An effective on –chip preloading scheme to reduce data access penalty. "In *Proceedings of the Supercomputing'91*, pp. 176-186, 1991

[14] J. Montanaro et al. A 160-MHz, 32-b, 0.5W CMOS RISC microprocessor. *Digital Technical Journal*, 9(1):49–62, 1997.

[15] M.B.Kamble and K.Ghose, Energy-efficiency of VLSI caches: a comparative study, in Proceedings of Tenth International Conference on VLSI Design, Jan. 1997, pp.261-267.

[16]M.B.Kamble and K.Ghosse, Analytical energy dissipation models for low power caches, in Proceedings of International Symposium on Low Power Electronics and Design, Aug. 1997, pp.143 -148.

[17] M. Guthaus, J. Ringenberg, T. Austin, T. Mudge, R. Brown, "MiBench: A free, commercially representative embedded benchmark suite, *in Proceedings of the IEEE 4th Annual Workshop on Workload Characterization*," Austin, TX, December 2001.

[18] M. Tomasko, S. Hadjiyiannis, and W. A. Najjar, "Experimental Evaluation of Array Caches", *IEEE TCCA Newslatters*, March 97, pp. 11-16.

[19]N. P. Jouppi, Improving direct-mapped cache performance by the Addition of a small fully associative cache and prefetch buffers, in Proceedings of the 17th ISCA, May 1990, pp. 364-373.

[20] O. S. Unsal, I. Koren, C. M. Krishna, C. A. Moritz, "The Minimax Cache: An Energy-Efficient Framework for Media Processors," *8th International Symposium on High-Performance Computer Architecture, HPCA8,* Cambridge, MA, February 2002, pp. 131-140.

[21] P. Jung-Wook, K. Cheong-Ghil, L. Jung-Hoon, K. Shin-Dug, "An energy efficient cache memory architecture for embedded systems" Proceedings of the 2004 ACM symposium on Applied computing, march 2004

[22] P.Ranganathan, S. Adve, and N.P. Jouppi, "Reconfigurable Caches and their Application to Media Processing," Int. Symp. on Computer Architecture, 2000.

[23] S.J.E.Wilton and N.P.Jouppi, "CACTI: an enhanced cache access and cycle time model," IEEE Journal of Solid-State Circuits, Volume: 31 Issue: 5 , May 1996, pp.677 -688.

[24]The MOSIS Service, http://www.mosis.org

[25] V. Milutinovic, M. Tomasevic, B. Markovic, and M. Tremblay, "The Split Temporal/Spatial Cache: Initial Performance Analysis," *SCIzzL-5*, Mar. 1996.

[26] Y. Sazeides and J. E. Smith, "The predictability of Data values", In Proceedings of the 30th Annual International Conference on Microarchitecture, pages 248-258, 1997.