A Study of Reconfigurable Split Data Caches and Instruction Caches

Afrin Naz afrin@cs.unt.edu
Krishna Kavi kavi@cse.unt.edu
Philip Sweany Philip@cse.unt.edu
Wentong Li wl@cs.unt.edu
Department of Computer Science and Engineering
P.O. Box 311366, University of North Texas, Denton, Texas 76203

Abstract

In this paper we show that cache memories for embedded applications can be designed to both increase performance and reduce energy consumed. We show that using separate (data) caches for indexed or stream data and scalar data items can lead to substantial improvements in terms of cache misses. The sizes of the various cache structure should be customized to meet applications' needs. We show that reconfigurable split data caches can be designed to meet wide-ranging embedded applications' performance, energy and silicon area budgets. The optimal cache organizations can lead to on average 62% and 49% reduction in the overall cache size, 37% and 21% reduction in cache access time and 47% and 52% reduction in power consumption for instruction and data cache respectively when compared to an 8k byte instruction and an 8k byte unified data cache for media benchmarks from MiBench suite.

Keywords: Embedded Systems, Cache memories, Split data cache, Victim Cache, Reconfigurability.

1 Introduction

Studies have found that the on-chip cache is responsible for 50% of an embedded processor's total power dissipation [3, 5]. For that reason we feel that it is worthwhile investigating new reconfigurable cache organizations to address both performance and the power requirements of embedded applications. Our experiments show that instruction cache with prefetching and split data caches (scalar data cache augmented with victim cache, and a separate array data cache) are effective in embedded systems when used in conjunction with dynamic reconfigurability of cache components.

Our goal is to reduce (silicon) area, access time, and dynamic power consumed by cache memories while retaining performance gains. In our design, we first address the problem of improving cache performance in embedded systems through the use of reconfigurability in separate array and scalar data caches. Then we extend our architecture by augmenting the scalar cache with a victim cache [14]. 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. Inspired by the reduction in silicon areas, and power consumptions resulting from our split caches we then implemented reconfigurability with our instruction cache, which is augmented by a small prefetch buffer. The prefetch buffer will utilize the silicon area savings achieved in our data cache designs. By setting a few bits in a configuration register, the cache can be configured by software for optimum sizes for each of our four structures (array cache, scalar cache, instruction cache and prefetch buffer) 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 hardware 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. 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 85% and 78%, cache access time by as much as 72% and 36%, and energy consumption by as much as 75% and 67% for instruction and data caches respectively (when compared with an 8k byte instruction and 8k byte unified data caches). These reductions can be profound when working with small L-1 caches often found in embedded systems. We believe there are three reasons behind the success of our cache architecture. First the separation of array and scalar data items eliminates mutual interference caused by these two types of data (and reduces conflict misses). The second reason is the greater reconfigurable design space afforded by our cache structures which allows more chances of improvement. Finally adding a small prefetch buffer to the instruction cache allows us to reduce cold misses in instruction access. Even with the additional power consumed by the prefetching for instruction cache, our studies show significant reductions in total energy consumed by our caches.

The space savings resulting from our cache structures may be used for many architectural features such as instruction reuse buffers and branch prediction buffers to further improve the performance of embedded applications. In this paper we survey possible optimization techniques to be implemented in our saved area.

The rest of the paper is organized as follows. Section 2 provides a survey of related work, while section 3 describes interactions between different cache parameters in embedded systems. In section 4, we describe the architectural design of our reconfigurable cache. In section 5 we evaluate our reconfigurable cache and in section 6 we provide a survey of different optimization techniques. Finally we present our conclusions in section 7.

2 Previous Work

Ranganathan *et. al* [17] proposed a reconfigurable unified data cache architecture for general purpose processors. They proposed dividing data 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 of reconfiguring caches for "instruction reuse". Albonesi *et al* [9] proposed "selective cache ways" to selectively disable portions of unified data cache, trading off performance with power. Neither of these analyzed the impact of reconfigurabilty on instruction cache. Work by Vahid *et. al.* [4] is closely related to our research, as they evaluate reconfigurable instruction and unified data caches for embedded applications. Unlike this research, we do not see associativity as an important reconfigurable design parameter. This is because, both our array and scalar data caches are designed as direct mapped caches, and we use a victim cache to effectively provide higher associativity for scalar data. Also inclusion of a very small instruction buffer allows us to remove the cold misses in our direct mapped instruction cache.

The main difference of our work when compared with others is, in addition to showing performance gains and power reductions, we also analyze silicon area savings obtained from our caches. The most significant aspect of our work is using separate data cache with reconfigurability. Previous research did not consider reconfigurable caches within the context of separate data caches [4, 9, 13, 15, 17].

3 Influence of different Parameters

In this section we will discuss the impact of power and associativity on our proposed cache system.

3.1 Dynamic power consumption

In CMOS circuits the major source of power consumption is dynamic power. Dynamic power dissipation is due to logic switching current and the charging and discharging of the load capacitances. In our power consumption evaluation we include energy consumed due to cache misses and off-chip accesses. Our model uses 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 [6]. 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 is calculated as $0.5 * Vdd^2 * (0.5 * W_{data} + W_{addr})) * 20pF$ [11, 18], 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. The last term is the load capacitance for off-chip destinations. For any miss the overhead for searching in cache is included in FTM (First Time Miss).

3.2 Influence of associativity

Higher associativity in both data and instruction caches is identified as the most important reconfigurable parameter by Vahid *et al*. [4]. In unified data caches, the associativity of the cache plays a significant part in the overall performance/power trade-offs. However, when the data cache is split (as in our case into array and scalar caches), we found that associativity is no longer a significant reconfigurable parameter. At L-1 cache (which is our primary concern), it is important to maintain a balance between miss rates and access times. In our design, direct mapped caches provide for such a balance, as conflicts between different classes of data (viz., arrays and scalars) are eliminated by our split cache organizations. Additionally, since we provide for a small victim cache with the direct mapped scalar cache, the miss rates are further reduced, without having to resort to higher associativities. For instruction cache, we believe that cold misses are more problematic to performance than conflict misses, and we use a small pre-fetch buffer to reduce cold misses.

4 Architectural design of proposed cache

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 implementations partition caches into multiple sub-arrays [7, 8]. 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 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). Figure 2 shows the structure of a cache using SRAM technology. This figure also includes the sections where additional multiplexors are added to implement reconfigurability (referred by shaded numbered blocks) [17].

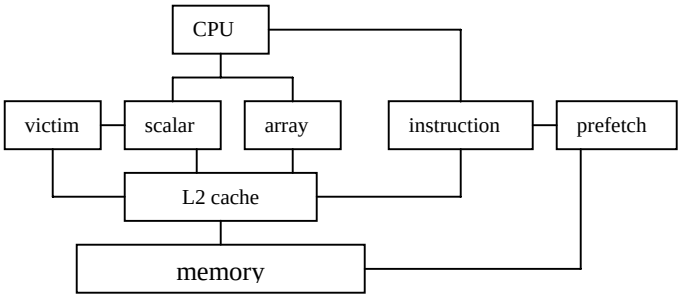

Figure 1 Reconfigurable split cache organization

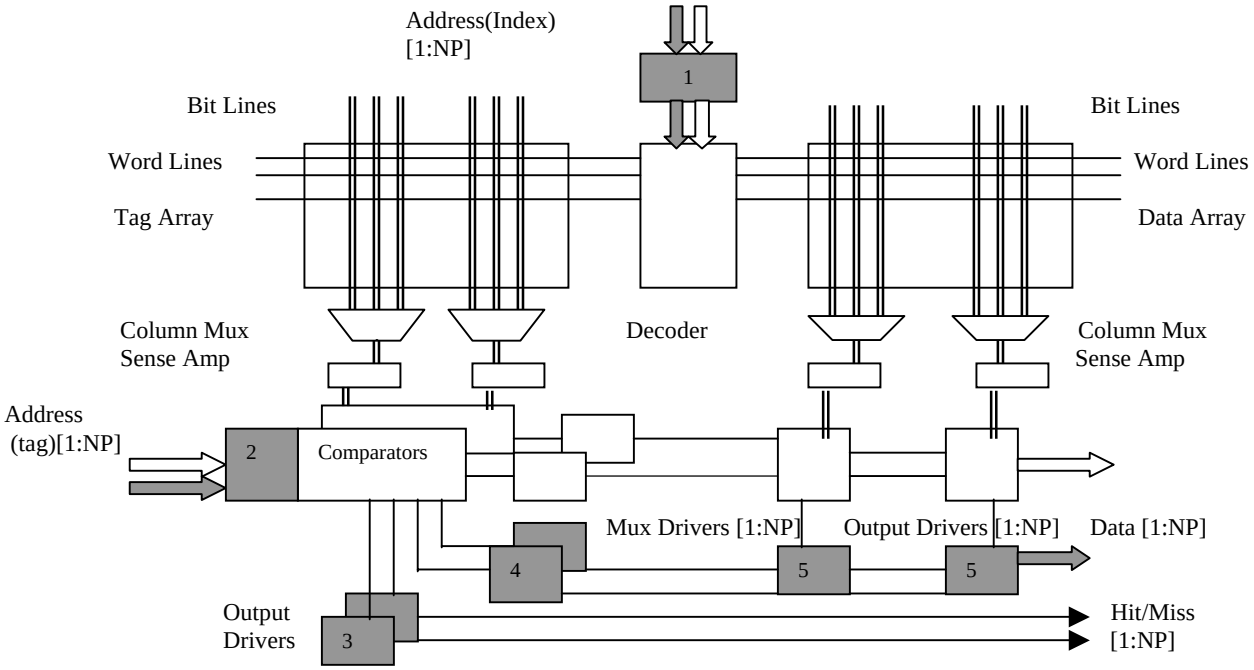

Figure 2 Additional logic for Reconfigurable cache [17]

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.* [17] 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).

The additional logic will add to silicon area, access time and power consumed. Ranganathan *et. al.* [17] 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 this paper we have used the CACTI timing model [18] to obtain values for these overheads of our reconfigurability.

Other major issues in designing reconfigurable split caches include determining how to find the best configuration and maintaining data consistency. A reconfigurable cache can be used in different ways. The best configuration for an application can be determined by extensive simulations (or actual executions). For

detailed information about maintaining data consistency see [17].

5 Evaluation of Reconfigurable Split Cache

In this section we present the results from our evaluation, comparing our cache organization with the base cache architecture consisting of 8k byte L1 instruction cache, 8k byte L1 data cache and a 32 k byte unified L2 cache.

5.1 Methodology

We use benchmark programs from the MiBench suite[12]. The descriptions of the benchmarks used in our studies are listed in Table 1.

Table 1: Descriptions of benchmarks

Benchmark	Description	% load/store	Name in fig
bit counts	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
rijndael	Encryption Standard	34	ri
String search	Search mechanism	25	ss
adpcm	PCM standard	7	ad
CRC32	Redundency check	36	cr
fft	Fast Fourier Transform	23	ff

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 used to compare our architecture uses an 8k byte L1 instruction cache, an 8k byte L1 data cache and a 32 k byte unified L2 cache.

5.2 Results

In Figure 3 we show the reduction in miss rates with increasing cache size for both instruction (a) and data (b) caches. For several benchmarks (“ad”, “cr”, “bc” in Figure 3(a) and “bc” in Figure 3(b)), the miss rates are too small (comparing to other benchmarks) to be visible in the figure. The three series in Figure 4 represent percentage reductions in power, area and access time for instruction and data caches respectively. In this figure we also show the average power, area and cache access time across all

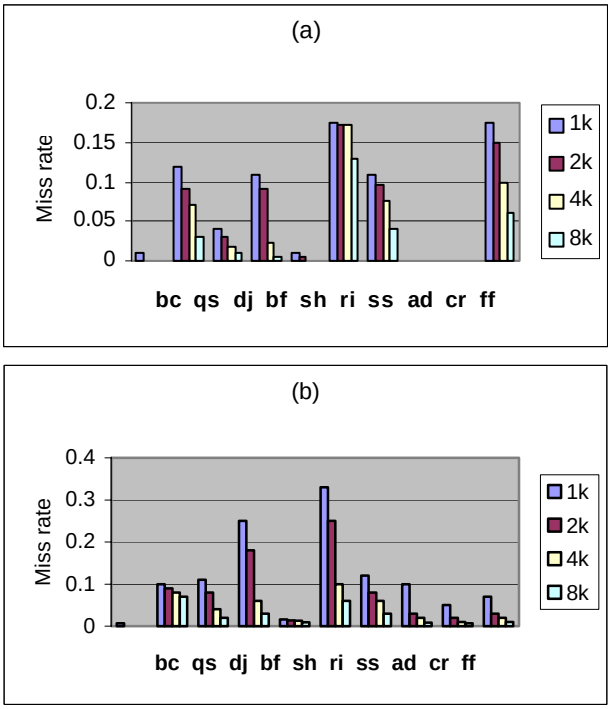

Figure 3: Instruction (a) and data (b) cache miss rates for increasing cache size

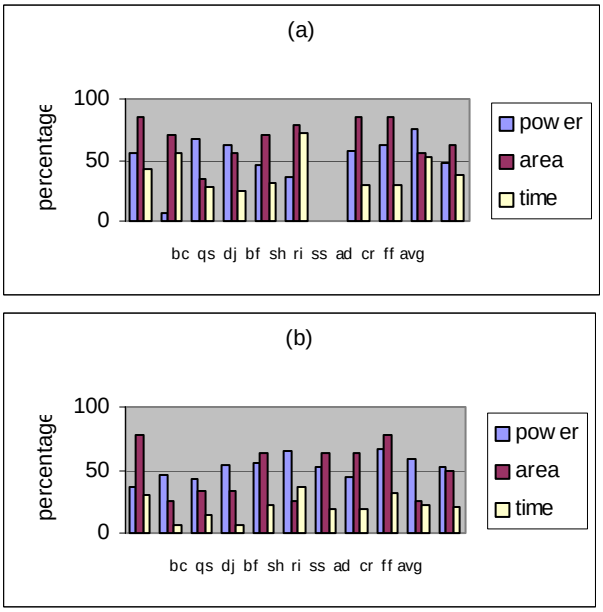

Figure 4: Percentage reduction of power, area and cycle for instruction (a) and data (b) caches

Table 2: Cache configurations yielding lowest power, area and cache access time

Benchmark	Instruction cache	Prefetch buffer	Array cache	Scalar cache
bit counts	256 bytes	256 bytes	512 bytes	512 bytes
qsort	256 bytes	512 bytes	1k	4k
dijkstra	1k	2k	512 bytes	4k
blowfish	1k	1k	512 bytes	4k
sha	256 bytes	512 bytes	512 bytes	1k
rijndael	512 bytes	512 bytes	1k	4k
string search	256 bytes	No prefetching	512 bytes	1k
adpcm	256 bytes	256 bytes	1k	512 bytes
CRC32	256 bytes	256 bytes	512 bytes	512 bytes
FFT	1k	1k	1k	4k

the benchmarks used in our experiments. As can be seen, for instruction cache, on average we achieve 47% reduction in power, 62% in area and 37% in access time.Here it should be mentioned that for benchmark "ss" the best configuration was 8k. Hence we did not achieve any reduction in power or area. For data caches, on average we show more than 50% reduction in both power and area. Each of the benchmarks also provides reduction in cache access time. For each benchmark we generated data and selected the best configuration in optimizing power, area and access times. In Table 2 we provide the optimum configurations for each benchmark.

6 Utilization of additional area

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 other processor activities. 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 and thus these techniques are viewed as inappropriate for embedded systems [5]. Since we show reductions in cache sizes in 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.

6.1 Hardware and software Prefetching

Prefetching or exploiting the overlap of processor computations with data access has proven to be effective in tolerating long memory latencies [2, 10]. Prefetching can be either hardware [10] or software based [2]. 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.

6.2 Hardware optimization techniques with look-up table

Modern processors utilize speculative execution of instruction based on branch prediction, instruction reuse and function reuse technique to improve performance [1, 16]. It has been found that many instructions and functions are repeatedly executed with the same inputs, producing same outputs [1]. Similarly for branch instructions, branch decisions are correlated and can be predicted. This observation can be exploited to reduce the number of instructions/functions executed dynamically as follows: by buffering the previous result of the instruction/function, future dynamic instances of the same static instruction (or function) can use the result by establishing that the input operands in both cases are the same [1]. For all of these optimization techniques as the microprocessor tries to make the prediction based on a record of what this instruction/function has done previously, having a larger look up table is very helpful [19]. Unfortunately none of these optimization techniques have been studied in detail for embedded applications. We anticipate that since we can save the space needed for cache memories using our cache structures (on average 62% for instruction cache and 49% for data cache), the saved space can be used to build needed look-up tables to implement instruction and function reuse.

7 Conclusions

In this paper we introduced a cache architecture for embedded microprocessor platforms. Our design 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. To further improve the proposed cache organization, we augment the instruction cache with a small prefetch buffer. Our cache architecture reduces the instruction and data cache size by as much as 85% (average 62%) and 78% (average 49%), cache access times by as much as 72% (average 37%) and 36% (average 21%), and energy consumption by as much as 75% (average 47%) and 67% (average 52%) respectively when compared with an 8KB L-1 instruction cache and an 8KB L-1 unified data cache with a 32KB level-2 cache (for both data and instructions).

Our design enables the cache to be divided into multiple partitions some of which 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 sub-array partitioning that is already present in modern caches, only minor changes to cache implementations are required. The reconfiguration only requires a small overhead in terms of silicon area, power and execution times.

In future we will explore how unused cache portions (obtained by our proposed method) can be used for instruction reuse, value prediction and branch predictions.

8 References

[1] A. Sodani and G. Sohi, Dynamic Instruction Reuse, in Proceedings of 24th Annual International Symposium on Computer Architecture, pp.194 - 205 June, 1997.

[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, pp. 222-233 Oct. 1996.

[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, pp.136 –146, June. 2003.

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

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

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

[10] 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

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

[12] 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.

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

[14]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, pp. 364-373, May 1990.

[15] 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, pp. 131-140, February 2002.

[16] P. Chen, K. Kavi and R. Akl. "Performance enhancement by eliminating redundant function execution", Proceedings of the IEEE 39th Annual Simulation Conference, Huntslville, AL,pp 143-150, April 2-6, 2006.

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

[18] 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:, pp.677 - 6885 , May 1996.

[19] 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.