Cache Design Trade-offs for Power and Performance Optimization: A Case Study

Ching-Long Su and Alvin M. Despain

Advanced Computer Architecture Laboratory
University of Southern California
{csu, despain} @usc.edu

Abstract

Caches consume a significant amount of energy in modern microprocessors. To design an energy-efficient microprocessor, it is important to optimize cache energy consumption. This paper examines performance and power trade-offs in cache designs and the effectiveness of energy reduction for several novel cache design techniques targeted for low power.

1 Introduction

Portable computing applications have shifted from conventional low performance products such as wristwatches and calculators to high throughput and computation intensive products such as notebook computers and cellular phones. The new portable computing applications require high speed, yet low energy consumption because for such products longer battery life translates to extended use and better marketability. This paper presents a case study of performance and power trade-offs in designing on-chip caches for the microprocessors used in portable computing applications.

Previous cache studies have mainly focused on improving performance. Studies of cache access times and miss rates for various cache parameters (e.g. cache size, block size, and degree of set associativity) of the single level caches can be found in [5,8]. Similar studies focusing on multi-level cache organizations can be found in [6,7].

The study of power consumption in caches is fairly now. Studies of instruction set design and its impact of cache performance and power consumption can be found in [1,3]. Studies of low power cache designs can also be found in [2,11].

This paper consists of five sections. Section 2 briefly describes the cache performance and energy models used in this study. Section 3 presents several experimental cache organizations which are designed for either improving performance or saving energy. Section 4 shows the experimental results of this study. Finally, concluding remarks are offered in Section 5.

2 Analytical Models for On-chip Caches

A conventional cache can be divided into three different components: address decoding path, cell arrays, and I/O path. The address decoding path includes address buses and address decoding logic. The cell arrays include read/write circuitry, tag arrays, and the data arrays. The I/O path includes I/O pads and buses to connect the address and data buses.

The on-chip cache cycle time is calculated based on an analytical model presented in [6,14] (which was based on the access time model of Wada et al in [13]). This time model, based on 0.8 μm CMOS technology, gives both cache cycle time (i.e. the minimum time required between the start of two accesses) and cache access time (i.e. the minimum time between the start and end of a single access) in terms of cache size, block size, and associativity. The features of this time model is that it uses SPICE parameters to predict the delays due to the address decoder, word-line driver, pre-charged bit lines, sense amplifiers, data bus driver, and data output drivers. The average time for an off-chip cache access is calculated by the average off-chip access and transfer times which is rounded to the next higher multiple of on-chip cycle time.

The on-chip cache energy consumption is based on an abstract model which considers only those cache components that dominate overall cache power consumption. In the address decoding path, the capacitance of the decoding logic is usually less than that of the address bus. Energy consumption of the address buses dominate the total energy consumption of the address decoding path. In the cell arrays, the read/write circuitry usually does not consume much power. Most energy consumed in the cell arrays is due to both tag and data arrays. The tag and data arrays in conventional cache designs can be implemented in dynamic or static logic. In a dynamic circuit design, word/bit lines are usually pre-charged before they are accessed. The energy consumed by the pre-charged cache word/bit lines usually dominates the overall energy consumption in the cell arrays. In a static circuit design, there are no pre-charges on the word/bit lines. The energy consumption of the tag and data arrays directly depends on the bit switch activities of the bit lines. In the I/O path, most energy is consumed during bit switches of the I/O pads.

The abstract cache energy model is presented below. The total energy consumption of a cache E_{cache} is

divided into three components: the energy consumption of the address decoding path $E_{decoding\_path}$, the cell arrays E_{cell_array}, and the I/O path E_{I/O_path}. $E_{decoding\_path}$ is estimated by multiplying the average number of bit switches on the address bus *Addr_bus_sb* with some scaling factor α. The E_{cell_array} for a cache design in dynamic logic is estimated by the multiplication of the number of memory cells per word line *Word_line_size* and the number of memory cells per bit line *Bit_line_size* with some scaling factor β; The E_{cell_array} for a cache design in static logic is estimated by the multiplication of the number of memory cells per word line *Word_line_size*, the number of memory cells per bit line *Bit_line_size*, and the bit switching rate of the bit lines *Bit_line_sb* with the scaling factor β. The E_{I/O_path} is estimated by multiplying the average number of bit switches on address pads *Addr_pad_bs* and data pads *Data_pad_bs* with some scaling factor χ. The scaling factors α, β, and χ is set to be 0.001, 2, and 20, based on one VLSI implementation in 0.8μm CMOS technology.

$$E_{cache} = E_{decoding\_path} + E_{cell\_array} + E_{I/O\_path}$$

$$E_{decoding\_path} = \alpha * Addr\_bus\_bs$$

$$E_{cell\_array} = \beta * Word\_line\_size * Bit\_line\_size * Bit\_line\_sb$$
(Static Logic)

$$E_{cell\_array} = \beta * Word\_line\_size * Bit\_line\_size$$
(Dynamic Logic)

$$E_{address\_path} = \chi * (Addr\_pad\_bs + Data\_pad\_bs)$$

Addr_bus_bsr	Number of bit switches on address buses per instruction
Word_line_size	Number of memory cells in a word line
Bit_line_size	Number of memory cells in a bit line
Bit_line_sb	Number of switching bit lines per instruction
Addr_pad_bs	Number of bit switches on address pads per instruction
Data_pad_bs	Number of bit switches on data pads per instruction
α,β,χ	Constants depending on the VLSI implementation.

3 Experimental Cache Organizations

3.1 Conventional Designs

Conventional cache designs include direct-mapped and set associative. A set associative cache usually has a better hit rate than a direct-mapped cache of the same size, although the access time for the set associative cache is usually higher than the direct-mapped cache. The number of bit line switches in the set associative cache is usually more than that in the direct-mapped cache, but the energy consumption of each bit line in a set associative cache is usually less than that in a direct-mapped cache of the same size.

3.2 Cache Designs for Low Power

This paper investigates three different cache design approaches to achieve low power: vertical cache partitioning, horizontal cache partitioning, and Gray code addressing

Vertical Cache Partitioning

The basic idea of vertical cache partitioning is to optimize the capacitance of each cache access by increase on-chip cache hierarchy (e.g. two-level caches). Accessing a smaller cache has a lower power consumption since a smaller cache has a lower load capacitance.

We use block buffering as an example of this approach. A basic structure of a block buffered cache [1] is presented in Figure 1. The block buffer itself is, in effect, another cache which is closer to the processor than conventional on-chip caches. The processor checks if there is a block hit (i.e. the current access data is located at the same block of the latest access data). If it is a hit, the data is directly read from the block buffer and the cache is not operated. The cache is operated only if there is a block miss.

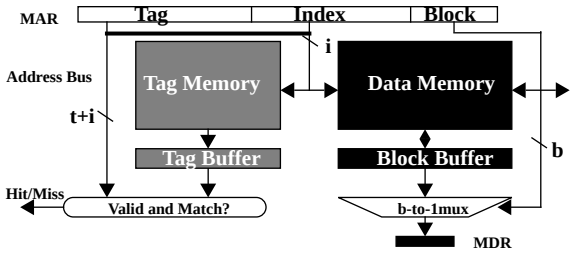

Figure 1: Block Buffering

A block buffered cache saves power by optimizing capacitance of each cache access. The effectiveness of block buffering strongly depends on the spatial locality of applications and the block sizes. The higher the spatial locality of the access patterns (e.g. an instruction sequence), the larger the amount of energy which can be saved by block buffering. The block size is also very important in block buffering. Excluding the impact to the cache hit rate of the cache block size, a small block may result in limiting the amount of energy saved by the block buffered cache and a large block may result in increasing unnecessary energy consumption by the unused data in the block.

Horizontal Cache Partitioning

The basic idea of the horizontal cache partitioning approach is to partition the cache data memory into several segments. Each segment can be powered individually. Cache sub-banking, proposed in [11], is one horizontal cache partition technique which partitions the data array of a cache into several banks (called *cache sub-banks*). Each cache sub-bank can be accessed (powered up) individually. Only the cache sub-bank where the requested data is located consumes power in each cache access. A basic structure for cache sub-banking is presented in Figure 2.

Cache sub-banking saves power by eliminating unnecessary accesses. The amount of power saving depends on the number of cache sub-banks. More cache sub-banks save more power. One advantage of cache sub-banking over block buffering is that the effective cache hit time of a sub-bank cache can be as fast as a conventional performance-driven cache since the sub-bank selection logic is usually very simple and can be easily

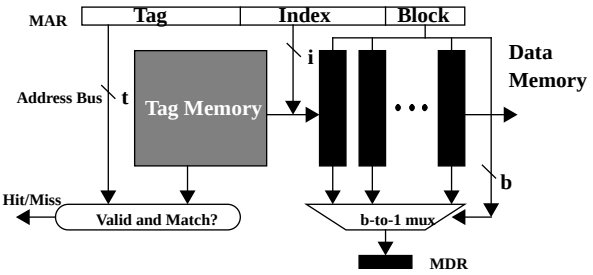

Figure 2: Cache sub-banking

hidden in the cache index decoding logic. With the advantage of maintaining the cache performance, cache sub-banking could be very attractive to computer architects in designing energy-efficient high-performance microprocessors.

Gray Code Addressing

Memory addressing used in a traditional processor design is usually in a 2's complement representation. The bit switching of the address buses when accessing sequential memory space is not optimal. Since there is a significant amount of energy consumed on the address buses and sequential memory address access are often seen in an application with high spatial locality, it is important to optimize bit switching activities of the address buses for low power caches.

Gray code addressing has been proposed to minimize the number of bit switches of the address buses when a consecutive memory space is accessed. A Gray code sequence is a set of numbers in which consecutive numbers have only one bit different [10]. Figure3 shows a comparison of bit switching when 2's complement binary or Gray code is implemented. In the case of a sequence of numbers from 0 to 16, there are 31 bit switches when the numbers are coded in binary representation. There are only 16 bit switches when these numbers are coded in a Gray code representation.

0	00000	00000
1	00001	00001
2	00010	00011
3	00011	00010
4	00100	00110
5	00101	00111
6	00110	00101
7	00111	00100
8	01000	01100
9	01001	01101
10	01010	01111
11	01011	01110
12	01100	01010
13	01101	01011
14	01110	01001
15	01111	01000
16	10000	11000
bits changed	31	16
(a)	2's complement	Gray

Figure 3:The bit switches of 2's Complement codes vs. Gray codes

4 Evaluation Results

This section evaluates performance and power consumption of the experimental cache structures on the VLSI-SLAM microprocessor [9]. First, we select a set of cache organizations for each experimental cache structure. These cache organizations differ by using different cache parameters in which the cache size ranges from 512 to 32 K bytes, the block size ranges from 8 to 32 bytes, and the degree of set associativity ranges from 1 to 4. These cache models are represented as follows,

<dm,2>	a direct-mapped cache with block size 2 words
<dm,4>	a direct-mapped cache with block size 4 words
<dm,8>	a direct-mapped cache with block size 8 words
<2lru,2>	a 2-way set associative cache with block size 2 words
<2lru,4>	a 2-way set associative cache with block size 4 words
<2lru,8>	a 2-way set associative cache with block size 8 words
<4lru,2>	a 4-way set associative cache with block size 2 words
<4lru,4>	a 4-way set associative cache with block size 4 word
<4lru,8>	a 4-way set associative cache with block size 8 words

We assume separated instruction and data on-chip caches. The address and data pins are shared for both instruction and data caches.

4.1 Evaluation Environment

The benchmark programs used in this study are described in Table 1. These benchmark programs are selected from the Aquarius benchmark suite [4]. Applications of these benchmark programs include list manipulation, data base queries, theorem provers, and computer language parsers. The benchmark programs are compiled through the Aquarius Prolog compiler front-end [12]and an optimizing compiler backend for VLSI-SLAM.An instruction-level simulator of VLSI-SLAM is used to monitor bit switching activities of cache behaviors when benchmark programs are running.

Benchmark	Instructions	Description
boyer	27,494,723	An a Boyer-Moore theorem prover.
browse	18,883,712	Build and query a database
nand	1,064,197	A circuit generator
reducer	1,695,417	List manipulation

Table 1: Benchmark programs

4.2 Miss Rates

Figure 4 shows the miss rates of direct-mapped and set-associative instruction and data caches. It is not surprising that the miss rates of both instruction and data caches are decreased when the cache size is increased. A large instruction cache can certainly help the hit rate. However, a large data cache may not help the hit rate that much. For example, the miss rates of <dm,2>, <dm,4>, and <dm,8> are 6.55%, 4.28%, and 3.42%, even though the cache size is 32K bytes.

Increasing block size significantly improves the instruction cache hit rates. A direct-mapped cache with a

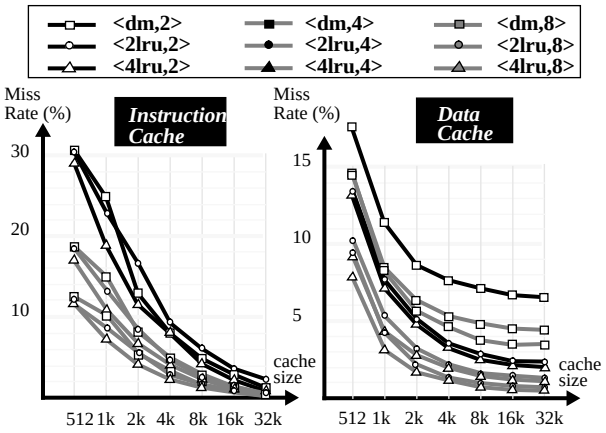

Figure 4: Miss rates

large block size performs better than a set associative cache of the same size. For example, the average hit rate of the 16K-byte instruction cache <dm,8> is 98.92% which is better than hit rates 96.30%, 97.64%, 98.20% and 98.55% of the <2lru,2>,<4lru,2>, <2lru,4>, and <4lru,4> data caches of the same size respectively. The impact to the hit rate of a data cache while increasing the block size is not as obvious as in the instruction cache.

Increasing the degree of set associativity significantly improves the data cache hit rates. A set associative cache with a large degree of set associativity performs better than a direct-mapped cache of the same size (even with a large block size). For example, the average hit rate of the 16K-byte data cache <2lru,2> is 97.53% which is better than hit rates 93.25%, 95.58%, and 96.46% of the <dm,2>, <dm,4>, and <4lru,8> data caches of the same size respectively. The impact to the hit rates of an instruction cache while increasing the degree of associativity is not as obviously as in the data cache.

4.3 Cache Access Time

Table 2 shows the cache access time is derived from the cache timing model presented in [14] based on 0.8 µm CMOS technology. In general, it takes significantly less time to access direct-mapped caches than set associative caches. For example, the cache access times of a 1K-byte direct-mapped and 2-way set associative caches with line size 16 bytes are 4.79 and 7.15 ns respectively. Accessing the 2-way set associative cache is about 50% slower than the direct-mapped cache.

	512	1K	2K	4K	8K	16K	32K
<dm,2>	4.85	5.10	5.47	6.09	6.76	7.55	8.61
<2lru,2>	6.92	7.11	7.37	7.83	8.49	9.35	10.27
<4lru,2>	7.31	7.41	7.64	7.94	8.62	9.28	10.24
<dm,4>	4.65	4.79	5.03	5.45	6.07	6.83	7.51
<2lru,4>	7.06	7.15	7.36	7.66	8.20	8.81	9.73
<4lru,4>	N/A	7.61	7.73	8.00	8.34	9.14	9.82
<dm,8>	4.56	4.60	4.73	5.02	5.40	6.07	6.77
<2lru,8>	N/A	7.46	7.57	7.81	8.16	8.84	9.43
<4lru,8>	N/A	N/A	8.19	8.35	8.66	9.14	9.99

Table 2: Cache Access Time (ns) (in 0.8 µm CMOS)

4.4 Performance and Energy for Traditional Caches

Figure 5 shows the average cache latencies among all experimental cache organizations. We assume each off-chip cache has a 100% hit ratio and the access time is 80 ns. Direct-mapped instruction caches have better (shorter) cache latency than set associative instruction caches. Set associative data caches have better (shorter) cache latency than direct-mapped data caches. Increasing cache line size helps to improve cache latencies.Figure 6 shows the average energy consumption of various cache organizations in dynamic logic.

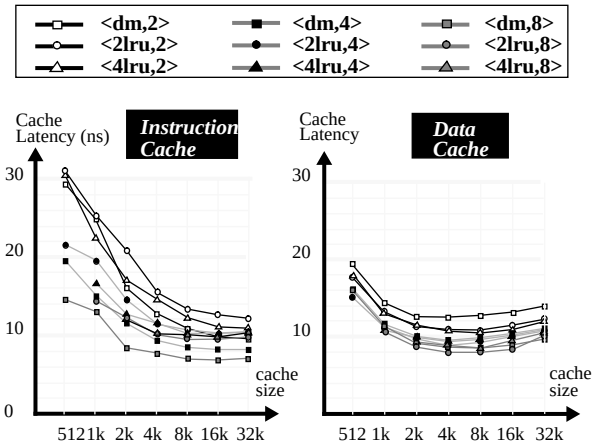

Figure 5: Average cache latency (ns)

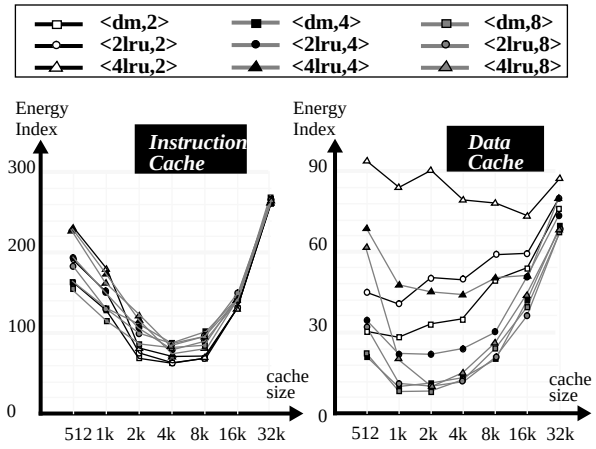

Figure 6: Cache energy consumption (in dynamic logic)

In the case of instruction caches in dynamic logic, direct-mapped caches consume less energy than set-associative caches when the cache line sizes are 8 and 32 bytes. However, direct-mapped caches consume more energy than set-associative caches when the cache line size is 16 bytes. An instruction cache with a 4K-byte cache size seems to consume the least energy among all experimental instruction caches.

In the case of data caches in dynamic logic, direct-mapped caches, in general, consumes less energy than set-associative caches. One exception is that two-way set

associative caches with cache sizes ranging from 4K to 16K bytes and a cache line size of 32 bytes consume less energy than direct-mapped cache with the same cache and line sizes. Similar to the instruction caches, small caches tend to consume less energy than large caches. Data caches with sizes 2K or 4K bytes consume the least energy among all experimental instruction caches. Figure 7 shows the average energy consumption of various cache organizations in static logic.

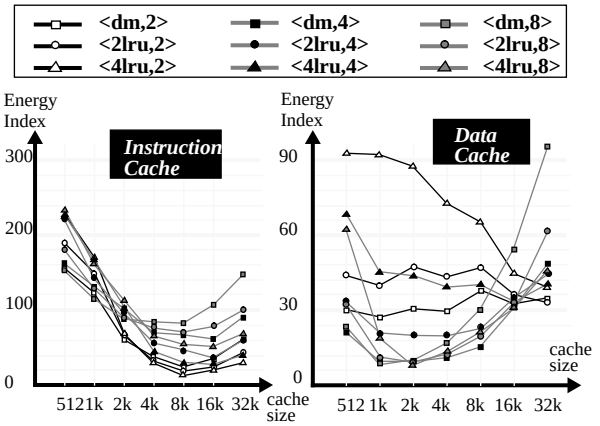

Figure 7: Cache energy consumption (in static logic)

In the case of instruction caches in static logic, set-associative caches consumes less energy than direct-mapped caches. Increasing set associativities can significantly improve energy consumption. However, increasing cache line sizes tends to consume more energy. An instruction cache with cache sizes 8K or 16K seems to consume the least energy among all experimental instruction caches.

In the case of data caches in static logic, set-associative caches consumes less energy than direct-mapped caches. Unlike instruction caches in dynamic logic, increasing set associativity does not improve energy consumption. For example, a two-way set associative cache consumes less energy than a four-way set associative cache with the same cache and line size. In general, increasing cache line sizes tends to consume less energy, although there exists some exceptions. A data cache with cache sizes ranging from 2K to 16K seems to consume the least energy among all experimental instruction caches.

4.5 The Impact of Cache Partitioning

Block hit rates of instruction caches are high when the block size is large. For example, the block hit rates are 42.80%, 64.35%, 74.19%, and 77.89% when the block size is 2,4,8, and 16 words (4 bytes per word). Except in the case of being able to put a loop in the same block, the limit of block hit rates is the average length of consecutive instruction accesses in the benchmark suite. Comparing the instruction caches, the block hit rates of data caches are relatively low. The block hit rates of the data caches are only 3.15%, 19.86%, 33.57%, and 39.20% when the block size is 2,4,8, and 16 words. In general, data cache miss rates are much higher than instruction cache miss rates when the cache size is very small.

Cache sub-banking reduces the energy consumed for each cache access by partitioning a cache into several banks. Each cache bank can be accessed individually. Since the tag memory is still consuming energy for each cache access, the amount of energy savings by a sub-bank cache not only depends on the number of cache banks in a cache, but also the cache size. The larger the cache, the smaller the amount of energy consumed in the tag memory. Like block buffering, cache sub-banking saves more energy when the block size is large. For example, the energy savings of cache sub-banking are 46.10%, 62.89%, 79.82%, and 89.45% in a 32K cache. Unlike block buffering in which its energy savings is limited to access pattern behavior, the energy savings of cache sub-banking is not limited. The larger the block size, the more energy savings.

In conclusion, the block buffering technique is good for instruction caches with reasonably large block size. Especially when the block size is large enough to contain the whole loop, the instruction cache is only accessed once during executing the loop. Cache sub-banking is good for both instruction and data caches. The energy savings by using both techniques can be more than 90% when the block size is more than 16 words.

4.6 The Impact of Gray Code Addressing

This section presents the normalized average bit switching rates of the address and data buses using traditional 2's complement addressing and Gray code addressing. On average, using Gray code address reduces 33% and 12% of the bit switches in the instruction and data address buses respectively. It is not surprising that the Gray code addressing has a significant energy reduction on the instruction cache since the sequential locality of instructions is generally quite high. The more sequential locality, the bigger the energy savings by using Gray code addressing.

4.7 Energy Reduction for Low Power Caches

This section presents a case study of the overall cache energy savings when both Gray code addressing and cache sub-banking approaches are applied. We use the cache energy model discussed in Section 2. to estimate the overall energy consumption of caches.

Table 3 shows the percentages of total energy consumption (including both instruction and data caches) when using Gray code addressing and cache sub-banking. Both caches in static (S column) and dynamic (D column) logic are presented. In general, these two techniques can save significant energy when they are applied on both caches in static and dynamic logic. The amount of energy saving for caches using dynamic logic is slightly higher than that in static logic.The amount of energy saving is increasing when one of the following situation occurs: (1) the degree of superpipelining is increased, (2) the cache line size is increased, and (3) the cache size is increased. For example, when we apply Gray code addressing and cache sub-banking on a 32K-byte four-way set associa-

tive cache with 32-byte cache lines, the overall cache energy consumption is only 23.11% of the total cache energy consumption when the cache without implementing both techniques.

5 Conclusion

The goal of this research is to investigate the performance and energy trade-offs in designing caches. The cache access time is calculated based on an analytical model that 0.8 µm CMOS technology is assumed. The energy consumption is calculated based on an abstract model where weighted bit switching rates of cache components are considered.

The experimental results suggest that direct-mapped instruction caches have better (shorter) cache latency than set associative instruction caches; set associative data caches have better (shorter) cache latency than direct-mapped data caches. The experimental results also suggest that direct-mapped caches (for both instruction and data) consume less energy than set associative caches when the caches are implemented in dynamic logic. On the other hand, set associative caches (for both instruction and data) consume less energy than direct-mapped caches when the caches are implemented in static logic. Caches with sizes ranging from 4K to 16K bytes tend to consume the least energy of all other cache organizations.

This paper also investigated the amount of energy reduction by using several novel techniques: Gray code addressing and cache sub-banking. Simulation results show that 33% and 12% of the bit switches on the instruction and data address buses can be reduced by using Gray code addressing. When both Gray code addressing and cache sub-banking are applied on a 32K-byte four-way set associative cache with 32-byte cache lines, the overall cache energy consumption is only 23.11% of the total cache energy consumption when the cache without applying the techniques.

Acknowledgments

We like to thank the anonymous referees for their constructive comments. We also thank Steve Crago and Kevin Obenland for their reviewing early drafts. The work was supported by ARPA under grant No. J-FBI-91-194.

References

[1] J. Bunda, W.C. Athas, and D. Fussell, “Evaluating Power Implication of CMOS Microprocessor Design Decisions,” In *Proc. of the 1994 International Workshop on Low Power Design*, April 1994.

[2] B. Burgress, et al., “The PowerPCTM603 Microprocessor: A High Performance, Low Power, Superscalar RISC processor,” In *Proc. of IEEE COMPCON*, February 1994.

[3] S.B. Furber, et al.,”AMULET1: A Microppipelined ARM,” In *Proc. of IEEE COMPCON*, February 1994.

[4] R. Haygood, “A Prolog Benchmark Suite for Aquarius,” Technical Report, Computer Science Department, University of California, UCB/CSD 89/509, 1989.

[5] M.D. Hill, “A case for Direct-mapped Caches,” *Computer*, Vol. 21, No. 12, 1988.

[6] N.P. Jouppi, S.J.E Wilton,. “Trade-offs in Two-Level On-Chip Caching,” In *Proc. of the 21st Annual International Symposium on Computer Architecture*, April 1994.

[7] S. Przybylski, M. Horowitz, and J. Hennessy, “Characteristics of Performance-Optimal Multi-level Cache Hierarchies,” In *Proc. of the 16th Annual International Symposium on Computer Architecture*, May 1989.

[8] A.J. Smith, “A Comparative Study of Set Associative Memory Mapping Algorithms and Their Use for Cache and Memory,” In *IEEE Transactions on Software Engineering*, Vol. 4, No. 2, March 1978.

[9] C.L. Su and Alvin M. Despain, “Branch With Masked Squashing In Superpipelined Processors,” in *Proc. of the 21th Annual International Symposium on Computer Architecture*, April, 1994.

[10] C.L. Su, C.Y. Tsui, and A.M. Despain, “Saving Power in the Control Path of Embedded Processors,” *IEEE Design & Test of Computers*, Vol. 11, No. 4, pp. 24-31, Dec. 1994.

[11] C.L. Su and Alvin M. Despain, “Cache Designs for Energy Efficiency,” in *Proc. of the 28th Hawaii International Conference on System Science*, January 1995.

[12] P. Van Roy and A. M. Despain, “High-Performance Logic Programming with the Aquarius Prolog Compiler,” *IEEE Computer*, January 1992.

[13] T. Wada, S. Rajan, S.A. Przybylski, “An Analytical Access Time Model for On-Chip Cache Memories,” *IEEE Journal of Solid-State Circuits*, Vol. 27, No. 8, Aug., 1992.

[14] S. Wilton, N. Jouppi, “An Enhanced Access and Cycle Time Model for On-Chip Caches,” Research Report 93/5, Digital, June, 1994.

	512		1K		2K		4K		8K		16K		32K	
	D	S	D	S	D	S	D	S	D	S	D	S	D	S
<dm,2>	82.18	82.43	81.85	82.52	81.90	84.40	78.44	84.04	74.61	84.09	68.65	76.11	65.43	69.27
<2lru,2>	81.35	81.48	79.52	79.84	78.06	78.84	74.35	76.16	70.19	72.80	67.51	70.58	65.09	66.79
<4lru,2>	81.52	81.42	79.19	78.96	76.83	76.28	73.80	73.13	69.85	69.80	66.69	66.99	64.65	64.93
<dm,4>	82.41	83.10	81.42	83.15	80.04	85.97	71.76	85.27	60.17	82.56	47.48	68.58	40.85	53.67
<2lru,4>	81.58	82.12	79.41	80.84	75.77	79.62	66.04	75.14	54.61	67.32	45.82	58.34	40.20	47.11
<4lru,4>	81.53	81.78	77.76	78.46	72.62	74.27	62.95	65.90	51.57	56.08	43.48	47.17	39.56	42.16
<dm,8>	84.50	85.30	83.66	85.69	82.71	89.48	72.76	89.59	56.14	86.82	36.87	72.42	26.70	52.96
<2lru,8>	83.24	83.98	81.02	83.02	77.32	82.93	65.22	80.49	48.59	73.02	32.26	55.70	24.55	38.49
<4lru,8>	82.76	83.29	78.15	79.75	71.35	75.50	57.49	66.72	42.53	55.58	29.93	41.05	23.11	28.71

Table 3: Percentages of energy consumption (when both Gray code addressing and cache sub-banking are applied) against the total energy consumption when both techniques are not applied.