Power Efficient Data Cache Designs

Jaume Abella*, Antonio González*+

* *Computer Architecture Department*
Universitat Politècnica de Catalunya
Barcelona (Spain)
jabella@ac.upc.es

+ *Intel Barcelona Research Center*
Intel Labs, Universitat Politècnica de Catalunya
Barcelona (Spain)
antonio@ac.upc.es

Abstract

This paper investigates some power efficient data cache designs that try to significantly reduce the cache energy consumption, both static and dynamic, with a minimal impact in performance. The basic idea is to combine different threshold voltages with different cache organizations that provide different levels of performance. Multi-banked organizations in combination with different approaches to allocate data to cache banks are explored. Some of the resulting cache architectures are shown to provide a good tradeoff between power and performance.

1. Introduction

Power dissipation has become an important issue in processor design. Dynamic power dissipation due to signal transitions is the main power dissipation source nowadays, but static power will become increasingly significant in upcoming processors. While dynamic power is directly related to the activity of the circuits, static power depends on the amount of powered-on transistors and their physical characteristics. Thus, large circuits are usually the main source of static power. Caches are normally the largest structures in the processor, so they are the most important sources of the static power dissipation. It is also known that increasing cache associativity and/or size to reduce the miss ratio and increase performance has an impact on static power and access time. On the other hand, a low access time is desired for performance. Power and performance also depend on the number of ports.

Due to all these factors, techniques to reduce cache power dissipation have to be applied carefully, because a reduction of the power requirements at the expense of losing too much performance can result in an increase of total energy consumption.

Aggressive techniques to reduce dynamic and static power in caches have been proposed in the past. They can be classified basically in two blocks: high-level and low-level techniques. We consider high-level techniques those that try to reduce power dissipation without changing the underlying technology, whereas low-level techniques are those that are based on using technology with different physical properties.

High-level techniques. Zhou et. al. [1] and Kaxiras et. al. [2] have proposed recently different techniques to reduce leakage power by powering off cache lines whose content is not expected to be reused. Ghose and Kamble [3] studied the effects of using subbanking, multiple line buffers and bit-line segmentation to reduce dynamic power dissipation in superscalar processor caches. Su and Despain [4] investigated vertical cache partitioning, horizontal cache partitioning and Gray code addressing to reduce dynamic power. Hezavei et. al. [5] studied the effectiveness of different low power SRAM circuit design strategies like divided bit line, pulsed word line and isolated bit line.

Low-level techniques. Kuroda et. al. [6] proposed a variable supply voltage scheme for low power high-speed CMOS digital design and explored the low supply voltage, low threshold voltage design space. R. Gonzalez et. al. [7] investigated the effect of lowering the supply and threshold voltages on the energy efficiency of CMOS circuits. Itoh et. al. [8] studied the effect of reducing supply voltage and increasing threshold voltage in order to reduce both dynamic and static power dissipation in caches.

This paper studies different cache organizations that reduce significantly dynamic and static power dissipation with a small performance loss. This study tries to guide processor designers to choose the cache organization with best trade-off between efficiency and power dissipation.

The rest of the paper is organized as follows. Section 2 introduces the model used to choose supply and threshold voltages. Section 3 details the definition of criticality that guides some of the evaluated cache systems. Some experimental cache organizations are presented in section 4 and their results are shown in Section 5. Finally, section 6 summarizes the main conclusions of this work.

2. Energy and delay models in CMOS circuits

CMOS power dissipation is given by [9][10]

$$P = P_{dyn} + P_{leak} \quad (1)$$

where dynamic power (P_{dyn}) and static power (P_{leak}) can be expressed as

$$P_{dyn} = p_t \cdot C_L \cdot V_{DD}^2 \cdot f_{CLK} \quad (2) \qquad P_{leak} = I_0 \cdot 10^{-V_{TH}/S} \cdot V_{DD} \quad (3)$$

respectively, where p_t is the switching probability, C_L is the load capacitance (wiring and device capacitance), V_{DD} is the supply voltage and f_{CLK} is the clock frequency. I_0 is a function of the reverse saturation current, the diode voltage and the temperature. V_{TH} is the threshold voltage. Finally, S

corresponds to the subthreshold slope and is typically about 100mV/decade. Using equation (3) can be observed that static power dissipation decreases by 10 times if V_{TH} increases 0.1V.

CMOS propagation delay can be approximated by the following simple α power model[1] [9]

$$Delay = k \cdot \frac{C_L \cdot V_{DD}}{(V_{DD} - V_{TH})^{\alpha}} \quad (4)$$

where k is a proportionality constant specific to a given technology. The α power reflects the fact that the transistors may be velocity saturated. α is compressed in the range [1.. 2], where $\alpha = 1$ implies complete velocity saturation and $\alpha = 2$ implies no velocity saturation. For the 0.18 µm technology assumed in this paper, α is typically 1.3.

From equations (2) and (3) can be deduced that decreasing V_{DD} reduces both dynamic and static power dissipation and slightly increasing V_{TH} reduces drastically leakage, but both parameters adjustments increase the propagation delay as equation (4) shows. Thus, there is a trade-off between reducing power dissipation and increasing delay propagation with minimum performance loss.

3. Criticality

In modern superscalar processors, where multiple instructions can be processed in parallel, deciding when a given resource should be assigned to an instruction is a well-known problem. For instance, when two ready instructions require the same functional unit to be executed, only one of them can be chosen. Different policies are used to take these decisions in existing processors, but usually they do not take into account the impact on performance of delaying any instruction. Some studies [11][12][13] have proposed techniques to heuristically obtain this information and use it to increase performance. Load instructions are especially harmful if they have high latencies and are in the critical path [14]. Thus, the criticality of load instructions is important information to handle them efficiently.

An exact computation of the criticality of each load instruction is not feasible due to its complexity. Thus, an approximation to the criticality is proposed. Then we propose an accurate predictor of criticality according to our definition. For the proposed criticality-based cache organization, we will only need to classify loads into two categories: critical and non-critical, so we need a mechanism to decide when a load can be delayed one or more cycles and when delaying it will significantly degrade performance.

3.1. Criticality estimation

In order to decide whether an instruction is critical or not, we consider if the data produced by the instruction (if any) is immediately used by at least another critical instruction. With this criterion only those instructions belonging to a chain of dependent instructions that are executed as soon as possible, are considered critical. For those instructions that do not produce data, like stores and branches, there is no information so another additional criterion is required.

If the number of cycles elapsed since an instruction has finished its execution until it commits is greater than a given threshold N, then the instruction is considered non-critical. Intuitively, this criteria indicates that the instruction belongs to a chain of dependent instructions which is not the longest one or that there is an instruction that stops the commit process (for instance a load that misses L1 cache), and thus, this chain may take some more cycles without performance degradation. In our experiments, after evaluating different values for N, we have observed that N=4 cycles gives the best results for the chosen cache organizations.

The criticality predictor has been implemented as a 2048 untagged entry table where each entry is a 2-bit saturated counter whose most significant bit is the prediction. Initially the table indicates that all the instructions are critical. The table is updated by every instruction that commits. If the committing instruction has been waiting for commit less than N cycles (N=4 in our experiments), or its produced data (if any) is forwarded to another critical instruction through a bypass and the depending instruction is issued immediately, the corresponding 2-bit counter is incremented, otherwise it is decremented.

The evaluation section describes how this criticality predictor has been validated.

4. Cache organizations

This section describes different cache organizations that are compared to a baseline L1 monolithic 1-cycle latency cache. Our proposals are based on two L1 cache modules implemented with different technologies. One of them is a 1-cycle latency cache implemented with the same technology than the baseline. It will be referred to as *Fast Cache* in the rest of the paper. The second one is a 2-cycle latency cache implemented with a technology with lower V_{DD} and higher V_{TH} than the baseline technology, in order to reduce both dynamic and static power dissipation at the expense of increasing the access time. It will be referred to as *Slow Cache* in the rest of the paper.

According to the formulas described in section 2 we are interested in decreasing V_{DD} and increase V_{TH} as much as possible with the following limitations: these parameters should be technologically feasible and the latency should be at most 2 times larger than the latency of the baseline cache.

Static power dissipation can be analytically estimated, but dynamic power depends on the program, thus optimal generic values for V_{DD} and V_{TH} cannot be computed. In order to guide the selection of these values, Figure 1 shows different valid combinations of these values that can be chosen and the expected dynamic and static power dissipation compared to the baseline technology. The assumed parameters for the baseline technology are V_{DD}=2.0V and V_{TH}=0.55V [15]. The rest of the parameters are described in section 2.

[1] The subthreshold current is considered to be a constant and it is assumed that transistors are in the current saturation mode.

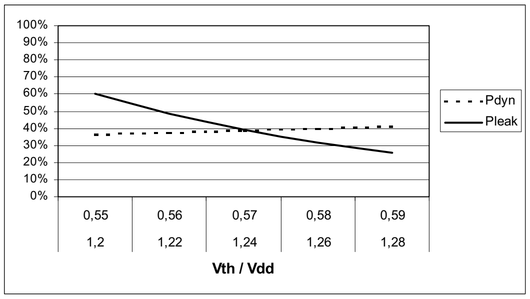

Figure 1. Power dissipation compared to the baseline technology for different V_{TH} and V_{DD} values

We have chosen V_{TH}=0.57V and V_{DD}=1.24V technology for the Slow Cache because it reduces both power dissipation sources to the same percentage.

4.1. Proposed cache organizations

Two different cache organizations are proposed. The first one is a hierarchical locality-based cache system where the fast cache is the first level data cache, the slow cache is the second level cache and the baseline's second level cache is the third level cache. In this organization the slow cache should be larger than the fast cache to be useful.

The second one is a criticality-based cache system where there is not the inclusion property (some data contained in fast cache may not be in slow cache and vice versa) as in the first proposal. Both the fast and the slow caches are accessed always in parallel. If a critical load hits in the slow cache and misses in the fast cache, the cache line is copied from the slow to the fast cache. If a critical load misses both caches, then the data fetched from the following cache level is allocated only in the fast cache. If a non-critical load hits at least in one of both caches, the data is not copied from one cache to the other. If a non-critical load misses in both caches, then the data fetched from the following cache level is allocated only in the slow cache. Finally, if a store hits at least in one cache there is no data copy, but if it misses, assuming that the used policy is write-allocate, the data is fetched to the fast or the slow cache depending on the criticality of the store instruction.

Another important consideration is the cache sizes used. Most of the existing processors have data caches which size is in the range [16K.. 64K]. Table 1 describes the different cache sizes used to compare the different alternatives. All caches described have 32-bytes cache lines and 2 read/write ports. The baseline and fast caches are 2-way associative. Due to space limitations, it is only shown the evaluation for the 16K baseline cache. For 32K and 64K similar conclusions have been obtained.

Table 1. Cache sizes used in the comparison

Baseline	Hierarchic system / Criticality-based (3-way slow)		Hierarchic system / Criticality-based (2-way slow)	
L1	**Fast**	**Slow**	**Fast**	**Slow**
16K	4K	12K	4K	8K

It can be seen in Table 1 that there are two versions for both proposals. In the first one the total size is the same than the baseline (slow cache is 3-way associative) and in the second one the total size is smaller than the baseline but the slow cache has the same associativy than the fast one (slow cache is 2-way associative). The cache sizes for both proposals are exactly the same, so their performance and power dissipation are comparable.

5. Performance evaluation

This section evaluates the accuracy of the criticality predictor and the performance and power dissipation of the different cache organizations in a superscalar processor.

5.1. Experimental framework

Our power dissipation and performance results are derived from Wattch [17], which is an architecture-level power and performance simulator based on SimpleScalar [16]. Table 2 shows the processor parameters.

Table 2. Processor configuration

Fetch, Decode, Issue, Commit width: *4 instructions/cycle*
Issue queue size: *40 entries* Reorder Buffer size: *64 entries*
IntALU's: *3 (1 cycle)* IntMult/Div: *1 (3 cycles pipelined mult, 20 cycles non-pipelined div)* FP ALU's: *2 (2 cycles pipelined)* FP Mult/Div: *1 (4 cycles pipelined mult, 12 cycles non-pipelined div)* Memory Ports: *2*
Branch Predictor: *Hybrid: 2K entry Gshare, 2K entry bimodal and 1K entry metatable* BTB: *2048 entries, 4-way*
L1 Icache size: *64K 2-way, 32-byte lines, 1 cycle latency* L1 Dcache size: *2-way, 32-byte lines* L2 Unified cache: *512Kb, 4-way, 64-byte lines, 10 cycles latency* Memory: *50 cycles, 2 cycles interchunk* TLB size: *128 entries, 30 cycles miss penalty*

5.2. Benchmarks

The whole SPEC2000 benchmark suite has been evaluated. These benchmarks have been compiled with the Compaq/Alpha compiler using –O4 –non_shared flags. For every benchmark we have simulated 100M instructions after skipping the initialization part, using the ref input set. The results shown correspond to the SpecINT2000, SpecFP2000 and Spec2000 averages (harmonic mean for IPC).

5.3. Criticality evaluation

For all the configurations, SpecINT2000 has near twice percentage of critical loads (60% approx.) than SpecFP2000 (30% approx.). The main reason for this difference is that integer applications have less ILP than FP ones, so they have proportionally more loads belonging to the critical path.

After studying what percentage of loads is considered critical, the next step consists in verifying that those loads are really critical. In order to verify that the criticality criterion detects the critical loads, we will compare the execution of the criticality-based 2-way slow cache organization versus the baseline in two ways:

- The loads considered as critical or non-critical are treated as critical or non-critical respectively.
- The same percentage of loads that were considered as critical ones in the previous simulation will be considered critical, but this time they will be chosen randomly.

As Figure 2 shows, the criticality scheme achieves significantly higher performance than the random scheme across all cache sizes. It can be seen that when loads are chosen as critical according to the criticality criterion the IPC loss is much less than in the randomly chosen scheme so, it can be concluded that the criticality criterion gives a good classification of loads that can be used to guide the criticality-based cache organization.

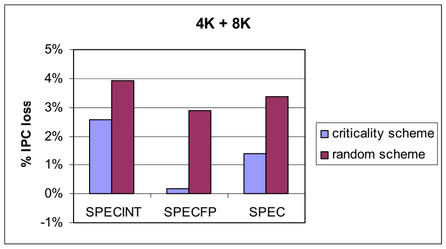

Figure 2. IPC loss of criticality-based cache for the guided and the random versions w.r.t. 16K baseline

5.4. Cache organizations comparison

The comparison between the locality-based and the criticality-based cache organizations versus the baseline has been done based on different metrics: performance (IPC), miss ratio, dynamic power dissipation and static power dissipation.

5.4.1. Performance

Figure 3 shows the IPC loss for both cache organizations versus the baseline. 2way and 3way stand for 2-way and 3-way associative slow cache respectively. The SpecINT2000, SpecFP2000 and SPEC2000 percentages have been computed using the harmonic means of the IPC's.

It can be observed that the locality-based scheme works better than the criticality-based scheme for the SpecINT2000 but the criticality-based scheme achieves better results than the locality-based for the SpecFP2000. We have observed that the loads can be classified as critical or non-critical, but it is common that the data fetched by a non-critical load is reused by a critical one and vice versa. Due to this, if there is no capacity limitation in the fast cache is better to fetch all data to the fast cache than fetching some data to the slow cache if it has to be fetched later to the fast cache by a critical load. In general, integer applications have small working sets so, the locality-based scheme that always fetches the data to the fast cache works better than the criticality-based scheme. But for floating point applications with huge working sets, this performance loss due to delaying some critical loads that find their data in the slow cache instead the fast cache, is compensated by retaining in the fast cache during more cycles data that will be reused by critical loads, instead of replacing it with data that only will be used by non-critical loads during that period of time.

It can be seen that for FP programs the criticality-based scheme may achieve better results than the baseline due to the beneficial effect of not placing data fetched by non-critical loads in the fast cache.

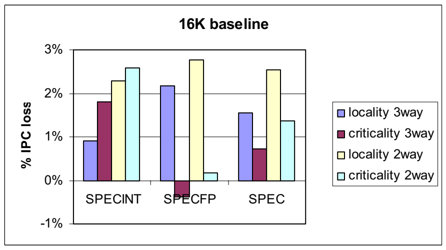

Figure 3. Performance loss of locality-based and criticality-based organizations w.r.t. 16K baseline

5.4.2. Miss ratios

Figure 4 shows the miss ratios for critical and non-critical loads. This figure classifies loads into L1 hits and misses for the baseline. For the other organizations the loads are classified into three groups: those that hit in Fast cache, those that miss in Fast cache but hit in Slow cache, and those that miss in both L1 caches. Note that the scale for all the figures begins at 50% for the sake of showing better the hit/miss distribution because the fast cache hit ratio is always higher than 50%.

Hit fast, hit slow and miss stand for hit in the fast cache, miss in the fast cache but hit in the slow cache, and miss in both L1 caches respectively.

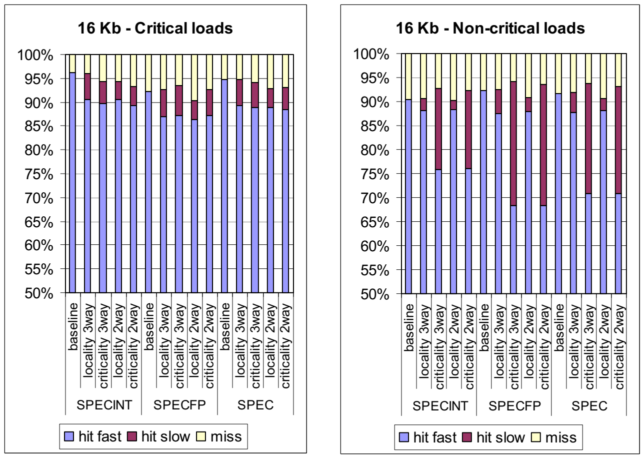

Figure 4. Miss ratios for 16K baseline cache

Figure above shows that in general for the SpecFP2000 the fast cache hit ratio of the critical loads in the criticality-based schemes is slightly higher than the same ratio in the locality-based schemes. For the SpecINT2000 higher hit ratios in the fast cache are achieved in the locality-based schemes because the working sets are small and critical loads reuse data fetched by non-critical loads to the fast cache.

The hit ratio for the critical loads in the slow cache is higher for both criticality-based schemes. Non-critical loads fetch data to the slow cache and critical loads later reuse this data. For the integer applications this means that in the locality-based schemes some critical loads find their data in the fast cache whereas in the criticality-based schemes they find their data in the slow cache, increasing their latency. For the FP programs this means that some critical loads that in the locality-based schemes do not find their data in L1 caches (fast and slow), in the criticality-based schemes find their data in the slow cache, decreasing their latency. For the floating point applications, with huge working sets, the data fetched by critical loads stays during more time in the fast cache because it is not evicted by data fetched by non-critical loads, but also the criticality-based schemes do not have the inclusion property in fast and slow cache so more different data can be stored in both caches.

For non-critical loads it can be observed that criticality-based schemes achieve lower miss ratios in the L1 caches and higher hit ratios in the slow cache because the classifying mechanism places data fetched by non-critical loads in the slow cache and this data is not evicted by data fetched by critical loads.

After analyzing the miss ratios it can be understood why the criticality-based scheme does not improve significantly the locality-based one in both kinds of benchmarks. The loads can be considered as critical or non-critical, but a critical load can use the data fetched by a non-critical one or vice versa (the same data or other data contained in the same cache line).

5.4.3. Dynamic power dissipation

After understanding the reasons that produce those performance differences between the proposed cache organizations, we will compare the power requirements of each organization in order to decide which one achieves the best tradeoff between power and performance. Figure 5 shows the percentages of dynamic power dissipation for every scenario. All percentages have been computed with respect to the baseline cache organization. The power dissipation of the locality-based scheme and the criticality-based scheme is broken down into different power dissipation sources: fast cache, slow cache, L2 cache power increase and, only for the criticality-based scheme, the additional structures to decide when a load is or not critical (table to decide when a load is critical or not and counters to know if an instruction that has finished its execution has been in the reorder buffer during more than 4 cycles).

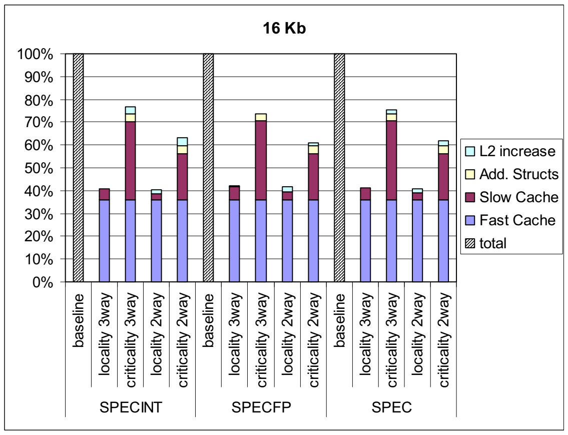

Figure 5. Dynamic power dissipation for a 16K baseline cache

As shown above, criticality-based schemes save near 25% data cache power for the 3-way slow cache configuration and near 40% for the 2-way slow cache configuration, whereas the locality-based organizations save near 60% dynamic power versus the baseline. The L2 power increase becomes negligible for larger caches because the miss ratios of the criticality-based and locality-based schemes are very similar to the baseline scheme miss ratio.

Slow cache power requirements are higher for the criticality-based schemes than for the locality-based schemes because fast and slow caches are accessed in parallel in the criticality-based schemes. In order to reduce these power requirements we did some experiments with different cache access policies like:

- Accessing only one cache and access the other just in case of miss.
- Access both in parallel for a critical load and only the slow cache for non-critical ones.
- Access both in parallel for a critical load and first the slow followed by the fast cache in case of miss for a non-critical load.

This kind of schemes showed to be especially harmful for performance because, as shown before, some critical loads miss in fast cache and hit in slow cache, so accessing sequentially to the L1 caches increases their latency. Additionally, even if load instructions access sequentially both caches, it is not saved as much power as in the locality-based schemes because in the criticality-based schemes the store instructions should access both caches in order to maintain cache coherence. In all cases the results did not show drastic power reduction but the performance loss was significant, so we decided to choose the policy with best performance even if it does not save as much power as the other organizations.

Finally, as figure shows, the additional structures power dissipation of the criticality-based schemes is quite small.

5.4.4. Static power dissipation

Figure 6 shows the static power dissipation for all scenarios with respect to the baseline. It can be observed that for the locality-based and the criticality-based schemes the fast cache and slow cache static power requirements are the same for every slow cache configuration (2-way or 3-way associative). As shown in the figure, the static power dissipated by the proposed cache organizations is substantially smaller than the static power dissipated by the baseline architecture because these organizations use a technology with less static power requirements for the slow cache and have less capacity for the 2-way associative slow cache organizations.

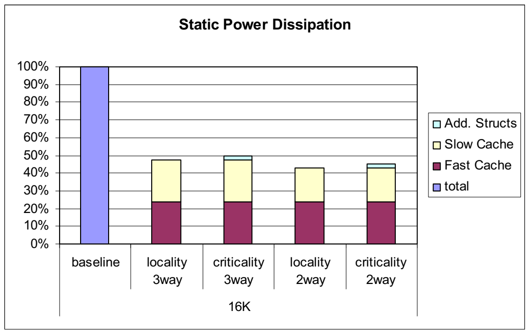

Figure 6. Static power requirements

6. Conclusions

This study shows how different L1 multi-banked data cache organizations can obtain similar performance to that where a monolithic cache is used, requiring at the same time less dynamic and static power and enabling to reduce the cache access time. It has been shown that different technology parameters can be combined to obtain high performance caches with small power requirements.

Another important conclusion is that the improvement in performance that a criticality-based scheme can obtain with respect to a locality-based scheme in some cases, does not justify the additional complexity to detect which instructions are critical and which not, and the additional power requirements. The criticality detection applied in a cache system cannot improve substantially performance because we use it to classify data, whereas the criticality is an instruction property, not a data property. Storing some data in a slow cache because a non-critical load fetched it can degrade performance if a critical load requires this data later.

Finally, we can conclude that a locality-based cache organization that combines different supply and threshold voltages can achieve high performance with small power requirements (both dynamic and static), small cache access time and reduced complexity.

Acknowledgements

This work has been supported by CICYT project TIC2001-0995-C02-01, the Ministry of Education, Culture and Sports of Spain, and Intel Corporation. We would like to thank the anonymous reviewers by their comments.

References

[1] H. Zhou, M. Toburen, E. Rotenberg, T. Conte, "Adaptative Mode Control: A Static-Power-Efficient Cache Design" in PACT'01, Barcelona, Spain, September 2001.

[2] S. Kaxiras, Z. Hu, M. Martonosi, "Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power" in ISCA'01, Göteborg, Sweden, June 2001.

[3] K. Ghose, M.B. Kamble, "Reducing Power in Superscalar Processor Caches Using Subbanking, Multiple Line Buffers and Bit-line Segmentation" in ISLPED'99, San Diego, California, August 1999.

[4] C.L. Su, A.M. Despain, "Cache Design Trade-offs for Power and Performance Optimization: A Case Study" in ISLPED'95, Dana Pt., California, April 1995.

[5] J. Hezavei, N. Vijaykrishnan, M.J. Irwin, "A Comparative Study of Power Efficient SRAM Designs" in GLSVLSI'00, Evanston, Illinois, March 2000.

[6] T. Kuroda et. al., "Variable Supply-Voltage Scheme for Low-Power High-Speed CMOS Digital Design" in IEEE Journal of Solid-State Circuits (JSSC) vol. 33 no. 3, March 1998.

[7] R. Gonzalez, B.M. Gordon, M.A. Horowitz, "Supply and Threshold Voltage Scaling for Low Power CMOS" in IEEE Journal of Solid-State Circuits (JSSC) vol. 32 no. 8, August 1997.

[8] K. Itoh, K. Sasaki, Y. Nakagome, "Trends in Low-Power RAM Circuit Technologies" in Proc. of the IEEE vol. 83 no. 4, April 1995.

[9] T. Sakurai, A.R. Newton, "Alpha-Power Law MOSFET Model and its Applications to CMOS Inverter Delay and Other Formulas" in IEEE Journal of Solid-State Circuits (JSSC) vol. 25 no. 2, April 1990.

[10] T. Sakurai, H. Kawaguchi, T. Kuroda, "Low-Power CMOS Design through V_{TH} Control and Low-Swing Circuits" in ISLPED'97, Monterey, California, August 1997.

[11] S.T. Srinivasan, R.D. Ju, A.R. Lebeck, C. Wilkerson, "Locality vs. Criticality" in ISCA'01, Göteborg, Sweden, June 2001.

[12] B. Fields, S. Rubin, R. Bodík, "Focusing Processor Policies via Critical-Path Prediction" in ISCA'01, Göteborg, Sweden, June 2001.

[13] R. Rakvic, B. Black, D. Limaye, J.P. Shen, "Non-vital Loads" in HPCA'02, Cambridge, Massachusetts, February 2002.

[14] M. Schlansker, V. Kathail, "Critical Path Reduction for Scalar Programs" in MICRO'95, Ann Arbor, Michigan, November 1995.

[15] G. Cai, C.H. Lim, "Architectural Level Power/Performance Optimization and Dynamic Power Estimation" in Proceedings of Cool Chips Tutorial, in conj. with MICRO'99, Haifa, Israel, November 1999.

[16] D. Burger, T. Austin, "The SimpleScalar Tool Set, Version 2.0" Technical report #1324, Computer Sciences Department, University of Wisconsin-Madison, June 1997.

[17] D. Brooks, V. Tiwari, M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations" in ISCA'00, Vancouver, Canada, June 2000.

[18] M.J. Charney, T.R. Puzak, "Prefetching and Memory System Behaviour of the SPEC95 Benchmark Suite" in IBM Journal of Research & Development vol. 41 no. 3 – Performance Analysis and its Impact on Design, 1997.