Ayodele Embry
Ben Serebrin
John Maly
Melvyn Lim

EE 482 Project
May 31, 2000

SCD2SE: A Proposed Method To Achieve Low Energy and High Performance

I. ABSTRACT

The proliferation of handheld/portable devices in the consumer electronics market has made energy consumption an increasingly important consideration in processor design. The goal in microchip design for handheld applications is to decrease energy consumption as much as possible without any significant loss in performance. We propose selective deactivation as a method to achieve a low energy, high performance design. Selective deactivation involves deactivating portions of the processor that are not contributing significantly to the performance of a given application. In this study, we analyze the tradeoffs between energy consumption and performance when deactivating various parts of the processor.

II. INTRODUCTION

As feature sizes shrink, computer architects are introducing more functionality and complexity to today's microprocessors. Most current processors have more than 500 million transistors, and some have over a billion transistors. Many of these high performance systems largely ignore power constraints. However, as chip size and functionality grow, so do power and energy consumption. This causes a problem for portable, battery-operated devices. Today, consumers may be willing to accept slightly less performance in return for longer battery life. However, as handheld and other portable devices become more prevalent, consumers will expect both higher performance and longer battery life.

There has been considerable work on low power processors in recent years. Many processors are scaling voltage, frequency, and feature sizes in order to lower energy requirements. Several architectures have proposed using software to maximize energy efficiency [Sinha]. Sinha and Chandrakasan proposed using wireless communication to make decisions about processing ability based on user input and sustainable battery life. Other architectures try to reduce mis-speculation on branches [Manne]. As cache size increases due to decreasing feature sizes and larger die sizes, the power consumed to access the cache becomes increasingly substantial. Some architectures solve this problem by banking the cache and only accessing cache bank that is being used [Albonesi].

With the increased use of battery-operated devices, standby current has become an important constraint in chip design. Most portable devices spend the majority of their time in standby mode, during which the system clock is inactive. In this mode, current is drawn by the device due to the static leakage current of the gates in the circuit. Although the standby current is several orders of magnitude lower than the active current during normal operation, it typically dominates battery life due to the large fraction of time the device spends in standby mode. When supply voltage is reduced (as is commonly done to reduce active power consumption), the device threshold voltage (V_t) is also reduced to maintain proper scaling for performance. The reduction of V_t causes the sub-threshold leakage current of the device to increase exponentially, thus making the leakage current of devices a major concern in next generation portable chip design [Blaauw].

Many types of applications do not use all the available processing power on a chip efficiently. For instance, some integer applications will not utilize the floating point hardware on a chip, which can occupy up to 20% of the chip area, thus wasting both dynamic power due to clock switching and static power due to leakage. Other applications, like databases, cannot take advantage of branch prediction and power is wasted fetching and executing from incorrect paths [Manne]. Still other applications have a small working set and do not need to utilize the entire cache.

If the programmer can identify the behavior of these applications or if the hardware or compiler can perform profiling to analyze repeated usage, unused units could be deactivated to save considerable power and energy without greatly impacting performance.

III. PROPOSED SOLUTION

One way to combat growing energy budgets for portable systems is to introduce an architecture that uses application, compiler, and runtime hints to deactivate inefficiently utilized hardware by gating clock and power supply inputs. Special instructions can be used to instruct the processor to deactivate certain hardware modules before the application begins to run. At the completion of the application, the processor will be returned to its completely active state. This will allow any applications that do not provide directives to run with full processor functionality.

Deactivation directives could be supplied in three ways. First, the programmer could include directives in the application code to tell the hardware which units may be inefficient and can be deactivated. Second, the compiler can be designed to keep track of instructions and deactivate units that are used infrequently. For example, if there are very few or no floating-point instructions, the FPU could be deactivated and those instructions could be emulated in software. Third, if a unit operates below a certain level of occupancy or accuracy (in the case of a predictor) at runtime, that unit can be deactivated. For example, if dynamic branch prediction accuracy is below a certain threshold, the branch predictor could be deactivated on the fly and static branch prediction could be used instead. Dynamic deactivation is not recommended for the cache since data may be spread among several banks and transferring all of the state to a needed space would be difficult.

While deactivating portions of the hardware can save energy, it is also very important that application performance not be greatly impacted. Only unused or unneeded hardware should be disabled. We initially chose to model four different types of hardware deactivation – the L2 cache, the ALU, the FPU, and the branch prediction unit.

L2 cache

Shrinking the cache size when possible is an obvious strategy for saving power. The trend in state of the art microprocessors today is to use growing transistor budgets to increase fast, on-chip cache memory [Kamble]. For the MIPS R10000, 30% of the chip area is devoted to cache memory and that number is increasing on newly released chips. Published reports also corroborate the fact that on-chip static RAM caches consume substantial fraction of overall chip power ranging from 25% for the DEC 21164 CPU to 43% for the StrongARM SA-110 from Digital [Kamble]. Therefore, reducing cache energy consumption can significantly reduce the energy consumption for the entire chip.

Banking caches has relieved some of the power stress on large caches, since only the bank being accessed adds to the line capacitance. However, as feature sizes shrink and die sizes increase, the impact of leakage current on cache power and energy dissipation also increases. At very low voltages, the power dissipation due to subthreshold leakage can become comparable in magnitude to the switching power dissipation of the circuit. In order to decrease the impact of leakage current, it must be possible to completely shut off unused or unneeded portions of the cache to save energy. Assuming unused cache components can be adequately deactivated (such that they will consume significantly less power than when activated), the savings could be vital.

ALU/FPU

Some chip designs use extra area to implement additional functional units such as ALUs and FPUs. This allows increased performance on computationally intensive applications that contain sufficient instruction-level parallelism. Deactivating additional ALUs may save power in applications that have considerable amounts of instruction dependence and are not able to take advantage of additional processing units.

Many present day portable computing applications do not consist of many floating-point operations. Most of the associated arithmetic is handled by the ALU for such tasks as graphical user interface updates. However, evolving microarchitectures and increased expectations by users for improved graphics may drive FPUs onto chips for portable devices. Power savings could be substantial if applications that do not need FPU functionality could be run with the unit deactivated. Any floating-point instructions that are encountered would be emulated with integer instructions, much like the StrongARM chip. The StrongARM has no FPU; instead, a slightly enhanced multiplier-adder unit is implemented in the ALU.

Branch Prediction Unit

Some applications, like unpredictable database-type applications, do not benefit from branch prediction. The energy usage of such applications can actually be increased by branch prediction. The branch predictor can be deactivated when running applications that have very low prediction accuracy. This will prevent additional energy being spent executing repeatedly down incorrect paths.

The motivation for this decision arose out of Montanaro's "A 160-MHz, 32-b, .5-W CMOS RISC Microprocessor", which avoided the use of a branch prediction unit to save power; branches incurred a one-cycle penalty for branches taken. A dedicated adder in the instructional unit was used to calculate branch target addresses, and this had to run every cycle (regardless of whether the instruction was a branch or not). But the tradeoff can still be worthwhile since the FPU occupies about 20% of the logic area of the chip.

IV. METHODOLOGY

A. Simulations

In our experiment, we looked at the effect of deactivating the hardware listed in Figure 1. We used the SimpleScalar simulator [Austin] configured approximately to the specifications of the MIPS R10000 processor [Vasseghi]. Daniel Citron provided the necessary configuration and code files [Citron]. Figure 2 details the processor features.

We ran several applications from the Spec95 benchmark suite to determine performance and energy consumption with selective hardware deactivation. Due to time and information limitations, we were only able to run four integer benchmarks: *go*, *compress*, *gcc*, and *m88ksim*. We skipped the first 500 million instructions to prime the cache, then ran the applications for 100 million instructions.

Hardware	**Action**	**Sizes**
L2 Cache	Shrink size of cache by deactivating unneeded banks	0KB, 64KB, 256KB, 1024KB
FPU	Deactivate FPU and emulate in software	
Branch prediction	Deactivate branch prediction unit and perform no speculation	Predict not taken with no speculation, Bimodal w/ 1024 entries
ALU	Remove ALU	1, 2, 4

Figure 1: Hardware deactivation options

Frequency	200 MHz
Voltage	3.3V
Process	0.35 μm
L1 Cache – I & D	32KB, 2-way
L2 Cache – U	512KB, 2-way
ALU	2

Figure 2: MIPS R10000 specifications

B. Estimated Energy Dissipation

Although many power studies solely use power as the metric, we use energy as our metric. Power can be misleading, especially when dealing with battery-powered devices. Even if the power consumption of a device is cut in half, the device will drain the same amount of stored battery energy if it takes twice as long to run the same application. Therefore, one must consider the requirements of the application: is the processor expected to continuously run but may be allowed to have reduced performance, or must it complete a finite-length task within a required time?

Energy dissipation data was estimated using information about the configuration of the various units and runtime data from the 4 chosen Spec95 benchmarks. The energy dissipation was measured in a slightly different manner for each of the units and described in the following section.

Cache

For cache energy consumption, we ran the applications with no L2 cache and L2 cache sizes of 64KB, 256KB, and 1024KB. We assumed 4 integer ALUs and bimodal branch prediction unit for all four cache size configurations. Using the Kamble and Ghose paper on analytical models for caches, we calculated the energy dissipated through bitlines and wordlines. We did not consider output line dissipation, although it constitutes a considerable percentage of cache power, due to time constraints and the lack of complete capacitive modeling information for cache output lines. The Kamble and Ghose model calculates the dissipated energy using information on cache organization and size and on dynamic behavior through misses and hits. We assumed minimum sized transistors for the core memory elements and realistically sized pass transistors. Therefore, our results are very conservative in the amount of power that the caches use. We model a combination of L1 and L2 cache energy in our results.

ALU

In order to determine ALU energy consumption, we ran the benchmarks with bimodal branch prediction and the maximum size cache of 1024KB. Energy was approximated by assuming that each individual ALU has an equal area and transistor count. Since the 2 ALUs in the R10000 occupy about 14% of the logic area, we assumed that each individual ALU would occupy about 7% of the logic area. Energy was determined using a percentage of transistor count equal to the percentage of logic area to calculate capacitance with minimum sized transistors.

For these energy calculations, interconnect power dissipation (which is, admittedly, a considerable amount) was ignored due to a lack of easy modeling.

Branch Prediction Unit

In order to evaluate branch prediction, we ran our benchmarks once using a not taken scheme and again using bimodal prediction. We could not determine explicit energy usage since we were unable to find information about the area footprint of the branch prediction hardware. We assume that the storage and logic for branch prediction is considerably smaller than other structures in the chip as justification for disregarding power usage due to the branch predictor itself. However, we examined the number of additional instructions that were executed because of incorrect branch prediction. Because branch prediction takes place every cycle, energy is dissipated every cycle when accessing the branch history table. In addition, energy is wasted due to taking the incorrect path. Instructions that are executed speculatively but will not ever commit take up space and expend energy.

FPU

Energy dissipated by the FPU was estimated by assuming that the number of logic transistors in the FPU is proportional to the percentage of logic area occupied by the FPU. The FPU takes up approximately 20% of the logic area of the chip. By shutting off the FPU completely when it is not efficiently used, a considerable amount of energy can be saved. This problem is similar to that of excessive large caches in that leakage power will become more important as time passes. However, on the performance side, IPC is decreased due to the additional cycles needed to emulate floating point operations.

V. RESULTS

We used an evaluation matrix to consider both energy savings and performance. Performance is measured as instructions executed per cycle and energy is measured as energy dissipated per instruction. Instructions per Cycle tells us how well the processor is performing while Energy per Instruction tells us how low the energy costs were. Both must be balanced in order to have an efficient, effective architecture. This allows us to measure tradeoffs between power and performance to determine whether or not these changes can be worthwhile.

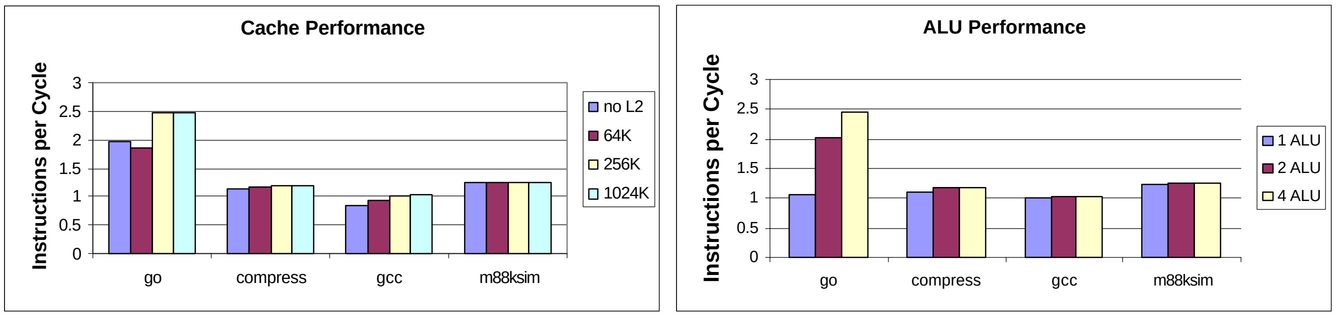


Figure 3: Performance measured in instructions per cycle

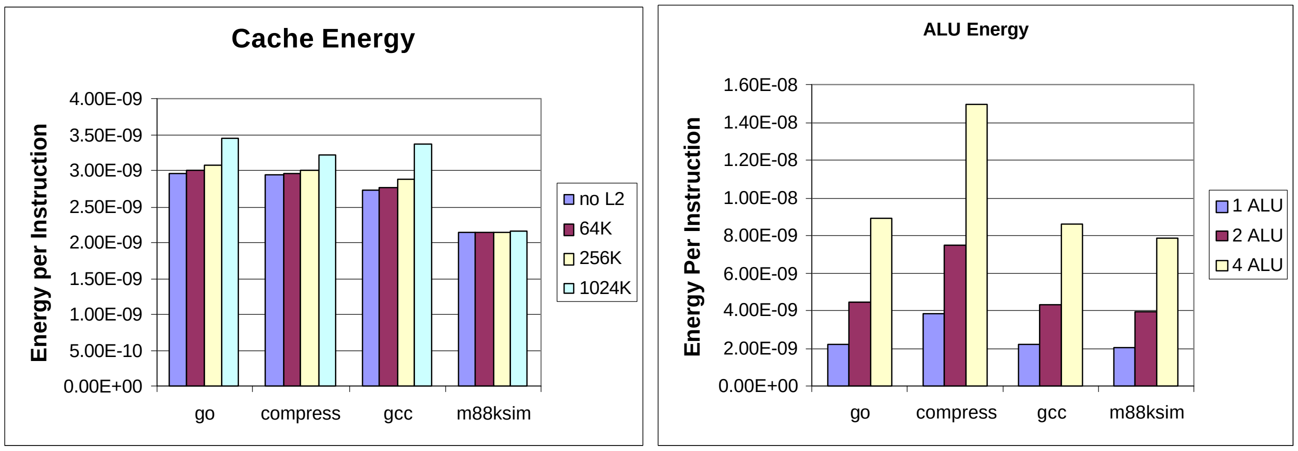


Figure 4: Energy dissipated per instruction. For the cache this calculation includes both the L1 and L2 cache energy due to wordline and bitline switching capacitance.

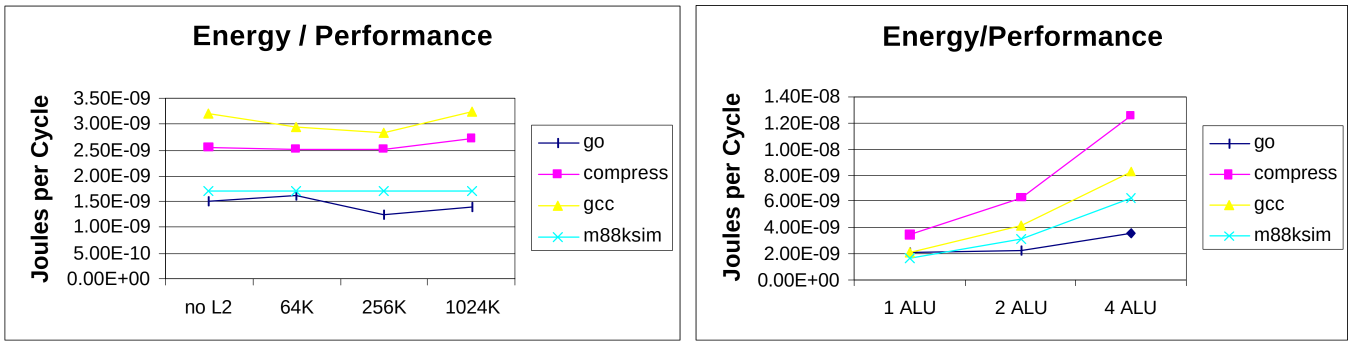


Figure 5: Power vs. Performance.

L2 Cache Results

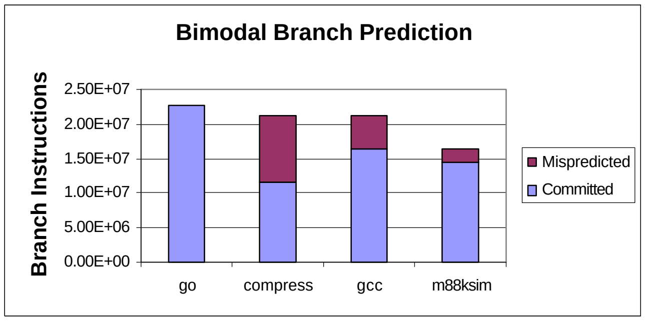
Cache energy per instruction is graphed for the combination of L1 (fixed size) and L2 (variable size) cache energy consumption due to bitline and wordline switching. The dynamic cache energy consumption for *m88ksim* does not change as the size of the L2 cache decreases. This is due to its extremely small working set, which fits completely in the L1 cache, and the large number of L1 accesses. L2 cache accesses are so infrequent that dynamic power does not suffer due to bitline and wordline loading. Likewise, the performance of *m88ksim* is also constant. However, we did not evaluate static leakage power dissipation that takes place in the L2 cache, so static power might dominate dynamic power for the cache due to the small number of accesses.

Gcc, *compress*, and *go* behave similarly in energy consumption. They show a small increase in dynamic power for cache sizes from 64KB to 256KB and then a jump in dynamic power due to wordline and bitline loading with a cache size of 1024KB. *Compress* and *go* show a negligible increase in performance with larger cache size. However, *go* shows a considerable increase in IPC with the 256KB cache size and no additional gain with 1024KB.

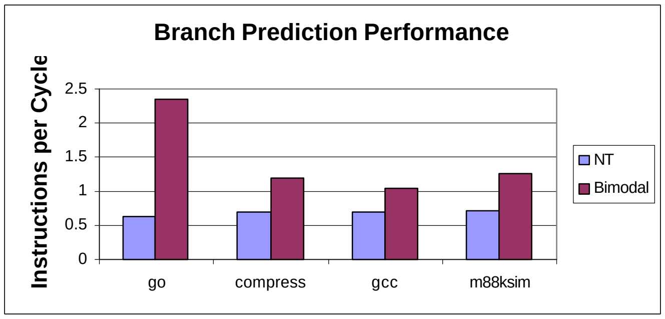
When we examine the results for our figure of merit, energy per cycle, for the applications, we see a small knee/saddle point for *gcc*, *go, and compress* at 256KB. This implies that these applications could benefit from cache deactivation, which will decrease power dissipation yet cause only a negligible impact on performance.

ALU Results

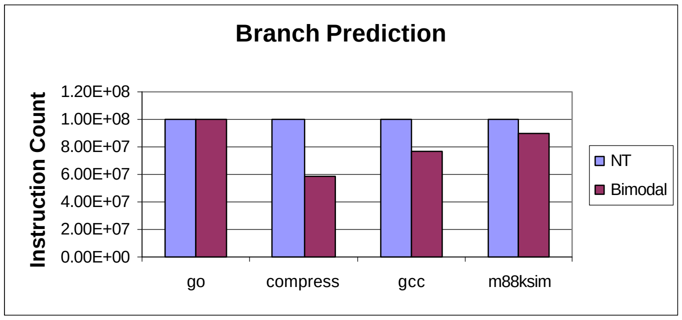
Compress, *gcc*, and *m88ksim* have very similar performance when the number of ALUs is scaled. Their performance does not increase when the number of ALUs is increased. Only *go* shows a performance benefit from increasing the number of ALUs beyond 1. Further, the energy per instruction increases considerably as the number of ALUs is increased. When we look at the relative energy per performance, we see that moving to 2 ALUs causes an increase in energy for all the applications except *m88ksim* and moving to 4 ALUs causes a substantial increase in energy dissipated. Thus, 2 ALUs seems to be the best compromise between performance and energy consumption, which is also the reason that most processors use no more than 2 ALUs. However, where low energy consumption is more essential than performance, applications could still benefit from allowing ALU deactivation to move from 2 ALUs to 1 ALU.



(a)



(b)



(c)

Figure 6: Branch prediction performance.
NT is a static not taken predictor.

Branch Prediction Results

The not taken predictor always guesses not taken and only begins to execute instructions that it knows will complete. On the other hand, the bimodal predictor will sometimes guess incorrectly and begin to fetch and execute down the wrong path. As evident from Figure 6a, *compress* and *gcc* have a considerable percentage of mispredicted branches. For *compress* the percentage is almost 45%. *Go*, however, has almost perfect branch prediction. After running tests with bimodal prediction, we noticed that 100 million instructions were not actually committed, but issued. *M88ksim* committed only 90 million, *compress* committed only 57 million, and *gcc* committed only 70 million instructions. However, since a branch occurs once every 6 instructions on average, the mispredicted instruction count accounts for the deviation – these instructions were issued/executed but not committed since they were belonged to the wrongly predicted path. Even with the large misprediction rate, performance improvements are substantial when moving from static to dynamic branch prediction. Even with a 45% misprediction rate, compress still almost doubles its performance. This means that special care must be taken in deciding to shut off branch prediction units to ensure that adequate performance is obtained while minimizing mispredictions and energy use. It may be better to improve the accuracy of the predictor than to completely deactivate it.

We were not able to calculate energy dissipation of the branch prediction unit due to the lack of information about chip power dissipation for instructions that begin execution but are later squashed.

Floating Point Results

For the FPU, performance is not actually measured with the unit disabled due to limitations in the simulation software that we used. Two of the applications, *go* and *m88ksim*, have no floating point instructions while *gcc* has a very small number of floating point instructions. Because of the considerable area occupied by the FPU, these applications would significantly enhance energy per performance if the FPU were deactivated.

VI. INSIGHTS

- It is difficult to obtain physical models for power, which rely on process parameters. The best way to get power information is to build the chip and measure it. However, at the same time, power constraints must be known before the chip is built. There is a major need for analytical models.

- Many assumptions had to be made to perform the power calculations. We tried to err on the conservative side, but we realized that without interconnect measurements, we could not get a true picture of the power consumption.

- Simply knowing die area is not enough to do efficient power calculations. The number of transistors dedicated to each unit and its functional and electrical behavior must also be known.

- It is easier to do studies using older technology because more information and architectural details have been published.

- It is also important to understand how the simulator measures values. For example, we had expected the "maximum instruction" (*–max:inst num*) command line option to specify the limit on the number of instructions committed, instead of the number of instructions issued/executed (as it turned out).

- Although leakage current is not currently a major factor in power dissipation, it will become more important in coming years as feature sizes continue to shrink and voltages are scaled.

VII. CONCLUSION

Using different combinations of cache size, number of ALUs, and branch prediction, we have shown that selective deactivation can help reduce the energy consumed by a processor per unit performance. This is especially pertinent to handheld/portable applications, which are rapidly growing in usage and popularity. Certain hardware units are more suitable for selective deactivation than others. Deactivating portions of the cache and the FPU produced improvements in energy per performance, while scaling down the number of ALUs brought a decrease in energy per performance and turning off branch prediction impacted performance significantly. Given this preliminary analysis, we believe that selective deactivation is a promising approach in low power, high performance design. Further work must be done with more detailed and accurate power models to explore this approach.

BIBLIOGRAPHY

[Kamble] Milind B. Kamble and Kanad Ghose. Analytical Energy Dissipation Models for Low Power Caches. ACM, 1997.

[Sinha] Amit Sinha and Anantha P. Chandrakasan. Energy Aware Software. *13th International Conference on VLSI Design*, January 2000.

[Manne] Srilatha Manne, Artur Klauser, and Dick Grunwald. Pipeline Gating: Speculation Control for Energy Reduction. *ISCA '98* , 1998

[Albonesi] David Albonesi. An Architectural and Circuit-Level Approach to Improving the Energy Efficiency of Microprocessor Memory Structures. 1999.

[Montanaro] James Montanaro et all. A 160-MHz, 32-b, 0.5-W CMOS RISC Microprocessor. *IEEE Journal of Solid State Circuits*, 1996.

[Blaauw] David Blaauw. Power Management Issues in High Performance Processor Design.
Proceedings of the IEEE Alessandro Volta Memorial Workshop on Low-Power Design, 1998.

[Vasseghi] Vasseghi et al, "200MHz Superscalar RISC Microprocessor Processor", *IEEE Journal of Solid-State Circuits*, 1996.

[Austin] Austin et al, “SimpleScalar: Simulation Tools for Microprocessor and System Evaluation”, www.simplescalar.org.

[Citron] Daniel Citron, C.S. Department, Hebrew University of Jerusalem, citron@cs.huji.ac.il. Personal communication.