Impact of Technology Scaling on Energy Aware Execution Cache-based Microarchitectures

Emil Talpes
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA, 15213
Phone: (412) 268-4275
etalpes@ece.cmu.edu

Diana Marculescu
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA, 15213
Phone: (412) 268-1167
dianam@ece.cmu.edu

ABSTRACT
Reducing total power consumption in high performance microprocessors can be achieved by limiting the amount of logic involved in decoding, scheduling and executing each instruction. One of the solutions to this problem involves the use of a microarchitecture based on an Execution Cache (EC) whose role is to cache already done work for later reuse.

In this paper, we explore the design space for such a microarchitecture, looking at how the cache size, associativity and replacement algorithm affect the overall performance and power efficiency. We also look at the scalability of this solution across next process generations, evaluating the energy efficiency of such caching mechanisms in the presence of increasing leakage power. Over a spectrum of SPEC2000 benchmarks, an average of 35% energy reduction is achieved for technologies ranging from 130nm to 90nm and 65nm, at the expense of a negligible performance hit.

Categories and Subject Descriptors
C.1.1 [**Computer Systems Organization**]: Processor Architectures – single data streams architectures.

General Terms
Performance, Design, Measurement.

1. INTRODUCTION
Superscalar processor designers have traditionally given priority to performance concerns over energy costs, power efficiency being addressed mainly at the technology level, through lower supply voltages, smaller transistors, etc. Nevertheless though, power dissipation is now the primary design constraints for modern processors, and thus microarchitecture designers must take it into consideration as well.

In this paper, we study the concept of caching previously executed work for further reuse with the purpose of reducing power consumption of superscalar, out-of-order processors. We focus on the design space of such a microarchitecture, and study how its behavior scales for deep submicron process technologies. Specifically, we are focusing on the *design space* for such a microarchitecture. Our contribution is threefold:

- **Design space exploration**. We study how the caching capacity impacts these designs, and how the trace replacement algorithm affect performance and power efficiency.
- **Eliminating or reducing performance bottlenecks**. We propose an adaptive scheme for the register file to deal with potential capacity limitations. Furthermore, we show that in several cases performance penalty can be completely eliminated by simply throttling the trace generation process.
- **Impact of technology scaling on the power efficiency.** While the concept of caching for power efficiency can be very successful in reducing dynamic power, it does not affect static power. As process technology evolves and transistors get smaller, leakage becomes a significant part of the total power budget. We evaluate how the optimum design point shifts in this process, focusing on three process technologies: 130nm, 90nm and 65nm.

ISLPED'04, August 9–11, 2004, Newport Beach, California, USA.

2. RELATED WORK
Gating input transitions for logic blocks that are not needed during a specific computation has been proposed before [1] as a method for controlling dynamic power. While this can be easily done using clock or input gating [1], it is generally hard to predict when such blocks will not be in use. Thus, special techniques have been proposed that essentially rely on caching partial computation results that can potentially be reused for better power efficiency.

Historically, on-chip caches have been used as a mechanism to reduce the average latency observed when accessing the main memory. Units like filter cache [3] or L-cache [4] function on essentially the same idea, servicing accesses locally as often as possible.

An interesting microarchitectural innovation implemented by Intel in the Pentium 4 microprocessor is the use of a special cache that shortens the branch misprediction path [5][6]. By storing decoded instructions (uops) in the trace-cache, the whole decode stage can be shut down for significant periods of time while the rest of the pipeline continues to work. In [7], this microarchitecture is studied focusing on several techniques for filtering out infrequently used traces.

Bringing the idea of caching for power efficiency one step further, an **Execution Cache (EC)** has been proposed [8] for storing instructions after they pass through the entire front-end of the pipeline. As this cache is placed after the Issue Stage, instructions that are fetched, decoded and have already had registers renamed can be stored in *issue-order* (and not in *program-order*) in the EC. Most of the time, instructions are fetched from the EC and fed directly to the execution engine.

3. MICROARCHITECTURE OVERVIEW
To achieve better power efficiency, the microarchitecture must be tuned to perform as little work as possible for each committed instruction. Using an EC placed after the Issue Stage, work performed by the front-end stages of the pipeline can be cached for later reuse. Instructions are fetched from the I-Cache, decoded, and physical registers are assigned to each logical register. The resulting instructions are placed in the Issue Window, waiting for their dependencies to be resolved. A number of independent instructions are issued to the functional units and, in parallel, added to a *Fill Buffer* in order to create program traces. When enough instructions are placed in this Fill Buffer, the entire program sequence is stored in the EC in **issue order**, for later potential reuse [8].

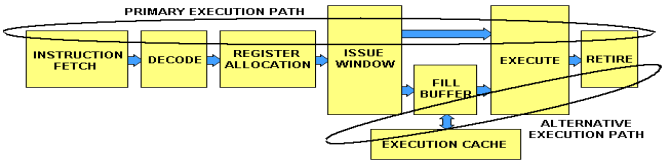

Figure 1. Superscalar microarchitecture using an EC for reusing scheduled instruction streams

In this microarchitecture, program execution can be split into two conceptual phases. Initially, when EC is empty, instructions are launched from the Issue Window, while a trace is built in parallel in the EC. This

step is called the ***trace build*** phase. Upon a mispredict (or a trace completion condition), a search is performed to identify a possible next trace starting at that point, and should a hit occur, instructions continue to be executed from the EC. In this ***trace replay*** phase, the processor behaves essentially like a VLIW core with instructions being fetched from the EC and sent directly to the functional units (Figure 1).

3.1. EC architecture overview

When stored in issue order, instructions lose their original, logical order, so they can only be retrieved on a sequential basis. To allow for traces to be reused, each trace is labeled with the address of its first instruction. Instructions from two consecutive traces cannot be interleaved. To minimize the overall performance impact associated with very frequent trace changes, these sequences must be long, limited as much as possible only by events that naturally occur in the processor pipeline (i.e., branch misprediction).

The EC structure consists of a ***Tag Array*** (TA) and a corresponding ***Data Array*** (DA). The TA is an associative cache, addressed using the translated program counter. When looking for a new trace, the TA is searched first and, if a tag is found, it is further used to access the DA. The DA can be either direct mapped or set associative, and is composed of multiple memory banks in order to reduce the energy per each access. While the address of the first access is dictated by information retrieved from TA, all subsequent accesses are made to sequential set addresses. By knowing beforehand which set will be accessed next, a new look-up for each DA read or write is avoided.

3.2. An adaptive register file

Since instructions come out of the Execution Cache without preserving their original, program order, registers cannot be renamed for such trace replays. For this microarchitecture, the Register Renaming operation is only performed in *trace build* mode. At the same time, operand values cannot be simply stored in the EC and reused, as they are generally different during each trace run. To handle this task, a special register file structure can be used [8] (Figure 2(a)). This structure employs a special pool of physical registers for renaming every logical register of the ISA. Unlike in a typical register file, where an architected register can be renamed to any physical register, here an architected register can be renamed to only physical registers of the corresponding pool.

Each trace generation is started under the assumption that the correct value for a register can be obtained from the first logical entry of the pool. If such a condition is met, all subsequent replays of a trace can be performed without further renaming the registers. The caveat is that this requires a *checkpoint* to be performed when a trace execution ends: the logical ordering must be changed inside the circular buffer, so each time it starts with the latest value for that architected register.

In this setup, the number of in-flight instructions that have the same logical destination is bounded by the number of physical registers available in the circular buffer. This limitation reduces the renaming capacity when compared against other register renaming schemes. However, we note that some registers are written very often (e.g., those holding local variables), while others are mostly read (e.g., Stack Pointer, Return Address, etc.). Following these guidelines, we can assign a different number of renaming registers for each architected register, thus alleviating the renaming capacity problem while also preventing the Register File structure from becoming too large.

We propose a simple, dynamic approach to solving this problem. A simple 16-bit counter (Stall Counter in Figure 2) is associated with every architected register, to keep track of capacity limitations. When a stall is introduced (due to the lack of available rename registers), the associated counter is incremented. During each specified interval, these counters are checked and register redistribution can be performed. Specifically, architected registers that have been detected to be bottlenecks are supplemented with additional physical entries, taken from other pools that are only infrequently accessed.

As the redistribution of the physical registers requires significant amount of time, this operations cannot be performed often. In our experiments, we assumed that the counters are tested every 1 million cycles, and, if a redistribution is needed, this operation will require an additional 100 cycles. Our results show that only a small fraction of the total architected registers (typically 10 to 15%) need more than four physical entries. Furthermore, the best configuration is mostly dictated by the conventions used for register allocation, so the number of redistribution is fairly small and steady state can be rapidly reached.

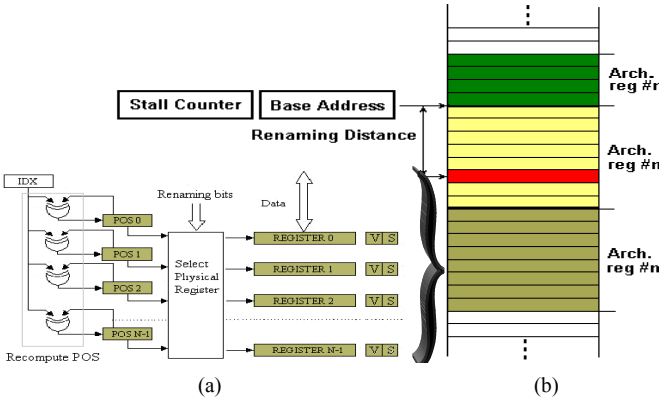

Figure 2. Physical register file organization: (a) Pool-based register file [8] and (b) the adaptive Register File

4. DESIGN TRADEOFFS

Most design decisions that need to be made impact both performance and power consumption. Furthermore, such decisions are affected by the increasing leakage power, and this effect is getting worse as the process technology scales down.

4.1. EC size

The most obvious parameter in this microarchitecture is the EC size. Since power savings can only be achieved when the processor uses the alternative execution path and turn off the front-end, we would ideally want to achieve very high hit rates for the EC.

However, for trace-based storage this is much harder to achieve than it is for normal instruction caches. In an I-Cache, any instruction can be potentially accessed directly if needed. In the EC, only the first instruction in the trace can be a branch destination; all the rest can be accessed sequentially, if the execution follows the same path as during trace build. Thus, multiple copies of the same instruction can very easily reside in the cache, bringing the required size up.

A larger EC requires significantly more power, but this problem can be controlled with sub-banking. While the address of the first access is dictated by information retrieved from TA and cannot be predicted, all subsequent accesses are made to sequential set addresses. By knowing beforehand which set will be accessed next, only the wordlines in the specified block need to be precharged, saving power.

While dynamic power can be controlled using sub-banking, as we reach deep submicron process technologies the leakage power starts to become a problem as well. Unless this problem is controlled, it can offset much of the benefits of the large EC, as we show in Section 6.

4.2. Trace build and replacement

When creating a new trace, instructions are added until an end condition is met. Such end conditions can occur when the trace grows beyond the maximum length that can be accommodated in the EC, when we encounter hard to predict instructions (like function returns or indirect branches) or when a branch mispredict occurs and the execution must resume from a different address. When fetching instructions from the EC, the execution is abandoned on trace-end or on a branch mispredict.

If predictions made while creating a trace prove to be wrong during its re-execution, the trace must be declared invalid and another one created. The most straightforward strategy is to observe the number of branch mispredicts generated by the trace. Thus, a trace is declared as invalid (and can be replaced) when *M* mispredicts are encountered in a

row while executing the same trace. As *M* increases, more branch mispredictions are tolerated and the usage of the pipeline's front-end decreases, leading to a better power efficiency.

In our experimental results, we study how this parameter affects the performance and energy by testing two alternative configurations. First, we assume that traces are aggressively erased when **two** branch mispredictions are encountered in a row when executing the same trace [8]. The second case that we consider waits until the misprediction is encountered **four** times in a row, and only then it evicts the trace.

4.3. Back-end throttling during trace-build

One potential drawback of the EC-based microarchitecture is its inability to schedule instructions around a variable latency operation. When scheduling is fixed during trace build mode, the ability of the processor to adapt to such conditions is lost.

However, this locked issue order can actually be turned into an advantage. If the Issue Window occupancy is sufficiently high, more independent instructions can be found and sent to execution. In our microarchitecture, in parallel with sending the instructions to execution they are captured and stored in the EC. If *more* instructions can be sent in parallel, a *more parallel* trace schedule will be captured and the replay will be characterized by higher performance. Thus, it might make sense to *slow down* the trace-build phase waiting to fill the Issue Window, since the performance penalty associated with it will be paid for just a fraction of the total execution time.

The implementation of this method is very simple. During trace build, the back-end execution core is throttled (half the speed in our setting). To avoid possible performance penalty, this policy has been adopted only if the processor spends less than 20% of the execution time in trace build phase. As the adaptive Register File requires certain tests to be performed at specified intervals (1 million cycles), we used the same checkpoints to check the front-end cycle counter and decide whether the throttling policy should be applied or not.

4.4. Future process technologies

Like all caching strategies, our method only reduces the active power, which is poised to become a smaller fraction of the total power budget. Following the 2003 ITRS guidelines [9], the leakage power will reach 40-50% at 90nm and even higher at a 65nm process technology.

As it only reduces active power, the EC mechanism becomes less efficient in such deep submicron process technologies. Furthermore, the microarchitecture is based on a fairly large cache structure and it also requires significantly more physical registers, so it can actually increase the leakage power. As it will be seen in Section 6, if not controlled, the increased leakage power can render any EC-based microarchitecture uninteresting for achieving energy efficiency.

One possible solution to the leakage problem has been presented recently [2], and is based on gating the ground off when the memory cell is not in use. As it increases the cycle time only very slightly, this method can be attractive for controlling leakage power. In our experimental setup, we evaluate both the behavior of the microarchitecture in the presence of uncontrolled leakage current, as well as the effect of using gated ground to control static power.

5. EXPERIMENTAL SETUP

For our experiments we have used a modified version of the SimpleScalar microarchitectural simulator *sim-outorder* engine in order to support the adaptive Register File and EC models, a longer pipeline and an operating mode based on inter-stage buffers instead of the SimpleScalar architecture based on a Register Update Unit (RUU). For the baseline microarchitecture, the Register Renaming mechanism chosen is similar to the one used by the MIPS R10000 processor [10].

The power models were based or similar to the ones used in Wattch [12]. For evaluating the leakage power we used the models proposed by Butts and Sohi [11]. The normalized leakage current per device was estimated using the ITRS 2003 guidelines [9].

The main parameters of the microarchitecture under consideration are presented in Table 1. We have accounted for the difference in global clock power due to an increased number of pipeline registers that have to be clocked. To validate our results, we have used several SPEC2000 benchmarks (both integer and floating point). All tests have been performed by fast forwarding over the first 500M instructions and then doing detailed simulation for the next 100M instructions.

Table 1. Microarchitecture parameters

Parameter	Value
Pipeline Width	Eight-way, out-of-order
Instruction Window	128-entries
Load Store Queue	64-entries
Regfile	MIPS-like baseline with 196 rename registers, 64 architected registers with at most 8 rename registers per block for the EC version
I-Cache	32K, 2 way, block size 32 bytes, LRU
D-Cache	32K, 4 way, block size 32 bytes, LRU
L2 Cache	Unified, 256K, 4 way, block size 64 bytes, LRU
L2 access time	10 cycles
Memory access time	100 cycles
Branch Predictor	Gshare, 10 bits history, 1024 entries
EC Data Array	Variable size, 2 way, LRU replacement policy
EC Tag Array	4K, 4 way, LRU replacement policy
Max. Trace Length	512 instructions
Trace Lookup Penalty	1 cycle
Functional Units	8 Integer ALUs, 4 Integer MUL/DIV 4 Memory address units 4 FP Adders, 2 FP MUL/DIV
Technology [9]	130nm - V_{dd} = 1.4V, V_t = 0.22V 90nm - V_{dd} = 1.2V, V_t = 0.20V 65nm - V_{dd} = 1.1V, V_t = 0.18V
Normalized leakage current per device [11]	130nm – 80 nA 90nm – 280 nA 65nm – 280 nA

6. EXPERIMENTAL RESULTS

Performance improves as the EC size is increased and the 128K version performs best with an average penalty of about 5%. The trace delete algorithm has only a minor influence in this case.

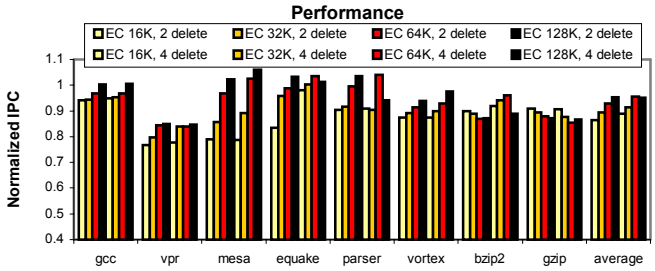

Figure 3. Normalized performance for EC-based microarchitecture, with varying EC size and replacement algorithms

In terms of energy consumption, we notice a significant reduction when increasing the number of mispredictions that can be tolerated. In this case, the best configuration (128K, four mispredictions delete algorithm) offers both best performance and best energy efficiency.

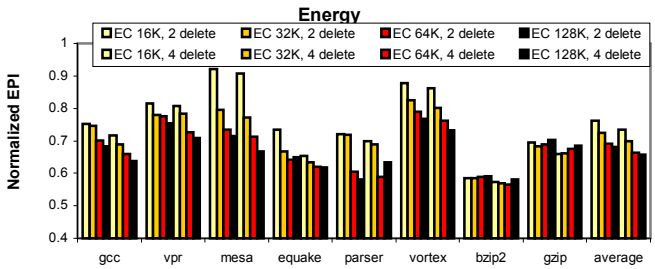

Figure 4. Normalized EPI for EC-based microarchitecture, with varying EC size and replacement algorithms

Looking at the adaptive Register File, our experiments show that it helps in 75% of the cases (6 out of 8 benchmarks), with differences of almost 8% for benchmarks like *bzip2* or *equake*. On average, it improves performance by about 5% (Figure 5). For this test, we used the best configuration found so far, with a 128K DA, two way associative and four mispredictions delete algorithm.

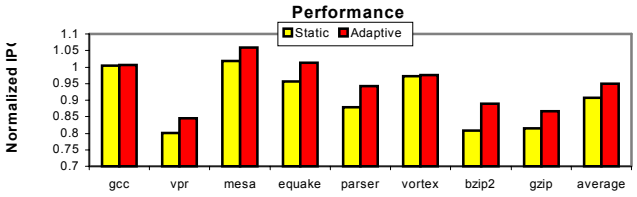

Figure 5. EC-based microarchitecture with static vs. adaptive Register File configuration, compared against the baseline

As we can see in Figure 6, the back-end throttling mechanism helps significantly in several cases, making the EC-based microarchitecture significantly faster (*equake*, *parser*, *bzip2*). On average, performance can be brought to the same level as the baseline. Increasing the EC size helps in this case as well, the only two benchmarks having a different behavior being *bzip2* and *gzip*. For *bzip2*, the time spent in trace replay mode is very high even with a small EC, so when increasing its size we don't see any real improvement. For *gzip*, the time spent in replay mode increases with increasing EC size, but it also has a much higher data cache miss rate (almost 3%). In this case, the replay mode is not able to schedule around the variable latency loads, so the trace-execution mode actually hurts the performance.

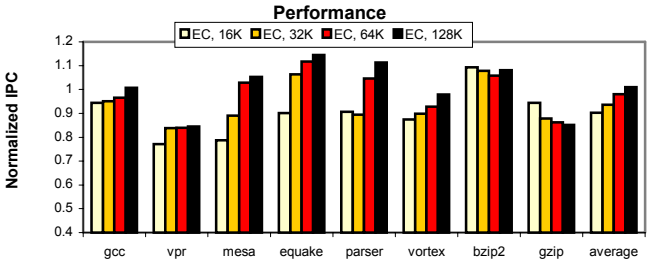

Figure 6. Performance of the EC-based microarchitecture using 50% back-end throttling, for various DA sizes

As process technology evolves and transistors are getting smaller, dynamic power becomes a smaller fraction of the total power used by the processor. In Figure 7, we study the evolution of the achievable energy savings in 130nm, 90nm and 65nm process technologies. While at 130nm this configuration saves more than 30% of the total energy, at 90nm it only saves around 13% and at 65nm this figure decreases even more, to 10% (first four bars in the three charts).

The last two bars on each chart are named Low Leakage (marked LL), and are obtained assuming that a leakage control mechanism is in place. We used the mechanism described in [2], which can essentially reduce by half the leakage current for each memory cell. Using such a mechanism, the energy efficiency increases, bringing the savings up to 19% at 65nm, 24% at 90nm and more than 40% at 130nm.

7. CONCLUSIONS

In this paper, we have studied an EC-based microarchitecture aimed at increasing the power efficiency of a superscalar out-of-order processor through reusing as much as possible from the work performed in the front-end of the pipeline. We have evaluated the design space available for such a processor, looking at different cache configurations and trace replacement algorithms, and we have proposed two mechanisms (adaptive Register File and back-end throttling) that can bring the performance to the same level offered by the superscalar out-of-order, no-EC, counterpart. We show that an average of up to 35% reduction in energy is possible, without sacrificing the overall performance.

We have also evaluated the scalability of this solution in deep submicron process technologies, and we found that, without a leakage controlling mechanism, most of the savings will be offset by the increased static energy. When controlling the leakage using the gated ground method, we obtained a 23% and respective 19% energy reduction for 90nm and 65nm process technologies.

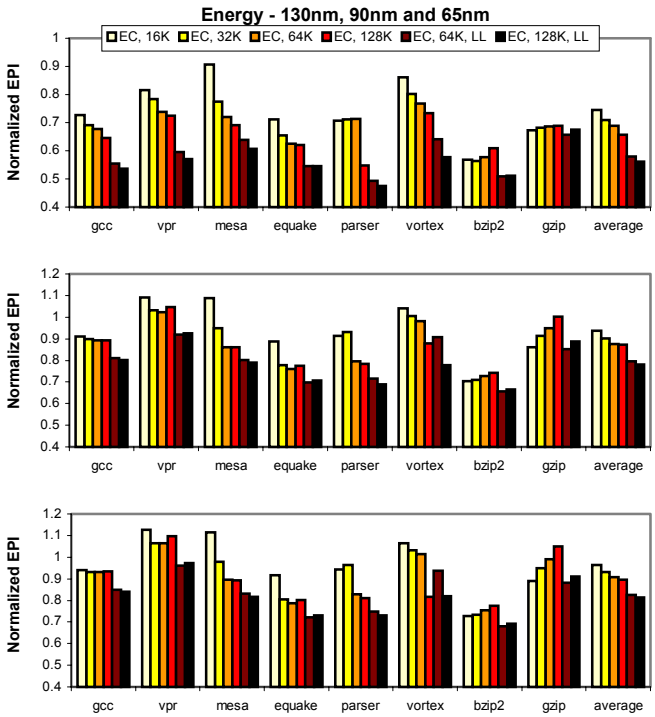

Figure 7. Normalized EPI for the EC-based architecture at 130nm (top), 90 nm (middle) and 65 nm (bottom) (two way associative DA, four mispredictions delete algorithm, back-end throttling)

8. REFERENCES

[1] F. Theeuwen and E. Seelen, "Power Reduction Through Clock Gating by Symbolic Manipulation," in Proceedings of the Workshop on Logic and Architecture Synthesis, 1996.

[2] A. Agarwal, H. Li, and K. Roy, "DRG-Cache: A Data Retention Gated-Ground Cache for Low Power", in Proceedings of DAC, 2002.

[3] J. Kin, Gupta Munish, and W. H. Mangione-Smith, "The filter cache: an energy efficient memory structure," in Proceedings of the International Symposium on Microarchitecture, pages 142-147, December 1997.

[4] N. Bellas and I. Hajj, "Architectural and Compiler Techniques for Energy Reduction in High Performance Processors," in Proceedings of ISLPED, pages 72-75, August 1998.

[5] INTEL Corp - US Patent US6170038 "Trace based instruction caching".

[6] B. Solomon, A. Mendelson, D. Orenstein, Y. Almog, and R. Ronen, "Micro-Operation Cache: A Power Aware Frontend for Variable Instruction Length ISA," in Proceedings of ISLPED, August 2001.

[7] R. Rosner, A. Mendelson, and R. Ronen, "Filtering Techniques to Improve Trace-Cache Efficiency," in Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, pages 37-48, September 2001.

[8] E. Talpes and D. Marculescu, "Power Reduction Through Work Reuse," in Proceedings of ISLPED, pages 340-345, August 2001.

[9] International Technology Roadmap for Semiconductors, 2003 Edition, http://public.itrs.net/

[10] K. C. Yeager, "The MIPS R10000 superscalar microprocessor", in Proceedings of the International Symposium on Microarchitecture, pages 28-40, April 1996.

[11] J. A. Butts, and G. S. Sohi, "A Static Power Model for Architects", in Proceedings of the International Symposium on Microarchitecture, pages 191–201, December 2000.

[12] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," in Proceedings of the ISCA, pages 83-94, June 2000.