Microprocessor Pipeline Energy Analysis

Karthik Natarajan*, Heather Hanson*, Stephen W. Keckler, Charles R. Moore, Doug Burger

Computer Architecture and Technology Laboratory
Department of Computer Sciences
*Department of Electrical and Computer Engineering
The University of Texas at Austin
cart@cs.utexas.edu — www.cs.utexas.edu/users/cart

ABSTRACT

The increase in high-performance microprocessor power consumption is due in part to the large power overhead of wide-issue, highly speculative cores. Microarchitectural speculation, such as branch prediction, increases instruction throughput but carries a power burden due to wasted power for mis-speculated instructions. Pipeline over-provisioning supplies excess resources which often go unused. In this paper, we use our detailed performance and power model for an Alpha 21264 to measure both the useful energy and the wasted effort due to mis-speculation and over-provisioning. Our experiments show that flushed instructions account for approximately 6% of total energy, while over-provisioning imposes a tax of 17% on average. These results suggest opportunities for power savings and energy efficiency throughout microprocessor pipelines.

Categories and Subject Descriptors

C.0 [**Computer Systems Organization**]: General

General Terms

Measurement, Experimentation, Performance

Keywords

power, energy, speculation, over-provisioning, alpha 21264

1. INTRODUCTION

Power consumption in high-performance microprocessors has increased with successive generations. High-end desktop and server processors such as the IBM Power 4 consume as much as 120 watts under typical conditions. A root cause of the dramatic power rise is the increase in clock rate to improve performance, which in turn has affected nearly every facet of superscalar microprocessor design. The drive for smaller and faster transistors has caused optimization in fabrication technologies that have increased leakage current. Dynamic circuits provide faster circuits at the expense of higher power. Deeper pipelines increase clock loading and overall circuit capacitance. In this paper, we examine two categories of microarchitectural features used in high-performance microprocessors that contribute to bottom-line performance at the cost of substantial power use: speculation and over-provisioning.

Superscalar microprocessors rely on speculation to feed their wide issue, out-of-order, and deep pipelines. Control speculation, data dependence speculation, hardware prefetching, cache way prediction, pipeline scheduling speculation, and other predictive mechanisms allow the processor core to make forward progress without waiting for long-latency operations to complete. Speculation offers opportunities for saving energy by filling the pipeline with useful work to do, thereby increasing throughput and reducing the program execution time. With fewer idle cycles spent resolving cache line addresses or branch targets, for example, the processor could finish tasks earlier, using less static power and allowing more opportunity to transition to a lower-power mode.

However, speculative techniques also cause a power burden from effort wasted on mis-speculated instructions. In addition to predictor structure control logic and arrays, speculative features also require additional resources throughout the chip, effectively providing extra room in the pipeline for extraneous instructions. Each instruction that is ultimately discarded contributes indirectly to elevated power levels due to the need for increased structure sizes, which lead to higher levels of transistor leakage current and more signal capacitance. A useless instruction also directly affects dynamic power through datapath switching activity until it is ejected from the pipeline. Our results show that approximately 6% of the total program energy is spent on mis-speculation.

The second microarchitectural feature examined in this paper is *over-provisioning* that results from excess capacity in the pipeline. Over-provisioned hardware structures have a wide range of effects on microprocessor power and energy consumption. Under-used array capacities and read/write ports cause the array to consume more dynamic power than necessary for signal switching in larger decoders and longer wordlines and bitlines. Furthermore, the excessive number of transistors contribute to greater leakage current. Secondary effects on neighboring functional units include longer interconnect to route around the overly large units and pos-

ISLPED'03, August 25–27, 2003, Seoul, Korea.

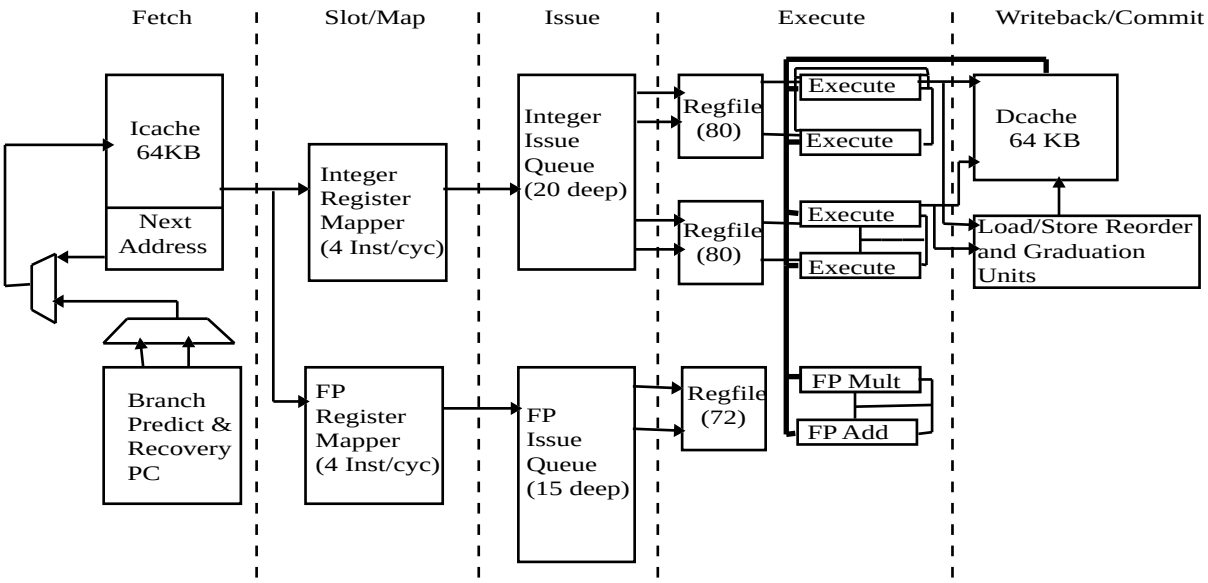

Figure 1: Alpha 21264 Pipeline Diagram

sibly increased temperature levels. This study examines the dynamic power wasted by hardware components that require full power levels to process partially filled pipeline stages and incur the maximum power penalty. The results indicate that approximately 17% of the total energy budget pays the power tax produced by over-provisioning in the map and issue stages.

Section 2 provides an overview of a wide-issue, out-of-order, superscalar microprocessor pipeline and Section 3 describes our performance and power model for this pipeline. Section 4 presents the experimental measurements of power losses due to speculation and over-provisioning. Section 5 discusses power management, including related work and future challenges. Section 6 concludes with final remarks and future directions for this study.

2. ALPHA 21264 PIPELINE

This section describes the pipeline of the Alpha 21264, a 4-wide issue, out-of-order, superscalar processor [7, 4, 3]. The 21264's core pipeline, illustrated in Figure 1 [7], relies on aggressive use of speculation and hardware resources in order to achieve high instruction throughput.

Fetch: The fetch stage reads four consecutive instructions from the instruction cache and enters them into the fetch queue. Cache line and cache way predictions, which speculatively select the next bytes to fetch every cycle, maintain high fetch rates.

Slot: The slot stage assigns four incoming instructions to integer and floating-point execution units each cycle. If the branch predictor opposes the cache line prediction made in the previous stage, the slot logic squashes the instructions and clears the fetch queues to correct the mis-speculation.

Map: Register renaming is performed in the map stage by separate integer and floating point mappers, which are 80-entry content-addressable memory arrays, each capable of handling four incoming instructions every cycle.

Issue: The issue stage places the instructions in the integer and floating point issue queues. The integer issue queue contains 20 instructions and the floating point holds 15 instructions. As operands and execution resources become available, the instructions issue from the queues to the ALUs.

Execute: The ALU units perform operations in the execute stage. Execution units and register files are separated into three distinct clusters: two integer and one floating-point cluster each contain two ALU units. Multi-cycle operations are pipelined to support a maximum of four integer and two floating point instructions entering the execution clusters each cycle.

Writeback: Once the instruction has completed, the results are written back to the destination registers.

Commit: The core pipeline terminates with the write-back stage, and then the commit stage follows to re-order instructions to their original fetched sequence and retire the instructions from the pipeline. In the commit stage, each instruction checks to see if it has generated traps, which occur due to incorrect execution or incorrect branch targets. If a trap has occurred, the entire processor pipeline is cleared and instructions are re-fetched. The trap types tracked in this study are branch mispredictions, load-store traps, load-load traps, and memory traps.

In the case of branch instructions, the commit stage compares the actual target with the predicted target. An address mismatch indicates a branch misprediction, which requires a pipeline flush to correct. Load-store traps occur when a load is speculatively issued before an older store to the same address is committed, potentially loading an incorrect value. Multiple load instructions with the same address may be speculatively issued out of order; detecting incorrect completion order and triggering load-load traps enforces consistency in shared-memory multiprocessor environments. The on-chip memory system is capable of handling multiple cache misses. However, if this capacity is exceeded or if there are conflicts between cache requests, the memory system will generate a trap for the offending instruction.

Table 1: Power Model

Component	Energy Per Cycle (nJ)	Average Power (W)
Clock	38.33	23.00
System	6.53	3.92
	Energy Per Access (nJ)	Average Power (W)
Fetch	3.49	6.74
FP mapper	1.05	2.51
FP issue queue	0.60	1.50
FP ALU	3.58	0.0
FP register file	1.67	0.0
Integer mapper	1.55	3.72
Integer issue queue	2.06	4.95
Integer ALU	2.33	2.34
Int register file	2.51	8.48
Load and store queues	4.25	0.98
Data cache	10.00	2.210
gzip Total:		64.48

3. EXPERIMENTAL METHODOLOGY

This section describes the simulation infrastructure used in this study to model a high-performance superscalar processor pipeline. We used a validated microarchitectural simulator, sim-alpha [5] that models behavior of the Alpha 21264 design, and incorporated the Wattch [1] power model with augmented power and performance measurements.

3.1 Microarchitectural Simulator

Sim-alpha models many low-level hardware features that support speculation throughout the pipeline. The cache model includes cache line and associative way prediction, and optimistically issues loads and stores as if there were no address conflicts and no port contention. The fetch unit uses the Alpha tournament branch predictor to speculatively determine the direction and target address of branch instructions [11]. The simulator also models trap detection and recovery, including clearing the pipeline and re-fetching instructions.

We augmented Wattch's power model with additional components, such as I/O pins and an estimate of bus and system interface power, and adapted the model for Alpha-specific features with a datapath width of 64 bits and processor frequency of 600 Mhz.

We designed the power model to produce power levels similar to published data [16, 8, 13]. Our baseline power rating with all units consuming full power is 71 watts. However, in our study, we separate the pipeline into structures that consume constant power each cycle regardless of the number of incoming instructions, such as map logic and issue queues, and other components such as ALUs and register files that vary in power consumption depending upon instruction activity. For the power-variable structures, we make the simplifying approximation that structures consume a fixed amount of power per instruction and are effectively clock-gated when idle; for example, an adder consumes the same amount of power for each add operation, regardless of the operand bit values. For the benchmarks in this study, our model produces power levels ranging from 54 to 62 watts, reflecting reduced power due to clock-gating and limited use of the floating-point cluster.

Table 1 lists components of the power and energy model. The second column shows the breakdown of energy per cycle for global components and energy per instruction for individual structures. To calculate energy use, we multiply the count of structure accesses by the power cost per access for individual components. We multiply the clock and system (including bus interface units and package pins) power costs per cycle with the program length for the global-structure energy total. The total energy is the sum of individual and global structures; average power is the total energy divided by program length. The third column in Table 1 shows results for average power for a representative program, *gzip*.

Our benchmark suite consists of several programs from the SPEC 2000 suite that represent a range of application behavior: *gzip, vpr, gcc, crafty, parser, eon, gap, bzip2* and *equake*. We simulated each benchmark for a total of one billion committed instructions after fast-forwarding through initialization code to the maximum amount allowed by the simulator.

3.2 Pipeline Monitoring

To gain insight into pipeline over-provisioning and speculation, we monitor the simulated pipeline by two simultaneous methods. We track each instruction's path through the pipeline and keep a record of its hardware structure accesses. Meanwhile, we compile histograms of accesses to each major structure in the pipeline. With these two measurements, we are able to observe pipeline utilization for programs in the benchmark suite, evaluate speculative mechanisms' ability to fill the 4-wide pipeline with useful work, and determine the power overhead of speculation and over-provisioning.

3.2.1 Speculation

We monitor the power overhead of speculation by observing each instruction throughout the pipeline. When an instruction exits the pipeline, its hardware access record is classified into one of six categories according to reason for termination. The categories separate work performed by the processor core into useful work for committed instructions (COM) and distinct causes of wasted effort: branch mispredictions (BR), cache line predictor (LP) mispredictions, load-load (LL) conflicts, load-store (LS) mis-speculation, and memory traps (MEM). For example, an ADD instruction ejected from the pipeline when another instruction triggers a memory trap would contribute its access history to the MEM category even though it did not cause the trap because effort expended on its behalf was wasted by the memory mis-speculation.

At the conclusion of program simulation, power models applied to structure access counts determine the total energy, or power accumulated over time. We maintain a separate energy account for supporting structures such as the global clock network, system interface, and I/O pins that are not attributed to individual instructions.

3.2.2 Over-provisioning

We monitor structure utilization throughout the pipeline by collecting information each cycle on the number of accesses to each structure. Then, we apply our per-access power model and sum the structures' power use over the duration of the program execution to estimate the total en-

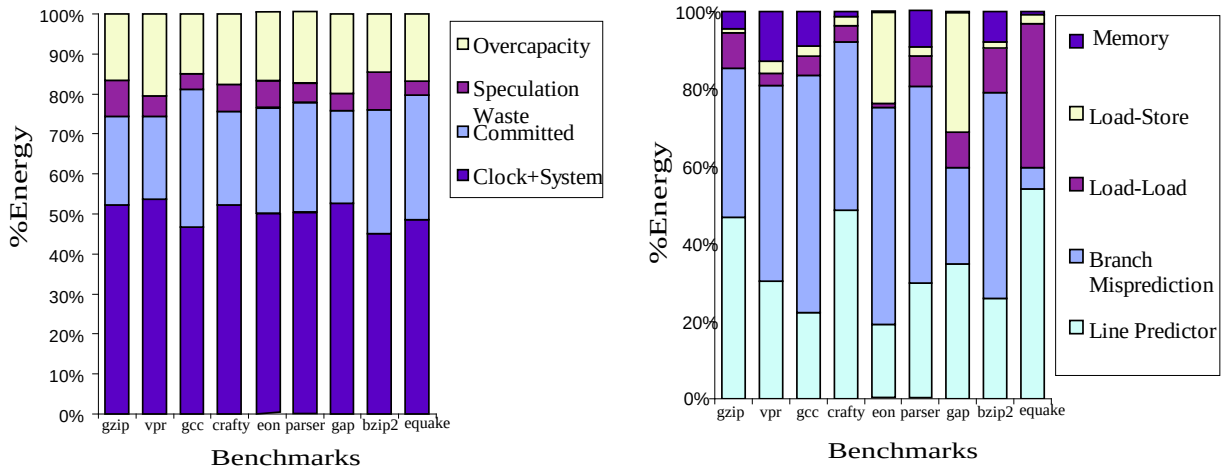

Figure 2: Energy Expenditure: (a) Overview (b) Speculation Energy by Category

ergy consumed.

Some microarchitectural structures, such as the integer and floating-point units and the caches can be designed to burn a negligible amount of power when they are unneeded. The calculated energy totals include contributions from these units according to the number of structure accesses, with no penalty calculated for over-provisioning.

However, other structures are typically accessed every cycle, regardless of how many instructions actually use them. This class of structures is designed with sufficient capacity and ports to handle peak throughput, but under typical loads add an excess power burden to the pipeline. For example, the floating-point mapper and issue queue run continuously even during predominantly integer programs as they search the incoming instruction stream for useful work to perform. In our model, the power consumed by the integer and floating-point mappers, integer and floating-point issue queues is separated into power spent on instructions and power wasted by unused slots in the pipeline. The instruction power is categorized into useful and non-useful work (as specified in section 3.2.1); the empty-pipe power is accounted separately as a distinct power overhead.

Note that in the over-provisioning analysis, some portion of a chip's global clock network and supporting circuitry is over-provisioned due to the extra capacitive load and area of the over-provisioned structures; this study does not include the global structures in the utilization accounting.

We include only dynamic power in this study, based on our model of the 21264's 350nm process technology with negligible static power due to leakage current. In more recent fabrication technologies, larger leakage currents significantly increase the penalty of unused and under-used structures throughout the pipeline.

4. EXPERIMENTAL RESULTS

Figure 2a shows the measured energy components for each benchmark. The chart shows the clock and system interface uses about half of the total energy. Energy spent on useful work that results in committed instructions contributes another 26%. Energy wasted on instructions that are flushed due to mis-speculation of all types combined constitutes about 6% of the total program energy. Finally, approximately 17% of the energy is spent on under-used map and issue structures that burn power each cycle.

4.1 Speculation

The energy wasted on pipeline flushes is a relatively small percentage of the total energy expended during program execution. However, it is significant (11% to 40%) when compared to the energy spent in the pipeline on committed instructions. Figure 2b shows the energy wasted by each trap type. In general, branch misprediction and incorrect cache line prediction are the main causes of wasted energy, though the energy spent on mis-speculation is dependent on program characteristics and varies widely among benchmarks. For example, wasted energy due to mispredicted branches is much greater for *bzip2* than for *gap*, which spends a higher percentage of energy on load-store mis-speculation. Also, the effect of branch misprediction is least in the benchmark *equake* as a result of its high branch prediction accuracy.

Table 2 shows the average number of instructions that are flushed from the pipeline as a result of each type of mis-speculation and how often the trap was triggered per benchmark. Even rare occurrences of mis-speculation can affect energy consumption. For example, *vpr* and *gzip* waste almost the same amount of energy due to mis-speculation and have an almost equal number of branches. However, *vpr* wastes more energy due to branch mispredictions than *gzip* despite a better branch predictor accuracy rate because it evicts more instructions in pipeline flushes. *Vpr* evicts an average of 53 instructions from the pipeline when a branch misprediction occurs, compared to 34 in *gzip*. With a fuller pipeline, *vpr* incurs a higher energy cost per pipeline flush, for an overall higher energy penalty.

4.2 Over-provisioning

We found that overprovisioned structures consume between 15% and 21% of the total energy. Our study highlights the inherent overprovisioning of the Alpha 21264 front-end structures. The pipeline has sufficient capacity to map a total of eight and issue a total of six instructions in the floating-point and integer clusters each cycle, despite a limited supply of four instructions fetched per cycle. The built-in extra space wastes effort on futile map and issue switch-

Table 2: Mis-speculation Occurrence and the Average Number of Instructions Evicted Per Flush

Benchmark SPECcpu2000	BR				LL		LS		MEM	
	Branch Pred. Accuracy	# branches (mil.)	# traps (mil.)	avg. # inst. evicted	# traps (mil.)	avg. #inst. evicted	# traps (mil.)	avg. # inst. evicted	# traps (mil.)	avg # inst. evicted
164.gzip	0.925	104	7.24	34	1.40	53	0.21	37	2.47	34
175.vpr	0.942	108	5.95	53	0.50	53	1.25	46	2.52	34
176.gcc	0.965	152	4.95	28	0.50	29	0.48	28	1.33	29
186.crafty	0.934	123	7.20	28	0.98	26	0.51	29	1.37	31
197.parser	0.960	162	6.22	31	0.90	34	0.49	34	2.37	32
252.eon	0.948	119	6.06	42	0.14	46	4.91	47	2.52	32
254.gap	0.957	781	3.04	34	2.19	26	1.31	34	5.27	24
256.bzip2	0.936	143	8.09	37	2.03	42	0.46	34	3.31	33
183.equake	0.993	174	0.38	32	3 2.0	32	0.20	30	3.02	25

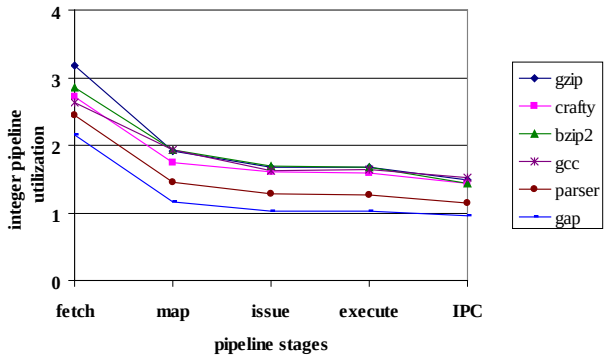

Figure 3: Pipeline Utilization

ing. With our predominantly integer benchmark suite, we found that the floating point mapper and issue queue were essentially unused but consumed power continuously.

The benchmarks did use integer structures, though not to their full capacity. Despite aggressive speculation to fill the pipeline with potentially useful instructions, the integer mapper and issue stages produced less than half of their peak output under typical conditions. Every missed opportunity to map and issue four integer instructions caused an additional power penalty. Thus, the map and issue stages spend most of their power on over-provisioning in these experiments.

4.3 Pipeline Analysis

The 4-wide pipeline with extra map and issue resources provides flexibility for high-throughput processing for diverse applications. Typical programs in our suite do not take full advantage of the wide datapaths, and the performance benchmark of instructions per cycle (IPC) hovers between one and two committed instructions per cycle.

Figure 3 illustrates the integer portion of the pipeline with a plot of average utilization per stage for selected benchmarks that use the integer clusters almost exclusively (less than 0.5% of these instructions use floating-point resources). The drop between fetch and map stages is largely due to frequent cache line mispredictions in our simulations. The instruction flow is fairly steady between the issue and execute stages, averaging between 1 to 2 instructions per cycle. A small percentage of "innocent bystander" instructions will be evicted during pipeline flushes, but most travel through the datapath unscathed. The majority will retire successfully at a rate between 1 and 2 instructions per cycle, as shown in the graph as the program IPC.

The useful pipeline width effectively narrows between fetch and commit stages from four down to less than two instructions wide for our benchmarks. The remaining space in the pipeline translates to under-used execution units, queue entries, and read/write ports throughout the design. The extent of over-provisioning pipeline resources is a function of hardware capacity and software behavior. We observed consistent under-use of resources throughout the pipeline in our predominantly integer suite. The pipeline is designed to handle a wide variation in workloads, with extra resources provided to support three pipeline clusters. Other computation-intensive programs could have higher utilization rates throughout the wide pipeline, or due to data dependences, might spend more time waiting for previous results. Seldom-used resources may provide a significant performance benefit for critical applications, and reduce the total energy expended.

5. POWER MANAGEMENT

The Alpha 21264 design team addressed that processor's critical power issues with a hierarchical clock distribution, conditional clocking in the floating-point cluster, and low-swing busses. [8], Other power management techniques for this class of superscalar processors approach the problem from a microarchitectural perspective. Previous work for reducing power wasted by speculation includes a pipeline-gating technique that limits speculatively issued instructions that are likely to be mispredicted [12]. One study investigated the power of branch predictors and evaluated pipeline speculation control in [14]. Another scheme is just-in-time instruction delivery [10], whereby instructions are fetched as late as possible, which reduces the number of instructions flushed from front-end queues. We suggest further exploration in front-end structures, such as sentries that monitor the instruction stream and select alternate lower-power structures or reduced-power modes as warranted.

Several researchers have proposed hardware resizing and reconfiguring techniques [2, 15, 6, 9] to address power wasted by oversized hardware structures. A power manager could

actively use information gathered from issue queue usage statistics to tailor the datapath width to fit the instruction stream, and direct unused or underused resources into a low-power mode.

Judicious use of hardware resources will become even more important as the projected increase in leakage current adds to the power liability of each transistor on die. The shifting balance of static and dynamic power is likely to dictate integrated power management strategies from circuit through architecture for future generations of microprocessors.

6. CONCLUSION

Our experimental infrastructure provides a detailed description of pipeline resource use (and misuse) within a 4-way issue superscalar processor. We identified the power overhead associated with two classes of microarchitectural features that boost performance: speculation and pipeline over-provisioning. We found that power wasted by mis-speculation accounts for approximately 6% of the total energy, and power spent on under-used map and issue resources contributes 17% of total energy, considering only dynamic power and explicit power overheads. We find that the beginning of the core pipeline is most directly affected by speculation-related effort, subject to clearing and refilling to correct mis-speculation. The tail end of the pipeline has the advantage of of containing fewer enqueued instructions subject to eviction upon a pipeline flush, and more information available from upstream stages for detection and power control of idle or under-used resources.

Our results suggest that a power management policy that provides effective speculation and hardware resource reconfiguration could be highly effective in reducing the power and energy in a wide-issue superscalar processor. Technology trends indicate continuing issues with dynamic power due to high clock rates, and emerging challenges due to static power from leakage current. In addition to semiconductor manufacturing and circuit styles, microarchitectural decisions provide an opportunity to effectively manage pipeline power and energy.

Future work for this project will include extensions to the microarchitectural simulator to evaluate the power overhead of other microarchitectural features and develop energy efficient techniques based on pipeline utilization statistics.

7. ACKNOWLEDGMENTS

The authors would like to thank Raj Desikan for his assistance with the sim-alpha simulator. This research is supported by the Defense Advanced Research Projects Agency under contract F33615-01-C-1892, NSF instrumentation grant EIA-9985991, NSF CAREER grants CCR-9985109 and CCR-9984336, two IBM University Partnership awards, and grants from the Alfred P. Sloan Foundation, the Peter O'Donnell Foundation, and the Intel Research Council.

8. REFERENCES

[1] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual Symposium on Computer Architecture (ISCA)*, pages 83–94, 2000.

[2] A. Buyuktosunoglu, S. Schuster, D. Brooks, P. Bose, P. Cook, and D. Albonesi. An adaptive issue queue for reduced power at high performance. In *Workshop on Power-Aware Computers Systems, held in conjunction with ASPLOS*, Nov 2000.

[3] Compaq Computer Corporation. *Alpha 21264 Microprocessor Hardware Reference Manual*, July 1999.

[4] Compaq Computer Corporation. *Compiler Writer's Guide for the Alpha 21264*, 1999.

[5] R. Desikan, D. Burger, and S. W. Keckler. Measuring experimental error in microprocessor simulation. In *Proceedings of the 28th Annual Symposium on Computer Architecture*, pages 266–277, 2001.

[6] D. Folegnani and A. Gonzalez. Energy-effective issue logic. In *28th International Symposium on Computer Architecture*, pages 230–239, July 2001.

[7] B. A. Gieseke, R. L. Allmon, D. W. Bailey, B. J. Benschneider, S. M. Britton, J. D. Clouser, H. R. F. III, J. A. Farrell, M. K. Gowan, C. L. Houghton, J. B. Keller, T. H. Lee, D. Leibholz, S. C. Lowell, M. D. Matson, R. J. Matthew, V. Peng, M. D. Quinn, D. A. Priore, M. J. Smith, and K. E. Wilcox. A 600 Mhz superscalar RISC microprocessor with out-of-order execution. In *IEEE International Solid-State Circuits Conference*, pages 176–177, 451, February 1997.

[8] M. K. Gowan, L. L. Biro, and D. B. Jackson. Power considerations in the design of the Alpha 21264 microprocessor. In *Proceedings of the 35th Design Automation Conference*, pages 726–731, 1998.

[9] A. Iyer and D. Marculescu. Run–time scaling of microarchitecture resources in a processor for energy savings. In *Cool Chips Workshop, held in conjunction with MICRO–33*, 2000.

[10] T. Karkhanis, J. E. Smith, and P. Bose. Saving energy with just in time instruction delivery. In *Proceedings of the 2002 International Symposium on Low Power Electronics and Design*, pages 178 – 183, 2002.

[11] R. Kessler. The Alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, March/April 1999.

[12] S. Manne, A. Klauser, and D. Grunwald. Pipeline gating: Speculation control for energy reduction. In *Proceedings of the 25th Annual Symposium on Computer Architecture (ISCA)*, pages 132–141, 1998.

[13] M. Matson, D. Bailey, S. Bell, L. Biro, S. Butler, J. Clouser, J. Farrell, M. Gowan, D. Priore, and K. Wilcox. Circuit implementation of a 600mhz superscalar RISC microprocessor. In *Proceedings of the International Conference on Computer Design*, pages 104–110, 1998.

[14] D. Parikh, K. Skadron, Y. Zhang, M. Barcella, and M. Stan. Power issues related to branch prediction. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture*, pages 211 –222, 2002.

[15] D. Ponomarev, G. Kucuk, and K. Ghose. Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources. In *34th International Symposium on Microarchitecture*, pages 90–101, Dec 2001.

[16] K. Wilcox and S. Manne. Alpha processors: A history of power issues and a look to the future. In *Cool Chips Tutorial: An Industrial Perspective on Low Power Processor Design*, pages 16–37, 1999.