# Power-Aware Operand Delivery

Erika Gunadi and Mikko H. Lipasti
Electrical and Computer Engineering Department
University of Wisconsin
1415 Engineering Dr
Madison, WI 53706
egunadi@ece.wisc.edu, mikko@ece.wisc.edu

## ABSTRACT
Based on operand delivery, existing microprocessors can be categorized into architected register file (ARF) or physical register file (PRF) machines, both with or without payload RAM (PL). Though many previous generation microprocessors use a PRF without PL, the trend of newer microprocessors targeting lower power environments seem to be moving towards ARF with PL. We quantitatively analyze power consumption of different machine styles: ARF with PL, ARF without PL, PRF with PL, and PRF only machine. Our result shows that PRF without PL consumes the least amount of power and is fundamentally the best approach for building power-aware out-of-order microprocessors.

## Categories and Subject Descriptors
B.1.2 [**Control Structures and Microprogramming**]: Control Structure and Design Aids - *automatic synthesis, simulation.*

## General Terms
Measurement, Performance, Design, Experimentation.

## Keywords
Power, Microarchitecture, Renaming.

## 1. INTRODUCTION
As power rapidly becomes a design constraint, the evolution of microprocessor technology has started to shift from performance-focused design to a more power-effective one. As more transistors are placed on a chip, more power is dissipated into heat. The heat increases the leakage power, leading to a destructive feedback cycle. Already, current microprocessors face challenges in cooling and packaging design. Researchers and industries are working vigorously to create a low-power design, especially for chips targeted for low power products such as rack-mounted servers and laptops.

Register renaming is used to support operand delivery in out-of-order machines. Two widely used design approaches for register renaming utilize either an ARF or a PRF. PRF-style machines have been around for more than a decade, starting with the MIPS R10000 [19], Alpha 21264 [12][13][5], IBM Power4 [17] and Power5 [16], and Intel Pentium4 [9]. ARF-style machines have a similar history, used in machines like the PowerPC 604 [15] and the Intel P6 [4][8] architecture, which formed the basis for the Intel Pentium Pro [14] and Pentium III [11]. ARF are also used in current generation microprocessors such as the AMD K8 [6][10] family, which includes AMD Athlon and Opteron, Intel Pentium M [7], as well as the recently launched Intel Core family. High-volume microprocessors, especially ones targeted for lower power consumption in laptops and rack-mount servers, are trending away from PRF. The fact that no quantitative analysis has been published to determine which design style is actually more power effective is our main motivation.

In existing designs, the ARF-based machines copy ready source operands at queue stage into a payload RAM (PL). In contrast, current PRF-based machines do not; instead, they are read from the PRF after the instruction issues. This difference, however, is not fundamental to the ARF vs. PRF approach. In fact, the two attributes can be mixed and matched in any combination: an ARF machine need not have a PL, and a PRF machine could have a PL.

In this work, we compared power consumption of different designs to evaluate whether the operand delivery method--ARF vs. PRF, PL or not--significantly affects power consumption. Our results show that there is in fact a significant difference: an equivalent-performance PRF machines consumes roughly 20% the dynamic power in the structures that are affected by this high-level design choice, leading to an overall reduction of roughly 6-7% of core power. This is a surprising and counterintuitive result since it is exactly the opposite of the prevailing industry trends: microprocessors intended for mobile and power-aware server applications (Intel Pentium M, Core Duo, AMD Opteron, and Turion) all employ an ARF-based approach, while notoriously power-hungry designs (Intel Pentium 4, IBM Power 4) employ the PRF-based approach. This paper shows that the industry trends are not due to this high-level design decision, but rather to additional factors like pipeline depth, frequency target, design legacy, or design methodology. Furthermore, our results suggest that microprocessors designed for low-power applications should employ the PRF style instead of the current ARF approach.

The rest of the paper is structured as follows. Section 2 describes different design spaces and their tradeoffs. Section 3 explains design methodology. Details of the design and structures modeling results are described in Section 4. Section 5 shows the experimental results and Section 6 concludes the paper.

## 2. DESIGN SPACE
We classify existing microprocessors based on operand delivery as shown in Table 1. Microprocessors can be divided into ARF and PRF style machine depending on where the speculative results are stored, and into PL or no PL based on when the operands read occur. Those two organizations are orthogonal and can be mixed and matched into four different combinations as shown in Table 1. However, existing ARF-style machines always use PL while PRF-style machines do not. All microprocessors reside in either the upper left or lower right quadrant.

ARF-style machines keep the non-speculative register values in a small ARF and store the speculative ones in the ROB. Execution results are first written to the ROB then copied to the ARF as instructions retire. In contrast to ARF-style machines, PRF-style machines store both speculative and non-speculative value in the PRF. Results are written only once in writeback stage.

A machine with PL reads the operands values before instructions are inserted into the reservation station (RS). Ready operands are copied to a PL while unready ones are delivered later via the bypass logic. Machines without PL only check the readiness of operands before inserting instructions into the RS. On issue, operands are read from necessary structures (either the PRF as in current machines like the Alpha 21264, or from the ARF+ROB).


*ISLPED'07* August 27-29, 2007, Portland, Oregon, USA.

**Table 1. Different machines models.**

| | ARF | PRF |
|---|---|---|
| Payload RAM (PL) | Intel P6, Core family, Intel Pentium M AMD K5, K8 family | None |
| No Payload RAM (No PL) | None | Intel Pentium 4, MIPS R10K, Alpha 21264, IBM Power4, Power5 |

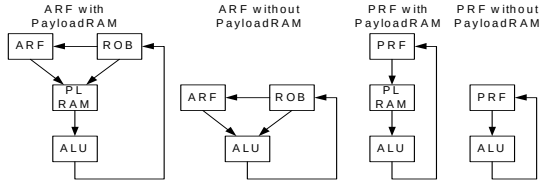


**Figure 1. Data movement in different machine models.**

In this work, we compare four microarchitectures representing the four quadrants in Table 1. Figure 1 shows the data movement diagrams for each model. Each block represents the structure while the arrow represents data movement. It can be inferred that PRF+noPL requires the least movement of data, intuitively reducing power.

Additional tradeoffs among the machine models--ARF+PL, ARF+noPL, PRF+PL, and PRF+noPL--can be seen in Table 2. We modeled 12-stage pipelines with two fetch and two decode stages. Operations done in key structures from rename to retire stage are described as read or write operation to a CAM or RAM structure. When a structure has a large portion of RAM in addition to tag-storage CAM , such as PL, it is referred to as CAM+RAM (written inTable 2 as CRAM). As RS have only a narrow portion that uses RAM, it is referred as CAM only structure.

Depending on the configuration, operand read is done at different time as illustrated in Table 2.If a PL is used, operand read is done before queue stage where instructions are inserted to the window. If noPL is used, operand read is done after issue stage. Though it seems trivial, the placement of the read stage for machines without PL adds an extra cycle between schedule and execute stage, thus increasing the load misscheduling penalty [1] and branch resolution loop.

In ARF+PL, the rename table (RAT) is accessed in the rename stage to read the mapping for input operands and to write the mapping for the output operands. Operands read is done from the ARF and/or the ROB one cycle after. The instruction is then inserted to the ROB while scheduling information is written to the RS. The operand tags and ready values are written to the PL. Once the instruction is issued, its tag is broadcasted to wake up its descendants. Operands are then read from the PL and/or bypass network and sent to the functional unit. The instruction then writes the result to the ROB, broadcasts the tag and data to PL, and updates the RAT status to ready. In the retirement stage, the data from the ROB is copied to the ARF. The RAT is updated again so that dependent instructions know that the data is now in the ARF. In ARF+noPL model, operands are read from the ARF and/or ROB after issue stage. Only the necessary tags are inserted in the PL at the queue stage.

In the third and fourth model, PRF is used instead of the ROB and the ARF. If PL is used, operands are read from PRF after rename stage and written to the PL in the queue stage. The result is written directly to the PRF. In PRF+noPL, the PRF is read after the instruction issues and the result is simply written back to the PRF.

Besides affecting the length of load misscheduling latency and branch resolution loop, these different configurations also affect the choice of solution to resolve branch mispredictions. The most widely known solution is by checkpointing the RAT on branch instructions. It can be trivially done for PRF-style machines as the PRF acts as the only operand source during the lifetime of a register. Hence, the entry in the RAT will never change once a mapping is defined. Resolving branches in out-of-order fashion has successfully been implemented in various PRF style machines such as MIPS R10000, Alpha 21264, IBM Power4, and IBM Power5.

**Table 2. Different Machine Tradeoffs.**

| | ARF + PL | ARF + No PL | PRF + PL | PRF + No PL |
|---|---|---|---|---|
| Rename | wr RAT, ram<br>re RAT, ram | wr RAT, ram<br>re RAT, ram | wr RAT, ram<br>re RAT, ram | wr RAT, ram<br>re RAT, ram |
| Read | re ARF, ram<br>re ROB, ram | | re PRF, ram | |
| Queue | wr RS, cam<br>wr ROB, ram<br>wr PL, cram | wr RS, cam<br>wr ROB, ram | wr RS, cam<br>wr ROB, ram<br>wr PL, cram | wr RS, cam<br>wr ROB, ram |
| Sched | wr RS, cam | wr RS, cam | wr RS, cam | wr RS, cam |
| Issue | re PL, ram | re PL, ram | re PL, ram | re PL, ram |
| Read | | re ARF, ram<br>re ROB, ram | | re PRF, ram |
| Exe | execute | execute | execute | execute |
| WB | wr ROB, ram<br>wr RAT, ram<br>wr PL, cram | wr ROB, ram<br>wr RAT, ram | wr PRF, ram<br>wr RAT, ram<br>wr PL, cram | wr PRF, ram<br>wr RAT, ram |
| Retire | re ROB, ram<br>wr ARF, ram<br>wr RAT, ram | re ROB, ram<br>wr ARF, ram<br>wr RAT, ram | re ROB, ram | re ROB, ram |

Unfortunately, implementing an out-of-order branch resolution becomes harder in an ARF style machine as valid register values can reside in either ROB or ARF. The RAT pointer has to be updated to ROB when an instruction enters the window and to the ARF when the instruction retires. Consequently, a simple checkpoint could have stale values. One solution would be to update all checkpointed copies during retirement. Alternatively, the machine could access both the ARF and ROB in parallel, to avoid using a stale value from the ROB. Nevertheless, both solutions could introduce additional complexity and extra power consumption. In fact, none of the current ARF-style designs employ out-of-order branch resolution; rather, they implement in-order resolution with two copies of the RAT, a regular and a retirement one. As a branch becomes the oldest, its prediction is checked. This can happen before a branch becomes the oldest instruction in the window. On a misprediction, the pipeline is flushed and new instructions are fetched. However, the newly fetched instructions cannot enter the window until the window is drained and the retirement RAT reflects the correct machine state. This retirement RAT is then copied to the regular one and new instructions are allowed to enter the window.

However, trusting that an elegant solution for out-of-order branch resolution is possible for an ARF style machine, we also modeled out-of-order branch resolution in our experiments to complement an in-order baseline similar to the current ARF-based microprocessors.

# 3. METHODOLOGY

To get accurate timing and power model, all the main structures in the OoO core are implemented in Verilog, synthesized using Synopsys Design Compiler and placed&routed using Synopsys Astro. We used LSI Logic's gflxp 0.11 micron CMOS standard cell library. RAM structures are synthesized with the latch-based RAM generator that uses the latch-based RAM cell provided by LSI Logic. Structures implemented include RAT, ROB, PL, RS, RF, and bypass network. According to our place&route results, it is reasonable to assume the same cycle time for all four models. Details of the design of these structures are presented in Section 4.

For microarchitectural simulation, we use a modified *Simplescalar / Alpha* 3.0 tool set [3]. Specifically, we extended sim-outorder to perform full speculative scheduling with squashing replay assuming constant execution latency. Our simulator also models aggressive load-store reordering with a memory dependence predictor similar to the Alpha 21264 machine. We implemented both out-of-order and in-order branch resolution as explained in Section 2.

**Table 3. Machine Configurations.**

| | |
|---|---|
| Out-of-order Execution | 4-wide fetch/issue/commit, 128 ROB, 32 ARF 24 LQ, 32 SQ, 32 sched, 12-stage pipeline |
| Branch Predictions | Combined bimodal (16k) / gshare (16k) with selector (16k), 16-entry RAS, 4-way 1k BTB |
| Functional Units | 2 iALU (1-cycle), 1 imult/idiv (3/20-cycle), 1 general memory ports (1+2 cycle) |
| Memory System (latency) | L1 I$: 64KB, DM, 64B line (2-cycle)<br>L1 D$: 16KB, 4-way, 64B line (2-cycle)<br>L2: 2MB, 8-way, 128B line (8-cycle)<br>Memory: 150-cycle |

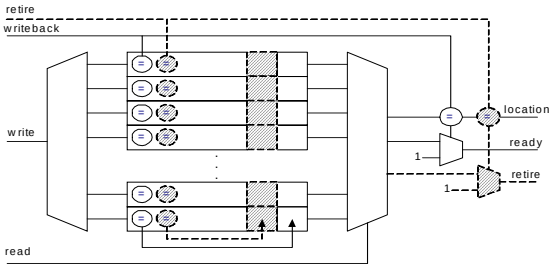


**Figure 2. RAT structure for ARF- and PRF- style machine.**

The SPEC CINT2000 benchmark suite is used for all results presented in this paper. All benchmarks were compiled with the DEC C and C++ compilers under the OSF/1 V4.0 operating system using -O4 optimization. Reference input sets and SMARTS [18] statistical sampling methodology were used for all benchmarks.

We used ARF+PL-style machine with configuration shown in Table 3 and in-order branch resolution for our baseline. The pipeline is illustrated in Table 2 with an addition of 2 fetch and 2 decode stage. All results are normalized to the baseline. We performed sensitivity analysis to get the PRF number to match the IPC of the base machine. A PRF with 96 entries results in less than 0.1% performance loss, well within the margin of error of the simulation.

# 4. DETAILS OF DESIGN

In this section, we present the details and synthesis results for the key structures. It is important to note that although our cycle time is not directly comparable to full-custom designs in a leading-edge process, it is competitive for a standard-cell design flow in 110nm and is certainly useful for making relative comparisons.

## 4.1.Register Alias Table (RAT)

The RAT is used in OoO microprocessors to resolve RAW and to eliminate WAR and WAW conditions. It provides renaming to make available a larger register set than is explicitly provided in the architecture. Before an instruction enters the window, it accesses the RAT to get physical locations of its input operands and allocates a new physical location for its destination register.

The RAT implementation is different on ARF- or PRF- style machines. In an ARF-style machine, RAT contains mapping to either ROB or ARF entries, ready bits, and retire bits. The ready bit tells that the value in the physical location pointed by the RAT is ready to read. The retire bit tells that the value is non-speculative and is located in the ARF. The RAT is accessed in three pipeline stages: rename, writeback, and retire stage. In the rename stage, it is used to locate the location of input registers and to rename the destination register into the ROB entry of the current instruction. The ready and retire bit for this entry are also cleared. In the writeback stage, the instruction updates the ready bit after successfully checking that the entry has not been renamed by subsequence instructions. A similar process is followed in the retire stage to update the retire bit.

Besides the RAM structure, the RAT needs to have comparators. Each read access has to first read the structure to get the location and the status. Then it has to compare that location with all writeback and retire tags to see if the ready or retire status needs to be changed. Precedence of updates has to be handled carefully since in the same cycle it is possible that all four rename, writeback, and retire ports need to update the same entry. In this case, the youngest rename has the highest priority and the oldest retire has the lowest one.

**Table 4. Delay (ns), area($mm^2$), power (mW) comparison.**

| | RAT-ARF | RAT-PRF | ROB-data | ROB-tag | ARF | PRF | PL | RS | BP |
|---|---|---|---|---|---|---|---|---|---|
| Delay | 2.07 | 1.98 | 3.21 | 2.01 | 2.25 | 2.46 | 2.12 | 2.03 | 1.93 |
| Area | 0.06 | 0.09 | 1.53 | 0.38 | 0.29 | 1.05 | 0.62 | 0.98 | 0.24 |
| Rename | 2.69 | 2.94 | N/A | N/A | N/A | N/A | N/A | N/A | N/A |
| Queue | N/A | N/A | N/A | 0.74 | N/A | N/A | 1.32 | 0.71 | N/A |
| Read | 2.83 | N/A | 2.83 | N/A | 1.68 | 2.52 | 1.42 | 1.21 | 4.58 |
| WB | 0.82 | 0.80 | 2.81 | 2.02 | N/A | 2.46 | 2.16 | 2.02 | N/A |
| Retire | 0.82 | 0.28 | 2.83 | 2.76 | 1.56 | N/A | N/A | N/A | N/A |

A RAT for a PRF-style machine does not have retire bits since both architectural and speculative values are kept in a single PRF structure. However, the PRF-RAT needs a scoreboard to keep track of ready bits and a free list to keep track of free registers. In the rename stage, instructions update the RAT. The scoreboard is updated in the writeback stage and the free list is updated in the retire stage.

To minimize the number of ports, the RAT is implemented using banks, one for rename with seven bits physical location, one for writeback with a single ready bit, and one for retire with a single retire bit as applicable. A freelist and a scoreboard are also implemented for the PRF-RAT. The RAT has 32 entries, 7 bits each. The freelist is a 96-entry RAM with 7-bit entries, implemented as four 32 entry RAMs to reduce port number and access time. The scoreboard is 96 entries of one ready bit. Figure 2 shows the block diagram of RAT for an ARF-style machine with one set of ports. For a PRF-style machine, the shaded part can be removed.

Table 4 shows the place&route results for our RAT implementation. The PRF-RAT has higher area due to the addition of the freelist. Rename power includes one write and two read in the RAT. Writeback power is the power needed to check and update the ready bit for the ARF-RAT and the power needed to update the scorecard for the PRF-RAT. For ARF-RAT, the retire power is needed to do the ownership checking and update the retire bit. For PRF-RAT it is needed to write the previous physical register destination back to the freelist.

## 4.2.ROB, Register File, Payload RAM

The ROB is used to keep track of the status of in-flight instructions. Instructions are inserted in the queue stage and removed in the retire stage. A ROB is implemented as a RAM-based circular queue with a head to remove retire instructions and a tail to insert new instructions. We implemented the ROB into two main structures, ROB-data and ROB-tags. ROB-data only exists in ARF-style machines, used to store speculative values. This structure is implemented as a 128-entry buffer with 64 bits per entry with 8 read ports and 4 write ports.

ROB-tags is used to store information needed while an instruction is in the window. It is separated into two main structures, one to store information inserted as instructions enter the window and another to store flags and control bits as instructions are executed. To reduce access time and power dissipation, the first is implemented as four banks of 32 entries with 40 bits each. The second is implemented as 128 entries with 16 bits data. The results are shown in Table 4.

A register file is a RAM-based structure used to store the execution results. We implemented a 32-entry register file for the ARF and a 96-entry register file for the PRF. Both implementation has eight read ports and 4 write ports. The results are shown in Table 4

PL is used to store input operands before instructions are executed. PL consists of a CAM structure for tags and a RAM structure for the values. As an instruction is inserted to the window, its input tags are inserted to the PL along with its ready input data. The tags are

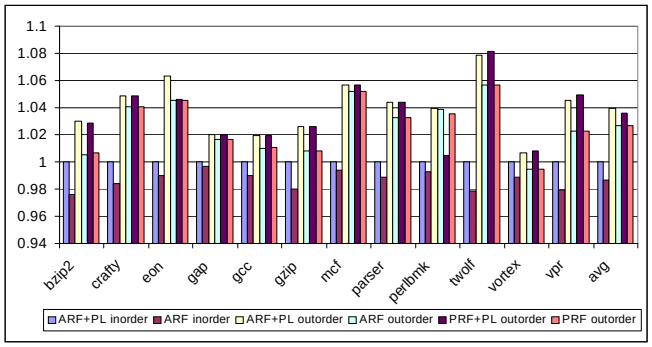


**Figure 3. Normalized IPC Comparison.**

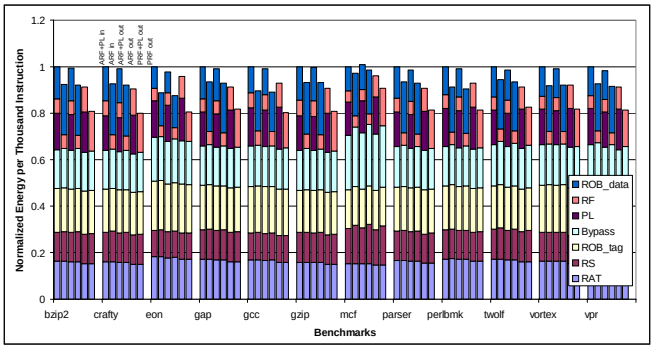


**Figure 4. Normalized Energy Comparison.**

searched associatively and data is latched as non-ready operands are broadcast by their producer in the writeback stage.

We model a 32-entry PL, with two tags and two 64-bit operands per entry. The place&route results for PL are shown in Table 4. Write and read power are power needed to activate one write port and one read port accordingly. Writeback power is for operand broadcasts. We estimate that each broadcast will result in one tag match and lead to the writing of a single entry; this is not completely accurate, since a broadcast might not match any entry or match up to 32x2 entries.

## 4.3.Reservation Station and Bypass Network

For completeness, a 32-entry RS with selection logic is modeled. Each entry has 43 bits to store input tags, destination tags, ready bits, and opcode. We also model a 2-level bypass network (BP) used to bypass data for back-to-back execution. The first level is to catch data from execute stage and the second one is to catch data from the writeback stage. Each level of bypass network has eight comparators and two muxes. Table 4 shows the place&route results.

# 5. RESULTS

Figure 3 shows normalized IPC comparison for six different configurations. The first two bars use in-order branch resolution while the rest use out-of-order branch resolution. As seen, out-of-order branch resolution adds 3% of performance on average, while doing operand read between issue and execute (noPL) decreases IPC by 1-2%.

Figure 4 shows energy per thousand instructions. PRF+noPL with out-of-order branch resolution consumes the least amount of energy: roughly 20% less energy than the baseline, saving 6-7% of total chip energy (assuming that these structures consume about 30% of total chip power). This assumption is not unreasonable as [2] shows that the ROB, RS, and RAT consumes 25.6% of total chip power in Intel Pentium Pro. The reduction mostly came from the elimination of PL and ROB-data. The energy spent by RAT, RS, ROB-tag, and bypass network does not change much across different configurations.

Interestingly, ROB-data and PRF do not consume the most amount of energy although they are the two largest structures in the system because they are not accessed for every operand read. Operands are often delivered via the bypass network, thus reducing the number of accesses significantly. It is also interesting that the energy spent by ARF+PL with out-of-order branch resolution is relatively the same as the amount of energy spent by the in-order baseline, despite the 3% increase in performance. It implies an increase in useless speculative activity from the out-of-order branch resolution, which could be addressed with some form of intelligent speculation gating.

# 6. CONCLUSIONS

Existing microprocessors can be categorized into ARF- or PRF-style, both with or without PL. Though many older generation microprocessors use PRF+noPL, the trend of newer microprocessors targeting lower power products seems to be moving towards ARF+PL. We quantitatively analyzed the power consumption of different machine styles: ARF+PL, ARF, PRF+PL, and PRF. Our result shows that PRF+noPL consumes the least amount of energy, and is fundamentally the best approach for building power-aware out-of-order microprocessors. On average, they consume 20% less energy than ARF+PL-style machine in the affected structures, which roughly translates to 6-7% of total chip power. Thus we believe that a PRF+noPL is the right approach for power-aware microprocessors. Additionally, PRF-style machine also simplify the implementation of out-of-order branch resolution, providing improved performance with comparable energy.

# 7. ACKNOWLEDGEMENTS

This research was supported in part by the National Science Foundation under grants CCR-0133437 and CCF-0429854, as well as grants and equipment donations from IBM and Intel.

# REFERENCES

[1] Borch, E., et al., Loose Loop Sink Chips, In *HPCA-2002*.
[2] Brooks, D., et al., Wattch: A Framework for Architectural-Level Power Analysis and Optimizations, In *ISCA-27*, 2000.
[3] Burger, D.C and Austin, T.M., The Simplescalar tool set, version 2.0, *Tech. Report CS-TR-97-1342*, UW-Madison, 1997.
[4] Colwell, R.P., and Steck, R.L., A 0.6um BiCMOS Processor with Dynamic Execution, *ISSCC Proceedings*, 1995.
[5] Compaq Computer Corporation, *Alpha 21264 Microprocessor Hardware Reference Manual*, July 1999.
[6] Diefendorff, K., K7 Challenges Intel, *Microprocessor Report*, vol. 12, no. 14, October 25, 1998.
[7] Gochman, S., et al., The Intel Pentium M Processor: Microarchitecture and Performance, *Intel Tech. Journal*, 2003.
[8] Gwennap, L., Intel's P6 Uses Decoupled Superscalar Design, *Microprocessor Report*, vol. 9, no. 2, February 1995, pp. 9-15.
[9] Hinton, G., et al., The Microarchitecture of the Pentium 4 Processor. *Intel Technical Journal*, no. Q1, February 2001.
[10] Keltcher, C.N., et al., The AMD Opteron Processor for Multiprocessor Servers, *IEEE Micro*, 2003.
[11] Keshava, J. and Pentkovski, V., Pentium III Processor Implementation Tradeoffs, *Intel Technical Journal*, no. Q2, 1999.
[12] Kessler, R.E., McLellan, E.J., and Webb, D.A., The Alpha 21264 Microprocessor Architecture, In *ICCD*, 1998.
[13] Matson, M., et al., Circuit Implementation of a 600 MHz Superscalar RISC Microprocessor, In *ICCD*, 1998.
[14] Papworth, D., Tuning the Pentium Pro Microarchitecture, *IEEE Micro*, April 1996, pp. 8-15.
[15] Peter, S., et al., The PowerPC 604 RISC Microprocessor. *IEEE Micro*, vol. 14, no. 5, October 1994, pp. 8-17.
[16] Sinharoy, B., et al., Power5 System Microarchitecture, *IBM Journal of Research and Development*, vol. 49, no. 4/5, 2005.
[17] Tendler, J.M., et al., Power4 System Microarchitecture, *IBM Journal of Research and Development*, vol. 46, no. 1, 2002.
[18] Wunderlich, R.E., et al., SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling, In *ISCA-30*, 2003
[19] Yeager, K.C., The MIPS R10000 Superscalar Microprocessor, In *IEEE Micro*, vol. 6, no. 2, 1996, pp. 28-40.