Reconfigurable Issue Logic for Microprocessor Power/Performance Throttling [*]

David Maze
Massachusetts Institute of Technology
Laboratory for Computer Science
dmaze@mit.edu

Edwin Olson
Massachusetts Institute of Technology
Laboratory for Computer Science
eolson@mit.edu

Andrew Menard
Massachusetts Institute of Technology
Laboratory for Computer Science
armenard@mit.edu

ABSTRACT
While most consumer microprocessors have used recent advances in computer architecture to improve performance, some applications, including portable computing, require lower power. We explore methods for programs to dynamically change the performance of a processor, allowing parts of programs that do not require the maximum performance of a processor to bypass complex issue logic for a low-power mode with decreased performance.

1. INTRODUCTION
Much of the thrust of recent computer architecture work has been in the direction of increased performance. Developments such as out-of-order issue and multiple-issue machines have led to performance increases in line with the prediction of Moore's Law. As performance has improved, software applications have in turn grown to take advantage of it. "Modern" programs will only run slowly, if at all, on the processors of 5 or 10 years ago.

Not all computer systems can afford the power requirements this improved performance brings. Laptop computers, for example, are limited by their battery life. While some research has gone into low-power architectures, these have largely gone into embedded systems where the expected performance is significantly lower.

For most interactive software applications, the vast majority of the computer's time is spent waiting for the user. Additionally, while some computations need the processor's full power to finish in reasonable time, others might only need to run fast enough to keep up with the user. It should then be possible for a program to tell the processor its performance requirements, and for the processor to then dynamically reconfigure itself to an appropriate mode to either maximize performance or minimize power usage.[2]

[*]Presented to 6.893, *Advanced VLSI Computer Architecture*, Spring 2000.

One major power drain is an out-of-order processor's issue logic. Every clock cycle, every instruction in the issue queue must be checked to see if it can be dispatched. This results in broadcasts of data on long bit lines. Some processors, such as the Alpha 21264, use queue compaction to simplify the search for a ready instruction, but this process requires even more energy. In the 21264, between 18 and 46 percent of the total power consumed by the processor goes to the issue logic.[7]

Some research has already been done in this direction. Much of it focuses on disabling sections of the processor's instruction or data cache to save power.[1] In a superscalar processor, issue and dispatch logic also takes up a significant fraction of the processor's total power usage. We therefore focus on techniquies to bypass the normal instruction issue logic to minimize power at the cost of performance.

2. METHODOLOGY
Our work targets the MIPS family of processors. MIPS historically has been a relatively straightforward architecture, and there are a number of tools that support simulations of it. Additionally, a number of real computer systems, including Silicon Graphics' workstations, have been built around a MIPS core.

A number of different tools support power and performance simulations of processors. One popular tool for performance simulations of is SimpleScalar.[4] SimpleScalar has an internal architecture, and performs performance simulations on this architecture. It has been used as the base for a number of other performance simulation projects. However, it does not provide any power usage statistics on the simulator processor.

The majority of our simulations were done with David Brooks' Wattch tool.[5] Watch is based on SimpleScalar. In addition to SimpleScalar's performance results, however, Wattch also performs power simulation, and returns the amount of power used by different parts of the processor core.

2.1 Power and Performance

Table 1: Issue Width and Power

Issue Width	1	2	4
IPC	0.57	0.64	1.39
Issue Power	1.75	2.06	6.12
Total Power	6.83	7.10	14.71
Issue Power %	25.59	29.04	41.61
IPC/Power	0.083	0.090	0.094
IPC/Issue Power	0.32	0.31	0.23

Table 2: Issue Window and Power

Issue Width	4	4	4	4
Window Size	32	16	8	4
IPC	1.41	1.39	1.23	0.89
Issue Power	7.47	6.12	4.75	3.14
Total Power	16.83	14.71	12.43	9.28
Issue Power %	44.35	41.61	38.24	33.83
IPC/Power	0.084	0.094	0.099	0.095
IPC/Issue Power	0.032	0.033	0.032	0.026

Running Wattch on its default model provides some interesting insight into the effects of varying issue width and instruction window width. Table 1 shows a processor's power consumption using issue widths of 1, 2, and 4 instructions. The "IPC/Power" line is most important: it indicates how much processing power is available for each unit of electrical power spent. These results suggest that, while the total power consumed by a single-issue processor is significantly lower than that consumed by a multiple-issue processor, the four-issue processor is more efficient in terms of power usage.[1]

Varying the size of the issue window provides more interesting results; these are summarized in Table 2. Again, the "IPC/Power" line indicates the amount of processing power available per unit of power consumed. While larger issue widths unambiguously lead to better IPC, there is not significant improvement in the performance of a 32-instruction window over the performance of a 16-instruction window. Note that an 8-instruction window gets IPC/Power of 0.099; this seems to be a near-optimal configuration in terms of power-efficient execution.

3. TECHNIQUES

Processors will have at least two modes: a high-performance mode, a low-power mode, and possibly a number of intermediate modes. Processors will have explicit instructions to switch from one mode to another. The switch may take significant time; consequently, we expect that applications or operating systems will only switch modes occasionally. It is nevertheless desirable to minimize the time and effort required to switch modes.

3.1 Side-by-side Cores

A simple technique for having a high-performance mode and a low-power mode is to have two completely separate processor cores. In high-performance mode, a superscalar core runs. For low power, a simple pipelined machine is placed alongside.

[1]The in-order processors (single- and double-issue) are poorly modelled. We believe the IPC figures for these processors to be too low, and their power consumption too high.

One disadvantage of this approach is the significant time required to switch modes. The switching involves completing the remainder of the instructions in either the pipeline or the superscalar issue window, and then transferring the contents of the register file from one processor to the other. There are additional complications involved with register renaming in the superscalar core. A straightforward approach is to ask the first processor to dump its register file to memory, switch to the second processor, and then restore it to memory, but this can be a very expensive operation.

3.2 Disabling Issue Logic

If the superscalar issue logic is disabled, a superscalar core can effectively become an in-order processor. The switching cost of this approach is low: we only need to wait for all outstanding instructions to finish. Aside from the disabled issue logic, all of the remaining parts of the processor are still useful.

3.3 Dynamically Varying Issue Width

The approaches discussed so far have taken an "all-or-nothing" approach: a processor is either in a high-performance mode or a low-power mode, with no middle ground. Power consumption varies significantly with a processor's issue width. By allowing varying issue widths, we can allow an intermediate level of performance with a very low switching cost.

4. REFERENCES

[1] D. H. Albonesi. Dynamic IPC/clock rate optimization. In *25th International Symposium on Computer Architecture*, pages 282–292. Association for Computing Machinery, June 1998.

[2] D. H. Albonesi. The inherent energy efficiency of complexity-adaptive processors. In *1998 Power-Driven Microarchitecture Workshop, held at the 25th International Symposium on Computer Architecture*, pages 107–112. Association for Computing Machinery, June 1998.

[3] L. Benini, A. Bogliolo, S. Cavallucci, and B. Ricco. Monitoring system activity for os-directed dynamic power management. In *International Symposium on Low Power Electronics and Design*, August 1998.

[4] D. Berger and T. M. Austin. The simplescalar tool set, version 2.0. Technical Report 1342, University of Wisconsin–Madison Computer Science Department, June 1997.

[5] D. Brooks, V. Tewari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *27th Annual International Symposium on Computer Architecture*, pages 83–94. Association for Computing Machinery, June 2000.

[6] R. Y. Chen and M. J. Irwin. An architectural level power simulator. In *25th International Symposium on Computer Architecture*. Association for Computing Machinery, June 1998.

[7] M. K. Gowan, L. L. Biro, and D. B. Jackson. Power considerations in the design of the Alpha 21264 microprocessor. In *35th Annual Conference on Design Automation*, pages 726–731. Association for Computing Machinery, June 1998.

[8] T. Pering, T. Burd, and R. Broderson. Dynamic voltage scaling and the design of a low-power microprocessor system. In *1998 Power-Driven Microarchitecture Workshop, held at the 25th International Symposium on Computer Architecture*, pages 107–112. Association for Computing Machinery, June 1998.

[9] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The design and use of simplepower: A cycle-accurate energy estimation tool. In *37th Design Automation Conference*, pages 340–345. Association for Computing Machinery, June 2000.