Issue Logic and Power/Performance Tradeoffs

Edwin Olson eolson@mit.edu, Andrew Menard armenard@mit.edu

***Abstract*—A growing need for computational power in mobile devices has spawned increased interest in low-power microprocessors. Some low-power applications require very high performance, such as real-time video decoding on Personal Digital Assistants. A growing body of work has examined how to provide this high performance when needed, while throttling performance so that power consumption can drop to very low levels when performance is not required. Observing that the issue logic in an out-of-order microprocessor consumes a significant amount of power, several groups have attempted to modify this part of the processor so that it can dynamically enter a low-power mode. We have revisited these topics and our work shows that simple approaches to modifying issue logic fail to reduce the average energy per instruction. We also look at the possibility of including a low-power single-issue processor on the same die as a high-performance multiple-issue processor. Swapping between these two processors allows a dynamic tradeoff between power and performance, but we show that this approach also struggles to reduce the average energy per instruction in the low-power mode.**

***Index terms*—Issue Window, Issue Logic, Out-of-Order, Low Power, Power/Performance Throttling**

I. Introduction

Much of the thrust of recent computer architecture work has been in search of increased performance. As transistor budgets have increased, more and more technologies from mainframes were incorporated in microprocessor designs. The product of this evolution was high performance microprocessors that sacrificed power consumption to maximize performance. With the emerging importance of low-power markets, these speed demons have been retrofitted to consume less power by incorporating clock gating, voltage scaling, and more recently, dynamic resizing of key architectural features such as the issue window.

Many existing techniques for reducing power are well established and extremely effective, including dynamically reconfiguring the cache[1] and voltage scaling[2]. Reducing the supply voltage of a microprocessor has a roughly linear effect on performance (due to weaker electric fields) but a squared effect on power dissipation (since power consumption is proportional to $\frac{1}{2}$*frequency*CV^2).

The authors are graduate students at the Massachusetts Institute of Technology, Cambridge, MA.

Voltage scaling must be taken into account when comparing two architectures for power efficiency. It is tempting to use a metric such as energy/instruction or the power delay product, however, one must also take into account the required performance level. A processor with seemingly poor energy/instruction characteristics that has more performance than required can be run at a lower voltage thus reducing performance and reducing the energy/instruction.

While voltage scaling is a very good way of providing additional power/performance modes, it has its limits. When operating voltage approaches the threshold voltage of the transistors, the performance of the transistors begins dropping off much faster than linearly. As threshold voltages are reduced, leakage currents increase, which, in turn, increases power consumption. The Semiconductor Industry Association predicts that in the year 2005, supply voltages for low power applications will be 0.9-1.2V [3], compared to a typical modern supply of 1.2V for a low-power processor like the Transmeta Crusoe. This implies a very limited ability for processors to exploit voltage scaling dynamically in order to scale power/performance. Clearly there is a need for additional power/performance throttling mechanisms. Other mechanisms for throttling performance, such as disabling portions of the cache, also have drawbacks; if the cache is made too small, the increasing miss rate will cause more power to be consumed by requiring main memory accesses.

It has been observed that one major power drain in modern out-of-order processors is the issue logic; every clock cycle, each instruction in the issue queue must be checked to see if it can be dispatched. Retired instructions broadcast the availability of new operands on long bit lines across the entire issue window. Some processors, such as the Alpha 21264, compact the issue queue in order to implement an oldest-first priority algorithm, and this process requires even more energy. In the 21264, between 18 and 46 percent of the total power of the processor is consumed by the issue logic [4].

In light of this, methods of scaling back the size of the issue window and the number of instructions issued each cycle have been proposed in order to reduce power consumption at the cost of reduced performance.[5,6] These methods are compatible with cache disabling and voltage scaling; for maximum reduction in power consumption the power management software could simultaneously reduce the

voltage to the lowest possible level, disable parts of the cache, and reduce the issue window size, or it could find intermediate power/performance points by doing only one or two of these optimizations, possibly by analyzing the type of code that is running to determine how much cache it needs or how much its performance would benefit from a large issue window. We also consider an alternate scheme of bypassing complex issue logic completely. We propose to do this by placing an in-order, single-issue core alongside the out-of-order multiple-issue core, with the OS able to swap between them, thus avoiding the complex out-of-order issue logic completely.

Several studies have shown that relatively simple modifications can allow an operating system to do performance throttling without spending an excessive amount of time profiling the code being executed, [7] and that relatively simple hardware structures can also monitor performance needs [5]. Thus, the power overhead required for dynamic throttling is minimal, which is necessary for it to be useful. Any overhead necessary to implement the strategies described in this paper is ignored from a power perspective; we assume that dynamic reconfigurations are perfectly efficient. Since we will demonstrate that these strategies do not reduce power, the omission of the overhead energy would only make the strategies even more unappealing.

II. METHODOLOGY

In order to conduct our study, we needed to measure the impact of changing architectural resource sizes, such as the number of slots in the issue window, on both power and performance. The SimpleScalar toolset provides detailed performance simulators [4]. SimpleScalar provides a performance simulator using a relatively unique microarchitecture built around a "Register Update Unit", an architectural resource combining the functions of the issue window and the register renaming unit. This is somewhat unfortunate, since it does not correlate well to actual chip designs.

However, the results we receive from these studies can still yield insight into the effects of scaling architectural features, and the relative results are still meaningful. Many other architectural studies have also used SimpleScalar, so our results can be directly compared with those. Future work may involve repeating our studies with a model more closely resembling commercially successful architectures.

SimpleScalar does not provide a mechanism for directly simulating power usage. However, several research groups have added power models to SimpleScalar, such as SimplePower [9], Wattch [10], and the Cai-Lim models [11]. Wattch's models are better suited to our study because its models are heavily parameterized and are therefore capable of reflecting various changes in configuration without needing to create SPICE models for each variation. We used version 1.02 of the Wattch power.c model.

Power estimation tools like Wattch and the Cai-Lim models have recently been the subject of considerable scrutiny [12]. Ghiasi, Grunwald and others have shown that not only are direct comparisons of energy measurements hopeless due to large differences in the energy predictions, but even relative comparisons often fail to agree. A key problem rests in the fact that an architectural description simply doesn't contain an adequate amount of information to properly estimate power, and even reasonable parameterized models quickly become unrealistic when the parameters are adjusted beyond a limited range. For example, the Wattch CAM model used for the RUU structure is a reasonable model for a 16 entry structure, but if the structure had 256 entries, it would have been implemented in a completely different way (multiple banks, perhaps). Therefore, we have limited our study to modifying parameters by relatively small factors, to minimize these effects.

We consider 4 issue and 8 issue processors with varying RUU sizes. The other characteristics of our processors are listed in Table 1. SimpleScalar allows many parameters to be adjusted, but we only changed a few. Table 1 is a list of non-default settings we used for the 4 and 8 issue architectures we studied.

Table 1.

	4 issue	**8 issue**
Decode Width	4	8
Commit Width	4	8
Load Store Queue Size	8	8
Integer ALUs	4	6
Integer Multipliers	1	2
FP ALUs	4	4
FP Mul/Div	1	2
Memory Ports	2	4

Our benchmarks are derived from the SpecInt95 suite. Due to the limited speed of the SimpleScalar simulator (about 90k instructions per second), it was impractical to run the entire suite, or even an entire single benchmark. Instead, as is the common practice in the simulator field, reduced input sets were used. These input sets take substantially less time to test, but still exercise the processor in ways similar to the official input sets. Therefore, the performance data we generated cannot be compared to actual SpecInt scores, but this is not an issue, as we are primarily interested in the relative performances of our various models.

Table 2.

Benchmark	**Input**
Li	Nqueens 6
Perl	test.in
compress95	5000 q 2131
mk88sim	ctl.dhry [50M instructions]

For all of these benchmarks, the kernel of the program, rather than initialization code, dominated the runtime. In addition, the simulator is completely deterministic, so there is no need to repeat simulations and average scores.

III. Determining Optimal RUU Capacity

Understanding the optimal size for the Register Update Unit is extremely important when determining what sizes to model. Several factors influence this optimal size. The goal of the RUU is to always have enough instructions ready to feed the available functional units. As the number of functional units increases, the size of the RUU should intuitively increase to provide more candidate instructions. However, due to data dependencies, it is often the case that the number of instructions that can be fetched is greater than the number that can be issued. In addition, we want the RUU to hold a certain "surplus" of instructions so that when an instruction miss occurs and fetch rate drops to zero, the functional units can be kept busy, but there is no reason to make the RUU unreasonably large, since once it reaches a certain size the inherent parallelism of the code will limit how many instructions can be issued, rather than any constraint on how many are in the window.

A. Bounds on RUU Usage

Our first experiment's goal was to determine an absolute upper bound on the size of the RUU. We configured SimpleScalar to use an extremely large RUU (128 entries) and made modifications to SimpleScalar to collect statistics on the size of the RUU every cycle. The resulting structure could hold enough instructions to keep the functional units busy for dozens of cycles, and is therefore excessive. However, it does provide an upper bound on the size of the RUU.

Figure 1.

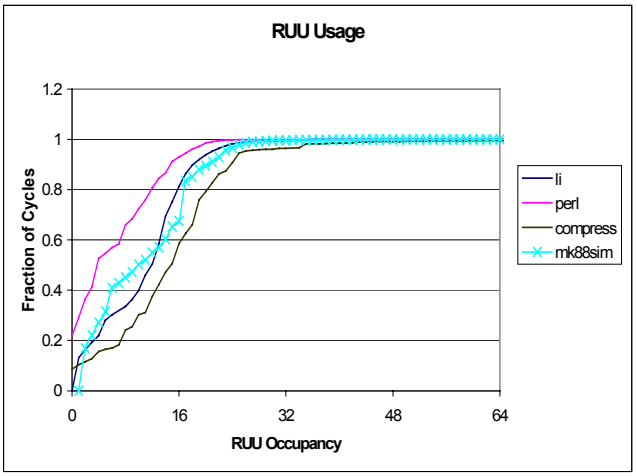


Figure 2.

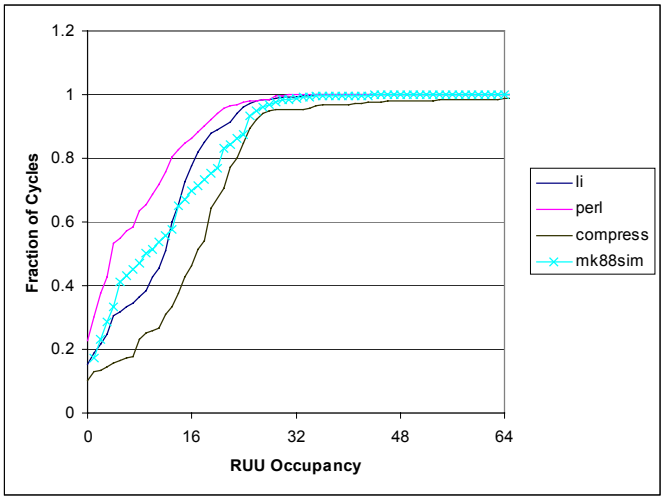
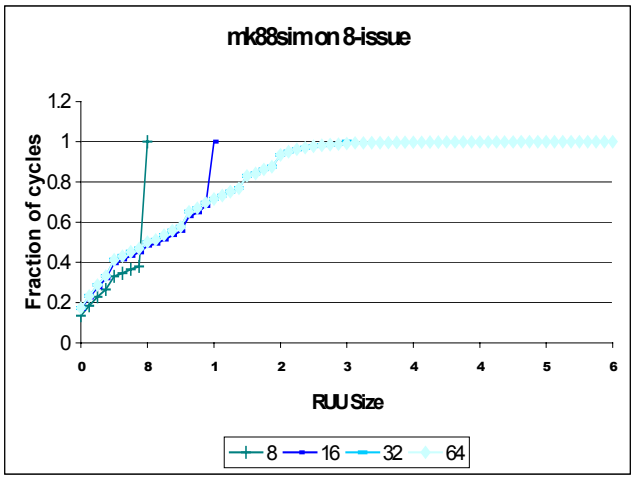


Figure 1 shows that for all four benchmarks, the RUU almost never contains more than 32 instructions at either issue width. Thus, making an RUU any larger than 32 would serve no function; the entries would be empty almost all the time. Figure 2 shows the results for the 8-issue case; they are almost. However, it still appears that 32 entries is sufficient.

When the RUU's physical size is bounded, the RUU usage closely mirrors the unlimited case, except that the RUU will "saturate". In figure 3, we show the cumulative occupancy statistics for a large RUU and a 16 entry RUU. We see that a 16 entry RUU has almost exactly the same occupancy characteristics when occupancy is between 0 and 15. The 16 entry RUU is fully occupied about as often as the unlimited RUU has 16 or more entries. This is as would be expected, and though Figure 3 uses the mk88sim benchmark, the other benchmarks demonstrate the same behavior, as does the 4-issue machine.

Figure 3.



B. IPC vs. RUU size

The important question now is: if the RUU capacity is limited beyond the ideal case, what happens to performance? We measured performance in terms of Instructions Per Cycle (IPC), since we cannot accurately determine changes in clock period from within SimpleScalar. If the length of all the pipeline stages are well-balanced, it's likely that dynamically reducing the size (and therefore delay) of the issue window would probably not allow the whole processor to be clocked faster.

Figure 4.

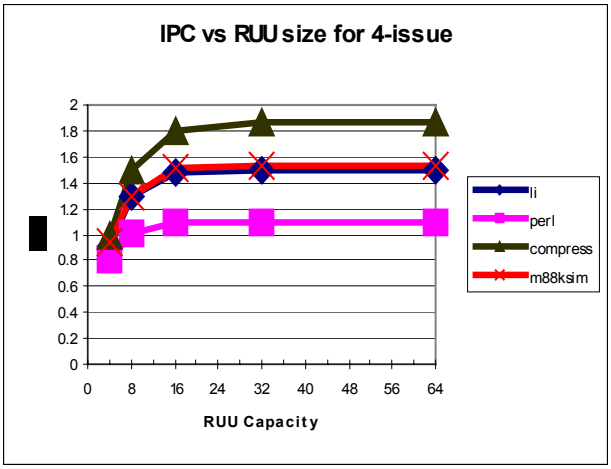


Figure 5.

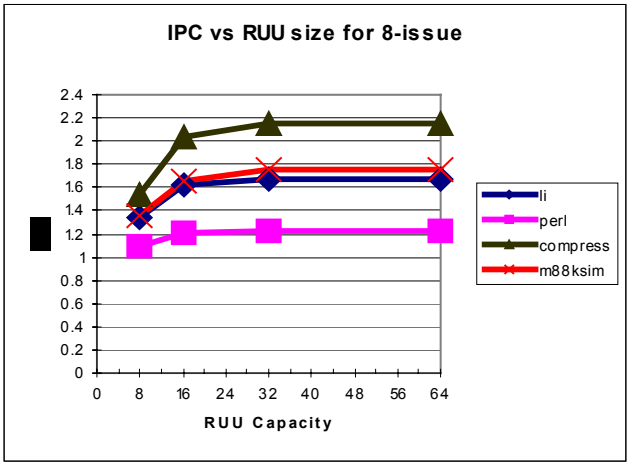


Figure 4 shows the performance of the processor, in terms of IPC, versus the capacity of the RUU. We notice immediately that the performance of the processor for compress and perl is *very* similar for RUU capacities of 16 and 32 for a 4-issue processor. There's a small increase for li. As we expected, there is almost no benefit in scaling the RUU beyond 32.

If we consider an 8-issue machine, we would expect the performance of the processor to drop off more rapidly than the 4-issue with decreasing RUU capacity. This is because the RUU could be depleted (potentially) twice as quickly, and the processor is therefore more likely to be unable to keep its functional units busy. We see precisely this behavior in Figure 5; there is a noticeable performance difference for both li and perl between RUU capacities of 16 and 32.

Some research groups have proposed dynamically varying the issue window capacity [5]. It is obvious that a parameterized model of an RUU is likely to predict substantially greater power consumption for a 32-entry RUU than a 16-entry RUU. We must resist the temptation to declare that throttling the RUU capacity between 16 and 32 is an effective way of throttling power/performance; there is very little performance difference between RUUs in that range, but there is a significant difference in power

consumption. A power-conscious architect is unlikely to make the RUU so much larger for such a miniscule return.

C. *Relationship between energy and RUU size*

However, an interesting question still remains. What happens to energy per instruction statistics as we decrease the RUU well into the region of decreased performance? It might be a good idea to allow a processor to dynamically decrease its RUU size, for example from 16 to 8, if the decrease in power more than offsets the decrease in performance.

Using the Wattch tool, we measured the power consumption of the processors and calculated the average energy per instruction assuming realistic clock gating (Wattch's cc3 models).

Table 3.

Structure	4x4	4x8	4x16	4x32	4x64
Energy/Inst (li)	15.8	13.0	**11.8**	12.8	14.1
Energy/Inst (perl)	16.5	14.3	**13.6**	14.7	16.1
Energy/inst (compress)	14.4	11.5	**10.6**	11.3	12.5
Energy/inst (m88ksim)	14.7	12.2	**11.4**	12.4	13.6

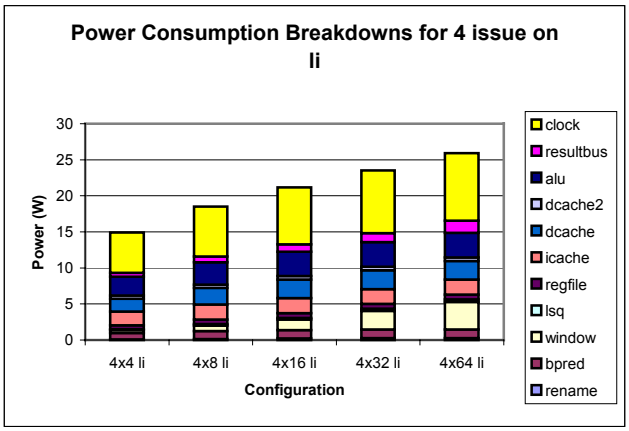
Table 3 shows the average energy per instruction for each benchmark, for various RUU capacities of a 4-issue processor. We already expected the 4x32 and 4x64 configurations to be suboptimal, since the RUU is essentially oversized. It's interesting, however, that the cost of executing instructions actually increases when the RUU is shrunk below 16 entries. While the power consumption of the issue logic is going down with decreasing RUU capacity, the performance is dropping super-linearly, due to the smaller amount of parallelism that the processor is finding.

We can also see that we're spending more energy per instruction on codes with less inherent parallelism (perl in particular). This makes sense since there are a lot of hardware resources in an out-of-order superscalar processor looking for parallelism to exploit, but there's simply very little parallelism to be found. This overhead cost is being amortized over very few issued instructions every cycle, and thus the average energy per instruction is higher.

We'll also note that we don't trust the power numbers for the extreme configurations of RUU (x4 and x64) since they comprise a significant factor of deviation from Wattch's baseline capacity.

The breakdown of power consumption is shown in figure 6. The patterns of power consumption are similar for all four benchmarks, so we show only the li case. One component that is consuming conspicuously more power as the RUU size is increased is the RUU itself (denoted as 'window'). Somewhat unexpected are the increases in energy in other areas of the chip. As discussed earlier, when the RUU size is increased, the total number of instructions (committed + speculated) executed increases 12-23%, depending on the benchmark. This causes increased activity in almost all of the major functional blocks. In addition to the window energy, we see significant increases in the clocking energy, the load store queue, and the result bus.

Figure 6.



In Table 4, we have energy per instruction statistics for an 8 issue processor. We see very similar trends as in the 4-issue processor. As with the 4-issue case, we observe that the 16-entry RUU is the minimum energy per instruction point.

Table 4.

Structure	8x8	8x16	8x32	8x64
Energy/Inst (li)	13.8	**12.5**	13.4	14.9
Energy/Inst (perl)	15.1	**14.7**	15.8	17.6
Energy/inst (compress)	12.4	**11.4**	11.9	13.3
Energy/inst (mk88sim	13.0	**12.1**	12.9	14.4

IV. Other Low-Power Modifications to Complex Processors

It seems as though scaling a processor's issue window will not provide the power/performance throttling we would like; the optimal energy per instruction point is near the optimal performance, so there is little to be gained from scaling it. There are many potential functional units that can be targeted for energy reduction, but other difficulties arise.

We see from Figure 6 that the register file consumes a significant percentage of power. SimpleScalar's RUU structure works in its favor for minimizing the complexity

of the register file by incorporating the renamed registers within the issue window and maintaining a separate (and smaller) architectural register file, whereas in a mainstream design the register file often contains both the renaming and architectural registers. In the latter case, the register file is both physically larger and may have additional ports, consuming even more power. A low-power mode might use a smaller set of renaming registers, or turn off register renaming completely. We do not consider any of these additional optimizations here.

V. Lobotomizing an Out-of-Order Processor

Our group considered several mechanisms for dynamically "lobotomizing" an out-of-order processor in order to provide new power/performance points. Our initial approaches mirrored those of other groups—dynamically resizing issue logic, but our study of the effects of RUU sizing discussed previously in this paper made this seem problematic.

Our second idea was to disable most of the logic accompanying the out-of-order issue logic, essentially causing instructions to be issued in-order. We modeled this in SimpleScalar by disabling out of order execution and speculative execution, then reducing the size of the RUU to one. We expected extremely poor performance based on our previous experiments in RUU sizing. We measured an effective IPC of 0.57. The power numbers returned by Wattch are not reported here since their accuracy with the oddly-sized models cannot be relied upon. The major cause of the poor performance is the very high latency of an out-of-order processor (compared to a simple pipelined machine) that causes many stalls when dependent series of instructions are run. An out-of-order machine spends a lot of time in order to find and exploit parallelism, and dramatically reducing the RUU's capacity causes most of this work to be wasted since it eliminates the possibility of having many instructions executing simultaneously. Moreover, the out of order machine still consumes excess power in structures designed to support out-of-order execution, for example the many-ported register file.

This led us to another idea-- completely bypassing the complicated issue logic of the microprocessor. If the renaming logic and issue window were bypassed completely—if a single instruction was passed immediately from the instruction fetch stage to the register file read stage—the latency of instructions would be substantially shorter and the performance would increase substantially. It would be tempting to use a banked register file as well, so that when register renaming was turned off, access to the register file would all come from a smaller "architectural register" register file. However, unless a completely separate register file was used for the low-power mode of operation, each bank of the register file would likely have far more ports than would be necessary for a single-issue processor, and this would put a bound on the amount of power savings that could be achieved. It would be worthwhile to build an accurate model of this and simulate it in detail in the future.

VI. Using a completely separate core

Since simply scaling back the size of the issue window does not seem to be an obvious win, we also considered eliminating it altogether. A large, complex, out-of-order core could be used when high performance was required, or a small, simple, in-order pipelined core could be used when low-power operation was needed. Our intuition suggested that a simpler core would likely have much lower energy/instruction given the same technology, given its lack of issue and renaming logic, its smaller register file, and so on. Also, since the cores are completely separate, each can be optimized separately; the in-order one being optimized for minimum energy use and the out-of-order one being more aggressively targeted at performance. If the on-die caches and other circuits could be used for both cores, the overhead in die area for a small in-order core would be very small, since it would require fewer functional units and much simpler.

We wished to get the most accurate data possible, and the Wattch and SimpleScalar models are not designed to support power modeling of in-order architectures, so rather than using them, we opted to conduct a survey of commercially available processors, looking for processor families where there is both an in-order and an out-of-order implementation of the same ISA in the same technology, and of processors that support some other form of power/performance tradeoff, for a comparison.

IBM's PowerPC line includes the model 440 CPU, a dual-issue, out-of-order machine, and the 405 CPU, a single-issue, 5-stage pipeline machine, both implemented in the same .18 micron copper process [15,16]. The 440, operating at 550MHz, consumes approximately 1.0W of power, and performs at 1000mips on the Dhrystone 2.1 benchmark, while the 405 operating at 266MHz consumes approximately 0.5W of power while performing 375mips on the same benchmark. Thus, the energy used per instruction on the 440 is actually lower than that of the 405. This is a very disappointing result; the faster processor is actually using less energy per instruction, so clearly you would benefit more from an approach like voltage scaling to reduce total energy used across a calculation, or even just use the faster processor until the calculation is finished and then put it into a sleep mode, both of which also avoid the significant area overhead of the dual processor approach.

A comparison of AMD's K6-2 line of 6-issue out-of-order processors and its AM5x86 line of in-order processors is slightly more hopeful; the 5x86 consumes slightly less power per instruction despite being implemented in an older technology. [15,16] However, a pair of recent processors from Intel and Transmeta demonstrate much more impressive results through voltage scaling and also demonstrate that cycling in and out of sleep mode can give a

linear power/performance tradeoff all the way down to nearly zero power.

The Intel Pentium III mobile versions utilize voltage scaling from 1.6V to 1.3V to achieve a more than 50% reduction in power consumption while still achieving 70% of the performance. Unfortunately, they only support a single high performance mode and a single low-power mode, with no intermediate modes. Intel's Xscale line of StrongARM-compatable chips uses voltage scaling at a much finer granularity to go from consuming 450mW at 800MHz to only 40mW at 150MHz[17], for a significantly better than linear tradeoff. The Transmeta corporation's Crusoe line of chips use dynamic voltage and frequency scaling from 600 MHz at 1.6V to 300MHz at 1.2V, achieving a significantly better than linear drop in power consumption for a linear drop in speed [18]. It also supports a sleep mode which consumes virtually no power, and which it can rapidly cycle into and out of, which allows it to continue throttling back performance for power savings all the way down to zero performance at almost zero power. This demonstrates that in mainstream processors with today's technology, voltage scaling and sleep modes are still the best approach to power/performance throttling; the methods we have proposed don't achieve any better results, and have significant complexity costs.

VII. Conclusions

While the bulk of recent computer architecture research has focused on increasing performance, many modern applications require both high performance and low power. Techniques such as voltage scaling and clock gating are well established as ways to reduce power consumption without adversely affecting performance. In this paper, we considered two techniques for switching between a high-performance mode and a lower-power, low-performance mode. Dynamically changing the processor's issue width seemed promising initially but yielded poor results when considering the power consumed per instruction. Installing a small in-order core alongside the out-of-order core also appeared promising, but making comparisons of energy usage between modern processors shows that this too fails to yield an overall benefit in terms of power used.

On a high-performance processor such as the Digital Alpha 21264, between 18 and 46 percent of the total power consumed by the processor goes to the issue logic. This suggests bypassing or reducing the issue logic as a route to minimizing power consumption if performance is not a concern. Using Wattch and SimpleScalar, we first determined that there is a maximum size for the SimpleScalar register update unit, or RUU. Increasing the size of this structure, roughly analagous to a real microprocessor's register renaming logic and issue window, yields no performance gains if this maximum size is exceeded. We then examined the power used by the RUU, and found that the performance of the processor drops faster than its power requirements when the RUU size is decreased. Thus, the power per instruction has a minimum, and we determined that the optimal size is at 16 for the 4 and 8 issue version we studied, at the point where performance gains begin to drop off rapidly.

Because of this result, changing the size of the issue window on an actual microprocessor would not be beneficial in terms of power consumed. Real-time applications require a certain number of instructions to be executed in a particular time frame. Our results show that the minimal power is used by doing this computation using the full power of the microprocessor, and then switching to a very low power sleep mode in which no instructions are executed for the remainder of the time period. Attempting to change the issue width would result in more power being used per instruction; since in this scenario the number of instructions executed per unit time is constant, this results in more power used per unit time. This approach does require rapid switching into and out of sleep mode, and a near zero expenditure of energy in the sleep mode, but modern processors have demonstrated that this is possible.

Given these discouraging results, we also considered the possibility of including a completely separate in-order core on the die of a larger microprocessor. We hoped that the in-order core would be small enough to fit into a modern superscalar machine without significantly impacting the layout. While the in-order machine would have much lower performance, it ideally would have had a much lower power usage since it lacked the expensive register renaming and instruction reordering logic present on the out-of-order machine.

A survey of real processors suggests that this unfortunately is not the case; the power savings from the in-order core would be minimal at best, and in some cases the out of order core used less energy. Other modern processors make good use of other approaches which do a better job, and research papers have proposed several additional approaches which also appear to be more promising.

In this paper, we have shown that neither modifying a processor's issue width nor adding a separate in-order core offers a possibility for a low-power mode as an alternative to a high-performance mode. We hope to be able to use better simulation tools to examine some of the options presented here, including completely bypassing the issue logic, using a separate register file for in-order execution, and including a separate in-order core on chip. However, current widely-used techniques, such as voltage scaling and clock gating, appear to offer the best power savings currently available for low-power applications.

VIII. References

[1] David H. Albonesi, "Dynamic IPC/Clock Rate Optimization," 25th International Symposium on Computer Architecture, 282--292, June, 1998

[2] T. Pering and T. Burd and R. Broderson, “Dynamic Voltage Scaling and the Design of a Low-Power Microprocessor System,” 1998 Power-Driven Microarchitecture Workshop, held at the 25th International Symposium on Computer Architecture, 107--112, June, 1998

[3] Silicon Industry Association, “International Technology Roadmap for Semiconductors, 1999 Edition”, http://public.irirs.net/files/1999_SIA_Roadmap/Home.htm

[4] Michael K. Gowan and Larry L. Biro and Daniel B. Jackson, “Power considerations in the design of the {Alpha} 21264 microprocessor, “35th Annual Conference on Design Automation”, 726--731, June, 1998

[5] Roberto Maro, Yu Bai, and R. Iris Bahar, “Dynamically Reconfiguring Processor Resources to Reduce Power Consumption in High-Performance Processors”

[6] Alper Buyutosunoglu, Stanley Schuster, David Brooks, Pradip Bose, Peter Cook, and David Albonesi, “An Adaptive Issue Queue for Reduced Power at High Performance”.

[7] L. Benini and A. Bogliolo and S. Cavallucci and B. Ricco, “Monitoring System Activity for OS-Directed Dynamic Power Management,” International Symposium on Low Power Electronics and Design, August, 1998

[8] Doug Berger and Todd M. Austin, “The SimpleScalar Tool Set, Version 2.0,” June, 1997

[9] W. Ye and N. Vijaykrishnan and M. Kandemir and M. J. Irwin, “The Design and Use of SimplePower: A Cycle-Accurate Energy Estimation Tool,” 37th Design Automation Conference, 340--345, June 2000

[10] David Brooks and Vivek Tewari and Margaret Martonosi, “Wattch: a framework for architectural-level power analysis and optimizations,” 27th Annual International Symposium on Computer Architecture, 83--94, June, 2000

[11] G. Cai and C. H. Lim, “Architectural level power/performance optimization and dynamic power estimation,” MICRO32, November, 1999

[12] Soraya Ghiasi and Dirk Grunwald, “A Comparison of Two Architectural Power Models,” Ninth International Conference on Architectural Support for Programming Languages and Operating Systems, November, 2000

[13] IBM Product Datasheet for the PowerPC 440 Core

[14] IBM Product Datasheet for the PowerPC 405 Core

[15] Enhanced AM5x86 Processor Family Datasheet http://www.amd.com/products/cpg/techdocs/datasheets/19715.pdf

[16] K6-2 Family Datasheet http://www.amd.com/products/cpg/techdocs/datasheets/18522.pdf

[17] Intel Xscale Microarchitecture Technical Summary http://developer.intel.com/design/intelxscale/XScaleDatasheet4.htm

[18] Marc Fleishmann, “Crusoe Power Management,” HotChips 12

[19]

[20] David H. Albonesi, “The Inherent Energy Efficiency of Complexity-Adaptive Processors,” 1998 Power-Driven Microarchitecture Workshop, held at the 25th International Symposium on Computer Architecture, 107--112}, June, 1998

[21] R. Y. Chen and M. J. Irwin, “An Architectural Level Power Simulator,” 25th International Symposium on Computer Architecture,” June, 1998

[22] Vivek Tiwari and Deo Singh and Suresh Rajgopal and Gaurav Mehta and Rakes Patel and Franklin Baez, “Reducing Power in High-performance Microprocessors,” 35th Annual Conference on Design Automation, June, 1998