Investigation of the Power–Performance Trade-off in High-Performance Processors

Todd Hoffenberg Kyle Wheeler

August 14, 2003

Abstract

Demand for devices that are power-conscious is obvious and growing, and the need for scaling back power dissipation for heat concerns is pressing. However, power does not linearly correspond to performance, and a balance can be achieved. Several design-space changes are considered and evaluated using sim-wattch. In cache design, an effective level 1 cache is an absolute necessity. Leakage power in level 2 cache (and lower levels) can be drastically reduced by transitioning unused blocks to a low-power state that preserves cache elements; dividing a level 2 cache into superblocks and introducing a buffer of superblocks to keep active can drastically cut leakage power at minimal performance cost. For a baseline configuration, issue width, decode width and RUU size are varied and are found to correspond directly to power consumption. Several branch prediction strategies are tested, showing bimodal to be the most useful, and 2-layer to be the most interesting.

1 Introduction

Demand for portable devices that are ever more capable and have ever more battery life continues to increase, while at the same time heat dissipation and the upward trend of CPU heat production (and thus power consumption) combine to make reducing power consumption one of the most pressing problems in microprocessor architecture today. There are many approaches to the problem which include things as exotic as redesigning the architecture to allow finer-grained pipelines to be turned off to conserve power, and things as approachable as reevaluating the utility of existing functional units and design decisions based on their performance to power consumption ratio. It is easy to take for granted that more power means better performance, and that less power means reduced performance, however power does not linearly lead to performance. Thus, in many situations, it may be possible to find a design-space "sweet spot," where the power to speed trade-off is the best—in other words, in what configuration do we get the most bang for the least buck. This paper is a step in that direction, using the Wattch simulator and power consumption model to evaluate several of the design decisions and functional unit sizes with regard to power consumption.

Some of the largest power consumers on a modern microprocessor die are the cache configuration and the branch predictor. However, while the two features use a lot of power, at the same time they are saving power by keeping the processor from doing unnecessary work. To test our ideas and to explore the power consumption relations, we used sim-wattch [4].

Sim-wattch is an extension to SimpleScalar that tracks power usage as well as the other performance metrics that SimpleScalar normally measures. Sim-wattch was run on MacOS X 10.2 and on RedHat Linux, kernel version 2.4.18-14 with minor changes to the source code to get it to compile. The wattch sim-outorder.c and cache.c were modified to support superblock switching simulations discussed herein.

2 Caches

At a minimum, caches dominate the chip area of a modern microprocessor, even those targeted for embedded systems. In the past, larger cache sizes did not mean a large associated increase in power. This is because most of the power consumed in a cache has been due to transistor switching, and only one cache block is addressed for each access. Consequently, accessing lower level caches naturally increases power consumption, so a good level 1 cache will provide the most obvious reduction in power. For the sake of minimizing access times, level 1 caches are typically direct mapped. However, level 2 caches are more often set-associative. For the sake of speed, all possible elements in a set are compared to the address tag for determining a cache hit in parallel. Therefore, increasing associativity will increase the amount of compare operations per access and correspondingly increase power.

In modern processors, most of the power consumed by a cache is in the form of keeping parasitic leakage involved in keeping volatile caches powered. This leakage power consumption is the prominent power drain at and below the 0.10 micron barrier [6]. Reducing the level 2 cache size should, therefore, reduce power consumption by the chip. This must be balanced against I/O speed and I/O power overhead to fully consider an appropriate configuration.

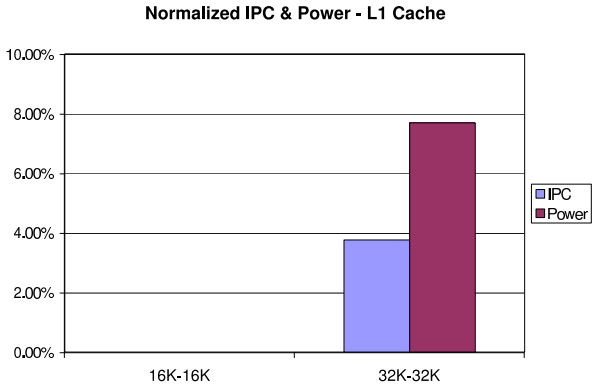

Figure 1: Performance and Power Consumption of Level 1 Cache Configurations

Additionally, work in reducing the leakage power has expanded by providing circuitry that actively holds state in selective cache blocks by selecting a lower rail voltage supply [13] [14]. Several strategies have been proposed by [11] to take advantage of this, but we propose a different technique that yields better results.

2.1 Level 1 Cache Configuration

Two different level 1 cache configurations are considered. These are the 32k–32k data/instruction cache and the 16k–16k data/instruction cache configurations. Figure 1 illustrates the relative performance and power of each as reported by sim-wattch [4].

The performance metric (IPC) and power difference are very small between the two caches. The 64k total level 1 cache size configuration is recommended because it avoids going to lower cache hierarchies more often, and other proposed power reduction strategies degrade level 2 cache performance. This setup is assumed for subsequent simulations.

2.2 Level 2 Cache Associativity

The performance and power consumption relative to level 2 cache associativity is highly correlated. This is due to the large speed and

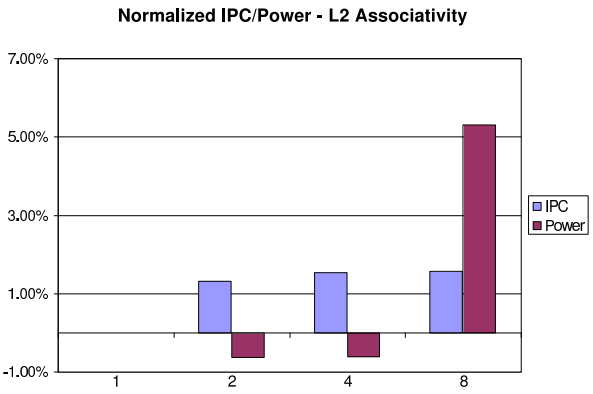

Figure 2: Performance and Power Consumption of Level 2 Cache Associativity Configurations

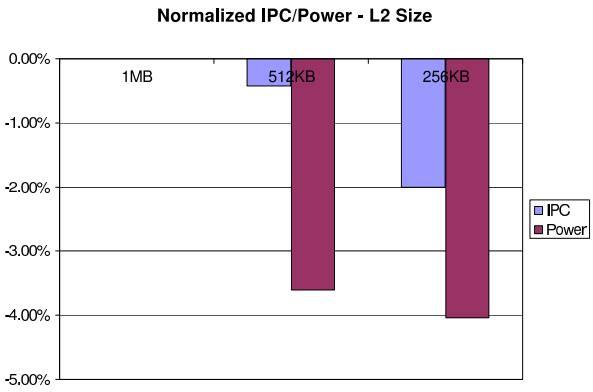

Figure 3: Performance and Power Consumption of Level 2 Cache Sizes

lower penalties associated with I/O and also due to the variance on the number of cache entries relative to cache associativity. Figure 2 illustrates the relative performance and power of different associativities.

The optimal associativity for this 1MB L2 cache with a 64-byte block size seems to be 4-way associative. This is assumed for subsequent simulations.

2.3 Level 2 Cache Size

To test the impact of leakage power, L2 cache size was varied. Smaller caches create more misses, so the impact of I/O power consumption must also be considered. Figure 3 illustrates the relative performance and power of different level 2 cache sizes.

The optimal performing cache is a 1MB cache. Leakage power is not much of a factor here, but this is likely due to the default setup of sim-wattch. As will be shown later, leakage power can be drastically reduced, so a 1MB cache is power-performance optimal and is assumed for subsequent simulations.

2.4 Leakage Power

2.4.1 Prior Work

In [17], gating Vdd is proposed as a mehtod of reducing cache leakage power. In this method, unused cache elements can be completely shut down by switching off the rail voltage supply. This technique was applied at the block level in [8]. However, so-called gated-Vdd cache configurations necessarily lose the data stored in a cache block when it is shut off. The determination of what data can be affordably eliminated is difficult, and the problem becomes more complex when highly associative caches are considered. The penalty for switching off the wrong cache block is severe, requireing an access to lower memory hierarchies. A solution to that problem has been proposed in [13] [14]. By using a small circuit, a lower supply voltage can be used for selected inactive blocks. This voltage is just enough to preserve cache state when it is not being accessed. However, an access to a “sleeping” cache block would require awakening the cache block. The circuit presented in [11] provides 120mv to sleeping cache blocks, but it requires a large time to allow settling of the virtual ground when putting cache blocks in a state-preserving state (some 50 cycles). This is further complicated by a minimum rise time for supplying

power to a sleeping cell of about 1V/10ns [14]. One additional consideration is that lower-voltage cells are more easily affected by alpha particle radiation [12]. Additional error correction bits assure correct operation, but the overhead in using this error correction, the frequency of incorrect readings, and the penalties associated with a bad cache read are not yet analyzed.

In [11], a speculative shut-down of cache blocks that have been moved from level 2 cache to level 1 cache performs well, saving more than 30% leakage power. Combining this with a Decay-II cache presented in [8] where cache blocks that have not been accessed after a certain time are plced in the low leakage state introduces a 75% savings in leakage power. These strategies are good, but there is potential or more savings.

2.5 Superblock Switching

Logically, a cache can be divided by the highest order address bits into partitions dubbed superblocks. If the number of superblocks is small (initially, we will assume four), then subsequent accesses to a cache are assumed to be in the same superblock. Thus, we can naively predict the superblock of the next cache access as simply the superblock of the previous cache access. This allows the remaining superblocks to be in the inactive state, saving a large amount of recoverable leakage power. These sequential type access patterns are typical of a level 1 cache, but this method is generally not applicable to this cache for three reasons. First, level 1 caches are small relative to lower levels and consequently consume less leakage power. Second, the overhead in determining a superblock is small, but for deep pipelined processors, it may result in an unacceptable penalty to level 1 access time. Finally, even if that overhead can be eliminated, a one cycle penalty for activating a new superblock in the event of a misprediction can seriously cripple performance.

Thus, utilizing superblock sectioning is limited to level 2 caches for our analysis. In the first scheme, superblocks are only awakened when they are accessed from a sleep state. The remaining superblocks are then put into a power-saving sleep state. For sequential cache accesses that span superblocks (i.e. large array accesses), it may be useful to preemptively activate superblocks before they are accessed. This can be done by first assuming a larger number of superblocks (say 8 or more) and then activating a fixed number of neighboring superblocks in either direction. However, this scheme is generally unsuccessful in surpassing the nominal 4-superblock approach. This supports the idea that level 2 cache accesses are more or less random at the superblock level. This is likely due to the fact that most sequential accesses are going to be covered by higher level cache space.

It can be reasonably assumed, however, that a given executing program is going to store data in the address space of only a few different superblocks. Which superblocks those actually are may be random, but repetitive access to superblocks may occur. Introducing a superblock buffer, a small cache of recently accessed superblock tags, can solve this problem. This provides additional superblocks to keep active, but it implies smaller superblocks and virtually eliminates the benefit of considering superblock neighbors. A fully-associative superblock buffer with an LRU replacement scheme would require the smallest buffer; but, for hardware simplicity sake, a direct-mapped buffer is used. This may also serve to minimize any additional latency imposed by the buffer. This additional latency ought to be nonexistent because the buffer must only be updated on a mispredicted superblock access where a cycle penalty is already assumed, and the buffer update can be done in parallel with the already

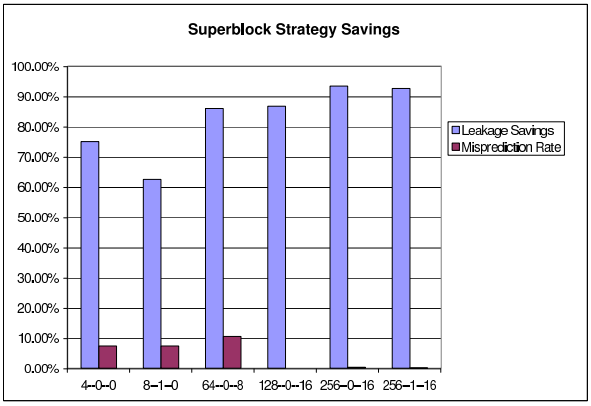

Figure 4: Leakage power savings and performance for various superblock switching strategies.

slow cache access.

Figure 4 presents the results for several different configurations of the number of superblocks, the number of superblock neighbors to either side of the previous access that are active, and the size of the superblock buffer (X–Y–Z, respectively in the figure elements). The graph contains the percentage recoverable savings in leakage power, which is simply the number of sleeping cache superblocks divided by the total number of superblocks. It also graphs the superblock misprediction percentage, which is the percentage of level 2 cache accesses that require waking up a cache superblock.

The simulated misprediction numbers are an average over the spec95 integer benchmarks. The best performance is by the 128-superblock 16-element buffer strategy, where all of the mispredictions are in fact compulsory and serve only to fill the buffer. Even more savings in power are possible by moving to a 256-superblock scheme with a 16-element buffer. In all benchmarks besides the JPEG benchmark, the misses are entirely compulsory. Even in the JPEG benchmark, the non-compulsory misses are very small. By activating 1 neighbor to either side of the last accessed superblock, nearly all non-compulsory misses are eliminated in the JPEG benchmark at only a slight increase in leakage power consumption.

3 Branch Prediction

When examining nearly any nontrivial trace of power usage for modern processors, one of the major power consumers is the branch predictor, which consumes around 22% of the power, with some small (less than one percent) variance depending on the prediction method used, the power conservation of the rest of the chip, and the specific load. Depending on the method of branch prediction used, however, the power used can vary significantly. This agrees with common sense, that complicated branch prediction uses a fairly sizable cache, and does a fair bit of computation for every conditional statement—and in typical code, conditional branches are very common.

3.1 Bimodal

While bimodal branch prediction uses a significant portion of power, it's CPI-per-power unit ratio remained fairly constant as the size of the bimodal buffer was varied (see Figure 6). This would seem to indicate that varying the size of the bimodal buffer has a very direct translation to performance, and that if this branch predictor is used, determining how much power can be saved is a merely a question of how much power to spend for performance, and that there is no obvious optimum balance.

3.2 2-Layer

Interestingly, 2-level branch prediction has rather unintuitive behavior, which is that the more area of silicon that is devoted to it, either by increasing the size of it's L1 layer or it's L2

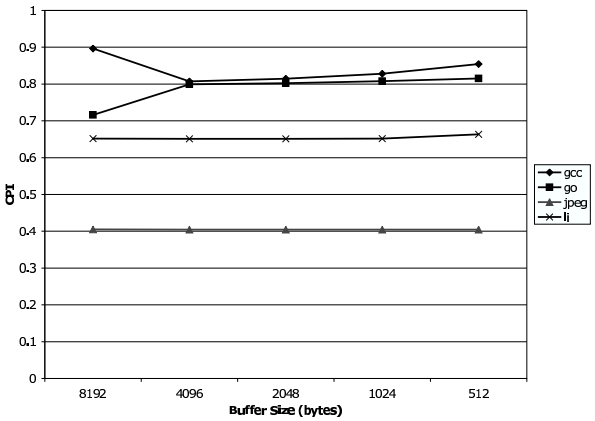

Figure 5: Bimodal CPI vs. Buffer Size

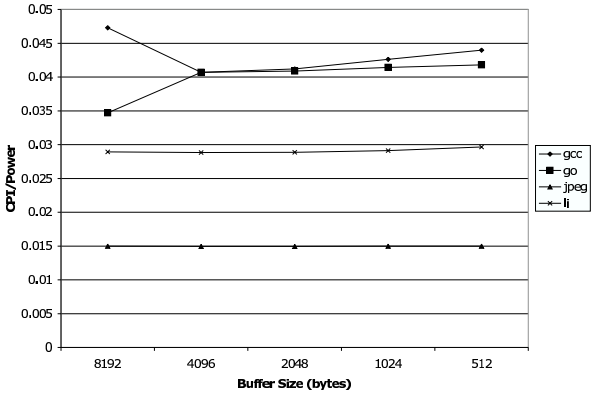

Figure 6: Bimodal CPI/Power vs. Buffer Size

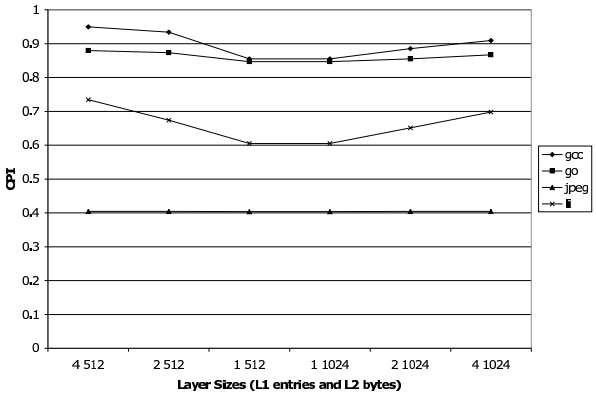

Figure 7: 2-Layer Branch Predictor CPI by Benchmark for Resource Variance

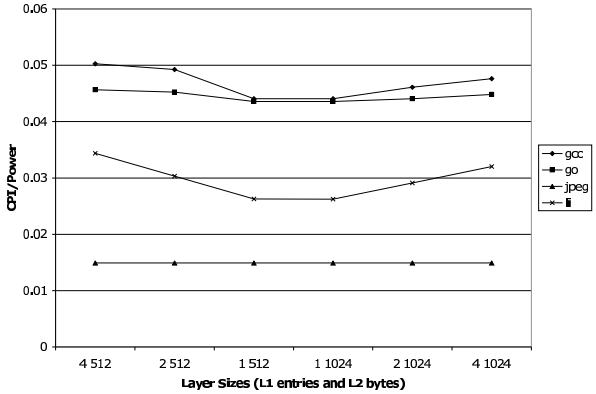

Figure 8: 2-Layer Branch Predictor CPI/Power by Benchmark for Resource Variance

layer, the less it helps the CPI (see Figure 7). Again counter-intuitively, the less silicon that is devoted to the 2-level branch predictor, the more power it uses (see Figure 8). One can only speculate why the power increases with the decrease in area; however, the useful conclusion one can draw from the data gathered is that the size of the second layer is not as important to the performance of the predictor (in terms of CPI per power) as the size of the first layer. Thus, with a smaller second layer, the CPI per power unit increases more per increase of the size of the first layer than with a larger second layer.

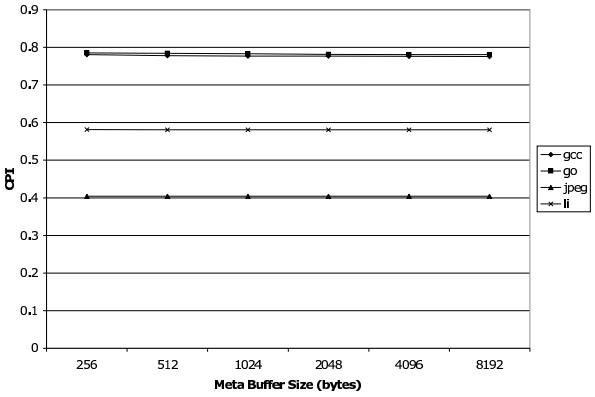

Figure 9: Combination Branch Predictor CPI by Benchmark for Meta Table Variance

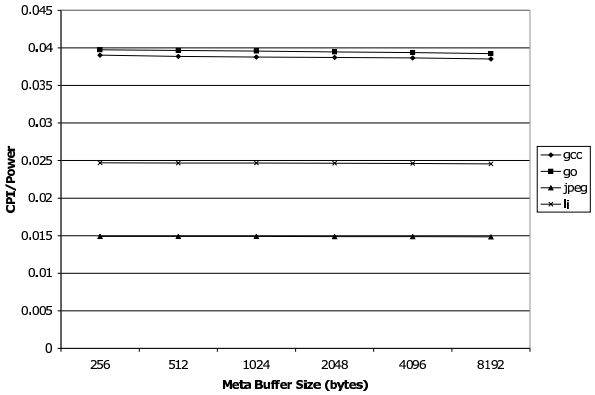

Figure 10: Combination Branch Predictor CPI/Power by Benchmark for Meta Table Variance

3.3 Combination

The combination branch predictor has very stable and linear characteristics, even more so than the bimodal branch predictor. The combination branch predictor, as the meta table size varies, has virtually the same CPI per benchmark (see Figure 9). By the same token, the combination branch predictor's CPI to power unit ratio is also very stable (see Figure 10). The CPI to power unit ratio for the predictor does decrease linearly (and universally) as the size of the meta table increases, but only a very little, suggesting that the meta table is a rather power-efficient design.

4 Feature Considerations

When examining the power consumption of the rest of the Simple Scalar processor functional units, a few features are obvious hot spots—the register update unit (RUU), the issue and decode widths, and the functional units themselves.[1] Similar to modifying the caches or the branch predictors, these features may not necessarily scale performance linearly with regard to power, and so there may be in many cases a configuration where the trade-off between the two is the best.

4.1 Register Update Unit

As the size of the Register Update Unit (RUU) is increased, the CPI of the benchmarks tested decreases rather dramatically up to about 16 entries, and then adds little further performance improvements (see Figure 11). The CPI per unit of power decreases along a similar curve (see Figure 12). Essentially, as RUU size increases (and thus so do the power requirements), the RUU does not provide as much more of a performance boost. The sweet spot for the Simple Scalar RUU seems to be around 8 or 4. While RUU sizes of 16 or more are definitely faster, the difference in CPI per power unit begins to get low enough that the extra power for more RUU units doesn't buy enough extra speed to be worth the extra power for power-conscious applications.

4.2 Issue and Decode Width

Issue and decode width are tightly coupled when it comes to modifying them to improve performance—extra issue width is not very useful without extra decode width, and vice versa,

[1] The clock is a major power consumer—more-so than perhaps any other single feature on a microprocessor. However, modifying the clock signal is outside the scope of this paper.

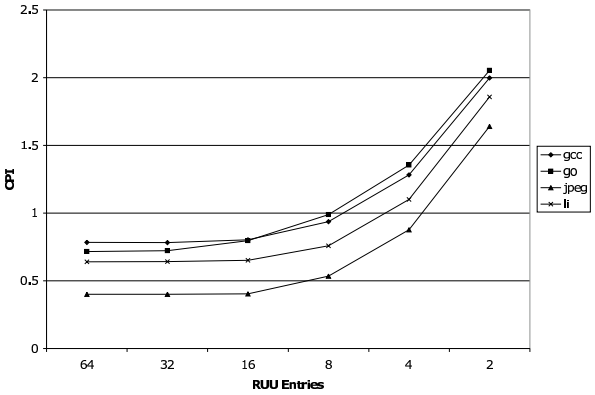

Figure 11: CPI by Benchmark for RUU Variance

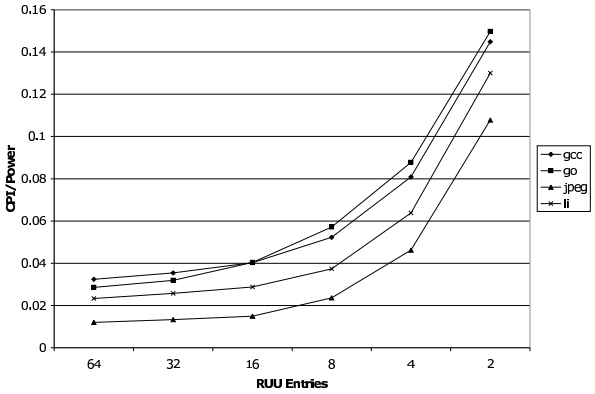

Figure 12: CPI/Power by Benchmark for RUU Variance

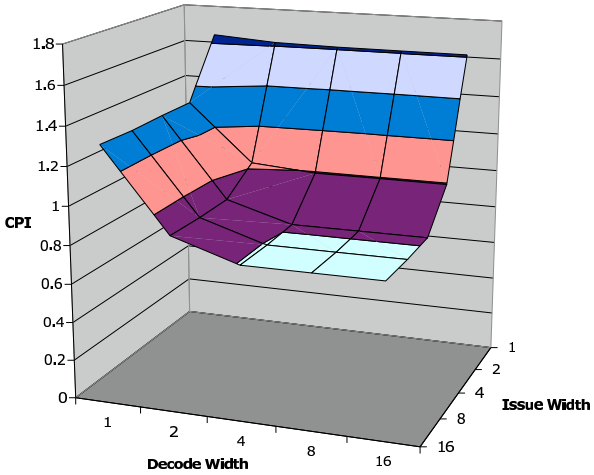

Figure 13: CPI for the Spec95 gcc Benchmark, for Issue Width and Decode Width

because they are bottlenecks for each other. Thus, instead of varying them separately, they must be varied together.

The CPI of the benchmarks tends to vary in small and very symmetric ways—that is, lessening the issue width by one yields the same performance penalty as yielding a decode width (Figure 13 is a good example of this symmetry). What is interesting is that the CPI per power unit ratios are very much not symmetrical at all, demonstrating across the board that issue units are much more important for performance than decode units are—indcating that decode units are preferable for turning off or leaving out of the die in order to save power.

4.3 Execution Units

The benchmarks used did not seem to use the integer multipliers at all,[2] so integer multipliers were ignored. Also, the utility of the integer units is limited by the issue and decode width. For the purposes of consistency, all testing was done with the default issue and

[2]Varying the number of integer multipliers in all but two circumstances (in which the effect was less than 0.001 CPI) had no effect on CPI.

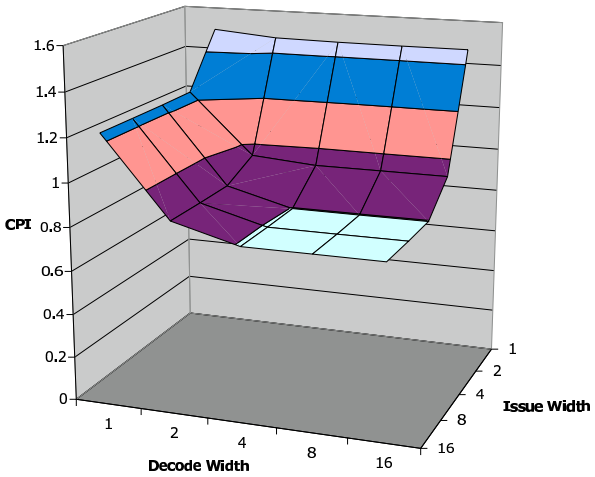

Figure 14: CPI for the Spec95 go Benchmark, for Issue Width and Decode Width

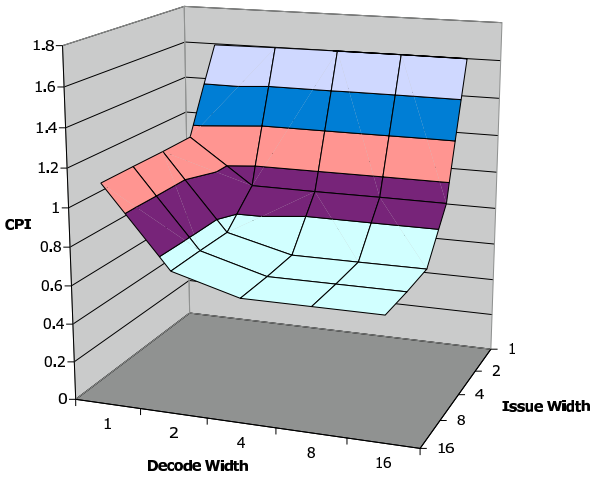

Figure 16: CPI for the Spec95 li Benchmark, for Issue Width and Decode Width

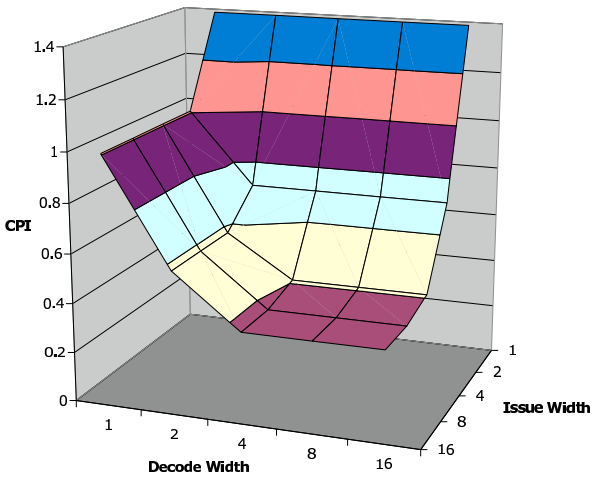

Figure 15: CPI for the Spec95 jpeg Benchmark, for Issue Width and Decode Width

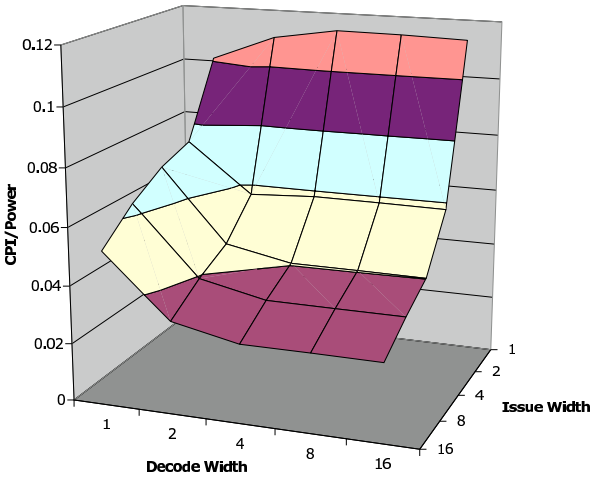

Figure 17: CPI/Power for the Spec95 gcc Benchmark, for Issue Width and Decode Width

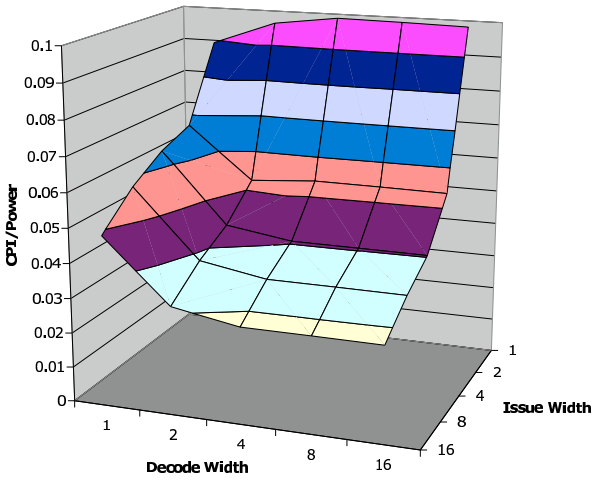

Figure 18: CPI/Power for the Spec95 go Benchmark, for Issue Width and Decode Width

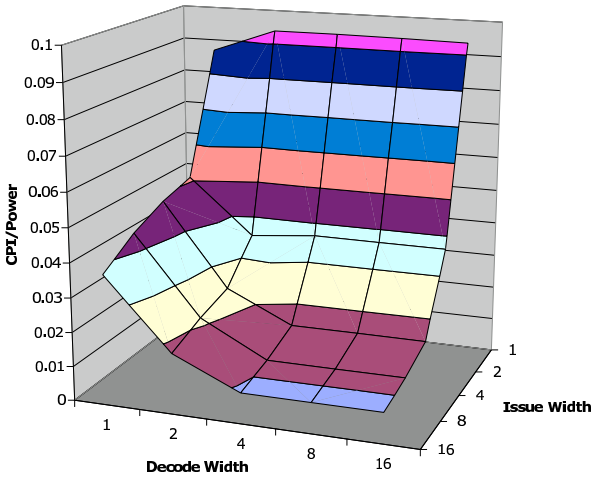

Figure 19: CPI/Power for the Spec95 jpeg Benchmark, for Issue Width and Decode Width

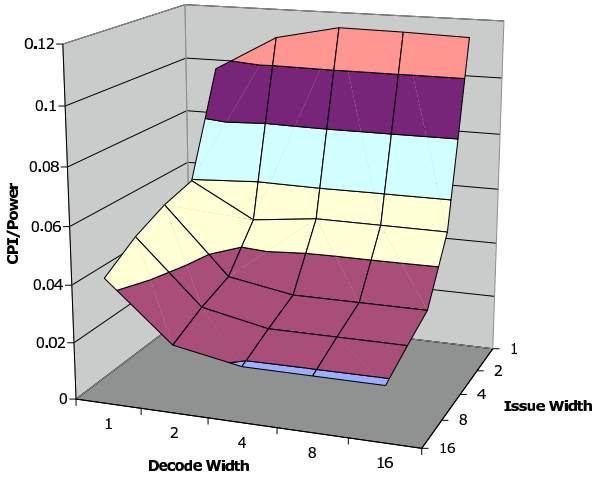

Figure 20: CPI/Power for the Spec95 li Benchmark, for Issue Width and Decode Width

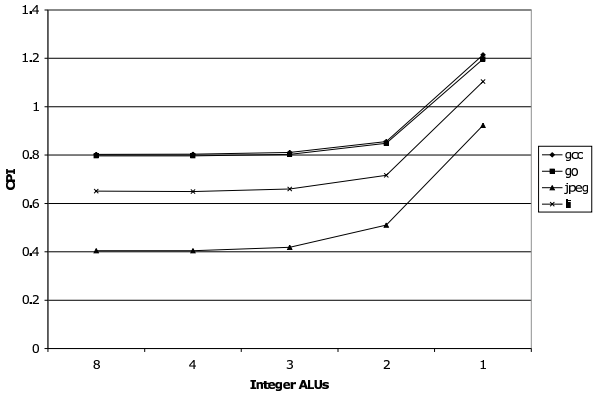

Figure 21: CPI by Benchmark for Integer ALU Count Variance

decode widths of four, which is why the CPI varies very little between four and eight ALUs (see Figure 21). For that matter, CPI doesn't vary much more between four and three ALUs. This information is essentially the exact same information that is conveyed by the CPI per power unit data (see Figure 22). After the second ALU is added to the system, the CPI per power unit differences are relatively minor, suggesting that for this issue and decode width of four, two integer ALUs is the best trade-off.

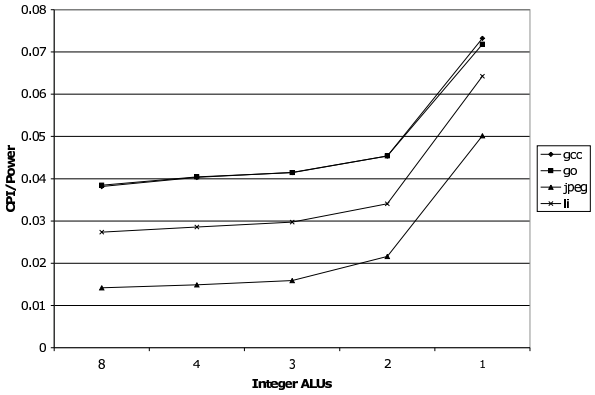

Figure 22: CPI/Power by Benchmark for Integer ALU Count Variance

5 Conclusions

Traditionally, the largest power consumption in modern processing has been directly related to the amount of instruction level parallelism support in a processor configuration. Specifically, the consumption by most architecture units is shown by [18] to scale proportional to a power of the issue width. This result is verified using the sim-wattch toolkit [4]. Additionally, varying the register update unit (which provides out-of-order execution) size are shown to scale performance non-linearly with respect to power. Generally, issue width should be optimized for performance rather than power consumption. Hardware controlled by the OS to selectively turn off parallel units could cut power drastically, but the performance detriment is too large to dismiss ILP hardware.

A non-trivial amount of power is consumed by cache hierarchies. As caches scale to 50% of chip area and beyond, the power spent in maintaining the state of cache blocks will begin to dominate. This leakage power overtakes all other chip consumption at the 0.10 micron feature size [6]. Prior techniques in reducing the leakage power consumption have reported up to 75% savings in leakage energy [11]. The impact on performance is not easily quantified, but a reported increase in energy-delay product of 2.3% suggests a very significant hit [11]. In simulation, a more effective strategy is proposed. Just as in [11] and [8], this strategy runs cache blocks at a minimum voltage when they are suspected to be unused. By separating caches into large superblocks and preemptively turning on superblocks before they are accessed, as much as 93% of the cache can be in a power saving state, and less than 1% of cache accesses result in a large clock-cycle penalty due to a mispredicted superblock access. The power overhead of superblock accessing ought to be smaller than all strategies presented in [11] because per-block control is not necessary. Clearly, this strategy holds great potential because of its comparatively minimal overhead and small performance hit. Even in processors where power consumption isn't a large consideration, this technique makes sense to reduce the heat production of a processor (this providing higher stable clock speeds).

Modern branch prediction units consume some 20% of the power of a typical processor, as reported by sim-wattch. Unsuccessful branch prediction strategies counteract the benefits of instruction level parallelism, forcing pipeline stalls and wasting processor power and performance. Thus, the focus for a low-power microprocessor must be to choose a branch prediction strategy that is accurate and is architecturally minimal. For the configurations tested, no single branch prediction is shown to be the most effective at reducing power and providing performance. Combined branch prediction methods are discouraged because they require the most hardware for marginal performance returns, and bimodal predictors do well, but are power expensive. 2-layer predictors should be investigated more, to explain their unique power characteristics.

6 Future Work

The effect of instruction level parallelism (ILP) on power is well characterized. Suggested implementations that can scale the level of ILP dynamic need to be explored further. This needs to be compared to voltage-stepping and clock reduction techniques.

For the reduction of leakage power, more robust benchmarking is an absolute necessity. The simulation code needs to be expanded to incorporate appropriate penalties for sleeping and reviving a cache block. A fully-associative superblock buffer with a least-recently used replacement scheme instead of a direct-mapped buffer could increase returns on smaller buffer sizes, but the configuration is untested. Additionally, the problem of multitasking is practically intractable for the simulation configuration; this is suspected to have a large impact on the level 2 cache access patterns, thus potentially degrading the performance of superblock switched caches. Combining this with other proposed techniques [11][8] for reducing leakage power consumption may prove even more useful at reducing leakage power consumption. Exploration of this technique in level 1 cache may be a worthy endeavor.

Lastly, branch prediction strategies are thoroughly characterized in the literature. The effect of branch misprediction on power (because of pipeline flushing) demands further study.

References

[1] Ning An, Suhanva Gurumurthi, Anand Sivasubramaniam, Narayanan Vijaykrishnan, Mahmut Kandemir, and Mary Jane Irwin. Energy-performance trade-offs for spatial access methods on memory-resident data. *The VLDB Journal — The International Journal on Very Large Data Bases*, 11(3):179–197, November 2002.

[2] Ana-Maria Badulescu and Alexander Veidenbaum. Power efficient instruction cache for wide-issue processors.

[3] Luca Benini and Giovanni de Micheli. System-level power optimization: Techniques and tools. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 5(2):115–192, 2000.

[4] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, Vancouver, British Columbia, Canada, 2000. ACM Press.

[5] Ramon Canal, Antonio González, and James E. Smith. Very low power pipelines using significance compression. In *Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture*, pages 181–190, Monterey, California, United States, 2000. ACM Press.

[6] A. Chandrakasan, W. J. Browhill, and F. Fox. Design of high-performance microprocessor circuits. 2001.

[7] Daniele Folegnani and Antonio González. Energy-effective issue logic. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 230–239, Göteborg, Sweden, 2001. ACM Press.

[8] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. *ISCA-28*, June 2001.

[9] Daehong Kim, Dongwan Shin, and Kiyoung Choi. Low power pipelining of linear systems: A common operand centric approach. In *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*, pages 225–230, Huntington Beach, California, USA, 2001. SIGDA, ACM Press.

[10] Jinson Koppanalil, Prakash Ramrakhyani, Sameer Desai, Anu Vaidyanathan, and Eric Rotenberg. A case for dynamic pipeline scaling. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 1–8, Greenoble, France, 2002. ACM Press.

[11] L. Li, I. Kadayif, Y-F. Tsai, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and A. Sivasubramaniam. Leakage energy management in cache hierarchies. In *Proceedings of the 11th International Conference on Parallel Architectures and Compilation Techniques*, 2002.

[12] T. May and M. Woods. Alpha-particled-induced soft errors in dynamic memories. *IEEE Trans. on Electronic Devices*, 26(1), January 1979.

[13] P.R.V.d. Meer and A. V. Staveren. Standby-current reduction for deep submicron VLSI CMOS circuits: Smart series switch. In *ProRISC/IEEE Workshop*, pages 401–404, December 2000.

[14] B. Nikolic. State-preserving leakage control mechanisms, September 2001.

[15] G. Palermo, M. Sam, C. Silvan, V. Zaccari, and R. Zafalo. Branch prediction techniques for low-power VLIW processors. In *Proceedings of the 13th ACM Great Lakes Symposium on VLSI*, pages 225–228, Washington, D. C., USA, 2003. ACM Press.

[16] Massoud Pedram. Power minimization in IC design: Principles and applications. *ACM Transactions on Design Automation of Electronic Systems*, 1(1):3–56, January 1996.

[17] S. Yang, M. D. Powell, B. Falsafi, K. Roy, and T. N. Vijaykumar. An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance i-caches. In *HPCA-7*, January 2001.

[18] V. Zyuban and P. Kogge. Optimization of high-performance superscalar architectures for energy efficiency. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, pages 84–89, Rapallo, Italy, 2000. ACM Press.