Instruction-Window Power Reduction Using Data Dependence Metric

Ziad Youssfi and Michael Shanblatt
Department of Electrical and Computer Engineering
Michigan State University
East Lansing, MI 48824
ziad@egr.msu.edu

Abstract

A large portion of the power consumed by a high-performance out-of-order microprocessor is attributed to units within the instruction window pipeline stages, from dispatch to commit. Power consumed within the window, and by the overall processor, is reduced by dynamic size adjustment of those units based on program needs. In this article, we introduce a new metric for estimating the Instruction Level Parallelism (ILP) over a group of instructions using a simple ratio of the longest data dependence path's length found in the group to the total number of instructions, and hence referred to as the Longest Dependence Path Ratio (LDPR). We couple this concept with a design where units in the instruction window are divided into segments that are dynamically disabled for power savings when they are unused. The maximum number of enabled segments that a program can utilize is limited based on the amount of ILP estimated using the LDPR. We also show how measuring the LDPR is implemented by a simple extension to the dependence-check-logic at dispatch stage. Results using the LDPR show that power is saved with a minimal impact on performance, by reducing the number of available segments, not only in low ILP periods, but also over those having very high ILP. Using the Wattch power models and the MinneSPEC benchmarks, the LDPR implementation achieves an average of 36.5% of window power savings and 11% overall processor power savings, and with an average performance loss of a very modest 1.9%.

I. Introduction

New generation high-performance microprocessors exhibit an increasing trend of power consumption due to device density and faster clocking. Higher power consumption results in more heat generation mandating more efficient, and usually expensive, cooling. Moreover, more power consumption implies that the system has to tolerate high current peaks and be able to deliver high and fast changing amounts of current. Together, these factors lead to a super-linear increase in cost [1]-[3]. The ITRS predicts that power consumption will exceed package limitations by a factor of 25 over the next 15 years and calls for innovative techniques at all levels of the design cycle to curb this trend [4].

As new microprocessor designs push toward higher performance, architects are using larger and more complex units to extract higher amounts of Instruction Level Parallelism (ILP). The issue queue with its wake-up and select logic is a case in point [8]. With complexity and power increasing due to longer bit-lines and words-lines of RAM arrays, and tag-lines and match lines of CAM arrays, the issue queue power consumption has become a major power-hungry component of the overall processor.

In this article, we present a dynamic technique to reduce the power consumption of units within the instruction window, based on a *direct* measurement of the available amount of ILP in the window. Using the SimpleScalar simulator [9], the Register Update Unit (RUU), and the Load-Store Queue (LSQ) are divided into segments of fixed number of entries—RUU being equivalent to reorder-buffer (ROB) plus issue queue (IQ). When a segment is unused, it is disabled for power savings. The maximum number of segments available to a program, however, is set dynamically based on the available amount of ILP measured using our new metric called the Longest Dependence Path Ratio (LDPR). This metric expresses the amount of ILP on a scale from zero to one; the highest ILP available is when LDRP is near zero, and conversely, the lowest ILP available is when the LDPR is near one. Using slightly extended dependence-check logic, the LDPR is measured every cycle at the dispatch stage and averaged over a short sample period. The maximum allowable number of segments a running program can use over the next sample period is based on a simple policy: the higher is the LDPR—meaning lower ILP—the lower the number of maximum segments available to the program. We also let the policy set a maximum allowable number of segments when the LDPR is very low—meaning when the ILP is very high. This second part of the LDPR policy may seem counterintuitive at first, but will be explained further in Section IV.

Presenting proposed designs addressing the increase in power and complexity of the IQ, a recent survey by Abella and González [5] groups the designs based on the type of technique used. One technique type changes the size of the IQ dynamically based on its contribution to performance. Our proposed technique falls under this dynamic sizing category; however, it has the advantage, of basing the decision *directly* on the amount of ILP in feed-forward fashion. Other techniques, such as the one presented by Buyuktosunogly *et al.* [11] for example, rely on measuring performance via the IPC (Instructions per Cycle) to dynamically adjust resources. Relying on the IPC has the drawback of needing a guard mechanism to prevent unnecessary performance loss; limiting resource size could result in low IPC, and this low IPC, in a feedback fashion, reinforces the decision to keep resource sizes small. A guard mechanism, therefore, is needed to periodically upsize resources to prevent a vicious feedback cycle. Our LDPR technique has the advantage of more power savings by not needing this periodic upsizing guard mechanism as well as the added advantage of adjusting the ROB size to match the available ILP. Another advantage of the LPDR technique is that it is purely based on data dependences between instructions, thus, it can be moved up to the software level; with knowledge about the microarchitecture, a compiler technique can provide ILP hints to the microarchitecture for power savings.

Power consumption can be accurately estimated after design floor-planning and layout. Chip architects, however, need to estimate power-performance tradeoffs early on in the design cycle. Thus, power models have been proposed that rapidly estimate power consumption with reasonable accuracy at the microarchitecture level. For example, the Wattch power model applied in this work uses four common circuit implementations to find the capacitive loadings of functional units [6]. These load capacitances help in quickly calculating the units' power consumption when they are accessed during functional simulations.

The rest of the paper is organized as follows: Section II presents the framework for our methodology including the rationale for the LDPR metric, the power models used in our study, as well as the simulated benchmarks; Section III presents the implementation of our segmentation design and the LDPR measurement at the dispatch stage; Section IV presents our power savings and performance results; and finally, Section V concludes with some suggestions on future directions.

II. Framework

A. The LDPR metric

Calculating the LDPR is based on a directed graph representation and a measurement of the Longest Dependence Path (LDP) in the graph. For any group of instructions taken from the dynamic instruction stream, data dependences between instructions can be represented as a directed graph with vertices representing instructions and edges representing data dependences. Because later instructions in a stream can have data dependences only on earlier instructions, the resulting directed graph is always acyclic (the edges are always in the direction of earlier instructions), and the LDP is always guaranteed to be found as the path with the maximum number of edges. In general, for any group of instructions, the larger is the LDP, the less there is available ILP, and vice versa.

In some graph cases, however, for a given group of instruction the LDP alone is not enough for measuring the ILP. For instance, the LDP is not an accurate measurement of the ILP if there are many instructions that are independent or belonging to short dependence paths apart from the LDP. Using the directed graph representation, Figure 1 shows two different groups of instructions with the same LDP; however, Figure 1-(a) has less available ILP than Figure 1-(b).

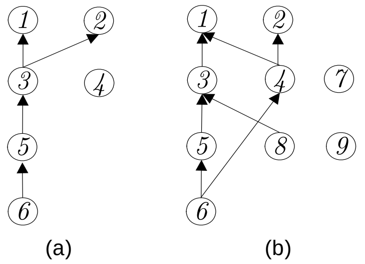

Figure 1: Example of two directed graph representations for data dependence where the LDP (Longest Dependence Path) is the same for both graphs, *i.e.* LDP = 3 edges. The total number of instructions in (a) is 6, giving an LDPR= 3/6. In (b) the LDPR = 3/9 indicating higher ILP than in (a).

To accurately measure the ILP and to compensate for the existence possibility of many independent instructions or external short dependence paths, a compensation factor is needed. One effective way to achieve this compensation is normalizing the LDP by taking its ratio to the total number of instructions in the graph. Thus, the LDPR is simply (LDP/n) where LDP is the longest dependence path and n is the total number of instructions in the graph. Since we have $0 \leq \text{LDP} < n$, the LDPR range is then $0 \leq \text{LDPR} < 1$, and the LDPR value is inversely proportional to the amount of ILP in the instructions graph. When most instructions in a graph do not belong to the LDP and the LDP itself is small, the LDPR tends toward zero indicating a higher level of ILP. Inversely, when most instructions in a graph are part of the LDP, the LDPR tends toward one indicating lower level of ILP.

Normalizing the LDPR to the number of instructions makes it suitable for comparing the amount of ILP across groups with different number of instructions. The LDPR, however, is still only an estimator, and not an absolute measure. Figure 2 shows two cases where the LDPR value is the 3/7 but the ILP is higher in case (b) than in (a).

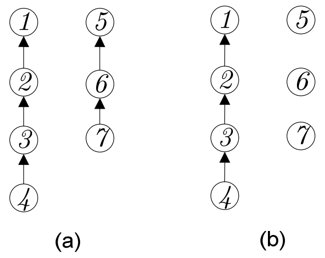

Figure 2: The LDPR is 3/7 in both cases (a) and (b). The ILP, however, is higher in (b) than in (a).

Despite some exceptions, the LDPR is still good estimator for dynamic sizing for the following two reasons:

1) The LDPR accuracy increases as its value tends toward its maximum and minimum values of one and zero, and it decreases in accuracy when it is in midrange around 0.5. Fewer exceptional cases can be found as the LDPR tends toward zero or one. With our LDPR policy, explained later, resources are gradually sized down as the LDPR nears the extreme ends, and they are set to maximum size when the LPDR value is around the 0.5 interval. In effect, power savings are gradually and increasingly acquired over the higher LDPR accuracy intervals so that performance is not negatively impacted.
2) The LDPR is averaged over a sample period of 256 cycles. Unless specific exceptional cases, such as in Figure 2, are sustained over the entire sample period, the averaging process will smooth out spurious cases.

The time complexity of measuring the LDP can be high for a large number of instructions such as the case, for example, in a large issue queue or over the entire instruction window. The LDP, however, is still effectively measured over a small number of instructions, such as the case at the dispatch stage, where the number of instructions is limited by the pipeline width. Since the LDPR is normalized to the number of instructions, averaging the LDPR at dispatch over a sample period constructs a good picture, albeit one chunck at a time, of the amount of ILP in the instruction window. Later in Section III, the complexity of the hardware structure for measuring the LDPR at the dispatch stage is addressed where it is shown how the dependence-logic-check is leveraged to obtain the LDPR.

To validate the LDPR metric, we developed two code segments where the ILP is deliberately made very low in the first and very high in the second. Consisting of a loop of nine instructions in both segments, every instruction in the first segment is dependent on the one before it, and conversely, all instructions in the second segment are independent of each other (except for the loop branch). When running these two code segments back to back, the LPDR is observed to be near zero during the run of the first segment and near one during the run of the second segment as shown in Figure 3, where the LDPR is almost an exact reverse correspondence to the IPC. To quantitatively confirm this negative correspondence, the correlation coefficient r between the IPC and LDPR over the simulation run is calculated and found to be r=-0.93, which is close to -1, indicating a strong linear correlation.

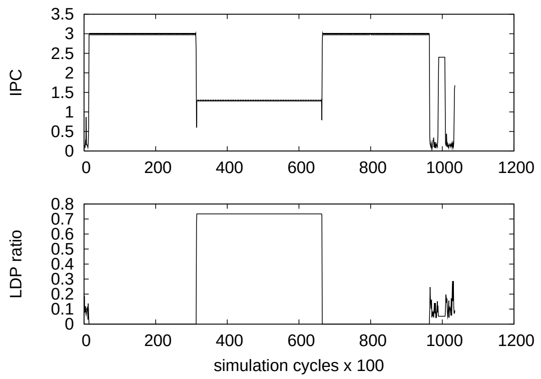

Figure 3: The IPC and LDPR dynamic measurements during the two test cases of code segments of high and low ILP periods.

B. *Power models*

To compute the load capacitance associated with dynamic power dissipation, Wattch uses a classification technique that groups common processor circuits into four categories: RAM array, CAM array, complex logic, and clock [6]. For each of these circuit categories, specific equations are used to compute the load capacitance using architecture parameters such as unit size, number of read and write ports, and number of entries.

Conditional clock gating is a general approach taken by architects and circuit designers to decrease processor power consumption. Wattch computes power consumption using three clock gating styles: all-or-nothing, linear, and linear with 10% overhead. The all-or-nothing style incurs the full power of a unit if it is accessed during a clock cycle, regardless of how many ports are accessed. In the linear style, however, the full power of a unit is scaled linearly by the number of accessed ports, while in the third style an extra 10% is added to the linear value to reflect overhead due the clock gating control circuits. To reflect a realistic clock gating, we use the linear style with the 10% overhead to compute power saving throughout our implementation. The architecture parameters used in our sim-outorder simulations are listed in Table 1.

Table 1: Architecture configurations

PARAMETER	CONFIGURATION
Instruction Fetch Queue Size	16 instructions, up to 2 branches
Branch Predictor	Combined bimodal 4096 entries, 2-level L1size 1, L2size 1024, history size 8
Decode, issue, commit widths	8 instructions
RUU/LSQ	RUU 128 entries, LSQ 64 entries
D-cache-L1	1024 sets, 4-way, 64 bytes per block, LRU, 1 cycle hit latency
I-cache-L1	4096 sets, direct, 64 bytes per block, LRU, 1 cycle hit latency
D/I-cache-L2	16K sets, 4-way, 64 bytes per block, LRU, 6 cycle hit latency
ALU	8 integer, 2 integer mult/div, 4 floating, 2 floating mult/div

C. *Benchmarks*

We used 18 of the MinneSPEC benchmarks [14], 13 integer and 5 floating-point, for our study. All benchmarks were compiled by the gcc SimpleScalar for the PISA instruction set. We also used the MinneSPEC large input data sets for our simulations, resulting in 700 million to 5 billion simulated instructions per benchmark.

III. Implementations

To implement the segmentation of units within the instruction window, we use a similar approach to that used by Ponomarev, *et al.*, [10] where the IQ bit-lines and tag-line are segmented for lower power consumption. Traditional circuit implementations of the instruction window are a mix of RAM and CAM arrays where RAM bit-lines and CAM tag-lines run the entire vertical length of the arrays, resulting in high load capacitance, and hence make a substantial contribution to power consumption. By segmentation of the bit-lines and tag-lines, each segment has a shorter length and a smaller number of pass-transistor connections resulting in lower load capacitance per segment and hence lower power consumption. During data read or write, if all entries in a segment are empty, the segment bit/tag-line is switched off, saving pre-charge energy. A segment is determined to be empty by using an extra busy bit for each entry. A logical OR of these bits controls the segment switch. The OR function and the segment switches result in extra overhead; the total power consumption, however, is reduced due to switching off empty segments and by optimizing the number of entries per segment. In our study, we find 16 entries per segment for the instruction window's units result in optimum power savings.

In our adaptive technique, we use the measured LDPR to limit the maximum number of active segments using a simple policy. We measure and average the LDPR over a 256 cycle sample period and translate this value, using a simple mapping function, to the number of maximum entries allowed during the next sample period. Every cycle, instructions are dispatched to the IQ, the ROB (or RUU), and the LSQ as long as the maximum allowable number of entries is not exceeded; otherwise, dispatching is temporarily halted (or RUU allocation is halted). Since an entire segment can be either turned on or off, the maximum number of entries is set to be modulo the number of entries per segment—for a 16 entries per segment, the possible allowable numbers of entries are 16, 32, 48, 64, etc. Figure 4 shows a mapping function for the ROB where the x-axis is the averaged LDPR value obtained at dispatch, while the y-axis is the maximum allowable number of active ROB entries for the current program having the given LDPR. The LSQ is always adjusted to be half the size of the ROB.

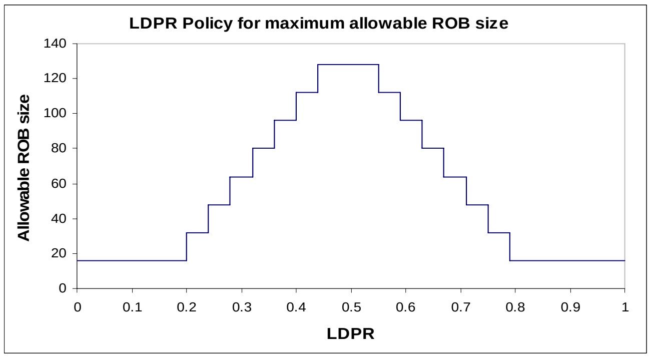

Figure 4: A simple LDPR policy to set the maximum allowable number of entries in the ROB (the y-axis) for a program given the current LDPR value. The number of physical entries in the ROB is 128.

The rationale for this policy is that when the estimated ILP is very low, a larger number of entries do not contribute to higher performance as there are very few or no instructions that can issue in parallel. Likewise, when the ILP is very abundant, having a larger number of entries beyond a certain point does not contribute to increased performance as the there are already enough instructions that can issue in parallel, and a smaller ROB/IQ/LSQ essentially provides the same performance.

Measuring the LDPR at dispatch is accomplished by extending the dependence-check-logic. As instructions are decoded, logical register operands are renamed to physical registers to remove any name dependences introduced by the compiler. When the dependence-check-logic determines that an instruction's source operand is dependent on a previous instruction's destination operand, it renames the producer and consumer to the same physical register. We extend this check-logic on register dependence so that each time data dependence is detected, its depth in the dependence chain is incremented by one and propagated to the next instruction in the chain. Since only the LDP needs to be measured, a comparator is needed to propagate only the maximum dependence for two source operands for the same instruction. In other words, if an instruction is the merge point of two dependence chains, only the longest branch of the chain is propagated to the consumer instruction. A final comparator determines the maximum of all dependence chains lengths to find the LDP. Note that if the LDPR is measured every time over the same number of instructions, *e.g.,* over 8 instructions at dispatch, the ratio is not needed and the LDP itself is equivalent to the LDPR. Figure 5 illustrates a simple example for obtaining the LDPR over 4 instructions. The magnitude of the measurement of the LDP is on the order of $(\log_2 n)$ where n is the number of instructions. And since n, the dispatch width, is small, the complexity and overhead to obtain the LDPR at dispatch is minimal compared to the overall dependence-check-logic, and hence ignored in our study. For instance, if the dispatch width is 4 instructions, the hardware needed to measure the LDPR is 4 comparators and 3 increment-by-1 counters all of which are two bits wide.

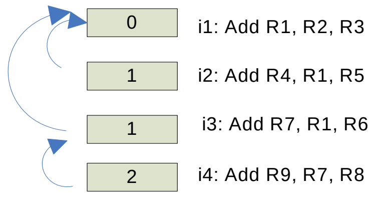

Figure 5: an example of four instructions at dispatch. The shaded boxes represent dependence depth counters. The maximum dependence value for all chains, or LDP, is 2 and the LDPR = 2/4.

IV. Results

Using the segmentation design and the MinneSPEC benchmarks, we ran a set of three simulations per benchmark. The first simulation serves as the comparison basis with unmodified sim-outorder and Wattch power models. The second simulation includes modifications to the Wattch power models to account for power savings when empty segments are disabled. And the third simulation uses modifications to the sim-outorder including the LDPR technique. For each benchmark, the percent power savings over the first unmodified base model relative to the second and third simulation were computed. The power savings for the instruction window and the entire processor are reported in Figure 6 and Figure 7, respectively. For every benchmark, the power savings due to the addition of the LDPR policy are greater than power savings due to only disabling empty segments.

The effect on performance due to limiting the allowable number of segments in the instruction window using the LDPR policy is shown in Figure 8 as speedup based on the IPC. Speedup values above 100% in the figure are performance improvement relative to the base model; values below 100% indicate performance degradation. Some benchmarks, such as 177.mesa, have speedup improvement. For such benchmarks with smaller block size, It is likely that some branch fetches are delayed due to a smaller instruction window until earlier branches are resolved, resulting in greater branch prediction accuracy. For instance 117.mesa showed a substantial retired-address-stack prediction improvement using the LDPR policy. This improvement effect is also similar to the one observed by Buyuktosunoglu *et al.* in their adaptive instruction fetch and issue [15]. The average performance degradation over all the benchmarks is a modest 1.9%.

V. Conclusion

A new metric called the LDPR is introduced to estimate the amount of ILP in the instruction window of high-performance out-of-order microprocessors. For a group of instructions, the LDPR is defined as the ratio of the longest dependence path's length found in the group to the total number of instructions. The LDPR value ranges from 0 to 1 where smaller values indicate higher ILP, and conversely, higher values indicate lower ILP. We validate the LDPR as a good predictor of performance and give a simple implementation technique by extending the dependence-check-logic at the dispatch stage.

We complement our LDPR concept with a design to segment units in the instruction-window so that empty segments are turned off for power savings. A limiting policy sets the maximum number of allowable segments for a program during a sample period based on the LDPR value averaged over the previous sample period. The maximum allowable number of segments is a simple mapping function from LDPR values. A mapping function example for optimum power savings is presented. Our simulations with the MinneSPEC benchmarks and the Wattch power models show power saving results of 36.5% for the instruction-window and 11% for the overall processor and with a modest performance loss of 1.9%.

The LDPR technique is orthogonal to other power savings techniques that can be combined with it such as pipeline gating presented by Manne *et al.*[12]. More power savings can also be achieved by disabling word-lines and match lines of ready entries in the IQ and LSQ as implemented by Folegnani, *et al.*[13].

Further research can be carried out for the feasibility of moving the LDPR measurement up into software level such as a compiler, or further work can be carried out to extend the LDPR implementation in relation to trace cache. The LDPR metric can also be used by reconfiguration techniques to enhance performance based on the available amount of ILP.

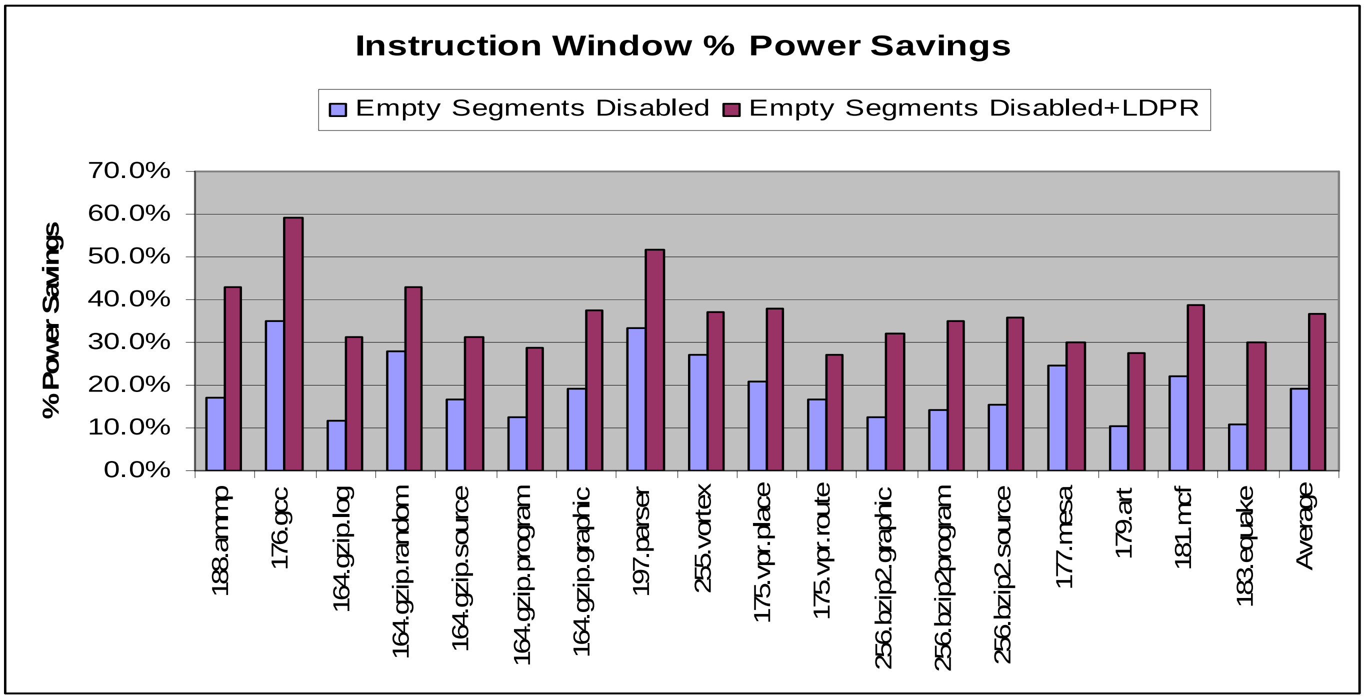

Figure 6: The % power saving over the MinneSPEC benchmarks for the instruction window. The % savings are with respect to the base unmodified model.

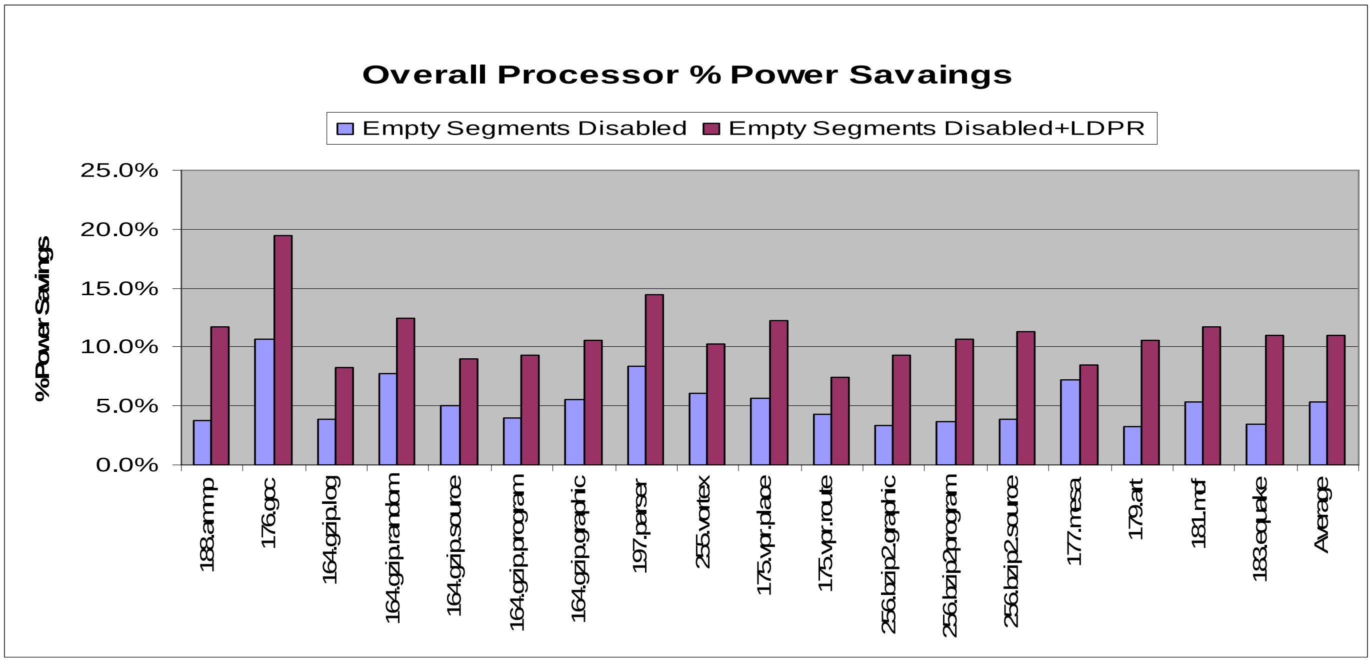

Figure 7: The % power saving over the MinneSPEC benchmarks for the overall processor. The % savings are with respect to the base unmodified model.

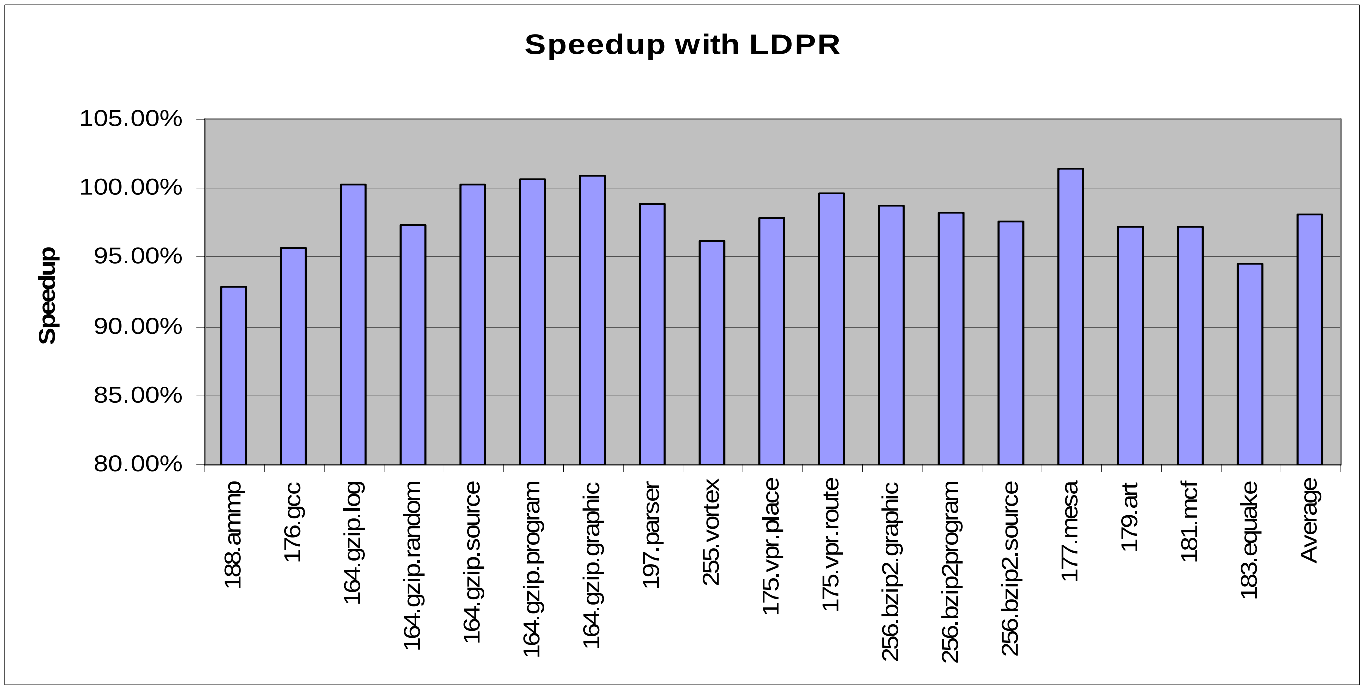

Figure 8: The percent speedup due to applying the LDPR policy. Note that the y-axis starts at 80%.

I. References

[1] T. Mudge, “Power: A First-Class Architectural Design Constraint,” *Computer*, pp. 52-58, Apr. 2001.

[2] V.Tiwari et al. “Reducing power in high-performance micro-processors,” 35th *Design Automation Conference*, 1998.

[3] D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J. D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook, “Power-Aware Microarchitecture: Design and modeling challenges for next-generation microprocessors,” *IEEE Micro*, pp. 26-24, Nov-Dec 2000.

[4] International Technology Roadmap for Semiconductors, “Executive Summary,” Tech. Rep., ITRS, 2003.

[5] J. Abella and A. González, “Power—And Complexity—Aware Issue Queue Designs,” *IEEE Micro*, pp. 50-58, Sept-Oct 2003.

[6] D. Brooks, V. Tiwari, and M. Martonosi, “Wattch: A Framework for Architectural-Level Power Analysis and Optimizations,” in *Proceedings of the 27th International Symposium on Computer Architecture,* 2000.

[7] A. J. Klein Osowski and David J. Lilja, “MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Research,” *Computer Architecture Letters*, vol. 1, June 2002.

[8] S. Palacharla, N. P. Jouppi, and J. E. Smith, “Complexity-Effective Superscalar Processors,” Proceedings of the 24th International Symposium on Computer Architecture, 1997.

[9] T. Austin and D. Burger, “The SimpleScalar Tool Set,” version 3.0, University of Wisconsin, 1999.

[10] D. V. Ponomarev, G. Kucuk, O. Ergin, K. Ghose, and P. Kogge, “Energy-Efficient Issue Queue Design,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 11, No. 5, October 2003.

[11] A. Buyuktosunoglu, S. E. Schuster, D. Brooks, P. Bose, P. W. Cook, D. H. Albonesi. “*A Circuit Level Implementation of an Adaptive Issue Queue for Power-Aware Microprocessors*,” 11th Great Lakes Symposium on VLSI (GLSVLSI-01), March 2001.

[12] S. Manne, A. Klauser, and D. Grunwald, “Pipeline gating: Speculation Control for Energy Reduction,” Proceedings *of the International Symposium on Computer Architecture*, June 1998.

[13] D. Folegnani and A. González, “Energy-Effective Issue Logic,” Proceedings *of the International Symposium on Computer Architecture*, June 2001.

[14] A.J KleinOsowski and D. J. Lilja, “MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research,” *Computer Architecture Letters,* vol. 1, June 2002.

[15] A. Buyuktosunoglu, T. Karkanis, D. H. Albonesi, and P. Bose, “Energy Efficient Co-Adaptive Instruction Fetch and Issue,” Proceedings of the *International Symposium on Computer Architecture,* June 2003.