Potential for Branch Predictor Adaptation at the Program and Phase Level for Performance and Energy-Efficiency

Univ. of Virginia Dept. of Computer Science Technical Report CS-2005-19
November, 2005

Michele Co, Dee A.B. Weikle, and Kevin Skadron
Department of Computer Science
University of Virginia
{micheleco, dweikle, skadron}@cs.virginia.edu

Abstract

Experiments to determine the potential for program-level and/or phase-level adaptation of branch predictor configuration for the purpose of total processor energy savings were performed. The performance and energy-efficiency of an 8-wide issue, out-of-order processor with six different branch predictors were evaluated on the SPECcpu2000 benchmark suite. Each branch predictor was compared to the branch predictor with the highest overall average IPC. The comparison was performed at both the program-level and at 1M instruction intervals to determine the potential of adaptation of the branch predictor configuration to improve overall processor energy-efficiency. The results of these experiments indicate little potential for branch predictor adaptation for the SPECcpu2000 benchmark suite using the branch predictors evaluated. Additional results show that the piecewise linear branch predictor consumes significantly more energy than its close-performing competitors and that the hashed perceptron predictor is comparable both in performance and energy to the overall best-performing branch predictor in this study, O-GEHL.

1 Introduction

Energy efficiency has become important for most new chip designs. Basically, for a processor to be energy efficient, the architectural requirements such as power consumption, energy consumption, and performance must be balanced. Extending battery life in mobile devices and reducing utility costs for wall-powered systems are growing concerns. Therefore, better understanding the factors that affect overall chip energy efficiency is important. Understanding the tradeoffs that affect overall chip energy efficiency helps system designers focus their efforts on the areas in which energy can be most effectively reduced without adversely impacting performance.

The fetch unit contributes a large portion of total power consumption in a microprocessor. For example, Montanaro *et al.* [14] measure the StrongARM's fetch engine power consumption at 27% of total chip power. Understanding how fetch organization affects processor energy efficiency is important to processor design. There has been prior work that examines segments of the fetch engine design space [7, 17, 19, 18, 9, 8, 1, 12, 22, 16], most of which do not consider energy efficiency.

Branch prediction is one critical function of the fetch engine. It is one of the most powerful and widely-used techniques to expose instruction level parallelism. Branch prediction accuracy has been shown to be critical to higher performance.

Recent work by Parikh et al. [16] explored the energy efficiency of branch predictors, and generally concluded that the better the predictor, the better the chip's energy efficiency. Existing and newly proposed branch predictors can contribute to more than 5% of overall chip power, so understanding the relationship between branch predictor energy-efficiency and ov erall processor energy-efficiency is important. For example, recently proposed branch predictor designs [11, 20] which have shown significantly better prediction accuracy at a larger area cost have not addressed the issue of energy-efficiency. The tradeoff between the power consumption, energy consumption, and improved performance of these designs is unclear and has not been previously evaluated. Understanding this tradeoff is necessary to determine the benefit of implementing these designs.

Program phase tracking and prediction [21] has been proposed and applied in guiding remote profiling [15]. It has shown promising results in terms of low power and resource overhead and in terms of profile collection efficiency. This technical report explores the energy efficiency potential of using program phase behavior to adapt branch predictor configuration.

2 Related Work

We are not aware of any research that adapts the branch predictor configuration for energy-efficiency.

Parikh *et al.* [16] explored the role of branch predictor organization on power, energy, and performance tradeoffs for fetch engine design. They found that although extra power may need to be expended to improve the branch prediction accuracy, overall processor power and energy dissipation can be reduced.

Co, Weikle, and Skadron develop a metric for determining a branch predictor energy budget for comparing branch predictor designs [5] based on Parikh's insight [16]. This work does not address adaptation of branch predictor configuration.

Several techniques have been proposed that are related to adaptation for energy- or power-efficiency. Dynamic optimization has been applied to adapting cache configuration in an adaptive computing environment [10] and has been shown to significantly reduce L1 data and L2 cache energy consumption. This technical report does not focus on caches, but rather focuses on evaluating the potential of branch predictor reconfiguration for overall processor energy-efficiency.

Within the adaptive computing environment research area, adaptive issue queues have also been proposed [3, 6]. Issue queues act as a bridge between the front-end and the back-end of the processor. Dynamically adapting the size of the issue queues based on monitoring the issue queue occupation rate and the utilization rate of the execution units was shown to achieve significant energy savings. This work achieves significant energy savings which affect overall processor energy efficiency. We evaluate the energy saving potential of adapting the branch predictor configuration within the fetch unit.

Program phase tracking and prediction [21] has been proposed and applied in guiding remote profiling [15]. It has shown promising results in terms of low power and resource overhead and in terms of profile collection efficiency, but has not been used to adapt the fetch unit for energy-efficiency.

Chaver, et al. [4] present the *Phase-Based Adaptive Fetch Mechanism (PBAFM)* which achieves significant fetch energy savings, while at the same time improving overall performance. Applications are statically divided into modules and a software-based feedback mechanism is used to monitor resource demands and make configuration choices. The hardware is adapted to carry out reconfiguration between four fetch schemes: SEQ1, SEQ3, concurrent trace cache access, and modified sequential trace cache access. Resizing the trace cache and the branch target buffer is also evaluated. A profiling pass using reduced inputs is used to determine the fetch policy and structure sizes for each phase. This information is encoded into the binary so that the hardware may adapt at runtime. Our work focuses on the adaptation potential of branch predictors only.

3 Simulation Technique

The SimpleScalar toolset's [2] out-of-order processor simulator (*sim-outorder*) augmented with Wattch power models was used. An 8-wide issue processor with 256 KB caches was simulated on the SPECcpu2000 benchmark suite.

The branch predictors listed in Table 1 were simulated. Total processor energy, branch predictor energy, energy-delay-squared product (ED^2), and instructions per clock (IPC) were collected for the entire simulation run. These statistics were also collected at 1M instruction intervals to capture any potential for phase adaptation. [1]

4 Experimental Results

An "overall best" branch predictor was determined for both the integer and floating point benchmark suites based on average best IPC for an entire benchmark run(at the program level). For both the integer and floating point benchmarks, the O-GEHL branch predictor was determined to be overall best. The hashed perceptron predictor was a close competitor to O-GEHL for some benchmarks for some statistics. The 32 KB piecewise linear branch predictor included in the evaluation often had lower IPC and higher ED^2 (both are worse) than O-GEHL.

In order for adapting branch predictor configuration potential to exist, for one benchmark or segment of a program, more than one branch predictor should be a significantly better choice than the overall best. In our plotted results, this corresponds to the branch predictor lines crossing over one another, and over the 0% (O-GEHL) line with statistical significance. The crossing over demonstrates that there is potential benefit from choosing a branch predictor different from the overall best for a program or program segment.

Generally, the lack of positive difference in predictor performance along all the statistics between programs indicates little potential for adaptation at the program level. Similarly, when examining the potential for adaptation at the phase level, few opportunities for adapting the branch predictor are available.

[1] 1M instruction intervals were chosen because Lau [13] and Sherwood [] use this interval and show that program phases are in the 10M+ instruction range on average for SPECcpu2000.

Branch Predictor	Area (KB)	Configuration
gshare	8	L1: 15-bit, L2: 32K 2-bit counters, XOR: on
hybrid	8	8K-entry metachooser 8K-entry bimodal L1: 14-bit global history register L2: 16K-entry 2-bit counters
piecewise linear	32	34360 7-bit general weights, 2396 bias weights, 220 16-bit local history registers, 26-bit global history register
bimodal	1	4K-entry 2-bit counters
hashed perceptron	8	64 global history x 10 perceptrons per history, 10 local history
O-GEHL	8	8 2K-entry 4-bit counter tables, 1K-entry 1-bit tag table

Table 1: Branch predictor configurations evaluated.

4.1 Program Level Potential for Branch Predictor Adaptation

At the program level, IPC, ED^2, total processor energy, and branch predictor energy were plotted for each benchmark, for each branch predictor simulated. Results show that when examining the raw data, there is little potential for adapting branch predictor configuration at the program level to make energy efficiency gains.

To verify this, percent difference of each branch predictor from the overall best branch predictor was calculated and plotted.

Figure 1 shows the results for the integer and floating point benchmarks respectively.[2] For IPC, there is no predictor in the study which performs significantly better (positive percent difference) than the calculated overall average best predictor (O-GEHL). In addition, the relationship between the branch predictors does not change or cross over. This indicates that the predictors have a clear hierarchy of performance regardless of benchmark. In other words, for the evaluated predictors, there is not a better choice depending on which benchmark is evaluated.

In terms of energy, only the hashed perceptron shows a significant improvement in branch predictor energy (negative percent difference) of approximately 19% on average and reasonable improvement in total processor energy (negative percent difference) of 1-4%. However, this does not translate to significant energy-efficiency improvement, as seen by the less than 0.5% improvement in ED^2. The reason for this result could be due to the fact that the branch predictor does not comprise much of the total processor energy. Rather, the larger structures such as caches could be dominating the processor energy.

Figure 2 shows the breakdown of total processor energy by component for *mcf*, *crafty*, and *gcc*. It shows that the branch predictor comprises a very small percentage of the total processor energy. So, in order for the branch predictor to make a significant improvement to the energy-efficiency of the processor, it must improve runtime enough to significantly improve the ED^2.

Overall, at the program level, these results show that there is not much room for adapting the branch predictor for significant energy-efficiency savings.

4.2 Phase Level Potential for Branch Predictor Adaptation

For analysis at the phase level, the same statistics analyzed for the program level were plotted, but at 1M instruction intervals (IPC, ED^2, total processor energy, branch predictor energy). To simplify the plots, each graph represents data for all the branch predictors for one benchmark. Again, the raw data showed little potential for adapting the branch predictor configuration for energy efficiency improvement. To verify this, percent difference of each branch predictor from the overall best branch predictor was calculated and plotted.

Figures 3-5 and Figures 6-8 show the IPC and ED^2 results for the integer and floating point benchmarks respectively.

For a branch predictor to be considered a good candidate in terms of IPC, the lines should be above the 0% line. With the exception of *mcf* which is memory-bound, and a few sections of *bzip*, none of the evaluated predictors show an IPC improvement greater than 2% over O-GEHL.

In terms of ED^2 negative percent difference is better (below 0% line). When combined with the IPC results, the ED^2 results show that when running *gcc*, *gzip*, or *parser* the processor could improve its ED^2 by selecting the hashed perceptron predictor, but with the potential of sacrificing some performance (IPC). Overall, the hashed

[2]For clarity of viewing the data, the data for the simplest branch predictor, bimodal, and the two branch predictors performing closest to O-GEHL are shown.

perceptron predictor is a close competitor to O-GEHL in IPC and ED^2, whereas the piecewise linear branch predictor included in the study has consistently slightly worse IPC, and much higher ED^2.

Figures 9-11 and Figures 12-14 show the percent difference from O-GEHL for total processor energy and branch predictor energy for the integer and floating point benchmarks respectively.

For both total processor energy and branch predictor energy metrics, lower is better. Negative percent difference is better. For all the benchmarks except *mcf*, the piecewise linear and bimodal predictors cause total energy to be worse than that of O-GEHL. Hashed perceptron is a close competitor to O-GEHL, but still generally performs on average 1-2% worse than O-GEHL. For *mcf*, hashed perceptron shows from 5-15% improvement in total processor energy over O-GEHL for the first 200M instructions.

In terms of branch predictor energy, the 32KB piecewise linear branch predictor consistently consumes greater than 50% more energy than the O-GEHL predictor and the hashed perceptron predictor consumes about 20% less.

Overall, the phase level the results show that for the branch predictors evaluated there is little potential for adapting the branch predictor for improved energy efficiency on the SPECcpu2000 benchmark suite.

5 Conclusions

Several branch predictor configurations were simulated using an 8-wide SimpleScalar out-of-order processor simulator on the SPECcpu2000 benchmark suite to determine the potential for branch predictor adaptation to improve overall processor energy efficiency. Statistics were gathered at the whole program and phase level and analyzed. Results show that there is little potential for adapting branch predictor configuration to improve overall processor energy efficiency. Of the branch predictors evaluated, there is no branch predictor which can offer significant energy efficiency gains over the overall best predictor for the benchmarks. Additional findings show that the 32KB piecewise linear branch predictor does not perform as well as the O-GEHL branch predictor at a much higher energy consumption and that the hashed perceptron predictor is a close competitor to O-GEHL.

Acknowledgments

This work is supported in part by the National Science Foundation under grant nos. NSF CAREER award CCR-0133634, CNS-0340813, and EIA-0224434.

References

[1] R. Iris Bahar, G. Albera, and S. Manne. Power and performance tradeoffs using various caching strategies. In *Proceedings of the 1998 International Symposium on Low Power Electronics and Design*, pages 64–69. ACM Press, 1998.

[2] D. C. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. *Computer Architecture News*, 25(3):13–25, June 1997.

[3] A. Buyuktosunoglu, T. Karkhanis, D. H. Albonesi, and P. Bose. Energy efficient co-adaptive instruction fetch and issue. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 147–156, New York, NY, USA, 2003. ACM Press.

[4] D. Chaver, M.A. Rojas, L. Pinuel, M. Prieto, F. Tirado, and M. C. Huang. Energy-aware fetch mechanism: trace cache and btb customization. In *Proceedings of the 2005 International Symposium on Low Power Eelctronics and Design*, pages 42–47, New York, NY, USA, 2005. ACM Press.

[5] M. Co, D. A.B. Weikle, and K. Skadron. A break-even formulation for evaluating branch predictor energy efficiency. In *Proceedings of the Workshop on Complexity-Effective Design*, pages 38–49, Madison, WI, June 5, 2005.

[6] D. Folegnani and A. Gonzalez. Energy-effective issue logic. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 230–239, New York, NY, USA, 2001. ACM Press.

[7] D.H. Friendly, S.J. Patel, and Y.N. Patt. Alternative fetch and issue policies for the trace cache fetch mechanism. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, pages 24–33, 1997.

[8] J.S. Hu, N. Vijaykrishnan, M. J. Irwin, and M. Kandemir. Using dynamic branch behavior for power-efficient instruction fetch. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2003)*, 2003.

[9] J.S. Hu, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. Power-efficient trace caches. In *Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition (DATE '02)*, 2002.

[10] S. Hu, M. Valluri, and L. K. John. Effective adaptive computing environment management via dynamic optimization. In *Proceedings of the 3rd Annual International Symposium on Code Generation and Optimization*, pages 63–73, Washington, DC, USA, 2005. IEEE Computer Society.

[11] D. A. Jiménez. Piecewise linear branch prediction. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, page TBD. IEEE Computer Society, 2005.

[12] N.S. Kim, K. Flautner, D. Blaauw, and T. Mudge. Drowsy instruction caches: leakage power reduction using dynamic voltage scaling and cache sub-bank prediction. In *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 219–230. IEEE Computer Society Press, 2002.

[13] J. Lau, S. Schoenmackers, and B. Calder. Transition phase classification and prediction. In *HPCA*, pages 278–289. IEEE Computer Society, 2005.

[14] J. Montanaro, R. T. Witek, K. Anne, A. J. Black, E. M. Cooper, D. W. Dobberpuhl, P. M. Donahue, J. Eno, G. W. Hoeppner, D. Kruckemyer, T. H. Lee, P. C. M. Lin, L. Madden, D. Murray, M. H. Pearce, S. Santhanam, K. J. Snyder, R. Stephany, and S. C. Thierauf. A 160-mhz 32-b 0.5-w cmos risc processor. Technical Report Vol. 9, Digital Technical Journal, Digital Equipment Corporation, 1997.

[15] P. Nagpurkar, C. Krintz, and T. Sherwood. Phase-aware remote profiling. In *Proceedings of the 3rd Annual International Symposium on Code Generation and Optimization*, pages 191–202, Washington, DC, USA, 2005. IEEE Computer Society.

[16] D. Parikh, K. Skadron, Y. Zhang, M. Barcella, and M. Stan. Power issues related to branch prediction. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture*, pages 233–44, Feb. 2002.

[17] E. Rotenberg, S. Bennett, and J. Smith. Trace cache: a low latency approach to high bandwidth instruction fetching. Technical Report 1310, Cs Dept. University of Wisconsin, Madison, April 1996.

[18] E. Rotenberg, S. Bennett, and J. E. Smith. A trace cache microarchitecture and evaluation. *IEEE Transactions on Computers*, 48(2):111–120, 1999.

[19] E. Rotenberg, S. Bennett, and J.E. Smith. Trace cache: A low latency approach to high bandwidth instruction fetching. In *Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 24–35, 1996.

[20] A. Seznec. Analysis of the O-GEometric History Length branch predictor. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, page TBD. IEEE Computer Society, 2005.

[21] T. Sherwood, S. Sair, and B. Calder. Phase tracking and prediction. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 336–349. ACM Press, 2003.

[22] Y. Zhang and J. Yang. Low cost instruction cache designs for tag comparison elimination. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, pages 266–269. ACM Press, 2003.

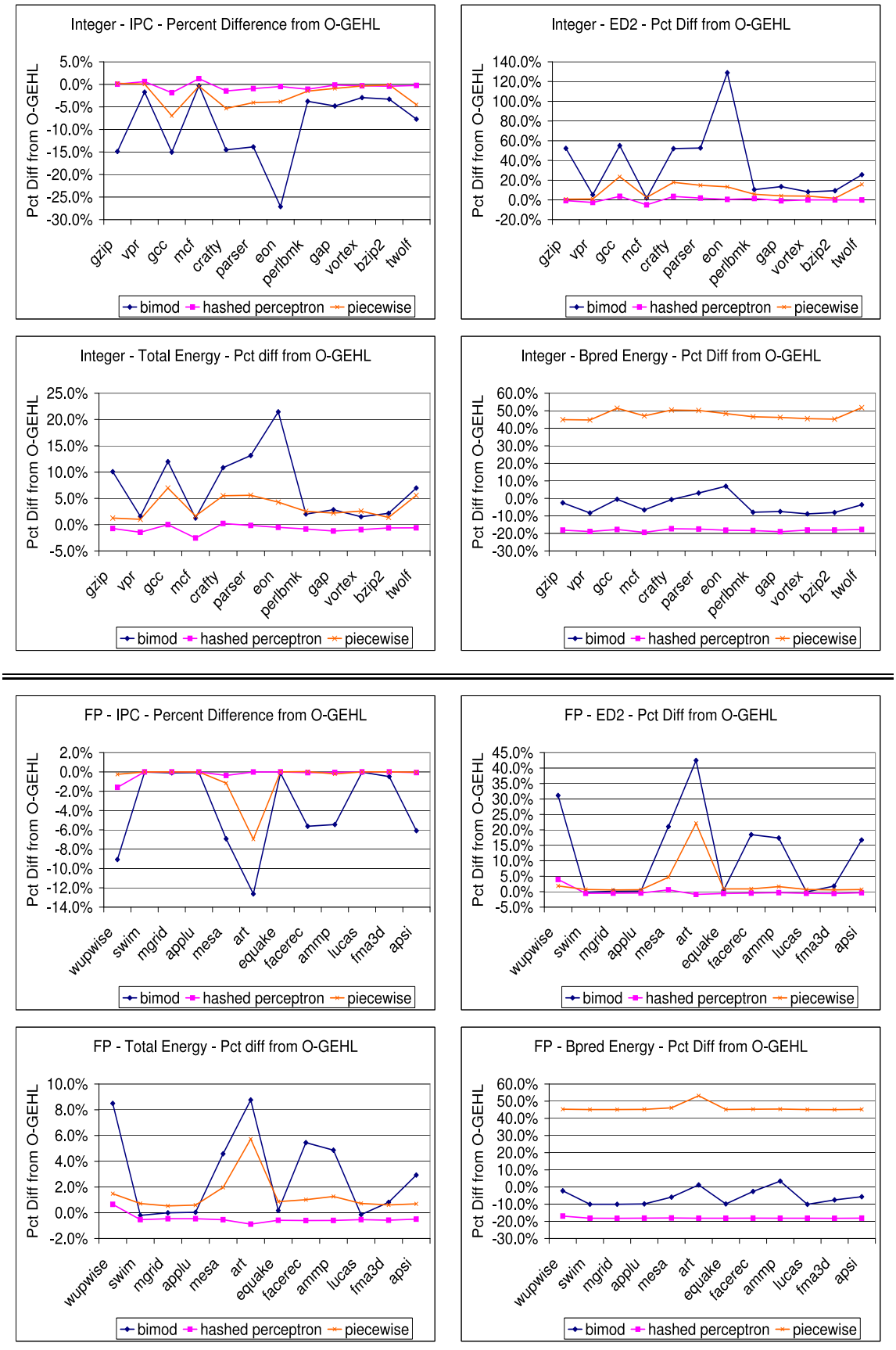

Figure 1: Percent difference from O-GEHL for integer (top) and floating point (bottom) benchmarks. Left-to-Right, Top-to-Bottom: IPC, ED^2, total processor energy, branch predictor energy.

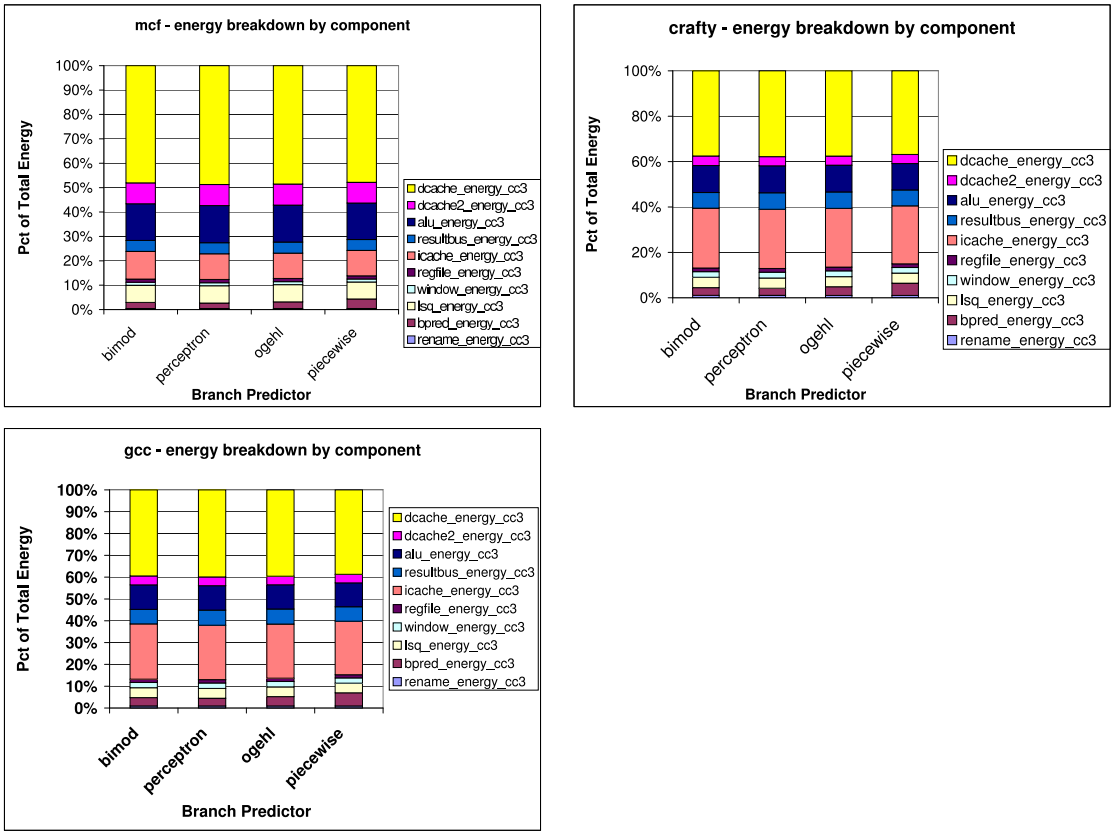

Figure 2: Breakdown of processor energy by component for *mcf*, *crafty*, and *gcc*.

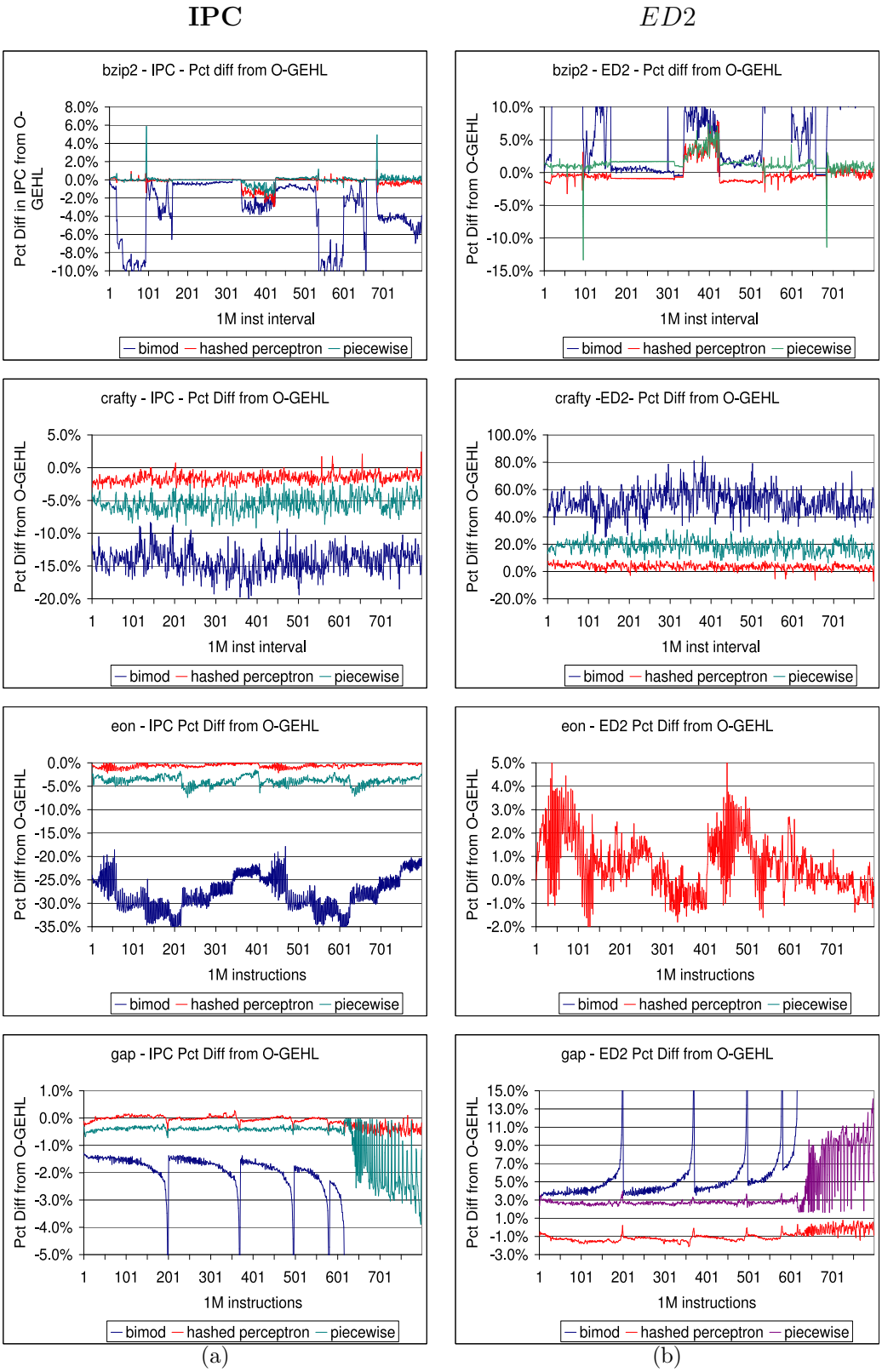

Figure 3: Integer Group 1: Percent difference from O-GEHL for integer benchmarks per 1M instruction interval for (a) IPC and (b) ED^2.

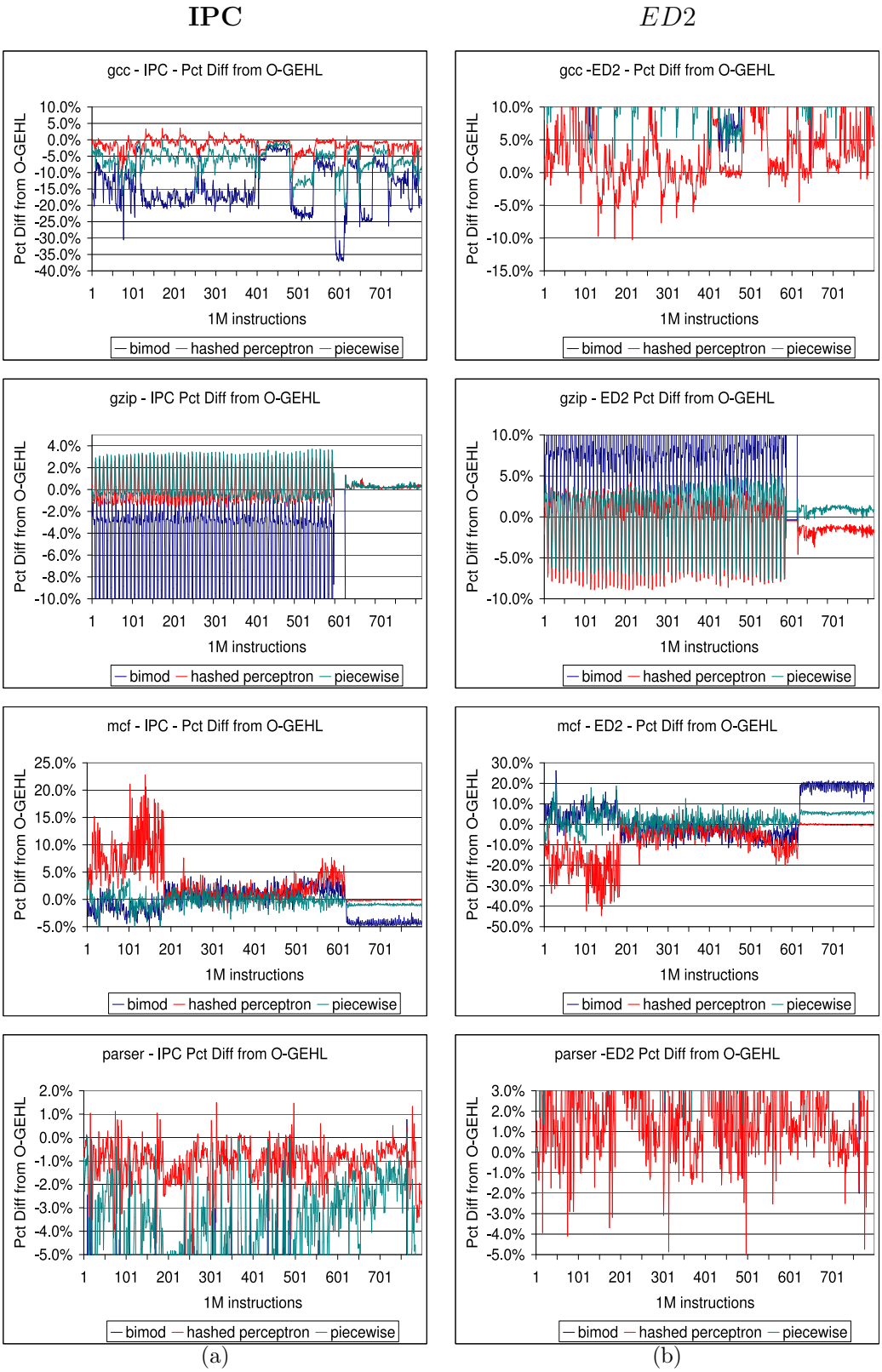

Figure 4: Integer Group 2: Percent difference from O-GEHL for integer benchmarks per 1M instruction interval for (a) IPC and (b) ED^2.

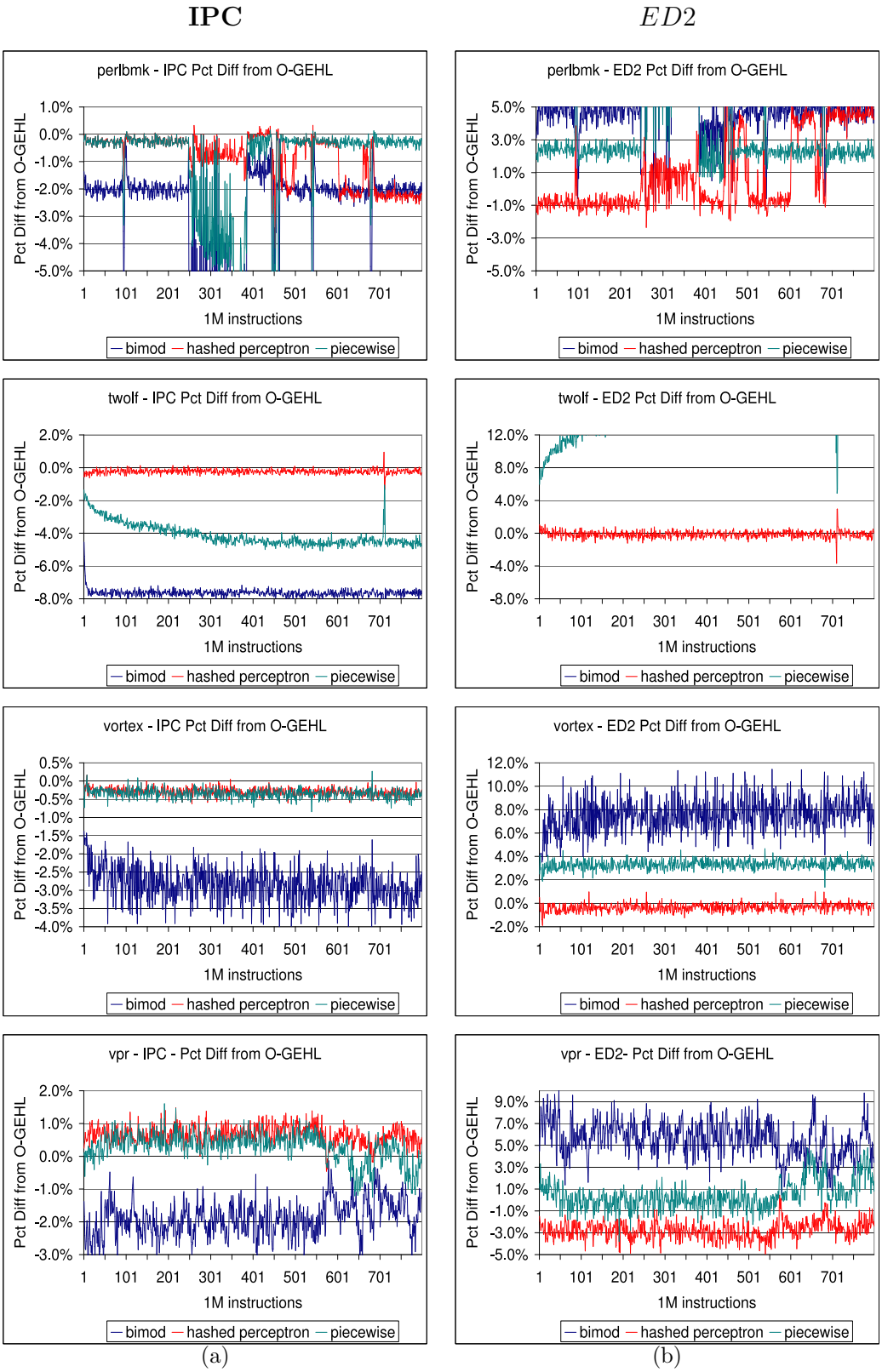

Figure 5: Integer Group 3: Percent difference from O-GEHL for integer benchmarks per 1M instruction interval for (a) IPC and (b) ED^2.

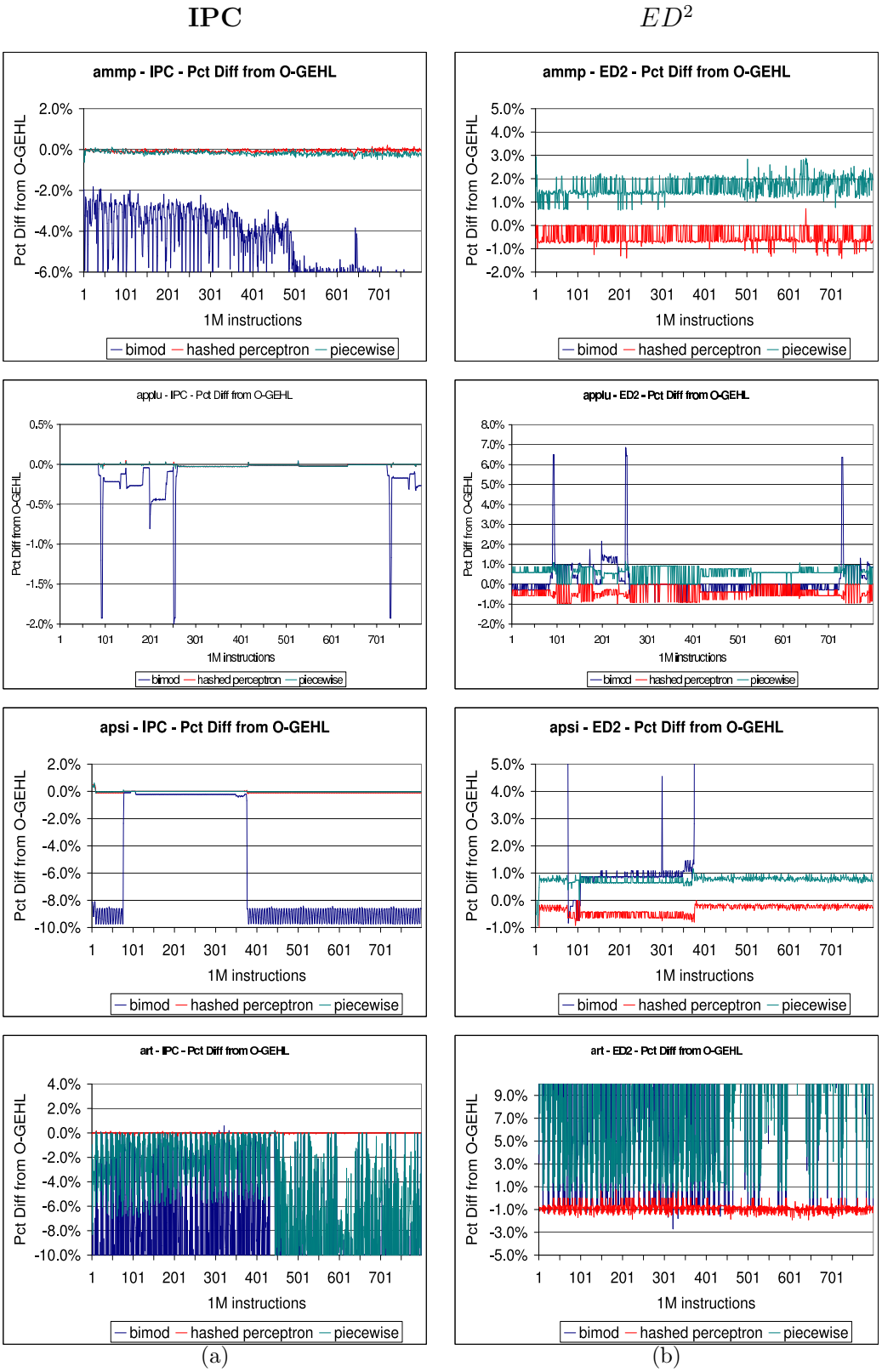

Figure 6: Floating Point Group 1: Percent difference from O-GEHL for floating point benchmarks per 1M instruction interval for (a) IPC and (b) ED^2.

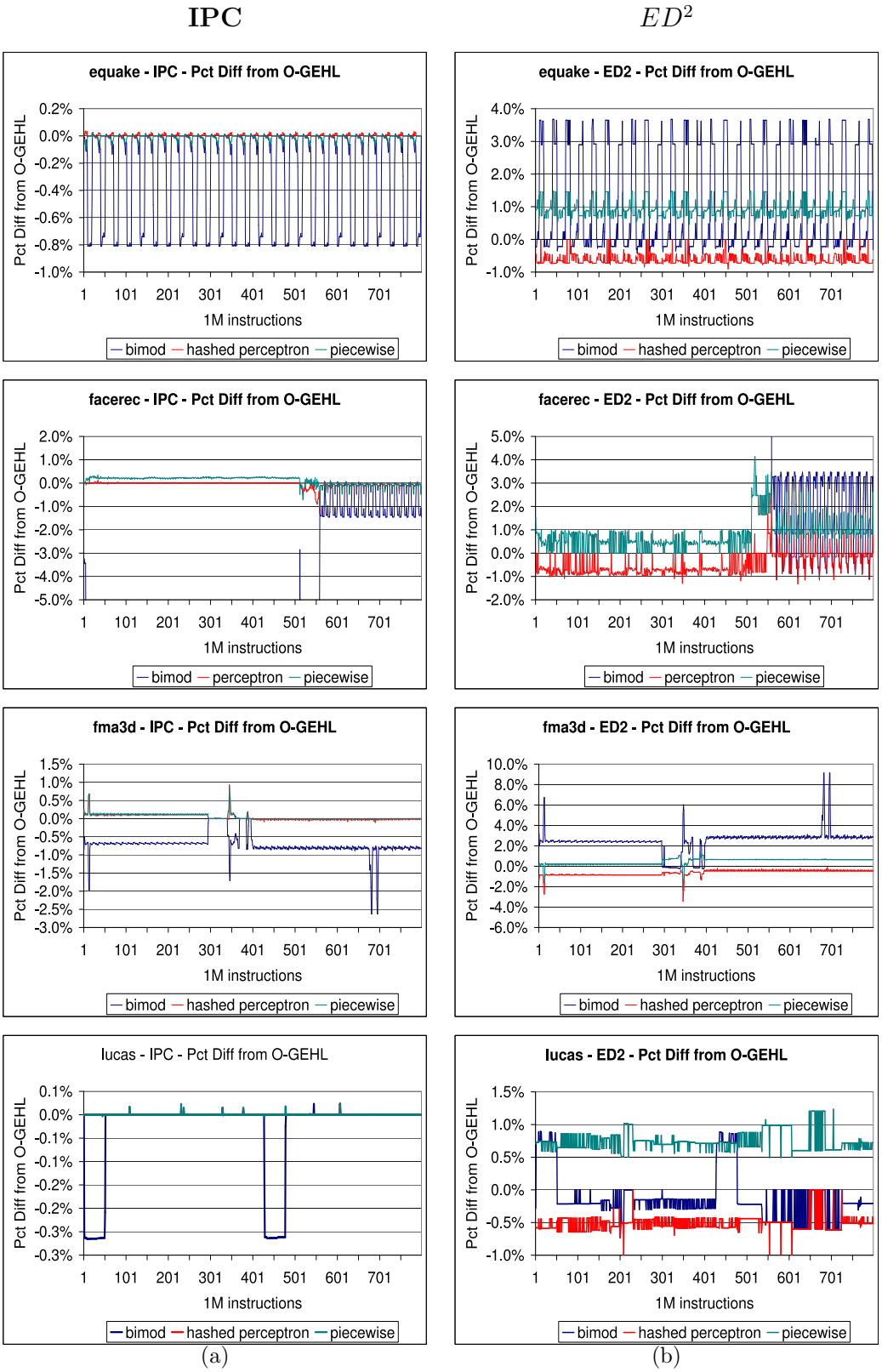

Figure 7: Floating Point Group 2: Percent difference from O-GEHL for floating point benchmarks per 1M instruction interval for (a) IPC and (b) ED^2.

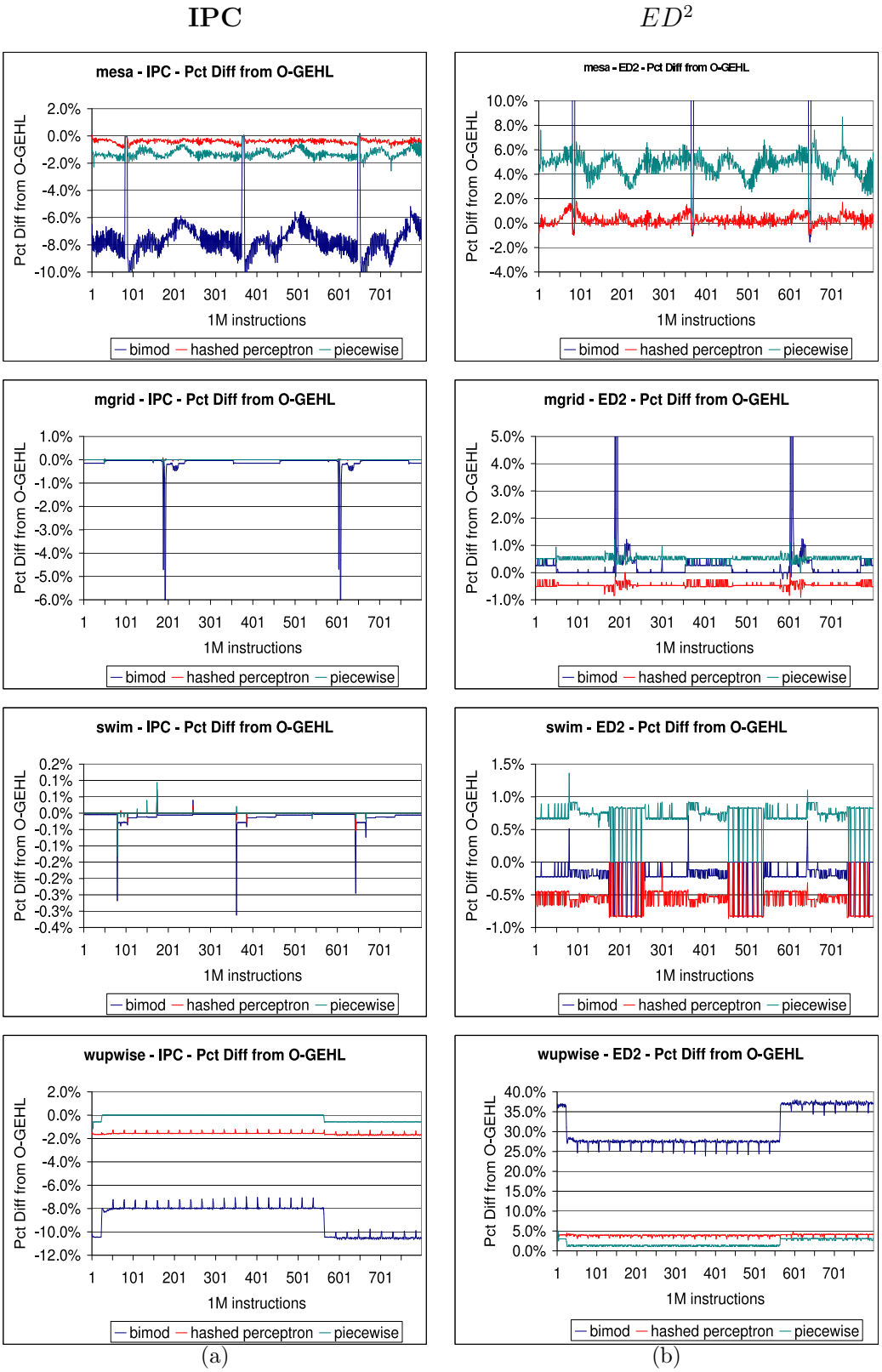

Figure 8: Floating Point Group 3: Percent difference from O-GEHL for floating point benchmarks per 1M instruction interval for (a) IPC and (b) ED^2.

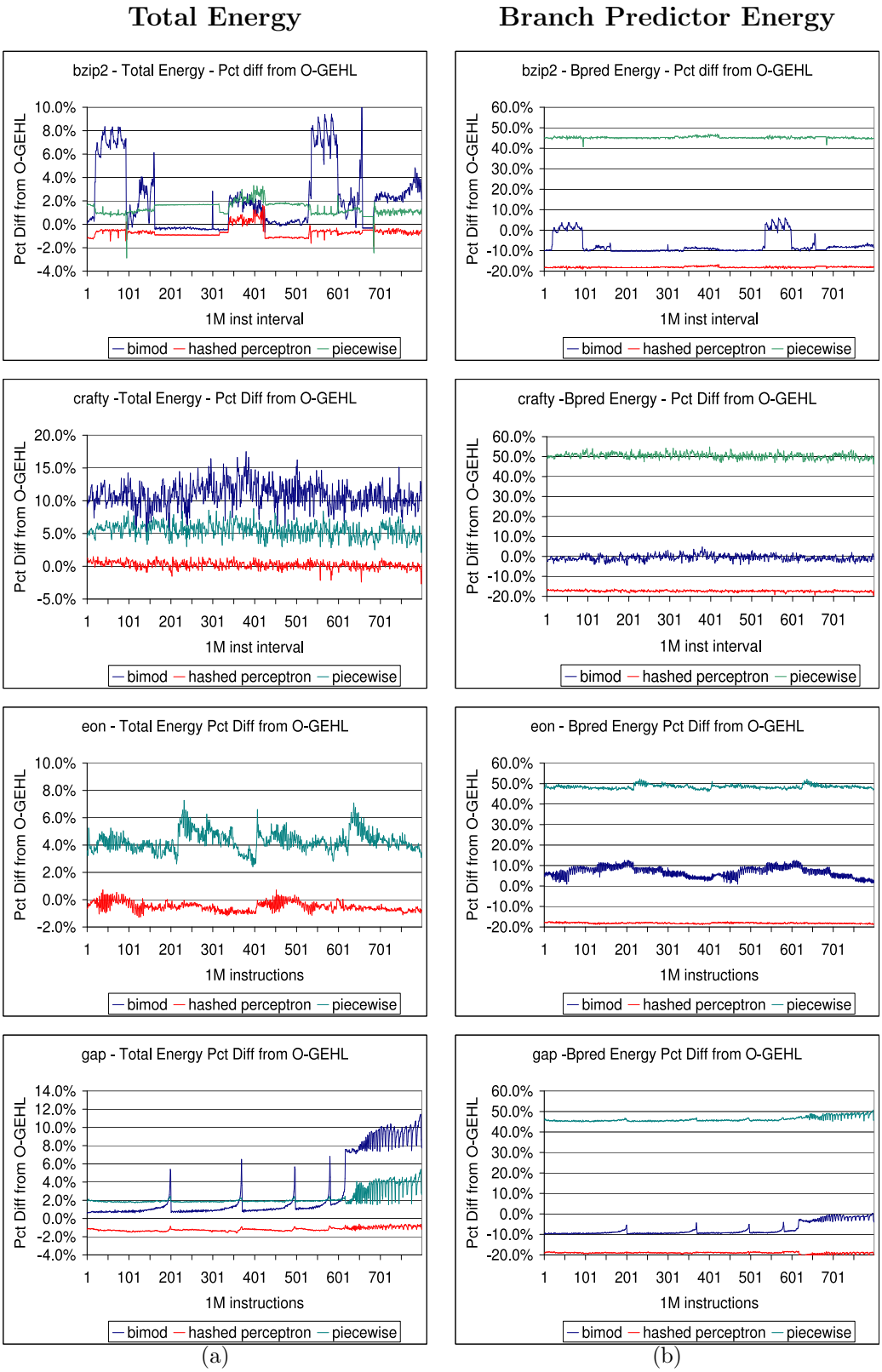

Figure 9: Integer Group 1: Percent difference from O-GEHL for integer benchmarks per 1M instruction interval for (a) total processor energy and (b) branch predictor energy.

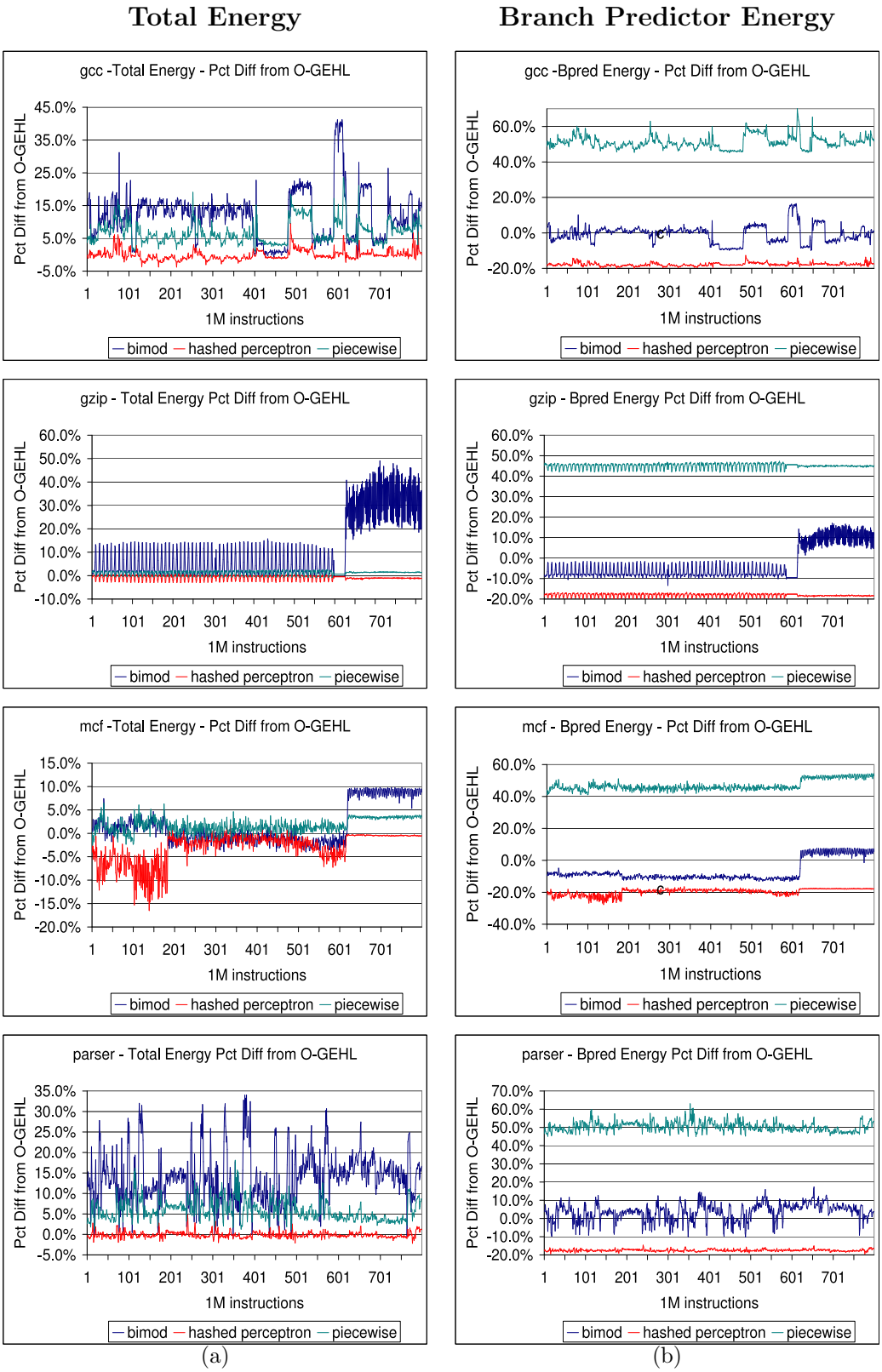

Figure 10: Integer Group 2: Percent difference from O-GEHL for integer benchmarks per 1M instruction interval for (a) total processor energy and (b) branch predictor energy.

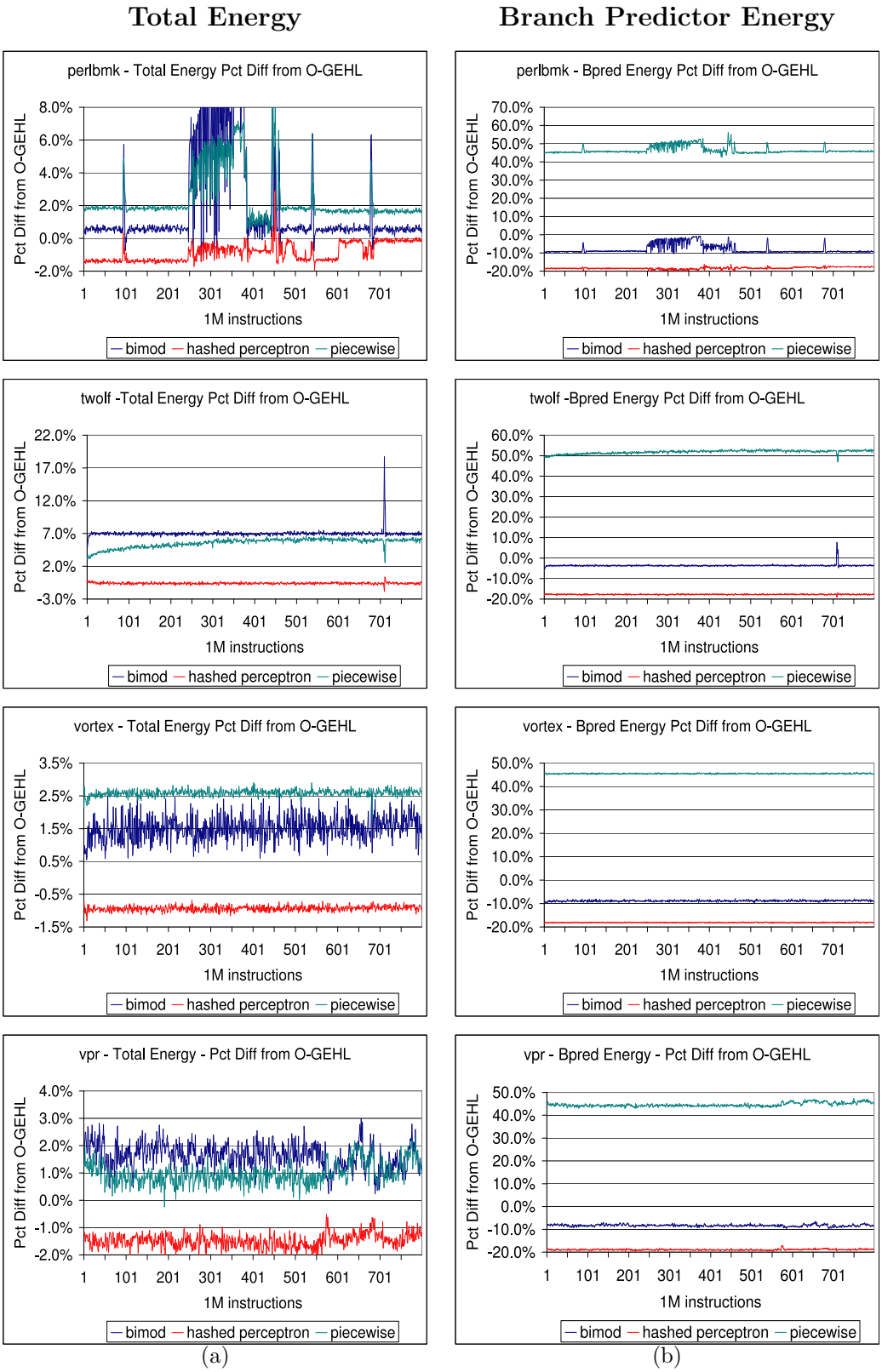

Figure 11: Integer Group 3: Percent difference from O-GEHL for integer benchmarks per 1M instruction interval for (a) total processor energy and (b) branch predictor energy.

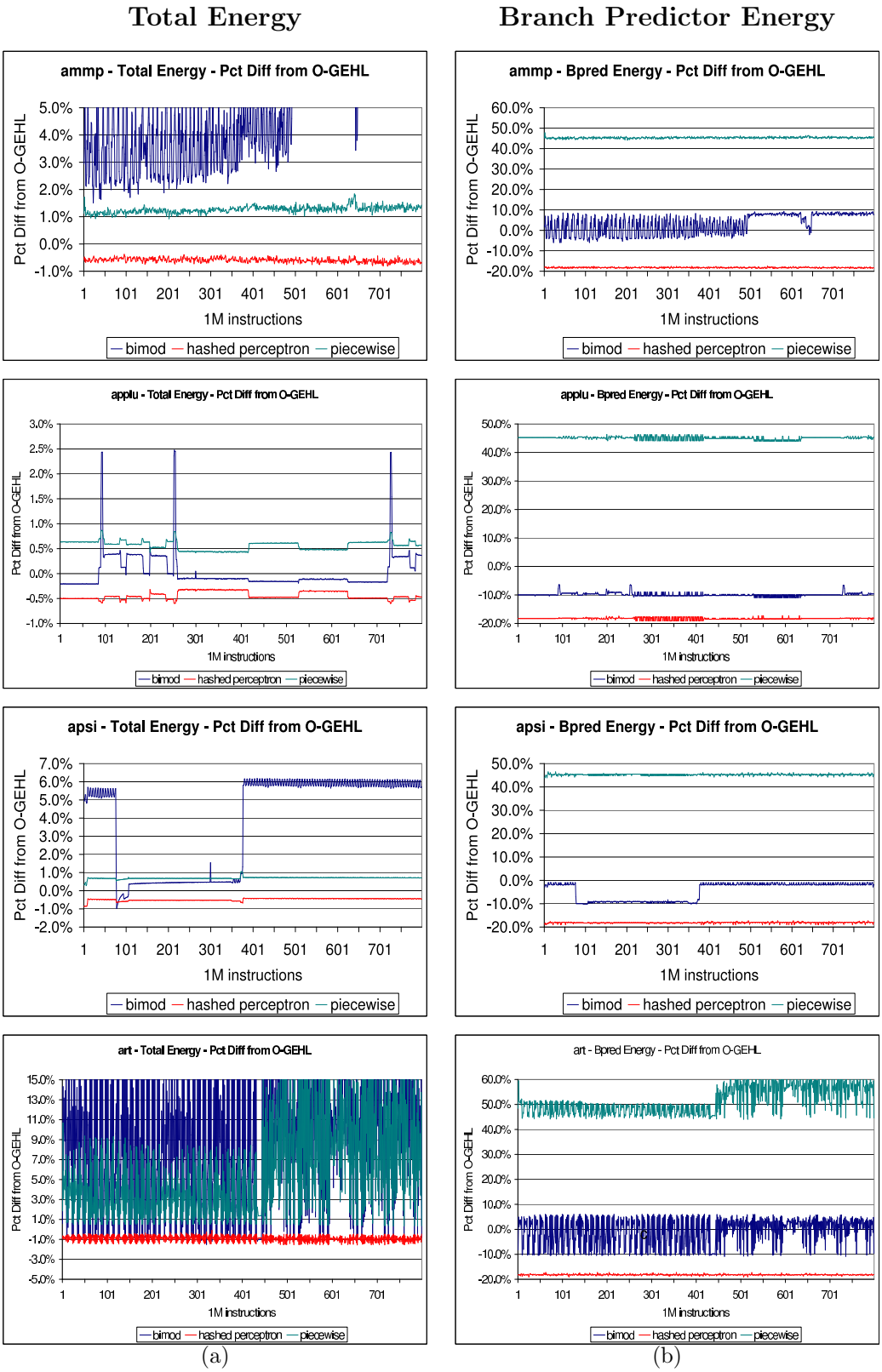

Figure 12: Floating Point Group 1: Percent difference from O-GEHL for floating point benchmarks per 1M instruction interval for (a) total processor energy and (b) branch predictor energy.

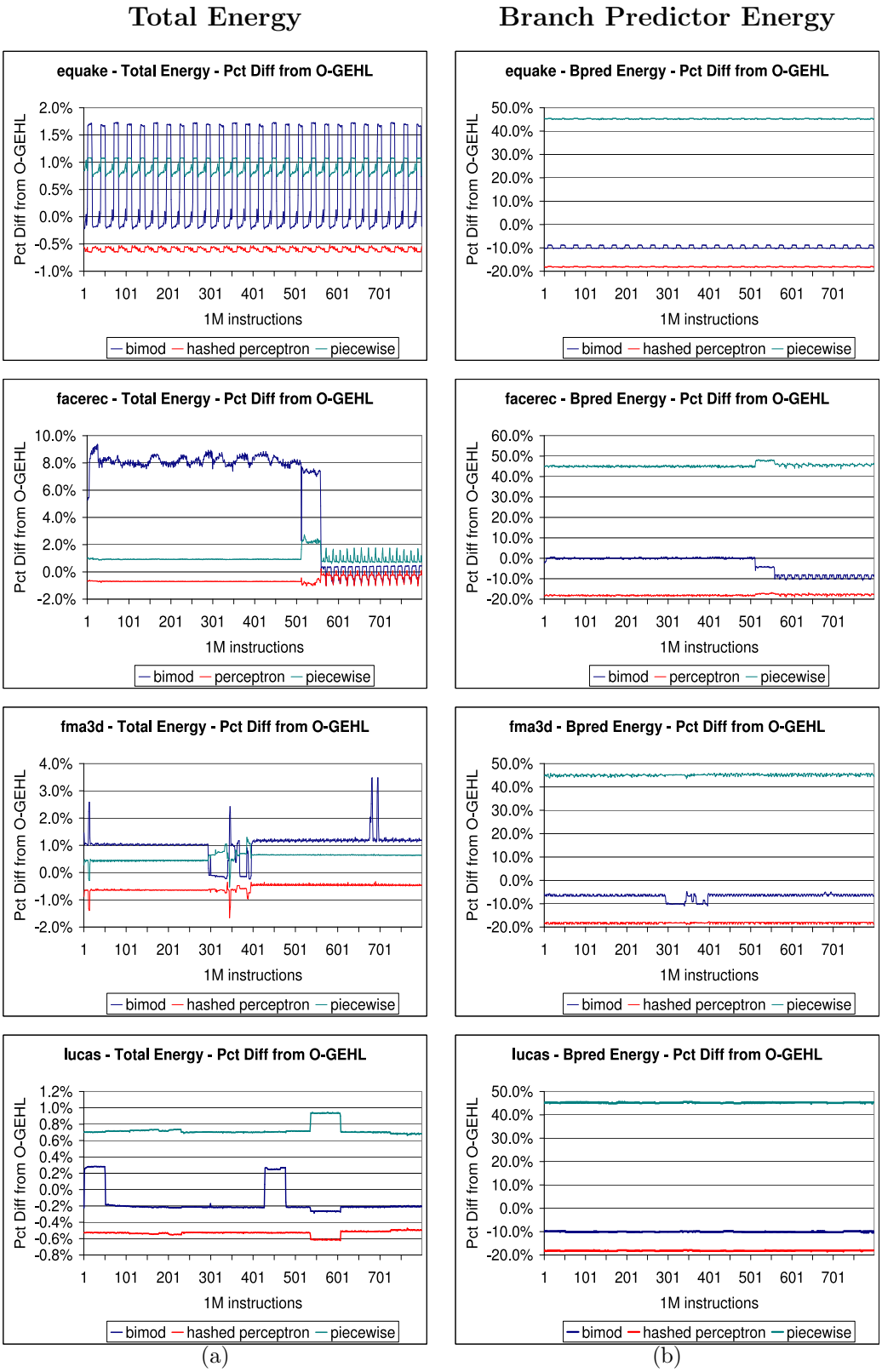

Figure 13: Floating Point Group 2: Percent difference from O-GEHL for floating point benchmarks per 1M instruction interval for (a) total processor energy and (b) branch predictor energy.

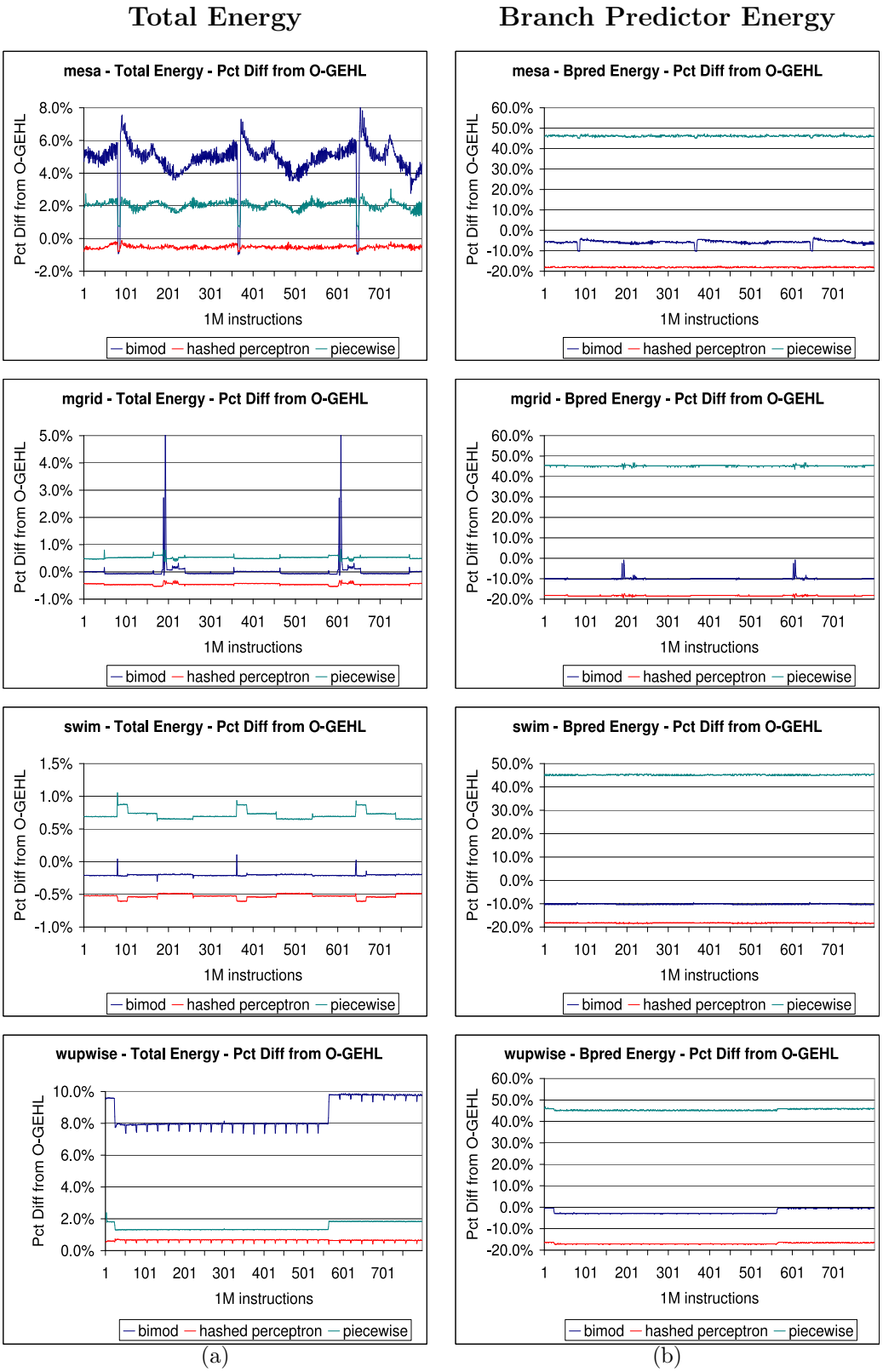

Figure 14: Floating Point Group 3: Percent difference from O-GEHL for floating point benchmarks per 1M instruction interval for (a) total processor energy and (b) branch predictor energy.