MLP-aware Instruction Queue Resizing: The Key to Power-Efficient Performance

Pavlos Petoumenos[1], Georgia Psychou[1], Stefanos Kaxiras[1], Juan Manuel Cebrian Gonzalez[2], and Juan Luis Aragon[2]

[1]Department of Electrical and Computer Engineering, University of Patras, Greece
[2]Computer Engineering Department, University of Murcia, Spain

Abstract. Several techniques aiming to improve power-efficiency (measured as EDP) in out-of-order cores trade energy with performance. Prime examples are the techniques to resize the instruction queue (IQ). While most of them produce good results, they fail to take into account that changing the timing of memory accesses can have significant consequences on the memory-level parallelism (MLP) of the application and thus incur disproportional performance degradation. We propose a novel mechanism that deals with this realization by collecting fine-grain information about the maximum IQ resizing that does not affect the MLP of the program. This information is used to override the resizing enforced by feedback mechanisms when this resizing might reduce MLP. We compare our technique to a previously proposed non-MLP-aware management technique and our results show a significant increase in EDP savings for most benchmarks of the SPEC2000 suite.

1 Introduction

Power efficiency in high-performance cores received considerable attention in recent years. A significant body of work targets energy reduction in processor structures, striving at the same time to preserve the processor's performance to the extend possible. In this work, we revisit a class of microarchitectural techniques that resize the Instruction Queue (IQ) to reduce its energy consumption. The IQ is one of the most energy-hungry structures because of its size, operation (fully-associative matches), and access frequency.

Three proposals by Buyuktosunoglu et al. [1], Folegnani and González [2], and Kucuk et al. [3] exemplify this approach: the main idea is to reduce the energy by resizing the IQ downwards to adjust to the needs of the program using at the same time a feedback loop to limit the damage to performance. The IQ can be physically partitioned into segments that can be completely turned off [1][3] or logically partitioned [2]. While physical partitioning and segment deactivation can be more effective in energy savings, the more sophisticated resizing policy of Folegnani and González minimizes performance degradation. We consider the combination of the physical partitioning of [1] and [3] and the "ILP-contribution" policy of [2] as the basis for our comparisons.

Studying these approaches in more detail we discovered that, in some cases, small changes in the IQ size bring about significant degradation in performance. New found understanding of the relation of the size of the instruction queue to performance, by the work of Chou et al. [7], Karkhanis and Smith [6], Eyerman and Eeckhout [4], Qureshi et al. [8], points to the main culprit for this: Memory-Level Parallelism (MLP) [5]. *In the presence of misses and MLP the single most important factor that affects performance in relation to the IQ size is whether MLP is preserved or harmed.*

While this new understanding seems, in retrospect, obvious and easy to integrate in previous proposals, in fact it requires a new approach in managing the IQ. Specifically, while prior feedback-loop proposals based their decisions on a coarse sampling period measured in thousands of cycles or instructions, an *MLP-aware* technique requires a much more fine-grain approach where decisions must be taken at instruction intervals whose size is comparable to the size of the IQ. The reason for this is that MLP itself exists if misses appear within the instruction window of the processor and must be handled at that resolution. More accurately, MLP exists if the distance between misses is less than the size of the reorder buffer (ROB) [6]. In our case, we use a combined IQ and ROB in the form of a Register Update Unit [9], thus the ROB size defaults to the size of the IQ. In the rest of the paper we discuss MLP with respect to the size of the IQ.

Contributions of this paper:

- We expose MLP (when it exists) as the main factor that affects performance in IQ resizing techniques.
- We propose a methodology to integrate MLP-awareness in IQ resizing techniques by measuring and predicting MLP in *fine-grain* segments of the dynamic instruction stream.
- We propose an example practical implementation of this approach and show that it consistently outperforms in *total* EDP (for the whole processor) an optimized non-MLP-aware technique as well as improving EDP across the board compared to base case of an unmanaged IQ.

Structure of this paper—Section 2 presents related work, while Section 3 motivates the need for IQ resizing using MLP. In Section 4 we present our proposal for MLP-awareness in the IQ resizing and in Section 5 we delve into some details. Section 6 offers our evaluation and Section 7 concludes the paper.

2 Related Work

IQ Resizing Techniques—The instruction window (including possibly a separate reorder buffer, an instruction queue and the load/store queue) has been prime target for energy optimizations. The reason is twofold: first they are greedy consumers of the processor power budget; second, typically these structures are sized to support peak execution in a wide out-of-order processor. Although there are many proposals for IQ resizing, we list here the three main proposals that are relevant to our work.

The proposals by Buyuktosunoglu et al [1] and Kucuk et al [3], resize the IQ by physically partitioning it in segments and disabling and enabling each segment[1] as needed. Both techniques use a feedback loop to control the size of the IQ. The Buyuk-

tosunoglu et al. technique [1] is based on the number of *active* entries (i.e., ready-to-issue entries) per segment to decide whether or not a segment deserves to remain active, while Kucuk et al. [3] argue that the *occupancy* of the IQ (in valid instructions) rather than active entries is a better indication of its required size. In both techniques the IPC is measured periodically; a sudden drop in IPC signals the need for IQ upsizing. However in both approaches there is no other mechanism to revert back to the full IQ size.

The Folegnani and González approach [2] is distinguished by being a logical resizing of the IQ (limiting the range of the entries that can be allocated) not a physical partitioning as the other two. In addition, the feedback mechanism is based on how much the youngest instructions in the IQ contribute to the actual ILP. If they do not contribute much, this means that the size of the IQ can be reduced further. However, there is no way to adapt the IQ to larger sizes except periodically revering back to full size. Nevertheless, the Folegnani and González resizing policy is very good at adjusting the IQ size so as to not harm the ILP in the program.

Modeling of MLP—Karkhanis and Smith presented a first-order model of a superscalar core [6] that deepened our understanding of MLP. This work exposed the importance of MLP in shaping the performance of out-of-order execution. Karkhanis and Smith show that in absence of any upsetting events such as branch mispredictions and L2 misses the number of instructions that will issue (on average) is a property of the program and is expressed by the so-called *IW characteristic* of the program.

The presence, however, of upsetting events, such as L2 misses, decreases the IPC from the ideal point of the IW characteristic depending on the apparent cost of the L2 miss. This is because, a L2 miss drains the pipeline, and eventually stalls it when the instruction that misses blocks the retirement of other instructions [6]. MLP, in this case, spreads the cost of accessing main memory over all the instructions that miss in parallel, making their apparent cost appear much less.

Existing IQ resizing mechanisms focus mainly on the variations of the ILP (effectively moving along the IW characteristic) ignoring what happens during misses. Our work, focuses instead on the latter.

MLP-Aware techniques—Qureshi et al. exploited MLP to optimize cache replacement [8]. MLP in their case is associated with data (cache lines) and the MLP prediction is done in the cache. Data that do not exhibit MLP are given preference in staying in the cache over data that exhibit MLP. Eyerman and Eeckhout exploit MLP to optimize fetch policies for Simultaneous Multi-Threaded (SMT) processors [4]. In their case, MLP is associated with instructions that miss in the cache. Our MLP prediction mechanism is similar to the one proposed by Eyerman and Eeckhout, but because we use it not to regulate the fetch stage, but to manage the entire IQ, we associate MLP information to larger segments of code.

---

1. Sequentially from the end of the IQ in [1] or independently of position in [3].

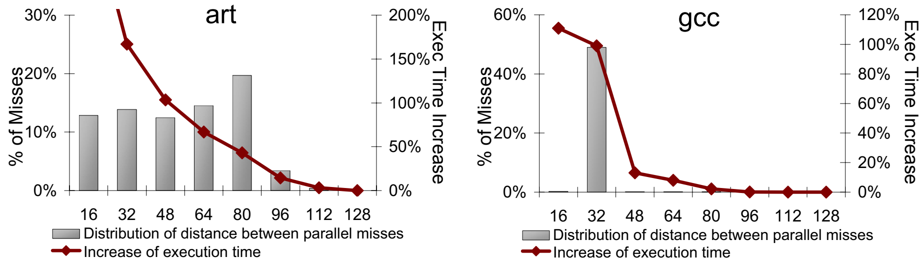

Figure 1. Comparison of the distribution of distances between parallel L2 misses and performance degradation due to IQ resizing for gcc and art

3 MLP and IQ resizing

In this section, we motivate the basic premise of this paper, i.e., that IQ resizing techniques must be aware of the MLP in the program to avoid excessive performance degradation. Figure 1 shows the distribution of the distances between parallel misses in two SPEC2000 programs: art and gcc. We assume here an IQ of 128 entries. The distance is measured in the number of intervening instructions between instructions that miss in parallel. Figure 1 also plots for each distance the increase in execution time if the IQ size is decreased below that distance.

Each time we resize the IQ, we eliminate the MLP with distance greater than the size of the IQ. In art, MLP is distributed over a range of distances, so its execution time is proportionally affected with the decrease of the IQ because we immediately eliminate some MLP. The more MLP is eliminated the harder performance is hit. In contrast, most MLP in gcc is clustered around distance 32. Below this point, we experience a dramatic increase in execution time (100%). For the intermediate IQ sizes, (between the maximum size and 32), execution time increases slowly due to loss of ILP. These examples demonstrate how sensitive performance is with respect to MLP and indicate that an efficient IQ management scheme must focus primarily on MLP rather than ILP.

Another important characteristic of MLP that necessitates a fine-grain approach to IQ management is that the distance among parallel misses changes very frequently. Figure 2 shows a window of 20K instructions from the execution of twolf. At each point the maximum observed distance among parallel misses is shown. Ideally, at each point in time, the IQ size should fit all such distances while at the same time be as small as possible. A blanket IQ size for the whole window, based on some estimation of the average distance between parallel misses, is simply not good enough since it would eliminate all MLP of distance larger than the average.

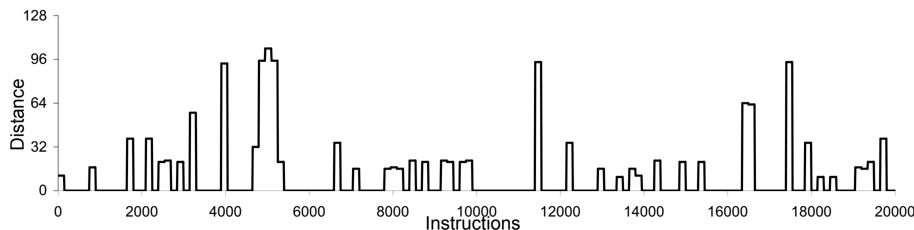

Figure 2. Maximum Distances between parallel misses of twolf

4 Managing the IQ with MLP

Our approach is to quantify MLP opportunities and relate this information back to the instruction stream via a prediction structure. Upon seeing the same instructions again, MLP information stored in the predictor guides our decisions for IQ resizing.

4.1 Quantifying MLP: *MLP-distance*

Our first concern is to quantify MLP opportunities in a way that is useful to IQ resizing. Two memory instructions are able to overlap their L2 misses, if there are no dependencies between them and the number of instructions dispatched between them is less than the size of the IQ. This number of instructions, called *MLP-distance* in [4], is also the basic metric for our management scheme.

A straightforward way to measure the MLP-distance among instructions is to check the LSQ every time a miss is serviced and find the youngest instruction which is still waiting for its data from the L2. This technique does not "fit" in our case, since it can only identify overlapping misses *for the current size of the IQ*. To overcome this problem we need to check for misses that could potentially overlap over a number of instructions, as many as the *maximum* number of instructions that can fit in the unmanaged IQ. Always keeping information for as many instructions as the maximum IQ size, partially defeats the purpose of resizing the IQ. Thus, instead of keeping information for each individual instruction, we keep aggregate information for *instruction segments*, groups of sequentially dispatched instructions (*which coincide with the segments that make up a physically partitioned IQ*).

This MLP information is kept in a small cyclic buffer —which we call *MLP distance buffer* or *MDB*— with as many entries as the maximum number of IQ segments (Fig.3). MDB is not affected by IQ resizing. A new entry is allocated for each segment, but in contrast to the real IQ entries, MDB entries are evicted only when it fills. This means that MDB "retires" a segment only when newly dispatched instructions are farther away than the size of the non-managed IQ, and thus could not possibly execute in parallel with the retiring segment. Our approach to measure MLP distance, is similar to [4] but based on segments for increased power efficiency. Each time an instruction causes an L2 miss, the corresponding MDB segment is marked as also having caused a miss. Upon eviction of an entry, the MDB is searched for the other entries which have caused L2 misses. If there are such entries, this means that there could be possible MLP among them. We update each entry's *MLP-distance field* with the distance —measured in segments— from the youngest entry with a miss, if this distance is longer than the previously recorded value. MDB is infrequently accessed and it is only processed whenever segments which caused L2 misses are retired.

The MLP-distance is not an entirely accurate estimation of actual MLP. To reside at the same time in the IQ is not the only requirement for two instructions to overlap their misses i.e. possible dependencies between the instructions may cause their misses to be isolated. In any case, the "actual" MLP-distance will be less than or equal to the value produced by our approach. This in turn means we might miss some opportunities for downward IQ resizing but we will not incur performance degradation due to loss of MLP. Our experiments showed that for most benchmarks falsely assuming parallel

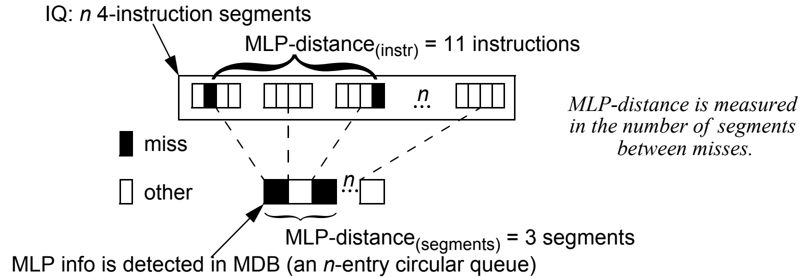

Figure 3. MLP Distance Buffer (MDB).

misses causes few overestimations of the MLP-distance. Considering the simplicity of our mechanism, these results indicate a satisfactory level of performance.

Measuring the MLP-distance allows us to control the IQ size for the relevant part of the code so as to not hurt MLP. We need, however, to relate this information back to the instruction stream so the next time we see the same part of the code we can react and correctly set the size of the IQ.

4.2 Associating MLP-Distance with Code

For the purpose of managing the IQ in a MLP-aware fashion, we dynamically divide program execution into fragments and associate MLP-distance information with these fragments. Execution fragments should be comparable in size to the size of the IQ. Much shorter fragments would be sub-optimal since the information they carry will be used to manage the whole IQ, which contains multiple such fragments. This could lead to very frequent and —many times— conflicting resizing decisions. Longer fragments, such as program phases, also fail to provide us with the fine-grain information we need to quickly react to fast-changing MLP-distances in the instruction stream.

To achieve the desired balance in the size of the fragments we use the notion of *superpaths.* A superpath is nothing more than an aggregation of sequentially executed basic blocks and loosely corresponds to a trace (in a trace cache) [13] or a hotspot [12]. The MLP-distance of a superpath is assigned by the MDB: when all of the MDB entries belonging to a superpath are "retired," the longest MLP-distance stored in these entries is selected to update the MLP-distance of the superpath. Note that the instructions which establish this MLP-distance do not have to belong to the same superpath. An MLP-distance that straddles a number of superpaths affects all of them (if it is the maximum observed MLP-distance in each of them).

The next time the same superpath is observed in dispatch, its stored information is retrieved to manage the IQ. A more detailed discussion about superpaths and how we actually implemented the aforementioned mechanisms can be found in Section 5.

4.3 Resizing Policy

When we start tracking a superpath in dispatch, we check whether we have stored information about its behavior, including its MLP-distance information. If there is such information then we have an indication about the minimum IQ size which will not affect the MLP of this superpath. In many cases, however, there is no MLP-distance information available or the MLP-distance does not restrict IQ downsizing.

The question is how much can we downsize the IQ is such cases? As explained in Section 3, an efficient IQ resizing scheme has to find the IQ size that minimizes energy consumption without hurting *performance*. This means that besides not hurting MLP we must also protect ILP.

This gap can be filled by any of the existing IQ resizing techniques. For example, the ILP-feedback approach of [2] can provide a target IQ size that does not hurt ILP while the MLP-aware approach judges whether this size hurts MLP and if so it overrides the decision. For the rest of this paper the ILP-feedback information will be provided by the decision making mechanism in Folegnani and González [2]. This mechanism examines the participation of the youngest segment of the IQ to the IPC within a specific number of cycles. If the contribution of the youngest part is not important, namely the number of instructions that issue from this segment is below a *threshold*, the IQ size is decreased. In our case, we deactivate the youngest segment when it empties.

The main idea in the Folegnani and González work is that if a segment contributes very little to ILP, deactivating it would not harm performance. However, this holds only for ILP —not for MLP. In other words, even if the contribution of a segment in issued instructions is very small, it can still have a significant impact on performance, if *any* of its issued instructions is involved in MLP. It is exactly for such cases where MLP-awareness makes all the difference. Further, in the Folegnani and González work, once the IQ is downsized, there is no way to detect whether the situation changes —all we can see is the contribution to ILP of the *active* portion of the IQ. Thus, periodically, the IQ is brought back to its full size, and the downsizing process starts again. In our case, the existence of MLP automatically upsizes the IQ, to a size that does not harm MLP.

5 Practical Implementation

5.1 IQ Segmentation

To allow dynamic variation of the IQ size, we divide the IQ in a number of independent parts referred as segments. For example, we use an 128-entry IQ partitioned into eight, sixteen-entry segments. Bitline segmentation is used to implement the resizing of the structure [10]. The structure of the IQ follows the one in [2]. The IQ is a circular FIFO queue, with every new instruction inserted at the tail; retiring instructions are removed from the head. The difference in our case, is that individual segments can be deactivated. A segment is deactivated if instructions from the youngest segment contribute less than *threshold* instructions in a *quantum* of time (1000 cycles) and a segment is reactivated every 5 *quanta*. This inevitably leads to constraints that have to be met during the resizing process, similarly to those faced by Ponomarev et al. [11]: downsizing of the IQ is only permitted if there are no instructions left to commit in the segment being removed and upsizing is constrained to activate segments that come after all the instructions currently residing in the IQ.

5.2 Superpaths

Our basic IQ management unit, the superpath, is characterized by its size and its first instruction. Sequential basic blocks are organized into superpaths at the dispatch stage and they contain at least as many instructions as the IQ size. Superpath creation ends

when we encounter a basic block which is at the head of a previously created superpath, in order to reduce both the number and *the overlap* of superpaths. For each newly created superpath we allocate an entry in a small hardware cache which keeps information about the superpath, as well as information about its MLP-distance. After performing an exploration of the design space, we chose a 4-way, 16-set configuration (indexed by the lower-order bits of the start address of the superpath).

We store 28 bits per superpath entry: 20 bits of the starting address (lowest-order bits above the indexing bits), the MLP-distance prediction (4 bits, again quantized in multiples of 16) and its confidence counter (3 bits) and a valid bit. For our 4x16 cache, our storage requirements add up to 1792 bits. According to CACTI [14] this structure contributes 9.5 mW to the total power consumption of the processor, which is a reasonable power overhead compared to the power consumption of the IQ (8.9W for the configuration described in Section 6).

5.3 MLP-distance Prediction

When all instructions of superpath commit, the stored superpath information is updated with the MLP-distance information of this particular execution. Different executions of a superpath are generally not identical in terms of MLP, so what we want to associate with the superpath is a dominant value for its MLP-distance. To manage this, in addition to keeping an MLP-distance prediction for each superpath entry, we employ a 3-bit saturating confidence counter which indicates our confidence that the stored MLP-distance is also the dominant value. The confidence counter is incremented for each MLP-distance update which agrees with the current prediction and decremented for each update which disagrees. When it reaches zero we replace it.

6 Evaluation

6.1 Experimental Setup

For our experiments we use a detailed cycle accurate simulator that supports a dynamic superscalar processor model and WATTCH power models [15]. The configuration of the simulated processor is described in Table 1. We use a subset of the SPEC2000 benchmarks, containing benchmarks with more than one long-latency load per 1K instructions for the smallest cache size we utilize. Benchmarks with even less misses present no interest for our work, since without misses our mechanism falls back to the baseline ILP-feedback technique. All benchmarks are run with their reference input. We simulate 300M instructions after skipping 1B instructions for all benchmarks except for vpr, twolf, mcf where we skip 2B instructions and ammp where we skip 3B instructions.

6.2 Results Overview

The experiments were performed for four different cache sizes. Three metrics are used in our evaluation: total processor energy, execution time and the energy × delay product. We first present the effects of MLP-awareness on the baseline ILP-oriented mechanism and then a direct comparison of the two mechanisms, the ILP-aware and the combination of the ILP-feedback and ILP/MLP techniques. All results (except otherwise noted) are normalized to the base case of an unmanaged IQ.

Parameter	Configuration
Fetch/Issue/Commit width	4 instructions per cycle
BTB	1024 entries,4-way set-associative
Branch Predictor	Combining, bimodal + 2 Level, 2 cycle penalty
Instruction Queue (combined with ROB)	128 entries
Load/Store Queue	64 entries
L1 I-cache	16 KB, 4-way, 64 bytes block size
L1 D-cache	8 KB, 4-way, 32 bytes block size
Unified L2 cache	256 KB/512 KB/1MB/2MB, 8-way, 64 bytes block size
TLB	4096 entry (I), 4096 entry(D)
Memory	8 bytes wide, 120 cycles latency
Functional Units	4 int ALUs, 2 int multipliers, 4 FP ALUs, 2 FP multiplier

Table 1. Configuration of simulated system

6.2.1 Effects of MLP-awareness

Figure 4 depicts the EDP *geometric mean* of all benchmarks for two different thresholds: an aggressive threshold of 768 instructions and a conservative threshold of 256 instructions. Note how difficult it is to improve the geometric mean of the whole-processor EDP with IQ resizing. This depends a great deal on the portion of the total power budget taken up by the IQ. In our case, this is a rather conservative 13.7%, so in the ideal case —significant energy savings and no performance degradation— we expect to approach this percentage in EDP savings. Indeed, this is what we achieve with our proposal.

As shown in the graph, the ILP-feedback technique works marginally with a conservative threshold while its combination with the MLP-aware mechanism improves the situation only slightly. However, with much more aggressive resizing, the ILP-feedback technique seriously harms performance and despite the larger energy savings, yields a worse EDP across all cache configurations. In this case, the incorporation of the MLP-aware mechanism can readily improve the results and turn loss into significant benefit, approaching 10% EDP improvement

As long as the ILP-feedback technique does not hurt the MLP, it yields benefits. When that changes, the performance loss is unacceptable. This hazard is avoided when the MLP mechanism is used because it works as a safety net. With the help of MLP mechanism, resizing can be pushed to the limit. .

6.2.2 Direct Comparison of ILP- and ILP/MLP-aware techniques

In this section, we compare the behavior of the two approaches, each evaluated for a configuration which minimizes its average EDP. An additional consideration was to find configurations that do not harm EDP over the base case for any benchmark. This, however, is not possible for the ILP-feedback technique, lest we are content with marginal EDP improvements. Thus, for the ILP-feedback we remove this restriction and we simply select the threshold that minimizes average EDP, which is 256-instructions. The

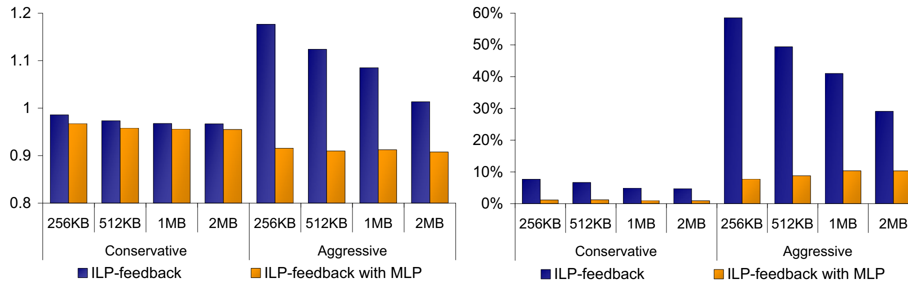

Figure 4. Average (geometric mean) Normalized EDP (left) and Performance Degradation (right) for ILP-feedback and ILP-feedback with MLP-awareness

ILP-feedback with the MLP mechanism can be pushed much harder as it is evident from Fig.4 with the 768-instruction threshold. However, EDP worsens over the base case for two programs (applu and art), even though the average EDP is minimized. The threshold that gives the second best average EDP —giving up less than 2% over the previous best case— for the combined ILP/MLP mechanism is the 512-instruction threshold which satisfies our requirement for EDP improvement across all benchmarks.

Figures 5, 6, 7 illustrate the normalized EDP, execution time increase and energy savings respectively for the "best" thresholds for each mechanism. The end result is that the very aggressive resizing of the ILP/MLP technique harms performance comparably to the conservative ILP-feedback technique but at the same time manages to reduce the IQ size more and produce significantly higher energy savings. This results in an EDP for the ILP/MLP technique that is consistently better than the EDP of the ILP-feedback technique, almost doubling the benefit on average (6.1-7.2% compared to 1.4-3.4% of the ILP-feedback technique). The added benefits of MLP-aware management diminish slightly with cache size, since with fewer misses we have less opportunities for MLP. Finally, note that the performance degradation of the ILP/MLP technique is kept at reasonable levels, while even for the conservative ILP-feedback it can vary considerably more (e.g., mcf execution time increases 67%-69%).

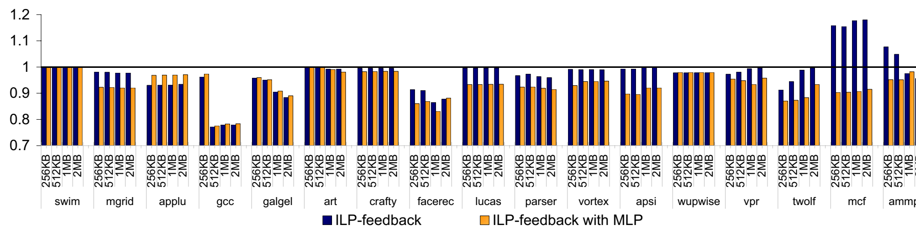

Figure 5. Normalized Energy-Delay Product for 'best' configurations: ILP-Feedback (256 threshold) and ILP-Feedback with MLP (512 threshold)

7 Conclusions

In this paper, we revisit techniques for resizing the instruction queue aiming to improve the power-efficiency of high-performance out-of-order cores. Prior approaches resized the IQ paying attention primarily to ILP. In many cases this results in considerable loss of performance while the energy gains from the IQ are bounded

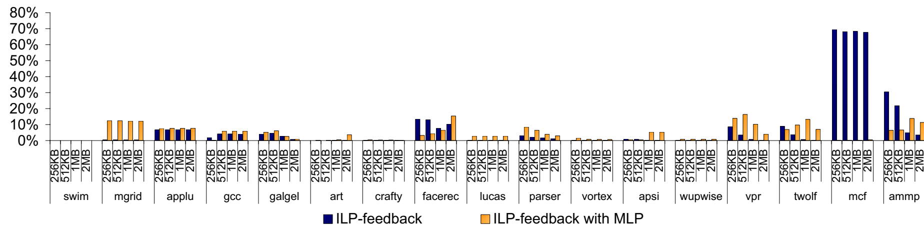

Figure 6. Execution Time Increase for ILP-Feedback and ILP-Feedback with MLP (each for its best configuration)

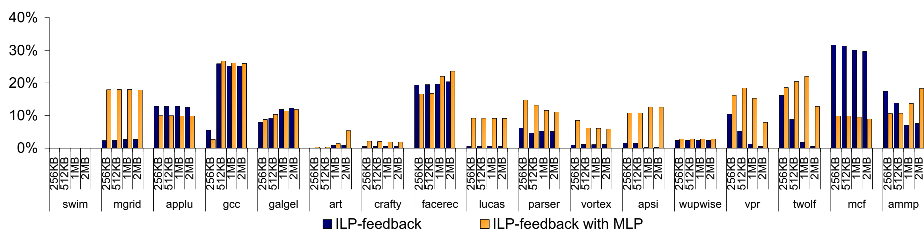

Figure 7. Normalized Energy Savings for ILP-Feedback and ILP-Feedback with MLP (each for its best configuration)

with respect to the energy of the whole processor. The result is that EDP improves in some cases but worsens in others making such techniques inconsistent.

The culprit for this is MLP —Memory-Level Parallelism. Resizing the IQ can reduce the amount of MLP in programs with serious consequences on performance. With this realization, we set out to provide a technique that can be applied on top of previous IQ resizing techniques. Our technique, detects possible MLP at runtime and uses prediction to guide IQ resizing decisions. Because we need to manage the whole IQ, our basic unit of management is a sequence of basic blocks, called superpath, comparable in the number of instructions to the maximum IQ size. MLP information is associated with superpaths and is used to override resizing decisions that might harm the MLP of the superpath. In absence of misses and MLP, resizing of the IQ is performed using already existing techniques.

Our results show that we can manage the IQ, considerably better than in prior approaches yielding consistently better EDP over the base case. At the same time, we can push the resizing of the IQ much more aggressively (to achieve better energy savings) knowing that our safety-net mechanism protects the MLP of the program and will not inordinately harm performance.

Acknowledgments—This work is supported by the EU FP6 Integrated Project - Future and Emerging Technologies, Scalable computer ARCitecture (SARC) Contract No. 27648, and the EU FP6 HiPEAC Network of Excellence IST-004408. The equipment used for this work is a donation by Intel Corporation under Intel Research Equipment Grant #15842.

8 References

[1] A. Buyuktosunoglu, D. Albonesi, S. Schuster, D. Brooks, P. Bose, P. Cook. A Circuit Level Implementation of an Adaptive Issue Queue for Power–Aware Microprocessors. In Proc. of Great Lakes Symposium on VLSI Design, 2001.

[2] D. Folegnani and A. González. Energy-effective issue logic. In Proc. of the International Symposium on Computer Architecture, 2001.

[3] G. Kucuk, K. Ghose, D. V. Ponomarev, and P. M. Kogge. Energy-Efficient Instruction Dispatch Buffer Design for Superscalar Processors. In Proc. of the International Symposium on Low Power Electronics and Design, 2001.

[4] S. Eyerman and L. Eeckhout. A Memory-Level Parallelism Aware Fetch Policy for SMT Processors. In Proc. of the International Symposium on High Performance Computer Architecture, 2007.

[5] Andrew Glew. MLP yes! ILP no! In ASPLOS Wild and Crazy Ideas Session '98. 1998

[6] T.S. Karkhanis and J.E. Smith. A first-order superscalar processor model. In Proc. of the International Symposium on Computer Architecture, 2004.

[7] Yuan Chou, B. Fahs, and S. Abraham. Microarchitecture optimizations for exploiting memory-level parallelism. In Proc. of the International Symposium on Computer Architecture, 2004.

[8] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt. A Case for MLP-Aware Cache Replacement. In Proc. of the International Symposium on Computer Architecture, 2006.

[9] G. S. Sohi and S. Vajapeyam. Instruction Issue Logic for High-Performance, Interruptable Pipelined Processors. In Proc. of the International Symposium on Computer Architecture, 1987.

[10] K. Ghose and M. B. Kamble. Reducing Power in Superscalar Processor Caches using Subbanking, Multiple Line Buffers, and Bit Line Segmentation. In Proc. of the International Symposium on Low Power Electronics and Design, 1999.

[11] D. Ponomarev, G. Kucuk, and K. Ghose. Dynamic Resizing of Superscalar Datapath Components for Energy Efficiency. IEEE Transactions on Computers, 2006.

[12] A. Iyer and D. Marculescu. Power aware microarchitecture resource scaling Design. Automation and Test in Europe, 2001.

[13] E. Rotenberg, J. Smith, and S. Bennett. Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching. In Proc. of the International Symposium on Microarchitecture, 1996.

[14] D. Tarjan, S Thoziyoor, and N. P. Jouppi. CACTI 4.0. Hewlett-Packard Laboratories Technical Report #HPL-2006-86, 2006.

[15] D. M. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In Proc. of the International Symposium Computer Architecture, 2000.

[16] R. Gonzalez and M. Horowitz, Energy Dissipation in General Purpose Microprocessors. IEEE J. Solid-State Circuits, 1996.

[17] V. Zyuban, D. Brooks, V. Srinivasan, M. Gschwind, P. Bose, P. N. Strenski, and P. G. Emma. Integrated Analysis of Power and Performance of Pipelined Microprocessors. IEEE Transactions on Computers, 2004.