Control Speculation for Energy-Efficient Next-Generation Superscalar Processors

Juan L. Aragón, *Member*, *IEEE Computer Society*, José González, *Member*, *IEEE Computer Society*, and Antonio González, *Member*, *IEEE Computer Society*

Abstract—Conventional front-end designs attempt to maximize the number of "in-flight" instructions in the pipeline. However, branch mispredictions cause the processor to fetch useless instructions that are eventually squashed, increasing front-end energy and issue queue utilization and, thus, wasting around 30 percent of the power dissipated by a processor. Furthermore, processor design trends lead to increasing clock frequencies by lengthening the pipeline, which puts more pressure on the branch prediction engine since branches take longer to be resolved. As next-generation high-performance processors become deeply pipelined, the amount of wasted energy due to misspeculated instructions will go up. The aim of this work is to reduce the energy consumption of misspeculated instructions. We propose *Selective Throttling*, which triggers different power-aware techniques (fetch throttling, decode throttling, or disabling the selection logic) depending on the branch prediction confidence level. Results show that combining fetch-bandwidth reduction along with select-logic disabling provides the best performance in terms of overall energy reduction and energy-delay product improvement (14 percent and 10 percent, respectively, for a processor with a 22-stage pipeline and 16 percent and 13 percent, respectively, for a processor with a 42-stage pipeline).

Index Terms—Control speculation, energy-aware systems, low-power design, processor architecture.

1 Introduction

Continuing advances in semiconductor technology lead to more powerful processors in which power dissipation has become an important design concern. Power dissipation translates directly into heat, which may cause chip malfunction due to some failures such as thermal runaway, junction fatigue, and electro-migration diffusion [21]. To keep temperature under control, high performance processors require the use of very expensive cooling schemes, which may significantly impact the final cost of the system. For mobile and embedded processors, battery life is another key design concern.

Conventional front-end instruction delivery—fetch, decode, rename, and dispatch—accounts for a significant portion of the overall energy consumed in a typical processor. For instance, the front-end was reported to consume about 25 percent of the total energy in the Alpha 21264 [9]. Front-end designs try to maximize the number of instructions supplied to the back-end using all the available bandwidth and resulting in some useless energy consumption and resource utilization. Furthermore, current processor designs use very deep pipelines to increase processor frequency and then performance [11], [12], [22], [23]. Augmenting the pipeline depth increases the number of cycles that a branch takes from being fetched to be resolved, which increases the number of misspeculated instructions for each branch misprediction. The result is that a very significant part of the dissipated power (around 30 percent, on average, based on data reported in [3]) may be wasted due to misspeculated instructions.

The goal of this work is to reduce the energy consumed by misspeculated instructions without hurting performance by means of *Selective Throttling*. Based on the confidence level assigned to each branch prediction, different processor structures are dynamically throttled: fetch unit, decode unit, or selection logic. For low confidence predictions, the most aggressive throttling heuristics are applied early in the pipeline (at the expense of a potentially higher performance penalty if the prediction turns out to be correct). On the other hand, for high confidence predictions, less aggressive techniques, both in terms of energy reduction and potential performance degradation, are applied depending on the confidence level.

Prior related work, such as *Pipeline Gating* [17], proposed an all-or-nothing mechanism which is very sensitive to the goodness of the confidence estimator since performance is highly penalized if the confidence estimation turns out to be incorrect (see Section 5.2 for further details).

In [3], we proposed different throttling policies that are selectively applied based on the branch confidence estimation. In addition, these policies have a certain degree of adaptability (i.e., complete fetch stall versus stalling fetch every four cycles). We also proposed a new throttling technique which disables the selection of instructions that are control dependent on a low confident branch.

- *J.L. Aragón is with the Departamento Ingenieria y Tecnología de Computadores, Universidad de Murcia, Facultad de Informática-Campus Espinardo, 30100 Murcia, Spain. E-mail: jlaragon@ditec.um.es.*
- *J. González is with the Intel Barcelona Research Center, UPC, Intel Labs Barcelona, Barcelona, Spain. E-mail: pepe.gonzalez@intel.com.*
- *A. González is with the Computer Architecture Department, Universitat Politècnica de Catalunya, c/ Jordi Girona 1-3, Mòdul D6, 08034 Barcelona, Spain. E-mail: antonio@ac.upc.es.*

Manuscript received 22 June 2004; revised 9 Aug. 2005; accepted 24 Aug. 2005; published online 20 Jan. 2006
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0212-0604.

The new contributions of this work are the following:

- A more up-to-date baseline architecture is chosen with a pipeline depth of 22 stages, similar to that of the Intel Pentium 4 [8], to evaluate our proposal for different throttling heuristics.
- The Sensitivity Analysis section is extended to evaluate the effect of our proposal when considering future design trends such as *superpipelining* [11], [12], [22], [23], considering pipeline lengths of up to 42 stages.
- The effect of using better branch predictors, such as the hybrid predictor implemented in the Alpha 21264, is evaluated.
- The effect of different issue window (IW) sizes is also evaluated. Since IW is responsible for a significant fraction of the overall energy consumption, we show that *Selective Throttling* can significantly reduce overall energy consumption and improve Energy-Delay product for processors with large IWs.
- Energy savings provided by *Selective Throttling* for different blocks of the processor are analyzed. In particular, it reduces the energy consumption of the I-cache by 40 percent and the energy consumption of the issue window by 18 percent.

The rest of the paper is organized as follows: Section 2 reviews some related work. Section 3 analyzes the power and energy consumption due to misspeculated instructions. The proposed *Selective Throttling* mechanism is described in Section 4. Section 5 analyzes performance and energy reductions of our proposal. Finally, Section 6 summarizes the main conclusions of this work.

2 Related Work

There have been many proposals for reducing the performance degradation caused by branch mispredictions. Some approaches try to improve branch prediction accuracy [1], [7], [18], [25]. Others try to minimize performance degradation by fetching and/or executing multiple paths [2], [13], [16], [24]. However, analyzing how misspeculated instructions influence energy consumption has not received so much attention.

Pipeline Gating [17] prevents wrong-path instructions from entering the pipeline and wasting energy. This is accomplished by using a confidence estimator to assess the quality of branch predictions [10], [14]. Confidence estimation is used to determine whether the processor is likely to fetch instructions that will not commit. The number of unresolved low confidence branches is used to determine when and for how long to gate. If this number exceeds a threshold, the fetch or decode stage is stalled, but previously fetched or decoded instructions continue flowing through the pipeline.

In [4], a fetch throttling mechanism is proposed to reduce power dissipation by enabling or disabling the fetch or decode stages based on certain heuristics. The authors introduce two control-flow heuristics to selectively turn the fetch stage off during three cycles.

JIT Instruction Delivery [15] dynamically limits the number of "in-flight" instructions. Whenever the instruction count exceeds a threshold, the fetch stage is stalled.

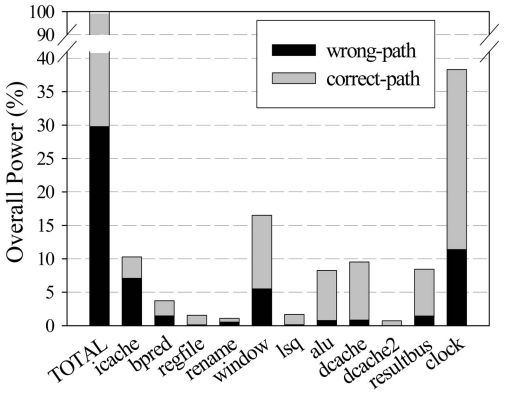

Fig. 1. Overall power breakdown for correct-path and wrong-path instructions.

Unlike *Pipeline Gating*, a confidence estimator is not used. Recently, a fetch gating mechanism driven by issue queue utilization has been proposed [6]. It is combined with dynamic issue queue size adaptation in order to further reduce overall energy consumption.

3 Energy Consumption of Wrong-Path Instructions

Processors use control flow speculation to predict the outcome of conditional branches. Speculation greatly improves performance, but it also increases energy consumption in case of misprediction. Furthermore, the amount of speculative activity increases with deeper pipelines, resulting in systems that waste a significant portion of energy. In [4], the number of incorrectly fetched instructions was reported to account for up to 80 percent of all fetched instructions.

In this section, we quantify the contribution of these useless instructions on the overall power dissipation. The SPECint2000 benchmark suite is analyzed with the *Wattch v1.02* power-performance simulator [5] for a baseline processor with 22 stages (see Section 5.1 for details on simulation methodology and processor configuration).

Fig. 1 shows the overall power breakdown for each block of the processor averaged across all benchmarks for both correct-path and wrong-path instructions. The significant impact that branch prediction causes in deeper pipelines can be seen, in terms of power dissipation: About 30 percent of the total processor power is dissipated by misspeculated instructions. This represents an upper bound of the power reduction that can be achieved with the techniques proposed in this paper. As expected, the fetch stage ("`icache`"+"`bpred`"), which dissipates about 14 percent of the overall power, wastes 57 percent of its power (9 percent of the total) due to wrong-path instructions. Similarly, the decode stage ("`rename`" + a fraction of "`regfile`"[1] + a fraction of "`window`"[2]) also contributes

1. According to the Wattch power model, the "`regfile`" activity counter is updated in the decode stage to account for reads of ready operands and also in the commit stage to account for register writes of committed values.

2. Wattch's "`window`" activity counter is updated in the decode stage to write ready operands into the ROB (that contains the physical registers), at *issue* to read ready operands from physical registers, and, finally, at *write-back* to write results into the corresponding physical registers.

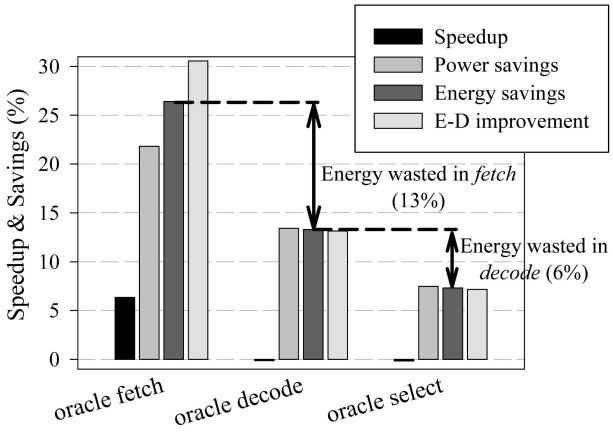

Fig. 2. Oracle fetch, decode, and select savings.

significantly to wasted power. The back-end of the processor, including the issue logic, wake-up and selection logic, ROB, LSQ, functional units, data caches, and the result bus still waste about 9 percent of the overall power as a consequence of misspeculated instructions. Finally, 12 percent of the power is wasted by the clock due to useless activities.

Because of the way the Wattch simulator measures dissipated power, it is not easy to directly determine the fraction corresponding to misspeculated instructions for each pipeline stage. Therefore, in order to precisely determine the energy wasted by wrong-path instructions on a per-stage basis, we also ran the following experiments:

1. *Oracle fetch*: Only fetches correct-path instructions for conditional branches. In case of misprediction, the processor does not fetch the misspeculated path.
2. *Oracle decode*: Uses realistic fetch, but only decodes correct-path instructions for conditional branches.
3. *Oracle select*: Uses realistic fetch and decode, but only selects for issuing correct-path instructions for conditional branches.

In particular, the difference between the energy savings of *oracle fetch* and *oracle decode* represents an upper bound of the energy wasted exclusively within the fetch stage. Analogously, the difference between the energy savings of *oracle decode* and *oracle select* constitutes an upper bound of the energy wasted only within the decode stage. Fig. 2 shows the average speedup as well as power and energy savings and energy-delay improvement for the three oracle experiments. It can be seen that 13 percent of the overall energy is wasted exclusively in the fetch stage, whereas 6 percent of the overall energy is wasted in the decode stage.

Note that the *oracle fetch* experiment provides overall energy savings similar to those reported in Fig. 1 although now only conditional branches are considered. However, since *Selective Throttling* will be applied only to conditional branches, the *oracle fetch* experiment provides a better upper bound on potential savings. Overall potential savings in power, energy, and energy-delay for *oracle fetch* are 22 percent, 26 percent, and 31 percent, respectively. Note that there is a speedup of 6 percent, mainly due to reduction in I-cache pollution and resource contention. Fig. 2 also shows that the potential energy savings by gating at decode are 13 percent (*oracle decode* experiment) and 7 percent (*oracle select* experiment) if we gate them at issue.

4 Selective Throttling

Selective Throttling attempts to reduce dynamic power dissipation and energy consumption while minimizing performance degradation. This is accomplished by limiting the number of misspeculated instructions fetched, decoded, and issued, which decreases useless activities in the processor.

Selective Throttling relies on branch confidence estimation to initiate a particular action. We propose using different throttling policies depending on the confidence level, with the goal of obtaining an optimal trade-off between power and performance.

4.1 Power-Aware Knobs

Various throttling knobs with different potential impact on performance are used:

- *Fetch throttling*: Reduces the fetch bandwidth to a half, a quarter, or it completely stalls the fetch unit.
- *Decode throttling*: Reduces the decode bandwidth to a half, a quarter, or it completely stalls the decode unit.
- *Selection throttling*: Avoids the selection for execution of those instructions control-dependent on a low confidence branch.

The last knob, *Selection throttling*, is a novel scheme that attempts to reduce the power dissipated when misspeculated instructions are executed. In order to do that, the selection of instructions control-dependent on a low confidence branch is disabled. This avoids useless activity in the issue logic. On the other hand, it has a minor impact on performance when it is erroneously applied (i.e., activated for correctly predicted branches) since instructions following the branch have already been fetched, decoded, and dispatched anyway and they can be issued immediately after the branch is executed.

The three knobs can be combined. For example, we could define a throttling policy that reduces the fetch bandwidth to a half, the decode bandwidth to a quarter, and avoids the selection of instructions fetched after a low confidence branch.

Fig. 3 depicts a block diagram of *Selective Throttling*. Limiting the fetch and decode bandwidth is achieved by alternating full activity cycles with idle cycles. For instance, in an 8-wide issue processor, reducing the fetch bandwidth to a half implies that eight instructions are fetched in a given cycle and zero instructions are fetched in the next one.

The *fetch* and *decode throttling* knobs work as follows: Four 2-bit registers are used to store the current and last fetch/decode bandwidths, as depicted in Fig. 3. Each register stores one of the following four values: full bandwidth, one half, one quarter, or idle.

In the fetch stage, if a branch is predicted with low confidence, new bandwidth parameters for both fetch and decode units are set according to the defined throttling policy and the previous bandwidths are backed up. In the *writeback* stage, if a low confidence branch turns out to be

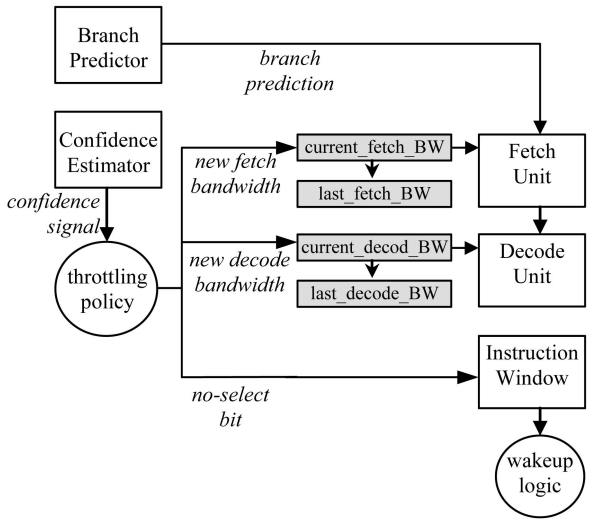

Fig. 3. Main components of *Selective Throttling*.

mispredicted, both fetch/decode bandwidths are restored to the last value. Otherwise, if the low confidence branch was correctly predicted, we restore fetch/decode bandwidths when the branch being *writebacked* is the one that set the current bandwidth. This last condition is necessary to allow multiple nested throttling policies for different low confidence branches, as explained in Section 4.2.

The *selection throttling* knob is fairly straightforward to implement. It just requires one bit in each issue window entry to disable selection. Fig. 4 shows how the *no-select* bit is used to avoid raising the request signal used by the selection logic. This heuristic works as follows: When a branch prediction has low confidence, all of the following instructions set the *no-select* bit to 1 when introduced in the issue window (IW). When the low confidence branch reaches the writeback stage, if it was mispredicted, all instructions after the branch are flushed from the pipeline and we have successfully avoided some useless activities and reduced power. Otherwise, if the low confidence branch was correctly predicted, all the *no-select* bits in the IW are reset to 0 using the "reset" line shown in Fig. 4. Resetting the *no-select* bits for all instructions in the IW may seem a conservative approach since *Selective Throttling* allows multiple nested low confidence branches. However, the reset is done very infrequently: Experimental results show that less than 0.5 percent of total executed instructions are conditional branches correctly predicted, but labeled as low confidence by the simulated 8 KB *BPRU* confidence estimator.

Note that the power overhead of the four 2-bit registers and the additional *no-select* bits, although small, has been modeled and taken into account in our experiments.

4.2 Confidence-Based Classification of Branches

The energy-efficiency of *Selective Throttling* strongly depends on the accuracy of the confidence estimator. If the confidence estimator assigns low confidence to a prediction and it turns out to be correct, *Selective Throttling* results in a power-performance penalty: Power dissipation is not reduced, whereas performance is degraded. On the other hand, if a prediction is assigned high confidence and the branch turns out to be mispredicted, performance is not degraded, but energy is wasted.

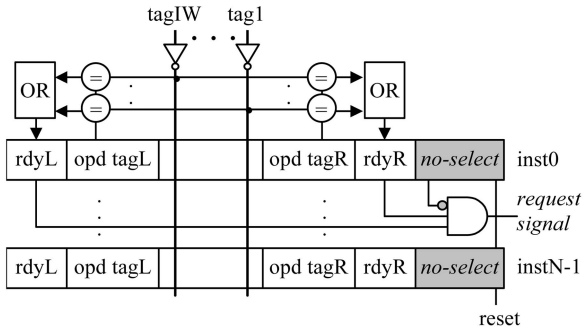

Fig. 4. Wake-up logic [19] modified to implement the *selection throttling* knob.

Thus, in order to obtain an optimal power-performance trade-off, instead of using the conventional two states (high/low) provided by a typical confidence estimator, we propose classifying each branch prediction into the following four states, based on the value of the confidence counter stored in each entry of the confidence estimator:

1. very-high confidence branches (VHC),
2. high confidence branches (HC),
3. low confidence branches (LC),
4. very-low confidence branches (VLC).

This classification allows the processor to better tune the different knobs to control power-performance. Note also that *Selective Throttling* applies a particular throttling scheme not only based on the confidence estimator, but also on the status of other in-flight branches. For instance, after applying a throttling policy for an LC branch, if a later branch is assigned VLC before the first branch is resolved, a more restrictive throttling policy may be applied.

4.3 Evaluated Confidence Estimators

According to the metrics introduced by Grunwald et al. [10], a good confidence estimator should have high SPEC and PVN.[3] This led us to use the confidence estimator proposed for the *Branch Prediction Reversal Unit* (*BPRU*) scheme [1], which makes use of predicted data values to assess the confidence of branch predictions. The *BPRU* is based on the fact that data values can be useful to determine branch mispredictions. Some mispredictions can be avoided by selectively reversing the prediction based on some processor parameters, in particular, predicted data values. The *BPRU* is effective to detect some pathological branches with a high probability of being mispredicted, such as hard to predict *if-then-else* structures.

The *BPRU* has been modified as follows to adapt it to the purpose of this work. It uses a tagged table. Branches that miss in that table use the saturating counter of the underlying branch predictor to provide the confidence estimation. If a branch is predicted as either *weakly* taken or *weakly* not-taken, the branch is assigned LC. As expected, this change increments the SPEC metric at the expense of

3. SPEC is defined as the fraction of incorrect predictions labeled as low confidence, whereas PVN is defined as the fraction of low confidence branches that are finally mispredicted.

TABLE 1
Benchmark Characteristics

benchmarks		input set	simulated instruc. (Mill.)	dyn.cond. branches (Mill.)	gshare 8 KB miss-rate
Spec95	compress	40000 e 2231	170	13	10.2%
	gcc	genrecog.i	145	19	9.2%
	go	9 9	146	15	19.7%
Spec2000	bzip2	input.source 1	500	43	8.0%
	crafty	test (modified)	437	38	7.7%
	gap	test (modified)	500	56	4.0%
	gzip	input.log 1	500	52	8.8%
	parser	test (modified)	500	64	6.8%
	twolf	test	258	21	11.2%
	vpr	test	500	45	7.4%

TABLE 2
Configuration of the Simulated Processor

Fetch engine	Up to 8 instr/cycle, 2 taken branches, 2 cycles of misprediction penalty.
BTB	1024 entries, 2-way
Execution engine	Issues up to 8 instr/cycle, 128-entries RUU, 64-entries load/store queue.
Functional Units	8 integer alu, 2 integer mult, 2 memports, 8 FP alu, 1 FP mult.
L1 Instr-cache	64 KB, 2-way, 32 bytes/line, 1 cycle hit lat.
L1 Data-cache	64 KB, 2-way, 32 bytes/line, 1 cycle hit lat.
L2 unified cache	512 KB, 4-way, 32 bytes/line, 6 cycles hit latency, 18 cycles miss latency.
Memory	8 bytes/line, virtual memory 4 KB pages.
TLB	128 entries, fully associative.
Technology	0.18㎛, Vdd = 2.0 V, 1200 Mhz.

Simulations with more modern technologies, such as 90nm, 1.2V, and 3 GHz, will lead to similar conclusions since we are reducing the activity of many units/structures in the processor, not their capacitance.

reducing the PVN metric. Simulations for an 8 KB *gshare* predictor along with an 8 KB *BPRU* obtain an average SPEC of 60 percent and a PVN of 45 percent for the evaluated benchmarks.

For comparison purposes, we have also evaluated *Pipeline Gating* using an 8 KB *JRS* confidence estimator with an MDC-threshold of 12. This confidence estimator obtains an average SPEC of 90 percent and a PVN of 24 percent, which is consistent with results reported in [10].

5 Experimental Results

5.1 Simulation Methodology

To evaluate the energy-efficiency of *Selective Throttling*, we run the 10 benchmarks from the SPECint95 and SPECint2000 suites that exhibit the highest branch misprediction rates. All benchmarks were compiled with maximum optimizations (`-O4 -fast`) by the Compaq Alpha compiler and were run using a modified version of the *Wattch v1.02* power-performance simulator [5]. All benchmarks were run to completion, using reduced input data sets in some of the applications to keep simulation time reasonable. Table 1 shows the characteristics of each particular benchmark.

Table 2 shows the configuration of the simulated architecture. A 22-stage pipeline (from fetch to commit) similar to that of the Intel Pentium 4 processor [8] has been considered as an example of a current microprocessor with a deep pipeline. These extra stages have been implemented in both Wattch's power model and sim-outorder's timing model. All results presented in this work use Wattch's clock-gating style "`cc3`," which scales power linearly with port/unit utilization, whereas inactive units still dissipate 10 percent of its maximum power.

5.2 Energy-Efficiency of *Selective Throttling*

In order to measure the energy-efficiency of *Selective Throttling*, we carried out a comprehensive set of experiments. These experiments are classified into three groups in order to study the effect of each power-aware knob. The first group of experiments exercises only the *fetch throttling* knob independently of the others. The second group of experiments evaluates the *decode throttling* knob independently and in combination with *fetch throttling*. Finally, the third group of experiments analyzes the *selection throttling* knob in conjunction with both *fetch throttling* and *decode throttling*. In all experiments, we compare configurations with equal total size in the required tables. The baseline configuration uses a 16 KB *gshare* branch predictor [18]. For *Selective Throttling*, the branch predictor is an 8 KB *gshare* and the confidence estimator is an 8 KB *BPRU* [1]. For comparison purposes, we have simulated *Pipeline Gating* with an 8 KB *gshare* and an 8 KB *BPRU* confidence estimator. We also include results for *Pipeline Gating* using an 8 KB *JRS* confidence estimator with an MDC-threshold of 12 and a gating threshold of 2, as proposed in [17].

Fig. 5 presents the results for the *fetch throttling* knob using different throttling policies, from less to more aggressive. The figure shows the speedup, energy savings, and energy-delay product improvement over the baseline.

Experiments A1, A2, and A3 reduce the fetch bandwidth to a half after an LC branch. Besides, after a VLC branch, the fetch bandwidth is reduced to a half, a fourth, or stalled, respectively. Such throttling policies have a negligible impact on performance, with an average slowdown lower than 1 percent. Reducing the fetch activity decreases power dissipation and results in average energy savings of 6 percent, 8 percent, and 10 percent, respectively. The E-D improvement is similar because of negligible performance loss. Experiments A4 and A5 correspond to more aggressive policies and, therefore, they further degrade performance (3 percent), but obtain greater energy savings (13 percent for A5), due to a drastic reduction in wrong-path instruction activity.

Finally, *Pipeline Gating* (experiments A6 and A7) also has a significant impact on performance, especially when using *BPRU*, with an average slowdown of 13 percent (up to 22 percent for *go*). This negative impact was also reported in [4], [15], [20]. *Pipeline Gating* obtains average energy savings of 11 percent, which is close to experiment A5. However, the average E-D improvement for *Pipeline Gating+BPRU* is negative (-1 percent) and it is just 2 percent for *Pipeline Gating+JRS*, whereas experiment A5 obtains an average E-D improvement of 10 percent. These results point out that completely stalling the fetch unit, as *Pipeline Gating*, is too aggressive in terms of power-performance, whereas limiting

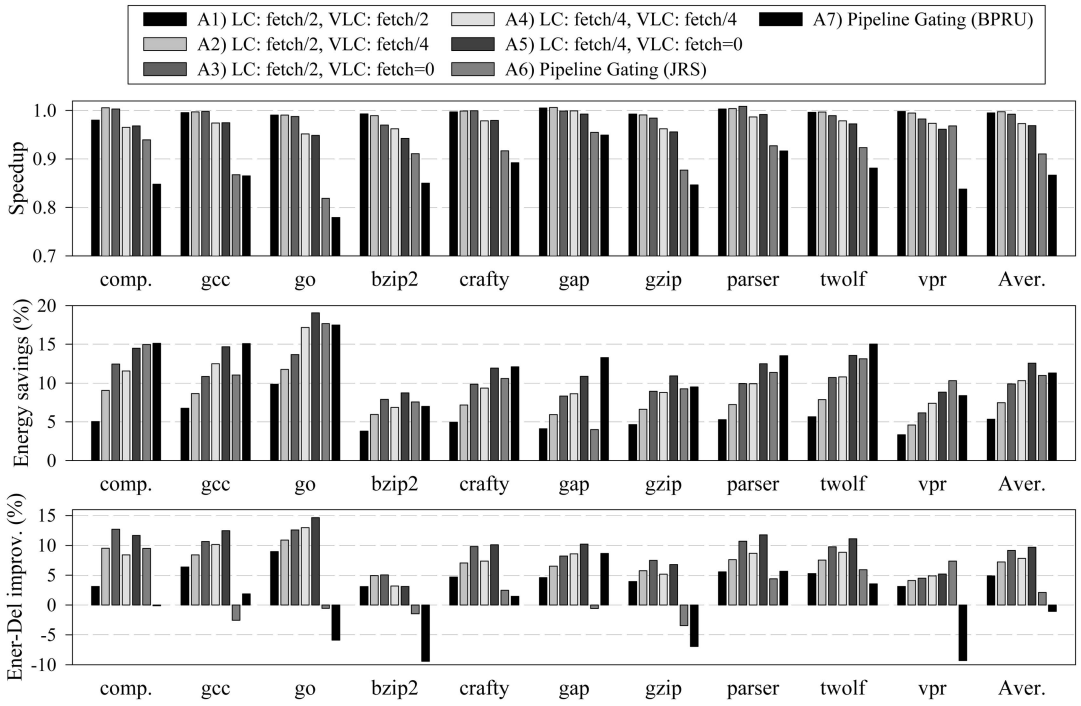

Fig. 5. Evaluation of the *fetch throttling* knob.

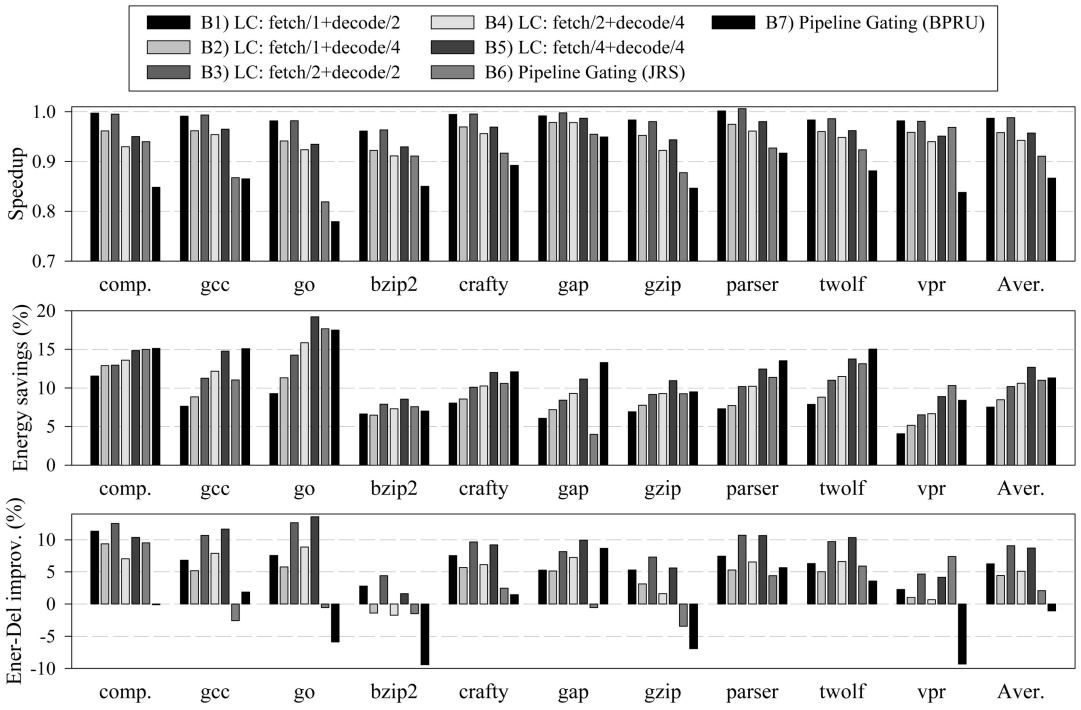

Fig. 6. Evaluation of the *decode throttling* knob independently and in combination with the *fetch throttling* knob.

the number of instructions entering the pipeline based on the confidence level assigned to the predicted branch represents a better trade-off between performance and power.

The next set of experiments evaluates the effect of *decode throttling* independently and in combination with *fetch throttling* knob. To limit the number of experiments, we have assumed that every VLC branch stalls the fetch unit since, in the previous analysis, the best trade-off was obtained by experiment A5. Fig. 6 shows the results using different throttling levels for the selected benchmarks. Again, results for *Pipeline Gating*, using both *BPRU* and *JRS* confidence estimators, are shown for comparison purposes.

Experiments B1 and B2 only change the decode bandwidth while keeping the fetch bandwidth at full speed when an LC branch is found. The speedup plot shows a noticeable impact on performance when the decode bandwidth is reduced to one fourth, obtaining an average slowdown of 4 percent in configuration B2. As expected, the

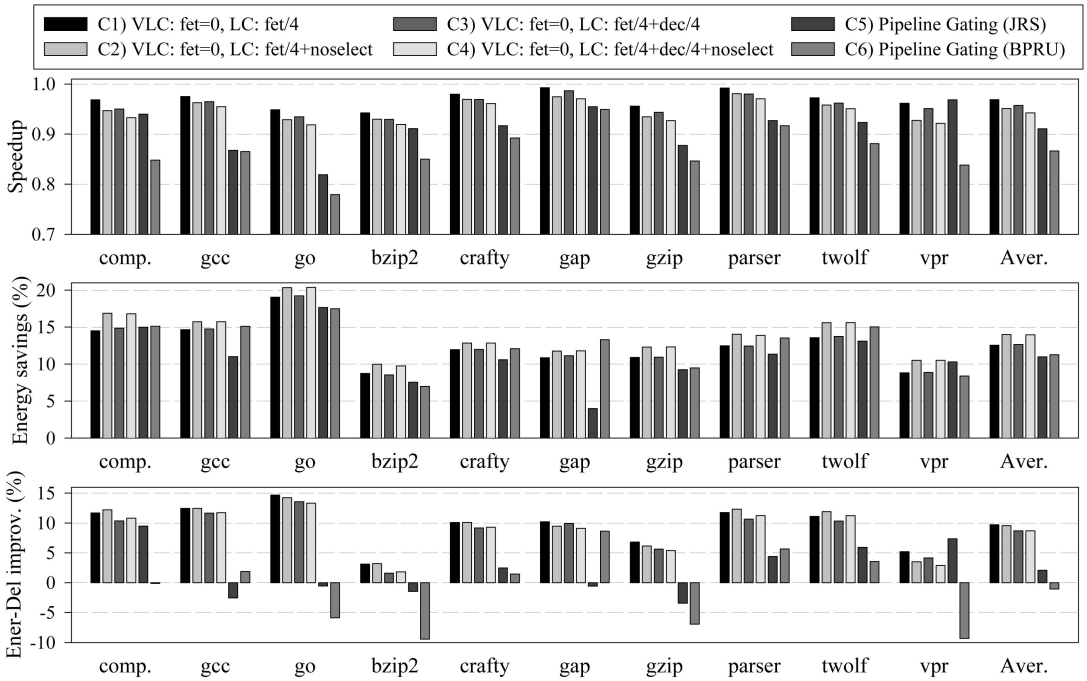

Fig. 7. Evaluation of the *selection throttling* knob.

reduction of the number of decoded instructions reduces power dissipation and results in average energy savings of 8 percent and 9 percent, respectively. Again, since experiment B2 limits the number of processed instructions more than B1, its average E-D improvement (4 percent) is lower than for configuration B1 (6 percent). Similar trends are experienced by experiments B3 and B4. This reveals that throttling the decode stage must be done carefully since aggressive policies may result in a significant impact on the E-D product.

Regarding the particular effect of *decode throttling* over the best configuration of *fetch throttling* (A5), configuration B5 represents an incremental change since an LC branch also reduces the decode bandwidth one fourth. The additional reduction in wrong-path instructions allows B5 to obtain similar energy savings and E-D improvement to A5.

The third set of experiments evaluates the effect of *selection throttling* knob combined with both the *fetch throttling* and *decode throttling* knobs. Fig. 7 plots the best configurations from the previous analysis (A5 and B5) along with the same configurations extended with the *selection throttling* knob. C1 and C3 are exactly the same experiments as A5 and B5, respectively, whereas C2 and C4 extend them with the *selection throttling* knob. Again, results for *Pipeline Gating* are shown for comparison purposes.

It can be seen that *selection throttling* scarcely degrades performance. Configuration C1 has an average slowdown of 3 percent, whereas C2 (with *selection throttling*) has a slowdown of 5 percent. C3 and C4 follow a similar trend: *Selection throttling* introduces an additional slowdown of about 2 percent. On the other hand, this knob reduces power dissipation due to lower issue and execution activities, resulting in higher average energy savings of 12 percent and 14 percent for experiments C1 and C2, respectively. *Selection throttling* provides 2 percent of additional energy savings while, at the same time, it does not harm the E-D product metric. This knob minimizes the impact of branches wrongly classified as low confidence since control-dependent instructions are still decoded and some of them are even awakened.

Summarizing, after evaluating the effect of the three power-aware knobs, the best results are obtained by configuration C2, which stalls the fetch unit after a VLC branch is encountered, reduces the fetch bandwidth to 1/4 after an LC branch, and avoids the selection of those instructions depending on an LC branch. This throttling policy obtains average energy savings of 14 percent (up to 20 percent for *go*) and an average E-D improvement of 10 percent (up to 14 percent for *go*), which is significantly better than that obtained by *Pipeline Gating*, with a negative E-D improvement of -1 percent, as seen before.

This analysis concludes that the classification of branches into multiple low confidence levels adds the flexibility to use different throttling policies that dynamically adjust the instruction traffic making an energy-efficient use of resources. Aggressive techniques are more effective for very low confidence predictions, whereas conservative, smart heuristics are more adequate for low confidence predictions. Completely gating the fetch unit is not always the best solution in terms of power-performance. A better trade-off is achieved by reducing the number of instructions at different stages in the pipeline based on the confidence level assigned to the predicted path.

It is also interesting to note that *Selective Throttling* reduces the energy consumption of many units of the processor. Fig. 8 shows the energy savings breakdown for configuration C2. As expected, the greatest energy savings are obtained in the I-cache (40 percent on average) because of an important reduction in the number of misspeculated instructions retrieved from the I-cache. Another important

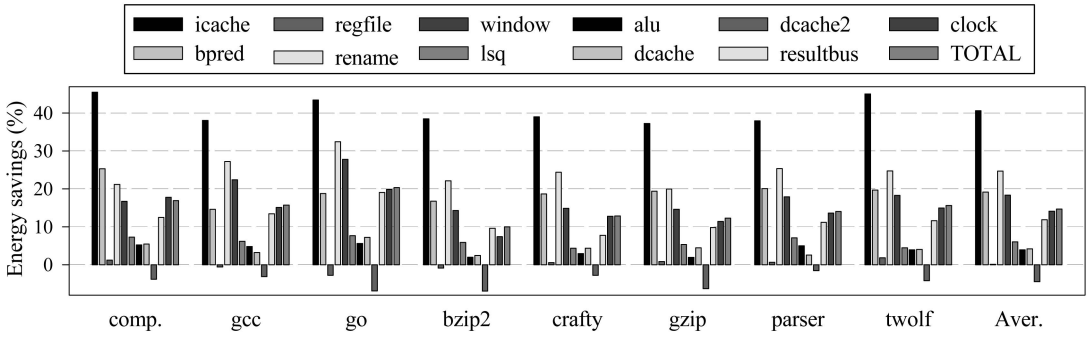

Fig. 8. Energy savings breakdown for several units of the processor.

structure is the issue window, which accounts for 18 percent of the total dissipated power of the simulated processor according to Fig. 1 (Section 3). For this particular structure, our proposal reduces energy consumption by 18 percent.

Finally, it is interesting to evaluate the energy-efficiency of a confidence-based scheme such as *Selective Throttling* for applications with highly predictable branches. In such applications, branch predictions are highly confident most of the time and, therefore, there are very few opportunities to initiate a throttling heuristic. Table 3 shows the relative performance and energy savings provided by the best configuration of *Selective Throttling* (C2) for six Spec2000 floating point benchmarks.[4] As expected, the average branch misprediction rate of those FP codes is just 1.5 percent. Looking at the worst case, *art*, the energy overhead of our proposal is just 1.3 percent and the E-D metric degrades by 5 percent. However, other predictable applications, such as *equake* (1.2 percent misprediction rate), obtain a positive E-D improvement of about 5 percent. Overall, *Selective Throttling* adds very little or no energy/performance overhead when speculation is highly confident most of the time.

5.3 Sensitivity Study of *Selective Throttling*

This section studies the energy-efficiency of *Selective Throttling* when some architectural parameters of the processor are varied. We report average speedup, power and energy savings, and E-D improvement for configuration C2 using the BPRU confidence estimator.

5.3.1 Pipeline Depth

The first group of experiments evaluates the effect of *superpipelining* [11], [23] on the energy-efficiency of *Selective Throttling*. In [12], [22], the authors proposed two analytical power-performance models that provide the variation of energy as a function of pipeline depth. The models were validated and refined using simulation data. As in those previous studies, we have extended the power-performance simulator to model different pipeline depths by adding extra stages "uniformly" across the pipeline. In particular, extra stages are added to the in-order front-end (fetch and decode stages), the cache access latency, and the latencies of the functional units, whereas processor frequency is not varied. We also modified the Wattch power model to account for the particular number of pipeline latches depending on the number of stages.

Increasing the number of cycles it takes for branches to be resolved augments the number of useless instructions entered into the pipeline, wasting resources and dissipating power. The longer the pipeline, the higher the wasted energy. *Selective Throttling* reduces the number of wrong-path instructions and, therefore, it provides higher benefits with deeper pipelines. This effect is illustrated in Fig. 9, which shows the percentage of overall wasted power by wrong-path instructions for pipeline depths from 5 to 42 stages in an 8-wide issue processor. Wrong-path instructions waste about 25 percent of overall power for a 5-stage pipeline. This percentage steadily increases to 31 percent for a 42-stage pipeline. When *Selective Throttling* is used, the percentage of wasted power drops to 10 percent and 14 percent, respectively.

In order to evaluate the energy-efficiency of *Selective Throttling*, Fig. 10 shows the relative performance, power, and energy savings as a function of pipeline depth. First, we can see that *Selective Throttling* is robust against pipeline length variations, with a performance degradation between 3 and 6 percent in all cases. However, power savings, energy savings, and, in particular, E-D improvement grow with pipeline depth, as expected. The average energy savings and E-D improvement go up to 16 percent and 13 percent, respectively, for 42 stages.

5.3.2 Reducing the Issue Width

The previous sensitivity analysis evaluated the effect of pipeline depth in an 8-wide issue processor. However, although the paper is focused on next-generation superscalar processors, it is also interesting to evaluate the

TABLE 3
Energy-Efficiency for Some SpecFP2000 Benchmarks

FP benchmarks	gshare 16 KB miss-rate	Relative Performance	Energy savings (%)	E-D improv. (%)
ammp	1.0%	1.00	-0.3	-0.7
art	0.5%	0.96	-1.3	-5.2
equake	1.2%	1.02	3.0	4.7
galgel	2.7%	0.99	2.5	1.4
lucas	0.8%	1.00	-0.4	-0.6
mesa	2.6%	1.01	5.6	6.3
Average	1.5%	1.00	1.5	1.0

4. We skip 100 million instructions and then simulate 500 million instructions for each benchmark.

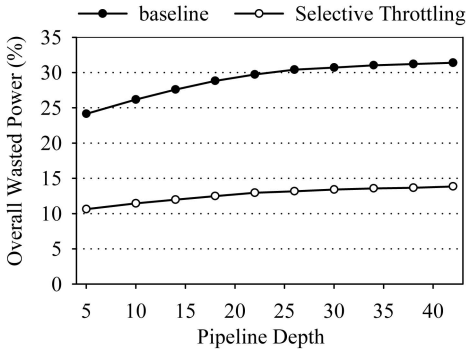

Fig. 9. Wasted power by wrong-path instructions (8-wide issue).

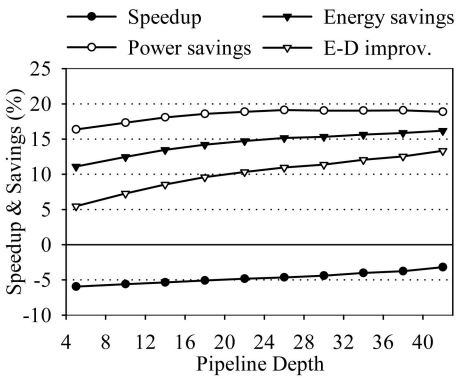

Fig. 10. Pipeline depth sensitivity of *Selective Throttling* (8-wide issue).

energy-efficiency of *Selective Throttling* in a current 4-wide issue processor design with a less power hungry front-end. The 4-wide issue configuration also allows a better comparison with the original *Pipeline Gating* scheme, which was tuned to a 4-wide issue, 7-stage processor [17]. Fig. 11 shows the energy savings and E-D improvement for a 4-wide issue processor as a function of pipeline depth. Results are shown for *Selective Throttling* (using *BPRU* confidence estimator) and for *Pipeline Gating* using both *BPRU* and *JRS* confidence estimators.

Despite the fact that the energy savings and E-D improvement are somewhat lower, *Selective Throttling* clearly outperforms previous gating techniques for almost all configurations (except for pipelines from five to seven stages). It is interesting to see that the energy savings provided by our proposal (which directly translates into battery savings and temperature reduction) for an eventual next generation mobile processor (4-wide issue, 20 stages) are close to 10 percent.

When compared with the original *Pipeline Gating* scheme in a 7-stage configuration, Fig. 11b shows that both *Selective Throttling* (using *BPRU*) and *Pipeline Gating* (using *JRS*[5]) obtain the same E-D improvement (1 percent) and about the same energy savings (5 percent). This shows that *Pipeline Gating* is a very energy-effective approach for short pipelines, but, when the number of stages is increased, *Pipeline Gating* is not able to reduce the E-D product even when using the same confidence estimator (*BPRU*), mainly due to the high impact on performance of stalling the fetch unit. As stated before, completely gating the fetch unit is not always the best solution in terms of power-performance, especially for deep pipelines. Reducing the bandwidth of the instructions that flow through the different stages of the pipeline enables a better compromise between performance loss and energy savings.

5.3.3 Branch Predictor and Confidence Estimator Sizes

The next group of experiments concerns the size of both the *gshare* branch predictor and the *BPRU* confidence estimator. In all cases, we always compare equal total sizes from 8 KB to 64 KB, as shown in Fig. 12. *Selective Throttling* devotes half of the total size to the branch predictor and the other half to the confidence estimator.[6]

5. The *JRS* confidence estimator has been tuned for every pipeline depth in order to increase its PNV.

6. Preliminary experiments with a different distribution of sizes showed that this is the best distribution.

As expected, branch prediction accuracy increases with predictor size for both the baseline and the *Selective Throttling*, leading to higher IPCs in both cases. However, the performance degradation provided by *Selective Throttling* is reduced as size grows because the confidence estimator becomes more accurate. On the other hand, power savings are reduced as size grows because there are fewer opportunities for improvement due to the higher prediction accuracy. These opposite trends of performance and power dissipation result in almost constant energy savings and E-D improvement of *Selective Throttling* with respect to the baseline as *gshare* size changes: between 14 percent and 13 percent energy savings and between 9 percent and 10 percent E-D improvement.

5.3.4 Using the Alpha 21264 Branch Predictor

The next group of experiments analyzes the benefits of our proposal when a more sophisticated branch predictor is used: the branch predictor of the Alpha 21264 processor. Again, we always compare equal total sizes from 8 KB to 64 KB, as shown in Fig. 13. As expected, when the size of the 21264 branch predictor is increased, power savings are reduced because there are fewer opportunities to reduce power dissipation due to the higher branch prediction accuracy. On the other hand, performance is also less degraded (around 2 percent slowdown for all evaluated sizes). The net effect is an average energy saving between 10 and 12 percent and average E-D improvement between 8 and 9 percent for *Selective Throttling* over the baseline configuration. The main conclusion of this analysis is that *Selective Throttling* provides significant energy savings even in the presence of highly accurate branch predictors.

5.3.5 Issue Window and Reorder Buffer Sizes

The last group of experiments evaluates the effect of the issue window and reorder buffer size. The studied sizes range from 64 to 512 entries, as shown in Fig. 14. The main conclusion of this analysis is that energy savings and E-D improvement provided by *Selective Throttling* increase as ROB and issue window size increases (21 percent and 16 percent, respectively, for 512 entries). The reason is that the reorder buffer and the issue window are responsible for an important fraction of the overall dissipated power. Large ROBs provide more opportunities to speculate, but they also incur more useless activity for each misprediction and, thus, the absolute energy consumption is increased. In such

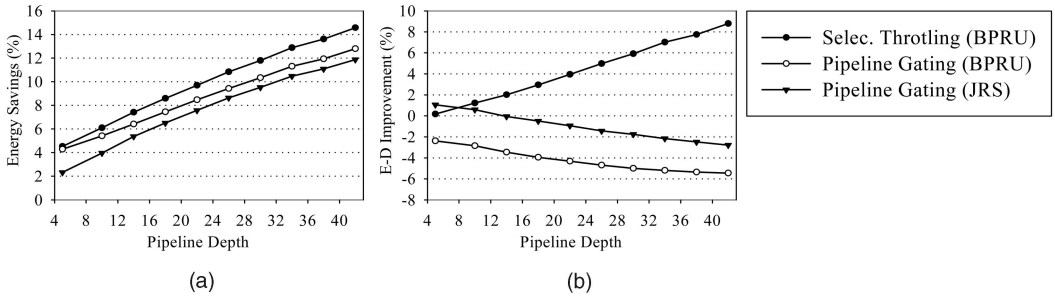

Fig. 11. Energy-efficiency for a 4-wide issue processor as a function of pipeline depth.

processors, a mechanism such as *Selective Throttling* can significantly reduce energy consumption and improve the E-D product.

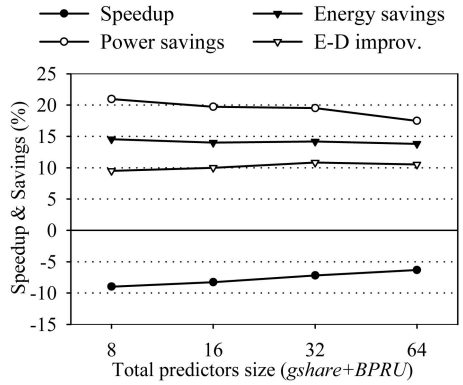

Fig. 12. Branch predictor size sensitivity.

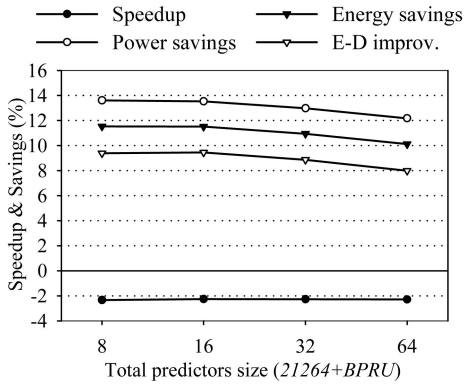

Fig. 13. *Selective Throttling* benefits for the *21264* branch predictor.

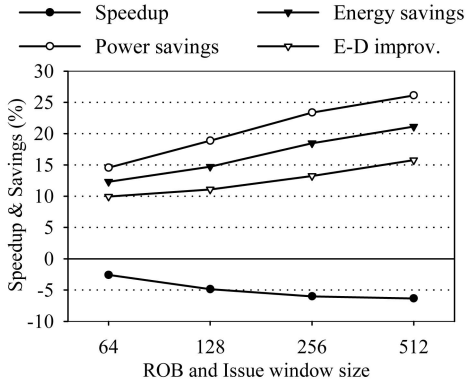

Fig. 14. Issue window and ROB size evaluation.

6 Conclusions

Modern superscalar processors waste a significant amount of energy in misspeculated instructions. In this work, we propose a mechanism, *Selective Throttling*, with the aim of reducing useless activities without compromising performance. *Selective Throttling* dynamically applies a different power-reduction scheme depending on the confidence estimation assigned to each particular branch prediction. We propose throttling at three different levels: fetch, decode, and instruction selection. Confidence estimation is used to select the appropriate level of throttling. For those branches that are likely to be mispredicted, aggressive throttling policies are applied. On the other hand, when confidence is not so low, less aggressive policies, both in terms of power reduction and performance degradation, are used. Results for a pipeline of 22 stages show that *Selective Throttling* achieves average energy savings of 14 percent and an average Energy-Delay improvement of 10 percent.

Finally, we have also shown that the power-performance efficiency of the *Selective Throttling* mechanism is robust against some architectural parameters. In particular, both energy savings and E-D improvement increase as the pipeline becomes deeper, obtaining 16 percent energy savings for 42 stages. Furthermore, benefits also increase as issue window and reorder buffer size augments. Benefits are also very important for a variety of branch predictor configurations.

Acknowledgments

This work has been partially supported by the Spanish Ministry of Education and Science under grants TIC2003-08154-C06-03 and TIN2004-03072 and EU Feder Funds.

References

[1] J.L. Aragón, J. González, J.M. García, and A. González, "Confidence Estimation for Branch Prediction Reversal," *Proc. Int'l Conf. High Performance Computing*, pp. 214-223, 2001.

[2] J.L. Aragón, J. González, A. González, and J.E. Smith, "Dual Path Instruction Processing," *Proc. Int'l Conf. Supercomputing*, 2002.

[3] J.L. Aragón, J. González, and A. González, "Power-Aware Control Speculation through Selective Throttling," *Proc. Int'l Symp. High Performance Computer Architecture*, 2003.

[4] A. Baniasadi and A. Moshovos, "Instruction Flow-Based Front-End Throttling for Power-Aware High-Performance Processors," *Proc. Int'l Symp. Low Power Electronics and Design*, 2001.

[5] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Frame-Work for Architectural-Level Power Analysis and Optimizations," *Proc. Int'l Symp. Computer Architecture*, 2000.

[6] A. Buyuktosunoglu, T. Karkhanis, D.H. Albonesi, and P. Bose, "Energy Efficient Co-Adaptive Instruction Fetch and Issue," *Proc. Int'l Symp. Computer Architecture,* 2003.
[7] P.Y. Chang, M. Evers, and Y.N. Patt, "Improving Branch Prediction Accuracy by Reducing Pattern History Table Interference," *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques,* 1996.
[8] P.N. Glaskowsky, "Pentium 4 (Partially) Previewed," *Microprocessor Report,* Aug. 2000.
[9] M.K. Gowan, L.L. Biro, and D.B. Jackson, "Power Considerations in the Design of the Alpha 21264 Microprocessor," *Proc. Design Automation Conf.,* June 1998.
[10] D. Grunwald, A. Klauser, S. Manne, and A. Pleszkun, "Confidence Estimation for Speculation Control," *Proc. Int'l Symp. Computer Architecture,* 1998.
[11] A. Hartstein and T.R. Puzak, "The Optimum Pipeline Depth for a Microprocessor," *Proc. Int'l Symp. Computer Architecture,* pp. 7-13, May 2002.
[12] A. Hartstein and T.R. Puzak, "Optimum Power/Performance Pipeline Depth," *Proc. Int'l Symp. Microarchitecture,* Dec. 2003.
[13] T.H. Heil and J.E. Smith, "Selective Dual Path Execution," technical report, Electrical and Computer Eng. Dept., Univ. of Wisconsin-Madison, 1997.
[14] E. Jacobsen, E. Rotenberg, and J.E. Smith, "Assigning Confidence to Conditional Branch Predictions," *Proc. Int'l Symp. Microarchitecture,* 1996.
[15] T. Karkhanis, J.E. Smith, and P. Bose, "Saving Energy with Just in Time Instruction Delivery," *Proc. Int'l Symp. Low Power Electronics and Design,* Aug. 2002.
[16] A. Klauser, A. Paithankar, and D. Grunwald, "Selective Eager Execution on the PolyPath Architecture," *Proc. Int'l Symp. Computer Architecture,* 1998.
[17] S. Manne, A. Klauser, and D. Grunwald, "Pipeline Gating: Speculation Control For Energy Reduction," *Proc. Int'l Symp. Computer Architecture,* 1998.
[18] S. McFarling, "Combining Branch Predictors," Technical Report #TN-36, Digital Western Research Lab., 1993.
[19] S. Palacharla, N.P. Jouppi, and J.E. Smith, "Complexity-Effective Superscalar Processors," *Proc. Int'l Symp. Computer Architecture,* 1997.
[20] D. Parikh, K. Skadron, Y. Zhang, M. Barcella, and M. Stan, "Power Issues Related to Branch Prediction," *Proc. High Performance Computer Architecture,* 2002.
[21] C. Small, "Shrinking Devices Put the Squeeze on System Packaging," *EDN,* pp. 41-46, Feb. 1994.
[22] V. Srinivasan, D. Brooks, M. Gschwind, P. Bose, V. Zyuban, P.N. Strenski, and P.G. Emma, "Optimizing Pipelines for Power and Performance," *Proc. Int'l Symp. Microarchitecture,* pp. 333-344, Dec. 2002.
[23] E. Sprangle and D. Carmean, "Increasing Processor Performance by Implementing Deeper Pipelines," *Proc. Int'l Symp. Computer Architecture,* pp. 25-36, 2002.
[24] S. Wallace, B. Calder, and D.M. Tullsen, "Threaded Multiple Path Execution," *Proc. Int'l Symp. Computer Architecture,* 1998.
[25] T.Y. Yeh and Y.N. Patt, "Two-Level Adaptive Branch Prediction," *Proc. Intl Symp. Microarchitecture,* 1991.

Juan L. Aragón received the MS and PhD degrees in computer engineering from the Universidad de Murcia, Spain, in 1996 and 2003, respectively. During 2003 and 2004, he did a one-year postdoctoral stay as a visiting assistant professor and researcher in the Computer Science Department at the University of California, Irvine. In 1999, he joined the Computer Engineering Department at the Universidad de Murcia, Spain, where he currently holds an assistant professor position. His research interests are focused on processor microarchitecture, energy-efficient architectures, embedded processors, branch prediction, and value prediction. He is a member of the IEEE Computer Society.

José González received the MS and PhD degrees from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain. In January 2000, he joined the Computer Engineering Department of the University of Murcia, Spain, and became an associate professor in June 2001. In March 2002, he joined the Intel Barcelona Research Center, where he is a senior researcher. Currently, he is working on new paradigms for the IA-32 family, in particular, thermal and power-aware clustered microarchitectures. He is a member of the IEEE Computer Society.

Antonio González received the MS and PhD degrees from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain. He is the founding director of the Intel-UPC Barcelona Research Center, whose research focuses on new microarchitecture paradigms and code generation techniques for future microprocessors. He joined the faculty of the Computer Architecture Department of UPC in 1986 and became a full professor in 2002. He currently holds a part-time professor position in this department. His research has focused on computer architecture, compilers, and parallel processing, with a special emphasis on processor microarchitecture and code generation. He has published more than 200 papers, has given more than 60 invited talks, and has filed for 13 patents in the areas of power-aware microarchitectures, clustered microarchitectures, speculative multithreaded processors, data value and data dependence speculation and reuse, cache architectures, register file architecture, modulo scheduling, code analysis and optimization, parallel algorithms, prolog-oriented architectures, instruction fetching mechanisms, and digital image processing. He is an associate editor of the *IEEE Transactions on Computers*, *IEEE Transactions on Parallel and Distributed Systems*, *ACM Transactions on Architecture and Code Optimization*, and *Journal of Embedded Computing*. He has served on more than 50 program committees for international symposia in the field of computer architecture, including ISCA, MICRO, HPCA, PACT, ICS, ICCD, ISPASS, CASES, and IPDPS. He was program (co)chair for ICS 2003, ISPASS 2003, and MICRO 2004, among other symposia. He is a member of the IEEE Computer Society.