Power-Aware Control Speculation through Selective Throttling

Juan L. Aragón[1], José González[2] and Antonio González[2,3]

[1]Dept. Ing. y Tec. de Computadores
Universidad de Murcia
Murcia, Spain
jlaragon@ditec.um.es

[2]Intel Barcelona Research Center
Intel Labs, UPC
Barcelona, Spain
josex.gonzalez.gonzalez@intel.com

[3]Dept. d'Arquitec. de Computadors
Universitat Politècnica de Catalunya
Barcelona, Spain
antonio@ac.upc.es

Abstract

With the constant advances in technology that lead to the increasing of the transistor count and processor frequency, power dissipation is becoming one of the major issues in high-performance processors. These processors increase their clock frequency by lengthening the pipeline, which puts more pressure on the branch prediction engine since branches take longer to be resolved. Branch mispredictions are responsible for around 28% of the power dissipated by a typical processor due to the useless activities performed by instructions that are squashed.

This work focuses on reducing the power dissipated by mis-speculated instructions. We propose Selective Throttling *as an effective way of triggering different power-aware techniques (fetch throttling, decode throttling or disabling the selection logic). The particular set of techniques applied to each branch is dynamically chosen depending on the branch prediction confidence level. For branches with a low confidence on the prediction, the most aggressive throttling mechanism is used whereas high confidence branch predictions trigger the least aggressive techniques. Results show that combining fetch bandwidth reduction along with select logic disabling provides the best performance both in terms of energy reduction and energy-delay improvement (14% and 9% respectively for 14 stages, and 17% and 12% respectively for 28 stages).*

1. Introduction

Power dissipation and energy consumption have become an important concern in the design of high performance microprocessors. In such systems it may be necessary the use of very expensive cooling schemes, which may have a significant impact on the final cost. For mobile systems, battery life is a key design concern. Furthermore, since power translates directly into heat, an increase in power dissipation may cause chip malfunction due to some failures such as thermal runaway, junction fatigue and electro-migration diffusion [26].

Current processor design trends lead to large pipelines in order to meet the cycle time requirements (e.g. 20 stages in the Pentium 4 [12]). In these architectures, a branch takes longer to be resolved and the processor is filled with many speculative instructions. Due to mispredicted branches, part of the power dissipated by a typical processor (around 28% on average) is due to mis-speculated instructions that waste energy performing useless activities.

In this work, we focus on reducing the energy wasted by mis-speculated instructions by means of *Selective Throttling*. According to the confidence level assigned to each branch prediction, different processor blocks are dynamically throttled: fetch unit, decode unit or selection logic (from more to less aggressive). Aggressive throttling will be applied for those branches with high probability of being mispredicted (at the expense of reducing performance if the branch hits). On the other hand, when the estimator is not sure about the correctness of the prediction, less aggressive techniques, both in terms of power reduction and performance degradation, are applied.

Among prior related work we can point out *Pipeline Gating* originally proposed by Manne *et al.* [21]. Since this scheme is an all-or-nothing mechanism, it is very sensitive to the goodness of the underlying confidence estimator in the sense that performance is highly penalized if a confidence estimation turns out to be wrong, and the fetch or decode stages had been completely stalled (see Section 5.2).

In this paper we make the following contributions:

- Throttling policies are selectively applied according to the branch confidence estimation. In addition, these policies have a certain degree of variation (i.e. complete fetch stall *vs*. stalling fetch every 4 cycles).
- A new throttling technique is proposed which avoids the selection of instructions that are control dependent on a low confident branch.
- The evaluation of the proposed scheme in terms of power and energy consumption, instead of using indirect and approximate metrics such as *Extra Work* [21] or *Instruction Traffic* [6].

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 analyzes the power and energy consumption due to mis-speculated

Proceedings of the The Ninth International Symposium on High-Performance Computer Architecture (HPCA-9'03)
1530-0897/02 $17.00 © 2002 IEEE

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. Background and Related Work

A very large body of research has been targeted at reducing the performance degradation caused by branch mispredictions. Many proposals try to improve branch prediction accuracy [29][22][10][2]. Others try to minimize performance degradation by fetching and/or executing multiple paths [15][19][28][3]. However, analyzing how mis-speculated instructions influence energy consumption has not received much attention.

As mentioned above, *Pipeline Gating* [21] 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 [14][16]. These confidence estimations are used to decide if the processor is likely to fetch and execute instructions that will not commit. The number *M* of unresolved low confidence branches is used to determine when and how long to gate. Thus, if *M* exceeds a threshold, the fetch or decode stage is stalled although previously fetched or decoded instructions continue traversing the pipeline. The authors evaluated their proposal for several confidence estimators with different hardware complexities. The best results, reported for an underlying *gshare* branch predictor [22], use the *JRS* confidence estimator [16] with an MDC-threshold of 12 and a *gating threshold* of 2.

In [6], Baniasadi and Moshovos propose a mechanism in order to reduce power dissipation, by enabling or disabling the fetch or decode stages according to certain heuristics. They introduce two control-flow heuristics that are orthogonal to confidence-based approaches: instead of fetching and decoding as many instructions as possible, they analyze instruction traffic identifying situations in which the additional parallelism that may be exposed does not improve performance. In such situations they propose to turn the fetch stage off during 3 cycles.

In [13] an extensive evaluation of the tradeoffs between power and performance for different architectural paradigms can be found. Many works have focused on power consumption in cache memories [18][4][17] since it is a critical component that is devoted a large portion of the chip. Several architecture-level power models have been developed for use in architecture power-performance research such as *Wattch* [8] and *SimplePower* [27].

In [1], it is proposed to balance the clock rate dynamically to match the requirements of the instruction stream. In [7], it is minimized power consumption of functional units, exploiting the fact that the sizes of operands are often less than the size of the available functional units. In [5], *Pipeline Balancing* dynamically tunes the resources of a general purpose processor to the needs of the application by monitoring performance. In [11], energy consumption of the issue logic is reduced by dynamically re-sizing the instruction queue and disabling the wake-up of ready operands. In [25], critical path prediction is used to separate high-speed functional units dedicated to critical instructions from low-power functional units dedicated to non-critical ones. Recently, several branch predictor schemes have been evaluated considering power-performance tradeoff [24]. Note that these proposals do not tackle the problem of consumption due to mis-speculated instructions.

3. Power and Energy Consumption of Mis-speculated Instructions

Conventional front-end designs rely on control flow speculation, which allows a processor to guess the target of a conditional branch without waiting for it to execute. While speculation greatly improves performance, it also increases power dissipation and energy consumption in case of a misprediction. As showed in previous works, the number of incorrectly fetched instructions can account for up to 80% of all instructions. Obviously, this extra traffic is greater in the front-end stages (fetch/decode), since fewer mis-speculated instructions reach the issue or execution stages.

Table 1. Overall power breakdown and the fraction wasted by mis-speculated instructions.

	Overall Power Breakdown	% of overall power wasted by mis-speculated instr.
Overall Power	56.4 Watts	27.9%
icache	10.0%	6.4%
bpred	3.8%	1.4%
regfile	1.6%	0.2%
rename	1.1%	0.5%
window	18.2%	5.6%
lsq	1.9%	0.2%
alu	8.7%	1.0%
dcache	10.6%	1.1%
dcache2	0.7%	0.0%
resultbus	9.5%	1.9%
clock	33.8%	9.5%

To understand how these extra instructions affect power dissipation, we ran the eight benchmarks from the SPECint95 and SPECint2000 that exhibit the highest misprediction rate using the *Wattch v1.02* power-performance simulator [8] (see Section 5.1 for details about the simulation methodology). The branch predictor is an 8 KB *gshare* [22] whose history register is speculatively updated.

Table 1 shows the overall power breakdown[1] for each block of the baseline microprocessor as well as the percentage of overall power wasted by all mis-speculated instructions for both conditional and unconditional branches. It can be seen that about 28% of the overall power is dissipated by incorrect instructions. This represents an upper bound of the power reduction we can achieve with the techniques proposed in this paper. As expected, the fetch stage ("`icache`"+"`bpred`"), which is responsible for 13.8% of the overall power, wastes 7.8% of the overall power processing incorrect instructions. Similarly, the decode stage ("`rename`" + a fraction of "`regfile`"[2] + a fraction of "`window`"[3]) wastes some power processing incorrect instructions. Finally, it is interesting to note that other portions of the processor such as the "`window`" –the fraction corresponding to the issue logic, wake-up and selection logic–, LSQ, functional units, data caches and the result bus still waste up to 9.8% of the overall power processing mis-speculated instructions, which is a considerable amount of power.

In order to precisely determine the potential of the proposed techniques, we ran the following experiments:

- *Oracle fetch*: only fetches correct-path instructions. In case of misprediction the processor does not fetch the mis-speculated path.
- *Oracle decode*: uses realistic fetch but only decodes correct-path instructions.
- *Oracle select*: uses realistic fetch and decode but only selects for issuing correct-path instructions.

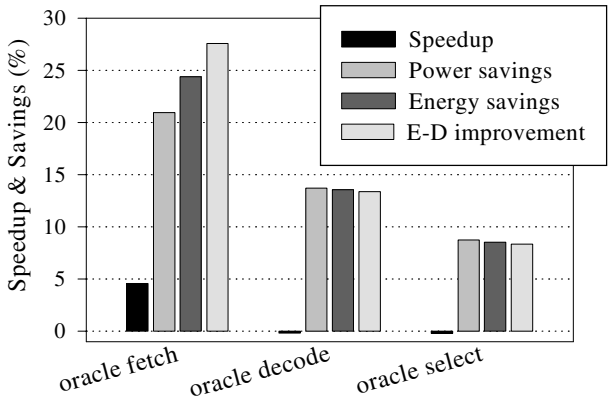

Figure 1. Oracle fetch, decode and select savings.

[1] Using the Wattch's clock-gating style `cc3`, which scales power linearly with unit usage. Inactive units still dissipate 10% of its maximum power.

[2] According to Wattch power model, the "`regfile`" activity counter is updated at decode stage to read a ready operand, and also at *commit* to write the result.

[3] The "`window`" activity counter is updated at decode stage to write ready operands into RUU (physical registers), at *issue* to read operands from physical registers and at *writeback* to write the result into the corresponding physical register.

Figure 1 shows the average speedup as well as power and energy savings and energy-delay improvement for the eight selected benchmarks. The *oracle fetch* experiment provides similar results to those shown in Table 1 except that now only conditional branches are considered. However, since confidence estimation is assigned to conditional branches, the *oracle fetch* experiment provides a precise upper bound about how much power is dissipated by mis-speculated instructions. Overall savings for power, energy and energy-delay are 21%, 24% and 28% respectively. Note also that the *oracle fetch* experiment obtains a speedup of 5%. This is mainly due to the I-cache pollution and to the fact that wrong-path instructions waste resources and may delay the execution of correct ones.

Because of how Wattch provides the dissipated power, it is not easy to determine the overall power fraction corresponding to mis-speculated instructions for each pipeline stage, since some Wattch's blocks belong to several stages. Experiments shown in Figure 1 allow obtaining the power wasted by wrong-path instructions on a per-stage basis. In particular, the difference between the power savings of the *oracle fetch* and *oracle decode* experiments represents an upper bound of the power wasted in the fetch stage (7.3%). Analogously, the difference between the power savings of the *oracle decode* and *oracle select* experiments constitutes an upper bound of the power wasted in the decode stage (5.0%). Energy consumption can be similarly calculated for the front-end stages (10.8% and 5.0% respectively).

Finally, this analysis demonstrates that, contrary to what it was suggested in previous work [21], a considerable amount of power and energy can be saved if we were able to gate at decode or issue stages whenever a wrong-path is being processed: up to 13.7% (power savings with *oracle decode*) and 8.7% (power savings with *oracle select*) respectively.

4. Selective Throttling

Selective Throttling is a mechanism that reduces dynamic power dissipation and energy consumption while attempting to minimize performance degradation. This is accomplished by limiting the number of mis-speculated instructions fetched, decoded and issued, and therefore decreasing the useless activity of the processor.

As previous proposals, *Selective Throttling* relies on branch confidence estimation to initiate a particular heuristic. We propose to use different policies depending on the confidence estimation, with the goal of obtaining an optimal tradeoff between power and performance.

4.1. Power-Aware Heuristics

The aim of the power-aware heuristics is to provide different throttling levels with different impact on performance whenever a heuristic is erroneously applied. We have evaluated the following ones:

1530-0897/02 $17.00 © 2002 **IEEE**

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

Limiting the fetch and decode bandwidth is achieved by alternating full activity cycles with stalled cycles. For instance, in an 8-way 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 aim of the novel *selection throttling* heuristic is to reduce the power dissipated when mis-speculated instructions are executed. In order to do that, the selection of instructions control-dependent of a low confidence branch is disabled. An incorrect instruction will not use buses to send its operands to the functional units, which, in turn, will not be unnecessarily wasted. The corresponding results will not be forwarded to incorrect dependent instructions (avoiding useless activity of the issue logic, especially waking-up mis-speculated instructions) that would store them at the reservation station entries (recall that, as shown in the *oracle select* experiment of Section 3, this useless work represents 8.7% of the total power dissipation).

On the other hand, this novel heuristic has a minor impact on performance when it is incorrectly applied (i.e. activated for correctly predicted branches), mainly when compared with more aggressive techniques such as *fetch throttling* or *decode throttling*.

Note also that there are no pipeline deadlocks when this heuristic is incorrectly applied: although control-dependent instructions cannot be selected, those data dependent on instructions prior to the branch are indeed awakened, and after the branch resolution, ready instructions can be quickly issued. Finally, the *selection throttling* heuristic is fairly straightforward to implement, requiring a bit in each instruction window entry to disable selection. Figure 2 shows how the *no-select* bit is used to avoid raising the request signal used by the selection logic.

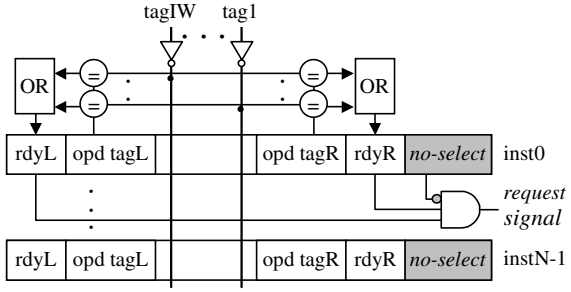

Figure 2. Wake-up logic [23] and the generation of the request signal used by the selection logic.

4.2. Confidence-Based Categorization of Branches

The power-performance efficiency of *Selective Throttling* strongly depends on the confidence estimator's accuracy. If the confidence estimator labels a prediction as low confidence and the prediction turns out to be correct, the heuristic triggered by *Selective Throttling* incurs in a serious penalization: power dissipation is not reduced whereas performance is degraded, which leads to a higher energy consumption. On the other hand, if a prediction is labeled as high confidence and the branch turns out to be mispredicted, performance is not additionally degraded but energy is again wasted.

Thus, in order to obtain an optimal power-performance tradeoff, instead of using the conventional two states (high/low) provided by the confidence estimator, we propose to categorize each branch prediction into the following four states: a) very-high confidence branches (VHC); b) high confidence branches (HC); c) low confidence branches (LC); and d) very-low confidence branches (VLC).

This categorization is carried out by using the value of the confidence counter stored in each entry of the confidence estimator, although each confidence estimator may require a particular implementation. Therefore, this categorization allows a fine grain decision concerning the heuristic applied depending on the likelihood of a prediction to be incorrect. Finally, to improve the *Selective Throttling* mechanism, after initiating a power-aware heuristic, if a later branch is labeled as VLC or LC before the first branch is resolved, a more restrictive heuristic can be initiated but not a less restrictive one.

4.3. Evaluated Confidence Estimators

According to the metrics introduced by Grunwald *et al* [14], a good confidence estimator should have high SPEC and PVN[4] metrics. This led us to use the confidence estimator proposed for the *Branch Prediction Reversal Unit* (*BPRU*) scheme [2], that makes use of predicted data values to assess the confidence of branch predictions.

In order to obtain a better power-performance tradeoff, it is necessary to label more branches as VLC or LC, and thus, initiating more power-aware heuristics. For this reason, the original *BPRU* behavior has been modified. Since *BPRU* uses a tagged table, whenever a branch misses in that table, the saturating counter of the underlying branch predictor is used to provide the estimation. If a branch is predicted as either *weakly* taken or *weakly* not-taken, the branch is considered as LC. As expected, this change increments the SPEC metric at the expense of reducing the PVN metric. Simulations for an 8

[4] 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.

KB *gshare* predictor along with an 8 KB *BPRU* obtain an average SPEC = 60% and a PVN = 45% for the eight selected benchmarks. Finally, whenever a branch hits in the *BPRU* table it is labeled as follows: since each entry has a 3-bit up/down saturating counter, values 0-1 of the counter assign VHC to the branch, values 2-3 assign HC, values 4-5 assign LC and finally, values 6-7 assign VLC to the branch.

We have also evaluated *Pipeline Gating* [21] using an 8 KB *JRS* confidence estimator with an MDC-threshold of 12. It obtains an average SPEC = 90% and a PVN = 24%, which is consistent with results reported in [14].

5. Experimental Results

5.1. Simulation Methodology

To evaluate the power-performance efficiency of *Selective Throttling*, we used the eight 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 [8]. Due to the large number of dynamic instructions in some benchmarks, we reduced the input data set while keeping a complete execution. Table 2 shows the characteristics for each particular benchmark.

Table 2. 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%
	gzip	input.source 1	500	52	8.8%
	parser	test (modified)	500	64	6.8%
	twolf	test	258	21	11.2%

Table 3 shows the configuration of the simulated architecture. The pipeline has been lengthened to 14 stages (from fetch to commit), following the pipeline scheme of the IBM Power 4 processor [20], 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 or unit usage, whereas inactive units still dissipate 10% of its maximum power.

The following metrics are used to evaluate the results:

- *Performance*: in terms of instructions committed per cycle (IPC).

Table 3. 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 reorder buffer, 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	0.18µm, Vdd = 2.0 V, 1200 Mhz.

- *Average* Instantaneous *Power (Watts)*: the total power dissipated in a per-cycle basis.
- *Energy (Joules)*: is equal to the product of the power dissipated and total execution time. It is more appropriate in low-end embedded and portable systems in which battery life is the primary index [9].
- *Energy-Delay* product *(Joules*sec)*: is equal to the product of energy and total execution time. It is more appropriate in high performance systems since the extra delay factor ensures a greater emphasis on performance [9].

5.2. Power-Performance Efficiency of *Selective Throttling*

In order to measure the power-performance efficiency of *Selective Throttling*, we carried out three set of experiments evaluating the effect of each power-aware heuristic. The first set of experiments exercises the *fetch throttling* heuristic independently of the other heuristics.

Figure 3 shows the speedup, power and energy savings as well as energy-delay (E-D) improvement using different throttling levels, from less to more aggressive, for the selected benchmarks. The underlying branch predictor is an 8 KB *gshare* [22] and the confidence estimator is an 8 KB *BPRU*. For comparison purposes, we have also evaluated *Pipeline Gating* using an 8 KB *JRS* confidence estimator with an MDC-threshold of 12 and a gating threshold of 2.

Experiments A1, A2 and A3 reduce the fetch bandwidth to a half after a LC branch, and 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 less than 1%. However, these limitations in the bandwidth of the fetch stage reduce the power dissipation resulting in average energy savings of 5.2%, 6.6% and 9.2% respectively. E-D improvements are consistent with

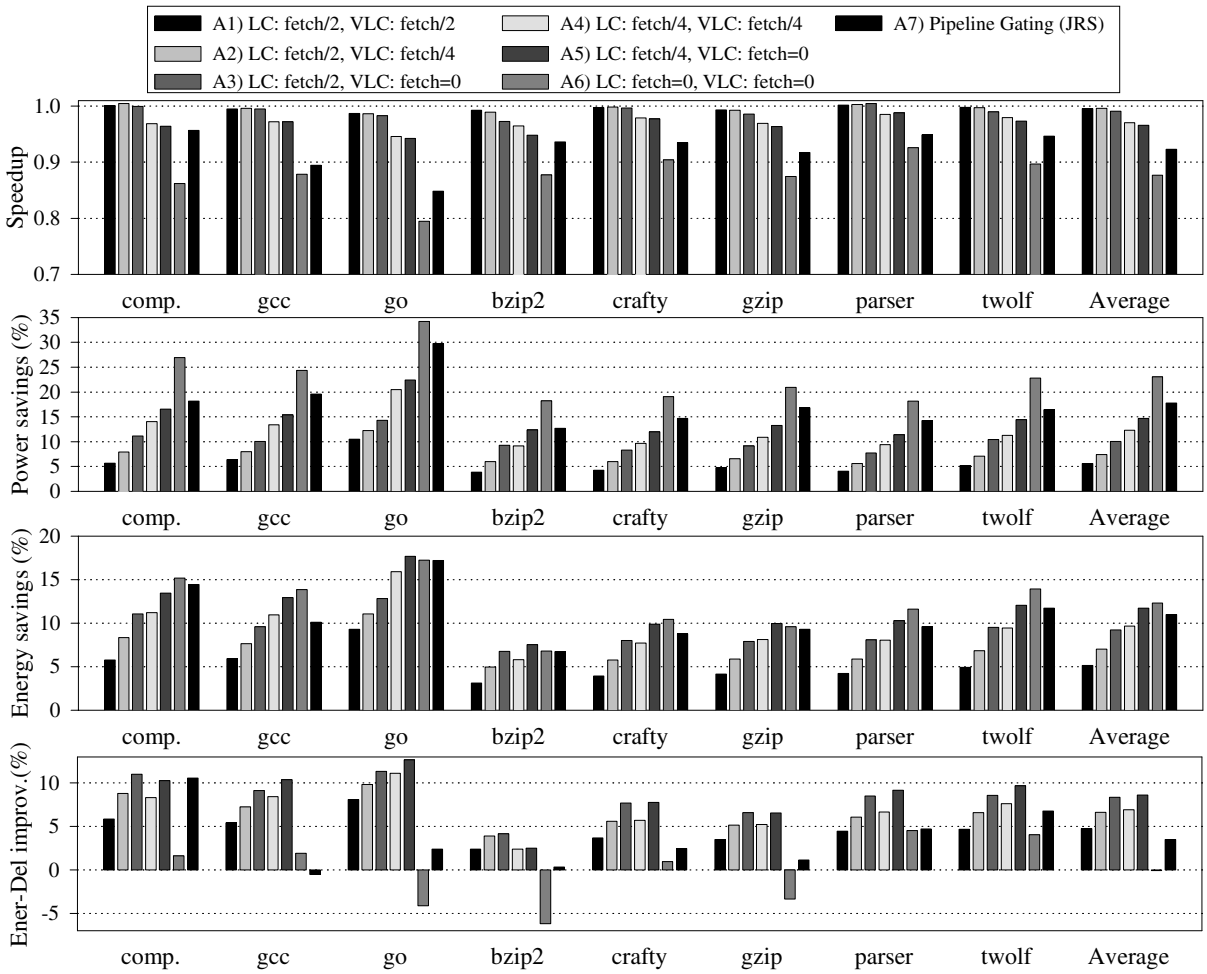

Figure 3. Evaluation of the *fetch throttling* heuristic.

energy savings. Experiments A4 and A5 are more aggressive, and therefore degrade more performance (3%). But they obtain greater energy savings (11.2%) than previous experiments because the flow of incorrect instructions is drastically reduced, pointing out the benefits of more aggressive throttling policies. However, E-D product metric is penalized in experiment A4, and especially in experiment A6, which stops the fetch unit completely for both LC and VLC branches (as *Pipeline Gating* without using the gating threshold). In this case, the average slowdown is 12%, resulting in a null E-D improvement. Therefore, those reductions of the fetch bandwidth are not appropriate for high performance systems, where the E-D product is the more interesting metric.

Finally, *Pipeline Gating* (experiment A7) also has a significant impact on performance, with an average slowdown of 8% (up to 15% for *go*). This negative impact is also reported in [6] and more recently in [24]. In addition, it obtains a higher average E-D improvement (3.5%) than experiment A6, showing how the use of the gating threshold may palliate the effect of the aggressive gating policy of stalling the fetch unit. Nevertheless, if we only consider energy savings, both experiments A6 and A7 obtain 12.3% and 11.0% respectively, showing such gating policies as appropriate for low-end systems in which battery life is the primary index [9].

Summarizing, the best tradeoff between power and performance is obtained by stopping the fetch unit when a VLC branch is encountered, and reducing four times the fetch bandwidth when a branch is labeled as LC (11.7% of energy savings and 8.6% of E-D improvement).

The second set of experiments evaluates the effect of the *decode throttling* heuristic independently and in combination with the *fetch throttling* heuristic. In order to limit the number of experiments, and since in the previous analysis the best tradeoff is obtained by experiment A5, we have assumed that every VLC branch stops the fetch unit. Therefore, this analysis exercises the *decode throttling* heuristic only when a LC branch is encountered. Figure 4 shows the results using different throttling levels for the selected benchmarks using an 8 KB *gshare* and an 8 KB *BPRU*. Again, results of *Pipeline Gating* are presented for comparison purposes.

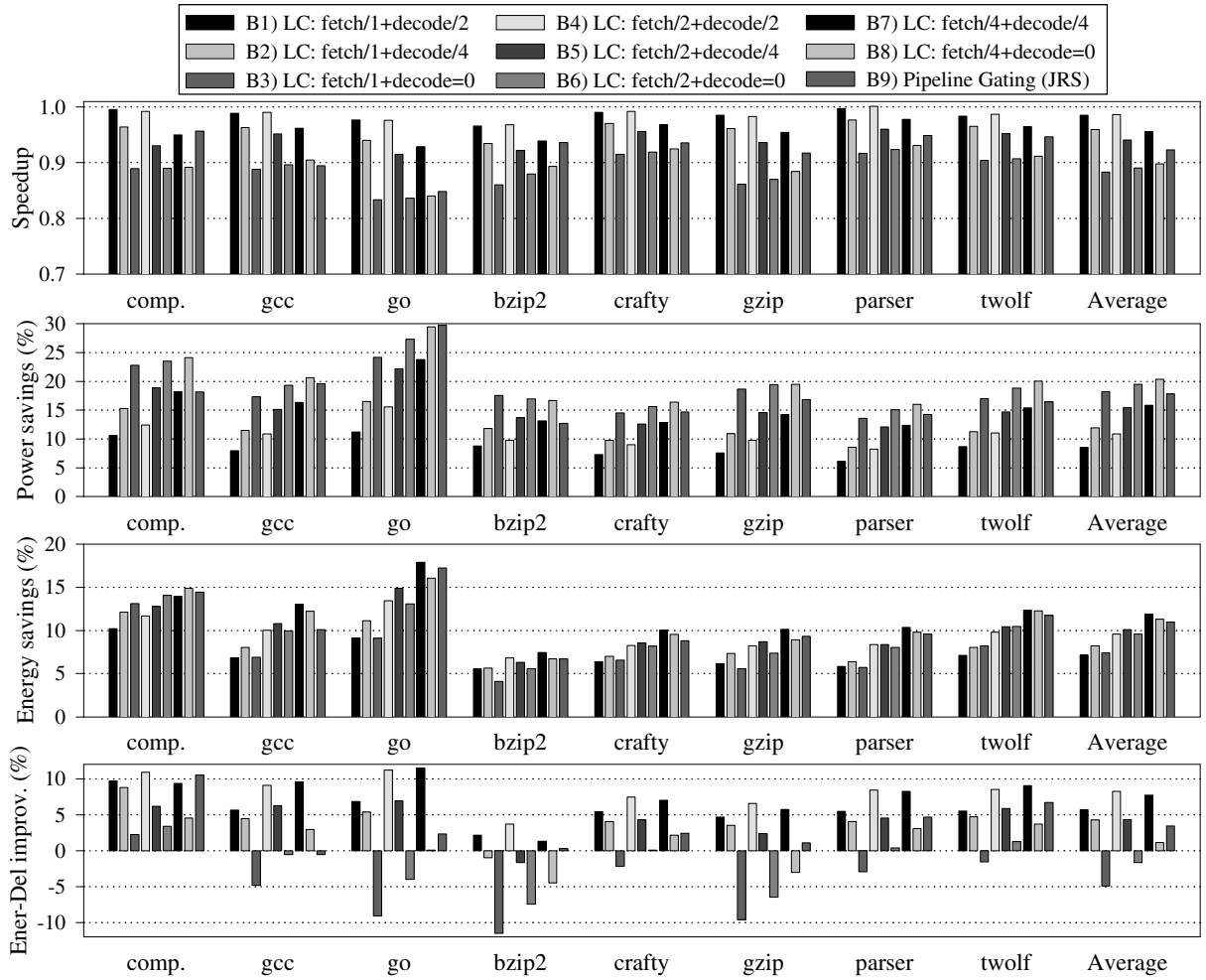

Figure 4. Evaluation of the *decode throttling* heuristic independently and in combination with the *fetch throttling* heuristic. In all experiments, the fetch unit is stalled when a VLC branch is found.

Experiments B1, B2 and B3 only change the decode bandwidth while leaving the fetch bandwidth unaltered at full speed when a LC branch is found. The speedup plot shows a significant impact on performance while the decode bandwidth is reduced, obtaining an average slowdown of 12% in experiment B3.

As expected, the reduction of the number of instructions traversing the decode and next stages reduces power dissipation resulting in experiment B2 having a greater average energy saving (8.2%) than B1 (7.1%). But this trend is not followed by experiment B3, which consumes more energy than B2. This negative behavior is more evident looking at the E-D improvement, being experiment B1 much better than B3 (−5.0%). This reveals that throttling the decode stage must be done carefully, since aggressive policies result in significant impact on the E-D product metric.

Regarding the particular effect of the *decode throttling* heuristic over the best experiment of the previous analysis (A5), experiments B7 and B8 represent an incremental change since a LC branch also reduces the decode bandwidth four times or stops it, respectively. The additional reduction of the traffic of incorrect instructions allows experiment B7 obtaining a slightly higher average energy savings (11.9%) than A5 (11.7%), but lower average E-D improvements (7.8%) than A5 (8.6%). Therefore, for low-end systems *decode throttling* provides additional benefits, but not for high performance systems.

The third set of experiments evaluates the effect of the *selection throttling* heuristic in combination with both *fetch throttling* and *decode throttling* heuristics. In order to properly determine its effect, Figure 5 plots again, the best experiments from the previous analysis (without the *selection throttling* heuristic) along with the same experiment using the *selection throttling* heuristic. Therefore, experiment C1 is the same as experiment A5, experiment C3 is the same as B5 and experiment C5 is the same as B7. Finally, experiments C2, C4 and C6 include the use of the *selection throttling* heuristic. Again, we plot the results of *Pipeline Gating* for comparison purposes.

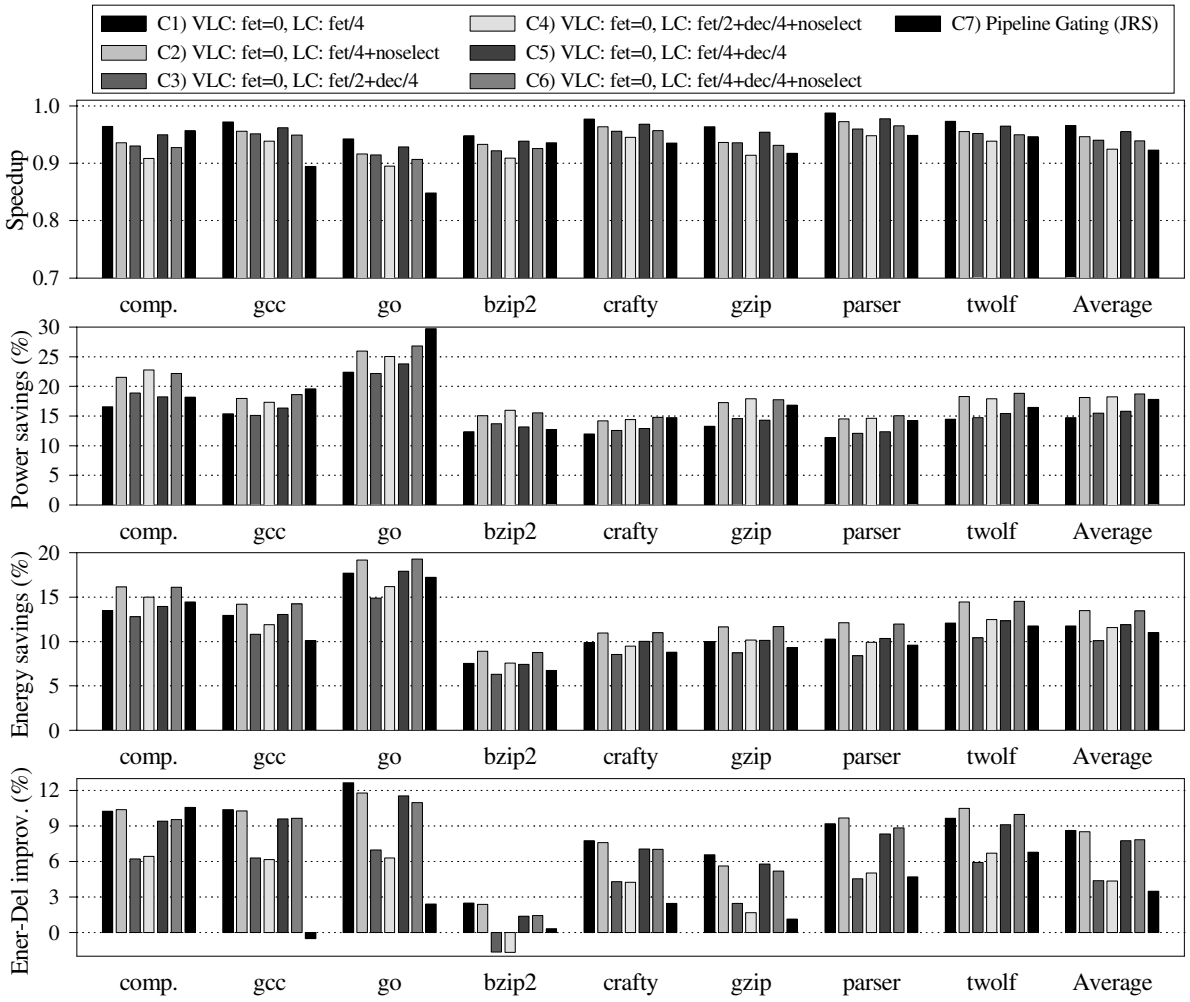

Figure 5. Evaluation of the *selection throttling* heuristic.

It can be seen that the inclusion of the *selection throttling* heuristic scarcely degrades performance. For instance, experiment C1 has an average slowdown of 3% whereas C2 (with *selection throttling*) drops it to 5%. The rest of experiments follows this trend. Thus, this heuristic introduces an additional slowdown of about 2%. On the other hand, power dissipation is reduced due to the lower issue and execution activities, resulting in higher energy savings due to the *selection throttling* heuristic. Experiment C2 increments the average energy savings of C1 from 11.7% to 13.5%. Similar results are obtained when comparing C3 with C4, and C5 with C6. The trend is about 2% of additional average energy savings provided by the use of *selection throttling*, which reveals that this heuristic obtains an additional balance in power-performance efficiency for low-end systems. Regarding E-D improvement, it can be seen that *selection throttling* does not provide additional benefits, although using it does not degrade the power-performance balance for high performance systems, as it is the case of *Pipeline Gating*.

Summarizing, after evaluating the effect of the three power-aware heuristics, the best approach is obtained by experiment C2, which stops the fetch unit after a VLC branch is encountered, reduces the fetch bandwidth four times for a LC branch as well as avoids the selection of those instructions depending on the LC branch. This experiment obtains an average energy saving of 13.5% (up to 19.2% for *go*) whereas *Pipeline Gating* obtains 11.0%. Furthermore, the average E-D improvement is 8.5% (up to 12% for go), which is significantly better than that obtained by *Pipeline Gating* (just 3.5%).

Therefore, results show that for the power of mispredicted branches to be reduced, aggressive techniques must be applied for non-confident predictions and conservative but smart heuristics must be applied to weak confident predictions.

The *selection throttling* heuristic is important when considering energy savings while at the same time it does not harm the E-D metric. This reveals the fine grain balance in power-performance efficiency provided by this

heuristic: in case of erroneously labeling a branch as LC, control-dependent instructions can be decoded and some of them even awakened, allowing a fast recovery.

5.3. Sensitivity Study

This section studies the power-performance efficiency of *Selective Throttling* when some architectural parameters of the processor are varied. We present the average speedup, power and energy savings as well as E-D improvement obtained by the best experiment C2, which stops the fetch unit after a VLC branch, reduces the fetch bandwidth four times for a LC branch and avoids the selection of instructions depending on the LC branch.

5.3.1. Pipeline Depth

As stated previously, current processor design trends lead to longer pipelines in order to meet the cycle time requirements. The first group of experiments evaluate the effect of pipeline depth on the power-performance efficiency of the *Selective Throttling* mechanism. We varied the pipeline depth by changing the number of stages of the in-order front-end (fetch/decode) and also incrementing the execution and L1 D-cache latencies[5]. Figure 6 shows the results for pipelines from 6 to 28 stages. First, we can see that *Selective Throttling* is robust against pipeline length variations, with a performance degradation between 5% and 6% in all cases. However, power savings, energy savings and E-D improvements grow with pipeline depth due to the fact that the energy wasted by useless instructions increases since they spend more cycles in the pipeline, and the *Selective Throttling* mechanism limits the number of wrong-path instructions traversing the pipeline. The average energy savings for 6 stages are 11%, going up to 17.2% for 28 stages. Finally, the E-D improvements are 5.4%, 8.5% and 12% for 6, 14 and 28 stages respectively. This shows the benefits of *Selective Throttling* as pipelines become longer.

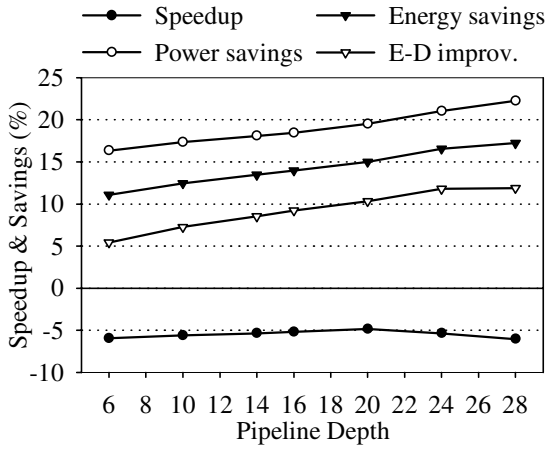

Figure 6. Pipeline depth evaluation.

[5] We have not varied the processor frequency.

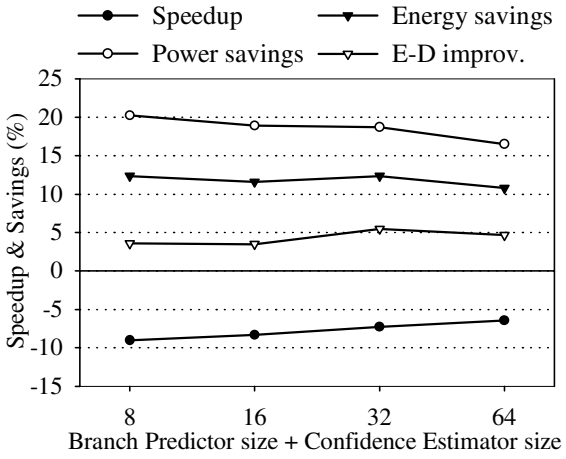

Figure 7. Table size evaluation.

5.3.2. Branch Predictor and Confidence Estimator Size

The second group of experiments concerns the size of the *gshare* branch predictor and the size of the confidence estimator. The studied total size ranges from 8 KB to 64 KB. In all cases, we always compare equal total sizes. Therefore, *Selective Throttling* devotes half of the total size to the branch predictor and the other half to the confidence estimator.

As expected, for both the baseline and the *Selective Throttling* experiments, the branch prediction accuracy is incremented as the branch predictor size becomes larger. This situation leads to higher IPCs in both cases. However, in Figure 7 we can see that the performance degradation derived from the *Selective Throttling* mechanism is reduced as size grows because the confidence estimator becomes more accurate determining wrong paths. On the other hand, the power savings derived from the *Selective Throttling* mechanism are reduced (20.3% for 8 KB and 16.5% for 64 KB) because there are less opportunities for improvement due to the higher prediction accuracy. These opposite trends in both performance and power dissipation lead to energy savings and E-D improvements almost constant respect to size changes: between 11% and 12% energy savings, and between 4% and 5% E-D improvements.

6. Conclusions

In modern superscalar processors around 28% of the dissipated power comes from mis-speculated instructions that waste energy performing useless activities. In this work we propose a mechanism, *Selective Throttling*, that depending on the confidence degree assigned to branch predictions, dynamically apply a different power-aware technique. We propose throttling at three different levels: fetch, decode and selection. Confidence estimation will be used to assign the appropriate level of throttling. The goal of our proposal is to obtain an optimal tradeoff between power and performance. For those branches likely to be

mispredicted, aggressive throttling will be applied. On the other hand, when the estimator is not quite sure about the confidence level of the prediction, less aggressive techniques, both in terms of power reduction and performance degradation, are used.

We have evaluated the proposed scheme in terms of power and energy consumption, instead of using indirect metrics. Results for *Selective Throttling* using an 8 KB *BPRU* confidence estimator obtain average energy savings of 13.5% (up to 19.2% for go). Furthermore, the average energy-delay improvement (which is an appropriate metric for high performance systems) is 8.5%, significantly higher than that obtained by *Pipeline Gating* (3.5%).

Finally, we have also shown that the power-performance efficiency of the *Selective Throttling* mechanism is robust against modifications of some architectural features. Both energy savings and E-D improvements are incremented as pipelines become deeper, as it is the current trend, obtaining 17% and 12% respectively for 28 stages.

Acknowledgements

We thank the anonymous referees for their valuable comments. This work has been supported by the Spanish Ministry of Science and Technology under grants TIC2000-1151-C07-03 and TIC2001-0995-C02-01.

References

[1] D. Albonesi. "Dynamic IPC/Clock Rate Optimization". *Proc. of the Int. Symp. on Computer Architecture*, 1998.

[2] J.L. Aragon, J. Gonzalez, J.M. Garcia and A. Gonzalez. "Confidence Estimation for Branch Prediction Reversal". *Proc. of the Int. Conference on High Performance Computing*, pp. 214-223, 2001.

[3] J.L. Aragon, J. Gonzalez, A. Gonzalez and J.E. Smith. "Dual Path Instruction Processing". *Proc. of the Int. Conference on Supercomputing*, 2002.

[4] I. Bahar, G. Albera and S. Manne. "Power and Performance Trade-Offs Using Various Caching Strategies". *Proc. of the Int. Symp. on Low Power Electronics and Design*, 1998.

[5] R.I. Bahar and S. Manne. "Power and Energy Reduction Via Pipeline Balancing". *Proc. of the Int. Symp. on Computer Architecture*, 2001.

[6] A. Baniasadi and A. Moshovos. "Instruction Flow-Based Front-end Throttling for Power-Aware High-Performance Processors". *Proc. of the Int Symp. on Low Power Electronics and Design*, 2001.

[7] D. Brooks and M. Martonosi. "Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance". *Proc. of the High-Perf. Comp. Arch.*, 1999.

[8] D. Brooks, V. Tiwari, and M. Martonosi. "Wattch: A Frame-Work for Architectural-Level Power Analysis and Optimizations". *Proc. of the Int. Symp. Comp. Arch.*, 2000.

[9] D. Brooks, P. Bose, S.E. Schuster, H. Jacobson, P.N. Kudva, A. Buyuktosunoglu, J.D. Wellman, V. Zyuban, M. Gupta and P.W. Cook. "Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors". *IEEE Micro*, Nov/Dec 2000.

[10] P.Y. Chang, M. Evers and Y.N. Patt. "Improving Branch Prediction Accuracy by Reducing Pattern History Table Interference". *Proc. of the Int. Conf. on Parallel Architectures and Compilation Techniques*, 1996.

[11] D. Folegnani and A. González. "Energy-Effective Issue Logic". *Proc. of the Int. Symp. on Comp. Architec.*, 2001.

[12] P.N. Glaskowsky. "Pentium 4 (Partially) Previewed". *Microprocessor Report*, August 2000.

[13] R. González and M. Horowitz. "Energy Dissipation in General Purpose Microprocessors". *IEEE Journal of Solid State Circuits*, 31(9), pp 1277-1284, 1996.

[14] D. Grunwald, A. Klauser, S. Manne and A. Pleszkun. "Confidence Estimation for Speculation Control". *Proc. of the Int. Symp. on Computer Architecture*, 1998.

[15] T.H. Heil and J.E. Smith. "Selective Dual Path Execution". Technical Report, Univ. of Wisconsin-Madison, ECE, 1997.

[16] E. Jacobsen, E. Rotenberg and J.E. Smith. "Assigning Confidence to Conditional Branch Predictions". *Proc. of the Int. Symp. on Microarchitecture*, 1996.

[17] M. Johnson and W. Mangione-Smith. "The Filter Cache: An Energy Efficient Memory Structure". *Proc. of the Int. Symp. on Microarchitecture*, 2001.

[18] M.B. Kamble and K. Ghose. "Analytical Energy Dissipation Models for Low Power Caches". *Proc. of the Int. Symp. on Low Power Electronics and Design*, 1997.

[19] A. Klauser, A. Paithankar and D. Grunwald. "Selective Eager Execution on the PolyPath Architecture". *Proc. of the Int. Symp. on Computer Architecture*, 1998.

[20] K. Krewell. "IBM's Power4 Unveiling Continues". *Microprocessor Report*, Nov. 2000.

[21] S. Manne, A. Klauser and D. Grunwald. "Pipeline Gating: Speculation Control For Energy Reduction". *Proc. of the Int. Symp. on Computer Architecture*, 1998.

[22] S. McFarling. "Combining Branch Predictors". Tech. Report #TN-36. Digital Western Research Lab., 1993.

[23] S. Palacharla, N.P. Jouppi and J.E. Smith. "Complexity-Effective Superscalar Processors". *Proc. of the Int. Symp. on Computer Architecture*, 1997.

[24] D. Parikh, K. Skadron, Y. Zhang, M. Barcella, and M. Stan. "Power Issues Related to Branch Prediction". *Proc. of the High Performance Computer Architecture*, 2002.

[25] J.S. Seng, E.S. Tune and D.M. Tullsen. "Reducing Power with Dynamic Critical Path Information". *Proc. of the Int. Symp. on Microarchitecture*, 2001.

[26] C. Small. "Shrinking Devices Put the Squeeze on System Packaging". EDN, pp. 41-46, February 1994.

[27] N. Vijaykrishnan, M. Kandemir, M. Irwin, H. Kim and W. Ye. "Energy-Driven Integrated Hardware-Software Optimizations Using SimplePower". *Proc. of the Int. Symp. on Computer Architecture*, 2000.

[28] S. Wallace, B. Calder and D.M. Tullsen. "Threaded Multiple Path Execution". *Proc. of the Int. Symp. on Computer Architecture*, 1998.

[29] T.Y. Yeh and Y.N. Patt. "Two-Level Adaptive Branch Prediction". *Proc. of the Int. Symp. on Microarchitec.*, 1991.