Efficient Register Renaming and Recovery for High-Performance Processors

Salvador Petit, *Member, IEEE,* Rafael Ubal, Julio Sahuquillo, *Member, IEEE,* and Pedro López, *Member, IEEE*

Abstract

Modern superscalar processors implement register renaming by using either RAM or CAM tables. The design of these structures should address both access time and misprediction recovery penalty. While direct-mapped RAMs provide faster access times, CAMs are more appropriate to avoid recovery penalties. However, the presence of associative ports in CAMs prevents them from scaling with the number of physical registers and pipeline width, negatively impacting performance, area, and energy consumption at the rename stage. In this paper, we present a new hybrid RAM-CAM register renaming scheme, which combines the best of both approaches. In a steady state, a RAM provides fast and energy-efficient access to register mappings. On misspeculation, a low complexity CAM enables immediate recovery. Experimental results show that in a 4-way state-of-the-art superscalar processor, the new approach provides almost the same performance as an ideal CAM-based renaming scheme, while dissipating only between 17% and 26% of the original energy and, in some cases, consuming less energy than purely RAM-based renaming schemes. Overall, the silicon area required to implement the hybrid RAM-CAM scheme does not exceed the area required by conventional renaming mechanisms.

Index Terms

Register renaming, misspeculation recovery, CAM complexity, energy consumption, energy efficiency.

I. Introduction

MODERN superscalar microprocessors implement out-of-order and speculative execution to increase performance. Many mechanisms have been devised aimed at enhancing the amount of instructions executing concurrently. These mechanisms require register renaming techniques in order to solve *write after read* (WaR) and *write after write* (WaW) data hazards.

Register renaming distinguishes two kinds of registers: *logical* and *physical* registers. Logical registers refer to those used by the compiler, while physical registers are those actually implemented in the machine. Typically, the number of physical registers is quite larger than the number of logical registers. When an instruction that produces a result is decoded, the renaming logic allocates a *free* physical register. The logical destination register is said to be mapped to that physical register. Subsequent data-dependent instructions rename their source registers to access this physical register. Renaming structures are accessed every cycle after instructions are decoded. The register renaming circuitry also deals with the register mapping table recovery on misspeculation. As these structures are highly accessed, renaming structures are critical due to their high power density [1], and new solutions must be devised to deal with this problem.

Random-access memories (RAMs) and content-addressable memories (CAMs) have been used for register renaming. Both of them present advantages and shortcomings, and the industry does not show a predominant trend. Some processors [2], [3] use the RAM approach, while others [4]–[6] include a CAM with a large number of checkpoints.

A logical source register is renamed by using its identifier to obtain the current mapping. This is performed faster and more efficiently in terms of energy with a RAM structure. The RAM is directly indexed by a source register, whereas this register is compared against all current mappings in the CAM. This associative search is a major concern not only because of its long access time, but also because it hinders scalability with the number of registers [7].

Regardless of the approach used, checkpoints allow for quick recovery of the correct mappings after misspeculation. in both RAM-based [2] and CAM-based [4]–[6] processors. However, when the number of checkpoints surpasses a certain limit, CAM checkpointing becomes faster and more energy-efficient than RAM checkpointing [7]. Table I shows the recovery penalty time (in processor cycles) of a RAM-based processor (with the baseline processor configuration described in Section IV). As observed, even when triggering the recovery at the *writeback* stage, the amount of cycles is not negligible.

In this paper, we propose a new scheme that tries to take the best of each implementation, that is, fast register renaming, fast register allocation, and fast recovery. We propose a hybrid approach that uses both a RAM and a CAM. During correct path execution, the RAM provides most of the mappings, acting as a CAM cache. The CAM is checkpointed whenever a branch is decoded, enabling quick misspeculation recovery, which is followed by an invalidation of the RAM contents. After recovery, the RAM is progressively refilled with correct mappings while new instructions enter the pipeline.

The advantages of the hybrid design stem from two main sources. On one hand, processors work in a non-speculative mode in the common case, so RAM invalidations are unusual. On the other hand, frequently executed instructions (e.g., loops) only use a small subset of the architected register file, so only few RAM updates suffice to recover the steady state. Thus, a reduction of the CAM complexity does not hurt performance, but lowers its power consumption, area, and access time.

The rest of this paper is organized as follows. Section II discusses typical renaming mechanisms. Section III describes the hybrid RAM-CAM approach. Section IV presents experimental results and Section V discusses some related work.

Table I
RECOVERY CYCLES IN A TYPICAL RAM-BASED APPROACH

Triggered at	SpecInt	SpecFP	Average
Commit stage	17.1	55.0	36.0
Writeback stage	4.7	25.7	15.2

```
A   beq r21,r0,destx
B   addi r5,r0,1        | r5->p11
C   addi r6,r0,2        | r6->p12
D   addi r7,r0,3        | r7->p13
E   addi r8,r0,4        | r8->p14
F   beq r22,r0,desty
G   addi r8,r0,5        | r8->p15
H   beq r23,r0,destz
I   add r7,r5,r6        | r7->p16
```

(a)

Logical Register	Physical Register
...	...
r5	p11
r6	p12
r7	p13→p16
r8	p15
...	...

(b)

Figure 1. RAM working example. (a) Code example. (b) RAM excerpt.

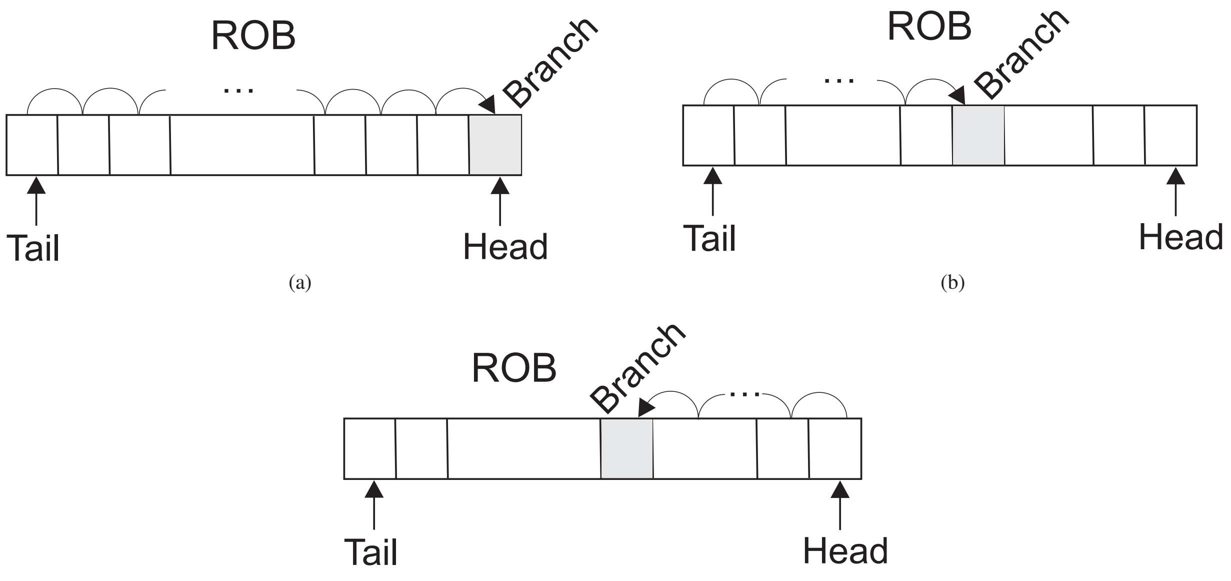


Figure 2. Recovery schemes. (a) Recover at commit. (b) Recover at writeback from the tail. (c) Recover at writeback from the head.

II. BACKGROUND

A. RAM Approach

The following example illustrates how a typical RAM-based approach works. Fig. 1a shows a code snippet consisting of nine instructions, whose four destination registers (*r5–r8*) are renamed into six physical registers (*p11–p16*). Fig. 1b shows the RAM contents at the time instruction *I* reaches the rename stage. At this point, its source registers are renamed to *p11* and *p12*. In addition, a free physical register (*p16*) is allocated to *r7*. To allocate a free physical register, RAM approaches use a *free register queue* (FRQ). After *I* is renamed, subsequent instructions having a data dependence on *r7* will be renamed to *p16*. The direct-mapped memory allows the mappings to be rapidly performed while taking up small area. Later, at the commit stage, physical registers are released by placing their identifiers back into the FRQ.

Since the RAM table is updated at the rename stage, it is modified by either non-speculative or speculative instructions. On misspeculation, the changes must be canceled, restoring the RAM to its previous state at the time the offending instruction entered the rename stage.

The simplest strategy for misspeculation recovery consists in waiting until the mispredicted branch reaches the ROB head (*recover at commit*, see Fig. 2a). As ROB entries contain the previous mapping for the destination register, the correct RAM state can be restored by scanning the ROB once the offending instruction reaches the ROB head.

Recover at commit incurs a penalty with two main components: i) the time elapsed since the misprediction is known until the mispredicted instruction reaches the commit stage, and ii) the time required to restore the correct mappings. The second component can be reduced by using two RAMs, a front-end RAM (FRAM) and a retirement RAM (RRAM) [3].

Physical	Logical	Current	Branch Checkpoints				
Register	Register	Mapping	cp_H	cp_F	cp_A	· · ·	Free
· · ·			· · ·				
P11	R5	1	1	1	0		0
P12	R6	1	1	1	0		0
P13	R7	1→0	1	1	0		0
P14	R8	0	0	1	0	· · ·	0
P15	R8	1	1	0	0		0
P16	R7	0→1	0	0	0		1→0
P17	–	0	0	0	0		1
· · ·			· · ·				

Figure 3. CAM table excerpt.

Physical	Logical	Current	Branch Checkpoints		
Register	Register	Mapping	cp_A	· · ·	Free
· · ·			· · ·		
P11	R5	1	0		0
P12	R6	1	0		0
P13	R7	1	0		0
P14	R8	1	0	· · ·	0
P15	R8	0	0		0→1
P16	R7	0	0		0→1
P17	–	0	0		1
· · ·			· · ·		

Figure 4. CAM misspeculation recovery from the branch checkpoint.

To reduce the first component of the penalty, recovery should be triggered as soon as the misprediction is known (*recover at writeback*, see Fig. 2c). This approach can rely on a single FRAM table, which is restored by walking the ROB from its tail towards the first misspeculated instruction. However, if there is a RRAM, the FRAM can also be restored by first copying the RRAM contents into the FRAM, and then walking the ROB from its head towards the first misspeculated instruction (see Fig. 2c).

B. CAM Approach

CAM structures have as many rows as the number of available physical registers. Each row maintains information for renaming, recovery, and register allocation as shown in Fig. 3. The first column indicates the mapped *logical register*, whereas the second column specifies whether this mapping is currently active or not.

Let us assume a simple design where register mappings are checkpointed each time a branch instruction is decoded. Fig. 3 shows an excerpt of a CAM table containing the *current mappings* and a set of *branch checkpoints*. The table contains the renaming information corresponding to the code shown in Fig. 1a at the time instruction *I* reaches the rename stage. Source registers *r5* and *r6* of instruction *I* are renamed to *p11* and *p12*, respectively. Destination register *r7*, previously mapped to *p13*, is remapped to *p16*, which is obtained by means of a priority encoder (PE) connected to the *free* column. Then, this mapping is updated in the corresponding entries (*logical register* and *current mapping*) of the CAM. At the same time, the *current mapping* entry of *p13* is reset. Finally, branch checkpoint columns cp_H, cp_F, and cp_A keep a copy of the *current mapping* column at the time the corresponding branch (i.e. *H*, *F* and *A*) was decoded. Depending on the implementation, checkpoints can be performed indiscriminately [6] or selectively [1], [8]. Creating a checkpoint merely involves copying the *current mapping* column.

Regarding register allocation, a physical register is assumed to be free when its *current mapping* bit is clear and it is not present in any checkpoint. In the example, *p11*, *p12*, *p15*, and *p16* are currently mapped, so they cannot be released. However, although *p13* is not currently mapped, it cannot be released until both branches *F* and *H* are resolved and known to be non-speculative, since *p13* appears in checkpoints cp_F and cp_H. This is also the case of *p14* in checkpoint cp_F. Finally, *p17* is free because it is neither currently mapped nor present in any checkpoint. Free registers can be obtained by simply *nor*-ing the *current mapping* and the *branch checkpoints* bits.

When dealing with multiple checkpoints, the *current mapping* and *branch checkpoints* columns can be organized as a circular queue following program order, where the current mapping is located at the tail, and the rest of the entries contain the branch checkpoints. This implementation allows for a reducing both the temporal penalty and power consumption of misprediction recovery.

Fig. 4 presents the CAM transition triggered when branch *F* is resolved as mispredicted. Simply by updating the tail pointer, the checkpoint cp_F column becomes the current mapping, and the mapping of *r7* from *p13* to *p16* is undone. This is also the case of *r8*, which is mapped to *p14* again. In addition, *p15* and *p16* are released, since it stops being allocated by the current mapping or any other checkpoint. Notice that youngest checkpoints (i.e. cp_F and cp_H) are discarded and only oldest checkpoints are kept (i.e. cp_A).

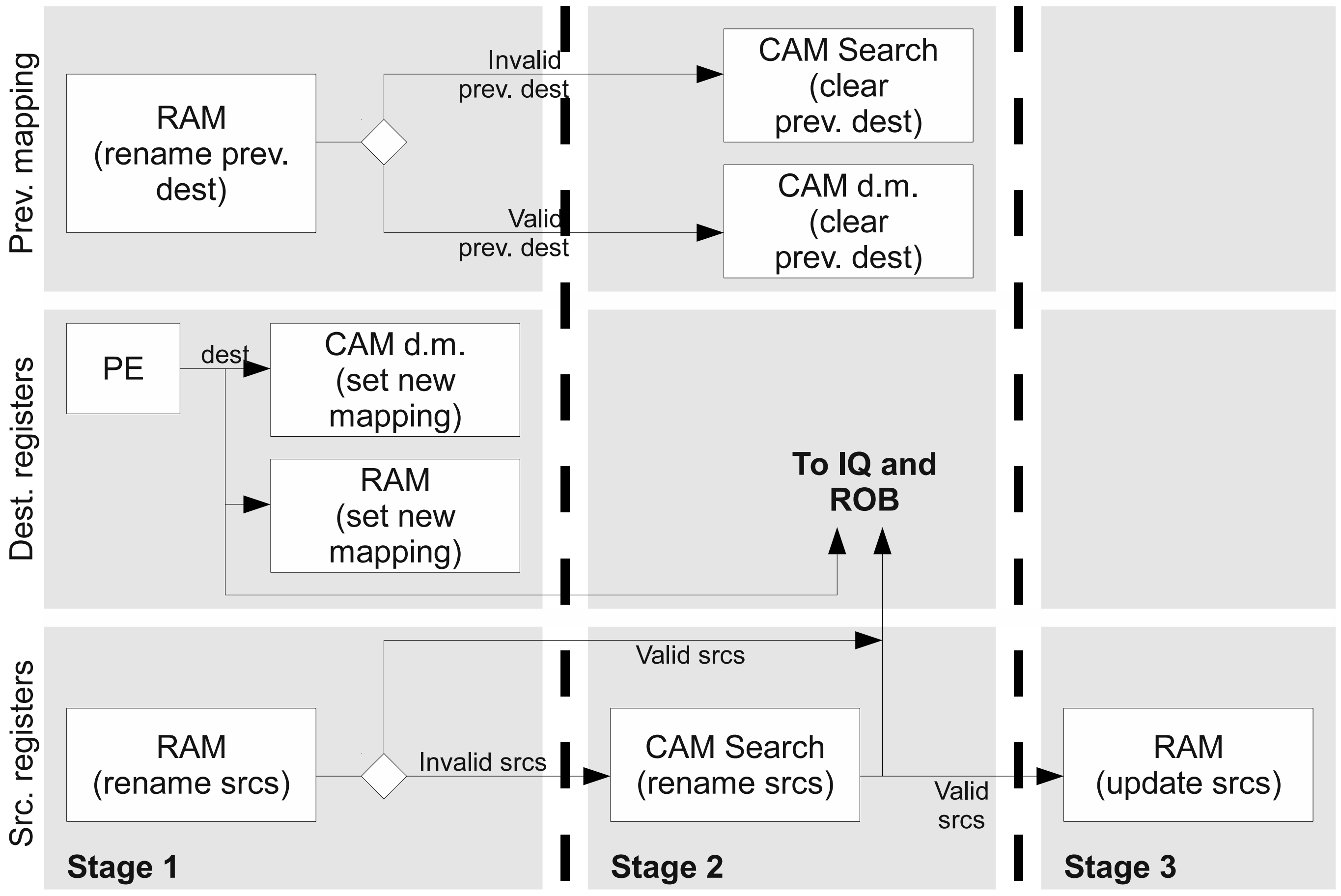


Figure 5. Block diagram of the Hybrid RAM-CAM renaming stages.

III. Hybrid RAM-CAM

The hybrid scheme uses two tables: i) a CAM containing all register mappings up to date, and ii) a RAM acting as a cache of the CAM, containing a subset of its renaming information. The CAM table can be indexed both directly by a physical register or associatively by a logical register, while the RAM is indexed by a logical register.

A RAM entry in the hybrid scheme may or may not contain a valid copy of the current mapping, as indicated by an additional *valid* bit per entry. Register renaming is performed by just accessing the RAM, while valid entries are hit. If an invalid entry is accessed, a RAM miss is said to occur, and the CAM is used to retrieve the current mapping. As a consequence, The CAM is not looked up in all renaming cycles, but only upon RAM misses, allowing for a lower number of CAM read/write ports compared to a typical CAM implementation. On misspeculation, the entire RAM contents are invalidated.

After recovery, the current mappings are only available in the CAM, since all RAM entries are invalid. Subsequent renaming of source registers will cause RAM misses, and the CAM will be looked up to obtain the mappings. Both CAM lookups and new register allocations will cause the RAM to be progressively updated, quickly reducing the RAM miss rate.

Let us analyze how the previous working example behaves on the hybrid RAM-CAM As instructions in Fig. 1a reach the rename stage, destination registers are mapped to new physical registers, and the new mapping is recorded both in the RAM and the CAM. After renaming instruction *I*, the contents of RAM and CAM tables is shown in Figs. 1b and 3, respectively.

Let us assume that register lookups are resolved in the RAM, but the branch instruction *F* is mispredicted, triggering the following recovery process. First, the RAM is invalidated by resetting all *valid* bits. Second, the CAM is recovered by restoring the current mapping with the branch checkpoint performed when instruction *F* was decoded (cp_F), returning to the state in Fig. 4.

Fig. 5 depicts a pipelined implementation of the hybrid scheme, where each box represents a table lookup. Lookups in the RAM are always direct-mapped, while CAM lookups can be either direct-mapped (*d.m.* in the figure) or associative searches. Only one table lookup is allowed in a single stage. This causes our proposal to be pipelined in three stages, though only two of them are in the critical path towards the instruction queues and the ROB.

The block diagram presented in Fig. 5 is also horizontally divided in three sections detailed next.

Clearing previous destination mappings. CAM entries corresponding to previous destination mappings are cleared. In the first stage, all previous destination mappings are looked up in the RAM. For those valid entries, physical register identifiers

Table II
MACHINE PARAMETERS

Branch predictor type	GShare: 16-bit GHR + 64K 2-bit counters Bimodal: 2K 2-bit counters
Branch Target Buffer	512-entry 4-way
Return Address Stack	32-entry
Issue policy	Out of order, 4 instructions/cycle
# of stages before rename	5
# of ROB entries	256
# of physical regs.	256
CPU to memory ports	2
L1 data cache	32KB, 4 way, 64 byte-line
L2 data cache	512KB, 8 ways, 64 byte-line
L1, L2, memory latencies	1, 10, 100 cycles

are obtained, which are then used in the second stage to directly index the CAM and clear the current mapping entries. On the contrary, invalid RAM entries cause previous mappings to be associatively cleared in the CAM.

Destination register renaming. Free physical registers are mapped to destination registers. The PE provides physical register identifiers from a set of free entries in the CAM. These identifiers are used to directly index the CAM and set the new mappings. New mappings are also updated in the RAM, which is indexed with the identifiers of the destination logical registers.

Source register renaming. For each source register, the associated mapped physical register identifier is obtained. In the first stage, the RAM is accessed. On a hit, mappings are available right away. Otherwise, an associative CAM search is performed in the second stage. Finally, those mappings retrieved from the CAM are updated in the RAM on the third stage. This last stage is optional and increases the RAM complexity. Nevertheless, it may provide performance and energy benefits if these updates avoid enough RAM misses.

Notice that previous mappings are cleared in the CAM in the second stage, while new mappings are set in the first stage. Thus, a hazard arises when an associative CAM lookup in the second stage for a given instruction accesses a mapping allocated by the same instruction in the first stage. This hazard can be avoided by flagging the new mapping entries at the end of the first stage in an additional single bit column in the CAM. The flags are reset at the end of the second stage.

IV. EXPERIMENTAL EVALUATION

A performance evaluation has been carried out on top of SimpleScalar, which has been modified to model the renaming approaches. Processor parameters are summarized in Table II.

Results have been obtained from the execution of the entire SPEC CPU2000 benchmark suite. Statistics were gathered using the *ref* input sets and single simulation points [9]. The SimpleScalar toolset has been configured for the Alpha ISA.

Five schemes have been analyzed, referred to as: *Commit*) RAM-based approach that triggers recovery at commit; *Writeback*) RAM-based approach that triggers recovery at writeback, walking the ROB from head to tail; *Writeback-fwalk*) RAM-based approach that triggers recovery at writeback, from tail to head; *Ideal CAM*) pure CAM-based approach; and *Hybrid*) RAM-CAM proposed approach. In addition, we will apply the suffix *-Iw* to the *Ideal CAM* and *Hybrid* schemes that reduce CAM complexity by limiting to I the number of instructions that the CAM can rename each cycle. As the baseline pipeline width is 4 instructions, values of I equal to or lower than 4 are evaluated.

The number of cycles incurred during misprediction recovery has been accurately modeled taking into account the position in the ROB of the mispredicted instruction, and the number of pipeline stages. In addition, the latency of the pipeline front-end to fetch the correct path is overlapped with the recovery penalty.

We assumed that a checkpoint is stored for each dispatched instruction group, as done in the IBM Power7 [10].

The baseline pipeline length resembles the ARM Cortex-A9 [11], with 10 stages (5 of them before rename). For the hybrid register renaming schemes, we assume a 12-stage pipeline.

A. Performance

1) Analysis on Short Pipelines: Unlike CAM-based, the Hybrid RAM-CAM approach performs associative searches on the CAM only upon RAM misses. Thus, a deliberate reduction of CAM complexity can have harmless consequences. This section explores the impact on performance of reducing the CAM complexity in the *Hybrid* approach.

Fig. 6 presents the IPC (Instructions executed Per Cycle) values for each benchmark under the *Ideal CAM-4w* renaming scheme. *Ideal CAM-4w* imposes an upper performance bound for the remaining models, since it takes just one processor cycle for both register renaming and misprediction recovery without negatively affecting the pipeline bandwidth.

Fig. 7 shows the performance slowdown for the analyzed schemes with respect to *Ideal CAM-4w*, calculated as $1 - \frac{IPC_{Renaming\ scheme}}{IPC_{Ideal\ CAM-4w}}$. Each bar in the figure represents the slowdown of the sequential execution of a benchmark set. Two variants of the hybrid scheme have been evaluated: *update sources* (US) and *non-update sources* (NUS). The NUS variant does not update the RAM for the source register mappings retrieved from the CAM. It requires one pipeline stage less than

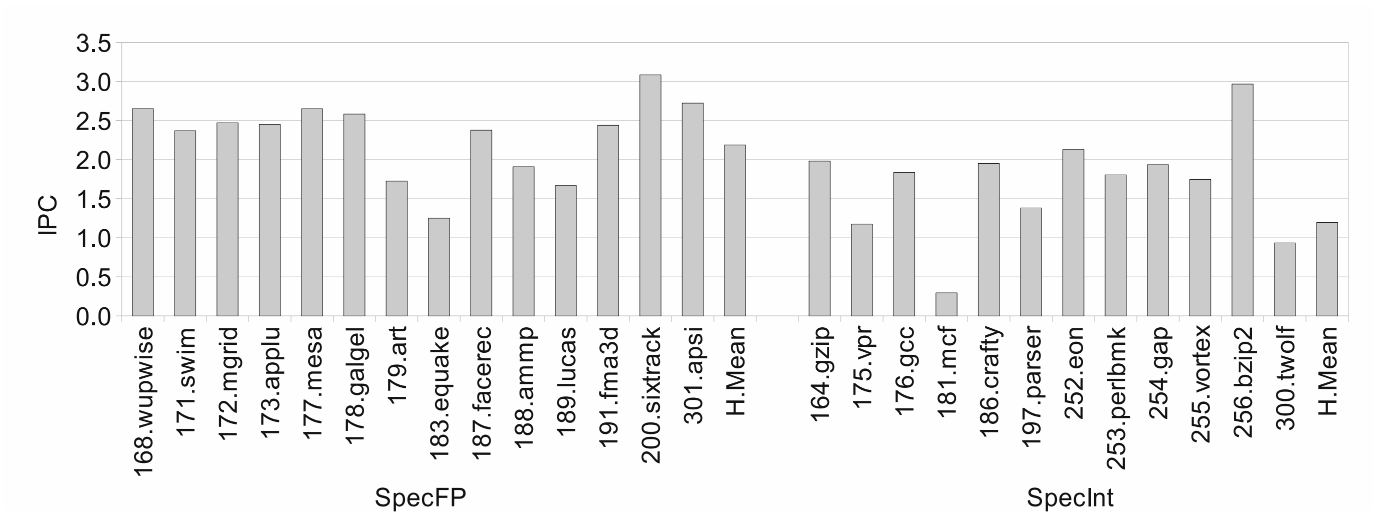


Figure 6. Performance of *Ideal CAM-4w*.

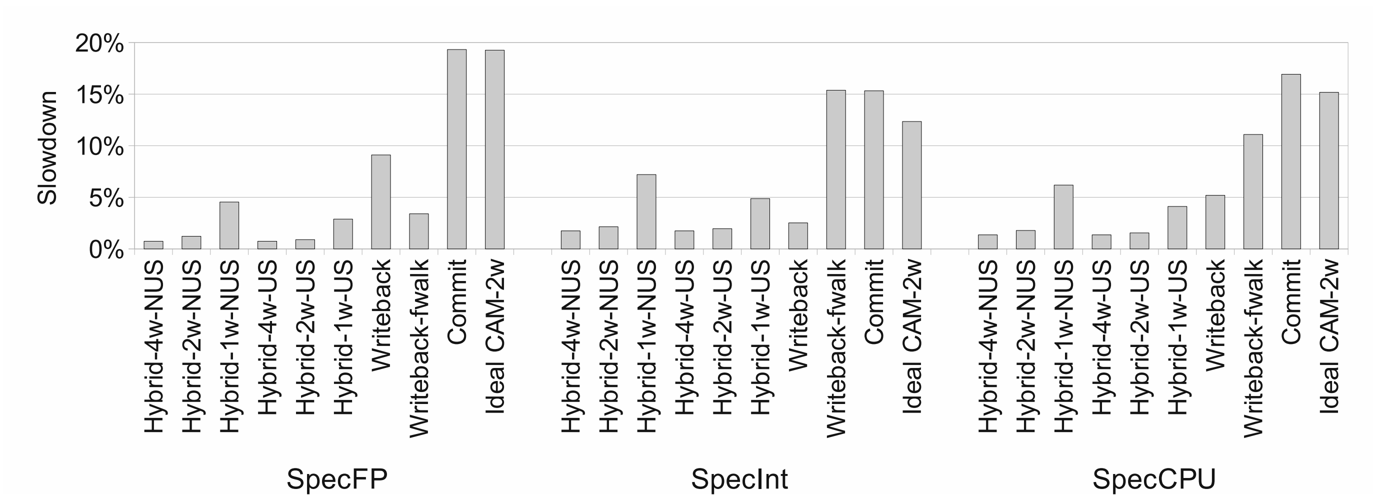


Figure 7. Performance slowdown with respect to *Ideal CAM-4w*.

the US variant, as the third stage shown in Fig. 5 is not longer needed. However, it incurs more RAM misses when looking for valid mappings in the RAM.

Writeback and *Writeback-fwalk* behave differently for integer and floating-point benchmarks. The reason is that the location of the mispredicted branch within the ROB is usually farther away from the ROB head in floating-point benchmarks than in integer ones. This is shown in Table III, which presents the average (arithmetic mean) number of instructions that must be scanned during recovery for each writeback variant. The reduction in number of scanned instructions is correlated with the results presented in Fig. 7 for the writeback approaches.

The *Commit* approach performs worse since its recovery penalty is usually higher. Also, a 2-way CAM-based configuration (*Ideal CAM-2w*) is included in the figure to show the impact on performance of reducing the CAM complexity by blindly halving the renaming bandwidth. Its slowdown for the whole SpecCPU (around 15%) is the second worst of the studied approaches. In contrast, *Hybrid-2w-NUS* presents a slowdown always smaller than 2.2%, suggesting that the additional RAM used in the Hybrid approach suffices to avoid that performance loss incurred by limited CAM ports.

Regardless of the hybrid variant, lower CAM bandwidths damage performance. The reason is that the rename stage stalls more often due to a lack of CAM ports. For SpecCPU, both variants of the hybrid approach outperform the conventional approaches, with the only exception of *Hybrid-1w-NUS*. Also, 2-way hybrid variants always provide better results than the writeback ones. Compared to *Ideal CAM-4w*, performance drops in the NUS variant by 1.4%, 1.8%, and 6.2% for *Hybrid-4w-NUS*, *Hybrid-2w-NUS*, and *Hybrid-1w-NUS*, respectively. These slowdowns are reduced by the US variant to 1.5%, and 4.1% for *Hybrid-2w-US* and *Hybrid-1w-US*, respectively. The reason behind this effect is that the US variant reduces the number of RAM misses, which in turn results in a lower number of searches on the CAM. This enhancement does not affect the performance of *Hybrid-4w* schemes because they have enough CAM ports to avoid stalling due to RAM misses. In fact, the

Table III
AVERAGE NUMBER OF SCANNED INSTRUCTIONS FOR WRITEBACK AND WRITEBACK-FWALK SCHEMES

Recovery scheme	SpecFP	SpecInt
Writeback	102.9	19.0
Writeback-fwalk	41.8	59.7

Table IV
RELATIVE CAM SEARCHES FOR THE HYBRID APPROACHES VARYING THE NUMBER OF CAM WAYS

	SpecInt	SpecFP	SpecCPU
Hybrid-1w-NUS	16.0%	8.4%	12.1%
Hybrid-2w-NUS	17.0%	8.5%	12.6%
Hybrid-4w-NUS	17.1%	8.5%	12.6%
Hybrid-1w-US	7.5%	2.9%	5.1%
Hybrid-2w-US	7.8%	2.9%	5.3%
Hybrid-4w-US	7.9%	2.9%	5.3%

slowdown observed in these cases is only due to the higher number of pipeline stages.

Table IV shows the percentage of CAM searches performed by the hybrid approaches with respect to *Ideal CAM-4w*. The US variant roughly halves the number of searches performed by NUS, which is the reason for its better performance. For SpecCPU, the percentage in the US variant lies around 5%, while this value is particularly low (below 3%) for floating-point benchmarks.

Performance of *Hybrid-2w* approaches falls very close not only to the *Hybrid-4w* ones but also to *Ideal CAM-4w*, for both integer and floating-point benchmarks. The reason can be inferred from Fig. 8, which presents the cumulative execution time for all benchmarks. As observed, the stalled time due to CAM ports constraints (black portion of each bar) is higher for integer benchmarks, which explains the higher slowdown exhibited by the hybrid approach. On average, the total bar heights for *Hybrid-2w* and *Hybrid-4w* are very similar, which means that a 2-way CAM is enough to avoid performance loss due to renaming constraints.

Finally, let us compare performance across individual benchmarks. Fig. 9 shows the results for *Commit*, *Writeback*, and *Hybrid-US* variants. The US variant has been selected as representative for the hybrid approaches since it offers better performance with a lower number of CAM accesses than the NUS variant. Although the decision on which scheme performs best is benchmark-dependent, *Hybrid-2w-US* and *Hybrid-4w-US* perform closest to the baseline for most applications. In some cases (e.g., *wupwise*, *galgel*, *facerec*, *ammp*, *vpr*, and *gap*), performance is especially affected by misprediction penalties, and both *Writeback* and *Writeback-fwalk* incur a slowdown higher than 5%. In contrast, the slowdown of *Hybrid-2w-US* is lower than this mark for all benchmarks except *eon*.

2) Impact of Long Pipelines: Long pipelines enable higher clock frequencies by simplifying the amount of work to be done in each pipeline stage. For example, [3], [12], and [10] have a pipeline depth of around 20 stages. As a side effect, branch misprediction penalties become more significant in terms of number of cycles. To analyze the performance of the hybrid approach with long pipelines, we assume in this section a 20-stage pipeline. Similarly to the Pentium 4 architecture, 6 of these stages are assumed to lie before the register renaming stage. Two extra stages (22 stages in total) are again assumed for the hybrid approach.

Fig. 10 shows the results as a slowdown (in processor cycles) with respect to the baseline *Ideal CAM-4w*. Comparing these results with the ones obtained for short pipelines, we can see that hybrid approaches exhibit insignificant variations in slowdown. In general, slowdowns slightly grow for floating-point benchmarks and shrink, also subtly, for integer benchmarks. *Writeback* shows the opposite trend: the number of occupied ROB entries is much higher for floating-point benchmarks, and thus misprediction penalty slightly increases with the pipeline depth. Overall, the most negatively affected renaming scheme is *Writeback-fwalk*, mainly because a late misspeculation detection increases recovery penalties when walking the ROB from the tail. This effect is especially significant in floating-point applications.

B. Hardware Complexity

Table V lists the implemented memory structures, as well as their number of read (r) and write (w) ports. The last column of the table summarizes the total area occupied by each renaming scheme. Results have been obtained with CACTI 6 toolset (http://www.hpl.hp.com/research/cacti/) for a 45nm technology node. [1]

Let us analyze the hybrid design (see Fig. 5) from a complexity point of view. In the first stage, the RAM table is queried for previous destination mappings, and the PE allocates new mappings at the same entries ($4r$ and $4w$ RAM ports required).

[1]The CAM was modeled as a fully associative SRAM_CACHE, with 256 entries, 1 byte/entry (only one bit —the valid bit— is actually required per entry, but 1 byte is the smallest entry size supported by CACTI), and a 6-bit tag size (i.e., 64 logical registers). The RAM, ROB, and FRQ were modeled as direct-mapped SRAM structures with no tags. RAM and RRAM are composed of 64 entries and store 1 byte per entry. The ROB contains 256 two-byte entries that store the current and previous mapping. Each entry of the ROB only considers information related with register renaming; other instruction metadata is neglected (canceled out) for comparison purposes. The FRQ table implements 256 one-byte entries to identity a free physical register.

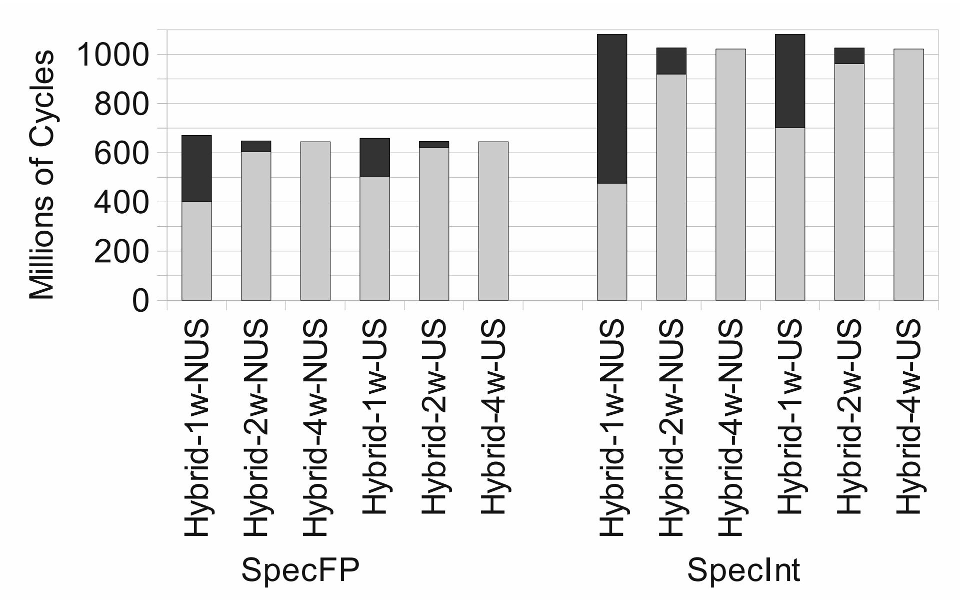


Figure 8. Total execution time and stalled time due to a lack of CAM ports.

Table V
OVERALL AREA REQUIRED BY EACH SCHEME AND COMPLEXITY OF ITS COMPONENTS

Scheme	PE	FRQ	ROB	RAM	RRAM	CAM assoc.	CAM d.m.	Area (mm^2)
Ideal CAM-4w	yes	–	–	–	–	8r+4w	4w	0.049
Commit	–	4r+4w	4r+4w	12r+4w	4r+4w	–	–	0.059
Writeback	–	4r+4w	4r+4w	12r+4w	4r+4w	–	–	0.059
Writeback-fwalk	–	4r+4w	4r+4w	12r+4w	–	–	–	0.055
Hybrid-1w-NUS	yes	–	–	12r+4w	–	2r+1w	8w	0.046
Hybrid-2w-NUS	yes	–	–	12r+4w	–	4r+2w	8w	0.052
Hybrid-4w-NUS	yes	–	–	12r+4w	–	8r+4w	8w	0.073
Hybrid-1w-US	yes	–	–	12r+6w	–	2r+1w	8w	0.050
Hybrid-2w-US	yes	–	–	12r+8w	–	4r+2w	8w	0.061
Hybrid-4w-US	yes	–	–	12r+12w	–	8r+4w	8w	0.092

Source operands are renamed by also accessing the RAM (additional $8r$ RAM ports). The US hybrid variant additionally updates the RAM with the source registers involved in previous RAM misses on the third pipeline stage (additional $2w$, $4w$, and $8w$ RAM ports for *Hybrid-1w-US*, *Hybrid-2w-US*, and *Hybrid-4w-US*, respectively).

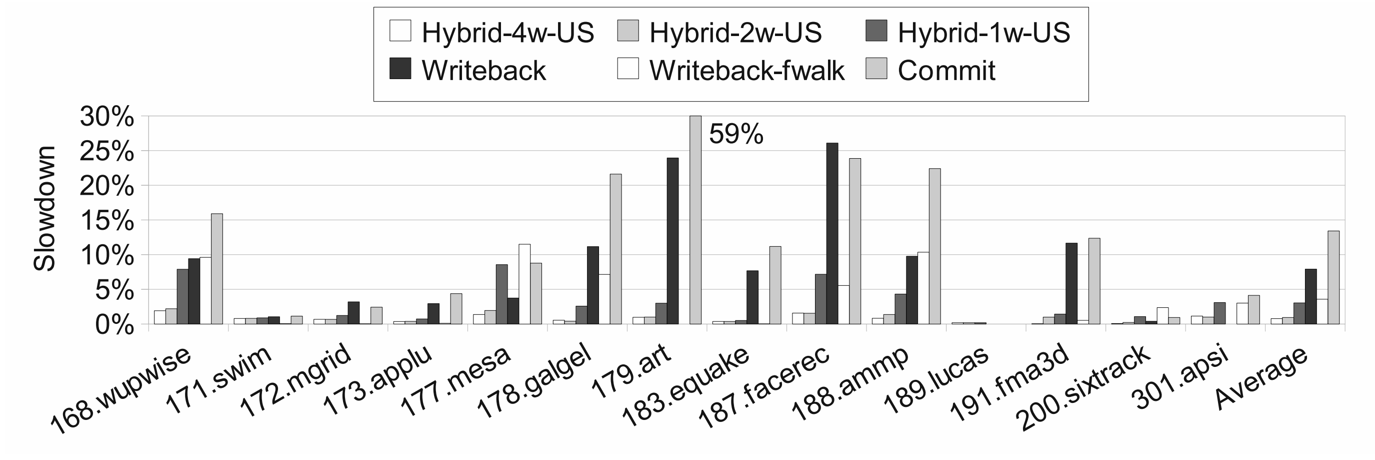
Associative CAM ports in the hybrid designs are used in the second stage to rename sources ($2r$, $4r$, or $8r$ CAM ports) and to clear previous destination mappings ($1w$, $2w$, or $4w$ CAM ports) that missed in the RAM. However, if physical registers are correctly provided by the RAM, previous destinations can be cleared with a direct-mapped (d.m.) access. Notice that direct-mapped ports are simpler than associative ports, so different hybrid configurations keep a constant number of them ($4w$ d.m. CAM ports). The remaining $4w$ direct-mapped ports ($8w$ d.m. CAM ports in total) are used in the first stage to allocate the new mappings provided by the PE.

Regarding *Ideal CAM-4w*, free physical registers are also provided by the PE. Therefore, as in the hybrid schemes, the CAM is accessed to allocate new destinations ($4w$ d.m. CAM ports). On the other hand, the CAM is associatively searched to rename source registers and clear previous destination mappings (in total, $8r$+$4w$ associative CAM ports).

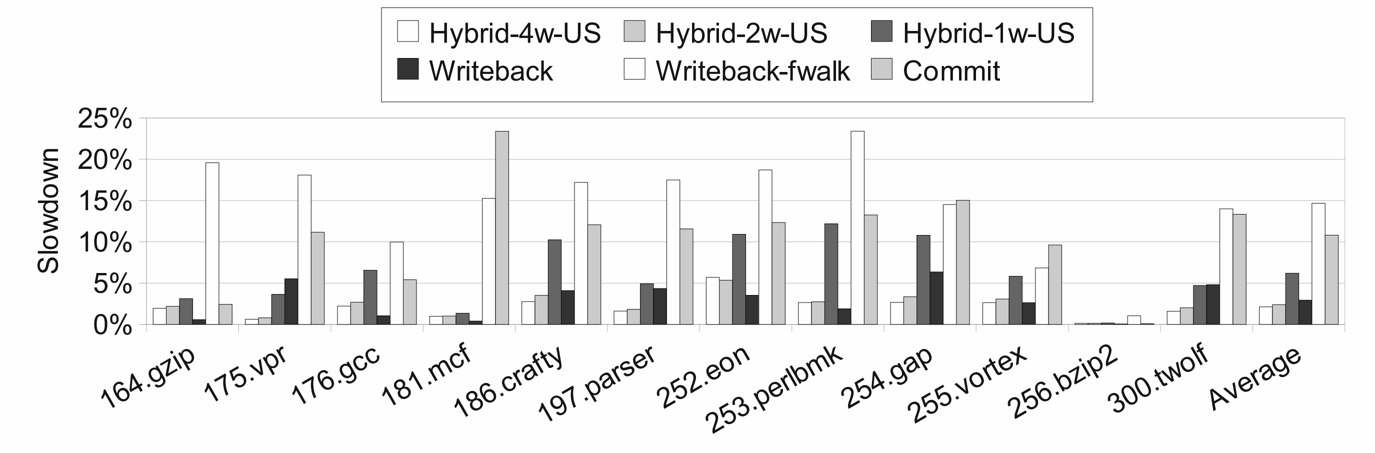
RAM-based designs (i.e., *Commit*, *Writeback*, and *Writeback-fwalk*) use the RAM to rename source registers ($8r$ RAM ports), as well as to look up previous destination mappings and update them with new values (additional $4r$ and $4w$ RAM ports). In addition, *Commit* and *Writeback* use a RRAM ($4r$ and $4w$ ports) where destination mappings are updated when instructions commit.

All RAM-based designs use the ROB to store ($4w$ ROB ports) and retrieve ($4r$ ROB ports) renaming information. In addition, RAM-based designs use an FRQ to allocate ($4r$ FRQ ports) and release ($4w$ FRQ ports) physical registers.

Table VI shows some technological features, including area, access time, energy per access, and leakage per nanosecond for each discussed hardware component. For comparison purposes, the area and access time of the PE required by our proposal have been assumed to be the same as the FRQ structure. This assumption is conservative, since consumption on the PE is a negligible fraction of the total CAM consumption [13]. In addition, for the CAM components, we include the features corresponding to the direct-mapped ports (labeled as d.m. in the table). Results also consider the contribution of the direct-mapped ports to the



(a)



(b)

Figure 9. Detailed slowdowns for individual benchmarks. (a) SpecFP. (b) SpecInt.

CAM area.

Table V shows the area occupied by each renaming scheme, calculated as the sum of each component. *Hybrid-1w* schemes have the smallest area occupancy thanks to the reduction of the CAM complexity, as well as the lack of ROB area devoted to renaming. On the other hand, the most area-hungry designs are the *Hybrid-4w* schemes, as they require both complex CAM and RAM structures. The *Hybrid-2w-US* scheme has an area (0.061 mm^2) close to *Writeback* and *Commit*.

Finally, the reported access time refers to the elapsed time since a table lookup starts until the operation completes. All components exhibit an access time lower than 0.25ns. Therefore, 4GHz is the maximum frequency at which the implementation proposed in Section III can function.

C. Energy Consumption

We measured dynamic energy as the total number of accesses to each component multiplied by the energy per access. Leakage (or static) energy was calculated as the total number of execution cycles times the total leakage energy per cycle, assuming a 1GHz clock frequency.

Fig. 11 shows the energy budget used for register renaming for a 10/12-stage pipeline. Dissipation of leakage energy lies between 20% to 35% of the total energy for all renaming schemes except *Ideal CAM-4w*. Leakage energy is lower for *Ideal CAM-4w* and *Hybrid* schemes because their execution is shorter and do not require additional ROB storage for renaming purposes.

Dynamic energy is distributed in the figure by component (FRQ, ROB, RAM, RRAM, direct-mapped CAM lookups, and associative CAM searches). The energy spent to recover a correct RAM state has been estimated by accounting for the copy of the RRAM renaming data to the RAM (*Commit* and *Writeback*) and the subsequent walk through the ROB (*Writeback* and *Writeback-fwalk*). The ROB, FRQ, and CAM structures are organized as circular queues, and recovered with the negligible cost of a pointer update. The cost of invalidating all RAM entries in the hybrid designs is also negligible, since this operation only entails resetting a single bit column.

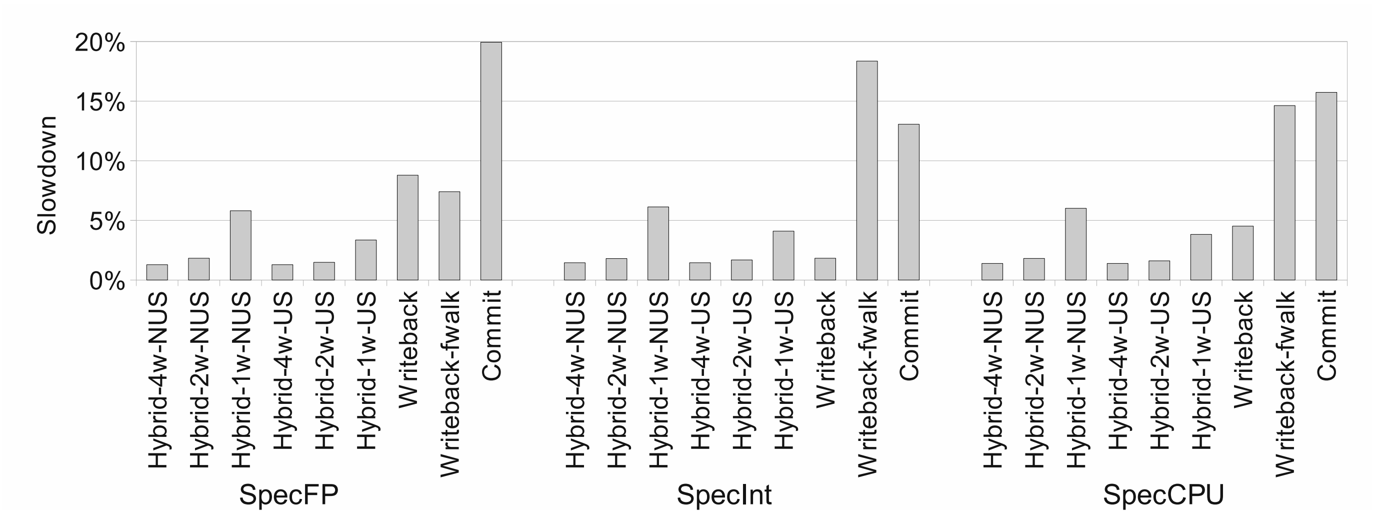


Figure 10. Impact of a 20/22-stage pipeline on performance slowdown results.

Table VI
AREA, ACCESS TIME, ENERGY PER ACCESS, AND LEAKAGE PER CYCLE FOR EACH COMPONENT

Component	Area (mm^2)	Access time (ns)	Energy per access (pJ)	Leakage per ns (pJ)
FRQ (PE)	0.014	0.157	0.434	0.871
ROB	0.026	0.200	0.892	1.714
RAM 12r+4w	0.015	0.168	0.406	0.448
RAM 12r+6w	0.019	0.176	0.448	0.491
RAM 12r+8w	0.023	0.185	0.490	0.534
RAM 12r+12w	0.033	0.202	0.574	0.618
RRAM 4r+4w	0.005	0.133	0.252	0.293
CAM 2r+1w (8w d.m.)	0.018	0.172	7.833 (0.327 d.m.)	1.302
CAM 4r+2w (8w d.m.)	0.024	0.189	8.751 (0.327 d.m.)	1.444
CAM 8r+4w (8w d.m.)	0.045	0.224	10.590 (0.327 d.m.)	1.731
CAM 8r+4w (4w d.m.)	0.035	0.224	10.590 (0.214 d.m.)	1.423

The *Ideal CAM-4w* design consumes about one order of magnitude more power than the rest of the schemes, since the CAM is associatively accessed every cycle for all mappings. On the contrary, *Writeback-fwalk* shows up as the best RAM-based in this regard. *Writeback* and *Commit* suffer mainly from RRAM costs. In the latter design, there are additional energy costs because of the late misspeculation detection, which causes more mispredicted instructions to be renamed before triggering recovery.

Besides providing performance close to *Ideal CAM-4w*, hybrid designs drastically alleviate energy dissipation. For SpecFP, they consume less than the *Commit* scheme. Indeed, all US variants except *Hybrid-4w-US* consume less energy than the *Writeback-fwalk* scheme. Regarding SpecInt, *Hybrid-2w-US* and *Hybrid-1w-US* present less energy consumption than both *Writeback* and *Writeback-fwalk*.

An increase of the CAM width in hybrid designs results in better performance but a higher energy cost. The reason is that the number of stalls due to limited CAM bandwidth decreases as the number of ports increases, but this also implies a higher number of useless accesses to the CAM when speculative execution takes place, as well as a higher cost per access due to more complex RAM and CAM structures. On the other hand, US variants show lower energy costs and better performance than NUS variants, in spite of requiring more complex RAM structures than NUS variants. The reason is that updating the RAM more often reduces the amount of associative CAM accesses.

Fig. 12 provides the energy consumption results for a 20/22-stage pipeline. The US variants provide again better consumption results than the NUS ones. The latter are excluded from the figure for the sake of clarity. In general, the energy dissipation for all presented designs is higher than in the short pipeline, because misspeculation is detected later.

Table VII shows the efficiency of the studied renaming approaches for SpecInt, SpecFP, and SpecCPU. Efficiency values have been quantified as the consumed energy multiplied by the square of execution time (energy-delay-square product), as suggested in [14]. In the short pipeline, the highest efficiency (i.e., the lowest product value) is provided by *Hybrid-2w-US* and *Hybrid-1w-US* across all sets of benchmarks except for SpecInt in the 20/22-stage pipeline.

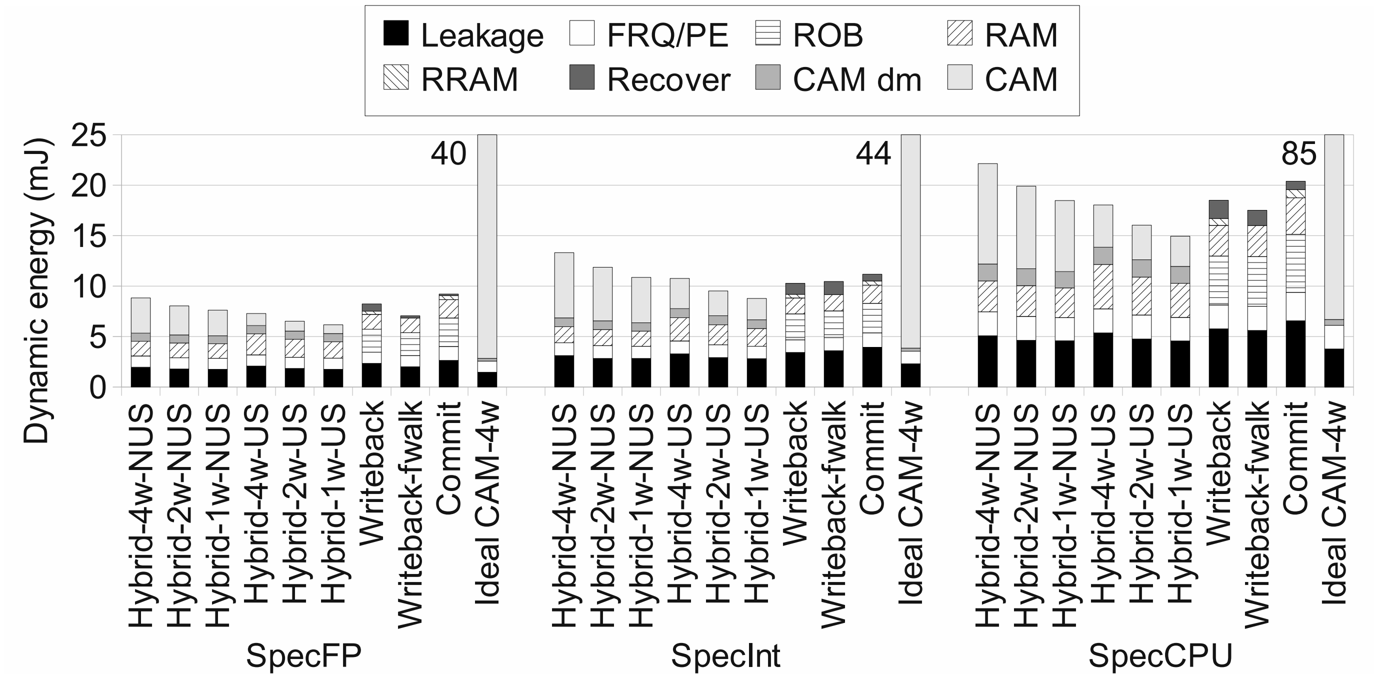


Figure 11. Dynamic energy spent in a 10/12-stage pipeline.

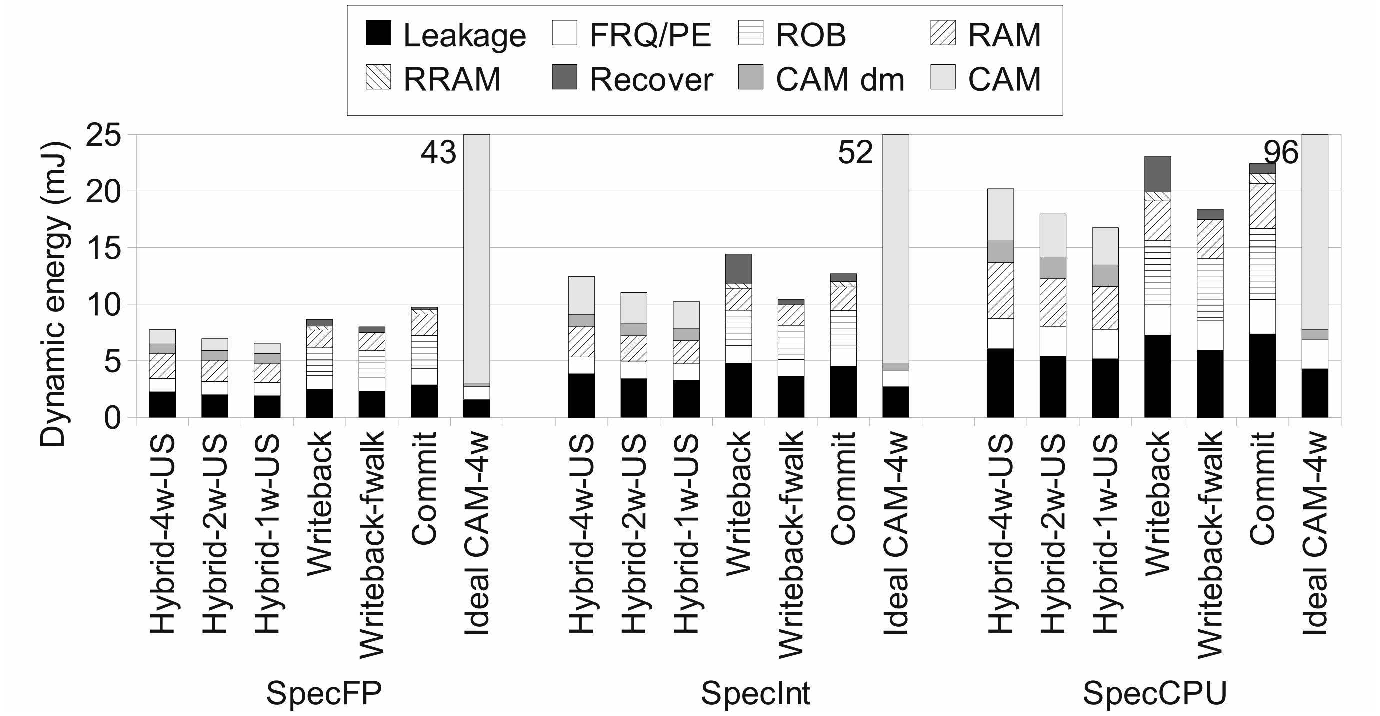


Figure 12. Dynamic energy spent in a 20/22-stage pipeline.

V. Related Work

Regarding RAM-based register renaming approaches, Moshovos [15] proposes to reduce the number of ports in the front-end RAM by detecting those instructions that do not use the maximum number of source and destination register operands. With the same aim, Kucuk et al. [16] further reduce the number of accesses to the front-end RAM by forwarding results of previous accesses performed by instructions nearby.

Concerning the recovery penalty incurred by RAM-based schemes, Moshovos [1] proposes an out-of-order release mechanism which reduces the number of RAM checkpoints to about one third. In [17], Akl et al. propose a ROB-like structure to accelerate

Table VII
EFFICIENCY ($Energy \cdot Delay^2/10^{24}$) (A) $10/12$-STAGE PIPELINE (B) $20/22$-STAGE PIPELINE

	SpecFP	SpecInt	SpecCPU
Commit	5.80	15.71	79.85
Writeback	4.08	10.89	55.64
Writeback-fwalk	3.10	14.72	59.87
Hybrid-1w-US	2.68	9.77	43.91
Hybrid-2w-US	2.72	9.98	44.71
Hybrid-4w-US	3.03	11.23	50.10

(a)

	SpecFP	SpecInt	SpecCPU
Commit	7.21	23.24	109.88
Writeback	4.79	29.96	110.19
Writeback-fwalk	4.56	14.93	70.21
Hybrid-1w-US	3.33	15.38	63.06
Hybrid-2w-US	3.40	15.79	64.62
Hybrid-4w-US	3.78	17.74	72.29

(b)

checkpoint recovery, which allows misspeculation recovery from specific branches. Similarly, a selective checkpoint mechanism to recover mispredictions and support large instruction windows is proposed in [18].

In a closely related work [19], Zhou et al. propose a mechanism to allow the processor to continue executing instructions after a misspeculation while the processor state is being restored, effectively hiding the recovery latency. This technique allows RAM-based approaches to dispatch instructions without waiting for the misspeculated branch instruction to reach the commit stage or scanning the ROB. It requires additional logic to correctly manage the issue stage and the instruction queues. In this sense, it is orthogonal to alternative mapping implementations like the one proposed in this work.

The RAM approach has been used in successful commercial processors [2], [3]. The CAM approach has similarly succeeded in aggressive designs [4], requiring only a single memory structure for fast recovery and multiple checkpoints [7]. It becomes then a major research concern to reduce the access time incurred by CAMs. Buti et al. [4] detail how this problem is addressed for the IBM Power4 processor. Also, Liu and Lu [20] explore the effect of circuit-level speculation to speed up the response of a CAM renaming table.

In a more recent work [7], Safi et al. compare the energy and latency of RAM and CAM approaches. They conclude that, when the number of checkpoints exceeds a given threshold, CAM approaches become more efficient and faster. They also propose to selectively disable CAM entries in order to optimize CAM energy consumption.

Finally, Wallace and Bagherzadeh also propose a hybrid RAM-CAM design [21]. Their goal is to reduce the RAM complexity and access time in RAM-based renaming schemes. The authors implement a small ROB-like FIFO queue located before the RAM which is also associatively addressable by a logical register identifier. This table reduces the number of required RAM ports (much like our design reduces the number of required CAM ports, which are more costly), and allows recovery of the correct mappings in one cycle only when mispredicted instructions have not updated the RAM. However, register release, pipelining, and other complexity issues are not tackled.

VI. CONCLUSION

In this paper, we have presented a renaming mechanism consisting of a RAM table and a low-complexity CAM table, as a hybrid design that takes the best of both approaches. Experimental results show that a 2-way hybrid approach achieves small performance slowdowns (about 2% and 1% for integer and floating-point benchmarks, respectively) with respect to a 4-way CAM-based renaming mechanism that is able to recover in one clock cycle. These small slowdowns are accompanied by a drastic reduction of the original associative searches carried out in the CAM-based approach to only 8% and 3%. Hybrid designs also reduce the dynamic energy by 16% and 12% with respect to the original CAM consumption, closing the dynamic energy consumption gap between CAM and RAM approaches. Besides general performance improvements, hybrid designs are proved to be more efficient than the simplest non-checkpointed RAM approaches in terms of both area and energy. Finally, experiments show that performance benefits span different processor configurations, whether with short or long pipelines.

REFERENCES

[1] A. Moshovos, "Checkpointing Alternatives for High-Performance, Power-Aware Processors," in *Proc. Int. Symp. Low Power Electron. Design*, 2003, pp. 318–321.

[2] K. C. Yeager, "The MIPS R10000 Superscalar Microprocessor," *IEEE Micro*, vol. 16, no. 2, pp. 28–40, 1996.

[3] G. Hinton, D. Sager, M. Upton, D. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Rousell, "The Microarchitecture of the Pentium 4 Processor," *Intel Technol. J.*, vol. 1, p. 2001, 2001.

[4] T. N. Buti, R. G. McDonald, Z. Khwaja, A. Ambekar, H. Q. Le, W. E. Burky, and B. Williams, "Organization and implementation of the register-renaming mapper for out-of-order IBM POWER4 processors," *IBM J. Res. Develop.*, vol. 49, no. 1, pp. 167–188, 2005.

[5] R. Kalla, B. Sinharoy, and J. M. Tendler, "IBM Power5 Chip: A Dual-Core Multithreaded Processor," *IEEE Micro*, vol. 24, no. 2, pp. 40–47, 2004.

[6] R. E. Kessler, "The Alpha 21264 Microprocessor," *IEEE Micro*, vol. 19, no. 2, pp. 24–36, Mar. 1999.

[7] E. Safi, A. Moshovos, and A. Veneris, "A Physical Level Study and Optimization of CAM-based Checkpointed Register Alias Table," in *Proc. 13th Int. Symp. Low Power Electron. Design*, 2008, pp. 233–236.

[8] H. Akkary, R. Rajwar, and S. T. Srinivasan, "Checkpoint Processing and Recovery: Towards Scalable Large Instruction Window Processors," in *Proc. 36th Annu. IEEE/ACM Int. Symp. Microarch.*, 2003, pp. 423–434.

[9] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," in *Proc. 10th Int. Conf. Archit. Support Program. Languages Oper. Syst.*, 2002, p. 45.

[10] B. Sinharoy, R. Kalla, W. J. Starke, H. Q. Le, R. Cargnoni, J. A. Van Norstrand, B. J. Ronchetti, J. Stuecheli, J. Leenstra, G. L. Guthrie, D. Q. Nguyen, B. Blaner, C. F. Marino, E. Retter, and P. Williams, "IBM POWER7 multicore server processor," *IBM J. Res. Develop.*, vol. 55, no. 3, pp. 191–219, 2011.
[11] ARM, "Cortex-A9 Processor Microarchitecture," Public Presentation, 2007, available at http://www.arm.com/products/processors/cortex-a.
[12] M. Butler, L. Barnes, D. D. Sarma, and B. Gelinas, "Bulldozer: An Approach to Multithreaded Compute Performance," *IEEE Micro*, vol. 31, no. 2, pp. 6–15, 2011.
[13] B. Agrawal and T. Sherwood, "Modeling TCAM power for next generation network devices," in *Proc. IEEE Int. Symp. Perform. Anal. Systems Softw.*, 2006, pp. 120–129.
[14] M. Martonosi, D. Brooks, and P. Bose, "Modeling and Analyzing CPU Power and Performance: Metrics, Methods, and Abstractions," *SIGMETRICS 2001 / Performance 2001 - Tutorials*, 2001.
[15] A. Moshovos, "Power-Aware Register Renaming," *Technical Report, Computer Engineering Group, University of Toronto*, 2002.
[16] G. Kucuk, O. Ergin, D. Ponomarev, and K. Ghose, "Reducing Power Dissipation of Register Alias Tables in High-Performance Processors," *IEE Proc. Comput. Digit. Techn.*, vol. 152, no. 6, pp. 739–746, 2005.
[17] P. Akl and A. Moshovos, "Turbo-ROB: A Low Cost Checkpoint/Restore Accelerator," *Lecture Notes in Computer Science*, vol. 4917, pp. 258–272, 2008.
[18] H. Akkary, R. Rajwar, and S. T. Srinivasan, "An Analysis of a Resource Efficient Checkpoint Architecture," *ACM Trans. Archit. Code Optim.*, vol. 1, no. 4, pp. 418–444, 2004.
[19] P. Zhou, S. Önder, and S. Carr, "Fast Branch Misprediction Recovery in Out-of-Order Superscalar Processors," in *Proc. 19th Int. Conf. Supercomput.*, 2005, pp. 41–50.
[20] T. Liu and S. L. Lu, "Performance Improvement with Circuit-Level Speculation," in *Proc. 33rd Annu. IEEE/ACM Int. Symp. Microarch.*, 2000, pp. 348–355.
[21] S. Wallace and N. Bagherzadeh, "A scalable register file architecture for dynamically scheduled processors," in *Proc. Conf. Parallel Archit. Compil. Techn.*, 1996, p. 179.

Salvador Petit received the PhD degree in computer engineering from the Universitat Politècnica de València, Spain.

Currently, he is an associate professor in the Department of Computer Engineering at the UPV where he has taught several courses on computer organization. His research topics include multithreaded and multicore processors, memory hierarchy design, as well as real-time systems.

Prof. Petit is a member of the IEEE and the IEEE Computer Society.

Rafael Ubal obtained his PhD Degree in Computer Engineering in 2010 from Universitat Politècnica de València (Spain).

He works currently as a Lecturer in the Electrical and Computer Engineering Department at Northeastern University (Boston, MA), and Postdoctoral Associate Researcher His research topics of interest include power-aware cache designs, clustered/multithreaded/multicore architectures and GPGPU. He is the main developer of the Multi2Sim simulation framework.

Julio Sahuquillo received his BS, MS, and PhD degrees in Computer Engineering from the Universitat Politècnica de València (Spain).

Since 2002 he is an Associate Professor at the Department of Computer Engineering. He has taught several courses on computer organization and architecture. He has published more than 100 refereed conference and journal papers. His current research topics include multi- and manycore processors, memory hierarchy design, cache coherence, and power dissipation.

Prof. Sahuquillo is a member of the IEEE Computer Society.

Pedro López received his PhD degree in Computer Engineering from Universitat Politècnica de València (Spain) in 1995.

Since 2002, he is a Full Professor in the Department of Computer Engineering at UPV, Spain. His research interests include high performance interconnection networks for multiprocessor systems, clusters and networks on chip, and, more recently, processor microarchitecture and cache design. Prof. López has published more than 120 refereed conference and journal papers.

Prof. López is a member of the IEEE Computer Society.