***Abstract*—This paper investigates how the latency and energy of register alias tables (RATs) vary as a function of the number of global checkpoints (GCs), processor issue width, and window size. It improves upon previous RAT checkpointing work that ignored the actual latency and energy tradeoffs and focused solely on evaluating performance in terms of instructions per cycle (IPC). This work utilizes measurements from the full-custom checkpointed RAT implementations developed in a commercial 130-nm fabrication technology. Using physical- and architectural-level evaluations together, this paper demonstrates the tradeoffs among the aggressiveness of the RAT checkpointing, performance, and energy. This paper also shows that, as expected, focusing on IPC alone incorrectly predicts performance. The results of this study justify checkpointing techniques that use very few GCs (e.g., four). Additionally, based on full-custom implementations for the checkpointed RATs, this paper presents analytical latency and energy models. These models can be useful in the early stages of architectural exploration where actual physical implementations are unavailable or are hard to develop. For a variety of RAT organizations, our model estimations are within 6.4% and 11.6% of circuit simulation results for latency and energy, respectively. This range of accuracy is acceptable for architectural-level studies.**

***Index Terms*—Computer architecture, delay and power modeling, implementation, microprocessors, register alias table (RAT).**

I. Introduction

THE Register Alias Table (RAT), the core of register renaming, is a performance-critical component of modern dynamically-scheduled processors. Register renaming eliminates false data dependencies and increases *instruction level parallelism* (ILP). The RAT is read and updated by all instructions in order as they are decoded. Hence, it must operate at the processor's clock frequency or it must be pipelined.

RAT complexity and size and, hence, latency and energy depend on several architectural parameters: The wider the issue width, the more heavily ported the RAT must be. Furthermore, as the instruction window size increases, so does the number of physical registers and, hence, the width of each RAT entry. RAT complexity and size also increase with the number of simultaneous threads supported by the processor. In modern processors, RAT complexity is increased further by the use of speculation, control flow, or otherwise. On mispeculations, the RAT content must be restored such that it does not contain any of the mappings introduced by incorrectly-speculated instructions. Accordingly, modern RAT designs incorporate a set of global checkpoints (GCs) to recover from mispeculations. A GC contains a complete snapshot of all relevant processor states including the RAT and, thus, can be used to recover from control-flow mispeculations. Recovery at an instruction using a GC is "instantaneous," i.e., it requires a fixed, low latency.

In early processor designs, GCs were allocated to every speculated branch [26]. This technique was feasible because very few GCs were sufficient to achieve high performance. Modern processors, however, use much larger instruction windows (e.g., 128 versus 32) and, hence, require considerably more GCs to maintain high performance. Accordingly, recent work assumed that the policy of allocating a GC to every speculated branch is impractical for modern processors [1], [2], [8], [18]. These studies developed GC count reduction techniques focusing on *instructions-per-cycle* (IPC) performance evaluation to compare alternatives. However, it is well understood that IPC does not predict the performance of the techniques that impact the clock period. Determining the actual relation between the number of GCs and performance is imperative for understanding whether existing state-of-the-art RAT checkpointing solutions work sufficiently well or whether further innovations are required.

This paper improves upon previous RAT checkpointing work by investigating how increasing the number of GCs affects the RAT latency and, thus, actual performance (execution time). Specifically, this work studies how the latency and energy of the RAT vary as a function of the number of GCs, the issue width, and the window size. The various RAT configurations are implemented in a commercial 130-nm technology. The results show that only a limited number of GCs can be implemented without impacting the clock cycle significantly, thus reducing overall performance. In addition, as energy consumption has become a major design consideration, this work also studies how energy varies for various checkpointing designs. To the best of our knowledge, no previous work determined RAT latency and energy variation trends as a function of the aforementioned architectural parameters.

Unlike previous work that primarily focused on architectural-level evaluation, this paper relies on both physical- and architectural-level evaluations to study the actual performance and energy impact of GC count. For the architectural-level evaluation, both conventional and state-of-the-art confidence-based methods for selectively allocating GCs are considered [3]. This paper shows that ignoring the actual delay of the RAT incor-

rectly predicts performance: In particular, performance does not monotonically increase with GC count as IPC measurements suggest. Two components determine actual performance: First, with more GCs, fewer cycles are spent recovering from mispeculations, hence improving performance. Second, introducing more GCs increases RAT latency and, hence, increases the clock period and decreases performance. In most cases, using very few GCs (e.g., four) leads to optimal performance.

Previous work relied on analytical models of register files, which were not adjusted to appropriately model a checkpointed RAT. To facilitate further RAT checkpointing studies, this paper presents analytical models for the latency and energy of the checkpointed RATs. These models can predict RAT latency and energy variation as a function of several parameters. These models can help computer architects estimate the latency and energy of various RAT organizations without involving in the actual physical-level implementation. These models are useful during early architectural-level exploration where physical-level implementation is either impossible to develop or cannot be afforded due to time and/or cost constraints. The model estimations are within 6.4% and 11.6% of Spectre circuit simulation results for the latency and energy, respectively. This range of accuracy for analytical models is acceptable for architectural-level studies.

In summary, this paper makes the following contributions: 1) It presents two full-custom implementations for the checkpointed RATs of 4- and 8-way dynamically scheduled superscalar processors in a 130-nm CMOS technology. These two implementations are representative of two commonly assumed checkpointing techniques for the RATs. The implementations differ in the way GCs are organized, allocated, and de-allocated. 2) For all RAT operations, it quantitatively determines the RAT latency and energy as a function of the number of GCs, issue width, and window size. 3) Using architectural-level simulations, it estimates how performance is affected by RAT latency for two RAT implementations taking a state-of-the-art selective GC allocation policy into consideration [3]. 4) It presents analytical models for the RAT latency and energy and compares the model estimations against physical-level simulation results.

The rest of this paper is organized as follows. Section II reviews the RAT's role in modern processors as well as related work. Section III discusses two checkpointed RAT implementations. Section IV presents the analytical models for the RAT latency and energy. Section V presents the results of the physical- and architectural-level evaluations. Additionally, it compares the simulation results against model estimations. Finally, Section VI summarizes our findings.

II. RAT Background

This section provides an overview of register renaming, RAT implementations, and RAT checkpointing. Moreover, this section reviews related work on the RAT implementation and RAT checkpoint reduction.

A. Role of the RAT in Register Renaming

The register renaming logic maps the architectural register names used by instructions into the physical registers implemented in the processor. Register renaming assigns a different physical register for each write to the same architectural register. As a result, this mapping removes false name dependences—write-after-write (WAW) and write-after-read (WAR)—that artificially limit ILP. The number of physical registers is larger than the number of architectural registers. Physical register names are recycled when their values are no longer needed (i.e., all instructions that might consume the values have executed). For each architectural destination register, renaming logic allocates a physical register and records this mapping in the RAT so that the subsequent architectural source registers will correctly reference the physical registers holding their latest value. Conceptually, the RAT is a table indexed by architectural register names, and each RAT entry contains a physical register name. Fig. 1 shows how false data dependences are removed by register renaming for the architectural register R2.

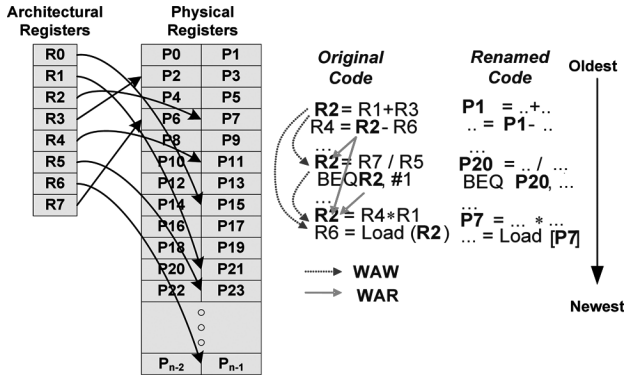

Fig. 1. Example of register renaming.

Renaming a single instruction through the RAT proceeds as follows: 1) reading the physical register names for the source register operands; 2) reading the current mapping of the destination register; this old mapping is saved in the reorder buffer (ROB) to support speculative execution (addressed in Section II-D); and 3) acquiring a physical register name from the pool of free registers and updating the RAT for establishing the new mapping of the destination register.

B. RAT Implementations

Two commonly used RAT implementations are based on static random access memory (SRAM) or content addressable memory (CAM) structures. This paper focuses on the SRAM-based RAT implementation (e.g., used in MIPS R10000 [26]). This implementation is similar in structure to a multi-ported register file and has as many entries as the number of architectural registers. The physical register name (or address) for an architectural register name is read/updated via a direct access to the corresponding RAT entry. The RAT entry width is equal to the physical register address (e.g., 7 bits for 128 physical registers). The CAM-based RAT has as many entries as the number of physical registers; each RAT entry stores the architectural register name assigned to a given physical register in addition to a valid bit indicating whether this RAT entry corresponds to the most recent instance of the architectural register [19]. In this implementation (e.g., used in the Alpha 21264 [13]), a RAT lookup involves an associative

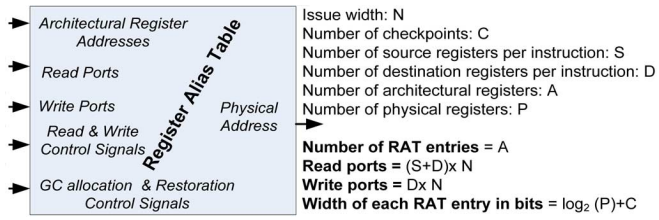

Fig. 2. RAT characteristics.

search on the CAM content using the architectural register address as the key. An investigation of the CAM-based RAT's latency and energy characteristics can be found in [22].

C. RAT Port Requirements

This work assumes a MIPS-like instruction set architecture, where the instructions may have at most two source registers and one destination register. Given this, the SRAM-based RAT needs to support $3 \times N$ reads and N writes per cycle, where N is the number of instructions required to be renamed per cycle. $2 \times N$ read ports are used to rename the two source operands, and another N read ports are needed to read the current mappings of the destination operands for the purpose of recovery using ROB (Section II-D). Finally, the N write ports are used to write new mappings for the destination registers. Fig. 2 shows a high-level block diagram of the RAT including its inputs and outputs.

D. Checkpointing Mechanisms: GCs Versus ROB

Modern processors utilize control-flow speculation to improve performance. When speculation is incorrect, all instructions along the mispeculated path must be squashed, i.e., any changes made by these instructions must be reversed. The fail-safe recovery mechanism, the ROB, allows recovery at any instruction including mispeculated branches. By maintaining a complete log of all changes made to the RAT, the ROB supports recovery at any instruction. Squashing an instruction amounts to reversing any changes it has made to the RAT. This recovery is done by writing back to the RAT the previous physical register name for the instructions' destination registers. To restore the RAT to the state it had at a particular instruction, all subsequent changes must be undone by traversing the ROB log in reverse order. Accordingly, in this recovery mechanism, reversing the effects of each mispeculated instruction requires time proportional to the number of squashed instructions.

Since branch mispeculations are relatively frequent, processors incorporate a number of GCs that are allocated at the decode time. A GC contains a complete snapshot of all relevant processor states, including the RAT. Using GCs, the previous state of all relevant processor components is retrieved "instantaneously," i.e., with a fixed, low latency. Recovery at a specific instruction without a GC can be done either through ROB or by recovering at an earlier GC and then re-executing all other instructions in between.

E. RAT Operations

The RAT operations are read (lookup), write (update), GC allocation, and GC restoration. Section II-A discussed RAT reads and writes. The other two operations are related to the RAT checkpointing function. A GC is taken by copying the main RAT bit into one of the backups (GC allocation). RAT recovery is done by copying one of the RAT backups (GCs) to the main RAT bit (GC restoration). The RAT checkpointing function is further discussed in Section III-B.

F. GCs and Performance

Earlier processors used few GCs (e.g., four) that were allocated at every predicted branch [26]. Few GCs were sufficient given the relatively small instruction window sizes (e.g., 32). However, modern processors use a lot larger windows and, hence, have a lot more unresolved branches. Moreover, modern processors use other forms of speculation such as memory dependence prediction and, thus, may require even more GCs (control-flow mispeculations remain dominant). Previous work showed that 24–48 GCs would be needed to maintain high performance for processors with 128 or more instructions in their windows [1], [13]. Assuming that embedding a large number of GCs into the RAT significantly increases RAT latency, previous work proposed using confidence estimators to allocate GCs selectively [11] and throttling control-flow speculation to achieve higher performance with four or fewer GCs [3]. While previous studies have assumed that increasing the number of GCs degrades performance and energy, they have not quantified this degradation as a function of the number of GCs. Instead, they have relied on IPC performance evaluation. Accordingly, it is not clear whether previous work has sufficiently reduced the number of required GCs and whether the conclusions are valid. This paper complements previous RAT checkpointing work by quantifying the actual performance (execution time) impact, taking into consideration both IPC and latency.

G. Checkpointed RAT: Related Work

Related work falls into two categories: The first category includes work on measuring and modeling SRAM-based RAT energy and latency. Bishop *et al.* present an implementation for a RAT with GCs for single-issue processors in a 350-nm technology and report its worst case delay [5]. De Gloria *et al.* report the latency of a 4-way superscalar RAT with embedded cross-bundle dependence detection logic and a stack of four GCs in a 350-nm technology [9]. Our work complements previous work in that it studies RAT latency and energy as a function of several architectural parameters as opposed to focusing on a particular design.

The second category includes work on reducing the number of RAT GCs or RAT ports while maintaining performance (e.g., [14], [20]). This paper complements these studies by focusing on both IPC and latency. As Section V shows, the tradeoffs are different when actual RAT latency is taken into consideration.

III. Physical-Level Design

This section presents two checkpointed RAT designs that are representative of early and recent proposals, respectively. Additionally, this section discusses our physical-level implementation of the SRAM-based checkpointed RAT.

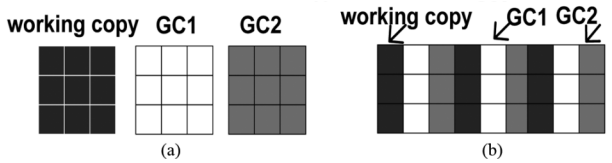

Fig. 3. RAT checkpointing. (a) Concept. (b) Implementation.

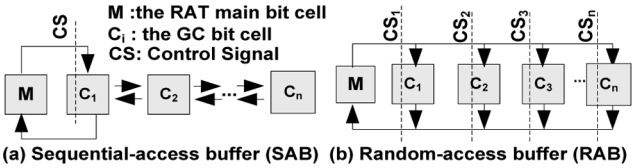

Fig. 4. Checkpointing organizations. (a) SAB. (b) RAB.

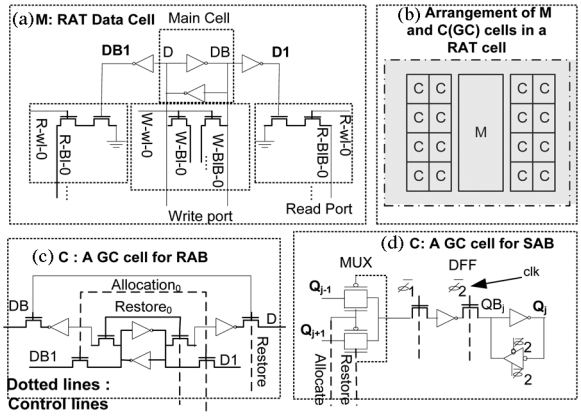

Fig. 5. (a) RAT main cell. (b) Layout of the main RAT bit and GCs. (c) RAB GC. (d) SAB GC.

A. Checkpointed RAT Designs

Fig. 3 illustrates the organization of a checkpointed RAT. Fig. 3(a) shows the conceptual organization where multiple RAT copies exist. Fig. 3(b) shows how, at the physical-level implementation, GCs can be interleaved and embedded into the RAT next to each main bit. Although GCs provide low-latency recovery, they affect RAT latency and energy whether the GCs are embedded into or placed out of the RAT. This increase is primarily due to the load increase that results from connecting GCs to the main RAT cell (discussed in Section III-B).

Two checkpointed RAT designs have been assumed in previous work. These designs differ in the way they implement GC allocation and GC restoration. The first design organizes the GCs in a bidirectional shift register. The second design organizes the GCs in a random access buffer. For clarity, the terms *serial access buffer* (SAB) and *random access buffer* (RAB) will be used to refer to these implementations. SAB requires point-to-point connections, whereas RAB provides maximum GC management flexibility.

Fig. 4(a) and (b) show the organizations of the RAT cells for SAB and RAB implementations, respectively. Every RAT main bit cell (marked as M) has $3 \times N$ read and N write ports; these ports are not shown in Fig. 4. The GC cells are marked as C_i. In SAB, GC allocation is done by shifting the C_i bits to the right, copying the RAT bit value to the adjacent vacant position. In SAB, restoring from a GC may require multiple steps since the appropriate value must be shifted into the RAT main bit. For example, restoring from C_2 requires two left shifts. GC restoration in SAB may take multiple cycles depending on the GC count.

In RAB, the GCs are organized in a random access buffer; hence, GC allocation latency and GC restoration latency are nearly the same for all GCs. If the number of GCs becomes large, recovery using RAB is much faster than recovery using SAB since no shifting is needed. Both designs require external controllers to track the number of available GCs and to coordinate GC operations. For SAB, the controller keeps track of the number of GCs that are currently in the shift registers. For RAB, the controller tracks the status of each GC. SAB is more compact than RAB since it requires fewer external signals: RAB requires one read/write control signal per individual GC, while SAB only requires a global shift left/right signal.

B. Physical-Level Implementation

A non-checkpointed RAT is simply a multi-ported register file. A checkpointed RAT, however, is a multi-ported register file with embedded GCs. The checkpointed RAT circuit consists of the precharge and equalization circuitry, sense amplifiers, write drivers, control circuitry, and decoders, along with an array of RAT cells connected by bitlines and wordlines. Fig. 5(a) shows the main RAT cell comprising two back-to-back inverters and several read and write ports. Fig. 5(b) shows a complete RAT cell with 16 GCs. Each GC requires an SRAM cell. The SAB and RAB GC cells are shown in Fig. 5(c) and (d), respectively.

The multi-ported RAT cell uses one wordline and two bitlines per each write or read port. Multiple read operations may access the same RAT entry. In RAB, all GCs are connected via pass gate/gates to the main cell, whereas in SAB, only one GC is connected to the main cell directly. Hence, the main cell must be capable of driving a capacitance proportional to the number of ports and connected GCs. To protect the data stored in the main cell during multiple accesses, decoupling buffers isolate the RAT main cell and the read ports [28]. Since the GCs are connected to the main cell as the read ports do, the buffers also isolate the GCs. Due to these isolating buffers, separate write bitlines are required. Differential read and write operations are used because they offer better power, delay, and robust noise margins. To reduce power, the following techniques are employed: 1) pulse operation for the wordlines, for the periphery circuits, and for the sense amplifiers; 2) multi-stage static CMOS decoding; and 3) current-mode read and write operations.

Fig. 5(c) and (d) show RAB and SAB GC cells, respectively. In SAB, GCs are organized as bi-directional shift registers with connections between adjacent cells; only one of the GCs is connected to the main RAT bit through pass gates. In SAB, a GC cell consists of a register and a multiplexer controlling the shift direction. The SAB's shift register uses two non-overlapping clocks. The SAB requires two external control signals irrespective of the number of GCs. In RAB, each GC cell is connected to the RAT main cell through separate pass transistors. Two pairs of pass transistors are used to copy the value from the RAT main cell to the GC and vise versa. Each RAB GC cell needs two external control signals.

IV. Analytical Models

Analytical models help computer architects estimate the latency and energy of architectural alternatives during architectural-level exploration. To the best of our knowledge, no analytical model exists for the checkpointed RATs. Hence, previous work relied on analytical models for register files. However, register file models do not consider the impact of GCs on latency and energy. This section presents analytical models for the worst case latency and energy of the SRAM-based checkpointed RAT implementation. Computer architects can use the models to estimate the latency and energy of RAT organizations in lieu of a physical-level implementation. Architectural-level power-performance simulators such as Wattch [6] and Simplepower [25] can incorporate these models as well.

This section is organized as follows: Section IV-A discusses the model-developing methodology and the model's input parameters. Sections IV-B and IV-C present the latency and energy models, respectively.

A. Methodology

To model latency and energy, we decompose the design into equivalent RC circuits. Our analysis methodology is similar to that of CACTI [24]. RC circuit analysis requires estimations of the gate capacitance (C_{gate}), the diffusion capacitance ($C_{\text{diffusion}}$), the overlap capacitance (C_{overlap}), the equivalent on resistance for nMOS transistors ($R_{\text{eq-nmos}}$), and the equivalent on resistance for pMOS transistors ($R_{\text{eq-pmos}}$). Information such as transistor sizes and interconnect lengths, required for capacitance and resistance estimations, is extracted from the full-custom layout.

For the base RAT, the transistors can be sized to achieve different speed/energy tradeoffs. In our implementation, we sized the transistors for the base 4- and 8-way RATs (addressed in Section V-A) such that the RAT read delay is less than the upper bound estimated by CACTI 4.2 [24]. In our models, the geometry and transistor sizes of the base 4- and 8-way RAT cells (cell port requirements are addressed in Section II-B) are extracted from our full-custom layout.

The models do not account for external loads since these loads are independent of the RAT implementation. Extending the models to predict latency and energy for other technologies is feasible, but it is beyond the scope of this work.

Table I lists the model's input parameters. The parameters fall under two broad classes: physical-level organizational parameters and technology-specific parameters. The physical-level organizational parameters are as follows: the number of entries (NoE), the width of each entry (WoE), the number of read ports (NoRP), the number of write ports (NoWP), and the number of GCs (NoGCs). In our analytical models, the relations among the physical-level organizational parameters and the architectural-level parameters are given by (1)–(4)

$$\text{WoE} = \log_2(\text{Number of physical registers}) \tag{1}$$

$$\text{NoE} = \text{Number of architectural registers} \tag{2}$$

$$\text{NoRP} = (\text{Number of source operands per instruction} + \text{Number of destination operands per instruction}) \times \text{issue width} \tag{3}$$

$$\text{NoWP} = (\text{Number of destination operands per instruction}) \times \text{issue width}. \tag{4}$$

TABLE I
Analytical Model Input Parameters

Physical Level Organizational Parameters	
NoE	*Number of entries*
WoE	*Width of each entry*
NoRP	*Number of read ports*
NoWP	*Number of write ports*
NoGC	*Number of global checkpoints (GCs)*
Technology Parameters	
C_w, R_w	*Per unit length capacitance and sheet resistance of metal layers*
Other parameters as in [24], such as capacitance and resistance per unit width: C_{gate}, $C_{ndiffarea}$, $C_{ndiffside}$, $C_{ndiffgate}$, $C_{pdiffarea}$, $C_{pdiffside}$, $C_{pdiffgate}$, $R_{eq.nmos}$, $R_{eq.pmos}$, V_{dd}.	

B. Delay Model

This section presents the analytical worst case delay model for the checkpointed SRAM-based RAT. For clarity, labels are assigned to the elements in the critical path. These labels are used as subscripts to specify the corresponding resistance and capacitance. The type of gates (e.g., inverter) and the type of capacitors (e.g., drain d, source s, and gate g) are also denoted in the subscripts.

Our RAT design is based on a synchronous multi-ported SRAM, i.e., a clock starts off the accesses. The SRAM consists of six main subblocks: a decoder to decode the input address, a memory core of bit cells arranged in rows and columns, a read logic comprising a read-column differential sense amplifier and output data drivers, a write logic to drive data onto the bitlines, and read and write control logic to control the write and read logic, respectively. The RAT read and write delays are given by (5) and (6), respectively. The following sections present the per component delay analyses.

$$T_{\text{RAT\_read\_access}} = T_{\text{decoder}} + T_{\text{wordline}} + T_{\text{bitline}} + T_{\text{sense\_amplifier}} \tag{5}$$

$$T_{\text{RAT\_write\_access}} = T_{\text{decoder}} + T_{\text{wordline}} + T_{\text{bitline}} + T_{\text{write\_driver}}. \tag{6}$$

1) Component delay—Decoder: A separate decoder is needed per read port and per write port. Fig. 6 shows the decoder's high-level architecture, critical path, and equivalent RC circuit. To estimate the RC delay, transistor sizes and interconnect lengths along the critical path are required. These parameters are a function of WoE, NoRP, and NoWP. Increasing the NoRP, NoWP, and NoGCs increases the RAT cell geometry and interconnect lengths between subsequent rows and columns.

The decoder has a hierarchical architecture. In the predecode stage, each 3-to-8 decoder generates a 1-of-8 code for every

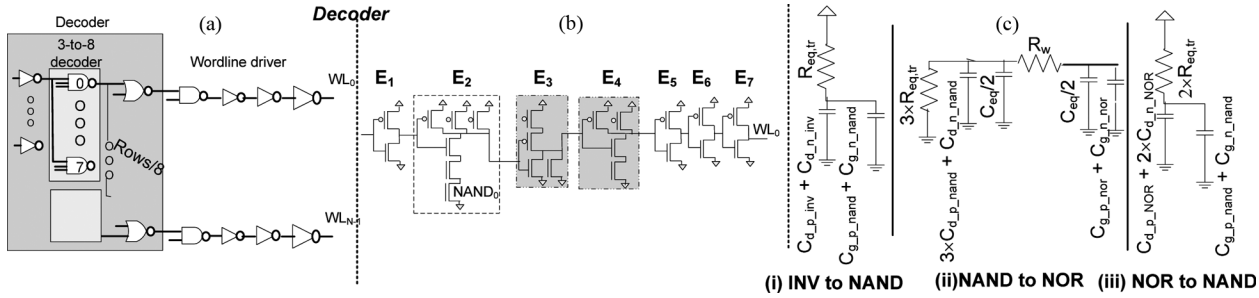

Fig. 6. (a) Decoder and wordline driver. (b) Equivalent critical path. (c) Equivalent RC circuit.

three address bits. If the number of address bits is not divisible by three, a 2-to-4 decoder or an inverter is used. Each x-to-2^x decoder consists of 2^x NAND gates and x inverters to complement the address inputs. In the second stage, the predecoder outputs feed NOR gates. The decoder delay is the interval between the moment the address input passes the INV(E1)'s threshold voltage and the moment the NOR(E3)'s output reaches the threshold voltage of the following NAND (E4). Equations (7)–(12) report the number of address bits (N_{addr}), the number of 3-to-8 decoders $(N_{3\,\text{to}\,8})$, the number of NOR gates (N_{nor}), and the NOR gate's fan-in $(N_{\text{nor-input}})$ as a function of the model's input parameters. Equations (9), $N_{2\,\text{to}\,4}$, and (10), N_{inv}, show if an additional 2-to-4 decoder or an inverter is required when N_{addr} is not divisible by three. Equation (13), $N_{\text{nor-a-nand}}$, calculates the number of NOR gates driven by each NAND gate. Given by (14), the length of the wire between two NOR gates fed by a specific NAND gate of the predecode stage is a function of the RAT cell's height and NoGC. The corresponding resistance and capacitance are calculated by (15). Equations (17)–(19) calculate time constants (τ) for the RC circuit shown in Fig. 6(c)

$$N_{\text{addr}} = \lceil \log_2 (\text{NoE}) \rceil \tag{7}$$

$$N_{3\,\text{to}\,8} = \lfloor 1/3 \times (N_{\text{addr}}) \rfloor \tag{8}$$

$$N_{2\,\text{to}\,4} = \lfloor 1/2 \times [(N_{\text{addr}}) - 3 \times (N_{3\,\text{to}\,8})] \rfloor \tag{9}$$

$$N_{\text{inv}} = (N_{\text{addr}}) - 3 \times (N_{3\,\text{to}\,8}) - 2(N_{2\,\text{to}\,4}) \tag{10}$$

$$N_{\text{nor}} = NoE \tag{11}$$

$$N_{\text{nor-inputs}} = N_{3\,\text{to}\,8} + N_{2\,\text{to}\,4} + N_{\text{inv}} \tag{12}$$

$$N_{\text{nor-a-nand}} = \frac{NoE}{8}, \qquad \text{if } N_{\text{addr}} \text{ is divisible by 3.} \tag{13}$$

$$L_{\text{cw}}\,(\text{in}\,\mu\text{m}) = \text{wire length between two NOR gates fed by the same NAND gate of the predecoder}$$

$$L_{\text{cw}} \approx \left(\text{Height of multi-ported cell}_{(4-\text{way or } 8-\text{way})} \times \left(1 + \sum_{i=0}^{i=\text{NoGC}} 0.01^i \right) \right) \times N_{\text{nor-a-nand}} \tag{14}$$

$$R_{\text{cw}}\,(\text{in ohms}) = R_{\text{Ohm}/\square} \times (L_{\text{wire}}/W_{\text{wire}})$$

$$C_{\text{cw}}\,(\text{in farads}) = C_{(\text{farad}/\mu\text{m})} \times L_{\text{wire}}\,(\text{in}\,\mu\text{m}) \tag{15}$$

$$\tau_{\text{INV-to-NAND}} = R_{\text{p}} \times (C_{d\_p\_\text{inv\_INV}} + C_{d\_n\_\text{INV}} + C_{g\_p\_\text{NAND}} + C_{g\_n\_\text{NAND}}) \tag{16}$$

$$C_{\text{eq}} = (C_{g\_p\_\text{NOR}} + C_{g\_n\_\text{NOR}}) \times N_{\text{nand-to-nor}} + L_{\text{cw}} \times C_{\text{metal}/\mu\text{m}} \tag{17}$$

$$\tau_{\text{NAND-to-NOR}} = 3\times R_n \times (C_{\text{eq}}/2 + 3 \times C_{d\_p\_\text{NAND}} + C_{d\_n\_\text{NAND}}) + (3 \times R_n + R_{\text{cw}}) \times C_{\text{eq}}/2 \tag{18}$$

$$\tau_{\text{NOR-to-NAND}} = 2\times R_p \times (C_{d\_p\_\text{NOR}} + 2 \times C_{d\_n\_\text{NOR}} + C_{g\_p\_\text{NAND}} + C_{g\_n\_\text{NAND}}). \tag{19}$$

2) Component delay—Wordline driver: Fig. 6 shows the critical path along the wordline driver and the inverter chain following it. The equivalent RC circuits [(20)-(22)] for the critical path is shown in Fig. 6(c). The NAND (E4) inputs are the decoder output and the *operation select* input (read, write, or "no operation"). The worst case delay occurs when one of the NAND inputs turns off and the last column's pass transistor turns on. Each wordline driver consists of a NAND gate followed by an INV or an INV chain, depending on the wordline capacitance. Wordline capacitance (23) depends on the number of pass transistors along it $2\times NoE$. Furthermore, the wordline capacitance depends on L_{cw}, the interconnect length between the wordline driver and the SRAM's last columns. L_{cw} is a function of NoGCs, the SAB cell's width, and the multi-ported SRAM cell's width. L_{cw}, given by (24), is used to estimate the equivalent resistance R_{eq} and capacitance C_{eq} required in (25).

$$\tau_{\text{NAND-to-INV}} = 2\times R_n \times (2\times C_{d\_p\_\text{NAND}} + C_{d\_n\_\text{NAND}} + C_{g\_p\_\text{INV}} + C_{g\_n\_\text{INV}}) \tag{20}$$

$$\tau_{\text{INV-to-INV}} = R_p \times (C_{d\_p\_\text{INV}} + C_{d\_n\_\text{INV}} + C_{g\_p\_\text{INV}} + C_{g\_n\_\text{INV}}) \tag{21}$$

$$\tau_{\text{INV-to-INV}} = R_n \times (C_{d\_p\_\text{INV}} + C_{d\_n\_\text{INV}} + C_{g\_p\_\text{INV}} + C_{g\_n\_\text{INV}}) \tag{22}$$

$$C_{\text{eq}} = (2\times C_{\text{gate\_pass}}) \times WoE + L_{\text{cw}} \times C_{\text{metal}/\mu\text{m}} \tag{23}$$

$$L_{\text{cw}} = \Big(\text{width of multi-ported cell}_{(4-\text{way or } 8-\text{way})} + \text{width of a SAB GC cell} \times \text{NoGCs}\Big) \times \text{WoE} \tag{24}$$

$$\tau_{\text{INV-to-Last\_Columns}} = R_p \times (C_{\text{eq}}/2 + C_{d\_p\_\text{INV}} + C_{d\_n\_\text{INV}}) + (R_{\text{eq,tr}} + R_{\text{wire}}) \times C_{\text{eq}}/2. \tag{25}$$

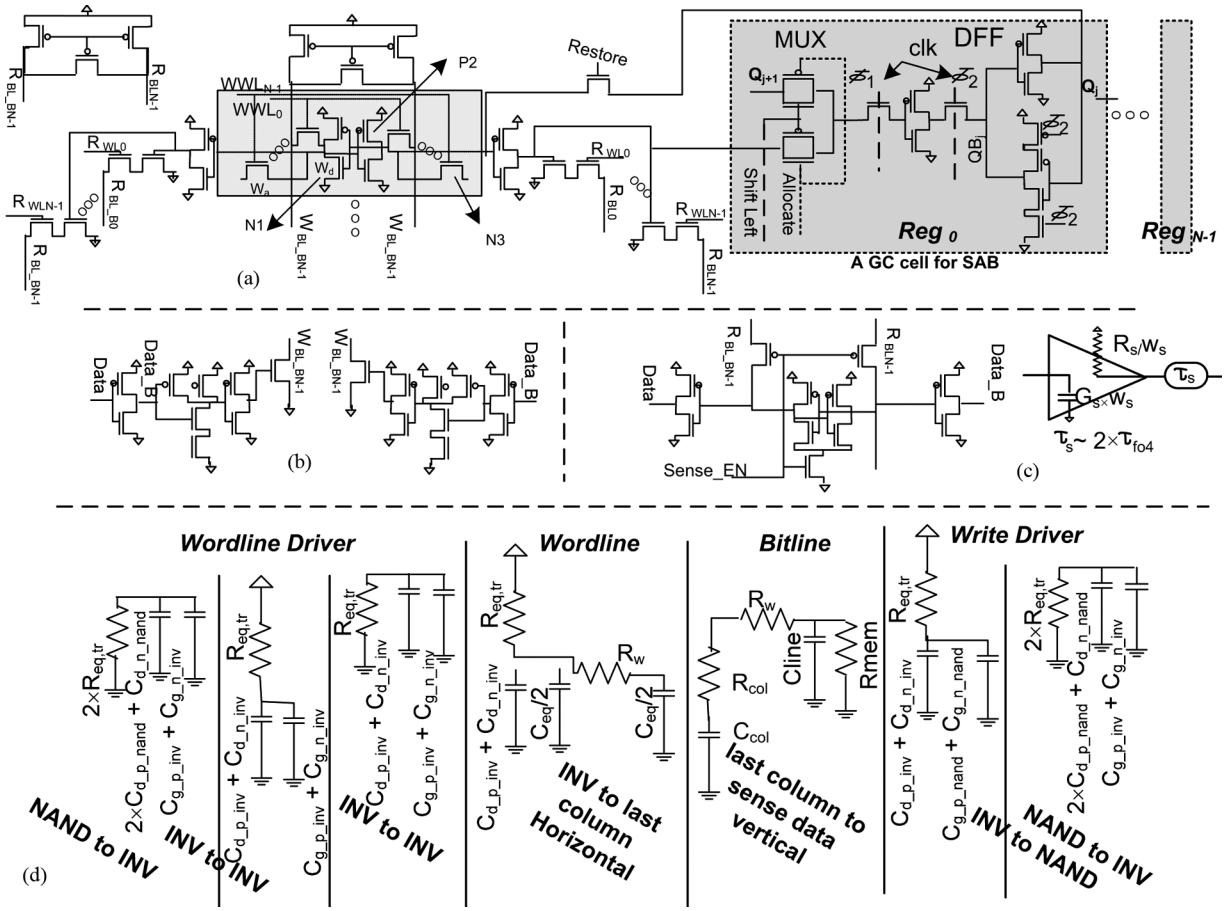

Fig. 7. Building blocks: (a) Multi-ported SRAM cell and its connection with a SAB GC cell. (b) Write driver. (c) Sense amplifier. (d) RC equivalent circuits.

3) Component delay—Bitline delay: This section discusses the components contributing to the bitline delay for read and write operations. The reading from or writing to a row is preceded by precharging all bitlines—bitlines (BLs) and bitline bars (BLBs)—to $V_{\text{precharge}}$ and the selection of a row by the decoder. The bitline precharge time is designed to be hidden under the address decoding time to achieve shorter read/write access time. As shown in Fig. 7(b), a wordline and a set of BLs/BLBs are selected to drive the contents of memory cell(s) to the sense amplifier for a read operation or to the write driver(s) for a write operation. During a read, when the wordline goes high, one of the pull-down transistors of the back-to-back inverters will begin to conduct, discharging BL or BLB. Column isolation pMOS transistors are turned on to allow the voltage difference between BL and BLB to develop to the sensing voltage (V_{sense}), helping the amplifiers to quickly sense the data. The BL delay of the read operation is the time interval between the moment the wordline goes high and the moment one of the bitlines reaches the voltage V_{sense} below its maximum value $V_{\text{precharge}}$.

The delay of the latch-based sense amplifier, shown in Fig. 7(c), consists of the latch and buffer delays. The latch delay depends on the voltage gain and the BL swing's speed. The latch's amplification delay is proportional to the logarithm of the required gain and the load on the amplifier outputs [4]. For a gain of about 20 with only the self-loading of the sense amplifier, [4] reports an amplification delay of about two fan-out-of-4 (FO4) inverter delays. We use the same estimation for the sense amplifier delay in our models.

Write drivers pull down the precharged BL/BLB to zero. As shown in Fig. 7(b), two nMOS transistors connect the write circuits to the write BL/BLB during write cycles. The write driver's critical path consists of an INV, a NAND, and another INV. The final inverter feeds the gate of an nMOS isolator transistor. The worst case BL delay for a write operation is the time interval between the moment that wordline goes high and the moment the cell content is inverted (V_{bit} should be a very low voltage near zero). Fig. 7(d) shows the RC equivalent circuit along the critical path for read and write operations (26)–(35).

4) Operation delay: The switching delay from input to output is referred to as propagation delay that is the time required for the output to reach 50% of its final value V_{dd} when the input changes. The output follows an exponential trend $(V_{\text{out}}(t) = V_{\text{dd}} \times \mathrm{e}^{-(T/\tau)})$, and the time it takes for the output to reach $V_{\text{dd}}/2$ is $\ln(2) \times \tau$. The read and write delays are given by (36) and (37), where the time constants $\tau_{\#}$ correspond to the equations with the same numerical subscript

$$\begin{aligned}\text{Delay}_{\text{Read}} = \ln(2) \times (&\tau_{17} + \tau_{18} + \tau_{19} + \tau_{20} + \tau_{21} \\ &+ \tau_{22} + \tau_{25} + \tau_{31} + \tau_{32}) \quad (36)\\ \text{Delay}_{\text{Write}} = \ln(2) \times (&\tau_{17} + \tau_{18} + \tau_{19} + \tau_{20} \\ &+ \tau_{21} + \tau_{22} + \tau_{25} + \tau_{31}). \quad (37)\end{aligned}$$

$$R_{\text{mem}} = R_{n,\text{Wd}} + R_{n,\text{access}} \tag{26}$$

$$C_{\text{bitline}} = 2 \times C_{d\_p\,(\text{Wbitpre})} + \text{NoE} \times (1/2 \times C_{d\_n\_\text{passWa}}) + \left(\text{Height of multi-ported cell}_{(\text{4 way or 8 way})} \left(1 + \sum_{i=0}^{i=\text{NoGC}} 0.01^i \right) \times C_{\text{metal}/\mu\text{m}} \right) \tag{27}$$

$$R_{\text{col\_read}} = R_{\text{eq},p} \qquad R_{\text{col\_write}} = R_{\text{eq},n} \tag{28}$$

$$C_{\text{col\_read}} = C_{d\_p\_\text{access\_SA}} \tag{29}$$

$$C_{\text{col\_write}} = C_{d\_n\_\text{write\_driver}} \tag{30}$$

$$\tau_{\text{Last\_Columns to Sense amplifier}} \left(\textbf{or } \tau_{\text{Last\_Columns\_discharge (for write)}} \right) = [R_{\text{mem}} \times C_{\text{bitline\_write}} + (R_{\text{mem}} + R_{\text{bitline\_write}} + R_{\text{col}}) \times C_{\text{col}}] \times \ln(V_{\text{precharge}}/(V_{\text{precharge}} - V_{\text{bit}})) \tag{31}$$

(For more details, refer to the RC tree analysis [24].)

$$\tau_{\text{Senseamplifier}} = 2 \times \tau_{\text{fo4}} \tag{32}$$

$$\tau_{\text{INV-to-NAND}} = R_{\text{eq,tr}}(C_{d\_p\_\text{inv\_INV}} + C_{d\_n\_\text{INV}} + C_{g\_p\_\text{NAND}} + C_{g\_n\_\text{NAND}}) \tag{33}$$

$$\tau_{\text{NAND-to-INV}} = 2 \times R_{\text{eq,tr}} \times (2 \times C_{d\_p\_\text{NAND}} + C_{d\_n\_\text{NAND}} + C_{g\_p\_\text{INV}} + C_{g\_n\_\text{INV}}) \tag{34}$$

$$\tau_{\text{INV-to-nmos}} = R_{\text{eq,tr}} \times (C_{d\_p\_\text{INV}} + C_{d\_n\_\text{INV}} + C_{g\_n\_\text{INV}}) \tag{35}$$

C. Energy Model

The four sources of power dissipation are as follows: First is the dynamic switching power due to the charging and discharging of the circuit capacitances. Second is the leakage power from the reverse-biased diodes and subthreshold conduction. Third is the short-circuit current power due to the finite signal rise/fall times. Fourth is the static biasing power found in some types of logic styles (e.g., pseudo-nMOS). For the given technology, circuit simulations suggest that the first two are the principal sources of energy consumption.

1) Dynamic power: Dynamic power is the result of the gate output transitions. Output transitions cause the capacitive load driven by the gate to be charged or discharged. To calculate approximately the capacitive load that is required for the energy per operation estimation, the gate (e.g., NAND) and interconnect capacitances in the signal path are added up. The energy dissipated per transition (0-to-1 or 1-to-0) is given by (38), where C_L is the load capacitance, V_{dd} is the supply voltage, and ΔV is the output's voltage swing

$$E_{\text{dynamic}} = 0.5 \times C_L \times V_{\text{dd}} \times \Delta V. \tag{38}$$

The analytical energy models use the capacitance estimations of the RC delay analysis. For instance, the decoder energy is calculated by adding up the gate and interconnect capacitances along the critical path [12].

2) Leakage power: To calculate the leakage current in a MOSFET, like [17], we used the model proposed by Zhang *et al.* [27] given by (39)

$$I_{\text{lkg}} = \mu_0 \times C_{\text{ox}} \times \frac{w}{l} \times \mathrm{e}^{b(V_{\text{dd}} - V_{\text{dd0}})} \times v_t^2 \times (1 - \mathrm{e}^{-V_{\text{dd}}/v_t}) \times \mathrm{e}^{\frac{-v_{\text{th}} - v_{\text{off}}}{n v_t}}. \tag{39}$$

As shown in [17], for a given threshold voltage (V_{th}) and temperature (T), all terms except the width (W) are constant for all the transistors in a given fabrication technology. Hence, (39) can be reduced to (40) where I_l is the leakage current of a unit-width transistor at a given T and V_{th}

$$I_{\text{lkg}} = W \times I_l(T, V_{\text{th}}). \tag{40}$$

When stacks of transistors (transistors connected in series drain to source) exist in a design, leakage current reduces significantly [17]. The leakage characteristics of nMOS and pMOS transistors can be different from each other in a given fabrication technology. We assume that the $I_{lP}(T, V_{\text{th}})$ and $I_{lN}(T, V_{\text{th}})$ for the given technology are available to the models. For leakage power estimation, we follow the methodology suggested in [17]. As an example, we discuss how the memory core's leakage current for the idle (or precharge) state is calculated. This memory core is assumed to comprise single-port SRAM cells similar in structure to the cell shown in Fig. 7(a). During the idle time, all wordlines are inactive, and BLs and BLBs are precharged to V_{dd}. We identify the off transistors during idle time and add up their leakage current as calculated in (41) and (42)

$$I_{\text{memCellIdle}} = I_{\text{leakage}}(\text{N1}) + I_{\text{leakage}}(\text{N3}) + I_{\text{leakage}}(\text{p2}) = (W_{\text{N1}} + W_{\text{N3}}) \times I_{lN} + W_{\text{P2}} \times I_{lP} \tag{41}$$

$$I_{\text{memCoreIdle}} = N_{\text{rows}} \times N_{\text{cols}} \times [(W_{\text{N1}} + W_{\text{N3}}) \times I_{lN} + W_{\text{P2}} \times I_{lP}]. \tag{42}$$

The same methodology is used for other components. Multiplying I_{leakage} by V_{dd} gives us the leakage power estimation.

V. Evaluation

This section discusses physical- and architectural-level evaluation results. Section V-A presents the physical-level results and, then, Section V-B builds upon these results, taking into consideration actual program behavior. Section V-D compares the model estimations against the circuit measurements.

A. Physical-Level Evaluation

1) Design assumptions and methodology: The base 4-way RAT has 12 read ports and 4 write ports, and the base 8-way RAT has 24 read ports and 8 write ports. These base RAT configurations include no GCs. We also assume that 64 architectural registers are available, typical of modern load/store architectures. Modern processors have about 128 physical registers.

However, future designs may include more physical registers to support larger scheduling windows and/or multiple threads. Hence, we vary the number of physical registers from 128 to 512. We focus on RAT designs with 0, 4, 8, or 16 GCs since previous work shows that, with GC prediction and selective GC allocation, 16 GCs are sufficient to achieve performance close to the performance achievable with an infinite GCs [1].

We developed full-custom layouts for both designs using the Cadence tool set in a commercial 130-nm fabrication technology with a 1.3-V supply voltage. For circuit simulations, we used Spectre, a vendor-recommended simulator. In this section, the worst case delay and energy values are reported.

Circuit designs can be tailored to achieve different latency and energy tradeoffs. In an actual commercial design, a target latency and/or energy is decided and used as a specification for tuning the individual components. In lieu of an actual specification for the target operating frequency, we used CACTI 4.2 [17] to obtain a reasonable upper bound on delay. CACTI is an integrated cache-access-time cycle-time area and power modeling tool that is commonly utilized by computer architects. Using CACTI, for the base 4-way RAT, we determined an upper bound on the critical path delay by estimating the delay of a 64-bit 64-entry SRAM with 12 read ports and 4 write ports. Similarly, we estimated the delay of a 64-bit 64-entry SRAM with 24 read ports and 8 write ports to determine an upper bound on the critical path delay for the base 8-way RAT. These upper bounds are reasonable approximations since the base non-checkpointed RAT designs are identical to register files. However, the data width of the register files modeled by CACTI is larger. To further corroborate these RAT delay estimations, we also considered the clock periods of processors built in 130-nm fabrication technology (e.g., 800-MHz SR71010B MIPS) given that a single-cycle register renaming has been assumed.

2) RAT delay: Fig. 8 shows RAT read and write delays for RAB and SAB as a function of the issue width, the window size, and the number of GCs. Fig. 8(a) and (b) show latencies of the 4- and 8-way superscalar RATs, respectively. As expected, RAT delay increases with increasing the window size, the issue width, and the number of GCs for both implementations. Adding more GCs increases delay, more so for RAB. The read and write operations of RAB are slower than those of SAB because, in RAB, the main RAT bit is connected via pass transistors to all GCs. The increase due to additional GCs is more pronounced for the 4-way superscalar RAT. The delay of the 8-way RAT, however, is dominated by the load of the extra read ports and, hence, GCs have less impact on overall latency.

To comment on the RAT delay variation as a function of the number of GCs, we focus, for instance, on SAB-512 and RAB-512. We compare the read and write delays for RATs with 4, 8, or 16 GCs with those of the non-checkpointed RAT. SAB RAT reads are 0.5%, 1.6%, and 5% slower depending on the number of GCs; SAB RAT writes are 1.8%, 4.8%, and 13.8% slower. RAB RAT reads are 0.8%, 2.7%, and 9.3% slower, whereas RAB RAT writes are 2.7%, 8%, and 25.3% slower, respectively. A SAB RAT with 16 GCs is faster than a RAB RAT with four GCs. These results suggest that a RAB RAT must improve IPC considerably over a SAB RAT to improve real performance.

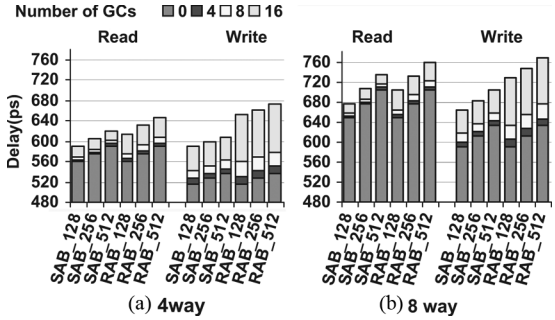

Fig. 8. RAT read delay and RAT write delay as a function of the number of GCs with window sizes of 128, 256, and 512. (a) 4-way. (b) 8-way.

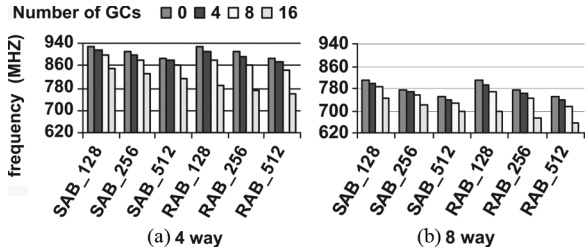

Fig. 9. Clock frequency as a function of the number of GCs for window sizes: 128, 256, and 512. (a) 4-way. (b) 8-way.

3) Operating frequency: In the simplest possible implementation, the RAT operates at the same frequency as the processor. In this implementation, instructions read from the RAT and then write new mappings back to it within a single cycle. Alternatively, the RAT can be pipelined to achieve a higher operating frequency at the expense of increased complexity and hardware cost. This paper focuses on single-cycle register renaming as this represents a reasonable and common design point. To further support the validity of this assumption, we present RAT latency in terms of the FO4 inverter delay. For instance, consider the 4-way RAT with 12 read and 4 write ports and no GCs. For the 130-nm fabrication technology that we use, the FO4 delay, measured by simulations, is about 40 ps. For the SAB RAT, 12 reads and 4 writes take 1077ps (27 FO4) given a non-pipelined RAT. The decoding of a RAT write can be overlapped with its preceding RAT data read. In this case, overall RAT access delay reduces to 20 FO4. This delay is comparable to the clock period of the processors used as a specification for our design (e.g., 800-MHz SR71010B MIPS).

Fig. 9 shows the maximum operating frequency for the 4- and 8-way RATs given that the RAT latency is the same as the clock period and the latching overheads are ignored. For a given window size and issue width, RAB's performance deteriorates more rapidly than SAB's performance as the GC count is increased.

B. Architectural-Level Evaluation

This section discusses the effects of architectural-level GC management policies on performance and energy.

1) Methodology: We used Simplescalar v3.0 [7] to simulate the processors detailed in Table II. We study 4- and 8-way dynamically-scheduled superscalar processors with 128-, 256-, or 512-entry window sizes. We compiled the SPEC CPU

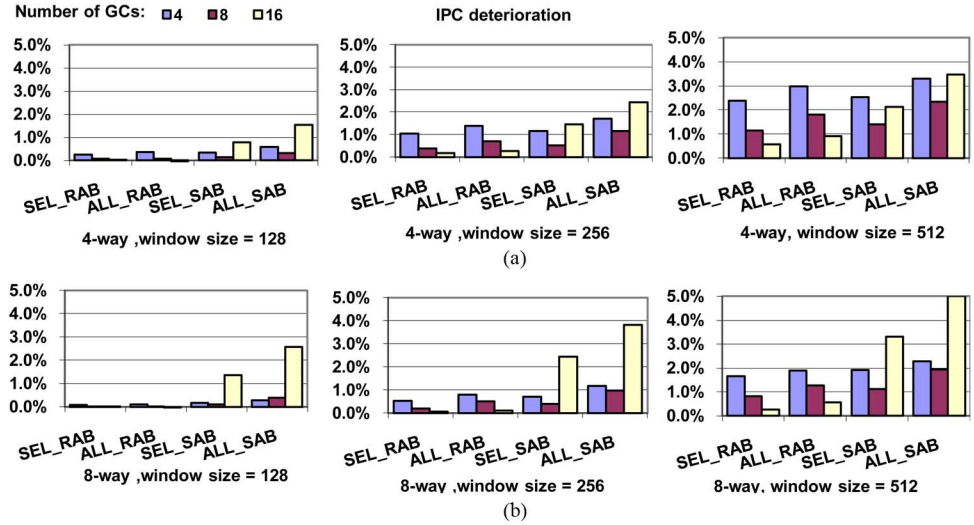

Fig. 10. IPC deterioration compared to a design with infinite GCs: (a) 4-way and (b) 8-way (all designs operate at the same frequency).

TABLE II
BASE PROCESSOR CONFIGURATION

Branch Predictor	*Fetch Unit*
8K-entry GShare and 8K-entry bi-modal 16K selector, 2 branches per cycle	**Up to 4 or 8 instr. per cycle, 64-entry Fetch Buffer, Non-blocking I-Cache**
Issue/Decode/Commit	*Scheduler*
any 4 or 8 instr./cycle	**128, 256 or 512-entry/half size LSQ**
FU Latencies	*Main Memory*
Default simplescalar values	**Infinite, 200 cycles**
L1D/L1I Geometry	*UL2 Geometry*
64KBytes, 4-way set-associative with 64-byte blocks	**1Mbyte, 8-way set-associative with 64-byte blocks**
L1D/L1I/L2 Latencies	*Cache Replacement*
3/3/16 cycles	**LRU**
Fetch/Decode/Commit Latencies	
4 cycles + cache latency for fetch	

2000 benchmarks for the Alpha 21264 architecture using HP's compilers and for the Digital Unix V4.0F utilizing the SPEC-suggested default flags for peak optimization. All benchmarks were run using a reference input data set. The following SPEC CPU 2000 benchmarks are used in the experiments: ammp, applu, apsi, art, bzip2, crafty, eon, equake, facerec, fma3d, galgel, gap, gcc, gzip, lucas, mcf, mesa, mgrid, parser, swim, twolf, vortex, vpr, and wupwise. We were not able to run the rest of the SPEC benchmarks either because we could not compile them or because they exhausted the available memory space during simulation. To achieve reasonable simulation times, samples were taken for one billion committed instructions per benchmark. Prior to collecting measurements, two billion committed instructions were skipped. Unless otherwise noted, we report the average over all benchmarks.

We limit our attention to two representative GC management policies. The first one, SEL, selectively allocates GCs only to low-confidence branches [1], [13]. Low-confidence branches are identified using a confidence estimator comprising a 1K-entry table of 4-bit resetting counters [11]. The second one, ALL, allocates GCs to all branches. Both use in-order GC allocation and deallocation. Unless otherwise stated, all performance results are normalized over an idealized RAT with infinite GCs that are allocated at all branches and can be accessed in a single cycle. Ignoring secondary effects, this RAT represents an upper bound on performance for the two GC allocation policies. We pessimistically assume that SAB can only shift by one bit per cycle. This assumption is valid for a simple SAB design. However, multiple shifts per cycle may be possible and, hence, execution time could be better at the cost of increased complexity.

2) IPC performance and execution time: Fig. 10(a) shows the average IPC deterioration for the 4-way processor given that all RAT implementations operate at the same frequency. These measurements ignore the actual implementation delay and compare just the IPC. For clarity, Fig. 10 excludes the results for the non-checkpointed RAT. Performance deterioration with a non-checkpointed RAT is on the average 5%, 9.4%, and 15% for the 128-, 256-, and 512-entry window sizes, respectively.

Irrespective of the GC allocation policy (SEL or ALL), with RAB, performance improves as the number of GCs increases because recovery latency for all GCs is nearly the same. As previous work reported, performance is better with SEL than with ALL when few GCs are used because SEL allocates GCs judiciously to branches that more likely trigger recoveries. With ALL, however, GCs are allocated to all branches indiscriminately and, hence, GCs get exhausted more often.

The same observation applies to SAB, where SEL still performs better than ALL with four or eight GCs. However, with SAB, performance does not always increase with the number of GCs. Specifically, performance degrades when GC count increases to 16 from 8, because the number of cycles required to recover from a specific GC is different depending on the location of the GC in the SAB's shift register. As the number of GCs increases, so does the expected number of shifts to retrieve that specific GC and, hence, the cycles needed to restore from

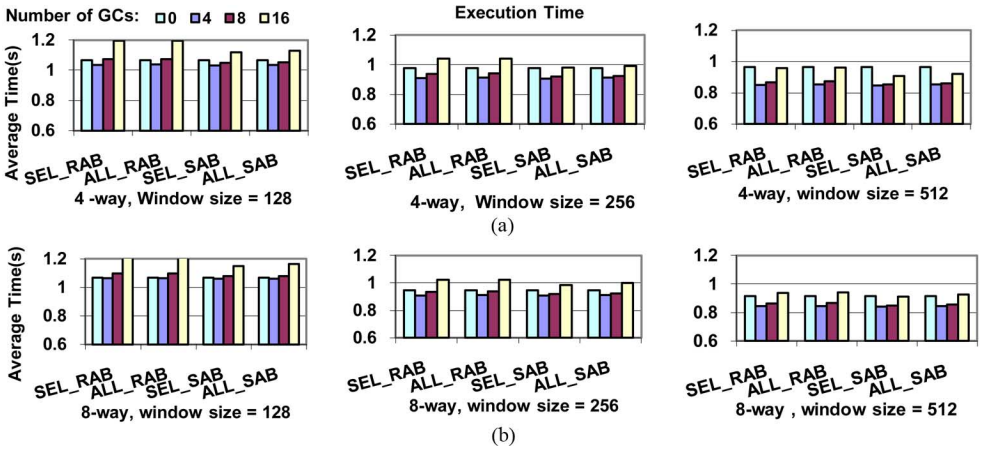

Fig. 11. Execution time for (a) 4- and (b) 8-way superscalar processors.

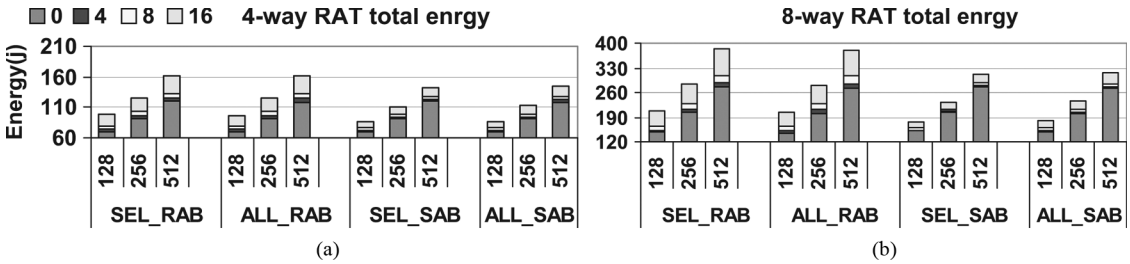

Fig. 12. Total energy of RAB and SAB for both SEL and ALL methods. (a) 4-way RAT. (b) 8-way RAT.

it. SEL requires more cycles for recovery as the number of GCs is increased to 16, because when SEL cannot find a GC associated with a specific mispredicted branch, it restores at a later GC and then walks back using ROB. The results show that RAB outperforms SAB if their implementation latencies are ignored.

Fig. 10(b) shows IPC deterioration for the 8-way superscalar processor. Wider (8-way) processors are capable of filling the window faster than 4-way processors. Hence, they are more likely to speculate further down the instruction path. However, the deeper the speculation, the less likely it will succeed. Hence, additional GCs will rarely be useful. In most cases, they only serve to introduce additional delay while shifting the right GC to the main RAT cell in the SAB RAT.

Fig. 11 shows execution time in seconds taking implementation delay into consideration. For this experiment, we assume that the RAT delay determines the processor's clock period. To comment on the execution-time variation as a function of the number of GCs, we focus on the 4-way RAT for a 512-entry window size, shown in Fig. 11(a). Irrespective of the GC allocation policy, going from four to eight GCs, we observe up to 2.24% and 0.94% increase in total execution time for RAB and SAB, respectively. Furthermore, going from 8 to 16 GCs, we observe up to 10.42% and 6.92% increase in total execution time. We observe similar variation trends for other window sizes. Thus, when we consider the delay overhead introduced by additional GCs, IPC does not correctly predict actual performance. In particular, IPC measurements suggest that performance always improves by increasing the number of GCs. Results show that, as predicted by IPC measurements, four or eight GCs improve the overall performance compared to the case no GCs exist, and recovery must exclusively be done using ROB. However, contrary to IPC prediction, the increase in the RAT latency with 16 GCs outweighs the IPC benefits. Furthermore, while eight GCs typically offer better IPC performance than four GCs, absolute performance in most cases deteriorates.

As expected, the best design when we focus on IPC alone would be different from the best design when we consider both IPC and latency. IPC measurements predict that the best performance is achieved by SEL_RAB with 16 GCs. Whereas, the best performance (execution time) is achieved by SEL_SAB with four GCs when the implementations' worst case delay is taken into consideration.

The results show that ignoring the actual latency of the RAT incorrectly predicts performance, and performance does not monotonically increase with increasing the number of GCs as IPC measurements suggest. Two components contribute to determining performance, and these components are at odds with each other. First, as more GCs are introduced, fewer cycles are spent recovering from mispeculations, hence improving performance. Second, introducing more GCs increases RAT latency and, consequently, increases the clock period and decreases performance. In most cases, using very few GCs (e.g., four) leads to optimal performance.

C. Energy

Fig. 12 shows the average total RAT energy as a function of the number of GCs and window size for both implementations and GC management policies. Each reported energy value totals the energy for the RAT reads, RAT writes, GC allocations, and GC restorations. SAB consumes less energy than RAB since

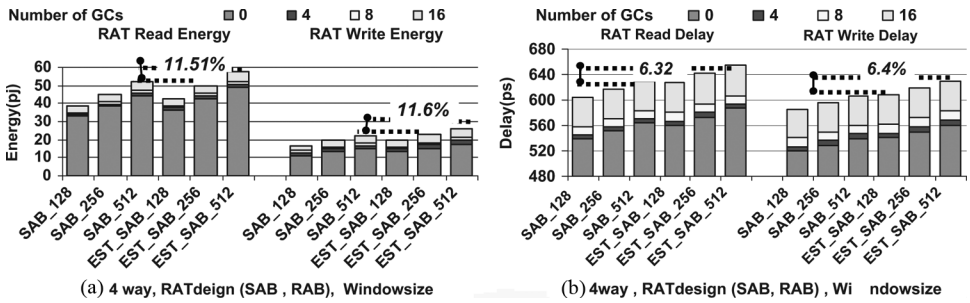

Fig. 13. (a) Energy and (b) delay as a function of number of GCs and window sizes for the 4-way RAT (simulation results and model estimations).

more than 90% of RAT accesses are reads and writes that SAB consumes less energy for them than RAB. The GC allocation and restoration of SAB are more energy consuming than those of RAB; however, these operations are relatively infrequent (10% of the total RAT accesses).

D. On the Accuracy of the Analytical Models

This section discusses the accuracy of the models. In this analysis, the relative estimation error is calculated by (43)

$$\%\,\text{Error} = \frac{\text{Analytical} - \text{Simulation}}{\text{Simulation}} \times 100. \qquad (43)$$

Fig. 13(a) and (b) show the circuit measurements with the analytical model estimations for energy and latency as a function of the number of GCs. The worst case relative error per operation is also shown. The worst case relative errors for energy and latency are within 11.6% and 6.4% of the Spectre simulation results, respectively. The errors are monotonic, and the estimations are in agreement with the physical-level simulation results in predicting delay and energy variation trends.

As expected, the analytical model estimations differ from the simulation results. Several sources of error cause this difference: Comparisons of the model-estimated and layout-extracted capacitances show that about 5.1% of the error for energy is due to capacitance estimation inaccuracy. The formulas used to calculate gate and diffusion capacitances are oversimplified, and the capacitances are assumed to be voltage independent [15], [17]. Each stage in our models (e.g., bitline and wordline) assumes that the inputs to the stage are step waveforms; however, actual waveforms are far from steps, hence impacting the delay of a stage. The energy model exhibits a worst case error of about 11.6%. Inaccurate capacitance estimation plays a key role in dynamic power consumption. The leakage power model accounts for 5.7% of this error. Leakage current largely depends on the state of the circuit and temperature. Hence, leakage current cannot be accurately quantified without circuit simulations.

VI. Conclusion

Previous RAT checkpointing work developed GC count reduction techniques focusing solely on IPC performance evaluation to compare alternatives. Although previous work assumed that increasing the number of GCs increases the RAT delay, in performance evaluation, they ignored the effect of the number of GCs on RAT delay and clock period. This paper improves upon previous work by determining quantitatively how RAT delay and energy vary as a function of the number of GCs, issue width, and window size, utilizing two representative full-custom checkpointed RAT implementations in a 130-nm fabrication technology. IPC performance evaluations show that performance improves monolithically by introducing more GCs. However, this paper demonstrates that, when RAT delay is taken into consideration, actual performance (execution time) rarely improves with more than four GCs for the SRAM-based checkpointed RAT. This paper also shows that, although RAB, representative of recent checkpointed RAT designs, offers better IPC performance, SAB is superior in achieving better actual performance since SAB offers faster RAT reads and writes. Additionally, this paper presents analytical delay and energy models for checkpointed RATs. Analytical models help computer architects estimate the latency and energy of various checkpointed RAT organizations during architectural-level exploration, where physical-level implementation is unavailable or unaffordable. Comparisons show that the estimations provided by the models are in satisfying agreement with the simulation results.

[