A Speculative Control Scheme for an Energy-Efficient Banked Register File

Jessica H. Tseng, *Student Member*, *IEEE*, and Krste Asanović, *Member*, *IEEE*

Abstract—Multiported register files are critical components of modern superscalar and simultaneously multithreaded (SMT) processors, but conventional designs consume considerable die area and power as register counts and issue widths grow. Banked multiported register files consisting of multiple interleaved banks of lesser ported cells can be used to reduce area, power, and access time and previous work has shown that such designs can provide sufficient bandwidth for a superscalar machine. These previous banked designs, however, have complex control structures to avoid bank conflicts or to buffer conflicting requests, which add to design complexity and would likely limit cycle time. This paper presents a much simpler and faster control scheme that speculatively issues potentially conflicting instructions, then quickly repairs the pipeline if conflicts occur. We show that, once optimizations to avoid regfile reads are employed, the remaining read accesses observed in detailed simulations are close to randomly distributed and this contributes to the effectiveness of our speculative control scheme. For a four-issue superscalar processor with 64 physical registers, we show that we can reduce area by a factor of three, access time by 25 percent, and energy by 40 percent, while decreasing IPC by less than 5 percent. For an eight-issue SMT processor with 512 physical registers, area is reduced by a factor of seven, access time by 30 percent, and energy by 60 percent, while decreasing IPC by less than 2 percent.

Index Terms—Low-power, register file, speculative control, superscalar, simultaneous multithreading.

1 Introduction

Multiported register files and bypass networks lie at the heart of a superscalar microprocessor and provide buffered communication of register values between producer and consumer instructions. With the deeper pipelines and higher instruction-level parallelism (ILP) of next generation out-of-order superscalar processor designs, both the number of ports and the number of required registers increase. These increased requirements cause the area of a conventional multiported regfile to grow more than quadratically with issue width [25]. The trend toward simultaneous multithreading (SMT) further increases register count as separate architectural registers are needed for each thread. For example, the proposed eight-issue Alpha 21464 design had a regfile that occupied over five times the area of the 64 KB primary data cache [16].

Many techniques have been previously proposed to reduce the area, energy, and delay of multiported register files. Some approaches split the microarchitecture into distributed clusters, each containing a subset of the register file and functional units [19], [14], [9], [12], [26], [18]. These schemes have the potential to scale to larger issue widths, but require complex control logic to map instructions to clusters and to handle intercluster dependencies. Alternatively, other approaches retain a centralized microarchitecture, but divide the physical register file into interleaved banks with fewer ports per bank [23], [2], [15], [13]. Provided that the number of simultaneous accesses to any bank is less than the number of ports on each bank, this structure can provide the aggregate bandwidth needs of a superscalar machine with significantly reduced area compared to a fully multiported regfile. These earlier banked schemes, however, require complex control logic with datapath buffering and stalls across multiple pipeline stages that would likely limit the cycle time of a high-frequency design.

In this paper, we present and analyze a banked multiported register file design together with a control scheme suitable for a deeply pipelined dynamically scheduled processor. Our control scheme does not place any register bank arbitration in the critical wakeup-select loop, but instead speculatively issues potentially conflicting instructions. If any conflicts are found after issue, a pipelined recovery scheme quickly repairs the issue window and reissues conflicting instructions. In contrast to previous work [23], [2], [15], [13], all conflicts are detected and resolved in one pipeline stage so that no write buffering or pipeline stalls are required. The main drawback of our scheme is that both bank conflicts and the extra pipeline stage used for port arbitration can impact processor performance. Bank conflicts add penalty cycles to repair the pipeline and delay the issuing of dependent instructions, while the additional pipeline stage causes an increase in branch misprediction latency.

To evaluate the scalability of this work, we first examine the performance of our proposed banked register file scheme in superscalar processors and then extend it to SMT processors. One might expect that the higher utilization of an SMT processor would raise the number of bank conflicts and, hence, reduce the applicability of our banked regfile design. Surprisingly, our data instead reveals that banking produces even better results for an SMT core (< 2 percent IPC degradation) than a single thread

• *The authors are with the MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139.*
E-mail: {jhtseng, krste}@csail.mit.edu.

Manuscript received 14 Mar. 2004; revised 16 Aug. 2004; accepted 8 Oct. 2004; published online 15 Apr. 2005.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TCSI-0091-0304.

superscalar core (< 5 percent IPC degradation). This result is partly due to SMT's ability to hide the increased branch mispredict penalty as, when one thread experiences a misprediction, other threads can continue to execute instructions. Also, the larger number of registers required by an SMT processor allows a larger number of regfile banks, reducing conflicts significantly. Consequently, we believe banked regfiles are a natural solution to the increasing register file demands of SMT processors.

The paper is structured as follows: We first describe our scheme in detail in Section 2, including the pipeline structure and required control logic. We then present area, energy, and delay numbers from detailed circuit layouts in Section 3. We present an analytical model to determine the expected probability of port conflicts in Section 4 and, in Section 5, we show the performance simulation results and compare these to the analytic model. We discuss the relationship of our scheme to previous work in Section 6, before concluding in Section 7.

2 Banked Regfile Design

In this section, we first describe the structure of the banked multiported register file datapaths. We then describe the operation of the control logic, including the overall pipeline design, the pipeline repair operation used after conflicts, and the control for the two read port optimizations, conservative bypass-skip and read sharing.

2.1 Register Bank Structure

A banked register file consists of multiple interleaved banks of non-fully-ported register cells. Fig. 1 shows one example of our register banking scheme for a four-issue processor. The regfile provides a total of eight global read ports and four global write ports using four interleaved register banks, each with two local read ports and two local write ports. Compared to a conventional multiported structure, each word of register storage has fewer ports and the storage cell size is dramatically smaller. But, now, additional multiplexing circuitry is required to connect the local port bitlines to the global port bitlines and the possibility of bank conflicts arises when too many global ports attempt to read or write the same bank.

As shown in Fig. 1, each functional unit needs two global read ports, which we term the left and right ports, to execute instructions with two register source operands. We simplify the local-global port crossbar by connecting one local port on each bank to only the global left operand buses and the other to only the global right operand buses. Therefore, all banks have at least two local read ports. This enables any instruction to retrieve both operands from the same bank in one cycle, but doesn't allow the use of the local left ports to fetch global right port operands. Apart from the reduction in mux circuitry, this restriction simplifies port arbitration logic by cutting in half the number of possible contenders for a local read port.

In contrast, the design presented in [2] employed banks with only a single read port. The single read port must connect to all global ports and, hence, requires the same local-global crossbar complexity as a dual read-port design that connects each local port to half the global ports. Our initial result reveals minimal area savings for the single read port design versus the split dual port design once the cost of the local-global crossbar is included. Moreover, the single read port bank requires considerably more complicated control logic to handle execution of an instruction that fetches both operands from the same bank and each read port arbiter has twice as many inputs.

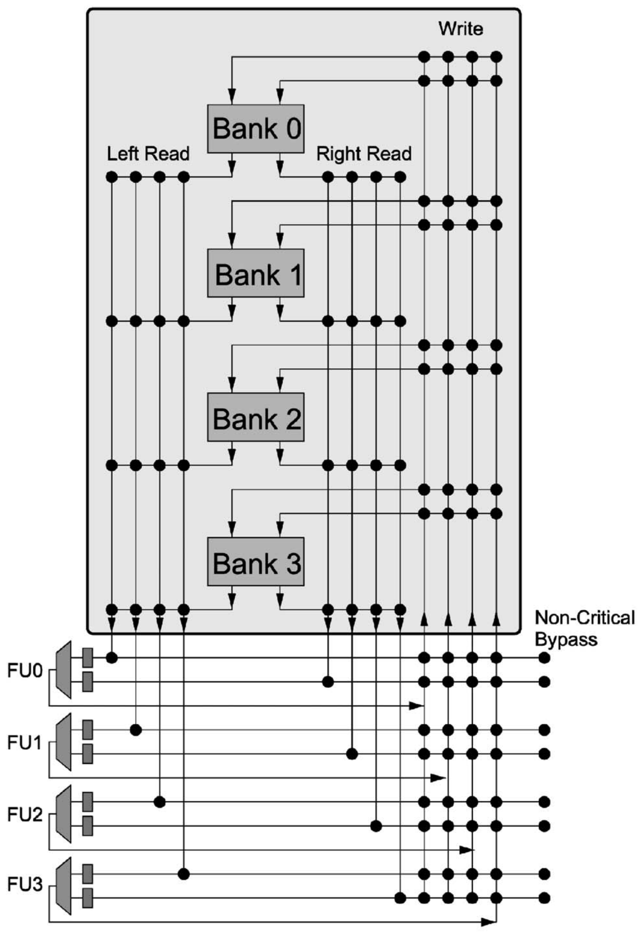

Fig. 1. An eight-read, four-write port register file implemented using four two-read, two-write port banks. The register file interconnect and bypass network are shown as distributed muxes where each dotted crosspoint represents a potential switched connection.

Fig. 1 also shows a portion of the bypass network for functional units with single cycle latency; critical multiple cycle units, such as load units, will require additional bypass paths. Register file writeback may require one or more cycles, in which case, additional bypass logic is required for results that have completed but which are not yet available from the register file. These delayed bypass paths are not latency critical and can be supplied by an early stage mux that feeds into the final latency-critical mux stage [10].

2.2 Control Logic

A banked multiported register file can provide sufficient bandwidth for a superscalar processor at a lower cost than a flat design. The main challenge is devising control logic that can handle the inevitable bank conflicts without compromising cycle time or adding excessive complexity.

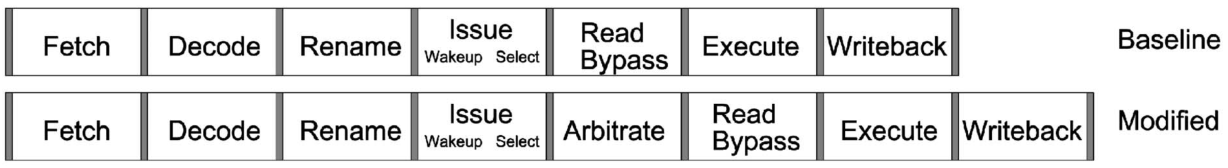

Fig. 2. Pipeline structures of processor with unified register file and processor with multibanked register file. An additional cycle is added for multibanked register file for read port arbitration and muxing. Read bank and write bank conflicts are also detected in this cycle.

2.2.1 Pipeline Design

We propose a pipelined control scheme where the pipeline first speculatively issues potentially conflicting instructions, then performs port arbitration in a later pipeline stage. If any conflicts are detected after issue, a pipelined recovery scheme, described below, quickly repairs the issue window and reissues conflicting instructions. Fig. 2 shows the baseline processor pipeline for the flat register file structure and the modified processor pipeline for the banked register file structure. Previous work has either placed additional arbitration logic in the select path to avoid conflicts or required that multiple pipeline stages be stalled [23], [2], [15], [13]. Both approaches complicate critical timing loops [3]. In particular, stalling a deep pipeline is usually prohibited in high-frequency designs due to the difficulty of generating and routing a global stall signal to a large number of pipeline registers.

For our baseline and modified processors, instructions are first decoded and renamed and then placed into an instruction window. Instructions wait in the instruction window until both operands are available. The instruction window pipeline stage contains the critical wakeup-select loop [14], where the wakeup phase is used to update operand readiness and the select phase picks a subset of the ready instructions to issue. Once a single-cycle instruction is selected, its result tag is immediately broadcast to the instruction window in the next wakeup phase to allow back-to-back issue of dependent instructions, even though the selected instruction will not produce its result for several cycles.

A conventional pipeline has a fixed mapping of issued instruction operands to register file ports, so operands of an instruction can be fetched from the regfile immediately after issue. A banked regfile scheme, however, must first mux operand addresses into the available register file ports. The extra arbitration pipeline stage shown in Fig. 2 detects both read and write regfile conflicts and muxes the winning addresses into the address decoders. The arbitration stage also manages requests to writeback results from long or variable latency operations such as divides or cache misses. These are given higher priority than newly issuing instructions. In Fig. 2 and in our evaluation, we allocate a whole pipeline stage to the arbitration and register address mux, but we believe the actual penalty would be much lower in practice.

All instructions that pass the arbitration phase can read from and write back to the regfile with no conflicts. This approach avoids register bank write buffers [2], which increase the size of the bypass network and cause pipeline stalls when full, but sufficient write ports must be provided such that write bank conflicts do not cause a large performance degradation. This approach is also much simpler than schemes that delay physical register allocation until writeback to avoid conflicts [15].

2.2.2 Repairing the Issue Window

The arbitration stage detects all bank conflicts, that is, when too many reads try to access the left or right side of a single bank or too many writes try to access the same bank. If such conflicts are detected, the processor must repair the issue window and reissue the conflicting instructions. Fig. 3 shows the method by which the pipeline state is restored after a conflict. A second group of instructions following the ones that encounter a conflict will have been speculatively issued into the pipeline in parallel with the detection of the conflict. This second group is killed, along with the instructions in the first group that were not granted a read or write port. The wakeup phase that would have been used to broadcast the tags of the second group of instructions is now used to repair the issue window by broadcasting the tags and resetting the ready and issued bits for the

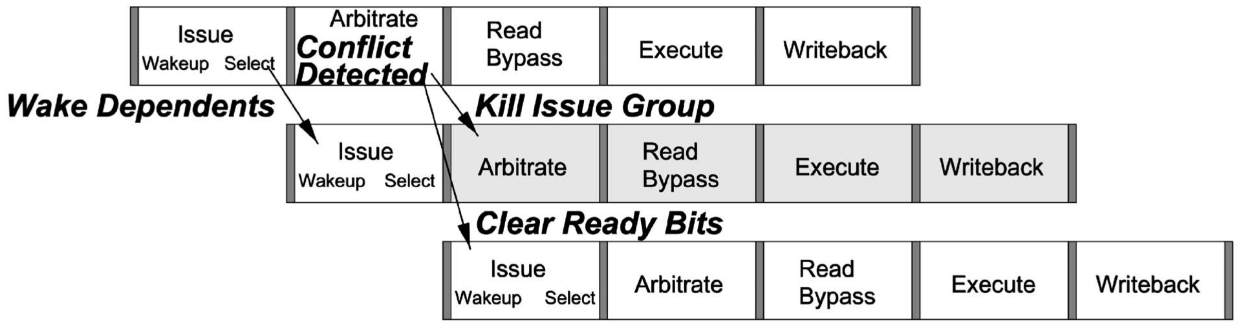

Fig. 3. Pipeline diagram shows repair operation after conflicts are detected. The wakeup tag search path is used to clear ready bits of instructions that had a conflict causing them to be reissued two cycles later. Any intervening instruction issues are killed.

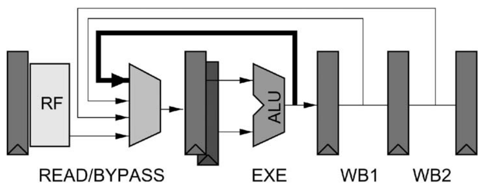

Fig. 4. Conservative bypass skip only avoids read port contentions when the value is bypassed from the immediately preceding cycle.

destinations of the killed instructions in the first group. Issue will now resume correctly in the select phase of this pipeline stage. This approach adds only a mux into the tag broadcast path of the critical wakeup phase.

2.2.3 Conservative Bypass-Skip

Previous work [21], [2], [15] indicates a great percentage of operands are either the dedicated zero register (R0 on MIPS, R31 on Alpha) or supplied from the bypass network. The number of requisite read ports can be reduced significantly if we have a separate zero input to the bypass mux and also if operands that will be sourced from the bypass network do not compete for access to the register file ports.

Avoiding read port contention for bypassed operands would at first appear to require that the arbitration logic wait until the bypass logic determines if operands will be bypassed. To avoid this increase in pipeline latency, the check can be folded into the wakeup phase. Previous work has described an optimistic bypass hint scheme [15] where an extra hint bit is added to each operand of instructions waiting in the issue window. The hint bit is cleared if the operand was ready before the instruction entered the issue window; otherwise, it is set. When the instruction is selected, an operand with the hint bit set will not contend for a read port as it is likely to be sourced from the bypass network. The disadvantage of this scheme is that it is only a prediction, which, when incorrect, requires stalling earlier pipeline stages to allow the instruction to access the read ports. The mispredictions also reduce performance in cases where the read could have been satisfied from a free read port if only it had been in contention.

We instead adopt a conservative bypass bit scheme, which is always correct but which only avoids contending for read ports for values bypassed from the immediately preceding cycle. Each operand in the instruction window has multiple comparators, one per tag write back port, to determine if the operand becomes ready in that cycle. We store the bypass bit for each instruction window operand in a latch that is loaded with the result of OR-ing together comparator results every cycle. If an instruction is selected in the same cycle where a tag match caused the instruction to wake up, the bypass bit will be set, indicating that the value is available from the bypass network. If the instruction is not selected for issue, the bypass bit latch will be cleared by the failing tag matches on the next wakeup phase. The bypass bit is conservative because it is only set for values that will be ready in the cycle before the current instruction executes, as shown in Fig. 4. Where there are several pipeline stages feeding the bypass mux (e.g., when register file access takes multiple cycles), this scheme will still compete for read ports even though these operands will be sourced from later pipeline stage bypasses. In practice, we find this lost opportunity causes negligible performance impact. To reduce datapath complexity, a microarchitecture might not support bypass from every functional unit. In this case, the wakeup tag search can be modified to broadcast a signal indicating whether the operand can be bypassed. This value can then be latched into the bypass bit on a successful tag match. Any register operand of an issued instruction which doesn't have the bypass bit set must contend for read ports.

2.2.4 Read Sharing

Read bank conflicts commonly occur when multiple instructions in an issue group try to read the same physical register [2]. Instructions that depend on the same register become ready on the same cycle and are likely to be issued together. The read port arbiter can detect this sharing and remove the conflict by setting enable signals such that a local port drives multiple global ports. Our register file structure only allows read sharing on either the left ports or the right ports and requires a second local port if a physical register is read from both sides.

3 Register File Layouts

This section describes the layouts of various sized banked register files we have undertaken to determine their area, delay, and energy. All designs were laid out in a 0.25 μm CMOS process from TSMC. The storage cells are a standard six transistor SRAM design, with differential write ports and single-ended read ports.

Metal 1 is used for local bitlines within a bank and metal 2 for word lines. The local ports from each bank then connect to the global bitlines running over the cells in metal 3. Most previous work has assumed that a large conventional multiported register file would have each port on a storage cell connected directly to the global bitline. With more metal layers, it is desirable to employ a hierarchical bitline structure, where each port on a cell connects to a local bitline which in turn connects to the global bitline [1], [10]. On each access, only one local bitline is connected to the global bitline. The parasitic drain capacitances of the storage cells in other banks are not driven, reducing delay and energy dissipation. Another benefit is that signal-to-noise ratio improves in the presence of leakage currents from off cells [1]. Adopting hierarchical bitlines in our baseline flat design reduces the relative energy and delay advantages of a multibanked design. To save area, we employ a single-ended global write bitline which is converted to a differential local bitline using a local inverter. To further save area, we pack two local storage cells into one global bit column where possible. This has the disadvantages that a 2:1 column mux is required, which adds area and delay, and that twice as many local bitlines are discharged on each access, which increases energy usage.

Table 1 shows the relative area of a variety of 64×32-bit eight read-ports and four write-ports multibanked register file designs in comparison to a unified design and Fig. 5 shows the detailed area breakdowns. Fig. 6 provides a graphical comparison of the floorplan of a few representative register file designs.

TABLE 1
Relative Area, Delay, Energy, and Leakage Numbers of Different 64×32-Bit Eight Global Read Port and Four Global Write Port Register File Designs

64×32b, 8 read ports, 4 write ports				
Area	8r4w	2r2w	2r1w	1r1w
1 banks	100.00%	-	-	-
4 banks	110.95%	28.99%	24.29%	22.88%
8 banks	122.55%	37.06%	31.97%	30.19%
Packing	1	2	2	2
Delay	8r4w	2r2w	2r1w	1r1w
1 bank	100.00%	-	-	-
4 bank	92.38%	79.05%	79.05%	81.90%
8 bank	83.33%	74.76%	74.76%	77.14%
Energy	8r4w	2r2w	2r1w	1r1w
1 bank	100.00%	-	-	-
4 bank	61.98%	57.93%	56.90%	40.54%
8 bank	61.41%	58.62%	57.55%	40.71%
Leakage	8r4w	2r2w	2r1w	1r1w
SRAM	100.00%	40.36%	29.32%	25.32%

Packing is the number of local bit cells packed per global bit column.

For the designs with eight read ports and four write ports per storage cell, moving to hierarchical bitlines adds area because of the interconnection overhead. An additional 11 percent area overhead is incurred when there are 16 words per local bitline and an additional 23 percent moving to eight words per local bitline. Bank conflicts do not occur in these designs.

As the number of local ports per bank is reduced, area drops dramatically. Compared to the baseline design, the designs with four banks are around one quarter the size and the designs with eight banks are around one third the size. Apart from the reduction in storage cell size, designs with smaller numbers of ports per bank have significantly less address decoder area than the highly multiported designs. Each bank has fewer decoders with narrower addresses. The design with four banks, each with one read port, cannot sustain eight global read port accesses and relies on the bypass network to supply the missing read operands.

Both Fig. 5 and Fig. 6 also show that multiplexing overhead dominates when there are only a few ports per cell. Designs with two read ports per bank are only a few percent larger than designs with a single read port per bank given that the single read port must connect to all global read ports, whereas each of the two read ports only connects to half of the global read ports. Also, increasing the number of write ports from one to two adds only 16-20 percent in area.

Table 1 also lists normalized delay and energy measures for the different multibanked register file designs compared to the unified design. Fig. 5 also shows the detailed delay and energy breakdowns of these designs. Delay and energy numbers were obtained from HSPICE simulations of an extracted layout with a 2.5 V supply voltage. For the fully ported storage cell designs, using hierarchical bitlines reduces energy by almost 40 percent and cuts delay by 8-17 percent. The lesser-ported bank designs are slightly faster, up to around 20 percent faster for the two read, two write port case. The energy reduction is also slightly greater

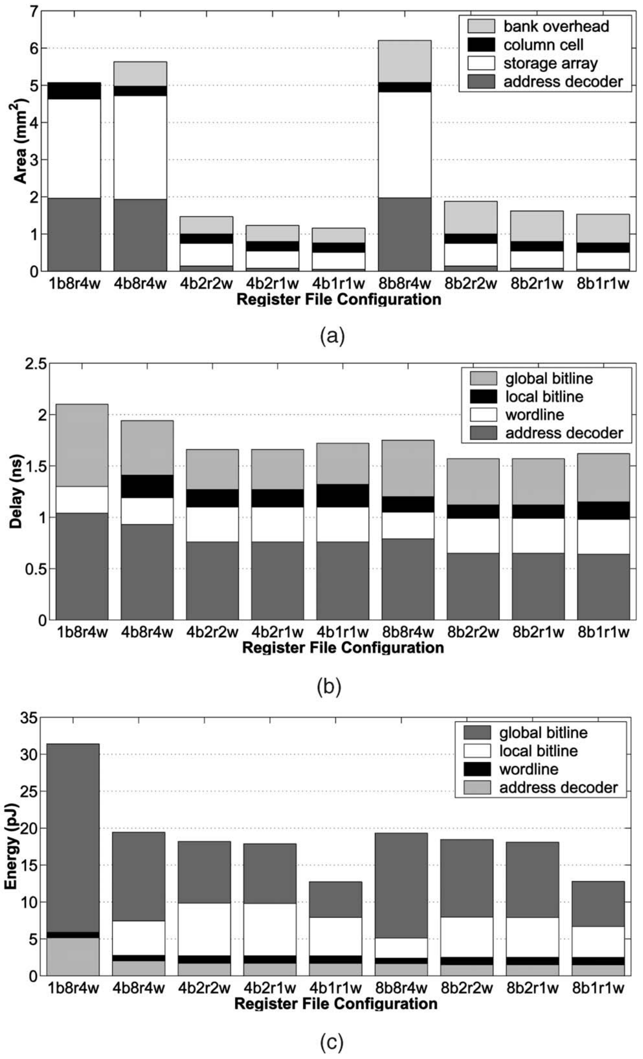

Fig. 5. Detail breakdown of various $64 \times 32b$ eight read-ports and four write-ports register file designs in terms of its (a) area, (b) read access delay, and (c) read energy consumption. For example, 1b8r4w refers to the baseline implementation—using one bank with eight read ports and four write ports.

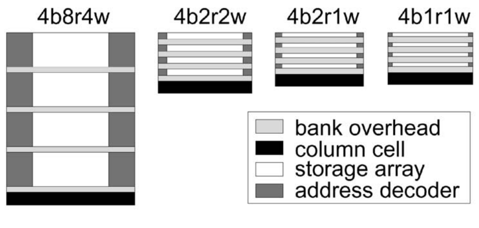

Fig. 6. Area comparison of four different $64 \times 32b$ regfiles for a quad-issue processor. The clear regions represent the storage cells, while the lighter shaded regions represent the overhead circuitry in each bank. The black shading at the bottom is the area required for the global bitline column circuitry. The medium-dark shading to the side is the area for address decoders.

TABLE 2
$F(d, A, B, N)$ for Various d Values

d	$F(d, A, B, N)$
2	$\sum_{j_0=N+1}^{A-(N+1)} \sum_{j_1=N+1}^{A-j_0} {}_AC_{j_0} \cdot {}_{A-j_0}C_{j_1} \cdot (B-2)^{A-j_0-j_1}$
3	$\sum_{j_0=N+1}^{A-2(N+1)} \sum_{j_1=N+1}^{A-j_0-(N+1)} \sum_{j_2=N+1}^{A-j_0-j_1} {}_AC_{j_0} \cdot {}_{A-j_0}C_{j_1} \cdot {}_{A-j_0-j_1}C_{j_2} \cdot (B-3)^{A-j_0-j_1-j_2}$
4	.
.	.
.	.

for the lesser-ported cells compared with just using hierarchical bitlines on the fully ported cells. The delay and energy reductions are not as great as might be expected from the area reduction as the packing of two local storage cells per global bit column slows the wordline drive and adds a column mux stage and also causes twice as many bitlines to discharge on a read. It might be possible to reoptimize the smaller ported cells for even smaller delay and energy, but this would add considerable additional area.

The primary source of energy dissipation for the 0.25 μm CMOS process is dynamic switching of load capacitance. Within a few process generations, it is expected that static leakage current will be responsible for a large fraction of total power dissipation [5]. Table 1 also shows the relative size of leakage energy across the three designs. The relative leakage energy numbers were obtained by calculating the total width of leaking transistors, assuming that 80 percent of stored values are zero and that the bit cell ports were optimized to reduce read energy for zero values [21]. Banking reduces leakage power by at least 60 percent over the baseline case and, so, we expect even greater relative power savings as leakage currents grow.

4 Modeling Local Port Contention

In this section, we develop an analytic model of register file bank conflicts under the assumption that register file accesses are uniformly randomly distributed across banks. We later use this model to help understand the results obtained by detailed microarchitectural simulation.

Bank conflicts occur when too many instructions compete for the same bank port during the same cycle. The port conflict probability (PCP) is the probability of having any conflicts in any regfile bank within a cycle. We can compute the expected PCP for a cycle based on the number of simultaneous accesses attempted by the processor and on the number and structure of banks in the design. The PCP is always zero for a regfile with fully ported storage cells.

For a two-banked regfile structure with $Bank0$ and $Bank1$, PCP can be expressed as (1). $P(Conflict_{Bank0} \cup Conflict_{Bank1})$ is the probability of either $Bank0$ or $Bank1$ encountering conflicts. $P(Conflict_{Bank0})$ is the probability of $Bank0$ having conflicts, $P(Conflict_{Bank1})$ is the probability of $Bank1$ having conflicts, and $P(Conflict_{Bank0} \cap Conflict_{Bank1})$ is the probability of both banks having conflicts in the same cycle.

$$\begin{aligned} PCP &= P(Conflict_{Bank0} \cup Conflict_{Bank1}) \\ &= P(Conflict_{Bank0}) + P(Conflict_{Bank1}) - \\ &\quad P(Conflict_{Bank0} \cap Conflict_{Bank1}). \end{aligned} \tag{1}$$

To calculate the PCP, we first need to determine M (2), which is the maximum number of banks that could encounter conflicts in a cycle given A accesses to a regfile with B banks of N ported storage cells. When M is zero, we have a conflict-free cycle. When M is one, we have the possibility of conflicts in only a single bank (e.g., $P(Conflict_{Bank0} \cap Conflict_{Bank1})$ is zero). When M is greater than one, we could have conflicts across multiple banks in the same cycle. Depending on the value of M, we calculate PCP differently (3). Since there are only two possible states, *conflict* and *nonconflict*, for each regfile bank, we use binomial coefficients to find conflicting sets (4). ${}_AC_x$ determines the number of instances that x out of A regfile accesses fetch values from the same bank regardless of the ordering and $\frac{(B-1)^{A-x}}{B^{A-1}}$ is its event probability. $O(A, B, N)$ adjusts for duplicate cases where multiple banks encounter bank conflicts in the same cycle.

$$M = \min\left(B, \left\lfloor \frac{A}{N+1} \right\rfloor\right), \tag{2}$$

$$PCP = \begin{cases} 0 & if\ M < 1 \\ \sum_{x=N+1}^{A} {}_AC_x \cdot \frac{(B-1)^{A-x}}{B^{A-1}} & if\ M = 1 \\ \sum_{x=N+1}^{A} {}_AC_x \cdot \frac{(B-1)^{A-x}}{B^{A-1}} - O(A,B,N) & \\ & if\ M > 1, \end{cases} \tag{3}$$

$${}_AC_x = \frac{A!}{x!(A-x)!}. \tag{4}$$

Analyzing the conflict overlap, $O(A, B, N)$, between multiple banks, we have two sets of binomial coefficients to find the two orthogonal sets of combination. One is for the bank numbers, while the other is for the accesses. In (5), ${}_BC_d \cdot F(d, A, B, N)$ determines the number of instances where at least d out of B banks encounter conflicts and $\frac{1}{B^A}$ is its event probability. Table 2 shows how $F(d, A, B, N)$ can be calculated for different numbers of overlapping conflict banks.

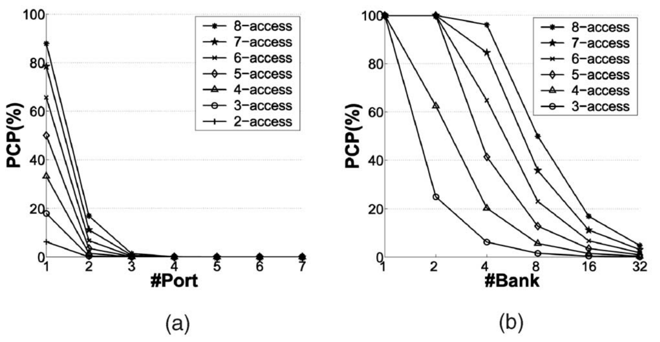

Fig. 7. PCP for designs with (a) 16 banks and (b) two local ports.

$$O(A,B,N) = \sum_{d=2}^{M} (-1)^d \cdot \frac{1}{B^A} \cdot {}_BC_d \cdot F(d,A,B,N). \quad (5)$$

Using (3), we plot the PCP for designs with 16 banks (varying the number of local ports) and designs with two local ports (varying the number of banks) for different numbers of accesses. Fig. 7 shows that having the additional second local port decreases conflicts by at least a factor of four and having a sufficient number of banks is crucial toward keeping the number of conflicts low.

5 Performance Evaluation

In this section, we evaluate the performance of our scheme for both superscalar and SMT processors. We describe the common methodology in Section 5.1. In Section 5.2, we report on simulations of four and eight bank configurations for a four-issue machine on Simplescalar [4] to determine the effects of different optimization techniques. Then, we extend our experiment to eight-issue SMT processors in Section 5.3 by simulating three different workloads on SMTSIM [22]. The *1-thread* workload models the behavior of superscalar processors, while *2-thread* and *4-thread* workloads account for the multithreading environment of SMTs. We compare our simulated results against the predicted results assuming uniform random regfile accesses in Section 5.4.

5.1 Methodology

We modified both Simplescalar and SMTSIM simulators to keep track of a unified physical register file organized into banks. We did not modify the register renaming strategy, simply taking the next available registers off a single FIFO free list regardless of bank allocation. The machine configurations are shown in Table 3. For designs with multibanked register files, we modeled the additional cycles required for read and write bank conflicts and pipeline repair. To account for the extra arbitration cycle, we increased branch misprediction latency by one cycle. The baseline design has a three-cycle latency for branch mispredictions, while other designs with multibanked register files have a four-cycle latency. The monolithic register file takes only one cycle to access, representing the most optimistic assumption.

For the Simplescalar simulations, we chose a subset of the SPEC CINT2000 and Mediabench benchmarks compiled with optimization for the PISA instruction set. The Mediabench benchmarks were used to provide some higher IPC codes that we would expect to cause greater register file traffic. For the SMTSIM simulations, we chose a complete set of SPEC CINT2000 benchmarks compiled with optimization for the Alpha instruction set. For *2-thread* and *4-thread* workloads, we randomly paired different combinations, as listed in Table 4. The Mediabench codes were run to completion. For the SPEC CINT2000 numbers, we used the methodology described in [17] to select a fast-forward period and sample length. We first simulate the baseline case, which uses a unified register file design and does not cause any register file port conflicts. Then, we analyze the performance of various multibanked register file schemes. By comparing the IPC between the monolithic and multibanked regfile simulations, we can determine the processor performance impact of using our banked regfile structure.

5.2 Performance Sensitivity

Fig. 8 and Table 5 show the resulting absolute and relative IPC numbers obtained for the four-issue machine with a 64-element register file. We label each configuration as (*#banks*)B(*#reads*)R(*#writes*)W(*bypass?*)(*sharing?*), where (*#banks*) is the number of banks, (*#reads*) is the number of local read ports, (*#writes*) is the number of local write ports,

TABLE 3
Simplescalar and SMTSIM Configurations

Item	Simplescalar	SMTSIM
L1 I-cache	32KB 2-way, 64-byte lines, 2 cycles	64KB 2-way, 64-byte lines, 2 cycles
L1 D-cache	32KB 2-way, 32-byte lines, 2 cycles	64KB 2-way, 64-byte lines, 2 cycles
L2 unified cashe	1MB 4-way, 64-byte lines, 10 cycles	1MB 4-way, 64-byte lines, 12 cycles
L3 unified cashe	N/A	8MB 8-way, 64-byte lines, 25 cycles
Fetch, dispatch, commit width	4	8
Integer ALUs	4	8
Memory instructions	2	4 (2-Load/2-store)
Register file	64 entries	512 entries

TABLE 4
Heterogeneous Multithreaded Workloads

Name	Applications
mix2.1	(bzip, twolf)
mix2.2	(gap, gzip)
mix2.3	(crafty, gcc)
mix2.4	(mcf, perl)
mix2.5	(parser, vortex)
mix2.6	(eon, vpr)
mix4.1	(crafty, gcc, gzip, votex)
mix4.2	(bzip, eon, mcf, twolf)
mix4.3	(gap, parser, perl, vpr)

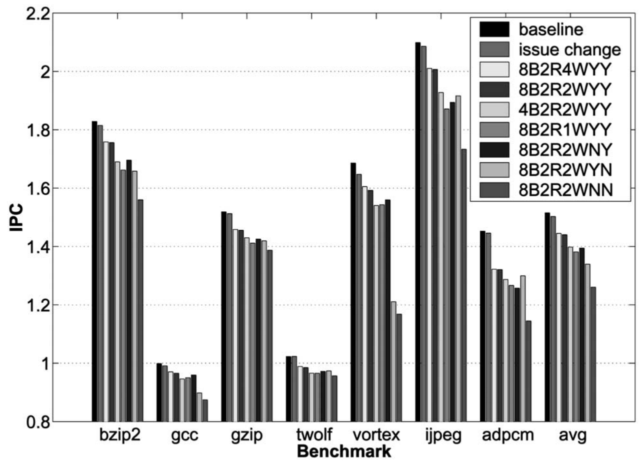

Fig. 8. IPCs for the 4-issue pipeline with register file of size 64.

(*bypass?*) indicates if values bypassed from the last execution cycle avoid competing for register ports, and (*sharing?*) indicates if local read ports can drive multiple global read ports to implement read sharing.

We have also included a configuration labeled *issue change* which shows the results if the select logic were changed to avoid register bank conflicts at issue time and where both bypassing and port sharing are used to reduce conflicts in a system with eight 2r2w banks. In this case, performance degradation was less than 1 percent compared to the baseline. In practice, this scheme would have a much slower wakeup-select loop, which would likely limit clock frequency and reduce total performance. The row labeled 8B2R2WYY shows the performance drop when we instead issue instructions without considering conflicts and kill conflicting instructions. Performance drops another 4-6 percent, but it is possible that this configuration could have a lower cycle time to make up for this difference.

Overall, we found the 8B2R2WYY configuration to perform well for this design point and we chose this as our center point in perturbing other parameters. Reducing the number of banks to four (4B2R2WYY) lowers performance by another 3-4 percent. We can also see that moving from one to two write ports (8B2R1WYY, 8B2R2WYY) improves performance by more than 4 percent, but having more than two write ports per bank (8B2R4WYY) only improves performance by another 0.3 percent. This is expected given that average IPCs are rarely above 2 and some instructions do not write to registers.

Omitting the bypass optimization (8B2R2WNY) degrades performance by over 3 percent. Omitting the sharing optimization degrades performance by around 7 percent. We confirmed the observation in [2] that groups of load and store instructions dependent on the stack pointer tend to issue together, probably at procedure call/return points. We also noticed that branch instructions dependent on the same register issuing together were another common source of read sharing. Omitting both bypassing and read sharing (8B2R2WNN) lowers performance by 12-14 percent.

5.3 Superscalar versus SMT

Fig. 9 shows the resulting absolute IPC obtained for *1-thread*, *2-thread*, and *4-thread* workloads. We found that

TABLE 5
Normalized IPC Percent for a Quad-Issue Machine with 64 Physical Registers

Type	bzip2	gcc	gzip	twolf	vortex	ijpeg	adpcm	average
baseline	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
issue change	99.3%	99.2%	99.6%	100.0%	97.7%	99.4%	99.5%	99.2%
8B2R4WYY	96.2%	97.2%	96.0%	96.7%	95.2%	95.8%	91.0%	95.4%
8B2R2WYY	96.1%	96.7%	95.8%	96.3%	94.5%	95.6%	90.9%	95.1%
4B2R2WYY	92.4%	94.7%	94.1%	94.4%	91.4%	91.9%	88.6%	92.3%
8B2R1WYY	90.9%	95.2%	92.9%	94.4%	91.5%	89.2%	87.2%	91.2%
8B2R2WNY	92.7%	96.1%	93.9%	95.0%	92.5%	90.2%	86.6%	92.1%
8B2R2WYN	90.7%	89.9%	93.5%	95.2%	71.8%	91.3%	89.5%	88.4%
8B2R2WNN	85.3%	87.6%	91.3%	93.5%	69.3%	82.6%	78.8%	83.2%

Configurations are labeled as (#banks)B(#local read ports)R(#local write ports)W(bypass skipped?)(read sharing?). Results are normaized to the IPC of the baseline case (unified with eight read and four write ports). Results for a configuation with bank arbitration in the issue logic (issue 8B2R2WYY) are also shown.

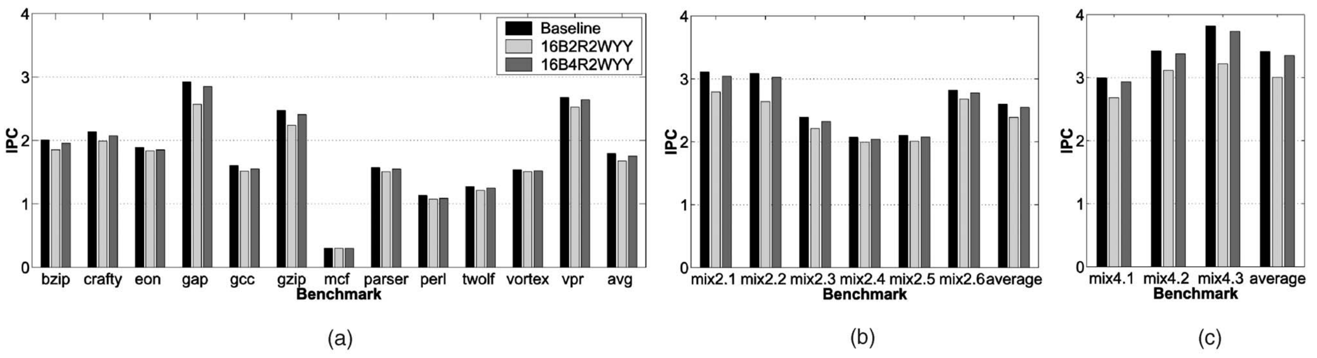

Fig. 9. IPCs for (a) 1-Thread, (b) 2-Thread, and (c) 4-thread workloads.

the average performance degradation of the 16B2R2WYY design increases from 5 percent, 8 percent, to 12 percent as we increase the workload from one, two, to four threads, respectively. This indicates that the register file does not provide sufficient throughput by having only two local read ports. After adding another pair of local read ports to the register file banks (16B4R2WYY), performance is restored to just a 2 percent IPC degradation in all three workload categories. In fact, the average IPC increases as we increase the workload, but the percentage of IPC degradation decreases. This is because the multithreading environment of SMT helps to hide the branch misprediction penalty as, when one thread experiences a misprediction, the other threads can continue to execute instructions. In comparison to a fully multiported design, our study shows that we can reduce regfile area by a factor of seven, access time by 30 percent, and energy by 60 percent if the 16B4R2WYY structure is used in an eight-issue SMT machine.

5.4 Correlation among Accesses

We now compare the observed conflicts against those predicted by the model developed in Section 4. We use the simulator to determine A, the number of accesses to the regfile in each cycle, then calculate the PCP, and sum over the whole run. We then compare the numbers against our SMTSIM simulation results for *4-thread* workloads on 16B4R2W regfiles with different combinations of optimization techniques in Fig. 10. We observe that, without the read port optimizations, there are many correlated accesses leading to a much higher conflict rate than that predicted by uniform random accesses. Applying *bypass-skipped* reduces the number of correlated accesses somewhat, but applying *read-sharing* causes fewer conflicts than a random distribution. We also achieve a slightly lower percentage of conflicting cycles than purely random accesses by applying both optimization techniques. However, for write port conflicts, the simulation numbers indicate that the write port allocation is less than uniformly distributed at random across all the banks. Unlike for reads, no optimization techniques are applied to remove or prevent write correlations.

We also investigated other register renaming policies, including LIFO and Random, to determine if different algorithms would have some impact on the correlations, but our results were unchanged for all cases.

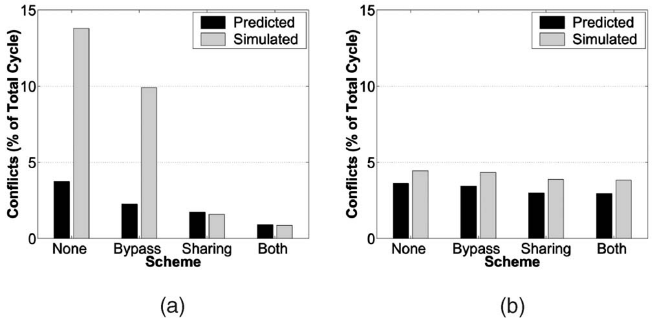

Fig. 10. Conflict cycle comparison for (a) reads and (b) writes.

6 Related Work

Monolithic regfile designs are known to scale poorly with increasing numbers of ports and registers. Many architects have explored alternative designs for implementing a large and fast multiported register file. One approach, used in the Alpha 21264 [12] and 21464 [16] designs, consists of dividing the functional units among two clusters and providing a copy of all registers in each cluster. This approach halves the number of read ports required on each copy of the regfile, but requires the same number of write ports on both regfiles to allow values produced in one cluster to be made available in the second cluster. An extension of this approach is to develop a clustered microachitecture that divides the registers among a number of clusters [19], [14], [9], [26], [18]. Clustered microarchitectures also allow the instruction window to be divided among clusters and have the potential to scale to larger issue widths at high clock frequencies. Clustering reduces the number of ports on each partition of the register file, but requires intercluster communication when a value is needed from a different cluster. The primary disadvantages of a clustered microarchitecture are the complexity of the intercluster control logic and the additional area required to achieve performance similar to a centralized architecture.

Other approaches retain a centralized and nonduplicated regfile structure, but explore different types of locality in regfile accesses. Locality of access has been used to design regfile structures that are similar to multilevel caches [7], [3], one-level less-ported [15], [13], and one-level multibanked [23], [2]. In [7], [3], registers are cached to reduce average access latency. Register caching can add considerable control complexity to an architecture as register caches have much worse locality than conventional data caches and determining the appropriate values to cache is nontrivial.

Using a less-ported structure and only allowing necessary regfile read accesses reduce the register file's area, energy, and access time. The designs in [15], [13] do not use banked reads to avoid increasing the complexity of the select logic. However, the select logic still has to select no more instructions than the number of available read ports after considering the bypass hint bits [15] or the prefetch flags [13]. Conversely, our scheme issues instructions without considering bypassability or conflicts and relies on rapid port arbitration and a nonstalling pipeline repair to reduce the pipeline latency impact. This enables the use of read banking to further reduce cell size.

The multibanking approach adopted in this paper and in previous work [23], [2] constructs a regfile from multiple interleaved register banks. The challenge is managing the complexity and added latency of the control logic needed to handle read and write bank conflicts and the mapping of register ports to functional units. A banking scheme that uses the bypass network to reduce unnecessary read port contention and usage is described in [23], but no description of the bypass check or read conflict resolution logic is given. Write conflicts are handled by delaying physical register allocation until writeback, at which point registers are mapped to nonconflicting banks. The primary motivation for this delayed allocation was to limit the size of the physical register file, but this can lead to a deadlock situation requiring a complex recovery scheme.

The scheme presented in [2] handles read bank conflicts by only scheduling groups of instructions without conflicts. This reduces the IPC penalty, but adds significant logic into the critical wakeup-select loop. A design with single-ported read banks is evaluated; however, this requires complex issue logic and functional unit datapaths to allow instructions where both operands originate from the same bank to be issued across two successive bank read cycles. As we found in our initial study, multiplexing circuits dominate the area of few-ported multibanked designs. Moving from a single read port to split dual read ports per bank has minimal area impact. In [2], write port conflicts are handled by buffering conflicting writes, which increases the size of the bypass network. Functional unit pipelines must also be stalled when conflicting writes queue up. In comparison, our scheme never has any write stalls because write bank conflicts are detected after issue and conflicting instructions are killed. By adding enough read and write ports per bank in our design, we are able to maintain acceptable performance. We justify the slight increase in the overall register file area in our scheme by its reduction in control logic complexity and bypass mux size.

The aforementioned work has focused on the design of high-bandwidth register files for dynamically scheduled superscalar processors with a single logical register file. Other work has examined the use of partitioned register files made visible to software. The SPARC architecture [24] has overlapping register windows where software explicitly switches between sets of registers. In-order superscalar implementations of the UltraSPARC exploit the fact that only one register window is visible to implement a dense multiported structure [20]. Clustered VLIW machines make the presence of multiple register file banks visible to software and the compiler is responsible for mapping instructions to clusters [11]. Vector machines have also long been designed with interleaved register file banks that exploit the regular access patterns of vector instructions to provide high bandwidth with few conflicts [6], [8].

7 Conclusion

We have presented an energy efficient banked multiported regfile design together with a speculative control scheme suitable for a high-performance dynamically scheduled processor. To prevent increases in cycle time, our control scheme avoids register bank arbitration in the already timing-critical issue logic, but instead adds an extra pipeline stage to detect and resolve any conflicts. Since bank conflicts can degrade performance, we keep the number of conflicts low without buffering by using a sufficient number of banks and ports and by removing the correlation between accesses to the same bank. Through layout studies, we show that, in comparison to the minimally ported design, adding a few ports to each bank only slightly diminishes the area, delay, and energy savings because the layout is dominated by bank interconnect for a small number of ports per bank. For a four-issue superscalar processor, we reduce the 64-entry regfile size by over a factor of three, access time by 25 percent, and access energy by 40 percent, while reducing IPC by under 5 percent with the 8B2R2WYY configuration. For an eight-issue SMT processor, the area of the 512-entry regfile is reduced by a factor of seven, access time by 30 percent, and energy by 60 percent, while IPC is degradated by less than 2 percent with the 16B4R2WYY configuration. We expect the power savings of using smaller and less ported storage cells to increase as leakage rises with decreasing feature size. Even though this banked regfile design exhibits a small performance penalty, the reductions in register file delay and power can potentially be used to increase the clock rate and lead to a more complexity-effective design. The ability to reduce area and power significantly with minimal performance degradation should also make this approach attractive for multiprocessor chips which are designed to provide the highest possible thread throughput at low cost.

Acknowledgments

The authors thank Dean Tullsen for providing and helping them with SMTSIM. This work was funded in part by a US National Science Foundation (NSF) graduate fellowship, an NSF CAREER award, grants from the US Defense Advanced Research Projects Agency (DARPA) PAC/C program and the DARPA HPCS project with IBM, and an equipment donation from Intel.

References

[1] A. Alvandpour, R. Krishnamurthy, K. Soumyanath, and S. Borkar, "A Low-Leakage Dynamic Multi-Ported Register File in 0.13 μm CMOS," *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED)*, pp. 68-71, 2001.

[2] R. Balasubramonian, S. Dwarkadas, and D.H. Albonesi, "Reducing the Complexity of the Register File in Dynamic Superscalar Processors," *Proc. 34th Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO-34),* Dec. 2001.
[3] E. Borch, E. Tune, S. Manne, and J.S. Emer, "Loose Loops Sink Chips," *Proc. High Performance Computer Architecture (HPCA),* pp. 299-310, Feb. 2002.
[4] D. Burger and T. Austin, "The Simplescalar Toolset, Version 2.0," technical report, Univ. of Wisconsin-Madison, June 1997.
[5] A. Chandrakasan, W.J. Bowhill, and F. Fox, *Design of High Performance Microprocessor Circuits.* IEEE Press, 2000.
[6] Unisys Corp., "Scientific Processor Vector File Organization," US patent 4,875,161, Oct. 1989.
[7] J.-L. Cruz, A. Gonzalez, M. Valero, and N.P. Topham, "Multiple-Banked Register File Architectures," *Proc. Int'l Symp. Computer Architecture (ISCA-27),* pp. 316-325, 2000.
[8] DEC, "Vector Register System for Executing Plural Read/Write Commands Concurrently and Independently Routing Data to Plural Read/Write Ports," US patent 4,980,817, Dec. 1990.
[9] K.I. Farkas, P. Chow, N.P. Jouppi, and Z.G. Vranesic, "The Multicluster Architecture: Reducing Cycle Time through Partitioning," *Proc. 30th Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO-30),* pp. 149-159, 1997.
[10] E.S. Fetzer et al., "A Fully-Bypassed 6-Issue Integer Datapath and Register File on an Itanium Microprocessor," *IEEE J. Solid-State Circuits,* vol. 37, no. 11, pp. 1433-1440, Nov. 2002.
[11] J.A. Fisher, "Very Long Instruction Word Architectures and the ELI-512," *Proc. 10th Int'l Symp. Computer Architecture (ISCA-10),* pp. 140-150, 1983.
[12] R.E. Kessler, "The Alpha 21264 Microprocessor," *IEEE Micro,* vol. 19, no. 2, pp. 24-36, Mar./Apr. 1999.
[13] N.S. Kim and T. Mudge, "Reducing Register Ports Using Delayed Write-Back Queues and Operand Pre-Fetch," *Proc. 17th Ann. ACM Int'l Conf. Supercomputing (ICS),* pp. 172-182, 2003.
[14] S. Palacharla, N. Jouppi, and J.E. Smith, "Complexity-Effective Superscalar Processors," *Proc. 24th Int'l Symp. Computer Architecture (ISCA-24),* pp. 206-218, June 1997.
[15] I. Park, M.D. Powell, and T.N. Vijaykumar, "Reducing Register Ports for Higher Speed and Lower Energy," *Proc. 35th Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO-35),* Nov. 2002.
[16] R.P. Preston et al., "Design of an 8-Wide Superscalar RISC Microprocessor with Simultaneous Multithreading," *Int'l Solid-State Circuits Conf. (ISSCC) Digest and Visuals Supplement,* Feb. 2002.
[17] S. Sair and M. Charney, "Memory Behavior of the SPEC2000 Benchmark Suite," technical report, IBM Research Report, Yorktown Heights, N.Y., Oct. 2000.
[18] A. Seznec, E. Toullec, and O. Rochecouste, "Register Write Specialization Register Read Specialization: A Path to Complexity-Effective Wide-Issue Superscalar Processors," *Proc. 35th Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO-35),* Nov. 2002.
[19] G.S. Sohi, S. Breach, and T.N. Vijaykumar, "Multiscalar Processors," *Proc. 22nd Int'l Symp. Computer Architecture (ISCA-22),* 1995.
[20] M. Tremblay, B. Joy, and K. Shin, "A Three Dimensional Register File for Superscalar Processors," *Proc. Hawaii Intl Conf. System Sciences (HICSS),* Jan. 1995.
[21] J. Tseng and K. Asanović, "Energy-Efficient Register Access," *Proc. 13th Symp. Integrated Circuits and Systems Design,* Sept. 2000.
[22] D.M. Tullsen, S. Eggers, and H.M. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism," *Proc. 22nd Int'l Symp. Computer Architecture (ISCA-22),* 1995.
[23] S. Wallace and N. Bagherzadeh, "A Scalable Register File Architecture for Dynamically Scheduled Processors," *Proc. Int'l Conf. Parallel Architectures and Compilation (PACT),* Oct. 1996.
[24] D.L. Weaver and T. Germond, *The SPARC Architecture Manual/Version 9.* Prentice Hall, Feb. 1994.
[25] V. Zyuban and P. Kogge, "The Energy Complexity of Register Files," *Proc. 1998 Int'l Symp. Low Power Electronics and Design (ISLPED),* pp. 305-310, Aug. 1998.
[26] V.V. Zyuban and P.M. Kogge, "Inherently Lower-Power High-Performance Superscalar Architectures," *IEEE Trans. Computers,* vol. 50, no. 3, pp. 268-285, Mar. 2001.

Jessica H. Tseng received the SM degree in electrical engineering and computer science from the Massachusetts Institute of Technology and the BS degree in electrical engineering from the University of Florida. She is pursuing the PhD degree in electrical engineering and computer science at the Massachusetts Institute of Technology. Her research interests include low-power computer architecture and VLSI design. She is a US National Science Foundation Graduate Fellowship recipient and a student member of the IEEE.

Krste Asanović received the BA degree in electrical and information sciences from Cambridge University in 1987 and the PhD degree in computer science from the University of California at Berkeley in 1998. He is an associate professor in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology (MIT) and a member of the MIT Computer Science and Artificial Intelligence Laboratory. His research interests are computer architecture and VLSI design. He is a member of the IEEE and the IEEE Computer Society.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.