

Asymmetrically banked value-aware register files for low-energy and high-performance [☆]

Shuai Wang, Hongyan Yang, Jie Hu ^{*}, Sotirios G. Ziavras

Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102, USA

Available online 6 November 2007

Abstract

Designing high-performance low-energy register files is of critical importance to the continuation of current performance advances in wide-issue and deeply pipelined superscalar microprocessors. In this paper, we propose a new microarchitecture, the *asymmetrically banked value-aware register file (AB-VARF)*, to exploit the prevailing narrow-width register values for low-latency and energy-efficient register file designs. The register bit-widths of different banks in our AB-VARF register files are specifically customized to capture different narrow-width values. Augmented with a value width predictor, the register renaming logic is slightly tuned to rename predicted narrow-width registers to the corresponding narrow-width banks. Our experimental evaluation with SPEC CINT2000 benchmark suite shows that AB-VARF reduces the energy consumption by 78.4% over a conventional register file, on the average, at the cost of a 0.7% performance loss to an ideal 1-cycle monolithic register file.

© 2007 Elsevier B.V. All rights reserved.

Keywords: Low power; Register file; Narrow-width value; Performance

1. Introduction

High-performance dynamically scheduled processors rely on register renaming to eliminate false data dependencies among the dynamic instruction stream and to expose instruction level parallelism (ILP) for out-of-order issue and execution. A large and multiported physical register file, which accommodates the temporary results produced by instructions in flight, is the key to this ILP exploitation in modern wide-issue and deeply pipelined

superscalar processors. These two requirements, large size and multiported design, however, significantly increase the energy consumption and the access latency of the register file. Notice that register file read is within the critical path of the processor pipeline and long access latency will result in lower clock frequency, consequently degrading the processor performance. Pipelining register file access into multiple cycles is particularly problematic since register file read is within both the branch resolution loop and the load resolution loop [4,7]. Simultaneous multithreading (SMT) processors make this situation even worse due to their much higher demands on physical registers [21]. For instance, the register file in the 8-issue/4-threaded Alpha 21464 microprocessor has 512 physical registers with an area of more than five times of the level one data cache [29]. Further, the energy consumption of the register file can be up to 20% of the total chip energy [36].

Many techniques have been proposed to improve the scalability, in terms of performance, energy, and area, of the register file in designing high-performance energy-efficient microprocessors. These techniques include hierarchical register files [30,33,23,41,3], register caching [40,10,28,4,7],

[☆] A preliminary version of this work was presented at the IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2007). This submission has extended the previous one by including (I) a detailed analysis of written/read register value width distribution for SPEC CINT2000 benchmarks, (II) an implementation of the writeport scheduler for AB-VARF register files, (III) a leakage energy model for AB-VARF, and (IV) additional experimental results for evaluating the width predictor, AB-VARF's performance sensitivity to the register file size, and a detailed dynamic/leakage energy breakdown for the AB-VARF.

^{*} Corresponding author. Tel.: +1 973 596 5273; fax: +1 973 596 5680.
E-mail addresses: sw63@njit.edu (S. Wang), hy34@njit.edu (H. Yang), jhu@njit.edu (J. Hu), ziavras@njit.edu (S.G. Ziavras).

and clustered register files [18,9,13,8]. Other proposals try to reduce the required register file size for reduced access latency and energy consumption by delaying the physical register allocation [14,26], sharing physical registers to exploit value locality [17,25,2,15,35], or exploiting narrow-width values [1,21,19,12]. Alternative approaches employ banked register files with reduced read/write ports [38,3,37] or/and the bypass hint scheme to reduce the read requests thus the number of read ports [27,37] for high-performance and low-energy. Resource classification based register assignment for application specific instruction set processors (ASIPs) has also been investigated in [20] for code generation, targeting at the very small physical register files of high speed and low area.

The presence of narrow-width data in general-purpose applications is well understood and has been utilized for energy, performance, and reliability optimizations [5,22,1,21,19,12,11,16]. In this paper, we propose to utilize a banking scheme [38,27,37] for designing asymmetrically banked value-aware register files (AB-VARF) to reduce the energy consumption as well as the access latency in large register files. The novelty of our AB-VARF register file over previous banked designs is that the bit-width of banks in AB-VARF is specially customized to handle both narrow-width values and regular-width values for improved energy efficiency. Based on the study of the value width distribution, our AB-VARF in general consists of 16-bit banks, 34-bit banks, and 64-bit banks, which also reduces the overall size of a banked design. To take advantage of this AB-VARF, we also re-design the register renaming logic such that the destination register of a decoded instruction is renamed/steered to a physical register in a particular bank, according to its predicted data-width. Notice that the width of each physical register is implicitly determined by its association with a particular bank. Although renaming a register with wider data (to be produced) to a narrow physical register is problematic, narrow-width registers have the flexibility to map to wider physical registers in case the free lists of narrow physical registers are empty. Notice that in AB-VARF, each generated register value at the execution stage is assigned a two-bit flag from the width-detection logic. The only purpose of this two-bit flag is to verify the value width prediction from the fetch stage. Our AB-VARF scheme also adopts the optimized bypass hint scheme [37] to reduce the number of read requests to the register file. To attack the bottleneck issue, i.e., write port conflict in banked register files, we adopt a similar scheme as in [3] to have the selection logic also take into account the read/write ports as scheduling resource.

Our experimental evaluation with SPEC CINT2000 benchmark suites shows that, in the simulated eight-wide superscalar microprocessor, the AB-VARF register file (consisting of 2 16-bit banks, 1 34-bit bank, and 1 64-bit bank) with 4 readports/2 writeports per bank, reduces the energy consumption (including both dynamic and leakage energy) to a 18.1% of a conventional monolithic design, which presents an additional saving of 44.4% over a partitioned value-aware register file [11]. Further, AB-VARF reduces the energy consumption by 28.6% over a banked register file [37] with the same banking configuration. In the mean time, by applying writeport scheduling our AB-VARF reduces the performance loss over an ideal 1-cycle monolithic register file to 6.6% from the loss of 11.8% in a conventional banked register file. By increasing the per bank writeports to 4, AB-VARF minimizes the performance loss to 0.7% with its energy efficiency still fully exploited, achieving a 78.4% energy reduction.

The rest of the paper is organized as follows. We use Section 2 to review some basics of register renaming and provide a detailed analysis of narrow-width register values. We elaborate on the AB-VARF microarchitecture, read/write port scheduling, and width prediction in Section 3. Section 4 covers the experimental evaluation. Finally, Section 5 concludes this paper.

2. Basics of register renaming and narrow-width register values

2.1. Register files and register renaming

Superscalar microprocessors dynamically exploit instruction level parallelism (ILP) to issue multiple instructions per cycle for improved performance. Register renaming is one of the fundamental techniques employed in superscalar architecture to increase the ILP by eliminating two false data dependences, namely, anti-dependences and output dependences. We implemented a MIPS R10000 [39] style register renaming scheme, where the architectural and physical register files are combined. Fig. 1 gives the superscalar microprocessor model simulated in this paper.

At register renaming stage, the logical register ids of the source operands in a decoded instruction are used to access the register alias table (RAT), a.k.a. register mapping table. The table entry indexed by the logical register id contains the physical register id that the source register was renamed to. For the destination register, a free physical register is allocated from the register free list and the RAT is updated as follows: the old physical register id is

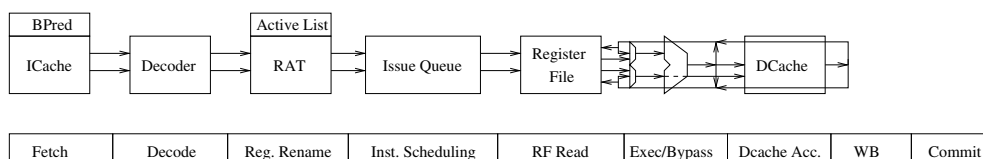


Fig. 1. Datapath and the pipeline stages of the simulated superscalar microprocessor.

read out from the RAT and stored in the active list entry allocated to the instruction, and then the new physical register id is written to the same RAT entry indexed by the logical destination register id. The destination register is said to have been re-mapped to the new physical register and the old physical register is said to have been unmapped. In case the register free list is empty, the renaming stage is stalled till some physical register is freed. Notice that a physical register cannot be freed until an instruction that previously unmapped this physical register is committed. To support wide-issue and deep pipelining, modern high-performance superscalar microprocessors employ a large multiported physical register file, which however significantly increases the energy consumption and also results in slow access.

2.2. Narrow-width register values

In high-performance 64-bit microprocessors, many generated register values during the execution do not require the full width of 64 bits. Values that can be represented by significantly less than 64 bits are generally referred as narrow-width values. The presence of narrow-width values has been well studied and exploited for performance, energy and reliability optimizations [5,22,21,12,19,1,16]. Different from the previous work, we exploit narrow-width register values for the design of partitioned and asymmetrically banked value-aware register files.

We present a similar study on the data-width distribution of integer register values for SPEC CINT2000 benchmarks (Alpha binaries) as in [5,21]. The detailed cumulative data-width distribution of written (destination) and read (source) register values is given in Fig. 2a and b, respectively. All integer register values are treated as signed numbers for this study. The data-width (W) of a given register value is determined as follows: $W = 64 - (\text{LS} - 1)$, where LS is the number of leading 0's or leading 1's. A data value V at width of W can be represented by W bits instead of 64 bits in the following way: $V = b_{W-1}b_{W-2} \cdots b_0$, where b_{W-1} is the sign bit, $b_{W-1} \oplus b_{W-2} = 1$, and $W \leq 64$.

Fig. 2a shows that on the average 48% of the generated register values have a data-width no more than 16 bits and an additional 9% of the values can be captured by a representation of 32 bits. However, there is a huge jump in the distribution from 33 bits to 34 bits, showing that 97% of the produced register values can be represented by no more than 34 bits. This significant jump (an additional 40%) is mainly contributed by the large number of memory address operations where the memory address of Alpha ISA uses 33 bits (plus 1 sign bit=34 bits). Please notice that the operations generating these memory addresses are different from the address calculation within a load/store instruction. The width distribution of read values in Fig. 2b shows a similar result. While the percentages of 16-bit and 32-bit values are reduced to 34% and 41%, 34-bit values still present 97% of the total reading register values. In this work, we exploit the large percentage of narrow-width values,

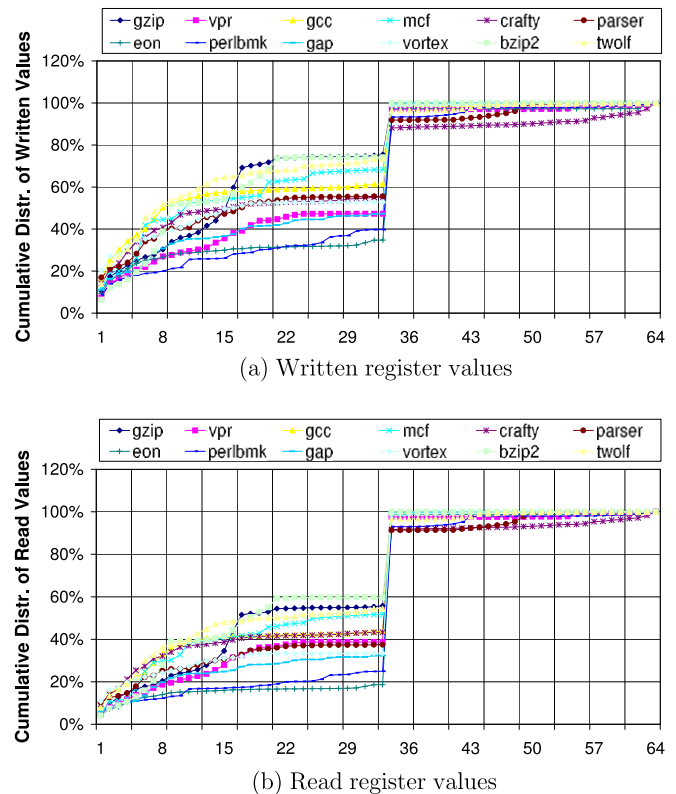


Fig. 2. A cumulative data-width distribution for (a) destination register values written to the register file, and (b) source register values read from the register file or the bypass network.

especially with width ≤ 16 bits and ≤ 34 bits, for designing high-performance and energy-efficient register files.

2.3. Narrow-width value detection

Based on the data-width analysis presented in the previous section, our design is particularly tuned to capture two types of narrow-width values: 16-bit values ($0^{48}0x^{15}$ or $1^{48}1x^{15}$) and 34-bit values ($0^{30}0x^{33}$ or $1^{30}1x^{33}$). Notice that 16-bit narrow-width values are just special cases or a subset of 34-bit values. The specific bit patterns of narrow-width data are given in Fig. 3. Note that a “x” in the bit pattern can be either a “1” or a “0”.

To capture narrow-width values, we utilize the existing zero (–1) detection logic within the functional units [24] to perform partial zero (–1) detection for the higher 31/49 bits in parallel with the regular zero (–1) detection. Thus, the narrow-width detection does not incur additional timing overhead to the functional units. The only hardware overhead might be very few additional AND gates to generate the zero signal for the higher 30/48 bits. After detection, two corresponding flag bits (N_1N_0) associated with each register value will be set to indicate the narrowness of the current value. The meaning of these two N_1N_0 bits is given in Table 1. The two flag bits will be used to verify the width prediction in AB-VARF.

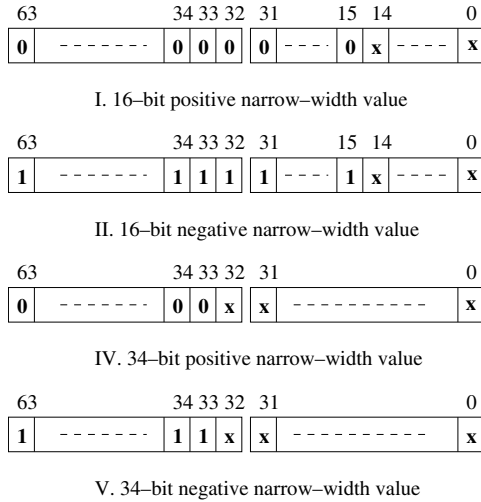


Fig. 3. Bit patterns for two types of narrow-width values considered: 16-bit value and 34-bit value. A “x” bit can be either “1” or “0”.

Table 1

The meanings of narrowness flag bits N_1N_0

N_1	N_0	Meaning
0	0	16-bit narrow-width value
0	1	34-bit (>16 bits) narrow-width value
1	0	Reserved
1	1	Regular value (>34 bits)

3. Asymmetrically banked value-aware register files (AB-VARF)

3.1. Value-aware register files (VARF)

Exploiting narrow-width values for energy reduction, a partitioned value-aware register file (P-VARF) was proposed in [11] to capture small 8-bit narrow values. Different from the asymmetrically ported register files [1], P-VARF here specifically partitions its wordlines/bitlines into three partitions: lower 16-bit, middle 18-bit, and upper 30-bit partitions. As shown in Fig. 4a, the access to the 18-bit and 30-bit partitions is controlled by the narrowness flag N_1N_0 associated with that register value. For example, the wordline signal from the decoder is gated before passing to the upper 30-bit segments when a 34-bit narrow value is accessed. Once the value is read out from the register file, sign extension will be automatically performed based on its narrowness flag. Thus, by serving the majority of register file accesses from the smaller 16-bit partition or 16-bit and 18-bit partitions (due to narrow values), great energy savings can be achieved. Notice that the sign extension logic exists in most datapath implementations and in many cases it is a part of the integer functional units since many instructions require automatic sign extension for their source operands before the actual ALU operation. Thus, sign extending the narrow-width register value will not cause any timing problem. However, the gated logic,

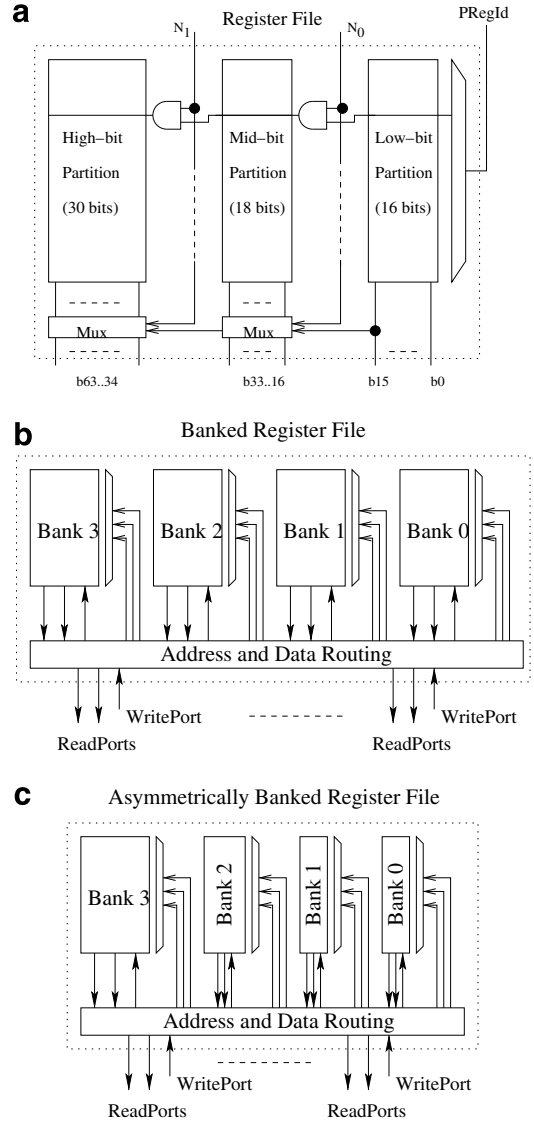


Fig. 4. A microarchitecture-level comparison among (a) a partitioned value-aware register file, (b) a conventional banked register file, and (c) the AB-VARF register file.

the storage and access of narrowness flags, and the multiplexer (for bypass or sign extension) in the data output path of 18-bit and 30-bit partitions may introduce additional timing, energy, and area overhead. Notice that we introduce P-VARF here only for reference purpose.

3.2. Asymmetric banking

P-VARF register files, on one side, could be a energy-efficient design compared to conventional ones. However, P-VARF does not address the increasing access latency or wire delay of large register files. In contrast, banked register files have been investigated for their efficiency in area, energy consumption, and access latency [38,3,27,37]. In this paper, we propose an asymmetrically banked VARF register file (AB-VARF) to exploit the majority of narrow values for further energy and performance optimization.

Different from the conventional banked register files, our AB-VARF consists of banks with word sizes of 16 bits, 34 bits, and 64 bits. Due to the reduced number of bitlines thus also the wordline length, the access to 16-bit or 34-bit banks incurs much less energy consumption than to 64-bit banks. As narrow-width values dominate the overall register values, the majority of register file accesses will be served by 16-bit and 34-bit banks. Thus the overall energy efficiency in AB-VARF register files is further improved over the conventional designs. Obviously, AB-VARF also occupies less area. Fig. 4 also compares a conventional banked register file and our AB-VARF register file, which has two 16-bit banks, one 34-bit bank, and one 64-bit bank. Each bank still has the same number of register entries. Since each physical register can only be in one bank, the physical register id itself implicitly indicates the width of the register value.

To support our asymmetrically banked VARF, the register renaming logic is slightly modified to maintain three free lists: one for 16-bit registers, one for 34-bit registers, and one for 64-bit registers. If the instruction is predicted to produce a 16-bit narrow value, its destination register will be renamed to a physical register from the 16-bit register free list. Similarly, if the predicted result value is a 34-bit narrow value, a 34-bit register is used to rename the destination register. Otherwise, the rename logic allocates a register from the 64-bit free list for a predicted full-size register value. To avoid any extra pipeline stalls due to unavailable free narrow registers, oversized register renaming is allowed in AB-VARF. The implementation of destination register renaming logic searches all three free lists simultaneously and uses the width prediction to select the best-matched free register. For example, if the result is predicted as a 16-bit value and the 16-bit free list is empty, then a register from the 34-bit free list will be allocated for the destination register instead. If the 34-bit free list is also empty, the last resort is to allocate a free 64-bit register. Otherwise, the renaming stage has to be stalled till some register is released and has enough bits to accommodate the predicted value. However, downsized renaming is strictly prohibited. For example, a destination register with a predicted 64-bit value shall never be renamed to a 34-bit or 16-bit free register. Notice that no change is required for source operand renaming or the register renaming table in the enhanced renaming logic.

3.3. Avoiding registerfile port conflict

As in conventional banked designs, bank conflict presents a major performance bottleneck in AB-VARF register files since each bank has significantly reduced read/write ports. Notice that in banked register files, additional address routing and data routing are required. To reduce the readport conflict in AB-VARF, we adopt the bypass hint scheme [37] that dramatically reduces the read requests to the register file. To support bypass hint scheme, each source operand of the instructions in the issue queue has

an additional bypass hint bit, which is set by the tag match operation during the wakeup stage. If the instruction is also selected for issue in the same cycle as the hint bit is set, the operand value will be from the bypass network instead of the register file. Thus the register file access can be inhibited for this operand. Otherwise, the bypass hint bit in an unselected instruction will be reset to zero during the next cycle due to the tag mismatch. Thus, the instruction is forced to read the register file for the required source operand value. Notice that this bypass hint scheme is non-speculative, which is different from the speculative version in [27]. A load miss will result in the replay of all issued dependent instructions (on the load value) thus shall not affect the bypass hint scheme itself.

With bypass hint scheme in place, the majority of readport conflict has been eliminated [27,37]. However, writeport conflict remains the major performance issue. We borrow the register port scheduling scheme from [3] to minimize both read and write port conflicts in banks. A similar idea was initially proposed in [32] to schedule the shared result bus in processor datapath using a result shift register. Register ports of each bank are treated as additional resources during instruction scheduling. In addition to the available functional unit, a ready instruction must also obtain register read ports (if reading any source operand from the register file) and the write port (if producing any result) before it can be selected for issue. If multiple instructions are competing for the same register file port, an oldest-first policy is employed to resolve the contention. For readport scheduling, only the current instructions generating request signals (for selection) need to be considered for possible bank conflict. However, writeport scheduling is more complicated than readport scheduling due to the various execution latencies of different operations.

We propose to maintain a scheduling bit vector for each write port in each bank. Fig. 5 presents an alternative implementation of the writeport scheduler for the AB-VARF register file. The size of the vector is set to accommodate the latency of the longest operation, e.g., 24 cycles for floating-point square root operation in the simulated microprocessor. The scheduler maintains a global (shared) pointer and a local scheduling pointer for each vector of each bank. Given the 32-bit vector size (to cover the longest 24-cycle operation), we employ a global 5:32 decoder that takes as its input the output of a 5-bit global counter that is ticked by the clock signal. The outputs of the decoder are connected to all vectors, one output to all vector bits in the corresponding column, as shown in the figure. At any cycle, only one output of the decoder will be active and act as the global pointer. This global pointer points to the bit position corresponding to the current cycle for all vectors and reset the pointed bits to 0s at the end of the writeback stage before it advances to the next position in the vectors as clock ticks the global counter every cycle. With the global pointer identified for the current cycle, a local scheduling pointer is required for each vector to locate the bit position corresponding to the cycle

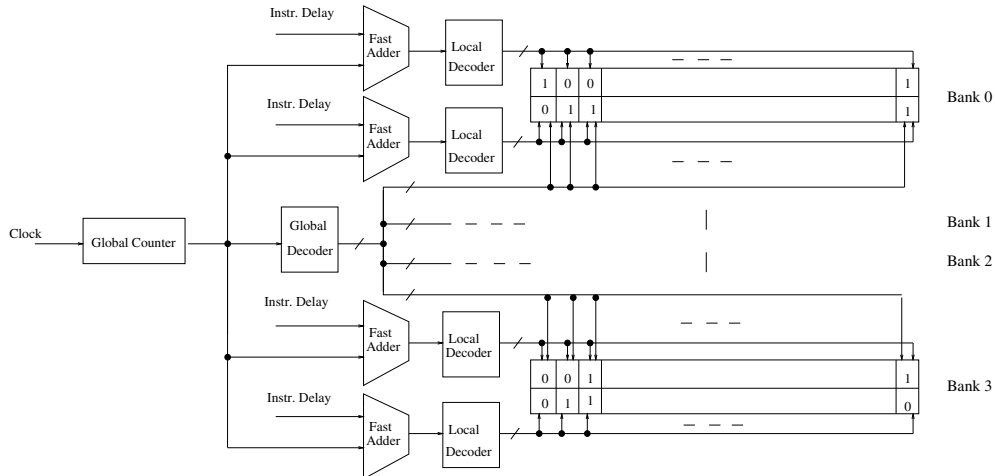


Fig. 5. A writeport scheduler for reducing writeport conflicts at banks in the AB-VARF register file.

(determined by the operation latency) when the requesting instruction is going to write its result to the target bank. A fast small 5-bit adder is used to add the operation latency as the offset to the global counter (as current cycle) and a local 5:32 decoder takes the sum from the adder to generate the local scheduling pointer, i.e., the active output of the local decoder. Notice that both the global pointer and local scheduling pointers work in a circular manner. Then, the writeport scheduling is quite straightforward. At the instruction selection stage, the vector bits pointed by the local scheduling pointers are checked for writeport availability. If the current bit value is 0, i.e., the writeport is available, the bit value is then set to 1 indicating the writeport being scheduled. If the current value is 1, the writeport has already been scheduled for a different instruction and the current requesting instruction cannot be selected due to unavailable writeport.

With our proposed writeport scheduling in place, any conflict among deterministic instructions (in terms of latency) can be removed. However, a load instruction may still cause bank writeport conflict since the actual latency of the load cannot be determined during instruction scheduling. A load instruction missing the L1 cache will be immediately scheduled (for writeport) as it will hit the L2 cache. If the load also misses higher level cache hierarchies (i.e., the latency exceeds the schedule vector size) and its registerfile write causes a port conflict, this load will have priority to proceed and simply stall all other scheduled writes for one cycle.

3.4. Value width prediction

In the AB-VARFs, a destination register in a decoded instruction is renamed to one of the three-sized registers: 16 bits, 34 bits, or 64 bits, based on the width prediction of the result. Since the major objective of AB-VARF is to reduce the energy consumption in banked register files, the width predictor itself should be a light-weighted one

in terms of energy consumption. In the mean time, it should provide sufficient accuracy to avoid considerable performance loss due to the misprediction. As shown in [22,12], a last value predictor is quite efficient in meeting these two requirements. We employ such a simple width predictor at the fetch stage. It predicts whether the produced result of the instruction is a 16-bit narrow value, a 34-bit value, or a 64-bit value. The predictor can be implemented as similar to a simple bimodal branch predictor, where the 2-bit counter indicates the value width of the last instance of the instruction. The default and initial value of counters is 3 (11), which predict a 64-bit value. If the width of last value is 34 bits, the corresponding counter is reduced to 1 (01). A 16-bit value resets the counter to 0 (00) and a 64-bit value changes the counter back to 3 (11).

The width prediction is verified against the width-detection result from the functional unit and the predictor is updated correspondingly. A width misprediction, if causing a situation that the renaming register is not able to accommodate the result value, will have to invoke the recovery mechanism, which flushes instructions following that mispredicted one and restores the renaming table by walking through the active list (or ROB). Since over-prediction does not have this bit space problem, it can still be treated as correct prediction.

4. Evaluation

4.1. Experimental setup

We derive our simulators from SimpleScalar V3.0 [6] to model a contemporary high-performance eight-issue microprocessor similar to Alpha 21464 [29]. Table 2 shows the detailed configuration for the simulated microprocessor. For experimental evaluation, we use SPEC CINT2000 suite compiled for Alpha instruction set architecture with “peak” tuning. We use the reference input sets for this study. Each benchmark is first fast-forwarded to its early

Table 2
Parameters for the simulated processor

	Processor core
Int/FP issue queue	128 entries
Load/store queue	256 entries
Active list (ACL)	512 entries
Int/FP register file	512/512 registers
Datapath width	8 instructions per cycle
Function units	8 IALU, 2 IMULT/IDIV, 4 FALU 2 FMULT/FDIV/FSQRT, 4 MemPorts
Branch predictor	Tournament predictor with a 4K meta-table A 4K bimodal predictor table, and a 2-level Gshare predictor with 12-bit history 2048-entry, 2-way BTB, and 32-entry RAS
Width predictor	2K bimodal predictor
<i>Memory hierarchy</i>	
L1 I/DCache	64KB, 2 ways, 64B blocks, 2 cycle latency
L2 UCACHE	4MB, 8 ways, 128B blocks, 12 cycle latency
Memory	225 cycles first chunk, 12 cycles rest
TLB	Fully assoc., 128 entries
<i>Power parameters</i>	
Technology	70 nm
Supply voltage	0.9 V
Clock frequency	5.6 Ghz

single simulation point (*gap* uses the standard single simulation point) specified by SimPoint [31]. Then, we simulate the next 100 million instructions in details.

4.2. Registerfile energy models

To model both the dynamic and leakage energy of different register files, we utilize Cacti 4.2 [34] to derive the energy numbers of major components within the register file. As shown in Eq. (1), the dynamic energy of accessing a conventional register file *Base* comes from five contributors: decoder (E_{Dec}), wordline (E_{WL}), bitline (E_{BL}), sense amplifier (E_{SA}), data output drive (E_{DR}).

$$E_{Base} = E_{Dec} + E_{WL} + E_{BL} + E_{SA} + E_{DR}. \quad (1)$$

In P-VARF register file, the per access dynamic energy consumption is determined by which partition(s) being accessed. The dynamic energy consumption of each partition (P_{part}) is scaled from the default 64-bit register width, as illustrated in Eq. (2). To simplify the energy model, we omit the energy overheads of partition control logic and output multiplexers, which shall be moderately small.

$$E_{P-VARF} = E_{Dec} + \sum E_{Part}, \quad (2)$$

$$E_{Part} = Part_width/64 * (E_{WL} + E_{BL} + E_{SA} + E_{DR}).$$

Banked register file reduces dynamic energy consumption by activating a single bank for each access. Notice that each bank has its own decoder. However, additional global routing is required in banked designs. The energy model is given by Eq. (3).

$$E_{BK-RF} = E_{Dec} + E_{Data} + E_{Routing}, \quad (3)$$

$$E_{Data} = E_{WL} + E_{BL} + E_{SA} + E_{DR}.$$

Our AB-VARF attacks the dynamic energy consumption in the bank data array (E_{Data}) using narrow banks, as shown in Eq. (4). The additional energy overhead of width prediction (E_{WPred}) is also included when evaluating AB-VARF's energy efficiency.

$$E_{AB-VARF} = E_{Dec} + BK_width/64 * E_{Data} + E_{Routing} + E_{WPred}. \quad (4)$$

We also derive the leakage power model for different register files from the Cacti 4.2 [34] by using a similar way as modeling the dynamic energy. In Cacti 4.2 [34], the leakage power of a monolithic *Base* register file is also contributed by the five major parts: decoder (P_{Dec}), wordline (P_{WL}), bitline (P_{BL}), sense amplifier (P_{SA}), data output drive (P_{DR}), as given in Eq. (5). Notice that the partition control circuit in P-VARF also introduces additional leakage power consumption. To compensate this leakage source, the model simply assumes that P-VARF has a similar leakage power number as *Base*.

$$P_{Base} = P_{Dec} + P_{WL} + P_{BL} + P_{SA} + P_{DR}. \quad (5)$$

Similarly, the leakage power of a single bank in a banked register file can be derived by Eq. (6), and the total leakage of the banked register file is given by Eq. (7), where n is the number of banks.

$$P_{Bank} = P_{Dec} + P_{Data} + P_{Routing}, \quad (6)$$

$$P_{Data} = P_{WL} + P_{BL} + P_{SA} + P_{DR}.$$

$$P_{BK-RF} = P_{Bank} * n \quad (7)$$

The leakage power of the narrow-width banks in AB-VARF is scaled from the one in Eq. (6) according to the bank width, BK_{width}^i . The overall leakage of a AB-VARF register file can be calculated as in Eq. (8), which also includes the leakage power in the width predictor P_{WPred} .

$$P_{AB-VARF} = \sum_{i=0}^{n-1} \{P_{Dec} + BK_{width}^i/64 * P_{Data} + P_{Routing}\} + P_{WPred}. \quad (8)$$

Table 3 lists the dynamic energy numbers for major components in a conventional monolithic register file with 16 readports and 8 writeports, *Base* (1B/16R/8W), a banked (four banks) register file with 4 readports and 2 writeports per bank, *BK-RF* (4B/4R/2W), and a banked (four banks) register file with four readports and four writeports per bank, *BK-RF* (4B/4R/4W), as well as the leakage power and access latency of these register files. The register file size is 512 entries as given in Table 2. The listed numbers clearly indicate that a banked design not only significantly reduces the energy consumption in the register file but also dramatically increases the access speed over a monolithic design. This feature of a banked

Table 3

Energy consumption of major components in three register files, Base (1B/16R/8W), BK-RF (4B/4R/2W), and Bk-RF (4B/4R/4W)

Dynamic energy (nJ)/speed (ns)	Base (1B/16R/8W)	BK-RF (4B/4R/2W)	Bk-RF (4B/4R/4W)
Decode (E_{Dec})	0.00178219	0.000180352	0.000250626
Wordline (E_{WL})	0.000342064	0.0000664371	0.0000843697
Bitline read (E_{BL})	0.0108764	0.000672728	0.000771746
Bitline write (E_{BL})	0.00362252	0.000349255	0.000416768
Sense amplifier (E_{SA})	0.000641605	0.000153092	0.000193168
Data output driver (E_{DR})	0.00612099	0.00252377	0.0033892
Global routing ($E_{Routing}$)	–	0.00244229	0.00249005
Total dynamic read energy (nJ)	0.019763249	0.0060386691	0.0071791597
Total dynamic write energy (nJ)	0.005746774	0.0005960441	0.0007517637
Total leakage power all bank (W)	0.36707159	0.128571725	0.180504201
Access latency (ns)	2.3415	0.481066	0.529453

Numbers derived from Cacti 4.2 [34].

register file is extremely important to the design of future high-performance wide-issue microprocessors.

4.3. Experimental results

Our AB-VARF register files exploits the large percentage of narrow-width register values for energy-efficient register file designs. Fig. 2 shows that, on the average, 48% of the generated register values have a width no more than 16 bits, and additional 49% can be represented by no more than 34 bits. Result values requiring more than 34 bits only present 3% of the total produced register values. The width distribution of source register values exhibits the similar trend as of the result values, where 16-bit and 34-bit values account for 34% and 63% of the total read values, respectively. The remaining 3% covers read values with a width more than 34 bits. These percentages imply great opportunities for energy optimization in the register file.

In this section, we evaluate the energy efficiency and performance of asymmetrically banked VARF register files (AB-VARF). For evaluation purpose, we also modeled a conventional monolithic register file with 16 readports and 8 writeports as the base design (Base), the base design with bypass hint scheme (Bypass), a P-VARF register file (P-VARF), and a base banked register file (BK-RF). Notice that both P-VARF and BK-RF register files employ the bypass hint scheme. Due to the banking overhead, we only consider banked register files with four banks for this experimental evaluation. Each bank by default has four readports and two writeports such that the aggregate port number is same as in a conventional register file. Notice that the base BK-RF utilizes a simple stall-one cycle scheme on bank writeport conflict.

4.3.1. Energy efficiency of AB-VARF register files

To integrate the performance impact, we present the energy numbers instead of power for different register file designs. Fig. 6 shows that the bypass hint scheme Bypass alone reduces the energy consumption by 23.8% over a conventional register file Base, and P-VARF further

improves this reduction to 37.5%. In comparison, the energy-efficient banked register file BK-RF achieves a dramatic 74.6% energy reduction. However, the performance loss is also non-trivial, a 11.8% on the average as shown in Fig. 8, due to the bank port conflicts.

For our proposed AB-VARF design, we evaluated two banking configurations, AB-VARF-121 with 1 16-bit bank, 2 34-bit banks, and 1 64-bit bank, and AB-VARF-211 with 2 16-bit banks, 1 34-bit bank, and 1-64-bit bank. Our results show that AB-VARF-211 achieves better energy/performance efficiency, which matches the register value width distribution in Fig. 2. For the following discussion, we focus on the AB-VARF-211 bank configuration (as AB-VARF). Different from the P-VARF and BK-RF, additional supports of register file read/write port scheduling and width prediction are required in AB-VARF. On the average, our AB-VARF achieves an astonishing energy reduction of 81.9% over Base and of 28.6% over BK-RF, as shown in Fig. 6. Even with its writeport number doubled to minimize writeport conflict (at both scheduling and writeback stages), AB-VARF-4w still maintains its energy efficiency, a reduction of 78.4% energy consumption over the Base register file. Notice that this energy number

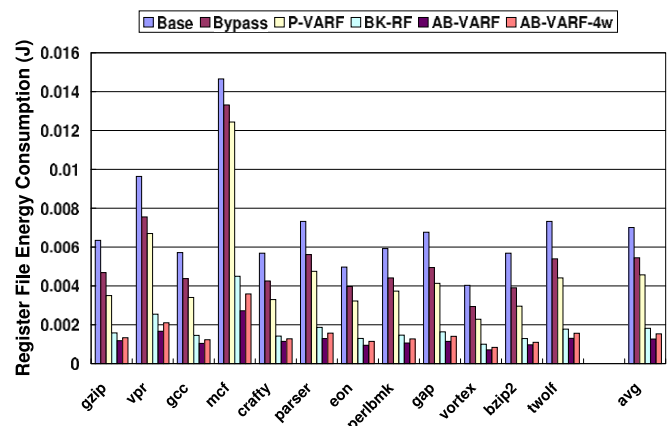


Fig. 6. A comparison of the overall energy consumption (including both dynamic and leakage energy) in different register files.

includes the energy consumption in the width predictor. Further, Fig. 7 provides a dynamic-leakage energy breakdown for the evaluated register files, an average across SPEC CINT2000 benchmarks. The leakage energy accounts for 51% of the total energy consumption in the Base register file. This number increases to 78% and 76% in P-VARF and BK-VARF register file, respectively. With AB-VARF, this percentage is effectively reduced to 55% due to narrow-width banks, leading to improved overall energy savings.

4.3.2. Performance of AB-VARF register files

To further evaluate our design choices in AB-VARF register files, we compare in Fig. 8 the performance of AB-VARFs with different assumptions. AB-VARF-Stall uses a simple pipeline-stall solution for bank port conflict just as in BK-RF. AB-VARF-Perf scheme implements readport scheduling as part of the instruction selection and assumes perfect value width prediction and perfect register file write, i.e., there is no write port conflict. AB-VARF-WS uses write port scheduling to avoid write conflict

and AB-VARF-WS-WPred implements a bimodal (2048-entry table) width predictor.

Fig. 8 shows that (1) simply stalling the pipeline on bank port conflict in AB-VARFs incurs significant performance degradation, an average of 17.3%; (2) readport scheduling has negligible performance impact mainly due to the dramatically reduced read requests by the bypass hint scheme; (3) writeport scheduling has successfully recovered 62.4% of the performance loss due to port conflict, achieving a performance within 6.5% of the perfect scheme; (4) a simple bimodal width predictor achieves a very close performance to the perfect predictor. On the average, AB-VARF-WS-WPred trades a 6.6% performance loss to an ideal 1-cycle conventional register file. By increasing the number of writeports per bank to 4, AB-VARF-WS-WPred-4w minimizes the instruction selection delay due to unavailable writeports, consequently reducing the performance loss to 0.7%. In the meantime, the energy efficiency of AB-VARF is still fully exploited, as shown in Fig. 6.

To study the performance sensitivity of AB-VARF to its size, register files of a small size, 128 entries, are also simulated. Results presented in Fig. 9 show that the performance loss (to the ideal 1-cycle Base register file) increases to 7.0% (from 6.6%) for AB-VARF-WS-WPred and to 3.8% (from 0.7%) for AB-VARF-WS-WPred-4w when compared to the 512-entry register files. This can be explained by the increased utilization in smaller register files and the increased pressure on wider banks, i.e., 34-bit/64-bit banks.

4.3.3. Width prediction distribution and accuracy

The performance results in Fig. 8 show that the simulated microprocessor with a simple last value width predictor only incurs a negligible performance loss (within 0.1%) to the one with a perfect width predictor. To further understand the performance impact of width predictors on microprocessors employing AB-VARF register files, we present the width prediction distribution and accuracy in

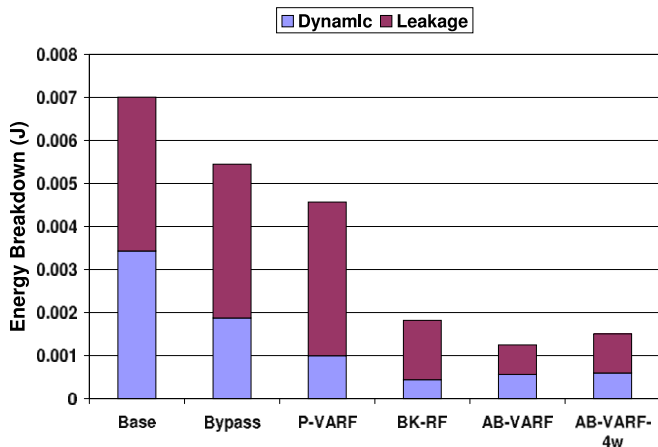


Fig. 7. A dynamic-leakage energy breakdown for evaluated register files, an average across SPEC CINT2000 benchmarks.

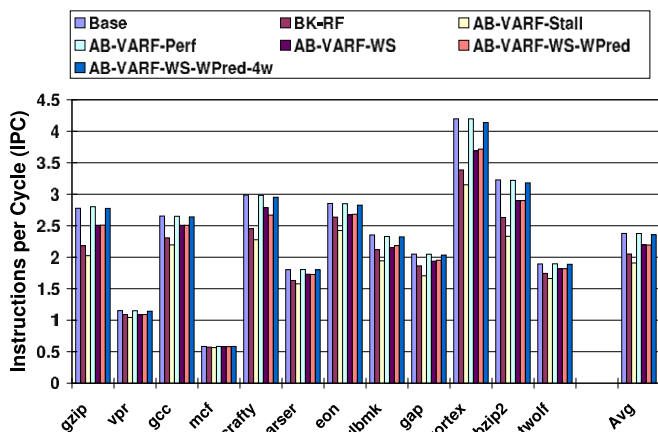


Fig. 8. Performance comparison among Base, Bk-RF, and AB-VARFs.

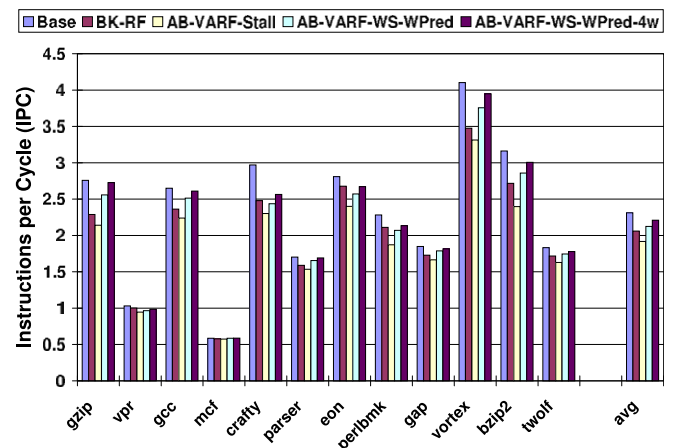


Fig. 9. Performance comparison among Base, Bk-RF, and AB-VARFs of a small size, 128 registers.

Fig. 10. Notice that the width predictor predicts the width of the result value produced by an instruction to be 16-bit, 34-bit, or 64-bit. The prediction of 34-bit means that the result value cannot be represented by 16 bits but requires no more than 34 bits. Fig. 10a shows that on the average, the width predictor predicts 48.5% of the result value to be 16-bit, 48.4% to be 34-bit, and 3.1% to be 64-bit, which closely matches the written register value width distribution presented in Fig. 2a in Section 2.2. A breakdown of the accuracy of these three predictions is given in Fig. 10b. The 16-bit prediction accuracy (Pred-16b) on the average is 92.6%. A misprediction in this category will allocate a 16-bit register for a value requiring more than 16 bits, resulting in misprediction recovery at the writeback stage. The prediction accuracy of 34-bit values (Pred-34b) is 75.5%, an average. There are two types of misprediction in this category, a 16-bit value predicted as 34-bit and a regular 64-bit value predicted as 34-bit. Notice that the first “misprediction” is actually an over-prediction and allocates a larger 34-bit register for a small 16-bit value, which will not cause any bit space problem. If we include this as

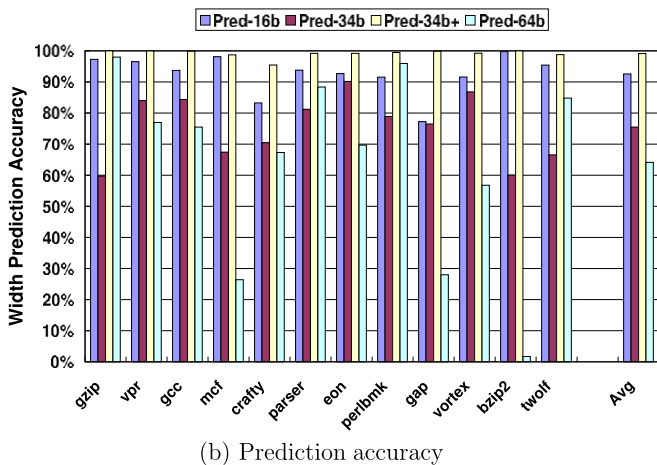
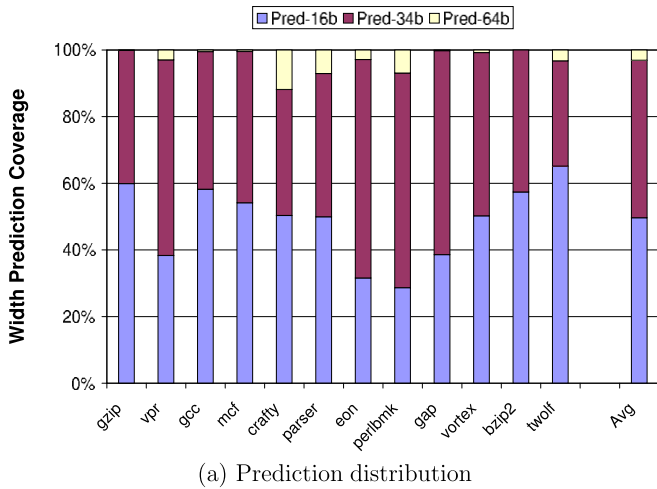


Fig. 10. (a) Width prediction distribution, and (b) width prediction accuracy of the last value width predictor employed to support the AB-VARF register file.

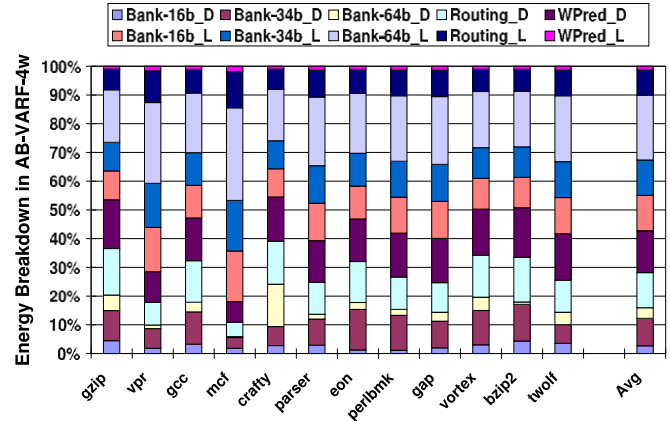


Fig. 11. A breakdown of the energy consumption in AB-VARF-4w.

correct prediction, the prediction accuracy (Pred-34b+) increases to 99.1%. The second type of 34-bit misprediction (0.9%) will cause the similar problem as the 16-bit misprediction. Even though the presented 64-bit prediction (Pred-64b) accuracy is only about 64.1%, all mispredictions in this category are over-predictions. In other words, we can always consider a 64-bit prediction correct. This high prediction accuracy (including over-predictions) explains the superior performance delivered by the simple width predictor.

4.3.4. Energy-delay product (EDP)

Finally, an energy breakdown of AB-VARF-4w is given in Fig. 11, where the leakage energy of the 64-bit bank Bank-64b_L contributes 22.5% to the total energy consumption, followed by the dynamic energy of the width predictor WPred_D (14.6%), the leakage energy of 16-bit banks Bank-16b_L (12.3%) and the 34-bit bank Bank-34b_L (12.3%), and the dynamic energy of the global routing Routing_D (12.2%). This result also indicates that the value width predictor introduces a major energy overhead in AB-VARFs and may require further optimizations such

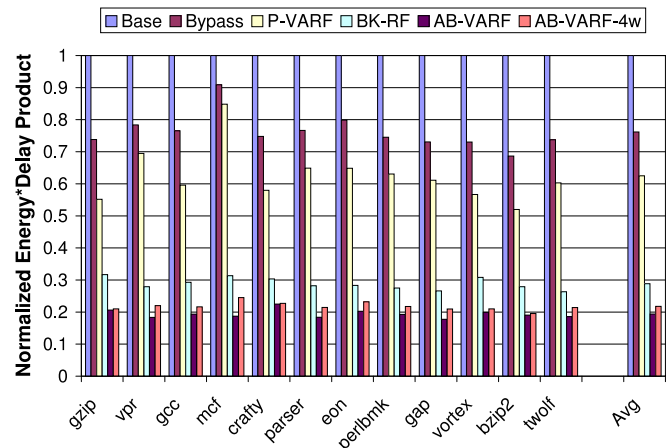


Fig. 12. Normalized energy * delay product (EDP) of different register files w.r.t. Base.

as compiler-assisted prediction schemes in order to achieve ultra low-energy AB-VARF designs.

To summarize the energy efficiency and performance benefit of our proposed AB-VARF, we present in Fig. 12 the normalized energy * delay product (EDP) with respect to the Base register file. By exploiting narrow-width value and asymmetric banking design, AB-VARF achieves the best EDP, a significant reduction of 81% over Base, 74% over Bypass, 68% over P-VARF, and 33% over BK-RF, on the average. This further confirms us that our AB-VARF can be an effective solution to high-performance low-energy register file designs in high-performance microprocessors.

5. Conclusions and future work

We proposed in this paper to exploit the prevailing narrow-width register values for reducing the energy consumption in register files. Our asymmetrically banked VARF (AB-VARF) improves the energy efficiency of banked register files by asymmetrically sizing the bit-width of banks such that the majority narrow values are stored in (retrieved from) the small and low-energy narrow-width banks. The experimental results have indicated that our AB-VARF register files can be suitable designs in high-performance energy-efficient microprocessors. For future work, we would like to investigate the area saving in AB-VARFs, compiler assistance in optimizing the energy consumption for the width predictor, and the impact of more asymmetric features, such as bank size and numbers of ports per bank, on the performance and energy of AB-VARF designs.

References

- [1] A. Aggarwal, M. Franklin, Energy efficient asymmetrically ported register files, in: Proceedings of ICCD'03, 2003.
- [2] S. Balakrishnan, G. Sohi, Exploiting value locality in physical register files, in: Proceedings of Micro-36, 2003.
- [3] R. Balasubramonian, S. Dwarkadas, D.H. Albonesi, Reducing the complexity of the register file in dynamic superscalar processors, in: Proceedings of the Micro-34, 2001.
- [4] E. Borch, E. Tune, S. Manne, J. Emer, Loose loops sink chips, in: Proceedings of the HPCA-8, 2002.
- [5] D. Brooks, M. Martonosi, Dynamically exploiting narrow width operands to improve processor power and performance, in: Proceedings of the HPCA-5, 1999.
- [6] D. Burger, T.M. Austin, The simplescalar tool set, version 2.0, Tech. Rep. 1342, Computer Sciences Department, University of Wisconsin, 1997.
- [7] J.A. Butts, G.S. Sohi, Use-based register caching with decoupled indexing, in: Proceedings of the 31st Annual International Symposium on Computer Architecture, 2004.
- [8] R. Canal, J.-M. Parcerisa, A. Gonzalez, Dynamic cluster assignment mechanisms, in: Proceedings of the Sixth International Symposium on High Performance Computer Architecture, 2000.
- [9] A. Capitanio, N. Dutt, A. Nicolau, Partitioned register files for vlvis: a preliminary analysis of tradeoffs, in: Proceedings of the 25th Annual International Symposium on Microarchitecture, 1992.
- [10] J.L. Cruz, A. Gonzalez, M. Valero, N.P. Topham, Multiple-banked register file architectures, in: Proceedings of the 27th Annual International Symposium on Computer Architecture, 2000.
- [11] O. Ergin, Exploiting narrow values for energy efficiency in the register files of superscalar microprocessors, in: 16th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS'06), Lecture Notes in Computer Science (LNCS), Springer-Verlag, 2006.
- [12] O. Ergin, et al., Register packing: Exploiting narrow-width operands for reducing register file pressure, in: Proceedings of the Micro-37, Portland, OR, 2004.
- [13] K.I. Farkas, P. Chow, N.P. Jouppi, Z. Vranesic, The multicluster architecture: reducing cycle time through partitioning, in: Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture, 1997.
- [14] A. Gonzalez, J. Gonzalez, M. Valero, Virtual-physical registers, in: Proceedings of the Fourth International Symposium on High-Performance Computer Architecture, 1998.
- [15] G. Gonzalez, A. Cristal, D. Ortega, A. Veidenbaum, M. Valero, A content aware integer register file organization, in: Proceedings of the 31st Annual International Symposium on Computer Architecture, 2004.
- [16] J. Hu, S. Wang, S.G. Ziavras, In-register duplication: Exploiting narrow-width value for improving register file reliability, in: Proceedings of the International Conference on Dependable Systems and Networks (DSN-2006), Philadelphia, PA, 2006.
- [17] S. Jourdan, R. Ronen, M. Bekerman, B. Shomar, A. Yoaz, A novel renaming scheme to exploit value temporal locality through physical register reuse and unification, in: Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture, 1998.
- [18] R.E. Kessler, The alpha 21264 microprocessor, *IEEE Micro* 19 (2) (1999) 24–36.
- [19] M. Kondo, H. Nakamura, A small, fast and low-power register file by bit-partitioning, in: Proceedings of the 11th International Symposium on High-Performance Computer Architecture, 2005.
- [20] C. Liem, T. May, P. Paulin, Register assignment through resource classification for asip microcode generation, in: Proceedings of the 1994 IEEE/ACM International Conference on Computer-Aided Design (ICCAD94), 1994.
- [21] M.H. Lipasti, B.R. Mestan, E. Gunadi, Physical register inlining, in: Proceedings of the 31st Annual International Symposium on Computer Architecture, 2004.
- [22] G.H. Loh, Exploiting data-width locality to increase superscalar execution bandwidth, in: Proceedings of the 35th International Symposium on Microarchitecture (MICRO), 2002.
- [23] L.A. Lozano, G.R. Gao, Exploiting short-lived variables in superscalar processors, in: Proceedings of the 28th Annual International Symposium on Microarchitecture, 1995.
- [24] D.R. Lutz, D.N. Jayasimha, Early zero detection, in: Proceedings of the 1996 International Conference on Computer Design (ICCD'96), 1996.
- [25] J.F. Martinez, J. Renau, M.C. Huang, M. Prvulovic, J. Torrellas, Cherry: checkpointed early resource recycling in out-of-order microprocessors, in: Proceedings of the Micro-35, 2002.
- [26] T. Monreal, A. Gonzalez, M. Valero, J. Gonzalez, V. Vinals, Delaying physical register allocation through virtual-physical registers, in: Proceedings of the Micro-32, 1999.
- [27] I. Park, M. Powell, T. Vijaykumar, Reducing register ports for higher speed and lower energy, in: Proceedings of the International Symposium on Microarchitecture, 2002.
- [28] M. Postiff, D. Greene, S. Raasch, T. Mudge, Integrating superscalar processor components to implement register caching, in: Proceedings of the 15th International Conference on Supercomputing, 2001.
- [29] R.P. Preston, et al., Design of an 8-issue superscalar risc microprocessor with simultaneous multithreading, in: Proceedings of the IEEE International Solid-State Circuits Conference, 2002.
- [30] R.M. Russell, The cray-1 computer system, *Commun. ACM* 21 (1) (1978) 63–72.
- [31] T. Sherwood, et al., Automatically characterizing large scale program behavior, in: Proceedings of the ASPLOS X, 2002.

- [32] J. Smith, A. Pleszkun, Implementing precise interrupts in pipelined processors, *IEEE Trans. Comput.* 37 (5) (1988) 562–573.
- [33] J.A. Swensen, Y.N. Patt, Hierarchical registers for scientific computers, in: *Proceedings of the ICS'88*, 1988.
- [34] D. Tarjan, S. Thoziyoor, N.P. Jouppi, Cacti 4.0, Tech. Rep., Compaq Western Research Lab, 2006.
- [35] L. Tran, N. Nelson, F. Ngai, S. Dropsho, M. Huang, Dynamically reducing pressure on the physical register file through simple register sharing, in: *Proceedings of the ISPASS'04*, 2004.
- [36] J. Tseng, K. Asanovic, Energy-efficient register access, in: *Proceedings of the 13th Symposium on Integrated Circuits and Systems Design*, 2000.
- [37] J. Tseng, K. Asanovic, Banked multiported register files for high-frequency superscalar microprocessors, in: *30th International Symposium on Computer Architecture (ISCA-30)*, San Diego, CA, 2003.
- [38] S. Wallace, N. Bagherzadeh, A scalable register file architecture for dynamically scheduled processors, in: *Proceedings of the 1996 Conference on Parallel Architectures and Compilation Techniques*, 1996.
- [39] K.C. Yager, The MIPS R10000 superscalar microprocessor, *IEEE Micro* 16 (2) (1996) 28–40.
- [40] R. Yung, N. Wilhelm, Caching processor general registers, in: *Proceedings of the International Conference on Circuits Design*, 1995.
- [41] J. Zalamea, J. Llosa, E. Ayguade, M. Valero, Two-level hierarchical register file organization for vliw processors, in: *Proceedings of the Micro-33*, 2000.