

# Asymmetrically Banked Value-Aware Register Files

Shuai Wang, Hongyan Yang, Jie Hu, and Sotirios G. Ziavras  
Department of Electrical and Computer Engineering  
New Jersey Institute of Technology  
Newark, NJ 07102  
{sw63, hy34, jhu, zivras}@njit.edu

## Abstract

*Designing high-performance low-power 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 power-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 suites shows that AB-VARF reduces the energy consumption by 92.6% over a conventional register file, on the average, at the cost of a 6.6% performance loss to an ideal 1-cycle monolithic register file.*

## 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 power 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]. 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 micro-

processor has 512 physical registers with an area of more than five times of the level one data cache [29]. Further, the power consumption of the register file can be up to 20% of the total chip power [35].

Many techniques have been proposed to improve the scalability, in terms of performance, power, and area, of the register file in designing high-performance power-efficient microprocessors. These techniques include hierarchical register files [30][33][23][40][3], register caching [39][10][28][4][7], and clustered register files [18][9][13][8]. Other proposals try to reduce the required register file size for reduced access latency and power consumption by delaying the physical register allocation [14][26], sharing physical registers to exploit value locality [17][25][2][15][34], or exploiting narrow-width values [1][21][19][12]. Alternative approaches employ banked register files with reduced read/write ports [37][3][36] or/and the bypass-hint scheme to reduce the read requests thus the number of read ports [27][36] for high performance and low power. 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 power, performance and reliability optimizations [5][22][1][21][19][12][11][16]. In this paper, we propose to utilize a banking scheme [37][27][36] for designing asymmetrically-banked value-aware register files (AB-VARF) to reduce the power 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 power 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 [36] 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 resources.

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 to a 7.4% of a conventional monolithic design, which presents an additional saving of 25.6% over a partitioned value-aware register file. 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 power efficiency still fully exploited.

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

## 2 Basics on 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 [38] style register renaming scheme, where the architectural and physical register files are combined.

At the 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 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 desti-

$N_1$	$N_0$	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)

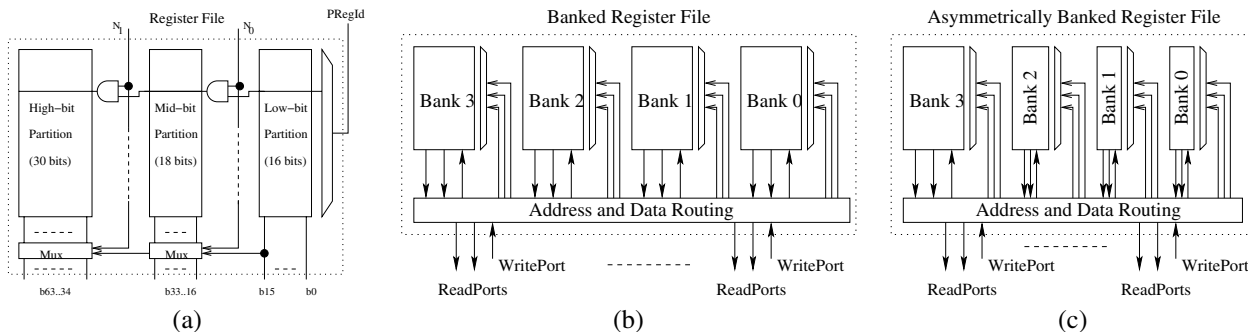
**Table 1. The meanings of narrowness flag bits  $N_1N_0$ .**

nation 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 power consumption and also results in slow access.

### 2.2 Narrow-Width Register Value and its Detection

The data-width ( $W$ ) of a given register value is determined as follows:  $W = 64 - (LS - 1)$ , 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:  $V = b_{W-1}b_{W-2}...b_0$ , where  $b_{W-1}$  is the sign bit,  $b_{W-1} \oplus b_{W-2} = 1$ , and  $W \leq 64$ . The study on the data-width distribution of integer register values for SPEC CINT2000 benchmarks (Alpha binaries) in [5][21][16] has shown that on the average 1) around 46% of the generated register values have a data-width no more than 16 bits, 2) an additional 8% of the values can be captured by a representation of 32 bits, and 3) there is a huge jump in the distribution from 33 bits to 34 bits, showing that 94% of the produced register values can be represented by no more than 34 bits. Consequently, 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.

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 30/48 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.



**Figure 1. A microarchitecture-level comparison among a partitioned value-aware register file (a), a conventional banked register file (b), and the AB-VARF register file (c).**

### 3 Asymmetrically-Banked Value-Aware Register Files (AB-VARF)

#### 3.1 Value-Aware Register Files (VARF)

Exploiting narrow-width values for power 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 Figure 1 (a), access to the 18-bit and 30-bit partitions is controlled by the narrowness flag  $N_1 N_0$  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 power savings can be achieved. However, the gated logic, 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, power, and area overhead. Notice that we introduce P-VARF here only for reference purposes.

#### 3.2 Asymmetric Banking

P-VARF register files, on one side, could be a power-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, power consumption, and access latency [37][3][27][36]. In this paper, we propose an asymmetrically-banked VARF register file (AB-VARF) to exploit the majority of narrow values for further power 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 and consequently the wordline length, the access to 16-bit or 34-bit banks incurs much less power consumption than to

64-bit banks. As narrow-width values dominate the overall set of register values, the majority of register file accesses will be served by 16-bit and 34-bit banks. Thus, the overall power efficiency of AB-VARF register files is further improved over conventional designs. Obviously, AB-VARF implementation also occupies less area. Figure 1 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 [36] that dramatically reduces the read requests to the register file.

With the bypass hint scheme in place, however, writeport conflict remains the major performance issue. We borrow the register port scheduling scheme from [3] to completely avoid both read and write port conflicts in banks. Register ports of each bank are treated as additional resources during instruction scheduling. 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. The size of the vector is set to the latency of the longest operation, e.g., floating-point square root. During the selection, the bit-positions (determined by the operation latency) in the scheduling vectors of the destination register bank are checked for available write ports. If a write port is allocated, the corresponding bit in the scheduling vector for this write port is set to 1. Otherwise, the instruction cannot be selected. A global pointer is maintained to point to the bit position corresponding to the current cycle for all vectors and reset the pointed bits to 0s at the end of the write-back stage before it advances to the next position in the vectors at every cycle. Notice that a load instruction may still cause bank writeport conflicts since the actual latency of the load cannot be determined during instruction scheduling. We employ a conservative strategy here to schedule the load instruction writeport for both hit and miss in the level-1 cache. If the load also misses at the 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 power consumption in banked register files, the width predictor itself should be a light-weighted one in terms of power 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 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 the counter is zero, which predict a 64-bit value. If the value width is 34 bits, the correspond-

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
Width Predictor	2048-entry, 2-way BTB, and 32-entry RAS
	2K bimodal predictor
Memory Hierarchy	
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
Power Parameters	
Technology	70nm
Supply Voltage	0.9V

**Table 2. Parameters for the simulated processor.**

ing counter is set to 1. A 16-bit value increases the counter to 2 and a 64-bit value resets the counter to 0.

The width prediction is verified against the width detection result from the functional unit, as the physical register id implies the predicted width or a width larger than the predicted one due to the unavailable narrow free registers. A width misprediction, if it causes the renaming register not to be 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 the ROB). Since over-prediction does not have this bit space problem, it can still be treated as the correct prediction.

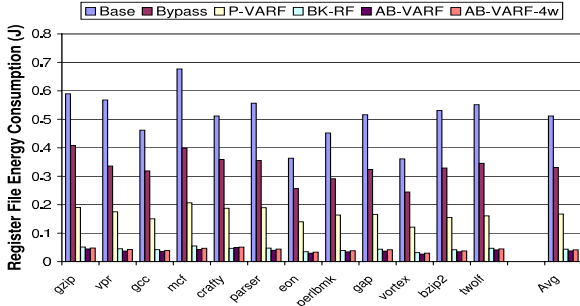
## 4 Evaluation

### 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 the SPEC CINT2000 suite compiled for the 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 single simulation point (*gap* uses the standard single simulation point) specified by SimPoint [31]. Then, we simulate the next 100 million instructions in detail.

### 4.2 RegisterFile Power Models

To model the per access dynamic power of different register files, we utilize Cacti 3.2 [32] for deriving the power numbers of major components within the register file. As shown in Equation 1, the dynamic power of accessing a



**Figure 2. A comparison of dynamic energy consumption in value-aware register files.**

conventional register file comes from five contributors: decoder ( $P_{Dec}$ ), wordline ( $P_{WL}$ ), bitline ( $P_{BL}$ ), sense amplifier ( $P_{SA}$ ), and data output drive ( $P_{DR}$ ).

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

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

$$\begin{aligned} P_{P-VARF} &= P_{Dec} + \sum P_{Part}, \\ P_{Part} &= Part\_width/64 * (P_{WL} + P_{BL} + P_{SA} + P_{DR}). \end{aligned} \quad (2)$$

The banked register file reduces power consumption by activating a single bank for each access. Notice that each bank has its own decoder. However, additional global address and data routing is required in banked designs. The power model is given by Equation 3.

$$\begin{aligned} P_{BK-RF} &= P_{Dec} + P_{Data} + P_{Routing}, \\ P_{Data} &= P_{WL} + P_{BL} + P_{SA} + P_{DR}, \\ P_{Routing} &= P_{Addr\_Routing} + P_{Data\_Routing}. \end{aligned} \quad (3)$$

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

$$P_{AB-VARF} = P_{Dec} + BK\_width/64 * P_{Data} + P_{Routing}. \quad (4)$$

We are also working on more detailed circuit-level implementation and simulation of our AB-VARF. The result will be presented in our future work.

### 4.3 Experimental Results

In this section, we evaluate the power efficiency and performance of asymmetrically-banked VARF register files (AB-VARFs). For evaluation purposes, we also modeled a conventional monolithic register file with 16 readports and

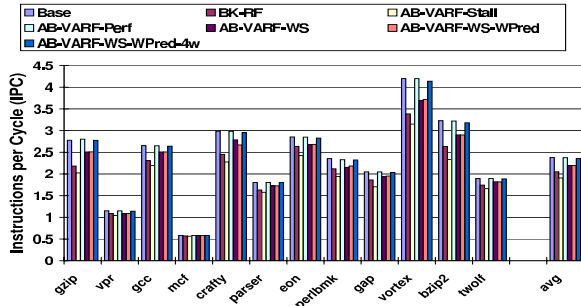
8 writeports as the base design (Base), the base design with the bypass hint scheme (Bypass), a P-VARF register file (P-VARF), and a base banked register file (BK-RF). Notice that both P-VARF and BK-RF register files employ the bypass hint scheme. Due to the banking overhead, we only consider banked register files with 4 banks for this experimental evaluation. Each bank by default has 4 readports and 2 writeports such that the aggregate port number is the same as in a conventional register file. Notice that the base BK-RF utilizes a simple stall-one-cycle scheme for bank writeport conflicts.

To investigate the impact on performance, we present energy numbers instead of power dissipation for different register file designs. Figure 2 shows that the bypass hint scheme Bypass alone reduces the energy consumption by 35% over a conventional register file Base, and P-VARF further improves this reduction to 67%. In comparison, the power-efficient banked register file BK-RF achieves a dramatic 91% energy reduction. However, the performance loss is also non-trivial, a 11.8% on the average as shown in Figure 3, due to 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 power/performance efficiency. For the following discussion, we focus on the AB-VARF-211 bank configuration. Different from the P-VARF and BK-RF, additional supports for register file read/write port scheduling and width prediction are required in AB-VARF. On the average, our AB-VARF achieves an astonishing 92.6% energy reduction, as shown in Figure 2. Notice that this energy number includes the energy consumption in the width predictor.

To further evaluate our design choices for AB-VARF register files, we compare in Figure 3 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. The 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.

Figure 3 shows that 1) simply stalling the pipeline on bank port conflicts 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) instead, writeport conflicts still present the major performance bottleneck and writeport scheduling causes a noticeable performance loss of 6.5% compared to the perfect scheme; 4) a simple bimodal width predictor achieves a very close performance to the perfect predictor. The dramatic performance loss in AB-VARF-Stall is due to the significantly increased writeport conflicts in 16-bit banks since half of the register file writes are steered to the two 16-bit banks. 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



**Figure 3. Performance comparison among Base, Bk-RF, and AB-VARFs.**

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 power efficiency of AB-VARF is still fully exploited, as shown in Figure 2.

## 5 Conclusions and Future Work

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

## References

- [1] A. Aggarwal and M. Franklin. Energy efficient asymmetrically ported register files. In *Proc. of ICCD'03*, pages 2–7, 2003.
- [2] S. Balakrishnan and G. Sohi. Exploiting value locality in physical register files. In *Proc. of Micro-36*, pages 265–276, 2003.
- [3] R. Balasubramonian, S. Dworkadas, and D. H. Albonese. Reducing the complexity of the register file in dynamic superscalar processors. In *Proceedings of Micro-34*, pages 237–248, December 2001.
- [4] E. Borch, E. Tune, S. Manne, and J. Emer. Loose loops sink chips. In *Proc. of HPCA-8*, pages 270–281, February 2002.
- [5] D. Brooks and M. Martonosi. Dynamically exploiting narrow width operands to improve processor power and performance. In *Proc. of HPCA-5*, January 1999.
- [6] D. Burger and T. M. Austin. The simplescalar tool set, version 2.0. Technical Report 1342, Computer Sciences Department, University of Wisconsin, 1997.
- [7] J. A. Butts and G. S. Sohi. Use-based register caching with decoupled indexing. In *Proceedings of the 31st annual international symposium on Computer architecture*, pages 302–313, 2004.
- [8] R. Canal, J.-M. Parcerisa, and A. Gonzalez. Dynamic cluster assignment mechanisms. In *Proceedings of the 6th International Symposium on High Performance Computer Architecture*, pages 132–142, 2000.
- [9] A. Capitanio, N. Dutt, and A. Nicolau. Partitioned register files for vlwls: a preliminary analysis of tradeoffs. In *Proceedings of the 25th annual international symposium on Microarchitecture*, pages 292–300, 1992.
- [10] J. L. Cruz, A. Gonzalez, M. Valero, and N. P. Topham. Multiple-banked register file architectures. In *Proceedings of the 27th annual international symposium on Computer architecture*, pages 316–325, 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, Sep. 2006.
- [12] O. Ergin et al. Register packing: Exploiting narrow-width operands for reducing register file pressure. In *Proc. of MICRO-37*, pages 304–315, Portland, OR, 2004.
- [13] K. I. Farkas, P. Chow, N. P. Jouppi, and Z. Vranesic. The multicenter architecture: reducing cycle time through partitioning. In *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, pages 149–159, 1997.
- [14] A. Gonzalez, J. Gonzalez, and M. Valero. Virtual-physical registers. In *Proceedings of the 4th International Symposium on High-Performance Computer Architecture*, pages 175–184, 1998.
- [15] G. Gonzalez, A. Cristal, D. Ortega, A. Veidenbaum, and M. Valero. A content aware integer register file organization. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*, June 2004.
- [16] J. Hu, S. Wang, and S. G. Ziavras. In-register duplication: Exploiting narrow-width value for improving register file reliability. In *Proc. of the International Conference on Dependable Systems and Networks (DSN-2006)*, pages 281–290, Philadelphia, PA, June 25–28 2006.
- [17] S. Jourdan, R. Ronen, M. Bekerman, B. Shomar, and 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*, pages 216–225, 1998.
- [18] R. E. Kessler. The alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, March-April 1999.
- [19] M. Kondo and 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*, pages 40–49, 2005.
- [20] C. Liem, T. May, and 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)*, pages 397–402, November 06–10 1994.
- [21] M. H. Lipasti, B. R. Mestan, and E. Gunadi. Physical register inlining. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*, pages 325–335, June 2004.
- [22] G. H. Loh. Exploiting data-width locality to increase superscalar execution bandwidth. In *the 35th International Symposium on Microarchitecture (MICRO)*, pages 395–405, November 18–22 2002.
- [23] L. A. Lozano and G. R. Gao. Exploiting short-lived variables in superscalar processors. In *Proceedings of the 28th annual international symposium on Microarchitecture*, pages 292–302, 1995.
- [24] D. R. Lutz and D. N. Jayasimha. Early zero detection. In *Proceedings of the 1996 International Conference on Computer Design (ICCD '96)*, pages 545–550, 1996.
- [25] J. F. Martinez, J. Renau, M. C. Huang, M. Prvulovic, and J. Torrellas. Cherry: checkpointed early resource recycling in out-of-order microprocessors. In *Proc. of Micro-35*, pages 3–14, 2002.
- [26] T. Monreal, A. Gonzalez, M. Valero, J. Gonzalez, and V. Vinals. Delaying physical register allocation through virtual-physical registers. In *Proc. of Micro-32*, pages 186–192, 1999.
- [27] I. Park, M. Powell, and T. Vijaykumar. Reducing register ports for higher speed and lower energy. In *Proceedings of the International Symposium on Microarchitecture*, Dec. 2002.
- [28] M. Postiff, D. Greene, S. Raasch, and T. Mudge. Integrating superscalar processor components to implement register caching. In *Proceedings of the 15th international conference on Supercomputing*, pages 348–357, 2001.
- [29] R. P. Preston et al. Design of an 8-issue superscalar risc microprocessor with simultaneous multithreading. In *Proc. IEEE International Solid-State Circuits Conference*, 2002.
- [30] R. M. Russell. The cray-1 computer system. *Commun. ACM*, 21(1):63–72, 1978.
- [31] T. Sherwood et al. Automatically characterizing large scale program behavior. In *Proc. of ASPLOS X*, October 2002.
- [32] P. Shivakumar and N. Jouppi. Cacti 3.0: An integrated cache timing, power, and area model. Technical report, Compaq Western Research Lab, 2001.
- [33] J. A. Swensen and Y. N. Patt. Hierarchical registers for scientific computers. In *Proc. of ICS'88*, pages 346–354, 1988.
- [34] L. Tran, N. Nelson, F. Ngai, S. Dropho, and M. Huang. Dynamically reducing pressure on the physical register file through simple register sharing. In *Proc. of ISPASS'04*, pages 78–87, 2004.
- [35] J. Tseng and K. Asanovic. Energy-efficient register access. In *Proc. of the 13th Symposium on Integrated Circuits and Systems Design*, 2000.
- [36] J. Tseng and K. Asanovic. Banked multiported register files for high-frequency superscalar microprocessors. In *30th International Symposium on Computer Architecture (ISCA-30)*, San Diego, CA, June 2003.
- [37] S. Wallace and N. Bagherzadeh. A scalable register file architecture for dynamically scheduled processors. In *Proceedings of the 1996 Conference on Parallel Architectures and Compilation Techniques*, page 179, 1996.
- [38] K. C. Yager. The MIPS R10000 superscalar microprocessor. *IEEE Micro*, 16(2):28–40, April 1996.
- [39] R. Yung and N. Wilhelm. Caching processor general registers. In *Proceedings of the International Conference on Circuits Design*, pages 307–312, 1995.
- [40] J. Zalamea, J. Llosa, E. Ayguade, and M. Valero. Two-level hierarchical register file organization for vlw processors. In *Proc. of Micro-33*, pages 137–146, 2000.