Selective Writeback: Exploiting Transient Values for Energy-Efficiency and Performance

Deniz Balkan Joseph Sharkey Dmitry Ponomarev Kanad Ghose
Department of Computer Science,
State University of New York, Binghamton, NY 13902-6000
{dbalkan, jsharke, dima, ghose}@cs.binghamton.edu

ABSTRACT

Today's superscalar microprocessors use large, heavily-ported physical register files (RFs) to increase the instruction throughput. The high complexity and power dissipation of such RFs mainly stem from the need to maintain each and every result for a large number of cycles after the result generation. We observed that a significant fraction (about 45%) of the result values are delivered to their consumers via the bypass network (consumed "on-the-fly") and are never read out from the destination registers. In this paper, we first formulate conditions for identifying such transient values and describe their micro-architectural implementation; then we propose a technique to avoid the writeback of such transient values into the RF. With 64-entry integer and floating point register files, our technique achieves an 11% performance improvement and 29% reduction in the RF energy consumption compared to the baseline machine with the same number of registers. Furthermore, for the same performance target, the Selective Writeback scheme results in a 38% reduction in the energy consumption of the RF compared to the baseline machine.

Categories and Subject Descriptors

C.1 [**Processor Architectures**]: Other Architecture Styles –*Pipeline processors.*

General Terms: Performance, Design

Keywords: Register Files, Energy-Efficiency

1. INTRODUCTION

The physical register file (RF) is one of the key datapath components of modern high-performance microprocessors. To support large instruction windows for effective exploitation of available ILP, large RFs are required in order to buffer speculative state of the program. However, as the number of entries in the RF increases, the access to the RF (which can potentially limit the cycle time [4], [17], [15], [8], [22]) is likely to require multiple cycles, especially in the era of higher frequencies and dominating wire delays. At the same time, read access to the RF typically lies on the critical schedule-to-execute path, which makes it desirable to perform this access within a single cycle. If the RF access is pipelined over several cycles, then the load-hit and branch misspeculation penalties increase and additional bypass stages are needed, negatively impacting both performance and complexity. Finally, large RFs also dissipate significant amount of power. The energy dissipated in the register file has been reported to account for 10% to 25% of the total chip energy [1], [4]. The situation is further exacerbated in the SMT processors, where the pressure on the register file is increased as larger physical register files are needed to support multiple thread contexts.

ISLPED'06, October 4–6, 2006, Tegernsee, Germany.

Traditional register management mechanisms are designed to easily support precise interrupts, but result in an inefficient use of registers. Specifically, a physical register allocated for the destination of an instruction is deallocated only when the next instruction writing to the same architectural (logical) register commits. Such a conservative register management guarantees that until all instructions between the two consecutive definitions of the same architectural register commit, the earlier definition is available and can be resurrected should the later definition be squashed as a result of a branch misspeculation, exception or interrupt. However, in this scheme registers remain allocated for a significant number of cycles, requiring large RFs to be used. An alternative to using large RFs is to use smaller number of registers, but manage them more effectively. To this end, a number of solutions have been proposed, including techniques for late register allocation, early register deallocation and register sharing. One set of solutions [17], [20], [21] targets early register deallocation. In all of these schemes, however, each and every result value is still written into the RF and the validity of the physical register is used as one of the conditions for its early deallocation. In this paper, we overcome this restriction and propose a scheme that in some cases deallocates physical registers right after instruction execution and *avoids the writebacks* of the produced values into the RF.

The specific contributions and the key results of this paper are:

- We introduce the *Selective Writeback (SWB)* mechanism - an aggressive scheme for early deallocation of physical registers whose values (which we call transient) are obtained by all of their consumers through the bypass network. We show that such early deallocations are possible in 45% of the cases across the SPEC 2000 benchmarks. Compared to the previously proposed early deallocation techniques, our solution is more proactive in nature, as we deallocate a register immediately after the instruction that targets this register completes the execution, *without even writing the produced transient value into the RF*. Such values are simply dropped from the datapath.

- We evaluate the impact of this mechanism on the performance as well as the energy dissipations within the RF. The RF energy is reduced because the writes of *transient* values are avoided. For 64-

entry RFs, *selective writeback* results in 11% IPC improvement and 29% dynamic energy savings within the RF compared to the baseline machine with the same number of registers. Furthermore, for the same performance target, our scheme achieves a 38% reduction in the energy consumption of the RF compared to the baseline machine (accounting for the dissipations in the additional logic).

- We compare the performance of the SWB scheme against some previously proposed techniques for register file optimizations and show that our mechanism outperforms the previous solutions for the majority of the benchmarks as well as on the average.

2. MOTIVATION AND DEFINITIONS

It is well-known that most of the register instances in a datapath are consumed within a very few cycles following their generation [18], [23]. Following the work of [23], we define a value targeting a register to be ***short-lived*** if the architectural register allocated as a destination of the instruction X has been redefined (used as a destination by another instruction) before the value generated by X is written back. We call the instruction that redefines a register allocated to hold a short-lived value as the ***redefiner***. In our simulations of the SPEC 2000 benchmarks, about 85% of all generated register values were identified as short-lived.

We define a produced result value as ***transient*** if the following conditions are true:

C1) The value must be short-lived.

C2) There must be at most one instruction that consumes the value.

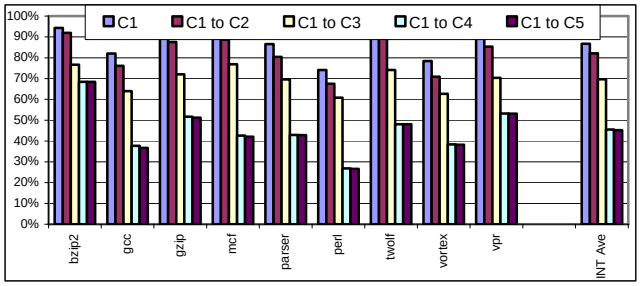
C3) The only consuming instruction must be selected for issue before the value is produced – this ensures that the value is obtained for consumption off of the bypass network.

C4) There must be no branch instructions between the value-producing instruction and its redefiner.

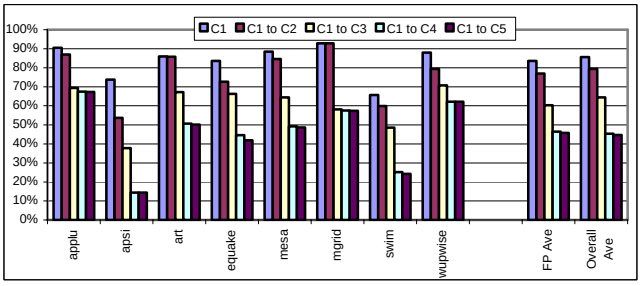
C5) The sole consumer of the value must not be subject to a replay caused by a load latency misprediction or a memory dependence misprediction

Figure 1 shows the percentage of transient values across the execution of SPEC 2000 benchmarks. The details of our simulation framework are given in Section 4. The bars from left to right correspond to the cumulative percentages of the various conditions (C1 through C5) described above. For example, the leftmost bars show the percentage of all generated values when condition C1 is valid, the next set of bars shows the percentage of cases when both conditions C1 *and* C2 are satisfied and so on. The rightmost set of bars depicts the percentage of transient values, as all 5 conditions are satisfied. On the average across all benchmarks, about 45% of the produced results are transient.

The results of Figure 1 imply that almost half of the generated results are not read from the RF, are not needed to recover from a branch misprediction, and the consumers of these results are not subject to any memory replay traps. The only reason to store these values in the register file is to allow for the reconstruction of the precise state after exceptions or interrupts. It is precisely the goal of this paper to introduce mechanisms that avoid writing such values into the RF and instead rely on periodic RF checkpointing to correctly handle exceptions and interrupts. The transient values are simply "dropped" from the datapath right after their generation and the corresponding physical registers are immediately released.



(a)



(b)

Figure 1. Various Statistics about Short-lived Values for integer (a) and floating-point (b) benchmarks.

3. DETECTING TRANSIENT VALUES

It is conceivable that some of the conditions for transient values (specifically, C2 and C4) can be determined by the compiler to simplify the hardware; however in the following discussions we describe an all-hardware implementation. Surprisingly, this mechanism is quite simple and relies on the use of two marker bits per rename table entry and two status bits per physical register. The two marker bits associated with each rename table entry are as follows:

a) The FC ("first consumer renamed") bit: this bit is set when the first consumer of a physical register representing the current instance of an architectural register is renamed.

b) The MCB ("multiple consumers or following branch") bit: this bit is set directly when a second or subsequent consumer of a physical register representing the current instance of an architectural register is renamed. This bit is also gang set for all entries in the rename table when a branch instruction is encountered.

When the rename table entry for mapping an architectural register is set up, the FC and MCB bits in the newly created entry *are both cleared*. The FC bit is set, as described above – on renaming the first consumer instruction of the associated physical register. When a consumer of a physical register is renamed, the FC bit of the rename table entry for that physical register is checked – if that bit is found to be already set, the MCB bit is set to indicate that more than one consumer exists for that register. A trivial extension of the logic also handles the case when more than one initial consumer of a physical register are renamed in the same cycle. The MCB bit is also set, irrespective of the number of consumers, as described above on encountering a branch. Thus if the MCB bit in the rename table entry for a physical register is set, the implication is that either

condition C2 or C4 are not valid, that is, the associated value is not transient.

The two additional bits – *both set to a one at the time of allocating a physical register* - are needed for detecting transient values are each associated with a physical register and are as follows:

c) The NT ("not transient") bit. This bit indicates if the associated register value is transient (NT = 0) or not (NT = 1). The initialization of NT marks the associated value as transient by default.
d) The CI ("consumer issued") bit, which is cleared when any consumer of a physical register issues. Recall that the physical register has to be read by the issued instruction, so this bit can be easily cleared as the issued instruction has initiated a register file read – either from the RF (or off the bypass network).

At the time of creating the new instance of an architectural register R (that is, at the time of renaming the renamer of the instruction that generates/generated the previous instance of R), the rename table entry to be overwritten is read out. This entry is for the previous instance of R, say physical register P. The MCB bit for P is then copied to the NT bit of P. If the MCB bit is a zero, the NT bit then indicates that conditions C2 and C4 are both valid, as well as the fact that a renamer was also encountered. Ignoring condition C5 for the time being, if both the NT and CI bits of a physical register are both zero at the time of initiating a write to the register, the writing of the register can be abandoned. Put in other words, the value targeting the register is transient, as it has no branch instructions preceding its renamer, nor multiple consumers (NT = 0) and its only consumer has issued (CI = 0) Note that the physical register's address is typically available at least one cycle preceding the actual write in modern datapaths for waking up dependent instructions.

In processors that use speculative scheduling based on load latency prediction, condition C5 is implemented by resetting the NT and CI bits of the affected physical registers. In addition, no value is dropped (even if it is determined as transient) during the time when such recovery takes place. To support SWB in processors that use memory dependence prediction [7], we use a technique similar to what is described in [20]. Specifically, we ensure that the consumer of a transient value is older than the oldest store instruction whose address has not yet been computed. As shown in Figure 1, the percentage of values identified as transient is reduced only slightly by imposing this additional condition. In our evaluations, we model both types of memory-related speculations.

The energy savings and potential performance gains from avoiding writebacks into the register file come at the cost of maintaining and managing just three bits per physical register, as described earlier.

4. MAINTAINING THE PRECISE STATE

When some register values are discarded right after their generation, traditional ROB-based mechanisms for restoring the precise state can no longer be used. Therefore, to maintain the precise state in the event of exceptions or interrupts, we rely on the use of checkpointing (We note here that branch mispredictions are handled just as in traditional designs due to the condition C4 as described in Section 2). Specifically, we create periodic checkpoints of the RF in the following manner. Assume that the instruction I1 is the youngest dispatched instruction in the ROB at the instant when the decision to create a checkpoint is made. At this moment, the most recent instance of every architectural register is either the value in the RF or it is defined by some in-flight instruction, whose result has not yet been generated. For example, if the value produced by the instruction writing to architectural register X has been dropped, then there is another in-flight instruction down the stream that writes into the same architectural register. Consequently, if we allow all the instructions between the *ROB_head* (which points to the commitment end of the ROB) and instruction I1 to unconditionally write their results to the register file, then at the time of I1's commitment a precise state of the register file will be available. At this point, the full checkpoint of the register file can be created. Basically, the selective dropping of values is avoided during the checkpoint creation period, i.e. all values are written into the register file. The specific circuit implementation of checkpointing is not central to this paper. One can either use a separate register file for this purpose (as in [20]), or embed the checkpoint within the register file itself, by backing up each bitcell with a shadow copy [9]. For our evaluations, we assumed the Checkpointed Register File (CRF) design proposed in [9], where each bitcell is backed up by a shadow cell.

Table 1. Configuration of the Simulated Processor

Parameter	Configuration
Machine width	4-wide fetch, 4-wide issue, 4 wide commit
Window size	64-entry issue queue, 64 entry LSQ, 128–entry ROB
Registers	Various sizes studied, as indicated in the text
Function Units and Latency (total/issue)	4 Int Add (1/1), 1 Int Mult (3/1) / Div (20/19), 2 Load/Store (2/1), 4 FP Add (2), 1FP Mult (4/1) / Div (12/12) / Sqrt (24/24)
L1 I–cache	32 KB, 2–way, 32 byte line, 1 cycle hit time
L1 D–cache	32 KB, 4–way, 32 byte line, 2 cycles hit time
L2 Cache unified	512 KB, 4–way, 128 byte line, 8 cycles hit time
BTB	1024 entry, 4–way set–associative. Minimum branch misprediction penalty – 10 cycles
Branch Predictor	Combined with 1K entry Gshare, 10 bit global history, 4K entry bimodal, 1K entry selector
Memory	128 bit wide, 120 cycles
TLB	64 entry (I), 128 entry (D), fully associative

When an exception occurs, the processor state is recovered to a previous checkpoint. However, if upon resumption of the execution from the checkpointed state we continue to drop transient values, the same exception can potentially reoccur again, thus inhibiting any forward progress. To prevent this scenario, we avoid dropping the values until the exception reoccurs, at which point the exception can be handled precisely. However, some exceptions may not reoccur in the course of re-execution, so we need some mechanism to identify when to resume selective dropping of the register values. Our solution is as follows. After the checkpoint is created, we maintain a counter of the number of instructions committed since the creation of the checkpoint. When an exception occurs for the first time (an exception is recognized when the instruction commits), the value of this counter is saved in a register. In the course of re-execution, we avoid dropping the values until the number of committed instructions reaches the saved value of the counter. At this point, the exception would have either reoccurred or is guaranteed not to be encountered again. Since our checkpointing period is rather small

and exceptions generally occur infrequently, the overall impact on the performance is negligible, as we detail in Section 5.

To buffer a large number of store instructions between two consecutive checkpoints, we use the approach described in [20] and also used in a few others works. The values are stored within the local cache hierarchy, but their propagation to the main memory is avoided until it is safe to do so. Each cache line updated in this manner is marked *Volatile*, using one extra bit for each cache line. When a processor needs to rollback to a checkpoint, all cache lines marked *Volatile* are invalidated using the gang-invalidate signal. When the precise state is created (as described in the previous section), the *Volatile* bits are cleared.

5. SIMULATION METHODOLOGY

For estimating the energy savings and the performance gains achieved by using the SWB scheme, we used a significantly modified version of the Simplescalar simulator [6] that explicitly models the issue queue, the reorder buffer, the load/store queue, the register renaming logic and other out-of-order execution mechanisms associated with a datapath where a unified register file is used. Register file is read after the instructions are issued. The studied processor configuration is shown in Table **1**. For load-latency prediction, we used the load hit/miss predictor which was used for the Alpha 21264 processor. A 5-bit saturating counter is used for each entry where the counter is incremented by 1 in case of hit and decremented by 2 in case of miss. Load instructions are predicted to hit in the cache if the most significant bit of the counter is 1. We also used the store set predictor described in [7] for speculating on memory dependencies.

We used 9 integer SPEC 2000 benchmarks *(gcc, gzip, parser, perlbmk, twolf, vortex, mcf, bzip* and *vpr*) and 8 floating point SPEC 2000 benchmarks (*applu, art, mesa, mgrid, swim, apsi, equake* and *wupwise*). We had difficulties compiling the other benchmarks (mostly those written in Fortran) in our simulation framework. Benchmarks were compiled using the Simplescalar GCC compiler (with –O4 optimizations) that generates code in the portable ISA (PISA) format. Reference inputs were used for all the simulated benchmarks. The results from the simulation of the first 1 billion instructions were discarded and the results from the execution of the following 500 million instructions were used.

For estimating the energy dissipated in the course of accessing the register files, the event counts gleaned from the simulator were used, along with the energy dissipations, as measured from the actual hand–crafted VLSI layouts using industry-standard Cadence® design tools. CMOS layouts for the register files and the bit–vectors in a 0.18 micron 6 metal layer process (TSMC) were used to get an accurate idea of the energy dissipations for each type of transition.

6. RESULTS AND DISCUSSIONS

We now describe the energy implications of the SWB scheme and then allude to the performance aspects of our mechanism, comparing the results against some previously proposed techniques.

6.1 Energy Considerations

We first present the energy implications of SWB on the RF, ignoring the dissipations in the auxiliary structures, such as the additional bit-vectors. We then take the dissipations of those components into consideration. First, we consider the reduction in the overall energy dissipations (or energy per task). Figure 2 presents the energy reduction achievable within the register file if the SWB scheme is used. The comparison is given for a datapath with 64 integer and 64 floating point registers. There are two bars depicted in the figure, the bar on the left presents the energy savings within the register file itself when the dissipations in the additional logic required by SWB are ignored. On the average, SWB reduces the RF's energy consumption by 36% - The energy reduction on individual benchmarks ranges from 50% for *bzip* to 13% for *apsi*. In the presented results, we assumed that the energy reduction is only a consequence of the smaller number of writebacks. The bar on the right depicts the energy reduction in the RF, if the dissipation in all additional bit-vectors and arrays required by the SWB scheme are taken into consideration. Even with the additional energy dissipation, there is still a reduction of 29% in the RF energy. The overall processor energy savings depends on the percentage of energy attributed to the RF. The range of these savings will be between 3% and 7% (considering that RFs contribute between 10% and 25% to the overall processor energy, as mentioned in the introduction).

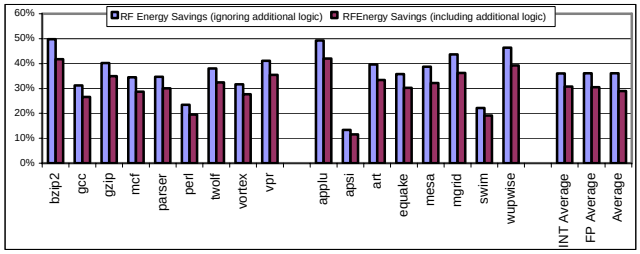


Figure 2. Percentage of Energy Savings within the RF.

Finally, we also evaluated the energy savings within the RF, if the machine employing the SWB scheme is configured to provide the same IPC performance as the baseline processor. As detailed later in the section, SWB with 80-entry RFs provides the same IPC performance on the average as the baseline machine with 96-entry RFs. However, higher energy savings can be achieved comparing these configurations, as in addition to having fewer writebacks, a smaller RF size also further decreases the energy per access. On the average, the energy savings (as well as the power savings since the performance is the same) are 45% in this case if the additional logic is ignored and 38% if the additional logic is accounted for.

6.2 Performance Considerations

Next, we evaluate the performance of the SWB scheme described in Section 3 for 64-entry register files, (64-entry integer RF + 64-entry floating point RF). Figure **3** depicts the commit IPC of the SWB scheme for different checkpointing frequencies. The figure has three curves, showing the average commit IPC of the integer benchmarks, the average commit IPC of the floating point benchmarks and the overall average IPC. For this experiment, we assumed that no exceptions or interrupts occurred, but we still avoided the dropping of the transient values in the course of constructing a checkpoint. Thus, performance difference among the various points on these curves only comes from the reduced pressure on the RF. It can be seen from the figure that the performance improvement achieved by the SWB scheme is small when checkpoints are created very frequently. For the checkpoining periods greater than 500 cycles, the performance gains are almost equal to those with no checkpointing in place – i.e. when all transient values identified as such are dropped. Based on the results presented in Figure **3**, we conclude

that the optimal checkpointing interval is 500 cycles, and we use this number for the rest of our experiments. We also simulated the effects of exceptional events, on the performance of the SWB scheme. Our experiments indicated that unless exceptions occur exceedingly often, such as a page fault occurring once every 1000 memory operations (which is, of course, an unrealistically high rate), the performance is very close to that of the system with no exceptions. We observed similar results for other RF sizes as well. For a checkpointing frequency of 500 cycles, the average performance gains of the SWB scheme compared to the baseline machine are almost 11%, ranging from 0.8% for *equake* to 39% for *art*. The average IPC gain for integer benchmarks is 6.2% and the average IPC gain for floating point benchmarks is 16.2%.

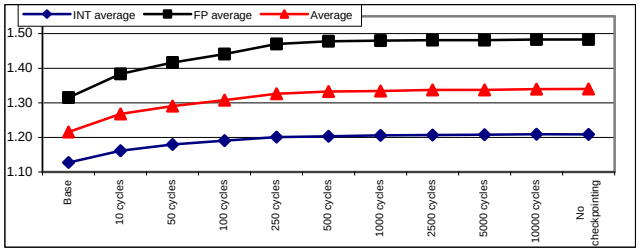


Figure 3. Commit IPCs of the SWB Scheme for different checkpointing periods.

6.3 Performance Comparison with Previous Techniques

Figure **4** compares the performance of SWB with some previously proposed techniques for optimizing register files. These alternative schemes are summarized in Section 6, we avoid repeating these discussions here for the sake of space.

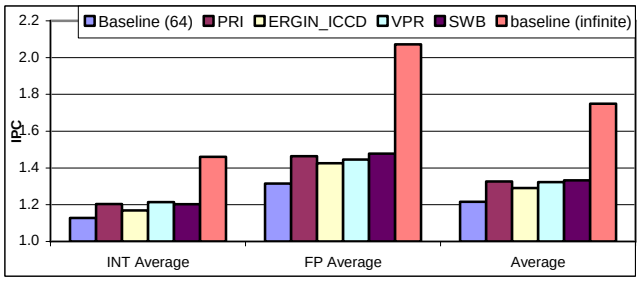


Figure 4. IPC comparison of the SWB scheme with PRI, VPR and Ergin_ICCD schemes.

In Figure 4, the first set of bars shows the performance of the baseline case. The second set of bars shows the IPCs of the scheme of [9], which deallocates the physical registers at the time of commitment or at the time of renaming the renamer instruction. We refer to this scheme as Ergin_ICCD to reflect the first author name and the publication venue. The third set of bars depicts the IPCs of the Virtual Physical Registers scheme of [11]. We refer to it as VPR in the rest of the paper. For the VPR scheme, we assume that 32 reserved registers are in use – the best configuration suggested in [11]. The next set of bars shows the performance of Physical Register Inlining [17], where narrow-width register values are stored directly within the rename table. We have implemented all these techniques in our simulation framework and therefore can compare the results on an even footing.

On average, SWB outperforms the baseline by 10.9%, compared to 5.35% for Ergin_ICCD scheme, 7.7% for VPR and 10.2% for PRI. It is interesting to notice that SWB outperforms the previous schemes on most of the benchmarks, but there are some exceptions. For example, PRI performs better than SWB for *gzip, swim, art* and *parser*. This is because a larger percentage of narrow-width values can be stored in the rename table for these benchmarks, than the percentage of values that can be detected and predicted as transient. For *bzip2, gzip, applu, vortex,* and *wupwise* the performance of SWB falls short of the VPR scheme and for *swim and wupwise*, Ergin_ICCD scheme outperforms SWB. In all other cases, SWB shows superior performance compared to other techniques. It is also interesting to observe either VPR or PRI perform better for various benchmarks with respect to each other.

6.4 Sensitivity Analysis to the RF Size

Figure **5** depicts the commit IPC values for the baseline machine, the SWB scheme for register files ranging from 48 to 128 entries. The commit IPC values are presented for the overall averages. Similar trends can be observed in all three graphs. For the same number of physical registers, the SWB scheme results in the following IPC improvements over the baseline machine on the average across all simulated benchmarks: 10.8% for 48 registers, 10.9% for 64 registers, 10.9% for 72 registers, 8.1% for 80 registers, and 7.7% for 96 registers. Note that with 128 registers only 1.57% IPC improvement is possible since the datapath with 128 registers already performs almost as well as the one with infinite number of registers, i.e. with 128 registers there are very few stalls due to the lack of physical registers.

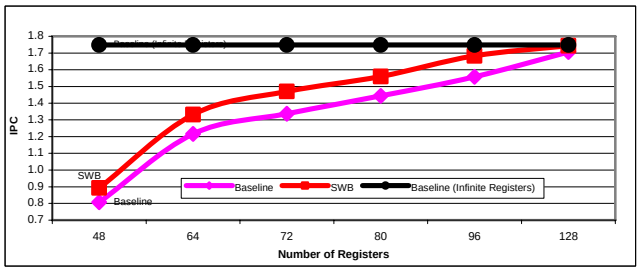


Figure 5. IPC values for the baseline machine, the SWB scheme with various register file sizes

It is also interesting to compare the number of registers needed to provide the same level of performance for the various schemes. As shown in Figure **5**, the performance of the baseline machine with 96-entry RFs is matched by the SWB scheme with 80-entry RFs and. Consequently, for the same IPC, the use of fewer registers with SWB can reduce the wire delays and possibly decrease the cycle time, if the register file access lies on the critical timing path. The quantification of these additional potential advantages of SWB is beyond the scope of this paper.

We do not directly compare the energy advantages of SWB with the previously proposed schemes for optimizing register files, because energy reduction was not the goal of those techniques. With those schemes, it is possible that the additional structures and/or data movements required to support the complicated register management will outweigh the energy advantages of having somewhat smaller register files. The energy advantages of SWB come from avoiding a significant fraction of writebacks to the register file, which is a unique feature of our solution. Alternative register file optimizations

for energy, such as banking [4][15][22][24] can be used in conjunction with our technique to provide additional energy reduction.

In summary, SWB achieves higher performance AND lower energy consumption per instruction (or per task) at the same time. It is possible that the power consumption may increase on some configurations due to much faster execution time (because more work is done per cycle). However, the desired power / performance targets can always be achieved using DVFS techniques.

7. RELATED WORK

Several techniques have been proposed in the recent literature to reduce the register file pressure by using the early deallocation of physical registers [9], [17], [20], [21]. These techniques are close in spirit to our selective writeback proposal, but there are fundamental differences. In all of these works, each and every generated result is still written into the register file and validity of the register value is one of the conditions for the earlier register deallocation. Energy reduction was not the goal of the previous proposals for the early deallocation of registers. The second set of solutions delays the actual allocation of physical registers until the time that the result is written back [11]. The third set of solutions reduces the number of registers through the use of register sharing [3], [14].

There is a large body of work that targets the energy reduction in the register files through reducing the number of register ports and register file banking [4], [15], [22], [24]. The common feature of these techniques is that they all encounter a small performance loss due to various reasons. In contrast, our techniques improve energy-efficiency and performance at the same time. Also, all proposed register file banking schemes can be used in conjunction with our technique. Alternative register file organizations (mainly using various forms of caching) have also been explored for reducing the access time (which goes up with the number of ports and registers), particularly in wire-delay dominated circuits [5], [8]. In [19], register file usage was optimized using compiler support to exploit dead value information. A compiler-assisted early register release is explored in [13]. In [10], a technique to pack multiple narrow-width results into the same physical register is proposed to reduce register file pressure. Narrow-width operands were also exploited to reduce the area, access time and energy consumption [16] as well as port complexity [1] of the RFs. The concept of partial value locality was exploited for reducing the register file power, area and delay in [12].

8. CONCLUDING REMARKS

We introduced the notion of transient values and showed that 45% of all the results produced in a typical superscalar datapath are transient in nature. We presented Selective WriteBack (SWB) - a microarchitectural technique that eliminates the writes of transient values to the register file which results in the following power/performance advantages compared to the traditional superscalar datapath:

For the same register file size, SWB improves the performance by 10.8% for 48 registers, 10.9% for 64 registers, 10.9% for 72 registers, 8.1% for 80 registers, and 7.7% for 96 registers. over the baseline machine on the average across the simulated SPEC 2000 benchmarks.

For a processor with 64-entry RFs, SWB also reduces energy by 29%, when the energy dissipated by the additional logic needed by the scheme is taken into account. For the same performance target, the SWB scheme results in 38% reduction in the RF energy compared to the baseline machine.

Finally, we also showed that SWB outperforms several recently proposed schemes for register file optimizations.

9. ACKNOWLEDGMENTS

We thank Oguz Ergin for his contributions during the early stages of this work. This research was supported in part by the National Science Foundation, award numbers CNS 0454298 and EIA 9911099, and by the Integrated Electronics Engineering Center at SUNY Binghamton.

10. REFERENCES

[1] Aggarwal, A., Franklin, M., “Asymmetrically-ported Register Files”, *in Proc. of ICCD 2003*.

[2] Azevedo, A., et.al., “Profile-based Dynamic Voltage Scheduling using Program Checkpoints in COPPER Framework”, *in Proc. of DATE, 2002*.

[3] Balakrishnan, S., Sohi, G., “Exploiting Value Locality in Physical Register Files”, *in Proc. of MICRO-36, 2003*.

[4] Balasubramonian, R.,et. al.., "Reducing the Complexity of the Register File in Dynamic Superscalar Processor", *in Proc. of MICRO-34, 2001*.

[5] Borch, E. et al, "Loose Loops Sink Chips", *in Proc. of HPCA-8, 2002*.

[6] Burger, D. and Austin, T. M., "The SimpleScalar tool set: Version 2.0", *Tech. Report, Dept. of CS, Univ. of Wisconsin-Madison, June 1997*.

[7] Chrysos G., J.Emer, "Memory Dependence Prediction using Store Sets", *in Proc. of ISCA-25, 1998*.

[8] Cruz, J-L. et. al., "Multiple-Banked Register File Architecture", *in Proc. of ISCA-27, 2000*.

[9] Ergin O., et.al., “Increasing Processor Performance through Early Register Release”, *in Proc. of ICCD, 2004*.

[10] Ergin O., et.al., “Register Packing: Exploiting Narrow-Width Operands for Reducing Register File Pressure”, *in Proc. of .MICRO-37, 2004*.

[11] Gonzalez, A., Gonzalez, J., Valero, M., “Virtual-Physical Registers”, *in Proc. of HPCA-4, 1998*.

[12] Gonzalez, R. et. al. “A content Aware Register File Organization”, *in Proc. of ISCA-31, 2004*.

[13] Jones T. et al., “Compiler Directed Early Register Release”, *in Proc. of PACT 2005*.

[14] Jourdan, S.,et. al. “A Novel Renaming Scheme to Exploit Value Temporal Locality through Physical Register Reuse and Unification”, *in Proc. of MICRO-31, 1998*.

[15] Kim, N., Mudge, T., "Reducing Register Ports Using Delayed Write-Back Queues and Operand Pre-Fetch", *in Proc. of ICS-17, 2003*.

[16] Kondo M. and Nakamura H. “A Small, Fast and Low-Power Register File by Bit-Partitioning”, *in Proc.of HPCA-11, 2005*.

[17] Lipasti, M., et.al., “Physical Register Inlining”, *in Proc. of ISCA-31, 2004*.

[18] Lozano, G. and Gao, G., "Exploiting Short-Lived Variables in Superscalar Processors", *in Proc. MICRO-28, 1995*.

[19] Martin, M., Roth, A., Fischer, C., “Exploiting Dead Value Information”, *in Proc. of MICRO-30, 1997*.

[20] Martinez, J., et. al.., "Cherry: Checkpointed Early Resource Recycling in Out-of-order Microprocessors", *in Proc. of MICRO-35, 2002*.

[21] Monreal, T., Vinals, V., Gonzalez, A., Valero, M. “Hardware Schemes for Early Register Release”, *in Proc. of ICPP-02, 2002*.

[22] Park, I., Powell, M., Vijaykumar, T., "Reducing Register Ports for Higher Speed and Lower Energy", *in Proc. of MICRO-35, 2002*.

[23] Ponomarev, D., et. al."Reducing Datapath Energy Through the Isolation of Short-Lived Operands", *in Proc. of PACT-12, 2003*.

[24] Tseng, J., Asanovic, K., "Banked Multiported Register Files for High Frequency Superscalar Microprocessors", *in Proc. of ISCA-30, 2003*.