A Compiler-assisted Banked Register File Architecture*

José L. Ayala†, Marisa López-Vallejo†, Alexander Veidenbaum‡

†Departamento de Ingeniería Electrónica
Universidad Politécnica de Madrid (Spain)
{jayala,marisa}@die.upm.es

‡Center for Embedded Computer Systems
University of California, Irvine (USA)
alexv@ics.uci.edu

ABSTRACT

The complexity of a register file is currently one of the main factors in determining the cycle time of high performance, wide-issue microprocessors. This complexity is directly related to the size and the number of ports of the register file. Advanced superscalar processors require a substantial simplification of this complex elements to allow their implementation and save energy. This work proposes a banked register file architecture in which the number of banks is derived from the more significant n bits of the physical register number. These n bits of the physical register number common to as the more significant n bits of the logical register number. This is accomplished by modifying the register renaming mechanism to partition the free list into n separate lists. A free list is then chosen according to the logical register number. Finally, the compiler is also modified to be aware of the register file banking and to assign source operands of the same instruction to different banks. This reduces the number of ports required. In addition, banks are normally in the "drowsy" mode and are awakened when needed by an instruction. This approach leads to energy savings that can reach 54% with no performance penalty.

1. INTRODUCTION

Current microprocessor designs move towards wider issue and increasingly complex out-of-order execution. This leads to more on-chip hardware and, consequently, an increase in power dissipation. In [13] the authors have described and analyzed those portions of a micro-architecture where complexity grows with increasing instruction-level parallelism. This is the case of the register rename logic, the wakeup logic, the selection logic, the data bypass logic, caches and instruction fetch logic.

In addition, in high performance, wide-issue processors the register file plays a critical role in determining the cycle time. Furthermore, it accounts for a significant fraction of the processor core's power consumption. The register file needs to be large to support multiple in-flight instructions and multi-ported to avoid stalling multiple instruction issue. For instance, the Alpha 21264, a 4-issue machine [6], had to partition the register file to make it implementable. More recent processors, e.g. IBM Power 5, have an even wider issue width.

*This work was supported by the Spanish Ministry of Science and Technology under contract TIC2003-07036

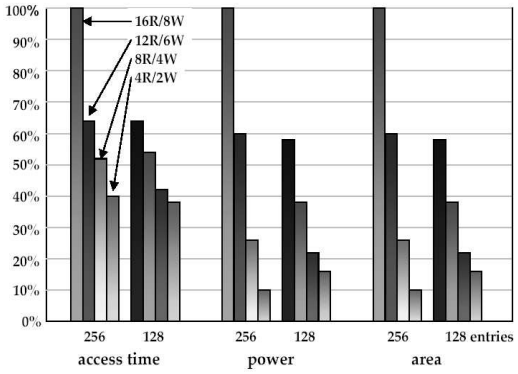

Figure 1: Effect of size and number of ports on access time, power and area

In figure 1 we can see the effects of the size and the number of ports on access time, energy and area for 128-, and 256-entry register files having several combinations of read and write ports [20]. The results were calculated using a modified version of CACTI 3.0 [16] assuming a 0.18 μm technology. The ports were chosen to support 8-, 6-, 4- and 2-issue machines, respectively. These values are normalized against a 256-entry register file with 16R/8W ports.

The current trend in superscalar machines is the design of multiple-pipeline architectures. Recent improvements in technology and computer architecture enable this growing complexity while reducing the cycle time. However, one of the major bottlenecks in these architectures is the access time to the multi-ported register file and the fact that it is not scalable. The goal of this work is to reduce the complexity of the register file and its energy consumption without a performance penalty.

The approach presented in this paper uses a banked register file architecture and with compiler support for register assignment to deal with banking. Unlike previous approaches, this technique does not rely on additional memory or complex logic which can increase the total energy dissipation of the system. Moreover, the proposed approach has no impact on system performance.

Finally, the technique presented here is combined with a low-power "drowsy cell" mechanism, which further reduces the energy consumption.

This paper is organized as follows. The next section presents the motivation for this work and discusses previous work

on the same topic. Section 3 describes the proposed technique. The experimental results and implementation details are given in section 4. Section 5 concludes the paper and discusses possible future direction for this research.

2. RELATED WORK

The energy complexity of register files has been carefully studied in recent years. In [20], Zyuban et al. compare several register file circuit design techniques and their energy efficiency, as a function of the number of registers and the number of ports. The dependence of register file access energy upon technology scaling was also studied by the authors. More recent approaches provide register file models for estimating delay, energy consumption and area on complex implementations of the register file. The work in [17] shows that partitioning the register file can reduce the cost of register storage and communication but with some performance impact. This work also develops a taxonomy of register architectures by partitioning and optimizing the hierarchical register organization to operate on streams of data.

Some works have proposed hierarchical and banked architectures for the register file but in these cases, unlike our, the goal was to reduce the size and number of registers in the register file. Some examples can be found in [3], [4] and [5]. Compiler-based approaches have also been presented, like the work in [19] focused on VLIW processors.

These previous approaches to reduce area and complexity of the register file usually required substantial changes in the pipeline and can lead to complications as noted in [14].

In [14], Park et al. propose to reduce the number of register ports through two proposals, one for reads and the other one for writes. The first approach needs an extra pipeline stage, while the second one performs bypass prediction. Both techniques can cause pipeline stalls due to mispredictions, with negative impact on system performance.

The interest in banked register files dates back several years to partitioned architectures proposed to reduce the complexity and area of the monolithic register file [9]. Such approaches did not consider the energy implications of the technique and the energy overhead of the extra logic. More recent work have improved upon these approaches using compile-time techniques for clustered processors to exploit the temporal locality of references to remote registers in a cluster [10]. But neither of these techniques considered the use of low power techniques and the design of power-aware logic to reduce the energy consumption of the register file as well as the number of ports.

An interesting work by Tseng et al. [18] achieves good energy savings in a banked register file without compromising the IPC of the machine. However, the number of register file ports is not reduced and it is even increased since the main goal of these authors is not to reduce the register file complexity.

Further, in [11] Kim et al. introduce techniques for reducing the number of ports in the register file by adding some auxiliary memory structures. This approach can effectively reduce the number of ports without noticeable impacting the system performance or IPC. However, the energy overhead of the extra hardware is not negligible.

Finally, the presented approach uses the well studied technique of voltage scaling to reduce the energy consumption in a large range of electronic devices. Moreover, voltage scaling has emerged as the preferred technique for power minimization in complex electronic systems at different abstraction levels [12, 15].

To the best of our knowledge, no single approach has obtained both reduction in the number of ports and register file energy without dramatically impacting the performance penalty. The work presented here avoids this limitation of previous approaches.

3. APPROACH

As was discussed above, there are two main goals in our approach: first, to reduce the number of ports of the register file in order to decrease the complexity and the access time of this module. And second, to reduce the energy consumption of this device.

These goals will be achieved by means of a modified register file architecture partitioned into banks. The number of ports is also reduced, assuming that the accesses will be distributed to different banks. Compiler support will be used to modify the register assignment in order to efficiently use those banks. Also, the low-power policy turns the unused banks per instruction into a low-power state to save as much energy as possible.

The following sections describe both architectural and compiler modifications of the proposed approach.

3.1 Banked Register File

The register file provides the *physical* registers assigned to the operands of an instruction (as specified by the *logical* registers). In out-of-order architectures, the register assignment performed by the compiler is modified by the register renaming logic to increase system performance and avoid data dependencies.

The physical register file is divided into several sub-banks which may also have fewer ports. The banks must provide the same amount of registers and the sufficient read and write ports for enabling the normal functioning of the pipeline without performance penalty. For instance, a register file with 256 physical registers and 22R/11W configuration, can be split into four 6R/3W banks with 64 physical registers per bank (Figure 2). In general, the number of ports can be reduced when increasing in the number of banks due to a better distribution of requests across banks.

Banks require additional multiplexing and conflict detection, with the corresponding time increase. This is compensated by a faster access time to a smaller bank. In addition, an extra cycle is required for wake up if "drowsy" mode is used. In the latter case, the bank number is available during instruction wakeup and scheduling and can be used to wake up the bank.

The reduction in ports and size of the register file simplifies the design, reduces the energy consumption per access and allows to follow the current trend on designing multi-superscalar machines of increased issue width.

3.2 Register Assignment

The proposed register assignment policy has been performed modifying the implementation of the GNU compiler gcc 3.2, which is publicly available, generates high quality code and supports multiple embedded and high-performance processors.

Normally the compiler performs a particular register assignment which is later modified by the register renaming

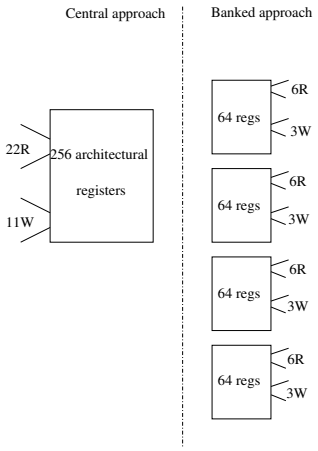

Figure 2: Banked register file

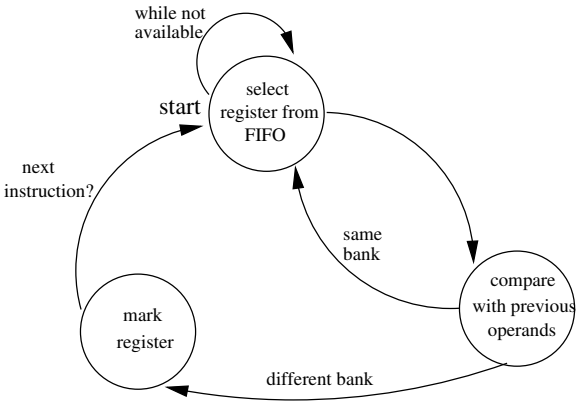

Figure 3: Register assignment algorithm

hardware to avoid hazards. The approach proposed here assumes a power-aware banked register renaming hardware described in [2]. This work proposes a banked register file architecture in which the number of banks is derived from the more significant n bits of the physical register number. These n bits of the physical register number common to as the more significant n bits of the logical register number. In this case, the compiler assignment should not be destroyed by the renaming mechanism because it restricts the renaming to the register file bank. This is accomplished by modifying the register renaming mechanism to partition the free list into n separate lists.

As was previously explained, our approach presents a static technique based on a modified compiler together with a dynamic technique based on the modification of the register renaming hardware. Both schemes will be outlined below.

Gcc, when assigning an architectural register to the instruction operands, retrieves the first available register from a list of free registers. Since gcc does not consider any restrictions on assigning the registers, these are selected sequentially. Thus, all the operands in a gcc assignment can come from the same register file bank unless the register assignment algorithm is modified.

The modified register assignment policy implemented in the compiler attempts to assign every operand in the instruction to a different register file bank. With this modification, the number of ports in every register file bank is reduced since, on average, fewer accesses to a bank are performed in parallel.

The algorithm used by the compiler to assign the architectural registers is shown in Figure 3. The first available register in the free list is selected. This register is double-checked to be free and not system-reserved and, after that, compared with the registers assigned to the other operands of the instruction. If the register file bank for the operand under assignment coincides with any of the other operands of the instruction, this register is assigned to the next bank and the procedure is repeated. When the register is selected, the liveness of the register is calculated and the annotation is generated.

After the register assignment is completed, every operand of an instruction has been placed in a different register file bank.

The CPU assigns a free physical register to an instruction during the renaming process. This is performed by looking for and reserving a free register and properly initializing the fields in a hardware free register list. The approach presented in this work modifies the free list management by splitting the 256-position list of free register into four 64-position lists. The size of these FIFOs has to be large enough to provide rename buffers to the destination registers but avoiding to become empty[1]. In this way, the problem of mapping logical registers to a physical bank as anticipated by the compiler has been largely solved.

Summarizing, the banked register file architecture is supported by a banked free register list, which assures that the renamed registers belong to a specific register file bank, and by a modified compiler, which spreads the instruction operands among the different banks. Therefore, the assignment of physical registers is partially controlled by the compiler.

3.3 Low-Power Policy

The implementation of the register file in several banks can also be used to reduce the energy consumption in this device. The low-power control logic turns the banks not used by the executing instructions into a low-power state.

This low-power policy applies a voltage scaling technique to reduce the power consumption of the unused banks to a minimum. A dual voltage source is needed as well as the control logic to detect which banks should be turned off and when to perform this reconfiguration. The memory cells are turned into a low-power state which requires to reinstate the cell content one clock cycle before the actual access [8].

The control logic only needs to check the operand labels in the instruction to detect the unused banks (a detailed explanation of this logic and such approach can be read in [2]). The detection of the banks required by the instruction has to be done in advance for turning them on before the access. The unused banks remain in the low-power state significantly reducing the energy consumption.

Figure 4 shows the stages of an out-of-order pipeline. During the fetch phase, the instructions are read from the instruction memory and placed into one of the fetch buffers. If no fetch buffer is available, then fetch stalls. Next, instructions are decoded, after that renamed, and then placed

[1]The correctness of the 64-position choice has been validated by simulation.

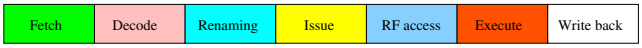

Figure 4: 5-stage pipeline

in the issue queue. When an instruction is enqueued, it may or may not have all of its operands, and it will wait in the issue queue until the availability of the operands. Once all the operands of an instruction are valid, they are read from the register file and sent to the execution units. Finally, the results of the instruction are written in the register file during the write back stage.

Because banked register file access takes less time, there should be enough time for turning the registers on before the access happens without any performance penalty. However, if this is not the case, the selection logic can be modified to generate bank ids for an even earlier bank wakeup.

4. IMPLEMENTATION AND EXPERIMENTAL RESULTS

The architecture previously described, as well as the new compiler modifications, have been implemented and simulated to verify the correctness of this approach and the energy savings obtained. The simulator has been derived from the SimpleScalar 4.0 tool suite [1]. SimpleScalar is an execution-driven simulator that implements a derivative of the MIPS-IV instruction set.

This section presents the results for the reduction in the number of ports in the banked register file, the energy savings obtained, and the energy savings also obtained when applying the low-power policy. Different configurations varying the number of banks are considered.

The simulated baseline configuration uses a multi-ported register file implementation with 256 registers and 6R/3 ports, e.g. it is configured for a 3-issue machine.

4.1 4-Bank Configuration

Initially, a 4-bank register file configuration was selected to analyze the impact of the proposed approach in the reduction of register file ports and energy consumption. This configuration has been used by the compiler to perform the modified register assignment, as was explained above. This initial point was selected to simplify the compiler implementation (the register assignment is easier when the number of banks is an integer divider of the number of architectural registers) and the expected energy savings were quite representative.

The baseline configuration for the 256-entry register file (Figure 2) with 6 read ports, 3 write ports is turned into four independent banks with 6 read ports, 3 write ports and 64 architectural registers per bank, but no power reduction is performed (to feed the 3-issue pipeline).

Next, the low-power policy is applied and analyzed. In this case, the unused register file banks per instruction are turned into a low-power state to reduce their power consumption to a minimum. In this case, the energy savings depend on the access pattern to the register file, the number of required operands per instruction, bypass logic, and, summarizing, the simulated source code. For the obtained results, no bypass policy has been considered.

For the 4-bank configuration with the low-power policy enabled, the set of benchmarks provided by the MiBench suite was simulated. Figure 5 shows the reduced energy consumption of the banked approach with respect to the traditional implementation, for a subset of benchmarks as well as the average energy consumption.

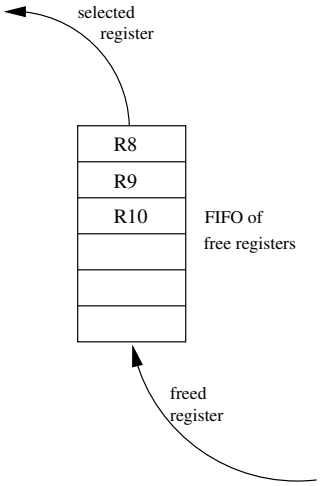

Figure 6: FIFO of free registers

As can be seen, the energy consumption is reduced to nearly 58%, on average, for the set of benchmarks, with most of them above 50% of the baseline consumption. As in the previous case when no low-power policy was applied, there is not performance penalty in splitting the register file into banks.

4.2 N-Bank Configuration

It is clear that increasing the number of sub-banks of the register file, higher energy savings can be expected since a bigger portion of the register file can be kept into the low-power state when it is not accessed. However, there are two main limitations to this idea. First, when increasing the number of sub-banks from one configuration to another, it may happen that the number of ports per bank has to be kept constant to meet the performance constraints and avoid pipeline stalls. The energy per access, which is related to the number of ports, does not change but there is a decrease in the energy savings due to the increased number of banks. Therefore, some energy penalty can appear when increasing the number of banks.

The second limitation is related to the number of live registers required by the application before any of them can be freed (*liveness*). When the compiler assigns registers during the register allocation phase, it takes them from a list (or FIFO) of free registers, and gives them back to the list when the live period of the register has finished and it can be freed (see Figure 6, where it is shown the reduced energy consumption of the banked approach with respect to the traditional implementation). If the compiler does not find any free register in the list it has to insert no-operation cycles waiting for available registers.

The analysis of the liveness of the registers assigned by the compiler shows the minimum size of the sub-banks to assure that they will never run out of free registers. This analysis is performed as a phase in the compilation with the unmodified compiler (the register assignment policy is not modified), and can be dumped into a text file containing

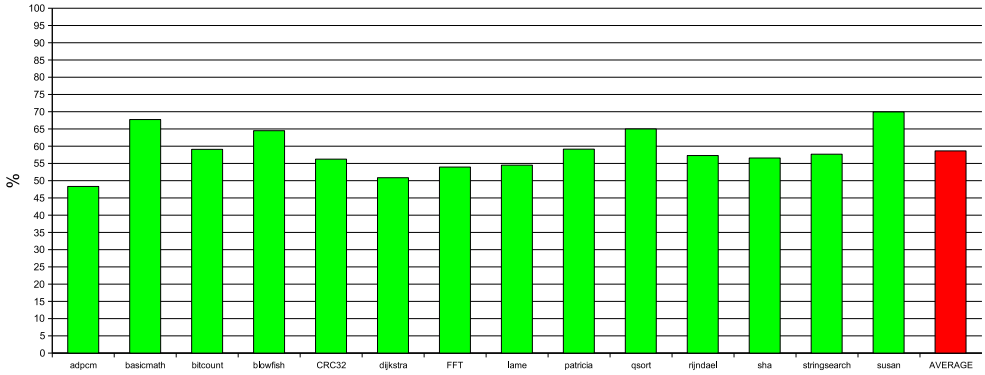

Figure 5: Energy consumption for the *4-bank configuration*

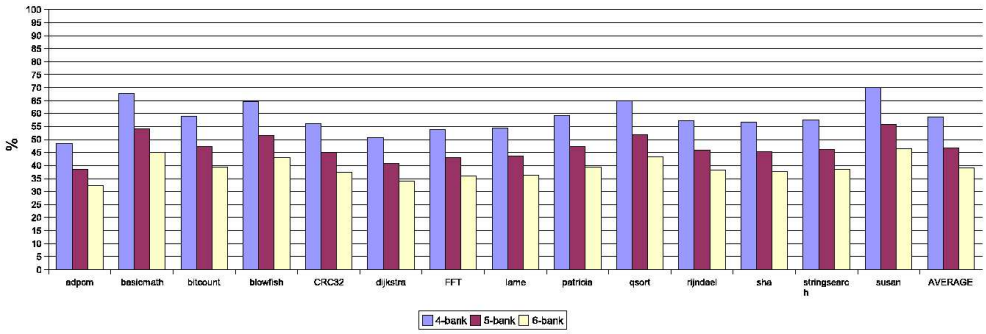

Figure 7: Energy consumption for the *N-bank configuration*

the liveness of every register and the beginning and ending instruction.

The presented analysis has been performed for the whole set of benchmarks used in the simulations, showing that the maximum number of banks that meets the performance constraints in all benchmark is six. Thus, two new configurations have been implemented and simulated to obtain the energy savings: *5-bank configuration* with 5 read and 3 write ports, and *6-bank configuration* with 4 read and 2 write ports.

Concerning the impact of the number of ports in the energy consumption, for the *5-bank configuration* the total energy consumption of the register file is reduced to a **72.37%** of the baseline configuration when splitting the register file into five banks. The *6-bank configuration* shows an increase in these savings (the total energy consumption is reduced to a **66.32%** of the baseline configuration). It should be noticed that configurations with more banks could not improve these numbers. For example, the 7-bank configuration reduces the energy consumption to a **86.84%** because the number of required ports per bank has to be kept equal to the 6-bank configuration (6 ports per bank in both cases). This is due to the first limitation on selecting the number of sub-banks, as was previously explained. It has also to be noticed that the complexity of the implementation and the access time to the register file have been dramatically reduced with these two last configurations.

Next, the low-power policy is applied to the *4-bank configuration*, to obtain greater energy savings. Figure 7 shows the resulting energy consumption of the register file for the three analyzed configurations when the low-power policy is applied. As can be observed, the average energy consumption is significantly reduced in both configurations since a bigger portion of the register file can be turn into the low-power state when it is not required by any instruction. Those benchmarks with less energy consumption correspond to those with more instructions requiring only one operand, while the consumption is increased when more multi-operand instructions are executed.

When increasing the number of register file banks (and thus, when decreasing the number of read and write ports) the pipeline could stall if the issue-width of the machine cannot be fed by a sufficient number of register file ports. In the selected configurations, such a behavior can be observed for the 6-bank configuration (Figure 8). As can be noticed, the performance penalty shown for the 6-bank configuration (2.3% on average) is in accordance with previous published works [7, 9].

It is interesting to remember that a configuration with no performance penalty and without modification of the pipeline behavior can be selected for efficient reduction in the number of register file ports and significant energy savings.

5. CONCLUSIONS

The growing complexity of the processor systems is reflected in the design of highly-parallel superscalar architectures. The performance of these processors relies on the availability of multi-ported register files that provide the operands to the parallel data-paths. The number of read

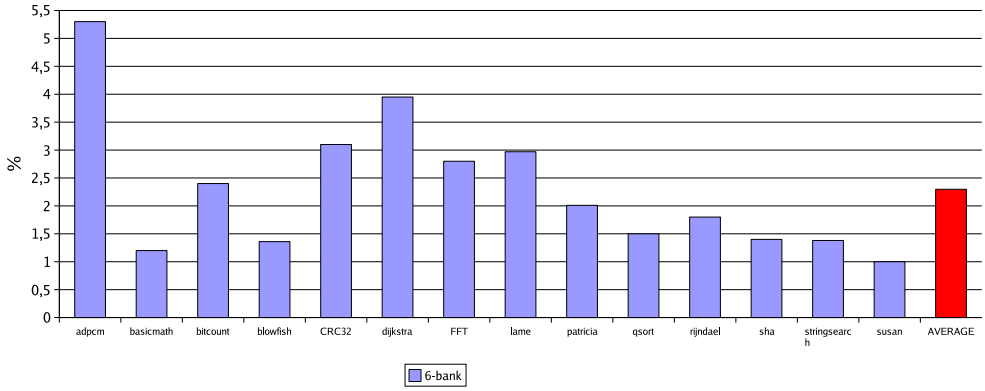

Figure 8: Performance penalty of the 6-bank register file

and write ports required by these memory structures makes impossible their implementation without a dramatic area and power overhead.

In this paper, a new approach based on a banked register file architecture and a low power policy has been presented. The described technique reduces the number of ports of the register file and provides important energy savings, as have been presented with experimental results. Both register file ports and energy reduction are obtained without a performance penalty.

6. REFERENCES

[1] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An Infrastructure for Computer System Modeling. *Computer*, 35(2):59–67, February 2002.

[2] J. L. Ayala, M. López-Vallejo, and A. Veidenbaum. Energy-efficient register renaming in high-performance processors. In *Workshop on Application Specific Processors*, 2003.

[3] R. Balasubramonian, S. Dwarkadas, and D. H. Albonesi. Reducing the complexity of the register file in dynamic superscalar processors. In *International Symposium on Microarchitecture*, 2001.

[4] E. Borch, S. Manne, J. Emer, and E. Tune. Loose loops sink chips. In *International Symposium on High-Performance Computer Architecture*, 2002.

[5] J. L. Cruz, A. González, M. Valero, and N. P. Topham. Multiple-banked register file architecture. In *nternational Symposium on Computer Architecture*, 2000.

[6] Digital. *Alpha 21264 Hardware Reference Manual*, 1999.

[7] K. I. Farkas, N. P. Jouppi, and P. Chow. Register file design considerations in dynamically scheduled processors. In *International Symposium on High Performance Computer Architecture*, 1996.

[8] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *International Symposium on Computer Architecture*, 2002.

[9] J. Janssen and H. Corporaal. Partitioned register file for TTAs. In *International Symposium on Microarchitecture*, 1995.

[10] K. Kailas, M. Franklin, and K. Ebcioglu. A partitioned register file architecture and compilation scheme for clustered ILP processors. In *International Euro-Par Conference*, 2002.

[11] N. S. Kim and T. Mudge. Reducing register ports using delayed write-back queues and operand pre-fetch. In *International Conference on Supercomputing*, 2003.

[12] G. Magklis, G. Semeraro, D. H. Albonesi, S. G. D. S. Dwarkadas, and M. L. Scott. Dynamic frequency and voltage scaling for a multiple-clock-domain microprocessor. *IEEE Micro*, 23(6):62–68, Nov.-Dec. 2003.

[13] S. Palacharla, N. P. Jouppi, and J. E. Smith. Complexity-effective superscalar processors. In *International Conference on Computer Architecture*, 1997.

[14] I. Park, M. D. Powell, and T. N. Vijaykumar. Reducing register ports for higher speed and lower energy. In *international symposium on Microarchitecture*, 2002.

[15] J. Pouwelse, K. Langendoen, and H. J. Sips. Application-directed voltage scaling. *IEEE Transactions on VLSI Systems*, 11(5):812–826, Oct. 2003.

[16] G. Reinman and N. Jouppi. An integrated cache timing and power model. Technical report, COMPAQWestern Research Lab, 1999.

[17] S. Rixner, W. J. Dally, B. Khailany, P. Mattson, U. J. Kapasi, and J. D. Owens. Register organization for media processing. In *International Symposium on High-Performance Computer Architecture*, 2000.

[18] J. H. Tseng and K. Asanović. Banked multiported register files for high-frequency superscalar microprocessors. In *International Symposium on Computer Architecture*, 2003.

[19] J. Zalamea, J. Llosa, E. Ayguadé, and M. Valero. Two-level hierarchical register file organization for VLIW processors. In *International Symposium on Microarchitecture*, 2000.

[20] V. Zyuban and P. Kogge. The energy complexity of register files. Technical report, University of Notre Dame, 1997.