Power-aware Register Renaming in High-Performance Processors using Compiler Support*

José L. Ayala, Marisa López-Vallejo
Dept. de Ingeniería Electrónica
Universidad Politécnica de Madrid
Spain
{jayala,marisa}@die.upm.es

Alexander Veidenbaum
Center for Embedded Computer Systems
University of California, Irvine
USA
alexv@ics.uci.edu

Abstract

This work presents an efficient multi-banked architecture of the register file, and a low-power compiler support which reduces energy consumption in this device by more than a 78%. The key idea of this work is based on a quasi-deterministic interpretation of the register assignment task, and the use of the voltage scaling techniques.

1 Introduction

Recently, energy consumption in embedded systems has become more and more important, mainly due to the fact that many systems are now being designed as mobile devices, i.e., they have to operate on battery power instead of using abundant power from wall sockets. One important aspect of such devices is their running time. Also, in high-performance systems, the total power dissipation of recently introduced microprocessors has been rapidly increasing, pushing desktop system cooling technology close to its limits.

Steadily increasing power consumption with each successive generation of processors has started to affect the system size and costs so adversely that this power/performance tradeoff has become increasingly difficult to justify in a competitive market. A look at a trend line borrowed from Intel (Figure 1) indicates the future power density of microprocessors exceeding the power density on the surface of the sun.

In pursuit of higher performance through higher clock rates and greater instruction level parallelism (ILP), modern microarchitectures are buffering an ever greater number of instructions in the pipeline. The larger window of in-flight instructions offers the microarchitecture hardware more opportunities to discover independent instructions to issue simultaneously. However, maintaining more instructions requires a corresponding increase in the buffering structures; in particular, a larger physical register file where generated results can be hold.

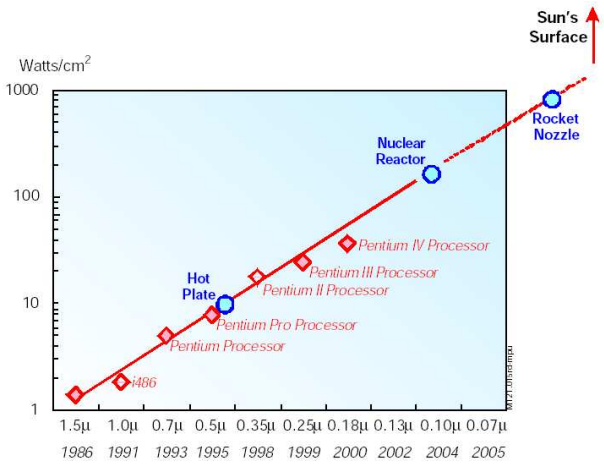

Figure 1: Power consumption trends in microprocessors

Multiported register files and bypass networks lie at the heart of a superscalar microprocessor core, providing buffered communication of register values between producer and consumer instructions. As issue widths increase, both the number of ports and the number of required registers increase, causing the area of a conventional multiported register file to grow more than quadratically with issue width [22]. The trend towards simultaneous multithreading also increases register count as separate architectural registers are needed for each thread. For example, the proposed eight-issue Alpha 21464 design had a register file that occupied over five times the area of the 64KB primary data cache. [6]

Many techniques have been proposed to reduce the area, energy, and delay of multiported register files. Some approaches split the microarchitecture into dis-

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

tributed clusters, each containing a subset of the register file and functional units. [7, 18, 23] These schemes have the potential to scale to larger issue widths but require complex control logic to map instructions to clusters and to handle inter-cluster dependencies. Alternatively, other approaches retain a centralized microarchitecture, but divide the physical register file into interleaved banks with fewer ports per bank [16, 2]. Provided the number of simultaneous accesses to any bank is less than the number of ports on each bank, the structure can provide the aggregate bandwidth needs of a superscalar machine with significantly reduced area compared to a fully multiported register file. These earlier banked schemes, however, require complex control logic with pipeline stalls that would likely limit the cycle time of a high-frequency design.

In this paper, we propose an innovative microarchitecture that based on compiler support achieves significant energy savings in the register file without any performance loss. The approach is suitable for deeply pipelined superscalar out-of-order processors. Our proposal is based on a deterministic variation of the renaming algorithm which allows to turn-off many registers of this structure if they are not used, with the corresponding energy savings. The required registers are turned-back on in advance and no performance penalty is accomplished.

For the functional implementation of this idea, a banked organization of the register file is used and the low-power state of the memory cells is accomplished by using *"drowsy"* memory cells [8]. The simple extra logic required for managing the banked register file does exhibit a reduced energy overhead. The approach is also supported by several experimental facts that show how the register assignment is performed by the compiler: related registers are assigned first. Compiler modifications allow us to improve the energy savings obtained by the proposed hardware structure.

The paper structure is as follows: section 2 presents the related work in this fied, section 3 covers the overview of the presented approach and the designed architecture is presented in 4. Finally, the experimental results are analyzed in section 5 and some conclusions are drawn in section 6.

2 Related Work

The energy complexity of register files has been carefully studied in the last years. In [22], Zyuban et al. compare various register file circuitry techniques for their energy efficiencies, as a function of the architectural parameters such as 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 following three axes reduces the cost of register storage and communication but with some performance impact. This work also develops a taxonomy of register architectures by partitioning and by optimizing the hierarchical register organization to operate on streams of data.

Some previous works have proposed hierarchical and banked architectures for the register file but in these cases, unlike our, the goal is to reduce the size and number of registers in the register file. Some examples are the works found in [2], [3] and [5]. Some compiler approaches have been also presented, like the work in [20] focused on VLIW processors.

These previous approaches oriented to reduce area and complexity of the register file usually require substantial changes to the pipeline and have the potential to create significant complications as noted in [16].

Compiler optimizations such as power-aware instruction selection, i.e., choosing the instruction sequence that will cause the lowest energy dissipation when the program is executed, will not require changes in the hardware to reduce energy consumption [15]. Often, optimizing for performance will result in a program that also yields good energy results. There are cases, however, where these two goals will result in different instruction sequences. The compiler optimization technique register pipelining is an example for this effect. This particular example can be found in [19].

Another compiler-based optimization for power is instruction scheduling. Subsequent instructions are ordered so as to incur as little change in circuit state as possible. Some techniques, aimed especially at very long instruction word (VLIW) architectures, are described in [13]. Besides these compiler-oriented optimizations, there is a further group of optimizations that combines architectural changes with modifications to the compiler.

One architectural parameter that also requires compiler support is the register file size. There have been several approaches to allow compilers to adapt to different target architectures by supplying them with architectural information about the target processor. In the Trimaran compiler [10], the machine description language MDES is used to model the underlying hardware. A different approach was taken in [14], where a compiler can be retargeted to different processors of the digital signal processor (DSP) domain described in MIMOLA [12], a VHDL-like hardware description language. One publication by Brooks et al. [4] presents a

framework enabling power analysis and investigation of the effect of hardware modifications as well as compiler optimizations. Their power analysis is based on parameterizable power models of common structures found in modern microprocessors. For VLIW architectures, Zalamea et al. [21] provide results concerning cycle time, area, and power consumption for register files of different sizes.

One of the closest works to ours is the register renaming by Jourdan et al. [11] The authors use register sharing to exploit value locality and reduce register presure in the Intel IA-32 architecture.

This paper extends previous works by proposing a hardware and software technique to efficiently reduce the energy consumption in the register file by means of simple extra logic (low overhead) and without any performance penalty.

3 Overview Approach

Our research work is oriented to the power reduction in processors with dynamic (hardware) register renaming, as usually can be found in high-performance processors. And the most common implementation of this technique is the *separate register file*, when rename buffers are implemented separately from the architectural registers (i.e, rename register values are held in a separate register file).

With i instructions or micro-operations issued per cycle each consuming (up to) two operands and producing (up to) one result, the register file needs to support $2i$ reads and i writes per cycle. These $2i+i$ registers have to be on to provide the source operands and store the results in a given cycle. But the rest of the N registers do not need to be on and are unused but energy-hungry resources. We will turn them in a low-power (*drowsy*) state.

The information in a memory cell is preserved while it is in the drowsy state. However, the data line must be restored to a high-energy mode before its contents can be accessed. One way to implement drowsy memory devices was proposed by Flautner et al. in [8], where a dynamic voltage scaling (DVS) technique is exploited to reduce static power consumption. The authors propose a memory cell architecture where the power supply can be reduced to save static power consumption while keeping the contents of the cell. The use of high threshold-voltage devices helps on reducing the leakage current powered from the cell.

The proposed solution deals with the "scalable" register file as follows. Instead of working with the whole register file, only the registers belonging to one (or more) of the pre-defined banks will be kept on. Once the register file bank that contains the required registers is known, the other registers are turned into a low power state by keeping the corresponding banks in the drowsy state. In this way, the power consumption is reduced to a minimum and the clock distribution network is gated as well.

Due to the necessity of restoring the register contents previously to the access, the accessed registers must be known at least one cycle before the access happens. However, the allocation of a free register is a non deterministic task and it is not possible to know in advance the rename buffer assigned to a destination register. The main goal of this work is to reduce this indetermination so that the required rename buffer can be known in advance to be turned on. For that, when a logical register is renamed to an architectural register, instead of using the whole register file, the target register will be selected from the bank, which reduces the intetermination to the registers inside the bank and allows to turn the other banks off.

4 New Architecture

The FIFO of free registers was split into smaller FIFOs. The size of these FIFOs has to be large enough to provide rename buffers to the destination registers but avoiding to become empty. In this way, the problem of indetermination has been mostly solved. Instead of assigning free rename buffers from the same FIFO to the whole set of logical registers, the first set of logical registers is assigned to the first FIFO, the second set to the second FIFO, and so on (Figure 2).

Splitting the FIFOs can represent a small penalty in terms of energy and area due to the need of extra logic. However, these area and power overheads are completely affordable, as was shown in [1].

Whenever a rename buffer has to be assigned to a destination register, the source FIFO providing the rename buffer is selected by simple logic. This logic selects the corresponding FIFO by means of the most significant bits of the logical registers coded in the instruction word (R_s, R_d and R_t). This simple logic has to be replicated for both source operands and for the destination registers, allowing in this way to turn the registers on for the read (decode stage) and write (write-back stage) of the operands. Also, those split FIFOs are designed with the right number of ports to allow concurrent access to the three possible operands of the executed instruction and the write back access of the former instruction in the pipeline.

In this way, for every single destination register, the set of permitted rename buffers are known in advance, allowing to keep the other registers off and this set on. The best execution case is when all the operands coded in the instruction word belong to the

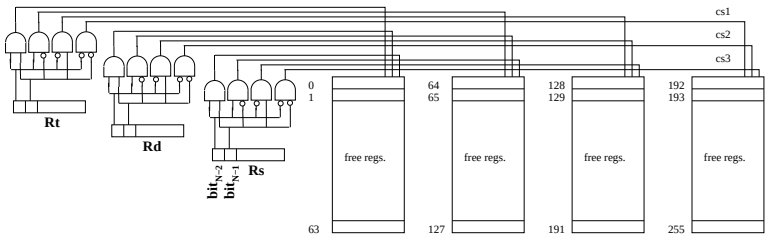

Figure 2: Modified allocation of free registers

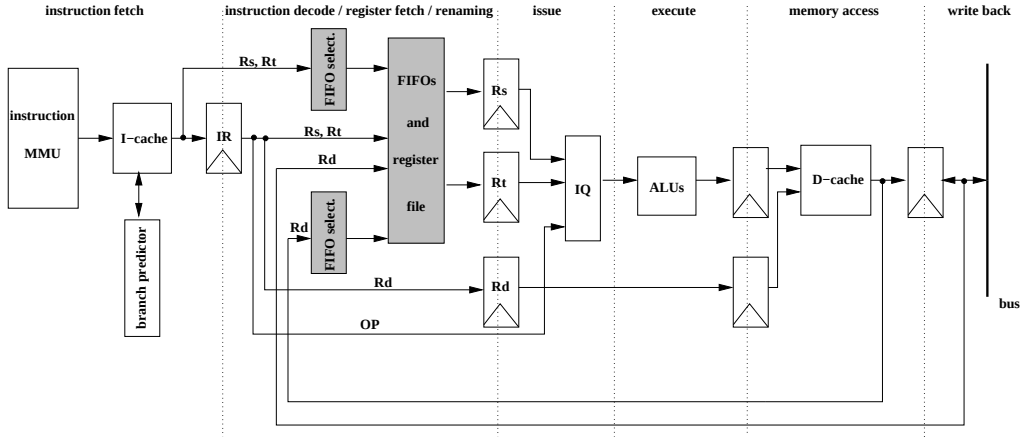

Figure 3: Schematic of the general architecture

same FIFO and the energy savings are the highest expected. The worst execution case happens when every operand read/written from/to the register file, and coded in the instruction words, belongs to different FIFOs (spread assignment). In this case, the expected savings are the lowest achievable.

During the fetch stage the instruction is loaded from I-cache into the instruction register. Our approach takes advantage of this fact by forwarding the operand fields of the instruction to the FIFO selection logic. This is possible due to the fixed instruction format used by RISC processors: the register designators are in a fixed position and can be easily extracted for the current instruction.

Figure 3 represents the general architecture of the processor supporting the proposed power aware register file. The shadowed modules are those modified or included in the presented approach to perform the energy-aware register renaming implementation. As can be seen, the FIFO selection logic has to be replicated to perform this task also during the write-back stage and, therefore, to turn the required register bank on before the write access.

Readers insterested in a more detailed description of the multi-bank hardware architecture are referred to [1], where this is shown in detail.

The multi-bank architecture that has been presented allows to exploit the low power capabilities of the split register file by turning the unused banks into a low power state. As much as the register file ussage is collapsed into a single bank per instruction, greater energy savings will be obtained. A low-power implementation of the register assigment task performed by the compiler will maximize this fact and will increase the energy savings achieved by the compiler.

Next section describes the design and implementation of this power-aware compiler.

4.1 Register Assignment

The compiler selected to perform the implementation of the improved register assignment policy is the GNU compiler gcc 3.2 which is publicly available. It generates high quality code and supports multiple embedded and high-performance processors.

The compiler performs a first register assignment by translating the logical registers coded in the intruction word, into the physical registers available in the hardware. This software assignment is later modified by the register renaming hardware to avoid hazards. The approach presented in the paper assumes a banked register renaming hardware like the one proposed in [1]

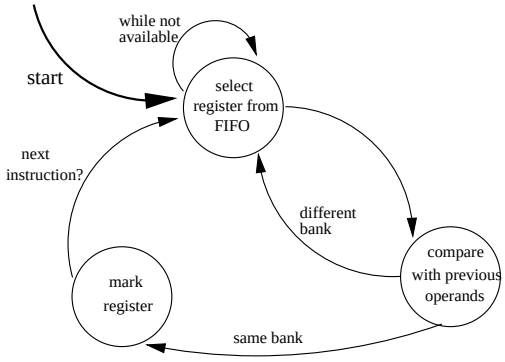

Figure 4: Register assignment algorithm

to save energy. In this way, the compiler decisions are not destroyed by the renaming mechanism because it restricts the renaming to the register file bank.

Gcc, when assigning an architectural register to the instruction operands, retrieves the first available register from a list (a FIFO) of free registers. The order of the registers inside the list is not representative and depends on the specific hardware architecture. Since gcc does not consider any restriction on assigning the registers, these are selected consecutively.

The register assignment policy implemented in the compiler modifies the traditional assignment by allocating every operand in the instruction to the same register file bank. With this modification, a higher number of unused banks per instruction can be kept in the low power state saving energy.

The algorithm followed by the compiler to assign the architectural registers is shown in Figure 4. First, the first available register in the list of free registers 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 does not coincide with any of the other operands of the instruction, this register is promoted 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, it results that every operand of the instruction has been disposed in the same register file bank for most of the instructions.

5 Experimental Results

The benchmarks used for the experiments belong to a precompiled set of the MiBench benchmark suite [9]. The benchmarks were run up to completion.

The baseline architecture selected for the experimentation has 256 rename buffers (and, therefore, the same number of positions in the FIFO of free registers), and 64 logical registers assigned by the compiler. This configuration represents a typical configuration for current-trend high-performance processors.

Different architectures have been simulated to analyze the effect of the size of the sub-bank on energy savings. In particular, 4-bank, 5-bank and 6-bank architectures have been studied. Architectures with more banks will not be able to provide the request operands without stalling the pipeline.

Figure 5 shows the maximal occupation of the FIFOs of free registers for the four configurations. As can be observed, the maximal occupation of these banks is always below the 100%. Therefore, the selection of the architecture configuration and size of the banks is correct (it does not cause pipeline stalls). Architectures with more banks are not able to provide the operands without stalling the pipeline (the occupation is above 100% for some benchmarks) and have not been analyzed in the experiments. Future research will work on adapting the register file architecture to the application requirements.

Before applying the power-aware algorithm for register assignment (compiler support), the energy savigns obtained with just the banked architecture were evaluated. Figure 6 shows the energy savings obtained for the four banked configurations when the compiler support is not used. The obtained results are quite representative, but in some cases they are fare away from the theoretical maximum due to the register assignment performed by the compiler. Therefore, the low-power compiler will help on improving these energy savings without any additional overhead.

The theoretical maximum for the energy savings would be achieved if all the operands per instruction belong to only one of the FIFOs. First column in Figure 7 shows the energy savings in the register file with the proposed approach for the 4-bank configuration. As can be seen, the achieved energy savings are above 70% on average and quite close to the theoretical maximum (75%). Differences between benchmarks and with the theoretical maximum[1] correspond to instructions that access spreadly assigned destination registers and cannot be mapped by the compiler into a single register file bank.

A similar analysis has been performed with the 5-bank architecture. Second column in Figure 7 shows the energy savings in the register file for this configuration. As can be seen the energy savings are increased due to the use of smaller banks which, when kept alive, demand less power.

[1] All the registers coded in the instruction word can be found in the same FIFO

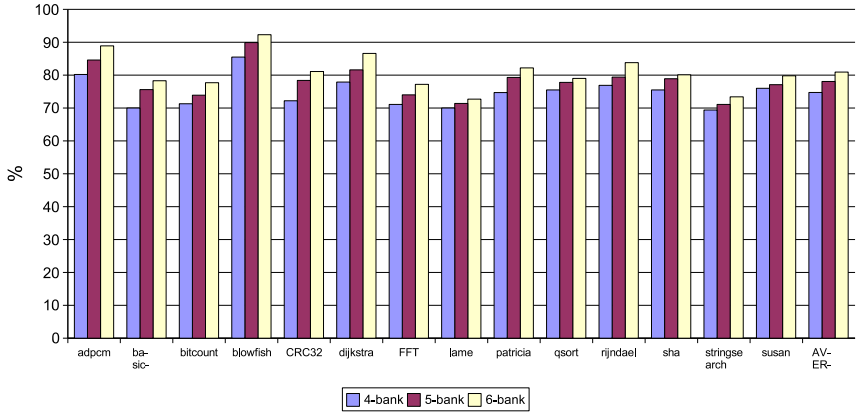

Figure 5: Maximal occupation of the banks

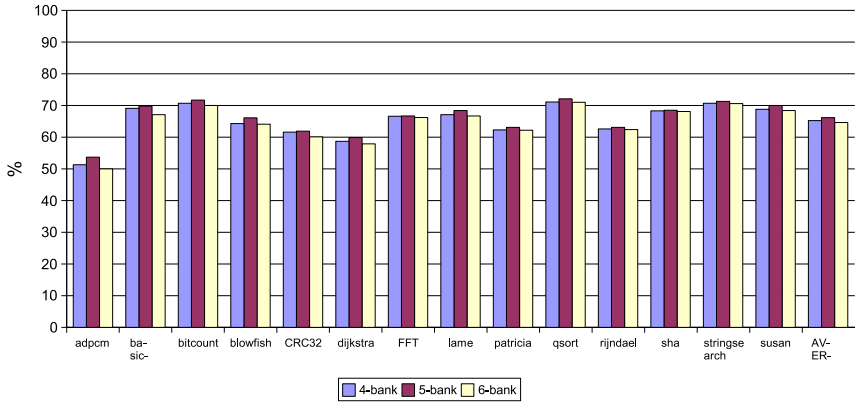

Figure 6: Energy savings in the register file without compiler support

Finally, the set of experiments has been simulated in a 6-bank architecture. Third column in Figure 7 shows the energy savings in the register file for this last configuration. In this case, most of the benchmarks present higher energy consumption than the previous configurations. For these benchmarks, the compiler fails on mapping the operands per intruction into a single bank and it has to use more than one register file bank (which represents more power consumption than the single bank assignment performed in previous configurations). Those benchmarks with reduced register ussage and where the compiler is able to perform the single bank assignment (basicmath, bitcount) exhibit lower energy consumption than the previous configurations.

Figure 7 summarizes the previous results in a single graph to allow the rapid comparison of the architectures.

6 Conclusions

In this work we have presented a combined hardware and software mechanism to efficiently reduce the energy consumption in the register file of out-of-order high-performance processors.

The approach proposes a multi-banked architecture and modifies the allocation of free registers, accomplished during the renaming stage, to reduce the indetermination of this task. Those unused banks are kept into a low power state thanks to the architectural modifications, while the compiler support increases the energy savings expected. The careful management of the access to the banks produces no performance penalty.

The approach has been simulated and validated with benchmarks from the MiBench suite, showing power savings above a 78% of the total power consumption for the register file.

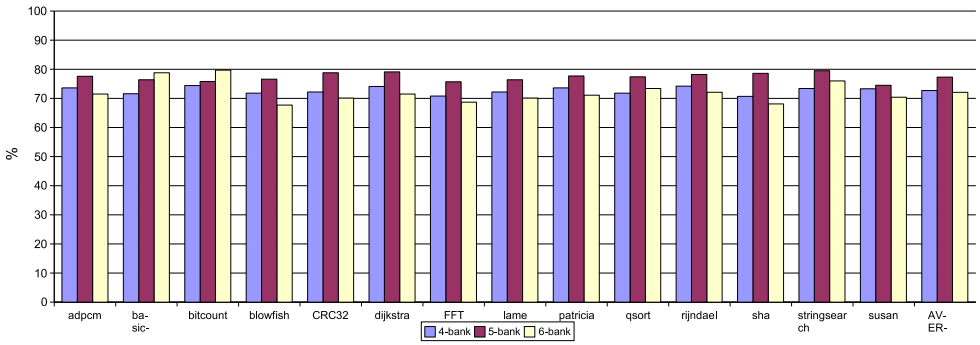

Figure 7: Energy savings in the register file

References

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

[2] 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.

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

[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations, set processor design and embedded systems. In *Int. Symp. Computer Architecture*, 2000.

[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] R. P. P. et al. Design of an 8-wide superscalar RISC microprocessor with simultaneous multithreading. *ISSCC Digest and Visuals Supplement*, 2002.

[7] K. I. Farkas, P. Chow, N. P. Jouppi, and Z. G. Vranesic. The multicluster architecture: Reducing cycle time through partitioning. In *MICRO*, 1997.

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

[9] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Annual Workshop on Workload Characterization*, 2001.

[10] T. Homepage. http://www.trimaran.org/, 2003.

[11] 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 *International Symposium on Microarchitecture*, 1998.

[12] R. Jhnk and P. Marwedel. *MIMOLA Reference Manual*. Dept. Comput. Sci. XII, Univ. Dortmund, Dortmund, Germany, 1993.

[13] C. Lee, J. Lee, T. Hwang, and S. Tsai. Compiler optimization on instruction scheduling for low power. In *Proc. 13th Int. Symp. System Synthesis*, 2000.

[14] R. Leupers. *Retargetable Code Generation for Digital Signal Processors*. Kluwer Academic Publishers, 1997.

[15] A. Parikh, S. Kim, M. Kandemir, N. Vijaykrishnan, and M. Irwin. Instruction scheduling for low power. *Journal of VLSI Signal Processing*, (37):129–149, 2004.

[16] I. Park, M. D. Powell, and T. N. Vijaykumar. Reducing register ports for higher speed and lower energy. In *MICRO*, 2002.

[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] A. Seznec, E. Toullec, and O. Rochecouste. Register write specialization register read specialization:

A path to complexity-effective wide-issue superscalar processors. In *MICRO*, 2002.

[19] S. Steinke, R. Schwarz, L. Wehmeyer, and P. Marwedel. Low power code generation for a RISC processor by register pipelining. Technical report, Dept. Comput. Sci. XII, Univ. Dortmund, Dortmund, Germany, 2001.

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

[21] J. Zalamea, J. Llosa, E. Ayguadé, and M. Valero. Software and hardware techniques to optimize register file utilization in VLIW architectures. In *Int. Workshop Advanced Compiler Technology for High Performance and Embedded Systems*, 2001.

[22] V. V. Zyuban and P. M. Kogge. The energy complexity of register files. In *International Symposium on Low Power Electronics and Design*, 1998.

[23] V. V. Zyuban and P. M. Kogge. Inherently lower-power highe-performance superscalar architectures. *IEEE Transactions on Computers*, 50(3):268–285, March 2001.