Babak Falsafi
T. N. Vijaykumar (Eds.)

LNCS 3471

Power-Aware Computer Systems

4th International Workshop, PACS 2004
Portland, OR, USA, December 2004
Revised Selected Papers

Springer

Babak Falsafi T.N. Vijaykumar (Eds.)

Power-Aware Computer Systems

4th International Workshop, PACS 2004
Portland, OR, USA, December 5, 2004
Revised Selected Papers

Springer

Volume Editors

Babak Falsafi
Carnegie Mellon University
Electrical and Computer Engineering Dept.
A305 Hamerschlag Hall, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA
E-mail: babak@cmu.edu

T.N. Vijaykumar
Purdue University
School of Electrical and Computer Engineering
Department of Computer Science
ECE/EE 465 Northwestern Avenue, West Lafayette, Indiana 47907-1285, USA
E-mail: vijay@ecn.purdue.edu

Library of Congress Control Number: 2005936777

CR Subject Classification (1998): B.7, B.8, C.1, C.2, C.3, C.4, D.4

ISSN	0302-9743
ISBN-10	3-540-29790-1 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-29790-1 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springeronline.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11574859 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 3471

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Preface

Welcome to the proceedings of the Power-Aware Computer Systems (PACS 2004) workshop held in conjunction with the 37th Annual International Symposium on Microarchitecture (MICRO-37). The continued increase of power and energy dissipation in computer systems has resulted in higher cost, lower reliability, and reduced battery life in portable systems. Consequently, power and energy have become first-class constraints at all layers of modern computer systems. PACS 2004 is the fourth workshop in its series to explore techniques to reduce power and energy at all levels of computer systems and brings together academic and industry researchers.

The papers in these proceedings span a wide spectrum of areas in power-aware systems. We have grouped the papers into the following categories: (1) microarchitecture- and circuit-level techniques, (2) power-aware memory and interconnect systems, and (3) frequency- and voltage-scaling techniques.

The first paper in the microarchitecture group proposes banking and write-back filtering to reduce register file power. The second paper in this group optimizes both delay and power of the issue queue by packing two instructions in each issue queue entry and by memorizing upper-order bits of the wake-up tag. The third paper proposes bit slicing the datapath to exploit narrow width operations, and the last paper proposes to migrate application threads from one core to another in a multi-core chip to address thermal problems.

The second group of papers on power-aware memory and interconnects starts with a contribution which proposes hardware–software co-operation to reduce main memory power dissipation. The paper suggests combining process-level information from the software and DRAM-bank-level information from the hardware for significant power reduction. The second paper in this group uses compiler-assist to make hardware prefetching more energy efficient by filtering out unnecessary and ineffective prefetching. The third paper explores modeling of external bus power dissipation and evaluation of coding techniques for bus power reduction. The last paper proposes context-independent coding for reducing power in off-chip interconnects to avoid the disadvantage of context-dependent coding not being applicable to commodity memories because of requiring collaboration between the memory controller and SDRAM.

The last group proposes frequency- and voltage-scaling techniques. The first paper in this group recommends throttling processor clock speed during low-utilization phases. The second paper scales the processor voltage according to the CPU-boundedness of the application. The third paper investigates the potential of hardware overprovisioning to increase throughput in data centers while remaining within a power budget. The last paper shows a detailed breakdown of power consumption in the various components of a modern laptop.

The success of PACS 2004 has been due to the high quality of the submissions, the efforts of the Program Committee, and the attendees. We would like to thank Vivek De for his informative keynote address, which described design challenges and opportunities for power-limited microprocessors. We would also like to thank Jose Gonzalez, Glen Reinman, Srikanth Srinivasan and other members of the MICRO-37 Organizing Committee who helped arrange the local accommodation and publicize the workshop.

December 2004

Babak Falsafi
T. N. Vijaykumar

PACS 2004 Program Committee

Babak Falsafi, *Carnegie Mellon University (Co-chair)*
T. N. Vijaykumar, *Purdue University (Co-chair)*

David Albonesi, *Cornell University*
Csaba Andras Mortiz, *University of Massuchusetts*
Krste Asanovic, *Massachusetts Institute of Technology*
Luca Benini, *Universitá di Bologna*
Frederic Chong, *University of California, Davis*
Kanad Ghose, *State University of New York, Binghamton*
Christoforos Kozyrakis, *Stanford University*
Uli Kremer, *Rutgers University*
Charles Lefurgy, *IBM, Austin Research Lab*
Yung-Hsiang Lu, *Purdue University*
Avi Mendelson, *Intel, Israel Microarchitecture Lab*
Andreas Moshovos, *University of Toronto*
Daniel Mosse, *University of Pittsburgh*
Vijaykrishnan Narayanan, *Pennsylvania State University*
Li-Shiuan Peh, *Princeton University*
Parthasarathy Ranganathan, *HP Labs*
Eric Rotenberg, *North Carolina State University*
Mircea Stan, *University of Virginia*
Se-Hyun Yang, *Samsung*

Lecture Notes in Computer Science

For information about Vols. 1–3712

please contact your bookseller or Springer

Vol. 3835: G. Sutcliffe, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning. XIV, 744 pages. 2005. (Subseries LNAI).

Vol. 3821: R. Ramanujam, S. Sen (Eds.), FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science. XIV, 566 pages. 2005.

Vol. 3818: S. Grumbach, L. Sui, V. Vianu (Eds.), Advances in Computer Science – ASIAN 2005. XIII, 294 pages. 2005.

Vol. 3814: M. Maybury, O. Stock, W. Wahlster (Eds.), Intelligent Technologies for Interactive Entertainment. XV, 342 pages. 2005. (Subseries LNAI).

Vol. 3809: S. Zhang, R. Jarvis (Eds.), AI 2005: Advances in Artificial Intelligence. XXVII, 1344 pages. 2005. (Subseries LNAI).

Vol. 3808: C. Bento, A. Cardoso, G. Dias (Eds.), Progress in Artificial Intelligence. XVIII, 704 pages. 2005. (Subseries LNAI).

Vol. 3807: M. Dean, Y. Guo, W. Jun, R. Kaschek, S. Krishnaswamy, Z. Pan, Q.Z. Sheng (Eds.), Web Information Systems Engineering – WISE 2005 Workshops. XV, 275 pages. 2005.

Vol. 3806: A.H. H. Ngu, M. Kitsuregawa, E.J. Neuhold, J.-Y. Chung, Q.Z. Sheng (Eds.), Web Information Systems Engineering – WISE 2005. XXI, 771 pages. 2005.

Vol. 3805: G. Subsol (Ed.), Virtual Storytelling. XII, 289 pages. 2005.

Vol. 3804: G. Bebis, R. Boyle, D. Koracin, B. Parvin (Eds.), Advances in Visual Computing. XX, 755 pages. 2005.

Vol. 3803: S. Jajodia, C. Mazumdar (Eds.), Information Systems Security. XI, 342 pages. 2005.

Vol. 3799: M. A. Rodríguez, I.F. Cruz, S. Levashkin, M.J. Egenhofer (Eds.), GeoSpatial Semantics. X, 259 pages. 2005.

Vol. 3798: A. Dearle, S. Eisenbach (Eds.), Component Deployment. X, 197 pages. 2005.

Vol. 3796: N.P. Smart (Ed.), Cryptography and Coding. XI, 461 pages. 2005.

Vol. 3795: H. Zhuge, G.C. Fox (Eds.), Grid and Cooperative Computing - GCC 2005. XXI, 1203 pages. 2005.

Vol. 3793: T. Conte, N. Navarro, W.-m.W. Hwu, M. Valero, T. Ungerer (Eds.), High Performance Embedded Architectures and Compilers. XIII, 317 pages. 2005.

Vol. 3792: I. Richardson, P. Abrahamsson, R. Messnarz (Eds.), Software Process Improvement. VIII, 215 pages. 2005.

Vol. 3791: A. Adi, S. Stoutenburg, S. Tabet (Eds.), Rules and Rule Markup Languages for the Semantic Web. X, 225 pages. 2005.

Vol. 3790: G. Alonso (Ed.), Middleware 2005. XIII, 443 pages. 2005.

Vol. 3789: A. Gelbukh, Á. de Albornoz, H. Terashima-Marín (Eds.), MICAI 2005: Advances in Artificial Intelligence. XXVI, 1198 pages. 2005. (Subseries LNAI).

Vol. 3788: B. Roy (Ed.), Advances in Cryptology - ASIACRYPT 2005. XIV, 703 pages. 2005.

Vol. 3785: K.-K. Lau, R. Banach (Eds.), Formal Methods and Software Engineering. XIV, 496 pages. 2005.

Vol. 3784: J. Tao, T. Tan, R.W. Picard (Eds.), Affective Computing and Intelligent Interaction. XIX, 1008 pages. 2005.

Vol. 3781: S.Z. Li, Z. Sun, T. Tan, S. Pankanti, G. Chollet, D. Zhang (Eds.), Advances in Biometric Person Authentication. XI, 250 pages. 2005.

Vol. 3780: K. Yi (Ed.), Programming Languages and Systems. XI, 435 pages. 2005.

Vol. 3779: H. Jin, D. Reed, W. Jiang (Eds.), Network and Parallel Computing. XV, 513 pages. 2005.

Vol. 3777: O.B. Lupanov, O.M. Kasim-Zade, A.V. Chaskin, K. Steinhöfel (Eds.), Stochastic Algorithms: Foundations and Applications. VIII, 239 pages. 2005.

Vol. 3775: J. Schönwälder, J. Serrat (Eds.), Ambient Networks. XIII, 281 pages. 2005.

Vol. 3773: A. Sanfeliu, M.L. Cortés (Eds.), Progress in Pattern Recognition, Image Analysis and Applications. XX, 1094 pages. 2005.

Vol. 3772: M. Consens, G. Navarro (Eds.), String Processing and Information Retrieval. XIV, 406 pages. 2005.

Vol. 3771: J.M.T. Romijn, G.P. Smith, J. van de Pol (Eds.), Integrated Formal Methods. XI, 407 pages. 2005.

Vol. 3770: J. Akoka, S.W. Liddle, I.-Y. Song, M. Bertolotto, I. Comyn-Wattiau, W.-J. van den Heuvel, M. Kolp, J. Trujillo, C. Kop, H.C. Mayr (Eds.), Perspectives in Conceptual Modeling. XXII, 476 pages. 2005.

Vol. 3768: Y.-S. Ho, H.J. Kim (Eds.), Advances in Multimedia Information Processing - PCM 2005, Part II. XXVIII, 1088 pages. 2005.

Vol. 3767: Y.-S. Ho, H.J. Kim (Eds.), Advances in Multimedia Information Processing - PCM 2005, Part I. XXVIII, 1022 pages. 2005.

Vol. 3766: N. Sebe, M.S. Lew, T.S. Huang (Eds.), Computer Vision in Human-Computer Interaction. X, 231 pages. 2005.

Vol. 3765: Y. Liu, T. Jiang, C. Zhang (Eds.), Computer Vision for Biomedical Image Applications. X, 563 pages. 2005.

Vol. 3764: S. Tixeuil, T. Herman (Eds.), Self-Stabilizing Systems. VIII, 229 pages. 2005.

Vol. 3762: R. Meersman, Z. Tari, P. Herrero (Eds.), On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops. XXXI, 1228 pages. 2005.

Vol. 3761: R. Meersman, Z. Tari (Eds.), On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, Part II. XXVII, 653 pages. 2005.

Vol. 3760: R. Meersman, Z. Tari (Eds.), On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, Part I. XXVII, 921 pages. 2005.

Vol. 3759: G. Chen, Y. Pan, M. Guo, J. Lu (Eds.), Parallel and Distributed Processing and Applications - ISPA 2005 Workshops. XIII, 669 pages. 2005.

Vol. 3758: Y. Pan, D.-x. Chen, M. Guo, J. Cao, J.J. Dongarra (Eds.), Parallel and Distributed Processing and Applications. XXIII, 1162 pages. 2005.

Vol. 3757: A. Rangarajan, B. Vemuri, A.L. Yuille (Eds.), Energy Minimization Methods in Computer Vision and Pattern Recognition. XII, 666 pages. 2005.

Vol. 3756: J. Cao, W. Nejdl, M. Xu (Eds.), Advanced Parallel Processing Technologies. XIV, 526 pages. 2005.

Vol. 3754: J. Dalmau Royo, G. Hasegawa (Eds.), Management of Multimedia Networks and Services. XII, 384 pages. 2005.

Vol. 3753: O.F. Olsen, L.M.J. Florack, A. Kuijper (Eds.), Deep Structure, Singularities, and Computer Vision. X, 259 pages. 2005.

Vol. 3752: N. Paragios, O. Faugeras, T. Chan, C. Schnörr (Eds.), Variational, Geometric, and Level Set Methods in Computer Vision. XI, 369 pages. 2005.

Vol. 3751: T. Magedanz, E.R. M. Madeira, P. Dini (Eds.), Operations and Management in IP-Based Networks. X, 213 pages. 2005.

Vol. 3750: J.S. Duncan, G. Gerig (Eds.), Medical Image Computing and Computer-Assisted Intervention – MICCAI 2005, Part II. XL, 1018 pages. 2005.

Vol. 3749: J.S. Duncan, G. Gerig (Eds.), Medical Image Computing and Computer-Assisted Intervention – MICCAI 2005, Part I. XXXIX, 942 pages. 2005.

Vol. 3748: A. Hartman, D. Kreische (Eds.), Model Driven Architecture – Foundations and Applications. IX, 349 pages. 2005.

Vol. 3747: C.A. Maziero, J.G. Silva, A.M.S. Andrade, F.M.d. Assis Silva (Eds.), Dependable Computing. XV, 267 pages. 2005.

Vol. 3746: P. Bozanis, E.N. Houstis (Eds.), Advances in Informatics. XIX, 879 pages. 2005.

Vol. 3745: J.L. Oliveira, V. Maojo, F. Martín-Sánchez, A.S. Pereira (Eds.), Biological and Medical Data Analysis. XII, 422 pages. 2005. (Subseries LNBI).

Vol. 3744: T. Magedanz, A. Karmouch, S. Pierre, I. Venieris (Eds.), Mobility Aware Technologies and Applications. XIV, 418 pages. 2005.

Vol. 3742: J. Akiyama, M. Kano, X. Tan (Eds.), Discrete and Computational Geometry. VIII, 213 pages. 2005.

Vol. 3740: T. Srikanthan, J. Xue, C.-H. Chang (Eds.), Advances in Computer Systems Architecture. XVII, 833 pages. 2005.

Vol. 3739: W. Fan, Z.-h. Wu, J. Yang (Eds.), Advances in Web-Age Information Management. XXIV, 930 pages. 2005.

Vol. 3738: V.R. Syrotiuk, E. Chávez (Eds.), Ad-Hoc, Mobile, and Wireless Networks. XI, 360 pages. 2005.

Vol. 3735: A. Hoffmann, H. Motoda, T. Scheffer (Eds.), Discovery Science. XVI, 400 pages. 2005. (Subseries LNAI).

Vol. 3734: S. Jain, H.U. Simon, E. Tomita (Eds.), Algorithmic Learning Theory. XII, 490 pages. 2005. (Subseries LNAI).

Vol. 3733: P. Yolum, T. Güngör, F. Gürgen, C. Özturan (Eds.), Computer and Information Sciences - ISCIS 2005. XXI, 973 pages. 2005.

Vol. 3731: F. Wang (Ed.), Formal Techniques for Networked and Distributed Systems - FORTE 2005. XII, 558 pages. 2005.

Vol. 3729: Y. Gil, E. Motta, V. R. Benjamins, M.A. Musen (Eds.), The Semantic Web – ISWC 2005. XXIII, 1073 pages. 2005.

Vol. 3728: V. Paliouras, J. Vounckx, D. Verkest (Eds.), Integrated Circuit and System Design. XV, 753 pages. 2005.

Vol. 3727: M. Barni, J. Herrera Joancomartí, S. Katzenbeisser, F. Pérez-González (Eds.), Information Hiding. XII, 414 pages. 2005.

Vol. 3726: L.T. Yang, O.F. Rana, B. Di Martino, J.J. Dongarra (Eds.), High Performance Computing and Communications. XXVI, 1116 pages. 2005.

Vol. 3725: D. Borrione, W. Paul (Eds.), Correct Hardware Design and Verification Methods. XII, 412 pages. 2005.

Vol. 3724: P. Fraigniaud (Ed.), Distributed Computing. XIV, 520 pages. 2005.

Vol. 3723: W. Zhao, S. Gong, X. Tang (Eds.), Analysis and Modelling of Faces and Gestures. XI, 4234 pages. 2005.

Vol. 3722: D. Van Hung, M. Wirsing (Eds.), Theoretical Aspects of Computing – ICTAC 2005. XIV, 614 pages. 2005.

Vol. 3721: A.M. Jorge, L. Torgo, P.B. Brazdil, R. Camacho, J. Gama (Eds.), Knowledge Discovery in Databases: PKDD 2005. XXIII, 719 pages. 2005. (Subseries LNAI).

Vol. 3720: J. Gama, R. Camacho, P.B. Brazdil, A.M. Jorge, L. Torgo (Eds.), Machine Learning: ECML 2005. XXIII, 769 pages. 2005. (Subseries LNAI).

Vol. 3719: M. Hobbs, A.M. Goscinski, W. Zhou (Eds.), Distributed and Parallel Computing. XI, 448 pages. 2005.

Vol. 3718: V.G. Ganzha, E.W. Mayr, E.V. Vorozhtsov (Eds.), Computer Algebra in Scientific Computing. XII, 502 pages. 2005.

Vol. 3717: B. Gramlich (Ed.), Frontiers of Combining Systems. X, 321 pages. 2005. (Subseries LNAI).

Vol. 3716: L. Delcambre, C. Kop, H.C. Mayr, J. Mylopoulos, Ó. Pastor (Eds.), Conceptual Modeling – ER 2005. XVI, 498 pages. 2005.

Vol. 3715: E. Dawson, S. Vaudenay (Eds.), Progress in Cryptology – Mycrypt 2005. XI, 329 pages. 2005.

Vol. 3714: H. Obbink, K. Pohl (Eds.), Software Product Lines. XIII, 235 pages. 2005.

Vol. 3713: L.C. Briand, C. Williams (Eds.), Model Driven Engineering Languages and Systems. XV, 722 pages. 2005.

Table of Contents

Microarchitecture- and Circuit-Level Techniques

Power-Aware Memory and Interconnect Systems

Frequency-/Voltage-Scaling Techniques

An Optimized Front-End Physical Register File with Banking and Writeback Filtering

Miquel Pericàs[1,3], Ruben Gonzalez[1], Adrian Cristal[1],
Alex Veidenbaum[2], and Mateo Valero[1,3]

[1] Computer Architecture Department, Technical University of Catalonia (UPC)
[2] Information and Computer Science, University of California at Irvine (UCI)
[3] Barcelona Supercomputing Center (BSC)
{mpericas, gonzalez, adrian, mateo}@ac.upc.edu,
alexv@matrix.ics.uci.edu

Abstract. Register file design is one of the critical issues facing designers of out–of–order processors. Scaling up its size and number of ports with issue width and instruction window size is difficult in terms of both performance and power consumption. Two types of register file architectures have been proposed in the past: a future logical file and a centralized physical file.

The centralized register file does not scale well but allows fast branch mis–prediction recovery. The Future File scales well, but requires reservation stations and has slow mis–prediction recovery. This paper proposes a register file architecture that combines the best features of both approaches. The new register file has the large size of the centralized file and its ability to quickly recover from branch misprediction. It has the advantage of the future file in that it is accessed in the "front end" allowing about 1/3rd of the source operands that are ready when an instruction enters the window to be read immediately. The remaining operands come from bypass logic / instruction queues and do not require register file access. The new architecture does require reservation stations for operand storage and it investigates two approaches in terms of power–efficiency.

Another advantage of the new architecture is that banking is much easier to use in this case as compared to the centralized register file. Banking further improves the scalability of the new architecture. A technique for early release of short–lived registers called *writeback filtering* is used in combination with banking to further improve the new architecture. The use of a large front–end register file results in significant power savings and a slight IPC degradation (less than 1%). Overall, the resulting energy–delay product is lower than in previous proposals.

1 Introduction

Memory-based structures in the core of modern microprocessors have increasing energy requirements as frequencies grow. One such structure is the register file. Its size and the number of read/write ports required increases with issue width making it difficult to implement at high clock frequencies.

Two main approaches to register file design were used in the past, neither of which solved the above-mentioned problems. One approach was an architecture based on the Future file, which has a logical register file updated in commit and the future register

B. Falsafi and T.N. Vijaykumar (Eds.): PACS 2004, LNCS 3471, pp. 1–14, 2005.

file in the "front–end" holding the most recent, uncommitted value for each logical register. The advantages of the future file are that it is not very large, has no renaming, can be read in the front–end and is not written if a more recent instruction assigning it is in the window. The disadvantages are that on branch mis–prediction, intermediate register values need to be recovered (typically after the mis–predicted branch commits), it needs reservation stations in the back-end, and its size cannot be increased. The mis–prediction recovery can lead to a significant IPC loss, especially given increasing memory latencies.

An alternative approach is a single, large physical register file, without a separate architectural register file. It is typically accessed after an instruction is scheduled to execute, even if source operand values were available when the instruction entered the window. This is the approach in the MIPS R10000 [1] and many later processors. Its advantages are increased size and fast mis–prediction recovery. Disadvantages are more complex renaming and longer value lifetime in the file due to lack of logical register file. Overall, it needs to be both large and heavily multi–ported, making it difficult to implement and increases its energy consumption significantly.

The new architecture proposed in this paper combines the best features of the two above–mentioned approaches: arbitrary size and fast mis–prediction recovery of the physical register file; and placement in the front–end, early operand read, and potential lack of write–back of the future file. It can be thought of as a physical register file moved to the front end and accessed after renaming. This allows a large fraction of operands to be accessed as an instruction enters the window, which is now the only read access to the register file. These values are stored in "reservation stations" integrated into the instruction queue, which can also be thought of as a replicated portion of the register file. A value coming from writeback may be written to this file if there are instructions waiting for it. Finally, many registers hold values for mis–prediction recovery, some of which can be released if they cannot affect recovery.

The approach proposed here uses a single register file containing all physical registers, the Front-end Physical Register File (FPRF). Thus restarting execution after a mispredicted branch can be done using a rename map recovery from check-points made on conditional branches.

As source operand registers are renamed, it can be determined if a register value has already been computed. The FPRF is read only in this case, significantly reducing its access frequency. Combined with the higher IPC due to faster branch recovery, it has a better energy-delay product compared to the two traditional approaches.

A new structure to hold such "early read" values is created in the instruction queue payload RAM. Its function is similar to that of reservation stations. It is smaller than the physical register file and thus consumes less energy. It is written into by completing instructions, if the produced value is a source operand of a waiting instruction.

This paper also investigate the use of banking in the FPRF architecture. Due to lower access frequency of the FPRF this is much easier to do than in a standard centralized physical register file Finally, writeback filtering, a technique to eliminate unnecessary writebacks into the register file is investigated and shown to be quite effective.

2 Related Work

The body of related work on register file energy optimization is large. Many recent papers have proposed mechanisms to reduce the number of the ports by means of modifying the register file architecture, such as [4] [5] [6] [7]. A reduced number of ports may be more efficient both in terms of energy and access time, which can improve performance.

A different approach is to reorganize the registers into several files, concentrating most activity on small files with low power consumption. [8] is an example of this approach based on the isolation of narrow operands. Hierarchical register files, such as those presented in [9] [10] [11] and clustering techniques such as [12] [5] are another example of this technique, which effectively trades size, speed and power consumption.

Another research direction has focused on changing the register allocation algorithm to reduce the register requirements of the architecture. *Early Release* frees registers before the commit stage of the next instruction that writes to the same logical registers [13] [14] [15]. *Virtual registers* [16] try to delay the allocation of the physical register until the writeback stage of the instruction. Another approach to reduce registers is to exploit repeated values in the registers [17] [18].

Our approach is somewhat based on the *Future File* organization which was proposed in 1984 [19]. In the original proposal, operands are provided to instructions via a logical register file in the front-end which received the name of *Future File*. The main difference with our architecture is that we are basing our design around the concept of physical registers to identify the state of the processor. Thus, while a Future File architecture can only recover from a mispredicted branch by draining the ROB, our proposed architecture can recover directly from the physical registers. Future File architectures are still being used in the form of the AMD K7 and K8 microarchitectures [20].

The future file can be extended with rename buffers to provide access to the full processor state at once. This has been implemented in the PPC620 [3] and POWER3 processors. However, these two processors still require the architectural state to be copied from the rename buffers during retirement. Having an architectural register file in the front-end shortens the pipeline one stage (access can be performed in parallel to rename stage) but increases the number of on-chip register transfers.

Research by Tseng et al. on banked register files [21] proposed an efficient implementation of banking for the register file of a MIPS R10000-like architecture. In the following sections is will be compared to the architecture proposed in this paper.

Finally, the *Writeback Filtering* technique, based on the release of short-lived values, is described in [22] and, in the context of VLIW architectures, in [23]. However, as will be shown here, the specific architectecture presented here allows to support *Writeback Filtering* in innovative ways.

3 Front-End Physical Register File

This section describes the *Front-End Physical Register File Architecture* (FPRF). The FPRF pipeline provides instructions with their operands as soon as the operands are available. Further, it implements a central physical register file in the front-end that

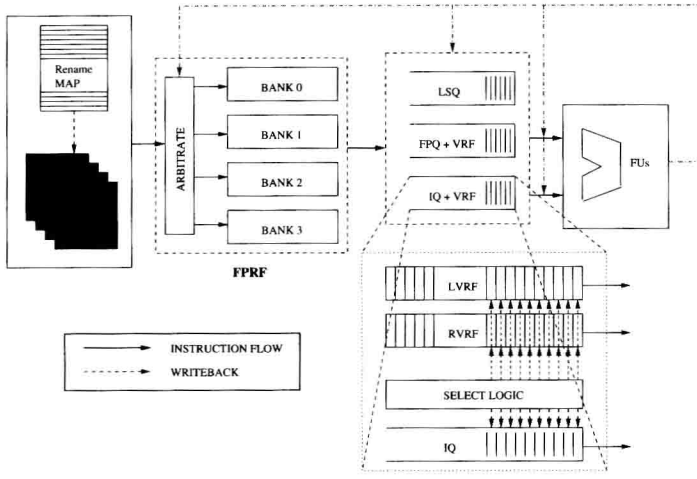

Fig. 1. The Front-End Physical Register File Architecture

allows for fast recovery with little complexity. It also allows to apply banking with high efficiency.

The FPRF Architecture, like a Future File, reads available registers in the front end. However, in this approach the registers are accessed via a mapping into a centralized register file that contains all registers. This has two implications:

1. Access to computed values in the front-end needs to be delayed until the rename stage has completed.
2. The number of registers in the front-end, being equal to the total number of registers, is much larger than it is in a Future File Architecture.

Figure 1 shows the FPRF architecture. Instructions, after going through the decode stage, enter the rename stage where source and destination registers are mapped to physical registers. Using this information an instruction may access the FPRF, a two stage process consisting of arbitration and data access. After available values have been given to the instruction, it is inserted into the corresponding instruction queue.

The back-end pipeline works as follows: When a functional units generates a result, the register tag is sent to all instructions in the queue. If there is an instruction waiting for it, the value is written into the corresponding entry of the Value Register File (VRF), which is part of the payload RAM of the instruction queue. The VRF is driven by the wakeup logic signals and can be implemented as a register file that does not require a decode stage. The value also gets written into the FPRF, as indicated by its physical register designator. There is also a possibility that a value is bypassed to a dependent instruction.

3.1 FPRF Pipeline

The pipeline of the FPRF Microarchitecture is shown in Fig. 2. It adds one stage to a commonly used 8-stage pipeline consisting of: fetch, decode, rename, queue, issue,

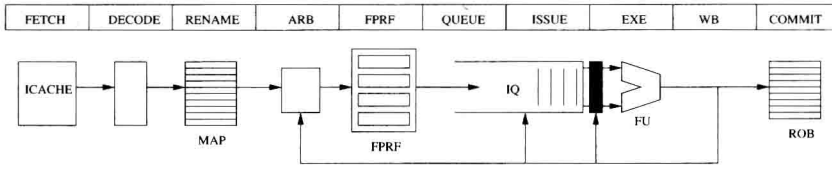

Fig. 2. The Pipeline of the FPRF Architecture

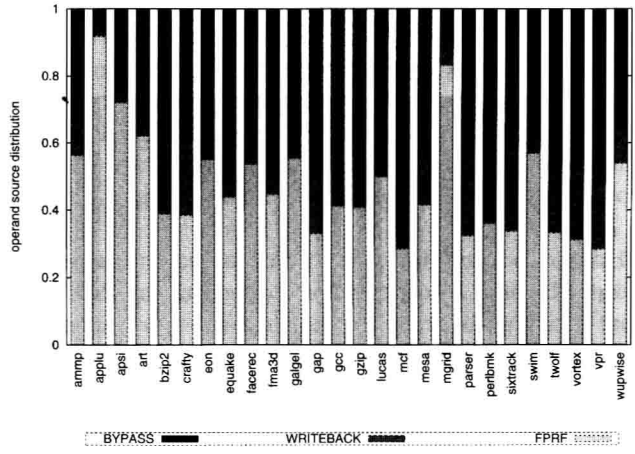

Fig. 3. Source of Integer Operands

operand read, execute, writeback and commit. To support FPRF access in the front-end, two stages are added to the pipeline: arbitration and FPRF Read. In the back-end the operand read stage disappears. This reduces the length to a single additional stage.

In the first new stage the source registers are analyzed to check for bank conflicts in the FPRF access. Conflicts stall all prior stages.

During rename, it is checked if the source registers have computed values. This is implemented via a bit vector with as many entries as logical registers. In the case of the Alpha ISA, modeled here, this requires maintaining two 32-bit vectors, one for the integer and another for the floating point registers. Each entry of this bit-vector indicates whether the corresponding logical register has a computed value. In case the computed value is available a read to the corresponding FPRF register is started.

During the arbitration cycle, priority is given to "older" instructions to access the operands. This makes sure that the front-end does not dispatch instructions to the instruction queues out-of-order.

The number of read ports for each bank has to be at least two, because some instructions must obtain both operands from the same FPRF bank.

Once arbitration has been performed the FPRF read occurs. After the instruction has read the available values it is inserted in the instruction queues. This happens during the *Queue* stage. At the same time, the register values are inserted into the VRF.

It is clear that the access rate to the FPRF is lower than to the centralized back-end physical register file. Lower access rate means that less conflicts will occur in the front-end and also that it will consume less energy. It was observed that the number of integer operands that are obtained from the FPRF is about 40% of the total while for floating point operands this number decreases to around 20%. Figures 3 and 4 show the distribution of integer and fp operand sources averaged over 100 million instructions for each Spec2000 benchmark. The boxes labeled *FPRF* account for operands that were read from the FPRF. The boxes labeled *WRITEBACK* are for operands that were written directly into the VRF from the writeback stage. Finally, the label *BYPASS* refers to those values that are sourced from the bypass network.

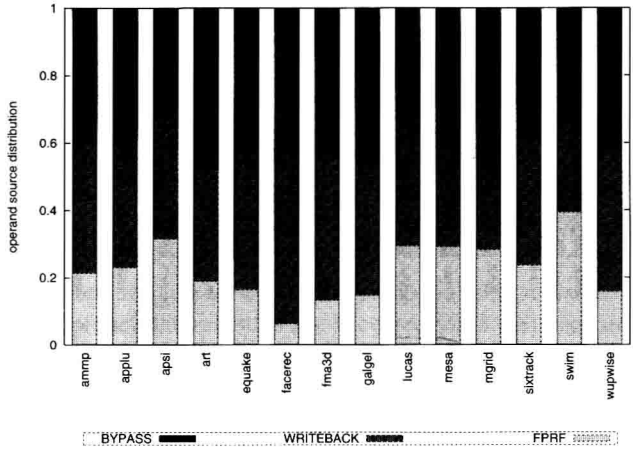

Fig. 4. Source of FP Operands

In the event of a branch misprediction the FPRF architecture behaves exactly like the MIPS R10000. First, the processor immediately aborts all instructions fetched along the mispredicted path. Next it restores the register mapping from the branch stack and finally, it starts fetching instructions from the correct path.

3.2 Read Sharing

Accesses to the same logical registers are often clustered during program execution. For example, many instructions use the same logical register for both register sources (eg `ADD R2, R2, Rd`). On the other hand it is also fairly common that the same register is sourced by several instructions without being written to. Such register accesses have a high probability of bank conflict. This suggests that conflicts in the access to the FPRF can be effectively reduced by using the technique known as read sharing [10]. Read sharing allows multiple reads of a same register to happen using a single local port which is connected to several global ports. Previous work on banked register files by Tseng et al. [21] has also used this approach. Read sharing will be evaluated later in the context of the FPRF architecture.