

Lecture Notes in Computer Science

942

Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Advisory Board: W. Brauer D. Gries J. Stoer

Günter Böckle

Exploitation of Fine-Grain Parallelism



Springer

Series Editors

Gerhard Goos

Universität Karlsruhe

Vincenz-Priessnitz-Straße 3, D-76128 Karlsruhe, Germany

Juris Hartmanis

Department of Computer Science, Cornell University

4130 Upson Hall, Ithaca, NY 14853, USA

Jan van Leeuwen

Department of Computer Science, Utrecht University

Padualaan 14, 3584 CH Utrecht, The Netherlands

Author

Günter Böckle

Zentralabteilung Forschung und Entwicklung, SiemensAG

Otto-Hahn-Ring 6, D-81739 München, German

E-mail: Guenter.Boeckle@zfe.siemens.de

Library of Congress Cataloging-in-Publication Data

Böckle, Günther, 1949-

Exploitation of fine-grain parallelism / Günther Böckle.

p. cm. -- (Lecture notes in computer science ; 942)

Includes bibliographical references (p.).

ISBN 3-540-60054-X (Berlin : acid-free paper). -- ISBN

0-387-60054-X (New York : acid-free paper)

1. Parallel processing (Electronic computers) I. Title.

II. Series.

QA76.58.B63 1995

004'.35--dc20

95-30348

CIP

CR Subject Classification (1991): D.3.4, D.3.2, B.1.4, B.6.1, B.2.1, C.1.2, D.1.3, F.1.2

ISBN 3-540-60054-X Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

© Springer-Verlag Berlin Heidelberg 1995

Printed in Germany

Typesetting: Camera-ready by author

SPIN: 10486363 06/3142-543210 - Printed on acid-free paper

Preface

The driving force for many research efforts is to provide higher performance for applications of computer systems. There are many computer systems architectures promising a high performance potential, chiefly among them parallel architectures. A whole zoo of parallel system architectures has been developed so far and many of those, although of intelligent design did not survive in the market. One of the main reasons for such failures is that the performance potential of such an architecture cannot be used efficiently by a sufficient number of applications.

This book presents methods to solve the problem of bringing the performance potential of parallel architectures to the applications so that the applications can use this potential efficiently. The focus is on parallelism on the instruction-set level which provides a high performance potential for a great variety of applications, not just the quite restricted sets of applications we see for many coarse-grain parallel architectures.

These methods have been developed in course of a project at Siemens Corporate Research and Development Department. A cooperation with the Institute for Computer Science at Munich Technical University was installed to join the forces for the development of corresponding methods and tools. This cooperation was then entered into a national research program at the Technical University, the Sonderforschungsbereich 342 "Methods and Tools for the Usage of Parallel Architectures".

Acknowledgements

Many people helped to do all the work presented in this book, however the actual project team was always quite small. We had help from our partners at the Institute for Computer Science at Munich Technical University, several Diplomarbeiten (master thesis) were made in this project.

I would like to deeply thank all those who contributed to the joined effort, above all the team members who did all an excellent job: Klaus Hahne, Jörg Schepers, Christof Störmann, Karl-Josef Thürlings, Siegfried Trosch, and Isolde Wildgruber.

Special thanks to Wolfgang Karl, our partner at Munich Technical University and to all the others, mainly: Uli Baumann, Ralf Karl, Michael Hardieck, and Martin Liebl.

Table of Contents

1	Introduction	1
2	Kinds of Parallelism	3
2.1	Classification by Structural Items	4
2.2	Classification by Items Processed	5
2.3	Classification by Instruction and Data Stream Parallelism	6
2.4	Classification by Memory Organization	7
3	Architectures for Fine-Grain Parallelism	8
3.1	Vertical Parallelism	8
3.2	Horizontal Parallelism	9
3.2.1	VLIW Architecture	9
3.2.2	Superscalar Processor Architectures	11
3.3	Classification of Pipelined Processor Architectures	12
3.4	Other Architectures for Fine-Grain Parallelism	15
3.4.1	Conditional Execution	16
3.4.2	History	17
4	VLIW Machines	18
4.1	Multiflow TRACE	18
4.2	IBM's VLIW Processor	19
4.3	Cydra	22
4.4	LIFE (Philips/Signetics)	24
4.5	The XIMD Architecture	26
5	Constraints on VLIW Architectures	28
6	Architectural Support for Exploitation of Fine-Grain Parallelism	32
6.1	Dynamic Instruction-Scheduling	32
7	Constraints for Instruction Scheduling	38
7.1	Data Dependences	38
7.2	Off-live Dependence	38
8	Instruction-Scheduling Methods	40
8.1	Local Instruction-Scheduling	42

8.2	Global Instruction-Scheduling	42
8.3	Global Scheduling Methods Based on Local Scheduling	43
8.3.1	Trace Scheduling	43
8.3.2	Other Global Scheduling Methods	46
8.4	Global Scheduling on the Program Graph	47
8.4.1	Percolation Scheduling	49
8.4.1.1	Move_op.....	50
8.4.1.2	Move_cj.....	51
8.4.1.3	Unify.....	52
8.4.1.4	Delete.....	53
8.4.2	Extensions to Percolation Scheduling	54
8.5	Loop Parallelization	55
8.5.1	Loop Unrolling	56
8.5.2	Software Pipelining	58
8.5.3	Perfect Pipelining	59
8.5.4	Integrating Loop Parallelization and Instruction Scheduling	61
9	Developing Instruction-Scheduling Methods	62
10	Tools for Instruction Scheduling	63
10.1	The C Compiler	64
11	The Machine Model	65
12	The Horizontal Instruction-Scheduler	67
12.1	The Interface Compiler - Scheduler	67
12.2	The Scheduler's Front-end	68
12.2.1	Mapping rtls to Machine Instructions	70
12.2.2	Flow Graph, Pointer Targets and Call Graph	74
12.2.3	Registers and Register Renaming	75
12.2.4	Memory-Access Analysis	81
12.2.5	Data-Dependence Analysis	90
12.2.6	Inserting Representatives for Multicycle Operations	92
12.2.7	Data-Flow Analysis	96
12.2.8	Standard Optimizations	101
12.3	The Scheduler's Central Part	103
12.3.1	Loop Parallelization	104
12.3.2	Partitioning the Program Graph into Regions	104
12.3.3	Extended Percolation Scheduling - Core Transformations	106

12.3.3.1	Standard Core Transformations.....	106
12.3.3.2	Core Transformations for Multicycle Operations	110
12.3.4	Extended Percolation Scheduling - Structure	115
12.3.5	Extended Percolation Scheduling - Control Tactics	117
12.3.5.1	Node-Oriented Tactics	117
12.3.5.2	Operation-Oriented Tactics	120
12.3.5.3	Which Tactics for what Application?	121
12.3.5.4	Control Tactics for Specific Code Patterns	121
12.3.5.5	Control Tactics Considering Resources	123
12.3.6	Extended Percolation Scheduling - Control Strategies	124
12.3.7	Moving Operations out of Regions and Functions	125
12.3.8	Resource Management	128
12.4	The Scheduler's Back-End	129
12.4.1	Register Allocation	129
12.4.2	Interactions Between Register Allocation and Instruction Scheduling	134
12.4.3	Rescheduling	135
12.4.4	Assignment of Operations to PEs	137
12.4.5	Instruction-Word Generation	138
12.4.6	Debug Information	140
13	Resource Management	142
13.1	Machine Description	144
13.2	Operators and Functions for Resource Management	146
13.3	Resource Allocation	147
13.4	Resource Assignment	148
13.4.1	Assignment of Processing Elements	148
13.4.2	Assignment of General Resources	152
14	Exceptions	156
15	Vertical Instruction-Scheduling	158
15.1	Vertical Core Transformations	159
16	Conclusion	164
17	References	165