

Distinguished Dissertations

Springer

London

Berlin

Heidelberg

New York

Barcelona

Hong Kong

Milan

Paris

Singapore

Tokyo

Other titles published in this Series:

Extensional Constructs in Intensional Type Theory

Martin Hoffman

Search and Planning Under Incomplete Information: A Study Using Bridge Card Play

Ian Frank

Theorem Proving with the Real Numbers

John Harrison

Games and Full Abstraction for a Functional Metalanguage with Recursive Types

Guy McCusker

Hardware Evolution: Automatic Design of Electronic Circuits in Reconfigurable Hardware by Artificial Evolution

Adrian Thompson

Models of Sharing Graphs: A Categorical Semantics of let and letrec

Masahito Hasegawa

Large Scale Collaborative Virtual Environments

Chris Greenhalgh

Radu C. Calinescu

Architecture-Independent Loop Parallelisation



Springer

Radu C. Calinescu, DPhil, MSc
Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford,
OX1 3QD, UK

Series Editor

Professor C.J. van Rijsbergen
Department of Computing Science, University of Glasgow, G12 8RZ, UK

ISSN 1439-9768

ISBN-13: 978-1-4471-1197-9 Springer-Verlag London Berlin Heidelberg

British Library Cataloguing in Publication Data
Calinescu, Radu C.

Architecture-independent loop parallelisation.-
(Distinguished dissertations)

1.Parallel processing (Electronic computers)

I.Title

004.3'5

ISBN-13: 978-1-4471-1197-9

Library of Congress Cataloging-in-Publication Data

Calinescu, Radu., 1968-

Architecture-independent loop parallelisation / Radu C. Calinescu.

p. cm. -- (Distinguished dissertations)

Includes bibliographical references and index.

ISBN-13: 978-1-4471-1197-9

e-ISBN-13: 978-1-4471-0763-7

DOI: 10.1007/978-1-4471-0763-7

1. Parallel processing (Computer science) 2. Computer architecture. I. Title. II.

Distinguished dissertations (Springer-Verlag)

QA76.58.C34 2000

004'.35--dc21

00-037369

Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act 1988, this publication may only be reproduced, stored or transmitted, in any form or by any means, with the prior permission in writing of the publishers, or in the case of reprographic reproduction in accordance with the terms of licences issued by the Copyright Licensing Agency. Enquiries concerning reproduction outside those terms should be sent to the publishers.

© Springer-Verlag London Limited 2000

Softcover reprint of the hardcover 1st edition 2000

The use of registered names, trademarks etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant laws and regulations and therefore free for general use.

The publisher makes no representation, express or implied, with regard to the accuracy of the information contained in this book and cannot accept any legal responsibility or liability for any errors or omissions that may be made.

Typesetting: Camera-ready by author

34/3830-543210 Printed on acid-free paper SPIN 10760296

Preface

This book addresses the automatic parallelisation of regular loop computations involving dense data structures. In order to achieve parallel code which is architecture-independent, scalable and of analytically predictable performance, scheduling in the bulk-synchronous parallel model of computation is considered.

Our parallelisation approach combines two types of scheduling in a novel way. A class of parallelisation techniques termed template-matching scheduling is used to build the parallel version of certain loop computations starting from predefined, highly optimised schedule skeletons. A more complicated technique called generic loop nest scheduling tackles the parallelisation of nested loops whose structure matches none of the recognised computation templates.

A collection of template-matching parallelisation methods is developed in the book. This collection builds on recent advances in automatic parallelisation and architecture-independent parallel programming, and includes two categories of scheduling techniques. The subset of techniques belonging to the first category is dedicated to the parallelisation of uniform-dependence perfect loop nests. The second category of techniques addresses the parallelisation of several loop constructs that appear frequently in imperative programs and comprise non-uniform dependences.

We also introduce a new scheme for the parallelisation of generic, untightly nested loops. This scheme comprises four steps: data dependence analysis, potential parallelism identification, data and computation partitioning, and communication and synchronisation generation. Due to the new algorithms employed in its last three steps, the scheme is able to identify coarse-grained potential parallelism, and to map it efficiently on the processor/memory units of a general purpose parallel computer.

The effectiveness of architecture-independent loop parallelisation is assessed through a series of case studies addressing the parallelisation of several scientific computing problems. For each problem, the best known parallel solution is compared with the one obtained using the automatic scheduling techniques, as well as with the parallel schedule generated by a research tool that implements a subset of these techniques. This study reveals that the new parallelisation approach is feasible, and can be successfully applied to many scientific computations involving dense data structures.

Except for a few minor corrections, this book represents the author's University of Oxford D.Phil. thesis.

Acknowledgements

I would like to express my gratitude to Bill McColl, my supervisor; he has provided invaluable advice and encouragement throughout the development of this thesis. I am also indebted to Alex Gerbessiotis, Constantinos Siniolakis, Stephen Donaldson, Alexandre Tiskin, Fabrizio Petrini, Jonathan Hill, Ronald Sujithan and all other current and former members of the Oxford BSP group for their comments on my work.

Special thanks are due to David J. Evans, Mike Giles, and Dan Stefanescu for their suggestions at various moments during my D.Phil. course, and to Stephen Donaldson and Tom Costello for proof-reading parts of the thesis.

Gaétan Hains and Fabrizio Petrini kindly accepted to examine this work, their valuable comments having led to several improvements in the final version of the thesis. I am also grateful to my examiners for suggesting that my thesis should be submitted to the Distinguished Dissertation competition.

To a great extent, this thesis represents the result of over twenty years of formal education in my life. I owe my success over all these years to many remarkable teachers. I am especially indebted to Octav Pastravanu for introducing me to the challenging world of research.

I am deeply grateful to my parents, Rodica and Dumitru, and to my wife, Ani, for their love, support and understanding; and to little Maria for reminding me that there is much more to life than computer science.

This work was funded by an Overseas Research Studentship, a Dulverton Scholarship, and an Oxford Overseas Bursary.

Contents

Glossary of Notations	xiii
------------------------------	-------------

List of Figures	xvii
------------------------	-------------

1 Introduction	1
1.1 Motivation	1
1.2 Parallelisation Approach Proposed in the Book	2
1.3 Organisation of the Book	3
2 The Bulk-Synchronous Parallel Model	5
2.1 Introduction	5
2.2 Bulk-Synchronous Parallel Computers	5
2.3 The BSP Programming Model	6
2.4 The BSP Cost Model	7
2.5 Assessing the Efficiency of BSP Code	8
2.6 The Development of BSP Applications	9
2.7 BSP Pseudocode	10
3 Data Dependence Analysis and Code Transformation	13
3.1 Introduction	13
3.2 Data Dependence	13
3.2.1 Definition	14
3.2.2 Data Dependence Representation	15
3.2.3 Dependence Tests	16
3.2.4 Dependence Graphs	16
3.2.5 Directed Acyclic Graphs	16
3.3 Code Transformation Techniques	17
3.3.1 Generalities	17
3.3.2 Loop Parallelisation	17
3.3.3 Loop Interchange and Loop Permutation	18
3.3.4 Loop Distribution	19
3.3.5 Loop Skewing, Wavefront Scheduling, and Iteration Space Tiling	20
3.3.6 Other Transformations for High-Performance Computing	22

4	Communication Overheads in Loop Nest Scheduling	23
4.1	Introduction	23
4.2	Related Work	25
4.3	Communication Overheads Due to Input Data	26
4.3.1	The Footprint Size of a Pure-Input Array	27
4.3.2	Input Communication Overheads Due to Input/Output Arrays	36
4.4	Inter-Tile Communication Overheads	38
4.5	Summary	42
5	Template-Matching Parallelisation	43
5.1	Introduction	43
5.2	Related Work	43
5.3	Communication-Free Scheduling	44
5.3.1	Scheduling Loop Nests Comprising Fully Parallel Loops . . .	45
5.3.2	Scheduling Loop Nests with no Fully Parallel Loop	47
5.3.3	Improving the Load Balancing of Communication-Free Scheduling	50
5.4	Wavefront Block Scheduling	52
5.4.1	Scheduling Fully Permutable Loop Nests	53
5.4.2	Extension to Generic Uniform-Dependence Loop Nests	58
5.4.3	Improving the Load Balancing of Wavefront Block Scheduling	60
5.5	Iterative Scheduling	61
5.5.1	Description of the Technique	61
5.5.2	Extension to Generic Loops and Load Balancing	64
5.5.3	Comparison with Wavefront Block Scheduling	65
5.6	Reduction Scheduling	66
5.7	Recurrence Scheduling	68
5.8	Scheduling Broadcast Loop Nests	70
5.8.1	Definition of a Broadcast Loop Nest	70
5.8.2	Scheduling Through Broadcast Implementation	73
5.8.3	Scheduling Through Broadcast Elimination	78
5.8.4	Comparison of the Two Approaches	81
5.9	Summary	82
6	Generic Loop Nest Parallelisation	85
6.1	Introduction	85
6.2	Related Work	86
6.3	Data Dependence Analysis	88
6.4	Potential Parallelism Identification	89
6.5	Data and Computation Partitioning	95
6.6	Communication and Synchronisation Generation	101
6.7	Performance Analysis	105
6.8	Summary	107

7	A Strategy and a Tool for Architecture-Independent Loop Parallelisation	109
7.1	Introduction	109
7.2	Related Work	109
7.3	A Two-Phase Strategy for Loop Nest Parallelisation	111
7.4	BSPscheduler: an Architecture-Independent Loop Paralleliser	112
7.4.1	The Structure of the Parallelisation Tool	112
7.4.2	The User Interface	113
7.4.3	The Parser Module	114
7.4.4	The Dependence Analysis Module	116
7.4.5	The Scheduling Modules	117
7.4.6	The Code Generation Module	120
7.5	Summary	123
8	The Effectiveness of Architecture-Independent Loop Parallelisation	125
8.1	Introduction	125
8.2	Matrix-Vector and Matrix-Matrix Multiplication	125
8.3	LU Decomposition	127
8.4	Algebraic Path Problem	129
8.5	Finite Difference Iteration on a Cartesian Grid	132
8.6	Merging	134
8.7	Summary	134
9	Conclusions	139
9.1	Summary of Contributions and Concluding Remarks	139
9.2	Future work directions	142
	Appendix A. Theorem proofs	145
	Appendix B. Syntax of the BSPscheduler input language	151
	Appendix C. Syntax of the BSPscheduler output language	155
	Appendix D. Automatically generated code for Example 7.5	157
	Bibliography	161
	Index	171

Glossary of Notations

Arithmetic

$ x $	the absolute value of x
$\lceil x \rceil$	the ceiling of x ($y \in \mathbf{Z}$ such that $y - 1 < x \leq y$)
$\lfloor x \rfloor$	the floor of x ($y \in \mathbf{Z}$ such that $y \leq x < y + 1$)
$\gcd(x_1, x_2, \dots, x_n)$	the greatest common divisor of x_1, x_2, \dots, x_n
$\min\{x_1, x_2, \dots, x_n\}$	the minimum of x_1, x_2, \dots, x_n
$\max\{x_1, x_2, \dots, x_n\}$	the maximum of x_1, x_2, \dots, x_n
$x \bmod y$	the remainder of the integer division of x by y

Sets

$\{\}$	the empty set
$\{x_1, x_2, \dots, x_n\}$	the set containing elements x_1, x_2, \dots, x_n
$\#X$	the number of elements in set X
A, B, C, \dots	sets
\mathbf{N}	the set of natural numbers $\{0, 1, 2, \dots\}$
\mathbf{Z}	the set of integer numbers $\{\dots, -2, -1, 0, 1, 2, \dots\}$
$x \in X$	set membership
$\{x \in T \mid P(x)\}$	set comprehension (the set of all x in T such that $P(x)$ holds)
$\mathbf{P}A$	the powerset of A ($\{X \mid X \subseteq A\}$)
$x..y$	the set $\{k \in \mathbf{Z} \mid x \leq k \leq y\}$, where $x, y \in \mathbf{Z}$
$X \subseteq Y$	set inclusion ($\forall x \in X \bullet x \in Y$)
$X \setminus Y$	set difference ($\{x \in X \mid x \notin Y\}$)
$X \cup Y$	set union ($\{x \mid x \in X \vee x \in Y\}$)
$X \cap Y$	set intersection ($\{x \in X \mid x \in Y\}$)
$X \times Y$	the Cartesian product of sets X and Y ($\{(x, y) \mid x \in X \wedge y \in Y\}$)

Logic

$\neg x$	negation (not x)
$x \vee y$	disjunction (x or y)
$x \wedge y$	conjunction (x and y)

$x \Rightarrow y$	implication (if x , then y)
$\forall x \in X \bullet \text{pred}$	universal quantification (<i>pred</i> holds for all $x \in X$)
$\exists x \in X \bullet \text{pred}$	existential quantification (<i>pred</i> holds for at least one $x \in X$)

Linear Algebra

A, B, C, \dots	matrices
$A = [a_1, a_2, \dots, a_n]$	matrix A has columns a_1, a_2, \dots, a_n
$A = [a_{i,j}]$	matrix A has elements $a_{i,j}$
A^T	the transpose of matrix A
$\det A$	the determinant of matrix A
$\text{rank } A$	the rank of matrix A
I_n	the $n \times n$ identity matrix
u, v, x, \dots	vectors
$x = [x_1, x_2, \dots, x_n]^T$	the n -dimensional vector x has elements x_1, x_2, \dots, x_n
(x_1, x_2, \dots, x_n)	the point of coordinates x_1, x_2, \dots, x_n in an n -dimensional space
$\text{diag}(x_1, x_2, \dots, x_n)$	the matrix $I_n[x_1, x_2, \dots, x_n]^T$
$\text{span}\{v^1, v^2, \dots, v^n\}$	the vector space spanned by the vectors v^1, v^2, \dots, v^n
N^x	the vector space of x -dimensional vectors with natural elements

Asymptotic Notation

$O(f(n))$	$\{g(n) \mid \exists(c > 0, n_0 > 0) \bullet \forall n \geq n_0 \bullet g(n) \leq cf(n)\}$
$o(f(n))$	$\{g(n) \mid \forall c > 0 \bullet \exists n_0 > 0 \bullet \forall n \geq n_0 \bullet g(n) \leq cf(n)\}$
$\Omega(f(n))$	$\{g(n) \mid \exists(c > 0, n_0 > 0) \bullet \forall n \geq n_0 \bullet cf(n) \leq g(n)\}$
$\omega(f(n))$	$\{g(n) \mid \forall c > 0 \bullet \exists n_0 > 0 \bullet \forall n \geq n_0 \bullet cf(n) \leq g(n)\}$
$\theta(f(n))$	$\{g(n) \mid \exists(c_1 > 0, c_2 > 0, n_0 > 0) \bullet \forall n \geq n_0 \bullet c_1 f(n) \leq g(n) \leq c_2 f(n)\}$

Automatic Parallelisation and the BSP Model

a, b, c, \dots	arrays
d, d^1, d^2, d^3, \dots	distance vectors
$\text{footprint}(a)$	the footprint of array a
g	the BSP communication parameter
\mathcal{G}	dependence graph
i, i_1, i_2, i_3, \dots	loop indices
\mathbf{i}	the index vector of a perfect loop nest ($\mathbf{i} = [i_1, i_2, \dots, i_K]^T$) or an iteration point of a perfect loop nest ($\mathbf{i} = (i_1, i_2, \dots, i_K)$)
I	the iteration space of a perfect loop nest
K	the number of loops in a perfect loop nest
l, l_1, l_2, l_3, \dots	loops in a computer program
L	the BSP synchronisation parameter
\mathcal{L}	loop nest

p	the number of processor/memory units of a BSP computer
S, S_1, S_2, S_3, \dots	statements in a computer program
$S_1 \delta S_2$	flow data dependence between statements S_1 and S_2
$S_1 \bar{\delta} S_2$	data antidependence between statements S_1 and S_2
$S_1 \delta^o S_2$	output data dependence between statements S_1 and S_2
$S_1 \delta^* S_2$	generic data dependence between statements S_1 and S_2
$\mathbf{v}, \mathbf{v}^1, \mathbf{v}^2, \mathbf{v}^3, \dots$	direction vectors

List of Figures

2.1	A bulk-synchronous parallel computation	7
3.1	Types of data dependence	14
3.2	The dependence graphs of the loop nests in Figure 3.1	16
3.3	Loop parallelisation	18
3.4	Loop interchange	19
3.5	Loop distribution	20
3.6	Loop skewing	21
4.1	A K -level perfect loop nest	24
4.2	The algorithm for the computation of the footprint size of a single array reference	31
5.1	The positive basis computation algorithm	48
5.2	A triangular loop nest and the block-cyclic partitioning of its iteration space	52
5.3	The wavefront block schedule of a fully permutable loop nest	53
5.4	The wavefront block scheduling of the loop nest in Example 5.5	57
5.5	The iterative schedule of a K -level uniform-dependence loop nest	62
5.6	Parallel schedule for Example 5.6	64
5.7	The tiles computed by a non-boundary processor in two successive supersteps of the iterative schedule in Example 5.6	64
5.8	A generic reduction loop nest	66
5.9	The generic form of a k -th-order recurrence loop	69
5.10	A K -level broadcast loop nest, $K \geq 2$	71
5.11	The j -th broadcast initialisation loop in Figure 5.10, $1 \leq j \leq J$	71
5.12	Triangular linear system solution by forward substitution	72
5.13	Gaussian elimination	73
5.14	Broadcast loop nest scheduling through broadcast implementation	74
5.15	Parallel solution of a triangular linear system	78
5.16	Augmented broadcast initialisation loop	79
5.17	A perfect loop nest equivalent to the broadcast loop nest in Figures 5.10-5.11	80

5.18	The ratio between the communication costs associated with the parallelisation of a three-level broadcast loop nest through broadcast implementation and broadcast elimination, respectively	82
6.1	The scheme for the architecture-independent scheduling of generic loop nests	85
6.2	A generic, untightly nested loop, and its dependence graph	89
6.3	The potential parallelism identification algorithm	93
6.4	The potential parallelism of the loop nest in Figure 6.2	95
6.5	The data partitioning algorithm	96
6.6	The computation partitioning algorithm	100
6.7	The partitioned parallel version of the loop nest in Figure 6.2	101
6.8	The synchronisation and communication generation algorithm	103
6.9	The BSP schedule of the generic, untightly nested loop in Figure 6.2	105
6.10	The performance analysis algorithm	106
7.1	The structure of the architecture-independent scheduling tool	112
7.2	The BSPSCHEDULER user interface	113
7.3	A nested loop, and its normalised version generated by the parser	115
7.4	The usage of the parser module	116
7.5	The dependence analysis step	117
7.6	The actual parallelisation step	120
7.7	The intermediate results of the parallelisation	121
7.8	The adjustment of array subscripts in the code generation step	122
8.1	Sequential LU decomposition	128
8.2	Parallel LU decomposition	129
8.3	The sequential solution of the algebraic path problem	130
8.4	The BSP version of the APP computation loop	130
8.5	Sequential finite difference iteration	132
8.6	The BSP version of Gauss-Seidel finite difference iteration	133
8.7	The sequential merging of two sequences sorted in increasing order	134