

Symbolic Parallelization of Nested Loop Programs

Alexandru-Petru Tanase • Frank Hannig
Jürgen Teich

Symbolic Parallelization of Nested Loop Programs



Springer

Alexandru-Petru Tanase
Friedrich-Alexander-Universität
Erlangen-Nürnberg (FAU)
Erlangen, Germany

Frank Hannig
Friedrich-Alexander-Universität
Erlangen-Nürnberg (FAU)
Erlangen, Germany

Jürgen Teich
Friedrich-Alexander-Universität
Erlangen-Nürnberg (FAU)
Erlangen, Germany

ISBN 978-3-319-73908-3 ISBN 978-3-319-73909-0 (eBook)
<https://doi.org/10.1007/978-3-319-73909-0>

Library of Congress Control Number: 2018930020

© Springer International Publishing AG 2018

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

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

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by the registered company Springer International Publishing AG part of Springer Nature.

The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Contents

1	Introduction	1
1.1	Goals and Contributions	4
1.2	Symbolic Outer and Inner Loop Parallelization	4
1.3	Symbolic Multi-level Parallelization	5
1.4	On-demand Fault-tolerant Loop Processing	5
1.5	Book Organization	6
2	Fundamentals and Compiler Framework	9
2.1	Invasive Computing	9
2.2	Invasive Tightly Coupled Processor Arrays	13
2.2.1	Processor Array	14
2.2.2	Array Interconnect	15
2.2.3	TCPA Peripherals	16
2.3	Compiler Framework	18
2.3.1	Compilation Flow	18
2.3.2	Front End	20
2.3.3	Loop Specification in the Polyhedron Model	22
2.3.4	PAULA Language	26
2.3.5	PARO	26
2.3.6	Space-Time Mapping	33
2.3.7	Code Generation	34
2.3.8	PE Code Generation	35
2.3.9	Interconnect Network Configuration	35
2.3.10	GC and AG Configuration Stream	36
3	Symbolic Parallelization	37
3.1	Symbolic Tiling	38
3.1.1	Decomposition of the Iteration Space	39
3.1.2	Embedding of Data Dependencies	41
3.2	Symbolic Outer Loop Parallelization	46
3.2.1	Tight Intra-Tile Schedule Vector Candidates	48
3.2.2	Tight Inter-tile Schedule Vectors	54

3.2.3	Parametric Latency Formula	60
3.2.4	Runtime Schedule Selection	63
3.3	Symbolic Inner Loop Parallelization	65
3.3.1	Tight Intra-Tile Schedule Vectors	67
3.3.2	Tight Inter-tile Schedule Vector Candidates	68
3.3.3	Parametric Latency Formula	71
3.3.4	Runtime Schedule Selection	74
3.4	Runtime Schedule Selection on Invasive TCPAs	76
3.5	Experimental Results	77
3.5.1	Latency.....	78
3.5.2	I/O and Memory Demand	82
3.5.3	Scalability.....	84
3.6	Related Work	86
3.7	Summary	92
4	Symbolic Multi-Level Parallelization	93
4.1	Symbolic Hierarchical Tiling	94
4.1.1	Decomposition of the Iteration Space	95
4.1.2	Embedding of Data Dependencies.....	97
4.2	Symbolic Hierarchical Scheduling	100
4.2.1	Latency-Minimal Sequential Schedule Vectors	101
4.2.2	Tight Parallel Schedule Vectors.....	106
4.2.3	Parametric Latency Formula	108
4.2.4	Runtime Schedule Selection	110
4.3	Experimental Results	112
4.3.1	Latency.....	112
4.3.2	I/O and Memory Balancing	115
4.3.3	Scalability.....	115
4.4	Related Work	117
4.5	Summary	121
5	On-Demand Fault-Tolerant Loop Processing	123
5.1	Fundamentals and Fault Model.....	124
5.2	Fault-Tolerant Loop Execution	126
5.2.1	Loop Replication	127
5.2.2	Voting Insertion.....	130
5.2.3	Immediate, Early, and Late Voting	132
5.3	Voting Functions Implementation	140
5.4	Adaptive Fault Tolerance Through Invasive Computing	142
5.4.1	Reliability Analysis for Fault-Tolerant Loop Execution.....	145
5.5	Experimental Results	146
5.5.1	Latency Overhead	146
5.5.2	Average Error Detection Latency.....	149
5.6	Related Work	150
5.7	Summary	152

6 Conclusions and Outlook 155

 6.1 Conclusions 155

 6.2 Outlook 157

Bibliography 159

Index..... 171

Acronyms

ABS	Anti-lock Breaking System
AG	Address Generator
AST	Abstract Syntax Tree
CGRA	Coarse-Grained Reconfigurable Array
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DMR	Dual Modular Redundancy
DPLA	Dynamic Piecewise Linear Algorithm
ECC	Error-correcting Code
EDC	Egregious Data Corruptiony
EDL	Error Detection Latency
FCR	Fault Containment Region
FSM	Finite State Machine
FU	Functional Unit
GC	Global Controller
GPU	Graphics Processing Unit
HPC	High-Performance Computing
<i>i</i> Ctrl	Invasion Controller
<i>i</i> -let	Invasive-let
ILP	Integer Linear Program
IM	Invasion Manager
<i>i</i> -NoC	Invasive Network-on-Chip
LPGS	Locally Parallel Globally Sequential
LSGP	Locally Sequential Globally Parallel
MPSoC	Multi-Processor System-on-Chip
NMR	N-Modular Redundancy

PE	Processing Element
PFH	Probability of Failure per Hour
PGAS	Partitioned Global Address Space
PLA	Piecewise Linear Algorithm
SER	Soft Error Rate
SEU	Single-Event Upset
SIL	Safety Integrity Level
SoC	System-on-Chip
SPARC	Scalable Processor Architecture
TCPA	Tightly Coupled Processor Array
TMR	Triple Modular Redundancy
UDA	Uniform Dependence Algorithm
VLIW	Very Long Instruction Word

List of Symbols

D^* – Set of tiled dependency vectors	42
$E[L_{E,early}]$ – The average error detection latency for early voting	137
$E[L_{E,imm}]$ – The average error detection latency for immediate voting	135
$E[L_{E,late}]$ – The average error detection latency for late voting	139
G – The number of quantified equations	23
I – Original iteration vector	23
In – Input space	60
J – Intra-tile iteration vector	40
K – Inter-tile iteration vector	41
K_f – The tile to be executed first by a symbolic schedule vector λ	61
K_l – The tile to be executed last by a symbolic schedule vector λ	61
L – Latency	33
L_g – Global latency	33
L_l – Local latency	33
$L_{E,early}$ – Error detection latency for early voting	137
$L_{E,imm}$ – Error detection latency for immediate voting	134
$L_{E,late}$ – Error detection latency for late voting	139
L_{opt} – Optimal latency	63
M – Maximal number of symbolic schedule candidates	51
Out – Output space	60
P – Tiling matrix	30
R – Replicated iteration vector	127
S – Path stride matrix	48
Φ – The allocation matrix	33
$\mathcal{B}$ – Set of protected variables	132
λ – Schedule vector	32
λ_J – Intra-tile schedule vector	47
λ_K – Inter-tile schedule vector	47
λ_R – Schedule vector of replicated iteration space	128
$\mathcal{P}$ – Processor space	33

$\mathcal{P}_{red}$ – Replicated processor space	128
$\mathcal{R}$ – Set of replicated iteration space	127
$\mathcal{C}$ – Set of I/O constraints	111
$\mathcal{F}$ – An operation to be scheduled	32
$\mathcal{I}$ – Set of original iteration space	24
$\mathcal{J}$ – Set of intra-tile iteration space	40
$\mathcal{K}$ – Set of inter-tile iteration space	40
$\mathcal{L}$ – Set of feasible symbolic schedule vector candidates	63
$\mathcal{M}$ – Set of memory constraints	111
$\mathcal{V}_k$ – Voting space for a protected variable x_k	131
τ – Relative start offset of an operation $\mathcal{F}$	32
d – Dependency vector	42
d^* – Tiled dependency vector	43
d_J – Intra-tile dependency vector	42
d_K – Inter-tile dependency vector	42
p – Processor index	33
r_v – Replica where the voting takes place	131
t – Start time	33
w_i – Execution time of a operation i	33
R_S – Replication space	127