LOOP PIPELINING WITH RESOURCE AND TIMING CONSTRAINTS

Autor: Fermín Sánchez

October, 1995
ACKNOWLEDGMENTS

I would like to thank the members of the Department of Computer Architecture for their support throughout the development of this work. In particular, I would like to thank Jordi Cortadella for his guidance and support throughout my graduate career. His enthusiasm for my early steps in the field gave me the confidence to pursue my own ideas. Besides being my advisor, he has been my best collaborator all these years. I would also like to give special thanks to Rosa M. Badia for her suggestions and comments, which have contributed to the improvement of this work. My gratitude goes also to the rest of the CAD-VLSI group. They have also contributed by their constant encouragement for me to finish this work.

I would like to thank my colleagues in the DAC, especially Anna del Corral, Josep Llosa, Angel Toribio, Mildred Sarmiento, Enric Pastor, Agustín Fernández and Montse Peiron. They have made my years in the University much more pleasant. From among all of them, my deepest gratitude goes to Josep Llosa for the great quantity of discussions that we have maintained in recent years, which have doubtlessly contributed to the enrichment of this work.

I thank Marc Noi for his help with the Farey's series, and Tricia for being my English advisor all these years. I also thank Tomás Lang, David Padua and Mateo Valero for giving me part of their valuable time, listening to my ideas and giving me their suggestions. I am equally grateful to Q. Ning, R. Govindarajan, Eric R. Altman and Guang G. Gao for supplying me the data dependence graphs used for comparisons in superscalar and VLIW processors.

Finally, I am greatly indebted to Ivette, who has always been understanding about my work. I would like especially to thank my brother David and my parents Herminio and Francisca. Their love and support have given me the courage to finish this work. I am privileged to belong to such a wonderful family. This work is dedicated to them.
To my family, the best in the world
CONTENTS

LIST OF FIGURES xi
LIST OF TABLES xviii
LIST OF ALGORITHMS xix
PREFACE xxii

1 INTRODUCTION 1
  1.1 Motivation of this work 1
  1.2 High-level synthesis and parallel architectures 2
    1.2.1 High-level synthesis 2
    1.2.2 Superscalar processors 4
    1.2.3 VLIW processors 4
  1.3 Internal representation of loops 6
    1.3.1 Program dependences 6
    1.3.2 Data dependence graph 7
  1.4 Coarse-grained parallelization 9
  1.5 Fine-grained parallelization 11
  1.6 Representation of algorithms in this work 13
  1.7 Summary 13

2 SOFTWARE PIPELINING 15
  2.1 Introduction 15
  2.2 State of the art 16
    2.2.1 Notation and classification 16
    2.2.2 Approaches which do not calculate MII 18
    2.2.3 Approaches which estimate the MII 21
    2.2.4 Approaches which analytically calculate MII 24
    2.2.5 Linear programming approaches 33
    2.2.6 Comparisons among the approaches 34
  2.3 Techniques proposed in this work 35
  2.4 Summary 38
# 3 Basic Definitions and Loop Transformations

## 3.1 Introduction

## 3.2 Representation of a loop

## 3.3 Representation of the architecture

### 3.3.1 Representation of resources

### 3.3.2 Representation of instructions

### 3.3.3 Example of representation of instructions

### 3.3.4 Example of architecture

## 3.4 Bounds on loop execution

### 3.4.1 Resource-constrained MII

### 3.4.2 Recurrence-constrained MII

### 3.4.3 Minimum initiation interval and throughput

## 3.5 Dependence retiming

### 3.5.1 Dependence retiming transformation

## 3.6 Loop unrolling

### 3.6.1 Loop unrolling transformation

### 3.6.2 H-graphs with integer MII

## 3.7 Summary and conclusions

# 4 Analysis of Data Dependences

## 4.1 Introduction

## 4.2 Schedule of a Π-graph

## 4.3 Scheduling dependences

## 4.4 Positive depth and height

### 4.4.1 Positive path

### 4.4.2 Maximal positive path, positive depth and height

### 4.4.3 Example of computing positive depth

## 4.5 ASAP and ALAP time

## 4.6 Negative depth

### 4.6.1 Negative restrictive dependences

### 4.6.2 Assigning negative depth to nodes

### 4.6.3 Example

## 4.7 Summary and conclusions

# 5 Scheduling a Π-Graph

## 5.1 Introduction

## 5.2 Scheduling graph

## 5.3 Overlapped schedule

## 5.4 List scheduling overview

## 5.5 Scheduling priority functions

### 5.5.1 The 0-mobility of a node

### 5.5.2 The positive depth of a node

### 5.5.3 The negative depth of a node

## 5.7 Summary and conclusions
Contents

5.5.4 The number of successors (not yet scheduled) of a node in the scheduling graph 79
5.5.5 The use of resources performed by an instruction 79
5.5.6 Complexity of selecting a node for scheduling 80
5.6 Scheduling algorithm 81
5.7 Summary and conclusions 81

6 UNRET: LOOP PIPELINING WITH RESOURCE CONSTRAINTS 83
6.1 Introduction 83
6.2 Exploring the solution space 85
6.2.1 Throughput diagram 85
6.2.2 Farey's series 87
6.2.3 Exploring Farey's series in decreasing order of magnitude 88
6.2.4 Reducing the solution space 91
6.2.5 Figures of merit 93
6.3 Retiming dependences 94
6.3.1 Range for retiming dependences 94
6.3.2 Retiming dependences not belonging to recurrences 95
6.3.3 Retiming dependences belonging to recurrences 96
6.4 Finding a schedule with maximum throughput 98
6.4.1 Quality of a r-graph 98
6.4.2 Finding a schedule in II cycles 99
6.4.3 General algorithm 102
6.5 Examples 104
6.5.1 Example 1 104
6.5.2 Example 2 105
6.5.3 Example 3 108
6.6 Experimental results 109
6.6.1 High-level synthesis 109
6.6.2 Superscalar and VLIW processors 111
6.7 Summary and conclusions 112

7 RESIS: REGISTER OPTIMIZATION 115
7.1 Introduction 115
7.1.1 Strategy overview 116
7.2 Previous work 116
7.3 Lower bounds on register pressure and RESIS strategy 120
7.3.1 Variable lifetime 120
7.3.2 Registers required for a dependence 121
7.3.3 Register pressure 122
7.3.4 Lower bounds on registers 124
7.4 SPAN reduction 126
7.4.1 Introduction
7.4.2 Heuristics to select a node to reduce the SPAN
7.4.3 Reduce index transformation
7.4.4 Reducing the number of scheduling dependences
7.4.5 Reducing local maxima
7.4.6 Scheduling
7.4.7 SPAN Reduction. Final algorithm
7.5 Incremental scheduling
7.5.1 Overview
7.5.2 Selecting an instruction to move
7.5.3 Moving an instruction
7.5.4 Re-scheduling
7.5.5 Swapping
7.5.6 Computational complexity of incremental scheduling
7.6 Experimental Results
7.6.1 High-level synthesis
7.6.2 Superscalar and VLIW processors
7.7 Summary and conclusions

8 TCLP: LOOP PIPELINING WITH TIMING CONSTRAINTS
8.1 Introduction
8.1.1 Strategy overview
8.2 TCLP Approach
8.2.1 Minimum initiation interval
8.2.2 Absolute lower bound on the set of resources
8.2.3 Increasing the number of resources
8.2.4 Reducing the set of resources
8.2.5 Increasing throughput
8.2.6 Reducing register pressure
8.2.7 TCLP. Execution time
8.3 Example
8.4 Experimental Results
8.5 Summary and conclusions

9 CONCLUSIONS AND FUTURE WORK
9.1 Contributions
9.1.1 Software pipelining: retiming and scheduling are separated into independent tasks
9.1.2 Analysis of data dependences and scheduling
9.1.3 Exploration of the solution space
9.1.4 Register reduction
9.1.5 Time-constrained loop pipelining
# LIST OF FIGURES

## Chapter 1
- 1.1 High-level synthesis system
- 1.2 Execution of instructions in different processors
- 1.3 Execution of instructions in a VLIW processor
- 1.4 Source code for the Livermore Fortran Kernel 3
- 1.5 Inner product compiled into a pseudo-assembly language
- 1.6 DDG of the inner product
- 1.7 Model for doacross scheduling

## Chapter 2
- 2.1 Software pipelining a loop
- 2.2 Example of DG and schedules for 4 adders

## Chapter 3
- 3.1 Representation of a loop by means of a π-graph
- 3.2 Equivalent π-graphs
- 3.3 Description of an architecture
- 3.4 Description of Cydra 5 Computer
- 3.5 Execution of compiled inner product
- 3.6 Recurrence in a loop
- 3.7 Equivalent π-graphs and their schedules
- 3.8 Unrolling a π-graph
- 3.9 Scheduling a π-graph and a multiple-instanced π-graph

## Chapter 4
- 4.1 Types of dependences in a schedule
- 4.2 Types of scheduling dependences according to δ(u, v)
- 4.3 Time frame for scheduling of a PSD and an NSD
- 4.4 Length of a positive path
- 4.5 Positive Depth of the nodes in a π-graph
- 4.6 NSD that does not constrain the scheduling process
- 4.7 Negative recurrence in a π-graph
- 4.8 II-graph with negative recurrences chained
- 4.9 Compute of negative depth
List of Figures

7.11 Flow diagram of incremental scheduling. 134

Chapter 8

8.1 Flow Diagram of TCLP 145
8.2 Resource responsible for not finding the schedule 148
8.3 Time-frame for scheduling 149
8.4 Exploration of the throughput diagram 150
8.5 Throughput exploration for FDCT 152

Appendix A

A.1 Cytron's example and Differential Equation 164
A.2 Algorithmic description of the differential equation 164
A.3 16-Point Digital FIR Filter 165
A.4 Fifth-Order Elliptic Filter 166
A.5 Fast Discrete Cosine Transform Kernel 167
A.6 Some examples of DDGs 169

Appendix B

B.1 Schedule for the differential equation 173

Appendix C

C.1 Comparing loop schedules for Spec Spice 10 benchmark 195
LIST OF TABLES

Chapter 2
2.1 Comparison among different software pipelining approaches 36

Chapter 6
6.1 Cytron’s example 110
6.2 Differential Equation 110
6.3 Fast Discrete Cosine Transform 110
6.4 Comparison for an architecture with 3 FP adders, 2 FP multipliers, 1 FP divisor and 2 load/store units 111

Chapter 7
7.1 Register reduction in a modulo scheduling algorithm for the Cytron example 138
7.2 Register reduction in a modulo scheduling algorithm for the 16-Point Digital FIR Filter 138
7.3 Register reduction in a modulo scheduling algorithm for the Fast Discrete Cosine Transform 138
7.4 Incremental scheduling after modulo scheduling for the Cytron example 139
7.5 Incremental scheduling after modulo scheduling for the 16-Point Digital FIR Filter 139
7.6 Incremental scheduling after modulo scheduling for the Fast Discrete Cosine Transform 140
7.7 Register reduction in a modulo scheduling algorithm by assuming a VLIW processor with 3 FP adders, 2 FP multipliers, 1 FP divisor and 2 load/store units 141

Chapter 8
8.1 Cytron’s example 153
8.2 Differential Equation 154
8.3 Fifth-Order Elliptic Filter with Non-Pipelined Multipliers 154
8.4 Fifth-Order Elliptic Filter with Pipelined Multipliers 154
8.5 Fast Discrete Cosine Transform 154

Appendix A
A.1 Benchmark loops 168
Appendix B

B.1 Cytron's example
B.2 Differential Equation
B.3 16-Point Digital FIR Filter
B.4 Fifth-Order Elliptic Filter with Non-Pipelined Multipliers
B.5 Fifth-Order Elliptic Filter with Pipelined Multipliers
B.6 Fast Discrete Cosine Transform
B.7 Results obtained by other approaches for superscalar processors by using an architecture with 1 FU of each type
B.8 Results obtained by UNRET for superscalar processors by using an architecture with 1 FU of each type. \( \text{Max II} = 15 \) for all cases except for (*), in which \( \text{Max II} = 50 \).
B.9 Comparison for an architecture with 3 FP adders, 2 FP multipliers, 1 FP divisor and 2 load/store units

Appendix C

C.1 Lower bounds for the Cytron's example
C.2 Lower bounds for the Differential Equation
C.3 Lower bounds for the 16-Point Digital FIR Filter
C.4 Lower bounds for the Fifth-Order Elliptic Filter with Non-Pipelined Multipliers
C.5 Lower bounds for the Fifth-Order Elliptic Filter with Pipelined Multipliers
C.6 Lower bounds for the Fast Discrete Cosine Transform
C.7 Register requirements for the Cytron's example
C.8 Register requirements for the differential Equation
C.9 Register requirements for the 16-Point Digital FIR Filter
C.10 Register requirements for the Fifth-Order Elliptic Filter with Non-Pipelined Multipliers
C.11 Register requirements for the Fifth-Order Elliptic Filter with Pipelined Multipliers
C.12 Register requirements for the Fast Discrete Cosine Transform
C.13 Register reduction in a modulo scheduling algorithm for the Cytron example
C.14 Register reduction in a modulo scheduling algorithm for the differential equation
C.15 Register reduction in a modulo scheduling algorithm for the 16-Point Digital FIR Filter
C.16 Register reduction in a modulo scheduling algorithm for the Fifth-Order Elliptic Filter with Non-Pipelined Multipliers
C.17 Register reduction in a modulo scheduling algorithm for the Fifth-Order Elliptic Filter with Pipelined Multipliers
C.18 Register reduction in a modulo scheduling algorithm for the Fast Discrete Cosine Transform
C.19 Incremental scheduling after modulo scheduling for the Cytron example
C.20 Incremental scheduling after modulo scheduling for the differential equation
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.21</td>
<td>Incremental scheduling after modulo scheduling for the 16-Point Digital FIR Filter</td>
<td>186</td>
</tr>
<tr>
<td>C.22</td>
<td>Incremental scheduling after modulo scheduling for the Fifth-Order Elliptic Filter with Non-Pipelined Multipliers</td>
<td>186</td>
</tr>
<tr>
<td>C.23</td>
<td>Incremental scheduling after modulo scheduling for the Fifth-Order Elliptic Filter with Pipelined Multipliers</td>
<td>186</td>
</tr>
<tr>
<td>C.24</td>
<td>Incremental scheduling after modulo scheduling for the Fast Discrete Cosine Transform</td>
<td>186</td>
</tr>
<tr>
<td>C.25</td>
<td>Register requirements in UNRET for superscalar processors by using an architecture with 1 FU of each type</td>
<td>187</td>
</tr>
<tr>
<td>C.26</td>
<td>Register requirements in UNRET for VLIW processors by using an architecture with 1 FU of each type</td>
<td>188</td>
</tr>
<tr>
<td>C.27</td>
<td>Comparison of register requirements for superscalar processors by using 1 FU of each type</td>
<td>189</td>
</tr>
<tr>
<td>C.28</td>
<td>Comparison of register requirements for VLIW processors by using 1 FU of each type</td>
<td>190</td>
</tr>
<tr>
<td>C.29</td>
<td>Register requirements in UNRET for superscalar processors by using an architecture with 3 FP adders, 2 FP multipliers, 1 FP divisor and 2 load/store units</td>
<td>191</td>
</tr>
<tr>
<td>C.30</td>
<td>Register requirements in UNRET for VLIW processors by using an architecture with 3 FP adders, 2 FP multipliers, 1 FP divisor and 2 load/store units</td>
<td>192</td>
</tr>
<tr>
<td>C.31</td>
<td>Comparison of register requirements for superscalar processors by using 3 FP adders, 2 FP multipliers, 1 FP divisor and 2 load/store units</td>
<td>193</td>
</tr>
<tr>
<td>C.32</td>
<td>Comparison of register requirements for VLIW processors by using 3 FP adders, 2 FP multipliers, 1 FP divisor and 2 load/store units</td>
<td>194</td>
</tr>
<tr>
<td>C.33</td>
<td>Register reduction in a modulo scheduling algorithm by assuming a superscalar processor with 1 FU of each type</td>
<td>196</td>
</tr>
<tr>
<td>C.34</td>
<td>Register reduction in a modulo scheduling algorithm by assuming a VLIW processor with 1 FU of each type</td>
<td>197</td>
</tr>
<tr>
<td>C.35</td>
<td>Register reduction in a modulo scheduling algorithm by assuming a superscalar processor with 3 FP adders, 2 FP multipliers, 1 FP divisor and 2 load/store units</td>
<td>198</td>
</tr>
<tr>
<td>C.36</td>
<td>Register reduction in a modulo scheduling algorithm by assuming a VLIW processor with 3 FP adders, 2 FP multipliers, 1 FP divisor and 2 load/store units</td>
<td>199</td>
</tr>
<tr>
<td>C.37</td>
<td>Incremental scheduling in a modulo scheduling algorithm by assuming a superscalar processor with 1 FU of each type</td>
<td>200</td>
</tr>
<tr>
<td>C.38</td>
<td>Incremental scheduling in a modulo scheduling algorithm by assuming a VLIW processor with 1 FU of each type</td>
<td>201</td>
</tr>
<tr>
<td>C.39</td>
<td>Incremental scheduling in a modulo scheduling algorithm by assuming a superscalar processor with 3 FP adders, 2 FP multipliers, 1 FP divisor and 2 load/store units</td>
<td>202</td>
</tr>
<tr>
<td>C.40</td>
<td>Incremental scheduling in a modulo scheduling algorithm by assuming a VLIW processor with 3 FP adders, 2 FP multipliers, 1 FP divisor and 2 load/store units</td>
<td>203</td>
</tr>
</tbody>
</table>
# LIST OF ALGORITHMS

Chapter 3

3.1 Algorithm to unroll a π-graph m times 54

Chapter 5

5.1 List Scheduling Algorithm 74

Chapter 6

6.1 Algorithm to compute a Farey fraction by using the next one 90
6.2 Retiming dependences not belonging to recurrences 95
6.3 Algorithm to find a schedule in a given number of cycles 100
6.4 Optimized retiming_and_scheduling algorithm 102
6.5 UNRET Algorithm 103

Chapter 7

7.1 Function reduce_scheduling_dependences 129
7.2 Function reduce_local_maxima 131
7.3 Function reduce_span 133
7.4 Function incremental_scheduling 135

Chapter 8

8.1 Algorithm to increase the architecture 149
8.2 Algorithm to reduce area cost 150
8.3 Algorithm to find the maximum-throughput schedule 151
This work presents three algorithms to solve three different problems:

- **UNRET** is proposed to solve loop pipelining with resource constraints.
- **TCLP** is proposed to solve loop pipelining with timing constraints.
- **RESIS** is proposed to reduce the number of registers required by a schedule.

Loop pipelining with resource constraints can be defined as follows: "given a set of resources, finding a pipelined schedule of a loop in the minimum number of cycles". Loop pipelining with timing constraints can be defined as follows: "given a maximum time to execute an iteration of a loop, finding a schedule which requires the minimum set of resources (or the minimum area)". Whilst loop pipelining with timing constraints is a typical problem in the high-level synthesis of VLSI circuits, loop pipelining with resource constraints is present in both the high-level synthesis of VLSI circuits and compilers for parallel architectures.

In parallel architectures, the number of registers available to store partial results (during loop execution) is limited, and it is defined by the architecture. In a VLSI circuit, a register consumes space in the chip. Therefore, it is interesting in both areas to obtain a schedule which requires as few registers as possible.

UNRET and TCLP are related to the extraction of the parallelism of a loop. Chapter 1 introduces different ways to exploit such a parallelism, as well as the subjects on which this work is focused: high-level synthesis of VLSI circuits and compilation techniques for superscalar and VLIW processors.

UNRET and TCLP belong to a family of techniques known as software pipelining. An overview and classification of such techniques is presented in Chapter 2.

In Chapter 3 we define two transformations to exploit the parallelism in a loop: dependence retiming and loop unrolling. Both transformations will be used by UNRET and TCLP. The maximum parallelism available for exploitation in a loop is limited. Chapter 3 also shows how this maximum parallelism can be calculated.

A data dependence exists between two instructions when the result produced by the first one is consumed by the second one. Data dependences impose a partial execution order in the instructions of a loop. According to whether a data dependence does not influence in the scheduling, or it influences the scheduling within an iteration or across consecutive iterations, we classify data dependences into three categories: free scheduling dependences, positive scheduling dependences and negative scheduling dependences. Chapter 4 presents the theory behind this classification.
Chapter 5 describes the scheduling algorithm used by UNRET and TCLP. The algorithm takes resources into account, as well as multiple-cycle (possibly pipelined) functional units and instructions that have complex execution patterns (they use several functional units during several cycles).

Chapter 6 presents UNRET. UNRET uses dependence retiming and loop unrolling to find a pipelined schedule of the loop. Loop unrolling is in general required to extract the maximum parallelism. Dependence retiming enables us to obtain different (but equivalent) configurations for the same (possibly unrolled) loop. Each configuration is scheduled by using the algorithm presented in Chapter 5, attempting to find a schedule which executes the loop with the maximum parallelism. When no schedule exists for any configuration of the loop, UNRET decides a new target parallelism (and unrolling degree) and explores new configurations.

Once a schedule has been found, the number of required registers is reduced while maintaining the parallelism. In Chapter 7 we propose RESIS, an algorithm oriented to such a purpose. RESIS works in two phases. First, several configurations of the loop are explored, attempting to reduce the number of different iterations involved in the pipelined schedule. Each configuration is independently scheduled by using the algorithm from Chapter 5. Following this, some instructions are individually rescheduled in order to reduce the register requirements.

Chapter 8 presents TCLP, an algorithm for loop pipelining with timing constraints. TCLP is based on ideas similar to UNRET. The timing constraint is given in the form of a maximum number of cycles to execute each iteration of the loop. TCLP analytically calculates a minimum set of resources (theoretically) required to execute the loop with the given timing constraint. Several configurations of the loop are explored in order to find a schedule by using the calculated set of resources. If no schedule is found, the set of resources is successively increased and new configurations are explored until a schedule is found. Once a schedule fulfilling the given timing constraint has been found, TCLP attempts to optimize several characteristics of the schedule. First of all, TCLP attempts to reduce the set of resources while maintaining the length of the schedule. Following this, it attempts to increase the parallelism of the schedule by exploring different unrolling degrees. Finally, RESIS is used to reduce the number of required registers.

Chapter 9 presents the conclusions of this work, summarizes the main contributions performed and indicates future areas of work.