Adaptive streaming applications : analysis and implementation models
Zhai, J.T.

Citation
Zhai, J. T. (2015, May 13). *Adaptive streaming applications : analysis and implementation models*. Retrieved from https://hdl.handle.net/1887/32963

Version:	Not Applicable (or Unknown)
License:	[Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden](#)
Downloaded from:	https://hdl.handle.net/1887/32963

Note: To cite this publication please use the final published version (if applicable).

Cover Page

Universiteit Leiden

Leiden University Repository

The handle http://hdl.handle.net/1887/32963 holds various files of this Leiden University dissertation

Author: Zhai, Jiali Teddy
Title: Adaptive streaming applications : analysis and implementation models
Issue Date: 2015-05-13

Adaptive Streaming Applications: Analysis and Implementation Models

Jiali Teddy Zhai

Adaptive Streaming Applications: Analysis and Implementation Models

PROEFSCHRIFT

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van Rector Magnificus prof.mr. C.J.J.M. Stolker,
volgens besluit van het College voor Promoties
te verdedigen op woensdag 13 mei 2015
klokke 10:00 uur

door

Jiali Teddy Zhai
geboren in 1982

Samenstelling promotiecommissie:

Promotor:	Prof. Dr. Ir. Ed F.A. Deprettere	Universiteit Leiden
Co-Promotor:	Dr. Todor P. Stefanov	Universiteit Leiden
Overige leden:	Prof. Dr.-Ing. Jügen Teich	Universität Erlangen-Nürnberg
	Dr. Ingo Sander	KTH Kungliga Tekniska högskolan
	Prof. Dr. Ir. Twan A.A. Basten	Technische Universiteit Eindhoven
	Prof. Dr. Joost N. Kok	Universiteit Leiden
	Prof. Dr. Farhad Arbab	Universiteit Leiden
	Prof. Dr. Harry A.G. Wijshoff	Universiteit Leiden

Adaptive Streaming Applications: Analysis and Implementation Models
Jiali Teddy Zhai. -
Dissertation Universiteit Leiden. - With ref. - With summary in Dutch.

Copyright © 2015 by Jiali Teddy Zhai. All rights reserved.

This dissertation was typeset using LATEX in Linux.
Cover designed by Shanshan Yang

Printed in the Netherlands.

Contents

Table of Contents v

List of Figures ix

List of Tables xiii

1 Introduction 1

1.1 Embedded Streaming System Design 2

1.1.1 Design Requirements 3

1.1.2 Application Characterization 4

1.1.3 Platform Implications 7

1.1.4 Model-based Design Methodology 9

1.1.5 Summary 14

1.2 Problem Statement 15

1.3 Research Contributions 17

1.4 Thesis Organization 19

2 Models-of-Computation (MoC) 21

2.1 Polyhedral Process Networks (PPN) 21

2.2 Actor-based Data Flow MoCs 29

2.2.1 Synchronous Data Flow (SDF) 30

2.2.2 Cyclo-Static Data Flow (CSDF) 32

2.3 Hard Real Time Scheduling of Acyclic (C)SDF Graphs 33

3 Automated Analysis Model Construction: Deriving CSDF from Equivalent PPN 39

3.1 The Algorithm 40

3.2 Experimental Results 47

4 **Exploiting Maximum Data-level Parallelism without Inter-processor Communication** **51**
4.1 Motivating Example 53
4.2 Related Work 57
4.3 Finding all Dependences in a PPN 58
4.4 Computing the Number of Communication-free Partitions 60
4.5 Communication-free Partitioning Algorithm 64
4.6 Experimental Results 69

5 **Exploiting Just-enough Parallelism in Hard Real-time Systems** **75**
5.1 Related Work 77
5.2 Unfolding of SDF Graphs 78
5.3 Problem Formulation 80
5.4 Period Scaling under Hard Real-time Scheduling 83
5.5 Bounding Solution Space 83
5.6 The Algorithm 87
5.7 Experimental Evaluation 92

6 **A New MoC for Modeling Adaptive Streaming Applications** **99**
6.1 Related Work 100
6.2 Model Definition 101
6.2.1 Parameterized Polyhedral Process Networks 101
6.2.2 Operational Semantics 104
6.3 Consistency 107
6.4 Experimental Results 111

7 **Hard Real-time Scheduling of Adaptive Streaming Applications** **115**
7.1 Related Work 117
7.2 Model Definition 118
7.2.1 Mode-Aware Data Flow (MADF) 118
7.2.2 Operational Semantics 123
7.2.3 Mode Transition 125
7.3 Hard real-time Scheduling of MADF 130
7.4 Case Study 139

8 **Summary and Outlook** **145**

Bibliography **149**

Samenvatting **163**

List of Publications **169**

Index **171**

List of Abbreviations **173**

List of Figures

1.1 Three examples of embedded streaming applications. 2
1.2 Block diagram of a H.264 decoder (taken from [18]). Each task is represented by a rectangular block. 5
1.3 Processing part of an MPSoC platform for mobile devices from Nvidia (taken and simplified from [11]). 8
1.4 Template of a baseband processor (taken and simplified from [103]). All interfaces and peripheries are omitted. 8
1.5 X-chart: a general design process (adopted from [50]). 9
1.6 Comparison of dataflow MoCs for streaming applications (taken and extended from [112]). The MoCs underlined are proposed in this thesis. 10
1.7 Daedalus^{RT} design flow. The grey boxes highlight the contributions of this thesis. The dashed box and lines denote the parts that are currently not fully implemented. 13

2.1 A polyhedron. 23
2.2 The polyhedral representation of the execution of function `read_image` in Listing 1. 29
2.3 PPN corresponding to the SANLP in Listing 1. 29
2.4 An example of an image filter algorithm modeled using the SDF MoC. 30
2.5 A CSDF graph G_1 of a pacemaker application (taken from [99]). . . . 33
2.6 An example of a CSDF graph and its real-time task-set representation. Since the execution of the actors repeats indefinitely, the last execution of $A_{2,1}$,$A_{3,1}$,$A_{3,2}$, and $A_{3,3}$ in the figure is truncated and shown in black. 36

3.1 CSDF graph equivalent to the PPN shown in Figure 2.3. 41
3.2 Domains of process snk in Figure 2.3. 42

3.3 Suffix tree for the sequence of process variants S_{snk}. The tildes represent the omitted part of the tree. 46

4.1 An example of a PPN and its communication-free partitions. 54
4.2 Mapping of the PPN in Figure 4.1(a) onto 2 PEs achieving the maximum performance. 55
4.3 Performance results of mapping the initial PPN and the alternative PPN after communication-free partitioning. 55
4.4 The PPN in Figure 4.1(a) after communication-free partitioning and its mapping. 56
4.5 Domain of PPN process P_3 in Figure 4.1(a). The input port domain of IP_3 (surrounded by the solid triangle), output port domain of OP_3 (surrounded by the dotted triangle), and dependence relation R_3 (denoted by the arrows between dots). 59
4.6 Finding transitive dependences of the PPN. 61
4.7 The PPN in Figure 4.1(a) after communication-free partitioning. . . . 66
4.8 Performance results of mapping the MJPEG encoder onto (a) FPGA-based MPSoC platforms and onto (b) a desktop multi-core platform. 70
4.9 Performance results of mapping the FM radio application onto (a) FPGA-based MPSoC platforms and onto (b) a desktop multi-core platform. 71

5.1 (a) An example of an SDF graph and (b) its equivalent CSDF graph by unfolding actor A_3 by factor 3. 80
5.2 G_3: Optimal alternative graph of G_1 in Figure 5.1(a) with unfolding factors $f_2 = 2, f_3 = 4$ when scheduled on 2 PEs. 85
5.3 The list produced by the algorithm for G_1 in Figure 5.1(a) on 2 PEs with $\rho = 0.95$. 91
5.4 Period ratio (lower is better). 94
5.5 Running time of our algorithm. 95
5.6 An example of an individual. The first replica of A_1 is allocated on the jth PE and the $\hat{f}_1$th replica of A_1 does not exist. 96
5.7 The ratios of total execution time Ω_t and total code size Ω_Θ for GA and our algorithm. 98

6.1 Comparsion between a PPN and a P^{3}N. 102
6.2 Control process and evaluation function. 106
6.3 Consistent execution of process P_2 and P_3 w.r.t. edge E_3. 108
6.4 Which combinations (M,N) do ensure consistency of P^{3}N? 108
6.5 Two alternatives of Function `Check` in Figure 6.2(b). 111

6.6 P^{3}N of our experiment 112
6.7 Performance results of PPN and P^{3}N implementations 113

7.1 An example of MADF graph (G_1). 119
7.2 Two modes of the MADF graph in Figure 7.1. 123
7.3 Execution of two iterations of both modes SI^1 and SI^2 under self-timed scheduling. 125
7.4 An execution of G_1 in Figure 7.1 with two mode transitions under the ST transition protocol. *MCR1* at time t_{MCR1} denotes a transition request from mode SI^2 to SI^1, and *MCR2* at time t_{MCR2} denotes a transition request from mode SI^1 to SI^2. 127
7.5 An illustration of the Maximum-Overlap Offset (MOO) calculation. 128
7.6 The execution of G_1 with two mode transitions under Maximum-Overlap Offset (MOO) protocol. 129
7.7 Upper bounds of earliest starting times for transition from mode SI^2 to SI^1. 133
7.8 Earliest starting times for transition from mode SI^2 to SI^1 with the MOO protocol. 134
7.9 Allocation of all MADF actors in Figure 7.1 to 3 PEs. 136
7.10 Earliest starting times for transition SI^2 to SI^1 on 2 PEs shown in Figure 7.9. 137
7.11 MADF graph of Vocoder. 141
7.12 Allocation of dataflow actors of Vocoder to 4 PEs. The control edges are omitted to avoid cluttering. 143

List of Tables

1.1 Processing requirements for video decoding (taken from [19]). 3
1.2 Processing requirements for image processing (taken from [43]). All applications are assumed to operate at 24 FpsFrame per second. 3

2.1 Mathematical notations. 22
2.2 Polyhedral notations. 23
2.3 Data flow notations. 31
2.4 Notations for HRT scheduling of CSDF MoCs. 34

3.1 Additional notations used in Chapter 3 besides the ones introduced in Chapter 2. 39
3.2 Consumption/production sequences for actor snk in Figure 3.1. 47
3.3 Characteristics of benchmarks and running times to derive their corresponding CSDF graphs. 48
3.4 Execution times of the phases in the Daedalus^{RT} flow for three streaming applications on a single MPSoC platform. 48

4.1 Execution time on benchmarks. 73

5.1 Additional notations used in Chapter 5 besides the ones introduced in Chapter 2. 81
5.2 Benchmark characteristics. 93
5.3 Parameters for the genetic algorithm. 97

7.1 Additional notations used in Chapter 7 besides the ones introduced in Chapter 2. 116
7.2 Mapping relation M_2 for actor A_2 in Figure 7.1. 120
7.3 Function MC_5 defined for actor A_5 in Figure 7.1. 120
7.4 Actor parameter for G_1 in Figure 7.1. 132
7.5 WCETs of all actors in Vocoder (in clk.). 142

7.6 Performance results of four modes of Vocoder in the steady-state. . . 143
7.7 Performance results for all mode transitions of Vocoder. 144