Imperial College London
Department of Computing

Optimising reconfigurable systems for real-time applications

Thomas Chun Pong Chau

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of the Imperial College
September 2014

Declaration of Originality

This thesis is a presentation of my original research work. The contributions of others are involved, every effort is made to indicate this clearly in the references to the literature and acknowledgement of collaborative research.

Copyright Declaration

Abstract

This thesis proposes novel approaches to design and optimise reconfigurable systems targeting real-time applications.

Our first contribution of this thesis is to propose novel data structures and memory architectures for accelerating real-time proximity queries, with potential application to robotic surgery. We optimise performance while maintaining accuracy by several techniques including mixed precision, function transformation and streaming data flow. Significant speedup is achieved using our reconfigurable accelerator platforms over double-precision CPU, GPU and FPGA designs.

The second contribution of this thesis is an adaptation methodology for real-time sequential Monte Carlo methods. Adapting to workload over time, different configurations with various performance and power consumption tradeoffs are loaded onto the FPGAs dynamically. Promising energy reduction has been achieved in addition to speedup over CPU and GPU designs. The approach is evaluated in an application to robot localisation.

The third contribution of this thesis is a design flow for automated mapping and optimisation of real-time sequential Monte Carlo methods. Machine learning algorithms are used to search for an optimal parameter set to produce the highest solution quality while satisfying all timing and resource constraints. The approach is evaluated in an application to air traffic management.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my advisor Professor Wayne Luk. Only through his enthusiasm for research, timely words of encouragement, and immense amount of patience, have I been able to develop into the person that I am today. His guidance has helped me improve my papers, presentations, and this thesis tremendously. His trust has always supported me throughout my research. I could not have imagined having a better advisor for my Ph.D. study.

I would like to thank my secondary advisor Professor Peter Y.K. Cheung and my master program's advisor Professor Philip Leong. They opened the door to reconfigurable computing research for me, and have inspire me throughout these years. I deeply appreciate their insightful comments.

Special thanks are due to Alison Eele and Professor Jan Maciejowski in University of Cambridge, and Benjamin Cope and Kathryn Cobden in Altera Corporation, for their collaboration in our project on sequential Monte Carlo methods and air traffic management.

It is also a real privilege for me to study in the Custom Computing Group, Department of Computing at Imperial College London. Thanks are specially given to Maciej Kurek, Xinyu Niu, Kuen Hung Choi, Gary Chow and Ka-Wai Kwok for all the invaluable discussions and collaboration. Thanks also to my past and present colleagues in the lab: James Arram, Tobias Becker, Brahim Betkaoui, Pavel Burovskiy, Bridgette Cooper, Kit Cheung, Gabriel De Figueiredo Coutinho, Stewart Denholm, Paul Grigoras, Ce Guo, Liucheng Guo, Eddie Hung, Gordon Inggs, Qiwei Jin, Adrien Le Masle, Nicholas Ng, Shengjia Shao, Timothy Todman, Anson Tse, Shulin Yan, Jinzhe Yang. I extend my gratitude to the project students, James Targett, Marlon Wijeyasinghe, Jake Humphrey, Georgios Skouroupathis, in the Department of Computing and the Department of Electrical and Electronic Engineering for all the productive work.

I am grateful for an internship opportunity working in ARM Ltd. at Cambridge, UK. It was a fruitful experience and I would like to thank my manager and mentor William Wang for sharing his expertise and broadening my knowledge.

I am thankful for the generous financial support provided by the Croucher Foundation, UK En-

gineering and Physical Sciences Research Council, and the European Union Seventh Framework Programme.

Lastly, my warmest thanks go to my wife, Kaijia, for her continued support throughout the years. She is such an excellent cook that keeps me eating and studying well.

Dedication

To my parents,

for making me be who I am, and giving me the best education you could;

and to Kaijia,

for her patience, understanding and care during my hours of research, contemplation and writing.

Publications

I T. C. P. Chau, X. Niu, A. Eele, J. M. Maciejowski, P. Y. K. Cheung, and W. Luk, "Mapping adaptive particle lters to heterogeneous recongurable systems," *ACM Transactions on Recongurable Technology and Systems*, accepted.

II T. C. P. Chau, M. Kurek, J. S. Targett, J. Humphrey, G. Skouroupathis, A. Eele, J. Maciejowski, B. Cope, K. Cobden, P. Leong, P. Y. K. Cheung, and W. Luk, "SMCGen: Generating reconfigurable design for sequential Monte Carlo applications," in *Proceedings of International Symposium on Field-Programmable Custom Computing Machines*, 2014.

III M. Kurek, T. Becker, T. C. P. Chau, and W. Luk, "Automating optimization of reconfigurable designs," in *Proceedings of International Symposium on Field-Programmable Custom Computing Machines*, 2014.

IV T. C. P. Chau, K.-W. Kwok, G. C. T. Chow, K. H. Tsoi, Z. Tse, P. Y. K. Cheung, and W. Luk, "Acceleration of real-time proximity query for dynamic active constraints," in *Proceedings of International Conference on Field-Programmable Technology*, 2013.

V T. C. P. Chau, J. S. Targett, M.Wijeyasinghe, W. Luk, P. Y. K. Cheung, B. Cope, A. Eele, and J. M. Maciejowski, "Accelerating sequential Monte Carlo method for real-time air traffic management," *SIGARCH Computer Architecture News*, vol. 41, no. 5, 2013.

VI T. C. P. Chau, X. Niu, A. Eele, W. Luk, P. Y. K. Cheung, and J. M. Maciejowski, "Heterogeneous reconfigurable system for adaptive particle filters in real-time applications," in *Proceedings of International Symposium Applied Reconfigurable Computing*, 2013.

VII A. Eele, J. M. Maciejowski, T. C. P. Chau, and W. Luk, "Parallelisation of sequential Monte Carlo for real-time control in air traffic management," in *Proceedings of International Conference Decision and Control*, 2013.

VIII A. Eele, J. M. Maciejowski, T. C. P. Chau, and W. Luk, "Control of aircraft in the terminal manoeuvring area using parallelised sequential Monte Carlo," in *Proceedings of AIAA Conference on Guidance, Navigation, and Control*, 2013.

IX X. Niu, T. C. P. Chau, Q. Jin, W. Luk, and Q. Liu, "Automating elimination of idle functions by run-time reconfiguration," in *Proceedings of International Symposium on Field-Programmable Custom Computing Machines*, 2013.

X T. C. P. Chau, W. Luk, and P. Y. K. Cheung, "Roberts: Reconfigurable platform for benchmarking real-time systems," *SIGARCH Computer Architecture News*, vol. 40, no. 5, 2012.

XI T. C. P. Chau, W. Luk, P. Y. K. Cheung, A. Eele, and J. M. Maciejowski, "Adaptive sequential Monte Carlo approach for real-time applications," in *Proceedings of International Conference Field Programmable Logic and Applications*, 2012.

Contents

Declaration of Originality **3**

Copyright Declaration **5**

Abstract **5**

Acknowledgements **9**

Dedication **11**

Publications **13**

List of Tables **21**

List of Figures **23**

Glossary **27**

1 Introduction **29**

1.1 Motivation 29

1.2 Research Challenges and Contributions 31

1.2.1 Precision Optimisation of Reconfigurable Data-paths 32

1.2.2 Run-time Adaptation of System Configuration 34

1.2.3 Design Flow for Domain-specific Reconfigurable Applications 35

1.3 Thesis Organisation 36

2 Background and Related Work **38**

2.1 Introduction 38

2.2 Reconfigurable Systems 38

2.2.1 Architecture 38

2.2.2 Design Flow 40

2.2.3 Domain Specific Languages 43

2.3 Real-time Systems 44

2.3.1 Real-time Applications 45

2.4 Summary 54

3 Precision Optimisation of Data-paths **55**

3.1 Introduction 55

3.2 Formulation of PQ 57

3.3 Optimisation for Reconfigurable Hardware 61

3.3.1 Transformation of Trigonometric and Search Functions 61

3.3.2 Applying Reduced Precision 62

3.3.3 Finding the Right Precision 63

3.4 Reconfigurable System Design 64
3.4.1 Streaming Data Structure 65
3.4.2 System Architecture 65
3.4.3 Performance Estimation 67
3.5 Experimental Evaluation 70
3.5.1 General Settings 70
3.5.2 Parallelism versus Precision 71
3.5.3 Ratio of Re-computation versus Precision 71
3.5.4 Comparison: CPU, GPU and Reconfigurable System 72
3.6 Summary 74

4 Run-time Adaptation of System Configuration 76

4.1 Introduction 76
4.2 Adaptive Particle Filter 77
4.3 Heterogeneous Reconfigurable System 81
4.3.1 Mapping Adaptive SMC to Reconfigurable System 82
4.3.2 FPGA Kernel Design 82
4.3.3 Timing Model for Run-time Reconfiguration 84
4.4 Optimising Transfer of Particle Stream 87
4.5 Experimental Results 90
4.5.1 System Settings 91
4.5.2 Adaptive SMC versus Non-adaptive SMC 92

4.5.3 Data Compression 93

4.5.4 Performance Comparison of Reconfigurable System, CPU and GPU . . . 94

4.6 Summary 98

5 Design Flow for Domain-specific Reconfigurable Applications 99

5.1 Introduction 99

5.2 SMC Design Flow 100

5.2.1 Specifying Application Features 102

5.2.2 Computation Engine Design 103

5.2.3 Performance Model 108

5.3 Optimising SMC Computation Engine 110

5.3.1 Compile-time Parameters 110

5.3.2 Run-time Parameters 111

5.3.3 Parameter Optimisation 113

5.4 Evaluation 114

5.4.1 Design Productivity 114

5.4.2 Application 1: Mobile Robot Localisation 115

5.4.3 Application 2: Air Traffic Management 117

5.5 Summary 119

6 Conclusion 120

6.1 Summary of Achievements 120

6.2 Future Work 123

6.2.1 Proximity Query Formulation 123

6.2.2 Adaptive Sequential Monte Carlo Methods 125

Bibliography **129**

List of Tables

2.1 SMC design parameters. Dynamic: adjustable at run-time; Static: fixed at compile-time. 50

2.2 Variables in air traffic management model. 54

3.1 Parameters of the performance model. 70

3.2 Comparison of PQ computation in 1 ms using CPU-based system (CPU), GPU-based system (GPU), double precision FPGA-based reconfigurable system (RS DP) and FPGA+CPU reconfigurable system with reduced precision (RS RP). . 74

4.1 Comparison of adaptive and non-adaptive SMC on reconfigurable system. 93

4.2 Performance comparison of reconfigurable system (RS), CPU and GPU. 96

5.1 Lines of code for two SMC applications under the proposed design flow. 115

5.2 Performance comparison of robot localisation. 116

5.3 Parameter optimisation of air traffic management system using machine learning approach. 117

5.4 Performance comparison of air traffic management. 118

List of Figures

1.1 Illustration of heterogeneous processing topologies: (a) Pre-processing by FPGAs; (b) Co-processing between FPGAs and CPUs. 33

1.2 Thesis organisation. 37

2.1 Island-style FPGA (L: LUTs and coarse-grained resources; C: Connection boxes; S: Switch boxes). 39

2.2 Design flow of FPGAs. 41

2.3 Design flow of FPGA with OpenSPL and OpenCL. 42

2.4 MathWorks' model-based design flow. 43

2.5 Various sets of points aligned on a series of contours; A set of points located on an arbitrary form of mesh. 46

2.6 (a) A virtual tube bounded by a series of contour denotes the configuration of an endoscope; (b) The corresponding three-dimensional distance map in grids of 86x48x43. 47

2.7 An overview of the air traffic control problem. 52

2.8 Aircraft model. 53

3.1 Various sets of points aligned on a series of contours; A set of points located on an arbitrary form of mesh. 58

3.2 Data structure: N_S points are processed in a group. Each point of a group is iterated for N_C times. Data are streamed in an order as indicated by the arrows. 65

3.3 System architecture: Solid lines represent communication on the FPGA board while dotted lines represent the bus connecting the reduced precision data-path on FPGA to the high precision data-path on CPU. 66

3.4 Memory array stores contour indices for re-computation. 68

3.5 Computation time and the level of parallelism versus different number of mantissa bits. 72

3.6 Ratio of re-computation and the number of points processed in 1 ms versus different number of mantissa bits. 73

3.7 Computation time for a PQ update with 100 contours versus the number of points. 75

4.1 Particle set reduction. 81

4.2 Heterogeneous reconfigurable system 83

4.3 A particle stream. 83

4.4 FPGA kernel design. 85

4.5 Power consumption of the reconfigurable system over time. 88

4.6 Compressing particle stream: After the resampling process, some particles are eliminated and the remaining particles are replicated. Data compression is applied so that every particle is stored and transferred once only. 89

4.7 Number of particles and components of total computation time versus wall-clock time. 93

4.8 Localisation error versus wall-clock time. 94

4.9 Effect on the data transfer time by particle stream compression. 94

4.10 Power consumption of reconfigurable system (RS), CPU and GPU in one time-step, notice that the computation time of the CPU system exceeds the 5-second real-time requirement 97

4.11 Run-time versus energy consumption of reconfigurable system (RS), CPU and GPU 97

5.1 Design flow (Compile-time and run-time) for SMC applications. Users only customise the application-specific descriptions inside the dotted box. 101

5.2 (a) Design of the SMC computation engine: Solid lines represent data paths while dotted lines represent control paths; (b) Data structure of particles represented by three data streams. 106

5.3 FPGA kernel design: The blocks that require users' customisation are darkened. The dotted box covers the blocks that are optional on FPGAs. 107

5.4 Parameter space of robot localisation system (N_A=8192, S=1): The dark region on the top-right indicates designs which fail localisation accuracy constraints, while those on the bottom-left indicates designs which fail real-time requirements. 111

5.5 Power consumption of the reconfigurable system with reconfiguration to low-power mode during idle 113

5.6 Illustration of automatic parameter optimisation: (a) Sampling parameter sets; (b) Building surrogate model; (c) Calculating expected improvement; (d) Moving to the point offering the highest improvement. 114

5.7 Number of particles and components of total computation time versus wall-clock time 116

5.8 Power consumption of reconfigurable system, CPU and GPU in one time-step . 117

6.1 Thesis contributions. 123

6.2 Image-guided catheterisation: Perform PQ based on a beating heart model, where light blue bubbles represent the control points registered on the surface and yellow spheres indicate the control points forming the centre line of the pathway [1]. 125

6.3 Altera SOC which integrates an ARM-based hard processor, peripherals, memory interfaces and FPGA fabric [2]. 125

6.4 Different schemes to put FPGA to sleep. 126

6.5 (a) Best-effort adaptive scheme described in Chapter 4; (b) Just-in-time adaptive scheme. 127

Glossary

AMBA Advanced Microcontroller Bus Architecture. 124

ASIC Application-Specific Integrated Circuit. 29, 126, 127

CLB Configurable Logic Block. 39

CPU Central Processing Unit. 31, 33–36, 41, 56, 64, 66–74, 76–78, 80, 82, 84–87, 91–98, 100, 102–104, 106–109, 112, 114–118, 120–122, 124, 127, 128

DCL Domain customisable Language. 43

DFS Dynamic Frequency Scaling. 126

DRAM Dynamic Random-Access Memory. 67, 69, 70, 82, 88, 109

DSL Domain Specific Language. 43, 44

DSP Digital Signal Processor. 29, 39, 40, 70, 91, 115, 120

FF Flip-Flop. 91

FPGA Field-Programmable Gate Array. 29–36, 38–44, 47, 48, 51, 55, 56, 61, 64–74, 76–84, 86, 87, 89–91, 93–98, 100, 102–105, 108–112, 114–118, 120–122, 124, 126, 127

GPU Graphics Processing Unit. 33–36, 41, 56, 70, 72–74, 77, 82, 92, 94–96, 98, 100, 114–118, 120–122

HDL Hardware-description Language. 35, 40, 42

HLS High-level Synthesis. 41, 42, 54

HPC High-Performance Computing. 30, 34, 40, 41

I/O Input/Output. 39, 120

IP Intellectual Property. 42

LUT Look-Up Table. 38–40, 91, 107, 115, 117, 127

PF Particle Filter. 48

PQ Proximity Query. 34, 36, 45–48, 55–57, 61, 62, 64, 65, 70–73, 121, 123, 124

RAM Random-Access Memory. 38–40, 91, 115, 117

RMSE Root-Mean-Square Error. 110, 111, 115, 116

RTL Register-transfer-level. 40, 42, 43, 54, 126, 127

RTOS Real-time Operating System. 30, 124, 127

SMC Sequential Monte Carlo. 34–37, 48–53, 76, 77, 90, 92, 93, 99–103, 110, 111, 113, 114, 119, 121, 122, 128

SOC System on a chip. 39, 124, 127

WCET Worst Case Execution Time. 30

Chapter 1

Introduction

1.1 Motivation

Reconfigurable system is a computing system which combines the flexibility of software with the high performance of hardware by deploying Field-Programmable Gate Array (FPGA) technology. The structure of a reconfigurable system can be changed by the end-user after the manufacturing process or even at run-time. The development of reconfigurable system has been driven largely by FPGAs, which are semiconductor devices with prefabricated logic and routing resources. The functionality and interconnection of an FPGA can be reconfigured with any new design multiple times. The reconfiguration enables application-specific tuning of designs and their adaptation to new standards. This reconfigurability make FPGAs suited to prototyping Application-Specific Integrated Circuit (ASIC) designs. In recent years, there has been a significant increase in the size of FPGAs. Modern state-of-the-art FPGAs contain numerous programmable logic cells, Digital Signal Processors (DSPs), memory blocks, high-throughput transceivers, peripheral devices, customisable IP blocks and even micro-processor cores [3, 4]. These components enable higher integration level, faster execution speed and lower power consumption through tailor-made data-paths, increased fine-grained parallelism and better memory utilisation. In addition, FPGA technology allows arbitrary precision floating-point arithmetic. This allows for reduction of the circuit area or increases parallelism without signifi-

cant impact to the accuracy of the results [5,6]. FPGA devices also support dynamic run-time reconfiguration enabling applications demanding adaptive and flexible hardware.

The advantages mentioned above facilitate the use of FPGAs in High-Performance Computing (HPC) and real-time computing. HPC has stringent speed, space and power consumption requirements. Examples include weather forecasting, brain simulation and molecular dynamics simulation. In the last decade, researchers have extended the scope of FPGAs from prototyping to accelerating a wide variety of software, such as Monte Carlo simulation [6] and video processing [7]. FPGAs show a lot of promise for HPC. Execution time and power consumption of applications running on FPGAs can be improved by up to several orders of magnitude compared to state-of-the-art microprocessors [5–9].

Real-time computing is driven by system being able to respond to input stimulus in finite intervals. Failing a deadline can cause degraded quality of service and even a total system failure. Real-time systems are found in a wide range of applications areas, from simple domestic appliances to financial systems, large scale process control and safety critical avionics. Examples include air traffic management [10,11], unmanned aerial vehicles [12], robotics [13], and medical surgery [14]. In some applications, the required response times are measured in milliseconds, in others it is seconds or even minutes. Nevertheless, they all must be satisfied.

FPGAs are considered as a platform for embedded real-time applications where software tasks running on micro-processors coexist with hardware tasks running on reconfigurable logics [15–18]. However, these implementations are software-based which means that multiple real-time tasks are managed by dedicated Real-time Operating System (RTOS) for reconfigurable devices. Real-time tasks running on micro-processors have to consider problems resulting from complex architectures. Pipelines, caches, branch prediction, or out-of-order execution result in complex state machines which to be managed by either software or hardware. Extensive research has been conducted on micro-processor-based real-time applications regarding time predictability, Worst Case Execution Time (WCET), task scheduling and management [19–21]. Nowadays, there are many real-time applications which push processing systems to their limit with the required response time. Real-time needs can be extremely hard when a large amount of data

have to be processed in a short period of time. For example, high-frequency trading is becoming popular with execution time bounded to microseconds [22].

FPGAs have potential to play an important role in high-performance real-time applications as they provide predictable timing performance, the ability to perform highly parallel calculations, better solution quality [23,24] and lower power consumption [25]. For instance, in Monte Carlo-based applications, FPGAs are able to simulate more paths, and therefore the result will be more accurate [24]. This thesis aims to study real-time systems from a different perspective, in particular about making use of the recent advancements in FPGA technology to bridge the gap between high performance and real-time computing. Essentially this research focuses on hardware-oriented approaches that utilise special-purpose reconfigurable hardware being tailored for target applications.

1.2 Research Challenges and Contributions

The objective of this thesis is to *optimise reconfigurable systems, particularly FPGAs, so as to improve the performance of real-time applications, and make the experience of implementing real-time applications on reconfigurable systems more convenient and effective.* The contributions of this thesis are based on an heterogeneous reconfigurable system. *Heterogeneous* means that more than one type of compute units are used to perform computation. In this thesis, heterogeneous reconfigurable system refers to one consisting of multiple FPGAs and Central Processing Units (CPUs). There are two different processing topologies that describe the distribution of workload between FPGAs and CPUs. Pre-processing topology in Figure 1.1(a) performs intensive number crunching in FPGAs prior to processing with CPUs, which only handle the non-compute-intensive calculations. Co-processing topology in Figure 1.1(b) splits the workload to take advantage of the different characteristics between FPGA and CPU. In this topology, FPGAs act as real-time co-processors having deterministic timing. The three main contributions made towards this goal are:

1. A precision optimisation approach that allows designers to maximise parallelism and

throughput subject to real-time requirements, without having to sacrifice accuracy of computed solutions. Aspects regarding streaming data structure, memory architecture, and hardware-friendly function transformation are also discussed. This work is published in paper [23], which leads to more effective use of arbitrary precision floating-point arithmetic offered by FPGA technology.

2. An adaptation technique that allows real-time system to reconfigure its hardware, software and algorithm at run-time for optimised performance while satisfying all the real-time constraints. This contribution employs the second type of heterogeneous processing topology as shown in Figure 1.1(b), numerically intensive processing is allocated to the FPGA. In papers [25, 26], we showed how FPGA technology enables dynamic workload management, frequency scaling, and bit-stream reconfiguration that lead to reduced energy consumption as well as better resource utilisation on real-time systems.

3. A design flow for generating efficient implementation of reconfigurable designs and reducing the development effort. The proposed design flow consists of a parametrisable computation engine and a software template, which maximise design reuse and minimise customisation effort. High-level functional description of the application is mapped to reconfigurable system automatically. Design parameters that are critical to the performance and to the solution quality are tuned using a machine learning algorithm. This process automatically maximises accuracy or minimises computation time without violating real-time constraints. This contribution is published in paper [24], which enables efficient mapping of a variety of designs to reconfigurable hardware.

The following three subsections present a brief overview of each contribution and the challenges involved. More detail are presented in the later chapters.

1.2.1 Precision Optimisation of Reconfigurable Data-paths

The first contribution of this thesis is a *precision optimisation approach to maximise real-time performance of reconfigurable systems.*

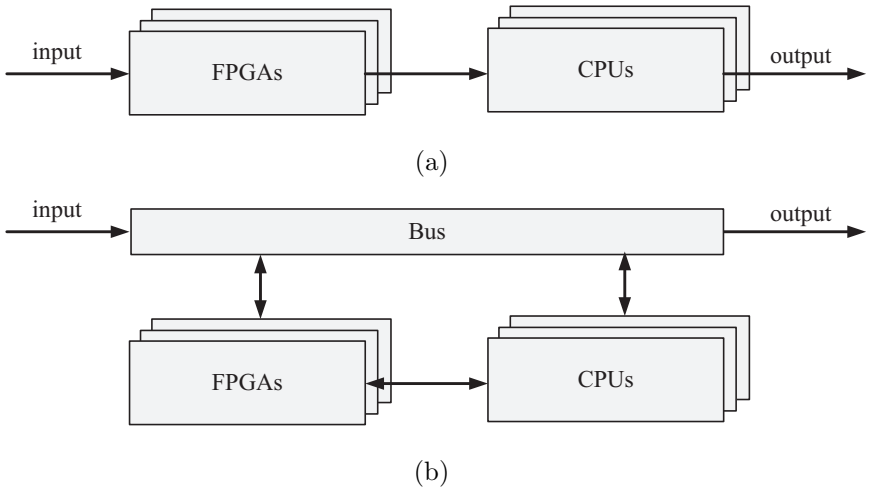


Figure 1.1: Illustration of heterogeneous processing topologies: (a) Pre-processing by FPGAs; (b) Co-processing between FPGAs and CPUs.

FPGAs have abundant fine-grained resources but the clock speeds of FPGAs are commonly 10 to 30 times slower than CPUs and Graphics Processing Unit (GPU), therefore the performance gains in FPGAs are obtained by designing algorithms such that many independent operations can occur simultaneously. A crucial step to unleash competitive performance of FPGAs is to provide massive parallelism and effective use of data. By employing significant data-path parallelism and deep pipelines where inputs and outputs continually stream through each cycle, hundreds or even thousands of operations are executed on each cycle of FPGAs to outweigh the slow clock frequencies. However, as each data-path requires replication of circuits and deep pipelines need numerous flip-flops, resource usage and bandwidth requirement are often the performance limitation for FPGA implementations [27].

Currently, FPGAs have the ability to support customisable data-paths with different precisions. Reduced precision data-paths consume less logic resource and hence allow for a higher degree of parallelism. Using reduced precision reduces I/O bandwidth and allows higher clock frequencies. Unfortunately, all the mentioned benefits come with an expense of lower accuracy of results. There are trade-offs between performance and accuracy in the implementation of data-paths.

Chapter 3 describes how reduced precision is applied to reconfigurable systems. A novel data structure and memory architecture are developed. They support reduced precision data-paths across multiple FPGAs. They maintain the accuracy of final results by re-computing a small

fraction of FPGA outputs on CPUs. This work employs the pre-processing topology (Figure 1.1(a)), and the data-paths on FPGAs compute and filter most of the data before sending the filtered data to CPUs for re-computation.

The proposed methodology is applied to an image-guided surgical robot application which employs the Proximity Query (PQ) process. Functional transformation further optimises the data-path for hardware-friendly implementation. Implementation in a reconfigurable platform with four FPGAs shows 58 times speedup over a 12-core CPU system, 3 times speedup over a GPU system, and 3 times speedup over the same reconfigurable platform without precision optimisation.

1.2.2 Run-time Adaptation of System Configuration

The second contribution of this thesis is *an approach that adapts the reconfigurable systems at run-time for reduced computation workload and energy consumption.*

Power and energy efficiency is becoming a major consideration for HPC systems. For example, the Green500 list [28] provides a ranking of the energy efficiency of world-wide supercomputers. CPUs are equipped with various technologies to reduce power dissipation [29, 30]. GPUs also have different power modes [31, 32]. As FPGAs are increasingly being deployed for HPC applications, power dissipation of FPGAs is also a concern. Apart from traditional power saving techniques such as clock gating and dynamic frequency/voltage scaling existing on other platforms, FPGAs' run-time reconfigurability could be exploited as an aggressive power saving technique. The power consumption of an FPGA depends on the circuit size and the clock frequency. Larger circuit uses more routing tracks which bring parasitic capacitance, and higher clock speed increases the switching activity on the routing tracks which causes significant power dissipation.

Chapter 4 explores an adaptation approach to reduce FPGA's energy consumption by run-time reconfiguration. In particular, Sequential Monte Carlo (SMC) applications are studied as they facilitate adaptation at algorithmic and system levels. At algorithmic level, an adaptive SMC

algorithm which adjusts the computation workload at run-time while maintaining the quality of results is proposed. At system level, run-time reconfigurability of FPGAs is used to switch the real-time system between computation mode and low-power mode. Low-power mode lowers the dynamic power by reducing circuit size and clock frequency. Compared to a non-adaptive and non-reconfigurable system, the proposed approach reduces idle power by 25-34% and the overall energy consumption by 17-33%.

This work employs the co-processing topology (Figure 1.1(b)), the FPGAs handle the computation which can be fully-pipelined, while the CPUs deal with non-sequential data access.

1.2.3 Design Flow for Domain-specific Reconfigurable Applications

The final contribution of this thesis is *the development of a design flow that reduces the development effort of real-time applications on reconfigurable systems.*

Although FPGAs show promising performance advantage for high-performance real-time systems, FPGA accelerators have not yet been accepted by mainstream application designers [27]. Low productivity and long design time have been a longstanding barriers to a more wide-spread usage. The design complexity of FPGA applications far exceeds that of CPUs and GPUs, and hence raises development cost and deter user acceptance. Traditionally, FPGA applications are developed using Hardware-description Languages (HDLs). Writing HDLs is timing-consuming and requires digital design expertise which is not common to mainstream designers. In addition, designers have to perform numerical analysis to determine an appropriate precision in order to achieve an FPGA's full potential, because FPGAs often achieve order of magnitude improvements when using fixed-point, integer, or bit-level operations. This numerical analysis process significantly increases design time. Another productivity bottleneck is lengthy compilation times due to the complexity of placement and routing. Common software design practices based on rapid compilation are no longer feasible for reconfigurable system design.

In Chapter 5, a design flow is proposed to address the above mentioned challenge. This chapter extends the SMC reconfigurable system described in Chapter 4 and focuses on making the sys-

tem parametrisable for a wide variety of SMC applications. In other words, it makes Chapter 4's reconfigurable system easier and more accessible to designers, especially those lack hardware design experience. Through templating the SMC structure, the proposed design flow enables efficient mapping of applications to multiple FPGAs. To reduce design space exploration effort, a machine learning algorithm based on surrogate modelling is used to tune design parameters that are crucial to the performance and solution quality. The design flow demonstrates its capability of producing reconfigurable implementations for a range of SMC applications that have significant improvement in speed and in energy efficiency over optimised CPU and GPU implementations.

1.3 Thesis Organisation

Chapter 2 offers a detailed background on reconfigurable architectures and systems, design flow which includes synthesis tools and programming languages, and applications of reconfigurable technologies on real-time systems. **Chapter 3** describes the first contribution, which demonstrates how precision optimisation is applied to reconfigurable real-time systems. The proposed methodology is applied to an application in imaged-guided surgical robot based on PQ process. **Chapter 4** presents the second contribution of this thesis, which describes the use of run-time reconfigurability of FPGAs to adapt real-time systems for reduced power and energy consumption. **Chapter 5** details the third contribution, which provides a design flow for automatically generating efficient implementation of reconfigurable designs. Lastly, Chapter 6 concludes this thesis, and presents the remaining outstanding challenges. Figure 1.2 shows how the three contributions of this thesis link together. Chapter 3 and 4 describe techniques to optimise reconfigurable real-time systems. Chapter 5 introduces a domain-specific design flow to address the long-standing programmability issues of FPGA.

Parts of this thesis have been published in [23–26]. During the course of this work, several related papers were also published. Papers [33–35] describe details concerning the acceleration of air traffic management systems, one of the SMC applications being studied in Chapter 5.

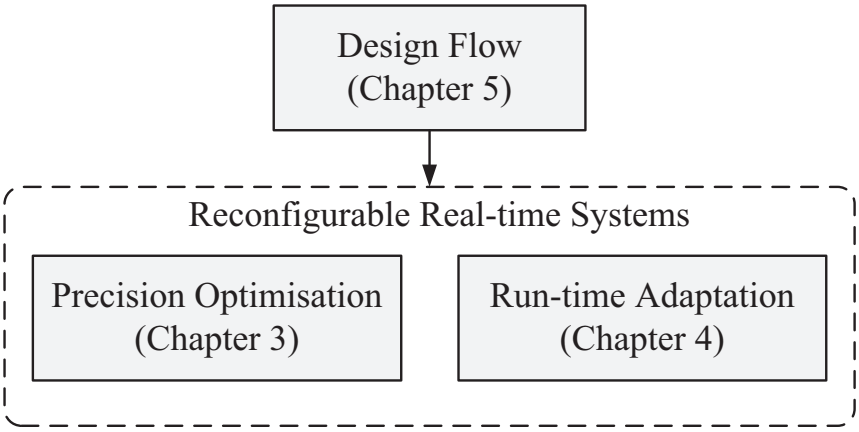


Figure 1.2: Thesis organisation.

Paper [36] presents details of surrogate modelling that enable machine learning approach in Chapter 5. Papers [37, 38] present initial work regarding the adaptive SMC method, which leads to the proposal in Chapter 4. Paper [39] provides a simple benchmarking platform for real-time systems. Due to page limitations, neither of these contributions are described in this thesis.

Chapter 2

Background and Related Work

2.1 Introduction

This chapter begins with a brief overview of reconfigurable systems in Section 2.2. The underlying system architecture and the design flow that maps designs to this system are illustrated. Real-time systems and their typical applications are covered in Section 2.3. Section 2.4 provides a summary.

2.2 Reconfigurable Systems

2.2.1 Architecture

The underlying technology of reconfigurable system is FPGA. In order to perform as functional circuits, FPGAs provide numerous fine-grained resources, namely Look-Up Tables (LUTs) implemented in small Random-Access Memorys (RAMs). LUTs implement combinational logic by storing the corresponding truth table and using the logic inputs as the address into the LUTs. FPGAs enable sequential circuits by providing registers along LUT outputs. By providing hundreds of thousands of LUTs and registers, FPGAs can implement massively parallel

circuits. Modern FPGAs also have coarse-grained resources such as Configurable Logic Blocks (CLBs), multipliers, DSPs, on-chip RAMs, and even microprocessor cores. Microprocessor cores can be dedicated hard processor, such as ARM Cortex A9 in Altera System on a chip (SOC)-FPGA [3] and Xilinx Zynq [4]. In addition, designers can use the LUTs to implement soft processors, such as Altera's Nios II [40] and Xilinx's MicroBlaze [41].

To combine these resources into larger circuits, FPGAs provide reconfigurable interconnect. In between each row and column of resources, FPGAs contain numerous routing tracks, which are wires carrying signals across the chip. Connection boxes provide programmable connections between resources Input/Output (I/O) and routing tracks, while switch boxes provide programmable connections between intersecting routing tracks. Such programmable connections allow a signal to be routed to any destination on the FPGA chip. This architecture is called an island-style fabric as shown in Figure 2.1.

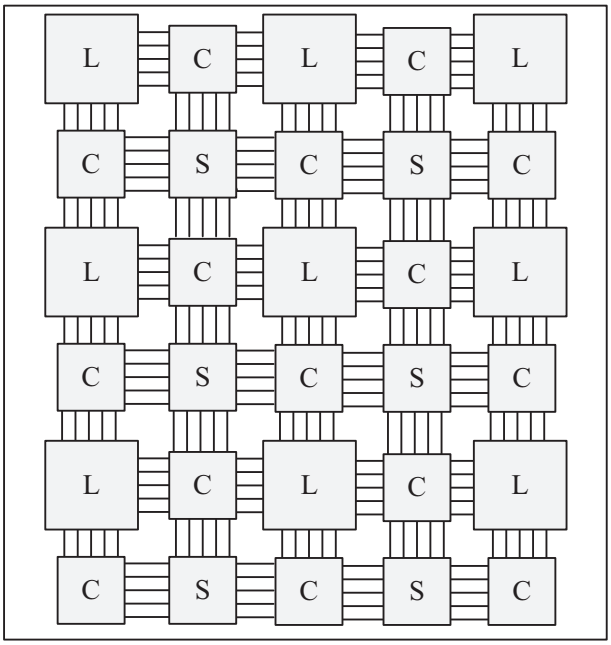


Figure 2.1: Island-style FPGA (L: LUTs and coarse-grained resources; C: Connection boxes; S: Switch boxes).

The reconfigurability of FPGAs leads to an unique feature which allows circuitry to be selectively updated on the fly, without disturbing the execution of the remaining system. This technique is referred to as run-time reconfiguration or dynamic reconfiguration. Many dynam-

ically reconfigurable architectures [42–45] and design approach [46–50] have been proposed. Run-time reconfiguration is also available in industry products such as Altera [51] and Xilinx [52].

FPGAs have also been employed in HPC. There exists a number of reconfigurable architectures which target compute-intensive applications. Convey HC-2 Computer [53] integrates an FPGA-based reconfigurable co-processor with Intel-based x86 host. The co-processor's FPGAs execute compute-intensive operations which take a large component of an application's runtime. The HC-2 system has a memory subsystem and crossbar which provide a highly parallel and high bandwidth (80 GB/s) connections between the FPGAs and the corresponding physical memory. It also employs a scatter-gather dual inline memory modules (SG-DIMMs) to increase performance of random memory access. Meanwhile, Maxeler Technologies develop a series of reconfigurable systems which consist of Intel-based x86 host and FPGA-based dataflow engines [54]. Their MPC-C500 machine can deliver over 400 GFlop/s computation speed and over 35GB/s of bandwidth to external physical memory.

2.2.2 Design Flow

To enable implementation of applications on FPGAs, FPGA tools generally support a design flow as shown in Figure 2.2. Firstly, *synthesis* takes source files written at Register-transfer-level (RTL), typically written in HDL, and converts them to design implementation in terms of logic gates. Secondly, *technology mapping* converts all logic gates into device resources such as LUTs, DSPs and block RAMs. Thirdly, *placement* maps each technology-mapped component onto physical locations of the chip. Finally, *routing* programs the interconnect to implement all connections in the circuit and generates a bit-stream which is downloaded to configure the target FPGA.

There are two major programming models for FPGAs. The most common model manually converts the code into a semantically equivalent RTL circuit, which designers typically specify using HDLs such as VHDL and Verilog. Designing RTL circuits is time consuming. Designers must specify the entire structure of the data path, define control for components, and manage

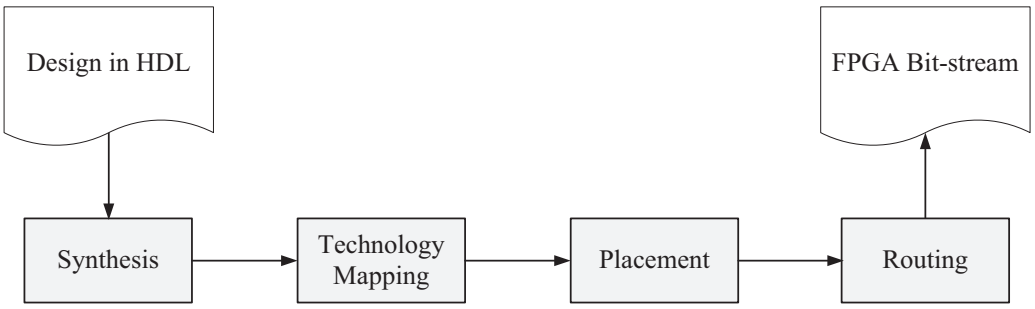


Figure 2.2: Design flow of FPGAs.

data movement from inputs to outputs which involve devices such as DDR memory, PCI Express bus, ethernet, and so on. Such complexity leads researchers to work on High-level Synthesis (HLS) tools. Commercial HLS tools become increasingly common. Example includes Xilinx's Vivado HLS [55], Impulse Accelerated Technologies' Impulse C [56], Calypto's Catapult C [57], Mentor Graphics' Handel-C [58], IBM's Lime [59], Bluespec [60], OpenSPL [61, 62], OpenCL in Altera FPGAs [63], and Mathworks' Simulink (via HDL Coder [64], DSP Builder [65] and System Generator [66]). Open-source tools such as LegUp [67] are also gaining researchers' attention.

As heterogeneous computing is becoming more popular, OpenSPL and OpenCL are two standards which have been proposed to provide a framework for writing programs that execute across FPGAs, CPUs and GPUs. Figure 2.3 illustrates the conceptual design flow of FPGA with OpenSPL and OpenCL. Both OpenSPL [61] and OpenCL include software-like programming language for developing kernel (functions that execute on hardware devices) as well as application programming interfaces that allow kernels communicate with software executable running on micro-processors. OpenSPL, which stands for Open Spatial Programming Language, is a programming language focusing on data-flow computing. Maxeler Technologies' MaxCompiler [62] is a commercial tool which supports OpenSPL and a series of reconfigurable HPC systems. Traditionally, designers targeting different FPGAs must make significant board-specific changes to RTL circuits which are described at low-level. OpenSPL simplifies the effort of customising an FPGA application for a specific model of FPGA by automatically generate interfaces such as buses between CPUs and FPGAs, as well as controller for external memory.

In addition, OpenSPL provides functional level simulation which reduces the debugging effort on RTL-level code simulation. On the other hand, OpenCL, which stands for Open Computing Language, is being promoted by Altera to target software developers who are new to FPGAs. As the requirement of hardware knowledge is low compared to traditional HDL development, OpenCL enables designers focus on high-level algorithm and software system design.

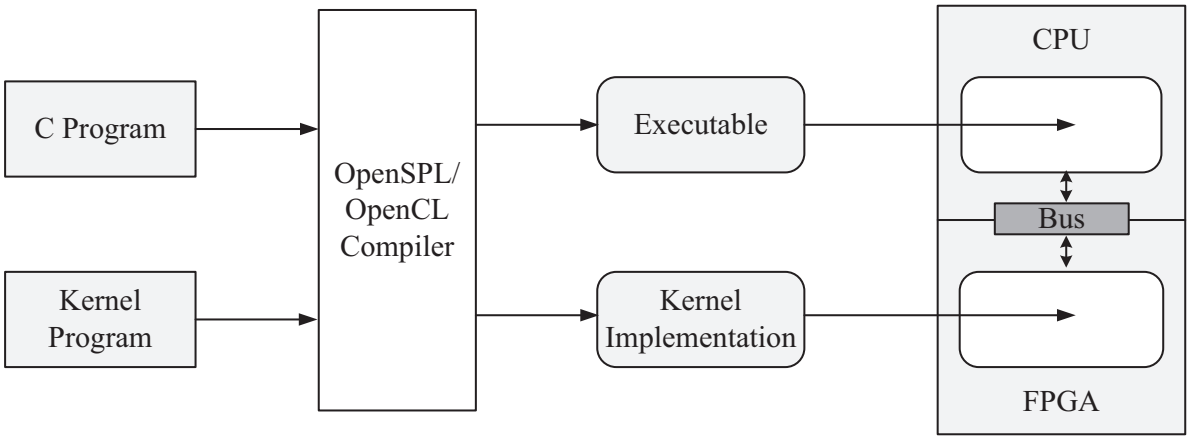


Figure 2.3: Design flow of FPGA with OpenSPL and OpenCL.

MathWork promotes HLS with a model-based design flow [68, 69]. As shown in Figure 2.4, designers first simulate and verify operations in the Simulink development environment, then FPGA Intellectual Property (IP) cores are generated from the Simulink models using HDL Coder, while software executables for ARM processor are compiled using Embedded Coder. The design flow also includes various board-support-packages which generate device-specific interfaces automatically.

Even though HLS tools ease the development effort of building parallelised applications that fully take advantage of FPGA, designers still suffer from long synthesis time which makes design space exploration very inefficient. Traditional software techniques relying on rapid recompilation are no longer feasible. The design space exploration of reconfigurable designs requires substantial effort from users who have to analyse the application, create models and benchmarks, and subsequently use them to evaluate the design. Sometimes such an approach is infeasible as numerical properties cannot result in a closed-form analytical model. One can proceed with automated optimisation based on an exhaustive search through design parameters, yet even au-

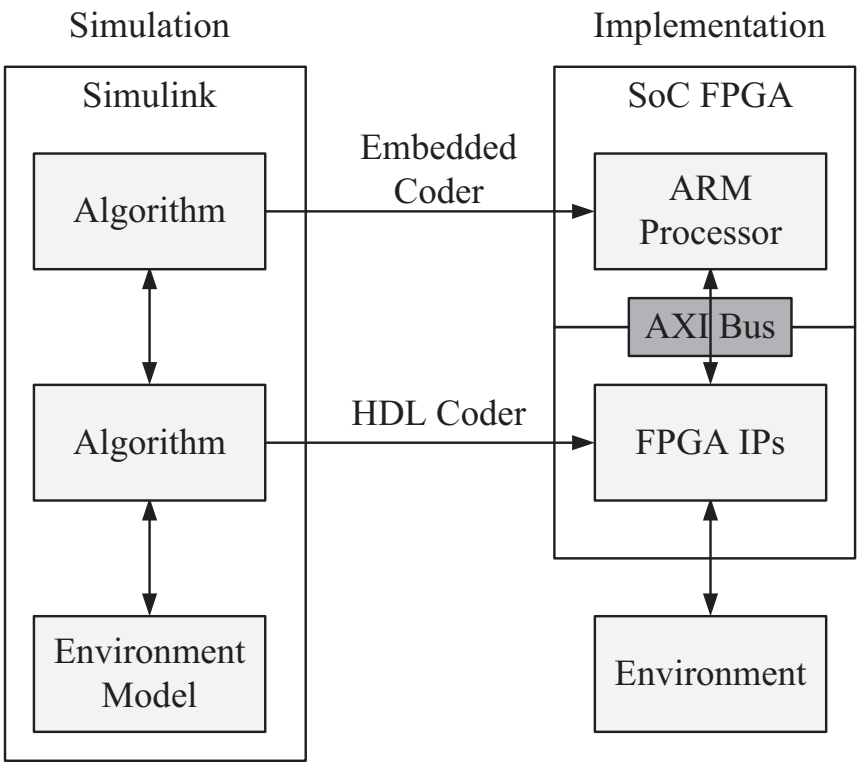


Figure 2.4: MathWorks' model-based design flow.

tomation of design space exploration is problematic because of the large number of evaluations needed. In dealing with large design space, an optimisation approach [70] is developed based on Efficient Global Optimisation (EGO) [71]. It has a surrogate model consisting of both a Gaussian process regressor [72] and a Support Vector Machine (SVM) classifier [73]. By using the surrogate model, the algorithm allows for automated design space exploration. The classification mechanism employed in the optimisation approach allows for constrained optimisation and it is particularly designed to cope with reconfigurable designs parameter tuning. This work is extended in [36] to offer automatic and calibration free optimisation.

2.2.3 Domain Specific Languages

At present, FPGAs are mainly programmed in RTL using Verilog or VHDL. The long development times and requirement for low-level, hardware-centric design expertise have served as a historical barrier for programmers and software engineers. RTL design is error prone and non-portable. Domain Specific Languages (DSLs) or Domain customisable Languages(DCLs) are

being promoted to increase programmer productivity and code quality. DSLs allow application to be described using abstractions that are closer to a problem domain.

GraphGen [74] is a vertex-centric framework that targets FPGA for graph computations. The framework accepts a vertex-centric graph specification and automatically compiles it onto an application-specific synthesised graph processor. The graph processor is customisable by user-defined graph instructions. There is also a special-purpose memory subsystem for graph computations. In the area of packet parsing, G [75] and PP [76] are high-level programming languages which can be compiled to produce high-speed FPGA-based packet parsers.

2.3 Real-time Systems

A system is defined as being real-time if it is required to respond to an input stimulus within a finite and specified time interval. The stimulus could either be an event at the interface to the system or an internal signal. The correctness of a real-time system is based on both the correctness of the outputs and their timeliness. However, the system does not have to be fast. A hard real-time system should guarantee a response to events within a timing bound which is normally referred to as a *deadline*. Missing an operation deadline can lead to catastrophic effects such as a total system failure. Soft real-time system is a loosen form where exact response time is not critical, but missing an operation deadline can cause degraded quality of service.

In summary, real-time systems must have the following properties to support critical applications [77]:

- Timeliness: Output values are produced before the deadlines.
- Robustness: The system should work when subject to a peak load.
- Predictability: The system behaviour is known before it is put into operation.
- Maintainability: The architecture of a real-time system should be designed to ensure that system modifications are easy to perform.

2.3.1 Real-time Applications

This thesis focuses on accelerating high performance real-time applications using reconfigurable systems. Three important applications as described below.

A. Proximity Query for Image-guided Surgery

Advanced surgical robots support image guidance and force-based haptic feedback for effective navigation of surgical instruments. Such image-guided robots rely on real-time computing the intersection or the closest point-pair between two objects in three-dimensional space; this computation is known as PQ.

PQ has been widely studied in areas such as robot motion planning, haptics rendering, virtual prototyping, computer graphics, and animation [78]. Robot motion planning is particularly demanding for the real-time performance of PQ [79]. In the past decade, PQ has also been used as a key task for active constraints [14] and virtual fixtures [80], which are collaborative control strategies mostly applied in image-guided surgical robotics. The clinical potential of this control strategy has been demonstrated by imposing haptic feedback [81] on instrument manipulation based on imaging data [82]. This haptic feedback provides the operator with kinaesthetic perception for sensing positions, velocities, forces, constraints and inertia associated with direct maneuvering of surgical instrument within the target anatomy.

Fast and efficient PQ is a pre-requisite for effective navigation through access routes to the target anatomy [14]. Haptic guidance, rendered based on imaging data, can enable a distinct awareness of the position of the surgical device relative to the target anatomy so as to prevent the operator from feeling disoriented within the surrounding organs. Such disorientation could potentially cause unnoticed major organ damage. Haptic guidance is particularly important during soft tissue surgery, which involves large-scale and rapid tissue deformations. A high update frequency above 1 kHz is required to maintain smooth and steady manipulation guidance. Due to its intrinsic complexity and this real-time requirement, PQ is computationally challenging. Various approaches have been proposed to achieve the required update rate [79, 83],

with objects represented in specific formats such as spheres, torus or convex surfaces. The only attempts that apply PQ to haptic rendering, while considering explicitly the interaction of the body with the surrounding anatomical regions, involve modelling the anatomical pathway or the robotic device as a tubular structure [1, 80]. The computation burden is increased by the need to compute the placement of anatomical model relative to the robot whose shape is represented by more than 1 million points.

Fig. 2.5 illustrates two objects acting as inputs to PQ. The object shown on the left is bounded by a series of contours C_j $\forall j \in [1, ..., N_C]$, each of which is outlined by a set of contour points. This object can be either a luminal anatomy or a robotic endoscope/catheter. On the right, the mesh comprises points which represent the morphological structure of either the robot or the target anatomy in complex shape. Essentially PQ computes δ_j, which describes how much the mesh deviates beyond the volumetric pathway bounded along the contours.

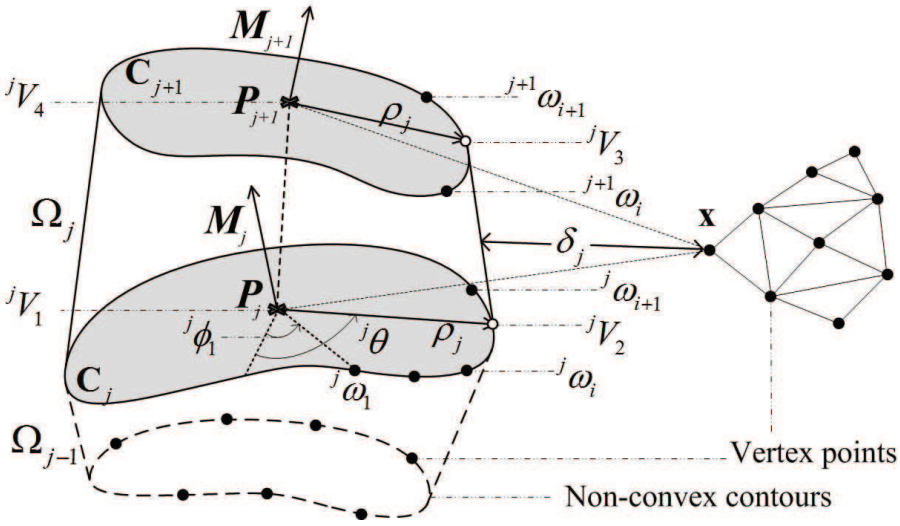
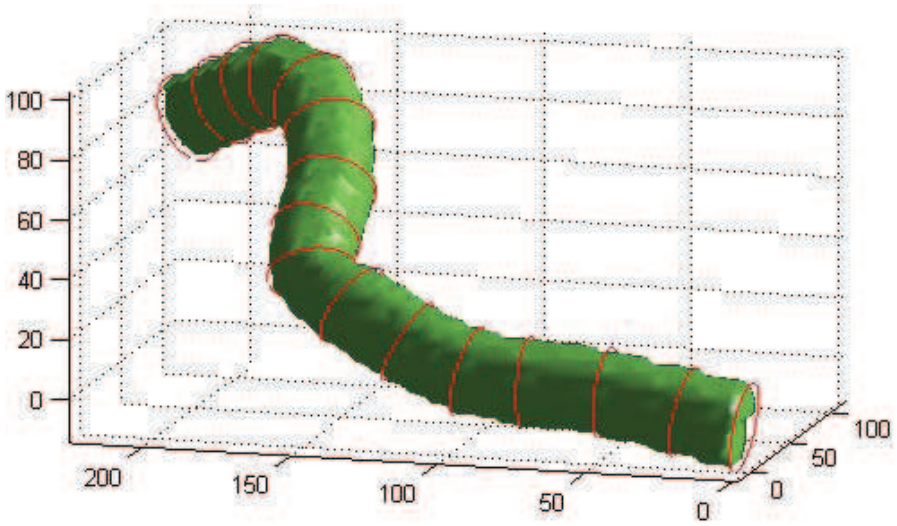
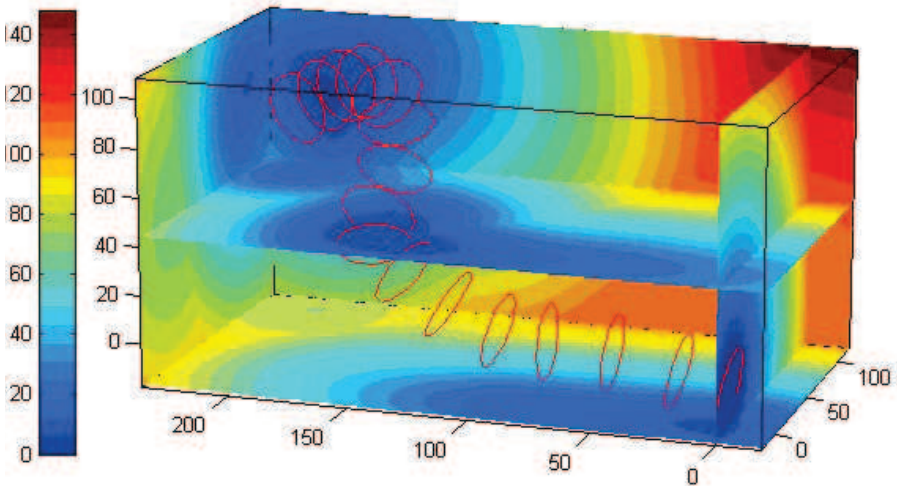


Figure 2.5: (Left) Various sets of points aligned on a series of contours; (Right) A set of points located on an arbitrary form of mesh.

As shown in Fig. 2.6(a), a series of circular contours fitted along a part of an endoscope, which passes through the rectum up to the sigmoid colon. These contours form a constraint pathway. Fig. 2.6(b) shows a distance map in three-dimensional space with 177k grid points. Distance from every grid point to the endoscope is computed by PQ. The warmer colour, the further the point is located beyond the endoscope. Each contour is denoted by C_j, $\forall j \in [1, ..., N_C]$. A single segment Ω_j comprises two adjacent contours C_j and C_{j+1}. P_j is the centre of the contour



(a)



(b)

Figure 2.6: (a) A virtual tube (in green) bounded by a series of contour (in red) denotes the configuration of an endoscope; (b) The corresponding three-dimensional distance map in grids of 86x48x43.

C_j. M_j is the tangent of centre line of contour C_j. ${}^j\omega_i = [{}^j\omega_{xi}, {}^j\omega_{yi}, {}^j\omega_{zi}]^T, (i = 1, ..., W)$ are the contour points, where W is the number of points outlining each contour.

There has been previous work on hardware acceleration of board-phase PQ, which involves detecting collisions between primitive objects, e.g. spheres [83] or boxes [84]. Such an object can be a bounding volume tightly containing a union of multiple complex-shaped objects. On FPGAs, the most relevant work is covered by Chow el at. [5]. On the other hand, narrow-phase PQ, which computes the shortest distance or penetration depth between polyhedra, such as GJK [85], V-Clip [86] and Lin-Canny [87], are difficult to be accelerated by hardware due

to algorithmic complexity. There is, thus far, no attempt of using FPGA. In addition, such approaches are restricted to the object represented in convex polyhedra. To this end, a PQ approach for complex-morphology object [1] is proposed but how it can be incorporated with FPGA is not elaborated.

To leverage the advantages of FPGAs for hardware acceleration, Chow et al. [5] proposed a mixed precision methodology. The work made an assumption that the data-path is short so that both the reduced precision and high precision implementations can fit in a single FPGA. They also assume that the communication between the reduced-precision data-path and the high-precision data-path is completed via a crossbar, thus the overhead is negligible. On the other hand, they are not applicable for complicated applications when the level of parallelism is limited.

Other researchers focus on studying bit-width optimisation which uses minimum precision in a data-path given a required output accuracy. Examples include interval arithmetic [88], affine arithmetic [89, 90] and polynomial algebraic approach [91]. However, a reduction of precision in any stage within a data-path will result in a loss in output accuracy which is uncorrectable. These studies require the use of accuracy models to relate output accuracy with the precisions of data-path.

B. SMC Methods for Finance, Robotic and Control

SMC methods, also known as Particle Filter (PF), are a set of a posterior density estimation algorithms that perform inference of unknown quantities of interest from observations [92]. The observations arrive sequentially in time and the inference is performed on-line. SMC methods are often preferable to Kalman filters and hidden Markov models, as they do not require exact analytical expressions to compute the evolving sequence of posterior distributions. SMC methods work well for dynamic systems involving non-linear and non-Gaussian properties, and they can model high-dimensional data using non-linear dynamics and constraints, are parallelisable, and can greatly benefit from hardware acceleration. SMC has been studied in various application areas including object tracking [93], robot localisation [94], speech recognition [95] and air

traffic management [11, 96]. For these applications, it is critical that high sampling rates can be handled in real-time. SMC methods also have applications in economics and finance [97] where minimising latency is crucial.

SMC keeps track of a large number of particles, each contains information about how a system would evolve. The underlying concept is to approximate a sequence of states by a collection of particles. Each particle is weighted to reflect the quality of an approximation. The more complex the problem, the larger the number of particles that are needed. One drawback of SMC is its long execution times so its practical use is limited.

In SMC, the target posterior density $p(s_t|m_t)$ is represented by a set of particles, where s_t is the state and m_t is the observation at time step t. A sequential importance resampling (SIR) algorithm [98] is used to obtain a weighted set of N_P particles $\{s_t^{(i)}, w^{(i)}\}_{i=1}^{N_P}$. The importance weights $\{w^{(i)}\}_{i=1}^{N_P}$ are approximations to the relative posterior probabilities of the particles such that $\sum_{i=1}^{N_P} w_t^{(i)} = 1$. This process is described in Algorithm 1. A more detailed description can be found in [92].

Algorithm 1 SMC methods.

```
for each time step t do
    idx1 ← 0
    Initialisation
    while idx1 ≤ itl_outer do
        idx2 ← 0
        itl_inner ← f(idx1)
        for each particle p do
            while idx2 ≤ itl_inner do
                Sampling
                Importance weighting
                idx2 ← idx2 + 1
            end while
        end for
        idx1 ← idx1 + 1
        if idx1 ≤ itl_inner then
            Resampling
        end if
    end while
    Update
end for
```

1. **Initialisation**: Weights $\{w^{(i)}\}_{i=1}^{N_P}$ are set to the same value, e.g. $\frac{1}{N_P}$.

Table 2.1: SMC design parameters. Dynamic: adjustable at run-time; Static: fixed at compile-time.

Parameters	Description	Type
itl_outer	Number of iterations of the outer loop	Dynamic
itl_inner	Number of iterations of the inner loop	
N_P	Number of particles	
S	Scaling factor for standard deviation of noise	
H	Prediction horizon	Static
N_A	Number of agents under control	

2. **Sampling**: Next states $\{s_{t+1}^{'(i)}\}_{i=1}^{N_P}$ are computed based on the current state $\{s_t^{(i)}\}_{i=1}^{N_P}$.

3. **Importance weighting**: Weight $\{w^{(i)}\}_{i=1}^{N_P}$ is updated based on a score function which accounts for the likelihood of particles fitting the observation. Within each iteration of *itl_outer*, the sampling and importance weighting stages are iterated *itl_inner* times so that those particles with sustained benefits are assigned higher weights. *itl_inner* increases as a function of $idx1$, because a larger $idx1$ implies that the set of particles reflects a more accurate approximation.

4. **Resampling**: Particles with small weights are removed and those with large weights are replicated. This process is repeated for *itl_outer* times in a time step to address the problem of degeneracy [99]. Without resampling, only a small number of particles will have substantial weights for inference.

5. **Update**: State s_{t+1} is obtained from the resampled particle set $\{s_{t+1}^{(i)}\}_{i=1}^{N_P}$ via weighted average or more complicated functions that will be shown below.

Table 2.1 summarises the parameters of the SMC methods described in Section 2.3.1.

Adaptive SMC methods have been proposed to improve performance or quality of state estimation by controlling the number of particles dynamically. Likelihood-based adaptation controls the number of particles such that the sum of weights exceeds a pre-specified threshold [100]. Kullback Leibler distance (KLD) sampling is proposed in [101], which offers better quality results than likelihood-based approach. KLD sampling is improved in [102] by adjusting the variance and gradient of data to generate particles near high likelihood regions. The above methods introduce data dependencies in the sampling and importance weighting steps, so they

are difficult to be parallelised. An adaptive SMC is proposed in [103] that changes the number of particles dynamically based on estimation quality. In [37], adaptive SMC is extended to a multi-processor system on FPGA. The number of particles and active processors change dynamically but the performance is limited by soft-core processors. In [104], both a mechanism and a theoretical lower bound for adapting the sample size of particles are presented.

Acceleration of SMC methods has been studied in applications such as finance, robotics and control. Applications related specifically to each of the contribution of this thesis are described below.

Robot Localisation

SMC methods are applied to mobile robot localisation [26, 94], and this application is used as an example throughout the paper. At regular time intervals, a robot obtains sensor values, identifies its location and commits a move. The robot needs to be aware of the locations of other moving objects in the environment.

The sampling stage is described by Equations 2.1 and 2.2:

$$\left(s_t^i\right) = \begin{pmatrix} x_t^i \\ y_t^i \\ h_t^i \end{pmatrix} = \begin{pmatrix} x_{t-1}^i + \delta_t'^i \cos(h_{t-1}^i) \\ y_{t-1}^i + \delta_t'^i \sin(h_{t-1}^i) \\ h_{t-1}^i + \gamma_t'^i \end{pmatrix}, \tag{2.1}$$

$$\left(r_t^i\right) = \begin{pmatrix} \delta_t'^i \\ \gamma_t'^i \end{pmatrix} = \begin{pmatrix} \mathcal{N}(\delta_t, \sigma_a^2) \\ \mathcal{N}(\gamma_t, \sigma_b^2) \end{pmatrix}, \tag{2.2}$$

where the robot estimates its updated state s_t based on the current known location (x, y),heading h, and external reference status r_t which contains displacement δ and rotation γ.

Both δ and γ are subject to Gaussian noises which are modelled as $\mathcal{N}(\delta_t, \sigma_a^2)$ and $\mathcal{N}(\gamma_t, \sigma_b^2)$ respectively. Importance weighting is used to calculate the likelihood of a location based on the observation, i.e. the sensor values.

Air Traffic Management

Air traffic management is crucial to air transport industry. An air traffic management system coordinates the movement of aircraft, and ensures safety by maintaining safe separation distances between aircraft during take-off, landing and cruising. These objectives have to be carried effectively that ensures air traffic flows smoothly with minimal expenses in terms of delay, fuel and administration costs. To cope with the growing demand in future air traffic, the capacity of the airspace has to be increased without compromising safety. However, the architecture of current air traffic management system relies on human-operated air traffic control services, which are rigid and saturated, impose a constraint in the growth of air traffic. Development of air traffic management aims to provide more accurate predictive information about aircraft trajectories. The uncertainty of aircraft trajectories can force air traffic control to use larger separations between aircraft to ensure safety, thus reducing the total number of aircraft that an airspace can accommodate, increasing the fuel consumption and time of arrival of aircraft.

SMC methods have been applied to air traffic management [33–35,96,105,106]. At discrete time intervals, control actions are determined by SMC and applied to adjust aircraft trajectories. Model predictive control is applied to optimise the air traffic management problem over a finite time horizon, which allows anticipating future events. Figure 2.7 provides an overview of the air traffic control problem depicted as a closed loop control system.

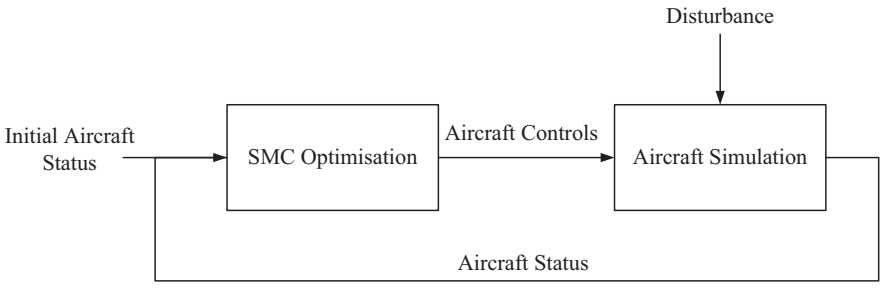
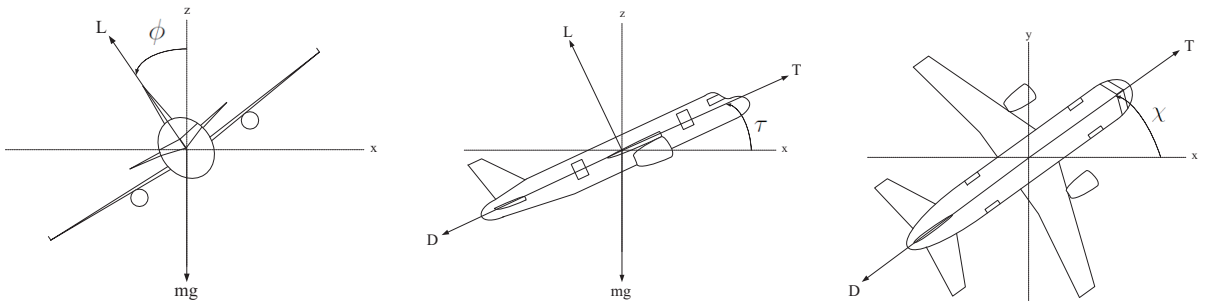


Figure 2.7: An overview of the air traffic control problem.

Figure 2.8 depicts the model that simulates the dynamic of an aircraft. The major variables include the aircraft position in 3 dimensional space (x, y, a), true air speed V, aircraft mass m,



(a) (b) (c)

Figure 2.8: Aircraft model.

heading angle χ, roll angle ϕ and pitch angle τ. The forces applied to the aircraft are its weight mg, the engine thrust T, and the aerodynamic forces of lift L and drag D. As illustrated in Equation 2.3, ϕ, τ and T are control variables summerised as a state which is optimised by SMC. The state s_t^i determines the movement of aircraft, which is affected by disturbances from varying wind and atmospheric conditions. Then s_t^i adjust the status of aircraft $r_t^{'i}$, which are the position, heading, speed and mass of the aircraft as described in Equation 2.4.

Table 2.2 summerises the variables used in air traffic management.

$$\left(s_t^i\right) = \begin{pmatrix} \phi_t^{'i} \\ \tau_t^{'i} \\ T_t^{'i} \end{pmatrix} = \begin{pmatrix} \mathcal{N}(\phi_t, \sigma_a^2) \\ \mathcal{N}(\tau_t, \sigma_b^2) \\ \mathcal{N}(T_t, \sigma_c^2) \end{pmatrix}, \tag{2.3}$$

$$\left(r_t^{'i}\right) = \begin{pmatrix} x_t^i \\ y_t^i \\ a_t^i \\ \chi_t^i \\ V_t^i \\ m_t^i \end{pmatrix} = \begin{pmatrix} x_{t-1} + V_{t-1}\cos(\chi_{t-1})\cos(\tau_t^{'i}) \\ y_{t-1} + V_{t-1}\sin(\chi_{t-1})\cos(\tau_t^{'i}) \\ a_{t-1} + V_{t-1}\sin(\tau_t^{'i}) \\ \chi_{t-1} + L\sin(\phi_t^{'i})/(M_{t-1}V_{t-1}) \\ V_{t-1} + (\frac{T_t^{'i} - D}{M_{t-1}} - g\sin(\tau_t^{'i})) \\ m_{t-1} - \eta T_t^{'i} \end{pmatrix}, \tag{2.4}$$

where $V_t^i \subset [V_{min}, V_{max}]$, $m_t^i \subset [m_{min}, m_{max}]$, $T_t^i \subset [T_{min}, T_{max}]$, $\phi_t^i \subset [\phi_{min}, \phi_{max}]$, $\tau_t^i \subset [\tau_{min}, \tau_{max}]$ are constraints.

Table 2.2: Variables in air traffic management model.

Variables	Description
(x, y, a)	Aircraft position in 3 dimensional space
V	True air speed
m	Aircraft mass
mg	Aircraft weight
χ	Heading angle
ϕ	Roll angle
τ	Pitch angle
T	Engine thrust
L	lift
D	drag

2.4 Summary

This chapter first reviewed the architecture and design flow of reconfigurable systems. Reconfigurable systems provide high flexibility and performance by customising numerous fine-grained resources and implementing massively parallel circuits. The primary challenge is that the traditional RTL design flow requires long synthesis time and substantial effort from user, therefore, latest development in HLS and domain-specific languages are discussed to provide a view on how academia and industry work on reducing the development effort of reconfigurable systems.

This thesis focuses on optimising reconfigurable systems to accelerate the above mentioned high-performance, real-time applications. The later part of this chapter reviewed real-time systems and several related applications, which specifically require high-performance. In this thesis, these applications are accelerated by various reconfigurable technologies, which relate to the contributions presented in Chapter 1.

Chapter 3

Precision Optimisation of Data-paths

3.1 Introduction

This chapter presents a precision optimisation approach to maximise real-time performance of reconfigurable systems. The proposed approach is applied to image-guidance of medical surgery robot.

PQ is an important compute-intensive and real-time application which requires substantial acceleration before it can be used in clinical setting. It is because fast and efficient PQ (update frequency above 1 kHz) is required to maintain smooth and steady manipulation guidance which is particularly essential for soft tissue surgery having large-scale and rapid tissue deformations. This real-time requirement, as well as the intrinsic complexity of the algorithm, make PQ computationally challenging as a high update frequency above 1 kHz is required. The computation burden is increased by the need to model the shape of tissue and surgery robot by more than 1 million points.

Due to its compute-intensive nature, PQ can greatly benefit from hardware acceleration. However, the massive amount of floating-point computations constitute a long data-path which is resource-demanding. Even if we could implement the data-path in an FPGA, the acceleration would be restricted by low parallelism and clock frequency. This challenge limits the

implementation of PQ on an FPGA.

In this chapter, we derive a PQ formulation which allows objects to be represented in complex geometry with points. To leverage the advantages of FPGAs, function transformation eliminates iterative trigonometric functions such that the algorithm can be fully-pipelined. We increase data-path parallelism by adopting a reduced precision data format which consumes fewer logic resources than high precision. To maintain the accuracy of results, potential incorrect outputs are re-computed in high precision. We design a novel memory architecture for buffering potential outputs and maintaining streaming data-flow. We further exploit the run-time reconfigurability of FPGA to optimise precision dynamically. To the best of our knowledge, our work is the first to apply reconfigurable technology to narrow-phase PQ computation.

The contributions of this chapter are as follows.

- A hardware-friendly PQ formulation for calculating the relative placement of objects modelled by points with complex morphology, which facilitates restructuring of trigonometric and search-functions to be amenable to parallel implementation in hardware.
- Enhanced parallelism by treating input points as a novel data structure propagating through pipelines, together with FPGA-specific optimisations such as adapting PQ to reduced precision arithmetic, supporting multiple precisions in a novel memory architecture, and automating precision management with run-time reconfiguration.
- Implementation in a reconfigurable platform with four FPGAs which is shown to be 478 times faster than a single-core CPU, 58 times faster than a 12-core CPU system, 9 times faster than a GPU, and 3 times faster than a 4-FPGA system implemented in double precision.

The rest of the chapter is organised as follows. Section 3.2 presents our proposed PQ formulation. Section 3.3 discusses the optimisation of PQ for reconfigurable system. Section 3.4 describes the system design that maps PQ to a reconfigurable system. Section 3.5 provides experimental results and Section 3.6 concludes our work.

3.2 Formulation of PQ

In this section, we derive our modified PQ process which was originally proposed in [1]. The significance of this modification is to formulate the PQ capable of processing the contours in complex shapes. As shown in Figure 3.1, PQ allows the analytical measure of the shortest Euclidean distance between an arbitrary set of points and a series of segments Ω_j (cf. Definition 1) which has been a well-known representation of a complex three-dimensional object [107]. Each segment Ω_j is enclosed by two adjacent contours which are outlined by points arranged in polar coordinates; hence, it outperforms the existing narrow-phase PQs which are only compatible with convex objects.

Definition 1. *Each contour is denoted by* C_j, $\forall j \in [1, ..., N_C]$. *A single segment* Ω_j *comprises two adjacent contours* C_j *and* C_{j+1}. P_j *is the centre of the contour* C_j. M_j *is the tangent of centre line of contour* C_j. ${}^j\omega_i = [{}^j\omega_{xi}, {}^j\omega_{yi}, {}^j\omega_{zi}]^T, (i = 1, ..., W)$ *are the contour points, where* W *is the number of points outlining each contour.*

Four steps are taken to calculate the point-to-segment distance δ_j, which is shown in Figure 3.1 as the shortest distance δ_j between a point $\boldsymbol{x}$ and the corresponding edge ${}^jV_2 \rightarrow {}^j V_3$. Before introducing these steps, we describe the computation using polar coordinates. Given a contour C_j, ${}^j\phi_i$ is the polar angle corresponding to contour points ${}^j\omega_i$. The polar angles of all the ${}^j\omega_i$ along the contour, i.e. ${}^j\omega_1, ..., {}^j\omega_W$, have to be computed. This computation can be further simplified by ignoring an axis coordinate. The poles and the contour points are then projected either on X-Y, Y-Z or X-Z plane based on the following conditions:

$$
\begin{aligned}
&\text{if } |M_{zj}| = \max\left(|M_{xj}|, |M_{yj}|, |M_{zj}|\right), \\
&\quad {}^j\omega_i' = [{}^j\omega_{1i}, {}^j\omega_{2i}]^T = [{}^j\omega_{xi}, {}^j\omega_{yi}]^T, P_j' = [P_{xj}, P_{yj}]^T, \\
&\text{if } |M_{xj}| = \max\left(|M_{xj}|, |M_{yj}|, |M_{zj}|\right), \\
&\quad {}^j\omega_i' = [{}^j\omega_{1i}, {}^j\omega_{2i}]^T = [{}^j\omega_{yi}, {}^j\omega_{zi}]^T, P_j' = [P_{yj}, P_{zj}]^T, \\
&\text{if } |M_{yj}| = \max\left(|M_{xj}|, |M_{yj}|, |M_{zj}|\right), \\
&\quad {}^j\omega_i' = [{}^j\omega_{1i}, {}^j\omega_{2i}]^T = [{}^j\omega_{zi}, {}^j\omega_{xi}]^T, P_j' = [P_{zj}, P_{xj}]^T,
\end{aligned}
\tag{3.1}
$$

where $M_j = (M_{xj}, M_{yj}, M_{zj})$ is the tangent of centre line of contour C_j.

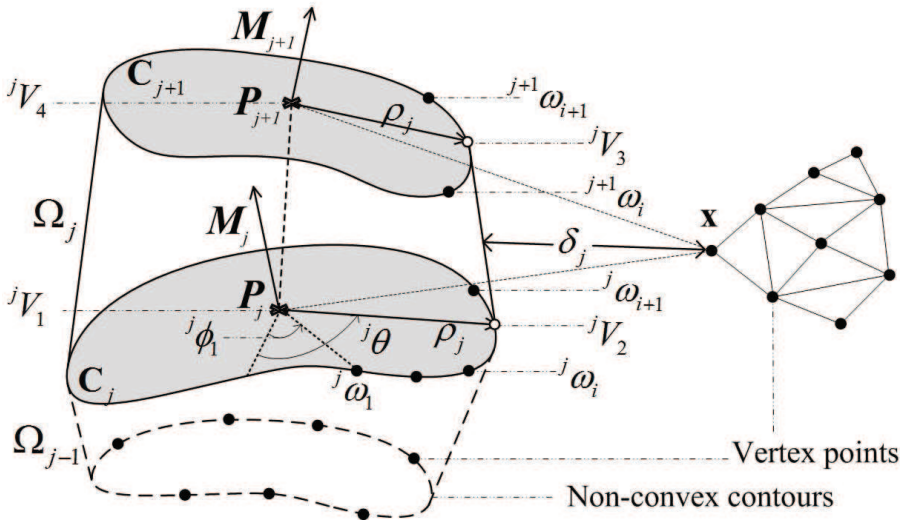


Figure 3.1: (Left) Various sets of points aligned on a series of contours; (Right) A set of points located on an arbitrary form of mesh.

Then $^{j}\phi_i$ is calculated as follows:

$$\overline{^{j}\omega_i'} = {}^{j}\omega_i' - P_j' \,, \qquad {}^{j}\phi_i = atan2\left(\overline{^{j}\omega_{2i}}, \overline{^{j}\omega_{1i}}\right). \tag{3.2}$$

We will explain the details of $atan2$ is Section 3.3.1.

Step 1: Find the normal of a plane containing three points: $\boldsymbol{x}$, P_j and P_{j+1}. The symbol $\times$ denotes a cross product of two vectors in three-dimensional space. The normal n_j is calculated by:

$$n_j = (P_j - x) \times (P_{j+1} - x)\,. \tag{3.3}$$

Step 2: Calculate vectors ρ_j and ρ_{j+1} which are respectively perpendicular to tangents M_j and M_{j+1} and are both parallel to the plane with normal n_j.

$$\rho_j = n_j \times M_j \,, \qquad \rho_{j+1} = n_j \times M_{j+1}. \tag{3.4}$$

Step 3: Determine a 4-vertex polygon outlined by ${}^{j}V_{i=1...4} \in \Re^{3\times 1}$ which is a part of the cross-section of segment Ω_j. This section is cut by a plane containing the point $\boldsymbol{x}$ and the line segment $P_j \rightarrow P_{j+1}$.

$$\begin{aligned} {}^{j}V_1 &= P_j, \qquad & {}^{j}V_2 &= P_j + t_j \cdot \rho_j, \\ {}^{j}V_4 &= P_{j+1}, \qquad & {}^{j}V_3 &= P_{j+1} + t_{j+1} \cdot \rho_{j+1}. \end{aligned} \tag{3.5}$$

At this stage, we need to calculate t_j and t_{j+1} of Equation 3.5. This can be achieved by mapping the values of ρ_j to a two-dimensional plane. The two-dimensional mapping of ρ_j is ρ'_j.

$$\begin{aligned} &\text{if } |M_{zj}| = \max\left(|M_{xj}|, |M_{yj}|, |M_{zj}|\right) \\ &\quad \rho'_j = [\rho_{1j}, \rho_{2j}]^T = [\rho_{xj}, \rho_{yj}]^T, \\ &\text{if } |M_{xj}| = \max\left(|M_{xj}|, |M_{yj}|, |M_{zj}|\right) \\ &\quad \rho'_j = [\rho_{1j}, \rho_{2j}]^T = [\rho_{yj}, \rho_{zj}]^T, \\ &\text{if } |M_{yj}| = \max\left(|M_{xj}|, |M_{yj}|, |M_{zj}|\right) \\ &\quad \rho'_j = [\rho_{1j}, \rho_{2j}]^T = [\rho_{zj}, \rho_{xj}]^T. \end{aligned} \tag{3.6}$$

Then we calculate ${}^{j}\theta$, the corresponding polar angle of ρ'_j by Equation 3.7.

$${}^{j}\theta = atan2\left(\rho_{2j}, \rho_{1j}\right). \tag{3.7}$$

A search is performed to find ${}^{j}\phi_i$ and ${}^{j}\phi_{i+1}$ such that ${}^{j}\phi_i \leq {}^{j}\theta \leq {}^{j}\phi_{i+1}$. The polar angles ${}^{j}\phi_i$ and ${}^{j}\phi_{i+1}$ are calculated from Equation 3.2.

Based on the value i obtained from the search, t_j, which is used in Equation 3.5, is calculated.

$$
\begin{aligned}
a &= [(P_j - {}^j\omega_i)({}^j\omega_{i+1} - {}^j\omega_i)][({}^j\omega_{i+1} - {}^j\omega_i)\rho], \\
b &= [(P_j - {}^j\omega_i)\rho]\|{}^j\omega_{i+1} - {}^j\omega_i\|^2, \\
c &= \|\rho\|^2\|{}^j\omega_{i+1} - {}^j\omega_i\|^2 - \|({}^j\omega_{i+1} - {}^j\omega_i)\rho\|^2, \\
t_j &= \frac{a-b}{c}.
\end{aligned} \tag{3.8}
$$

Step 4: Define the shortest distance to be zero if the point $\boldsymbol{x}$ lies inside the polygon ${}^jV_{i=1...4}$ on the same plane. Referring to [108], it can be determined by three variables $\lambda_{i=1,...,3}$ calculated as follows:

$$
\begin{aligned}
\lambda_i &= n_j \cdot \psi_i, i = 1, ..., 3 \\
\text{s.t. } \psi_i &= ({}^jV_i - \mathbf{x}) \times ({}^jV_{i+1} - \mathbf{x}).
\end{aligned} \tag{3.9}
$$

Here n_j denotes the normal defined in Equation 3.3 and ψ_i denotes the normal of the plane containing ${}^jV_{i=1...4}$. For all $\lambda_{i=1,...,3} \geq 0$, the shortest distance δ_j from point $\boldsymbol{x}$ to the segment Ω_j is assigned to zero such that $\delta_j(\boldsymbol{x}) = 0$. Otherwise $\delta_j(\boldsymbol{x})$ will be considered as the distance from the point $\boldsymbol{x}$ to the line segment ${}^jV_2 \rightarrow {}^j V_3$. Referring to [109], such a point-segment distance in three-dimensional space can be calculated as shown in Equation 3.10:

$$
\begin{aligned}
{}^j\mu &= \frac{({}^jV_2 - X) \cdot ({}^jV_3 - {}^j V_2)}{||{}^jV_3 - {}^j V_2||^2}, \\
\chi_j &= (1 - {}^j\mu){}^jV_2 + {}^j\mu{}^jV_3, \\
\delta_j(\mathbf{x}) &= ||\mathbf{x} - \chi_j||.
\end{aligned} \tag{3.10}
$$

In consideration of many points and segments, Equation 3.11 generally expresses the deviation in distance from a single coordinate x_i to a series of constraint segments $(\Omega_1, ..., \Omega_{N_C-1})$, where $i = 1, ..., N_P$, N_P is the total number of points belong to the mesh model, $N_C - 1$ is the number of segments involved in the calculation.

$$
{}_i\delta_{N_C-1} = \min\left(\delta_1(\boldsymbol{x}_i), \delta_2(\boldsymbol{x}_i), ..., \delta_{N_C-1}(\boldsymbol{x}_i)\right). \tag{3.11}
$$

The point with the maximum deviation, also known as penetration depth, is obtained below:

$$d^{N_C-1} = \max_{i=1,\ldots,N_P} \left({}_i\delta_{N_C-1}(\boldsymbol{x}_i)\right). \tag{3.12}$$

3.3 Optimisation for Reconfigurable Hardware

The PQ formulation sketched in the previous section is not entirely hardware-friendly. In this section we discuss several techniques allowing PQ to benefit from FPGA technology.

3.3.1 Transformation of Trigonometric and Search Functions

The search process in step 3 of PQ (described in Section 3.2) checks whether ${}^j\phi_i \leq {}^j\theta$.

$$\begin{aligned} {}^j\phi_i &= atan2\left(\overline{{}^j\omega_{2i}}, \overline{{}^j\omega_{1i}}\right), \\ {}^j\theta &= atan2\left(\rho_{2j}, \rho_{1j}\right). \end{aligned} \tag{3.13}$$

$atan2(a,b)$ is not a hardware-friendly operator. It requires the calculation of $tan^{-1}(a,b)$ and then determines the appropriate quadrant of the computed angle based on the signs of a and b. $tan^{-1}(a,b)$ is resource and timing expensive [110] and not available in many FPGA libraries, therefore, we transform Equation 3.13 to another form as shown below:

$$\begin{aligned} {}^j\phi_i &= tan^{-1}\left(\frac{\overline{{}^j\omega_{2i}}}{\sqrt{\overline{{}^j\omega_{1i}}^2 + \overline{{}^j\omega_{2i}}^2} + \overline{{}^j\omega_{1i}}}\right), \\ {}^j\theta &= tan^{-1}\left(\frac{\rho_{2j}}{\sqrt{\rho_{1j}^2 + \rho_{2j}^2} + \rho_{1j}}\right). \end{aligned} \tag{3.14}$$

$atan2$ is transformed to tan^{-1} which is then cancelled out on both sides. As a result, the comparison becomes:

$$\frac{\overline{{}^{j}\omega_{2i}}}{\sqrt{\overline{{}^{j}\omega_{1i}}^2 + \overline{{}^{j}\omega_{2i}}^2} + \overline{{}^{j}\omega_{1i}}} \leq \frac{\rho_{2j}}{\sqrt{\rho_{1j}^2 + \rho_{2j}^2} + \rho_{1j}}. \tag{3.15}$$

In this case, square root calculation is much easier to be mapped to hardware.

3.3.2 Applying Reduced Precision

Reduced precision data-paths consume less logic resource at the expense of lower accuracy of results. To benefit from reduced precision data-paths without compromising accuracy, we partition the computation of PQ into two data-paths:

- Reduced precision data-path: Compute the deviations based on Equation 3.3 to 3.11.
- High precision data-path: Re-compute those deviations which are not accurate enough and calculate the penetration depth according to Equation 3.12.

In Equation 3.11, there are $N_C - 2$ comparisons involved to find the minimum value. The only item of interest is the minimum value ${}_i\delta_{N_C-1}$, rather than the exact values of every $\delta_j(\boldsymbol{x}_i)$. Based on this insight, we define the comparison operation:

$$\begin{aligned} {}_i\delta^{min}_{1,\ldots,j} &= \min\left(\delta_1(\boldsymbol{x}_i), ..., \delta_j(\boldsymbol{x}_i)\right), \\ D &= {}_i\delta^{min}_{1,\ldots,j} - \delta_{j+1}(\boldsymbol{x}_i). \end{aligned} \tag{3.16}$$

When computed in reduced and high precision, the values of D are denoted as D_{p_L} and D_{p_H}, respectively. D_{p_L} might have a flipped sign compared with D_{p_H}. We use the following three steps to make sure the results of Equation 3.11 is correct.

1. Evaluate Equation 3.16 using a reduced precision data format.
2. Estimate the maximum and minimum values of the value in high precision, i.e. $min(D_{p_H})$ and $max(D_{p_H})$, as shown in Equation 3.17:

$$
\begin{aligned}
E_{p_L}(D_{p_L}) &= E_{p_L}({}_i\delta^{min}_{1,\ldots,j}) + E_{p_L}(\delta_{j+1}(\boldsymbol{x}_i)),\\
\min(D_{p_H}) &= D_{p_L} - E_{p_L}(D_{p_L}),\\
\max(D_{p_H}) &= D_{p_L} + E_{p_L}(D_{p_L}),
\end{aligned}
\tag{3.17}
$$

where $E_{p_L}(y)$ is the absolute error of y in reduced precision p_L.

$E_{p_L}(\delta_{j+1}(\boldsymbol{x}_i))$ is computed at run-time and the details will be discussed later in Section 3.3.3.

3. Determine whether the comparison result should be re-computed or dropped.

   Case A: $\min(D_{p_H}) > 0$, $\delta_{j+1}(\boldsymbol{x}_i)$ is smaller. No re-computation is necessary.

   Case B: $\max(D_{p_H}) < 0$, ${}_i\delta^{min}_{1,\ldots,j}$ is smaller. No re-computation is necessary.

   Case C: Cannot determine which value is smaller. Store both values for re-computation using high precision p_H.

In case A and B, the difference between the values is large enough to distinguish the sign of D_{p_H} even in the presence of errors introduced by reduced precision computations. In case C, the difference is small compared with the uncertainty introduced by reduced precision, and therefore re-computation in high precision is necessary. The frequency of case C is lower than case A and B, therefore the gain in computation speed from using reduced precision outweighs the re-computation overhead.

3.3.3 Finding the Right Precision

We optimise the error bound $E_{p_L}(D_{p_L})$ based on feedback from run-time environment. Although the error bound can be derived statically [89], the estimated error bound grows pessimistically as it propagates along the data-path. Thus, we calculate the error bound using Equation 3.18:

$$
E_{p_L}(y) = y \cdot RE_{p_L}. \tag{3.18}
$$

where y is the run-time data and RE_{p_L} is the relative error which is profiled using a number of test vectors relative to a double precision data-path.

On the other hand, we need to decide the precision used in the reduced precision data-paths. A lower precision increases the level of parallelism and hence increases the throughput of reduced precision data-path. However, it increases the ratio of re-computation and the total run-time. It is important to find an optimal precision for the best performance. When the properties of data set do not change, the ratio of re-computation can be determined by offline profiling. Otherwise, the optimal precision has to be searched at run-time using our proposed method as shown in Algorithm 2. When a new data set is applied or the ratio of re-computation exceeds a threshold, Algorithm 2 is invoked on the CPU to reconfigure the FPGA with a higher precision. On a system with multiple FPGAs, one of the FPGAs is reconfigured to approach the optimal precision over a number of time steps while the remaining FPGAs keep the system running. TH_{comp,p_L}, which will be seen in Equation 3.19 in Section 3.4.3, is the run-time measured throughput when using reduced precision p_L.

Algorithm 2 Run-time tuning of precision for system with $N \geq 2$ FPGAs

1: Get the list of precisions P
2: $TH_{comp,p_{test}} \leftarrow 0$
3: **repeat**
4: $TH_{comp,p_L} \leftarrow TH_{comp,p_{test}}$
5: $p_{test} \leftarrow \min(p \in P)$
6: Remove p_{test} from P
7: Configure the $FPGA_1$ with precision p_{test}, $FPGA_{2...N}$ are not reconfigured
8: Compute PQ and get $TH_{comp,p_{test}}$
9: **until** $TH_{comp,p_{test}} < TH_{comp,p_L}$

3.4 Reconfigurable System Design

In this section, we present our design which treats input points as a data stream that propagates through the customised system architecture. We also propose an analytical model for performance estimation.

3.4.1 Streaming Data Structure

In PQ, there are N_P points to represent a mesh. Referring to Equation 3.10, PQ computes the shortest distance from each point to the segment boundary defined by N_C contours. An intuitive implementation is to stream one point into the FPGA at the beginning, then the contours are streamed in the subsequent N_C iterations. In other words, Equation 3.3 to 3.11 are iterated for $N_C - 1$ times. However, since every comparison operation in Equation 3.11 takes more than one clock cycles of latency (denoted as L_{cmp}), the next comparison can only start after the current one completes. This significantly reduces the FPGA's throughput for L_{cmp} times because the pipeline is not fully-filled.

To tackle this problem, we propose a data structure for efficient streaming. As shown in Figure 3.2, data are streamed in an order as indicated by the arrows. In each iteration of N_S cycles, N_S (a number greater than L_{cmp}) points are processed together as a group. A new contour value is streamed in at the beginning of each iteration. In this manner, N_S points are being processed together in the pipeline to retain one output per clock cycle.

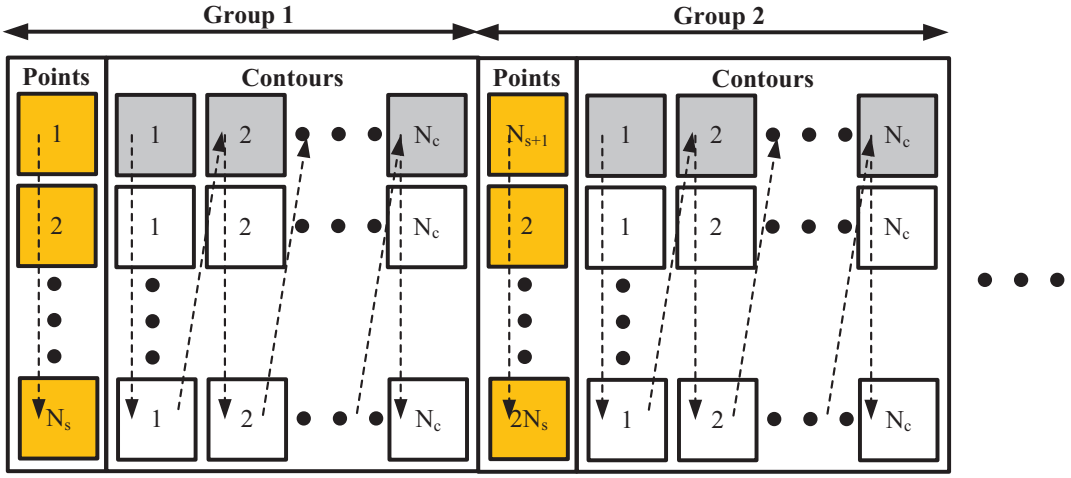


Figure 3.2: Data structure: N_S points are processed in a group. Each point of a group is iterated for N_C times. Data are streamed in an order as indicated by the arrows.

3.4.2 System Architecture

Figure 3.3 shows our proposed system architecture which consists of three major components.

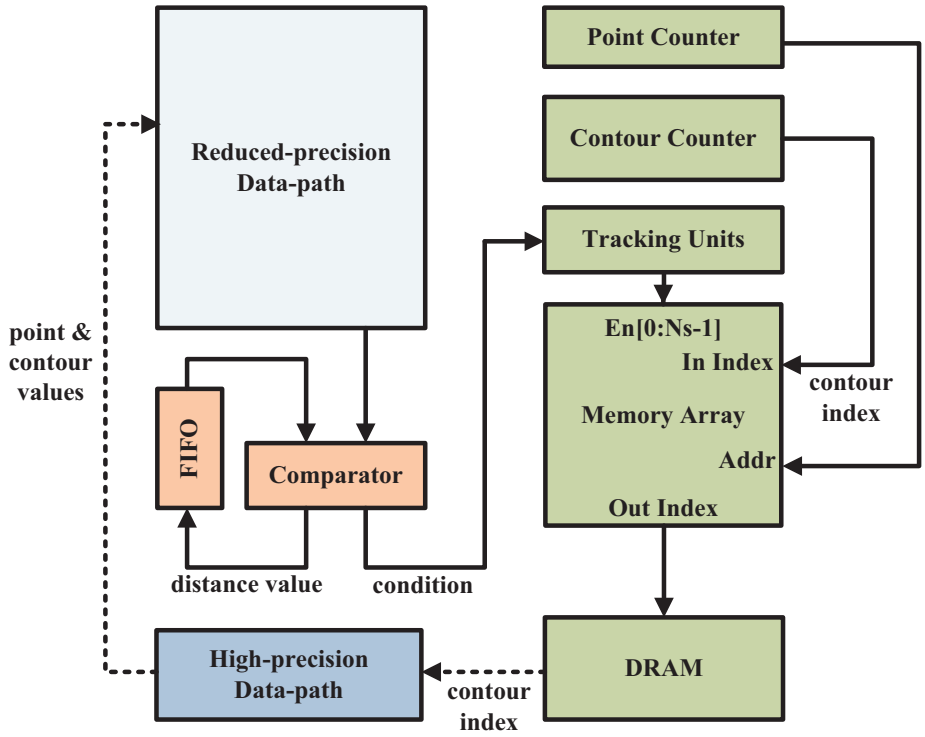


Figure 3.3: System architecture: Solid lines represent communication on the FPGA board while dotted lines represent the bus connecting the reduced precision data-path on FPGA to the high precision data-path on CPU.

Data-paths: As mentioned in Section 3.3, we employ reduced precision on FPGA to compute the deviations. The high precision data-path on CPU re-computes the deviations which are not sufficiently accurate, and then it calculates the penetration depth based on the minimum deviation. The reduced precision and high precision data-paths are interfaced by a comparator and a memory architecture as described below.

Comparator: The comparator compares the values of two point-segment distances and determines which one is smaller. Consider a group of N_S points (i.e. $\boldsymbol{x_1}$, $\boldsymbol{x_2}$, ..., $\boldsymbol{x_{N_S}}$) being processed together in the pipeline, we use a FIFO of length N_S where each slot of the FIFO stores the latest minimum deviation corresponding to a point. Since the point-segment distances are calculated in reduced precision, according to Section 3.3.2, either one of the three conditions happens: (A) The distance from the data-path is smaller; (B) The distance stored in the FIFO is smaller; (C) The difference between the two distances is too small, so re-computation in high precision is necessary.

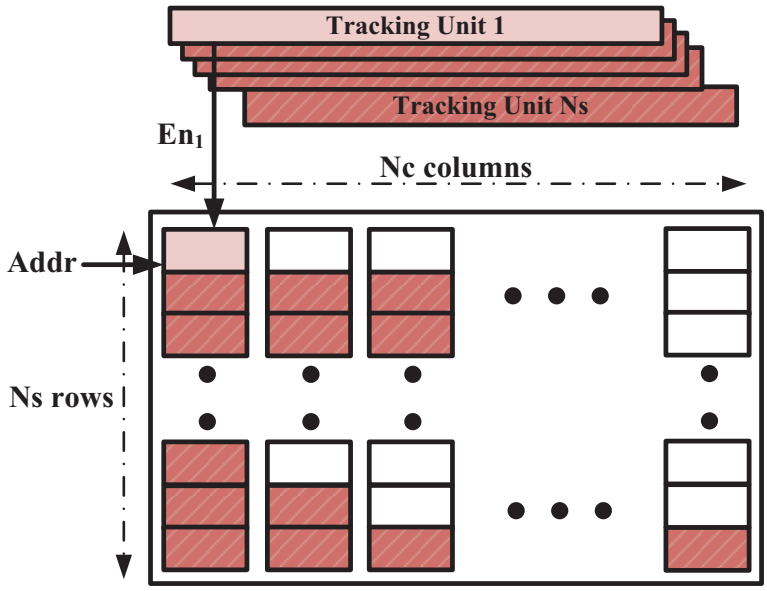
Memory Architecture: The purpose of the memory architecture is to store the contours that require re-computation. We design a memory array as shown in Figure 3.4. There are N_S rows, each of which corresponds to the computation of one point which is addressed by a *point counter*. Each row consists of N_C elements and it serves as a buffer for contours that may need re-computation. N_C elements are needed as in the worst case all the contours have to be re-computed. Instead of storing the contours in three-dimensional coordinate, we store their indices so as to save memory space. The indices are counted by a *contour counter*. There are N_S *tracking units*, each for one row, to keep track of the latest elements where the indices should be written.

To understand the mechanism of memory architecture, consider the example in Figure 3.4(a). First, the deviation in distance of *point 1* is being calculated. If the comparator indicates *condition A*, the value from the reduced precision data-path is the smallest, and all previous values stored in that row will be cleared. Second, the index corresponding to the new value is written to *element 1* of *row 1*. Third, *tracking unit 1* is updated to point to that element. If *condition B* is indicated, the minimum value is already stored in the memory and no update is required. Consider another example in Figure 3.4(b) where the calculation of *point* N_S indicates *condition C*. Both the indices in the memory and from the data-path should be stored. Thus, a contour index is written to the next element and *tracking unit* N_S advances one element further.

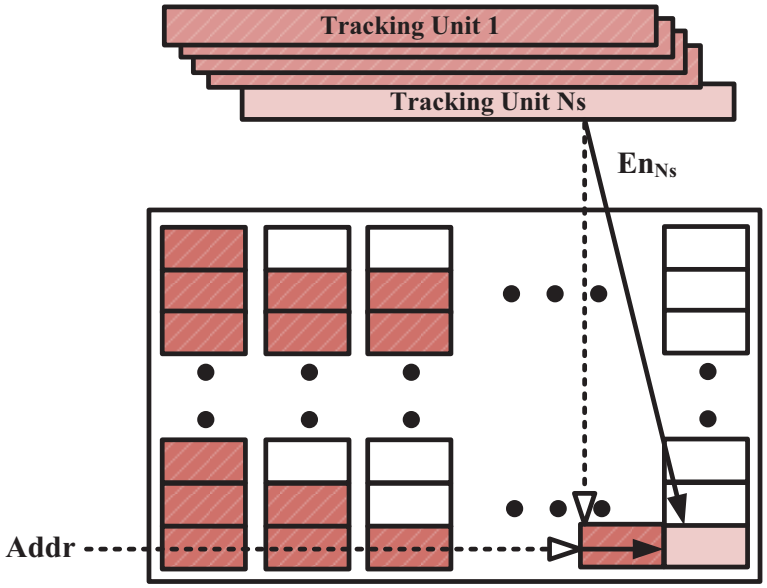
After a group of points are processed, the contour indices stored in the memory array are transferred to the Dynamic Random-Access Memory (DRAM) on the FPGA board. The data on DRAM will be accessed by the high precision data-path. To fully utilise the memory bandwidth, only non-empty memory columns are transferred in burst to the DRAM.

3.4.3 Performance Estimation

We derive a performance model to make the most effective use of the FPGA's resources and to address real-time requirements. The results will be presented in Section 3.5.2 and 3.5.3. The total computation time T_{comp} is affected by the time spent on three parts: (1) the reduced precision data-path on FPGA, (2) the high precision data-path on CPU, (3) the data



(a) Condition A: the value from the reduced precision data-path is the smallest, *tracking unit 1* points to the *element 1* of *row 1*. Previous vales stored in *row 1* are cleared.



(b) Condition C: both the value in the memory and the index from the data-path should be stored. A contour index is written to the next element and *tracking unit* N_S advances one element further.

Figure 3.4: Memory array stores contour indices for re-computation.

transfer through the bus connecting the CPU to FPGA. Equation 3.19 shows the three parts respectively:

$$\begin{aligned} T_{comp,p_L} &= T_{p_L} + T_{p_H} + T_{tran}, \\ TH_{comp,p_L} &= \frac{1}{T_{comp,p_L}}, \end{aligned} \tag{3.19}$$

where T_{p_L}, T_{p_H} and T_{tran} represent the time spent on (1), (2) and (3) respectively.

The computation time of FPGA, T_{p_L}, is shown in Equation 3.20:

$$T_{p_L} = \frac{N_P \cdot (N_C + L_{output})}{freq_{p_L} \cdot N_{p_L}} + L_{p_L}, \tag{3.20}$$

where N_P is the number of points, N_C is the number of contours, L_{p_L} is the length of the data-path but this term is usually negligible when compared with the amount of data being processed. Each point needs L_{output} cycles to output indices on the memory array to DRAM.

L_{output} is affected by the bit-width available between the FPGA and the DRAM and their relations are shown in Equation 3.21:

$$L_{output} = \frac{N_C}{N_{output}}\ , \quad N_{output} = \frac{W_{DRAM}}{W_{idx} \cdot N_{p_L}}. \tag{3.21}$$

The computation time of CPU, T_{p_H}, is related to the amount of data and the ratio of re-computation:

$$T_{p_H} = \alpha \cdot R \cdot N_P \cdot N_C. \tag{3.22}$$

The data transfer time from the DRAM to CPU, T_{tran}, is judged by the amount of data, the ratio of re-computation, R,and the bandwidth of the bus connecting the CPU to FPGA, BW_{bus}:

$$T_{tran} = \frac{R \cdot N_P \cdot N_C \cdot W_{idx}}{BW_{bus}}. \tag{3.23}$$

With the model, designer can ensure that the system parameters (Table 3.1) do not cause the system to fail real-time application's deadline.

Table 3.1: Parameters of the performance model.

N_P	Number of points
N_C	Number of contours
N_{p_L}	Number of reduced precision data-path
N_{p_H}	Number of high precision data-path
L_{p_L}	Length of the data-path
N_{output}	Number of outputs per data-path per cycle
L_{output}	Number of output cycles
L_{cmp}	Latency of a comparison operation
R	Ratio of re-computation
W_{DRAM}	Bit-width of FPGA-DRAM connection
W_{idx}	Bit-width of one contour index
$freq_{p_L}$	Clock frequency of reduced precision data-path
α	Empirical constant of CPU speed
BW_{bus}	Bandwidth of the bus connecting the CPU to FPGA

3.5 Experimental Evaluation

3.5.1 General Settings

We use the MPC-C500 reconfigurable system from Maxeler Technologies for our evaluation. The system has four MAX3 cards, each of which has a Virtex-6 XC6VSX475T FPGA with 476,100 logic cells and 2,016 DSPs. The cards are connected to two Intel Xeon X5650 CPUs and each card communicates with the CPUs via a PCI Express gen2 x8 link. The CPUs have 12 physical cores and are clocked at 2.66 GHz. We develop the FPGA kernels using MaxCompiler which adopts a streaming programming model supporting customisable floating-point data formats.

We also build a CPU-based system by implementing the PQ formulation on a platform with two Intel Xeon X5650 CPUs running at 2.66 GHz. The code is written in C++ and compiled by Intel C compiler with the highest optimisation. OpenMP library is used to parallelise the program for multiple cores. IEEE double precision floating point numbers are used.

For the GPU-based system, we use an NVIDIA Tesla C2070 GPU which has 448 cores running at 1.15 GHz.

Our PQ implementation supports 100 contours and we set an update rate of 1 kHz as the

real-time requirement.

3.5.2 Parallelism versus Precision

Figure 3.5 shows the overall computation time (T_{Comp}) and the degree of parallelism of PQ versus different number of mantissa bits. Please note that all different configurations of mantissa bits have the same output accuracy. The data set includes 73k points and 100 contours. The computation times are obtained using our analytical model in Section 3.4.3 and they are verified experimentally using the implementation. The degree of parallelism is obtained by filling the FPGA with data-paths until the logic cell utilisation exceeds 80% after the placement and routing process. The degree of parallelism is the highest when we start with four mantissa bits. Using more mantissa bits decreases the parallelism as well as the ratio of re-computation, therefore T_{p_L} increases but T_{p_H} decreases. As shown by the dotted line in the figure, a minimum computation time is achieved when 10 mantissa bits are used. Note that when the number of mantissa bits is more than 36, only one data-path can be mapped onto the FPGA. In such cases, we can implement the data-path in double precision directly which does not require any re-computation on CPU. This is indicated by the last data points of both curves.

3.5.3 Ratio of Re-computation versus Precision

The dotted line in Figure 3.6 shows the ratio of re-computation versus the number of mantissa bits. The results are obtained from a software version of PQ implementation with precisions adjusted using MPFR library [111]. For each point, 100 computations of deviation in distance are required. The ratio of re-computation drops exponentially as the number of mantissa bits increases. From the performance perspective, to the left the ratio of re-computation is too high, to the right the decrease of re-computation cannot offset the impact brought by the decrease in parallelism. When the number of mantissa bits is four, in average 2.66 out of 100 computations need to be re-computed using high precision, i.e. the ratio of re-computation is 2.66%. When the number of mantissa bits is greater then 15, the ratio of re-computation drops to 1% which is

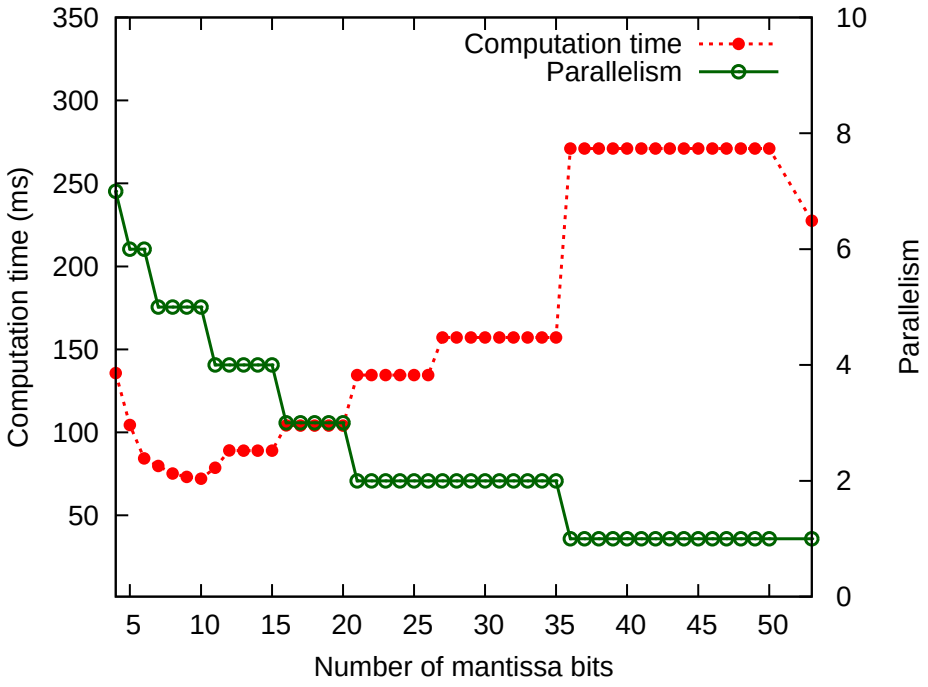


Figure 3.5: Computation time (dotted line) and the level of parallelism (solid line) versus different number of mantissa bits.

the minimum value as only one out of 100 values is re-computed. The last data points of both curves indicate the situation when double precision is used on the FPGA and no re-computation is necessary.

The solid line in Figure 3.6 shows the number of point processed in 1 ms versus the number of mantissa bits. The application has a real-time update requirement of 1 kHz so the results are updated every 1 ms. The number of required points is based on the user specification of the model resolution in three-dimensional space. When the number of mantissa bits is 10, the maximum number of points can be processed. It is because the throughput is the highest by balancing the ratio of re-computation and the degree of parallelism. Since more points can be processed in real-time, we can handle a more complex robot model with a finer resolution.

3.5.4 Comparison: CPU, GPU and Reconfigurable System

Table 3.2 compares the performance of PQ running on CPU, GPU and FPGA in double precision arithmetic, and our proposed reconfigurable system with CPUs and FPGAs.

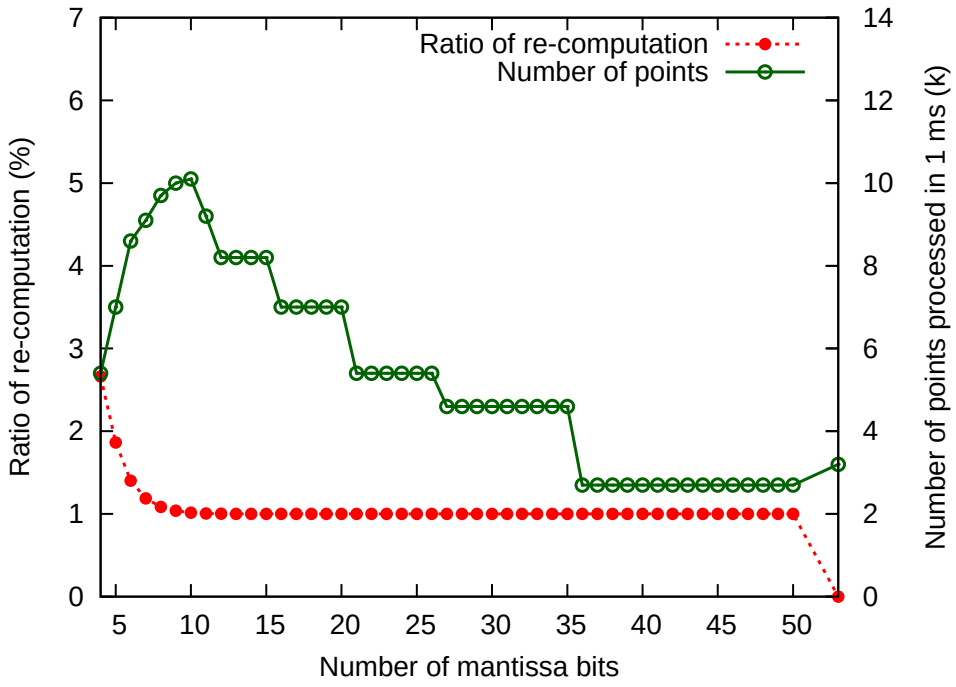


Figure 3.6: Ratio of re-computation (dotted line) and the number of points processed in 1 ms (solid line) versus different number of mantissa bits.

In 1 ms, our proposed system is able to process 58 times more points than a 12-core CPU system, and 9 times more points than a GPU system. Without any optimisation, we can only implement one double precision data-path on an FPGA. Our proposed approach can support five reduced precision data-paths to be implemented in parallel on one chip, i.e. 20 data-paths in total on the 4-FPGA system. The clock frequency is also higher because reduced precision simplifies routing of signals. The performance gain over a double precision FPGA implementation is over 3 times.

Figure 3.7 shows the computation time for a PQ update against the number of points. The black solid line indicates the real-time bound of 1 ms. In the CPU-based system, even with the fastest configuration (12 cores), only 173 points can be processed in real-time. Meanwhile, the performance of our proposed 1-FPGA reconfigurable system is on-par with a 4-FPGA reconfigurable system in double precision. Our proposed 4-FPGAs system can process 10,094 points within the 1 ms interval.

Table 3.2: Comparison of PQ computation in 1 ms using CPU-based system (CPU), GPU-based system (GPU), double precision FPGA-based reconfigurable system (RS DP) and FPGA+CPU reconfigurable system with reduced precision (RS RP).

	CPU	GPU	RS DP	RS RP
Clock frequency (MHz)	2,660	1,150	80	130 & 2,660 [a]
Number of cores	12	448	4	20
Number of mantissa bits	53	53	53	10 & 53 [b]
Number of p_L eval. (k)	0	0	0	1009.4
Number of p_H eval. (k)	173	106	320	10.1
Number of total eval. (k)	173	106	320	1019.5
Evaluated in p_H (%)	100	100	100	1
Number of points in 1 ms	173	1,064	3,200	10,094
Normalised speedup	1x	6.15x	18.5x	58.35x
Reduced precision gain	-	-	1x	3.15x

[a] FPGA and CPU clock frequencies.
[b] Reduced precision and high precision.

3.6 Summary

This chapter presents a reconfigurable computing solution to proximity query computation. To the best of our knowledge, this is the first application of FPGAs to this problem.

We transform the algorithm to enable pipelining and apply reduced precision methodology to maximise parallelism. Run-time reconfiguration is employed to optimise precision automatically. We then map the optimised algorithm to a reconfigurable system with four Virtex-6 FPGAs and 12 CPU cores. Our proposed reconfigurable system achieves 478 times speedup over a single-core CPU, 58 times speedup over a 12-core CPU system, 9 times speedup over a GPU, and 3 times speedup over an FPGA implementation in double precision. Since more points can be processed in real-time, we can handle a more complex robot model with a finer resolution.

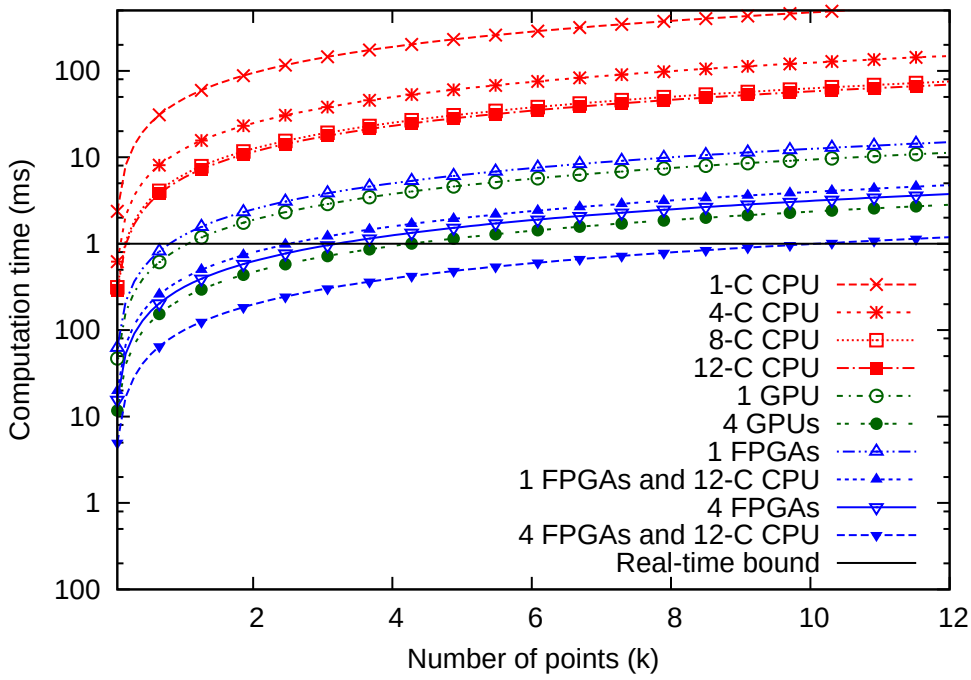


Figure 3.7: Computation time for a PQ update with 100 contours versus the number of points.

Chapter 4

Run-time Adaptation of System Configuration

4.1 Introduction

This chapter presents an adaptation approach for reconfigurable systems. The approach provides an efficient solution to real-time SMC methods. We derive an adaptive SMC algorithm that adjusts its computation complexity at run time based on the quality of results. To map our algorithm to a reconfigurable system consisting of multiple FPGAs and CPUs, we design a pipeline-friendly data structure to make effective use of the stream computing model. Moreover, we accelerate the algorithm with a data compression scheme and data control separation.

The key contributions of this chapter include:

- An adaptive SMC algorithm which adapts the size of particle set at run-time. The algorithm is able to reduce computation workload while maintaining the quality of results.
- Mapping the proposed algorithm to a scalable and reconfigurable system by following the stream computing model. A novel data structure is designed to take advantage of the architecture and to alleviate the data transfer bottleneck. The system uses the run-time reconfigurability of FPGA to switch between computation mode and low-power mode.

- An implementation of a robot localisation application targeting the proposed system. Compared to a non-adaptive and non-reconfigurable implementation, the idle power of our proposed system is reduced by 25-34% and the overall energy consumption decreases by 17-33%. Our system with four FPGAs is up to 169 times faster than a single core CPU, 41 times faster than a 1U CPU server with 12 cores, and 3 times faster than a modelled four-GPU system.

The rest of the chapter is organised as follows. Section 4.2 describes the proposed adaptive SMC methods. Section 4.3 presents the heterogeneous reconfigurable systems which is optimised for adaptive SMC methods. Section 4.4 discusses techniques which reduce the transfer overhead of particle stream. Section 4.5 provides experimental results and Section 4.6 concludes our work.

4.2 Adaptive Particle Filter

This section introduces an adaptive SMC algorithm which changes the number of particles at each time-step. The algorithm is inspired by [104] and we transform it to a pipeline-friendly version for mapping to the stream computing architecture. This algorithm is shown in Algorithm 3 which consists of four stages.

Stage 1 - Sampling and Importance Weighting (line 8 to 9): At the initial time-step ($t = 0$), the number of particles P_0 is initialised with P_{max} which is the maximum number of available particles. At the subsequent time-steps, the number of particles is denoted as P_t. Initially, the particle set $\{s_t^{(i)}\}_{i=1}^{P_t}$ is sampled to $\{s_{t+1}^{'(i)}\}_{i=1}^{P_t}$. Then a weight from $\{w^{(i)}\}_{i=1}^{P_t}$ is assigned to each particle. As a result, $\{s_{t+1}^{'(i)}\}_{i=1}^{P_t}$ and $\{w^{(i)}\}_{i=1}^{P_t}$ give an estimation of the next state.

During sampling and importance weighting, the computation of every particle is independent of each other. The mapping of computation to FPGAs will be described in Section 4.3.

Algorithm 3 Adaptive SMC algorithm.

$P_0 \leftarrow P_{max}$
$\{X_0^{(i)}\}_{i=1}^{P_0} \leftarrow$random set of particles
$t = 1$
for each step t **do**
  $r = 0$
  while $r \leq itl\_repeat$ **do**
    —On FPGAs—
    Sample a new state $\{s_{t+1}^{'(i)}\}_{i=1}^{P_t}$ from $\{s_t^{(i)}\}_{i=1}^{P_t}$
    Calculate unnormalised importance weights $\{w^{'(i)}\}_{i=1}^{P_t}$ and accumulate the weights as w_{sum}
    Calculate the lower bound of sample size $\widetilde{P}_{t+1}$ by Equation 4.1
    —On CPUs—
    Sort $\{s_{t+1}^{'(i)}\}_{i=1}^{P_t}$ in descending $\{w^{'(i)}\}_{i=1}^{P_t}$
    if $\widetilde{P}_{t+1} < P_t$ **then**
      $P_{t+1} = max\left(\lceil \widetilde{P}_{t+1} \rceil, P_t/2\right)$
      Set $a = 2P_{t+1} - P_t$ and $b = P_{t+1}$
      –Do the following loop in parallel–
      for i in $P_t - P_{t+1}$ **do**
        $s_{t+1}^{'(i)} = \frac{s_{t+1}^{'(a)} w^{'(a)} + s_{t+1}^{'(b)} w^{'(b)}}{w^{'(a)} + w^{'(b)}}$
        $w^{'(i)} = w^{'(a)} + w^{'(b)}$
        $a = a + 1$ and $b = b - 1$
      end for
    else if $\widetilde{P}_{t+1} \geq P_t$ **then**
      $a = 0$ and $b = 0$
      for i in $P_{t+1} - P_t$ **do**
        if $w^{'(a)} < w^{'(a+1)}$ and $a < P_{t+1}$ **then**
          $a = a + 1$
        end if
        $s_{t+1}^{'(P_t+b)} = s_{t+1}^{'(a)}/2$
        $s_{t+1}^{'(a)} = s_{t+1}^{'(a)}/2$
        $w^{'(P_t+b)} = w^{'(a)}/2$
        $w^{'(a)} = w^{'(a)}/2$
        $b = b + 1$
      end for
    end if
    Resample $\{s_{t+1}^{'(i)}\}_{i=1}^{P_t}$ to $\{s_{t+1}^{(i)}\}_{i=1}^{P_{t+1}}$
    $r = r + 1$
  end while
end for

Stage 2 - Lower Bound Calculation (line 10): This stage derives the smallest number of particles that are needed in the next time-step in order to bound the approximation error. The adaptive algorithm seeks a value which is less than or equal to P_{max}. This number, denoted as $\widetilde{P}_{t+1}$, is referred to as the lower bound of sampling size. It is calculated by Equation 4.1:

$$\widetilde{P}_{t+1} = \sigma^2 \cdot \frac{P_{max}}{Var(\{s_{t+1}^{'(i)}\}_{i=1}^{P_t})}, \tag{4.1}$$

where

$$\begin{aligned}\sigma^2 = &\sum_{i=1}^{P_t}\left(w^{(i)} \cdot s_{t+1}^{'(i)}\right)^2 - 2 \cdot E(\{s_{t+1}^{'(i)}\}_{i=1}^{P_t}) \cdot \sum_{i=1}^{P_t}\left((w^{(i)})^2 \cdot s_{t+1}^{'(i)}\right) \\ &+ \left(E(\{s_{t+1}^{'(i)}\}_{i=1}^{P_t})\right)^2 \cdot \sum_{i=1}^{P_t}(w^{(i)})^2,\end{aligned} \tag{4.2}$$

$$Var(\{s_{t+1}^{'(i)}\}_{i=1}^{P_t}) = \sum_{i=1}^{P_t}\left(w^{(i)} \cdot (s_{t+1}^{'(i)})^2\right) - \left(E(\{s_{t+1}^{'(i)}\}_{i=1}^{P_t})\right)^2, \tag{4.3}$$

$$E(\{s_{t+1}^{'(i)}\}_{i=1}^{P_t}) = \sum_{i=1}^{P_t} w^{(i)} \cdot s_{t+1}^{'(i)}. \tag{4.4}$$

As shown in Equation 4.2 to 4.4, $w^{(i)}$ is a normalised term. To calculate $w^{(i)}$, a traditional software-based approach is to iterate through the set of particles twice. The sum of weights w_{sum} and unnormalised weight $w^{'(i)}$ are calculated in the first iteration. Then $w^{(i)}$ is obtained by dividing $w^{'(i)}$ by w_{sum} in the second iteration. However, this method is inefficient for FPGA implementation. It is because 2 P_t cycles are needed to process P_t pieces of data, which makes the throughput reduced to 50%.

To fully utilise deep pipelines targeting an FPGA, we perform function transformation. Given $w^{(i)} = \frac{w^{'(i)}}{w_{sum}}$, we move w_{sum} from Equation 4.2 to 4.4. By doing so, we obtain a transformed form as shown in Equations 4.5 to 4.7.

$$\begin{aligned}\sigma^2 =& \frac{1}{(w_{sum})^2} \cdot (\sum_{i=1}^{P_t} \left(w^{'(i)} \cdot s_{t+1}^{'(i)} \right)^2 - 2 \cdot E(\{s_{t+1}^{'(i)}\}_{i=1}^{P_t}) \cdot \sum_{i=1}^{P_t} \left((w^{'(i)})^2 \cdot s_{t+1}^{'(i)} \right) \\ &+ \left(E(\{s_{t+1}^{'(i)}\}_{i=1}^{P_t}) \right)^2 \cdot \sum_{i=1}^{P_t} (w^{'(i)})^2),\end{aligned} \tag{4.5}$$

$$Var(\{s_{t+1}^{'(i)}\}_{i=1}^{P_t}) = \frac{1}{w_{sum}} \cdot \sum_{i=1}^{P_t} \left(w^{'(i)} \cdot (s_{t+1}^{'(i)})^2 \right) - \left(E(\{s_{t+1}^{'(i)}\}_{i=1}^{P_t}) \right)^2, \tag{4.6}$$

$$E(\{s_{t+1}^{'(i)}\}_{i=1}^{P_t}) = \frac{1}{w_{sum}} \cdot \sum_{i=1}^{P_t} w^{'(i)} \cdot s_{t+1}^{'(i)}. \tag{4.7}$$

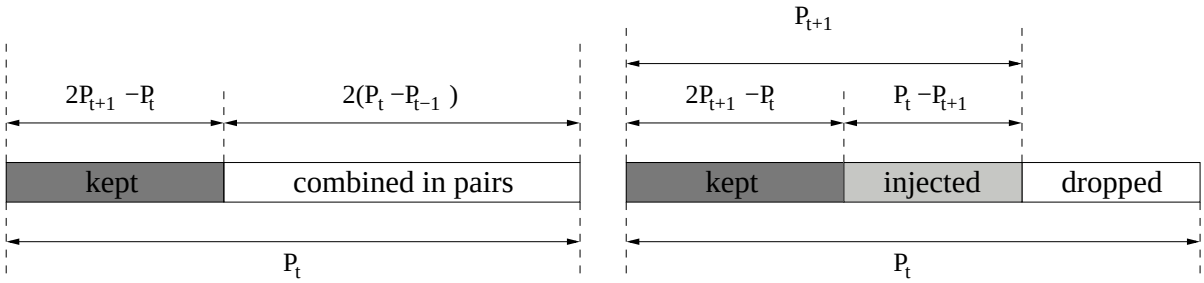
w_{sum} and $w^{'(i)}$ are computed simultaneously in two separate data paths. At the last clock cycle of the particle stream, σ^2, $Var(\{s_{t+1}^{'(i)}\}_{i=1}^{P_t})$ and $E(\{s_{t+1}^{'(i)}\}_{i=1}^{P_t})$ are obtained. The details of the FPGA kernel design will be explained in Section 4.3.

Stage 3 - Particle Set Size Tuning (line 12 to 34): The adaptive approach tunes the particle set size to fit the lower bound P_{t+1}. This stage is done on the CPUs because the operations involve non-sequential data access that cannot be mapped efficiently to FPGAs.

The particles are sorted in descending order according to their weights. As the new sample size can increase or decrease, there are two cases:

- **Case I: Particle set reduction when $\widetilde{P}_{t+1} < P_t$**

  The lower bound P_{t+1} is set to $max\left(\lceil \widetilde{P}_{t+1} \rceil, P_t/2\right)$. Since the new size is smaller than the old one, some particles are combined to form a smaller particle set. Figure 4.1 illustrates the idea of particle reduction. The first $2P_{t+1} - P_t$ particles with higher weights are kept and the remaining $2(P_t - P_{t+1})$ particles are combined in pairs. As a result, there are $P_t - P_{t+1}$ new particles injected to form the target particle set with P_{t+1} particles. We combine the particles deterministically to keep the statements in the loop independent of each other. As a result, loop unrolling is undertaken to execute the statements in



(a) Combining the last $2(P_t - P_{t+1})$ particles with lower weights

(b) P_{t+1} new particles are formed

Figure 4.1: Particle set reduction.

parallel. The complexity of the loop is in $\mathcal{O}\left(\frac{P_t - P_{t+1}}{N_{parallel}}\right)$, where $N_{parallel}$ indicates the level of parallelism.

- **Case II: Particle set expansion when $\widetilde{P}_{t+1} \geq P_t$**

  The lower bound P_{t+1} is set to $\widetilde{P}_{t+1}$. Some particles are taken from the original set and are inserted to form a larger set. The particles with larger weight would have more descendants. As shown in line 22 to 34, the process requires picking the particle with the largest weight at each iteration of particle incision. Since the particle set is pre-sorted, the complexity of particle set expansion is $\mathcal{O}(P_{t+1} - P_t)$.

Stage 4 - Resampling (line 35): Resampling is performed to pick P_{t+1} particles from $\{s_{t+1}^{'(i)}\}_{i=1}^{P_t}$ to form $\{s_{t+1}^{(i)}\}_{i=1}^{P_{t+1}}$. The process has a complexity of $\mathcal{O}(P_{t+1})$.

4.3 Heterogeneous Reconfigurable System

This section describes the proposed heterogeneous reconfigurable system. It is scalable to cope with different FPGA devices and applications. The reconfigurable system also takes advantage of the run-time reconfiguration feature for power and energy reduction.

4.3.1 Mapping Adaptive SMC to Reconfigurable System

The design of reconfigurable system is shown in Figure 4.2. A heterogeneous structure is employed to make use of multiple FPGAs and CPUs. FPGAs and CPUs communicate through high bandwidth buses. FPGAs are responsible for (1) sampling, (2) importance weighting, and (3) lower bound calculation. The data paths on the FPGAs are fully-pipelined. Each FPGA has its own on-board DRAM to store the large amount of particle data. On the other hand, the CPUs gather all the particles from FPGAs to perform particle set size tuning and resampling.

Resampling requires a collective operation over the weights which makes it less readily parallelised in hardware. Different resampling methods have been proposed aiming to parallelise the algorithm on FPGAs [112] and GPUs [113]. Direct resampling methods such as stratified [114] and systematic resampling [115] can achieve certain degree of parallelism by removing data dependency. Monte Carlo based methods such as Metropolis [116] and rejection sampling [117] strategies are more straightforward to be implemented in parallel devices. However, Metropolis method results in a biased sample, while rejection results in non-deterministic timing. Despite the parallelisation effort, these methods do not address the problem of non-sequential memory access pattern which has significant impact on performance when the particles are stored in off-chip memory instead of on-chip memory.

4.3.2 FPGA Kernel Design

Sampling, importance weighting and lower bound calculation are the most computation intensive stages. In each time-step, these three stages are iterated for itl_repeat times. An FPGA kernel enabling efficient acceleration is proposed.

Figure 4.4 shows the components of the FPGA kernel. The kernel is fully pipelined to achieve one output per clock cycle. It can also be replicated as many times as FPGA resource allow and the replications can be split across multiple FPGA boards. The kernel takes three inputs from the CPUs or on-board DRAM: (1) states, (2) controls, and (3) seeds. Application specific parameters are stored in ROMs. Three building blocks corresponding to the sampling,

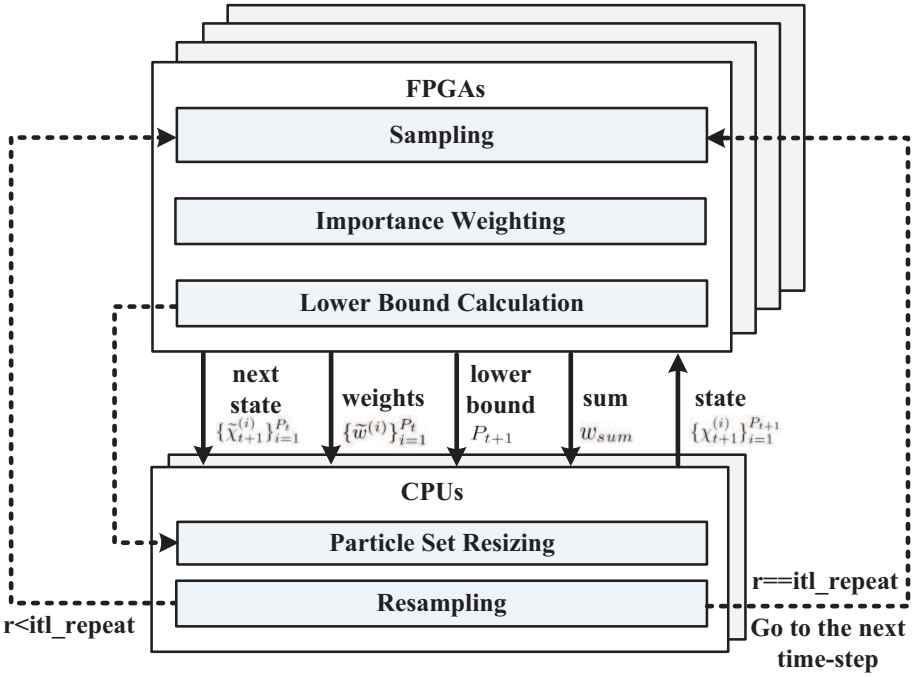


Figure 4.2: Heterogeneous reconfigurable system (Solid lines: data paths; Dotted lines: control paths).

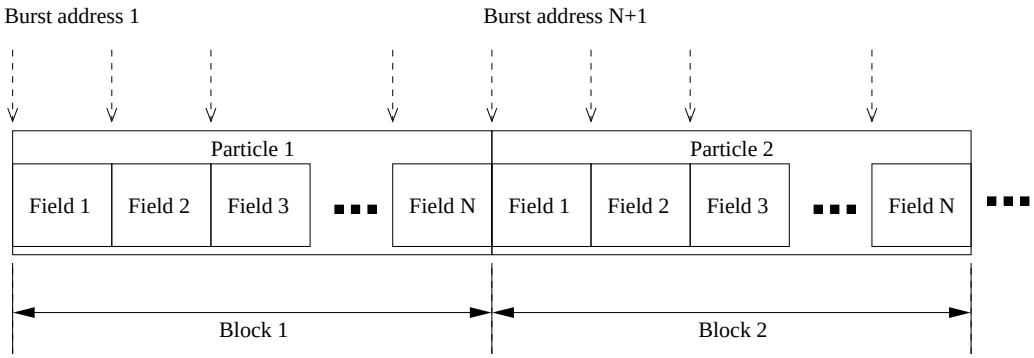


Figure 4.3: A particle stream.

importance weighting and lower bound calculation stages are described in Section 4.2.

For sampling and importance weighting, the computation of each particle is independent of each other. Particles are fed to the FPGAs as a stream shown in Figure 4.3. Each block of the particle stream consists of a number of data fields which store information of a particle, where the number of data fields is application dependent. In every clock cycle, one piece of data is transferred from the onboard memory to an FPGA data path. Each FPGA data path has a long pipeline where each stage is filled with a piece of data, and therefore many particles are processed simultaneously. Fixed-point data representation is customised at each pipeline stage to reduce the resource usage.

Meanwhile, the accumulation of w_{sum} introduces a feedback loop. A new weight comes along every cycle which is more quickly than the floating-point unit to perform addition of the previous weight. In order to achieve one result per clock cycle, fixed-point data-path is implemented while ensuring no overflow or underflow occurs.

4.3.3 Timing Model for Run-time Reconfiguration

We derive a model to analyse the computation time of reconfigurable system. The model helps us to design a configuration schedule that satisfies the real-time requirement and, if necessary, amend the application's specification. The model will be validated by experiments in Section 4.5.

The computation time, T_{comp}, of reconfigurable system consists of three components: (1) Data path time $T_{datapath}$, (2) CPU time T_{cpu}, and (3) Data transfer time T_{tran} as shown in Equation 4.8:

$$T_{comp} = itl\_repeat \cdot (T_{datapath} + T_{cpu} + T_{tran}), \tag{4.8}$$

where itl_repeat represents the number of times that the sampling, importance weighting and resampling processes are repeated in every time-step.

Data path time, $T_{datapath}$, denotes the time spent on the FPGAs.

$$T_{datapath} = \left(\frac{P_t}{f_{fpga} \cdot N_{datapath}} + L - 1 \right) \frac{1}{N_{fpga}}, \tag{4.9}$$

where P_t denotes the number of particles at the current time-step and f_{FPGA} denotes the clock frequency of the FPGAs. L is the length of the pipeline. $N_{datapath}$ denotes the number of data paths on one FPGA board. N_{fpga} is the number of FPGA boards in the system.

CPU time, T_{cpu}, denotes the time spent on the CPUs.

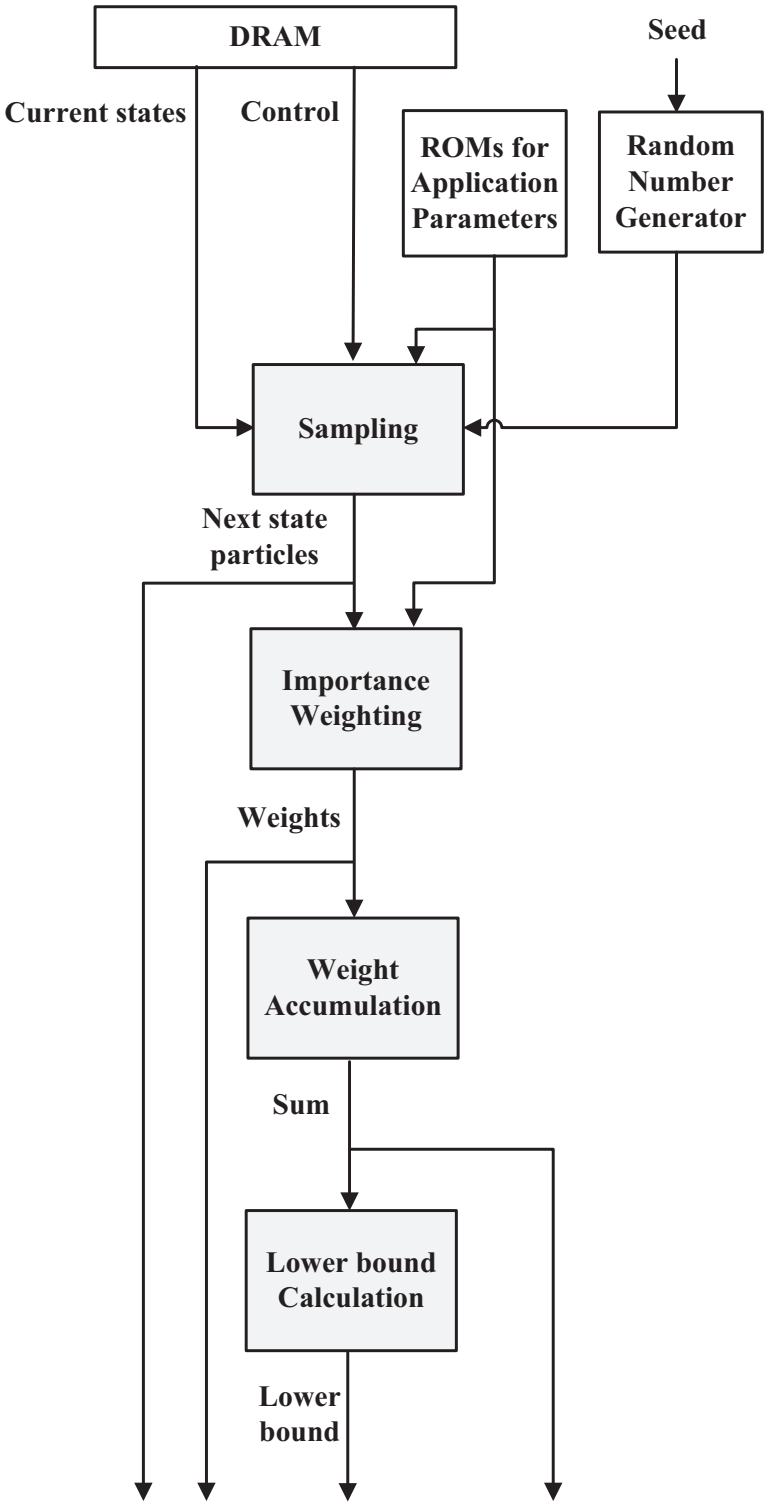


Figure 4.4: FPGA kernel design.

$$T_{cpu} = \alpha \cdot \frac{P_t}{f_{cpu}} \cdot \left(1 - par + \frac{par}{N_{thread}}\right), \tag{4.10}$$

where the clock frequency and number of threads of the CPUs are represented by f_{cpu} and N_{thread}

respectively. *par* is an application-specific parameter in the range of $[0, 1]$ which represents the ratio of CPU instructions that are parallelisable, and α is a scaling constant derived empirically.

Data transfer time, T_{tran}, denotes the time of moving a particle stream between the FPGAs and the CPUs.

$$T_{tran} = \frac{(2 \cdot df + 1) \cdot bw_{data} \cdot P_t}{f_{bus} \cdot lane \cdot eff \cdot N_{fpga}}, \tag{4.11}$$

where df is the number of data fields of a particle. For example, if a particle contains the information of coordinates (x, y) and heading h, $df = 3$. Given that the constant 1 represents the weight and the constant 2 accounts for the movement of data in and out of the FPGAs, and bw_{data} is the bit-width of one data field, the expression $(2 \cdot df + 1) \cdot bw_{data}$ is regarded as the size of a particle. f_{bus} is the clock frequency of the bus connecting the CPUs to FPGAs and $lane$ is the number of bus lanes connected to one FPGA. Since many buses, such as the PCI Express Bus, encode data during transfer, the effective data are denoted by eff (in PCI Express Gen2 the value is 8/10). In our previous work [25], the data transfer time has a significant performance impact on reconfigurable system. To reduced the data transfer overhead, we introduce a data compression technique that will be described in Section 4.4.

In real-time applications, each time-step is fixed and is known as the real-time bound T_{rt}. The derived model helps system designers to ensure that the computation time T_{comp} is shorter than T_{rt}. An idle time T_{idle} is introduced to represent the time gap between the computation time and real-time bound. It is calculated by Equation 4.12:

$$T_{idle} = T_{rt} - T_{comp}. \tag{4.12}$$

Figure 4.5(a) illustrates the power consumption of a reconfigurable system without run-time reconfiguration. It shows that the FPGAs are still drawing power after the computation finishes. By exploiting run-time reconfiguration as shown in Figure 4.5(b), the FPGAs are loaded with a low-power configuration during the idle period. Such configuration minimises the amount of

active resources and clock frequency. Equation 4.13 describes the sleep time when the FPGAs are idle and being loaded with the low-power configuration. If the sleep time is positive, reconfiguration would be helpful in these situations, where the sleep time is expressed as:

$$T_{sleep} = T_{idle} - T_{config}. \tag{4.13}$$

Configuration time, T_{config}, denotes the time needed to download a configuration bit-stream to the FPGAs:

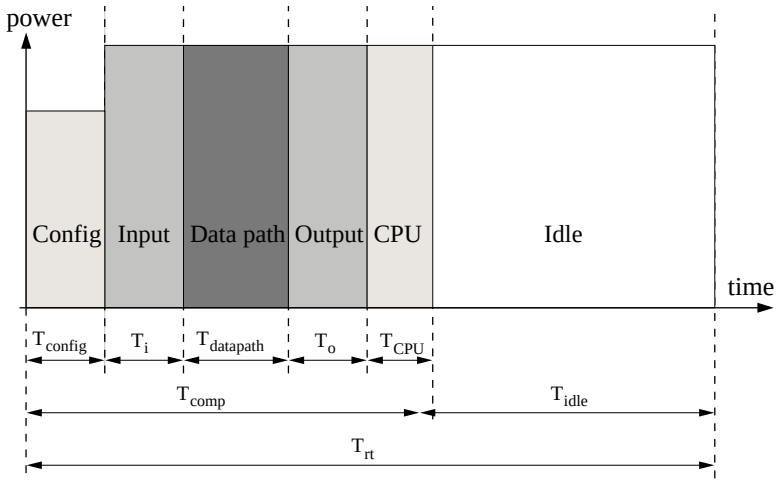
$$T_{config} = \frac{size_{bs}}{f_{config} \cdot bw_{config}}, \tag{4.14}$$

where $size_{bs}$ represents the size of bitstream in bits, and f_{config} is the configuration clock frequency in Hz and bw_{config} is the width of the configuration port.

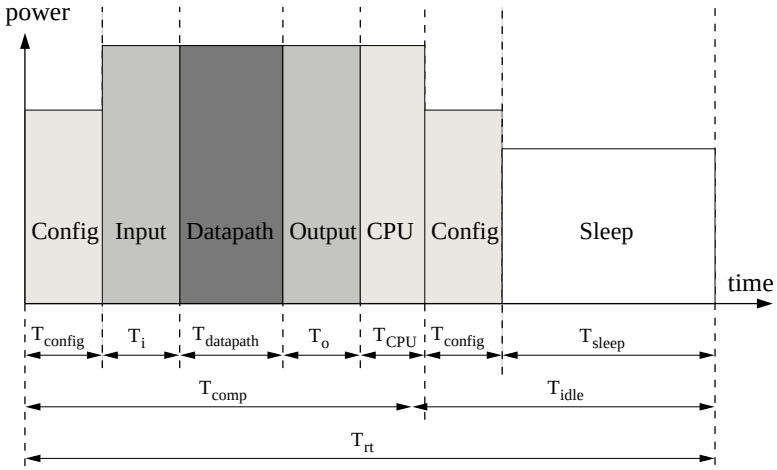
4.4 Optimising Transfer of Particle Stream

In Section 4.3, the data transfer time depends on the number of particles and the bus bandwidth between the CPUs and FPGAs. It can be a major performance bottleneck as depicted in [25]. Refer to Figure 4.6(a), each block stores the data of a particle. When the CPUs finish processing, all data are transferred from the CPUs to the FPGAs. The data transfer time cannot be reduced by either implementing more FPGA data paths or increasing the FPGAs' clock frequency because the bottleneck is at the bus connecting the CPUs and FPGAs.

To improve the data transfer performance, we design a data structure which facilitates compression of particles. The idea comes from an observation of the resampling process - some particles are eliminated and the vacancies are filled by replicating non-eliminated particles. Replication means data redundancy exists. For example, in the original data structure shown in Figure 4.6(a), particle 1 has three replicates and particle 2 is eliminated, therefore, particle 1 is stored and transferred for three times.



(a) Without reconfiguration

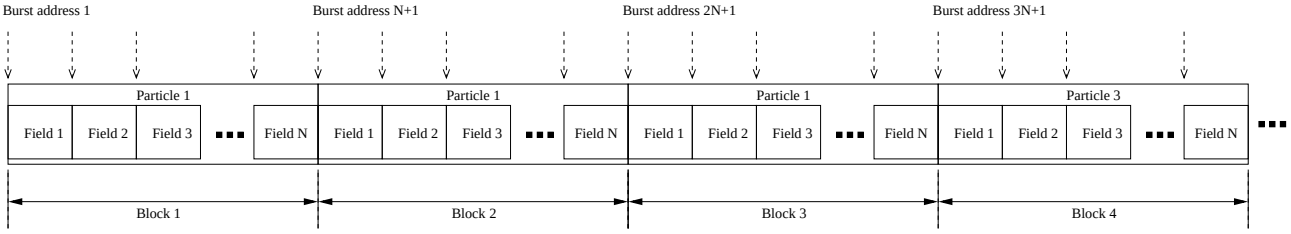


(b) With reconfiguration to low-power mode during idle

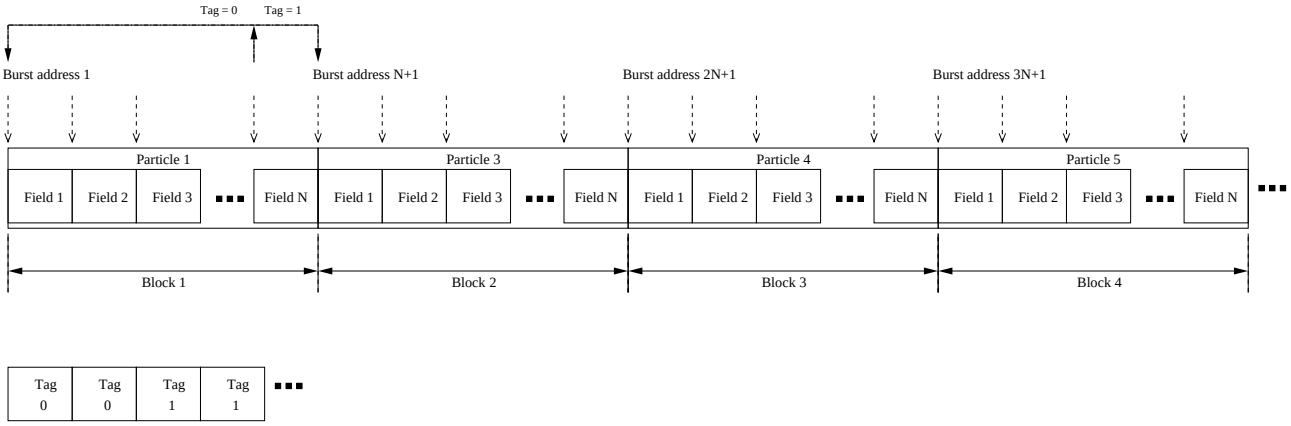
Figure 4.5: Power consumption of the reconfigurable system over time.

By using the data structure in Figure 4.6(b), data redundancy is eliminated by storing every particle once. Each particle is also transferred once. As a result, the data transfer time and memory space are reduced.

A reconfigurable system often contains DRAM which transfers data in burst in order to maximise the memory bandwidth. This works fine with the original data structure where the data are organised as a sequence from the lower address space to the upper. However, using the new data structure, the data access pattern is not sequential any more, the address goes back and forth. The DRAM controller needs to be modified so that the transfer throughput would not be affected by the change of data access pattern. As illustrated in Figure 4.6(b), a tag sequence is used to indicate the address of the next block. For example, after reading the data



(a) Particle stream before compression



(b) Compressed particle stream

Figure 4.6: Compressing particle stream: After the resampling process, some particles are eliminated and the remaining particles are replicated. Data compression is applied so that every particle is stored and transferred once only.

of particle 1, the burst address is at N. If the tag is one, the next burst address will point to the address of the next block at $N+1$. Otherwise, the burst address will point to the start address of the current block (which is 1). The data are still addressed in burst so the performance is not degraded.

The data transfer time with compression, T_{Tran}, is shown below:

$$T_{tran} = \frac{(\frac{df}{Rep} + df + 1) \cdot bw_{data} \cdot P_t}{f_{bus} \cdot lane \cdot eff \cdot N_{FPGA}}, \tag{4.15}$$

where Rep is the average number of replication of the particles, and therefore the size of the resampled particle stream is reduced by a ratio of Rep when compared to that without compression. The range of Rep is from 1 to P_t, depending on the distribution of particles after the resampling process. The effect of Rep on data transfer time will be evaluated in the next section.

4.5 Experimental Results

To evaluate the performance of the reconfigurable system and make comparison with the other systems, we implement an application which uses SMC for localisation and tracking of mobile robot. The application is proposed in [94] to track location of moving objects conditioned upon robot poses over time. Given a priori learned map, a robot receives sensor values and moves at regular time intervals. Meanwhile, M moving objects are tracked by the robot. The states of the robot and objects at time t are represented by a state vector X_t:

$$X_t = \{R_t, H_{t,1}, H_{t,2}, ..., H_{t,M}\}. \tag{4.16}$$

R_t denotes the robot's pose at time t, and $H_{t,1}, H_{t,2}, ..., H_{t,M}$ denote the locations of the M objects at the same time.

The following equation is used to represent the posterior of the robot's location:

$$p(X_t|Y_t, U_t) = p(R_t|Y_t, U_t) \prod_{m=1}^{M} p(H_{t,m}|R_t, Y_t, U_t), \tag{4.17}$$

where Y_t is the sensor measurement and U_t is the control of the robot at time t. The robot path posterior $p(R_t|Y_t, U_t)$ is represented by a set of robot-particles. The distribution of an object's location $p(H_{t,m}|R_t, Y_t, U_t)$ is represented by a set of object-particles, where each object-particle set is attached to one particular robot-particle. In other words, if there are P_r robot-particles representing the posterior over robot path, there are P_r object-particle sets, each has P_h particles.

In the application, the area of the map is 12m by 18m. The robot makes a movement of 0.5m every five seconds, i.e. $T_{rt} = 5$. The robot can track eight moving objects at the same time. A maximum of 8192 particles are used for robot-tracking and each robot-particle is associated with 1024 object-particles. Therefore, the maximum number of data path cycles is $8 \times 8192 \times 1024 = 67,108,864$. Each particle being streamed into the FPGAs contains

coordinates (x,y) and heading h which are represented by three single precision floating-point numbers. For the particle being streamed out of the FPGAs, it also contains a weight in addition to the coordinates. From Equation 4.11, the size of a particle is $(2 \cdot 3 + 1) \cdot 32$ bits = 224 bits.

4.5.1 System Settings

Reconfigurable system: Two reconfigurable systems from Maxeler Technologies are used. The system is developed using MaxCompiler, which adopts a stream computing model.

- *MaxWorkstation* is a microATX form factor system which is equipped with one Xilinx Virtex-6 XC6VSX475T FPGA. The FPGA has 297,600 LUTs, 595,200 Flip-Flops (FFs), 2,016 DSPs and 1,064 block RAMs. The FPGA board is connected to an Intel i7-870 CPU (4 physical cores, 8 threads in total, clocked at 2.93 GHz) via a PCI Express Gen2 x8 bus. The maximum bandwidth of the PCI Express bus is 2 GB/s according to the specification provided by Maxeler Technologies.

- *MPC-C500* is a 1U server accommodating four FPGA boards, each of which has a Xilinx Virtex-6 XC6VSX475T FPGA. Each FPGA board is connected to two Intel Xeon X5650 CPUs (12 physical cores, 24 threads in total, clocked at 2.66 GHz) via a PCI Express Gen2 x8 bus.

To support run-time reconfigurability, two FPGA configurations are used:

- *Sampling and importance weighting configuration* is clocked at 100 MHz. Two data paths are implemented on one FPGA to process particles in parallel. The total resource usage is 231,922 LUTs (78%), 338,376 FFs (56%), 1,934 DSPs (96%) and 514 block RAMs (48%).

- *Low-power configuration* is clocked at 10 MHz, with 5,962 LUTs (2%), 6,943 FFs (1%) and 12 block RAMs (1%). It uses minimal resources just to maintain communication between the FPGAs and CPUs.

CPU: The CPU performance results are obtained from a 1U server that hosts two Intel Xeon X5650 CPUs. Each CPU is clocked at 2.66 GHz. The program is written in C language and optimised by Intel Compiler with SSE4.2 and flag *-fast* enabled. OpenMP is used to utilise all the processor cores.

GPU: An NVIDIA Tesla C2070 GPU is hosted inside a 4U server. It has 448 cores running at 1.15 GHz and has a peak performance by 1288 GFlops. The program is written in C for CUDA and optimised to use all the cores available. To get more comprehensive results for comparison, we also estimate the performance of multiple GPUs. The estimation is based on the fact that the first three stages (sampling, importance weighting, lower bound calculation) can be evenly distributed to every GPU and be computed independently, so the data path and data transfer speedup scales linearly with the number of GPUs. On the other hand, the last two stages (particle set resizing, resampling) are computed on the CPU no matter how many GPUs are used. Therefore, the CPU time does not scale with the number of GPUs.

4.5.2 Adaptive SMC versus Non-adaptive SMC

The comparison of adaptive and non-adaptive SMC is shown in Table 4.1. Both model estimation and experimental results are listed. Initially, the maximum number of particles are instantiated for global localisation.

For the non-adaptive scheme, the particle set size does not change. The total computation time estimated and measured are 1.328 seconds and 1.885 seconds, respectively. The measured computation time is longer due to the difference between the effective and maximum bandwidth of the PCI Express bus.

For the adaptive scheme, the number of particles varies from 573k to 67M, and the computation time scales linearly with the number of particles. From Table 4.1, both the model and experiment show 99% reduction in computation time.

Figure 4.7 shows how both the number of particles and the components of total computation time vary over the wall-clock time (passage of time from the start to the completion of the

Table 4.1: Comparison of adaptive and non-adaptive SMC on reconfigurable system (Max-Workstation with one FPGA, no data compression is applied).

	Non-adaptive SMC		Adaptive SMC	
	Model	Experiment	Model	Experiment
Number of particles	67M		573k	
Data path time $T_{datapath}$ (s)	0.336	0.336	0.003	0.003
CPU time T_{cpu} (s)	0.117	0.117	0.001	0.001
Data time T_{tran} (s)	0.875	1.432	0.007	0.012
Total computation time T_{comp} (s)	1.328	1.885	0.011	0.016
Comp. speedup (higher is better)	1x	1x	120.7x	117.8x

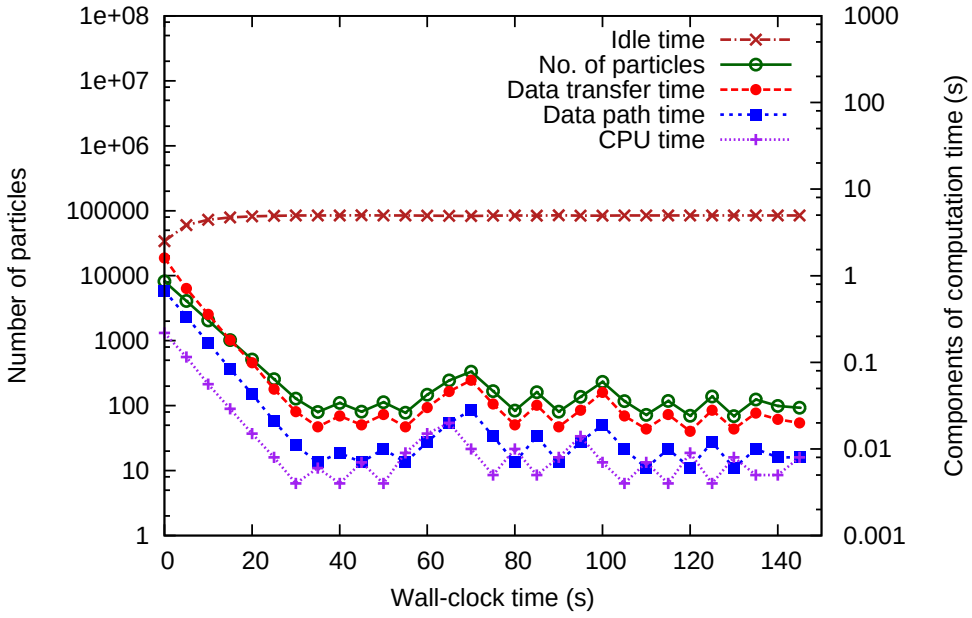


Figure 4.7: Number of particles and components of total computation time versus wall-clock time.

application). Although the number of particles is reduced in the proposed design, the results in Figure 4.8 show that the localisation error is not adversely affected. The error is the highest during initial global localisation and it is reduced when the robot moves.

4.5.3 Data Compression

Figure 4.9 shows the reduction in data transfer time after applying data compression. A higher number of replications means a lower data transfer time. The data transfer time has a lower bound of 0.212 seconds because the data from the FPGAs to the CPUs are not compressible. Only the particle stream after the resampling process is compressed when it is transferred from

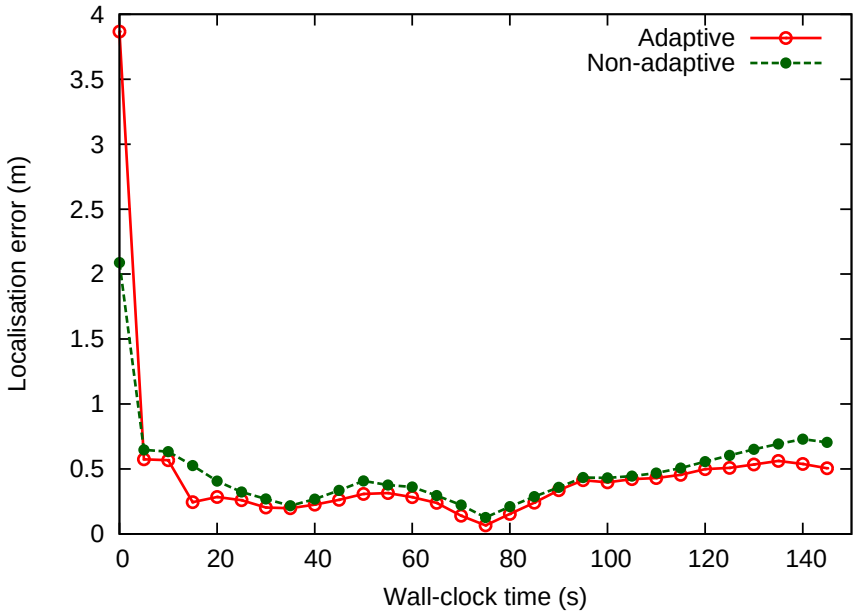


Figure 4.8: Localisation error versus wall-clock time.

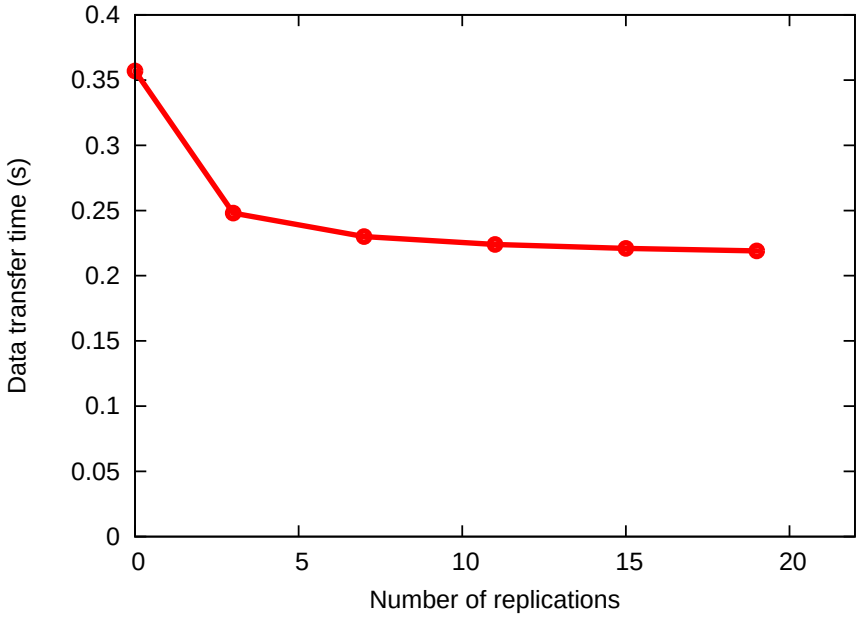


Figure 4.9: Effect on the data transfer time by particle stream compression.

the CPUs to the FPGAs.

4.5.4 Performance Comparison of Reconfigurable System, CPU and GPU

Table 4.2 shows the performance comparison of the CPUs, GPUs and reconfigurable system.

Data path time: Considering the time spent on the data paths only, the reconfigurable system is up to 328 times faster than a single-core CPU and 76 times faster than a 12-core CPU system with 24 threads. In addition, it is 12 times and 3 times faster than one GPU and four GPUs, respectively.

Data transfer time: The data transfer time of reconfigurable system is shown in three rows. The first row shows the situation when the PCI Express bandwidth is 2 GB/s. The second row shows the performance when PCI Express gen3 x8 (7.88 GB/s) is used such that the bandwidth is comparable with that of the GPU system. When multiple FPGA boards are used, the data transfer time decreases because multiple PCI Express buses are utilised simultaneously. The third row shows the performance when data compression is applied and it is assumed that each particle is replicated for 20 times in average.

CPU time: The CPU time of reconfigurable system is shorter than that of the CPU and GPU systems because part of the resampling process of object-particles is performed on the FPGA using Independent Metropolis-Hastings (IMH) resampling algorithm [118]. IMH resampling algorithm is optimised for the deep pipeline architecture where each particle occupies a single stage of the pipeline. On the CPUs and GPU, the computation of the particles are shared by threads and therefore IMH resampling algorithm is not applicable.

Total computation time: Considering the overall system performance, reconfigurable system is up to 169 times faster than a single-core CPU, 41 times faster than a 12-core CPU system. In addition, it is 9 times faster than one GPU, and 3 times faster than four GPUs. Notice that the CPUs violate the real-time constraint of 5 seconds.

Power and energy consumption: In real-time applications, we are interested in the energy consumption per time-step. Figure 4.10 shows the power consumption of reconfigurable system, CPUs and GPU over a period of 10 seconds (two time-steps). The system power is measured using a power meter which is connected directly between the power source and the system. All the curves of reconfigurable system show peaks when the system is at the computation mode and troughs when it is at the low power mode. The power during the configuration period lies between the two modes. On the reconfigurable system with one FPGA, run-time reconfiguration

Table 4.2: Performance comparison of reconfigurable system (RS), CPU and GPU.

	CPU(1) [a]	CPU(2) [a]	GPU(1) [b]	GPU(2) [b]	GPU(3) [b]	RS(1) [c]	RS(2) [d]	RS(3) [d]
Clock frequency (MHz)	2660	2660	1150	1150	1150	100	100	100
Precision	single	single	single	single	single	single + custom	single + custom	single + custom
Level of parallelism	1	24	448	896	1792	2+8 [e]	4+24 [e]	8+24 [e]
Data path time (s)	27.530	6.363	1.000	0.500	0.250	0.336	0.168	0.084
Data path speedup	1x	4.3x	27.5x	55.1x	110.1x	81.9x	163.9x	327.7x
Data transfer time (s)	0	0	0.360	0.180	0.090	1.432 [f] 0.363 [g] 0.223 [h]	0.716 [f] 0.182 [g] 0.111 [h]	0.358 [f] 0.091 [g] 0.056 [h]
CPU time (s)	0.420	0.334	0.117	0.117	0.117	0.030	0.025	0.025
Total comp. time (s)	27.95	6.697	1.477	0.797	0.457	0.589	0.304	0.165
Overall speedup	1x	4.2x	18.9x	35.1x	61.2x	47.5x	91.9x	169.4x
Computation power (W)	183	279	287	424	698	145	420	480
Computation power eff.	1x	0.7x	0.6x	0.4x	0.3x	1.3x	0.4x	0.4x
Idle power (W)	133	133	208	266	382	95	360	360
Idle power eff.	1x	1x	0.6x	0.5x	0.4x	1.4x	0.4x	0.4x
Energy. (J) [i]	677/5115	673/1868	1041/1157	1331/1456	1911/2054	489/595	1896/1914	1994/2012
Energy eff.	1x	1x/2.7x	0.7x/4.4x	0.5x/3.5x	0.4x/2.5x	1.4x/8.6x	0.4x/2.7x	0.3x/2.5x

[a] 2 Intel Xeon X5650 CPUs @2.66 GHz (12 cores supporting 24 threads).
[b] 1/2/4 NVIDIA Tesla C2070 GPUs and 1 Intel Core i7-950 CPU @3.07 GHz (4 cores supporting 8 threads).
[c] 1 Xilinx XC6VSX475T FPGA and 1 Intel Core i7-870 CPU @2.93 GHz (4 cores supporting 8 threads).
[d] 4 Xilinx XC6VSX475T FPGAs and 2 Intel Xeon X5650 CPUs @2.66 GHz (12 cores supporting 24 threads).
[e] Number of FPGA data paths and number of CPU threads.
[f] Each FPGA communicates with CPUs via a PCI Express bus with 2 GB/s bandwidth.
[g] Each FPGA communicates with CPUs via a PCI Express Gen3 x8 bus with 7.88 GB/s bandwidth.
[h] Each FPGA communicates with CPUs via a PCI Express Gen3 x8 bus with data compression.
[i] Cases for 573k and 67M particles in a 5-second interval.

reduces the idle power consumption by 34% from 145W to 95W. In other words, over a 5-second time-step, the energy consumption is reduced by up to 33%. On the reconfigurable system with four FPGAs, the idle power consumption is reduced by 25% from 480W to 360W, and hence the energy consumption decreased by up to 17%.

The run-time reconfiguration methodology is not limited to the Maxeler systems, it can be applied to other FPGA platforms. The resource management software of our system (MaxelerOS) simplifies the effort of performing run-time reconfiguration, and hence we can focus on studying the impact of run-time reconfiguration on energy saving.

To identify the speed and energy trade-off, we produce a graph as shown in Figure 4.11. Each data point represents the computation time versus energy consumption of a system setting. Among all the systems, the reconfigurable system with one FPGA has the computation speed satisfying the real-time requirement, while consuming the smallest amount of energy. All the

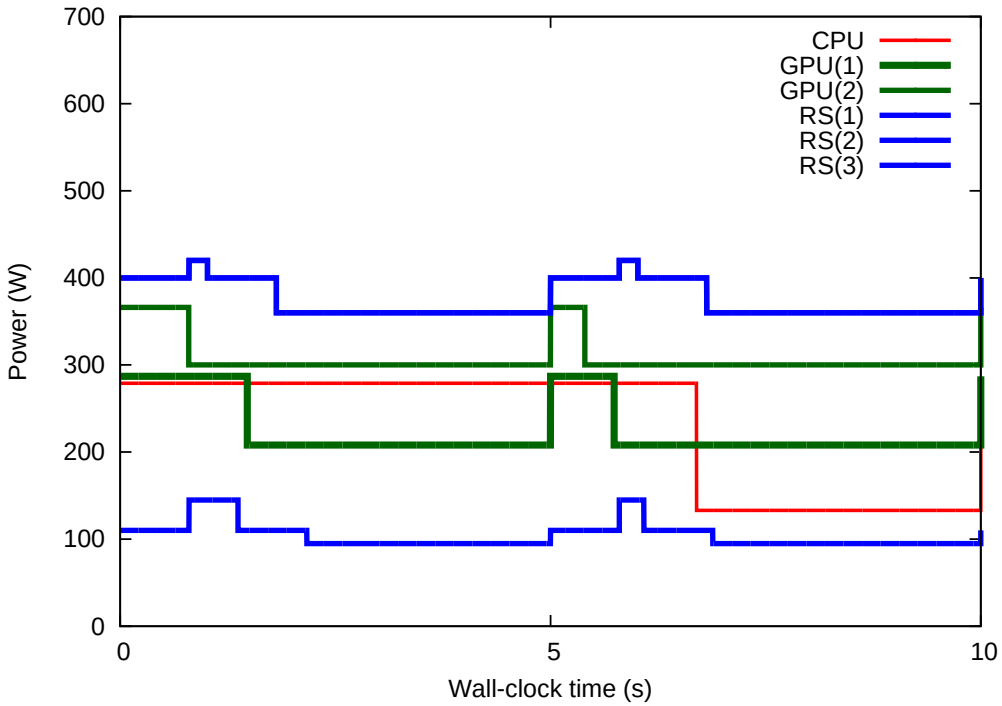


Figure 4.10: Power consumption of reconfigurable system (RS), CPU and GPU in one time-step, notice that the computation time of the CPU system exceeds the 5-second real-time requirement (The lines of RS(2) and RS(3) overlap).

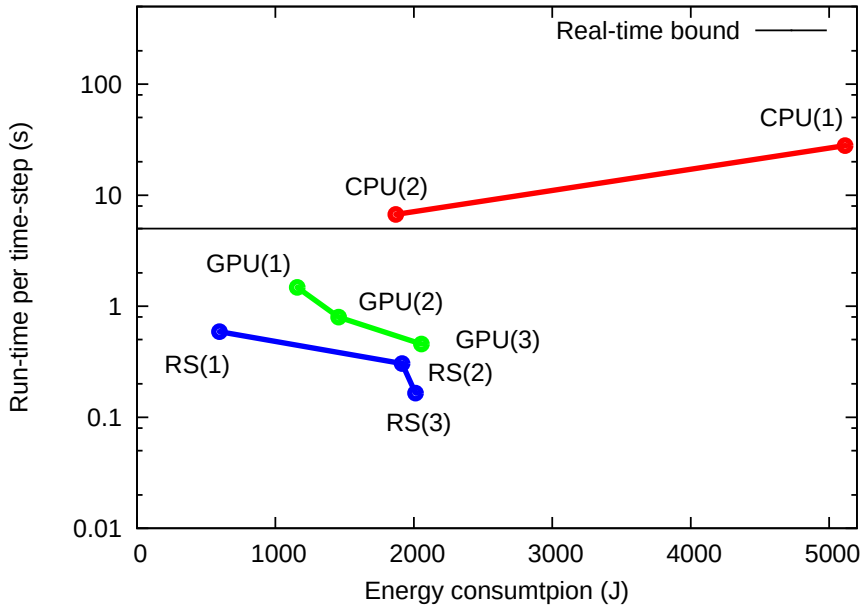


Figure 4.11: Run-time versus energy consumption of reconfigurable system (RS), CPU and GPU (5-second time-step, 67M particles; Refer to Table II for system settings).

configurations of CPU system cannot meet the real-time requirement. RS(3), the reconfigurable system with four FPGAs, is the fastest among all the systems in comparison, therefore it is able to handle larger problems and more complex applications.

4.6 Summary

This chapter presents an approach for accelerating adaptive particle filter for real-time applications. The proposed heterogeneous reconfigurable system demonstrates a significant reduction in power and energy consumption compared with CPU and GPU. The adaptive algorithm reduces computation time while maintaining the quality of results. The approach is scalable to systems with multiple FPGAs. A data compression technique is used to mitigate the data transfer overhead between the FPGAs and CPUs.

Chapter 5

Design Flow for Domain-specific Reconfigurable Applications

5.1 Introduction

In this chapter, we propose a domain-specific design flow for reconfigurable hardware, targeting SMC in particular. The main objective of this design flow is to reduce the development effort and optimise the performance of real-time SMC applications. Users can specify application-specific features which are automatically converted to efficient hardware so redesign effort is minimised. A computation engine captures the generic control structure which is shared among all SMC applications. All these features are enabled by a framework for mapping software to hardware. To enable rapid learning of a large design space, a timing model relates design parameters to performance constraints, and a machine learning algorithm is used to automatically deduce characteristics of the design space.

The contributions are as follows:

- A design flow that reduces the development effort of SMC applications on reconfigurable systems (Section 5.2). The reconfigurable system is generalised based on the one mentioned in Chapter 4. Through templating the SMC structure, users can design efficient,

multiple-FPGA SMC applications for arbitrary problems without any knowledge of reconfigurable computing. Moreover, the software template is open-source.[1]

- A machine learning approach that explores the SMC design space automatically and tunes design parameters to improve performance and accuracy (Section 5.3). The resulting parameters can be applied to the hardware design at run-time without the need for resynthesis. It is demonstrated that parameter optimisation enables the design space to be explored an order of magnitude faster without sacrificing quality. When compared with previous work [26, 33], our approach provides better quality of solutions and faster designs.

- The benefit of this approach in terms of design productivity and performance is quantified over a diverse set of SMC problems. Two applications are implemented on Altera and Xilinx-based reconfigurable platforms, with varying numbers of FPGAs. For these problems, the number of lines of code for the FPGA implementation is reduced by approximately 76%, and significant speedup and energy improvement over CPU and GPU implementations (Section 5.4) are demonstrated.

The rest of the chapter is organised as follows. Section 5.2 describes the design flow for generating reconfigurable SMC designs. It covers the software template, the computation engine and the performance model. Section 5.3 discusses how the SMC computation engine can be optimised both at compile-time and run-time. Section 5.4 evaluates the design flow using two different SMC applications and Section 4.6 concludes our work.

5.2 SMC Design Flow

This section introduces a design flow for generating reconfigurable SMC designs. The design flow has two novel features to minimise hardware redesign efforts: (1) A generic high-level mapping where application-specific features are specified in a software template and automatically converted to hardware. The template supports the parameter optimisation described in

[1] Available online: `http://cc.doc.ic.ac.uk/projects/smcgen`

Section 5.3. (2) A parametrisable SMC computation engine which is made up of customisable building blocks and generic control structure that maximises design reuse. We will start with three high-level stages as shown in Figure 5.1, and look into the features as we go through this section.

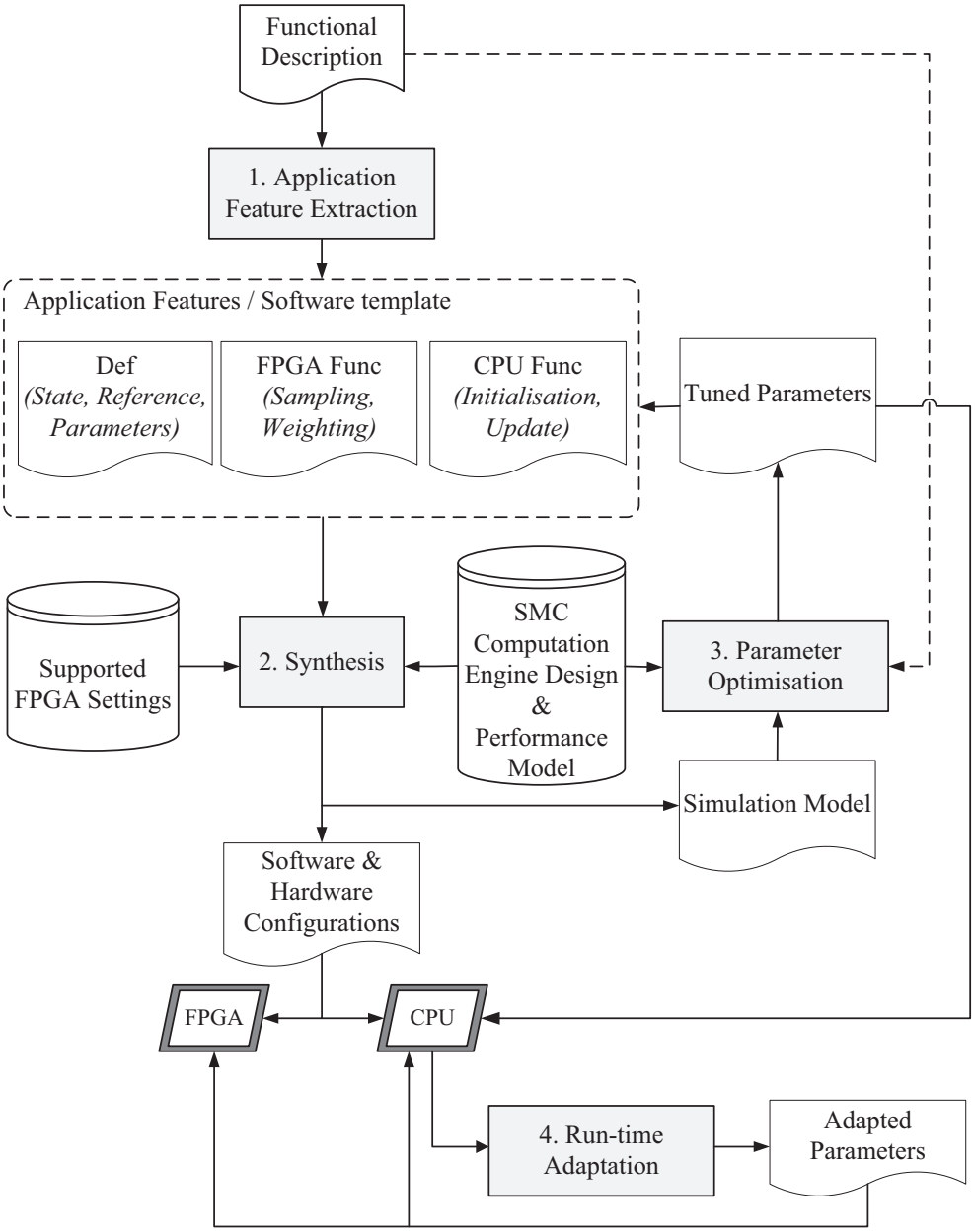


Figure 5.1: Design flow (Compile-time and run-time) for SMC applications. Users only customise the application-specific descriptions inside the dotted box.

Figure 5.1 shows the proposed design flow:

1. Starting with a functional description such as a software code or a mathematical formu-

lation, the users identify and code application-specific features (Section 5.2.1). Generally only the application-specific features are of interest, other features which are common to all SMC applications are handled by the design flow, so the functional description does not necessarily have to be a complete software code.

In this work the synthesis tool employed is Maxeler's MaxCompiler, which uses Java as the underlying language. MaxCompiler also supports FPGAs from multiple vendors, such that low level configurations, such as I/O binding, are performed automatically. Our approach can be extended to support other tools and devices, for example by having the appropriate templates in VHDL or Verilog.

2. The synthesis step automatically weaves the application-specific features with the computation engine (Section 5.2.2) to form a performance model (Section 5.2.3), a simulation model, and a complete configuration for the targeted reconfigurable system.

3. The design flow also consists of a parameter optimisation step (Section 5.3) which takes the simulation model and performance model as inputs to produce a set of performance or accuracy optimised parameters. Generally a simulation model is sufficient for performing optimisation, if a complete software code is provided, it can be used to accelerate the optimisation process.

4. The design of SMC computation engine allows further adaptation of design at run-time. The adaptation is based on the solution quality. For example, a better solution quality means that fewer particles could be used for performing SMC, and vice versa.

5.2.1 Specifying Application Features

Users create a new SMC design by customising the application-specific Java descriptions inside the dotted box of Figure 5.1. These descriptions correspond to *Def* (Code 1), *FPGA Func* (Code 2) and *CPU Func*.

Def: Code 1 illustrates the class where number representation (floating-point, fixed-point with different bit-width), structs (state, reference), static parameters (Table 2.1) and system

parameters are defined. Users are allowed to customise number representation to benefit from the flexibility of FPGA and make trade-off between accuracy and design complexity. State and reference structs determine the I/O interface. Static parameters are defined in this class, while dynamic parameters are provided at run-time. System parameters define device-specific properties such as clock speed and parallelism. Lastly, application parameters define properties that are tied to specific applications.

FPGA Func: *Sampling and importance weighting* are the most computation intensive functions, and are accelerated by FPGAs. Code 2 gives a simple example on how these two FPGA functions are defined. Given current state *s_in*, reference *r_in* and observation *m_in* (sensor values in this example), an estimation state *s_out* is computed. Weight w accounts for the probability of an observation from the estimated state. The weight is calculated from the product of scores over the horizon. In this example, the weight is equal to the score as the horizon length is one.

CPU Func: *Initialisation and update* are functions running on the CPU. They are responsible for obtaining and formatting data and displaying results. *Resampling* is independent of applications so users need not to customise it.

5.2.2 Computation Engine Design

In Chapter 4, a heterogeneous reconfigurable system has been designed for accelerating SMC applications. In this section, the system is extended to improve flexibility in terms of customisability and design friendliness.

To allow customisation of the computation engine, the engine and data structure are designed as shown in Figure 5.2(a) and 5.2(b) respectively. The computation engine employs a heterogeneous structure that consists of multiple FPGAs and CPUs. FPGAs are responsible for sampling, importance weighting and optionally resampling index generation, and are fully pipelined to maximise throughput. To exploit parallelism, particle simulations (sampling and importance weighting) are computed simultaneously by every processing core on each FPGA.

```
public class Def {
  // Number Representation
  static final DFEType float_t =
    KernelLib.dfeFloat(8,24);
  static final DFEType fixed_t =
    KernelLib.dfeFixOffset(26,-20,SignMode.TWOSCOMPLEMENT);
  // State Struct
  public static final DFEStructType state_t = new
  DFEStructType(
    new StructFieldType(''x'', float_t);
    new StructFieldType(''y'', float_t);
    new StructFieldType(''h'', float_t););
  // Reference Struct
  public static final DFEStructType ref_t = new
  DFEStructType(
    new StructFieldType(''d'', float_t);
    new StructFieldType(''r'', float_t););
  // Static Design parameters (Table I)
  public static int NPMin = 5000, NPMax = 25000;
  public static int H = 1, NA = 1;
  // System Parameters
  public static int NC_inner = 1, NC_P = 2;
  public static int Clk_core = 120, Clk_mem = 350;
  public static int FPGA_resampling = 0, Use_DRAM = 0;
  // Application parameters
  public static int NWall = 8, NSensor = 20;
}
```

Code 1: State, control and parameters for the robot localisation example.

Processing cores can be replicated as many times as FPGA resources allow. In situation where the computed results have to be grouped together, data are transferred among FPGAs via an inter-FPGA connection. To maximise the system throughput, remaining non-compute-intensive tasks that involve random and non-sequential data accesses are performed on the CPUs. FPGAs and CPUs communicate through high bandwidth connections such as PCI Express or InfiniBand.

From the control paths (dotted lines) of Figure 5.2(a), we see that there are three loops: (1) inner, (2) outer, and (3) time step. First, the inner loop iterates *itl_inner* number of times for *sampling* and *importance weighting*, *itl_inner* increases with the iteration count of the outer loop. Second, the outer loop iterates *itl_outer* times to do *resampling*. The resampling process is performed *itl_outer* times to refine the pool of particles. The particle indices are scrambled after this stage and the indices are transferred to the CPUs to update the particles. Third,

```
28 public class Func {
29   public static DFEStruct sampling(
30     DFEStruct s_in, DFEStruct c_in){
31     DFEStruct s_out = state_t.newInstance(this);
32     s_out.x = s_in.x + nrand(c_in.d,S*0.5) * cos(s_in.h);
33     s_out.y = s_in.y + nrand(c_in.d,S*0.5) * sin(s_in.h);
34     s_out.h = s_in.h + nrand(c_in.r,S*0.1);
35     return s_out;
36   }
37   public static DFEVar weighting(
38     DFEStruct s_in, DFEVar sensor){
39     // Score calculation
40     DFEVar score = exp(-1*pow(est(s_in)-sensor,2)/S/0.5);
41     // Constraint handling
42     bool succeed = est(s_in)>0 ? true : false;
43     // Weight accumulation
44     DFEVar w = succeed ? score : 0; //weight
45     return w;
46   }
47 }
```

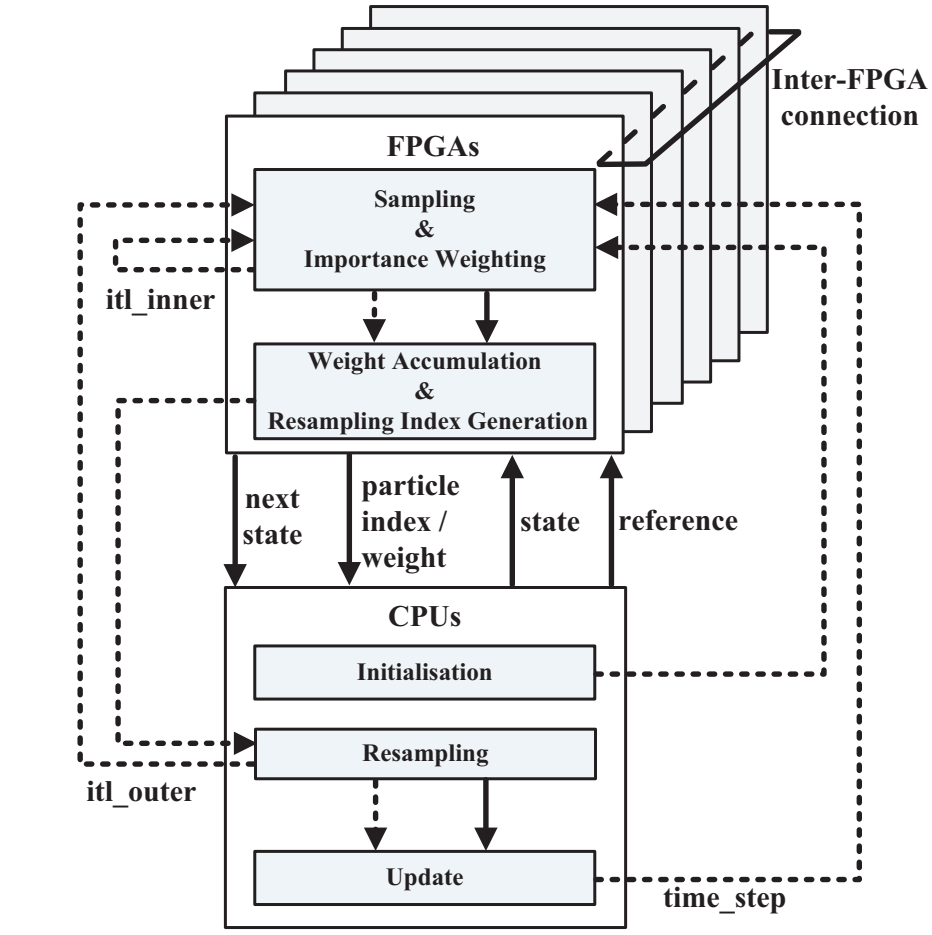
Code 2: FPGA functions (Sampling and importance weighting) for the robot localisation example.

the time loop iterates once per time step to obtain a new control strategy and to update the current state.

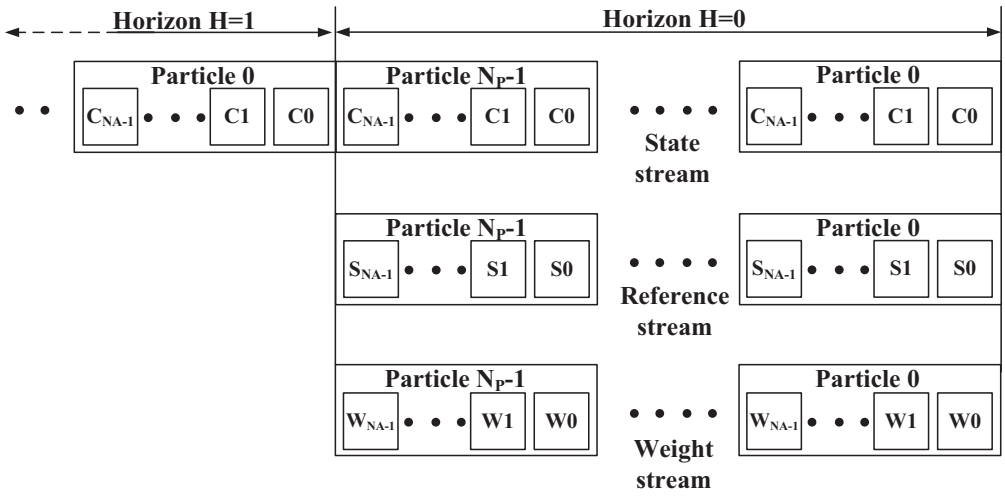
Based on this fact, the data structure shown in Figure 5.2(b) is derived. Each particle encapsulates three pieces of information: (1) state, (2) reference, and (3) weight, each being stored as a stream as indicated in the figure. The length of the *state stream* is $N_P \cdot N_A \cdot H$ where H means each control strategy predicts H steps into the future. The *reference* and *weight streams* have information of N_A agents in N_P particles.

The engine design and data structure do not only offer compile-time parametrisation, but also allow changing the values of *itl_outer*, *itl_inner* and N_P at run-time. It is because these parameters only affect the length of the particle streams, but not the hardware data path. The computation engine is fully pipelined and outputs one result per clock cycle.

Figure 5.3 shows the design of the FPGA kernel. Blocks that require customisation are darkened. The sampling function in Code 2 is mapped to the **Sampling** block which accepts a state and a reference on each clock cycle and calculates the next state on the prediction horizon.



(a)



(b)

Figure 5.2: (a) Design of the SMC computation engine: Solid lines represent data paths while dotted lines represent control paths; (b) Data structure of particles represented by three data streams.

After getting a state from the CPU at the beginning ($itl\_inner = 0$ and $H = 0$), the data will be used by the kernel $itl\_inner \cdot N_P$ times. An optional *state RAM* enables reuse of state data

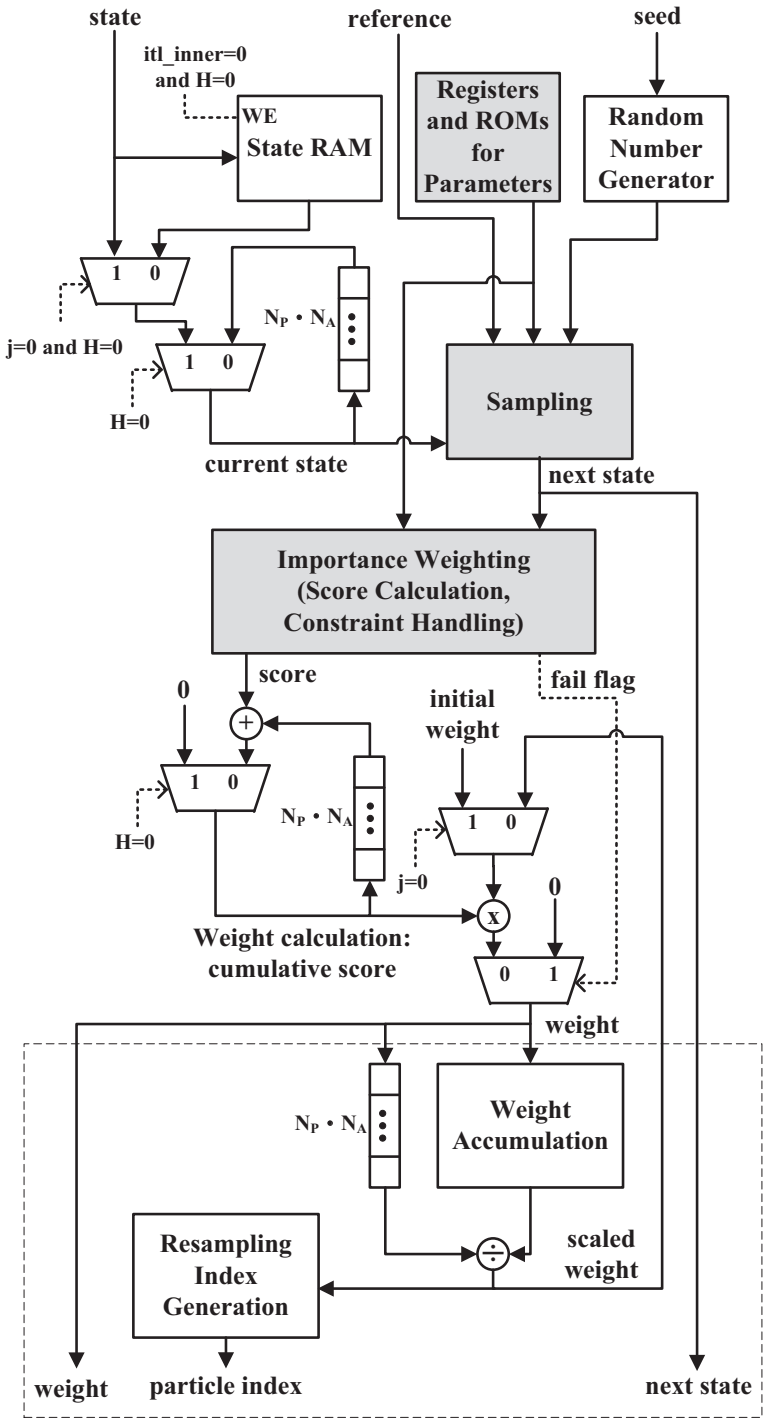


Figure 5.3: FPGA kernel design: The blocks that require users' customisation are darkened. The dotted box covers the blocks that are optional on FPGAs.

and improves performance when the value of *itl_inner* is large. An array of LUT-based random number generators [119, 120] is seeded by CPU to provide random variables; application parameters are stored in registers; and a feedback path stores the state of the previous $N_P \cdot N_A$ cycles.

The **Importance weighting** block computes in three steps. Firstly, *Score calculation* uses the states from the *Next state* block to calculate scores of all the states over the horizon. A feedback loop of length $N_P \cdot N_A$ stores the cost of the previous horizon and accumulates the values. Secondly, *Constraint handling* uses the states from the *Next state* block to check the constraints. The block raises a fail flag if a constraint is violated. Lastly, *Weight calculation* combines the scores of the states over the horizon.

Part of the resampling process is handled by the **Resampling index generation** and **weight accumulation** blocks. Weights are accumulated to calculate the cumulative distribution function, then particles indices are reordered. These two blocks can either be computed on FPGAs or CPUs.

All the blocks allow precision customisation using fixed-point or floating-point number representation. Users have the flexibility to make trade-off between result accuracy and design complexity.

5.2.3 Performance Model

We derive a performance model to analyse the effect of parameters on the processing speed as well as resource utilisation of the computation engine. It will be used in Section 5.3 for parameter optimisation.

The processing time of a time step is shown in Equation 5.1. It has four components which are iterated *itl_outer* times.

$$T_{step} = itl\_outer \cdot (T_{s\&i} + T_{resample} + T_{cpu} + T_{tran}) \,. \tag{5.1}$$

$T_{s\&i}$ is the time spent on sampling and importance weighting in the FPGA kernels.

$$T_{s\&i} = \frac{itl\_inner \cdot N_P \cdot N_A \cdot H}{N_C \cdot N_{Board} \cdot freq} \cdot \min\left(1, \frac{bandwidth}{sizeof(state) \cdot freq}\right). \tag{5.2}$$

Since the data is organised as a stream as described in Section 5.2.2, the time spent on sampling and importance weighting is linear with N_P, N_A and H. It is iterated *itl_inner* times in the inner loop. The sampling and importance weighting process can be accelerated using multiple cores, such that each of them is responsible for part of the inner loop iterations or particles. N_C represents the number of processing cores being used on one FPGA, and N_{Board} is the number of FPGA boards being used. $min(1, \frac{bandwidth}{sizeof(state) \cdot freq})$ accounts for the limitation of bandwidth between FPGAs and CPUs.

$T_{resample}$ is the time spent on generating the resampling indices.

$$T_{resample} = \frac{N_P \cdot PW + N_P \cdot N_A + 3 \cdot PL \cdot N_P}{freq}. \tag{5.3}$$

It takes $N_P \cdot PW + N_P \cdot N_A$ cycles to generate the cumulative probability distribution function, and a further $3 \cdot PL \cdot N_P$ cycles to generate particle indices. PW and PL are the length of the pipelines. $T_{resample}$ can be omitted if resampling is processed by the CPUs.

T_{cpu} is the time spent on resampling and updating the current state on the CPUs.

$$T_{cpu} = \alpha_1 \cdot H \cdot N_P \cdot N_A. \tag{5.4}$$

The time is related to the amount of data and the speed of the CPU. α_1 is the scaling factor of the CPU speed.

T_{tran} is the data transfer time that accounts for the time taken to transfer the state stream between CPUs and DRAM on an FPGA board. T_{tran} can be omitted if no DRAM is used.

$$T_{tran} = \frac{N_P \cdot N_A \cdot (H \cdot sizeof(state))}{bandwidth}. \tag{5.5}$$

5.3 Optimising SMC Computation Engine

The design parameters in Table 2.1 have great impact on the performance. Three questions manifest when finding optimised customisation of the engine: **(1) Which sets of parameters have the best accuracy? (2) For the same accuracy, which sets of parameters meet the timing requirement? (3) How can we reduce the design parameter exploration time?** This section discusses some techniques about parameter optimisation.

5.3.1 Compile-time Parameters

Referring to Table 2.1 in Chapter 2, the SMC computation engine has up to six design parameters, each of which adds a dimension to the design space. It is ineffective to exhaustively search for the best set of parameters. Furthermore, the performance curve of each dimension can be non-linear and constrained by both the real-time requirement and FPGA resources.

To answer **questions 1 and 2**, consider the robot localisation application. Its solution quality is measured by Root-Mean-Square Error (RMSE) in localisation [94]. We study the effect of changing design parameters using the functional specification in Figure 5.1, e.g. a C program. Software functional specification has fast build time, and it helps us to perform analysis effectively. To meet real-time operation requirement, software functional specification is too slow without acceleration of the SMC computation engine. The run time of the computation engine is estimated by the timing model described in Section 5.2.3.

When N_P and *itl_outer* are explored together as shown in Figure 5.4, we see an uneven surface. Although non-linear, it is evident that RMSE decreases as N_P and *itl_outer* increase. The valid parameter space is constrained by the real-time requirement: the parameter space is darkened for those parameters leading to an RMSE greater than 1 m (Question 1); the dark region with a run-time longer than the 5 seconds real-time requirement is marked as invalid (Question 2).

If the value of S (scaling factor for the standard deviation of noise) is also considered, the parameter optimisation problem expands to three dimensions as shown in Equation 5.6:

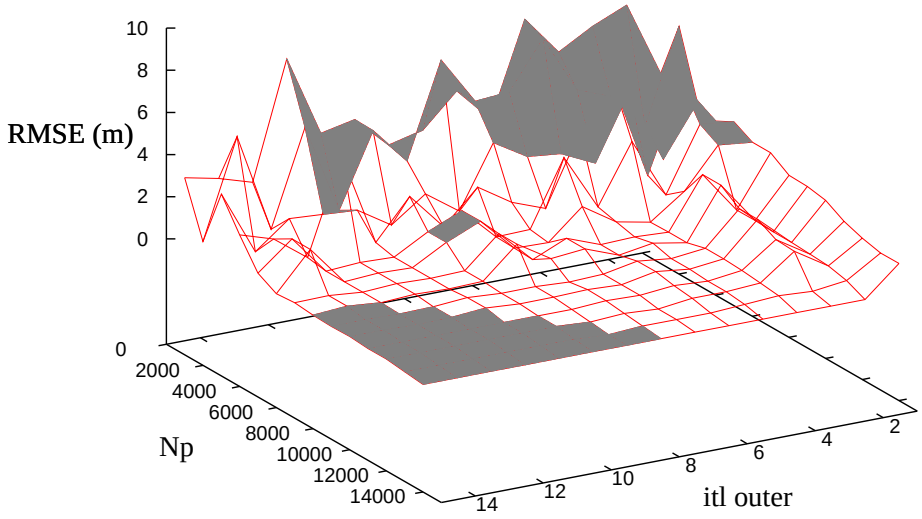


Figure 5.4: Parameter space of robot localisation system (N_A=8192, S=1): The dark region on the top-right indicates designs which fail localisation accuracy constraints, while those on the bottom-left indicates designs which fail real-time requirements.

$$\begin{aligned} &\text{minimise } RMSE = f(N_P, itl\_outer, S), \\ &\text{subject to } RMSE \leq 1 \text{ m}, T_{step} \leq 5\text{s}, . \end{aligned} \tag{5.6}$$

5.3.2 Run-time Parameters

In Chapter 4, we proposed an algorithm which changes the number of particles based on run-time condition. The computation workload decreases with the number of particles and hence introduces an idle period between the finishing time of computation and the end of real-time interval. The power consumption during the idle period is reduced by reconfigurating the FPGAs to low-power mode, where the FPGAs runs at a lower frequency and all the unnecessary features are disabled or removed. The run-time reconfiguration and parameter adaptation are applied to the proposed SMC computation engine in this chapter. For convenience, Algorithm 4 recaptures the approach, with modifications made to cope with the generalised computation engine. In particular, an inner loop *itl_inner* is included to deal with multiple iteration of sampling and importance within a time step. P_{t+1} describes the lower bound of particle size which is used in Algorithm 4:

$$\widetilde{P}_{t+1} = \sigma^2 \cdot \frac{P_{max}}{Var(\{\widetilde{\chi}_{t+1}^{(i)}\}_{i=1}^{P_t})}. \tag{5.7}$$

Algorithm 4 Adaptive SMC algorithm

$P_0 \leftarrow P_{max}$
$\{X_0^{(i)}\}_{i=1}^{P_0} \leftarrow$random set of particles
$t \leftarrow 1$
for each step t **do**
  $idx1 \leftarrow 0$
  Initialisation
  while $idx1 \leq itl\_outer$ **do**
    $idx2 \leftarrow 0$
    $itl\_inner \leftarrow f(idx1)$
    —On FPGAs—
    while $idx2 \leq itl\_inner$ **do**
      Sample a new state $\{\widetilde{\chi}_{t+1}^{(i)}\}_{i=1}^{P_t}$ from $\{\chi_t^{(i)}\}_{i=1}^{P_t}$
      Calculate unnormalised importance weights $\{\widetilde{w}^{(i)}\}_{i=1}^{P_t}$ and accumulate the weights as w_{sum}
      $idx2 \leftarrow idx2 + 1$
    end while
    Calculate the lower bound of sample size $\widetilde{P}_{t+1}$ by Equation 5.7
    —On CPUs—
    Sort $\{\widetilde{\chi}_{t+1}^{(i)}\}_{i=1}^{P_t}$ in descending $\{\widetilde{w}^{(i)}\}_{i=1}^{P_t}$
    if $\widetilde{P}_{t+1} < P_t$ **then**
      $P_{t+1} = max\left(\lceil \widetilde{P}_{t+1} \rceil, P_t/2\right)$
      Set $a = 2P_{t+1} - P_t$ and $b = P_{t+1}$
      –Do the following loop in parallel–
      for i in $P_t - P_{t+1}$ **do**
        $\widetilde{\chi}_{t+1}^{(i)} = \frac{\chi_{t+1}^{(a)}\widetilde{w}^{(a)} + \chi_{t+1}^{(b)}\widetilde{w}^{(b)}}{\widetilde{w}^{(a)} + \widetilde{w}^{(b)}}$
        $\widetilde{w}^{(i)} = \widetilde{w}^{(a)} + \widetilde{w}^{(b)}$
        $a = a + 1$ and $b = b - 1$
      end for
    else if $\widetilde{P}_{t+1} \geq P_t$ **then**
      $a = 0$ and $b = 0$
      for i in $P_{t+1} - P_t$ **do**
        if $\widetilde{w}^{(a)} < \widetilde{w}^{(a+1)}$ and $a < P_{t+1}$ **then**
          $a = a + 1$
        end if
        $\widetilde{\chi}_{t+1}^{(P_t+b)} = \widetilde{\chi}_{t+1}^{(a)}/2$
        $\widetilde{\chi}_{t+1}^{(a)} = \widetilde{\chi}_{t+1}^{(a)}/2$
        $\widetilde{w}^{(P_t+b)} = \widetilde{w}^{(a)}/2$
        $\widetilde{w}^{(a)} = \widetilde{w}^{(a)}/2$
        $b = b + 1$
      end for
    end if
    $idx1 \leftarrow idx1 + 1$
    if $idx1 \leq itl\_inner$ **then**
      Resample $\{\widetilde{\chi}_{t+1}^{(i)}\}_{i=1}^{P_t}$ to $\{\chi_{t+1}^{(i)}\}_{i=1}^{P_{t+1}}$
    end if
  end while
  Update
end for

Figure 5.5 illustrates the effect of adapting parameters at run-time. Power consumption is reduced by reconfiguring the FPGAs to sleep mode, at the expense of reconfiguration overhead.

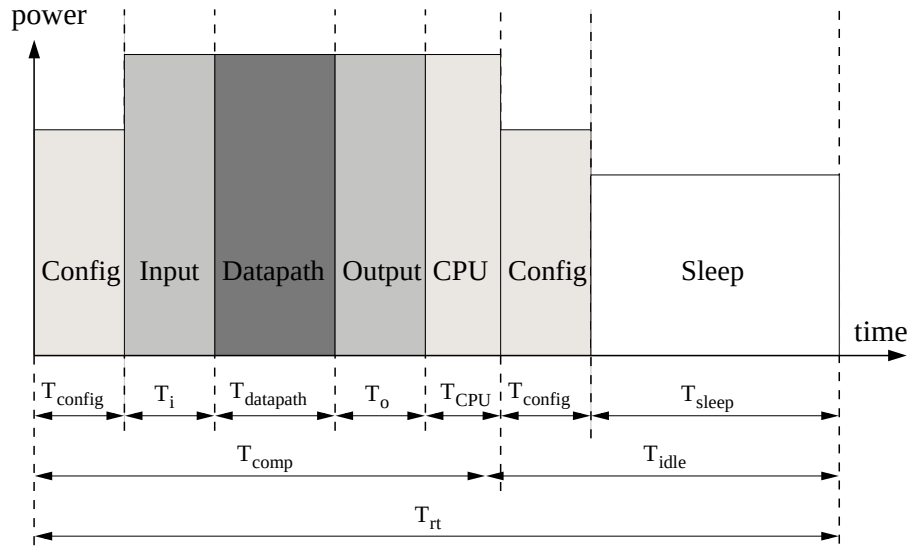


Figure 5.5: Power consumption of the reconfigurable system with reconfiguration to low-power mode during idle

5.3.3 Parameter Optimisation

Now we come to **question 3**, the parameter optimisation problem, which is difficult as construction of an analytical model combining timing and quality of solution is either impossible or very time consuming. Furthermore the design space is constrained by multiple accuracy and real-time requirements. We cannot use a design unless the results are within certain error bound. The problem is further aggravated by *the curse of dimensionality*. We use an automated design exploration approach which is facilitated by a machine learning algorithm developed in [36]. The approach allows the performance impact of different parameters to be determined for any design based on our SMC computation engine.

A surrogate model is employed to enable rapid learning of the valid design space and to deal with a large number of parameters. The idea is illustrated in Figure 5.6. Firstly, a number of randomly sampled designs is evaluated (Figure 5.6(a)). Secondly, the results obtained during evaluations are used to build a surrogate model. The model provides a regression of a fitness function and identifies regions of the parameter space which fail any of the constraints (Figure 5.6(b)). Thirdly, the surrogate model output is used to calculate the expected improvement (Figure 5.6(c)). Finally, the exploration converges to the parameter set that is expected to offer the highest improvement. Parameter sets in the invalid region are disqualified (Figure 5.6(d)).

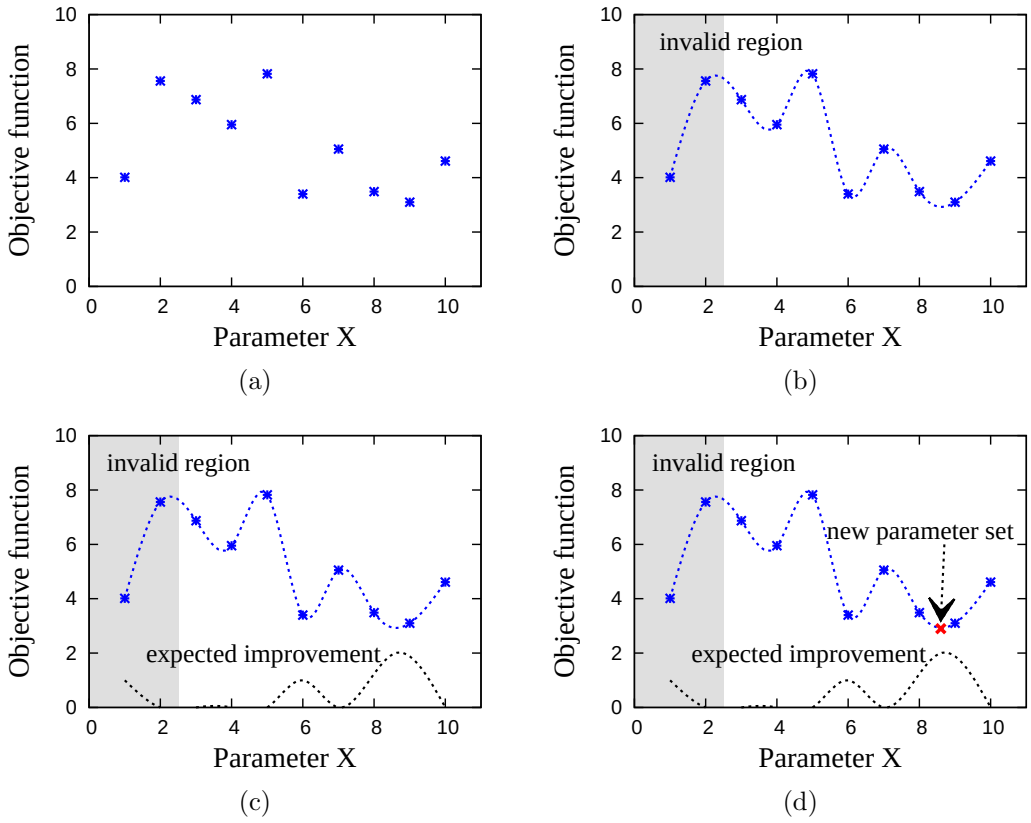


Figure 5.6: Illustration of automatic parameter optimisation: (a) Sampling parameter sets; (b) Building surrogate model; (c) Calculating expected improvement; (d) Moving to the point offering the highest improvement.

Our SMC computation engine is made customisable to benefit from this optimisation approach which is also applicable to CPUs and GPUs.

5.4 Evaluation

5.4.1 Design Productivity

We first analyse how the proposed design flow can reduce design effort. In Table 5.1, user-customisable code is classified into three parts: (a) *Def* is the definition of state, reference and parameters. (b) *FPGA Func* is the description of sampling and importance weighting functions. (c) *CPU Func* is the initiation, resampling and update part running on CPU. On average, users only need to customise 24% of the source code. Moreover, automatic design space optimisation greatly saves the overall design time. As we will see in the applications below, we are able to

choose the optimal set of parameters without conducting an exhaustive search.

Table 5.1: Lines of code for two SMC applications under the proposed design flow.

	Custom code			All code	Custom %
	Def	FPGA Func	CPU Func		
Robot loc.	54	143	56	1,113	22.7
Air traffic	45	360	70	1,360	35.0

5.4.2 Application 1: Mobile Robot Localisation

Our design flow is used in targeting a robot localisation application to a Xilinx Virtex-6 XC6VSX475T FPGA. Two processing cores clocked at 120 MHz are instantiated in the FPGA. Core computation in the sampling and importance weighting process is implemented using fixed-point arithmetic to optimise resource usage. The implementation utilises 148,431 LUTs (50%), 1,278 DSPs (63%) and 549 block RAMs (26%).

The design space has three dimensions: itl_outer, N_P and S. Out of 945 sets of parameters, 52 sets are evaluated to minimise the localisation error within the 5 seconds real-time constraint.

Table 5.2 compares the performance of our reconfigurable system with CPU, GPU and a previous system in [26] which has not been optimised by our proposed approach. With parameter tuning that maximises accuracy, our work achieves a better RMSE than the previous work (0.15m vs. 0.52m). In other words, parameter tuning improves accuracy by 3.5 times. GPU is also optimised using the same set of parameters, but it consumes double the power of our reconfigurable system. Compared to CPU, FPGA is 24 times more accurate. It is because CPU has lower performance, and a different set of parameters is applied to meet the 5 seconds real-time requirement at an expense of accuracy.

In this application, the number of particles are adapted to the run-time environment. Figure 5.7 shows the effect of the number of particles on the breakdown of computation time. The largest amount of particles are used to determine the robot's initial location (known as global localisation). Then the number of particles needed decreases sharply, only a small amount of particles are used to keep tracking the robot's movement. Reduction in the number of particles

Table 5.2: Performance comparison of robot localisation.

	CPU opt. [a]	This work opt. [b]	Ref. sys. [26] w/o opt. [b]	GPU opt. [c]
Clock frequency (MHz)	2,930	120	100	1,150
Number of cores	4	2	2	448
Run-time / step (s)	5.0	3.7	1.6	4.5
RMSE (m)	3.64	0.15	0.52	0.15
Power (W)	130	145	145	287

[a] Intel Core i7 870 CPU, optimised by Intel Compiler with SSE4.2 and flag *-fast* enabled.
[b] Maxeler MaxWorkstation with Xilinx Virtex-6 XC6VSX475T FPGA and Intel Core i7 870 CPU, developed using MaxCompiler.
[c] NVIDIA Tesla C2070 GPU, developed using CUDA programming model.
[d] Parameters with optimisation for FPGA and GPU: itl_outer=2, N_P=14000, S=1.2;
Parameters with optimisation for CPU: itl_outer=1, N_P=3000, S=1;
Parameters without optimisation: itl_outer=1, N_P=8192, S=1.

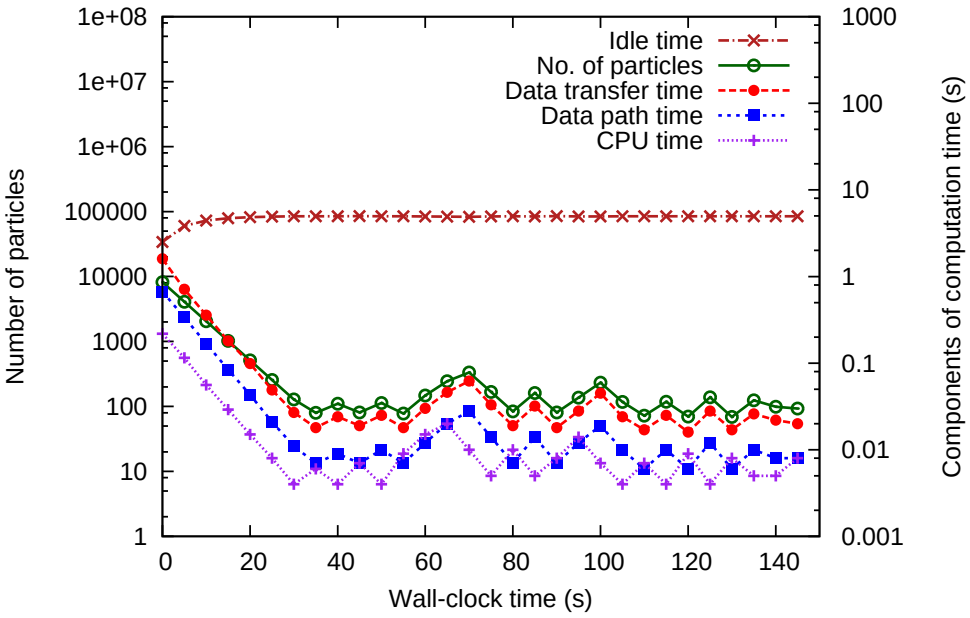


Figure 5.7: Number of particles and components of total computation time versus wall-clock time

implies decrease in the computation time, and hence results in a longer idle time when the FPGA runs in low-power mode.

The effect of running the FPGA in low-power mode is shown in Figure 5.8. The power of FPGA peaks at 135W when in compute-mode and drops to 95W when in idle-mode. Short periods of 110W are observed when the FPGA is switching between the two modes. The power of CPU and GPU are also shown in the figure.

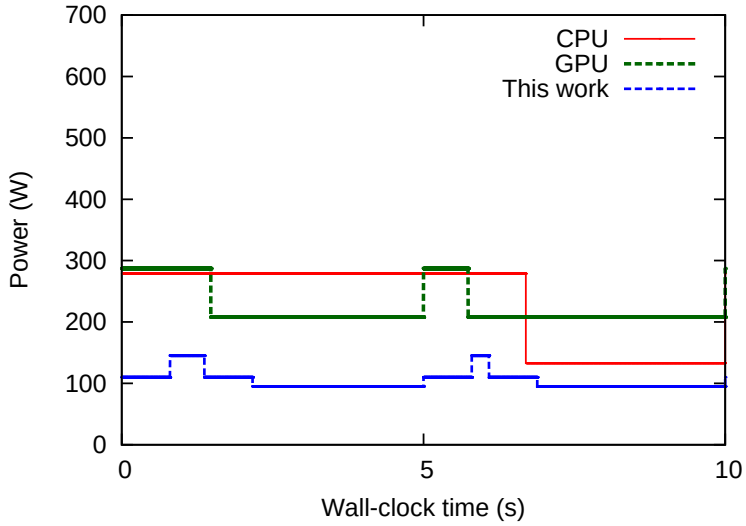


Figure 5.8: Power consumption of reconfigurable system, CPU and GPU in one time-step

5.4.3 Application 2: Air Traffic Management

The air traffic management system is able to control 20 aircraft simultaneously. The FPGA part runs on a 1U machine hosting 6 Altera Stratix V GS 5SGSD8 FPGAs clocked at 220 MHz, each of which has a single precision floating-point data path that consumes 166,008 LUTs (63%), 337 multipliers (9%) and 1,528 block RAMs (60%). The CPU part runs on 2 Intel Xeon E5-2640 CPUs clocked at 2.53GHz. Both parts are connected via InfiniBand.

This application has four design parameters leading to a space with 4000 sets of parameters. The optimisation target is to minimise the time of aircraft spending in the air traffic control region, i.e. the number of time steps required for all aircraft to reach their destinations. Each time step is subject to a real-time requirement of 30 seconds. The machine learning approach reduces the number of evaluations to 1% as indicated in Table 5.3. Hence, the parameter optimisation time is reduced from days to hours.

Table 5.3: Parameter optimisation of air traffic management system using machine learning approach.

N_A	Parameter sets Evaluated / Total	Parameter set obtained			
		itl_outer	H	N_P	S
4	41 / 4000	20	5	500	0.1
20	31 / 4000	100	8	5000	0.05

Table 5.4 summarises the performance of the CPU, GPU and reconfigurable system. To ensure

Table 5.4: Performance comparison of air traffic management.

		CPU w/ opt. [a]	GPU w/ opt. [b]	This work w/ opt. [c]	Ref. FPGA [33] w/o opt. [d]
	Clock frequency (MHz)	2,660	1,150	220	150
	Number of cores	24	1,792	6	5
	Power (W)	550	1100	600	N/A
4 aircraft	Run-time / step (s)	0.80	0.12	0.03	2.2
	Total steps	25	25	25	25
20 aircraft	Run-time / step (s)	Failed	28.25	11.6	N/A
	Total steps	Failed	41	41	N/A

[a] 4 Intel Xeon X5650 CPUs (scaled), optimised by Intel Compiler with SSE4.2 and flag *-fast* enabled.
[b] 4 NVIDIA Tesla C2070 GPUs (scaled), developed using CUDA programming model.
[c] Maxeler MPC-X2000, with 6 Altera Stratix V GS 5SGSD8 FPGAs and 2 Intel Xeon X5650 CPUs, developed using MaxCompiler.
[d] Altera Stratix IV EP4SGX530 FPGA.
[e] Parameters with optimisation: refer to Table 5.3;
Parameters without optimisation: itl_outer=100, N_P=1024, S=0.05, H=6.

fair comparisons, we scale the CPU and GPU systems to similar form factors with the reconfigurable system. The scaling is based on the fact that the sampling and importance weighting process is evenly distributed to every GPU and computed independently, while the resampling process is computed on the CPU no matter how many GPUs are used. The reconfigurable platform is faster and more energy efficient than the other systems.

In the case with 4 aircraft, all systems are able to finish with the minimal number of steps without violating the real-time requirement of 30 seconds per step. However, for the case with 20 aircraft, CPU fails to obtain a parameter set which gives a valid solution within 30 seconds.

We also compare the performance of our work with a reference implementation that uses an Altera Stratix IV FPGA [33]. That implementation is only large enough to support 4 aircraft and it does not have the flexibility to tune parameters without re-compilation. Our design exploration approach is able to select the set of parameters that produces the same quality of results and is up to 73 times faster.

5.5 Summary

This chapter demonstrates the feasibility of generating highly-optimised reconfigurable designs for SMC applications, while hiding detailed implementation aspects from the user. A software template makes the computation engine portable and facilitates code reuse, the number of lines of user-written code being decreased by approximately 76% for an application. We further establish that a surrogate software model combined with machine learning can be used to rapidly optimise designs, reducing optimisation time from days to hours; and that the resulting parameters can be utilised without resynthesis.

Chapter 6

Conclusion

This thesis has described three contributions that enable more effective and efficient implementation of high-performance real-time applications on reconfigurable systems. In this concluding chapter, we recap the key challenges and provide a summary of individual contributions and the significance of each. Then we will describe the current limitations of this thesis and suggest future research directions.

6.1 Summary of Achievements

An FPGA contains numerous prefabricated logic and routing resources which allow the functionality and interconnection to be reconfigured. Many modern FPGAs have a high level of integration with coarse grained components such as DSPs, memory blocks, high-throughput transceivers, I/O peripheral devices, customisable IP blocks and micro-processor cores. Benefiting from the reconfigurability and abundance of computation resource, FPGAs have been increasingly adopted to designs with high performance requirements. FPGAs' deterministic performance also makes them preferable over CPUs and GPUs in real-time systems. However, FPGAs are restricted in their floating-point computation capability and ability to design in mainstream programming languages. In addition, the use of FPGA for real-time applications still lacks focus on high-performance computing capability. This thesis works toward three key

areas to address the above mentioned challenges.

The computation capability of FPGAs is restricted by the number of logic components available. Chapter 3 discusses how we take advantage of FPGAs' programmability to fit more floating-point operators to an FPGA chip. Instead of using standard IEEE floating-point arithmetic (single and double precision), floating-point operators are implemented in reduced precision, which consumes less logic resource and hence allow a higher degree of parallelism, higher clock frequencies and lower I/O bandwidth. The accuracy loss introduced by reduced precision is compensated by re-computation on CPUs using the required higher precision. This chapter proposes a novel data structure and a memory architecture to interface the reduced precision domain on FPGA and the high precision domain on CPU. As a result, the accuracy of output is the same as an equivalent system fully implemented with high precision data-paths. We demonstrate that an optimal precision and performance point can be chosen that balance the number of FPGA data-path and the amount of re-computation on CPUs. The proposed methodology is applied to an image-guided surgical robot application which employs the PQ process. The resultant reconfigurable platform implementation shows a significant speed-up over CPU, GPU and the same platform that has not applied our methodology.

FPGAs' data-path can be customised and reconfigured for one particular application, so it usually demonstrates better power and energy efficiency compared to CPUs and GPUs. However, the power of FPGAs cannot be neglected as they are increasingly used in the high performance computing domain. Apart from traditional power saving techniques such as clock gating and dynamic frequency/voltage scaling, Chapter 4 explores how the unique run-time reconfigurability of FPGAs could be used as an efficient power saving technique. The proposed reconfigurable system has two configurations, which allows the FPGA to run and switch between computation mode and low-power mode. In computation mode, the FPGA is clocked at the maximum frequency and all the available resources are utilised to boost performance. In contrast, for low-power mode, the FPGA is loaded with a configuration which has the slowest possible clock and uses only the minimal amount of resource. The proposed run-time reconfiguration approach is applied to a robot localisation application which employs adaptive SMC methods. Compared to a non-adaptive and non-reconfigurable system, the proposed approach reduces idle power by

25-34% and the overall energy consumption by 17-33%.

Although techniques proposed in Chapter 3 and Chapter 4 enhance the computation and energy efficiency of reconfigurable systems, the design complexity and compilation time of FPGA applications far exceed that of CPUs and GPUs, making FPGAs difficult to be accepted by mainstream application designers. Chapter 5 discusses the programmability challenges, and describes a design flow which extends the SMC reconfigurable system mentioned in Chapter 4. To make the proposed reconfigurable system more user-friendly, this chapter focuses on making the system parametrisable for a wide variety of SMC applications. A surrogate modelling-based machine learning algorithm is employed to tune design parameters for improved performance and solution quality. The design flow enables efficient mapping of applications to multiple FPGAs, reduces design space exploration effort, and is capable of producing reconfigurable implementations for a range of SMC applications. Significant improvement in speed and energy efficiency are achieved over optimised CPU and GPU implementations.

To conclude, Figure 6.1 recaptures the thesis organisation chart in Chapter 1, and it shows the connections of three contributions that enhance reconfigurable systems for real-time applications. Unique features of FPGA technology, in particular customisable precision in Chapter 3 and run-time reconfiguration in Chapter 4, have been applied to optimise reconfigurable real-time systems. The long-standing programmability issues of FPGA has also been addressed by a domain-specific design flow in Chapter 5. The enhanced computing power brought by reconfigurable technologies enlarges the set of compute-intensive algorithms that can have realistic applications in daily life. For example in Chapter 3, we discusses the potential of clinical setting in surgical robots. The use of customisable precision allows more sophisticated models and higher update rates so that surgeons who use surgical robots are able to response promptly. In Chapter 5, the design flow reduces the effort of implementing a high-performance air traffic management system. In addition, the SMC computation engine provides sufficient computing power in dealing with the growing demand of future air traffic. An efficient air traffic management system reduces the level of human control, improves fuel consumption, decreases the time of arrival of aircraft, and increases the capacity of airspace.

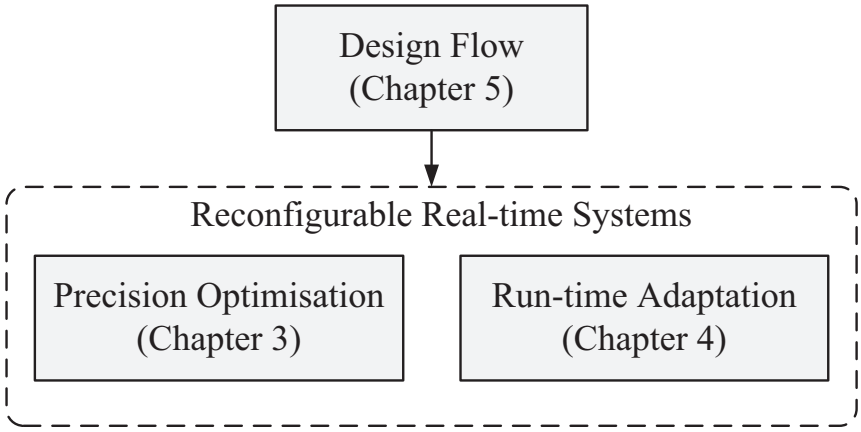


Figure 6.1: Thesis contributions.

6.2 Future Work

This section will elaborate on the current limitations of this thesis, and suggest directions in which future research can address them.

6.2.1 Proximity Query Formulation

The work in Chapter 3 shows the acceleration of PQ with reconfigurable system. PQ has substantial potential in medial surgery which involves human-robot collaborative control. The proposed reduced precision approach can be extended to cover applications which could not be applied to clinical setting due to complex models and stringent real-time requirements. One example is image-guided catheterisation as illustrated in Figure 6.2. To deal with rapid deformation of the heart and the associated blood vessels, it is vitally important to provide the operator of surgical robot online guidance in real time, for which fast and efficient PQ computation is essential. The current implementation of PQ has three limitations that can be improved in the future:

- PQ is currently modelled with points and contours, for example in minimal invasive heart surgery, the surgical instrument is described by a cloud of points and the aorta vessel is modelled by a series of contours. The data structure and memory buffer are

designed specifically for this point-contour model. In the future, the PQ formulation can be extended to point-point model to maximise the flexibility. Such an extended model will increase the computation requirement, thus a faster reconfigurable system is necessary.

- The proposed heterogeneous reconfigurable system connects FPGAs and CPUs via the PCI Express bus. Data accessed by FPGAs have to be copied from the main memory hosted by CPUs, and vice versa. The performance of such decoupled heterogeneous architecture is restricted by high latency and limited bandwidth. In the future, we can investigate closely-coupled platform where CPU and FPGA fabric lie in the same board or even the same chip. One example is SOC-FPGA introduced by Altera [3] and Xilinx [4]. Figure 6.3 shows the block diagram of an Altera SOC-FPGA which integrates an ARM-based hard processor, input/output peripherals, memory interfaces and FPGA fabric. Advanced Microcontroller Bus Architecture (AMBA) provides high throughput interconnect between CPUs and FPGAs. This closely-coupled platform follows the topologies described in Figure 1.1. The FPGAs partition acts as a deterministic real-time co-processor which connects to peripherals. The ARM processor runs a RTOS to serve real-time requests in software.

- At present, the run-time reconfiguration is done on full chip basis, which means that the entire FPGA is loaded with a new bit-stream each time it is reconfigured. On our targeting platform, full chip reconfiguration takes around one second, which precludes its usage in many applications that require fast response time. In Chapter 3, we try to overcome this drawback by reconfiguring one FPGA at a time while keeping the remaining FPGAs operating. This method needs multiple FPGA boards to support individual reconfiguration. It is worth investigating partial reconfiguration technique, where only a subset of the design is changed at run-time. To do this, the design is partitioned into two regions. The critical sections of the data-path, such as those having reduced precision arithmetic, are run-time reconfigurable. The remaining parts, such as PCI Express interface and memory controller, can be kept static. Instead of disabling an entire FPGA board for reconfiguration, the proposed scheme still allows some data-paths to be functioning during reconfiguration.

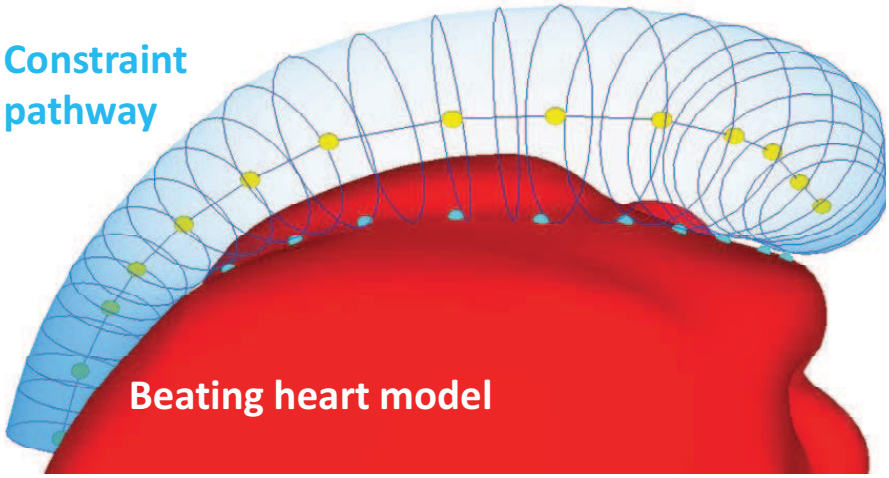


Figure 6.2: Image-guided catheterisation: Perform PQ based on a beating heart model, where light blue bubbles represent the control points registered on the surface and yellow spheres indicate the control points forming the centre line of the pathway [1].

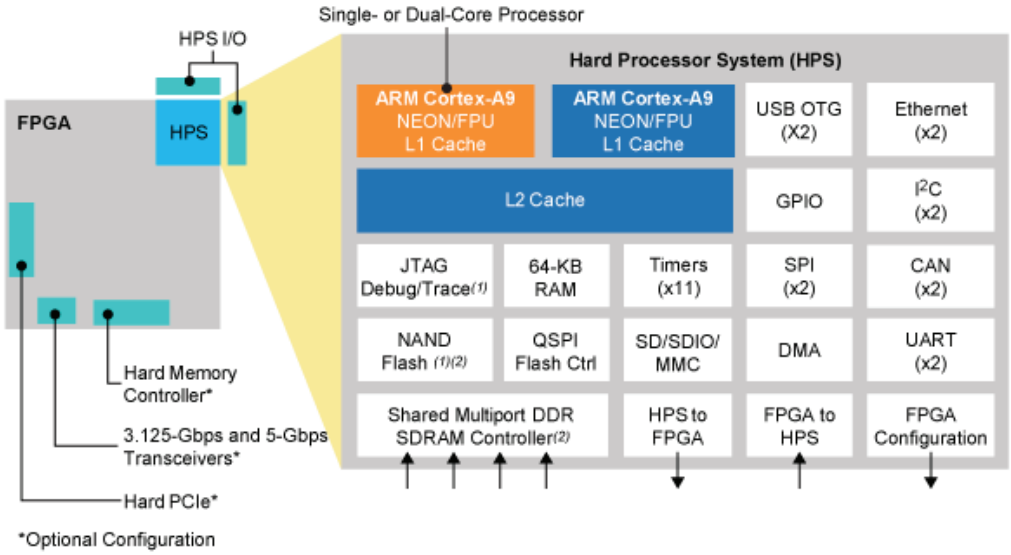


Figure 6.3: Altera SOC which integrates an ARM-based hard processor, peripherals, memory interfaces and FPGA fabric [2].

6.2.2 Adaptive Sequential Monte Carlo Methods

In Chapter 4, the proposed reconfigurable system switches between computation mode and low-power mode. Currently, the system performance is restricted by full-chip reconfiguration, as it consumes time and energy, shortens idle time, and keeps applications which require fast response time away from this system. In fact, PCI Express interface and memory controller should be in place for both configurations as these components are crucial to maintain functionality. To reduce reconfiguration overhead, these components need not to be reconfigured.

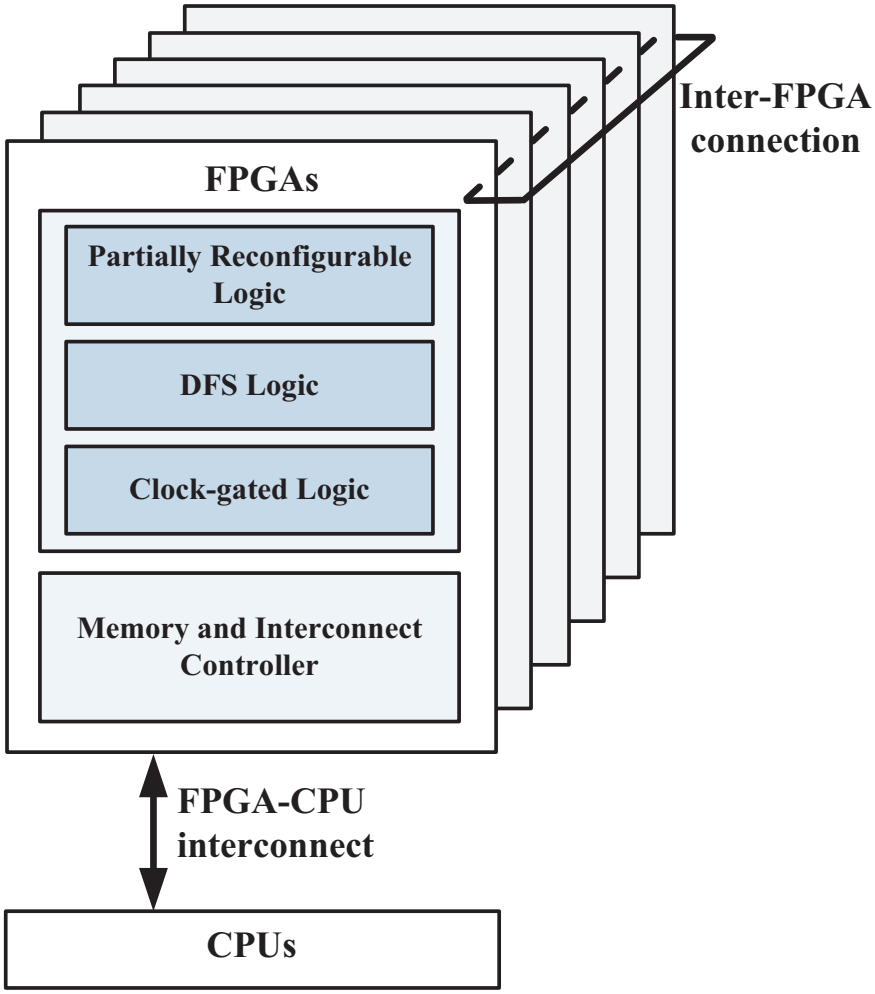


Figure 6.4: Different schemes to put FPGA to sleep.

Apart from partial reconfiguration, the fixed computation interval of real-time system can be exploited by other power saving techniques as summarised in Figure 6.4:

- Dynamic Frequency Scaling (DFS): Instead of the "best-effort" approach which finishes the computation as quickly as possible (Figure 6.5(a)), we can use a "just-in-time" approach (Figure 6.5(b)) which lowers the clock speed to an extent that the system could finish just within the real-time interval. To enable this approach, effective and efficient real-time scheduling should be studied to guarantee meeting real-time requirement.

- Clock gating: This is a common power optimisation technique employed in both ASIC and FPGA designs to eliminate unnecessary switching activity and thus dynamic power consumption. To enable clock gating, designers need to add additional gating components to the RTL code. The added components disable unnecessarily active sequential elements

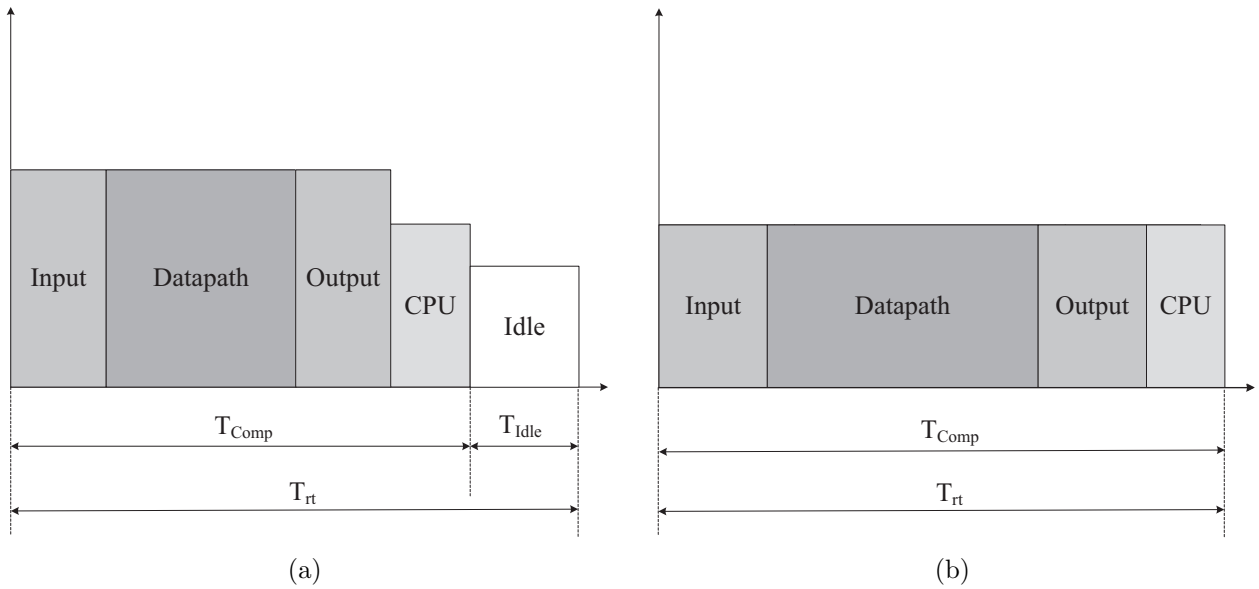


Figure 6.5: (a) Best-effort adaptive scheme described in Chapter 4; (b) Just-in-time adaptive scheme.

which need not to switch states. In ASICs, the clock tree that distributes the clock to all sequential elements is built specifically for each device. The clock tree can be added with any gating component to gate particular groups of clocks, and the delays introduced by these gating components are specifically handled. In FPGAs, the clock trees are fixed because dedicated nets and buffers are responsible for distributing the clock to all logic elements. It is not possible to do the arbitrary gating that is possible in ASICs. In addition, adding gating components and determining portions of gated circuit require intimate knowledge of the design and numerous changes to the RTL. For example, if a clock is gated using a LUT, the clock needs to leave the clock network and be routed using general routing resources to a LUT, and then be routed back to the desired clock. The LUTs used for clock gating can introduce glitch and the extra routing can make the clock arrive later than any clock that stays on the clock tree.

As mentioned earlier in this chapter, the proposed heterogeneous reconfigurable system consists of FPGAs and CPUs which are not closely coupled. In Chapter 4, particle data are transferred frequently between FPGAs fabric and CPUs, and hence significant processing time and power consumption are introduced. The above mentioned SOC-FPGA device (Figure 6.3) with closely-coupled CPUs and FPGA fabric has promising opportunities. An RTOS, such as

VxWork [121] and MicroC/OS-II [122], can run on the CPUs to guarantee real-time capability.

The SMC design flow described in Chapter 5 can be extended for better accessibility and user-friendliness. At present, only application-specific parameters, such as the number of particles and the number of iterations, are being considered in the optimisation approach. The advantage is that the parameters can be studied using a software model, which is fast as no hardware generation is involved. On the flip side, the effect of device-specific parameters, such as the precision of number format, the level of parallelism and the clock speed, are not taken into account. Optimising application-specific and device-specific parameters together can provide more promising results. For example, we can reduce the precision of number format but compensate the loss in accuracy by using more particles. However, there are challenges when bringing in device-specific parameters to the optimisation approach. In particular, the optimisation time will be significantly longer because the time required to generate and benchmark the hardware configurations is extremely long, and these new parameters introduces more dimensions to the optimisation space. To address these challenges, an initial study has been conducted in [36], where an ARDEGO algorithm is proposed to offer automatic optimisation of device-specific parameters in reconfigurable designs. The time spent on hardware generation is reduced by only exploring the parameters that are most likely to give better results, rather than doing exhaustive search. Lastly, to make our proposed design flow more accessible and usable to software programmer, the design flow can be enhanced. It will allow generation of both hardware and software from designs captured in software programming languages (e.g. R, MATLAB) to reconfigurable implementations, and extend the software template in VHDL/Verilog to support a wider range of systems apart from the current Maxeler platform.

Bibliography

[1] K.-W. Kwok, K. H. Tsoi, V. Vitiello, J. Clark, G. C. T. Chow, W. Luk, and G.-Z. Yang, "Dimensionality reduction in controlling articulated snake robot for endoscopy under dynamic active constraints," *IEEE Transactions on Robotics*, vol. 29, no. 1, pp. 15–31, 2013.

[2] "Cyclone V SoCs hard processor system," http://www.altera.com/devices/fpga/cyclone-v-fpgas/hard-processor-system/cyv-soc-hps.html, 2014.

[3] "Cyclone V SoCs: Lowest system cost and power," http://www.altera.com/devices/processor/soc-fpga/cyclone-v-soc/cyclone-v-soc.html, 2014.

[4] "Zynq-7000 all programmable SoC," http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/, 2014.

[5] G. C. T. Chow, K. W. Kwok, W. Luk, and P. H. W. Leong, "Mixed precision processing in reconfigurable systems," in *Proceedings of International Symposium Field-Programmable Custom Computing Machines*, 2011, pp. 17–24.

[6] G. C. T. Chow, A. H. T. Tse, Q. Jin, W. Luk, P. H. Leong, and D. B. Thomas, "A mixed precision Monte Carlo methodology for reconfigurable accelerator systems," in *Proceedings of International Symposium on Field Programmable Gate Arrays*, 2012, pp. 57–66.

[7] Z. Guo, W. Najjar, F. Vahid, and K. Vissers, "A quantitative analysis of the speedup factors of FPGAs over processors," in *Proceedings of International Symposium on Field Programmable Gate Arrays*, 2004, pp. 162–170.

[8] S. Craven and P. Athanas, "Examining the viability of FPGA supercomputing," *EURASIP Journal on Embedded Systems*, vol. 2007, no. 1, p. 13, 2007.

[9] O. Pell and O. Mencer, "Surviving the end of frequency scaling with reconfigurable dataflow computing," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 4, pp. 60–65, 2011.

[10] E. Crisostomi *et al.*, "Combining Monte Carlo and worst-case methods for trajectory prediction in air traffic control: A case study," in *Proc. Eurocontro Innovative Research Workshop and Exhibition*, 2007.

[11] A. Eele and J. M. Maciejowski, "Comparison of stochastic optimisation methods for control in air traffic management," in *Proceedings of IFAC World Congress*, 2011.

[12] A. Ortiz and N. Neogi, "Color optic flow: A computer vision approach for object detection on uavs," in *Proc. Digital Avionics Systems Conf.*, 2006, pp. 1–12.

[13] F. Dellaert *et al.*, "Monte Carlo localization for mobile robots," in *Proc. Int. Conf. Robotics and Automation*, 1999, pp. 1322–1328.

[14] K.-W. Kwok, V. Vitiello, and G.-Z. Yang, "Control of articulated snake robot under dynamic active constraints," in *Proceedings of International Conference Medical image computing and computer-assisted intervention*, 2010, pp. 229–236.

[15] R. Paul, S. Saha, S. Sau, and A. Chakrabarti, "Real time communication between multiple FPGA systems in multitasking environment using RTOS," in *Proceedings of International Conference on Devices, Circuits and Systems*, 2012, pp. 130–134.

[16] M. Schoeberl, "A Java processor architecture for embedded real-time systems," *Journal of Systems Architecture*, vol. 54, no. 1-2, pp. 265–286, 2008.

[17] J. Whitham and N. Audsley, "The scratchpad memory management unit for Microblaze: Implementation, testing, and case study," University of York, Tech. Rep. YCS-2009-439, 2009.

[18] *Drive-On-Chip Reference Design*, Altera, 2014.

[19] A. Burns and A. J. Wellings, *Real-Time Systems and Programming Languages:*, 3rd ed. Addison Wesley, 2001.

[20] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Computing Surveys*, vol. 43, no. 4, pp. 35:1–35:44, 2011.

[21] P. Puschner and A. Burns, "A review of worst-case execution-time analysis (editorial)," *Real-Time Systems*, vol. 18, no. 2/3, pp. 115–128, 2000.

[22] M. J. McGowan, "The rise of computerized high frequency trading: use and controversy," *Duke L. & Tech*, 2010.

[23] T. C. P. Chau, K.-W. Kwok, G. C. T. Chow, K. H. Tsoi, Z. Tse, P. Y. K. Cheung, and W. Luk, "Acceleration of real-time proximity query for dynamic active constraints," in *Proceedings of International Conference on Field-Programmable Technology*, 2013, pp. 206–213.

[24] T. C. P. Chau, M. Kurek, J. S. Targett, J. Humphrey, G. Skouroupathis, A. Eele, J. Maciejowski, B. Cope, K. Cobden, P. Leong, P. Y. K. Cheung, and W. Luk, "SMCGen: Generating reconfigurable design for sequential Monte Carlo applications," in *Proceedings of International Symposium on Field-Programmable Custom Computing Machines*, 2014, pp. 141–148.

[25] T. C. P. Chau, X. Niu, A. Eele, W. Luk, P. Y. K. Cheung, and J. M. Maciejowski, "Heterogeneous reconfigurable system for adaptive particle filters in real-time applications," in *Proceedings of International Symposium Applied Reconfigurable Computing*, 2013, pp. 1–12.

[26] T. C. P. Chau, X. Niu, A. Eele, J. M. Maciejowski, P. Y. K. Cheung, and W. Luk, "Mapping adaptive particle filters to heterogeneous reconfigurable systems," *ACM Transactions on Reconfigurable Technology and Systems*, 2014, accepted.

[27] G. Stitt, "Are field-programmable gate arrays ready for the mainstream?" *IEEE Micro*, vol. 31, no. 6, pp. 58–63, 2011.

[28] "The Green500 list," http://www.green500.org/, 2014.

[29] "What is difference between deep and deeper sleep states?" http://www.intel.com/support/processors/sb/CS-028739.htm, 2014.

[30] "Intel Turbo Boost Technology 2.0," http://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html, 2014.

[31] "AMD Enduro power management technologies," http://www.amd.com/en-us/innovations/software-technologies/enduro, 2014.

[32] "NVIDIA PowerMizer technology," http://www.nvidia.com/object/feature_powermizer.html, 2014.

[33] T. C. P. Chau, J. S. Targett, M. Wijeyasinghe, W. Luk, P. Y. K. Cheung, B. Cope, A. Eele, and J. M. Maciejowski, "Accelerating sequential Monte Carlo method for real-time air traffic management," *SIGARCH Computer Architecture News*, vol. 41, no. 5, 2013.

[34] A. Eele, J. M. Maciejowski, T. C. P. Chau, and W. Luk, "Parallelisation of sequential Monte Carlo for real-time control in air traffic management," in *Proceedings of International Conference Decision and Control*, 2013.

[35] ——, "Control of aircraft in the terminal manoeuvring area using parallelised sequential Monte Carlo," in *Proceedings of AIAA Conference on Guidance, Navigation, and Control*, 2013.

[36] M. Kurek, T. Becker, T. C. P. Chau, and W. Luk, "Automating optimization of reconfigurable designs," in *Proceedings of International Symposium Field-Programmable Custom Computing Machines*, 2014, 201-213.

[37] T. C. P. Chau, W. Luk, P. Y. K. Cheung, A. Eele, and J. M. Maciejowski, "Adaptive sequential Monte Carlo approach for real-time applications," in *Proceedings of International Conference Field Programmable Logic and Applications*, 2012, pp. 527–530.

[38] X. Niu, T. C. P. Chau, Q. Jin, W. Luk, and Q. Liu, "Automating elimination of idle functions by run-time reconfiguration," in *Proceedings of International Symposium on Field-Programmable Custom Computing Machines*, 2013, pp. 97–104.

[39] T. C. P. Chau, W. Luk, and P. Y. K. Cheung, "Roberts: Reconfigurable platform for benchmarking real-time systems," *SIGARCH Computer Architecture News*, vol. 40, no. 5, 2012.

[40] "Nios II Processor," http://www.altera.com/devices/processor/nios2/ni2-index.html, 2014.

[41] "Microblaze soft processor core," http://www.xilinx.com/tools/microblaze.htm, 2014.

[42] J. R. Hauser and J. Wawrzynek, "Garp: A mips processor with a reconfigurable co-processor," in *Proceedings of International Symposium on Field-Programmable Custom Computing Machines*, 1997, p. 12.

[43] S. Trimberger, D. Carberry, A. Johnson, and J. Wong, "A time-multiplexed FPGA," in *Proceedings of International Symposium on Field-Programmable Custom Computing Machines*, 1997, pp. 22–28.

[44] Z. A. Ye, A. Moshovos, S. Hauck, and P. Banerjee, "Chimaera: A high-performance architecture with a tightly-coupled reconfigurable functional unit," *SIGARCH Computer Architecture News*, vol. 28, no. 2, pp. 225–235, 2000.

[45] T. Fujii, K.-i. Furuta, M. Motomura, M. Nomura, M. Mizuno, K.-i. Anjo, K. Wakabayashi, Y. Hirota, Y.-e. Nakazawa, H. Ito, and M. Yamashina, "A dynamically reconfigurable logic engine with a multi-context/multi-mode unified-cell architecture," in *Proceedings of International Solid-State Circuits Conference*, 1999, pp. 364–365.

[46] L. Shang and N. Jha, "Hardware-software co-synthesis of low power real-time distributed embedded systems with dynamically reconfigurable FPGAs," in *Proceedings of Asia and South Pacific Design Automation Conference*, 2002, pp. 345–352.

[47] N. Shenoy, A. Choudhary, and P. Banerjee, "An algorithm for synthesis of large time-constrained heterogeneous adaptive systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 6, no. 2, pp. 207–225, 2001.

[48] B. Jeong, S. Yoo, S. Lee, and K. Choi, "Hardware-software cosynthesis for run-time incrementally reconfigurable FPGAs," in *Proceedings of Asia and South Pacific Design Automation Conference*, 2000, pp. 169–174.

[49] Y. Li, T. Callahan, E. Darnell, R. Harr, U. Kurkure, and J. Stockwood, "Hardware-software co-design of embedded reconfigurable architectures," in *Proceedings of Design Automation Conference*, 2000, pp. 507–512.

[50] J. Noguera and R. Badia, "A HW/SW partitioning algorithm for dynamically reconfigurable architectures," in *Proceedings on Design, Automation and Test in Europe Conference and Exhibition*, 2001, pp. 729–734.

[51] "Stratix V FPGAs: Ultimate flexibility through partial and dynamic reconfiguration," http://www.altera.com/devices/fpga/stratix-fpgas/stratix-v/overview/partial-reconfiguration/stxv-part-reconfig.html, 2014.

[52] "Partial reconfiguration," http://www.xilinx.com/tools/partial-reconfiguration.htm, 2014.

[53] *The Convey HC-2 Architectural Overview*, Convey Computer Corporation, 2014.

[54] "Maxeler Technologies: Products," http://www.maxeler.com/products/, 2014.

[55] "Vivado high-level synthesis," http://www.xilinx.com/products/design-tools/vivado/integration/esl-design/, 2014.

[56] "Impulse accelerated technologies," http://www.impulseaccelerated.com/, 2014.

[57] "Catapult," http://calypto.com/en/products/catapult/overview/, 2014.

[58] "DK design suite: Handel-c to FPGA for algorithm design," http://www.mentor.com/products/fpga/handel-c/dk-design-suite/, 2014.

[59] "Liquid Metal," http://researcher.watson.ibm.com/researcher/view_group.php?id=122, 2014.

[60] "Bluespec," http://www.bluespec.com/, 2014.

[61] "Open SPL," http://www.openspl.org/, 2014.

[62] "MaxCompiler," https://www.maxeler.com/products/software/maxcompiler/, 2014.

[63] "Altera SDK for OpenCL," http://www.altera.com/products/software/opencl/opencl-index.html, 2014.

[64] "HDL Coder: Generate Verilog and VHDL code for FPGA and ASIC designs," http://www.mathworks.co.uk/products/hdl-coder/, 2014.

[65] "DSP Builder," http://www.altera.com/products/software/products/dsp/dsp-builder.html, 2014.

[66] "Xilinx System Generator and HDL Coder," http://www.mathworks.co.uk/fpga-design/simulink-with-xilinx-system-generator-for-dsp.html, 2014.

[67] "Leg Up," http://legup.eecg.utoronto.ca/, 2014.

[68] "Model-based design," http://www.mathworks.co.uk/model-based-design/, 2014.

[69] S. Sharma and W. Chen, "Using model-based design to accelerate FPGA development for automotive applications," SAE Technical Paper, Tech. Rep., 2009.

[70] M. Kurek, T. Becker, and W. Luk, "Parametric optimization of reconfigurable designs using machine learning," in *Proc. Int. Symp. Applied Reconfigurable Computing*, 2013, pp. 134–145.

[71] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *J. Global Optimization*, vol. 13, no. 4, pp. 455–492, Dec. 1998.

[72] C. Rasmussen, "Gaussian processes in machine learning," in *Advanced Lectures on Machine Learning*, ser. Lecture Notes in Computer Science, O. Bousquet, U. von Luxburg, and G. Rtsch, Eds. Springer, 2004, vol. 3176, pp. 63–71.

[73] A. Basudhar, C. Dribusch, S. Lacaze, and S. Missoum, "Constrained efficient global optimization with support vector machines," *Structural and Multidisciplinary Optimization*, vol. 46, no. 2, pp. 201–221, 2012.

[74] E. Nurvitadhi, G. Weisz, Y. Wang, S. Hurkat, M. Nguyen, J. C. Hoe, J. F. Martínez, and C. Guestrin, "GraphGen: An FPGA framework for vertex-centric graph computation," in *Proceedings on International Symposium on Field-Programmable Custom Computing Machines*, 2014, pp. 25–28.

[75] G. Brebner, "Packets everywhere: The great opportunity for field programmable technology," in *Proceedings of International Conference on Field-Programmable Technology*, 2009, pp. 1–10.

[76] M. Attig and G. Brebner, "400 Gb/s programmable packet parsing on a single FPGA," in *Proceedings of Symposium on Architectures for Networking and Communications System*, 2011, pp. 12–23.

[77] G. C. Buttazzo, *Hard Real-Time Computing Systems*, 3rd ed. Springer US, 2011.

[78] E. G. Gilbert and C. P. Foo, "Computing the distance between general convex objects in three-dimensional space," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 1, pp. 53–61, 1990.

[79] N. Chakraborty, J. Peng, S. Akella, and J. E. Mitchell, "Proximity queries between convex objects: An interior point approach for implicit surfaces," *IEEE Transactions on Robotics*, vol. 24, no. 1, pp. 211–220, 2008.

[80] M. Li, M. Ishii, and R. H. Taylor, "Spatial motion constraints using virtual fixtures generated by anatomy," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 4–19, 2007.

[81] D. Constantinescu, S. E. Salcudean, and E. A. Croft, "Haptic rendering of rigid contacts using impulsive and penalty forces," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 309–323, 2005.

[82] M. Jakopec, F. Rodriguez y Baena, S. Harris, P. Gomes, J. Cobb, and B. L. Davies, "The hands-on orthopaedic robot "acrobot": Early clinical trials of total knee replacement surgery," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 5, pp. 902–911, 2003.

[83] M. Benallegue, A. Escande, S. Miossec, and A. Kheddar, "Fast C1 proximity queries using support mapping of sphere-torus-patches bounding volumes," in *Proceedings of International Conference Robotics and Automation*, 2009, pp. 483–488.

[84] X. Zhang and Y. J. Kim, "Interactive collision detection for deformable models using streaming aabbs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 2, pp. 318–329, 2007.

[85] E. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal of Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[86] B. Mirtich and B. Mirtich, "V-Clip: Fast and robust polyhedral collision detection," *ACM Transactions on Graphics*, vol. 17, pp. 177–208, 1998.

[87] M. C. Lin and J. F. Canny, "A fast algorithm for incremental distance calculation," in *Proceedings of International Conference Robotics and Automation*, 1991, pp. 1008–1014.

[88] C. F. Fang, R. A. Rutenbar, and T. Chen, "Fast, accurate static analysis for fixed-point finite-precision effects in dsp designs," in *Proceedings of International Conference Computer-aided Design*, 2003, pp. 275–282.

[89] D.-U. Lee, A. Abdul Gaffar, W. Luk, and O. Mencer, "MiniBit: Bit-width optimization via affine arithmetic," in *Proceedings of Design Automation Conference*, 2005, pp. 837–840.

[90] W. G. Osborne, J. Coutinho, R. C. C. Cheung, W. Luk, and O. Mencer, "Instrumented multi-stage word-length optimization," in *Proceedings of International Conference Field-Programmable Technology*, 2007, pp. 89–96.

[91] D. Boland and G. A. Constantinides, "Automated precision analysis: A polynomial algebraic approach," in *Proceedings of International Symposium Field-Programmable Custom Computing Machines*, 2010, pp. 157–164.

[92] A. Doucet, N. de Freitas, and N. Gordon, *Sequential Monte Carlo methods in practice*. Springer, 2001.

[93] M. Happe, E. Lübbers, and M. Platzner, "A self-adaptive heterogeneous multi-core architecture for embedded real-time video object tracking," *Journal of Real-Time Image Processing*, pp. 1–16, 2011.

[94] M. Montemerlo, S. Thrun, and W. Whittaker, "Conditional particle filters for simultaneous mobile robot localization and people-tracking," in *Proceedings of International Conference Robotics and Automation*, 2002, pp. 695–701.

[95] J. Vermaak, C. Andrieu, A. Doucet, and S. J. Godsill, "Particle methods for Bayesian modeling and enhancement of speech signals," *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 3, pp. 173–185, 2002.

[96] N. Kantas, J. M. Maciejowski, and A. Lecchini-Visintini, "Sequential Monte Carlo for model predictive control," in *Nonlinear Model Predictive Control*, ser. Lecture Notes in Control and Information Sciences, 2009, pp. 263–273.

[97] D. Creal, "A survey of sequential Monte Carlo methods for economics and finance," *Econometric Reviews*, vol. 31, no. 3, pp. 245–296, 2012.

[98] N. J. Gordon, D. J. Salmond, and A. F. M. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *Proceedings of Radar and Signal Processing*, vol. 140, no. 2, pp. 107–113, 1993.

[99] G. Kitagawa, "Monte Carlo filter and smoother for non-gaussian nonlinear state space models," *Journal of Computational and Graphical Statistics*, vol. 5, no. 1, pp. 1–25, 1996.

[100] D. Koller and R. Fratkina, "Using learning for approximation in stochastic processes," in *Proceedings of International Conference Machine Learning*, 1998, pp. 287–295.

[101] D. Fox, "Adapting the sample size in particle filters through KLD-sampling," *International Transactions on Robotics*, vol. 22, no. 12, pp. 985–1003, 2003.

[102] S.-H. Park, Y.-J. Kim, and M.-T. Lim, "Novel adaptive particle filter using adjusted variance and its application," *International Journal of Control, Automation and Systems*, vol. 8, no. 4, pp. 801–807, 2010.

[103] M. Bolic, S. Hong, and P. M. Djuric, "Performance and complexity analysis of adaptive particle filtering for tracking applications," in *Proceedings of Asilomar Conference Signals, Systems, and Computers*, vol. 1, 2002, pp. 853–857.

[104] Z. Liu, Z. Shi, M. Zhao, and W. Xu, "Mobile robots global localization using adaptive dynamic clustered particle filters," in *Proceedings of International Conference Intelligent Robots and Systems*, 2007, pp. 1059–1064.

[105] I. Lymperopoulos and J. Lygeros, "Sequential monte carlo methods for multi-aircraft trajectory prediction in air traffic management," *International Journal of Adaptive Control and Signal Processing*, vol. 24, no. 10, pp. 830–849, 2010.

[106] I. Lymperopoulos, "Sequential monte carlo methods in air traffic management," Ph.D. dissertation, ETH Zurich, 2010.

[107] J. Ponce, D. Chelberg, and W. B. Mann, "Invariant properties of straight homogeneous generalized cylinders and their contours," *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 11, no. 9, pp. 951–966, 1989.

[108] F. P. Preparate and M. I. Shamos, *Computational Geometry.* Springer, 1985.

[109] E. Weisstein, "Point-line distance–3-dimensional," http://mathworld.wolfram.com/Point-LineDistance3-Dimensional.html.

[110] *Floating-Point Megafunctions User Guide*, Altera, 2013.

[111] L. Fousse, G. Hanrot, V. Lefère, P. Péissier, and P. Zimmermann, "MPFR: A multiple-precision binary floating-point library with correct rounding," *ACM Transactions on Mathematical Software*, vol. 33, no. 2, pp. 13:1–13:15, 2007.

[112] M. Bolic, P. M. Djuric, and S. Hong, "Resampling algorithms and architectures for distributed particle filters," *IEEE Transactions on Signal Processing*, vol. 53, no. 7, pp. 2442–2450, 2005.

[113] L. M. Murray, A. Lee, and P. E. Jacob, "Parallel resampling in the particle filter," Tech. Rep., 2014. [Online]. Available: http://arxiv-web3.library.cornell.edu/abs/1301.4019?context=cs

[114] C.-E. Särndal, B. Swensson, and J. Wretman, *Model assisted survey sampling*. Springer, 2003.

[115] R. Douc and O. Cappe, "Comparison of resampling schemes for particle filtering," in *Image and Signal Processing and Analysis, 2005. ISPA 2005. Proceedings of the 4th International Symposium on*, Sept 2005, pp. 64–69.

[116] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[117] R. M. Neal, "Slice sampling," *Annals of statistics*, pp. 705–741, 2003.

[118] L. Miao, J. J. Zhang, C. Chakrabarti, and A. Papandreou-Suppappola, "Algorithm and parallel implementation of particle filtering and its use in waveform-agile sensing," *Journal of Signal Processing Systems*, vol. 65, no. 2, pp. 211–227, 2011.

[119] D. B. Thomas and W. Luk, "High quality uniform random number generation using LUT optimised state-transition matrices," *Journal of VLSI Signal Processing Systems*, vol. 47, no. 1, pp. 77–92, 2007.

[120] ——, "An FPGA-specific algorithm for direct generation of multi-variate Gaussian random numbers," in *Proceedings of International Conference on Application-specific Systems Architectures and Processors*, 2010, pp. 208–215.

[121] "VxWorks RTOS," http://www.windriver.com/products/vxworks/, 2014.

[122] "uC/OS-II overview," http://micrium.com/rtos/ucosii/overview/, 2014.