Generative Programming Methods for Parallel Partial Differential Field Equation Solvers

A thesis presented in partial fulfilment of the requirements
for the degree of

Doctor of Philosophy
in
Computer Science

at Massey University, Albany
New Zealand

Daniel Peter Playne

2011

Abstract

This thesis describes a generative programming system that automatically constructs parallel simulations of complex systems that are based on field equations using finite differencing and explicit Runge-Kutta integration methods. Programming computational simulations by hand for different parallel architectures is both tedious and time consuming. Simulation frameworks struggle to target different architectures without losing performance. Automating the process of constructing simulation codes can significantly improve productivity.

Three computational models based on field equations are discussed in depth along with numerical methods for discretising and simulating them. The Cahn-Hilliard model of phase separation, the Ginzburg-Landau model of superconductivity and the Lotka-Volterra model of interacting populations are discussed in detail and are used as examples. A number of modern parallel computing architectures and associated parallel programming languages are also discussed and simulation implementations that run upon these architectures are presented. The performance results from these implementations are used to compare the parallel architectures and their relative performance capabilities for processing each type of simulation.

The key elements of a simulation are identified as being: the computational model, the stencil operators, the explicit integration methods and the configuration information. A domain specific language for defining these elements is developed and presented. A generative programming system is described that parses these element definitions and combines them together to form a complete simulation definition. It is then shown that code generators can be readily developed to produce very efficient implementations from these simulation definitions in any parallel programming language including C, TBB, MPI and CUDA for which explicit examples are given. It is shown that this method is effective, extensible and can considerably reduce the programmer effort required to develop fast parallel simulations of complex systems.

Acknowledgements

First of all I should like to thank Ken Hawick and Chris Scogings for their wisdom, patience and encouragement throughout this work. My deepest thanks to my parents David and Cheryl, my brother Matthew and Rebecca Pearman for their continual love and support. Also to my colleagues Arno Leist, Anton Gerdelan, Teo Susnjak and Guy Kloss for their advice and many discussions. I wish to acknowledge The North Harbour Club, Massey University and the New Zealand Tertiary Education Commission for their financial support. Finally I would like to thank everyone who has supported and contributed to making this work possible.

Contents

1 Introduction **1**
- 1.1 The Challenge of Simulation Development . . . 1
- 1.2 Introduction to Complex Systems . . . 1
- 1.3 Introduction to Field Equations . . . 2
- 1.4 Introduction to Parallel Computing . . . 3
- 1.5 Introduction to Automatic Parallelization . . . 4
- 1.6 Introduction to Generative Programming . . . 5
- 1.7 Previous Work . . . 6
- 1.8 Aim of this Thesis . . . 7
- 1.9 Thesis Structure . . . 8

2 Models **9**
- 2.1 Introduction . . . 9
- 2.2 Cahn-Hilliard Equation . . . 10
- 2.3 Time Dependent Ginzburg-Landau Equation . . . 11
- 2.4 Lotka-Volterra Equation . . . 14
- 2.5 Model Commonalities . . . 18
- 2.6 Conclusions . . . 18

3 Numerical Methods **19**
- 3.1 Introduction . . . 19
- 3.2 Finite-Differencing . . . 20
- 3.3 Boundary Conditions . . . 22
- 3.4 Time Integration Methods . . . 25
- 3.5 Butcher Tableaux . . . 27
- 3.6 Commonalities in Numerical Methods . . . 28
- 3.7 Higher-Order Runge-Kutta Methods . . . 29
- 3.8 Conclusions . . . 30

4 Parallel Architectures and Languages **31**
- 4.1 Introduction . . . 31
- 4.2 Central Processing Units . . . 32
- 4.3 Cluster Computers . . . 34
- 4.4 Graphical Processing Units . . . 36
- 4.5 Multi-GPU . . . 41
- 4.6 GPU-Accelerated Clusters . . . 42
- 4.7 Commonalities in Parallel Architectures . . . 44
- 4.8 Conclusions . . . 44

5 Parallel Algorithms **45**
- 5.1 Introduction . . . 45
- 5.2 Equation Computation . . . 46
- 5.3 Boundary Conditions . . . 49
- 5.4 Sequential Implementation . . . 50
- 5.5 Parallel Decomposition . . . 53
- 5.6 Multi-Core - POSIX Threads . . . 54
- 5.7 Multi-Core -Threading Building Blocks . . . 56
- 5.8 Cluster - MPI . . . 57
- 5.9 Graphical Processing Units - CUDA . . . 59

5.10 Multi-GPU - CUDA & Pthreads . . . 63
5.11 GPU Cluster - CUDA & MPI . . . 67
5.12 Simulation Commonalities . . . 68
5.13 Conclusions . . . 69

6 Performance Results **71**
6.1 Introduction . . . 71
6.2 Model Computation . . . 71
6.3 Single-Core CPU . . . 72
6.4 Multi-Core CPU . . . 73
6.5 Cluster . . . 76
6.6 Tesla GPU . . . 77
6.7 Fermi GPU . . . 81
6.8 Multi-GPU . . . 83
6.9 GPU Cluster . . . 83
6.10 Performance Comparison . . . 84
6.11 Conclusions . . . 85

7 Simulation Description Language **87**
7.1 Introduction . . . 87
7.2 Domain Specific Language . . . 88
7.3 Equation Parser . . . 89
7.4 Stencil Operators . . . 93
7.5 Integration Methods . . . 95
7.6 Configuration . . . 97
7.7 Conclusions . . . 98

8 Simulation Representation **99**
8.1 Introduction . . . 99
8.2 Equation Trees . . . 100
8.3 Stencil Nodes . . . 102
8.4 Integration Trees . . . 105
8.5 Merging Equation and Integration Trees . . . 108
8.6 Conclusions . . . 114

9 Automatic Code Generation **117**
9.1 Introduction . . . 117
9.2 Equation Generation . . . 118
9.3 Generating Target Specific Code . . . 121
9.4 Generating C code . . . 121
9.5 Generating TBB code . . . 124
9.6 Generating MPI code . . . 127
9.7 Generating CUDA code . . . 131
9.8 STARGATES Results . . . 135
9.9 Conclusions . . . 136

10 Conclusions **137**
10.1 Thesis Overview . . . 137
10.2 Major Contributions . . . 139
10.3 Implications for Automatic Parallelism . . . 140
10.4 Implications for Computational Simulations . . . 141
10.5 Future Work . . . 142

A Adaptive Stepsize Methods **147**
A.1 Introduction to Adaptive Stepsize Methods 147
A.2 Runge-Kutta Methods with Adaptive Stepsizes 147

B Generated Code Example **151**

List of Figures

2.1 Three-dimensional Cahn-Hilliard visualisations . . . 11
2.2 The three quenching phases of the Cahn-Hilliard model . . . 12
2.3 Coarsening behaviour of the Cahn-Hilliard model . . . 13
2.4 Two-dimensional visualisation methods for the Ginzburg-Landau equation . . . 14
2.5 Three-dimensional Ginzburg-Landau visualisations . . . 15
2.6 Three-dimensional Lotka-Volterra visualisations . . . 16
2.7 Lotka-Volterra Species Populations . . . 17

3.1 The Laplacian operator . . . 21
3.2 The Biharmonic operator . . . 22
3.3 Periodic boundary conditions . . . 23
3.4 Dirichlet boundary conditions . . . 24
3.5 Neumann boundary conditions . . . 25

4.1 Single-Core CPU architecture . . . 32
4.2 Multi-Core CPU architecture . . . 33
4.3 Cluster computer architecture . . . 35
4.4 Tesla GPU architecture . . . 37
4.5 Fermi GPU architecture . . . 39
4.6 Multi-GPU architecture . . . 42
4.7 GPU cluster architecture . . . 43

5.1 Two-dimensional lattice decomposition . . . 53
5.2 Three-dimensional lattice decomposition . . . 54

6.1 Cahn-Hillard, Ginzburg-Landau and Lotka-Volterra processing time on a CPU . . . 72
6.2 Cahn-Hillard, Ginzburg-Landau and Lotka-Volterra processing time on a GPU . . . 73
6.3 Comparison of integration methods on a single CPU core . . . 74
6.4 Comparison of thread numbers for Pthreads implementation . . . 75
6.5 Comparison of Pthreads and single-core implementations . . . 75
6.6 Performance of different TBB task block sizes . . . 76
6.7 Comparison of the TBB and single-core implementation . . . 77
6.8 Comparison of MPI nodes . . . 78
6.9 Comparison of the MPI and single-thread implementation . . . 78
6.10 Comparison of thread block sizes for Tesla GPUs . . . 79
6.11 Comparison of memory types for Tesla GPUs . . . 80
6.12 Comparison of the Tesla and single-core implementations . . . 80
6.13 Comparison of thread block sizes for Fermi GPUs . . . 81
6.14 Comparison of memory types for Fermi GPUs . . . 82
6.15 Comparison of the Fermi and single-core implementations . . . 82
6.16 Comparison of the Multi-GPU and single-GPU implementations . . . 83
6.17 Comparison of all implementations . . . 84

7.1 Laplacian, Biharmonic and SpatialAverage stencils . . . 94
7.2 Extension of Laplace stencil to higher dimensions . . . 95

8.1 Cahn-Hilliard equation tree . . . 101
8.2 Ginzburg-Landau equation tree . . . 101
8.3 Lotka-Volterra equation tree . . . 102

8.4 Stencil operator arrays 103
8.5 Rearranged Cahn-Hilliard equation tree 104
8.6 Application of two stencil operators 105
8.7 Euler method integration tree 107
8.8 RK2 method integration tree 108
8.9 Cahn-Hilliard Euler simulation tree 111
8.10 Cahn-Hilliard RK2 simulation tree 112
8.11 Lotka-Volterra RK2 simulation tree 115

List of Tables

3.1 General form of a Butcher Tableau 28
3.2 Euler method Butcher tableau 28
3.3 RK4 method Butcher tableau 28
3.4 RK4(2) method Butcher tableau 29
3.5 RK5 method Butcher tableau 30
3.6 RK6 method Butcher tableau 30

6.1 Comparison of parallel architectures and models 85

7.1 General form of a Butcher tableau 95

A.1 Merson method Butcher tableau 148
A.2 Fehlberg $5^{th}/6^{th}$ method Butcher tableau 149
A.3 Verner $5^{th}/6^{th}$ method Butcher tableau 150
A.4 Dormand-Prince $4^{th}/5^{th}$ method Butcher tableau 150