

Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.

Data-parallel Structural Optimisation in Agent-based Modelling

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.

Alwyn Visser Husselmann
May 2014

Abstract

AGENT-BASED MODELLING (ABM) IS PARTICULARLY SUITABLE for aiding analysis and producing insight in a range of domains where systems have constituent entities which are autonomous, interactive and situated. Decentralised control and irregular communication patterns among these make such models difficult to simulate and even more so to understand. However, the value in this methodology lies in its ability to formulate systems naturally, not only generating the desired macroscopic phenomena, but doing so in an elegant manner. With these advantages, ABM has been enjoying widespread and sustained increasing use.

It is then reasonable to seek advances in the field of ABM which would improve productivity, comparability, and ease of implementation. Much work has been done towards these, notably in terms of design methodology, reporting, languages and optimisation. Three issues which remain despite these efforts concern the efficient construction, performance and calibration of agent-based models.

Constructing a model involves selecting agents, their attributes, behaviours, interaction rules, and environment, but it also demands a certain level of programming ability. This learning curve stymies research effort from disciplines unrelated to computer science. It is also not clear that one methodology and software package is suitable for all circumstances. Domain-specific languages (DSLs) make development much simpler for their application area.

Agent-based model simulation sometimes suffer tremendously from performance issues. Models of situations such as algal cultivation, international markets and pedestrians in dense urban areas invariably suffer from poor scaling. This puts large system sizes and temporally distant states out of reach. The advent of scientific programming on graphical processing units (GPUs) now provides inexpensive high performance, giving hope in this area.

It is also important to calibrate such models. More interestingly, the problem of calibrating model *structure* is given particular emphasis. This ambitious task is difficult for a number of reasons, and is investigated with considerable thought in this work.

In summary, the research shows that appropriate use of *data-parallelism* by *multi-stage programming* in a simple *domain-specific language* affords *high performance*, *extensibility* and *ease of use* which is capable of effective automatic model structure optimisation.

Key words: Agent-based Modelling, Domain-specific Languages, GPUs, Optimisation.

Acknowledgements

To the memory of David Frederik Husselmann Sr., 1929-2003.

The invaluable guidance of several people have made this project possible. Firstly, I would like to express my gratitude to my supervisors, Prof. Ken Hawick and A/Prof. Chris Scogings, without whom this dissertation would certainly not have been possible.

I am also greatly indebted to the members of the Complex Systems and Simulations Group at Massey University, especially Dr. Daniel Playne and Dr. Arno Leist for continuing advice and valuable suggestions for improvements to the manuscript.

I have been very privileged to have been able, in November of 2013, to marry my now-wife, Leah. To her I express my appreciation for her neverending support, especially during times of profound difficulty and late work nights.

My family has been very supportive, and I would like to thank my parents particularly, Erika and David, for a place to live and study. Without their support, I certainly could not have studied full time, and would not be in a position to progress my career for several years. My brothers, sisters-in-law and grandmother were also highly supportive during this time.

Finally, I would like to acknowledge the financial support I received from the Massey University Vice-Chancellor's Doctoral Scholarship.

Abstract	iii
Acknowledgements	v
List of Tables	x
List of Figures	xi
I Introduction	1
1 Overview	3
1.1 Structure and Contributions	3
1.2 Agent-based Modelling	4
1.3 Evolutionary Algorithms	5
1.4 Data-parallel Scientific Computing	6
1.5 Domain-Specific Languages	7
1.6 Model Calibration and Optimisation	8
1.7 Summary	10
2 Parallel Agent-based Modelling and Simulation	11
2.1 Introduction to Agent-Based Modelling and Simulation	11
2.1.1 Conway’s Game of Life	15
2.1.2 Flocking Simulations	16
2.1.3 Predator-Prey Models	18
2.1.4 Amorphous Computers	21
2.1.5 Visualisation and Analysis	21
2.2 Introduction to Parallel ABMS	22
2.2.1 Compute Unified Device Architecture	24
2.2.2 Spatial Partitioning	29
2.2.3 Multiple GPUs	31

II	Parallel Evolutionary Algorithms	35
3	Continuous Global Optimisation	37
3.1	Introduction	38
3.1.1	First-order Optimisation	39
3.1.2	Stochastic Derivative-free Optimisation	39
3.1.3	Calibrating Agent-based Models	40
3.2	Evolutionary Optimisers	41
3.2.1	Genetic Algorithm	44
3.2.2	Particle Swarm Optimisation	45
3.2.3	Firefly Algorithm	47
3.3	Advanced Space Exploration	48
3.3.1	Convergence Results using MOLPSO	49
3.3.2	Discussion	51
3.4	Parallel Implementation	52
3.4.1	MOL Particle Swarm Optimiser	52
3.4.2	Firefly Algorithm	56
3.4.3	Discussion	58
3.5	Calibrating Metaheuristic Optimisers	58
3.5.1	Methodology	59
3.5.2	Results	60
3.5.3	Discussion	62
3.6	Higher Dimension Visualisation	64
4	Combinatorial Optimisation	67
4.1	Introduction	67
4.1.1	Symbolic Regression	70
4.1.2	Santa Fe Ant Trail	71
4.2	Linear Representations and Karva	73
4.3	Data-parallelism using GPUs	76
4.4	Karva, Genetic Programming and Parallelisation	77
4.4.1	Parallelisation of K-GP-GPU	77
4.4.2	Experimental Results	79
4.5	Geometric Optimisation	82
4.6	Geometric Particle Swarm Optimiser	83
4.6.1	GPSO for Karva-expressions	83
4.6.2	Experimental Results	85
4.7	Geometric Firefly Algorithm	88
4.7.1	Parallelisation	90
4.7.2	Method	92
4.7.3	Experimental Results	92
4.8	Program-space Visualisation	96
4.8.1	Recursive Space Subdivision	97
4.8.2	Graph Rendering Methods	102

III	MOL: Domain-Specific Language for ABM	105
5	Domain Specific Language for ABM	107
5.1	Introduction	107
5.2	Multi-stage Programming and Terra	109
5.3	The MOL Compiler	112
5.3.1	Lattice-oriented ABM	112
5.3.2	Spatial ABM	115
5.4	Examples and Selected Results	117
5.4.1	Conway's Game of Life	117
5.4.2	Predator-Prey Model	119
5.4.3	Flocking	122
5.5	Discussion	123
6	Parallel Domain-specific Optimisation in ABM	125
6.1	Introduction	125
6.2	Model Structure Optimisation using MOL	128
6.3	The Single-threaded Evolutionary Optimiser	131
6.4	Experiments and Convergence Results	132
6.4.1	Single-statement Selection	134
6.4.2	Permutation Extraction	134
6.4.3	Recombination	135
6.5	Parallel MOL	141
6.6	Selected Results	143
6.6.1	Santa Fe Ant Trail	143
6.6.2	Predator-Prey Model	145
6.7	Discussion	148
7	Photobioreactor Modelling	149
7.1	Introduction	149
7.2	Conventional ABM	151
7.2.1	Hydrodynamic Flow and Illumination Approximation	151
7.2.2	Cell Division	157
7.2.3	Mutual Cell Shading and Negative Biomass Growth	163
7.3	Modelling using MOL	166
7.3.1	Implementation Detail	169
7.3.2	Results	169
7.3.3	Structural Optimisation Experiment	170
7.4	Discussion	175

IV	Discussions and Conclusions	177
8	Discussions	179
8.1	Parallel Agent-based Modelling and Simulation	179
8.2	Parallel Continuous Optimisation	181
8.3	Parallel Combinatorial Optimisation	182
8.4	Domain-specific Languages for ABM	183
8.5	Parallel Domain-specific Optimisation in ABM	184
8.6	Photobioreactor Modelling	185
9	Conclusions	187
9.1	Overview	187
9.2	Domain-specific Languages for ABM	187
9.3	High Performance ABM and Optimisation	188
9.4	Model Structures, Parameters, and Calibration	189
9.5	Summary	190
9.6	Future Work	191
	Bibliography	192
A	List of Publications	207
	Glossary	209
	Acronyms	211

LIST OF TABLES

2.1	Predator-Prey model: Simple ruleset for predators.	19
2.2	Predator-Prey model: A simple ruleset for prey.	20
3.1	Continuous Optimisation: Testing functions, their constraints and optima.	44
3.2	Advanced Space Exploration: Convergence data across testing functions with various random motion methods.	51
3.3	Parallel MOLPSO: Convergence and performance data, comparing Lévy flights and the original.	54
3.4	Continuous Optimisation: Single-threading vs GPU-parallel Firefly algorithms.	56
3.5	Meta-optimisation: Mean results generated by the MOLPSO meta-optimiser.	61
3.6	Meta-optimisation: Best sub-optimiser parameters generated by the meta-optimiser.	61
4.1	K-GP-GPU: Performance data for GP-CPU and the K-GP-GPU algorithms.	80
6.1	MOL: Additional parameters used throughout simulations and optimisation algorithms.	136
6.2	MOL: Parameters used in optimising model structure for the Santa Fe Ant Trail problem.	139
6.3	MOL: Parameters used for the Santa Fe Ant Trail problem and the Predator-prey model for performance comparisons.	144
7.1	Photobioreactors: Parameters used in the photobioreactor model during tests.	165
7.2	Photobioreactors: MOL extensions used in the photobioreactor model.	168

LIST OF FIGURES

1.1	The Agent-based Modelling research process.	5
1.2	Flow diagram of the typical Evolutionary Algorithm.	6
1.3	The architectures of typical compiled internal and external domain-specific languages.	8
1.4	Flow diagram of agent-based model calibration using a continuous global optimiser.	9
2.1	Flocking: A complex system state of a flocking simulation, depicting complex behaviour which many scientists term “emergence”.	14
2.2	Sample spatial configurations of agent-based models.	15
2.3	Conway’s Game of Life: A time lapse of a sample pattern in Conway’s Game of Life.	16
2.4	Flocking: An example of a multi-species Boids model.	18
2.5	Predator-Prey model: Lattice coverage by time in an implementation of the Predator-Prey model.	19
2.6	A time lapse image showing several screenshots of a Particle Swarm Optimiser written in Proto.	21
2.7	Flocking: Spatial histograms of the Boids model.	22
2.8	Flocking: Plot of Boid cluster counts using a component labelling algorithm.	23
2.9	Example renders of an agent-based photobioreactor model.	23
2.10	A flowchart indicating the process that an application typically follows when using an NVIDIA GPU.	25
2.11	Performance plot of system sizes against a variety of NVIDIA graphics hardware.	28
2.12	Flocking: A visualisation of one million Boids, rendered using pixel shaders.	28
2.13	Flocking: Visualisation of a uniform grid partitioning scheme over a set of agents.	29
2.14	Flocking: Visualisation of simple and complex constructed k-D trees.	30
2.15	Multiple-GPU performance results for datastructure construction and timestep computation.	33
3.1	Surface plots of the Schwefel, Rastrigin, Ackley and Rosenbrock test functions.	43
3.2	A visualisation of a large number of Firefly particles attempting to optimise the 3-parameter Rosenbrock function.	48
3.3	Sample random walks generated by Brownian motion, and Cauchy, Gaussian and Lévy flights.	49
3.4	Parallel MOLPSO: Particle population size scaling and dimension count scaling characteristics.	55

3.5	A population of 262,244 fireflies optimising a 3-parameter Rosenbrock function. Step sizes were deliberately smaller to accentuate movement through space and discovery of better solutions. . . .	57
3.6	Meta-optimisation: An example high-level view of a meta-optimiser.	60
3.7	Meta-optimisation: A snapshot of the super-optimiser and the results of the sub-optimisers on the Rosenbrock function superimposed.	61
3.8	Meta-optimisation: Mean fitnesses obtained across a range of test functions.	63
3.9	High Dimension Visualisation: Example particle renders.	64
4.1	Santa Fe Ant Trail Problem: Initial lattice state.	71
4.2	Santa Fe Ant Trail Problem: Initial random 3D lattice state of a modified Santa Fe Ant Trail problem.	72
4.3	Santa Fe Ant Trail Problem: A simple solution attempt.	72
4.4	Santa Fe Ant Trail Problem: Perfect solution generated by Koza.	72
4.5	Karva: An abstract syntax tree of a sample <i>karva</i> -expression.	74
4.6	Karva: Example results from a simple <i>karva</i> -expression crossover operation.	75
4.7	K-GP-GPU: Convergence results for the modified 3D Santa Fe Ant Trail problem.	79
4.8	K-GP-GPU: Population construction timing.	81
4.9	Genetic Programming: A typical agent generated by the “Full” genetic programming initialisation method.	81
4.10	Genetic Programming: A highly effective generated agent as an abstract syntax tree.	82
4.11	K-GP-GPU: Modified 3D Santa Fe Ant Trail problem convergence results using elitism.	86
4.12	K-GPSO-GPU: Modified 3D Santa Fe Ant Trail problem convergence results with elitism.	86
4.13	K-GPSO-GPU: Modified 3D Santa Fe Ant Trail problem convergence results compared against the K-GP-GPU algorithm.	88
4.14	K-GPSO-GPU: Modified 3D Santa Fe Ant Trail problem compute times for fitness and generation computation.	89
4.15	GFA: Sample expression tree with full score for a symbolic regression problem.	92
4.16	GFA: Comparison plot of different decay parameters.	93
4.17	GFA: Fitness plot without observed score decay.	93
4.18	GFA: Fitness plot with decay parameter set to 0.01.	94
4.19	GFA: Success rates on the Symbolic Regression problem by different decay values.	95
4.20	GFA: Sample decay curves with various γ settings.	95
4.21	GFA: Success rates by decay parameter setting.	96
4.22	GFA: Average compute performance for timesteps and population construction by generation.	97
4.23	Program Space Visualisation: Visualising a population of agents using a tree structure.	98
4.24	Program Space Visualisation: Sample populations throughout execution of the K-GP-GPU algorithm.	100
4.25	Program Space Visualisation: Early population visualisation using recursive space division.	101
4.26	Diversity plot of the K-GP-GPU and K-GPSO-GPU algorithms during a typical run.	101
4.27	Program Space Visualisation: Visualisation of the transition between timestep 1 and 2 of the K-GPSO-GPU algorithm with a lower crossover probability.	102

4.28	Successive scaling to the location of the global optimum.	103
4.29	Program Space Visualisation: 2D graph visualisation of a newly initialised population.	103
4.30	Program Space Visualisation: Graph rendering of the 500th generation's population.	104
4.31	Program Space Visualisation: Graph rendering of the 1000th generation's population.	104
5.1	A flow diagram indicating the initial program flow of the DSL framework presented here.	113
5.2	Performance comparison between hand written GNU C++ code (Listing 5.7) and Terra (LLVM) compiled code (Listing 5.8).	116
5.3	Live/dead cell count fractions including like-like (LL) bond fractions.	119
5.4	A plot of the predator and prey lattice coverage (discussed in Chapter 2), duplicated here for convenience).	120
5.5	A screenshot of a flocking model implemented in MOL, using code similar to that of Listing 5.12.	123
6.1	Optimisation: Visualisation of a small population of candidate models.	127
6.2	Optimisation: Flow diagram of the underlying optimisation processes in the optimising MOL language.	132
6.3	Optimisation: Screenshot during evaluation stage.	137
6.4	A screenshot of the optimised program from Listing 6.5 using the code block from Listing 6.6.	137
6.5	The best candidate generated by the MOL recombination optimiser (executed using the code shown in Listing 6.7). The score this candidate achieved was zero, which is the best possible score.	140
6.6	Optimisation: Santa Fe Ant Trail problem fitness graph using MOL.	140
6.7	Optimisation: Fitness by generation of the Santa Fe Ant Trail problem using the single-threaded version of MOL.	144
6.8	Optimisation: Performance plots of timestep and population computing times by CPU and GPU.	146
6.9	Optimisation: Performance plots for different system sizes of Predator-prey models.	147
7.1	A visualisation of a simulated photobioreactor at a very late time step. Cubes represent algal cells.	152
7.2	The canonical Kawasaki exchange model.	152
7.3	Kawasaki model with weak gravitational forces at different timesteps during simulation.	153
7.4	Competing species in a Kawasaki site-exchange model.	155
7.5	Photobioreactors: Growth stagnation visualisation.	155
7.6	Photobioreactors: Results gathered from modelling a photobioreactor using a conventional method.	156
7.7	Photobioreactors: State diagram of the Photosynthetic factory model with additions.	158
7.8	Photobioreactors: Spatial configurations showing areas of preferential growth.	159
7.9	Photobioreactors: Plot of activation state counters.	160
7.10	Photobioreactors: Fill fraction plot by timestep and decay parameters.	160
7.11	Photobioreactors: Log-log plot of brightness parameter and half-life fill fractions.	161
7.12	Photobioreactors: Growth plots of exponent values against time taken to reach half-life.	162
7.13	Photobioreactors: Plot of illumination function with constant γ and no mutual cell shading effects.	163

7.14 Photobioreactors: Mutual shading effects on illumination absorption.	164
7.15 Photobioreactors: Example of dense culture growth due to photoinhibition and cell shading. . . .	165
7.16 Photobioreactors: Sample fill fractions for varying parameters in a photobioreactor model.	167
7.17 Photobioreactors: Example of isosurface rendering.	167
7.18 Performance results of MOL, including comparison with custom C++ code.	169
7.19 Fitness plot of the recombination of state transition code over 300 runs.	172
7.20 Candidate model program with redundancies removed, as a flow diagram.	173
7.21 Candidate model program in the form of an abstract syntax tree.	173
7.22 Symbol histograms of k -expressions for generations 0, 5, 10, and 40 in a sample run.	174