University of Exeter
College of Engineering, Mathematics and Physical Sciences

Genetic programming and cellular automata for fast flood modelling on multi-core CPU and many-core GPU computers

Michael J Gibson

24 August 2015

Supervised by Dr. Edward C. Keedwell and Prof. Dragan Savić

Submitted by Michael J Gibson to the University of Exeter as a thesis for the degree of Doctor of Philosophy in Computer Science, 24 August 2015.

I certify that all material in this thesis which is not my own work has been identified and that no material has previously been submitted and approved for the award of a degree by this or any other University.

(signature) ……..... M.J.Gibson

Abstract

Many complex systems in nature are governed by simple local interactions, although a number are also described by global interactions. For example, within the field of hydraulics the Navier-Stokes equations describe free-surface water flow, through means of the global preservation of water volume, momentum and energy. However, solving such partial differential equations (PDEs) is computationally expensive when applied to large 2D flow problems. An alternative which reduces the computational complexity, is to use a local derivative to approximate the PDEs, such as finite difference methods, or Cellular Automata (CA). The high speed processing of such simulations is important to modern scientific investigation especially within urban flood modelling, as urban expansion continues to increase the number of impervious areas that need to be modelled. Large numbers of model runs or large spatial or temporal resolution simulations are required in order to investigate, for example, climate change, early warning systems, and sewer design optimisation. The recent introduction of the Graphics Processor Unit (GPU) as a general purpose computing device (General Purpose Graphical Processor Unit, GPGPU) allows this hardware to be used for the accelerated processing of such locally driven simulations. A novel CA transformation for use with GPUs is proposed here to make maximum use of the GPU hardware. CA models are defined by the local state transition rules, which are used in every cell in parallel, and provide an excellent platform for a comparative study of possible alternative state transition rules. Writing local state transition rules for CA systems is a difficult task for humans due to the number and complexity of possible interactions, and is known as the '*inverse problem*' for CA. Therefore, the use of Genetic Programming (GP) algorithms for the automatic development of state transition rules from example data is also investigated in this thesis. GP is investigated as it is capable of searching the intractably large areas of possible state transition rules, and producing near optimal solutions. However, such population-based optimisation algorithms are limited by the cost of many repeated evaluations of the fitness function, which in this case requires the comparison of a CA simulation to given target data. Therefore, the use of GPGPU hardware for the accelerated learning of local rules is also developed. Speed-up factors of up to 50 times over serial Central Processing Unit (CPU) processing are achieved on simple CA, up to 5-10 times speedup over the fully parallel CPU for the learning of urban flood modelling rules.

Furthermore, it is shown GP can generate rules which perform competitively when compared with human formulated rules. This is achieved with generalisation to unseen terrains using similar input conditions and different spatial/temporal resolutions in this important application domain.

I would like to thank my supervisors Dr. Ed Keedwell and Prof. Dragan Savic for their patience, support, vision and experience during my PhD. Secondly I would like to thank both Dr. Albert Chen and Dr. Michele Guidolin whose work with UIM and the CADDIES API/flooding frame work respectively have helped immensely with this project. I would also like to thank my friends and family who support and understanding during my PhD has allowed me to follow my passion; I would especially like to thank my mother Christine McAdam without whose support it would not have been possible to complete this work. I would also like the thank my friends Robert Parker, Ian Alldis, and Ben Morgan for their patience and support. Special thanks to my Badminton crew Paul Ritchie, Lena Sundukova, Arjaree Saengsatien and so many more colleague and friends for their support.

This was partially supported by EPSRC grant number: EP/H015736/1 - Simplified Dual-Drainage Modelling for Flood Risk Assessment in Urban Areas.

Contents

List of Tables

List of Figures

Publications

Some of the material presented in this thesis has previously been published in the following:

Journals:

- "*An investigation of the efficient implementation of Cellular Automata on Multi-core CPU and GPU hardware*", M Gibson, E Keedwell, D Savic, (2015), Journal of Parallel and Distributed Computing 77, p11-25 (available online at: http://www.sciencedirect.com/science/article/pii/S0743731514002044)

Conferences:

- "*CADDIES: A New Framework for Rapid Development of Parallel Cellular Automata Algorithms for Flood simulation*", M Guidolin, A Duncan, B Ghimire, **M Gibson**, E Keedwell, A Chen, S Djordjevic, D Savic, 2012. Proceedings of the 10th International Conference on Hydroinformatics (HIC 2012), Hamburg, Germany, 14-18 July 2012, (available online at: https://ore.exeter.ac.uk/repository/handle/10036/3742).
- "*Understanding the efficient parallelisation of cellular automata on CPU and GPGPU hardware*", M Gibson, E Keedwell, D Savic, 2013, Proceeding of the fifteenth annual conference companion on Genetic and Evolutionary Computation Conference (GECCO), p171-172, Amsterdam, ACM, (available online at: http://dl.acm.org/citation.cfm?id=2464660)
- "*Genetic Programming for Cellular Automat Urban Inundation Modelling*", M Gibson, E Keedwell, D Savic, 2014,Proceeding of the 11th international Conference on Hydroinformatics (HIC 2014), New York, USA, 17-21 August, (available online at: http://academicworks.cuny.edu/cc_conf_hic/414/)

Chapter 1: Introduction

With the urbanisation of the modern world, there is an ever increasing replacement of permeable with impermeable surfaces, which leads to greater amounts of run-off for sewer systems to handle. Furthermore, climate change leads to a greater uncertainty and variability in rainfall. Therefore, the need for computationally efficient flood modelling methods is steadily increasing; both in the need for early warning system and for modern city's resilient design. Such models come in three forms, with each suiting a particular type of modelling: 1D models, which are often used to model flow within pipes and sewer systems, 2D systems used to model overland and pluvial flow, and finally 3D modelling commonly used for free-surface water flow or atmospheric weather modelling. Where lower dimensionality models can be used they are far less accurate for each scenario and with the increased dimensionality there is an increase in computational costs. Similarly, the spatial and temporal resolution of these models maybe increased to obtain greater accuracy, but this comes again at an increased computational cost. For very large resolution models, over large spatial areas and simulation times, or where large ensembles of simulation are required for design validation, the processing time becomes intractable or at least infeasible.

The work in this thesis is concentrated on 2D models for pluvial and overland flooding models; of which there are a wide variety of existing software packages, as shown by the UK Environment Agency Report from 2013 "*Benchmarking the latest generation of 2D hydraulic modelling packages*" [1], with as many as 13 different tools being used. This is due to there being no agreement on the best way to accurately model flooding at various scales, without using full Navier-Stokes based simulations, which are computationally expensive. The majority of these models are based on a regular grid, and on the Shallow Water Equations (SWE) which themselves are computationally complex to solve for large areas. A more recent approach makes use of finite difference models which approximate the derivative of the SWE. These are then solved locally to each cell or inter-cell edge. This approach bears similarities with the computer science technique for modelling complex system, in particular Cellular Automata (CA). These approaches present the opportunity to increase processing speeds

through the use of modern multi-core and many-core technologies, due to the parallelism of the algorithm. Cellular Automata require a state transition rule which governs the change in state of the grid from one iteration to the next, in a similar way to the approximate derivative in the finite difference model. The same rule is processed for many cells and their neighbourhoods, which gives the CA algorithms its inherent parallelism.

In this thesis two main approaches are taken to tackle the computational complexity of the problem within urban flood modelling systems. Firstly, Cellular Automata based models are developed to utilise modern multi-core CPU and many-core GPGPU, and a series of experiments are carried out to understand the efficiency of this parallelisation in relation to the algorithmic parameters. Secondly, as the temporal and spatial resolutions of the CA model directly influence the processing time, then coarser models are required to maintain accuracy to a reasonable level in order to gain further increases in processing speed efficiency. The model accuracy at different resolutions is limited by the local approximation represented by the state transition rule of the CA that drives the various flow rates. The state transition rule must take account of different spatial and temporal resolutions, as well as the various terrain and water levels to produce a reasonably accurate approximation of the globally driven rules. There exists a body of literature where researchers and engineers have derived state transition rules for flood modelling to maximise the temporal resolution of their CA models, while maintaining accuracy at different spatial resolutions. Therefore, experimentation is carried out to understand the feasibility of learning such a specific state transition rule, via the use of an artificial intelligence algorithm. Genetic Programming (GP) is chosen as it is capable of learning from data, and creating innovative results, as well as searching an intractable search space while producing '*good*' solutions ('*good*' in terms of being reasonably close to the global optimum). However, unlike other Evolutionary or Genetic Algorithms, GP does not have a fixed sized chromosome and so is capable of evolving entire computer programs, or formulae. This allows for the possibility of entirely automatic derivation of the local state transition rule from given data, where many other algorithms can only tune a set number of parameters of a model. Having developed a system that is capable of encoding a number of the human formulated state transition rules for flood modelling, these are tested against the

generated rules to make a comparison. The hypothesis tested in this thesis is that the derived local rules should be capable of operating in similar conditions but with different input terrain and water level distributions due to way that they are only programmed with local knowledge. Therefore, the Genetic Programming Cellular Automata system (GPCA) may be trained once, and then allow for the operation of the rules on unseen data sets, and experimentation is carried out within this thesis to validate this theory.

All evolutionary algorithms, of which GP is member, require a method for the evaluation of the fitness of potential solutions. In this case it requires the processing a CA simulation and comparison to example input. The evaluation of the fitness function is known to be the overwhelmingly large computational element of Evolutionary Algorithm's (EA), especially due to the need for the evaluation to be carried out for multiple candidate solutions. The GPCA system proposed in this thesis, requires the simulation of a CA model using the given GP rules as the state transition rule, and therefore makes additional use of the earlier work with many and multi-core processors. This then achieves parallelisation and acceleration of the learning of state transition rules, through the use of a novel combination of the parallelism drawn from both the GP and the CA algorithms together.

Later work in this thesis will tackle the questions of whether the GP can evolve real world CA state transitions rules at a single set of spatial and temporal resolutions and how well these may generalise to other simulation inputs like terrain layouts, and different rates of rain input. The experimentation is then extended to include first various temporal resolutions at a single spatial resolution, and finally a scale of both spatial and temporal resolutions. This method allows for a comparison of the effectiveness of training on a single temporal resolution compared to a spread of temporal resolutions. Finally, this investigation begins to tackle the trade-off between the resolution of the simulations and the accuracy of the simulations produced by each rule. As demonstrated in Chapter 3: (Sections 3.5.1, 3.5.2, 3.5.7), the spatial and temporal resolutions of the simulation heavily influence the real world processing times. Therefore, this trade-off actually weighs the computational time against the accuracy of simulations. This leads to the exciting opportunity to use both

computer science methods and modern hardware to accelerate the processing of real-world urban flood modelling.

1.1 Background

1.1.1 Urban flood Modelling

The underlying physics of hydraulic movement of liquids is reasonably well understood in the Navier-stokes equations [2] [3], which are partial differential equations. Such equations describe the preservation of mass, momentum and energy on a global scale. Therefore, performing full Computational Fluid Dynamics (CFD) is very computationally expensive, as it must simulate many particles in a 3D environment calculating each particle's velocity while balancing the mass, momentum and energy between all particles within the system. For the purposes of modelling very large systems, these kinds of simulations are completely impractical. Therefore, modelling communities have used simplified models, in order to perform simulation in a tractable amount of time.

1.1.2 Cellular Automata

Recently, urban flood models have been based on CA systems, which are locally driven deterministic simulations. CA are based on a grid, where each grid location is referred to as a '*cell*'. Each cell is updated using the same state transition rule, and all cells are updated in parallel. The state transition rule of each cell only uses local neighbourhood state information to that cell. Cellular Automata present an abstract model of complexity based on the emergent behaviour of many simple identical interacting parts. In this way they create an abstract model of the universe, where the laws of physics are encoded as the local state transition rules. Having a single rule which precisely and deterministically describes the movements of fluid over a surface greatly reduces the computational complexity of the hydraulic simulation, over the full 3D Navier-Stokes models. These methods represent water volumes within each cell, and thus represent the 3D structure of the fluids in a 2D format with the addition of water depths within each cell.

In computer science, simple CA models have been well studied [4] [5], such as the game of life (described in section 2.1.1), which only has two states - dead

or alive (zero or one respectively). Even with very few states and a very simple state transition rule, there still are many complex interactions which can occur. However, creating the state transition rules with specifically desired global complex interactions is difficult for human developers. This is because any single change to a state transition rule will affect the entire simulation. Therefore, a way must be found to only affect the desired sections through the complex interactions, known as the inverse problem.

1.1.3 Genetic Programming

Evolutionary and Genetic algorithms (EA/GA) [6] [7] are powerful search methods, although they are limited by the fact that they can only search a fixed number of decision variables. They must use an implicit model based on these decision variables in order to establish the given fitness for each individual. By contrast, GP operates on a variable number of decision variables, and even selects for the important variables as well as optimising the solution. GP can develop entire computer programs, and thus can develop an entire model. Genetic Programming, offers a powerful search algorithm, capable of exploring variable degrees of complexity within its solutions. Parsimony can be included as part of the selection criteria for GP, allowing the evolutionary power to be harnessed to search and optimise the solution's accuracy and parsimony.

1.2 Hypothesis

1. Using Genetic Programming to train the state transition rules for CA, and presenting entire simulations as training data will create rule sets which have good generalisation properties to other unseen initial conditions. I.e. the training set of a simulation may not contain all the state transitions of the target or the underlying CA rule, but by having a distributed rule programmed by GP, the rule generated should interpolate well to other input conditions. Such a method should be applicable to almost any type of CA system, so long as an interface between the local CA rule and GP system can be established. This interface should declare how the GP rule is instantiated within the local CA neighbourhood, in order to guarantee uniformity, and mass preservation if required. Also the interface should

declare how the neighbouring states in the local CA neighbourhood lead to the GP variables and the next cell state output.

In Chapter 4:, the GPCA system uses a Game of Life simulation and attempts to find the Game Of Life rule set. The results show that the majority of rules generated match the Game of Life rules even though the simulation presented does not represent all state transitions of the Game of Life. In Chapter 5: the GPCA system is trained upon the output of a real-world hydraulic modeller (UIM), and is tested on unseen water level inputs, and different terrain inputs to determine the generated rules ability to generalise to unseen data. The limits of this generalisation are seen in Chapter 6:, where the training and testing are extended to include multiple spatiotemporal resolutions, and testing also includes radically different inflow conditions.

2. For real world CA systems, the spatial and temporal resolution variables (Cell size and time step) are static for all cells and iterations of a single simulation, and alter the entire dynamics of the resulting simulation. Training on a number of different simulations, each with different values for one of the simulation static variables of cell size and/or time step, will allow the GPCA system to learn the higher level dynamics. This should create rules which can not only generalise to different initial conditions, but also to different temporal and spatial resolutions. In this way it may be possible to create rules which can operate at higher time step factors than previous rules with acceptable accuracy, thereby producing faster computational rule sets.

   In Chapter 6: training and testing of the GPCA system is conducted at different spatiotemporal resolutions, and testing is also conducted on different water level and terrain inputs. In section 6.2 rules are trained and tested for their generalisation to the timestep property, and by creating rules which can operate at higher time steps, this creates faster rules. Finally in section 6.4, rules generated with both cell size (spatial) and time step (temporal) variation, are used to tackle the more complex trade-off between speed and accuracy, and demonstrate that the system can learn the rules behind this complex dynamic competitively with the most modern human formulated rules.

1.3 Aims of research

The primary aim of this research is to automatically produce fast and accurate flood modelling systems, through the application of computer science methodologies including algorithmic parallelisation and Artificial Intelligence. A further aim is to compare the automatically generated rules against existing (human derived) rules, and be able to compare their effectiveness in a quantitative manner.

1.3.1 Objectives

The following is a list of objectives tackled within this thesis:

1. The investigation of the parallelisation of CA systems upon modern many-core GPGPU technologies, and the effect of varying the standard CA parameter such number of cells, initial configuration and activity, number of states, neighbourhood size, and number of generations on the speed-ups obtained. Also to investigate the effects on the relative speed-ups obtained, of varying GPGPU parameters such as the workgroup size, GPU memory type, and the base data type used to store states. This investigation is intended to ensure that the relationship between the CA parameters and the relative speed-ups of the GPGPU over the CPU are well understood, such that when later work in this thesis can maximise speed-ups from the GPGPU when combining GP and CA systems.

2. The development of a CA system for flood modelling, based on existing models from literature, which is capable of expressing a spectrum/range of variable state transition rules. It is intended that these state transition rules should always ensure uniformity to direction of flooding flow and should preserve the water volume across the grid. This will allow for the derivation of state transition rules which can concentrate on finding the correct flow rates given the water, terrain levels and spatial and temporal resolutions across the grid. Leading to the development of a GP system for the optimisation of CA state transition rules. Such a system should take advantage of previous research conducted after Objective 1, in order to obtain the best speed-ups

possible by accelerating the evaluation of CA fitness functions upon the GPGPU.

3. An investigation of the effectiveness of the combined GPCA system from Objectives 1 and 2, to learn a known CA rule set such as the Game of Life. This will allow for the calibration and confirmation that the system can find the correct underlying state transition rule from an example CA simulation.

4. An investigation of the effectiveness of the combined GPCA to learn flooding modelling state transition rules based on example simulation data.

    4.1. Quantify the simulation time needed during training on a fixed set of spatial and temporal resolutions, and prove that the combined GPCA system can learn state transition rules which are competitive amongst human CA flood modelling rules.

    4.2. The proof of hypothesis 1, through the testing of derived state transitions rules from objective 4.1 on unseen data, including unseen sections of the training test case and completely different terrain.

    4.3. An investigation of the effectiveness of the combined GPCA system to learn flood modelling CA state transition rules which are capable of operating competitively at a range of temporal resolutions. By creating rules which can produce competitive accuracies at higher time step factors (temporal resolutions) than human formulated CA state transition rules, this will begin to tackle the trade-off problem of creating faster rules. Thereby tackling the ultimate aim of creating faster rule sets through the use of machine learning techniques to derive the CA state transition rules for flood modelling systems (hypothesis 2).

    4.4. An investigation of the limits of hypotheses 1, by testing of those rules generated during training conducted in Objective 4.3, upon different inputs including: unseen parts of the training test case, a completely

different terrain, and finally on different '*boundary conditions*' (the type of inflow used in the test cases, e.g. uniform rain, or a lateral inflow).

4.5. Finally, an investigation of the ability of the GPCA system to learn CA state transition rules that can operate successfully at a range of both spatial and temporal resolutions. Demonstrating how the proposed system can adapt to the complex set of inputs including spatial and temporal resolutions, and the local terrain and water levels in order to further tackle the complex trade-off created by the resolution of the simulation (both spatial and temporal) and the accuracy of the resulting water movements over the entire simulation area and duration. A comparison can then be made between the performance in terms of this trade-off with the very latest human formulated CA flooding modelling rules and those generated by the proposed GPCA system (hypothesis 2).

1.4 Thesis structure

Chapter 1: introduces the problem and the scope of this thesis, its background, aims, hypothesis and novelties.

Chapter 2: performs an in-depth literature review, starting off with the origin and purpose of Cellular Automata systems within computer science, leading to their use for real world urban flood modelling. Then the literature pertaining to Genetic Programming, and alternative methods used to learn cellular automata state transition rules is reviewed. Finally, the literature that involves the application of many-core hardware (GPGPU's) to speed up both CA and genetic programming systems is investigated.

Chapter 3: performs an in-depth investigation of the effects of utilising modern many-core GPGPU hardware to speed up the processing of CA. Specifically, this chapter investigates the effects of varying the CA parameters such as lattice size, initial configuration, number of states and amounts of activity, data types, neighbourhood size, and number of generations/iterations. Furthermore, the effects of varying the GPGPU specific parameters such as work group size and GPU memory types used are also investigated. This allows for a

better understanding of how speed-up on modern hardware is affected, and thus leads onto the methodology in the next chapter.

Chapter 4: carries out experimentation using GP to learn CA state transition rules for the simple binary Game of Life rule set. Chapter 4:, section 4.2 introduces the methodology of using genetic programming to find specific cellular automata state transition rules. This chapter details how the GP is interfaced with the CA neighbourhood for the Game of Life. Details of the fitness function and evolutionary algorithm used to drive the GP system are also given here. Lastly, details are given of a novel method of using parallelism for both the GP algorithm population and the CA's cell population. This allows for smaller training cases to be used while still saturating the many-core GPGPU hardware with enough parallel elements, and therefore reducing overall processing time for the optimisation process. Chapter 4:, section 4.3 then describes experiments used to demonstrate this system's ability to find representations of the Game of Life rule set given a target CA simulation. As the game of life rule set is known (i.e., the global optimum for the GP search), the system can be verified before tackling the real-world problem where the best rule sets are not so clear.

The GPCA methodology is then extended in Chapter 5: to tackle the real-world problem of finding rule sets which can perform urban flood modelling. The methodology for the updated interface between the GP and the CA neighbourhood for real-world flood models is described in section 5.2. Section 5.3 details the experimental set-up used for the real-world experiments, including the flooding test cases and the human competitor rules sets from literature. The first set of real world experiments, in section 5.4 are carried out on a fixed set of spatial and temporal resolutions (cell size and time step), and vary the amount of simulation time used for training. Experiments are then carried out in section 5.5 to see how well rules trained on different lengths of simulation generalise to different input conditions. For example, different terrain configurations, initial water levels, and different rain input conditions are used to test this ability. These experiments are intended to prove hypothesis 1, by demonstrating the ability of the GPCA system to learn rules which can then generalise to other initial conditions and inputs (at a single spatial and temporal resolution).

Chapter 6: then extends the experimentation on real-world flood models to include an investigation into the development of rules that can operate at different spatial and temporal resolutions. It is anticipated that this will then tackle the trade-off between the real-world processing time and the accuracy of simulations produced from the trained rule sets, and thereby provide a weight of evidence for hypothesis 2. The trade-off is tackled by training rule sets which can produce reasonable accuracy at competitively high time steps against the latest human rule sets from literature. First in section 6.2, rule sets are trained on a single spatial resolution and a sparse number of temporal resolutions, then the generalisation of Genetic Programs trained on a single time step are compared to those trained on many. Lastly in section 6.4, experiments are carried out to train GPCA state transition rules that are capable of operating over many spatial and temporal resolutions. There is a further relation between the cell size (spatial resolution) and the trade-off of accuracy and computational speed determined by the temporal resolution, which is tackled by training rules for multiple spatial and temporal resolutions and accuracy. Finally, Chapter 7: draws conclusions and final discussions from the thesis.

1.5 Novelty of the work

- Genetic Programming has been used previously to create the state transition rules for Cellular Automata systems, given the expected large scale outcomes. These have only been implemented on small scale 1D CA cases, and where the final solution of the CA was the expected outcome of the combination of many instances of the state transition rule in a spatial configuration. In contrast, this thesis explores a new approach of using GP for the creation of very specific and complex 2D CA simulations, i.e., where the entire course of the CA simulation in space and time is the desired global reaction.

- A novel method for utilising the combined parallelisation of the GPCA system, such that it can be completed in a reasonable amount of time, is developed in this thesis. The method harnesses the parallelism drawn from the both the multiple individuals within the GP population and the multiple cells of CA. This method allows for the use of a smaller target simulation for the fitness function, while still saturating the GPGPU with

sufficient work. Therefore, the method allows the entire process to be completed within a tractable amount of time. Finally, the methodology is extended to include the parallelism from multiple populations of GP individuals, during trials.

- A novel method for the acceleration of processing CA on modern highly parallel GPU hardware is developed and validated. The specific texture memory of the GPU hardware, which has four layers (Red, Green, Blue, and Alpha) is exploited to process 4 cells per thread and to allow for the use of the GPUs wider memory lanes. While the utilisation of these four layers by itself is not unique, the method of folding the lattice such that neighbouring values for cells on different layers can be collected efficiently using hardware '*swizzling*' operations is novel.

- The multi-state interpretations of the Game of life (Sections 3.3.1). These are novel integer state CA state transitions rules, which produce interesting patterns for study. These rules are extensions on the Game of life rules, and reproduce it when the number of states is two (i.e. binary), but produce different behaviour when using greater number of states.

- The interpretation of the Game of Life rule set has an element of novelty, as there are many ways to represent the same state transition rule, using decision trees. Not only are the human formulations of the Game of life decision tree novel, but so are the trained representations.

- The extensive testing of the effects of the CA parameters on the speed-ups obtained from many-core GPU is novel and contains novel discoveries, in that it confirms for the main causes of computational complexity are the number of cells and CA generations including for the GPU. It is found that the speed-ups increase up to a plateau, as the over heads of parallelisation are overcome. Finally, the investigation yields novel discoveries of how the neighbourhood size (amount of memory look-up per cell) and the amount of *'activity'* (number of cells carrying out calculations) affect the relative performance of many-core CPU over multi-core CPU.

- Using an example CA simulation of the Game of Life rule set as target (which only contains a sub-set of all the state transitions) in order to learn the specific under lying CA rule set, has not been attempted before. The use of Genetic Programming on a continuous scale CA system, to learn the binary state Game of Life rules sets, is novel.

- Application of GP for learning the state transition rule of a continuous CA, i.e., real world applications flooding applications.
    - GP has not been applied to learning a real world continuous-CA state transition rule, until now.
    - No one has before considered the effects of the spatial and temporal resolution of a real world model will have on a GP system learning the state transition rules of a CA (sections 6.2, 6.4).
- The comparison on the current best competing urban flood modelling state transition rules from the literature with those created by the automated GPCA system.

1.6 Glossary of terms

1.6.1 Definitions

Chromosome	In an analogy with natural genetics, where a group of individual '*genes*' are often referred to as a chromosome
Cross-over	Recombining two or more candidate solutions to create a new candidate solution.
Elitism	When applied to evolutionary algorithms, this refers to the number of best individuals within the population which are directly passed to the next generation so as the ensure that the best do not get any worse
Fitness	This is an inverse measure of the error of an individual; this allows for a minimisation problem to become a maximisation one and that fitness proportionate selection can be performed.

Gene	An individual variable within the genetic information of an individual candidate solution
Genotype/Genotypical behaviour	The literal encoding of the genetic information of a candidate solution
Locus/Loci	The point or points within the genetic information of candidate solution where the genetic information is divided.
Phenotype/Phenotypical behaviour	The resulting solution created by the interaction of the genetic information of candidate solution

1.6.2 List of terms

ADF	Automatically Defined Functions
ALU	Arithmetic logic unit
ANN	Artificial Neural Network
CA	Cellular Automata
CFL condition	Courant-Friedrichs-Lewy condition
CGP	Cartesian Genetic Programming
CPU	Central Processing Unit
CUDA	Compute Uniform Device Architecture
EA	Evolutionary Algorithm
EAT	Environment Agency Test
FHP	Frisch Hasslacher Pomeau - Lattice gas Boltzmann method
FPGA	Field Programmable Gate Array
GA	Genetic Algorithm
GEP	Gene Expression Programming
Gflops	Giga Floating Point Operation per Second
GOL	Game Of Life
GP	Genetic Programming
GPCA	Genetic Programming Cellular Automata
GPGPU	General-Purpose Graphics Processing Unit
GPU	Graphics Processing Unit
HPP	Hardy Pomeau Pazzis - Lattice gas Boltzmann method
LGP	Linear Genetic Programming
MEP	Multi-Expression Programming

MLP	Multi-Layer Perceptron
MSGOL	Multi-State Game Of Life
NaN	Not A Number
PDE	Partial Differential Equations
RAM	Random Access Memory
RGB	Red Green Blue
RMSE	Root Mean Squared Error
SIMD	Single Instruction Multiple Data
SIMT	Single Instruction Multiple Thread
SWE	Shallow Water Equations
TSP	Travelling Salesman Problem
UIM	Universal Inundation Model
vant	Virtual Ant
VSM	Virtual State Machine

Chapter 2: Literature review

In the following sections, the current literature pertaining to Cellular Automata (CA) is reviewed in section 2.1. This section covers a brief history of CA, and then examines a number of CA models, working towards the more complex continuous CA for hydraulic modelling which are used for flood modelling. In section 2.2, Genetic Programming (GP) literature is reviewed, and section 2.2.3 focuses on the few examples of GP applied to learning CA state transition rules. Finally, both CA and GP algorithms are highly parallelisable, where the fitness function of the GP is known to take the majority of processing time. Therefore the literature pertaining to GPU processing of both CA and GP is investigated in section 2.3.

2.1 Cellular automata

2.1.1 Introduction

The umbrella term Cellular Automata (CA), represents a spatially discrete grouping of Automata (or simple abstract machines), and thus the collection forms a cellular grouping of many small component parts. The most common and basic instantiations of CA use a regular grid, where all the cells follow the same automaton (state transition rule), and commonly a binary state is use. A key element of the CA model is that interactions between the cells are local and parallel. The local cells form what is known as the neighbourhood, which defines which cells are adjacent to which other cells, commonly forming a regular pattern. Cellular Automata are of great interest to the computer science, physics, mathematical fields, due to their theoretical importance and capability to simulate physical systems.

John Von Neumann [8] was one of the founding fathers of field of study currently labelled as Computer Science who, through the study of Logic and Automata, and early digital computers, put forward ideas of a new field between Logic and ‘Neurology’, and noted similarities with the field of thermodynamics [9]. His initial work in philosophy, mathematics, psychology, and neurophysiology guided his attempt to construct a general theory of automata. He used both man-made mechanical and electrical devices, and natural complex mechanisms to forward his argument for the study of how complex systems originate from the

combination of simple mathematical logical operators. Von Neumann was keenly interested in “self-reproducing automata”, whereby the word automata simply means a machine or mechanism of some form.

Von Neumann is attributed with the general architecture of today’s electrical digital computer systems (Henceforth referred to simply as a computer device), known as the ‘Von Neumann Architecture’. Such a computer device requires an ALU (Arithmetic Logic Unit), a memory, and a controller, as well as information buses between the above that connect the controller to the input/output system. The beauty of this architecture lies in its simple generic nature, in that the ALU is capable of processing a number of simple mathematical and logic operations upon some given data. The controller is responsible for collecting the instruction for the relevant mathematical or logic operation, and the data upon which is operates from the memory, delivering these to the ALU, collecting the result and storing it in memory again. The controller is also responsible for interpreting inputs (storing results to memory if required to affect processing), as well as which bit of memory to interpret as instruction and which are data; finally, it is responsible for interpreting the instruction from the ALU or memory to give the necessary outputs. Thus the true beauty in this architecture is that the idea of a ‘program’ has been developed, in a sense a virtual machine, which operates upon a generic simple machine, and exists in the same space as the data upon which it operates. In a sense, a CA is an extension of this design, in that each automaton in each cell represents an ALU. Instructions in CA cells are received from the neighbouring cells data and in a similar way the program and data may exist in the same memory space. So a CA can be viewed as an abstractly distributed computing device, containing many ALU, and where the spatial distribution of data has implications for its purpose.

Von Neumann was critically aware of the other founding father of computer science, Alan Turing; noting that

“*For the question which concerns me here, that of ‘self-reproduction’ of automata, Turing's procedure is too narrow in one respect only. His automata are purely computing machines. Their output is a piece of tape with zeroes and ones on it. What is needed for the construction to which I referred is an automaton whose output is other automata*.” [9].

So Von Neumann's true intention in the creation of his first Cellular Automata is likely to have been the search for truly undirected evolution, and the study of its properties. However, Von Neumann was also keen that these automata not only be self-reproductive but also perform some useful computation, thereby being a truly undirected evolution of computation, which may answer many questions about our own existence. Unfortunately Von Neumann never finished his work in this area due to his tragic early death in 1957 and this work remained under-reported until Arthur W. Burk collected his works on the 'general theory of [complex] automata' together in 1966 [10].

Von Neumann's CA has 29-states and was explicitly designed to demonstrate the idea of a possibly useful CA (or rather a certain level of complexity) capable of self-reproduction. McMullin [11] examines the importance of Von Neumann's contribution to the area of self-reproductive automata, and indeed the enigmatical question of exactly why this interested him so. In McMullin's 2000's paper, he puts forth the idea that the self-reproductive elements Von Neumann's designs are "*trivial,* though highly serendipitous" [11], in that Von Neumann was searching for what he calls "the *evolutionary growth of complexity*". I.e. the question is how do machines construct other machines that are more '*complex*', as we are aware occurs in biological machines (i.e. biological life seems to tend to evolve towards greater complexity). Whereas Von Neumann himself pointed out how it is obvious that most man made machines can only generally construct simpler machines, and are far less resistant to error. His idea was to investigate both the mechanical and computational power that biological evolution has endowed upon us humans.

Von Neumann's original construction can be viewed from a slightly different perspective; i.e. in the light of the Von Neumann architecture, the cellular space and state transition rule represent an abstract view of a complex emergent system based on local state transition rules and from another perspective an abstract model of the universe. Even global rules like gravity (which is the obvious example Burk uses as a rule based on the distance of two objects, i.e. is not locally driven) could plausibly be driven by local approximate rules, given some medium through which to communicate the information, in the case of gravity this is space itself. We must acknowledge that any model we create of the universe will have some element of approximation due to it being a model which is not at

the same resolution as the universe itself. By acknowledging this we know that our model will inevitably have some element of approximation. The base mathematical concept of Finite Volume Methods takes such a global rule as the conservation of mass and energy, and creates an approximate mathematical derivative of the partial differential equations, and in a similar way any rule we generate locally based on such global rules will have some element of approximation, which is covered later in this thesis.

In Von Neumann's original description he made great effort to ensure that the formulation of his state transition rule was not the source of the Turing complete computation capability, but that the state transition rule represents the laws of the abstract universe the cellular space creates [12]. However, in the time since his death other great minds have taken up the idea of the using the cellular space, and different state transition rules to investigate a number of different scientific fields. There are even countless variations including: irregular meshed CA [13] [14] [15], and heterogeneous CA [16] [17] where the state transition rule is different in different cells. However, this thesis focuses on just the 'standard CA' as it is already a large umbrella term. Lastly on the subject of the origin and purpose of CA, one of the classical neighbourhood patterns commonly used is attributed to Von Neumann due to the it's use within the original 29-state CA he developed.

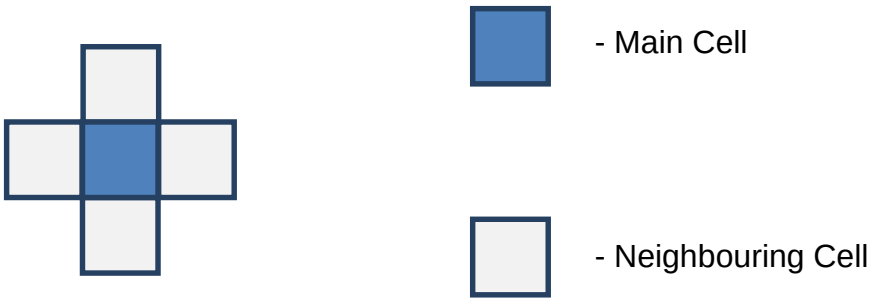


Figure 2.1, Von Neumann Neighbourhood, attributed to the original 29-state Von Neumann Cellular Automaton.

The next advance in the field of Cellular Automata was developed by John Conway with the publication of his initial study of the CA he called 'The Game Of Life' (GOL), in the 1970's [4]. The GOL has only two states (dead – 0, and alive – 1), in a 2 dimensional infinite regular field of cells. The Moore neighbourhood is used which, similarly to the Von Neumann neighbourhood (Figure 2.1) only

includes cells within a radius of 1, however within the Moore neighbourhood interpretation the diagonal cells are included as shown in Figure 2.2.

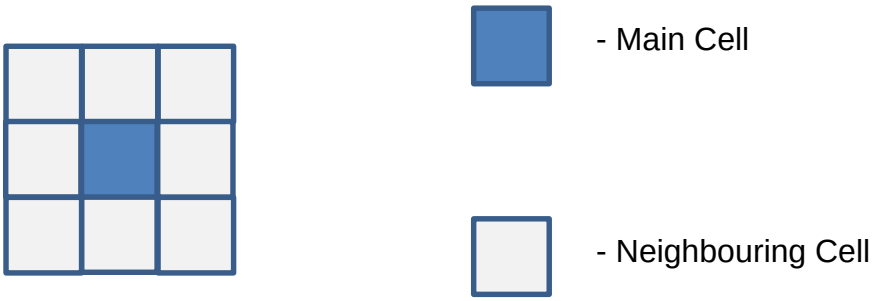


Figure 2.2, Moore Neighbourhood.

The rule set for the game of life is originally described as follows:

1. Survivals. Every cell with two or three neighbouring cells alive, survives until the next generation.
2. Deaths. Every cell with four or more live neighbours dies from overpopulation. Every cell with one live neighbour or none dies from isolation.
3. Births. Each dead cell with exactly three live neighbours (no more, no less), becomes alive in the next generation.

Thus it can be determined what should happen when a cell is currently alive and how it transitions to dead, similarly it can be determined when a dead cell becomes alive, forming a complete state transition rule, for all configurations. Importantly due to the way that the state transition rule is based on the number of live cells within the neighbourhood, it is spatially uniform. These rules were carefully chosen by Conway via experimentation in order to produce interesting pattern development; and at this early point several observations of the global behaviour of the game of life were made, including the nature for apparently random patterns to emerge with a great level of complexity. However, many populations will converge to collections of “still lifes”, which either don’t change at all or oscillate back and forth between two states. The apparent emergence of very complex patterns from simple local rules pushed the study of CA forward greatly; with Conway offering a $50 reward for those persons who could prove the existence of an initial starting configuration which could go on expanding indefinitely. I.e. a “gun” which would produce “gliders”, which would then self-

propagate themselves outwards; or a “puffer train” a configuration which would move and leave a trail of “smoke” behind” [4]. This prize was indeed won with the discovery of a continuous glider gun. Later the game of life was shown to be Turing complete in 1982 [18].

Although as described by McMullin [11] and others like Langton and Herman [19] who have tackled the specific problem Von Neumann faced of producing self-reproductive and Turing complete automata, the Game of Life allowed computer scientists to look at problems from a different angle. If such very simple state transition rules as the game of life can lead to such complex patterns and interactions (to the level of Turing completeness), then how can such simple rules lead to such complex behaviours, i.e. do all rules lead to such behaviour. This problem was tackled by Stephen Wolfram who developed a classification for cellular automata.

In 1984 Stephen Wolfram, made an extensive study of 1D cellular automata in order to investigate the properties of the emergent patterns in space time [20] [5]; whereby he classify the behaviours into four distinct classes:

1. Spatially homogenous, most patterns evolve quickly to a stable state with no change thereon.
2. Sequence of simple stable or periodic structures.
3. Chaotic aperiodic behaviour, nearly all patterns evolve in a pseudo-random way, and any stable structures are quickly destroyed by noise.
4. Complicated localized structure, some propagating; this is the most interesting class which is capable of all the behaviours of the above classes and is thought to be capable of universal computation (Turing completeness).

In fact, Wolfram likened these to the four forms of form language:

1. Regular languages: no memory required.
2. Context-free languages: memory arranged as a last-in, first-out stack.
3. Context-sensitive languages: memory as large as input word required.
4. Unrestricted languages: arbitrarily large memory required (general Turing machines)

Logically also he deduces that if other rules sets can be Turing complete, then the famous ‘halting-problem’ should apply, such that for a given starting configuration it is impossible to determine if it will reach a stable conclusion in a

finite time; of which the game of life is an example in 2D. At the same time as Wolfram studied the more mathematical and computer science (complex systems) elements of CA, Christopher Langton was studying their use further for the study of artificial life.

As well as tackling the self-replication problems from different angles, Langton [19] looks at the deeper question, asked by Lehninger in a previous Biochemistry text:

"*If living organisms are composed of molecules that are intrinsically inanimate, why is it that living matter differs so radically from non-living matter, which also consists of inanimate molecules? Why does the living organism appear to be more than sum of its inanimate parts? Philosophers once answered that living organisms are endowed with mysterious and divine life-force. But this doctrine, called vitalism, has been rejected by modern science, which seeks rational and, above all, testable explanations of natural phenomena. The basic goal of the science of biochemistry is to determine how the collections of inanimate molecules that constitute living organisms interact with each other to maintain and perpetuate the living state…*" [19].

This further brings the study of Cellular automata back toward that of the real world, and not just that of mathematics and computer science, but of underlying physics and chemistry of the real world. Indeed, Langton uses the idea that he is modelling '*artificial molecules*' which are free to roam around in an "abstract computer space" and interact, by means of '*virtual automata*'. Langton performs an in-depth study of several different models of artificial life interactions, and concludes that

"Cellular automata provide us with good artificial universes within which we can embed artificial molecules in the form of virtual automata. Since virtual automata have the computational capacity to fill many of the functional roles played by the primary biomolecules …" [19].

Since these seminal works Cellular Automata have been used to model an enormous variety of different modelling purposes.

2.1.2 Applications

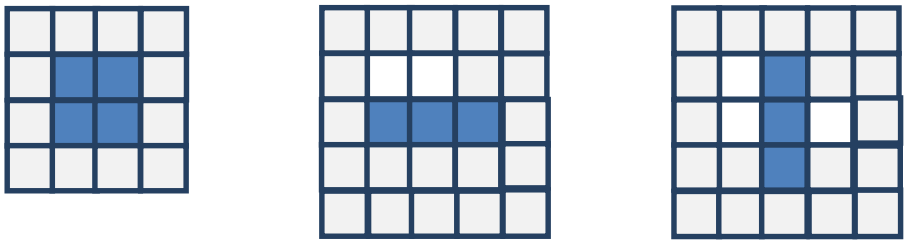
There are a wide variety of scientific papers using CA models, for a large number of modelling uses and disciplines. These applications include but are not limited to:

- Physics (Fluid/turbulent fluid flow [21] – lattice Boltzmann methods [2] [3] [22] [23] [24], reaction-diffusion [25] [26] [27], laser dynamics [28], magnetization [29], collision detection, fracture modelling [30])
- Chemistry/Biophysics (reaction-diffusion [31], artificial life)
- Biology (system biology [32], cell simulation internal-chemical/reaction diffusing within an e-coli cell [31], groups of cells-Keratinocyte skin cells [33], cardiac tissue [34], artificial life/systems [35] [36], tumour cell growth, bacteria swarming, epidemiology [37], viral infection/epidemic spreading [38])
- Computer Science(image processing/visualisation [20] [35] [36] [37] [39] [40] [41], algorithmic study/benchmarking [24] [21] [38] [33] [37] [42] [43] [44] [45] [46] [47] [48], cellular programming/GA/GP-classification [43] [49] [50], cognitive science [51], cryptography [38], computer graphics and animation, distributed computing [27])
- Geography/Environmental sciences (population movements/dynamics [38] [52], land uses/deforestation [53] [46] [54], forest-fires [38], wildfires [55])
- Engineering (wet chemical etching [40], designing hardware (FPGA) to run CA, communications [38])
- Mathematics [5]
- Hydroinformatics (fluid dynamics [24] [21] [49], sewer optimisation [56], pluvial flood modelling [57])
- Economics (stock markets [58] [59] [38])

This variety of applications demonstrates the wide applicability of CA models and in many cases illustrates that the discretisation of time and space for use with a CA model is able to provide results of acceptable accuracy with greater efficiency than traditional models. A number of these models will now be discussed, paying attention to how the state transition rules lead to the overall modelling behaviour, starting with a simpler rule set of the game of life [18], and leading on to more complex rules and behaviours.

The game of life (GOL) state transition rule is remarkably simple, as has been demonstrated above (section 2.1.1). However this gives rise to an even greater variety of emergent behaviours in the form of small collections of live cells [25] [50]. Such behaviours as ‘static lifes’ or ‘still lifes’ which either tend to remain

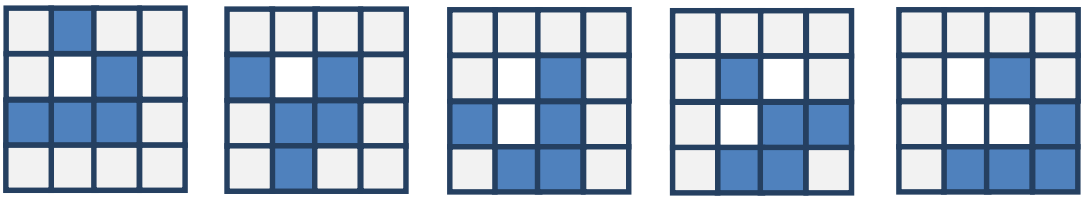
completely static or oscillate back and forth between two states while remaining still in the cellular space; examples include the ‘blinker’ (oscillates with a period of 2 iterations) and ‘block’(a still life) (shown in Figure 2.3).



Block Blinker (1) Blinker (2)

Figure 2.3, A ‘still life’ (Block shown left), and an oscillating life (Blinker shown middle and right).

The ‘Glider’ is another important form of emergent life, which has the property of *moving* through the cellular space; or from another point of view it doesn’t move but rather through its period it recreates a new version of itself in a new position, having destroyed the original version. Shown below is the most famous of these discovered in the game of life, known as ‘The glider’ (Shown in Figure 2.4).



1 2 3 4 5

Figure 2.4, A South-East aligned Glider, Showing the 5 steps required to move the entire glider one step [50].

These emergent behaviours can be observed at various levels; indeed using the knowledge of some of the more basic life-forms, it is possible to construct basic logic gate circuits within the cellular space [50]. All this richness in global behaviour is created from the interaction in space and time of a very simple finite state automaton (Shown in Figure 2.5)

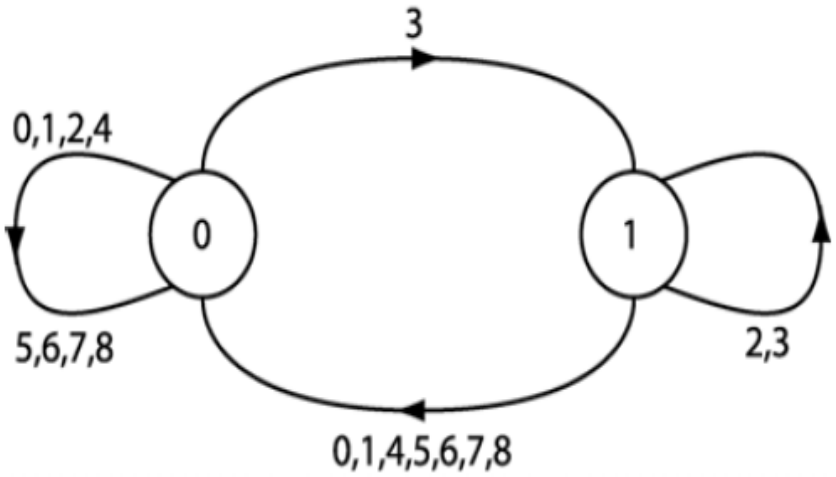


Figure 2.5, The finite state automaton representation of the game of life state transition rule [50].

Figure 2.5 demonstrates that the state transition rules may be represented in a number of forms; so long as the rule is able to create a new state for the main cell based on the states of the neighbouring cells. The GOL rule set requires that the number of live neighbour cells is counted, in order that the finite state automaton is able to produce the new state of the main cell. It is this element of counting the neighbouring cell which maintains the uniformity of both the local rule and global behaviours (for example the same glider behaviour seen in Figure 2.4 can be created in all four diagonal directions).

Stephen Wolfram uses another interesting rule set variation [60], whereby every possible combination/configuration of the main and neighbouring cells is given either a value for the new state (which is either 0 or 1); thus a binary string is created (and more easily read by humans as an integer number). Known as a full look-up table, as the state transition rule no longer needs to count the total live neighbouring cells, it simply determines which configuration within the table matches the current configuration and looks-up the resulting main cell's new state. Such a method does not necessarily preserve the uniformity of the given rule sets if they are varied. However, either method preserves the fact that the state transition rule is completely deterministic, in that for every configuration of the main and neighbouring cells, there is a corresponding new state for the main cell and it is always the same resulting new state.

In 1986 Christopher Langton developed an interesting rule set; which can be viewed as either a Cellular Automaton, or as 2-dimensional Turing Machine

[19] [61]; whereby we imagine that at least one virtual-ant (vant) is placed within the cellular space, and this vant has current direction (North, South, East or West), and all cells have one of two states (known as to-left and to-right). Obviously if we are to view this system as a cellular automaton then we need one single local rule which takes into account the interactions of the vant with the other cells; thus we can create what Langton calls a Virtual State Machine (VSM). It is the VSM that represents the global behaviours of the vant in the local neighbourhood, and represents the concept of the vant *moving* through the system. Langton's Ant rule provides a variation to the Game of life. In the Game of Life, the movement of collections of cells is not directly encoded within the local state transition rule, while in Langton's Ant rule set the vant's movement is clearly written into the local rule. However, if we view the movement/change in the global pattern (i.e. movement of the trail left by vant's) these elements of movement are an emergent behaviour. It is possible to use knowledge of how the vant moves through the system, and *move*/alter the trails in order to construct circuits and logic gates within the cellular space. Using this we are able to construct a system which can simulate the functions of a Turing machine and is therefore capable of universal computation. If we assume there is only ever one vant within the simulation then we would only require a 10 state system, 2 non vant states, and 8 states of the vant (two different states of the underlying system multiplied by the four possible vant directions). In this way Langton has created a local state transition rule which at each turn destroys the vant in its current position and knows which cell should receive the *new* vant. This would generate a uniform local state transition rule, which would be able to receive the vant from any direction dependent on its current state.

An important CA rule set used for hydrodynamics is the lattice-gas or Lattice Boltzmann model. These models have been well studied, and are based on modelling the convection movements within incompressible gases or liquids. Frisch et. al. [2] discuss how a Lattice-gas system is modelled by placing particles of unit mass and unit speed upon the grid, and having these particles moving between cells in the directions of the lattice. No more than one particle is to be found at a given time and node (as shown in Figure 2.6 taken from Frisch [2]). Such systems, model the universe at the level of particles, and use simple deterministic collision rules which will conserve mass and momentum. These are

performed in two stages; firstly, the collision stage using simple bitwise states to indicate the direction, and the second step is one of propagation.

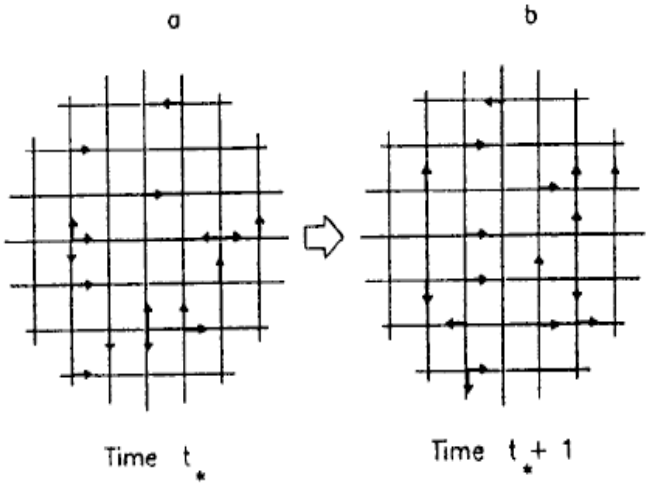


Figure 2.6, The Hardy-Pomeau-Pazzis (HPP) model. The black arrows are for cell-occupation. In (a) and (b) the lattice is shown at two successive times (taken from Frisch et. al. [2]).

More commonly, it has been found more effective to use a hexagonal grid, due to the fact that energy would be conserved within each column and row of a square grid, causing a less realistic spread of particles (Shown in Figure 2.7). Shown in Figure 2.9 and Figure 2.10, are the simple deterministic collision rules for the system.

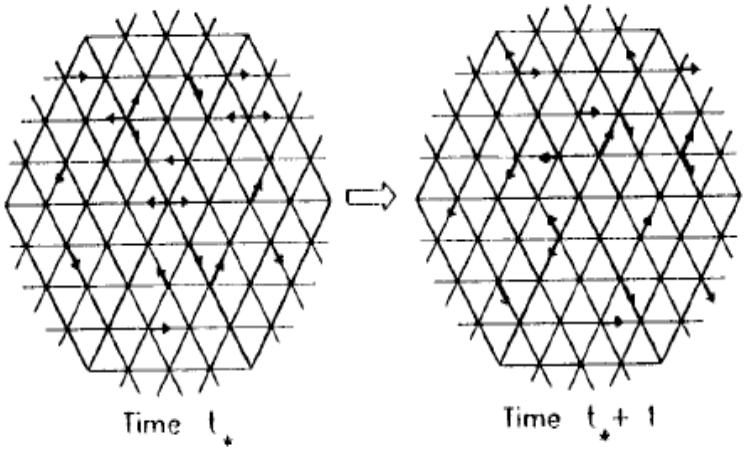


Figure 2.7, The Frisch-Hasslacher-Pomeau (FHP) model with binary head-on and triple collisions at two successive times. (Frisch et. al. [2]).

In actual fact as it is the vertices of the hexagonal grid upon which the particles sit and move, it is actually a triangular mesh used; as shown by Szkoda et. al. [21] in Figure 2.8.

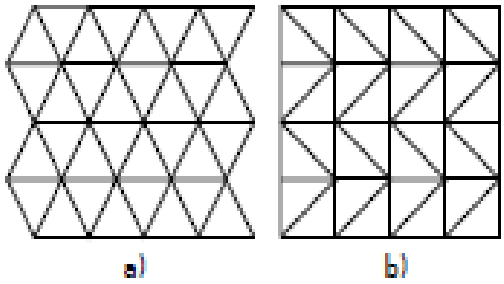


Figure 2.8, Szkoda et. al.'s method for creating a triangular (a) lattice our of regular lattice by shifting every second row by half the lattice constant (b) [21].

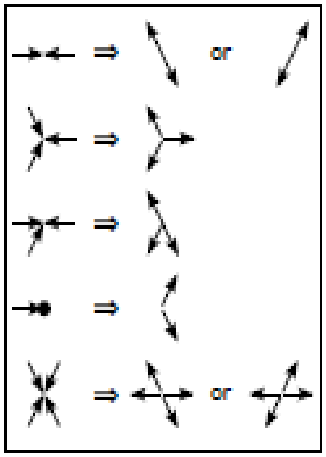


Figure 2.9, Collisions rules for the FHP Cellular Automata system [21].

Figure 2.10, The state of each node is represented by an 8-bit word. Bits 0-5 mapped into particles with given non-zero velocities, bit 6 corresponds to a particle at rest and but 7 controls whether the node is a boundary node [21].

Considering a simple incompressible fluid within a regular square lattice, and using the von Neumann neighbourhood, assume gravity drives the water level down to a median level, and friction stops the majority of the effects of momentum on a flat terrain. In this case, the rule set is very simple, in that water levels within the neighbourhood can be averaged out (mean) to produce the new water level.

While this can produce some very elegant global reactions, it has two major drawbacks in that it can only operate within a flat terrain, and operates at a fix rate with little to no reference to real world simulation time.

2.1.3 Urban flood modelling

The physics of fluid dynamics are well understood, where the Navier-Stokes Equations [3] describe the movements of incompressible liquids such as water, known as the Shallow Water Equations (SWE). These equations are partial differential equations which preserve the volume, momentum/energy of the system. However, solving such equations requires large amounts of computational power as the Navier-stokes equations do not describe the

movements at any particular point, but rather the preservations of volume and energy across the system.

With the increase of urban creep, whereby cities tend to grow and the amount of impervious ground is enlarged, city planners, their residents and planners face increasing challenges with flooding. Also with more volatile and uncertain weather conditions, further increases the problems for engineers in the design of cities and their sewer networks. In order to design better cities, sewer systems and other Sustainable Urban Drainage System (SUDS), and in order to test such system under many conditions before the expensive process of construction, many simplified modelling systems have been created. Due to the need to produce high detail models, and varying conditions such as different weather inputs, and different engineering solutions to high risk flooding areas, or early warning systems, simplified 2D models are generally used [62] [1].

The UK Environment Agency, along with the water companies are responsible for maintaining public water ways and sewer systems in the UK. In this capacity they have benchmark tested a number of state of the art modelling packages including ANUGA, Flowroute-*i*, InfoWorks ICM, ISIS 2D, ISIS2D GPU, JFLOW+, MIKE FLOOD, SOBEL, TUFLOW, TUFLOW GPU, TUFLOW FV and UIM [1]. These packages which use simplified equations, which have been grouped in to three categories:

1. LISFLOOD-FP and RLSM EDA, which solve a version of the SWE neglecting the advective acceleration term (referred to as ‘3-term’ models)
2. ISIS Fast Dynamic, which utilizes Manning’s uniform flow law and UIM which solves the SWE without the acceleration terms (referred to as ‘2-term’ models)
3. ISIS Fast and RFSM direct, which are based mainly on continuity and topographic connective, and therefore predict only a ‘final’ state of inundation, that is, there are no variations in time (referred to as ‘0-term’ models) [1]

The majority of these models use a 2D storage cell system based on a regular raster grid. Due to the simplification of the full SWE equations used by

many modern models, they form a good approximation of the models using the full equation, with much less computational complexity. Processing times are increasingly important for large spatial scale problems (large extent, fine grain resolutions, or even large numbers of simulations), while maintaining reasonable accuracy. In recent years a shift towards localised and even CA models has been proposed to reduce the computational complexity further still.

The models studied in this thesis, are 2D non-inertia models, where UIM [62] and the Hunter [63]/Bates [64] models are based on the Saint Venant equations in which the inertial terms are neglected by the assumption that the acceleration terms of the water flow on the land surface are relatively small compared with gravity and friction terms [65].

2.1.4 CA for Urban flood modelling

Open Channel systems, are examples of continuous CA, which represent the state as a floating point/real value. Open Channel system represent water depths within each cell, also a terrain height is often stored, where the water volume sits on top of the simulated terrain (Shown in Figure 2.11). The terrain levels are fixed and do not change during a simulation, but the water depths and levels change over time.

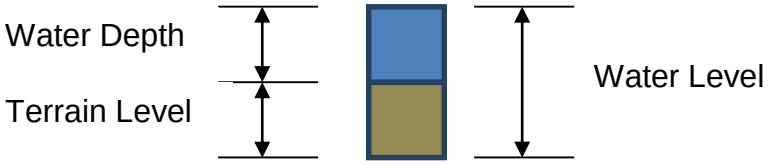


Figure 2.11, Side view a cell represented by the continuous values terrain level, and water depth, which summed together equal the water level, stored within each cell of an open channel CA system.

The global physics of water are well understood, using the Navier-Stokes equations, which preserve the mass, momentum and energy within a system. Open Channel CA simplify this approach by preserving mass locally and therefore globally, and assuming that gravity drives most of the movement, while friction negates most of the effects of conservations of energy/momentum. These are the CA systems which are investigated in more detail, and later in this thesis used for experimentation.

CA systems use a similar two stage system to the Lattice-gas models and must first establish the outflows from a cell in the four orthogonal directions; i.e. using the Von Neumann Neighbourhood, as shown in Figure 2.12 and Figure 2.13.

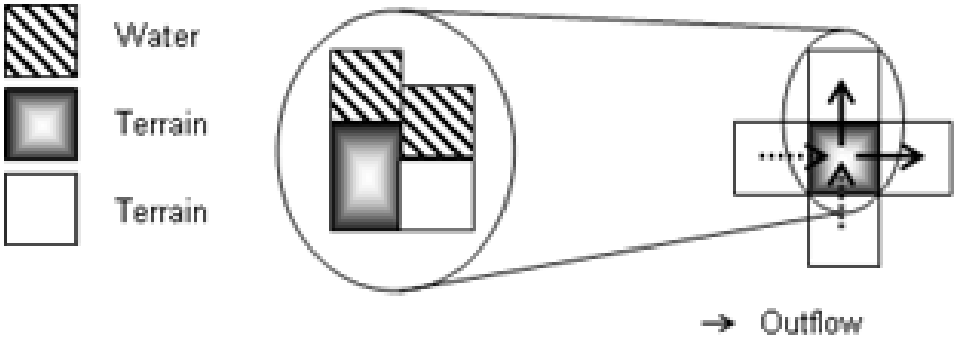


Figure 2.12, Demonstrates how the outflows are calculated within the Cellular Automata system, between the main cell and each neighbour of the Von Neumann neighbourhood. Centrally showing a side view of the terrain and water levels of the selected two cells, and a plan view of the neighbour on the right.

As is common to most known Open Channel CA, they make the basic assumption that the water can only flow from the cell with the higher surface Level to cell with the lower water surface level, which establishes the direction of flow (also assuming momentum is largely negated by frictional forces). In order to balance the water volume, the model only calculates outflows from each main cell to its neighbours, as those neighbours where the water level is higher, will themselves create an outflow to this main cell. The total amount of water leaving a cell in up to all four directions cannot be allowed to exceed the volume of water within the main cell otherwise volumes will be created and/or destroyed across the grid. Therefore a two stage system is utilised where in the first stage all the cells established outflows to all the necessary neighbours, as shown in Figure 2.13. A second stage then removes outflows from the current water level, while adding inflows from other cells and in doing so balances the water mass across the grid while allowing for lateral and horizontal movement of water.

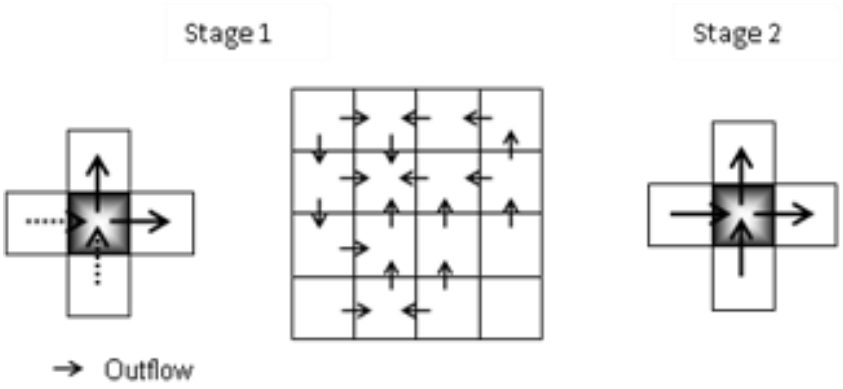


Figure 2.13, The two stages of the CA flood system. Stage 1 for every pair of cells an outflow is calculated, stage 2 every cell updates water depths by means subtracting outflows and adding inflows.

This kind of two stage CA system is rather different from most previous models but maintains the key elements of local state transition rules, which are uniform in each direction. They are also complete in that they provide an output for every possible input. However, this is in the form an equation commonly representing the flow rate from one upstream cell to another. In the following sections a number of key open-channel systems are investigated.

2.1.4.1 Dottori and Todini technique

A key example of this kind of Open Channel CA is that of Dottori and Todini [66], as they use a very direct method of calculating the outflows from each cell to its neighbouring cells. As has become a common approach with open channel techniques, the Manning's formula is utilised (Equation 2.1), which can calculate the flow in an open channel such as a river for example. The Manning's formula only calculates the flow rates, and needs to be coupled with the discharge formula, shown in Equation 2.2, in order to calculate the volume of water transferred in the given time step.

Equation 2.1 $$V = \frac{1}{n} R^{\frac{2}{3}} S^{\frac{1}{2}}$$

Equation 2.2 $$Q = VAT$$

The Manning's formula is shown in Equation 2.1; Where V represents the volume metric flow rate, n is the Manning's frictional coefficient, R is the hydraulic

radius, and S is the hydraulic gradient. The discharge formula is then used to calculate the volume of water transferred, shown in Equation 2.2. Where Q is the transfer volume, A is the cross-sectional area of flow, and T is the amount of time at this flow rate. The hydraulic radius (R) interpretation differs from the Ghimire [65], Dottori and Todini [66], and Bates [64] methods; and the hydraulic gradient (S) is the difference between the water levels divided by the distance between the centroids of the cells (which in a regular grid is the cell size). The Dottori and Todini method uses the arithmetic mean between the main cell and the neighbouring cells depth, to calculate the hydraulic radius (R) in the Manning's formula. They are unclear as to exactly how they control the flows when the total from one cell exceeds the volume present, saying:

"Every discharge calculation step includes a control on volumes; which avoids that the volume of water flowing out of a cell is greater than the sum of the volume stored in the cell itself and the incoming volumes from adjacent cells." [66].

The system then proceeds to use two simple test cases to test the stability of the schema. Firstly, an open 1D channel of length 50km, and width 250m, and a slope of 10^{-4}, and a Manning's roughness factor of 0.05 $m^{-1/3}s$. Using 3 different cell sizes, of 125mx250m, 250mx250m, and 500mx250m, Dottori and Todini test different spatial resolutions (notably by varying the longitudinal grid resolution). Their results are validated against the Hydraulic Engineering Centre (HEC), Hydraulic Reference Manual (HEC RAS). I.e. this is a well-known hydraulic test case. Table 2.1, shows their results, where they test at various time steps on each of the cell sizes.

Table 2.1, Dottori and Todini results on the open 1D channel tests at various cell sizes, and time steps. Where an "N" indicates that the simulation produces significant oscillations on the solution, while "Y" indicates a stable solution. NC ts indicates the minimum time step computed by the Neumann condition (discussed later) [66].

$\Delta X - \Delta t$	1s	2s	5s	10s	15s	30s	60s	NC ts
125m	Y	Y	N					≈ 1.2s
250m	Y	Y	Y	N				≈ 5s
500m	Y	Y	Y	Y	Y	Y	N	≈ 22s

Here in Table 2.2, Dottori and Todini note that "*As expected, the reduction of cell size decreases the model stability, and vice versa; however, the accuracy*

of the solution is not compromised by spatial discretisation until water level oscillation becomes significant " (Shown in Table 2.2). This demonstrates how there is a relation between the cell size and the maximum time step, at which their schema is capable of operating. They use a common approach of calculating the RMSE (Root Mean Squared Error) of their model against their control model (HEC-RAS), and find the maximum to be well below 4cm. It is also noted that the control mode (HEC-RAS) uses the full De Saint Venant Equations. Processing of their simulation is performed in less than 4 seconds, showing the computational efficiency of the local schema.

The second test case is that of horizontal plane, as they note "*routing on a flat slope is a case in which hydraulic models may be more subjected to instability, particularly when flow velocity and water surface slope are reduced. Considering explicitly diffusive models like CA models, the two instability factors are the use of high resolution grids (cell sizes below 10m) and the presence of deep water stages*." [66]. They use a horizontal plane, at two grid resolutions of 10x10 cells of 20x20m size, and 20x20 with a 10x10m size. The water depths are set to 3m across the whole plane, with no subsequent incoming water. The water is drained via a weir located in one corner. They also noted that "*such initial conditions are chosen because tests with incoming discharge have shown that the magnitude of oscillations seem to be only a function of the water stage and cells size*" [66]. The results are shown in Table 2.2, and Figure 2.14.

Table 2.2, Results for Dottori and Todini case 2, where a "N" indicates that the simulation produces significant oscillations on the solution, while "Y" indicates a stable solution. NC ts, indicates the minimum time step computed by the Neumann condition.

$\Delta X - \Delta t$	0.01s	0.02s	0.05s	0.1s	0.2s	NC ts
10m	Y	N				$\approx 5*10^{-4}$s
20m	Y	Y	Y	N	N	$\approx 2*10^{-3}$s

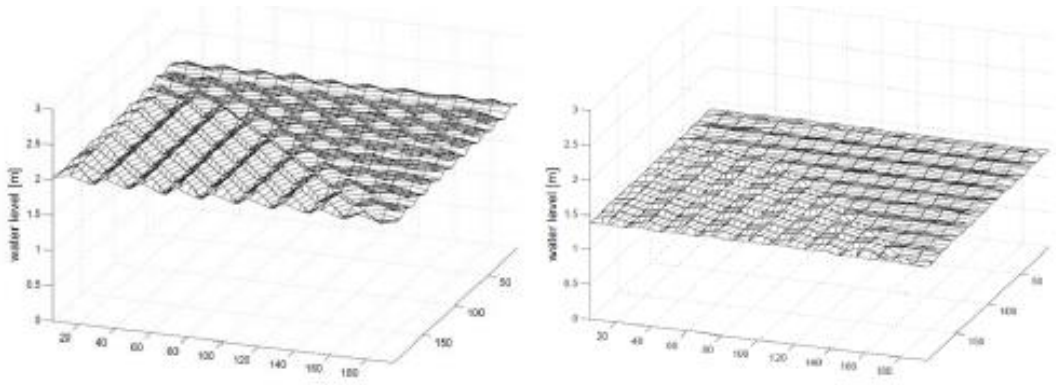


Figure 2.14, Dottori and Todini, case 2, water stages/depths computed by CA model after 30 minutes (left) and 1 hour (right) from simulation start. The outlet is located in the lower right corner.

From Table 2.2, and Figure 2.14, it can be seen that the instability increases when increasing the time step or decreasing the cell size, and vice versa. They conclude that stability is primarily dependent on the spatial and temporal resolution, like other explicit models, and therefore a stability condition is required. They first consider the Von Neumann stability condition, shown in Equation 0.3 [66]. Where Δt is the minimum time step, and Δx is the cell size, n is the manning roughness coeffiecnt, R is hydraulic radius and S is the hydraulic gradient.

Equation 2.3 $$\Delta t = \frac{\Delta x^2}{4} min\left(\frac{2n}{R^{5/3}} S^{\frac{1}{2}}\right)$$

Importantly they note that the FLO2D model, based on the full shallow water equations, uses the Courant-levy-Friedrich (CLF) condition, shown in Equation 2.3.

Equation 2.4 $$\Delta t = C\Delta x/(v)$$

Where Δt is the time step, C is a coefficient which depends on the adopted explicit algorithm, Δx is the cell size, and v is the velocity (generally the largest within the grid); although Dottori and Todini note that the CFL condition is not suitable for CA models since the diffusive approximation needs more stringent conditions. However, it was found that the models maintained stability with greater time steps than those computed by the Neumann stability condition (Equation 2.4).

2.1.4.2 Ghimire et. al.'s technique

The work by Ghimire et. al. [65] distributes the water from the main cell to the lowest downstream neighbour, before working its way up to the next most downstream neighbour until the water levels are matched, or all the water of the main cell flows out to the downstream neighbours (as shown in Figure 2.15, taken from [65]). They then use the Manning's formula to establish how much time has passed given the largest flows (both within each neighbourhood and then globally across the grid), thus allowing it to adapt the time step to the flow conditions. The method uses the water depth within the main cell as the hydraulic radius (R) within the Manning's formula. The authors found it necessary to use a relaxation parameter in order to stop excessive oscillations from occurring as it over-shot in some areas, by trying to go at the maximum limits of the CFL condition.

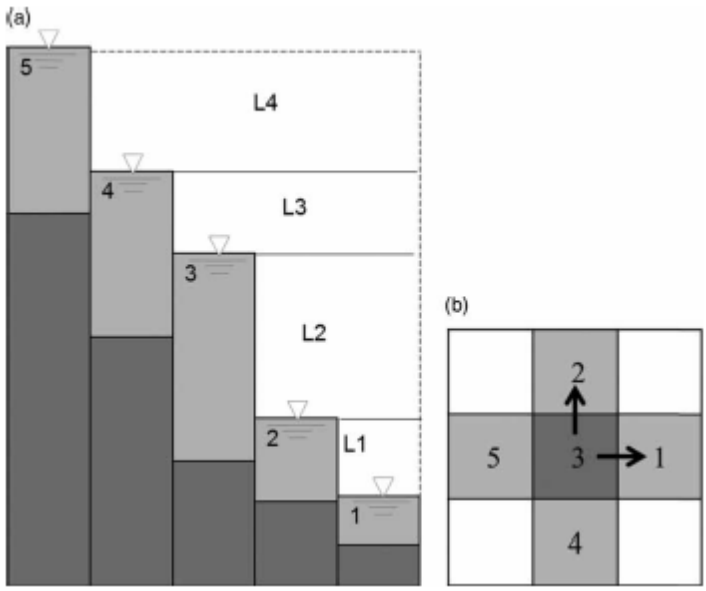


Figure 2.15, Ghimire CA flooding state transition rule: (a) Cells ordered in NH according to their ranks;L1-L4 are layers of free spaces between the water levels of the two cells that area available within NH for water distribution, the numbers shown are cell ranks. In this diagram the ground level for each cell is shown in dark grey and the water level light grey, (b) an example of the outflow fluxes (shown by arrows) from the central cell having rank 3 to its neighbouring cells [65].

They then employ a hypothetical terrain for testing, consisting of 30 x 20 cells, at a 50m resolution; "*The terrain consists of both forward and reverse slopes of 0.2%. It also has a lateral slope of 0.1 toward the outlet*", where the outlet was removed for consistency (Shown in Figure 2.16). A roughness factor

0.01n is applied across the terrain, and a rain fall of 20mm/h for the first hour of the simulation is used as input for the water depths.

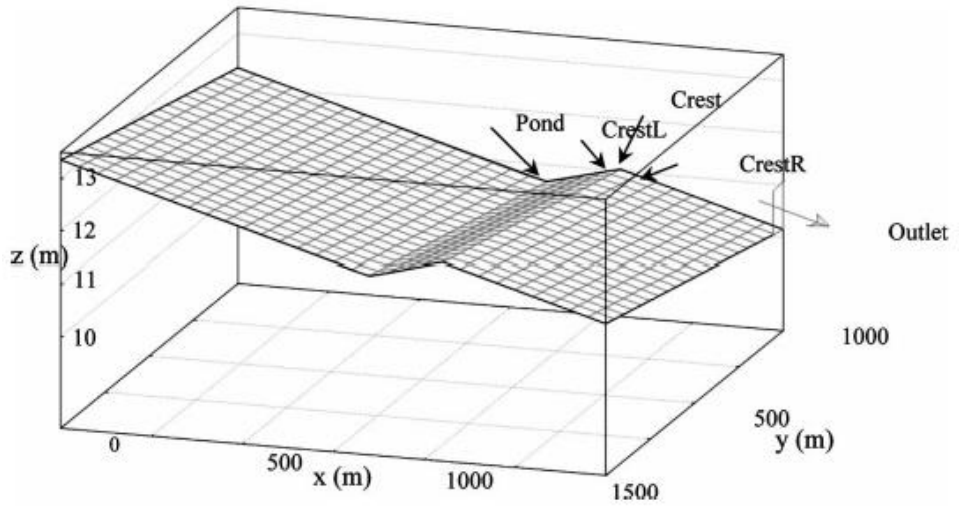


Figure 2.16, Hypothetical 'Hill and Pond' terrain, and given test points; taken from Ghimire et. al. [65].

The Ghimire rule set represents a truer CA approach in that the rule set for establishing outflows is based on the neighbouring values, as opposed to a formula between a pair of cells replicated for each neighbour. However, it then only uses the downhill neighbours, and calculates the outflows for each edge in a similar 2 stage system to the Dottori and Todini technique. Ghimire et. al. do use a novel ranking (or ordering) system to establish outflows within their rule set. Shown in Figure 2.17 are the results of the of the Ghimire et. al. [65] model compared to the UIM model [62] on the Hill and Pond test case. This test case is later used for training and validation in Chapter 4:.

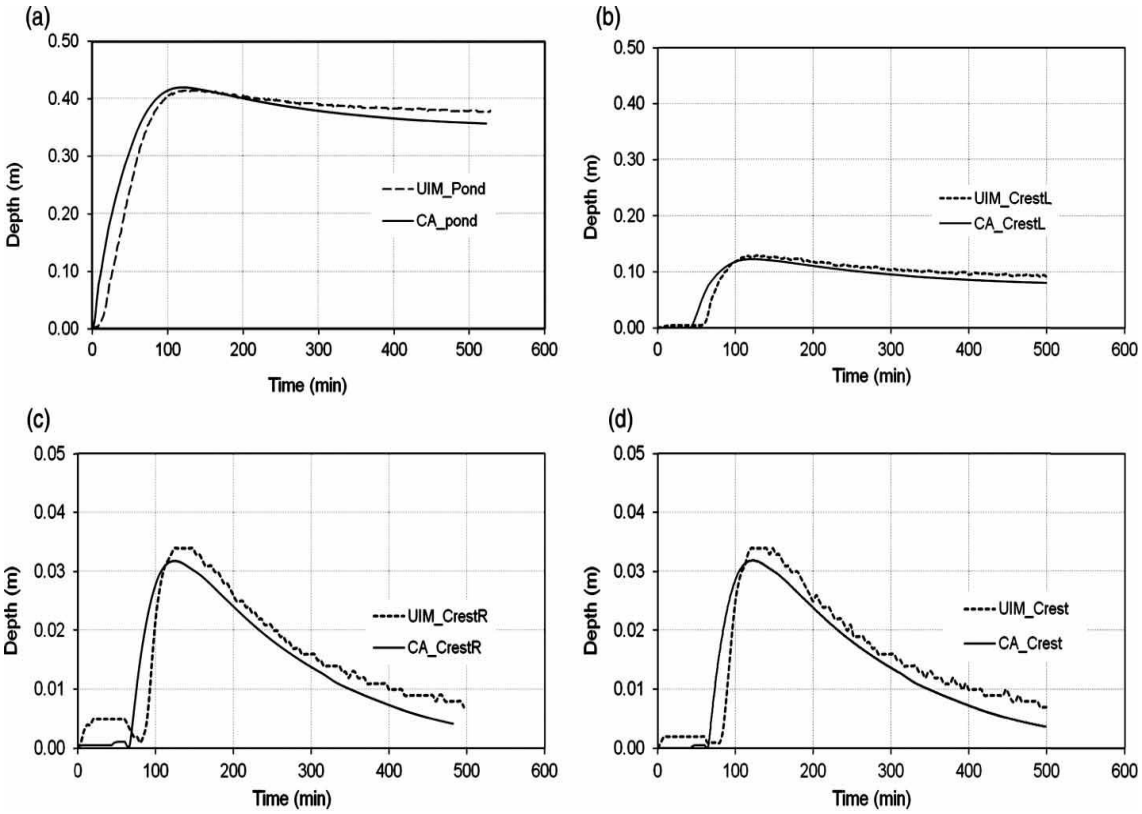


Figure 2.17, Resulting water depths using UIM and the Ghimire et. al. rule [65] at the (a) pond,(b) left of crest, (c) right of crest, and (d) crest points of the hypothetical 'Hill and Pond' terrain, and given test point.

It is noted that the results shown in Figure 2.17 are using a 0.7 relaxation parameter setting. It should also be noted that they [65] use a capped version of the Manning's formula in their time step calculations. Ghimire et. al., also use a real world test case, with a 2m resolution, Keighley from the UK, to test the rule set (shown in Figure 2.18), Consisting of 377 x 269 cells.

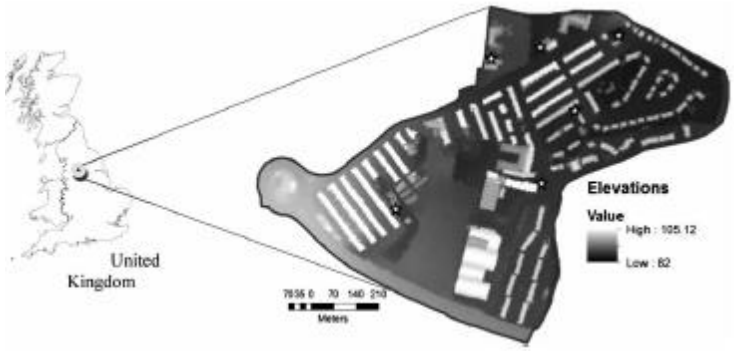


Figure 2.18. Stockbridge Keighley terrain, with sample points 1-6 drawn [65].

The abstract shape of the terrain is represented by using an encompassing regular terrain and 'no data' cells where the terrain extent is not covered. The

resulting comparisons of the water depths for the rule set on the Keighley test case are shown in Figure 2.19, for the 6 test points.

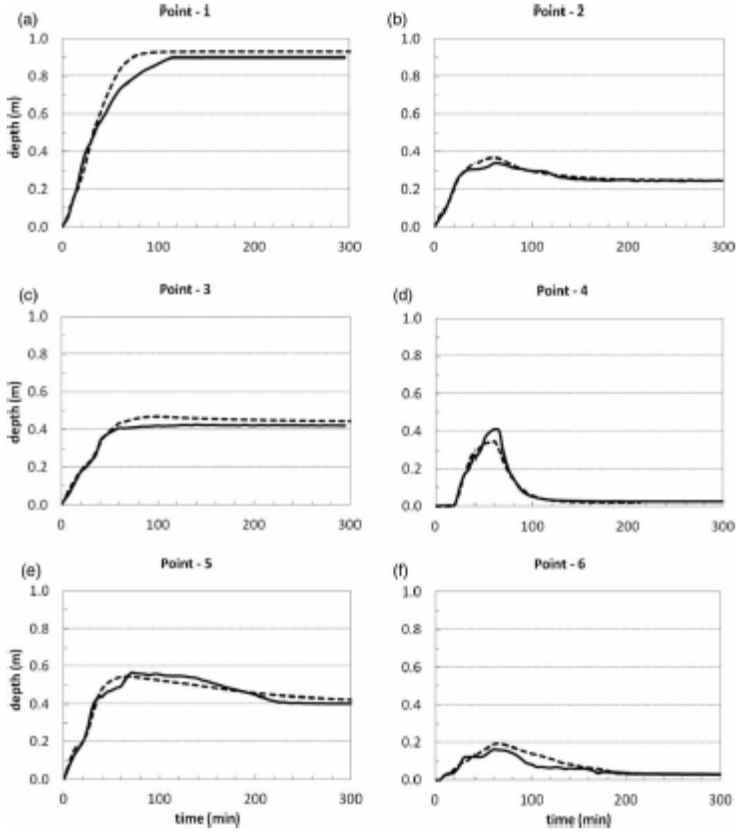


Figure 2.19, Resulting water depths from UIM and the Ghimire rule set, for the 6 test points of the Keighley test case [65].

The results from Figure 2.17 and Figure 2.19, show there is certainly a degree of variation between the models, however the general trends are also certainly followed. In order to calculate the time step that generates a stable simulation the authors use the CFL condition, in a slightly different form to that of Equation 2.3, shown in Equation 2.4. Although the use of a relaxation parameter within the main rule set, means that the entire process can be scaled back for a particular amount for a particular simulation, and they conclude that a method is require for the automatic calibration of this relaxation parameter (α).

Equation 2.5 $$\Delta t = \alpha \Delta x / v$$

2.1.4.3 Hunter and Bates et. al.'s technique

First established by Hunter et. al. in 2005 [63], then advanced by Bates et. al. 2010 [64] this method functions by firstly simplifying the Manning's formula

and discharge formula together and separating out the time step element (shown in Equation 2.6). I.e. the volume transfer rate is calculated.

Equation 2.6

$$Q = \frac{1}{n} R^{\frac{5}{3}} S^{\frac{1}{2}} \Delta y$$

They also use the difference between the water level (free surface height) in the main cell, and the highest of the two terrain levels between the main and neighbouring cell, to calculate the hydraulic radius (R). This key element ensures that where the water level of the main cell is higher, but the terrain level of the main cell is lower than the neighbouring cell (as shown in Figure 2.20), then water is only allowed to flow through the smaller area. The hydraulic radius (R) has been calculated as the entire water depth of the main cell (Ghimire), or the mean of the two cell water depths (Dottori and Todini).

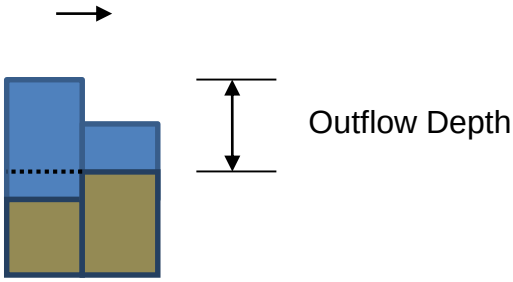


Figure 2.20, A pair of cells, where the left cell is the main outflowing cell, as it has the higher water level. However, terrain level of the main (left) cell is lower than that of the cell it is outflowing to. It makes sense that water between the dotted line and terrain level of left cell, shouldn't be included in outflow calculations, as it is the higher water level that drives the outflow.

The area of outflow (A), which is the cell size, multiplied by the outflowing depth, is factored into the Manning's formula leaving just the time step outside (shown in Equation 2.6). Notably the hydraulic radius in this formula is raised to the power of 5/3, as opposed to 2/3 in previous techniques, which would then multiply by the outflowing depth (which is equivalent to the hydraulic radius). Having then multiplied by the cell size, the outflow area (A) from the discharge formula is completely factored into this single formula, leaving the time step as an independent variable. Apart from the change to the hydraulic radius (R) interpretation in the above techniques, these changes only constitute mathematical simplification and manipulation.

However, Hunter and Bates et. al., develop a key technique for limiting the flow rates, by considering the physical limitations of applying the Manning's formula to such a CA system. That is, the Manning's formula calculates a flow rate (or velocity per unit area) in the given lateral direction across the grid, however when this is combined with an especially large time step, the flow according to the Manning's formula will cover a range greater than that of next cell (neighbourhood radius), as shown in Figure 2.21. As it is not possible for water to move further than a radius of single neighbourhood within a single iteration, then these excessive flows cause an incorrectly large water level in the next cell in the next iterations, as opposed to correctly flowing further laterally. This in turn cause the false diffusion effect shown in Figure 2.22, whereby even one incorrectly large flow causes feedback that spread across the simulation destroying its overall quality.

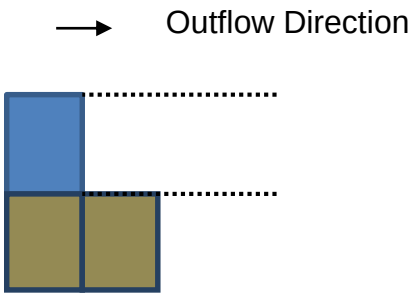


Figure 2.21, Demonstrates what the physics of the flow rate means, i.e. that water will perturb through the given area, by multiplying the time at that flow rate, finds the distance of flow. Effectively the entire block of water is seen to have the given velocity.

This also occurs when the water levels are very similar, as is shown by Hunter & Bates [63] work in Figure 2.22.

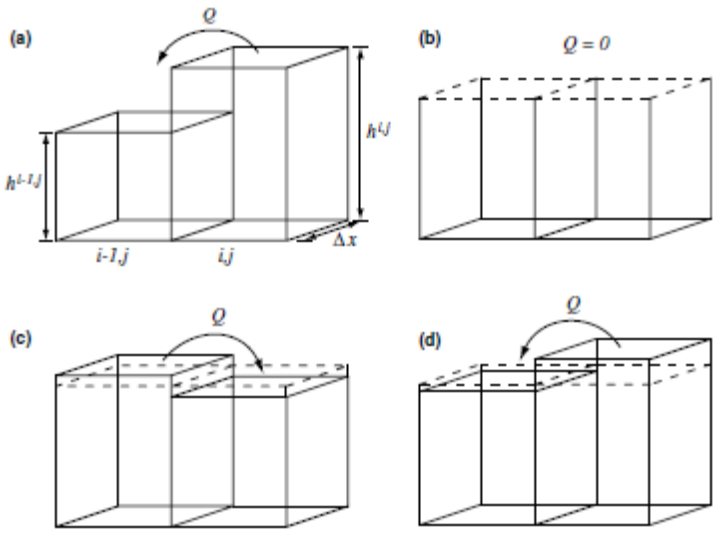


Figure 2.22, Illustration of the chequerboard oscillations between two adjacent cells [63]. (a) At end of time step t, the level in the cell i, j has for the first time risen above that of cell i-1,j. (b) At the end of the time step t + Δt, the discharge from i,j to i-1,j, should be equal to zero as the levels in each cell are equal. (c) However, an oscillation begins to develop as a result of the low free surface gradient between the two cells. (d) The erroneously high flow causes a back flow at t + 2Δt.

Therefore Hunter and Bates, develop a flow limiter to ensure that the flow does not 'over' or 'undershoot', and is a function of flow depth, grid cell size and time step (Shown in Figure 2.23).

$$Q_x^{i,j} = \min\left(Q_x^{i,j}, \frac{\Delta x \Delta y\left(h^{i,j} - h^{i-1,j}\right)}{4\Delta t}\right)$$

Figure 2.23, Flow limiter formula, used by Hunter and Bates et. al. where the flow rates are first calculated by the Manning's formula (Shown in Equation 0.6), then the minimum between the above and that outflow is calculated [63]..

Hunter and Bates note that "*This limiter replaces fluxes calculated using Manning's equation with values dependant on model parameters, and hence when the flow limiter is in use floodplain flows are sensitive to grid cell size and time step, and insensitive to Manning's n*." [63]. While they note there is still a stability issue with small cell sizes, and/or high time steps; stability is increased over that of previous works by a factor of 2, and the Hunter-Bates rule set with the flow limitation uses the Von Neumann stability condition shown in Figure 2.24. This operates at much higher time steps than previous formulations.

$$\Delta t = \frac{\Delta x^2}{4} \min\left(\frac{2n}{h_{\text{flow}}^{5/3}}\left|\frac{\partial h}{\partial x}\right|^{1/2}, \frac{2n}{h_{\text{flow}}^{5/3}}\left|\frac{\partial h}{\partial y}\right|^{1/2}\right)$$

Figure 2.24, Hunter and Bates et. al. formulation of the Von Neumann stability condition, which the minimum flow in the neighbourhood, and the square of the cell size to calculate the time step for stability.

2.1.5 Conclusion

This section has studied the applications and science behind Cellular Automata. There are many different applications of CA, for a wide variety of fields of study and modelling environments. There is an even greater variety of methods for the definition of the state transition rules, however the majority of rule sets and methods for defining variable rule sets are explicitly designed to ensure that provide the same result given the same input from different directions (referred to as uniformity to direction). The design of the state transition rule is highly dependent on each application.

While limitations and approximations of the systems exist, overall CA are seen as a good modelling system. The models investigated in this thesis are limited to non-inertia models, being based on the Saint Venant equations in which the inertial terms are neglected by the assumption that the acceleration terms of water flow on land surfaces are relatively small compared with gravity and frictional terms [65]. CA are included amongst these 2D non-inertia models and are shown to be capable of making a reasonably good approximation of actual flow patterns. Due to the local nature of the CA, there is a limiting factor for the maximum flow rate, as flows can only propagate across a single cell within a single iteration (known as light speed within CA). Excessive flows have been shown to cause large oscillations, which destroy the quality of the overall simulation. A number of flow limiters have been created however there remains a relation between the maximum time step, the cell size, and the flow rates. Within these limitations CAs can offer a good approximation of the global mechanics of fluid dynamics while using a local state transition rule to drive the system. By using a locally driven rule, simulation can be performed in a much less computationally complex framework. Furthermore, CA and such locally driven

systems are inherently parallel, which may make good use of modern hardware to further speed up the production of reasonably good simulations.

2.2 Genetic Programming

2.2.1 Introduction

Genetic Programming (GP) first used by J Koza [6] [7] is an evolutionary algorithm which uniquely operates on a variable sized chromosome, commonly in a tree structure. The algorithm maintains a population of candidate solutions, with each tested to establish a fitness score, where fitness is calculated by testing each candidate solution and finding its error compared to some given model. Fitter candidate rules are selected stochastically, taking into account the fitness of the solution, for example by using fitness proportionate roulette selection or tournament selection. The selected candidates then have genetic operators applied upon them, such as crossover and mutation. A given set of operators and terminal values are used in the tree structure, with some operators such as division needing protection from spurious inputs (e.g. divide by zero).

The umbrella term of Genetic Programming covers most techniques that evolve computer programs and there are many different varieties of GP including: Cartesian GP [67] [68], Linear GP [69] [70] [71] where linear GP also Includes Multi-Expression Programming (MEP), Gene Expression Programming (GEP) and Grammatical Evolution (GE) [72]. However, with the ‘standard’ Koza style GP the key element is that they operate upon a tree structure (or program, as they too can be represented as an abstract syntax tree). A program that is a list of instructions may be viewed as a tree structure, or a basic mathematical formula (as shown in Figure 2.25, [73] which shows the GP parse tree for Equation 2.7).

Equation 2.7 $(x * x) + 2$

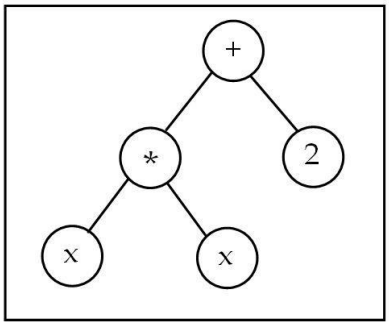


Figure 2.25, A very basic GP parse tree for Equation 2.7.

The key underlying idea, and variation between Genetic Algorithm (GA) and Genetic Programming, is they will operate upon/represent a variable size of chromosome. The tree structure is ideal for representing logical groupings (or nested groups) of functions and operators (sub-formulae), which in the case of Equation 2.7, might be the $(x * x)$ element/sub-tree. The reason for this representation is primarily to allow for cross-over of two different sized chromosomes. While GA and EA produce a static sized chromosome which tends to require further interpretation, GP produces a variable program or formula. Where machine learning techniques such as an Artificial Neural Network use a fixed number of hidden units and can only match a certain degree of polynomial, GP can express both a very simple formula and a very complex one, which allows for a lot more freedom of movement within the search space.

Crossover in GA and EA is easily performed because the two parent chromosomes will be of the same length. They can therefore be aligned easily and equal amounts of material can be exchanged. This will either be a multi-point crossover or a single point crossover, where genes are taken from one parent chromosome up until one of these locus points, and then genes are taken from the alternate parent until the next locus point (a 2 point cross-over of linear chromosomes of the same size is shown in Figure 2.26). Often GA systems will use a probability of selecting a gene from either particular parent. However, GP has variable length chromosomes, and so these simple forms of cross-over are not so easily executed.

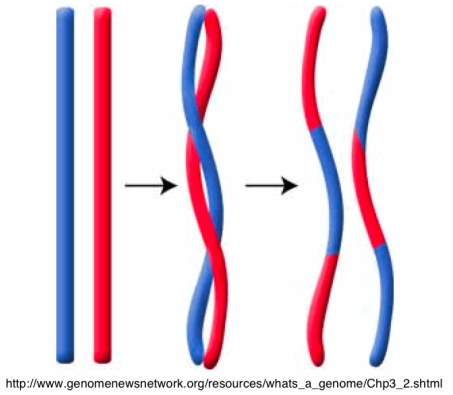


Figure 2.26, 2-point cross-over of same sized linear chromosomes, commonly used in EA and GA systems.

To tackle this, Koza developed the sub-tree crossover system as shown in Figure 2.27, whereby a sub-tree is selected from both parent trees, and both entire sub-trees are exchanged between the position where the former sat in the opposite parent tree. This is analogous to the 2-point cross-over shown in Figure 2.27, in that a block of one parent is exchanged with a continuous block from the other parent. However, a major difference is that the locations of the blocks of code can change, and therefore so can their form and function.

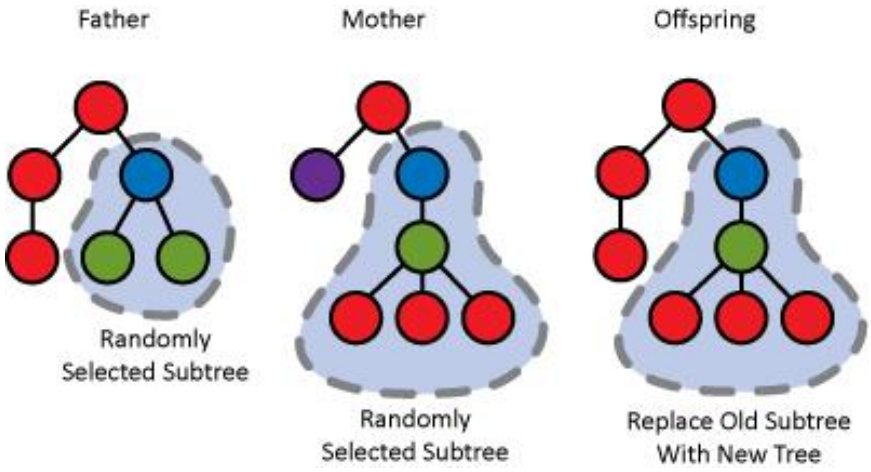


Figure 2.27, Sub-tree cross-over in GP, two different sub-trees are selected from the two parent trees, and exchanged to create the new off-spring tree [74].

Due to the tree structure, expressions are commonly expressed in reverse Polish notation, for example using languages like LISP. An example LISP expression is shown in Figure 2.28, demonstrating how more complex formulae can be composed, and commonly 2-3 branch nodes are used.

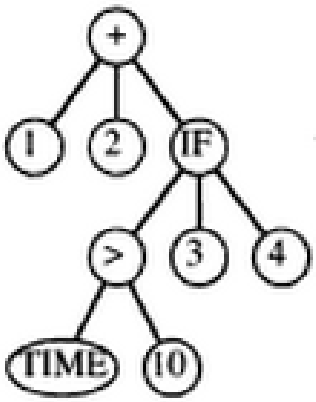


Figure 2.28, A parse tree for the list for the LISP S-expression `(+ 1 2 (IF (> TIME 10) 3 4))` *depicted as rooted, point-labelled tree with ordered branches [6].*

A key problem with having a variable sized chromosome in GP is what is known as ‘program bloat’ [75] [76] [77] [78] [79] [80]. It is commonly thought that, expanding volumes of ‘junk’ code which does not affect the individual GP tree’s performance (sometimes using the biology analogy “introns”) [81], causes the bloat. Experiments by Langdon and Poli [81] on dynamic fitness cases, find that large penalties do not affect program bloat. Often an upper cap is placed upon the number of nodes (branch and terminal/leaf nodes) or the depth of the tree. [82]

Genetic Programming is a relatively new field, with its first conception in the early 1990’s by Koza, however it does extend the relatively well established fields of evolutionary algorithms, and machine learning. As shown by a recent community survey (2013) [83], there are a wide variety of GP application domains (shown in Table 2.3).

2.2.2 Applications

Table 2.3, Problem domains used in EuroGP and GECCO GP track papers 2009-2012, from the 'Better GO benchmarks: community survey results and proposals' [83] .

Category	Number of papers	Percentage
Symbolic regression	77	36.2
Quartic polynomial	15	7.0
Classification	45	21.1
UCI database examples	23	10.8
Predictive modeling	30	14.1
Boolean	37	17.4
Parity	31	14.6
Multiplexer	21	9.9
Path finding and planning	44	20.7
Artificial ant	24	11.3
Control systems	5	2.4
Game playing	5	2.4
Dynamic optimisation	7	3.3
Traditional programming	8	3.8
Constructed benchmarks	12	5.6
Other	10	4.7
Max problem	5	2.4

Technically GP could be applied to many more machine learning applications/problems, as at its core is the idea of being able to evolve entire computer programs. However, as it is a relatively new field, there is still much research to be done to fully understand the field. Therefore, researchers perhaps more often opt for 'better understood' machine learning algorithms. It is clear from Table 2.3, that a large number of applications are concentrated on symbolic regression problems. This is due to a number of factors, firstly the fact that unlike other machine algorithms (e.g. Artificial Neural Networks (ANN)), GP produces a human-readable formula. Secondly by limiting GP to operating without loops or recursion (such as mathematical formula, as shown in Figure 2.25 and Figure 2.28), there is no need to tackle the halting problem. I.e. a GP with forms of memory, and/or capable of looping, could either take a very long to time to come to its conclusion, or never stop.

The third reason for symbolic regression GP application volume seen in Table 2.3 is possible due to the ease with which data can be produced and tested.

Classification and Boolean application figures seen in Table 2.3 are probably explained by similar reasons to those of symbolic regression. Whereas path finding and planning demonstrates how the tree structure lends itself to these kinds of problems.

Table 2.4, 'Better GP benchmarks: community survey results and proposals' [83] A proposed blacklist of benchmark problems.

Domain	Problem
Symbolic regression	Quartic polynomial
	Low-order polynomials
Boolean	Parity (fixed input size)
	Multiplexer (fixed input size)
	Other simple one-output Boolean functions
Classification	Pima, Iris, Wine datasets
Path-finding and planning	Artificial ant
	Lawnmower

Table 2.4, 'Better GP benchmarks: community survey results and proposals' A proposed blacklist of benchmark problems. shows a number of problems the GP community has suggested for '*blacklisting*', due to fact that many of these problems are simply "too easy" [83] for GP, or rather GP is known to lend itself to solving these problems easily. A number of more complex problems are suggested, including multiple output multipliers to replace parity and multiplexers and more complex classifiers and planning and control applications including Mario gameplay and physical TSP (Traveling Salesman Problem). Also the community begins to establish some benchmarks for the symbolic regression cases, which are shown in Table 2.5.

Table 2.5, 'Better GP benchmarks: community survey results and proposals' [83] A list of proposed benchmark problems for symbolic regression for GP. In the training and testing sets, U[a,b,c] is c uniform random samples drawn from a to b, inclusive, E[a,b,c] is a grid of points spaced with an interval of c, from a to b inclusive.

Name	Variables	Equation	Training Set Testing Set
Keijzer-6 [25] [46]	1	$\sum_{i}^{x} \frac{1}{i}$	E[1, 50, 1] E[1, 120, 1]
Korns-12 [27]	5	$2 - 2.1\cos(9.8x)\sin(1.3w)$	U[−50, 50, 10,000] U[−50, 50, 10,000]
Vladislavleva-4[50]	5	$\frac{10}{5+\sum_{i=1}^{5}(x_i-3)^2}$	U[0.05, 6.05, 1,024] U[−0.25, 6.35, 50,00]
Nguyen-7 [33]	1	$\ln(x+1)+\ln(x^2+1)$	U[0, 2, 20] None
Pagie-1 [36]	2	$\frac{1}{1+x^{-4}}+\frac{1}{1+y^{-4}}$	E[−5, 5, 0.4] None
Dow Chemical (see Sect. 6.1)	57	Chemical process data[6]	747 points 319 points
GP Challenge [56] (see Sect. 6.1)	8	Protein energy data	1,250–2,000 per protein None

The problems suggested in Table 2.5, show a minimum amount of complexity for which GP should be applied, as well as demonstrating how GP tends to be trained and tested on similar ranges. GP searches multiple levels of complexity by combining (stringing together or nesting) relatively simple mathematical operations and terminal values (static values and variables), and therefore should be applied to reasonable level of complexity for the target formula.

In a recent paper titled '*open issues in genetic programming*' [84], they again confirm that "GP has not reached the popularity of other machine learning methods. At the current time, GP does not seem to be universally recognized as a mainstream and trusted problem solving strategy, despite the fact that in the last decade GP has been shown to outperform some of these methods in main important applications" [84]. However, there is room for optimism, as GP has been accepted in platforms like MatLab, and Mathematica. Also GP has been shown to be capable of solving real-world problems, and demonstrating routine human competitiveness.

The research field of GP has a number of open issues identified, including [84]:

- Identifying appropriate representation for GP

There are a number of different representations besides the standard predominant tree-based form popularised by Koza, including binary string machine code, finite state automata, and generative grammatical encodings. Other representations include graphs, strongly-typed, linear, linear-trees, and linear-graph [84].

- Fitness landscapes and problem difficulty in GP

The choice of genetic operators and fitness functions will have a large sway on how the GP is capable of learning the given system. More research is required in order to establish the links between types of fitness functions, and different operator sets, and performance of GP.

- Static versus dynamic problems

Given that natural evolution is only really concerned with the survival and reproduction of species, the challenges that are presented for each individual's survival tends to be fairly different. It is currently thought that more dynamic test environments for the GP training would result in greater diversity and perhaps better generalisation (i.e. increased adaptiveness of individuals) [84].

- The influence of biology on GP

The fields of evolutionary computation and GP have two main goals, firstly to reverse engineer or rather come to understand the mechanics of natural evolution better. Secondly to harness and understand the mechanics/algorithms of natural evolution such that they can be applied to other problem areas [84].

- Open-ended evolution in GP

Stemming from work by Von Neumann and Turing, and others, is the idea of evolution with no clear goal. Natural evolution appears to have no clear fitness function, yet somehow the complexity of life forms has increased continually.

Recent work by Moore and co-workers [85], show the essential ingredients of open-ended evolution are (i) a dynamically changing fitness landscape, (ii) availability of co-evolutionary processes, (iii) search with continuously injected randomness [84].

- Generalisation in GP

How to ensure that the evolved GP have good properties of generalisation (i.e. does not over fit the training data)? "*A large amount of literature and of well-established results exists concerning the issue of generalisation for many non-evolutionary Machine Learning strategies*" [84]. A common agreement of many researchers is the so called "minimum description length principle", which states that the best model is the one that minimises the amount of information needed to encode it. However it has been noted in the aptly named paper "The role of Occam's razor in knowledge discovery" [86] , that the above argument of minimum description length, should be taken with care as too much emphasis on minimising complexity can prevent the discovery of more complex yet more accurate solutions. It has also been suggested that bloat is related to over fitting, however recent work by Vanneschi et. al. (2009) [87] clearly shows that GP systems can be defined that bloat and do not over fit data, and vice versa. Thus, bloat and over fitting seem to be two different phenomena.

- GP Benchmarks

As GP is capable of solving such a wide range of problems, and in a wide variety of ways, some of the unique facets of GP mean that the community continues in very recent years to attempt to establish a better set of benchmark problems specifically for the GP field, as shown by [83].

- GP and modularity

Many modern high level computer languages have the concept of functions, and even nature begins to block groups of DNA together in to chromosomes. How can this kind of modularity be incorporated or even derived by the evolutionary system? The first attempts came through Koza's [6] Automatically Defined Function (ADFs), although little study was done on the theoretical background. The first steps towards a theoretically motivated study of ADFs is probably

represented by [88], where an algorithm for the automatic discovery of building blocks in GP called ‘adaptive representation through learning’ is proposed. Linear GP has other ways to evolve modularity, by reusing contents of registers; memory in LGP can be considered a substitute for ADFs in tree-based GP [84].

2.2.3 Genetic Programming and Cellular Automata

There have been a number of cases of the application of GP for the learning of CA state transition rules, although most of these studies do not use continuous CA, but most commonly the binary state CA or a similar low number of states. For example Andre & Koza discover a better than any known rule by means of genetic programming for the majority classification problem [89]. In this case a 1 dimensional binary state CA is used, with a neighbourhood radius of 3, which allows for complex interactions of the rule sets. The problem is to create a rule set which will after a certain number of generations finds the state the majority of cells were in at the initial condition. It does this by means of altering the states of the cells, such that at the conclusion all cells have changed to the majority case. The resulting state transition rule is encoded as a 7x7 bit string of binary values, totalling a 149 bit string. The best evolved system results in an accuracy of 82.326%, which exceeds that of any human designed rules. The number of test cases used for each fitness case is in the order of 10^6 to 10^7.

Other examples of Genetic Programming in CA include [90], again Koza, although this time working on using 1D CA to produce pseudo-random sequences and using entropy as the fitness function. Koza in this paper states “*The problem of designing a state-transition rule that, when it operates in each cell of the cellular space, produces a desired overall emergent behaviour is called the ‘inverse’ problem for Cellular Automata*” [90].

The only sources of learning of continuous CA state transition rules, comes from the use of CA for image processing, often for edge detection [91]. However, such methods use Genetic Algorithms or EA, and use very simple state transition rules, where the optimisation is only calibrating the human created rule.

2.2.4 Alternatives to GP for learning CA state transition rules

The majority of works that have used machine learning algorithms to develop CA state transition rules, use a genetic algorithm and tend to operate on binary state CA [92] [93] [94]. The majority of these tackle problems such as the majority task, as Andre & Koza [89] did with GP. GA is a good match for this sort of problem because of the binary state available to each cell, where there is a limited number of combinations at the neighbourhood level, and a binary gene may represent the resulting state of the main cell for each neighbourhood configuration. I.e. using a numbering system like Wolfram's code [5], it is possible to represent easily every combination in a binary string.

There are a number of CA which are used to model land uses and people movements, and a number of these methods use machine learning algorithms to calibrate a number of the model parameters [95]. The underlying mathematical framework for the state transition rules are developed by humans, but by allowing an element of calibration, such methods can easily learn an effective state transition rule.

A number of other methods propose adaptive or self-programming state transition rules [96] [58] [53] , where the state transition rule learning is done during the evolution of the CA. The majority of these kinds of systems are developed for either stochastic or heterogeneous CA.

Another interesting CA technique is the use of what is termed a Cellular Neural Network [36] [97] [98], whereby the state transition rule of the CA is a form of neural network (Shown in Figure 2.29).

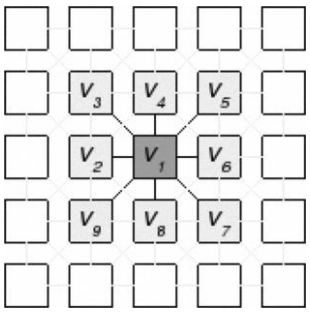


Figure 2.29, Neighbourhood for a Cellular Neural Network, where weighted and possibly even function based elements connect the main cell to each of the Moore neighbourhoods cells [36].

Cellular Neural Networks are commonly used for image processing, due to their ability to represent a wide range of different graphics kernels, and the analogy with graphical filtering. They tend to use a very direct relation between the neighbouring cells and the new state, and may not be uniform, and they are not designed to represent complex functions between each cell.

2.2.5 Conclusion

Genetic Programming is capable of learning complex formulae, and has been shown to be capable of deriving state transition rules for simple CA systems. While there is a limited amount of work pertaining to the learning of state transition rules, the majority is aimed at either binary state CA or calibration of simple land use CA. A number of genetic algorithms have been applied to learning CA state transition rules, or their calibration and this has proven successful in a number of areas. Therefore, it appears reasonable to conclude that applying genetic programming to the learning of more complex continuous CA state transition rule should be able to derive rules with some success. Furthermore, GP is capable of producing human readable formula as the result of his optimisation, which may then be of use to human designers.

2.3 GPGPU computing

2.3.1 Introduction

The Graphics Processing Unit (GPU) has in recent years become not just a powerful graphics engine, but also a highly parallel programmable processor, featuring peak arithmetic and memory bandwidth that substantially outpaces its CPU counterparts [99]. In fact in recent years, due to the heavy parallelism of the GPU hardware, their processing power (Giga Floating Point Operations Per Second, GFLOP/s), has increased at a greater rate than their CPU counter parts, as shown in Figure 2.30.

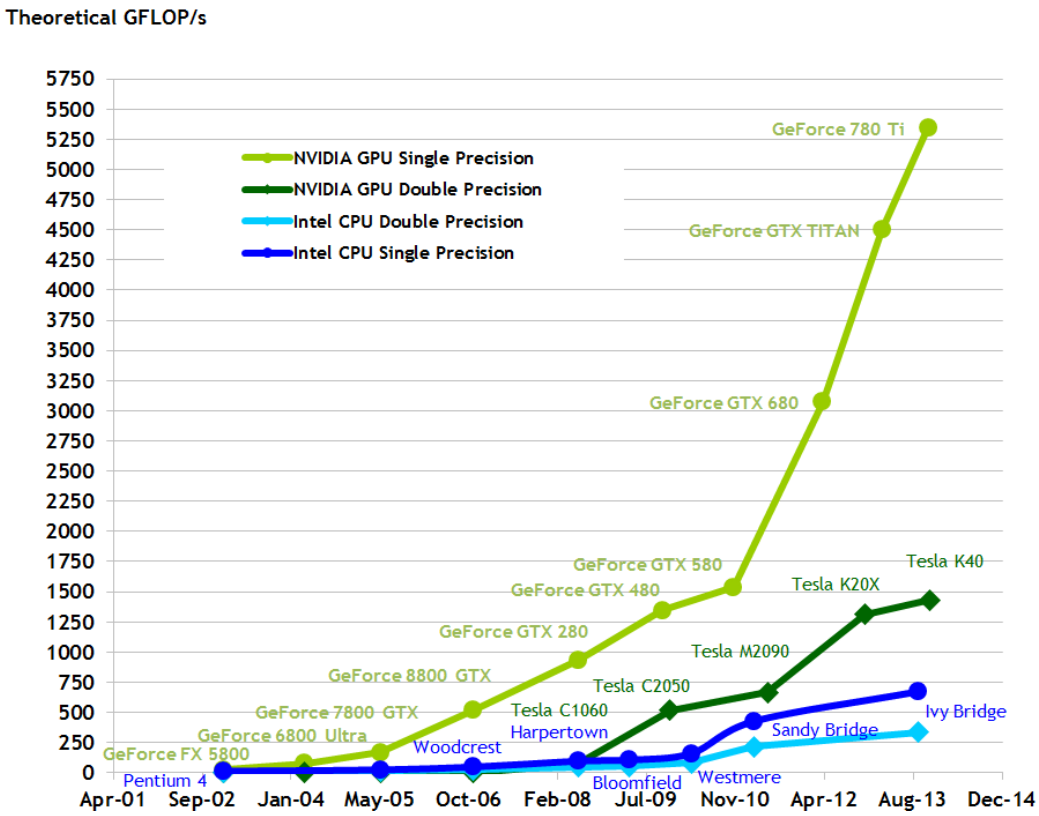


Figure 2.30, Theoretical maximum processing power (measured in Giga Floating Point Operations Per Second, GFLOP/s), between modern CPU and GPU, in both single and double precision [100].

Although CPU and GPU have become progressively more parallel in recent years, the number of independent cores within a CPU is dwarfed by those available in a GPU. This difference is exemplified by the differing terminology used for CPUs, multi-core computing, and GPUs many-core computing. This different terminology refers not just to the sheer number of cores within the GPU compared to the CPU, but also the way in which the CPU cores are more independent than GPU cores.

In recent years, the GPU has been harnessed for more general purpose processing than solely graphics, and can be known as a GPGPU (General Purpose Graphics Processing Unit). By harnessing appropriately parallel algorithms, which fit the GPU's radically different hardware, large increases in processing speed over than of a modern CPU can be gained [46] [48] [101] [23] [26].

A key attribute of both Cellular Automata and Genetic Programming, is the parallelism of the algorithms. Both CA and GP have different properties at an algorithm level and therefore different amounts of parallelism. The key

parameters of each algorithm, for example with CA the size of the grid (number of cells), number of iterations, complexity of the state transition rules, and neighbourhood size, will all have an effect of the level of parallelism of the algorithm. For GP the size of the tree which has to be parsed and the complexity of the operations and perhaps even some of the evolutionary parameter will have a play on the parallelism. The following observations were made in our paper '*An investigation of the efficient implementation of cellular automata on multi-core CPU and GPU hardware*' [101], referring to existing literature on the properties of CA acceleration upon GPU hardware.

2.3.2 Cellular Automata GPU computing

There are few attempts in the literature to develop parallel CA algorithms and to investigate how exactly they will interact with many-core technologies. There are a number of examples of implementations which are discussed below; however, few of these investigate the spread of possible speed-ups, or how the variation of the CA's base parameters (e.g. lattice size, number of generations, number of states/data types, neighbourhood sizes, or rule complexity) affects these speed ups.

Recent approaches to the use of CPU and GPU computing to speed up CA execution include Lopez-Torres et. al. (2012) who used CA to simulate laser dynamics, and noted in summary of recent CA-GPGPU works that "*Depending on application, they are offering a 10 to 100 times speed up at price points extremely affordable*" [38]. Rybacki et. al. (2009) examine and benchmark CA algorithms and investigate different levels of multi-threading with either a "brute force" or sparse ("discrete" which only applies the rule set to those cells that might change) method of implementation, on both the CPU and GPGPU of several machines. They use five different rules sets: the game of life, parity, majority, wireworld, and a benchmark case. They find that "*there is no perfect algorithm for everything*" [42], which is largely due to the discrete algorithms being outperformed on the GPGPU, but they note that this is due to the small size of the grid and/or the small number of living cells after the first few generations. This work highlights the issue of sufficient parallelism, if a CA with a low number of cells (and therefore low number of parallel elements) is applied to hardware with a large number of cores there is a high likelihood that computational resources

will be wasted due to the lack of sufficient algorithmic parallelism. The algorithmic representation must match the many-core nature of the GPGPU and sparse representations either don't work well or are difficult to code on the GPGPU. Also, if a rule set is somehow known to produce little to no *activity* (the number of cells alive and/or changing value over the whole simulation) within a given grid and initial configuration then it may still be more efficient to use a sparse representation on the CPU, as there is relatively little computational work to be done.

There are circumstances where a sparse implementation has been implemented on a GPGPU, for example Ferrando et. al. (2011) [40] have employed an Oct-tree representation which subdivides a 3 dimensional cube of space into 8 smaller cubes at each branch of the tree. Although this does mean that the tree structure must be stored and manipulated using the CPU, a lot of processing can still be carried out on the GPGPU. This is done by means of the CPU organising the tree structure, which then issues commands to the GPGPU to order the respective array of 'memory clusters'. These memory clusters are organised linearly upon the GPGPU, and each contains all the information needed to process a single cell (i.e. the cell and its neighbour's states), these can then be processed in bulk by the GPGPU. The optimised use of further GPGPU data structures are used to minimise the amount of traffic between the CPU and GPGPU, which is known to be a bottleneck. However, Ferrando et. al. [40] are more keenly interested in carrying out the high resolution of simulation in feasible amounts of time, and so do not directly claim that this approach provides speed ups because their system works as a co-operation of the CPU and GPGPU.

Of particular interest is work by Zaloudek et. al. (2010) [43], in which they examine the evolution of 1D CA systems, using Nvidia's Compute Uniform Device Architecture (CUDA). CUDA describes both Nvidia's architecture and high level language for its manipulation. Zaloudek et. al. examine the possibility of parallelisation at the level of cells, but also at an evolutionary CA system level which requires a population of solutions be evaluated, often with each possible solution (state transition rule set) needing to be run under a number of initial conditions in order to reach an average fitness. They have examined the possibility of parallelising their algorithm in terms of 'training vectors' and 'individual solutions', as well as by cells. The results are encouraging in favour of

using parallelisation at the ‘training vectors’ and ‘individual solutions’ levels. However, this is due to the fact that they confine themselves to the very closest, fastest and conversely smallest forms of memory on the GPGPU (known as ‘local memory’, analogous with cache memory on the CPU). This severely limits the size of CA grids which they can simulate mainly due to the way that synchronisation works differently on a GPGPU with current limits at 1024 threads/cells. They show that this local memory can allow for a huge processing speed increases where they show a CA simulation (without any evolution) for 50,000 generations/fitness evaluation has a speed up of 489.75 times on one machine and 621.68 on another [43]. These speed-ups are exceptional and are at the high-end of the findings here. One possible source of disparity between their results and those shown in Chapter 3:, is the use of local memory, although experiments by the authors of this paper, tailored towards these hardware specific parameters [48] explains in greater detail how this local memory may benefit some machines, and the limitations of using the GPGPUs specific memory types. This work showed that local memory is indeed faster in all machines than the GPGPUs main (global) memory, but due to limitation on the number of threads/cells that speed-up factors of 2.5x-5x are obtained with these local memory implementations. However, by using the global memory to allow for synchronisation of much larger grids, greater speed-ups of up to nearly 50x are obtained. The final significant finding of this study is they show that the workgroup (OpenCL) or block size (CUDA), is vitally important to the speed up of GPGPU processing, and should be selected from the small spectrum of possible sizes though empirical testing. In the work below, this limiting factor is investigated along with the effects of these models on the GPGPU. Lastly Brodtkorb et. al. [47] perform a good review of current trends in GPGPU computing, and say “*reporting a speedup of hundreds of times or more holds no scientific value without further explanation supported by detailed benchmarks and profiling results*.” [47].

Collectively, the literature demonstrates that there is considerable interest in the use of multi-core and GPU computing to parallelise cellular automata for specific applications. Papers such as [43] have investigated a more general-purpose approach to the parallelisation of the technique but this experimentation is conducted with a hybrid 1D CA/EA algorithm and with variety of vectors but

single lattice size. However, the literature is lacking a discussion of how the base CA parameters such as lattice size, neighbourhood size, number of states and iterations (generations) affects processing speed-ups on the GPGPU.

2.3.3 Genetic Programming GPU literature

Work by Harding and Wolfgang [49] investigates the properties of parallelising Genetic Programming, stating "*it is well known that fitness evaluation is the most time consuming part of the genetic programming (GP) system. This limits the types of problems that may be addressed by GP, as large number of fitness cases make GP runs impractical.*" [49]. At this point in time (2007), they are limited to the use of OpenGL, which is primarily intended for the generation of graphics but was used for more general purpose process until the introduction of OpenCL and CUDA. Figure 2.31, shows how the graphics pipeline is used for more general purpose programming.

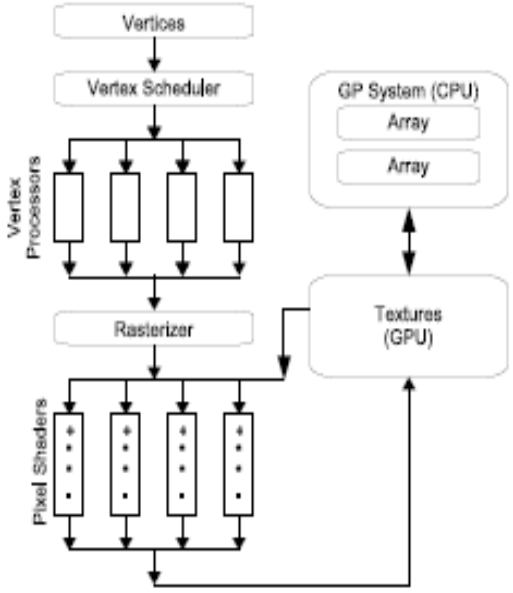


Figure 2.31, Illustration of how arrays, representing the test cases, are converted into textures. These textures are then manipulated (in parallel) by small programs inside each of the pixel shaders. The result is another texture, which can be converted back to a normal array for CPU based processing. [49]

Harding and Wolfgang [49], state that "Typically parsing a GP expression involves traversing the expression tree in a bottom-up, breadth first manner. At each node visited the interpreter performs the specified function on the inputs to the node, and outputs the results as the node output. The tree is re-evaluated for every input set. Hence, for 100 test cases, the tree would be executed 100 times"

[49]. Using the GPU, they are able to parallelise the process, so the GP tree is only parsed once, with each test case applied in parallel. One advantage of such a system, it is claimed by the authors, is that this reduces the amount of times that the switch case needs to be accessed; although this may not be the case, dependant on how hardware accesses the data for each parallel element. The GP population and genetic algorithm are applied on the CPU, with the GPU performing the evaluation of the GP trees.

An important factor in the final performance is that the "release build configuration" is utilised. I.e. this means that compiler optimisation is enabled, which results in much faster processing programs, and makes for a reasonable comparison of processing speeds. In their experiments the GP trees are randomly generated, with a given number of nodes. Experiments are carried on floating point, binary and real world test cases. The function set used on the floating point tests, are +, -, *, and /. The resulting speed up factors, are recorded in Table 2.6 (where a value greater than 1 indicates the GPU is faster, and less than 1 the CPU).

Table 2.6, Results showing the number times faster evaluating floating point based expressions is on the GPU, compared to the CPU implementation. An increase of less than 1 shows that the CPU is more efficient [49].

	Test Cases					
Expression Length	64	256	1024	4096	16384	65536
10	0.04	0.16	0.6	2.39	8.94	28.34
100	0.4	1.38	5.55	23.03	84.23	271.69
500	1.82	7.04	27.84	101.13	407.34	1349.52
1000	3.47	13.78	52.55	204.35	803.28	2694.97
5000	10.02	26.35	87.46	349.73	1736.3	4642.4
10000	13.01	36.5	157.03	442.23	1678.45	7351.06

The results here are of that of Cartesian GP, although the authors claim similar advantages for linear and tree based GP. They also state that "*many typical GP problems do not have large sets of fitness cases for two reasons: First, evaluation has always been considered computationally expensive. Second, we currently find it very difficult to evolve solutions to harder problems. With the ability to tackle large problems in reasonable time we have to also find innovative approaches that let us solve these problems. Traditional GP has difficulty with*

scaling. For example, the largest evolved multiplier has 1024 fitness cases on a GPU" [49]. The authors confirm that small sets of fitness cases, the overheads of transferring data and the shader programs to the GPU outweighs the advantages of the increased processing speed, resulting in overall the CPU being more efficient at these small numbers of test cases. The results in Table 0.6, demonstrate that increasing the number of test cases, and/or increasing the number of expressions in the GP tree to be evaluated, will both result in large performance increases upon the GPU. Although the precise scale of these speed-ups seems rather high if we consider the brute processing power of modern GPU's (in terms of FLOPS, floating point operations per second), is on average about 5-10 time that of the modern CPU.

Experimentation is also carried out by Harding and Wolfgang [49] on a toy/benchmark test case, where they perform regression upon Equation 2.8.

Equation 2.8 $$x^6 - 2x^4 + x^2$$

A number of test cases are drawn randomly from a uniform distribution between -1 and +1. They allow for the length of GP expression (which is again implemented using Cartesian GP), to vary between 1 and the maximum size indicated in Table 2.7. The GP was run for 200 generations to allow for convergence, and again using the same simple function set (notably division by zero results in "infinity").

Table 2.7, Results for regression experiments, showing the number of times faster evaluation evolved GP expressions is on the GPU, compared to CPU implementations. The maximum expression length is the number nodes in the CGP graph [49].

	Test Cases			
Max Expression Length	10	100	1000	2000
10	0.02	0.08	0.7	1.22
100	0.07	0.33	2.79	5.16
1000	0.42	1.71	15.29	87.02
10000	0.4	1.79	16.25	95.37

It is unfortunate that the authors do not also report the actual processing times of these experiments, as even with these large speed-up factors, there may still be a limiting point. However, with at least 100 test cases, and at least a

maximum number of expressions of 100, speed-ups can be gained from use of the GPU. The greatest speed ups are once again, with either very large number of test cases and/or large numbers of GP node expressions.

Where we plan to run Genetic Programming within a Cellular Automaton, we can expect a similar pattern, whereby sufficiently sized GP trees and test cases will be required in order to gain speed-up upon the GPU.

2.3.4 Conclusion

There is obviously great interest in the parallelisation of both CA and GP algorithms, and a great variety of speed-ups for both algorithms have been demonstrated on various different test cases. There is an understanding that the number of nodes within the GP tree relates to the speed-ups gained, where larger number of nodes means a greater amount of time spent on each and therefore more parallelism. Whereas CA demonstrate a more a number of very different parameters relating to their spatial and temporal layouts, and there is a much larger variety of speed-ups reported. This may be attributed to the different types of memory used in the GPU hardware, and the different access patterns. There is little understanding in the literature demonstrated of how the key CA parameter of grid size, number of generations, number of states/base data type, size and shape of neighbourhood, or complexity of state transition rule affect the relative speed-ups obtained.

Chapter 3: GPU computing

The introduction has illustrated that the potential for cellular automata as a modelling tool in areas such as urban flooding are only just starting to be realised. Although CA are computationally efficient in comparison with other models, they still represent a significant computational cost if the model run has to be repeated a large number of times. As the ultimate goal is to use genetic programming to determine new rulesets for a CA, the CA will be executed within the objective function calculation of a population based metaheuristic. This will necessitate many thousands of runs of the CA and so greater computational savings are required. This chapter investigates the potential for cellular automata to be parallelised on modern GPU systems. Programming for GPU systems requires a detailed understanding of the underlying hardware and novel lattice representations to take full use of the additional computational power. This chapter investigates these aspects on the well understood Game of Life ruleset and some novel extended multi-state rulesets, as the investigation is intended to bridge the gap between such well understood rulesets potentially much more complex real world rules such as urban flood modelling. The investigations in this chapter are carried out on two different graphics cards, in order to determine any major difference between the generations of GPU. The literature on the effects of the Genetic Programming parameter on the GPU speed-ups shows how the number of nodes within the GP trees affects the processing speeds for example. The effects of the base parameters of CA on GPU speeds-up factors are less well understood. Therefore, the experimentation in this chapter investigates the effects of such CA base parameters as the lattice size, neighbourhood size, number of generations, the number of states and data types used to store the state, and the initial configurations and activity levels within the simulation. Experimentation is also carried out on the GPU specific parameters which relate to the CA execution such as the workgroup size and GPU memory types. Parts of the following are drawn from the papers [101] and [48].

3.1 Introduction

Modern hardware is becoming increasingly parallel in nature with modern commercially available single CPUs equipped with up to 8 cores, and Graphical Processing Units (GPUs) having many hundreds or even thousands of cores (e.g.

the latest Nvidia cards have between 3072 - 5740 cores [102] [103]). This increase in parallelism provides the opportunity for such inherently parallel algorithms as CA to provide large speed increases in processing. However, there is the need to understand the scalability of this effect, especially with regards to the CA base parameters and the specification difference between the CPU and GPU hardware in question.

In recent years the development of the GPU into a processor capable of General Purpose processing has received particular attention, due to fact that GPUs have needed even greater parallelism than their CPU counterparts. The literature shows that although methods for parallelisation on the CPU are fairly well established and understood that some of the unique architectural differences between the General Purpose-GPU (GPGPU) and CPU are not so well known. GPGPU computing is still very much an expert field, which means that there are few comprehensive studies of the performance and scalability of the performance gains possible through GPGPU computing, particularly for cellular automata. Cellular automata (CA) are excellent techniques for the efficient simulation of a wide variety of systems and natural phenomena, in addition to being interesting from a theoretical perspective.

In this chapter, the new open standard OpenCL is used to perform benchmark tests using the well-studied ‘game of life’ cellular automaton [25] along with some novel variants. Experimentation is conducted on a variety of different parameterisations of cellular automata that impact performance, notably the lattice size, the number of states, the neighbourhood size, the complexity of the state transition rule sets, and population levels within the random initial configuration by assigning the chances of a cell being alive in the initial configuration (initial configuration distribution probability). Also experimentation is carried out on GPU specific parameters such as the data types used to store the states, and the different GPU memory types available. With the availability of such unique memory structures as the ‘image/texture’ memory, which is designed to store a vector of 4 values Red, Green, Blue, and Alpha levels, a novel method for the utilisation of such memory structure for the processing of CA is proposed in the Folding method.

It is found that each of these key parameters affects the ability of multi-core CPU and GPU architecture to speed-up CA execution. In addition, these key CA parameters cover the majority of variations in CA that might be implemented to simulate a variety of natural systems. Through the intensive study of the multi-core/many-core speed-up available for a wide variety of parameter settings, it is possible to infer some general properties of CA efficiency operating on multi-core CPU and GPGPU hardware. This is particularly useful as it should also be possible to extend these inferences to the more complex urban flood modelling rules sets, and possible variant rule sets that GP could create. This kind of extensive study would be more difficult with the real world urban flood modelling rules sets due to the much larger space of possible variations.

3.1.1 Multi-core CPU and Many-core GPU computing

With the wide variety of disciplines and applications for CA suggested in the literature review, a growing number of modellers are harnessing the inherent parallelism of the CA algorithm in modern hardware, i.e. multi-core CPUs and many-core GPUs. This is motivated by the idea that the multi-core nature of most modern CPUs which is allowing Moore's law to continue to predict processing speed increase. [33]. Also several sources suggest that co-processors like GPGPUs, with their inherently many-core nature, may be increasing in performance at a quicker rate than their CPU counter parts [30], with Fan et. al. stating that: "*Driven by the game industry, GPU performance has approximately doubled every 6 months since the mid-1990s, which is much faster than the growth rate of CPU performance that doubles every 18 months on average (Moore's law), and this trend is expected to continue.*" [23]. Although this publication dates from some eight years ago and it is now an established fact that significant increases in computational power in many areas will need to be achieved through the use of parallelism rather than the increase in performance of a single core. Therefore, the scientific community needs algorithms which will scale to take advantage of this emerging parallelism, such as CA, and to understand how these algorithms will scale to the emerging hardware available.

3.2 Relevant literature

A vast number of different speed ups have been reported [32] [36] [46] [23] [40] [37] [55] [44] [57] [28] [27] [24] [38] [33] [21], Which vary drastically from small scale speed ups of 5-10x, up to large scales of the thousands. Where it should be noted that the overall computing power of the modern GPU is on average approximately 5-10x in terms of GFLOPs, although the different memory access patterns on different hardware and different levels of algorithmic parallelism may allow for some higher scores. Understanding how the key CA parameter such as grid size, number of generations, number of states, neighbourhood size, initial configuration set up and the complexity of the state transition rules will affect the different levels of algorithm parallelism will be very important in understanding the levels of relative speed-ups obtained.

3.3 Method

It is suspected that the rule set will play a large role in the computational properties of CA on GPU hardware, and in order to investigate the different effects of the base CA parameters a simple well known and investigated rule set of the Game of life is utilised. As more complex real world rule sets like urban flood modelling will use far more states, perhaps even a continuous scale, these investigations include experimentation on novel extensions of the Game of life so as to use multiple states. Furthermore, experimentation is also carried out on the base data type which carries the state of each cell of the CA, be that an integer or a floating point (i.e. continuous data type).

3.3.1 Rule sets

In the majority of tests, the well-studied ‘game of life’ rule set is used, which has 2 states and a Moore neighbourhood. This is often instantiated by means of some form of look up from the possible previous states of a cell and its neighbours, mapped to the corresponding next state of the main cell (current cell under evaluation). However, to allow for more complex systems with variable number of states and neighbourhood sizes, a programmatic function is used here (a series of C if-statements) which forms the basis of a decision tree. The basic definition of the game of life states there are two states known as dead (zero), and alive (one), and that if a cell is currently dead and has 3 live neighbours then

it becomes alive, and if it is currently alive and has 3 or 2 live neighbours then it remains alive otherwise becoming dead. This is interpreted in pseudo-code as follows in sections 3.3.1.1, 3.3.1.2, and 3.3.1.3 .

Since the ‘game of life’, is confined specifically to two states and a Moore neighbourhood, a number of new rule sets have been created based on the decision tree which demonstrates the compactness and simulation variety possible. Also this enables the testing of the effect of variable numbers of states and neighbourhood sizes without excessively large look-up tables. Two of the most interesting rule sets which demonstrate the relationship between activity and speed increases are shown in rule sets MSGOL (Multi-State Game Of Life) and MSGOL4 (Multi-State Game Of Life version 4) in section 3.3.1.2, and 3.3.1.3, respectively. In order to test the effects of different neighbourhood sizes, the adaptive nature of such decision tree based rule sets is demonstrated further, by using the same ‘game of life’ rule set (section 3.3.1.1) and an extended Moore neighbourhood where the size of the radius of the neighbourhood is defined by a user-specified parameter.

3.3.1.1 Pseudo code for the game of life rule set function

Below is the code interpretation of the game of life rule set. Where ‘`mainCell`’ variable contains the current value (state) of the central main cell, and should finish with the next state of the main cell. The ‘`NH_Count`’ variable holds the number of live cells in the neighbourhood excluding the main cell.

```
if(mainCell == ALIVE)
      if(!(NH_Count == 3 OR NH_Count == 2))
            mainCell = DEAD;
else
      if(NH_Count == 3)
            mainCell = ALIVE;
```

3.3.1.2 Pseudo code for the Multi-State Game Of Life (MSGOL) rule set function

This rule set implements a simple multistate conversion of the Game of Life where the state represents the energy of the organism rather than a simple alive/dead delineation. The rules are constructed so that the rule that would usually lead to a cell becoming alive, increases energy and the rule leading to

death decreases it. Death only occurs when the cell reaches the lowest possible state.

```
if(NH_Count == 3)
      if(mainCell != states-1)
            ++mainCell;
else
      if(NH_Count != 2)
            if(mainCell != DEAD)
                  --mainCell;
```

3.3.1.3 Pseudo code for the Multi-State Game Of Life (MSGOL4) rule set function

MSGOL4 is a modified version of MSGOL and has a more stringent requirement for life in addition to a distinction between having too many or too few neighbours which results in immediate death or loss of a single energy level respectively. Also once a cell is at its maximum energy level another increase will cause immediate death, except when there are only two states, which maintains the rule sets ability to mimic the 'Game of life'. This rule set has been developed to gain a better understanding of the effect of activity levels on potential GPU speed-up.

```
if(NH_Count == 3)
      if(mainCell != states-1)
            ++mainCell;
      else
            if(mainCell != 1)
                  mainCell = DEAD;
else
      if(NH_Count != 2)
            if(NH_Count < 2)
                  if(mainCell != DEAD)
                        --mainCell;
            if(NH_Count > 3)
                  mainCell = DEAD;
```

3.3.2 Novel CA-GPU Representation

In section 3.5.2, a novel use of the GPU's image/texture memory is created, termed as Lattice folding. As the GPU texture memory is specifically designed to carry a vector of four values for the Red, Green, Blue, and Alpha values of a pixel. This can be utilised to store the states of four cells, and therefore increase the amount of processing within each thread, and utilise the wider memory lanes

specifically designed to carry the per pixel data (i.e. a vector of four values). This method also makes use of specifically designed hardware operations within the GPU called swizzling, which can efficiently reorder the vectors of values within hardware.

The method is called lattice folding, as it mimics the process of folding a piece of paper into four quarters, as shown in Figure 3.1, and Figure 3.2. The method of utilising the different colours of the texture image is not entirely novel (Figure 3.2), however the orientation of folding the grid for the application of the neighbourhood based CA system is.

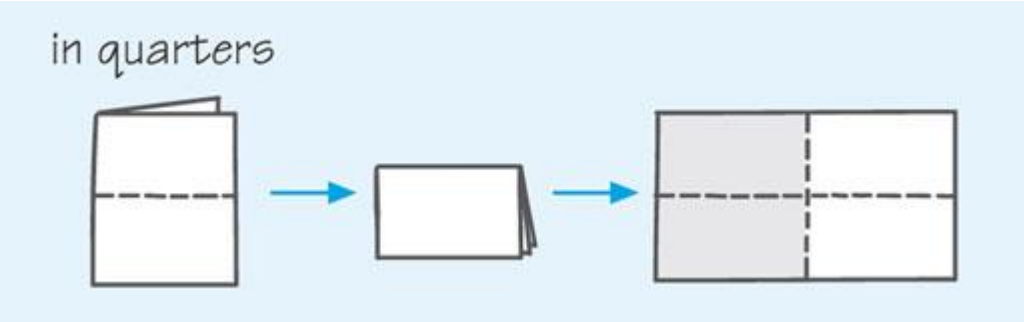


Figure 3.1, How the four quadrants of the single grid are folded into a single grid with four layers Red, Green, Blue and Alpha [104].

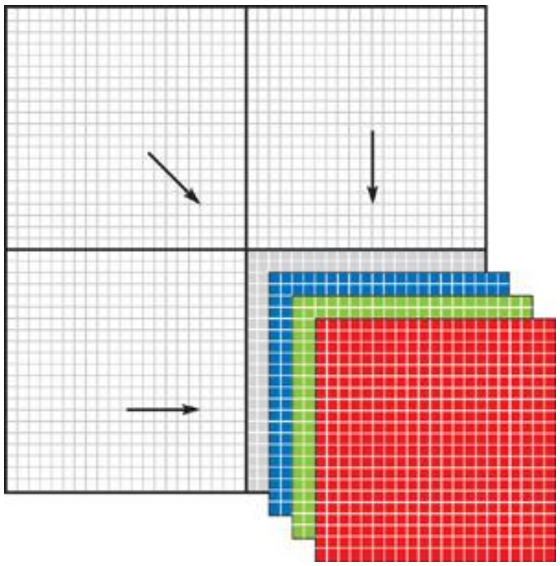


Figure 3.2, How the four quadrants of the single grid are folded into a single grid with four layers Red, Green, Blue and Alpha [105].

By folding the grid, the orientation of the border conditions can be controlled. Cells which are not located directly next to a fold in the grid, can collect the neighbouring values from the folded grid and thusly collect the states of four

neighbourhoods in a single thread. The resulting 4-layered grid has two borders which represent folds in the original grid, and two borders which represent the original borders. On the folded borders, the neighbouring vectors are reflected inwards, and then depending on which way the grid was folded and which border determines the correct swizzling operations. The swizzling operation re-orders the vector of values, which re-align each colour value which the correct colour layer for neighbouring collection at the folded borders.

3.4 Experimental Set up

The C/C++ language and MSVC 2010 SDK compiler are used, and an application profiler was used to ensure that the program did not make excessive memory allocations, which were found to cause large slowdowns in processing in the CPU implementation that could therefore skew comparisons. The '/O2' level of compiler optimisation was also used. The state value of each cell was stored in a single byte (C style 'char' or 'unsigned char'), and used a single array to store the lattice, apart from in section 3.5.5, where experimentation is carried out using char, int, floating and double data types. The experiments below are limited to square grids, and only use a static border condition (of dead cells). Although other border conditions exist, such as wrap-around or reflect inwards, these would require slightly more work at each generation. It was determined that the best way to deal with border conditions is to pad the grid with a border apron of cell values as large as the neighbourhood radius (one in the case of the classic Moore neighbourhood).

A second grid (also padded with this border apron) is created and these two memory spaces are used alternately as the current grid and the new grid, for each generation. Importantly the implementation increments a counter for each live cell in each cell's neighbourhood, as opposed to adding the value of every neighbouring cell to a counter. This becomes very important in demonstrating how variable arithmetic (computational work) can cause variable speed-ups (I.e. conditional branching around the arithmetic work dependant on the automaton's current state and neighbouring states causes far greater variation in processing speed with the CPU compared that of the GPGPU). Another method to accomplish this would be to use a look up table, however this look-up table would be very large, and every cell would need access to it. This design would be sub-

optimal for the CPU, but may be favourable for the GPU's relative speed-up. However, since more complex rules are likely to be built from more programmatic forms, the method of having a count of neighbouring values and a programmatic function is utilised to investigate these properties.

A simple testing framework is developed where the initial grid and the parameters (e.g. grid size, number of generations) are passed into a function which processes the whole CA simulation, and the system clock is used to record how long it takes for the resulting final grid to be returned. Since it is expected to be difficult to time the processing on the GPU at intervals within the simulation fairly, the simulations are repeated for each generational experiment. Each experiment is repeated 15 times in order to gain an average timing result.

In order to achieve parallelisation on the CPU the common shared memory model called OpenMP [106] is used, which generates worker threads at each generation for each cell in the lattice. The compiler then generates code which composes a single master thread, and at each generation launches worker threads for each cell, these are then distributed and time-sliced by the operating system between the CPU cores.

Finally, the new open-standard language/API called OpenCL is used, designed specifically for parallel hardware such as the GPGPUs, and multi-core CPUs. For detailed information the reader is referred to the OpenCL specification [107]. The experiments include a small amount of compilation time for the kernel in every test (although this is from an intermediate form), and the transfer time to and from the GPGPU. Special hardware in the GPGPU time-slices these threads, and load balances between hardware sub-groups of each workgroup which need to access memory and those which need processing, and in this way can hide a lot of processing time behind memory latency. Work by Zaloudek et. al. [43] shows that the workgroup size affects the processing time by affecting how the hardware time-slices threads (referred to a SIMT – single instruction multiple threads). Therefore, initially 2-3 different workgroup sizes are used, to determine the fastest, and the workgroup size is set to this for the remainder of the tests; however, it has been shown that each different machine may require auto-tuning in order to determine the optimum workgroup size.

A key aspect of a study such as this is the hardware used to determine the level of speed-up obtained by the algorithms. The comparison between a single core benchmark and the multi-core implementation will depend to a great degree on the hardware involved. Therefore, during testing two very different machine set ups have been used, firstly Machine A is a Dell XPS M1530 laptop, and is approximately 4-5 years old at the time of testing. Machine B is a recently constructed PC tower unit which contains a modern Core-i7 processor and latest-generation 'Fermi' Nvidia graphics card. The full specification of each machine is shown below.

Table 3.1, Full specifications of machines used for testing [108] [109].

Machine	Machine A	Machine B
Type	Dell XPS M1530 laptop	PC workstation
Age	4-5 years	Recent
CPU	Intel Core2 Duo T8100 @ 2.1GHz	Intel Core-I7-2600 @3.4Ghz
CPU cores	2	4 (8 with Hyper-Threading)
GPGPU	Nvidia GeForce 8600M	Nvidia GTX 560 Ti
GPU Processing elements (CUDA cores)	32	384
GPU Compute cores	4	8
GPU speed (Core, Shader, Memory MHz)	475, 950, 700	1645, 822, 1000/4000
GPGPU bus type	PCI-E x 16	PCI-E 2.0 x 16

Visualisation of the rule sets was achieved through a basic OpenGL interface; also the outputs from the GPGPU algorithms were found to match the CPU implementations exactly. OpenCL possesses interoperability with OpenGL which opens up the possibility to be able to accelerate visualisation as well as processing.

A key difference between the CPU and GPGPU is that the GPGPU is firstly a co-processor which has its own independent RAM memory, meaning that data must be transferred along the bottleneck of the PCI connection. Secondly the GPGPU has distinct architectural differences to the CPU, for example whereas CPU cores may run independently from each other (i.e. operate on different sections of code at the same time, by virtue of each possessing its own program counter) the modern GPGPU has a hierarchy of processing cores. OpenCL calls each core capable of operating independently a ‘compute unit’ (Nvidia/CUDA call this a ‘Streaming-Multi Processor’); each compute unit may possess one to many ‘processing elements’ (Nvidia/CUDA call this a CUDA core), where each processing element may run a thread in an SIMD (Single Instruction Multiple Data) fashion within each compute core. A single workgroup is only ever processed on a single compute core, which allows the GPGPU hardware further parallelism by allowing it to distribute workgroups to compute cores as it sees fit. It attempts to best use the hardware (number of compute cores) available, much in the same way that the operating system and CPU distribute threads amongst its cores.

However, the modern GPGPU uses yet another level of parallelism within each workgroup and compute core, known as SIMT (Single Instruction Multiple Thread). Where a workgroup possesses more threads than processing elements with the compute core, the hardware may swap between many groups of threads with each group at different stages within the code. This allows the GPGPU to put its processing elements to best use, i.e. if one group of threads is waiting on a memory request, then another group which isn’t may be used for processing. This allows the GPGPU hardware to maintain far more simultaneous thread processing compared to the CPU. The majority of this is abstracted away from the programmer (Shown in Figure 3.3), apart from the workgroup size. OpenCL and the underlying GPGPU hardware stipulate both an upper limit on the number of threads within a workgroup (size), and that the lattice of cell/threads must be a multiple of the workgroup size in each dimension.

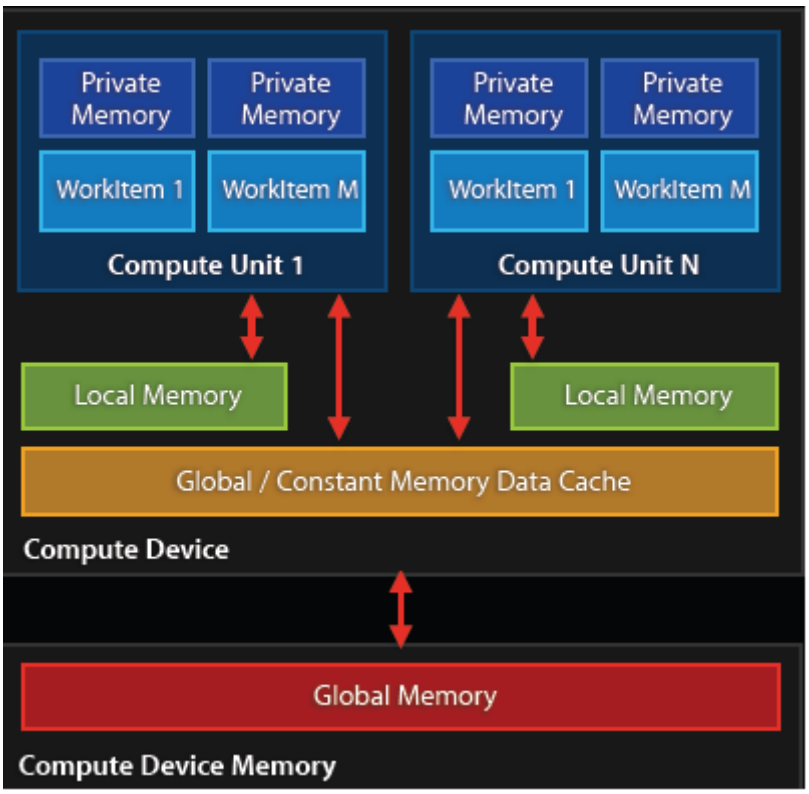


Figure 3.3, The abstract hierarchy presented by OpenCL [107].

The abstract hierarchy for the OpenCL standard is shown in Figure 3.3, where the workgroup specifies which thread/work items are to be performed on the same compute unit. Firstly, there are more threads per work group than there are processing elements within each compute unit, therefore the hardware can be responsible for the firstly layer of SIMT whereby groups of waiting threads can be swapped for groups of threads needing processing within each compute unit. The second layer of SIMT again allows the hardware to decide which workgroup to process on which compute unit, and in which order, as again there are likely to be more workgroups than compute units. Since each workgroup is isolated to a single compute unit, the hardware's distribution of these to the number of hardware compute unit can cause what is called 'load balancing'. This occurs when the number of workgroups and the amount of time they take to process does not suit the number of hardware compute units. A simplified example might be, if the hardware in question has 8 compute cores, and is asked to 9 workgroups, and then no matter how it distributes the workgroups, it will have entire compute cores standing waiting.

3.5 Experimentation

In sections 3.5.1, the lattice size (i.e. the number of cells) and the size of the workgroups are varied to understand this relationship. In section 3.5.2 these experimentation are extended to include different memory types available upon the GPU and the use of the novel lattice folding methods are applied. In sections 3.5.3, the effects of the initial configurations and activity throughout the simulations are investigated. In section 3.5.4 the novel multi-state extensions of the Game of Life are utilised to investigate the variation that a larger numbers of states might have. This is extended in section 3.5.5, by using different base data types such as char, int, float and double to carry the state of each cell. In section 3.5.6 the effects of changing the neighbourhood size is investigated, and finally in section 3.5.7 the effects of various length of CA simulation are investigated.

3.5.1 Lattice size and workgroup tests

The wide variety of application domains for cellular automata means that a commensurate range of lattice sizes are possible. The lattice size is therefore the first variable to be investigated here.

3.5.1.1 Method

In order to allow the processing of any size of lattice, any size which is not a multiple of the workgroup size is padded up to the nearest with threads/cells which do nothing. For these reasons two sets of lattices size tests are conducted with the first testing a wider spectrum of lattice sizes, and second testing a smaller spectrum but at much finer granularity along with testing 3 different workgroup sizes (8x8, 16x16, and 32x32).

3.5.1.2 Experimental set up

A random initial lattice configuration of live and dead cells is created, using a seed value and a 50% chance (initial configuration distribution probability) of each cell being made alive or dead. Tests are run for 1,000 generations on Machine A, and 10,000 generations on Machine B because Machine B is much faster making such longer runs more feasible. Lattice sizes begin at 128x128 and proceed at increments of 32x32 up to 2048x2048, also a spread of lattice sizes from 500x500 to 600x600 at increments of 1x1 are presented. Experiments are

conducted at workgroups sizes of 8x8 and 16x16 on both machines, but as Machine A is limited to a maximum of 512 threads per workgroup and Machine B is limited to a maximum of 1024, a workgroup size of 32x32 could only be used on Machine B. With these experiments OpenCL is utilised on the both the CPU and GPGPU.

3.5.1.3 Experimental results

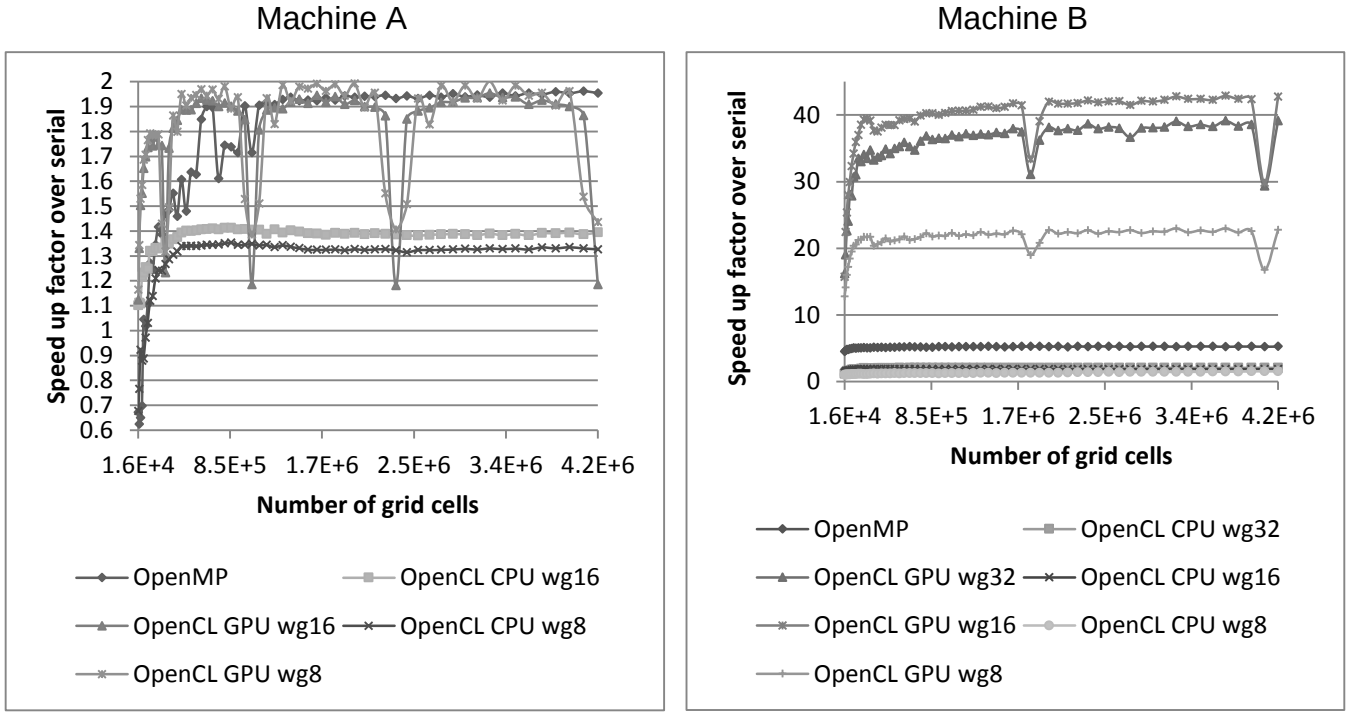


Figure 3.4, Speed ups over the serial implementation for OpenMP and OpenCL on the GPU and CPU, at 1,000 generations on Machine A, and 10,000 generations on Machine B, for lattice sizes of 128x128 to 2048x2048, at increments of 32x32.

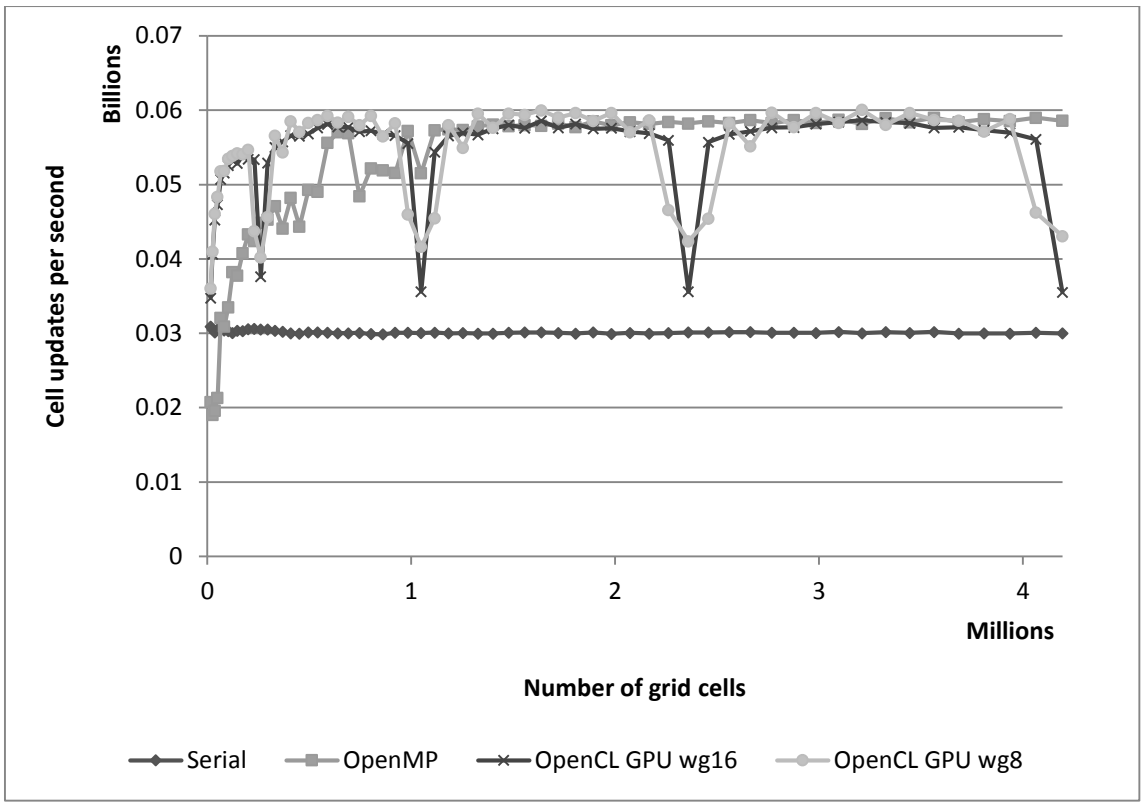


Figure 3.5, Cell update rates (per second) for the serial implementation, OpenMP and OpenCL on the GPU and CPU, at 1,000 generations on Machine A, for lattice sizes of 128x128 to 2048x2048, at increments of 32x32.

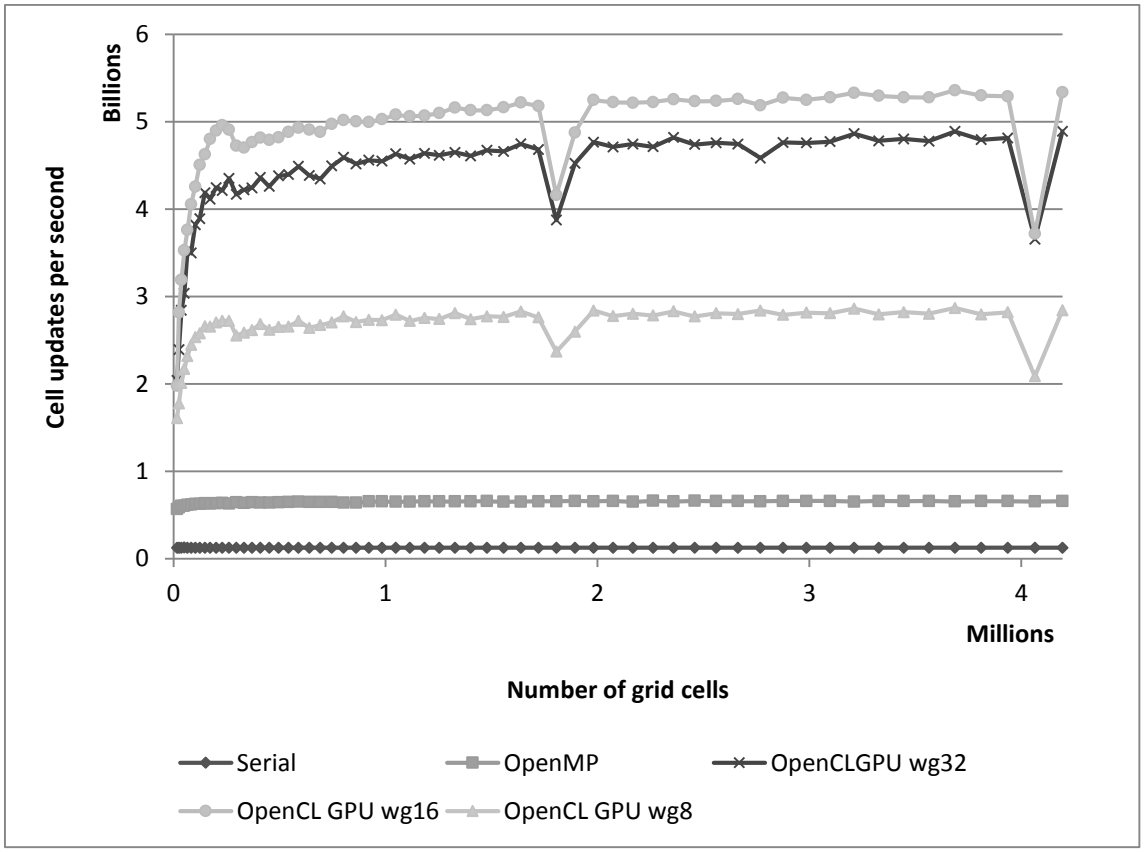


Figure 3.6, Speed ups over the serial implementation for OpenMP and OpenCL on the GPU and CPU, at 10,000 generations on Machine B, for lattice sizes of 128x128 to 2048x2048, at increments of 32x32.

Figure 3.4 demonstrates several key factors involved with lattice and workgroup sizes; firstly OpenCL running on the CPU is not very competitive, whereas OpenMP appears to scale fairly well to the number of CPU cores. Although OpenCL on the GPGPU performs up to 40x that of the serial CPU implementation on Machine B (the newer of the two machines), this is not in scale with the number of processing cores, although later in this chapter the question of which factors lead to this level of performance increase are addressed. Secondly it is clear to see that small lattice sizes are affected by the overheads of parallelisation which on the GPGPU include the transfer and as such gain lesser performance increases than larger lattices, with this stabilising at approximately 800x800 sized lattices. A workgroup size of 16x16 shows the best performance increases across both machines, and is therefore utilised in the experiments in the rest of this chapter. Finally it can be seen in Figure 3.4 that for particular grid sizes, performance decreases abruptly. On Machine A this occurs at lattice sizes of 512x512, 1024x1024, 1536x1536, and 2048x2048; whereas on Machine B this occurs at lattice sizes of 1344x1344 and 2016x2016. This is shown in Figure 3.7 to be due to the number of workgroups, and is therefore attributed to load balancing of the number of work groups to the number of hardware compute cores.

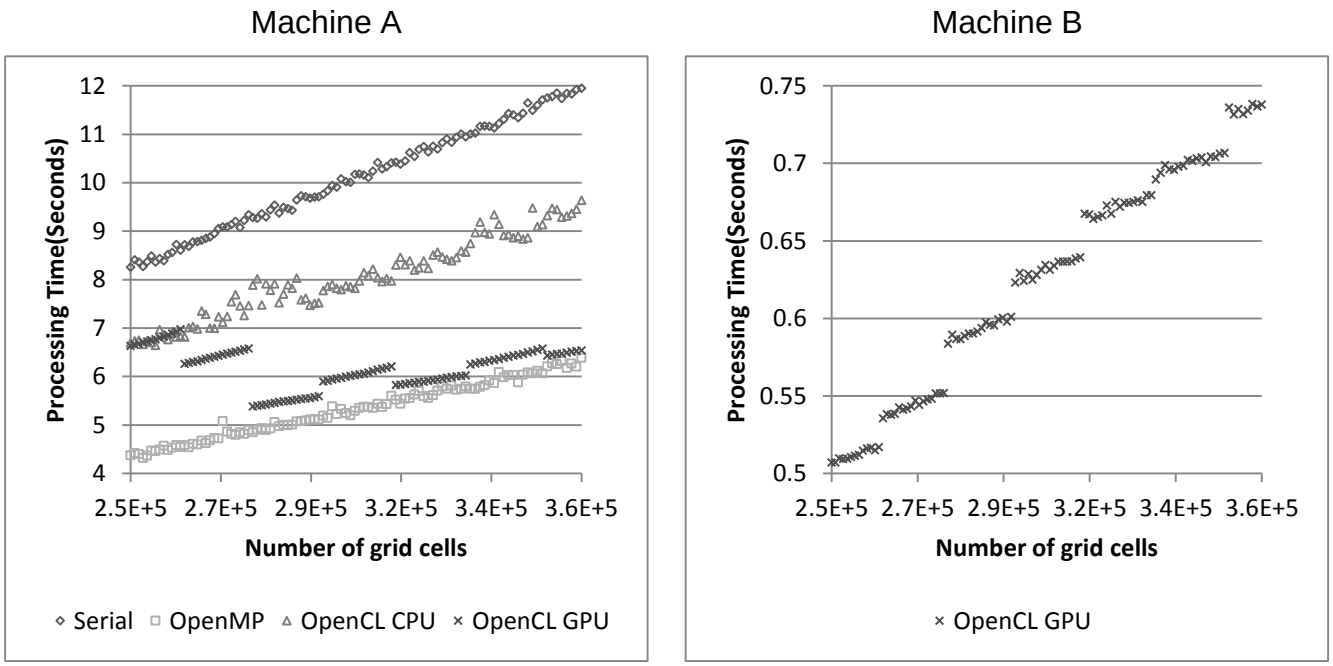


Figure 3.7, Processing times for OpenCL on the GPGPU, and for Machine A only on the OpenCL CPU, OpenMP and Serial implementations, for 500x500 to 600x600 lattice sizes in increments of 1x1, with a workgroup size of 16x16.

Figure 3.7 shows that both the CPU (sequential and parallel), and GPGPU approaches all scale linearly with the number of cells being processed, which demonstrates that the CA's complexity is based around the work in each cell. Where the GPGPU shows abrupt performance decreases (shown in Figure 3.7), this is associated with ‘load balancing’ of the workgroups to available compute cores of the GPGPU. Figure 3.7 indicates that the load balancing should be associated with the number of workgroups within the given lattice sizes. This is illustrated by the way that the performance on the GPGPU steps abruptly after each 16 successive grid size, where the number of workgroups changes.

3.5.1.4 Conclusion

Both CPU and GPGPU show linear increases in processing time with the number of cells/threads, however due to transfer and other overheads there is an offset, and due to the greater computational power of the GPGPU its processing times increase at a lesser gradient; therefore, a sufficiently large grid is required in order to gain the most efficient use of GPGPU hardware and thus the greatest speed-up factor. Where this result is somewhat expected, it is informative to see the scale of this threshold number of cells/threads is relatively large compared to the number of hardware cores upon the GPGPU. In addition, there are exceptions where particular lattice sizes or rather the number of workgroups within, give lesser performance than lattice sizes with similar numbers of workgroups; this is attributed to the way GPGPU hardware distributes workgroups to be processed between available compute cores.

3.5.2 Lattice size and GPU Memory types tests

3.5.2.1 Experimental Set up

The GPGPU present three memory types including ‘global memory' which is essentially the RAM on the GPU card; ‘local memory’ which is on-chip and therefore much faster but limited in space and scope to a single compute core; and finally ‘image memory’ (sometimes referred to as texture memory) which is memory specific to the native task of the GPGPU as a graphics processor and is cache-lined even in older models as well as having special hardware for dealing with border conditions. Tests are conducted using the novel texture based

memory layout described in section 3.3.2, and finally, tests are also performed with vectorisation (folding) and global memory.

3.5.2.2 Experimental Results

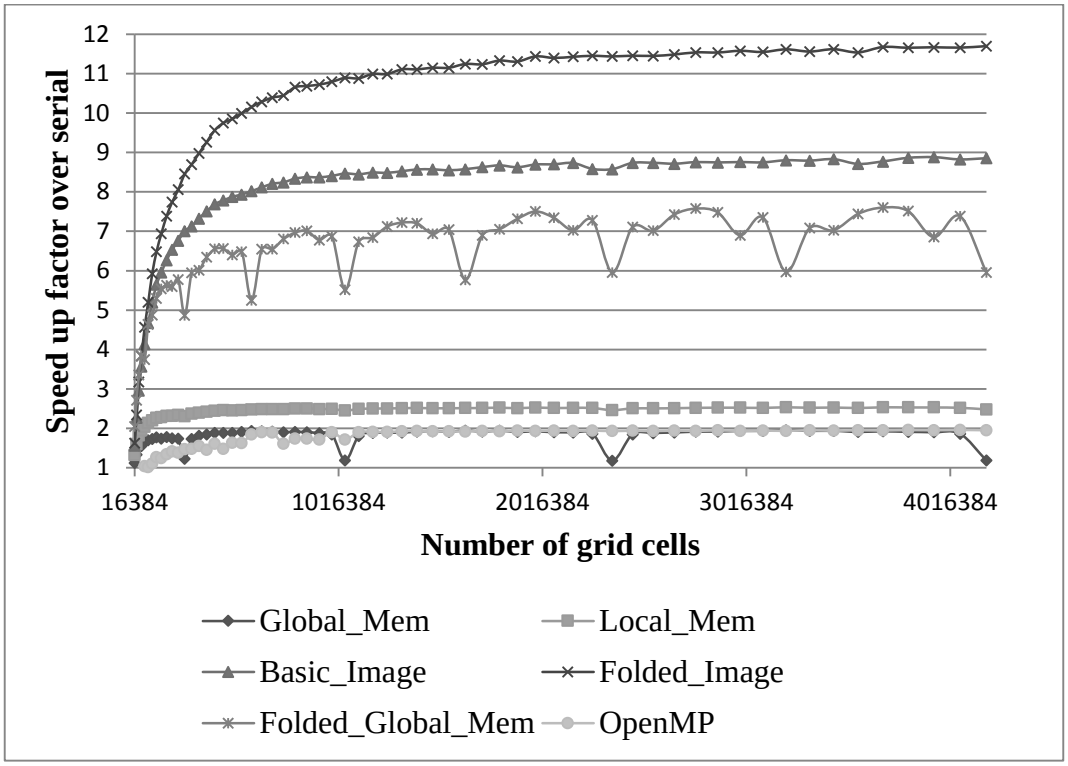


Figure 3.8, Machine A, Speed ups over CPU serial implementation for parallel CPU (OpenMP), and the OpenCL memory algorithms on the GPGPU.

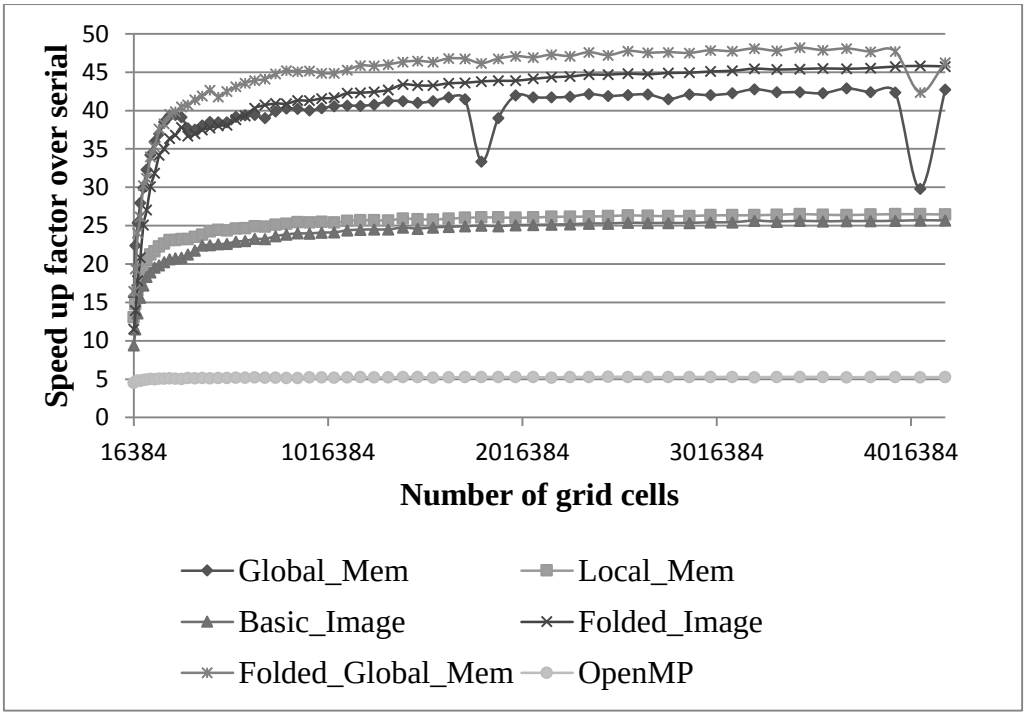


Figure 3.9, Machine B, Speed ups over CPU serial implementation for parallel CPU (OpenMP), and OpenCL memory algorithms on the GPGPU.

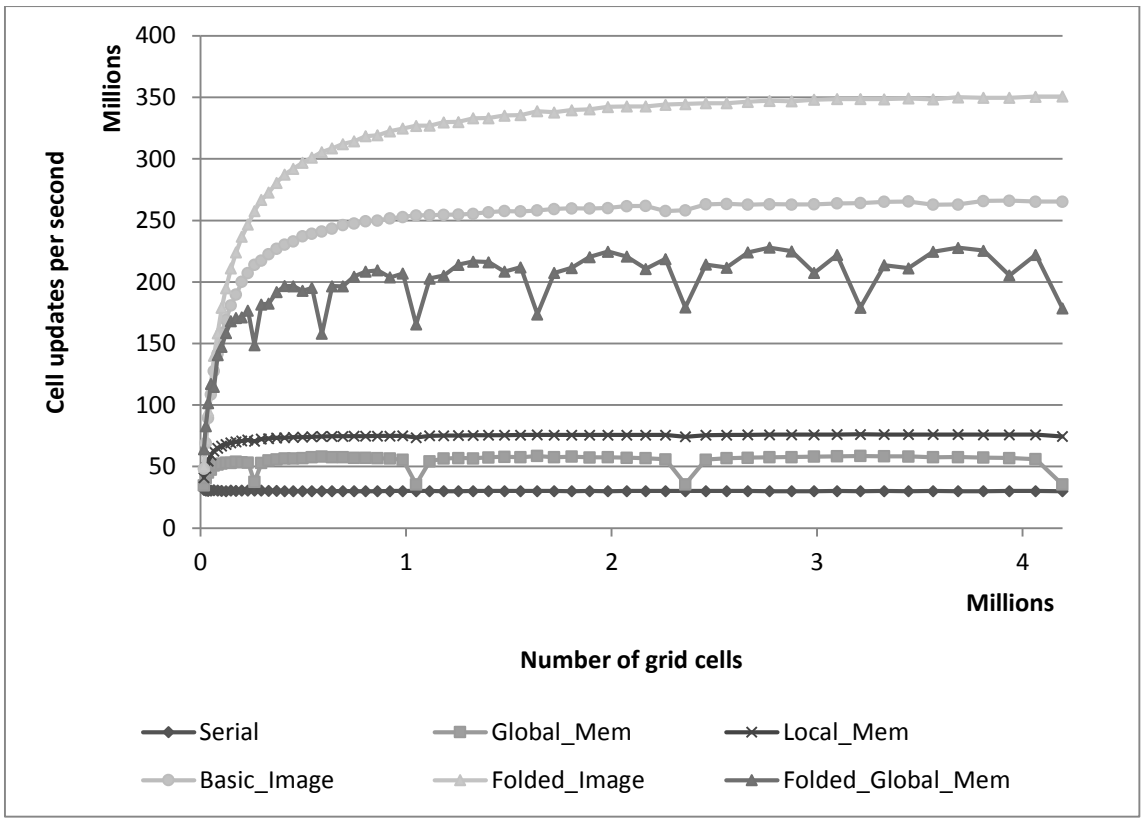


Figure 3.10, Machine A, Cell update rates (per second) for CPU serial implementation, parallel CPU (OpenMP), and the OpenCL memory algorithms on the GPGPU.

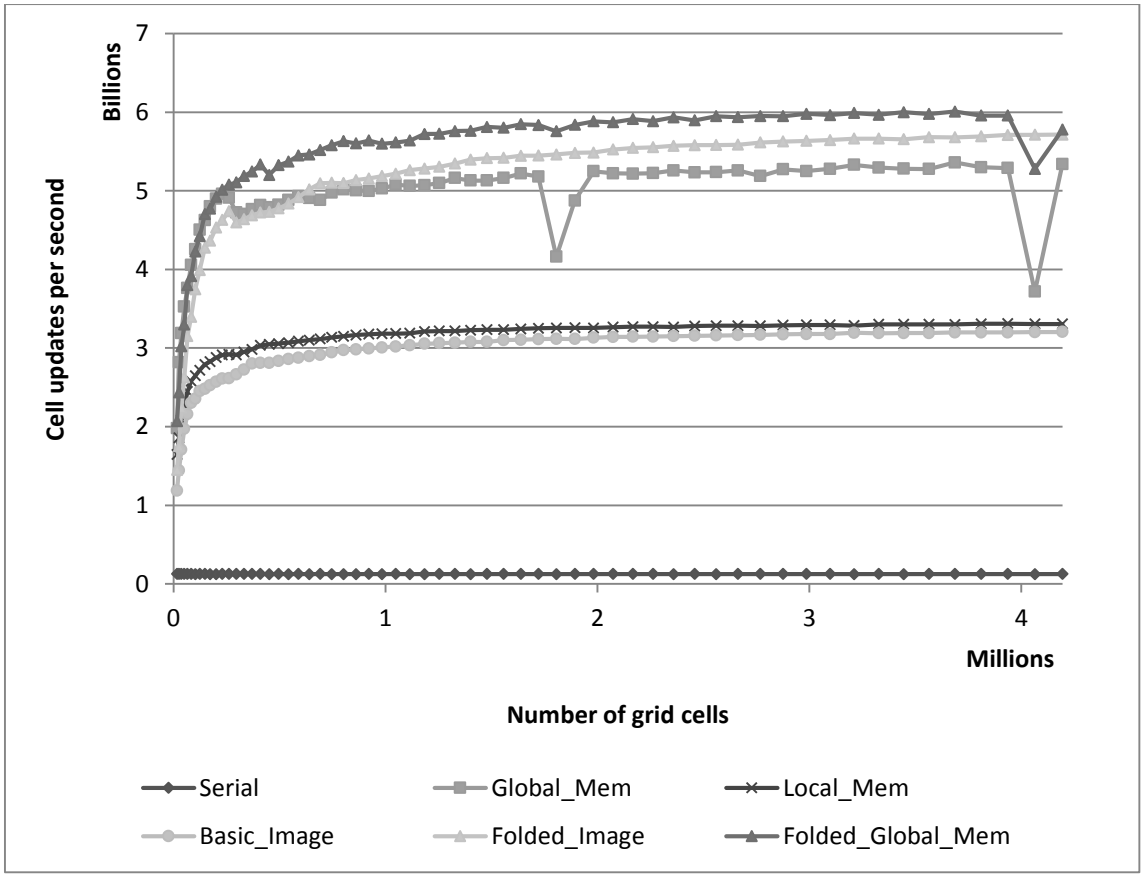


Figure 3.11, Machine B, Cell update rates (per second) for CPU serial implementation, parallel (OpenMP) implementations, and OpenCL memory algorithms on the GPGPU.

3.5.2.3 Discussion

One of the major advantages of the folded texture method is that it has automatic use of caching so as the re-accessing time of the data, when each cell is accessed as both a main cell and a neighbour for a number of other cells. However, the more modern of the two GPU's has automatic caching on its global memory which leads to a less gain in using this method compared to that of the older GPU. Additional gains are still possible by using such a tailored method to the graphical hardware in question.

3.5.2.4 Conclusions

Figure 3.8 and Figure 3.9 shows a marked difference in performance between the two tested machines, due to the introduction of cache-lined global memory in the Fermi (Machine B) generation of GPGPU's. Local and image memory gain greater speed-ups than global memory alone on Machine A, whereas on Machine B local and image memories are less efficient than global memories due to the more efficient caching and the need to explicitly copy data to the local and image memories. When vectorisation (folding) is applied both global and image memories show an increase in performance. For Machine A, the vectorised image/texture memory performance best, but for Machine B, it is the vectorised global memory that is the top performer. This is due to the more efficient cache-line global memory of the Fermi chip with Machine B.

3.5.3 Initial configuration distribution probability and Activity tests

A further variation in cellular automata is the extent to which their formulation in terms of starting conditions and rule sets leads to activity (i.e. the number of 'alive' cells) over the life of the CA. The standard 2-state game of life from random conditions for instance is known to produce a set of short-lived static and mobile structures (e.g. gliders) and will eventually converge on a stable state that will include some oscillating structures. Clearly, the number of alive cells in the CA will change over time, and in the case of the game of life, will start high and converge to a stable minimum. The following work investigates the impact of the ratio of 'alive' cells in the initial CA and records the level of activity in the CA to determine their effect on the potential speed-up of the CA on GPU hardware. This allows for an investigation of the effects of the resulting simulations produced

by the CA have on the processing speed up provided by the GPU, where such understanding can only come from an understanding of the rule set in questions.

3.5.3.1 Experimental set up

A separate implementation is used to count the number of live cells (those with a state of 1) and the number of live neighbouring cells for each cell to ensure that the timing results are not biased. Having counted the number of live cells and neighbours for every cell in every generation, an average proportion is calculated by dividing by the number of cells and generations. Tests are performed over a spectrum of lattice sizes, and initial configuration distribution probabilities which are used in the creation of each cell being alive or dead in the initial configuration. Ten different seed values for the random number generator are used in these experiments. The timing tests are repeated for 15 trials, but this is not necessary for the counts, as they are deterministic. Tests were conducted on a 512x512 lattice size for 1,000 generations on both machines with a workgroup of 16x16 are used.

3.5.3.2 Experimental results

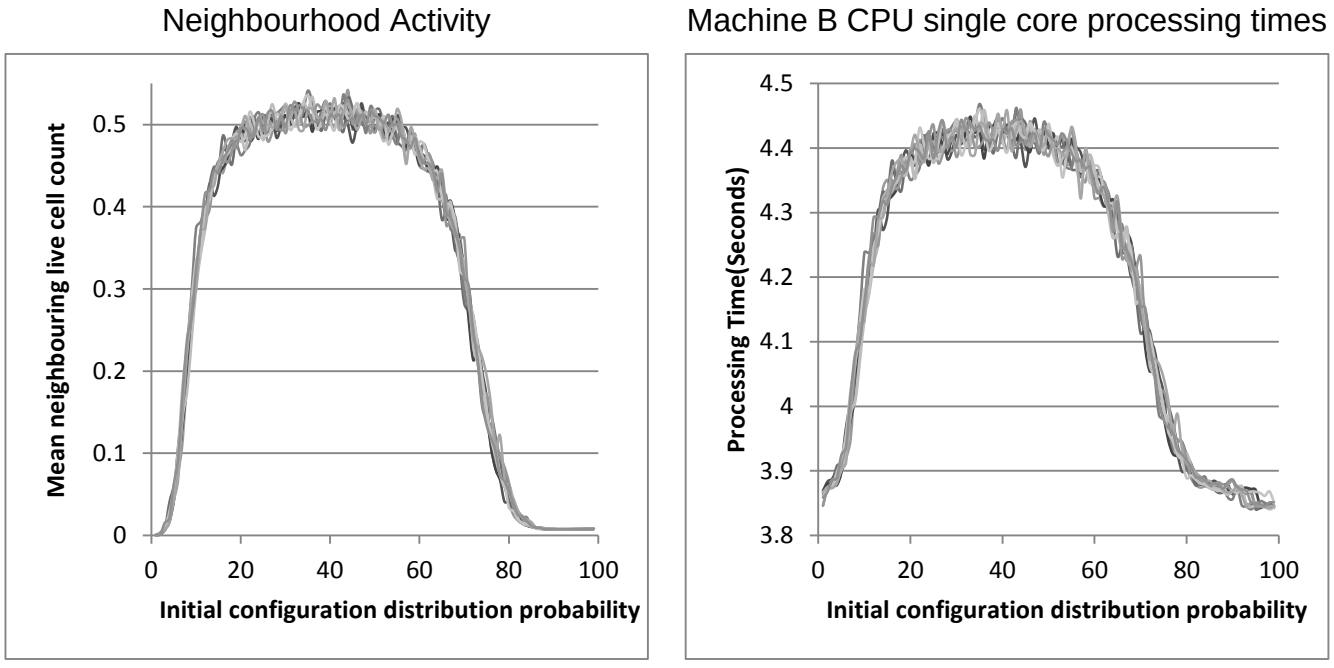


Figure 3.12, Average (mean) neighbourhood live cell counts per cell over the entire simulation for a range of initial configuration distribution probability/chances of live cell creation in the initial configuration (left), and the processing time on a single CPU core for the same ranges (right), processed at a lattice size of 512x512 for 1,000 generations.

Figure 3.12 shows firstly that as the neighbourhood counts are averaged over the entire simulation, this restricts the variation due to the difference in the underlying patterns formed through the differently seeded simulations, therefore these averages are a measure of *Activity* over the entire simulation. Secondly Figure 3.12 shows that there is little to no activity below the initial configuration distribution probability levels of approximately 5% and above 80%, and between these, the level rises to a plateau. This plateau level is surprisingly low with an average live neighbourhood cell count of approximately 0.5 live neighbours out of a possible 8. The right pane of Figure 3.8 shows that processing time is highly correlated with the activity levels seen in the left pane. It can also be seen in Figure 3.12 (right) that with no activity levels, the processing time is dominated by other work within each cell; i.e. a baseline non-varying amount of arithmetic computational work and the memory look up of the neighbourhood within each cell.

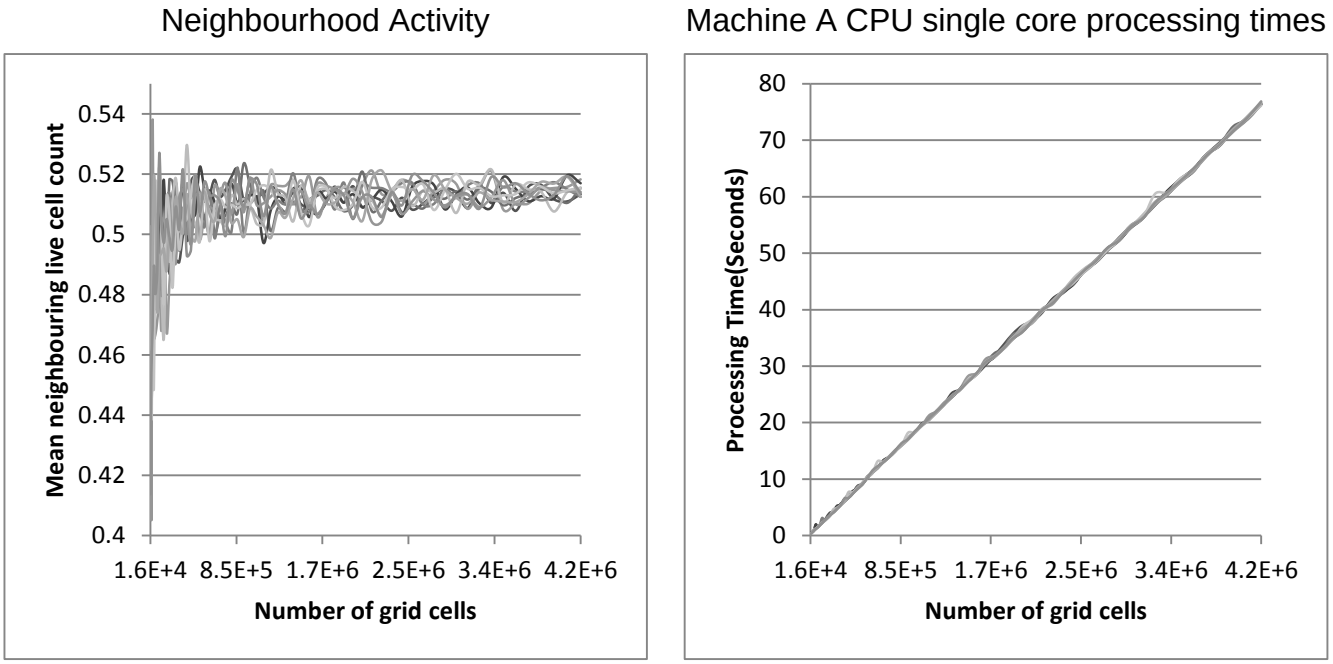


Figure 3.13, Average neighbourhood live cell counts per cell over the entire simulation, when using an initial configuration distribution probability of 50% for a range of lattice sizes of 128x128 to 2048x2048, at increments of 32. (left) Note the difference in the scale of the y-axis, and the processing time on a single CPU core for the same ranges (right), for 1,000 generations.

Figure 3.13 shows activity variation when using the same 50% initial configuration distribution probability for different lattice sizes. This variation is present due to the difference in the underlying patterns formed, but is small and decreases as the lattice size increases. The processing times accordingly show

very little variation and are thus dominated by the symmetrical work within each cell of the lattice.

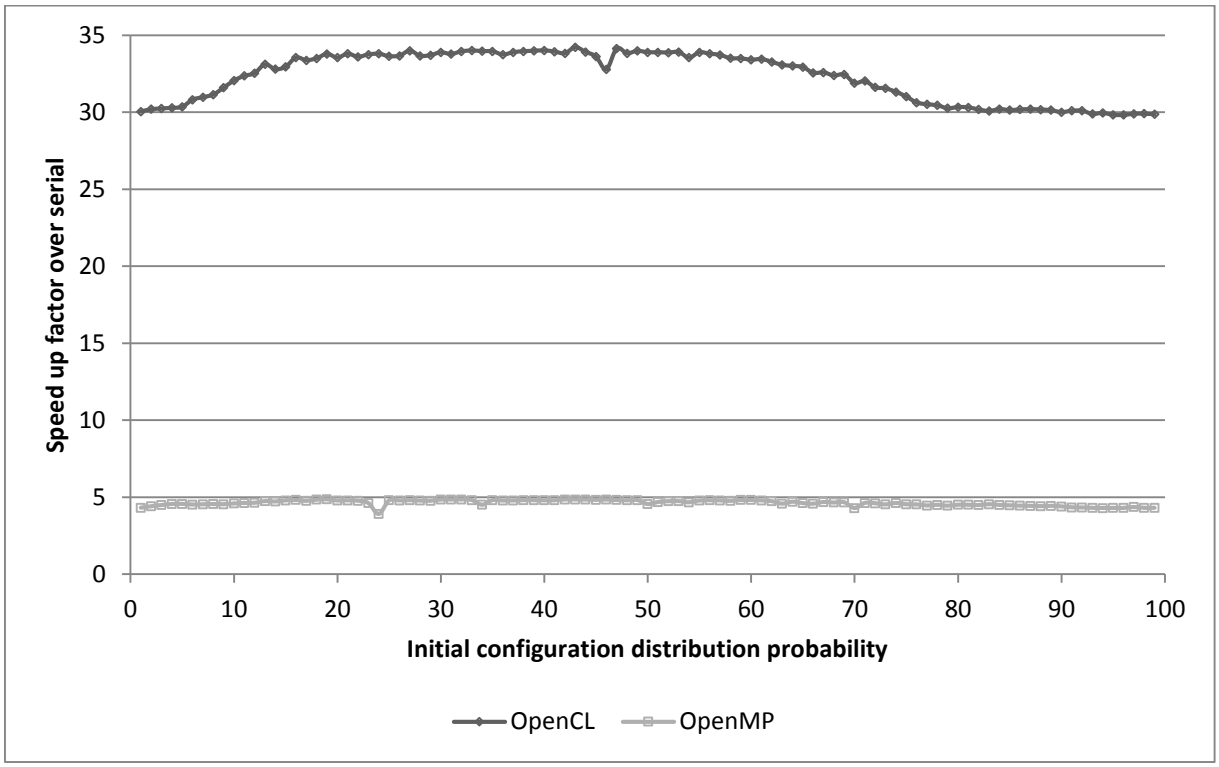


Figure 3.14, Speed-ups relative to the serial implementation for OpenMP, and OpenCL on the GPGPU (workgroup size of 16x16) on a 512x512 lattice size at 1,000 generations, over a range of initial configuration distribution probability values from 1% to 99% at intervals of 1%; results shown for Machine B.

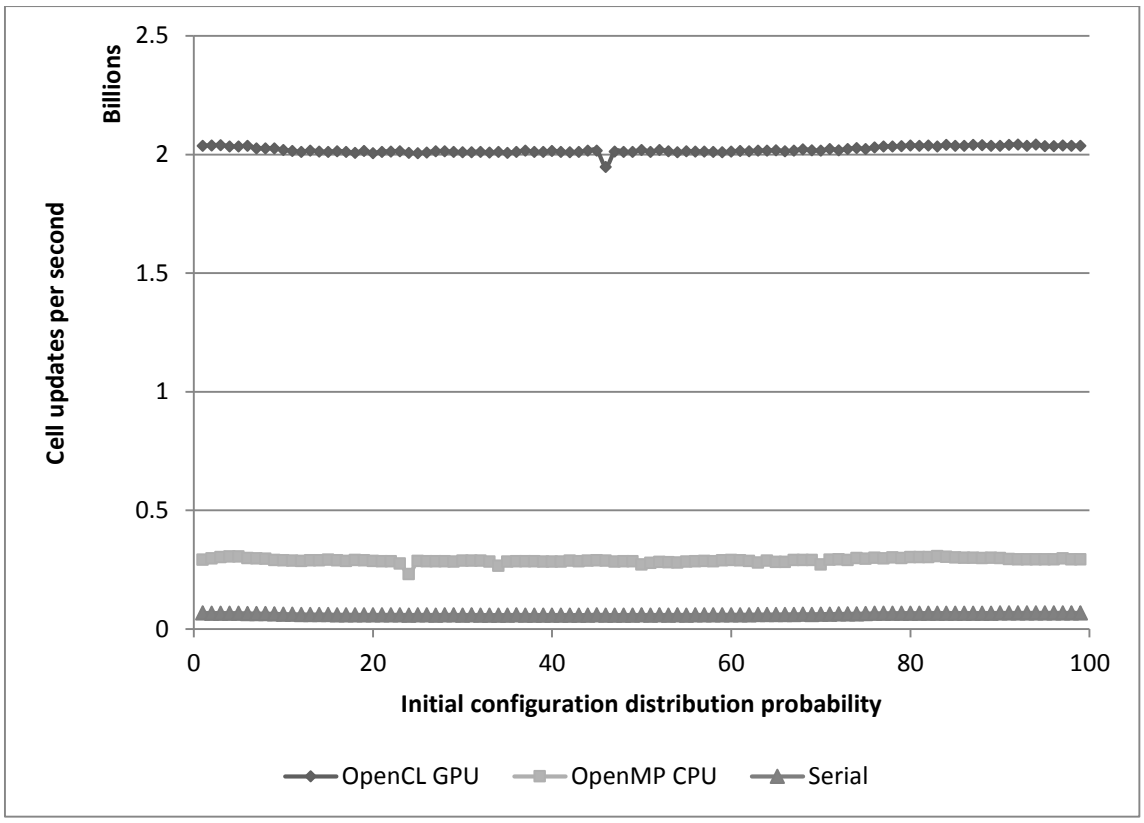


Figure 3.15, Cell update rates (per second) for serial implementation, OpenMP, and OpenCL on the GPGPU (workgroup size of 16x16) on a 512x512 lattice size at 1,000 generations, over a range of initial configuration distribution probability values from 1% to 99% at intervals of 1%; results shown for Machine B.

Figure 3.14 shows that it is this increase in arithmetic from the counting of 'alive' cells, due to the correlation with the activity levels shown in Figure 3.12 which, when parallelised, leads to proportional increases in relative performance between the parallel approaches and the sequential approach. This effect is more greatly noticed in the GPGPU due to the greater level of hardware ALU (Arithmetic Logic Unit) parallelism. Here, results from only Machine B and are only shown for a single seed; however, these are representative of results recorded from all initial configurations and clearly show that the GPU, and to a lesser extent, the parallel CPU are able to increase speed-up when activity levels are high.

3.5.4 Number of states tests

The intention with these experiments is to extend the inferences made on these simple CA to more complex CA with more states, and even to continuous states. The experimentation in this section uses the novel multi-state variants of the game of life, to test how the number of states might affect the processing

times and speed-ups of the GPU. As more complex real world rule sets with inevitably use much large number of states and possible even continuous scales, there is the need to understand how the number of states will affect the relative speed-ups of the GPU.

3.5.4.1 Method

Since the game of life rule set specifically has only two states, it has been adapted here to a multi-state interpretation in order to test the effects of a variable number of states. Many such interpretations have been created and two interesting rule sets are presented which are called MSGOL (Multi-State Game Of Life, section 3.3.) and MSGOL4 (version 4, section 3.4.). MSGOL at 3 or 4 states produces large areas of what appears as chaotic behaviour, where small snake like collections of cells are born, move around and die in between the chaos. At larger numbers of states, this forms maze-like patterns over the whole grid, with fluctuations which move over the grid as if searching for a stable global pattern. MSGOL4 at 3 and 4 states look more like the game of life, so much so that new and distinct gliders are detected at both of these numbers of states. However, as the number of states is increased in MSGOL4, larger areas of what appears to be chaotic behaviour consume the simulation.

Videos of the MSGOL and MSGOL4 rule sets are various numbers of states can be viewed online @ :
http://www.sciencedirect.com/science/article/pii/S0743731514002044#appd003

In the timing tests, the lattices are populated with the same initial configuration as before. The number of states is modified by using a parameter within the decision tree rule sets. The decision tree is able to represent an increasing number of state transitions with the same decision tree because of the way it programmatically maps the relation between each state, as opposed to using an increasingly large look-up table.

3.5.4.2 Experimental set up

Experimental results are shown for MSGOL runs from 2 to 10 states. In these experiments a lattice size of 512x512, with a workgroup size of 16x16 are

again used (which notably is a badly load balanced size on Machine A), and run for 1,000 generations on Machine A, and 10,000 on Machine B.

3.5.4.3 Experimental Results

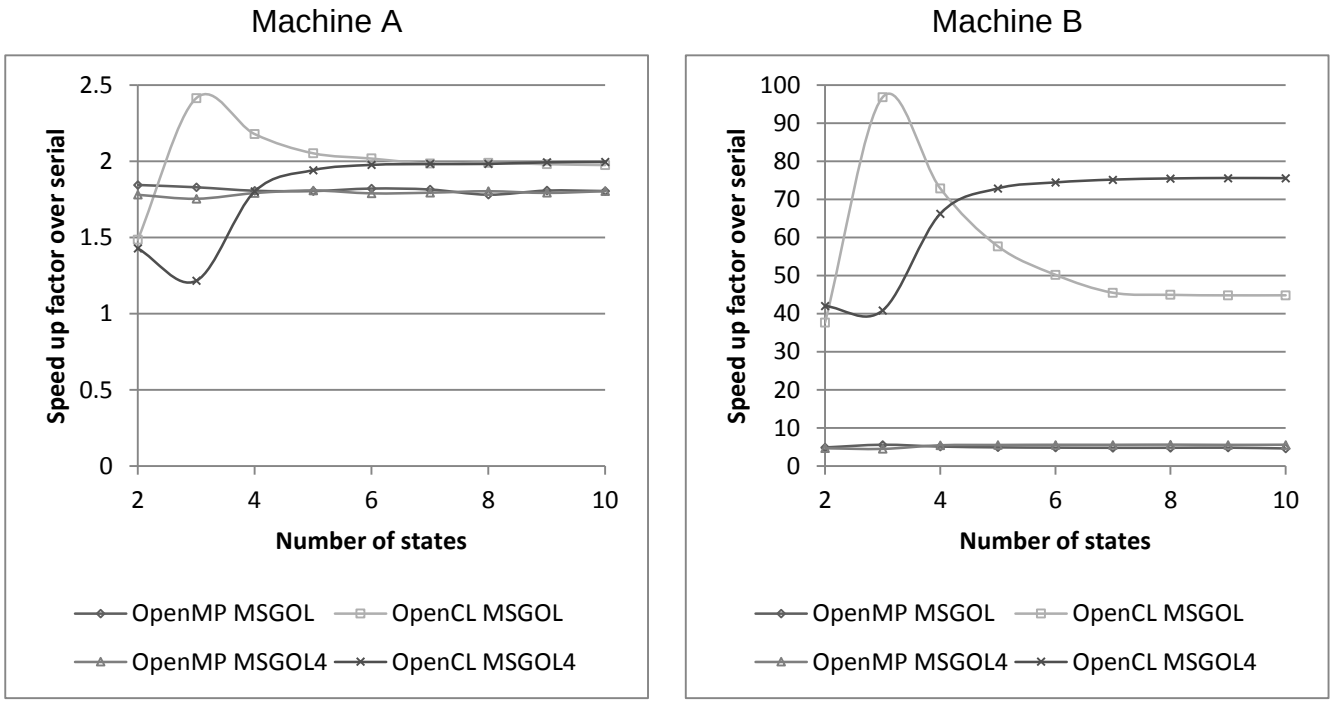


Figure 3.16, Speed-ups over the serial implementation for OpenMP, and OpenCL at a lattice size of 16x16, for 1,000 generations on Machine A, and 10,000 on Machine B. Showing resulting for the MSGOL and MSGOL4 rule sets with 2 to 10 states.

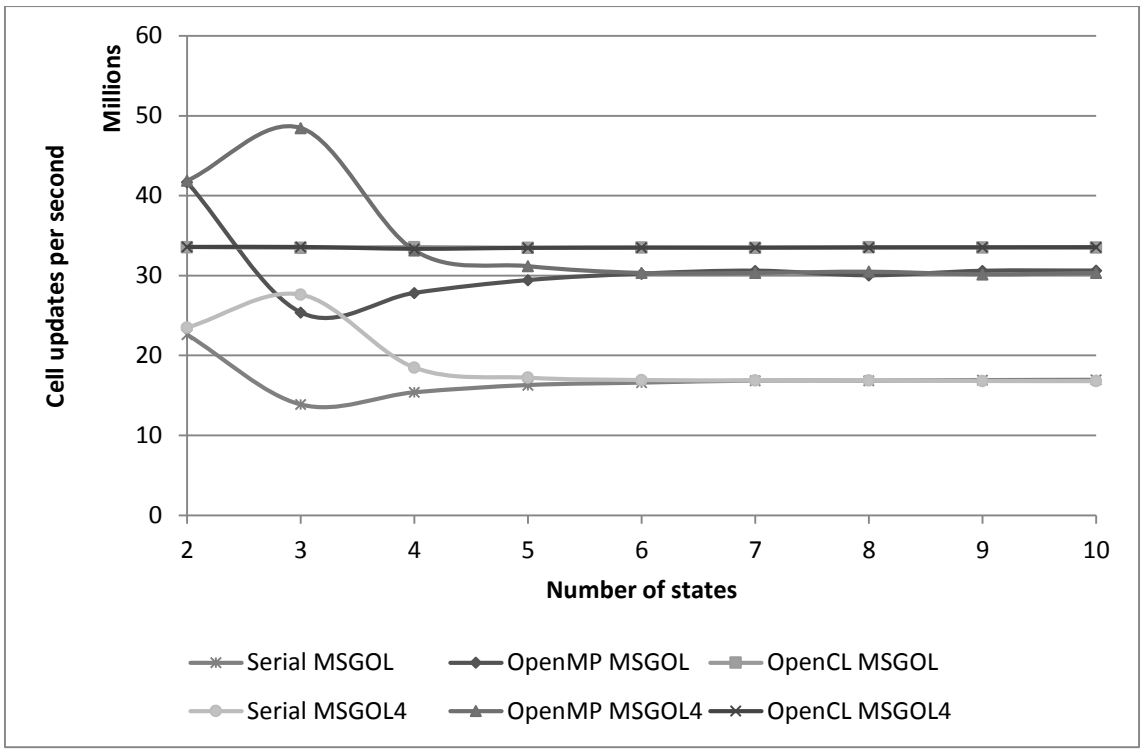


Figure 3.17, Cell update rates (per second) for serial implementation, OpenMP, and OpenCL at a lattice size of 16x16, for 1,000 generations on Machine A. Showing results for the MSGOL and MSGOL4 rule sets with 2 to 10 states.

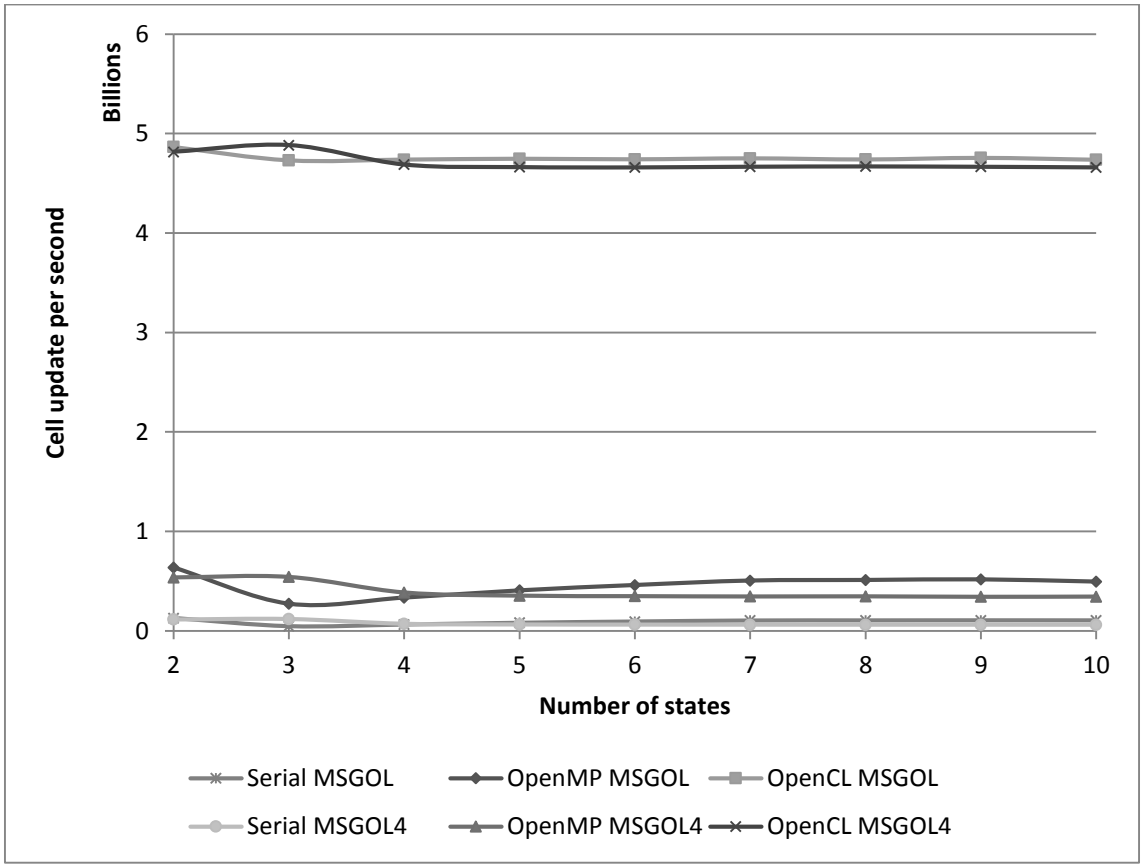


Figure 3.18, Cell update rates (per second) for serial implementation, OpenMP, and OpenCL at a lattice size of 16x16, 10,000 on Machine B. Showing results for the MSGOL and MSGOL4 rule sets with 2 to 10 states.

Figure 3.16 shows the relationship between the number of states (from 2-10) and the speed-up on the GPU. The graphs for the GPU and MSGOL show a peak around 3 states which then drops down to a converged rate of speed-up later for higher numbers of states. For MSGOL4, this situation is reversed with a dip in speed-up at 3 states. It is therefore clear that for the modified game of life rule sets, at least, the number of states does have an effect on the speed-up possible from GPUs but that the hardware has a very much larger effect (note the difference in axes ranges for Machine A and Machine B). However, the discrepancy between 3 states and the others was not expected and required further experimentation (shown in section 3.5.4.4).

3.5.4.4 Further experimentation with multi-state game of life variants

In the majority of cases the simulations take approximately the same amount of processing time irrelevant of the number of states, however in the area of the most variation in activity a large spike in performance for both rules can be observed in Figure 3.16, Figure 3.20, Figure 3.21, and Figure 3.22, as they change from one type of behaviour to another. As shown in Section 3.5.3, it is the variable amount of arithmetic carried out in each cell which directly relates to the processing time and consequent increases in performance. Therefore, it is necessary to account for the variable amount of arithmetic from the decision tree. The MSGOL and MSGOL4 rule sets both have leaf nodes which carry out a simple plus/minus-one calculation for the next state, therefore counting implementations have been created which, as well as counting the live cells and live neighbours per cell, also count the number of cells taking each leaf node of the decision tree rule sets. Figure 3.19 shows the binary decision tree represented by the MSGOL rule set (section 3.3.1.2), with leaf nodes labelled A-E, where it is node B that carries out an increment to the current state, and node E carries out a decrement to the current state in order to find the next state, and all other leaf nodes represent leaving the current state of main cell as it was in the previous generation. A similar decomposition of the MSGOL4 rule set is performed, with leaf nodes A-G, where nodes C and F are responsible for arithmetic operations. With both rules sets, as with the game of life rule, the operation time also depends on the number of live neighbours for each cell. The average live neighbourhood counts, and proportion of cells over the entire simulation for each leaf node of the decision tree for MSGOL and MSGOL4 rule

sets are shown in Figure 3.20 and Figure 3.21 respectively. In Figure 3.22, first the timing results from the MSGOL and MSGOL4 rule sets for the parallel CPU approach on Machine A are shown; this is compared to the combination of the variable amount of arithmetic (i.e. the average live neighbour counts, plus the leaf nodes, which carry out arithmetic), in order to demonstrate how it is again the variable amount of activity which causes the difference in processing time on the CPU. The GPGPU is found to have much smaller variations in processing time over the same area, which leads to huge computational speed up in the area of high activity, shown in Figure 3.22 and consequently explains the difference in performance seen in Figure 3.16.

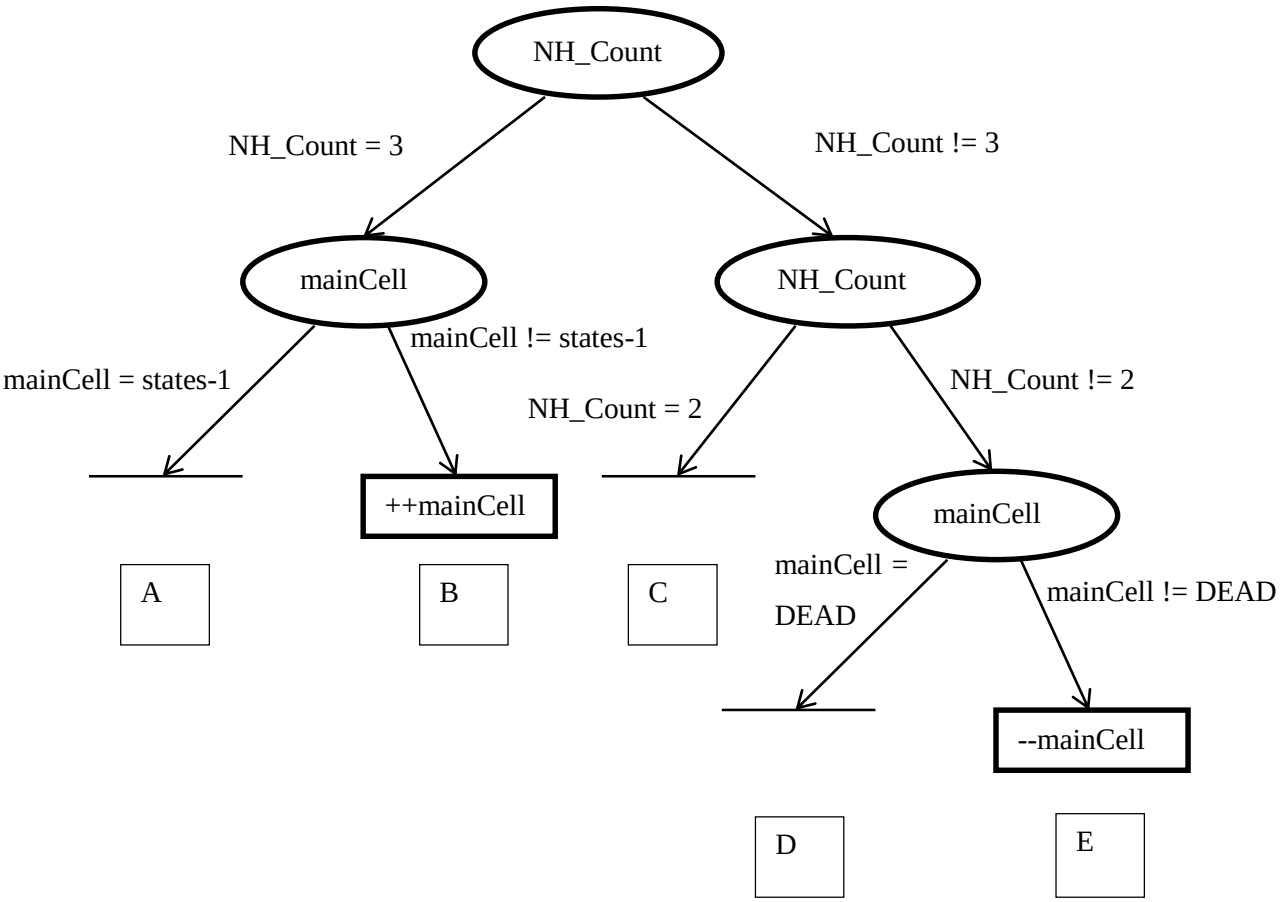


Figure 3.19, Binary decision tree version of the MSGOL rule set, with leaf nodes labelled A-E. With the variables '`NH_Count`' which represents the number of live neighbouring cells, and '`mainCell`' to represent the central main cell's value, and finally '`states`' to represent the number of states variable.

3.5.4.5 Experimental Results

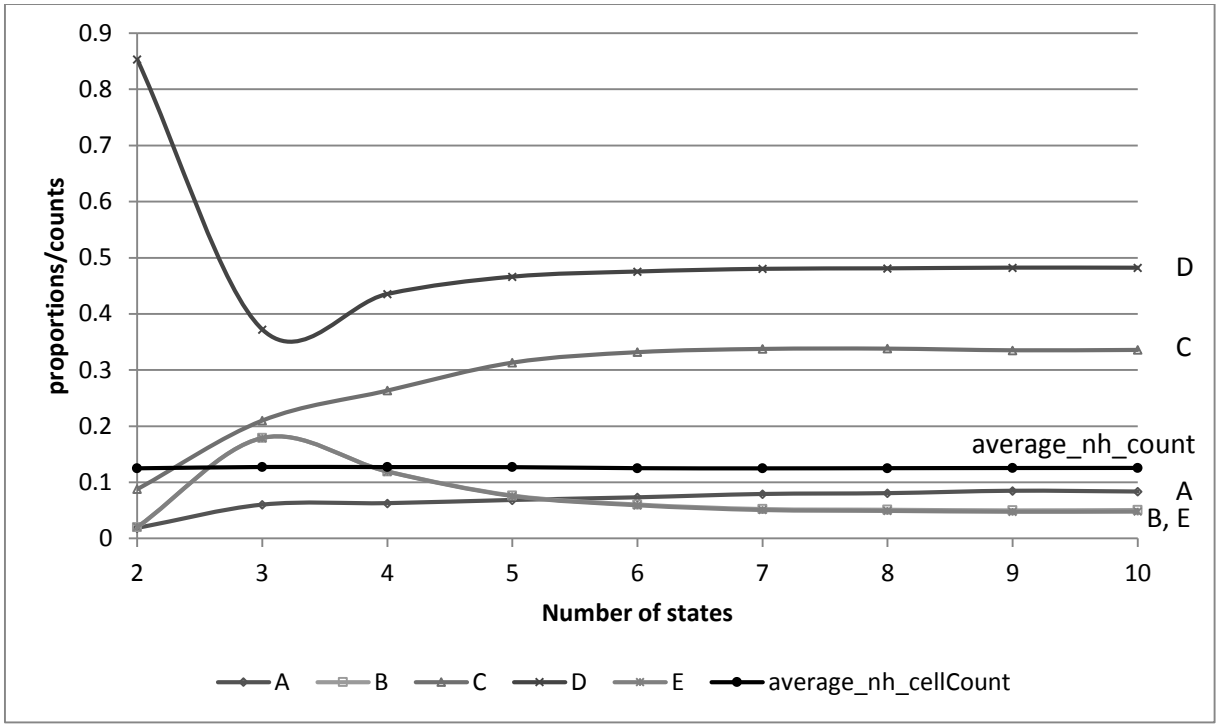


Figure 3.20, Average neighbouring cell counts for each cell and the proportion of cells over the entire simulation taking each possible leaf node through the rule sets MSGOL (which has leaf nodes A-E as shown in Figure 3.19.

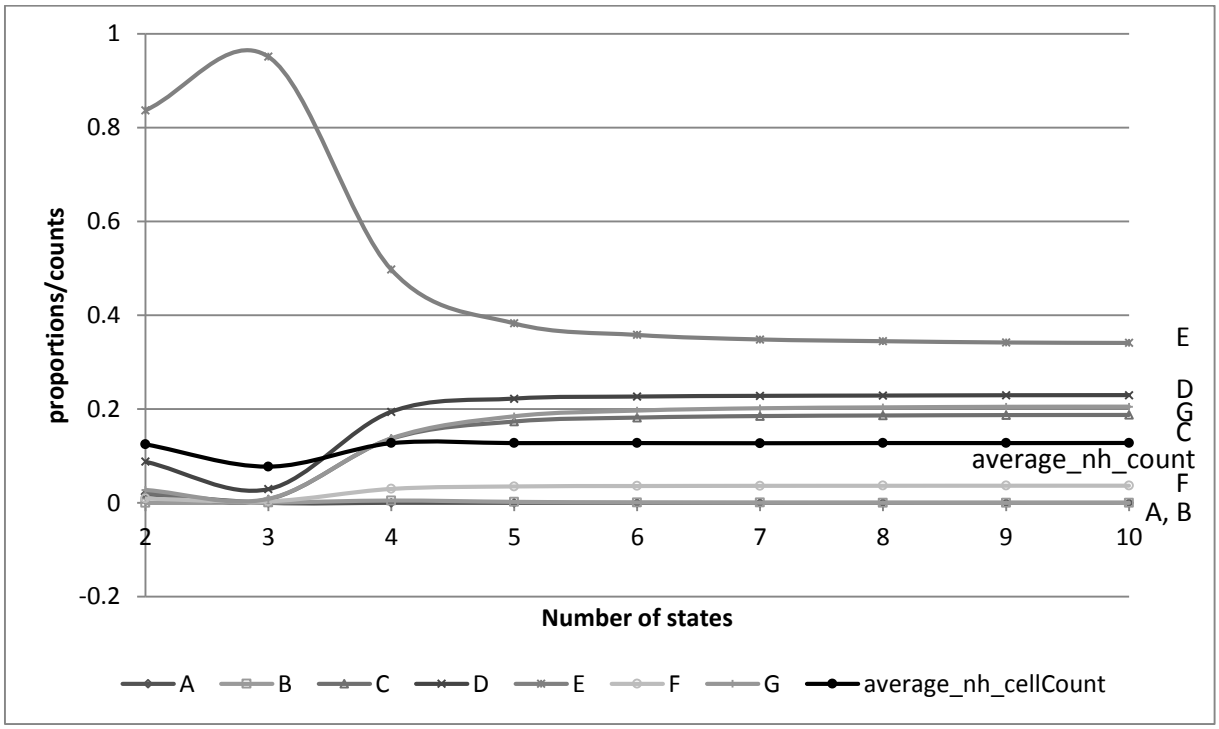


Figure 3.21, Average neighbourhood live cell counts, and proportion of cells over the simulation taking each leaf node for MSGOL4 rule set, on Machine A.

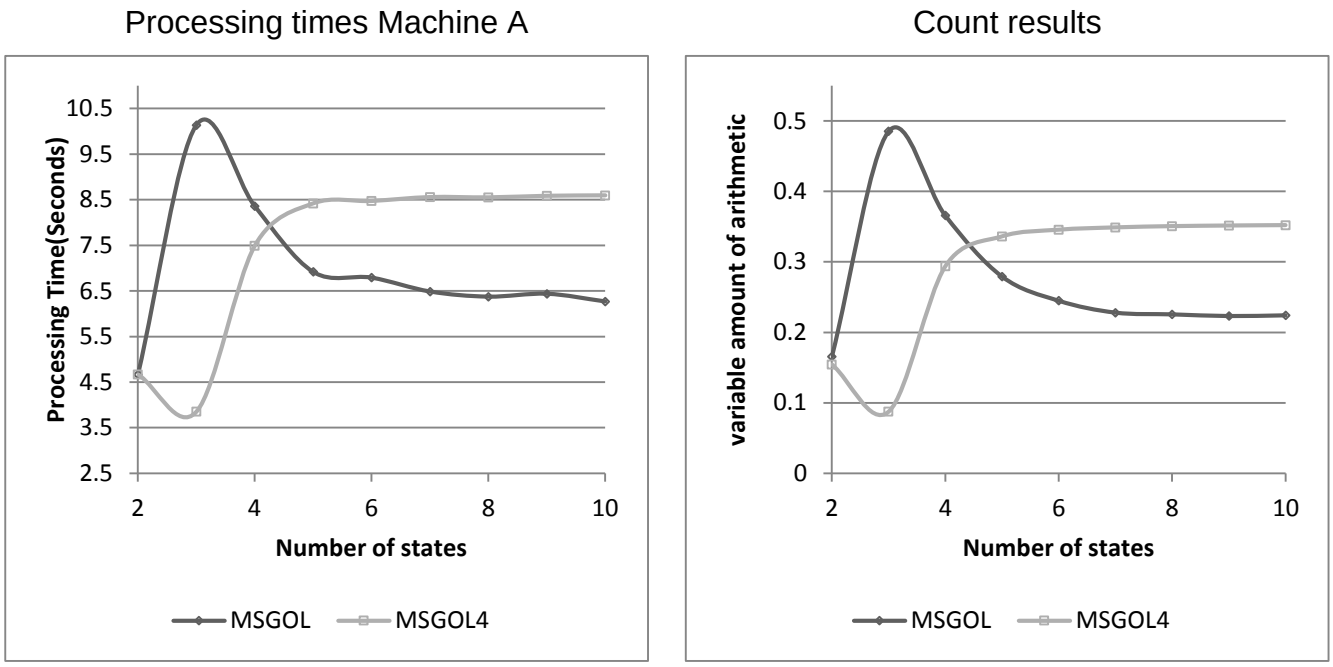


Figure 3.22, Processing times of the OpenMP implementations of MSGOL and MSGOL4 in comparison to each other for 2-10 states (left), shown (right) the theory of the arithmetic complexity by showing the average neighbourhood count (shows the number of increments of a counter, on average), plus the proportions of cells on average over the whole simulation which perform an arithmetic operation. In the case of MSGOL this is leaf nodes B and E, and for MSGOL4 leaf node C and F.

Figure 3.20 and Figure 3.21 demonstrate that the rule sets generate most of the extra arithmetic complexity compared to the neighbourhood counting. Figure 3.22 shows how it is indeed this arithmetic complexity caused by the resulting behaviour which causes the relative slowdown in the CPU processing, and Figure 3.16 shows how this also causes a relative speed performance increase from the GPGPU over the CPU in the same area.

3.5.4.6 Conclusions

This work showed that again, it is the level of arithmetic that is conducted by the rule set that is the main driver of speed-up. The specifics of the rule set and the decision tree implementation mean that the (relatively fast on GPU) addition and subtraction operations only occur at specific leaves of the tree. Simply put, the more often these leaves are used, the greater the speed-up on the GPU. Of course, this depends on the specifics of the rule set and the decision tree implementation, but this does mean that the optimisation of the rule set to maximise arithmetic and minimise memory operations is an important element of parallelising CA with GPUs.

3.5.5 Data types

As the state of the CA is stored in a specific data types varies e.g. integer, or floating point, and the level of precision given by the number of bytes used, this varies the performance of hardware. It is useful to know how previous experimental results will relate to a continuous CA which is used for flood models that will be investigated in Chapter 5: and Chapter 6:.

3.5.5.1 Experimental set-up

Experiments are conducted on two types of integer and two types of floating point data types, at a range of grid sizes. The char data type is a single byte integer, whereas the int type is a 4 byte integer. The float type is a 4 byte floating point, and the double is an 8 byte floating point type.

3.5.5.2 Experimental Results

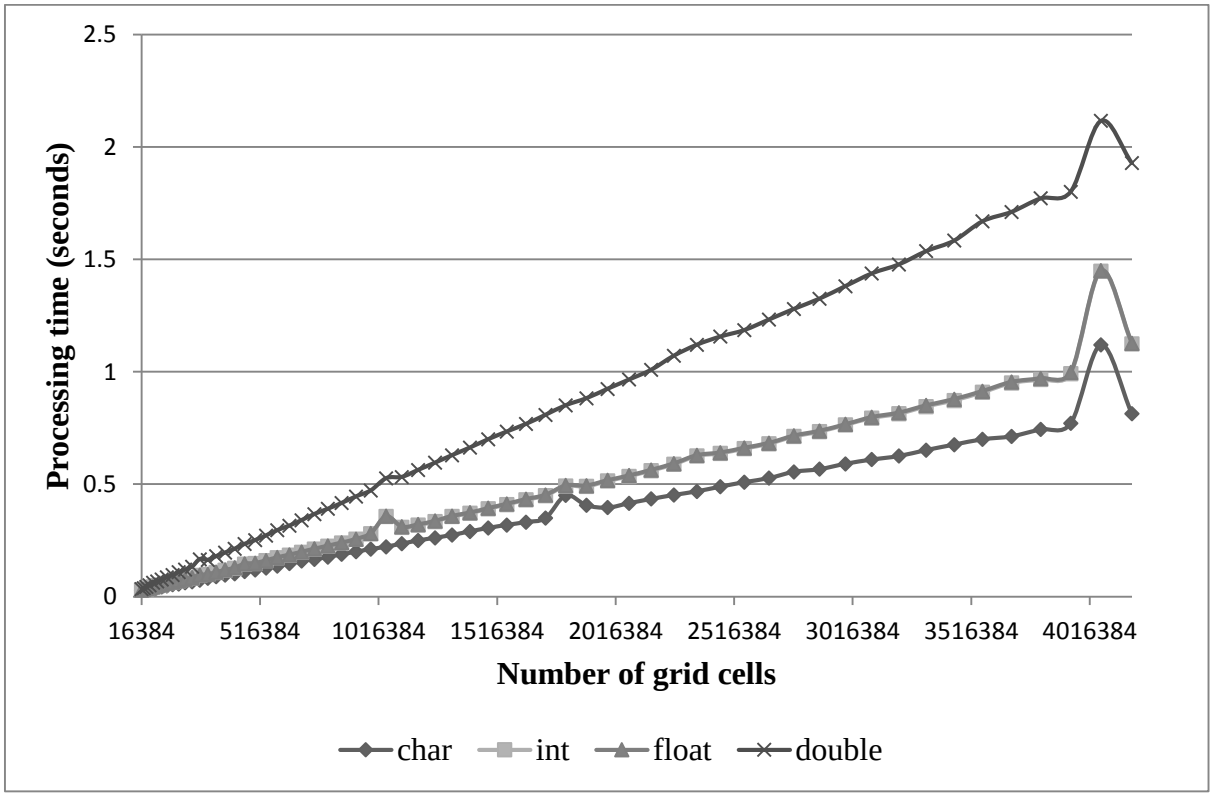


Figure 3.23, Processing times from the game of life with a 50 percent active distribution and run for 1,000 generation, at various grid sizes, and with the 4 different data types, on the GPU.

The results of Figure 3.23 demonstrate there is very little difference in processing times for the two data types which are the same size (int and float, both being four bytes), the char type with only a single bytes takes the least time,

where the double floating point types takes the most times. It would appear that the size of the data types has large effects on the processing time of the GPU.

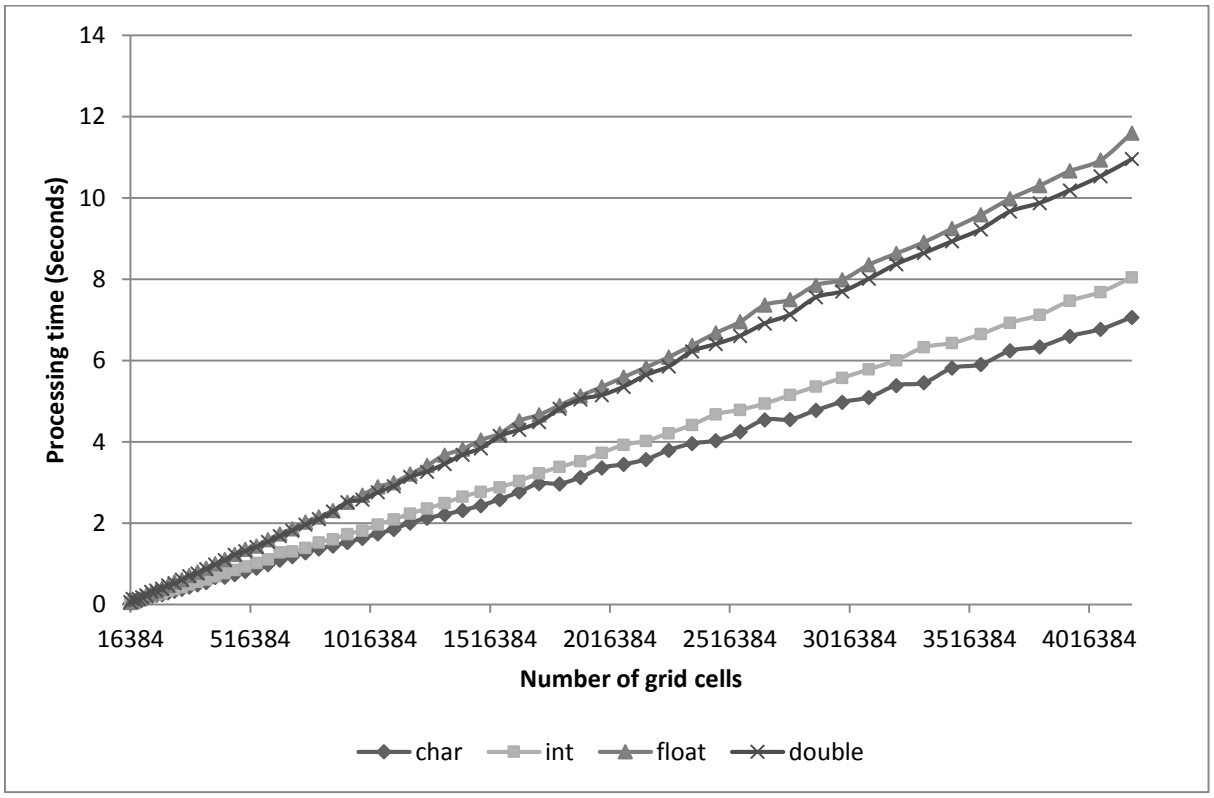


Figure 3.24, Processing times for the OpenMP implementation, with 1,000 generations of the game of life with a 50 percent initial distribution configuration.

Figure 3.24 shows the processing times for the CPU, where the char type does take the least amount of time as with the GPU, however the int type which is 4 times as large, only takes a marginally longer amount of time to process on the CPU. It is thought this is because it is primarily hardware constrained and that the hardware being 64/32bit is tailored for these larger data types. Both floating point data types take a sizeable amount of extra time to process compared to the integer types, however there is even less difference between the processing time of the two different floating point types (float and double). In fact, since the CPU has for some years been tailored to operate with the higher precision double floating point types, it can be seen in Figure 3.24 that it is actually slightly faster at processing this types, even though it is twice the size of the float type.

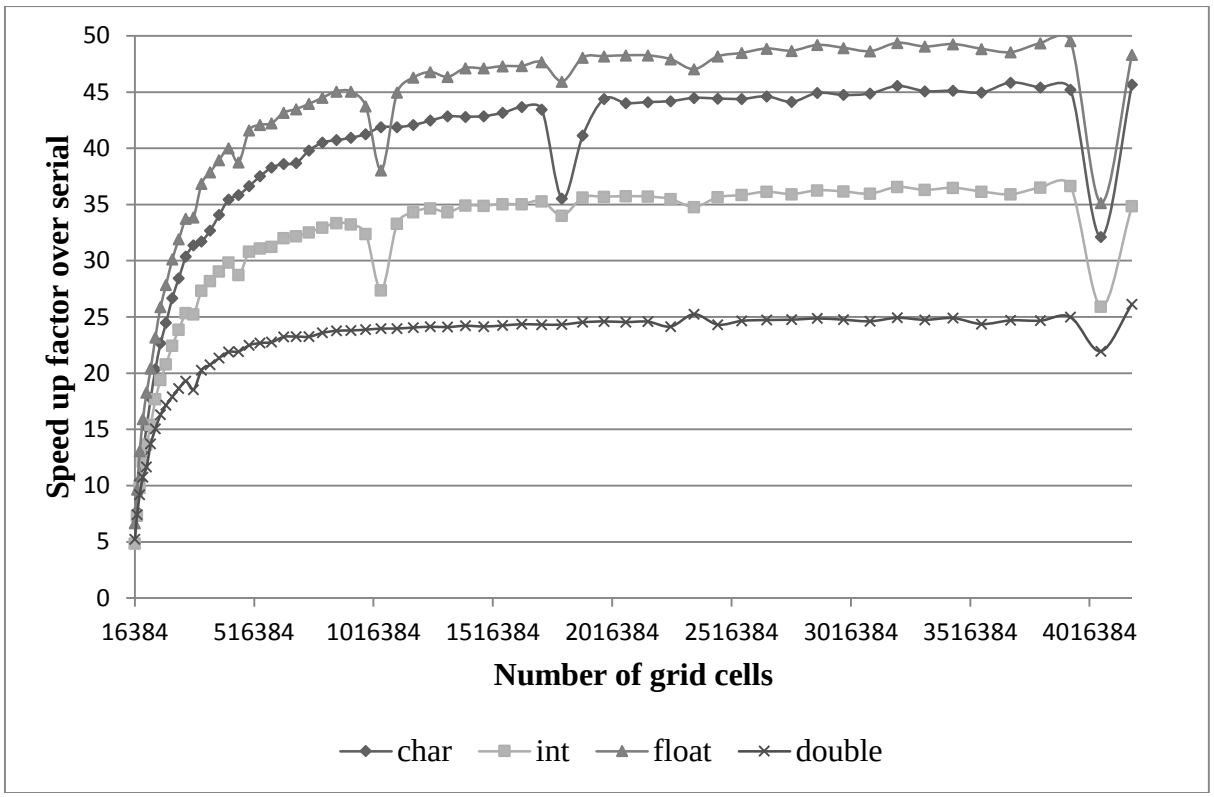


Figure 3.25, Speed-ups of the GPGPU over the CPU of several different grid sizes, for 1,000 generations, using different base data types of char, int, float and double floating point numbers.

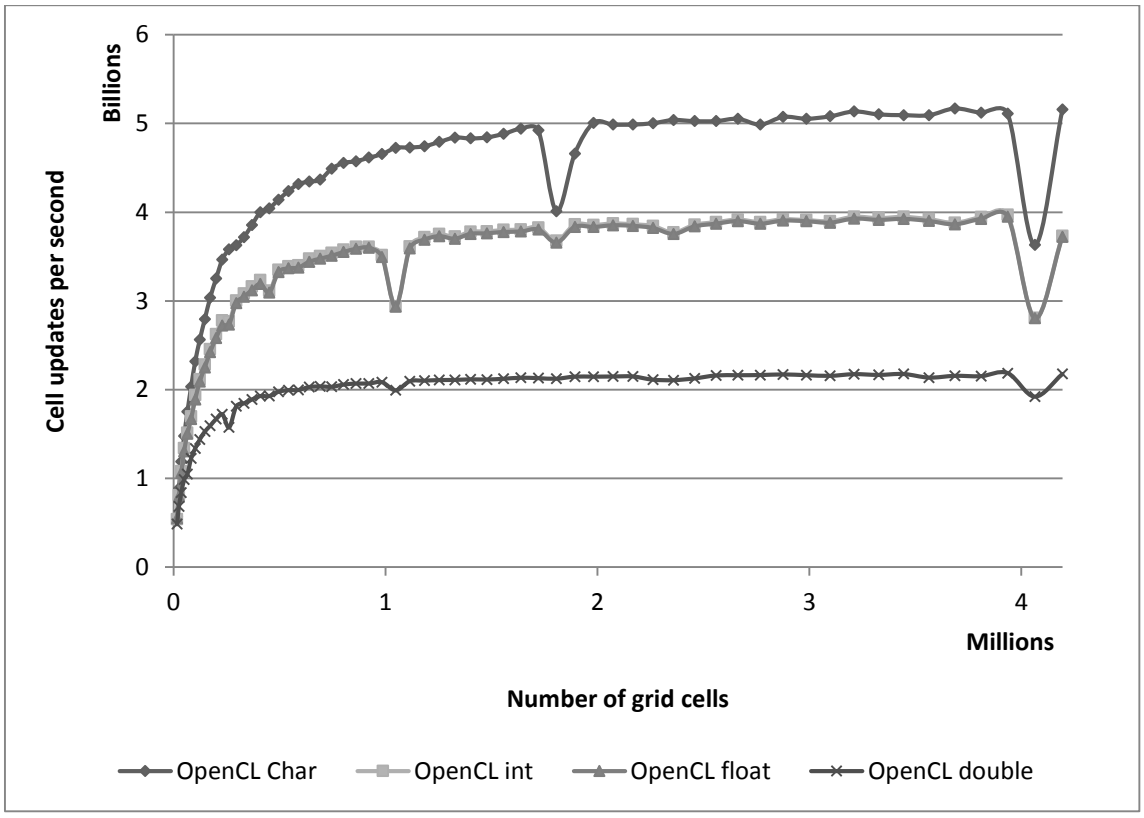


Figure 3.26, Cell update rates (per second) on the GPGPU at a range of different grid sizes, for 1,000 generations, using different base data types of char, int, float and double floating point numbers, on machine B.

Figure 3.27, Cell update rates (per second) on the serial CPU implementation at a range of different grid sizes, for 1,000 generations, using different base data types of char, int, float and double floating point numbers, on machine B.

Due to the different natures of the processing times shown in Figure 3.23 and Figure 3.24, the speed-ups factors shown in Figure 3.25 are notably different. The starkest difference is between the float and double types, as the CPU is tuned to perform well for the 64bit double floating point type, whereas the GPU performs roughly half as fast as it does with the float types. Notably the floating point type has the largest speed-up factor, even greater than the smaller data types of the char (which is also a simple type being an integer).

3.5.6 Neighbourhood size tests

A further parameter that varies among applications in cellular automata is the neighbourhood size. An in-depth investigation is conducted here into the effect of modifying neighbourhood size, in conjunction with activity levels to determine possible speed-up on a GPU.

3.5.6.1 Method

The Moore neighbourhood is extended and defined by the size of the radius (r), where the standard Moore neighbourhood has a radius of 1 as each (central,

main) cell is surrounded by 1 cell in either direction, forming a square neighbourhood where the number of cells is defined as $(2r+1)^2$. The 'game of life' decision tree rule set is used (it should be noted that the GOL rule set only uses assignment for its state changes, and therefore the only variable arithmetic is within the counting of live neighbouring cells); and the collection of neighbouring live cell counts is altered to a set of two loops which takes the radius parameter, and finally each neighbouring cell is counted as it is visited, as opposed to storing the entire neighbourhood which is more difficult for the GPGPU. Since the 'game of life' rule set looks for specifically 2 or 3 live cells in order to trigger activity, as the neighbourhood size is increased, the range of possible live neighbouring cells also increases; thus the chance of finding 2 or 3 live neighbours decreases when the initial configuration is seeded with the same 50% initial configuration distribution probability in the creation of live cells. However, it is found to be possible to generate long lasting patterns in all radius sizes tested from 1 to 5 for the decision tree game of life. It was consequently found that the initial configuration needed to be seeded with fewer live cells as the neighbourhood size was increased. So, similar activity tests as in section 5.2 were repeated for each neighbourhood size by using a separate implementation to ascertain the cell and neighbourhood counts over a range of initial configuration distribution probability for initial live cell creation.

It is determined that there are ranges of values within the initial configuration distribution probabilities which favour activity (shown in Figure 3.12 and Figure 3.28). Within these ranges the initial population levels are neither too few nor too many to generate widespread amounts of live cells both spatially and temporally, and as such are termed as 'habitable spectrum' of initial configuration distribution probabilities. Unfortunately it can be seen in Figure 3.28, that the 'habitable spectrum' for each radius of the extended Moore neighbourhood shifts dramatically towards the lower end of the initial configuration distribution probabilities, so much so that using a 50% initial configuration distribution probability with a radius greater than 2 would not likely yield high activity levels. Therefore a simple estimation of the centre of these 'habitable spectra' is utilised, which also coincides with using a 50% initial configuration distribution probability as before with the tests using a neighbourhood radius of 1 (as in sections 3.5.1-3.5.3). Equation 3.1 shows the initial configuration distribution probabilities

relative to the neighbourhood radius used in the preceding time experiments to ensure that high activity levels are generated for all neighbourhood radius sizes. Interestingly, the centres of these habitable spectra can be approximately calculated using the golden ratio of 1.618, a ubiquitous constant in natural systems. This leads to the initial configuration distribution probabilities as shown in Table 3.2.

Equation 3.1

$$Initial\ Configuration\ Distribution\ Probability = \frac{1}{radius^{1.618}+1}$$

Table 3.2, Estimations of biases for the first 5 neighbourhood radius sizes used in experiments in this section, which correspond roughly to the centre of the discovered habitable zones.

Radius	1/(radius$^{1.618}$+1)
1	50%
2	24.57%
3	14.46%
4	9.59%
5	6.89%

3.5.6.2 Experimental set up

Both the initial configuration distribution probability estimates produced by Equation 1 that yield high activity levels and a zero initial configuration distribution probability, which gives all dead cells in the initial configuration and thus the rest of the simulation, are tested. Again a workgroup size of 16x16, on a 512x512 grid is used, at 1,000 generations on Machine A, and 10,000 on Machine B.

3.5.6.3 Experimental Results

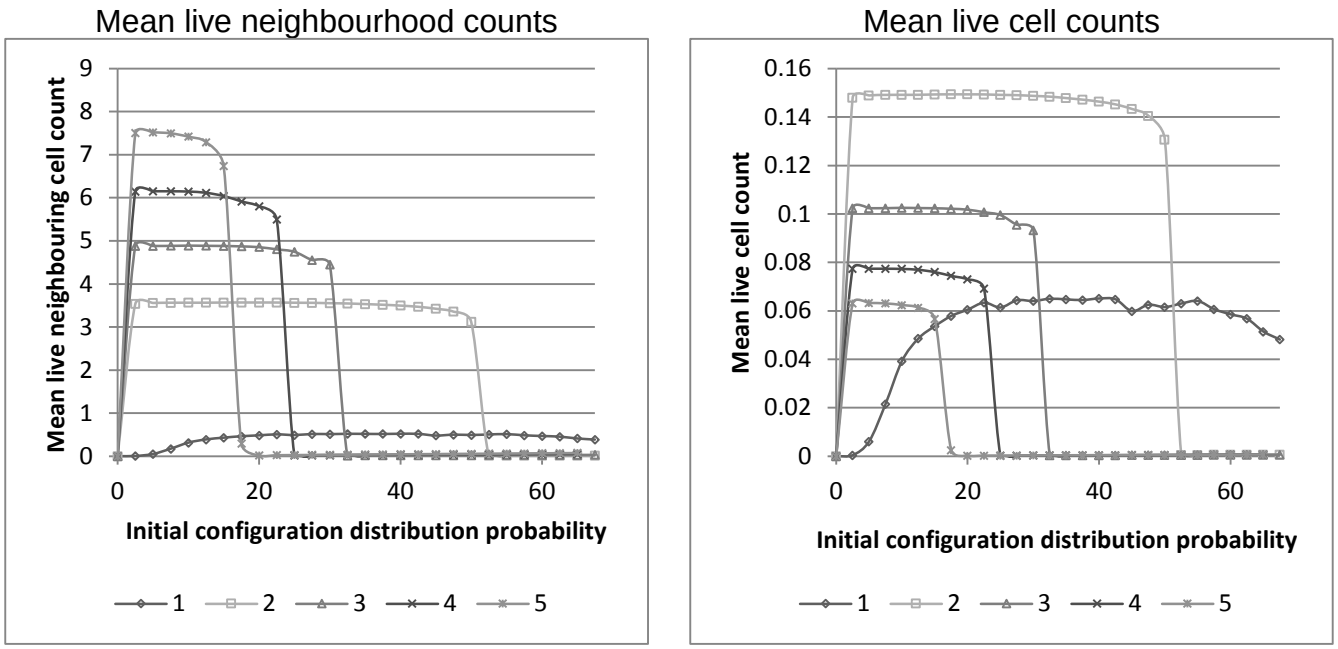


Figure 3.28, Average live neighbours and live cell counts for initial configuration distribution probability of 0% to 67.5% at intervals of 2.5%, for a 512 lattice size and 1,000 generations, for the neighbour radius sizes 1 to 5.

Figure 3.28 shows the average amount of activity and the described plateaux or habitable spectra. Figure 3.28 also shows that there is a reasonably large jump in neighbourhood count activity between neighbourhood radius sizes 1 and 2, but after that appears to follow a fairly linear increase in activity as the radius of the neighbourhood is increased. Interestingly the live cell counts follow a very different pattern.

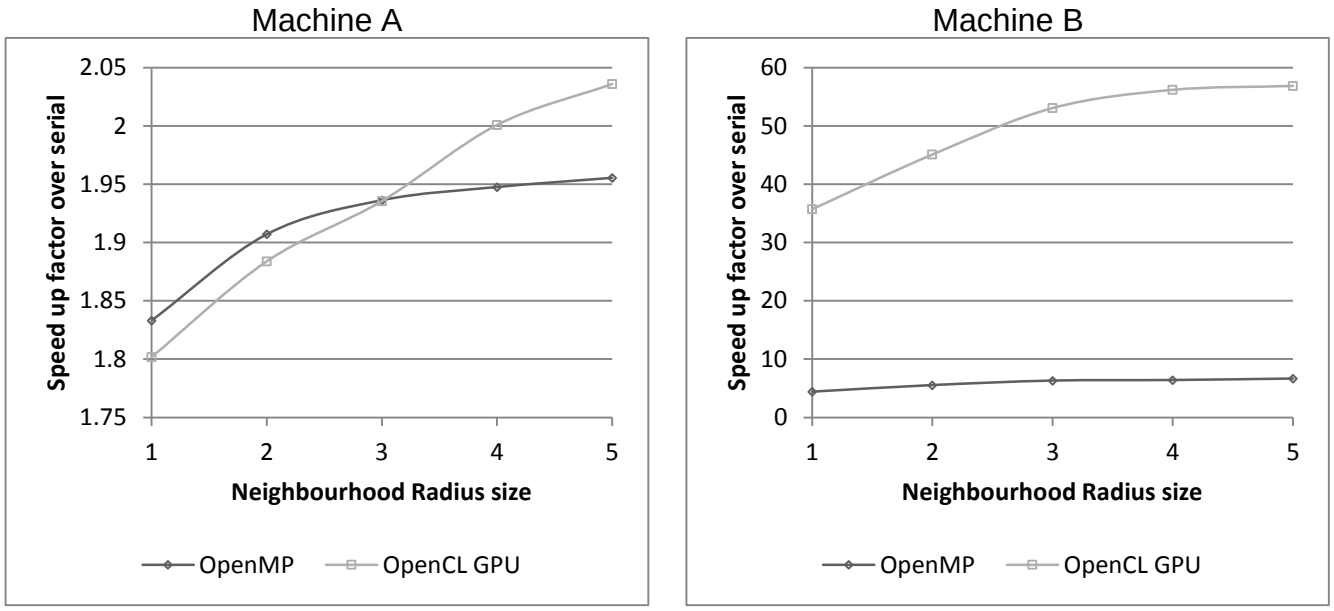


Figure 3.29, Relative speed improvements of the GPGPU and the OpenMP implementation, over the serial implementation with a variable neighbourhood size. Results shown for a 512x512 sized lattice, for 1,000 generations on Machine A, and 10,000 generations on machine B. Seeding with a zero initial configuration distribution probability and therefore no activity.

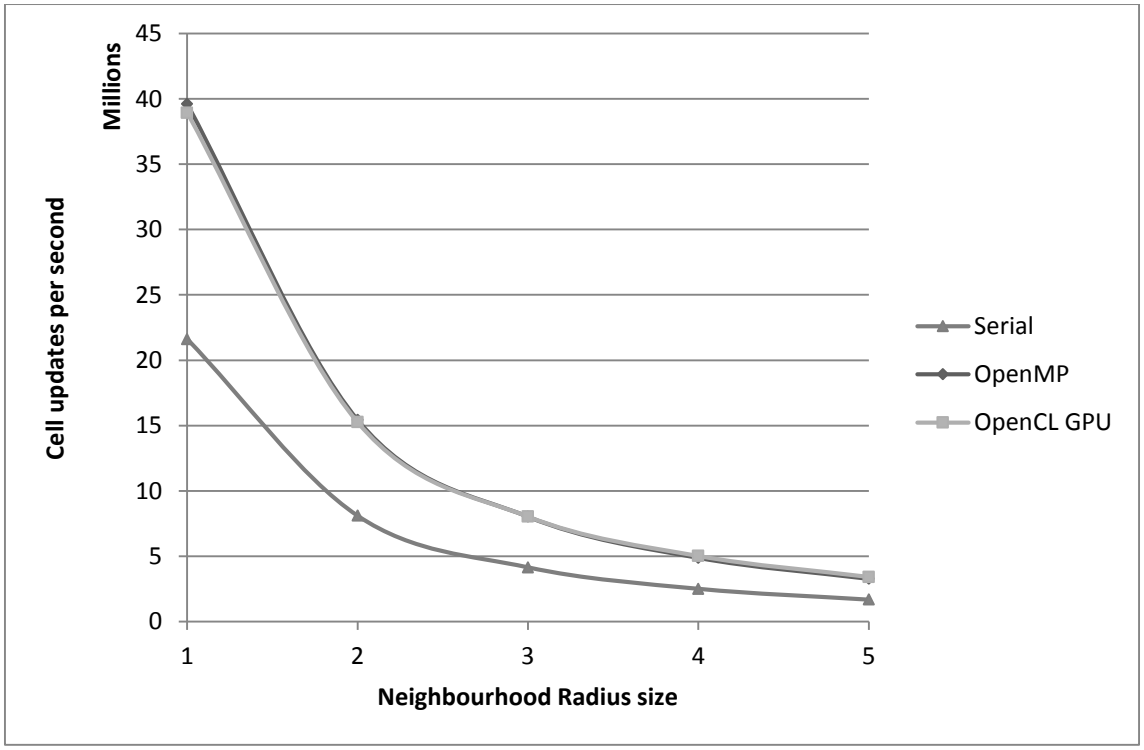


Figure 3.30, Cell update rates (per second) for the serial CPU implementation, GPGPU and OpenMP, with a variable neighbourhood size. Results shown for a 512x512 sized lattice, 1,000 generations on machine A. Seeding with a zero initial configuration distribution probability and therefore no activity.

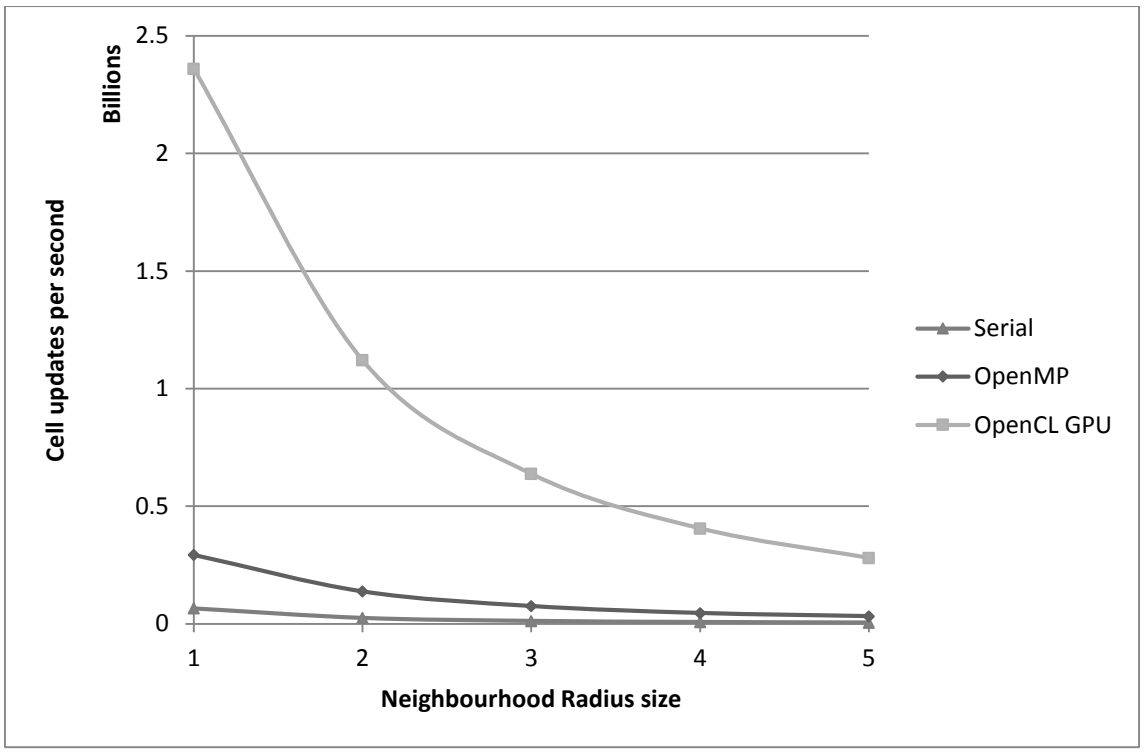


Figure 3.31, Cell update rates (per second) for the serial CPU implementation, GPGPU and OpenMP, with a variable neighbourhood size. Results shown for a 512x512 sized lattice, 10,000 generations on machine B. Seeding with a zero initial configuration distribution probability and therefore no activity.

Figure 3.29 shows that where there is no activity, only a slight increase in performance for larger neighbourhood sizes is observed, with the exception of the GPGPU on Machine B which shows larger increases. This is attributed to a more efficient use of the cache of the GPGPU in Machine B allowing for a greater use of the extra hardware parallelism with the additional memory work in each cell.

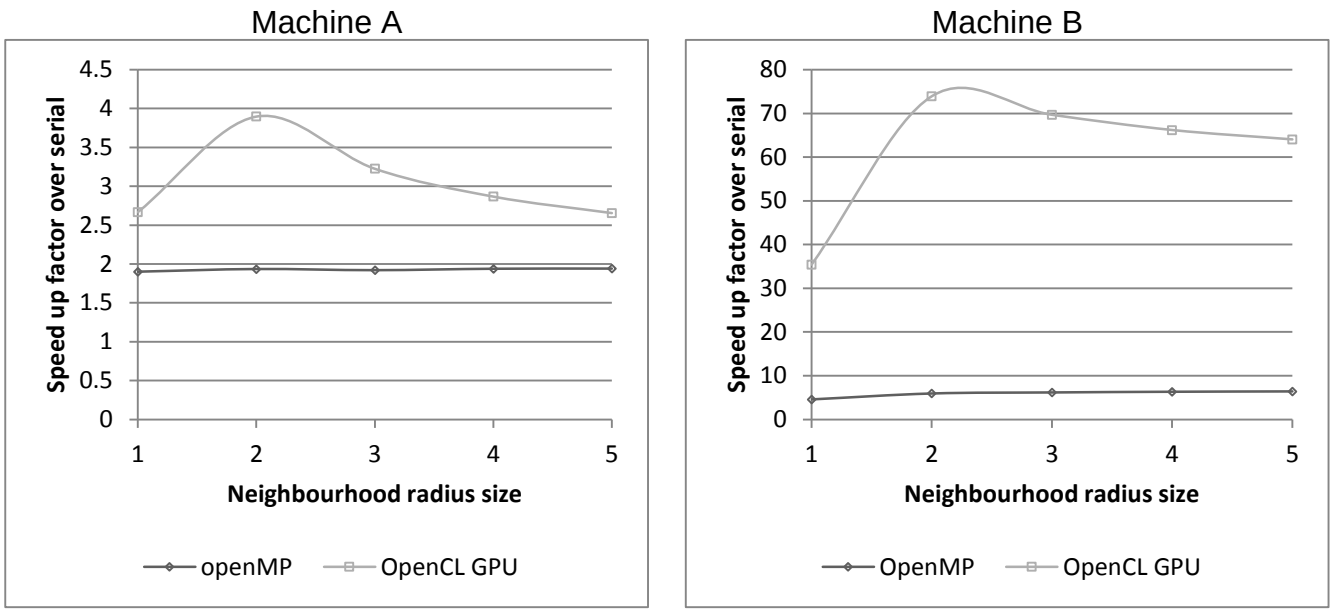


Figure 3.32, Relative speed improvements of the GPGPU and the OpenMP implementation, over the serial implementation with a variable neighbourhood size. Results shown for a 512x512 sized lattice, for 1,000 generations on machine A, and 10,000 generations on machine B. Seeding with the 'initial configuration distribution probability relative to the radius' as shown in Equation 1, to produce activity in all simulations.

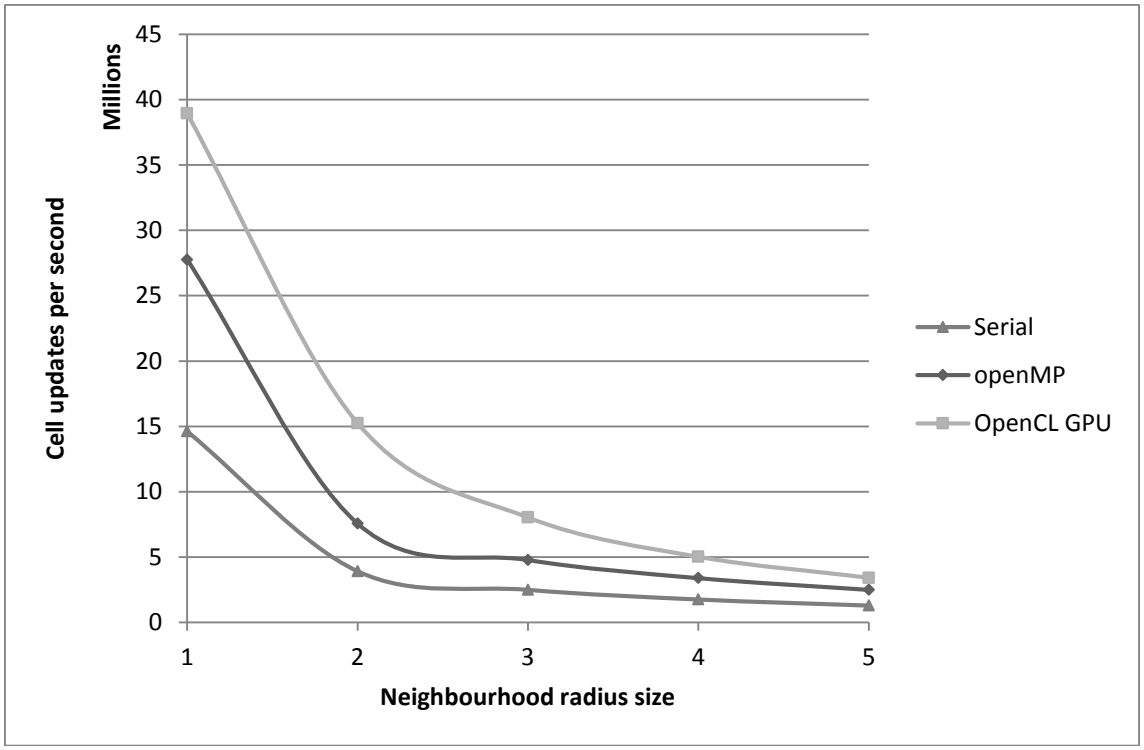


Figure 3.33, Cell update rates (per second) for Serial CPU implementation, GPGPU (OpenCL), and parallel CPU (OpenMP) with a variable neighbourhood size. Results shown for a 512x512 sized lattice, for 1,000 generations on machine A. Seeding with the 'initial configuration distribution probability relative to the radius' as shown in Equation 1, to produce activity in all simulations.

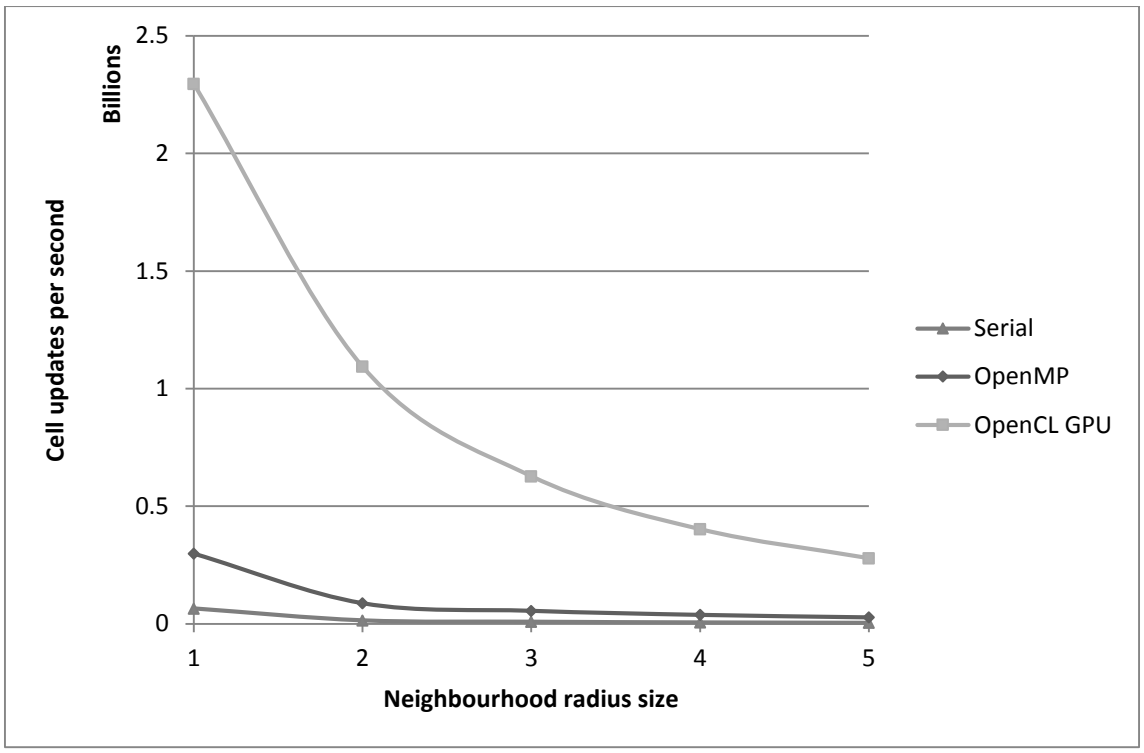


Figure 3.34, Cell update rates (per second) for Serial CPU implementation, GPGPU (OpenCL), and parallel CPU (OpenMP) with a variable neighbourhood size. Results shown for a 512x512 sized lattice, for 10,000 generations on machine B. Seeding with the 'initial configuration distribution probability relative to the radius' as shown in Equation 1, to produce activity in all simulations.

Figure 3.32 shows that where there is activity, a very different pattern in the speed ups of the GPGPU compared to the CPU (parallel) approach can be observed, whereby there is a spike in the performance at a radius of 2. It is proposed that not only is there a link between the amount of arithmetic and speed up, but there is there is also a relation between the proportions of arithmetic to memory accesses, shown in Equation 3.2. I.e. as the number of memory accesses is increased, caused by the larger neighbourhoods, this proportion is reduced along with the speed ups.

Equation 3.2

$$Speedup\ Factor \propto \frac{Arithmetic\ per\ cell}{Memory\ requests\ per\ cell}$$

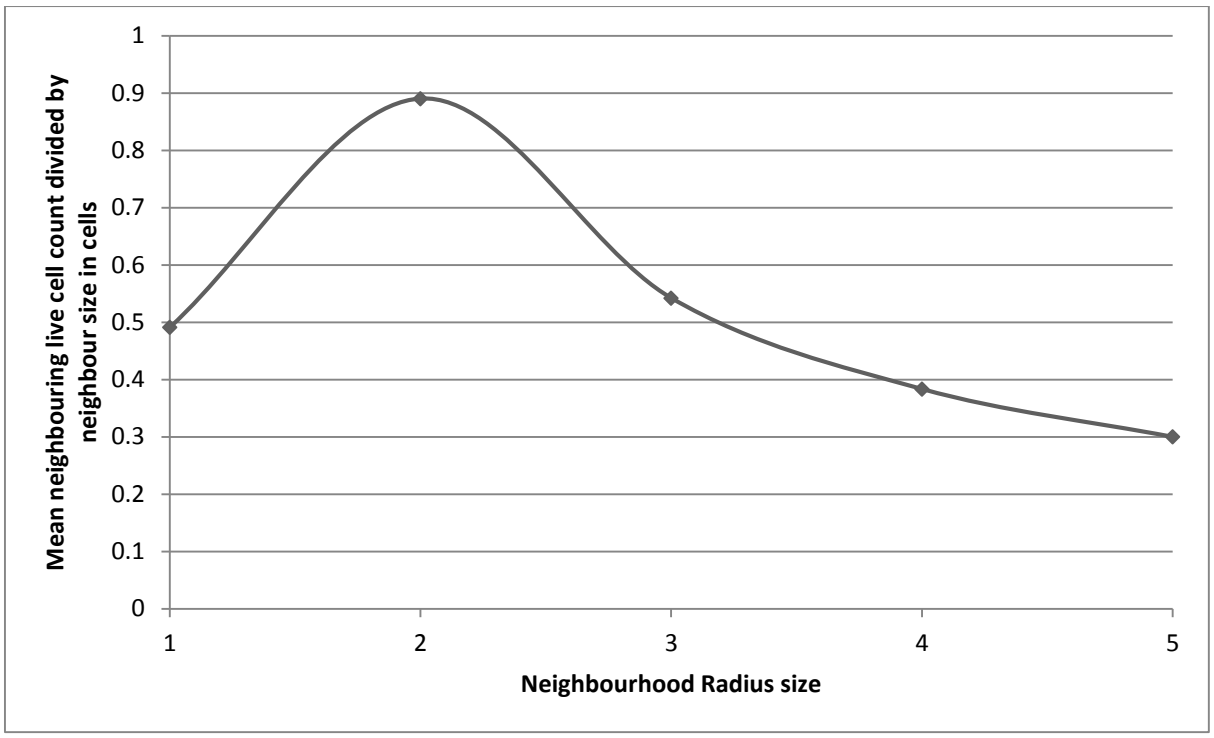


Figure 3.35, Ratio of the average live neighbouring cell count (activity) for each radius, against the neighbourhood size in cells, for a 512 sized lattice and 1,000 generations (I.e. the predicted speed-up level from Equation 2).

This relation (Equation 3.2) is shown in Figure 3.35, where the average live neighbouring cell counts from each radius divided by the number of cells in each neighbourhood are plotted. This measure of the level of activity within the neighbourhood relative to the neighbourhood size, for each neighbourhood radius clearly mimics the shape of the GPGPU speedup curves for both machines in Figure 3.32. For machine B, there is a slight trend to increase in performance with larger radius sizes, which is attributed to the increase in speedup seen in Figure 3.29 where the performance increases with neighbourhood size irrespective of activity level.

3.5.7 Generational size tests

Clearly, longer CA runs will benefit more greatly from any speedup that the GPU can provide. However, there are overheads associated with the implementation of a CA on the GPU and so these experiments attempt to characterise the length of run under which speedup on the GPU will be maximised. This is especially important for more complex CA to understand how different numbers of CA iterations will affect the GPU speed ups.

3.5.7.1 Method

In order to avoid excessive transfer to and from the GPGPU when testing over a range of CA generations, for each number of generations tested, a full simulation is repeated up to the required number of generations, as opposed to running a single long simulation and timing it at sample intervals. Earlier sections (3.5.3-3.5.4) have shown how activity affects the entire simulation; experiments are now conducted to see how this effect correlates with the number of CA generations by again counting the live average cell and neighbourhood activity as well as a separate implementation purely for timing results.

3.5.7.2 Experimental set up

In a similar fashion to the lattice size tests in section 3.5.1, and 3.5.2, tests were run on a spectrum of generation sizes and fully independent runs (repeated 15 times for an average) were conducted for each total count of generations. Tests were run at a single generation, and then at increments of 100 on Machine A up to 1,000, and at increments of 1,000 on Machine B up to 10,000. These tests are performed at lattice sizes of 512x512, 1024x1024, and 2048x2048, which are noted to be the particular lattice sizes where Machine A suffers from load balancing issues on the GPGPU, with a workgroup size of 16x16. Tests are also repeated on Machine B, at smaller lattice sizes of 480 and 448, in order to confirm the theory for the difference in the Machine B's response at a 512 lattice size. Machine B's GPGPU has more cores and therefore processes each generation quicker than Machine A. However, the overheads of parallelisation do not change as greatly for Machine B and this therefore means it takes more generations to overcome them at these smaller sized lattices.

3.5.7.3 Experimental Results

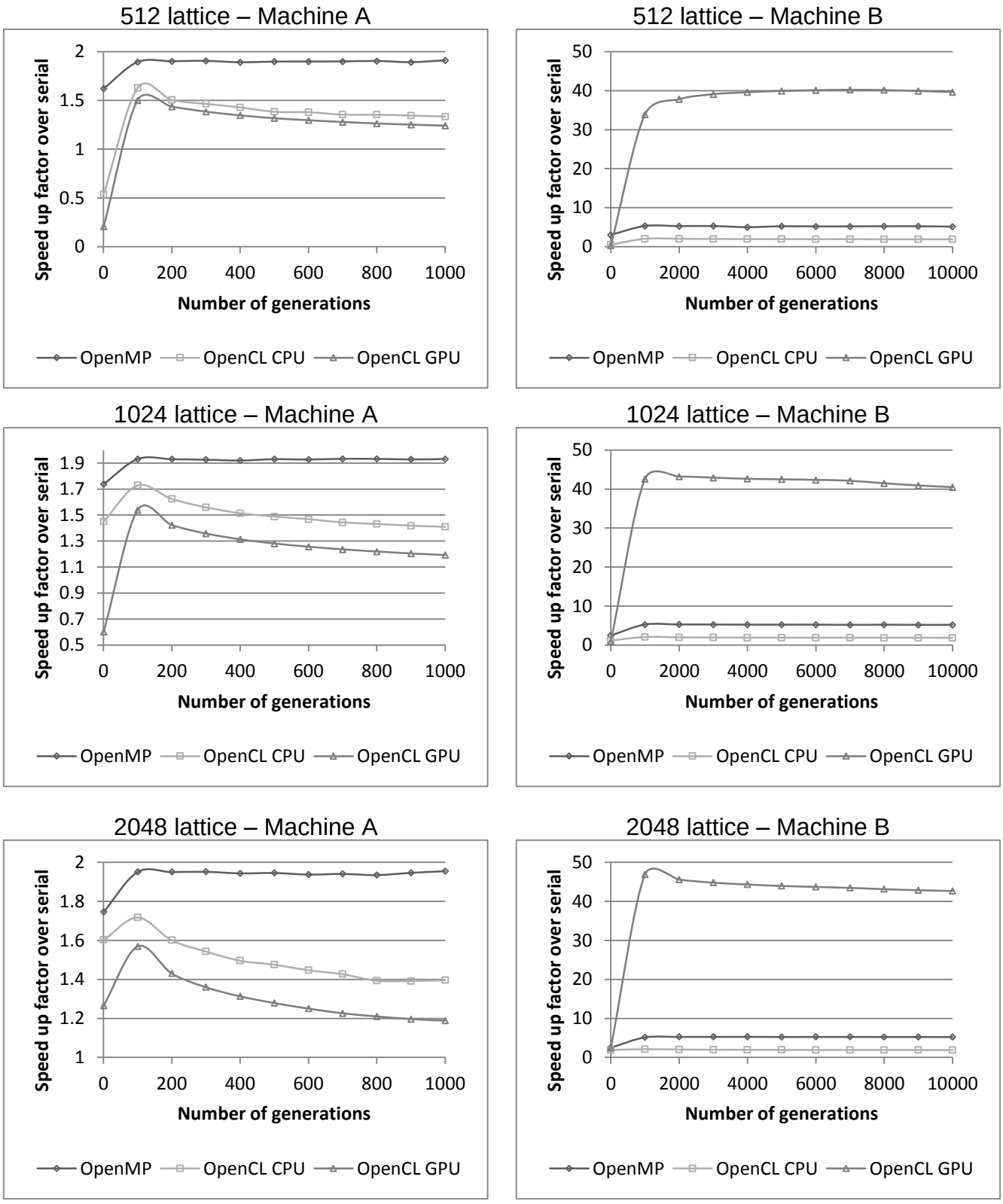


Figure 3.36, Speed ups over the serial implementation, for OpenMP and OpenCL on the CPU and GPU, for a spread of generations, at lattice sizes of 512x512, 1024x1024, and 2048x2048.

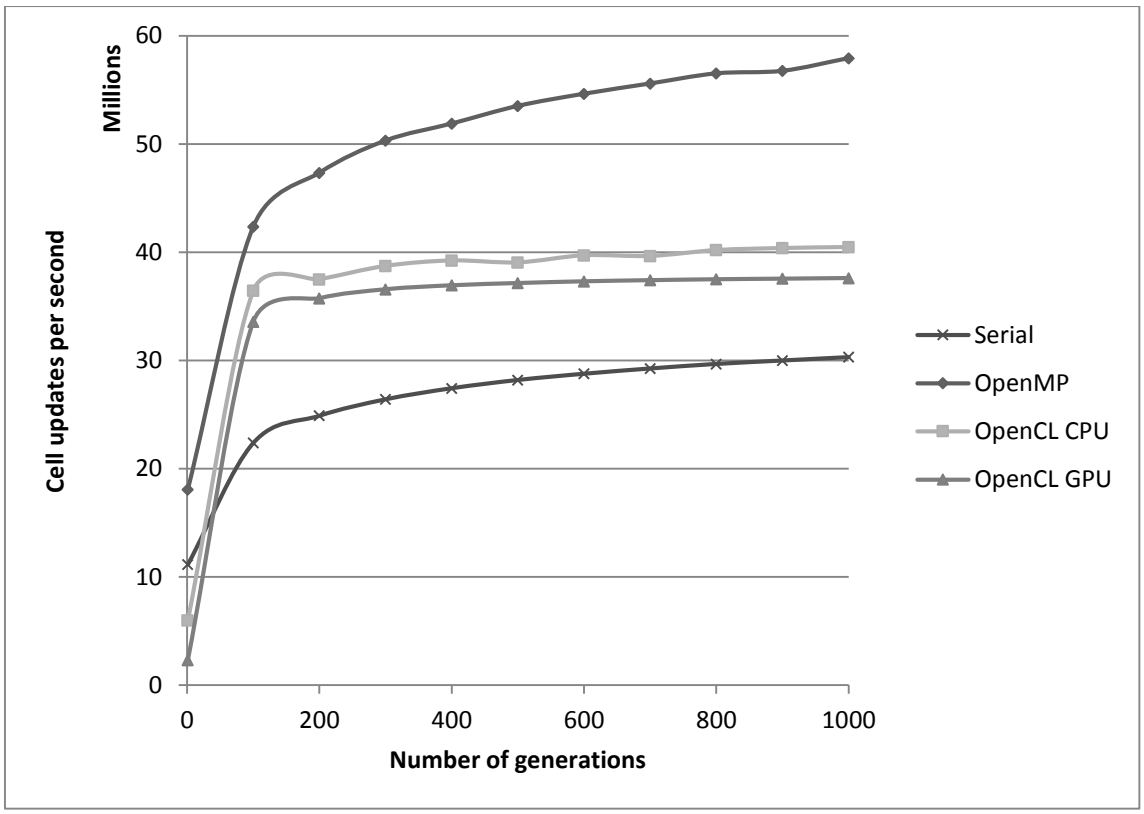


Figure 3.37, Cell update rates (per second) for serial implementation, OpenMP, and OpenCL on the CPU and GPU, for a spread of generations, at lattice sizes of 512x512, on Machine A.

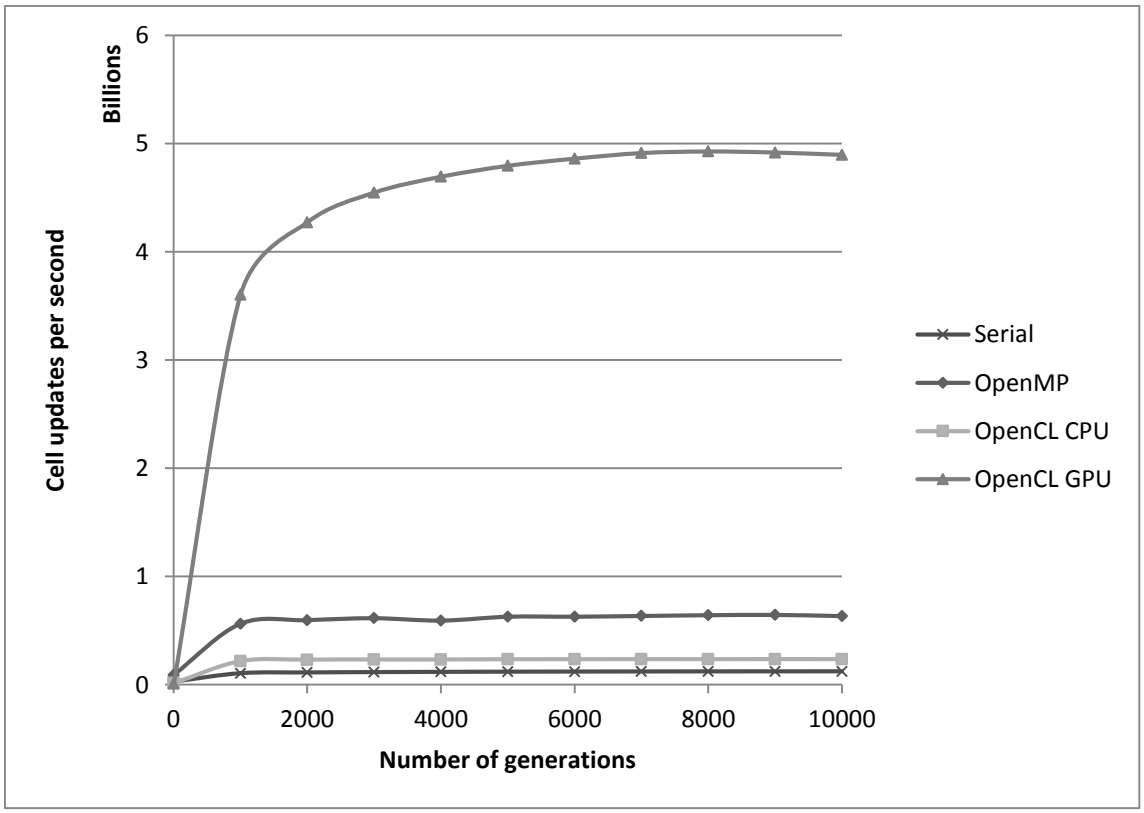


Figure 3.38, Cell update rates (per second) for serial implementation, OpenMP, and OpenCL on the CPU and GPU, for a spread of generations, at lattice sizes of 512x512, on Machine B.

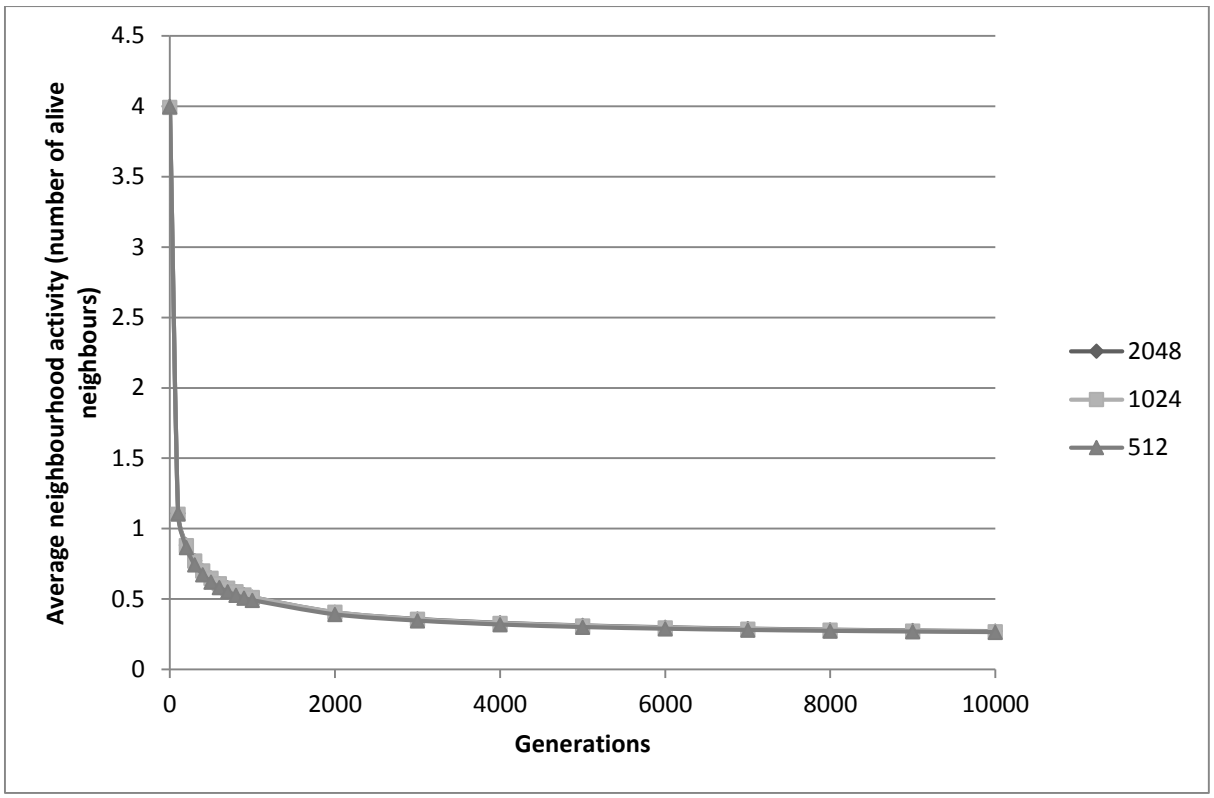


Figure 3.39, Average neighbourhood counts (Mean number of live neighbouring cells), for a spread of generations, at lattice sizes of 512x512, 1024x1024, and 2048x2048.

It was found that once again, the timing results indicate a fairly linear relation between the number of generations and the processing time. However overheads of parallelisation which, on the GPGPU, include the transfer time to and from the device and small amount of compilation time for the kernel, mean that a curvature up to a plateau can be observed in the relative performance shown in Figure 3.36. In many of the relative performances of the parallel approaches, as the number of generations increases the parallelisation overheads are overcome as a larger proportion of the whole algorithm is parallelised. However, it appears that, if the number of generations is increased further, a slower drop in relative performance of the parallel approaches can be observed. Sections 3.5.3-3.5.4 have shown that over the course of the entire simulation the activity is directly proportional to the relative performance of the parallel approaches and can be partially attributed to the slower drop in activity, in the later stages of the simulations (Shown in Figure 3.39). However experimental results appear to show a greater drop at the higher end of the tested spectrum of generations, especially on the older/slower machine A, which may be due to the continued bottleneck of load balancing, which occurs at that grid size. After a single generation the highest amount of activity are observed in Figure 3.36, and in Figure 3.39 the lowest relative

performance (even a marginal decrease in some cases) is observed. This demonstrates the extent to which simulations with very few generations are dominated by these parallelisation overheads, specifically for the GPGPU the transfer time bottleneck. Figure 3.36, for Machine B at a 512 lattice size shows that it takes more generations to overcome the initial bottleneck. This pattern is due to the use of relatively small lattice in relation to the level of hardware parallelism and the transfer bottleneck; i.e. as Machine B has a greater level of hardware parallelism compared to Machine A, it can process each generation of the smaller lattice relatively faster compared to the transfer bottleneck, and therefore takes more generations to overcome this effect.

3.6 Discussion

The experimentation here has shown that in order to gain the greatest performance from the GPGPU the lattice size, or rather the number of threads used, must exceed the number of hardware cores by more than an order of magnitude in order to best utilise all layers of hardware parallelism. Particularly difficult lattice sizes are shown to be caused by the lack of fit between the numbers of workgroup threads and the number of hardware's independent sub-groups of processors. Therefore, caution is suggested when using particular lattice sizes, and claiming acceleration rates for the GPGPU. It is also shown that these load balancing effects are caused by the number of workgroups, and one solution to this problem might be to pad the number of threads and therefore workgroups. An advantage to this approach is that these extra threads need only check if they are within the actual lattice or if the padded threads area, and thus need not transfer additional data.

It is noted that part of these findings are due to many of the intricacies of way the implementation has been formulated, even though only a brute force approach is investigated. For example, a programmatic function/decision tree is used, which doesn't require a look-up. Also, apart from the final tests with neighbourhood sizes, unrolled loops have been used for the collection of neighbouring cell values. Also, in the 'game of life' with two states, it would be possible to implement this by means of summing the values of all neighbouring cells in order to gain a count of living cells, although this would be sub optimal for serial implementations, it would yield a very high speed up factor. The

experimentation shows how important variable amounts of arithmetic are to the speed improvement shown by the GPGPU, and how this causes variations in GPGPU accelerations over the CPU. It is demonstrated through the implementation of state transition rules with variable amounts of arithmetic, that where the conditions of the CA simulation result in large variations in the arithmetic levels across the grid then the GPGPU's performance varies proportionately. This may go some way to explain the wide variation of reported speed increases from the literature. All CA will need to follow a similar memory pattern, as all CA need each cell to access the neighbouring cells states; therefore, any variation in speed-up between different CA will come from both the different neighbourhood size, in addition to the amount of certain arithmetic and variable arithmetic within the given state-transition rule of the CA. It is shown for the investigated rule sets that the majority of processing time can be attributed to the number of memory look-ups for each cell however the variation in the arithmetic work of rule set can still cause relatively large difference in speed-up. Therefore, for a given size of neighbourhood used (the amount of memory requests per cell), one should if possible increase the amount of arithmetic within the rule set on a per cell basis, in order to gain greatest speed-ups. Conversely if the rule set uses minimal arithmetic compared to memory access, it is expected that speed-ups will be smaller.

Multi-state interpretations of the game of life have been created and experimental results are shown. The variation in the number of live cells over the course of the simulation, and the average specific course through the decision tree are shown to affect the processing speed of the CPU implementation due mainly to the varied amount of arithmetic necessary. Whereas the GPGPU may hide this variation in arithmetic complexity behind the memory latency and within the hardware parallelism, and as a result produce much more predictable processing times in the presence of such variation. The variation is a consequence of the behaviour produced, and therefore very hard, if not impossible, to predict without prior knowledge of the given rule set. Activity levels effect the processing time by different amounts at each generation and, with the game of life rule set, the activity was found the be greatest in the early stages; after an initially large drop tends to a slow decrease. In these early generations/short simulations the bottleneck of transferring to and back from the

GPGPU for processing is found to by far outweigh the acceleration in processing. This suggests that this bottleneck is especially critical to short lived simulation, which would certainly have consequences for distributed/cluster systems; i.e. it is even more important to run long CA simulations (large number of generations) than it is to use larger lattices, in order to overcome the overheads of parallelisation upon co-processors such as the GPGPU.

The cache in the newer Fermi generation of GPGPUs facilitates better use of larger neighbourhoods. This is because the additional hardware parallelism makes good automatic uses of this fast memory. A relation between the proportions of arithmetic activity to the amount of memory re-use is proposed (shown in Equation 3.2), again suggesting the need for a fine balancing act between the amount of arithmetic and memory complexity within a CA system for best performance increases. The decision tree interpretation of the game of life rule set when applied to larger neighbourhoods is found to produce very interesting patterns: - 'habitable spectra' within the initial configuration distribution probabilities exist for the incitement of such patterns. A relation between the radius of the neighbourhood and these habitable spectra are proposed which appears to follow an interesting mathematical pattern found from nature, i.e. the golden ratio.

3.7 Conclusions

As a lattice-based, parallel method of computation, cellular automata lend themselves to parallelisation on GPUs very well. This work has thoroughly investigated the performance increases that can be expected from this parallelisation for a wide range of expected cellular automata parameters. The results have provided some expected results; that CA run for longer generations provide increasing speed-up and that the machine type (and in particular relative speeds of GPU and CPU) have a large bearing on the level of speed-up possible on these machines. The results have also provided some less obvious insights into GPU parallelisation. Firstly, that the maximum speed-ups are found when maximising the arithmetic use of the GPU, while minimising the amount of memory look-ups per cell of the CA also increases the speed-up factor. Secondly, that the amount of activity in the CA has a large effect on performance. With a high dependency on the specific implementation, CAs with more cells that carry

out lengthy processing during their evolution are more amenable to parallelisation than those which have low cellular computational complexity and are therefore able to exploit the parallel GPU more effectively. Thirdly, that the choice of lattice size is important in the speed-ups possible on the GPU and care should be taken to ensure that the lattice size fits with the underlying hardware where possible. Fourthly, that there exists a complex relationship between the number of states, neighbourhood size, state-transition rules and the level of activity (and therefore effective parallelism) within a CA. This relationship will ultimately determine the specific level of speed-up available to a CA implementation and is, for obvious reasons, problem specific. However, using the results from this study the likely speedup for a CA implementation on a GPU can be estimated based on the lattice size, activity levels observed within the CA and the number of generations required. This estimation can aid decision making when considering whether the degree of speedup is sufficient to warrant a GPGPU implementation on an application by application basis.

Finally, the work above has shown that for a simple CA such as the Game of Life, speed-ups of between 50 and 100 times are possible on modern hardware. It should be noted that this is likely to be a conservative estimate as this figure is a comparison with one of the fastest modern CPUs available and the Game of Life is comparatively simple. More complex rule sets will make better use of the GPUs native capability for performing fast parallel calculations. The best speed-ups will occur when the CA is sufficiently complex and run for a large number of generations, which is beneficial to the field in that the most complex CA will benefit the most from parallelisation.

This all leads to the idea that CA when given enough lattice size and enough iterations, relative the given GPU hardware can be accelerated in processing. This brings into real terms the possibility of using CA simulations as fitness functions for a Genetic Programming system. This is where the parallelism of the CA and GP system can combine well together, as it appears from the research contained within this chapter, that to gain the best speed-up the CA needs to be at least an order of magnitude larger than the number of core within the GPU device. Since GP will require the evaluation of a fitness function for a number of individuals, this can be used to enlarge the total number of cell processed without enlarging the individual test cases, which would still scale with the total number

of cells. Such a method is described in greater detail in later chapters of this thesis, as this is implemented in order to train a CAGP system for a real-world problem in urban flood modelling.

Chapter 4: GP learning of Cellular Automata state transition rules

4.1 Introduction

4.1.1 Background

GP was introduced previously in section 2.2 as a method of optimisation, symbolic regression and rule learning, and section 2.2.3 shows how GP has been used previously to derive the state transition rules for relatively simple binary state, and 1 dimensional CA. The fitness of the GP state transition rules, are in these types of problems normally determined by the final state of number of CA simulations.

The work in this chapter introduces the combined GPCA methodology for learning the state transition rules of 2D CA using GP, and investigates the feasibility of training CA state transition rules where the desired output is that of an entire CA simulation. In such a process, not only would the final state of the given fitness simulation be tested, but also all the intermediary states. Instead of evolving a distributed computer program, a distributed simulator is evolved, as the method the CA uses to get the final results is as important as the final resulting state.

The GPCA method may be applied to any CA simulation application so long as an interface between the local neighbourhood of the CA and the GP can be produced. This interface must declare the basic type of CA (binary, integer, continuous)/number of states, and how the states of the CA neighbourhood relate to GP variables. The work in this chapter demonstrates how the GPCA system may work on such a simple binary rule set as the Game of Life.

4.1.2 Chapter Structure

Section 4.2 introduces the general methodology of the GPCA system for learning CA state transition rules, and introduces the interface system used to learn the Game of Life rule set. This section also describes the GP fitness function and evolutionary algorithm applied in the rest of this thesis.

Finally section 4.2.4 introduces the method of combining the modern many-core parallelism of the GPGPU, to not just increase the processing time of CA simulations, but also to increase the speed of GPCA training. This novel extension of simply processing each CA fitness case on GPGPU hardware, allows for all the fitness cases of the single GP generation to be processed simultaneously on the GPGPU. By processing all the fitness cases in a single batch, the bottleneck between the CPU and GPGPU can be minimised, and make full use of the combined work of all the fitness cases to satisfy the GPGPU's needs for sufficient workloads in order to gain speed-ups through parallelism. In fact the experiments in Chapter 5:, on fixed spatial and temporal resolution real-world flooding experiments with the GPCA system would take 3.75 weeks when run in parallel on the CPU, and only take 125 hours on the GPGPU. If these experiments were to be carried out in serial on the same CPU, it would take approximately 18.75 weeks to carry out just the experimentation for Chapter 5:. Therefore, it is only the use of this novel method of combining the parallelism of the GP and CA algorithms which allows for the feasibility of the range of experiments carried out in this thesis.

This system is first applied to the simple rule set of the Game of Life in section 4.3, which shows that Genetic Programming is capable of discovering this simple rule set. This experimentation is intended to demonstrate the ability of GP to search the space of possible rulesets and to locate the global optimum 'correct' solution. The advantage of using a system such as the GOL is that it has an identifiable global optimum which is well known, whereas the real-world flooding applications do not. It can therefore be used to modify the GP system to be as optimal as possible before tackling the much larger real-world problem. Having shown that the GPCA a system can learn such a simple rule set as the Game of Life (section 4.3), this methodology is then extended to the development of a real world urban flood modelling rule set in Chapter 5:. Finally chapter 4.4, draws conclusions upon the feasibility of using the GPCA system to learn such a simple CA rule set as the Game of Life.

4.2 Methodology

In this section a methodology is developed that uses GP to evolve specific CA state transition rules, using example CA simulations to learn from. The standard Koza [6] type of GP is employed, in that a GP tree system and sub-tree cross-over are employed. The GP tree is applied as a decision tree within each cell of the cell of the cellular automaton to determine the new state for each cell.

4.2.1 GP CA interface/representation

The method employed to interface the GP within each CA neighbourhood differs for each type of CA, and the type of rule which is being learnt. There are two main methods employed, the first for the Game of Life type of binary state rule sets (described in section 4.2.1.1), and the second for use modelling real world hydraulic movements of water (described later in section 5.2.1). The interface, dictates on which state variables the GP will operate and therefore the variable terminal values for the GP tree. Since the prospective formulae expected for each system are so different a different set of operates are used in each experiment, detail in the experimental setups for each.

4.2.1.1 Game Of Life binary state GP interface

The Game of Life style GP implementation uses the Moore neighbourhood (shown in Figure 2.2), and the rule set that was previously described in section 2.1.1, in which the states of cells are either '*alive*' (state 1) or '*dead*' (state 0). The number of alive neighbouring cells is counted for each cell in the CA grid, and this is provided to the single GP tree within each cell as one of the variables inputs (This variable is called: `NH_count`). The only other variable input to the system is the state of the central cell of each neighbourhood (This variable is called `mainCell`). Finally, two static variables are introduced for each of the binary states `ALIVE`, and `DEAD` to represent states 1 and 0 respectively.

This system guarantees the uniformity to the direction of input for all possible rule sets, by only presenting a single cumulative input from all the neighbouring cells to a single GP tree within each cell. In the experiments in this chapter on learning the Game of Life rule set, both the GP tree (at each node) and the states of the cells are allowed the full variable expression of floating point

values (double precision). Therefore, it is up to the GP trees to develop rules which only output binary states. However, this means that the counting system which determines the number of alive neighbours (`NH_count`), must deal with floating point values. In this case, the neighbourhood count variables (`NH_count`), counts any cell states with a state equal to or greater than one as alive.

4.2.2 GP CA Fitness function

Each resulting CA simulation for each GP rule is tested for similarity to the given target CA simulation, which then returns an error score for the given GP individual. The error is calculated between each cell at each iteration and the target simulation and thus produces a similarity for the entire simulation. The inverse of the error score is used as the fitness score (where fitness must be maximised), as this is used for fitness proportionate roulette wheel section. Effectively the system is asked to solve the inverse problem for CA, whereby given a global reaction, the system must deduce the local state transition rule which creates this global reaction. Evolutionary algorithms, of which GP is a member, are characterised by large numbers of fitness evaluations which can be very computationally expensive. However, they are known to be capable of searching otherwise intractably large search spaces and finding near optimal solutions. Therefore, the evolutionary system should be able to create local rules which are reasonably close approximations to the global maximum. Since such a rule will have to cope with many hundreds or thousands of different sets of parameter/variable inputs, and the complex interactions required to closely match the global space-time pattern, these local rules produced should generalise well to other conditions and initial inputs. As the same local rule in different configurations which is responsible for the different global reaction observed. For example in the Game of Life experiments in Section 4.3, using a 100x100 grid, and 10 generations of CA simulation as the fitness function, presents each GP with 100,000 different input and output sets. Since a CA will need to be evaluated with each GP individual, and a comparison made against the target CA, then the target CA details are loaded once at the start of the optimisation process, to avoid the loading bottleneck during the process.

In the real-world rule set experiments in Chapter 5: and Chapter 6:, it is important to consider the real-scaled nature of the underlying model (i.e. real

world). This must be represented in a computer system as discrete points in time, known as the sampling rate, regardless of whether this data comes from another 'trusted model' or from the real world. The target simulation is represented in memory by holding a grid of values for every second of the simulation (sampling rate used throughout this thesis), even though the model used to produce it run at a much finer grain (smaller time step). This single data set is used to represent the continuous movements fluid masses around the grid, so when a CA iteration falls directly on a given second it will use this grid for comparison. When a CA iteration falls between two given grids (seconds), then linear interpolation is utilised between these two grids, on a cell by cell basis. This allows for the same space time pattern to be targeted at different temporal resolutions.

4.2.3 GP CA Evolutionary Algorithm

A generational evolutionary algorithm is used to drive the GP system, shown in Figure 4.1, where all the individuals within the current population are evaluated and then order/sorted according to their fitness scores. Then a top percentage of individuals are copied without alteration to the new population (known as the elitism rate), and so long as at least one elitist individual is copied to the new population at each generation the algorithm's current best solution will not get any worse. The rest of the new population is made from individuals which are selected from the entire current population, using the given selection strategy (e.g. fitness proportionate roulette wheel, or tournament selection). Selected individuals are given a certain chance of being directly passed into the new population without cross-over (cross over rate), although there is still a certain chance of mutation (mutation rate). If an individual is assigned for cross-over, then another individual is selected (again using the given strategy) and used as the second parent. A cross-over of two parent GP trees, will produce two child GP trees, but only a single arbitrary child is created, and given a chance of mutation before being place in the new population.

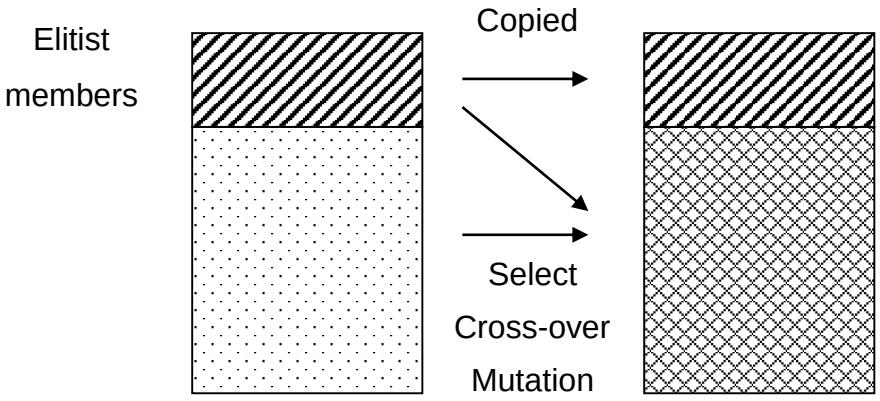


Figure 4.1, Illustration of how a new population is derived from the current population within the generational GP system.

The entire system is represented by the flow chart (shown in Figure 4.2). Targets are provided by UIM (the Urban Inundation Model) [62] for the real world experiments, which uses an adaptive time step, where the minimum time step is much lower than those utilized within training of the GP system, in order to provide a finer grain of detail.

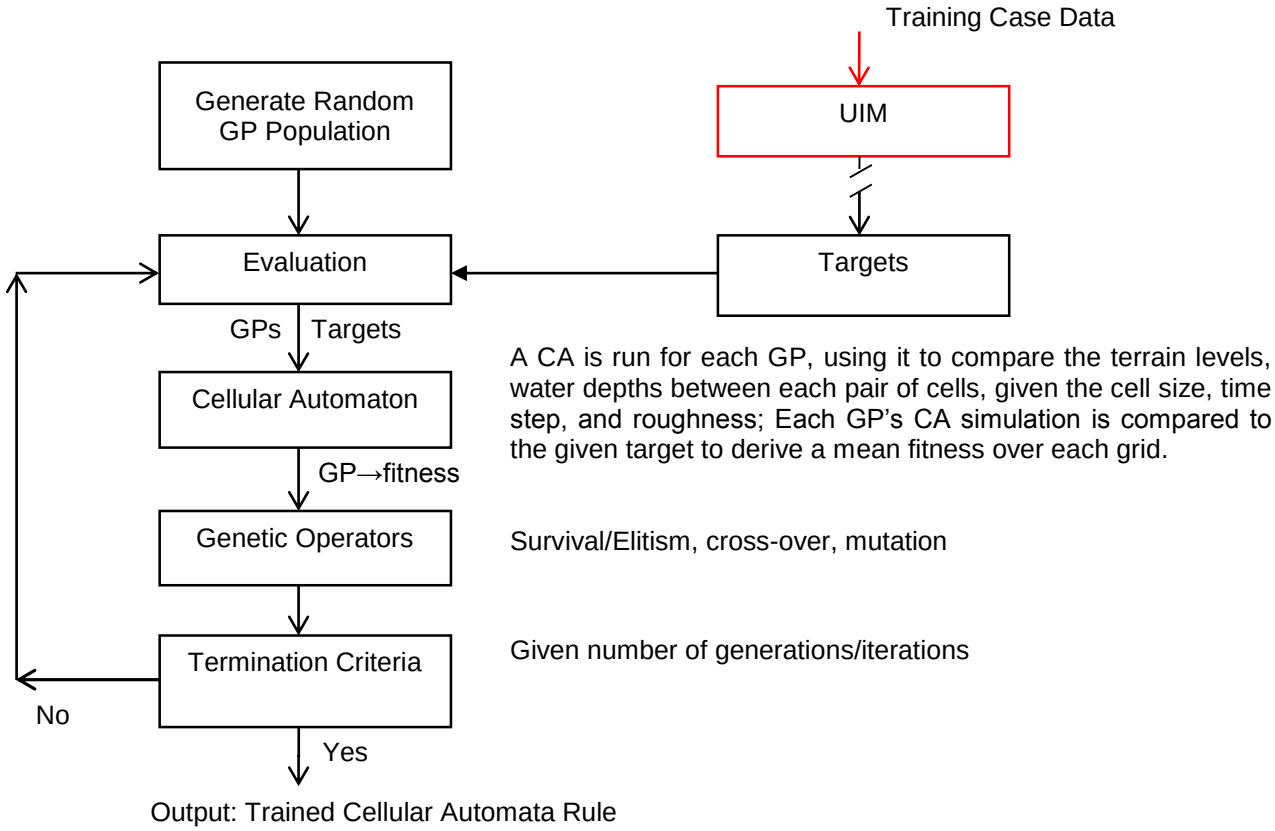


Figure 4.2, Flow chart of the GPCA optimisation systems process.

4.2.4 GP CA GPU computing method

4.2.4.1 Novel GP CA method for combined parallelism for more efficient GPU computing

In order to expand the number of parallel elements (cells) being processed, without expanding the size of each individual test cases, a novel system was developed which harnesses the parallel nature of both the GP and the CA algorithms. Parallelism may be drawn from both the CA with its many cells performing the state transition rules for each cell in parallel, but also this process is repeated for each GP individual, which may also be performed in parallel. Previous experiments have shown that there is a minimum threshold of the number of cells required in order gaining sufficient parallelism, and the size of the CA evaluated for the single GP individual maybe much smaller than this, otherwise the whole processing time is multiplied. Therefore, the evaluation of the CA for each GP individual is carried in out in a closely parallel fashion. In that one large CA grid is created, which includes a CA grid for each GP individual, where particular attention is made in the kernel code to avoid any interaction between these grids. Therefore a much larger number of cells per CA generation can be carried out while evaluating all the GP individuals in parallel (Shown in Figure 4.3). Since multiple populations may also be required to be run, it is possible to use a third layer of parallelism by performing the evaluation of every GP individual in each population in parallel. This method allows for the usage of a smaller terrain in terms of the number of cells, to be used as the target CA for each GP individual, while still overall maintaining a high enough parallel elements (cells) in a single process in order to satisfy the conditional number of parallel elements found in Chapter 3:, to be required to take full advantage of the parallel power of the GPGPU.

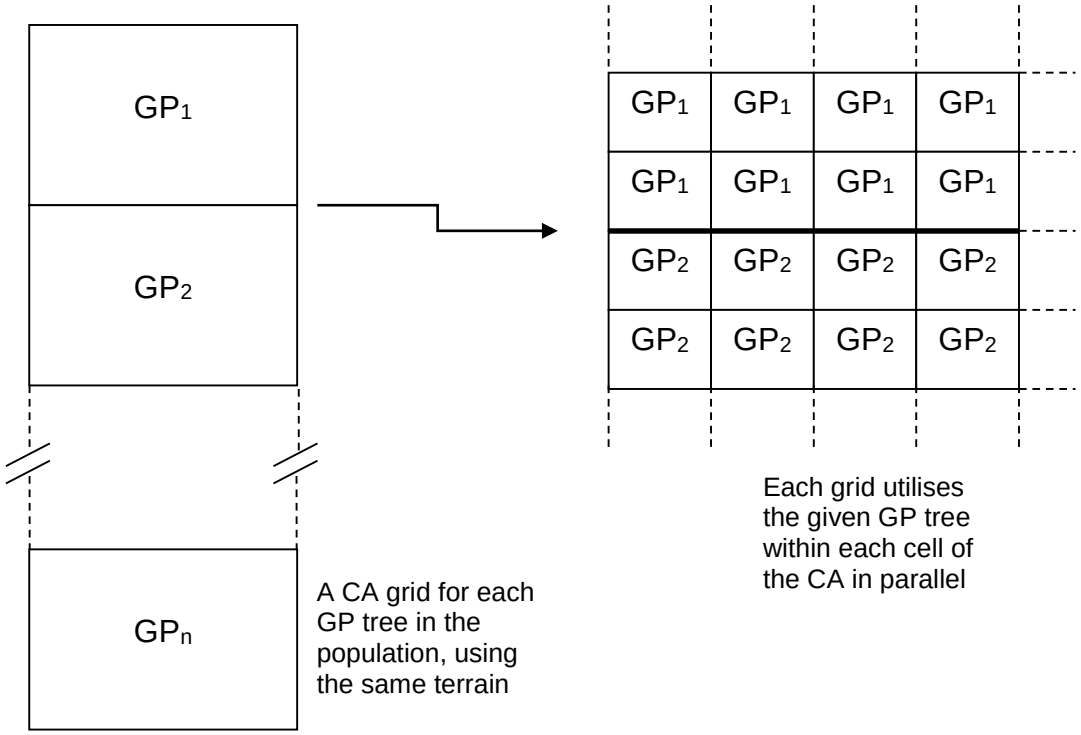


Figure 4.3, Demonstrates how the system is parallelised, on the left, the CA grid is extended as many times as there are GP decision trees which are applied as the state transition rule for all the cells in each section (where no interaction between section/repetition of the terrain is allowed).The subscript after the GP denotes which GP tree of the population is currently being utilised, and n is a variable from 1 to the number of GP in the population. On the right, within each section of the CA that particular GP decision tree is applied within every cell.

For example the terrain used in [65] has only 600 cells, and experiments in Chapter 3:, have shown this is not enough parallel elements. Where the number of hardware cores in current GPU's is in the order of hundreds or thousands, it is shown in Chapter 3: that the number of threads/cells needs to be between one to two orders of magnitude larger than the number of cores in order to make the best use of the available hardware. However, when this number is multiplied by the number of GP individuals requiring evaluation at each generation of the GP, the number of parallel elements can be dramatically increased. For example, if all 100 individuals require evaluation, and then a sizeable 60,000 cells are evaluated in parallel on the GPU. Furthermore, the majority of the experiments run a number of differently seeded GP populations, and if 10 populations are evaluated in parallel then 600,000 cells can be evaluated in parallel on a single GPU. Clearly the use of the CA's parallelism at this scale, would be insufficient to take full advantage of the GPU hardware, however this method of utilising both the GP and CA parallelism simultaneously allows for far greater parallelism to be harnessed, without the need to expand the original test case. It is noted that this

method only works if each of the CA to be evaluated in parallel are the same size, i.e. the same test case is applied to each GP.

4.2.4.2 GP decision tree evaluation

Evaluating a standard GP tree on the GPU is made more difficult by the lack of recursion or variable sized arrays. Where recursion is the most obvious way to evaluate a variable sized tree structure with a standard CPU based programming paradigm. Due to the need for the additional hardware to hold the return address and recursion details, the GPU cannot physically perform recursion in the same way. Therefore, a looping system is created, which keeps track of the current and the previous nodes visited, using an indexing system, where the details of the tree are stored in an array of indices, parent and children indices. By knowing which node of the tree the process is currently on and which node was last visited (either a parent of child), the system can know which node is next to visit and if the current node should be evaluated, or if the system is traversing back up or down the tree.

The tree is then evaluated in a depth first fashion, where upon reaching a terminal node, the value is pushed on to the stack. The key element that is a stack is again made difficult by the lack variable sized arrays on the GPU. Therefore, a hardcoded limited sized array is created within each GP tree instantiation, and a variable is used to keep track of the current top of the stack. It is upon traversing back up the tree that the operations are carried out, for example for a standard binary operation such as addition, then two values are popped from the stack and the result of the operation is then push on to the stack. Therefore, two switch cases are required, one for the possible operations and another for the possible terminal variable values, where the static values are stored literally. Due to the way that values are stored and then retrieved during the return traversal, the maximum size of the stack is equal to the maximum depth of the given GP tree.

4.2.4.3 Hardware difference of the power function between CPU and GPU

Appendix 9.1 explains details of how the outputs of the power function may be different on the different CPU and GPU hardware, another difference between the CPU and GPU implementations is the protection of the power function. As it

is difficult to predicted which inputs will cause outlier output values, such as Not-A-Number (NaN) values, indeterminate, infinity (plus or minus), post calculation checks are utilised to capture these spurious results. While the CPU is capable of capturing all of these spurious results directly after its calculation within the GP decision tree, the current OpenCL API is unable to capture all of these, particularly the indeterminate values. Therefore, both implementations capture only the Not-A-Number (NaN) values, infinity (plus or minus) values, after each calculation step and return a zero value for that particular node. This however leaves the occurrences of indeterminate values to be passed down the GP decision tree, and then can occur as states within the simulations. Since results are returned from processing the simulation and fitness function on either CPU or GPU device, return finally to CPU for the processing of genetic operators, then at this stage any remaining spurious results can be captured. This rare occurrence means that the entire GP decision tree must have a penalised error/fitness score.

4.2.4.4 Parallel fitness function

The GPU is responsible for the processing of a CA simulation for each GP individual, where the GP individual is used as the state transition rules for every cell of that particular CA simulation. The fitness function is also particularly tailored, so as to not to cause losses of parallelisation, in that a grid of error values for each cell is maintained during the CA simulation. This leaves the reduction process of finding a single error value for the entire grid, until the end of the CA simulation. Therefore, an RMSE of the entire simulation is obtained, i.e. a RMSE of every cell in every iteration of the CA.

4.3 GP CA - The Game of Life experimentation

The methodology described in the previous section, allows the GPCA system to be applied to any CA system so long as an interface between the CA neighbourhood and the GP tree system is created. In this section, experiments are conducted to show that such a system can learn a relatively simple rule set such as the Game of Life (which was previously described in section 2.1.1). The Game of Life rule set is so simple it allows for the calibration and verification of

the GPCA system, where the global optimum is known, or rather can be tested for.

The example simulation may not contain examples of every state transition, as is the case with the simulation used in these experiments. However, the since it is possible to easily check for every state transition required for the game of life, the resulting rule sets can be tested to see how well they have learnt the underlying Game of Life rule sets.

4.3.1 Experimental setup

In these experiments a random distribution of live (state 1) and dead cells (state 0) are created by using a 50% chance of either on each cell, using a 100 x 100 grid. This is then used as the initial configuration for a simulation of the Game of Life, for a total of 10 generations and the data from this simulation is then used as the target for the GPCA experimentation.

In order to maintain the uniformity of the rule set, the counting of the neighbouring states is explicitly programmed where if the neighbouring state is equal to or higher than the alive state (one) then it adds one to the value of the counter variable (`NH_count`), for each of the eight possible neighbours in the Moore neighbourhood. The state of the main (central) cell of each neighbourhood is contained in the variable (`mainCell`). Therefore, there are only two variables in the rule set, the `NH_count` variable which can range from 0 to 8 inclusively, and the `mainCell` variable which should be either alive (state 1) or dead (state 0). However, since this is a simple problem for the GP, the experiments do not limit the output of the GP to just zero or one, but allow any output (continuous floating point values), and thus force the GP system to learn the correct outputs. From these possible variable inputs, and the expected outputs of the GOL rule set, there are only 16 combinations (Shown below in Table 4.1). However, if this continuous system is allowed to use the outputs from one generation as the inputs for the next, there will be a much larger range of possible value for the `mainCell` variable.

Table 4.1, The 16 possible variable inputs (where mainCell column shows the current state of the main cell, and Live neighbouring cell count shows the number of alive neighbour), and expected outputs of the main cell in the next time step.

GOL expected output	`mainCell`	Live neighbouring cell count (`NH_count`)
0	0	0
0	1	0
0	0	1
0	1	1
0	0	2
1	1	2
1	0	3
1	1	3
0	0	4
0	1	4
0	0	5
0	1	5
0	0	6
0	1	6
0	0	7
0	1	7
0	0	8
0	1	8

A number of suitable terminals and operators are chosen in order to tackle this problem, which is shown in Table 4.2. Also a number of static variables are supplied to be available to the GP system, integer values ranging from 0 to 9 inclusively, and the fraction 0.1 up to 0.9, at increments of 0.1 are all supplied to the GP as additional terminals. For the purposes of the game of life rule set, two static variables are introduced which represent the alive state of one, and dead state of zero.

Table 4.2, Terminal and Operator set used for the GP system, for learning the Game of Life rule set.

Node Name	Operator/ Terminal	Description
add (+)	Binary Operator	Adds two values
Subtract (-)	Binary Operator	Subtracts right value from left
Protected division (%)	Binary Operator	Divide left by right, unless right is zero, in which case returns zero
Multiply (*)	Binary Operator	Multiply two values
Greater Than (>)	Binary Operator	If left value is larger than right value return one, else zero
Smaller Than (<)	Binary Operator	If left value is smaller than right value return one, else zero
Equality (==)	Binary Operator	If both values are equal return one
If-then-else (if)	Ternary Operator	If left value is greater than zero, then return value of middle branch, else right branch
And(&&)	Binary Operator	If left value is greater than zero, and right value is greater than zero, return one, else zero
Or(||)	Binary Operator	If left value is greater than zero, or right value is greater than zero, return one, else zero
ALIVE (static state 1)	Terminal	Static value of 1.0
DEAD (static state 0)	Terminal	Static value of 0.0
mainCell	Terminal	Variable – The value of the main cell
NH_count	Terminal	Variable – The count of the neighbouring value which are one or more. 0 to 8, inclusively.

Figure 4.4, shows an instantiation of the Game of Life rule set, which will gives the correct GOL output for each of the possible variables inputs, although there are many possible GP trees that will replicate these state transitions.

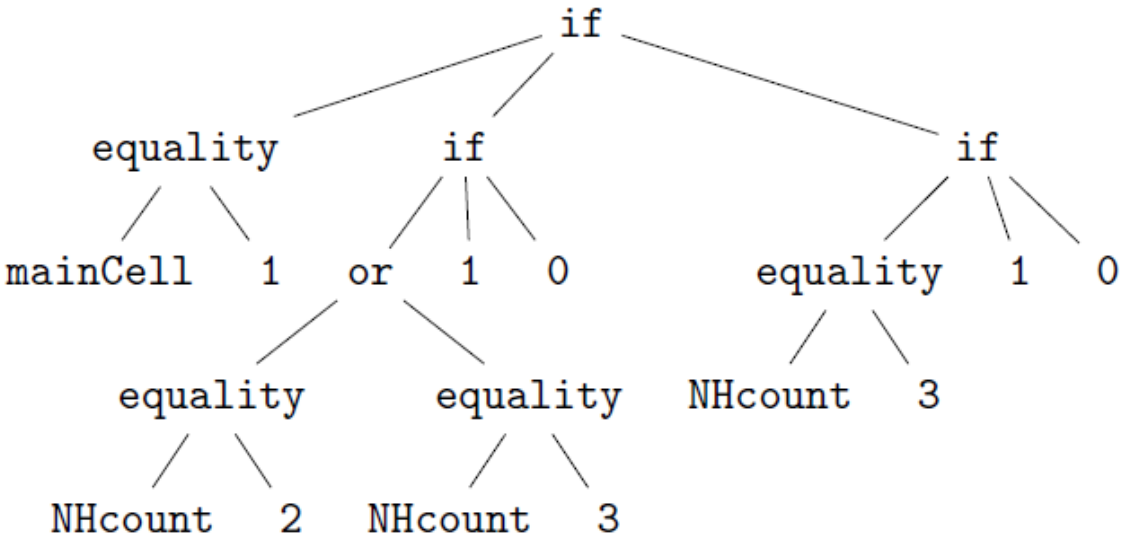


Figure 4.4, A human programmed GP tree which will clearly produces the required state transition in Table 4.1, and therefore is valid version of the game of life rule set (one of many possible instantiations).

10 populations each with a different seed values are run, with the GP parameters described in Table 4.3. The termination criterion for the experimentation is limited to 500 GP generations.

Table 4.3, The Genetic Programming parameters applied game of life in tests.

Population size	100 (*10 run in parallel)
Initial population set-up	Depth 3 full growth
Mutation types	Change, Insert, Remove nodes, and replace sub-tree (maximum depth 3)
Mutation level	2.5% chance per node
Selection type	Tournament from 10 random individuals
Cross-over chance	80%
Elitist individuals	1%
Maximum GP tree depth	10

4.3.2 Experimental results

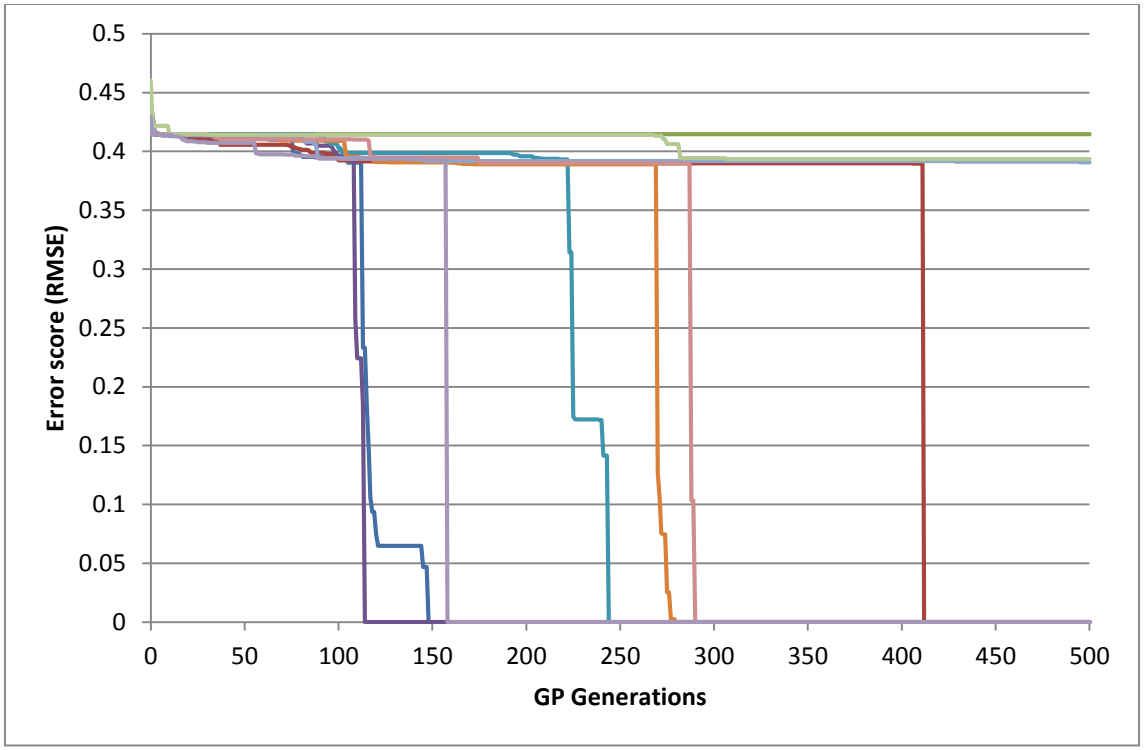


Figure 4.5, Error score (RMSE) of the fittest individual with each of the 10 populations.

In the results shown in Figure 4.5, there are 7 out of 10 populations which successfully match the target simulation data, and out of these 6 out of 7 of those populations match all the GOL state transitions (i.e. on one occasion the system has discovered an alternative ruleset for generating the required outcomes which does not match the Game of Life state transitions). This is because the target 10 CA iterations of the Game of Life, at this grid size and the initial configuration does not contain examples of every state transition. This is a problem with the fact that the GOL rule set, has been shown to tend to converge towards a low average neighbourhood counts (Figure 3.39) causing a lower overall chance of high neighbourhood counts occurring in the target simulation. Also due to the initial configuration using a 50% chance of a cell being alive, the chance of a high live neighbouring cell count are initially low, as well as the rules tendency to lower counts. I.e. it is not likely that a single game of life simulation with an initial random distribution of 50% chance of alive or dead cells will have the state transitions with high counts of alive neighbours present. It is however encouraging that the majority of learnt rules sets have learnt the underlying system, even considering that it is possible to find alternative rules. Figure 4.1 shows the state transitions of the GP individual which scored zero error on the target simulation, but does

not match all the GOL state transitions. It is clearly this lack of state transitions with higher neighbourhood cell counts, which are lacking from this target example.

Table 4.4, The 16 possible variable inputs, and expected outputs, and the one GP run which matched the target but didn't perfectly match all the state transitions. Incorrect outputs are highlighted in bold.

GOL expected output	GP output	MainCell	NH_count
0	0	0	0
0	0	1	0
0	0	0	1
0	0	1	1
0	0	0	2
1	1	1	2
1	1	0	3
1	1	1	3
0	0	0	4
0	0	1	4
0	**1**	0	5
0	**1**	1	5
0	**1**	0	6
0	**1**	1	6
0	**1**	0	7
0	**1**	1	7
0	**1**	0	8
0	**1**	1	8

An example evolved GP tree which successfully scored both zero error on the target simulation and fully replicates all the binary state transitions of the Game of Life is shown in Figure 4.6. It is relatively easy to read, although a rather different approach to the well-known human formulation. The left side checks to see if the neighbourhood count is less than 4, i.e. if it is 3, 2, or 1, resulting in 1 if so. Then the right side of the tree, checks if the main cell's value is dead or alive, which results in the entire right hand side of the tree checking if the neighbourhood count is less than either 3 or 2, performing the latter if the cell is alive. Since the left side is subtracted from the right side, this results in the correct output where if the neighbourhood count is 3 it always makes the cell alive, and if the main cell is alive and 2 or 3 other neighbours are also alive then the result is a live cell in the next iteration.

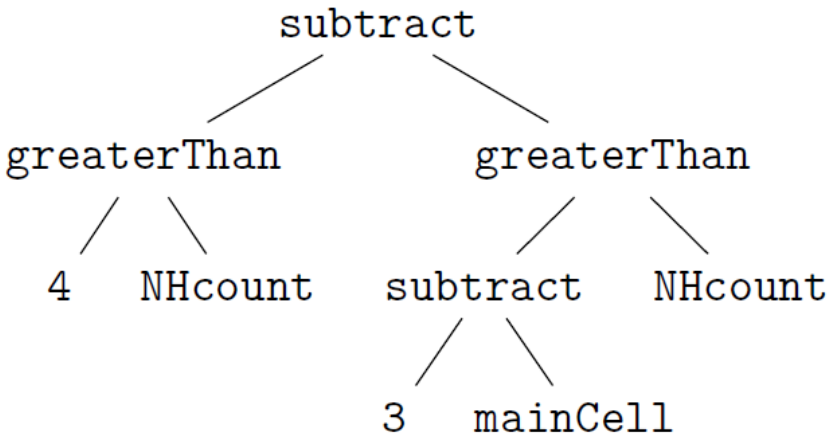


Figure 4.6, An example evolved version of the game of life rule set.

4.4 Conclusions

The GPCA system has been shown effectively learn the state transition rule of relatively simple Game of Life, even considering that the GP system is not limited to binary states, it has managed to learn the correct output in the majority of cases. The fact that the system is not constrained to purely the use of binary outputs and intermediary values at each node means that it has produced particularly readable formulae outputs, as they must directly produce the correct output. I.e. because the GP system can produce such a large variation of outputs, and these outputs are directly used to represent the new cell states (no re-interpretation is used), then it must produce a formula that directly produces the correct outputs. This is guided to the correct outputs by the fitness function.

The tests show that for this simple rule set, there is a large jump in fitness landscape between the perfect score and other fit individuals (Shown in Figure 4.5). This is because if a fit (i.e. near perfect) individual makes some small errors within its state transition rule, then these values are used as the input for the next CA iteration, and so on for the full 10 CA iterations. This allows for a small error in the state transition rule to cause larger error scores, also because there is no relation of the cell states to concepts such as mass or energy, and therefore no such concepts of mass/energy conservation, then it is possible to make large jumps in the patterns between CA iterations.

Signs of convergence are difficult to quantify with a generational GP population, due to way the populations best may have no change for some time,

meanwhile the mean score of population may get worse, and this may lead to eventual improvements in the best and the average. The length of 500 generations of GP optimisation was chosen as it would appear that the majority of populations have performed the majority of their optimisation, also to establish fair and even termination criteria in the testing. However, it is possible for populations to make a sudden and large jump in the fitness landscape which can quickly change the majority of the population. It does appear that due to the initial growth of GP trees, that optimisation appears to slow down shortly after the maximum depth is reached.

The results in Figure 4.5 could be interpreted as discouraging due to the large jump in fitness for those successfully achieving zero error; however this is due to the limited size of the search space, the binary nature of the CA and the chaotic nature of the Game of Life rule set. It is however encouraging that most (7 out of 10) achieved the goal and most of them (6 out of 7) generalised to the unseen state transitions for the Game of Life. Also given the binary nature of the Game Of life rule set, this search is too simple for GP, and similar results could possibly be obtained from a random search. However, it is expected that when using the GPCA methodology for real-world continuous CA rule sets, this problem will be largely overcome by a larger and generally smoother search space. The final conclusion for this chapter is that the GPCA system is indeed capable of learning CA rules for the simple Game of Life system, although the search is made more difficult by the binary states and complex system produced by the Game of Life.

Chapter 5: GP CA real-world flood modelling

5.1 Introduction

This chapter describes the experimentation using the GPCA system to train real world urban flood modelling rule sets, at a single fixed spatial resolution and temporal resolution. The work in this chapter meets objectives 4, 4.1, and 4.2 by firstly ascertaining how well the GPCA system can learn a CA local state transition rule for a real world system at a single resolution, and then testing this on unseen terrain, and water input data. Limiting the training and testing in this chapter to a single spatial and temporal resolution is designed to test the ability of the GPCA system to learn a real world rule set, without also considering what are effectively different rules at different resolutions. This work is then extended in the next chapter to train and test over different spatial and temporal resolutions.

5.1.1 Chapter Structure

Section 5.2 extends the methodology established in section 4.2 for the GPCA system with a new interface for the new modelling. I.e. as the CA now has continuous states and more states than previously used in the Game of Life, and physical limitations must also be imposed over the GP within the local CA neighbourhood, a new interface is required. Section 5.3 details the experimental set-up for the real-world cases used in this chapter and the next, including specifics of the training and testing cases utilised. Also the best known flood modelling CA rules from literature are shown in Section 5.3.3, which are used as human competition/benchmarks for the experiments.

After this, the chapter is split into two main sets of experiments, training in section 5.4 and testing on unseen data in section 5.5. In the training experiments performed in section 5.4 the aim is establish how much training data is required in order that the GPCA system can train effective rules. This is then tempered by testing the rules generated in section 5.4 on unseen data in section 5.5, in order to establish which amount of training data presented creates the most effective rules at generalising to unseen data. Finally conclusions are drawn from the combination of the training and testing rules in section 5.6.

5.2 Methodology

The GPCA methodology introduced in section 4.2 is utilised with a modified GP-CA interface which is detailed below in section 5.2.1.

5.2.1 Real world hydraulic GP interface

The system for modelling of hydraulic water/fluid movements within CA has been described previously section 2.1.3 and 2.1.4; this section introduces how the GP interfaces with this system. The modelling of water and terrain levels requires a continuous CA, where the states and values of the cells are represented by floating point (double precision) values. The state transitions are represented by continuous formulae, which will therefore output a value for every input and each input will always output the same value. Each cell has two main values, the terrain level and the water depth. The terrain level does not change for each cell during a simulation, but changes from cell to cell. The water level is the combination of the water depth and terrain level of the cell, as shown in Figure 5.1. The grid itself has certain static variables which are the same for every cell in the grid, for example the cell size, time step and roughness factors (although the roughness can in certain circumstances vary across the grid).

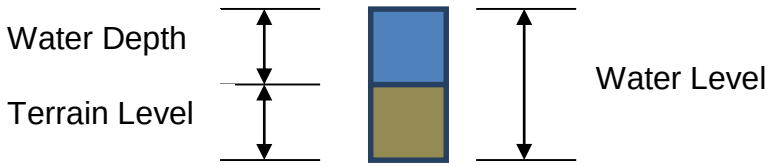


Figure 5.1, Side view of a cell as represented by the continuous values the terrain level, and water depth, which summed together equal the water level, stored within each cell of an open channel CA system (repeated from Figure 2.11).

A number of methods for flood modelling in the literature [110] [64] [111] [63] [65], use a method whereby the state transition rules are formed from repeated use of a single formula between the main (central) cell of the neighbourhood and each of the neighbouring cells. This establishes the outflow rates in up to four directions, and then the total outflows are capped such that the total outflow from all four directions does not exceed the volume of water within the main cell (Shown in Figure 5.2). In order that an outflow is written by only a

single cell between each pair of cell in the grid, outflows are only calculated to neighbouring cells which have a lower water level. This method is designed such that water volume can be preserved across the grid, so water volumes within each cell can neither be created nor destroyed, and can only move laterally across the grid in the x and y axes.

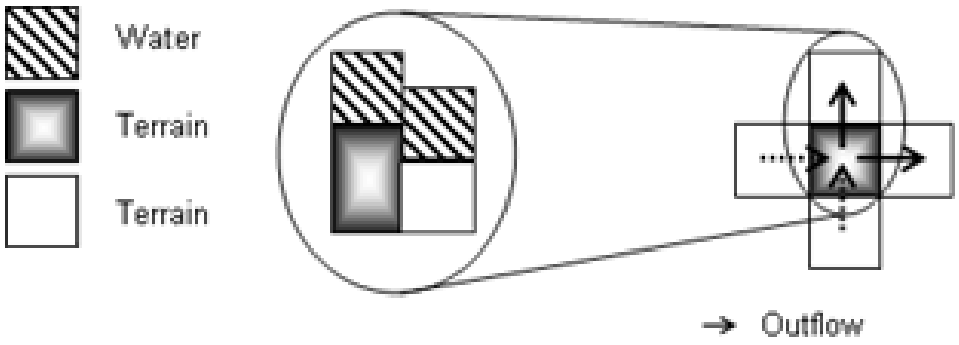


Figure 5.2, Demonstrates how the outflows are calculated within the Cellular Automata system, between the main (central) cell and each neighbour of the Von Neumann neighbourhood (repeated from Figure 2.12). Centrally showing a side view of terrain and water levels in the pair of cells highlighted in the Von Neumann neighbourhood, Right, showing a plan view of the neighbourhood.

This requires a two stage system to be employed, where the GP formula is used up to four times per cell to establish the outflows in each direction for each cell. If the total flows exceed the volume of the central cell, then the flows are normalised by the amount of water within the central cell. The flows are normalised proportionately to their calculated values but do not cumulatively exceed the volume of water with the central cell. However, since the calculated outflow dictated that more water should move than is available, it appears logical to reduce this to just as much as is available. Only outflows from the central cell are calculated in the first stage, as an inflow to the current cell is an outflow from another cell. Due to the way that outflows may be normalised by the amount of water within the main cell, when the total outflow exceeds this level, then the individual outflows are not calculable from neighbouring cells. Instead these values must be written to an intermediary grid of edge values (an edge buffer, shown in Figure 5.3). A second stage is then utilised where, each cell reads from the edge buffer for every one of its four neighbouring cells. In this stage both inflows and outflows are read from the edge buffer, where of course an inflow to one cell is read as an outflow from another cell. Once all inflows and outflows are read from the edge buffer by a cell, then the new water level is calculated by subtracting the volumes of outflows and adding the inflows to the current water

volume of the central cell. In this way the water is always balanced across the grid, regardless of what outflows are calculated between each pair of cells.

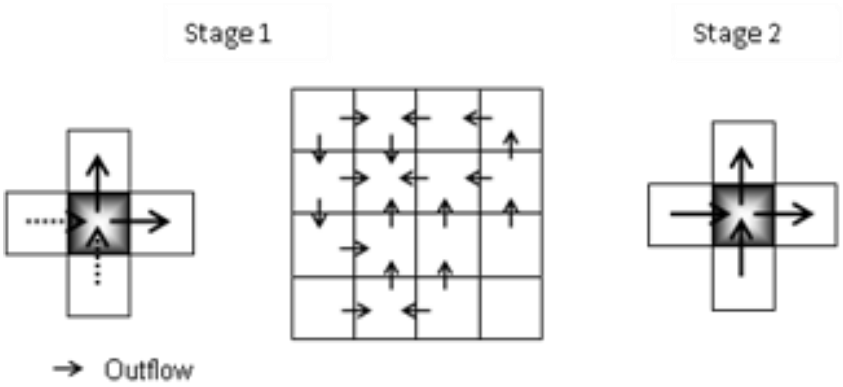


Figure 5.3, Two stages of the CA flood system. Stage 1 for every pair of cells an outflow is calculated, stage 2 every cell updates water depths by means subtracting outflows and adding inflows (repeated from Figure 2.13).

In this system, the GP formula represents the relation between a pair of cells, within the constraint of not exceeding the volume preservation rules of the Von Neumann neighbourhood. Outflows are only calculated from a main cell to a neighbouring cell when the neighbouring cell's water level is lower than that of the main cell, which prevents outflows being calculated in two opposing directions. However, this means that the value taken from any GP formula cannot be negative, and therefore the absolute of the GP tree's calculated output value is used to represent the pre-normalised outflows.

5.3 Experimental setup

Operators and terminals are selected so as to be suitable for the processing of such a rule as the Manning's formula (Shown in Equation 2.1, in section 2.1.4.1), therefore the following operators are utilised (Shown in Table 5.1).

Table 5.1, Terminal and operator set used for the GP system when applied to flood modelling, where the new power operator is highlighted in bold.

Node Name	Operator/ Terminal	Description
Add (+)	Binary Operator	Adds two values
Subtract (-)	Binary Operator	Subtract right value from left
Protected division (%)	Binary Operator	Divide left by right, unless right is zero, in which case returns zero
Multiply (*)	Binary Operator	Multiply two values
Greater Than (>)	Binary Operator	If left value is greater than right value return one, else zero
Smaller Than (<)	Binary Operator	If left value is smaller than right value return one, else zero
Equality (==)	Binary Operator	If both values are equal return one
If-then-else (if)	Ternary Operator	If left value is greater than zero, then return value of middle branch, else right branch
And(&&)	Binary Operator	If left value is greater than zero, and right value is greater than zero, return one, else zero
Or(||)	Binary Operator	If left value is greater than zero, or right value is bigger than zero, return one, else zero
Power (pow)	Binary Operator	Raise the left value to the power of the right value (If is Nan or Infinity, return zero)
Main Cell Water Depth	Terminal	The water depth of the main cell
Main Cell Water Level	Terminal	The water level (water depth plus terrain level) of the main cell
Main Cell Terrain Level	Terminal	The terrain level of the main cell
Neighbouring Cell Water Depth	Terminal	The water depth of the neighbouring cell
Neighbouring Cell Water level	Terminal	The water level (water depth plus terrain level) of the neighbouring cell
Neighbouring Cell Terrain Level	Terminal	The terrain level of the neighbouring cell
Cell size	Terminal	The cell size of the grid in meters
Time step	Terminal	The time step applied between each CA iteration
Roughness factor (1/n)	Terminal	One over the roughness factor of the grid

The division function is protected, such that an attempt at division by zero results in a zero value. This kind of operator protection is reasonably standard for

Genetic Programming. However the use of a power function is rare within GP, and it is found to be especially problematic (shown in Appendix 9.1), as the results of the power function for the CPU and GPU are not guaranteed to be the exactly same value. However, the Manning's formula uses both a square, square root and a cube root in its calculation and therefore it is reasonable to assume that the GP system will require some form of similar operator(s). Since it is difficult to determine which input values for the power function will result in out of bounds return values, a check is performed on the results of the calculation of the power function, whereby any out of bounds results (including +/- infinity, Not-A-Number (NaN), or indeterminate values), are reduced to a resulting zero value. It is noted in Appendix 9.1 that this still does not guarantee identical results from the power function between the different hardware platforms, however it guarantees as sensible a result as possible, and ensures consistency on repeated runs of the same platform.

It is noted that the water level variables provided are the addition of the water depth and terrain level of each cell (as shown in Figure 5.1), and that it should of course therefore be possible for the GP system to create the water level variable. However, since GP is capable of both creating composite variables and selecting important variables, both the water levels and the depths, and terrain levels are provided. The cell size and roughness factors also have a role to play in the amount of flow, and are also therefore provided to the GP system. Finally, the time step at which the grid is set to operate is provided, and should play a vital role in how large flows can be. Once again a number of static variables are supplied to be available to the GP system, integer values ranging from 0 to 9 inclusively, and decimals 0.1 up to 0.9, at increments of 0.1 are all supplied to the GP as additional terminals. The Root Mean Squared Error (RMSE) of every cell, at every iteration of the CA simulations is calculated, by summing all the squared errors at every iteration, square rooting, and dividing the number of cell in each grid multiplied by the number of iterations. The GP parameters used in the experiments are displayed in Table 5.2.

Table 5.2, The Genetic Programming parameters applied in real-world urban flood modelling tests.

Population size	100 (*10 run in parallel)
Initial population set-up	Depth 3 full growth
Mutation types	Change, Insert, Remove nodes, and replace sub-tree (maximum depth 3)
Mutation level	2.5% chance per node
Selection type	Tournament from 10 random individuals
Cross-over chance	80%
Elitist individuals	1%
Maximum GP tree depth	10
Termination Criteria	500 Generations

The GPGPU device used for these experiments differs from that of previous experiments in previous sections, and is described in Table 5.3. This more modern GPU has an even large number of processing cores, although each compute core now contains an increase to 192 processing units each (Nvidia Kepler GPUs).

Table 5.3, Full specifications of machines used for in real-world urban flood modelling tests [112]

Machine	Machine B-2
Type	PC workstation
Age	Recent
CPU	Intel Core-I7-2600 @3.4Ghz
CPU cores	4 (8 with Hyper-Threading)
GPGPU	Nvidia Tesla K20
GPU Processing elements (CUDA cores)	2496
GPU Compute cores	13
GPU speed (Core, Memory - MHz)	706, 2600

5.3.1 Hill and Pond - Training case

The Ghimire et. al. hypothetical test case [65] (previous described in section 2.1.4.2), is utilised as a training case. Consisting of 30 x 20 cells, at a 50m resolution; “The terrain consists of both forward and reverse slopes of 0.2%. It also has a lateral slope of 0.1 toward the outlet”, where the outlet is removed for consistency (Shown in Figure 5.4). This example specifies a roughness factor 0.01n is applied across the terrain, and a rain fall of 20mm/h for the first hour of the simulation is used as input for the water depths, where a full simulation time is considered 12 hours.

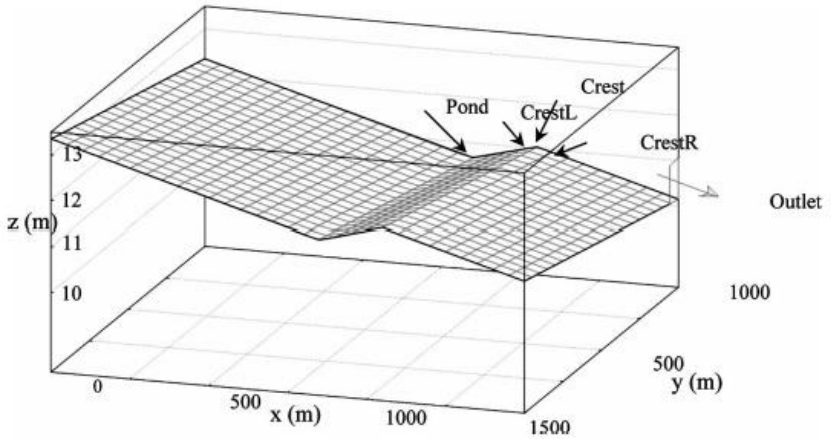


Figure 5.4, Hypothetical ‘Hill and Pond’ terrain, and given test points; taken from Ghimire et. al. [65] (also shown in Figure 2.16).

This terrain (Shown in Figure 5.4) is selected for providing sufficient hydraulic examples while not being an overly large spatial size. This selection is made primarily due the size and processing times of this training simulation, in that because this simulation will need to be repeated for every new individual at every GP generation, the processing times will quickly mount up for the entire optimisation process. For example, even with a very short processing time of 1.0 seconds for each CA simulation, and a population of 100, and 500 generations would equate to approximately 14 hours of processing. Therefore, the processing time of single fitness evaluation is very important, and the selection of the size of the test case used, must be made carefully.

In order to carry out meaningful investigations of the system, multiple separate populations will need to be evaluated. Research in (Sections 3.5.1 and 3.5.2), shows that in order to achieve a significant speed-up factor from

parallelisation that a large enough number of parallel elements are required, i.e. the number of cells in the CA. However as described in the previous paragraph a single CA simulation needs to limited in size in order to avoid the multiplication of many evaluations required for the GP system. This paradoxical problem is answered in the next section, by means of the novel method of harnessing the parallelism of both the CA and the population of individuals with the GP system (described in Section 4.2.4.1).

5.3.2 Testing and validation simulation cases

Larger terrains are possible for testing and validation, due to fact that a single simulation need only be run on each, as opposed to training which requires many. For these larger terrains, using a sampling rate of one second, it is possible to exceed the limitations of modern hardware memory limitation, where this may total many hundreds of Gigabytes of data. Therefore, the entire simulation target is not loaded into memory, but rather only the required point in time for the simulation that is being tested. This adds the loading time into the simulation time, and brings these test simulations in to the range of minutes to perform each. Where these scales of real world simulation processing times are acceptable for the testing purposes they are clearly too large for the training purposes.

It is hoped that the system where a single local rule is trained, will be able to generalise very well to other configurations. Therefore other test cases are selected, this time from the UK Environment agency Benchmark test suite [1]. However, the resulting trained GP trees are firstly tested on the latter 6 hours of the ‘hill and pond’ simulation; i.e. from 6 up to 12 hours. Also validation is carried out on the same ‘hill and pond’ terrain, for the full 12 hours of simulation, but with a different rain profile, thus this is entirely unseen simulation.

5.3.2.1 EAT-2 Test case

The primary test cases utilised with a different terrain is the EAT-2 (Environment Agency Test), which is shown below (Figure 5.5). EAT2 in its original formulation is a 2000m square grid, using a cell size of 20m, and is therefore 100 by 100 cells, totalling 10,000 cells. In order to test on the same cell sizes as those trained upon; this terrain is scaled up to a 50m cells size, and therefore occupies an area of 5000m^2. The Plan view of the terrain layout of UK

Environment Agencies Test case 2 (EAT2) is shown in Figure 5.5. A rain profile is applied for the majority of tests as opposed to the prescribed inflow conditions, and this represents the primary reason for selecting this terrain, in that it is a viable test case with uniform rain applied. A uniform Manning's roughness factor of 0.01n is used throughout validation, so the system has been trained on only a single set of static variables and tested on simulation with same static variables.

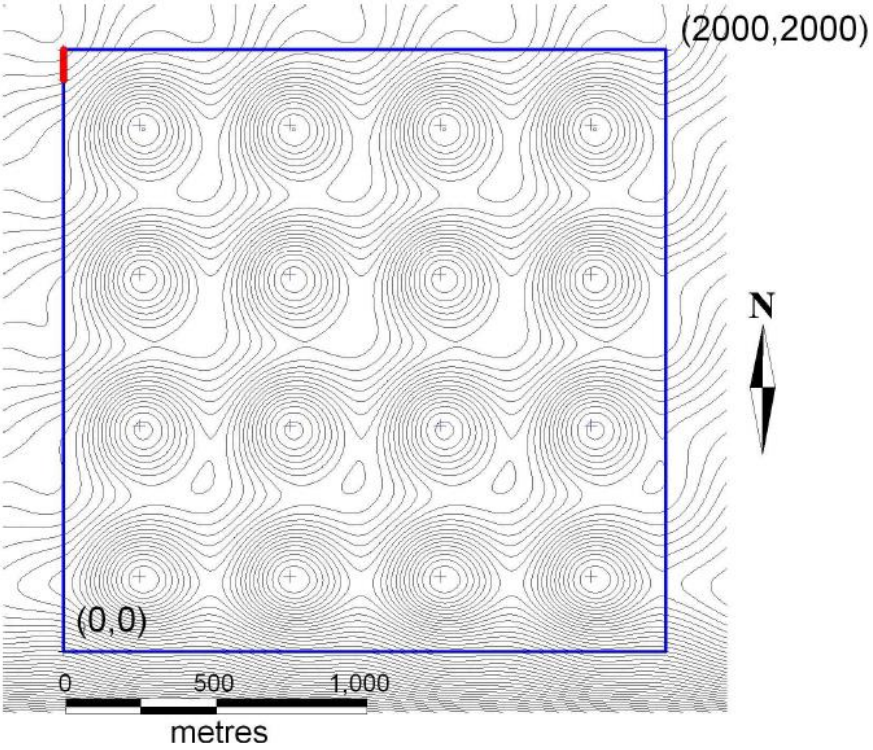


Figure 5.5, EAT2 test case original terrain (Plan view), at 2,000m square with 100x100 cells; which is scaled up to a 5,000m square terrain by increasing the cell size to 50m

5.3.2.2 EAT-1 Test case

The next simpler terrain but very different input conditions test case, is EAT1, where the terrain is essentially a 1 dimensional channel as there is no change in the y-axis. The modelled domain is a 700m X 100m rectangle (Shown in Figure 5.6), using a 10m cell size in its original description, this gives 700 cells total, although due to there being no difference in the y-axis this is simplified down to only 70 cells.

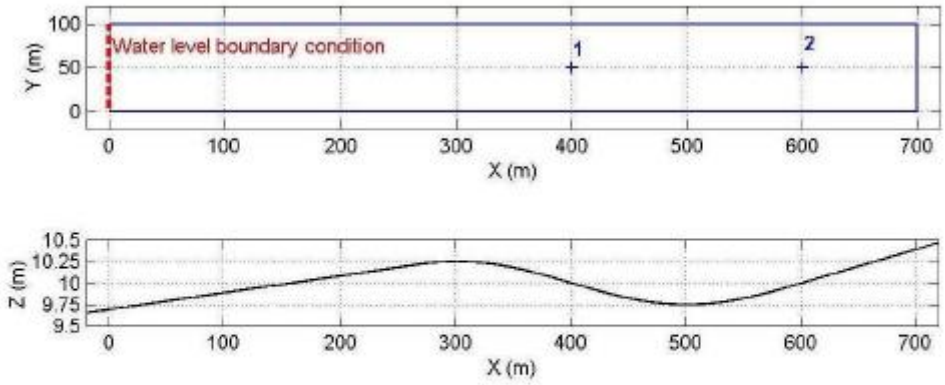


Figure (a): Plan and profile of the DEM use in Test 1. The area modelled is a perfect rectangle extending from X=0 to X=700m and from Y=0 to Y=100m as represented.

Figure 5.6, Plan (top) and profile (bottom) views of the EAT1 terrain (DEM - Digital elevation Model), also showing the two test points in the plan view.

The major difference for the EAT1 test case, compared to that of previous EAT2 cases is that it is designed purely for a different type of inflow. Where previous test cases all use a uniform rain pattern, at a certain rate for a certain amount of simulation time and this rate is applied equally to all cells in the terrain. The EAT1 test cases has what is termed a water level event, whereby at one of the borders (shown in Figure 5.6, plan view - Top) the bordering water level is varied during the simulation. This is used to test a simulated lateral inflow, or outflow, for example a dam break or over flow. Once again this terrain is scaled up to a 50m cell size from its original resolution, so as to be a fair test.

The water level inputs used in the EAT1 test cases are shown in Figure 5.7), where the water level starts at 9.7m which is equal to the terrain level at the input/output border. It is then raised linearly up to 10.35m which is higher than the terrain level, and therefore there is an inflow, which should proceed along the terrain in an easterly direction. Since the higher level of 10.35 is also higher than the peak in the terrain along the x-axis at 300m, then water should flow over this into the pond (located between 300m to 650m, and central at 500m). The water level should then settle at the 10.35m point across the terrain, until the point in time at 11 to 12 hours of simulation time, when it is again lowered to 9.7m. At this point the waters should recede back out of the terrain, but leaving water within the ponding area. The waters on the westerly side of the peak at 300m along the x-axis should fully recede, whereas the water within the ponding area should leave the pond water level at 10.25m, due to the level of the terrain peak at 300m along the x-axis.

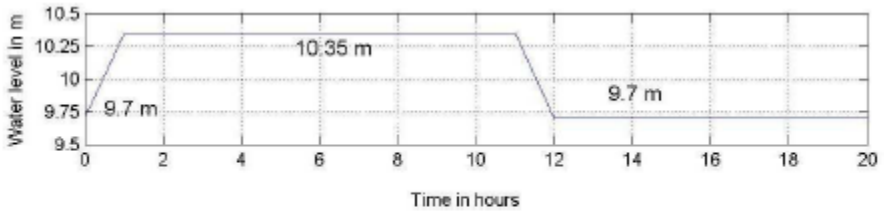


Figure (b): Water level hydrograph used as dowstream boundary condition (table provided as part of dataset).

Figure 5.7, Varied bordering water level event which drives the input to EAT1 test case.

5.3.3 Human competition

Based on the work of previous models four different GP versions of the Manning's formula are formulated, based on the Equation 2.1, Equation 2.2, Equation 2.6 and Section 2.1.4. In order to be able to better encode these human formulations and compare the code, a simple recursive descent parser was created, and the paired look ahead level one language it encodes. This allows a human programmer to program in much more natural format, i.e. avoiding using lists of token in reverse polish notation, and thusly avoiding costly mistakes. Code shown in Figure 5.9, Figure 5.11, Figure 5.13, and Figure 5.16 utilises the simple recursive descent language created by the author and specified in Appendix 9.2. This language accepts C-style comments, and is designed to facility easier and more error resistance human programming of GP decision trees.

5.3.3.1 Ghimire formulation

Ghimire et. al. [65], interpreted the hydraulic radius (R) as the water depth of the main cell, as shown in Figure 5.8 and Figure 5.9. Whereas the full original Ghimire rule set uses a ranking system to distribute the water from the main cell to downhill neighbours, instead their formulation of the Manning's formula is applied within the more standard framework. Thus a fair comparison can be made between the different formulations, when working within this standard schema.

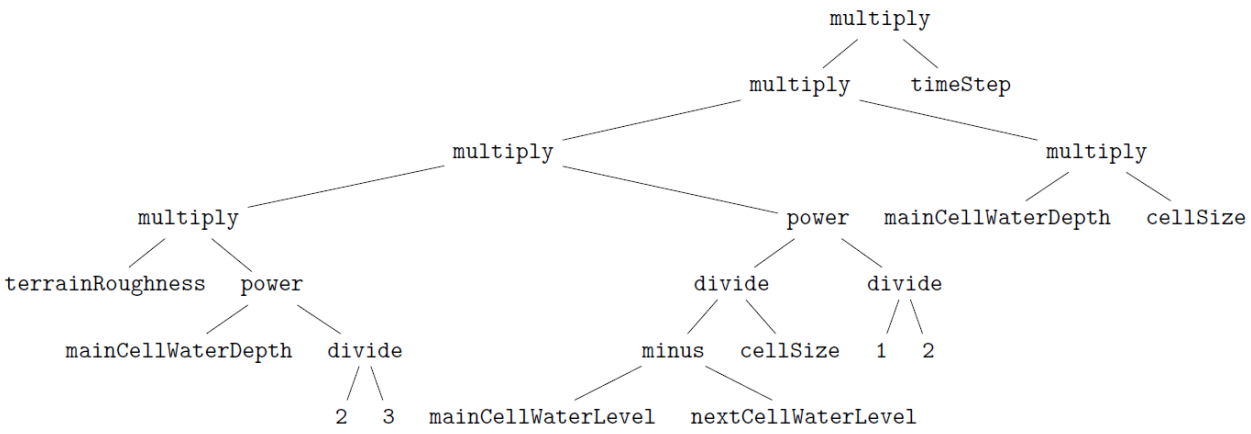


Figure 5.8, Manning's formula, combined with the discharge formula, in GP tree form; used to calculate the volume of water to transfer between a pair of cells, using the Ghimire implementation of the hydraulic radius.

```
25  GP =
26
27
28  //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
29  // (1/n)* (S^(1/2)) * (R^(2/3)) = flowRate;
30  (
31      // (1/n) = terrainRoughness
32      terrainRoughness *
33
34      // R    = mainCell_WaterDepth
35      (mainCell_WaterDepth^(2/3)) *
36
37      // S    = (mainCell_WaterLevel-nextCell_WaterLevel) /cellSize)
38      (((mainCell_WaterLevel-nextCell_WaterLevel) / cellSize)^(1/2))
39  )
40  //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
41  //flowRate * A = volPerSecond;
42
43  //A     = mainCell_WaterDepth * cellSize
44  * (mainCell_WaterDepth * cellSize)
45  //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
46  //volPerSecond * timeStep = VolTransfer;
47  * timeStep
48  //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
49  //VolTransfer / area = waterDepthDifference; !!!
50  //////// (cellSize*cellSize)
51  //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
52  ; // end of GP formula
```

Figure 5.9, Manning's formula, combined with the discharge formula, in GP tree code form (scaled down version of C code); used to calculate the volume of water to transfer between a pair of cells, using the Ghimire implementation of the hydraulic radius.

5.3.3.2 Dottori and Todini formulation

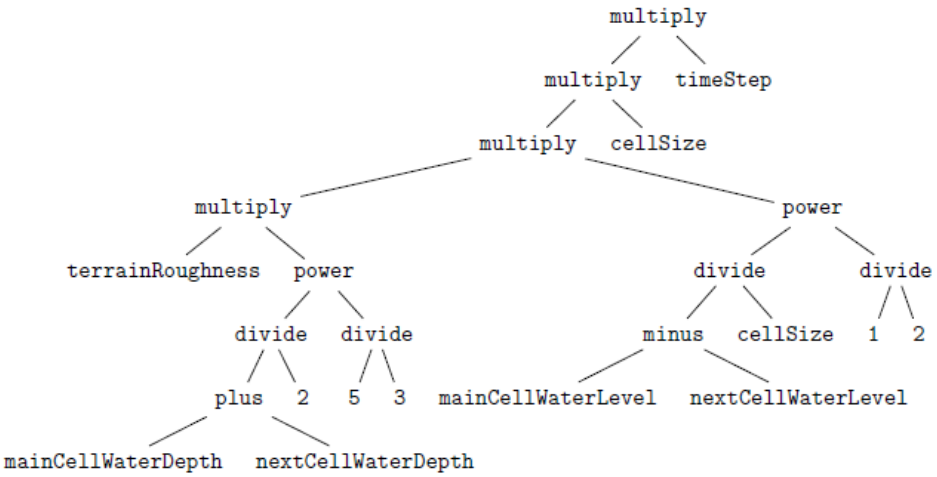


Figure 5.10, Manning's formula, combined with the discharge formula, in GP tree form; used to calculate the volume of water to transfer between a pair of cells, using the Dottori and Todini implementation of the hydraulic radius.

```
GP =


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// (1/n)* (S^(1/2)) * (R^(2/3)) = flowRate;
(
    // (1/n) = terrainRoughness
    terrainRoughness *

    // R    = ((mainCell_WaterDepth+nextCell_WaterDepth)/2)
    (((mainCell_WaterDepth+nextCell_WaterDepth)/2)^(5/3)) *

    // S    = (mainCell_WaterLevel-nextCell_WaterLevel) /cellSize)
    (((mainCell_WaterLevel-nextCell_WaterLevel) / cellSize)^(1/2))
)
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//flowRate * A = volPerSecond;

//A     = mainCell_WaterDepth * cellSize
//* (mainCell_WaterDepth * cellSize)

// B = the width of the contact face.
* cellSize

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//volPerSecond * timeStep = VolTransfer;
* timeStep
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//VolTransfer / area = waterDepthDifference; !!!
//////// (cellSize*cellSize)
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; // end of GP formula
```

Figure 5.11, Manning's formula, combined with the discharge formula, in GP tree code form (scaled down version of C); used to calculate the volume of water to transfer between a pair of cells, using the Dottori and Todini implementation of the hydraulic radius.

The Dottori and Todini formulations (Shown in Figure 5.10 and Figure 5.11) major distinction is the use of the arithmetic mean of the pair of cells water depths, in the place of the hydraulic radius (R). Also they perform a mathematical simplification of moving the hydraulic radius element of the outflow area (A), and

into the power function. I.e. they have raised the hydraulic radius the power of 5/3rds instead of the 2/3rds and only multiplied by the cell size and not the cell size and the water depth of the main cell as in the Ghimire method.

5.3.3.3 Bates and Hunter formulation

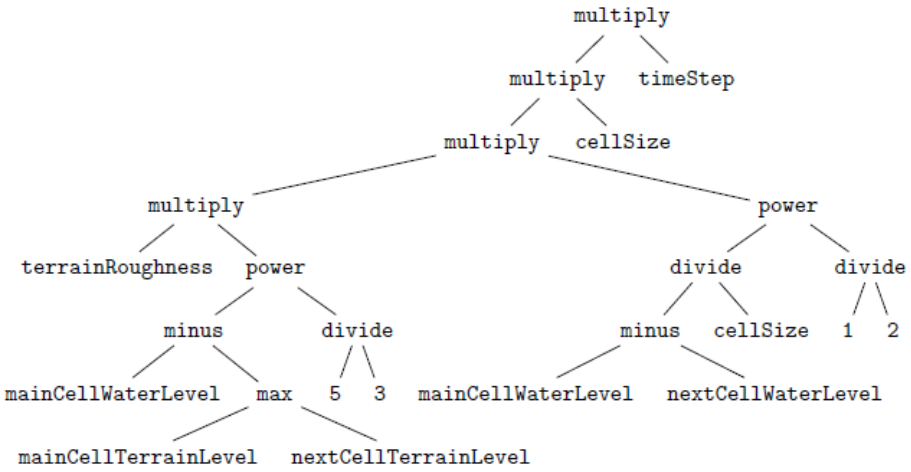


Figure 5.12, Manning's formula, combined with the discharge formula, in GP tree form; used to calculate the volume of water to transfer between a pair of cells, using the Bates and Hunter implementation of the hydraulic radius.

```
GP =


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// (1/n)* (S^(1/2)) * (R^(2/3)) = flowRate;
(
    // (1/n) = terrainRoughness
    terrainRoughness *

    // R    = mainCell_WaterLevel - max(mainCell_terrainLevel,nextCell_terrainLevel)
    ((mainCell_WaterLevel - max(mainCell_TerrainLevel,nextCell_TerrainLevel))^(5/3)) *

    // S    = (mainCell_WaterLevel-nextCell_WaterLevel) /cellSize)
    (((mainCell_WaterLevel-nextCell_WaterLevel) / cellSize)^(1/2))
)
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//flowRate * A = volPerSecond;

//A     = mainCell_WaterDepth * cellSize
//* (mainCell_WaterDepth
* cellSize
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//volPerSecond * timeStep = VolTransfer;
* timeStep
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//VolTransfer / area = waterDepthDifference; !!!
//////// (cellSize*cellSize)
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; // end of GP formula
```

Figure 5.13, Manning's formula, combined with the discharge formula, in GP tree code form (scaled down version of C code); used to calculate the volume of water to transfer between a pair of cells, using the Bates and Hunter implementation of the hydraulic radius.

The major difference between the Bates and Hunter formulation of the Manning's formula is that they use the difference between the main (outflowing and therefore higher) water level and the larger of the two terrain levels, as the hydraulic radius.

5.3.3.4 Bates and Hunter Flow Limited formulation

Bates and Hunter [64], develop a flow limiter to ensure that the flow does not 'over' or 'undershoot', and is a function of flow depth, grid cell size and time step (Shown in Figure 5.14, and previous discussed in section 2.1.4.3).

$$Q_x^{i,j} = \min\left(Q_x^{i,j}, \frac{\Delta x \Delta y\left(h^{i,j} - h^{i-1,j}\right)}{4\Delta t}\right)$$

Figure 5.14, Flow limiter formula, used by Hunter and Bates et. al. where the flow rates are first calculated by the Manning's formula (Shown in Equation 6.1 then the minimum between the above and that outflow are calculated previous shown in Figure 2.23)

This cap is also included as part of the full limited Bates and Hunter formulation, is displayed in Figure 5.15 and Figure 5.16

.

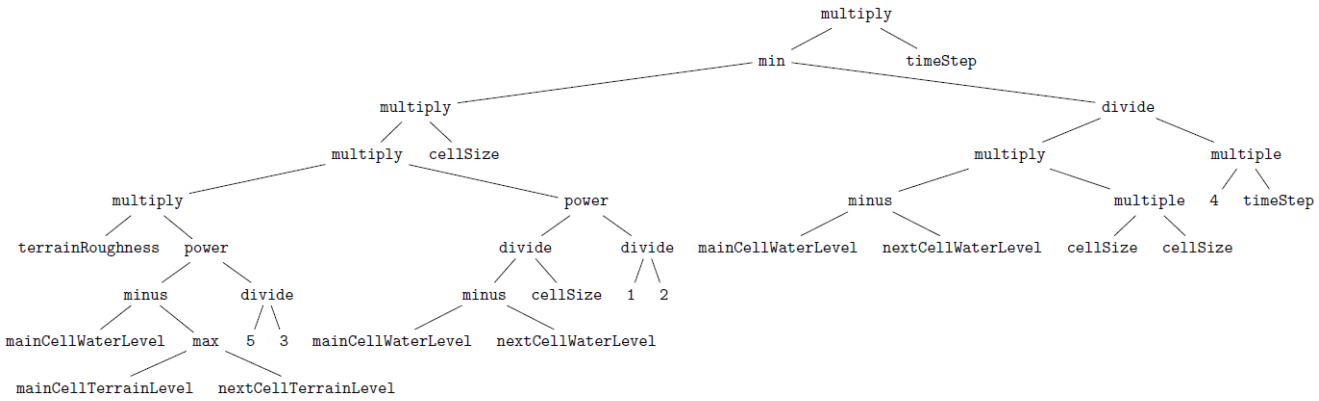


Figure 5.15, Manning's formula, combined with the discharge formula, and Bates & Hunter limiting cap, in GP tree form; used to calculate the volume of water to transfer between a pair of cells, using the Bates and Hunter limited implementation of the hydraulic radius.

```
GP =


min(

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // (1/n)* (S^(1/2)) * (R^(2/3)) = flowRate;
    (
        // (1/n) = terrainRoughness
        terrainRoughness *

        // R    = mainCell_WaterLevel - max(mainCell_terrainLevel,nextCell_terrainLevel)
        ((mainCell_WaterLevel - max(mainCell_TerrainLevel,nextCell_TerrainLevel))^(5/3)) *

        // S    = (mainCell_WaterLevel-nextCell_WaterLevel) /cellSize)
        (((mainCell_WaterLevel-nextCell_WaterLevel) / cellSize)^(1/2))
    )
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    //flowRate * A = volPerSecond;

    //A     = mainCell_WaterDepth * cellSize
    //* (mainCell_WaterDepth
    * cellSize
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    //volPerSecond * timeStep = VolTransfer;

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    //VolTransfer / area = waterDepthDifference; !!!
    //////// (cellSize*cellSize)
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    ,//------------------------------------------------------------------------ next part of min

    ((mainCell_WaterLevel-nextCell_WaterLevel)*(cellSize * cellSize))/
    (4 * timeStep)


)// end of min
* timeStep
; // end of GP formula
```

Figure 5.16, Manning's formula, combined with the discharge formula, and Bates & Hunter limiting cap, in GP tree code form (scaled down version of C); used to calculate the volume of water to transfer between a pair of cells, using the Bates and Hunter limited implementation of the hydraulic radius.

The key difference in the Bates and Hunter limited implementation, is the cap placed on volume transfers which is relative to the time step and total water level volume difference. I.e. the combined Manning's and discharge formula are calculated and then the minimum between this, and the volume difference determined by the water levels, divide by four times the time step, and the result used as the flow rate. Since the calculated flow rate is then multiplied by the time step, this caps the maximum flow in any particular direction to a fourth of the difference in volume between the water level differences.

5.4 Training GP with fixed spatial and temporal resolution

5.4.1 Introduction

In the earlier binary CA systems, the size of cells and the amount of time between CA iterations are both abstract concepts, however real world CA

simulations of urban flood modelling represent a discretisation of movement of incompressible fluids through space and time. The ultimate idea is to have a rule which would be able to model to same real movements of water within space and time. However, this section considers only a single static spatial and temporal resolution of the CA simulations in training and testing. By training in this way on a single spatial and temporal resolution, and testing on the same resolution, the aspects of generalisation that are different terrains and water level inputs can be investigated. The experimental question asked in this section, is how much training is required to enable the system to optimise effectively? Is there a point of diminishing returns, as the volume of training (the amount of simulation time presented) will scale the optimisation time linearly?

5.4.2 Experimental setup

Experiments have been conducted with the GP optimisation using the Ghimire 'Hill and Pond' training case (shown in Section 5.3.1). These spatial and temporal settings used throughout this sections, as well as the rain input used for this test case are shown in Table 5.4.

Table 5.4, Details of the training simulation utilised in this section.

Cell Size	50 Meters
Time Step	1 Second
Roughness Factor	0.01
Water inputs	Initial Dry, uniform rain applied to all cells of 20mm/h for first hour.
Full simulation time	12 hours

Experimentation is conducted for varying lengths of the simulation during training, in order to establish how much of a single CA simulation is required that the system can begin to generalise the space time pattern at this spatial and temporal resolution. Experimentation is conducted using the first 1, 2, 4, and 6 hours of the simulation for training. Notably since a 1 second time step is maintained throughout these experiments this does mean a different number of CA iterations is also carried out for each experiment of 3,600 seconds, 7,200 seconds, 14,400 seconds, and 21,600 seconds. 10 separate populations are run on the CPU implementation (using OpenMP to take full advantage of its

parallelism), and 10 separate populations are processed on the GPGPU (Tesla K20, shown in Table 5.3). This experimental set-up is used to get an average result of the heuristic algorithm and to demonstrate the speed difference in performing of training on the two different architectures. Training is conducted using the GP parameters shown in Table 5.2.

For all these cases the Manning's formula in its 3 forms and the Bates-hunter limited formulation, are run to provide a human competitiveness benchmark level. Also a zero flow candidate GP, and an arbitrarily large flow (1000 units of volume) in order to see both relatively good solutions and two bad solutions fitness scores for each particular test case Table 5.5.

Table 5.5, Fitness scores on the hill and pond test case, starting t = 0 and progressing up to the respective time. The best scores are highlighted in bold.

Simulation end time	ZeroFlow	LargeFlow	Manning's -Ghimire	Manning's -Dottori & Todini	Manning's -Bates & Hunter	Manning's-Bates & Hunter limited
1	46.5212	31.5022	598.054	195.842	611.029	**612.728**
2	22.9313	19.9719	284.828	186.307	519.186	**560.568**
3	19.4262	17.5724	186.053	167.323	484.355	**565.787**
4	18.0948	16.5968	162.283	156.77	470.673	**597.408**
5	17.371	16.0509	155.48	154.691	459.692	**629.897**
6	16.9076	15.6954	154.251	158.037	453.322	**661.575**
7	16.5888	15.4482	155.494	163.297	449.987	**691.427**
8	16.3575	15.2673	157.881	169.129	449.276	**720.369**
9	16.1814	15.1285	160.732	174.993	449.861	**747.818**
10	16.0421	15.0184	163.761	180.705	451.217	**772.992**
11	15.9291	14.9287	166.83	186.189	452.987	**795.763**
12	15.8356	14.8541	169.869	191.422	454.962	**816.273**

With the majority of human formulations (excluding the Bates and Hunter limited formulation), as the amount of simulation time upon which the fitness scores is tested is increased, the rule set converges towards a given value. Also it is clear to see that there is a much larger variation in fitness scores over the first hour, and the first two hours. This is due to both the short simulation period and the fact that because the rain profile is applied during the first hour, there is little water movement during this time and so the training phase is atypical of the rest of the simulation.

5.4.3 Training Results

Table 5.6, Fitness scores (1/RMSE) for the training case from t = 0 up to the respective time shown, for the CPU and GPU trained populations; also showing the maximum and mean fitness for both groups of populations and all GP individuals at each training time. Manning's formulations, limited, zero and large flows are shown for reference. Those highlighted in bold have exceeded the score of the human formulations on the respective training simulation time.

		Hours of training/simulation					
		1	2	3	4	5	6
GPU GP Population	0	**1161.44**	**536.226**		407.985		381.729
	1	**705.405**	**583.692**		417.757		382.537
	2	**940.497**	**856.501**		325.641		372.382
	3	**1064.82**	**506.382**		473.573		117.383
	4	531.4	**626.534**		426.831		611.569
	5	**963.936**	395.724		412.152		341.041
	6	**1162.82**	**626.116**		491.456		424.292
	7	576.466	**607.179**		406.298		424.292
	8	**1063.51**	**881.868**		456.979		341.923
	9	**729.003**	**729.631**		413.994		427.672
CPU GP Population	0	**1016.32**	442.884		340.636		451.643
	1	**1475.1**	415.5		371.077		391.982
	2	**724.696**	**697.367**		371.077		406.925
	3	462.642	458.237		352.405		401.958
	4	**768.077**	467.964		445.557		442.205
	5	**1289.19**	415.621		375.65		395.907
	6	**721.977**	**795.036**		453.281		370.583
	7	**700.701**	314.12		510.021		419.748
	8	**1222.82**	343.572		475.547		329.604
	9	**728.677**	**666.693**		517.33		466.355
GPU GP Maximum		**1162.82**	**881.868**		491.456		611.569
GPU GP Mean		**889.9297**	**634.9853**		423.2666		382.482
CPU GP Maximum		**1475.1**	**795.036**		517.33		466.355
CPU GP Mean		**911.02**	501.6994		421.2581		407.691
Combined GP Maximum		**1475.1**	**881.868**		517.33		611.569
Combined GP Mean		**900.4749**	**568.3424**		422.2624		395.0865
Bates-Limited		612.728	560.568	565.787	597.408	629.897	661.575
Bates-Manning's		611.029	519.186	484.355	470.673	459.692	453.322
Ghimire-Manning's		598.054	284.828	186.053	162.283	155.48	154.251
DT-Manning's		195.842	186.307	167.323	156.77	154.691	158.037
Zero Flow		46.5212	22.9313	19.4262	18.0948	17.371	16.9076
large Flow		31.5022	19.9719	17.5724	16.5968	16.0509	15.6954

From the results in Table 5.6 it would appear that the first hour or the first two hours of the training simulation present little challenge for the GP, however there are slightly higher scores for even the zero and large flow formulations during this period. This is attributed to the lack of change in water levels during this period, and while there are obviously some flows, both these periods of training prove much easier for the system to gain higher scores on. The GP system has outperformed all the human formulations during these periods, and as the amount of simulation time presented is increased the overall performance decreases. While for all periods the GP output performs the Ghimire, and Dottori & Todini implementations, on the longest periods of simulation time the Bates formulation outperform the GP systems. From these training results, GP has managed in all cases to perform competitively amongst the best human formulations, although performance appears to decrease with the additional simulation time presented.

5.4.4 Processing times and speed-ups from GPU computing

At this stage it is stressed how much processing time is required for these runs, especially considering full use is made of modern multi-core I7 (4/8 cores) CPU's processing times still take in the order or days to complete. Where previous experiments with CA and GPU have measured the difference in speed-up between the serial CPU implementations and OpenMP/parallel CPU implementations and then the GPU implementations, due to the sheer scope of processing times, the difference between the OpenMP - parallel CPU implementations and the GPU implementations are measured here. Shown in Table 5.7 and Figure 5.17 are the processing times for the entire optimisation processes on each hardware.

Table 5.7, Processing times for each complete GP optimisation run, for both the CPU and the GPGPU, given the number of hours of the training simulation applied. The speed-up factor of the GPGPU over the CPU is shown, along with a breakdown of the processing times in minutes, hours, and days.

	Hours Training	1	2	4	6
	Total processing time(Seconds) CPU:	282622.5	426462.9	633659.5	924994.7
	Total processing time(Seconds) GPU:	50167.78	86652.42	145673.6	167668.6
	speed-up factor (CPU/GPU)	5.633546	4.921535	4.349856	5.516804
CPU	Minutes	4710.375	7107.716	10560.99	15416.58
	Hours	78.50624	118.4619	176.0165	256.943
	Days	3.271093	4.935914	7.334022	10.70596
GPU	Minutes	836.1297	1444.207	2427.894	2794.476
	Hours	13.93549	24.07012	40.4649	46.5746
	Days	0.580646	1.002922	1.686038	1.940608

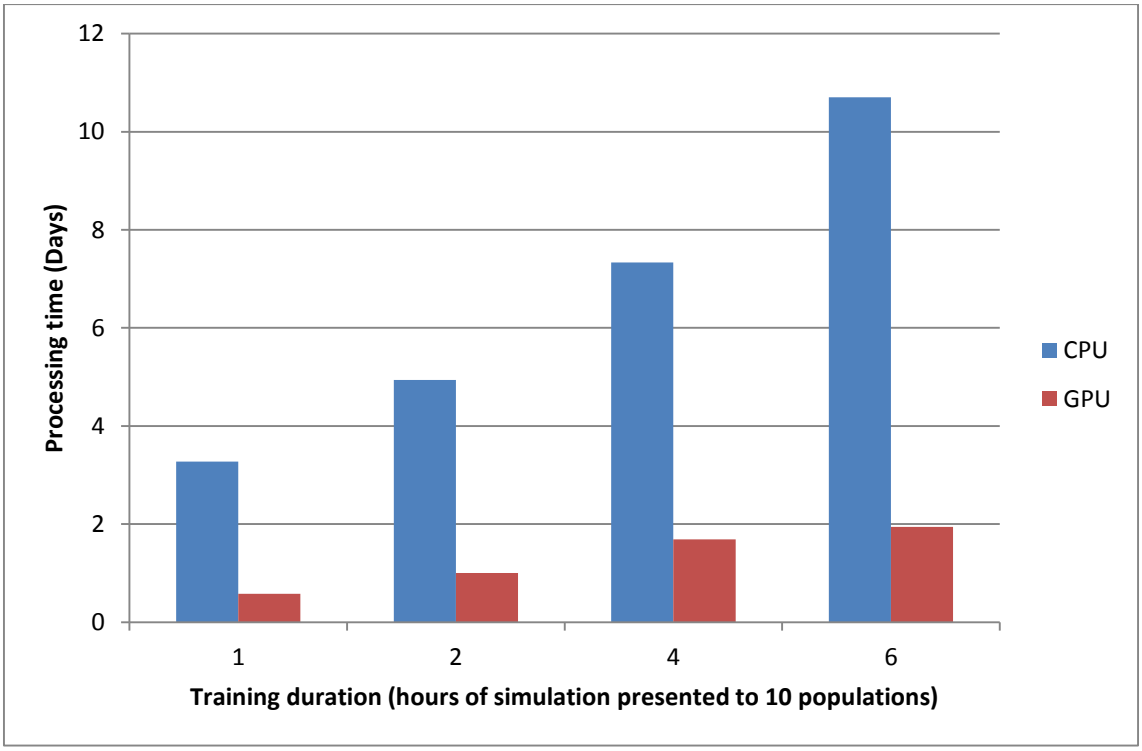


Figure 5.17, Processing times for each complete GP optimisation run for both the CPU and GPGPU in days of processing time, given the number of hours of the training simulation applied.

It is clear to see in Table 5.7 that the processing times are directly related to the amount of the training simulation applied. Also that there is a degree of variation to the speed-up factors of the GPGPU over the CPU, however they remain relatively constant, and at near the predicted levels of approximately a 5x speed up (predicted in section 3.7). This is due the fact that extending the amount of simulation time of the training simulation applied extends the serial processing of the CA simulation (i.e. it may only serve to reduce the amount of overall parallelism). Due to the fact that there are minor difference in the GPGPU implementation and that the use of different seed values, the actual GP individuals in each population are different at each generation, which explains the variation in processing times and speed-up factors. Shown in Figure 5.18, is the processing of each generation for the CPU, and in Figure 5.19 for the GPGPU, and in Figure 5.20 the speed-up factor for each generation is shown. Note that because of the parallelism that is used between the multiple separate populations (Shown in section 4.2.4.1), that a single processing time is provided for each generation of all 10 populations.

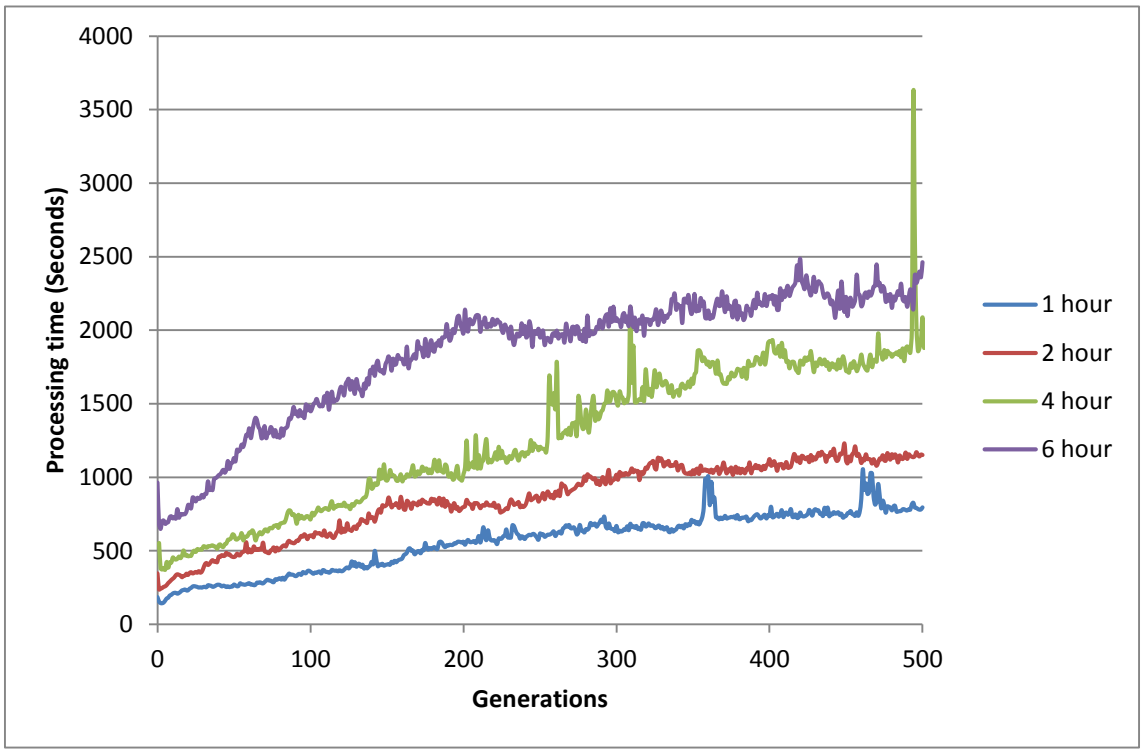


Figure 5.18, Processing time in seconds for each generation on the CPU, which includes all 10 population processed at the in the same batch, for each amount of simulation training time used.

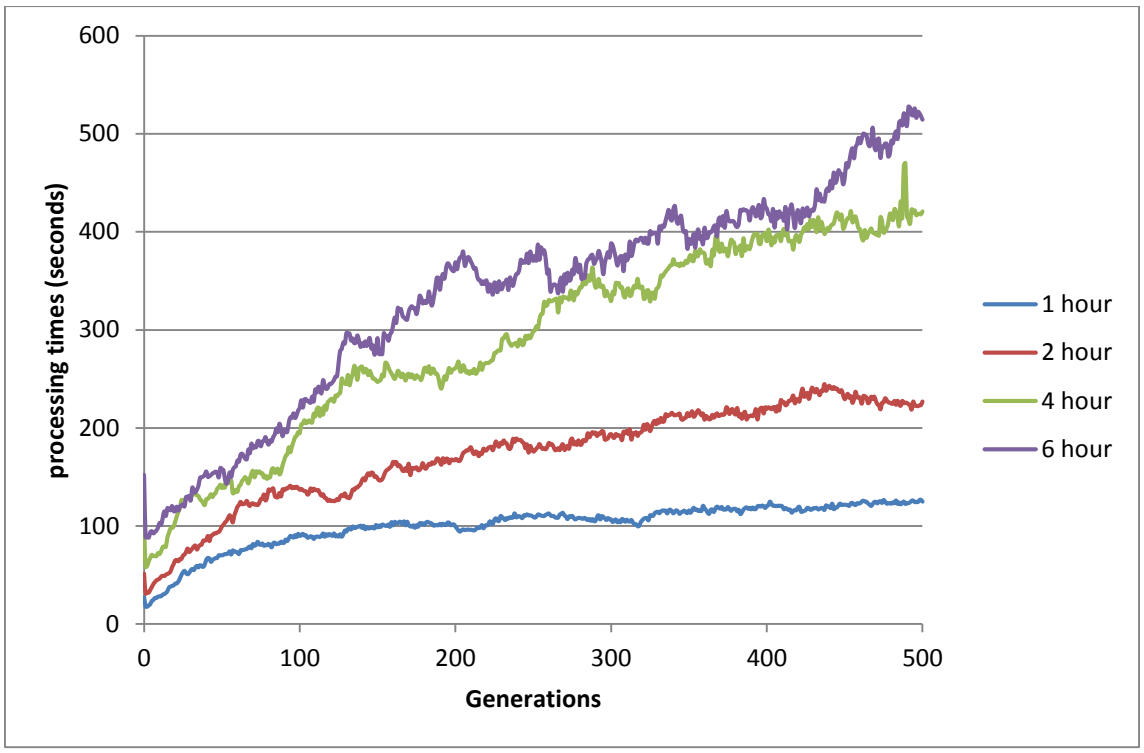


Figure 5.19, Processing time in seconds for each generation on the GPGPU, which includes all 10 population processed at the in the same batch, for each amount of simulation training time used.

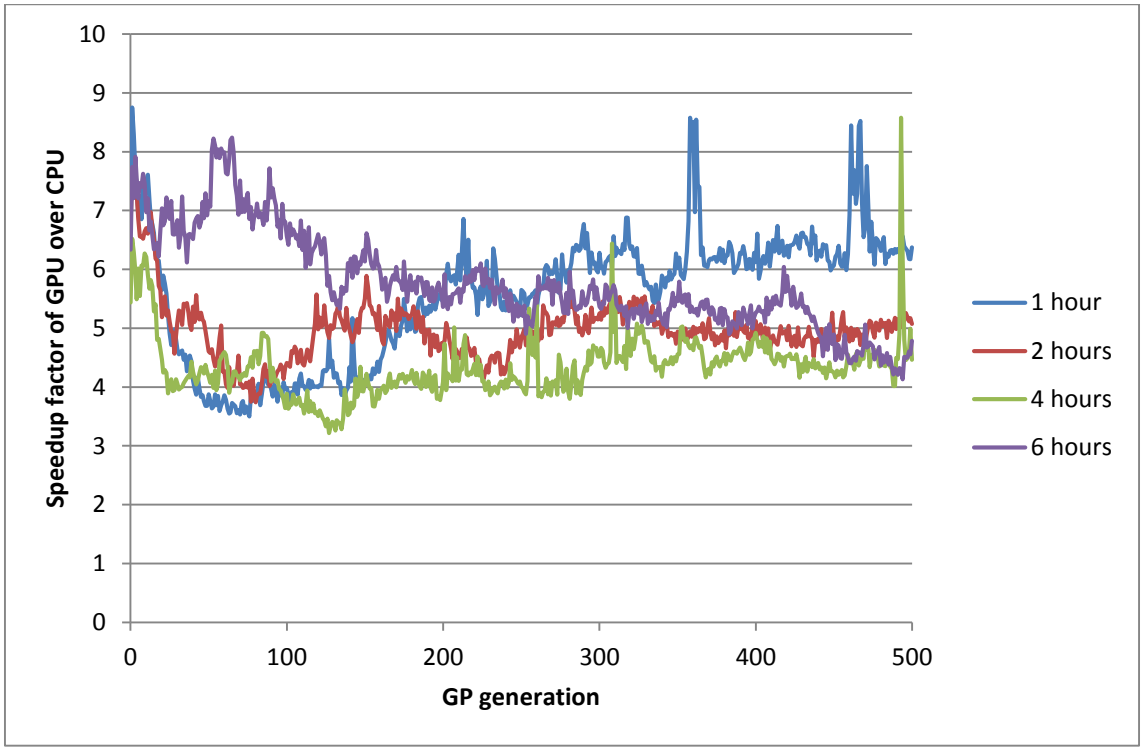


Figure 5.20, Speed-ups of the GPGPU over the CPU runs for each generation (including all 10 populations in each generation).

The processing time of both the CPU (Figure 5.18) and GPGPU (Figure 5.19) increase during the optimisation process, which is attributed to the tendency for the complexity of the GP trees within the population to increase over time. This complexity increase is in both terms of the number of nodes with each GP tree, and their respective computational cost. I.e. with the progression from simple trees to more complex and more accurate in terms of matching the target, it is also expected that there will be an increase in terms of the complexity of the resulting CA simulation. Finally the GPGPU is shown to be an invaluable tool in decreasing the processing time of the training runs, bringing them into a more feasible range, including saving up to 8 days of computational time (Table 5.7).

5.5 Testing of trained GP with fixed spatial and temporal resolution

5.5.1 Introduction

Having trained GP rule sets in the previous section 5.4 on the Ghimire Hill and pond test case for varying amounts of simulation time, it was found that the system could create rules which can match the target pattern competitively amongst human formulations. However, in to ascertain how well this training has captured the underlying general mechanics of fluid movements, this section applies each set of rules to different test cases upon which they were not trained (i.e. unseen data sets).

5.5.2 Experimental setup

Several stages of testing have been utilised, to test if the trained rules are capable of generalising to different initial and input conditions (water levels/depths and terrains). All experiments use a fixed cell size of 50m and time step of 1 second, which on the EAT1 test case is unviable due to poor performance of all human formulations.

5.5.2.1 Remainder of training case

Firstly, the remaining duration of the simulations are used as validation, as this will test how the same rules react with the same terrain but different initial conditions. Since the pattern of the water movements in space and time are

different, this should show how well the rules are learning the underlying mechanics of the fluid flow relative to fluid level/depths and the terrain levels.

In the first experiment the time from t = 0 up to t = 1 hours is used for training, and so then the time from t = 1 hours up to t = 12 hours can be used for testing. This is achieved by starting the simulation from the target examples grid state at the appropriate time. Similarly, where experimentation is conducted with training from t = 0 up to t = 2. Then testing/validation can be performed from t = 2 up to t = 12 hours, but also testing/validation can be carried out on the 1 hour trained versions of this case too, and similarly for 4 and 6 hour trained simulations (demonstrated in Table 5.8).

Table 5.8, Testing time periods applicable to each training case in this section, when using the remainder of the training simulation case for testing.

	t = 1 to t = 12	t = 2 to t = 12	t = 4 to t = 12	t = 6 to t = 12
1 hour training	X	X	X	X
2 hour training		X	X	X
4 hour training			X	X
6 hour training				X

5.5.2.2 Validation on the same terrain with different rain profile

The second stage of testing which has been utilised uses the same ‘hill and pond’ test case terrain, but with an altered rain profile to create a completely different test simulation. This will test the generated rules capability to generalise to different input conditions (i.e. different water level inputs) through the course of the simulation. Also it tests the capability of the rules to operate over a longer simulation period than they were previously trained on. The rain profile for this second test case is altered to 10mm/h for 2 hours, as opposed to 20mm/h for hour. In order to fully and fairly test the variously trained GP batches, all experimental batches are tested on the entire 12 hours of validation simulation, for each of the trained GP candidates from the 1, 2, 4, and 6 hour training. This means that a much fairer comparison of the quality of training from each different training volume can be determined.

5.5.2.3 Testing on a different terrain (EAT2)

The third stage of testing is to train one terrain and test on another. For this purpose, the EAT2 terrain (as described in section 5.3.2) has been utilised, and a rain profile applied, as opposed to prescribed inflow condition, run for 4 hours of simulation time. This finally tests the capability of the rules to generalise to a completely different and unseen terrain and water levels/depths inputs. As with all of the above test/validation cases, this case has been scaled up to 50m cell size, and use a roughness factor of 0.01n, as this was the only training variety provided; however, the training model was re-run with the same parameter for comparison.

5.5.3 Results

5.5.3.1 Remainder of training case

Having trained on the first hour, validation/testing can be performed on the remained of the simulation. This is done by starting the water depths of the grid at the state of the target at the given start time and continuing with the simulation. Test are performed from t = 1, 2, 4, and 6 hours up to t = 12 hours, as shown in Table 5.9.

Table 5.9, Testing time periods applicable to each training case, when using the remainder of the training simulation case for validation, and which table display these results.

	t = 1 to t = 12	t = 2 to t = 12	t = 4 to t = 12	t = 6 to t = 12	Results Table
1 hour trained	X	X	X	X	Table 5.10
2 hour trained		X	X	X	Table 5.11
4 hour trained			X	X	Table 5.12
6 hour trained				X	

Table 5.10, Fitness scores (1/RMSE) for the training case from respective time shown up to t =12, for the 1 hour CPU and GPU trained populations; also showing the maximum and mean fitness for both groups of populations and all GP individuals at each training time. Manning's formulations, Limited, zero and large flows are shown for reference.

		Test simulation start time					
		1	2	3	4	5	6
GPU GP Population	0	242.983	222.117		262.536		319.237
	1	122.791	128.665		136.496		144.012
	2	514.912	282.234		409.822		509.438
	3	499.122	298.614		383.707		478.916
	4	480.172	288.833		361.897		496.174
	5	318.002	215.876		230.741		262.736
	6	31.5126	264.633		327.087		20.8154
	7	256.514	262.871		334.93		378.155
	8	309.404	468.183		380.775		520.5
	9	431.677	377.62		495.966		580.558
CPU GP Population	0	306.06	400.387		390.353		486.266
	1	159.476	163.577		180.122		197.202
	2	116.383	117.634		124.233		130.721
	3	62.6691	60.169		58.5109		58.6517
	4	170.914	162.705		172.757		187.189
	5	177.604	188.866		225.467		257.375
	6	295.239	330.591		464.4		550.758
	7	215.386	243.641		332.008		445.303
	8	214.31	230.633		291.456		366.602
	9	157.638	145.097		153.266		164.54
GPU GP Maximum		514.912	468.183		495.966		580.558
GPU GP Mean		320.709	280.9646		332.3957		371.0541
CPU GP Maximum		306.06	400.387		464.4		550.758
CPU GP Mean		187.5679	204.33		239.2573		284.4608
Combined GP Maximum		514.912	468.183		495.966		580.558
Combined GP Mean		254.1384	242.6473		285.8265		327.7575
Bates-Limited		868.219	1006		1242.04		1298.87
Bates-Manning's		447.986	461.173		517.147		552.1
DT-Manning's		197.104	222.466		287.386		357.138
Ghimire-Manning's		162.991	174.341		215.826		248.954
Zero Flow		31.1719	52.9909		150.501		341.91
large Flow		14.581	15.4065		14.9763		15.3828

It should be noted for these tests that when the starting time of the simulation is increased up to 6 hours, the zero flow GP individual scores increasingly well, up until the validation from t = 6 up to t = 12, where it actually scores better than a number of the Manning's formulations. Clearly most of the water has moved prior to this point in time and the simulation is settling down. However this does represent a good test since it would be hoped that evolved GP programs don't just move water all the time but equally see when water should not move, or at least not move as much and potentially be converged.

CPU population 3 appeared to be trapped in local a maxima within training (shown in Table 5.6) which explains its poor validation scores (shown in Table 5.10). However, it would appear that for GPU population 4 and 7, which also didn't outperform human formulations on training, still generalise well enough outside of the training set. Of the rest that did outperform human formulations on the training set, the GPU populations 1 and partial 6, CPU populations 1, 2, 4 and 9; which would appear to have over trained to the amount of water movement within the first hour of the simulation compared to the latter parts of the simulation. The over training over these populations is indicated as they scored better than Manning's on the training, however these rules perform poorly on the validation (which has low flow). They have scored better on training, but poorly on validation, therefore have either not picked up the underlying rules very well, or have rather concentrated on a rule which performs well on just the training data set. While none of the rules sets on validation perform better than the Bates-limited formulations, the scores for this method are exceptional even amongst the human formulations.

Finally, however, a good number of populations score better than some Manning's formulations on both training and then on the validation areas of the simulation. While the mean of the populations outperforms a number of the Manning's formulations on all the validation cases, on the very last case from t = 6 to t = 12, the Ghimire formulation did not outperform the zero flow. While it does however score very close, this is perhaps an indication that the training, when using t = 0 to t = 1, is heavily weighted in terms of high flows.

Table 5.11, Fitness scores (1/RMSE) for the training case from respective time shown up to t =12, for the 2 hour CPU and GPU trained populations; also showing the maximum and mean fitness for both groups of populations and all GP individuals at each training time.

		Test simulation start time					
		1	2	3	4	5	6
GPU GP Population	0		235.059		272.128		323.151
	1		411.195		423.494		521.056
	2		413.415		419.41		553.95
	3		267.536		320.842		390.146
	4		372.171		396.995		502.418
	5		430.2		526.71		569.206
	6		399.77		398.04		521.101
	7		275.463		312.439		379.203
	8		459.397		441.651		563.695
	9		363.054		376.368		482.266
CPU GP Population	0		434.888		479.6		604.632
	1		256.229		285.209		321.7
	2		413.167		456.649		571.564
	3		165.125		252.042		321.127
	4		281.574		299.01		347.34
	5		270.184		291.356		338.498
	6		403.605		435.937		563.012
	7		196.551		203.697		219.06
	8		277.854		290.224		345.012
	9		387.319		391.244		496.961
GPU GP Maximum			459.397		526.71		569.206
GPU GP Mean			362.726		388.8077		480.6192
CPU GP Maximum			434.888		479.6		604.632
CPU GP Mean			308.6496		338.4968		412.8906
Combined GP Maximum			459.397		526.71		604.632
Combined GP Mean			335.6878		363.6523		446.7549
Bates-Limited			1006		1242.04		1298.87
Bates-Manning's			461.173		517.147		552.1
DT-Manning's			222.466		287.386		357.138
Ghimire-Manning's			174.341		215.826		248.954
Zero Flow			52.9909		150.501		341.91
large Flow			15.4065		14.9763		15.3828

Table 5.11 shows the validation/testing results for those populations trained on the first 2 hours of the training simulation. It can be seen that there appears to be far less cases of over training, and of those that didn't outperform the zero flow. It is noted that by having 2 hours of the training simulation and specifically the first two hours, the rain has fallen during the first one hour and left the next hour for the water flow to be driven by the flow that exist from the previous rain fall. It should also be noted that the peak of concentration for the ponding point in the training simulation occurs at 1h 45mins. Therefore, there is a small amount of training for when the water should be draining away, having been driven primarily by the terrain and gravity during this time.

Table 5.12, Fitness scores (1/RMSE) for the training case from respective time shown up to t =12, for the both the 4 and 6 hour, CPU and GPU trained populations; also showing the maximum and mean fitness for both groups of populations and all GP individuals at each training time.

		4 hour trained			6 hour trained		
		Test simulation start time					
		4	5	6	----------	6	----------
GPU GP Population	0	297.237		349.942		358.4	
	1	337.003		386.793		359.361	
	2	276.24		329.392		449.94	
	3	343.558		441.98		270.686	
	4	389.321		440.139		630.292	
	5	324.718		364.004		383.731	
	6	494.904		557.207		472.983	
	7	464.388		525.182		383.81	
	8	414.911		505.111		548.466	
	9	471.051		529.891		370.66	
CPU GP Population	0	458.938		583.941		643.891	
	1	319.26		381.822		580.413	
	2	296.005		330.114		481.309	
	3	301.921		359.452		548.267	
	4	461.701		558.845		550.742	
	5	388.226		464.427		522.458	
	6	498.279		557.922		550.96	
	7	451.326		576.098		550.96	
	8	409.949		498.648		458.291	
	9	465.151		569.642		624.869	
GPU GP Maximum		494.904		557.207		630.292	
GPU GP Mean		381.333		442.964		422.833	
CPU GP Maximum		498.279		583.941		643.891	
CPU GP Mean		405.076		488.091		551.216	
Combined GP Maximum		498.279		583.941		643.891	
Combined GP Mean		393.204		465.528		487.025	
Bates-Limited		1242.04		1298.87		1298.87	
Bates-Manning's		517.147		552.1		552.1	
DT-Manning's		287.386		357.138		357.138	
Ghimire-Manning's		215.826		248.954		248.954	

In the cases that were trained on the first 4 hours, and 6 hours of the training simulation (Shown in Table 5.12). There can be seen a much better response, however it can be argued that these populations where trained at a closer time frame to these test cases. This does show that these validation cases, testing the remaining end of the simulation may be primarily testing the rules ability to predict rather low flow. There are a very few cases that are out performed by the zero flow, comparatively far less than on the 1 and 2 hour trained simulations.

5.5.3.2 Validation on the same terrain with different rain profile

For the next stage of testing, runs are conducted on the training simulation terrain, and evenly for the full period from t = 0 to t = 12 hours, but with different rain conditions of 10mm/h for 2 hours, as opposed to the training/validation case which utilised 20mm/h for 1 hour. Therefore, the entire hydrograph produced from each cell will be different from the training case used. The scores for Manning's formulations and limited, zero flow and the arbitrarily large flow are shown in Table 5.13, and the comparable score for all the GP population in Table 5.14. Compared to previous validation testing, all batches of trained GP are now tested on the same test case, which makes for easier comparison.

Table 5.13, Fitness scores (1/RMSE) for the Manning's formulations and limited, zero flow, and large flow (1,000) on the entire validation case, using the hill and pond terrain but modified rain profile.

GP Program	Fitness score
Bates-Limited	865.414
Bates-Manning's	478.223
Dottori and Todini-Manning's	199.433
Ghimire-Manning's	174.179
Zero Flow	16.3422
large Flow	15.3644

Table 5.14, Fitness scores (1/RMSE) for the validation case from t = 0 up to t = 12, for the CPU and GPU trained populations trained at the respective length on the training simulation; also showing the maximum and mean fitness for both groups of populations and all GP individuals at each training time.

		Hours of training/simulation					
		1	2	3	4	5	6
GPU GP Population	0	255.212	252.483		265.789		305.702
	1	134.659	441.873		318.972		304.702
	2	259.051	459.783		267.496		350.233
	3	275.922	264.809		334.722		141.821
	4	277.938	362.204		340.743		530.114
	5	225.772	324.701		331.3		313.089
	6	265.384	474.943		429.314		401.914
	7	231.443	323.025		416.414		304.483
	8	246.8	499.648		424.842		364.654
	9	337.264	397.023		372.259		334.973
CPU GP Population	0	320.472	365.137		368.623		443.5
	1	171.307	279.395		315.586		411.474
	2	124.015	462.718		294.518		350.541
	3	67.9683	183.087		294.403		432.707
	4	177.452	283.497		390.887		454.383
	5	185.511	261.447		325.516		364.497
	6	280.06	406.704		454.23		374.495
	7	226.605	200.812		411.811		452.153
	8	226.483	289.061		388.723		262.063
	9	164.763	442.018		438.915		478.182
GPU GP Maximum		337.264	499.648		429.314		530.114
GPU GP Mean		250.9445	380.0492		350.1851		335.1685
CPU GP Maximum		320.472	462.718		454.23		478.182
CPU GP Mean		194.4636	317.3876		368.3212		402.3995
Combined GP Maximum		337.264	499.648		454.23		530.114
Combined GP Mean		222.7041	348.7184		359.2532		368.784

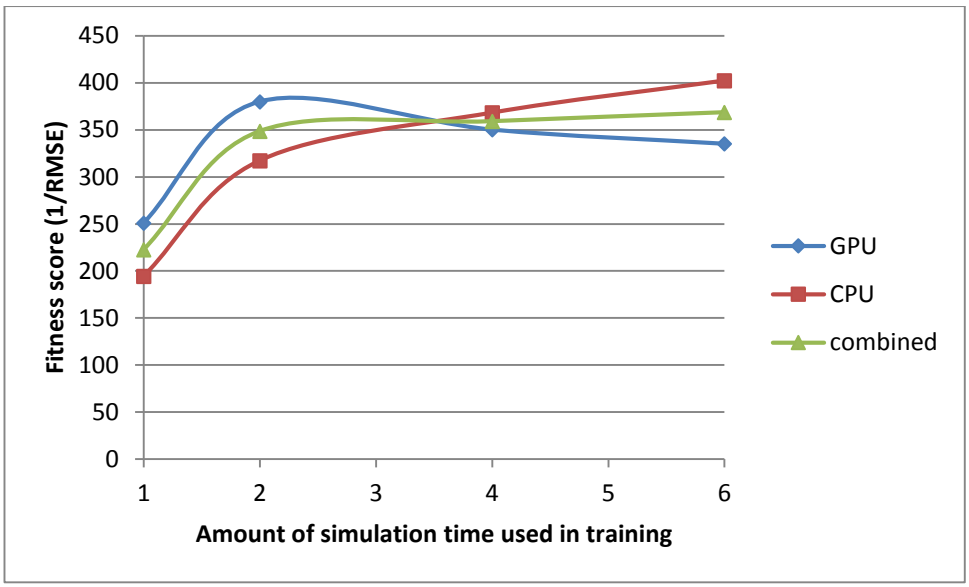


Figure 5.21, Mean fitness score (1/RMSE) of the GPGPU, CPU, and both combined runs for varying amounts of simulation time used for training, for the hill and pond test case with the altered rain profile (From Table 5.13).

It can be seen from Figure 5.21 and Table 5.14 that more populations have failed to capture the underlying dynamics with a shorter training period of just the first hour. However, a large number still manage to exceed the score of the lower scoring manning's formulations on this unseen case or come very close. It can be seen from the average maximum scores (shown in Figure 5.21), that out of the length of training cases that have been utilised, clearly 1 hour training would appear to lack sufficient example for the system to generalise well. After this point there is a minor disagreement between the CPU and GPU scores, although both are also examined together this would appear to indicate that from 2 hours' worth of simulation time onwards it does not increase the generalisation much if at all. Clearly this gives the water enough time to concentrate and be drawn down, which gives the GP optimisation enough training example. After this point it would seem to be a case of diminishing returns, since it takes a directly proportionate amount of time to run the GP optimisations as the amount of training simulation used.

5.5.3.3 Testing on a different terrain (EAT2)

For the final stage of validation, the terrain for the UK Environment Agency Test second case (EAT2) is utilised, which is a much larger and a different terrain configuration compared to the training case. The terrain has been scaled up to a 50m cell size, and used a roughness factor of 0.01n, a rain profile at 40mm/h for

an hour has been applied, and the simulation run for 4 hours' worth of simulation time. The scores for the Manning's formulations, limited, zero flow and large flow are showing in Table 5.15, and the GP population for both CPU and GPU are shown in Table 5.16.

Table 5.15, Fitness scores (1/RMSE) for the Manning's formulations, limited, zero flow, and large flow (1,000) on the entire validation case, using the EAT2 terrain scaled up to 50m, with 0.01n roughness factor, and a rain profile of 40mm/r for the first hour; simulation was run up to t = 4 hour.

GP program	Fitness score
Bates-Limited	332.333
Dottori and Todini-Manning's	297.973
Bates-Manning's	281.595
Ghimire-Manning's	254.318
Zero Flow	14.3493
large Flow	12.6236

Table 5.16, Fitness scores (1/RMSE) for the EAT2 scaled to 50m, and 0.01n roughness factor (with 40mm/h rain for first hour) validation case from t = 0 up to t = 4, for the CPU and GPU trained populations trained at the respective length on the training simulation; also showing the maximum and mean fitness for both groups of populations and all GP individuals at each training time. Those highlighted bold have outperform the Manning's formulations.

		Hours of training/simulation					
		1	2	3	4	5	6
GPU GP Population	0	**364.532**	**415.567**		209.903		189.762
	1	122.791	**498.207**		292.786		**369.183**
	2	**514.912**	**617.317**		298.058		**359.934**
	3	**499.122**	217.652		**420.556**		95.2937
	4	**480.172**	**510.179**		264.763		202.902
	5	318.002	178.124		243.873		**385.266**
	6	31.5126	172.421		173.98		162.58
	7	256.514	**518.85**		**340.576**		**337.502**
	8	309.404	**564.988**		**471.42**		134.292
	9	**431.677**	**568.708**		278.359		150.337
CPU GP Population	0	**457.02**	289.122		259.168		305.981
	1	**370.823**	275.54		216.192		**432.401**
	2	106.013	**708.808**		257.238		170.561
	3	235.649	186.953		161.221		295.629
	4	159.431	**349.744**		**366.243**		95.2969
	5	**396.31**	312.775		**397.11**		**426.365**
	6	**374.749**	**581.848**		**333.474**		215.428
	7	195.095	149.006		**504.691**		**406.521**
	8	**389.258**	**348.501**		**400.064**		229.293
	9	148.462	286.786		**495.113**		126.596
GPU GP Maximum		**514.912**	**617.317**		**471.42**		**385.266**
GPU GP Mean		**332.8639**	**426.2013**		299.4274		238.7052
CPU GP Maximum		**457.02**	**708.808**		**504.691**		**432.401**
CPU GP Mean		283.281	**348.9083**		**339.0514**		270.4072
Combined GP Maximum		**514.912**	**708.808**		**504.691**		**432.401**
Combined GP Mean		308.0724	**387.5548**		319.2394		254.5562
Pass/fail rate		10/10	11/9		9/11		7/13

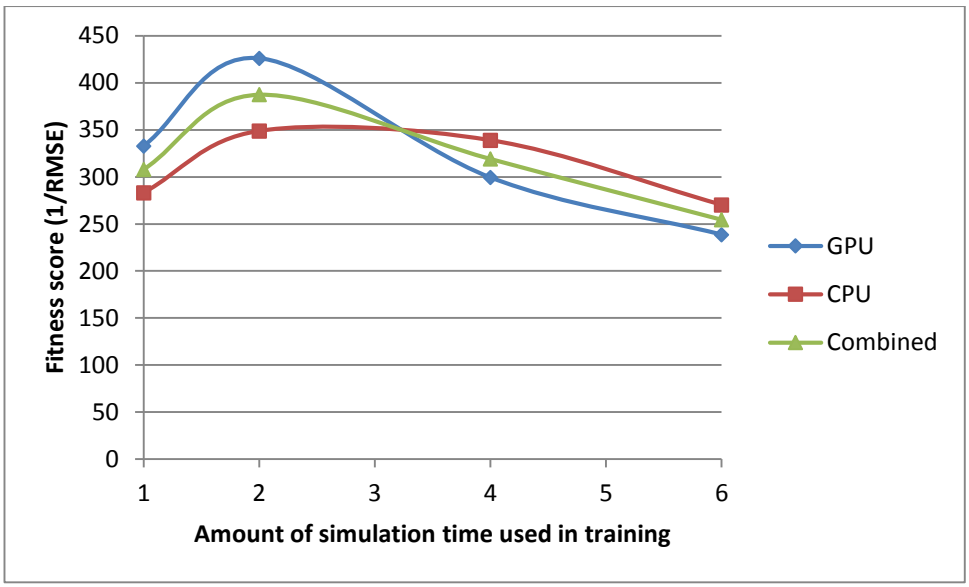


Figure 5.22, Mean fitness score (1/RMSE) of the GPGPU, CPU, and both combined runs for varying amounts of simulation time used for training, for the EAT2-rain test case.

In this case (Shown in Table 5.16 and Figure 5.22) there are a number of additional populations which don't perform very well in testing, showing their over training. The validation scores certainly show a peak with 2 hours of training simulation provided, and on this validation case even show a marked drop in performance with longer periods of training simulation time provided. However, this might be explained by the length of the test cases at only 4 hours long, where the training cases length of training and the testing simulation length maybe playing a role. Although this could also be explained by over training of the 6 hours trained GP trees to the longer draining down period of the training simulation. It is hard to say without further investigation or greater volume of test cases. Considering the test cases that have been utilised in this thesis, it appears more likely that these longer runs might be over trained.

5.6 Conclusions

While it appears easier for the GP system to optimise and gain high scores on very short amounts of simulation time/CA iterations presented (Section 5.4), in this section testing on unseen test cases shows that those GP trained with the shorter simulation times of 1 hour do not capture the underlying dynamics as well others. However, there is not a large increase in testing performance after 2-4 hours of training simulation time presented, and even in the last validation case a marked drop in performance. When the processing time of these large amounts

of training data being provided to the system is also considered, then there appears to be a marked point of diminishing returns at approximately 2-4 hours. It is thought that this is due to the difference in the flows, as the point of concentration is at approximately t = 1h and 45mins, and that sufficient example of both the flows before and after this point are needed in order to generalise well. Therefore, considering the additional computational times required to provide more simulation time during training, and the peak in generalisation performance at 2-4 hours of simulation, this represents and optimal amount of simulation time for training. In most conditions the majority of (average) rules out perform the Ghimire, Dottori and Todini rules sets, and on the last test case of EAT2 a number outperform all rule sets including the Bates formulation.

Chapter 6: GP CA real-world flood modelling generalisation to spatiotemporal resolution

6.1 Introduction

It is shown in the last chapter (Chapter 5:) that the GPCA system can train rules at a single spatiotemporal resolution, and that given sufficient amounts of training data it can generalise to other input conditions on the same single spatiotemporal resolution. The work in this chapter investigates the effects of varying the spatiotemporal resolutions during training and testing of the GPCA system. The same GPCA methodology is used as in Chapter 4: and Chapter 5:.

6.1.1 Chapter Structure

The work in section 6.2 investigates the effects of temporal generalisation while still maintaining a single static spatial resolution during all experimentation. Firstly the GP trained and tested in Chapter 5: is tested upon a range of different temporal resolutions. Then the GPCA system is trained and tested on a set of different temporal resolutions. Here an investigation of the GP bloat during the processing is also carried out. Section 6.3 further investigates the generalisation of these newly trained GP to include other input such as different terrains, water level inputs and finally different inflow types, in order to find the limits of this generalisation. By creating rules which can generalise to different terrains, water level inputs as well as different temporal resolutions, this work tackles the trade-off between processing time and accuracy, by creating rules which operate at larger time steps and maintain accuracy levels higher than a number of human implementations. This work is extended to cover the full breadth of this problem, whereby some human implemented rules drop their performance in terms of the largest time step at which they produce acceptable accuracy, as the spatial size of cells are decreased. Therefore section 6.4 investigates the effectiveness of training the GPCA system on a set of different spatial and temporal resolutions, and their competitiveness with the very latest human models in this respect. Finally section 6.5 draws conclusions from the process of training and testing the GPCA system with variable spatiotemporal resolutions, about its advantages and limitations.

6.2 Training GP for temporal generalisation of CA rules

6.2.1 Introduction

The previous chapter's experimentation (section 5.4) has concentrated on training, and then testing (section 5.5) upon a single set of the grid static variables (Cell size, roughness factor, and time step). However, the human formulated rules are capable operating at different temporal and spatial resolutions (time step and cell sizes). Also a large body of literature exists on how well each model operates at different combinations of cell size and time step (Section 2.1.4). Each rule attempts to model the same physical reaction but at different temporal scales (I.e. each rule relates each cell to a spatial scale (cell size) and each iteration to a temporal scale (time step), in the real world). This shows how much learning and research work has been performed by humans in order to formulate these rules, and how a lot of this learning/research work has been directed towards relating each cell to a spatial area and each iteration of the grid states to a temporal area. This section examines how trade-off of accuracy against processing time affects the training of CA rules, given variables discretisations of time (time step) and a static spatial resolution.

The reasoning for such a need for rules to adapt to different spatial and temporal scales is because the processing time of a simulation is directly related to the number of iterations and the number of cells of the CA. Where the number of iterations is inversely proportional to the temporal resolution (time step), and the number of cells is inversely proportional to the spatial resolution (cell size) in each dimension. It is possible to consider the same area of real time, and the same real area of space in the simulation but at very different CA resolutions, and therefore it is possible to model the same simulation and decrease the processing time either by increasing the time step, or increasing the cell sizes. Therefore, there exists a fairly well understood trade-off between the processing time of the simulation and the possible accuracy. Again considering the real world event, the CA can only model movements of up to one cell at any given time step (CA iteration). Therefore, the question is how well the given rule can approximate the 'real' water levels given its spatial and temporal resolution.

The target could of course be actual real world data, however volumes of real world data with a scale of conditions is not feasible to capture. Therefore, a reasonably trusted model must be utilised to create the target data, and a large part of the trust element comes from running the target model at a very small time step in order to produce very accurate underlying data. Within this thesis the target UIM model was run with an adaptive time step, and a minimum value of 0.0125 seconds, and a sampling rate of 1 second. Once the continuous shape of movements of water levels (state changes) can be represented through time, then a comparison can be made using different discretisation schema of time.

The work of Dottori & Todini [66], has showed that the use of the Manning's formula and discharge formula have a maximum time step, after which the viability of the rule begins to degrade. Sometimes this is called an "explosion" [63], or 'wild/checkerboard oscillations', or 'artificial diffusion'. This is due to the discharge formula part of the schema, which indicates a direct proportionality of the flow to the time step. This means that there is a direction relation between the maximum time step, and an inverse relation to the velocity of the lateral fluid flows at which these formulations will operate successfully (section 2.1.4.1). After a critical time step is reached then at least one cell within the simulation will cause a disproportionate outflow, which in turn causes larger and larger feedback, and the observed oscillations which destroy the overall quality of the simulation.

It is slightly less than clear exactly how the rules adapt, when the time step is altered due to the complex nature of the system. I.e. altering the time step within the rules operable area changes the flow rates in the very initial iterations of the algorithm. However, under successive iterations of the CA algorithm, the state transition rule accepts the altered water levels and produces further altered flow rates, such that overall very similar water levels are produced over the spatial and temporal area of the simulation. Therefore, the resulting simulation is effectively the same/similar resulting output up until an excessively large time step is used. At this limiting point, at least one flow within the simulation in space and time will be excessively large to the point whereby other cells using the Manning's formula on successive iterations, cannot cope with the larger flow, and only serve to exacerbate the problem. This is especially true, as a very large flow will empty a cell completely of water volume, and that flow rate being limited by the main cell's water volume will be relatively less than prescribed by the

Manning's formula to maintain the desire global actions. Secondly, since that cell emptied of water, it would be left completely dry while its neighbours now certainly have some water; they will certainly flow water back into the original cell in the next time step; starting the checkerboard style of oscillations.

6.2.2 Human formulations and static temporal resolution trained GP performance.

The fitness function can clearly detect this point (Figure 6.1), where the fitness scores of the human formulated rules are tested on 4 hours of the Ghimire hill and pond test case, each at varying time step factors. Each single simulation is run with a static time step, but a different simulation is processed at each time step. The Manning's formulations (Ghimire, Dottori & Todini, and Bates) drop dramatically between 1-3 seconds and onwards. The Bates Limited formulation is designed to extend this limiting time step and operates successfully at far higher time step factors. Also shown in Figure 6.1, are the average fitness scores of the GP trained in section 5.4 on only a one second time step. Since previous training examples only provided a single example for each of the static variables including the time step at 1 second, it can be expect that the system will have over trained/generalised to the given training case, which is demonstrated in Figure 6.1. Where clearly these rules generated have indeed over train/generalised to the single training example of 1 second, especially as their performance decrease as the time step is also decreased, where the Manning's formula (and discharge formula, combined schema) maintain their performance. Both the Manning's formulations and the GP tree trained on a single second all fall in performance after 1 second up to 4 second time steps. Whereas the Bates limited formulation holds its fitness up to a much larger time steps. Overall this demonstrates that as expected, the GP overfits to the time step on which it has been trained and in this window, it is competitive with the Bates Limited formulation. However, outside of this window, performance decreases rapidly.

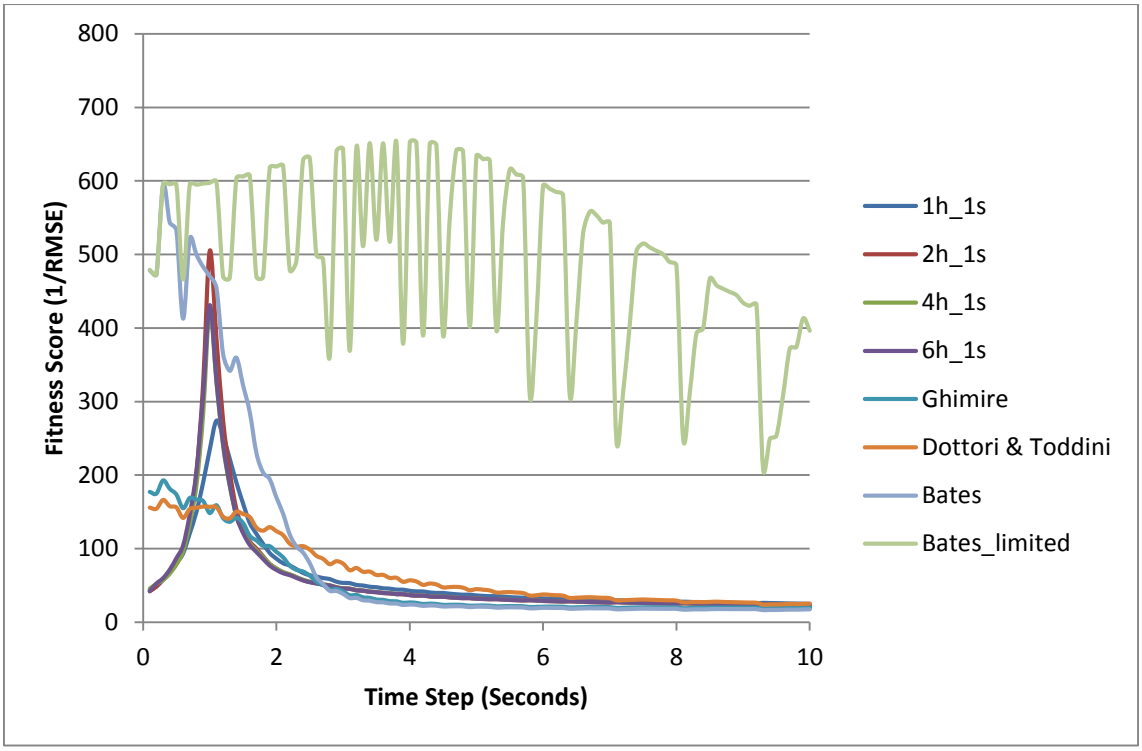


Figure 6.1, Fitness scores the Manning's formulations (Ghimire, Dottori and Todini, Bates) and the limited Bates formulation, along with the fitness scores for the previous trained GP populations (average of all best individuals, all trained at 1 second time step, for 1, 2, 4 and 6 hours of the training simulation). Results shown for the Hill and pond test case for the 4hours, at various time steps from 0.1 seconds up to 10 seconds, at intervals of 0.1 seconds.

6.2.3 Experimental setup

Experimentation has been conducted providing training variation in the time step variable used. Where each simulation maintains a static time step for the duration of the simulation, and therefore a number of different simulations are processed for same time period but each using a different time step, 0.5 seconds, 1 second and 2 seconds. The fitness is established as the reciprocal of the mean of various simulations RMSE. As the experimentation in section 5.4, established that a minimum amount of simulation time required was 2-4 hours, and therefore the hill and pond training case is again utilised for 4 hours with 0.5 seconds, 1 second and 2 seconds time steps.

6.2.4 Training results

The resulting fitness's on the training cases, along with the fitnesses of the human competitors are displayed in Table 6.1. Note that there is difference between the fitness score of each individual with multiple test cases, and the

mean of the fitness, as the fitness is the reciprocal of the mean of the errors, as opposed to the mean of the reciprocal of each error for each simulation.

Table 6.1, Fitness scores (1/RMSE of all cells in all time steps) of the Manning's formula and a the GP populations trained with a 0.5, 1, and 2 second time step; run on the hill and pond test case for 4 hours of simulation time. Also shown are the fitness scores of the Manning's formulations and Limited on the same simulation time and time steps.

			Time step simulation fitness		
GP population	Mean fitness	Fitness score	0.5	1	2
0	**882.779**	881.8331786	**868.448**	**923.468**	**856.421**
1	358.4483333	358.2186513	371.321	350.857	353.167
2	452.9616667	450.5610854	483.948	466.533	408.404
3	463.8523333	462.1535741	496.371	466.96	428.226
4	316.0313333	314.0722402	283.157	341.783	323.154
5	400.8396667	382.0149379	520.581	377.078	304.86
6	297.6156667	297.5704827	302.82	294.833	295.194
7	259.6966667	259.351231	270.031	261.797	247.262
8	375.994	373.5759816	382.584	408.66	336.738
9	297.6023333	229.3584866	363.93	193.57	193.57
GP maximum	**882.779**	881.8331786	**868.448**	**923.468**	**856.421**
GP mean	410.5821	400.8709849	434.3191	408.5539	374.6996
Bates-Limited	603.947	603.748513	594.97	597.408	619.463
Bates	391	303.2317069	532.58	470.673	169.746
DT	145.47	143.707376	155.945	156.77	123.694
Ghimire	143.761	133.9718724	173.442	162.287	95.5546

Interestingly now three examples of time step are provided, in having 3 whole simulations of 4 hours, a large number of the GP individuals have beaten some of the Manning's formulations on the training data sets, this excludes the Bates formulation. It is also interesting to note that in the majority (mean fitness score) of cases, GP individuals do have slightly lower score at the higher step, but in a much less respect than the Manning's formulations. Clearly the fitness scores are closer in performance in terms of generalisation to the Bates limited formulation, and one individual even exceeds these scores, on the training data set. This individual population (GP population 0), has performed exceptionally well and has exceeded the scores of all human formulations, on all time step settings tested here. The results in Figure 6.2 (which display the mean and best individuals trained on multiple time steps), contrasted to those of Figure 6.1 which

show those trained on the single time step, clearly demonstrate how those trained on a single time step have over generalised, and those trained with even a sparse a number of time steps, 3 points (different time steps) generalise much better with less dramatically reduced performance outside of the training time steps.

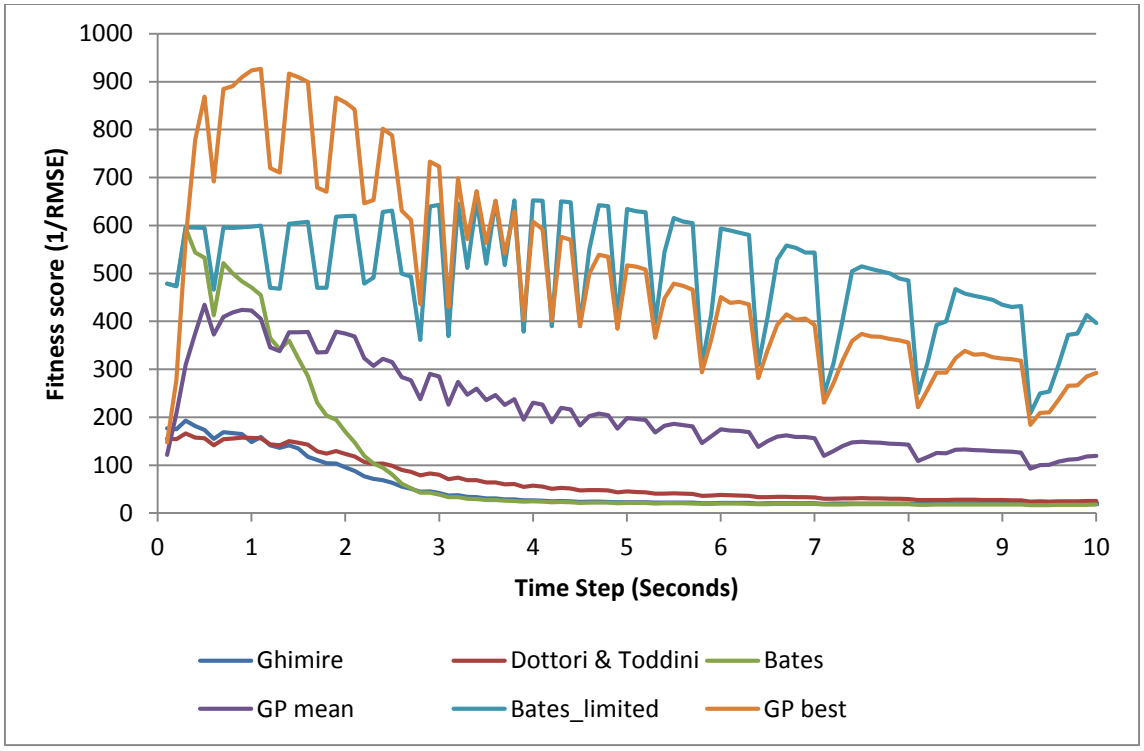


Figure 6.2, Fitness scores of the Manning's formulations (Ghimire, Dottori and Todini, Bates) and the limited Bates formulation, along with the fitness scores for the trained GP populations (all trained at 0.5, 1, and 2 second time step, for 4 hours of the training simulation), showing the best individual and the mean of all the 10 best individuals. Results are shown for the Hill and pond test case for the 4hours, at various time steps from 0.1 seconds up to 10 seconds, at intervals of 0.1 seconds.

6.2.5 GP bloat Results

Finally the traces of the fittest individual within each of the 10 populations are shown in Figure 6.3, are contrasted against the depths of the GP trees (shown in Figure 6.4), and the number of nodes in Figure 6.5.

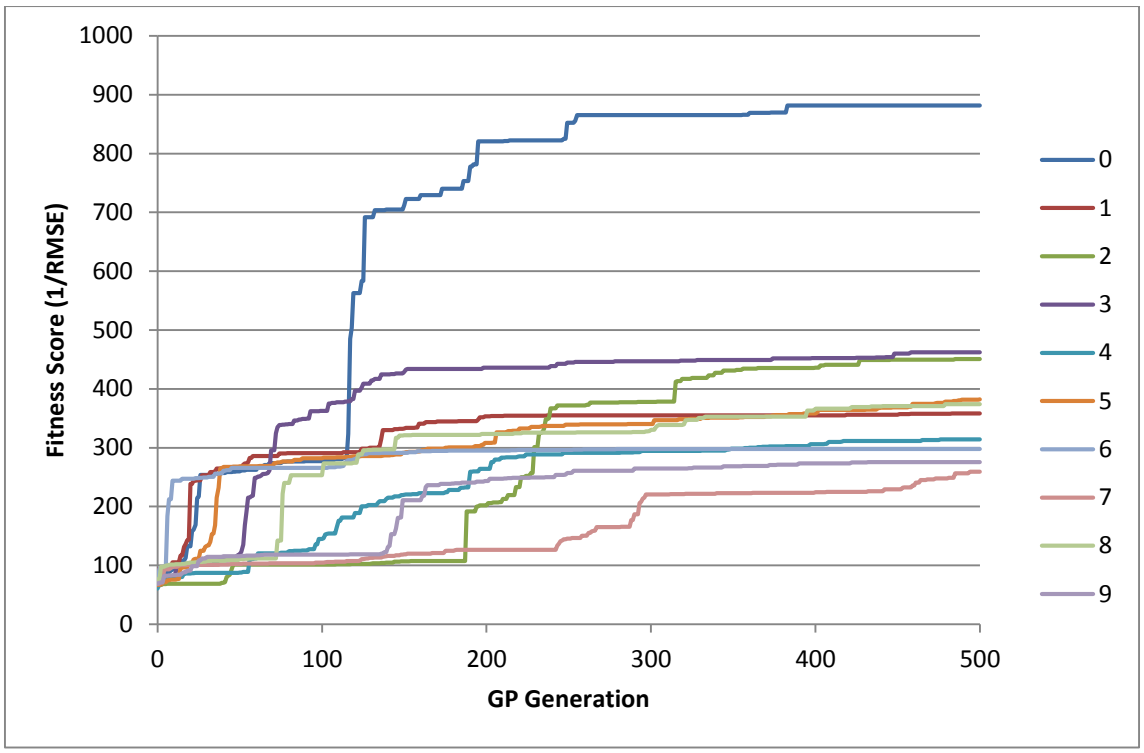


Figure 6.3, Fitness of the fittest individual within each of the 10 populations, trained on hill and pond test case at 50m cell size, and 0.5, 1, and 2 second time steps.

It can be seen in Figure 6.3 that the best fitness individuals take very different routes through the fitness landscape. While in Figure 6.4, it can be seen how within very few generations the maximum depth of 10 is reached. While in Figure 6.5, the number of nodes increases reasonably rapidly during this period, it is nowhere near a full tree for these depths. The number of nodes increase, does then slow down after the maximum depth is reached, however both the fitness and the number of nodes continue to slowly increase after this point. This shows firstly how the bloating preference in our system is for rapid increases in GP tree depth growth, secondly however the GP system is able to continue to optimise within this constraint, and therefore must be able to re-organise the GP tree to accommodate new nodes. Finally the limiting factor of a maximum GP tree depth, leads to a majority of very ‘long, thin’ GP trees in comparison to the maximum size (number of nodes) of the tree at these depths (shown in Equation 6.2).

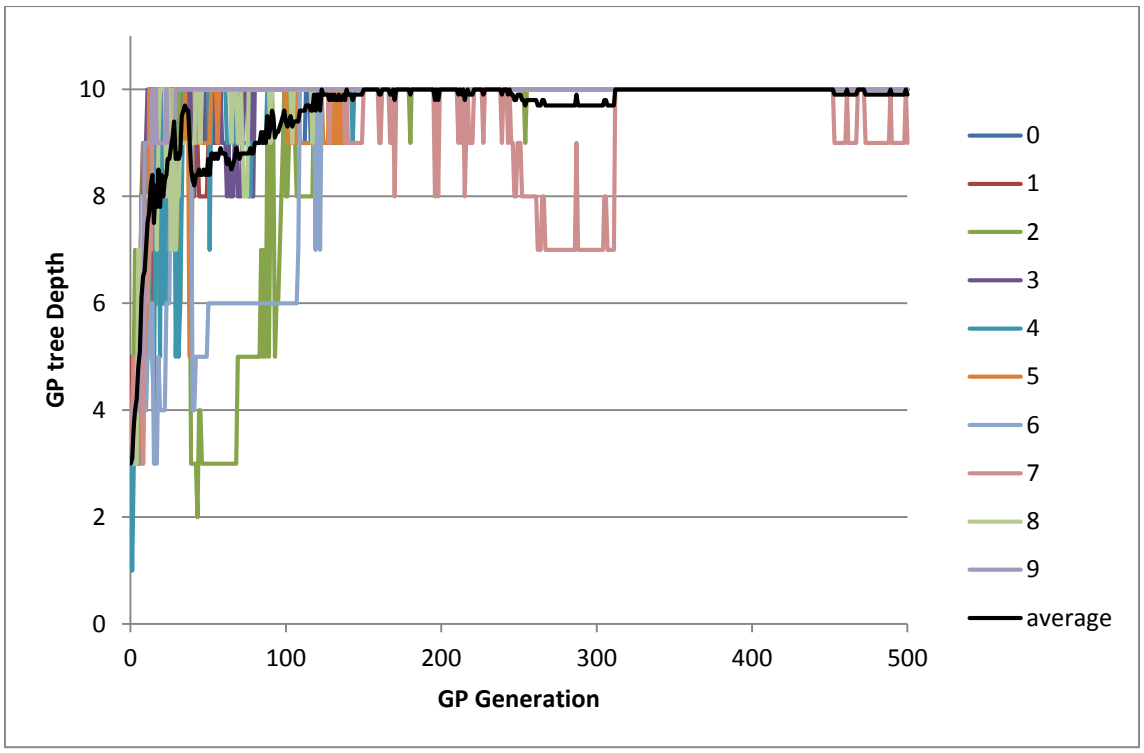


Figure 6.4, Depths of the fittest individual within each population and the mean of each of these 10 individuals at each generation of the optimisation process.

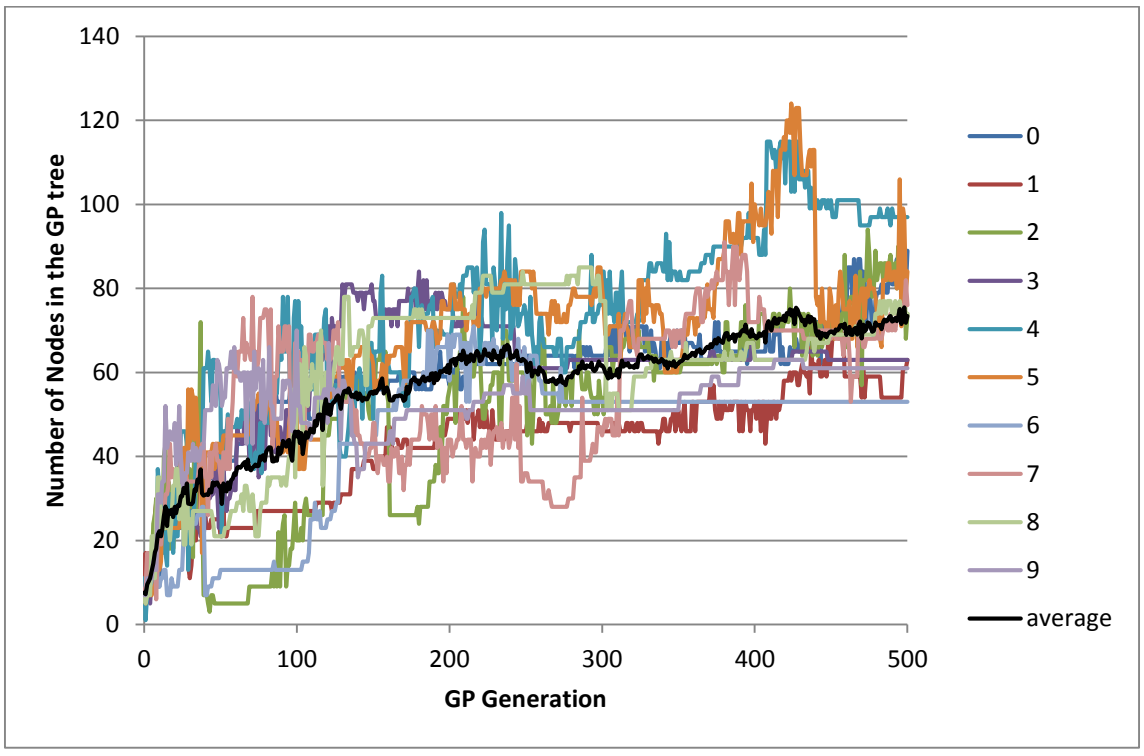


Figure 6.5, Number of nodes within each of the fittest GP tree for each of the 10 populations and the mean of these is displayed in black.

Where experimentation was conducted with larger maximum GP tree depths, it was found that much larger number of nodes GP tree would be formed, although these where still a fraction of the total available size of GP tree's. Equation 6.2 demonstrates the maximum number of nodes for a binary tree at a

given depth, and Equation 6.2 the maximum for a ternary tree system. The calculated maximum number of nodes in a binary or ternary tree are shown in Table 6.2.

Equation 6.1 $$\sum_{i=1}^{n} 2^{(i-1)} = 2^n - 1$$

Equation 6.2 $$\sum_{i=1}^{n} 3^{(i-1)}$$

Table 6.2, Maximum number of nodes possible for full GP trees at each depth, for both binary and ternary trees.

GP Tree depth	Maximum number of nodes of binary tree	Maximum number of nodes of ternary tree
1	1	1
2	3	4
3	7	13
4	15	40
5	31	121
6	63	364
7	127	1093
8	255	3280
9	511	9841
10	1023	29524

Our experimentation has only a single ternary operator, and therefore is heavily biased towards the maximum size of binary trees. Often terminal values are applied early in tree branch, which reduces the number of available nodes. I.e. it is unlikely to find trees near the maximum size, as there is no room for the tree to change shape, and these trees are dominated by operators. In fact, during experimentation where tree sizes are initiated at much larger depths, the trees first begin to shrink down before re-joining this type of growth behaviour. This is thought to be due to two main factors, firstly the mutation operators, especially the growth operators, which has a 50% chance of depositing terminal values as opposed to operators during its growth. Secondly the cross-over operators, where both these operators can be very destructive to trees which are near the

maximum size, they can also be the primary source of growth for small trees. This is especially where the maximum depth is not limited, the cross-over operator is capable of adding very large amount of nodes to GP trees.

6.2.6 Conclusions

In this section it has been shown that providing a single static and grid global variable (e.g. time step) across training will result in over trained rules to that specific variable. By providing even a small number of separate simulations with different static and grid global variables, this can then provide the system with the additional information required to begin to generalise across different time step values. Considering the range of possible time step values, training in this section has used a very sparse number of data points (only 3 time step values of 0.5, 1, and 2 second) in a very small range. Yet the generated rules appear to generalise well to a wide range of time step settings as shown in Figure 6.2 is very encouraging. It is thought this is due to interrelation of the time step and the flow rates within each CA simulation. While testing is limited to 500 GP generations within these experiments so as to make fair comparison between the different lengths of training simulation, in Appendix 9.3 much longer training is carried out on the hill and pond test case at 50m cell size, with 0.5, 1, and 2 second times steps on 4 hours of training simulation time, up to 2,500 GP generations, in order to test how effective this limiting termination criteria has been on these experiments.

6.3 Testing GP trained for temporal generalisation of CA rules

6.3.1 Introduction

For the GP trained in the previously section 6.2, it would appear that a greater generalisation has been achieved, in terms of the fitness scores at different time step factors, by providing the extra training cases each with a different time step variable. However this is primarily in the area of the larger time steps, and the average of the GP results in Figure 6.2, shows this is not always as good performance as the Bates-limited formulation.

However, in order to claim fully that we have not reduced the generalisation of these rules to variation in the rain profile (water depth inputs), and terrain

variation, similar validation experimentations are conducted on the trained GP as in section 5.5. Therefore, the GP are tested on the Hill and pond simulation for the remainder of the training case simulation from t = 4 hours up to t = 12hours, with a different rain profile for a full simulation, and a completely different terrain (EAT2) for 4 hours of simulation time.

Lastly, the previous testing has only investigated test cases which used a uniform rain input condition, and therefore excluded EAT1 test case, and the inflow variation of EAT2. This section also investigates the relative performance of the human rules and those trained with multiple time step inputs. All of these test cases have been scaled to use the same spatial resolutions so as the only variation in grid global static variables is that of the time step.

6.3.2 Experimental setup

Once again (similarly to 5.5) several stages of validation have been utilised, to test if the trained rules are capable of generalising to different initial and input conditions (water levels/depths and terrains). However these tests are performed on those trained on a number of different temporal resolutions of simulation, from section 6.2). All experiments use a fixed cell size of 50m, however tests are conducted firstly on the 3 simulation time steps, and these test cases are detailed in the sections below.

6.3.2.1 Remainder of training case

Firstly, the remaining elements of the training test case (Ghimire, hill and pond case) are used for testing, as this will test how the same rules react with the same terrain but different initial conditions. Since the pattern of the water movements in space and time are different, should show how well the rules are learning the underlying mechanics of the fluid flow relative to fluid level/depths and the terrain levels.

As only 4 hours of training simulation was selected for training purposes in section 6.2, then the remainder of the hill and pond simulation extends from t = 4 hours up to t = 12 hours This is achieved by starting the simulation from the target examples grid state at t = 4 hours, and proceeding up to t = 12hours.

6.3.2.2 Testing on the same terrain with different rain profile

The second stage of testing which has been utilised uses the same 'hill and pond' test case terrain, but with an altered rain profile to create a completely different test simulation. This will test the generate rules capability to generalise to different input conditions (i.e. different water level inputs) through the course of the simulation. Also it tests the capability of the rules to operate over a longer simulation period than they were previously trained on. The rain profile for this second test case is altered to 10mm/h for 2 hours, as opposed to 20mm/h for hour.

6.3.2.3 Testing on a different terrain (EAT2) with uniform rain input

As the third stage of testing the EAT2 terrain (as described in section 5.3.2) has been utilised, and a rain profile applied, as opposed to the prescribed inflow condition, run for 4 hours of simulation time. This tests the capability of the rules to generalise to a completely different and unseen terrain and water levels/depths inputs. As with all of the above test/validation cases, this case has been scaled up to 50m cell size, and use a roughness factor of 0.01n, as this was the only training variety provided; however, the training model was re-run with the same parameter for comparison.

6.3.2.4 Testing on a different terrain (EAT1) with inflow conditions

The fourth stage of testing, utilises the EAT1 test case (shown in section 5.3.2.2), which has lateral inflow condition, which is radically different from the uniform rain input condition used for both training and testing previously. Tests are conducted on the human formulation over a range of time steps, to show a range of time step values where it might be feasible to model. Tests are conducted with the trained GP trees are the 3 time steps upon which they were trained.

6.3.2.5 Testing on a different terrain (EAT2) with inflow conditions

The fifth and final stage of testing, utilises the EAT2 test case (shown in section 5.3.2.1), and uses a lateral inflow condition, which is radically different from the uniform rain input condition used for both training and testing previously. Tests are conducted on the Human formulation over a range of time steps, to

show a range of time step values where it might be feasible to model. Tests are conducted with the trained GP trees are the 3 time steps upon which they were trained.

6.3.3 Rain condition results

6.3.3.1 Remainder of the training simulation validation

The first stage of validation, runs the trained GP on the remainder of the Hill and Pond simulation, i.e. The grid of water depth states are started in the state of the simulation at the t = 4hour, and proceed up to t = 12hours, testing both the Manning's formulations and the newly trained GP individuals (Shown in Table 6.3).

Table 6.3, Fitness scores (1/RMSE of all cells in all time steps) of the Manning's formulation and Bates limited, as well as the GP populations trained with a 0.5,1, and 2 second time step; run on the hill and pond test case for 8 hours of simulation time, from t = 4hours up to t = 12hours.

			Time step simulation fitness		
GP population	fitness score	fitness score	0.5	1	2
0	423.299	412.3180986	483.022	453.729	333.147
1	386.995	371.7563368	470.677	400.102	290.205
2	426.434	424.0592547	400.954	405.815	472.533
3	362.685	353.5214469	426.27	372.234	289.55
4	294.998	292.1049256	337.503	277.401	270.091
5	400.966	381.6974553	486.888	427.573	288.435
6	358.931	350.3397986	285.069	386.337	405.387
7	331.188	329.0016599	295.523	339.409	358.633
8	332.03	328.0233323	290.673	325.091	380.324
9	310.288	302.5494629	272.309	275.016	383.541
GP maximum	426.434	424.0592547	486.888	453.729	472.533
GP mean	362.7814	354.5371772	374.8888	366.2707	347.1846
Bates-Limited	1233.69	1233.216417	1200.85	1242.04	1258.19
Bates	620.304	236.3985878	1243.29	517.147	100.478
Dottori & Todini	262.222	207.3978676	379.527	287.386	119.754
Ghimire	203.945	139.1267407	323.758	215.826	72.2492

Once again all GP individuals' validation scores are better than that of either the Ghimire and in many case the Dottori & Todini - Manning's formulations. Also the average scores for each time step show that it appears harder to match the higher time step, although there are individuals which are the exception to this

rule, for example GP individual 9, which improves its score for the higher time step, counter to its training scores.

6.3.3.2 Testing on the same terrain with different rain profile

In the next testing case, a full 12 hour simulation is run on the same Hill and Pond terrain, with a different rain profile and therefore the water depth inputs and results simulation are different; a rain fall of 10mm/h for 2 hours is used, as opposed to 20mm/h for an hour for testing (and noting the training simulation was only for the first 4 hours), shown in Table 6.4.

Table 6.4, Fitness scores (1/RMSE of all cells in all time steps) of the Manning's formulations and Bates limited, and the GP populations trained with a 0.5,1, and 2 second time step; run on the hill and pond test case, with a different rain fall profile (10mm/h for 2 hours), for a full 12 hours of simulation time, from t = 0 hours up to t = 12hours.

			Time step simulation fitness		
GP population	Mean fitness	fitness score	0.5	1	2
0	455.975	449.0340739	492.987	495.119	379.818
1	331.047	326.9474276	368.079	342.553	282.508
2	371.712	364.551928	324.602	343.382	447.152
3	311.604	310.8110975	321.321	323.682	289.808
4	268.094	267.4062226	273.986	249.572	280.725
5	355.059	338.2086624	453.37	348.059	263.749
6	281.313	279.1863467	301.123	294.901	247.915
7	230.289	229.4902646	246.233	231.422	213.213
8	334.884	328.0554417	320.016	401.087	283.549
9	275.895	275.7836851	278.915	280.588	268.181
GP maximum	455.975	449.0340739	492.987	495.119	447.152
GP mean	321.5872	316.947515	338.0632	331.0365	295.6618
Bates Limited	872.658	872.4938989	862.958	865.414	889.602
Bates	438.251	251.1592381	718.347	478.223	118.183
DT	178.23	167.0299052	214.577	199.433	120.681
Ghimire	153.79	128.4979821	209.225	174.178	77.9674

Table 6.4, shows similar results to previous generalisation tests, demonstrating that the generalisation maintains, even with a different rain/water depth input; i.e. it continues to generalise well over the spatial area of the simulation. Considering by now, with a single time step, some obvious cases of

over training had occurred, it would appear that the additional training data has managed to add to the total volume of training data for generalisation purposes, producing better overall rules.

6.3.3.3 Testing on a different terrain (EAT2) with uniform rain input

In the next test case, experimentation is conducted using a completely different terrain (the EAT2 terrain), and rain fall profile of 40mm/h for an hour. The test simulations are run for 4 hours from t = 0 to t = 4 hour (Shown in Table 6.5).

Table 6.5, Fitness score (1/RMSE of all cells in all time steps) of the Manning's formulations and Bates limited, and a the GP populations trained with a 0.5,1, and 2 second time step; run on the EAT2 case for 4 hours of simulation time, from t = 0 hours up to t = 4 hours. Those score which have exceeded that of all the human competitors are highlighted in bold.

			Time step simulation fitness		
GP population	Mean fitness	fitness score	0.5	1	2
0	**559.055**	**555.3597239**	**604.448**	**574.199**	**498.518**
1	317.561	271.982337	**428.817**	**353.486**	170.382
2	**464.514**	**462.2318612**	**480.761**	**492.753**	**420.027**
3	304.887	262.5315703	**481.28**	241.204	192.177
4	283.641	274.3068947	**359.319**	234.589	257.014
5	208.606	208.3577101	212.272	214.892	198.654
6	**370.648**	**363.6616796**	**410.977**	**398.641**	302.327
7	80.6829	78.32310393	99.4542	77.2422	65.3522
8	288.209	283.4970442	260.458	**342.732**	261.438
9	286.954	252.9755025	**411.258**	278.133	171.471
GP maximum	**559.055**	**555.3597239**	**604.448**	**574.199**	**498.518**
GP mean	316.4758	301.3227428	**374.9044**	320.7871	253.736
Bates-Limited	332.181	332.1797629	332.98	332.333	331.231
Dottori & Todini	261.615	214.4585808	359.502	297.973	127.371
Bates	231.355	160.9381136	329.515	281.595	82.9549
Ghimire	222.344	162.0334113	325.865	254.315	86.8522

Table 6.5, demonstrates that some individuals have over trained to the specifics of the Hill and Pond trained case, be that either the terrain or the water depths provided, as they perform well on the other hill and pond test cases but not on these cases. It would appear that the Bates formulations (both limited and not) score particularly well on the Hill and pond test case at 50m, however it would

appear that many of the trained GP rules generalise better than even these rules. Clearly the rules are finding it easier at the lower time steps, which is similar to many of the human programmed Manning's formulations, however these are closer to the Bates limited formulation in their generalisation. A number of rules now exceed the resulting scores of the human formulations, including GP population 0 which score particularly well on the training data.

6.3.3.4 Results summary

It can be seen in Figure 6.6, that the Bates-limited formulation scores particularly well on all of the hill and pond test cases, but performs less well on the EAT2 test case. The GP mean score maintains its score across the test cases, where the best GP individual outperforms all human formulations on the training case, and is only out performed by the Bates-limited on the hill and pond test cases, but still scores well.

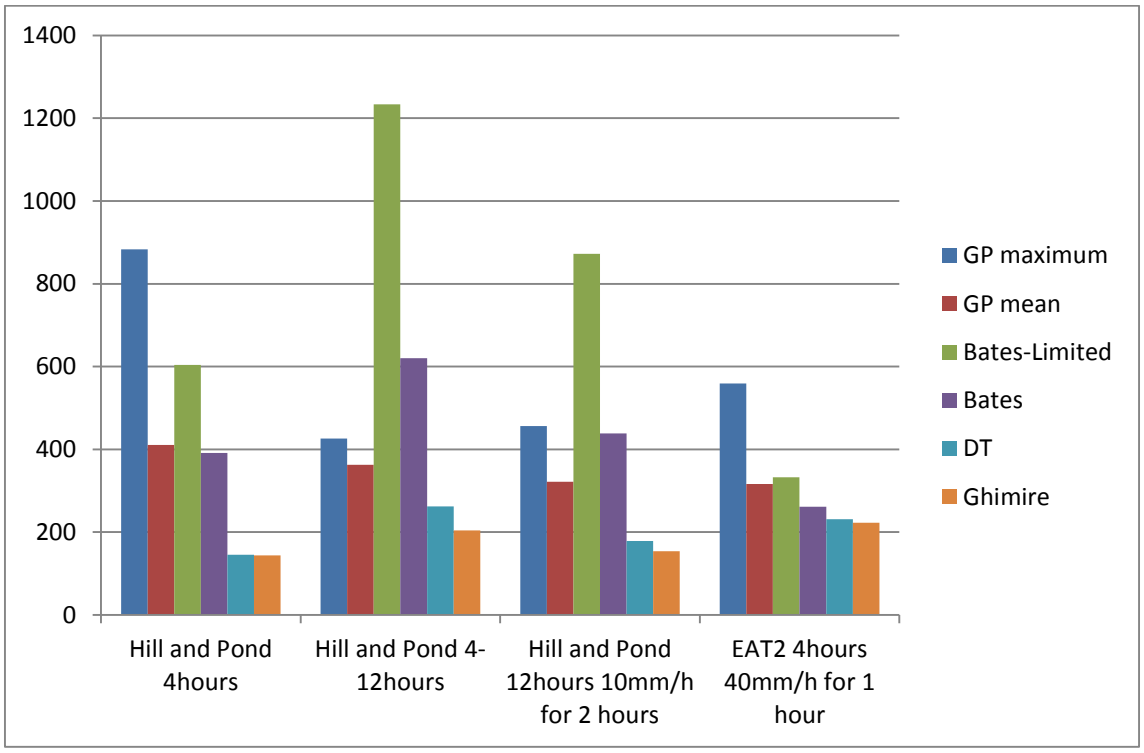


Figure 6.6, GP's Maximum and Mean scores of the 10 populations for each of the test cases, as well as that of the 4 different human formulations.

6.3.4 Discussion

Examples below show GP 0 population from the above trained GP, tested on the hill and pond test case for 12 hours, and shows the water depths at the Ponding, Crest (left, centre, right) and old outlet points; run on a 2 second time step (shown in Figure 6.7, Figure 6.8, Figure 6.9, Figure 6.10 and Figure 6.11).

This is contrasted against the hydrographs for the Ghimire Manning's formulation and the UIM target data. The Ghimire Manning's formulation is selected for this example as with this time steps settings it will begin to breakdown its simulation quality, and demonstrate how the GP formulation are capable exceeding the capabilities of number of the human formulation by avoiding excessive oscillations at higher time steps while reasonably matching the required pattern in space and time.

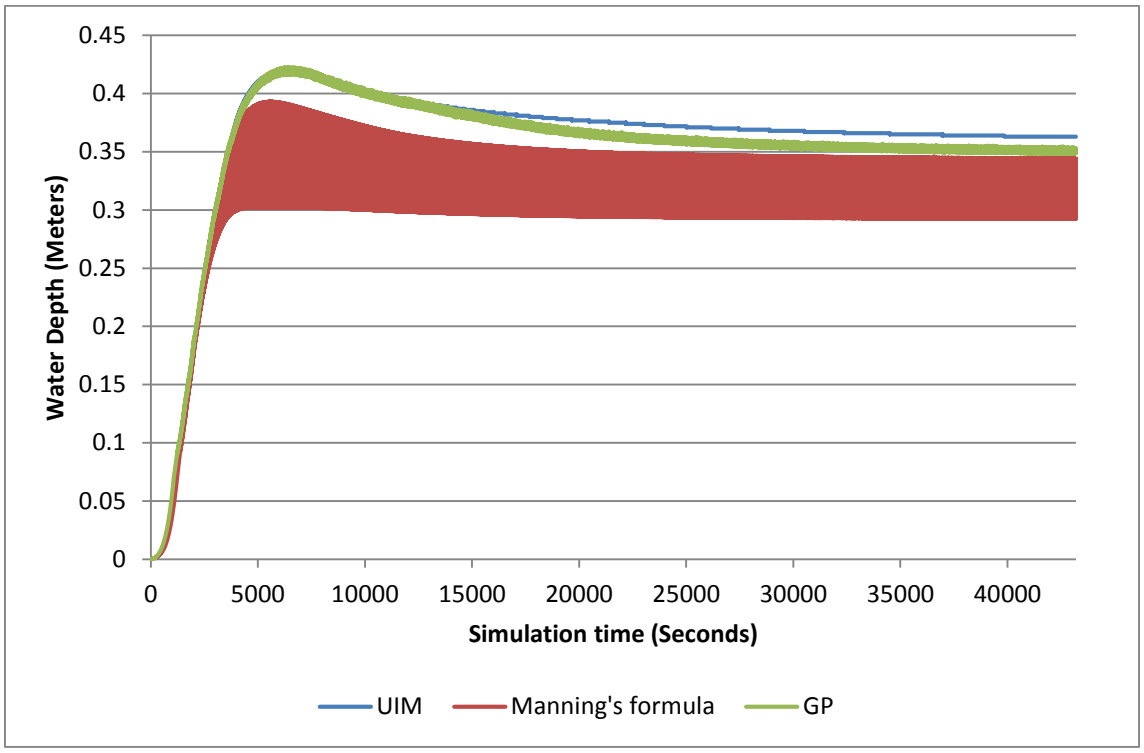


Figure 6.7, Water depths at the ponding point in the hill and pond test case, for UIM, and the Manning's formula, and the GP 0 individual, over the course of the 12 hours of simulation; with a 2 second time step for the CA models.

Clearly oscillations have started to occur with one the standard form the Manning's formula at this time step, however our trained GP generalises much better (Shown in Figure 6.7). However not every cell location within the grid has large oscillations as shown in Figure 6.8, Figure 6.9, and Figure 6.10. Although in Figure 6.10 there exists very minor oscillations and the Ghimire rule does not match the required hydrograph as well as the GP individual. This is thought to be due to nearby large oscillations.

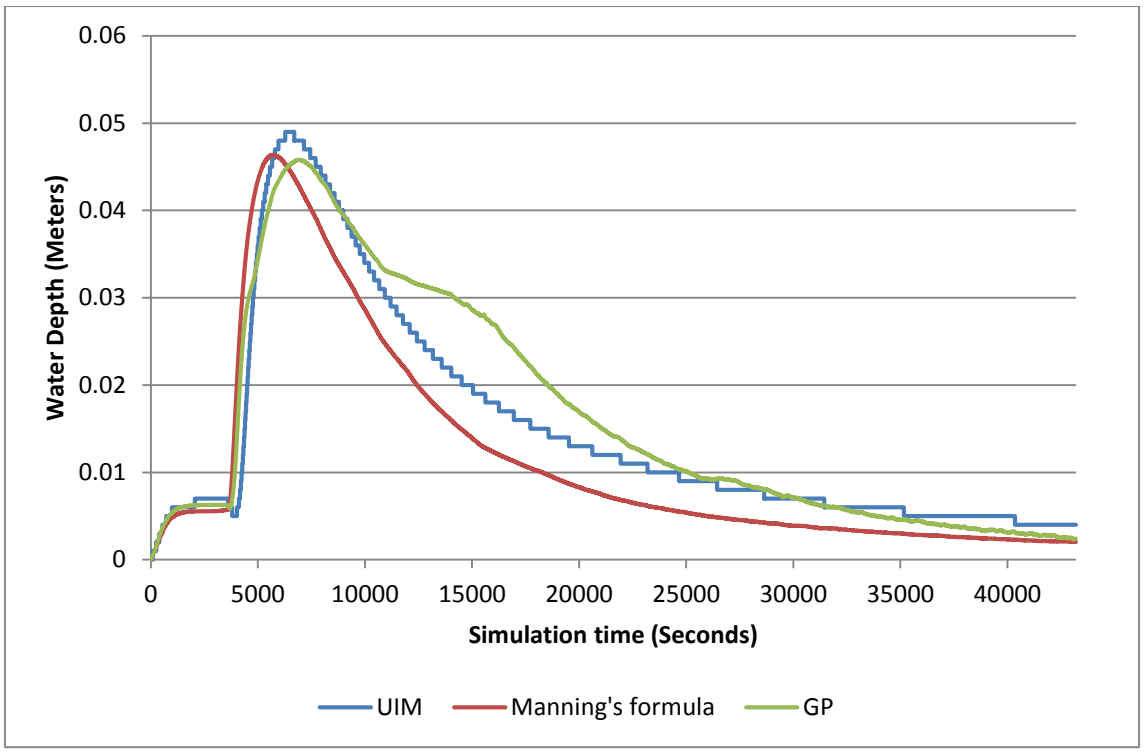


Figure 6.8, Water depths at the Crest Left point in the hill and pond test case, for UIM, and the Manning's formula, and the GP 0 individual, over the course of the 12 hours of simulation; with a 2 second time step for the CA models.

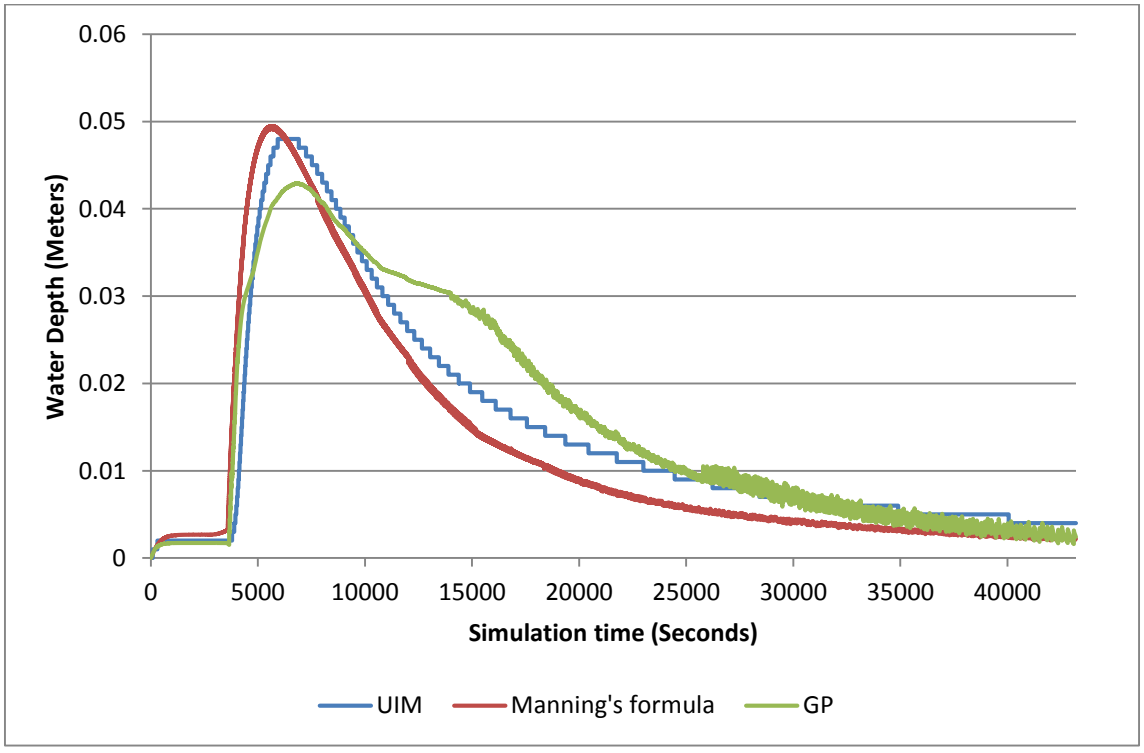


Figure 6.9, Water depths at the Crest Centre point in the hill and pond test case, for UIM, and the Manning's formula, and the GP 0 individual, over the course of the 12 hours of simulation; with a 2 second time step for the CA models.

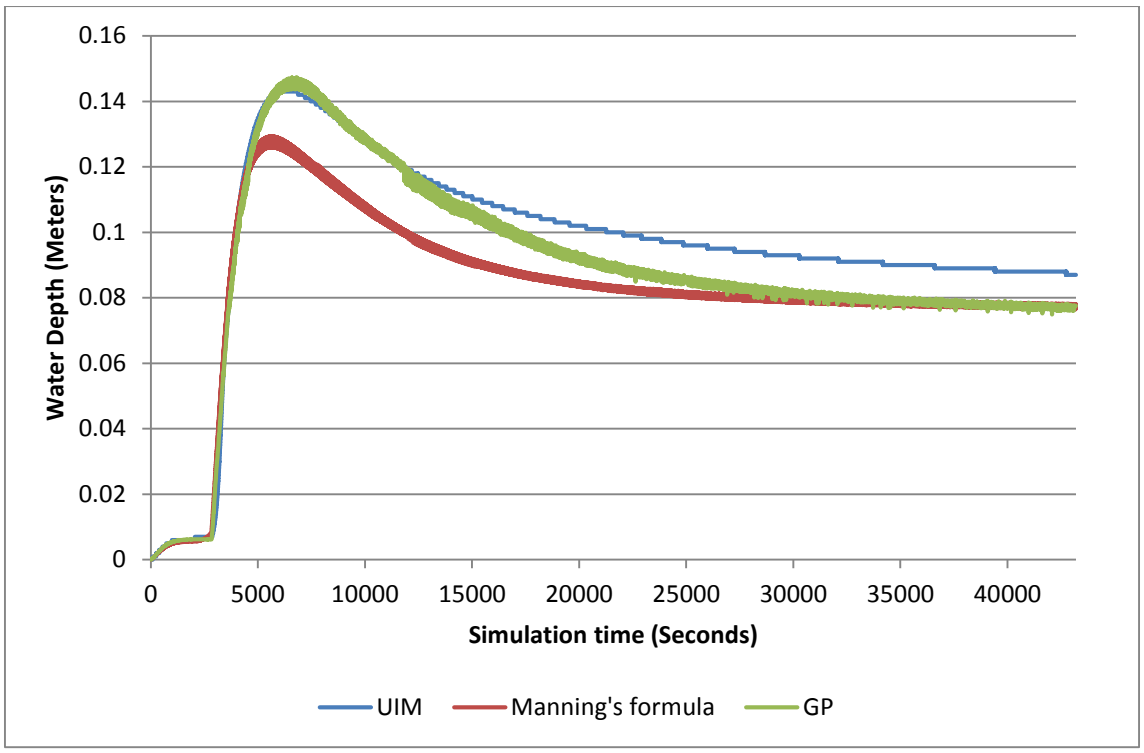


Figure 6.10, Water depths at the Crest Right point in the hill and pond test case, for UIM, and the Manning's formula, and the GP 0 individual, over the course of the 12 hours of simulation; with a 2 second time step for the CA models.

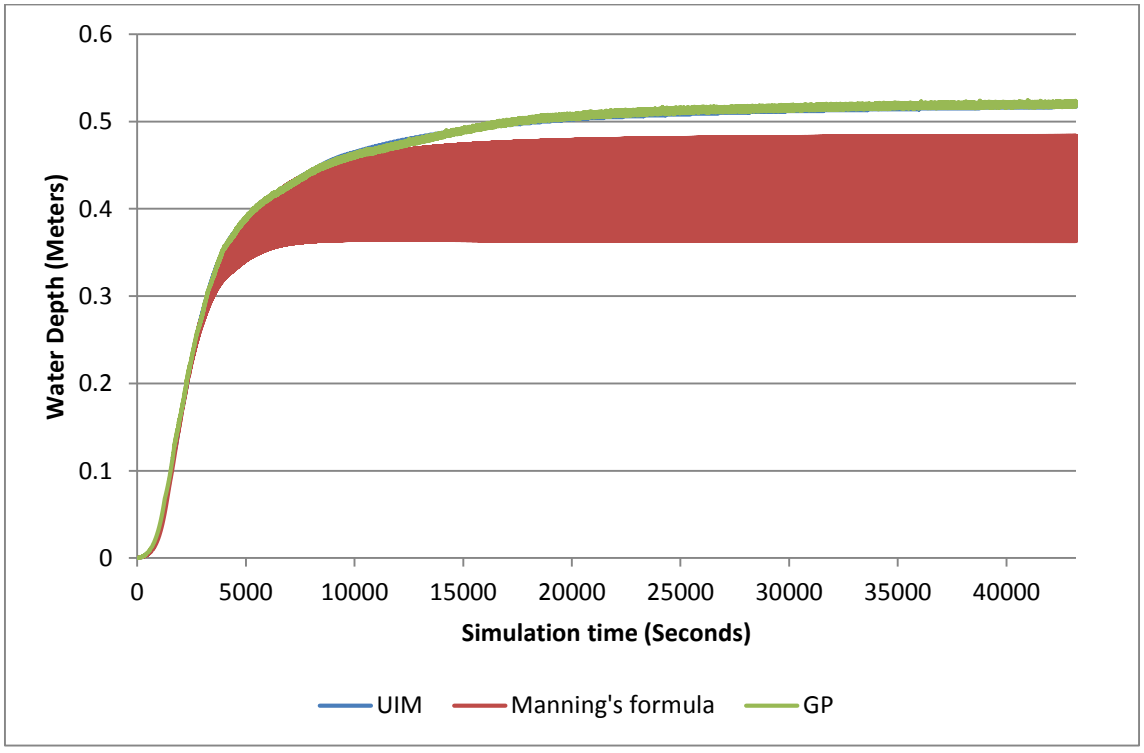


Figure 6.11, Water depths at the Old Outlet point in the hill and pond test case, for UIM, and the Manning's formula, and the GP 0 individual, over the course of the 12 hours of simulation; with a 2 second time step for the CA models.

Clearly seen in Figure 6.7 and Figure 6.11, which is where the water depths are larger within the grid, the Manning's formulation (Ghimire) has begun to

oscillate at a 2 second time step, whereas the GP formula generalises reasonably well. Although the particular candidate may have over trained slightly as it has minor oscillates, and ceases to follow the general pattern as well after the first 4 hours (Shown in Figure 6.8, Figure 6.9, and Figure 6.10). However, the GP individual far out performs the human formulation in terms of accuracy, and at a large time step factor. Thus this GP rule is capable of operating a quicker processing rate than the Ghimire rule, while still maintaining reasonable accuracy. This is an improvement in terms of multiple objectives of both speed and accuracy, via a method of optimisation for only the single objective of accuracy but over multiple time step factors. While this method does not produce a Pareto front, the idea of dominance might not work in the same way. I.e. A single rule or candidate solution does not produce a single point in the multi-objective space, but rather each rule may produce a front/trade-off between the two objectives. Therefore, the comparison of two different rules in this multi-objective space and the idea of dominance in the traditional sense are made more difficult. However, when considering a single time step factor, each rule will produce a single metric in both objectives of speed and accuracy for which traditional dominance could be established.

As the time step is so directly related to the speed objective, at a given time step rules are likely to produce very similar processing speeds. Also there is a requirement within the original objectives to create faster rules. This could be introduced as a preference within the trade-off towards speeds over accuracy. For example, if the speed difference between the rules at the same time step is consider negligible, and at a lower time rule A is more accurate than rule B. However, at a higher time step and therefore faster processing speed rule B produces greater accuracy than rule A, than it could be considered that rule B is fitter. What is likely required is to establish a threshold of acceptable accuracy, where more accurate models are acceptable, and to find the rule which can match or surpass this threshold at the largest time step, and therefore fastest speed possible. This area of multi-objective optimisation requires further work and research.

6.3.5 Inflow condition results

6.3.5.1 Testing on a different terrain (EAT1) with inflow conditions

Earlier training and testing limited to a single time step of 1 second, limits the number and types of viable test cases which can be utilised due to requirements for much lower time steps for human formulations to successfully operate. This includes EAT1 and EAT2 test cases in their originally prescribed inflow conditions, which is demonstrated in Figure 6.12 and Figure 6.14, where the most successful scores from the human formulations are at much lower time steps.

The EAT1 case is scaled up to a cell size of 50m and altered to have the same roughness factor as the training set (0.01), but maintains use of the water level event to drive the inflow. In these tests the human formulations are not scoring as well as on the original cell size and roughness factor, but also strangely the human formulation scores slightly worse on much lower time steps after a peak in their performance (shown in Figure 6.12).

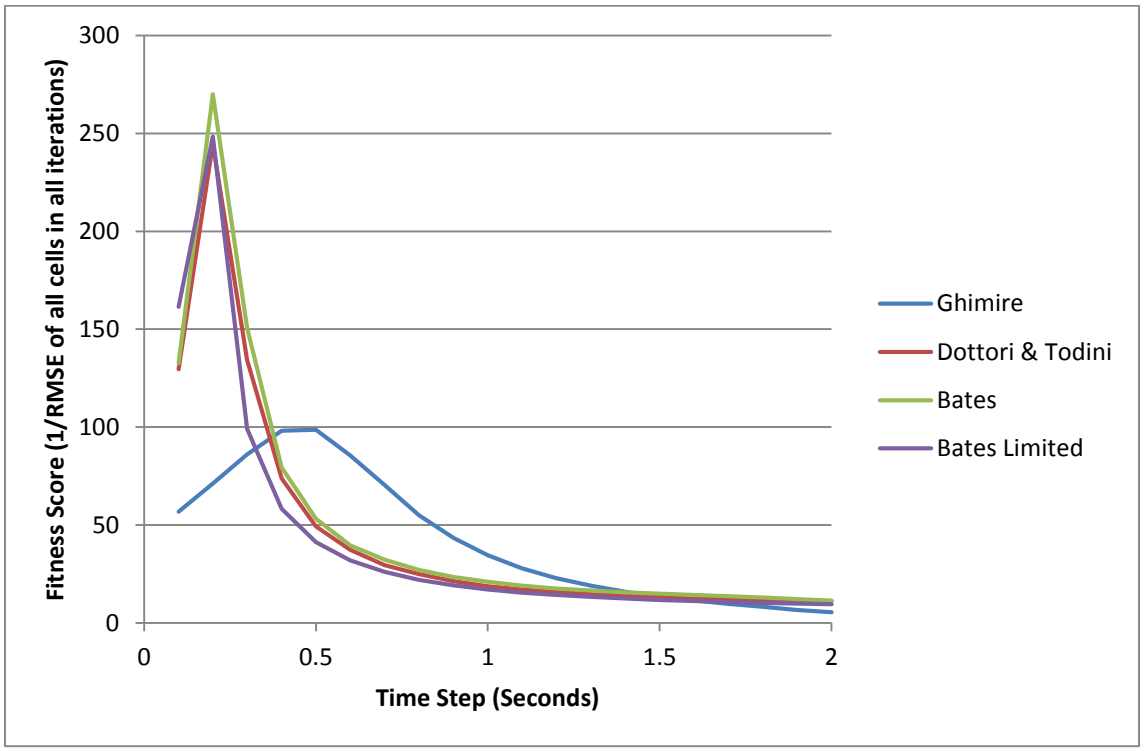


Figure 6.12, Fitness scores (1/RMSE of all cells in all iterations) of the Manning's formulations, on the EAT1 case scaled to 50m cell size and made 1 Dimensional, and 0.01n roughness factor; on various time step.

The GP trees that were trained on the hill and pond test case with rain conditions are now tested on the EAT1 scaled case, at 0.5, 1, and 2 second time steps, shown in Table 6.6.

Table 6.6, Fitness scores (1/RMSE of all cells in all time steps) of the Manning's formulations and a the GP populations trained with a 0.5,1, and 2 second time step; Tested on the EAT1 case for a full 20 hours of simulation time, from t = 0 hours up to t = 20 hours.

			Time step simulation fitness		
GP population	Mean fitness	fitness score	0.5	1	2
0	9.9798	9.609415619	12.7584	9.15182	8.02915
1	7.65107	7.548575143	8.95054	7.08551	6.91717
2	5.14602	5.063622589	6.02223	4.99574	4.4201
3	6.67526	6.474194886	8.22923	6.42996	5.36659
4	32.7277	15.06462784	58.2566	33.347	6.57943
5	9.47597	8.999509593	9.5209	6.8924	12.0146
6	8.01556	7.761859652	8.32962	9.52286	6.1942
7	3.8383	3.837939749	3.88461	3.8363	3.79398
8	8.14917	7.482624147	5.67382	11.5212	7.25246
9	22.4954	13.63357941	41.8607	18.6033	7.02236
GP maximum	32.7277	15.06462784	58.2566	33.347	12.0146
GP mean	11.41543	8.547594863	16.34867	11.13861	6.759004
Bates-Limited	22.5962	15.97452585	41.1879	17.0719	9.52887
Ghimire	46.1699	13.42200931	98.5847	34.5017	5.42341
Bates	28.4659	19.37550566	53.1203	20.949	11.3283
DT	26.4317	18.42106631	49.3245	18.775	11.1956

Seen in Table 6.6, there is a clear indication that some element of the EAT1 case is missing from the training example; since the difference in our experiments here make use of lateral inflow, and the fact that the EAT1 case is designed to alter the direction of flow towards the outflow. It is clear that the system has over trained/under generalised, or perhaps is missing enough training example of circumstances which occur uniquely within the EAT1 case with the water level event. Since only two of our GP rules have scored close to the score of the Manning's formula, these individual's performance is investigated (GP individuals 4 and 9), also one of the individuals which scored well on previous test cases but poorly on this one is investigate (GP individual 0), Shown in Figure 6.13.

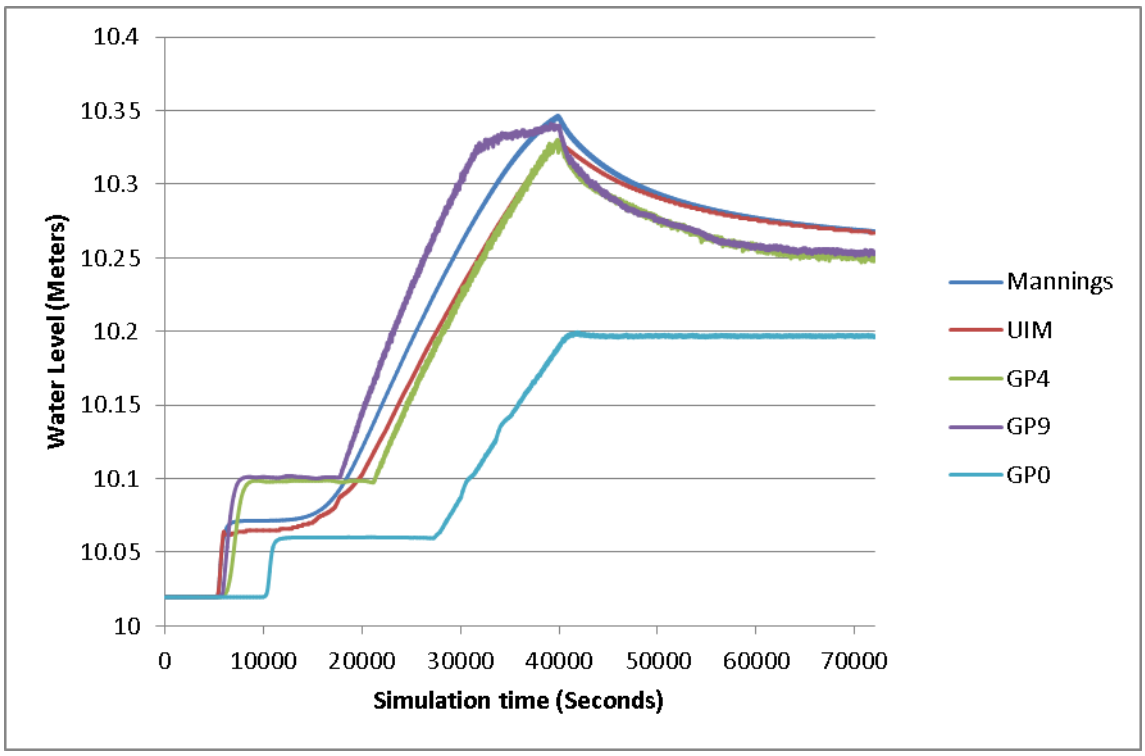


Figure 6.13, Water level at the test point 1, on the EAT1 case scaled to 50m cell size, and 0.01n roughness factor, UIM is shown at its original time step settings, but the Ghimire version of the Manning's formula and trained GP individuals 0, 4, and 9 are shown at a time step of 0.5 seconds. Time period shown from t = 0 seconds up to t = 72,000 seconds, which equates to 20 hours of simulation time.

Figure 6.13 shows that actually our two good candidate GP individuals 4 and 9, are reasonably close to the target UIM simulation, and that of GP 0 is performing much worse, as our fitness function confirms. It is difficult to determine exactly how the system has over trained to rain conditions, compared to inflow/water level events, but appears to an over training/under generalisation to this condition. Perhaps the more similar water levels form cell to cell, and general tendency for rain events to tend towards a single convergence in space and time, is too radically different from this test case. Also since the minimum time step that the human formulations operate well on this test case are between 0.2 -0.5 seconds, and these rule do not appear to generalise well to lower time step values.

6.3.5.2 Testing on a different terrain (EAT2) with inflow conditions

Finally since the EAT2 terrain has been utilised for testing successfully with a uniform rain inputs to each cell (section 5.5.3.3), now tests are conducted on the EAT2 terrain with the prescribed inflow condition. Figure 6.14 demonstrates

a similar pattern to Figure 6.12, where the human formulations are tested on the EAT2 terrain with inflow conditions, at various time steps.

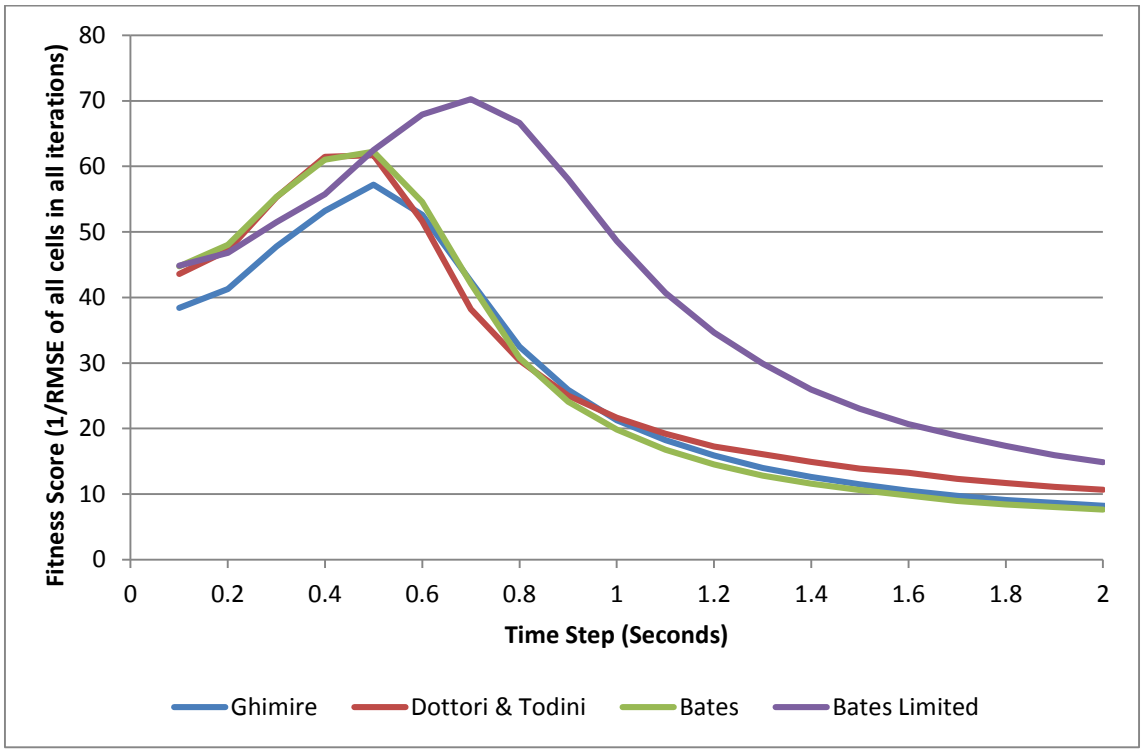


Figure 6.14, Fitness scores (1/RMSE of all cells in all iterations) of the Manning's formulations and Bates Limited, on the EAT2 case scaled to 50m cell size, and 0.01n roughness factor, and inflow conditions; at various time step.

We see peak performance in Figure 6.14 at approximately 0.4 seconds, and reduces on lower time steps, as well as higher time steps. This occurs for all those human formulations that don't use a flow limiter. However, the spike in performance for the Bates limited formulation occurs at a slightly larger time step of 0.7 seconds. This could be due to the interpolation between each second of the target, and the variances in exact amounts of in and out flows, it is difficult to say. These are however much lower scores than compared to on the same terrain with a rain profile (section 6.3.3.3), demonstrating how inflow conditions test cases, are harder to approximate. This is probably due to the large variance in water depths, due to the existence of dry cells receiving large amounts of water as the water front moves across the grid.

Similar results to those of EAT1 (Table 6.6) are shown in Table 6.7 for EAT2, in that the validation scores are very low due to the inflow conditions. These results are vastly different to those of the same terrain with rain conditions shown in Table 6.5, demonstrating both that the inflow condition test cases are harder

to optimise, but also that the GP system appears to be over trained to the type of condition it was trained upon.

Table 6.7, Fitness scores (1/RMSE of all cells in all time steps) of the Manning's formula and a the GP populations trained with a 0.5,1, and 2 second time step; run on the EAT2 case for 8 hours of simulation time, from t = 0 hours up to t = 8 hours; with inflow conditions.

			Time step simulation fitness		
GP population	Mean fitness	fitness score	0.5	1	2
0	5.08953	4.928335	6.29615	4.91383	4.0586
1	6.96491	6.964451	6.89286	6.97153	7.03033
2	0.249989	0.249989	0.250219	0.249919	0.249829
3	3.7366	3.613535	4.64828	3.58994	2.97157
4	8.1764	7.206672	12.2413	7.15544	5.13251
5	6.18522	6.158069	6.72133	6.117	5.71733
6	0.360174	0.356838	0.343788	0.327092	0.409642
7	0.268826	0.268245	0.285758	0.265212	0.255509
8	4.23827	4.233514	4.40682	4.24786	4.06013
9	3.9344	2.802652	6.9029	3.29201	1.60828
GP maximum	8.1764	7.206672	12.2413	7.15544	7.03033
GP mean	3.920432	3.67823	4.898941	3.712983	3.149373
Bates Limited	41.9822	28.88388	62.4699	48.6148	14.8618
Bates	29.9119	15.21516	62.2435	19.8414	7.65075
Dottori and Todini	31.3431	19.21382	61.7231	21.6357	10.6705
Ghimire	28.9008	16.12153	57.1933	21.2871	8.22198

6.3.6 Conclusions

The results in Section 6.3.3 show that the GP tree trained on multiple time step simulations tests similarly to those trained on a single time step in section 5.5.3. When tested on different input conditions of rain inputs levels, initial water levels and even terrain, however they demonstrate greater generalisation to time step factors. Producing these high scores on unseen data where the complex set of flow rates that lead to high scores varies for each time step, demonstrates the extra level of complexity introduced into the training of these rules sets.

Testing on lateral inflow conditions, demonstrates that the system has trained to the specifics of the type of inflow conditions (i.e. rain conditions). Possible due to the way water is flowing into a dry simulation from a particular point which produces a greater possibility to lead to spatial variation in the water levels. The GP rules do not perform as well as on the rain conditions tests, although the human formulation also find it harder approximating these simulation, although the relative performance of the GP rules is much worse. This is thought to indicate a level of over training within the GP rules to the rain conditions of the training cases.

Where the rules do generalise well to other input conditions, like variance in the initial water levels, rain levels, and terrain inputs, it has been possible to create rules which can perform similar simulations to the training set on unseen data at higher time steps than a number of the human formulations. This would allow for simulations to be processes at a quicker real world computational rate while still maintaining a reasonable level of accuracy to the original simulation.

6.4 Training GP for temporal and spatial generalisation of CA rules

6.4.1 Introduction

As real world simulation uses such static variables as cell size and time step to represent the different discretisation of time and space, this expands the breadth and scope of what the human formulation are capable of. While section 6.2 has already investigated the limits of training GP and human formulation under various time steps/temporal discretisation's, and found that at the particular cell size of 50m there is a limiting highest time step before oscillations are created within the simulation which destroy its quality. However, literature demonstrates that for the Manning's formula alone there is a relationship between the cell size and maximum time step at which the rule set will operate successfully (i.e. without large destructive oscillations). This is demonstrated in Figure 6.15, where the Manning's formula fitness is clearly related to the cell size, whereby the maximum time step for smaller cell size is much smaller also. However, the Bates-limited formulation is specifically designed to overcome this problem to a greater degree. In Figure 6.15, the original hill and pond test case test has been scaled to different

cell sizes, by means of using the same terrain heights but simply labelling the cell as different sizes. Since this effectively resizes the terrains total simulation size, this makes the slopes of the terrain models steeper. Since a uniform rain profile is applied, the same water levels fall into each cell, but given as these are different volumes with different slopes, then there are different flow rates which should prove as good training examples for GP. These differently scaled simulations are re-run through the hydraulic modelling software UIM, to produce different target sets of data for each scaled test case.

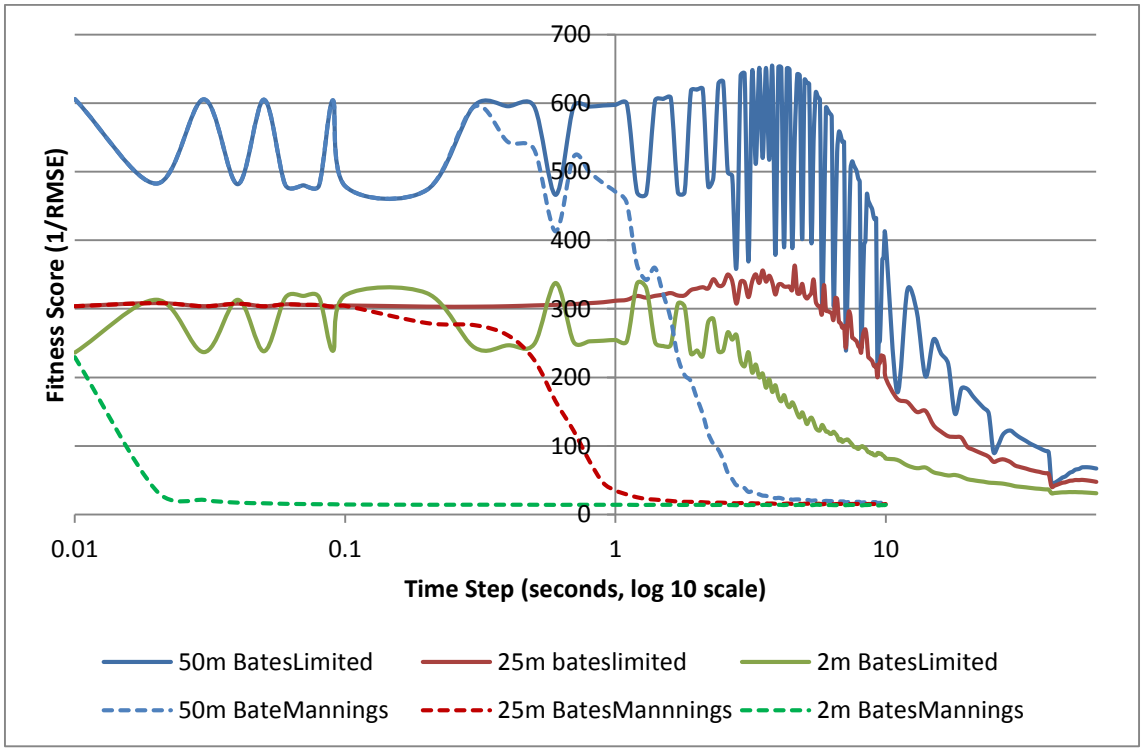


Figure 6.15, Fitness scores of Bates Manning's formulation and the Bates Limited formulation, on the Hill and Pond test case, with a 50m, 25m, and 2m cell sizes. Note a logarithmic base 10 scale is used on the time step (x-axis).

It can be seen in Figure 6.15 that the Bates formulations score particularly well on the 50m test cases, where the fitness scores plateau between 500~600. Although there are large oscillations in the fitness score of particularly the 50m test cases with the Bates Limited formulation, this drops as low as the scores on the 25m test cases. It is thought that this is mainly caused by two factors, firstly a particular factor of each time step allows the model to fit the underlying 50m target model more accurately (i.e. it could be an element of coincidence), especially as the 25m and 2m test cases plateau out at approximately the same fitness score of 300. Secondly due to the interpolation of the original target UIM model, which was run once for each cell size, as the cell size creates a different

model output. Those human formulations which directly use the Manning's and discharge formula, show their particular weakness when the space of cell sizes and time steps are explored together (as in Figure 6.15). Where the Manning's formula formulations have a very small maximum time step at which it performs adequately for the very small cell sizes, this would be an unfeasible time step at which to run training due to the number of CA iterations required. However, the Bates Limited formulation is capable of maintaining reasonably high fitness on lower cell sizes at much higher time steps.

The true challenge of this final section is firstly to train rule sets which can find the correct fluid flows given the varying water depths/level, and terrain levels, but also adapt to different spatial and temporal resolutions (cell size and time step). Secondly to see if the GP system can match, or even exceed the most advanced human formulation in terms of the upper limits on the time step and performance. Effectively this tackles the trade-off between overall processing speed and accuracy, by explicitly controlling the processing speed though the spatial and temporal resolution of the simulations.

6.4.2 Experimental set-up

In attempts to train a system which is both capable of generalisation to the variation of multiple static variables, and different inputs, the system is trained upon a number of different time steps at each of the given cell sizes. Using 50m, 25m, and 2m test cases derived each from a different simulation run on the hydraulic modelling software, UIM. Each target model is run for four hours of simulation time, at 0.5, 1, 2, 5, 10, and 25 second time steps, totalling 15 different test cases. The combined fitness is calculated as the reciprocal of the average RMSE from each case. Shown in Table 6.8 are the fitness scores and the average fitness scores across the 15 different test cases (i.e. the reciprocal of each RMSE), for each of the human formulations.

Table 6.8, Fitness scores (1/ average RMSE of each test case) and the Mean fitness's (1/RMSE of each test case) for the human formulated rule sets.

Rule set	**Fitness score**	**Mean Fitness**
Bates Limited	173.2149287	304.78285
Bates	19.88530422	90.76921667
Dottori and Todini	24.86212055	51.42916111
Ghimire	22.35578729	45.49682778

Clearly this particular test set has been designed to highlight the strength of the Bates Limited formulation, as all other human formulations fall to much lower fitness scores at the lower cell sizes with these times steps. This does however make the whole operation feasible, as training at lower time steps would require more iterations, and extend the processing time to unfeasible ranges. Shown in Table 6.9 are the fitness scores for the Bates Limited rule set upon each of the test cases.

Table 6.9, Fitness scores (1/RMSE) for each of the test cases, for the bates Limited formulation, on the hill and pond test case for 4 hours of simulation time, at various combinations of cell size and time step.

	50m	25m	2m
0.5seconds	594.97	304.987	248.68
1 s	597.408	311.863	254.602
2 s	619.463	329.614	239.33
5 s	634.199	319.885	141.091
10 s	396.327	198.228	81.5564
25 s	90.9623	76.758	46.1676

6.4.3 Experimental Results

Shown in Figure 6.16 are the fittest individuals from each of the 10 population during the 500 GP generations applied, along with the average of the 10 fitnesses.

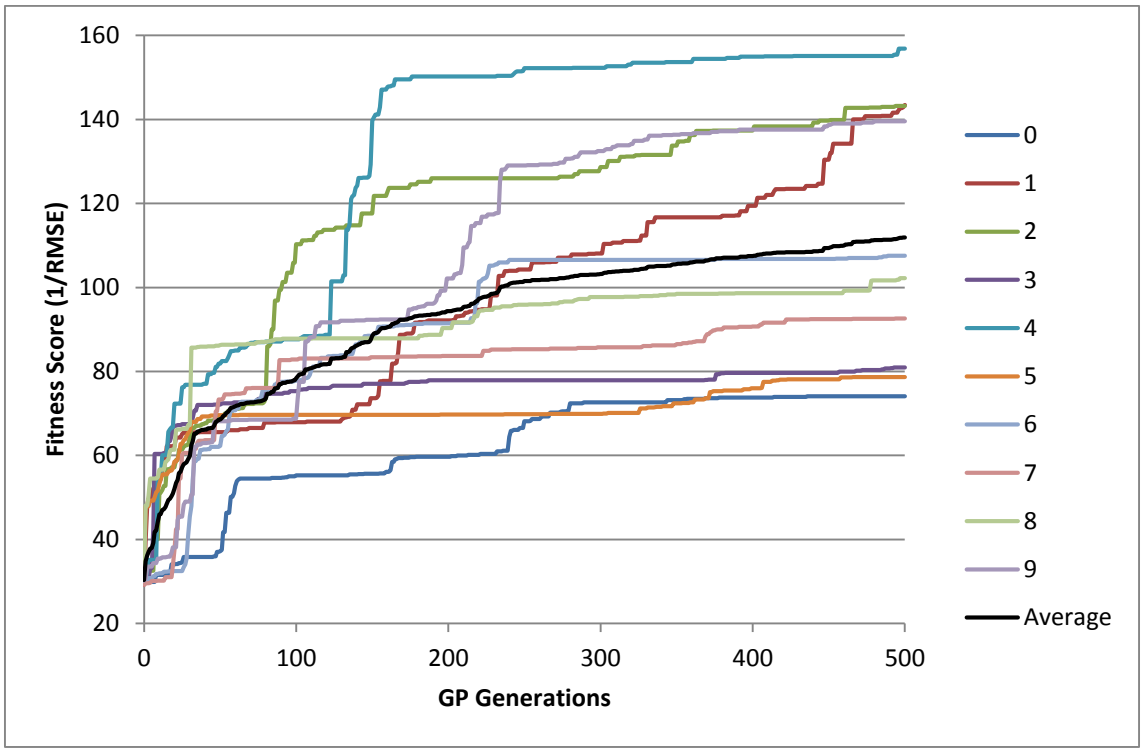


Figure 6.16, Fitness scores (1/average RMSE of each test case) of the fittest individual within each of the 10 populations, and the average of these 10 fitness scores.

A reasonably wide variation in fitness can be observed in Figure 6.16, as this is now a much larger search space. Shown below in Table 6.10 are the fitness scores from each of the specific test cases for the fittest individual in the fittest population of the 10 differently seeded populations.

Table 6.10, Fitness scores (1/RMSE) for each of the test cases, for the fittest individual from the fittest of the 10 populations (GP 4).

	50m	25m	2m	Average
0.5seconds	453.496	163.277	311.357	309.3767
1 s	441.914	162.799	318.059	307.5907
2 s	424.51	160.85	302.952	296.104
5 s	350.758	161.494	177.848	230.0333
10 s	275.346	151.331	84.2666	170.3145
25 s	89.1788	68.6467	45.4046	67.74337
Average	339.2005	144.733	206.6479	

Clearly this individual is performing better on the two extreme cell sizes of 50m, 2m than at 25m cell size, and performs reasonably well up to the 10 second time step, where thereafter it begins to degrade. Figure 6.17 below shows the fitness scores for the best GP program from the best population (GP4, as shown in Table 6.10), on the hill and pond test case for 4 hours simulation time, at various time steps.

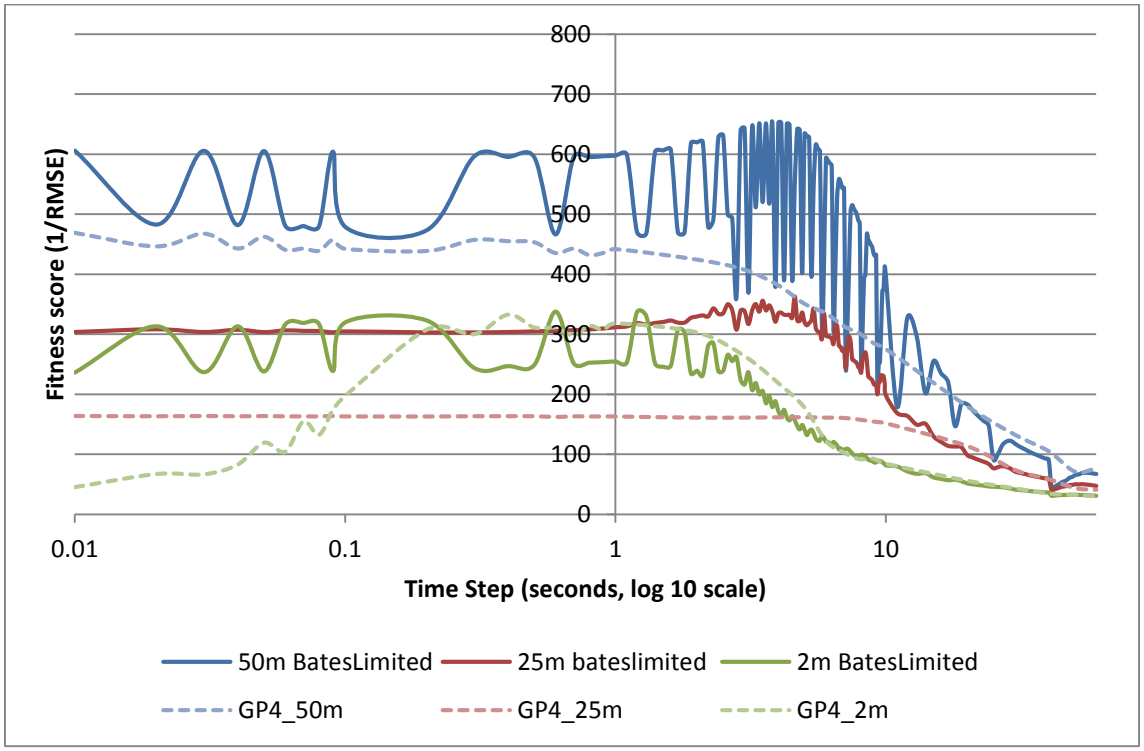


Figure 6.17, Fitness scores of Bates Manning's formulation and the Bates Limited formulation, on the Hill and Pond test case, with a 50m, 25m, and 2m cell sizes. Finally, these are contrasted against the best scoring GP individual in the best population (GP4). Note a logarithmic base 10 scale is used on the time step (x) axis.

It can be seen in Figure 6.17 that better time step generalisation at 50m has been obtained, compared to that seen in Figure 6.1 or Figure 6.2. The generalisation extends to both the smaller scaled time steps and shows a very similar maximum time step to the Bates limited formulation. While the scores are not as high as the Bates limited at 25m, again a good generalisation to the time step is observed. The 2m cases shows an over specialisation to the large time steps, but again shows a similar maximum time step to that of the Bates limited formulation.

6.4.4 Conclusions

There is a wide degree of variation in the routes taken through the evolutionary landscape by each different population, where only the random seed is varied. This is thought to be due to a number of reasons including the following:

- The reduction of the many different phenotypical behaviours into a single objective function/score means that due to the different initial seeding, it may select very similarly scored individuals, but this will translate into very different genotypes. This in turn will lead to different possible future routes through the evolutionary landscape.
- Unlikely GA type algorithms, the chromosome has no real sense of alignment, in that different parts of the chromosome can be responsible for different phenotypical behaviours, where this appears to vary more across individuals. Also crossover is capable of shifting pieces of code/GP tree radically in position.
- The growth and mix of introns and active code within early GP tree, and the hard cap placed on the depth of the tree forces the system to move within this search space.
- Specific heuristics used, i.e. the parameter settings, and their hard coded equivalent (e.g. number of operations, and terminals, even mutation and cross-over levels, as well as population levels).

From Figure 6.16 and the wide variety of resulting scores, we conclude that when training such a complex rule set, and all the specific parameter settings utilised in these test, has a greater chance of falling into a local minima than those shown in Figure 6.3. However, a number of populations do manage to achieve reasonable scores, and when these are compared to the Manning's formulae, the GP outperform all but the Bates Limited formulations in all cases. However, this is a result of the specific time steps targeted are aligned with such a high time step rule set. I.e. in this single objective system, the secondary objective of total real world processing time is controlled explicitly through the use of the time steps targeted. As the time step is negatively correlated with the number of iterations and therefore the processing time, a rule which can maintain accuracy at higher time steps, can simulate to a reasonable level of accuracy the given water levels at a faster real world processing time.

Throughout this thesis a static time step for each simulation has been maintained, although it a number of works in the literature propose an adaptive time step. This is partial to try to overcome to problem of excessive flows within the simulation at a cellular level, by setting the time step to match the greatest flow. However, it is also takes advantage of the underlying assumption that gravity is the main driver and friction negates most of the effects of momentum, and therefore simulation will eventually settle down and cease most movement. This in turn should mean that a system which links the timestep during the simulation to the maximum flows, should also eventually become larger and larger, and move up to the maximum. Indeed, even with adaptive time steps, there is the need for maximum and minimum time steps, at which there is a very similar problem of which rule best approximates the 'real' or target simulation, given that it is asked to use a higher time step than if flows are greater than the minimum time step. However, it might be expecting too much for a rule to be generated which operates across all known spatial and temporal resolutions given the training set has only a limit number of examples of such grid static variables.

6.5 Conclusions

In conclusion the experimental results have shown that it is possible to train rules which can learn near optimal local water movements such that the global water movements are nearly consistent with simulated targets, where such rules can also be trained to operate with reasonably accuracy upon different spatial and temporal resolutions, and will generalise to other input conditions. This is a facet of having trained a single rule which operates in a distributed fashion across a cellular automata system, i.e. in order to enforce any change to the global behaviour in any particular area a change must be made to the rule which operates in every cell. This in turn encourages some selectivity in the rulesets between the given variables.

In section 5.4, it was determined that a minimum amount of simulation time is required such that the rules generated begin to pick up sufficient hydraulic of water flows. While the system scored highly during training on simulation with less than this minimum amount of simulation time, these rules would not generalise well to other input conditions. Rules generated after this minimum

point did not perform much better on validation/testing cases, and increasing the amount of simulation training time (while maintaining the same time step) increases the amount of real world processing time for the optimisation linearly. Therefore, the use of extra simulation time in training after this minimum amount represents a case of diminishing returns. This is thought to be due to the nature of the hydraulic examples presented within the training case, where the point of concentration at the ponding point is at approximately 1hour 45minutes, and examples of both drawing up to and down from this concentration point are required.

In section 6.2 this methodology is extended to begin to tackle the trade-off between real-world processing speed and accuracy, by means of changing the time step applied during simulation and optimisation. It is firstly demonstrated that previous training upon a single spatial and temporal resolution (cell size and time step) results in over training to those specific examples. Training is then extended such that each GP individual is presented with a sparse number of different time step examples, by re-running the training case simulation at different time steps. This forces the training GP rules to adapt to the different settings presented in each case, while trying to match the same space-time pattern. Since those rules which are capable of operating with reasonable accuracy at larger time steps take less real world processing time, this allows the system to tackle the trade-off problem and is capable of producing rules which are more efficient than some human formulations. The generalisation properties however are still centred upon the conditions present within the training case, for example when having trained the system upon uniform rain input conditions, the rule sets generated do not generalise well to lateral inflow conditions.

Lastly in section 6.4, this methodology is extended to the final stage of training rules which can operate at different spatial and temporal resolutions, by presenting each GP rule set with simulations of the training case at different cell size and time steps. As literature demonstrates the most difficult challenge for human programmers of such real world CA rules, is to be able to create functional rules which operate at smaller cell sizes, and higher time steps. This also challenges the GP system to understand how many different variables affect the system.

Since in cellular automata systems, the size of cells and the amount of time between iterations are both abstract terms, the linkage of cell size and time step within a real world CA system presents a novel problem for training a local state transition rule. This has been systematically tackled within this chapter.

Chapter 7: Conclusions and discussion

7.1 Conclusions

The aim of this thesis has been to accelerate urban flood modelling, through the use of CA on modern multi-core and many-core hardware. An additional goal was to use GP to learn the CA state transition rules, with further acceleration on many-core hardware. This thesis investigates and begins to understand the trade-off presented by the simulation resolution and accuracy. This thesis has tackled all of these challenges, and draws the following conclusions and contributions:

1. Objective 1 - “The investigation of the parallelisation of CA systems upon modern many-core GPGPU technologies, and the effect of varying the standard CA parameter such number of cells, initial configuration and activity, number of states, neighbourhood size, and number of generations on the speed-ups obtained. Also to investigate the effects on the relative speed-ups obtained, of varying GPGPU parameters such as the workgroup size, GPU memory type, and the base data type used to store states. This investigation is intended to ensure that the relationship between the CA parameters and the relative speed-ups of the GPGPU over the CPU are well understood, such that later work in this thesis can maximise speed-ups from the GPGPU when combining GP and CA systems”. Objective 1 is tackled in Chapter 3:, where it is concluded that the main driver for CA speed-ups is the number of cells relative to the number of cores of the GPGPU. The best speed-ups are obtained when the number of cells are between one to two orders of magnitude greater than the number of GPU cores.

2. Objective 2 – “The development of a CA system for flood modelling based on existing models from literature, which is capable of expressing a spectrum/range of variable state transition rules. It is intended that these state transition rules should always ensure uniformity to direction of flooding flow and should preserve the water volume across the grid. This will allow for the derivation of state transition rules which can concentrate on finding the correct flow rates given the water, terrain levels and spatial and temporal resolutions across the grid. Thereby, a GP system is generated for the optimisation of CA

state transition rules. Such a system should take advantage of the research conducted to satisfy Objective 1, in order to obtain the best speed-ups possible by accelerating the evaluation of CA fitness functions upon the GPGPU". Objective 2 is tackled in section 4.2 and section 5.2 where a methodology is developed for using GP to develop CA state transition rules from example data. Furthermore this GPCA methodology takes advantage of conclusion 1, and combines the parallelism from the GP and CA algorithms in order to be able to gain the best speed-ups on the GPU during optimisation. This is achieved while only using relatively small test cases, and therefore minimal overall processing time for optimisation processing.

3. Objective 3 – "An investigation of the effectiveness of the combined GPCA system from Objectives 2 and 3, to learn a known CA rule set such as the Game of Life. This will allow for the calibration and confirmation that the system can find the correct underlying state transition rules from an example CA simulation". The experimentation in Chapter 4:, section 4.3 demonstrates that the GPCA system is capable of learning a state transition rule for a known system, i.e. the Game of life, thereby meeting Objective 3. Although the Game Of Life is shown to have large jumps in the fitness landscape between those close to the global optimum and the actual global optimum (Figure 4.5), it is thought that this large jump in the fitness landscape is due to the binary nature of the Game Of Life style of rule set, and its capacity to radically change with simulation outputs with small changes to the rule set.

4. Objective 4: "An investigation of the effectiveness of the combined GPCA to learn flood modelling state transition rules based on example simulation data". Chapter 5: and Chapter 6: undertake the investigation required by Objective 4, where the effectiveness of the GPCA system is investigated to learn flood modelling CA state transition rules.

    4.1. Objective 4.1: "Quantify the simulation time needed during training on a fixed set of spatial and temporal resolutions, and prove that the combined GPCA system can learn state transition rules which are competitive amongst human CA flood modelling rules", and Objective 4.2: "The proof of hypothesis 1, through the testing of derived state transitions rules from

objective 4.1 on unseen data, including unseen sections of the training test case and completely different terrain".. This is tackled in Chapter 5: where the GPCA system is trained with a number of different lengths of training simulation, and tested on unseen data. Section 5.4 concludes that it is easier for the GPCA system to more closely match the training simulation with less simulation time (shown by the higher scores for shorter training simulation times in Table 5.6). However section 5.5, which tests the generated rules on unseen data concludes that the rules trained on the shortest simulation times (test cases of 1 hour), have indeed over trained to the limited movement during this period (shown by Table 5.10, Table 5.14, Figure 5.21, Table 5.16, and Figure 5.22). Rules generated with 2-4 hours of training simulation time, gained the best testing performance relative to their training performance. Increases in simulation time for training, extends the computation time for optimisation. Further increases in the amount of simulation time for training, presents a point of diminishing returns in terms of generalisation to unseen data. This provides a good weight of evidence for hypotheses 1, in that given a suitable amount of training data the system can generate rules which are capable of operating on unseen data, thusly tackling Objectives 4.1 and 4.2.

4.2. Objective 4.3: "An investigation of the effectiveness of the combined GPCA system to learn flood modelling CA state transition rules which are capable of operating competitively at a range of temporal resolutions. By creating rules which can produce competitive accuracies at higher time step factors (temporal resolutions) than human formulated CA state transition rules, this will begin to tackle the trade-off problem of creating faster rules. Thereby this investigation tackles the ultimate aim of creating faster rule sets through the use of machine learning techniques to derive the CA state transition rules for flood modelling systems (hypothesis 2)". Section 6.2 concludes that the methodology of using multiple simulations with different temporal resolutions is capable of creating rule sets, which can operate at competitively low temporal resolutions (large time steps) with some of the most advanced human CA formulations from modern literature (Shown in Figure 6.2). This begins to tackle the trade-off of

overall processing speed versus the accuracy of the resulting simulation, by creating rules which can maintain higher accuracies at lower temporal resolutions and thus addresses Objective 4.3, and provides and provides a weight of evidence for hypothesis 2.

4.3. Objective 4.4: “An investigation of the limits of hypotheses 1, by testing of those rules generated during training conducted in Objective 4.3, upon different inputs including: unseen parts of the training test case, a completely different terrain, and finally on different ‘*boundary conditions*’ (the type of inflow used in the test cases, e.g. uniform rain, or a lateral inflow)”. Section 6.3 investigates the limits of the generalisation of the GPCA system, by testing the rules generated in section 6.2, on not just unseen terrain and water levels with uniform rain conditions, but also with lateral inflow types. The results demonstrate that the generalisation to unseen data is limited by the training conditions, i.e. since the training test cases lack lateral inflow this is the main reason for the lack of generalisation to these types of tests. Furthermore, since there is little change between each cell in test cases involving a uniform rain condition, those with lateral inflow type conditions make it more difficult for rules to match target data. These experiments address Objective 4.4, and therefore demonstrate the limits of the hypotheses 1 and 2.

4.4. Objective 4.5: “Finally an investigation of the ability of the GPCA system to learn CA state transition rules that can operate successfully at a range of both spatial and temporal resolutions. The investigation demonstrates how the proposed system can adapt to the complex set of inputs including spatial and temporal resolutions, and the local terrain and water levels in order to further tackle the complex trade-off created by the resolution of the simulation (both spatial and temporal) and the accuracy of the resulting water movements over the entire simulation area and duration. A comparison can then be made between the performance in terms of this trade-off with the very latest human formulated CA flood modelling rules and those generated by the proposed GPCA system (hypothesis 2)”. Lastly section 6.4 demonstrates the ability of the GPCA system to effectively learn flood modelling rules which can adapt to different spatial

and temporal resolutions, and thusly tackles the final objective 4.5. The GPCA system is shown to be able to create rules which can perform competitively on the complex trade-off between the resolution (in time and space, which leads to processing time), and accuracy of the flooding over the area and duration of the test simulations (shown in Figure 6.17).

The work in this thesis has shown that the GPCA system can effectively learn CA state transition rules for both simple, and real-world CA systems. The GPCA system should therefore be capable of learning other CA models, with a minimal amount of human design for the GP-CA interface/representation for the modelling purpose/environment. Considering the large number of CA applications (demonstrated in section 2.1.2), and the difficulty for human programmer to create distributed CA state transition rules, the GPCA methodology is expected to be a valuable tool for future researchers. Given sufficient training in terms of quantity and quality the GPCA methodology demonstrates an ability to be trained once, and generalise well to other input conditions.

Although real-world CA models are discretisations of time and space, they are modelling the analogue real-world and therefore is a range of spatial and temporal resolutions that can be modelled. Although the spatial and temporal resolutions are static variables across each CA simulation, by training the system on a few examples, the GPCA methodology can create rules which can generalise well to the complex simulations of local cell variables, over a spectrum of different simulation static variable settings (spatial and temporal resolutions). The earlier work in this thesis (Chapter 3:) concludes that primary driver for the computational complexity of CA models is the number of cells and iterations, which directly relates to the temporal and spatial resolution. Therefore, using the GPCA methodology to create CA state transition rules which can make a good approximation of the global model behaviour at lower temporal resolutions given the spatial resolution will create faster models. I.e. by optimising the trade-off between temporal resolution at different spatial resolutions and the accuracy, the trade-off between speed and accuracy of the simulations is optimised. Coupling any acceleration gained through the optimisation and creation of the CA state transition rules, modern many-core hardware in the form GPGPUs can further

reduce the computational processing time of both training and any potential general use of the rules.

7.2 Discussions and future work

The work in this thesis has developed a methodology which allows genetic programming to create local state transitions rules for flood modelling, which guarantee the preservation of mass and uniformity of flow to terrain direction. However, portions of the state transition rule are explicitly pre-programmed. In order to ensure uniformity of flow to terrain direction, the same GP tree is instantiated up to four times, once for each neighbouring cell. While this method proves successful in allowing the system to concentrate on the amount of water movements, this does mean that the system is only developing part of the state transition rule, and still requires an element of human design for the interface system. However, this could be due to the limited amount of information present, and the extra element of learning required to ensure mass preservation and uniformity of flow to terrain direction. Advancement could be made on the system by increasing the radius of the neighbourhood and allowing the system to access more information about what's beyond the current horizon of the neighbourhood. Such a system would require testing for the levels of uniformity of flow to terrain direction and preservation of mass. Such a method would have the advantage of avoiding the two stage system required to ensure that the water levels are preserved across the grid. However, the method established in this thesis can successfully train real world flood state transition rules with limited human design of the interface.

Ultimately it would be desirable to use a system such as the GPCA system, and/or use the rules generated from these training runs, on real world data and/or on data from the full Navier-Stokes equations (for example using openFoam software). There are possible elements of limitations from the target model which could be picked up by the GP training, and it could be the case that the current experiments comparing the best human formulations to one of the best human simplified model might have an element of bias. However, as the target model was run at a much smaller time step in order to generate a more accurate base target model, there may still exist some bias. One problem with using real world data is that there would be an element of uncertainty in the accuracy of the

recorded data, and it would be expected that the GP system would deal with the additional noise well. There is the possibility of slight bias due to the use of a target model, or noise due to uncertainty in the real world data.

There have been a limited number and spread of test cases utilised within this thesis and it would be preferable to have more test cases in order to gain a better idea of the generalisation of the rule sets to different input conditions. This is highlighted by one of the rules having exceptional performance on one of the particular test cases (Bates formula on the Hill and pond test cases at 50m). Also it would be beneficial to better cover the possible variation in inflow type conditions, since it is thought that the greater amount of spatial variation in water levels, coupled with the lack of training examples causes the poorer performance of rules trained on rain conditions.

Finally, it would be interesting to tackle the trade-off problem of the maximum time step at which the rules can generate a reasonable score, as a multi-objective problem, as opposed to simply a single objective problem. There is still room for debate over how to decide which rules are most successful in the cases where one rule dominates another at some resolutions, but is dominated by the other rule at other spatial resolutions.

All the tests in this thesis have been constrained to a fixed time step, however there are advantages to using an adaptive time step factor, although due to the need for a minimum and maximum time step these advantages are still limited. It would be interesting to allow a GP rule to learn the temporal rule which determines the time step as well as the spatial rules which can move the water around the grid. In this way a rule could be optimised to generate the fastest and most accurate rules possible.

Chapter 8: Bibliography

[1] UK Environment Agency, “Benchmarking the latest generation of 2D hydraulic modelling packages,” UK Environment Agency, Bristol, 2013.

[2] U. Frisch, D. d'Humieres, B. Hasslacher, P. Lallmand, Y. Pomeau and J.-P. Rivet, “Lattice Gas Hydrodynamics in Two and Three Dimensions,” *Complex Systems,* vol. 1.4, pp. 649-707, 1987.

[3] R. Denzi, S. Succi and M. Vergassola, “The Lattice Boltzmann equation: Theory and application,” *Physics reports,* vol. 222, no. 3, pp. 145-197, 1992.

[4] M. Gardner, “Mathematical games: The fantastic combinations of John Conway’s new solitaire game “life”,” *Scientific American,* vol. 223.4, pp. 120-123, 1970.

[5] S. Wolfram, “Cellular automata as models of complexity,” *Nature,* vol. 311, no. 5985, pp. 419-424, 1984.

[6] J. R. Koza, Genetic Programming: on the programming of computers by means of natural selection, vol. 1, Cambridge: MIT press, 1992.

[7] W. Banzhaf, Genetic programming : and introduction on the automatic evolution of computer programs and its applications, San Francisco: Morgan Kaifmann publishers, 1998.

[8] J. Von Neumann, Theory of Self-Reproducing Automata, University of Illinois press, Urbana, 1966.

[9] J. Von Neumann, “The general and logical theory of automata,” *Cerebral mechanism in behaviour,* pp. 1-41, 1951.

[10] A. W. Burk, “The General Theory of [complex] Automata [by] John Von Neumann,” University of Illinois Press, Urbana, 1966.

[11] B. McMullin, “John Von Neumann and the Evolutionary Growth of Complexity: Looking Backwards, Looking Forwards...,” *Artificial Life,* vol. 6, no. 4, pp. 347-361, 2000.

[12] E. Sapin, O. Bailleux, J.-J. Chabrier and P. Collet, “A new universal cellular automaton discovered by evolutionary algorithms,” in *GECCO*, Berlin Heidelberg, 2004.

[13] D. Stevens and S. Dragicevic, “A GIS-Based irregular cellular automata model of land-use change,” *Environment and planning B -planning and design,* vol. 34, no. 4, p. 708, 2007.

[14] A. Flache and R. Hegselmann, “Do irregular grids make a difference? Relaxing the spatial regularity assumption in cellular models of social dynamics,” *Journal of Artificial Societies and Social Simulation,* vol. 4, no. 4, 2001.

[15] D. O'Sullivan, “Exploring spatial process dynamics using irregular cellular automaton models,” *Geographical Analysis,* vol. 33, no. 1, pp. 1-18, 2001.

[16] j. Goldenberg, B. Libai and E. Muller, “Using complex systems analysis to advance marketing theory development: Modelling heterogeneity effects on new product growth through stochastic cellular automata,” *Academy of Marketing Science review,* vol. 9, no. 3, pp. 1-18, 2001.

[17] S. J. Steacy and J. McCloskey, “What controls an earthquake's size? Results from a heterogeneous cellular automaton,” *Geophysical Journal International,* vol. 133, no. 1, pp. F11-F14, 1998.

[18] E. R. Berlekamp, J. H. Conway and R. K. Guy, Winning Ways for your Mathematical Plays, New York: Academoic Press, 1982.

[19] C. Langton, “Studying artificial life with cellular automata,” *Physica D: Nonlinear Phenomena,* vol. 22, no. 1, pp. 120-149, 1986.

[20] S. Wolfram, “Universality and complexity in cellular automata,” *Physica D: Nonlinear Phenomena,* vol. 10, no. 1, pp. 1-35, 1984.

[21] S. Szkoda, Z. Koza and M. Tykierko, “Accelerating cellular automata simulations using AVX and CUDA,” ARXIV, 2012.

[22] J. A. Somers, “Direct simulation of fluid flow with cellular automata and the lattice-Boltzmann equation,” *Applied Scientific Research,* vol. 51, no. 1, pp. 127-133, 1993.

[23] Z. Fan, F. Qiu, A. Kaufman and S. Yoakum-Stover, “GPU Cluster for High Performance Computing,” in *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, Washington, DC, USA, 2004.

[24] M. G. B. Johnson, D. P. Playne and K. A. Hawick, “Data-Parallelism and GPUs for Lattice Gas Fluid Simulation,” Massey University, New Zealand, Auckland, 2010.

[25] B. Chopard and M. Droz, Cellular Automata Modeling of Physical Systems, Cambrige University Press UK, 1998.

[26] A. Kolnoochenko, P. Gurikov and N. Menshutina, “General-purpose graphics processing units application for diffusion simulation using cellular automata,” in *21st European Symposium on Computer Aided Process Engineering – ESCAPE 21*, 2011.

[27] M. J. Harris, G. Coombe, T. Scheuermann and A. Lastra, “Physically-Based Visual Simulation on Graphics Hardware,” in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, Saarbrucken, Germany, 2002.

[28] J. L. Guisado, F. Jiménez-Morales and F. Fernández De Vega, “Cellular automata and cluster computing: An application to the simulation of laser dynamics,” *Advances in Complex Systems,* vol. 10, no. supp01, pp. 167-190, 2007.

[29] G. Y. Vichniac, “Simulating physics with cellular automata,” *Physica D: Nonlinear Phenomena,* vol. 10, no. 1-2, pp. 96-116, 1984.

[30] J. Ning, H. Xu, B. Wu, L. Zeng, S. Li and Y. Xiong, “MCA-based Animation of Fracturing Heterogeneous Objects,” in *Computer-Aided Design and Computer Graphics (CAD/Graphics), 2011 12th International Conference on*, Jinan, 2012.

[31] E. Roberts, J. E. Stone, L. Sep´ulveda, W.-M. W. Hwu and Z. Luthey-Schulten, “Long time-scale simulations of in vivo diffusion using GPU hardware,” in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, Rome, 2009.

[32] L. Dematte and D. Prandi, “GPU computing for systems biology,” *Briefings in bioinformatics,* vol. 11, no. 3, 2010.

[33] P. Richmond, D. Walker, S. Coakley and D. Romano, “High performance cellular level agent-based simulation with FLAME for the GPU.,” *Briefings in Bioinformatics,* vol. 11, no. 3, pp. 334 - 347, 2010.

[34] J. Tran, D. Jordan and D. Luebke, “New Challenges for Cellular Automata Simulation on the GPU,” in *SIGGRAPH*, Los Angeles, 2004.

[35] S. Gobron, F. Devillard and B. Heit, “Retina Simulation using Cellular Automata and GPU Programming,” *Machine Vision and Applications,* vol. 18, no. 6, pp. 331-342, 2007.

[36] R. Dolan and D. Guilherme, “GPU-Based Simulation of Cellular Neural Networks for Image Processing,” in *Proceedings of International Joint Conference on Neural Networks*, Atlanta, Georgia, USA, 2009.

[37] S. Gobron and D. Mestre, “Information Visualization of Multi-dimensional Cellular Automata using GPU Programming,” in *Information Visualization, 2007. IV '07. 11th International Conference*, Zurich, 2007.

[38] M. R. L´opez-Torres, J. L. Guisado, F. Jim´enez-Morales and F. Diaz-del-Rio, “GPU-based cellular automata simulations of laser dynamics,” jornadassarteco.org, Seville, Spain, 2012.

[39] S. Gobron, H. Bonafos and D. Mestre, “GPU accelerated computation and visualisation of hexagonal cellular automata,” in *Cellular Automata - Lecture Notes in Computer Science*, vol. 5191, U. Hiroshi, M. Shin, N. Katsuhiro, K. Toshihiko and B. Stefania, Eds., Springer Berlin / Heidelberg, 2008, pp. 512-521.

[40] N. Ferrando, M. A. GosÃ¡lvez, J. CerdÃ¡, R. Gadea and K. Sato, “Octree-based, GPU implementation of a continuous cellular automaton for the simulation of complex, evolving surfaces,” *Computer Physics Communications,* vol. 182, no. 3, pp. 628 - 640, 2011.

[41] C. Kauffmann and N. Piche, “Cellular automaton for ultra-fast watershed transform on GPU,” in *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, Tampa, FL, 2008.

[42] S. Rybacki, J. Himmelspach and A. M. Uhrmacher, “Experiments with Single Core, Multi-core, and GPU Based Computation of Cellular Automata,” in *Advances in System Simulation, 2009. SIMUL '09. First International Conference on*, Porto, 2009.

[43] L. Žaloudek, L. Sekanina and V. Šimek, “Accelerating cellular automata evolution on graphics processing units,” *International Journal on Advances in Software,,* vol. 3, no. 1-2, p. 294–303, 2010.

[44] C. Grelck and F. Penczek, “Design and Implementation of CAOS: An Implicitly Parallel Language for the High-Performance Simulation of Cellular Automata,” in *Cellular Automata - Simplicity Behind Complexity*, A. Salcido, Ed., 2011, pp. 545 - 566.

[45] J. Singler, “Implementation of Cellular Automata using a Graphics Processing Unit,” in *Proceedings of ACM Workshop on General Purpose Computing on Graphics Processors*, Los Angeles, 2004.

[46] J. Drieseberg and C. Siemers, “C to Cellular Automata and Execution on CPU, GPU and FPGA,” in *High Performance Computing and Simulation (HPCS), 2012 International Conference on*, Madrid, Spain, 2012.

[47] A. R. Brodtkorb, T. R. Hagen and M. L. Sætra, “Graphics processing unit (GPU) programming strategies and trends in GPU computing,” *Journal of Parallel and Distributed Computing,* vol. 73, no. 1, pp. 4-13, 2013.

[48] M. J. Gibson, E. Keedwell and D. Savić, “Understanding the efficient parallelisation of Cellular Automata on CPU and GPGPU hardware,” in *Genetic and Evolutionary Conference(GECCO)*, Amsterdam, 2013.

[49] S. Harding and W. Banzhaf, “Fast Genetic Programming and Artificial Developmental Systems on GPUs,” in *21st International Symposium on High Performance Computing Systems and Applications(HPCS'07)*, Saskatoon, SK, 2007.

[50] J.-P. Rennard, Implementation of logical functions in the game of life, Spinger, 2002, pp. 491-512.

[51] L. M. Rocha and W. Hordijk, “Material Representations: From the Genetic Code to the Evolution of Cellular Automata,” *Artificial Life,* vol. 11, no. 1-2, pp. 189 - 214, 2005.

[52] R. Koenig and M. Daniela, “Cellular-Automata-Based Simulation of the Settlement Development in Vienna,” in *Cellular Automata - Simplicity Behind Complexity*, A. Salcido, Ed., 2011, pp. 23-46.

[53] J. G. Hasbani, N. Wijesekara and D. J. Marceau, “An Interactive Method to Dynamically Create Transition Rules in a Land-use Cellular Automata Model,” in *Cellular Automata - Simplicity Behind Complexity*, A. Salcido, Ed., 2011, pp. 3-22.

[54] A. C. Ximenes, C. M. Almeida, S. Amaral, M. I. S. Escada and A. P. D. Aguiar, “Spatial dynamic Modelling of deforestation in the Amazon,” in *Cellular Automata - Simplicity Behind Complexity*, A. Salcido, Ed., 2011, pp. 47-66.

[55] S. D. Gregorio, G. Filippone, W. Spataro and G. A. Trunfio, “Accelerating wildfire susceptaibility mapping through GPGPU,” *Journal of Parallel and Distributed Computing,* vol. 73, no. 8, pp. 1183-1194, 2013.

[56] Y. Guo, G. Walters, S.-T. Khu and E. Keedwell, “Optimal Design of Sewer Networks using hybrid cellular automata and genetic algorithm,” in *Proc., 5th IWA WorldWater congress*, Beijing, China, 2006.

[57] M. Guidolin, A. Duncan, B. Ghimire, M. Gibson, E. Keedwell, A. S. Chen, S. Djordjević and D. Savić, “CADDIES: A New Framework for Rapid Development of Parallel Cellular Automata Algorithms for Flood Simulation,” in *10th International Conference on hydroinformatics, HIC*, Hamburg, Germany, 2012.

[58] M. Bartolozzi and A. W. Thomas, “Stochastic Cellular Automata Model for Stock Market Dynamics,” *Physical Review E,* vol. 69, no. 4, p. 046112, 2004.

[59] G. Qui, D. Kandhai and P. M. A. Sloot, “Understanding the complex dynamics of stock markets through cellular automata,” *Physical Review E,* vol. 75, no. 4, p. 046116, 2007.

[60] H. V. McIntosh, “Wolfram's class IV automata and a good life,” *Physica D: Nonlinear Phenomena,* vol. 45, pp. 105-121, 1990.

[61] A. Gajardo, A. Moreira and E. Goles, “Complexity of Langton's ant,” *Discrete Applied Mathematics,* vol. 117, no. 1, pp. 41-50, 2002.

[62] A. S. Chen, S. Djordjević, J. Leandro and D. A. Savić, “The urban inundation model with bidirectional flow interaction between 2D overland surface and 1D sewer networks,” *NOVATECH,* pp. 465-472, 2007.

[63] N. M. Hunter, M. S. Horritt, P. D. Bates, M. D. Wilson and M. G. Wener, “An adaptive time step solution for raster-based storage cell modelling of floodplain inundation,” *Advances in Water eresources,* vol. 28, no. 9, pp. 975-991, 2005.

[64] P. D. Bates, M. S. Horritt and T. J. Fewtrell, “A simple inertial formulation of the shallow water equations for efficient two-dimensional flood inundation modelling,” *Journal of Hydrology,* vol. 387, no. 1-2, pp. 33-45, 2010.

[65] B. Ghimire, A. S. Chen, M. Guidolin, E. C. Keedwell, s. Djordjević and D. A. Savić, “Formulation of a fast 2D urban pluvial flood model using a cellular automata approach,” *Journal of Hydroinformatics,* vol. 15, no. 3, pp. 676-686, 2012.

[66] F. Dottori and E. Todini, “A 2D flood inundation model based on cellular automata approach,” in *International Conference on Water Resources*, Barcellona, 2010.

[67] J. F. Miller, Cartesian genetic programming, Springer, 2011.

[68] S. L. Harding, J. F. Miller and B. Wolfgang, “Self-modifying cartesian genetic programming,” in *Cartesian genetic Programming*, Springer, 2011, pp. 101-124.

[69] M. F. Brameier and W. Banzhaf, Linear genetic programming, Springer Science and Business Media, 2007.

[70] M. Brameier and W. Banzhaf, “A comparison of linear genetic programming and neural networks in medical data mining,” *Evolutionary Computation, IEEE Transactions on,* vol. 5, no. 1, pp. 17-26, 2001.

[71] A. Abraham and V. Ramos, “Web usage mining using artificial ant colony clustering and linear genetic programming,” in *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, IEEE, 2003, pp. 1384-1391.

[72] M. Oltean and C. Grosan, “A comparison of several linear genetic programming techniques,” *Complex Systems,* vol. 14, no. 4, pp. 285-314, 2003.

[73] S. Luzia Vidal de and P. T. Aurora, Genetic programming and Boosting Technique to Improve Time Series Forecasting, Intech, 2009.

[74] [Online]. Available: http://geneticprogramming.us/Genetic_Operations.html. [Accessed 24 feb 2015].

[75] E. De Jong, R. Watson and J. Pollack, “Reducing bloat and promoting diversity using multi-objective methods,” Citeseer, 2001.

[76] S. Bleuler, M. Brack, L. Thiele and E. Zitzler, “Multi-objective genetics programming: Reducing bloat using SPEA2,” *Evolutionary Computation, 2001. Proceeding of the 2001 Congress on,* pp. 536-543, 2001.

[77] R. Poli, “A simple but theoretically-motivated method to control bloat in genetic programming,” in *Genetic Programming*, Springer, 2003, pp. 204-217.

[78] S. Luke, Issues in scaling genetic programming: breeding strategies, tree generation, and code bloat, research directed by Dept. of Computer Science.University of Maryland, College Park, 2000.

[79] W. B. Langdon, “Size fair and homologous tree crossovers for tree genetic programming,” *Genetic programming and evolvable machines,* vol. 1, no. 1-2, pp. 95-119, 2000.

[80] W. B. Langdon and W. Banzhaf, “Genetic Programming bloat with semantics,” *Parallel Problem Solving from from nature PPSN VI,* pp. 201-210, 2000.

[81] W. B. Langdon and R. Poli, Genetic programming bloat with dynamic fitness, Springer, 1998.

[82] S. Luke and L. Panait, “A comparison of bloat control methods for genetic programming,” *Evolutionary Computation,* vol. 14, no. 3, pp. 309-344, 2006.

[83] D. R. White, J. McDermott, M. Castelli, L. Manzoni, B. W. Goldman, G. Kronberger, W. Jaśkowski, U.-M. O'Reilly and S. Luke, “Better GP benchmarks: community survey results and proposals,” *Genetic Programming Evolvable Machines,* vol. 14, no. 1, pp. 3-29, 2013.

[84] M. O'Neill, L. Vanneschi, S. Gustafson and W. Banshaf, “Open issues in genetic programming,” *Genetic Programming and Evolvable Machines,* vol. 11, pp. 339-363, 2010.

[85] J. H. Moore, P. C. Andrews, N. Barney and B. C. White, “Development and evaluation of an open-ended computational evolution system,” in *Evolutionary Computation, Machine Learning and Data Mining in Bioinfomatics*, Springer, 2008, pp. 129-140.

[86] P. Domingos, “The role of Occam's razor in knowledge discovery,” *Data mining and knowledge discovery,* vol. 3, no. 4, pp. 409-425, 1999.

[87] S. Silva and L. Vanneschi, “Operator equalisation, bloat and overfitting: a study on human oral bioavailability prediction,” in *Proceeding of the 11th Annual conference of Genetic and evoltionary computation*, 2009, ACM, 2009, pp. 1115-1122.

[88] J. P. Rosca, “Towards automatic discovery of building blocks in genetic programming,” in *Working Notes for the AAAI Symposium on Genetic Programming*, vol. 445, MIT, Cambridge, MA, USA, 1995, pp. 78-85.

[89] D. Andre, F. H. Bennett III and J. R. Koza, “Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem,” in *Proceedings of the First Annual Conference on Genetic Programming*, MIT Press, 1996, pp. 3-11.

[90] J. R. Koza, “Discovery of rewrite rules in Lindenmayer systems and state transition rules in cellular automata via genetic programming,” *Symposium on pattern formation (SPF-93), Claremont, California,* 1993.

[91] D. Safia, D. Oussama and B. M. Chawki, “Image segmentation using continuous cellular automata,” *Programming and Systems (ISPS), 2011 10th International Symposium on,* pp. 94-99, 2011.

[92] M. Michell, J. P. Crutchfield and P. T. Hraber, “Evolving cellular automata to perform computations: Mechanisms and impediments,” *Physica D: Nonlinear phenomena,* vol. 75, no. 1, pp. 361-391, 1994.

[93] R. Das, M. Mitchell and J. P. Crutchfield, “A genetic algorithm discovers particle-based computation in cellular automata,” in *Parallel problem solving from nature-PPSN III*, Springer, 1994, pp. 344-353.

[94] M. Sipper, “Co-evolving non-uniform cellular automata to perform computations,” *Physica D: Nonlinear Phenomena,* vol. 92, no. 3, pp. 193-208, 1996.

[95] F. Wu, “Calibration of stochastic cellular automata: the application to rural-urban land conversions,” *International journal of geographical information science,* vol. 16, no. 8, pp. 795-818, 2002.

[96] S.-U. Guan and S. K. Tan, “Pseudorandom number generation with self-programmable cellular automata,” *Computer-Aided Designed of*

integrated Circuits and systems, IEEE Transactions on, vol. 23, no. 7, pp. 1095-1101, 2004.

[97] L. O. Chua and L. Yang, “Cellular Neural networks: Applications,” *Circuits and Systems, IEEE Transactions on,* vol. 35, no. 10, pp. 1273-1290, 1988.

[98] F. Werblin, T. Roska and L. O. Chua, “The analogic Cellular neural network as a bionic eye,” *International journal of circuit Theory and Applications,* vol. 23, no. 6, pp. 541-569, 1995.

[99] J. D. Owens, M. Houston, D. Leubke, S. Green, J. E. Stone and J. C. Phillips, “GPU computing,” *Proceeding of the IEEE,* vol. 96, no. 5, pp. 879-899, 2008.

[100] “Nvidia CUDA toolkit 6.5,” Nvidia, 11 Feb 2015. [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/#axzz3RS9dXtPY. [Accessed 11 Feb 2015].

[101] M. J. Gibson, E. C. Keedwell and D. A. Savić, “An investigation of the efficient implementation of cellular automata on multi-core CPU and GPU hardware,” *Journal of arallel and Distributed Computing,* 2014.

[102] N. Corporation, “K40 specification,” Nvidia Corporation, 2014. [Online]. Available: http://www.nvidia.com/object/tesla-servers.html. [Accessed May 2014].

[103] Nvidia, “Nvidia Titan Z specifications,” Nvidia. [Online]. [Accessed 28 March 2015].

[104] “The Open University,” [Online]. Available: http://www.open.edu/openlearnworks/mod/oucontent/view.php?id=608&section=2.2. [Accessed 28 March 2015].

[105] Nvidia, “GPU GEMS2 Chapter 35,” Nvidia, 28 March 2015. [Online]. Available: http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter35.html. [Accessed 28 March 2015].

[106] “The OpenMP Architecture Review Board Home Page,” The OpenMP Architecture Review Board, 2012. [Online]. Available: http://openmp.org/. [Accessed August 2012].

[107] “OpenCL 1.2 Specification,” Khronos group, 2012. [Online]. Available: http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf. [Accessed March 2012].

[108] Nvidia, “GeForce 8600M specification,” Nvidia, 2012. [Online]. Available: http://www.nvidia.com/object/geforce_8600M.html. [Accessed 2012].

[109] Nvidia, “GeForce GTX 560 Ti,” Nvidia, 2012. [Online]. Available: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-560ti/specifications. [Accessed 2012].

[110] P. D. Bates and A. P. J. De Roo, “A simple raster-based model for flood inundation simulation,” *Journal of Hydrology,* vol. 236, no. 1-2, pp. 54-77, 2000.

[111] N. M. Hunter, P. D. Bates, M. S. Horritt and M. D. Wilson, “Simple spatially-distributed models for predicting flood inundation: A review,” *Geomorphology,* vol. 90, no. 3-4, pp. 208-225, 2007.

[112] Nvidia, “Nvidia Tesla K20 specifications,” Nvidia, 6 May 2015. [Online]. Available: http://www.nvidia.co.uk/content/PDF/kepler/Tesla-K20-Passive-BD-06455-001-v07.pdf. [Accessed 6 May 2015].

[113] Nvidia, “CUDA Faq,” Nvidia, 18 April 2015. [Online]. Available: https://developer.nvidia.com/cuda-faq. [Accessed 18 April 2015].

[114] N. Whitehead and A. Fit-Florea, “Precision & Performance: Floating Point and IEEE 754 Compliance for NVIDIA,” Nvidia, 2011.

Chapter 9: Appendices

9.1 Appendix 1: The power function, differences on CPU and GPU hardware

9.1.1 Introduction

It is known that the power for different CPU and GPU hardware that the power function will produce slightly different results for some inputs [113] [114]. Due to the limited precision presented by floating point numbers in representing real valued numbers, not all real values can be accurately represented by a finite number of bits. An example given in Literature is that of the real value of 2/3rd, which represented as a binary value 0.10101010… to an infinite number of bits after the binary point. Therefore the binary representation of the real value 2/3rd must be rounded, where the rounding modes are specified by the IEEE 754 standard for binary floating point arithmetic [114]. The IEEE 754 standard requires support for only a handful of operations, these include the arithmetic operations add, subtract, multiply, divide, square root, fused-multiply-add, remainder, conversion operations, scaling, sign operations and comparisons. The results of these operations are guaranteed to be the same for all implementations of the standard, for a given format and rounding mode. However more complex functions like the power (x, y) operator, which raises the value x to the power of the value y, are not guaranteed to produce the exact same results on the differing hardware's. This has an affect can be multiplied by the heuristic optimisation process of the CAGP system, and therefore a specific set of experiments are carried out to determine the level of this affect.

9.1.2 Experimental setup

The version of the build created to be compatible with the GPU, is used for both the CPU and GPU experiments contained in this section. I.e. the GPU implementation requires the GP tree to be evaluated by a loop with a fixed sized stack, as opposed to be means of recursion. This mean exact same code is used for both of these tests, at the point of the evaluation of the GP trees, barring the protection of the power function. This protection is once again implementation specific, in that attempts are made to encode the power function to capture any spurious results including NaN (Not a Number), and +/-infinity at the operator

level, and return a value of zero. However, this does not include denormals which are very small number which are close to zero, are not captured until after the evaluation of the entire CA and error score. This is due to the fact that OpenCL does not provide a method in its API to capture these denormal values.

In these experiments, two identical populations are created by using the exact same seed value for the optimisation process. Due to the difference in the fitness scores, the heuristic optimisation will likely select some differing individuals during it process and therefore only the first generation can be guaranteed to be the same GP trees. Since both final populations are appearing to result in different optimisations, i.e. different resulting GP trees, and evaluation of each population is made by running each population on the alternate hardware. These experiments are conducted using the Hill and Pond test case, at 50m, with the original rain fall profile of 10mm/hour for 1 hour, where the simulations are run for 4 hours of simulation time, at 0.5, 1, and 2 second time steps.

9.1.3 Experimental Results

Out of the 100 identical individuals produced in the first generation, 5 of these failed to agree on the fitness scores. It is noted that these 5 that disagree and within the 6 worst of the population, and that this instantly creates a different ordering of the population.

Table 9.1, Fitness scores (1/average RMSE), of the 6 worst cases of the two population of the different CPU and GPU hardware. The one fitness score highlighted between the two implementations has scored the same value but is placed in a different position.

GP Tree	**95**	**96**	**97**	**98**	**99**	**100**
CPU Fitness score	16.6039	16.329	14.5613	**11.335**	11.0148	6.51328
GPU Fitness score	16.621	16.3288	14.5627	11.682	**11.335**	5.71232
absolute difference	0.0171	0.0002	0.0014	0.347	0.3202	0.80096

This small difference in the population to start off with (shown in Table 9.1), coupled with the continued small difference in the interpretation of the power function, results in populations as shown by the resulting optimisation scores in Figure 9.1. Where there is only a small divergence in the beginning of the process, these later results in quite a stark difference.

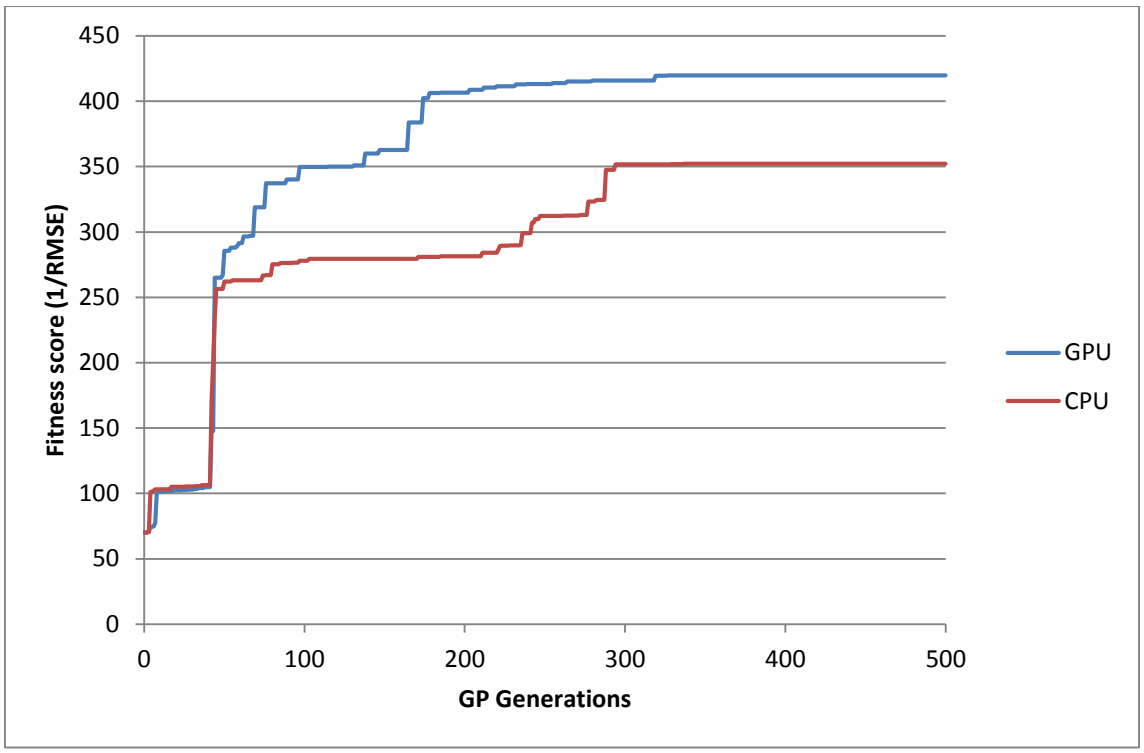


Figure 9.1, Fitness scores (1/average RMSE) of the fittest individual in each population, run on the CPU and GPU. Where the fitness scores are calculated by the respective hardware during each optimisation run.

It can be seen in Figure 9.2 that when comparing the final two populations from each optimisation runs on the alternate hardware, that a larger number and variety of rankings of GP have interpretation differences, than compared to that of the first generation (which is identical for both runs).

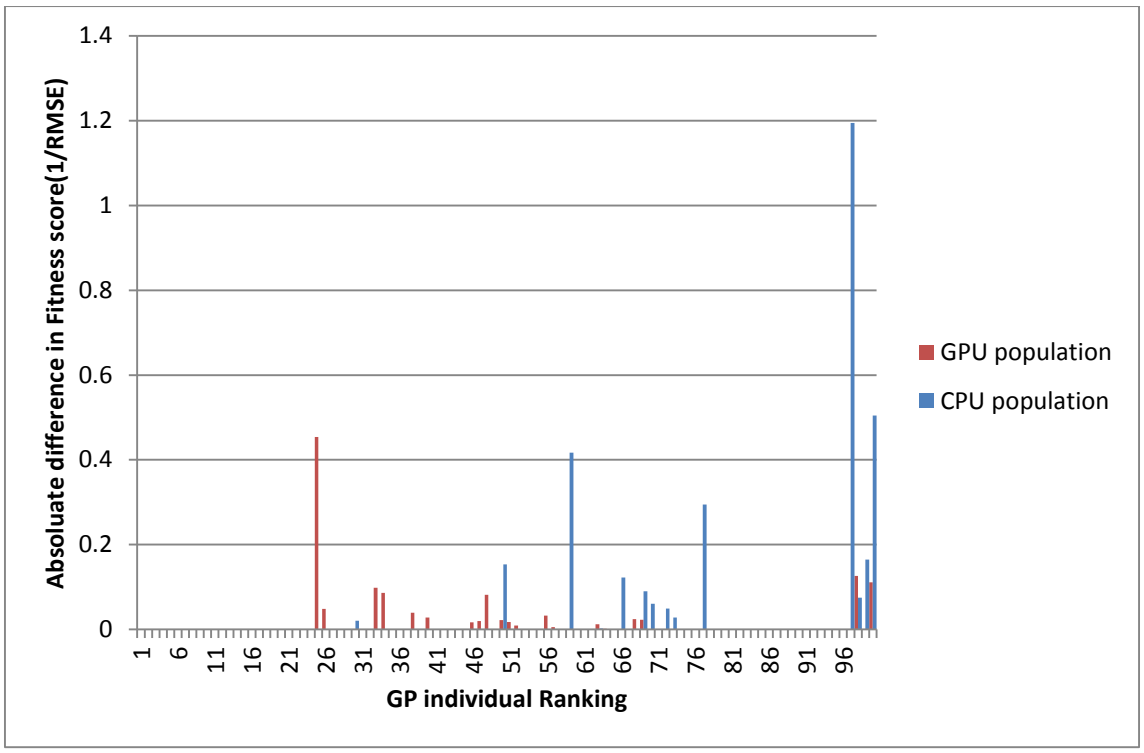


Figure 9.2, Absolute difference between the fitness scores of the each final population evaluated on the alternate hardware, where the x-axis represents the ranking of the GP individual within the population.

9.2 Appendix 2: The simple GP language

In order to make the construction of GP formulae easier, and to ensure that there are less errors in those formulations, a simple recursive decent compiler is created. This is designed to be a scaled version of C, and therefore accepts C style comments, both multi-line comments (/* … comments …*/), and in-line comments (lines beginning with //). This is designed to more easily facilitate the human construction of complex GP decision trees, as opposed to attempting to program in reverse polish notation. The specification for this language is shown in Figure 9.3.

```
program     = "GP" "=" condition";"

condition   = expression { ( "<", ">", "==", "&&", "||" ) expression }

expression  = term { ( "+" | "-" ) term }

term        = factor { ( "*" | "/" | "^" ) factor }

factor      = variable | number |"(" condition ")" | function

function    = ( dualFunc, trippleFunc )

dualFunc    = ( "min", "max", "sin", "cos" "pow" ) "(" condition ","
                  condition ")"

trippleFunc = "if" "(" condition "," condition "," condition ")"

variable    = any token starting with a letter(alpha), then followed
              by other letters, numbers, "_" under scores, excluding
              above key words.
              Must one of the predefined variables.

num         = any token starting with 0-9
```

Figure 9.3, Specification of the simple recursive decent language used in this thesis to specify GP decision trees.

9.3 Appendix 3: Extended training with GP for temporal generalisation of CA rules

9.3.1 Introduction

Having trained in section 6.2, on the Hill and pond test case at 50m, with a 0.5, 1, and 2 second time steps, in order to gain generalisation over the different time steps, for a limited 500 GP generations. It is difficult to determine how well the system would be able to optimise the rules set given more optimisation time, therefore these extended tests use the same experimental settings except the termination criteria is set to 2,500 GP generations.

9.3.2 Experimental set-up

The hill and pond training case is again utilised for 4 hours of simulation time with 0.5 seconds, 1 second and 2 seconds time steps, where the fitness is established as the reciprocal of the mean of various simulations RMSE. Once again 10 differently seeded populations are utilised, and differently seeded to those 10 populations in section 6.2.

9.3.3 Training results

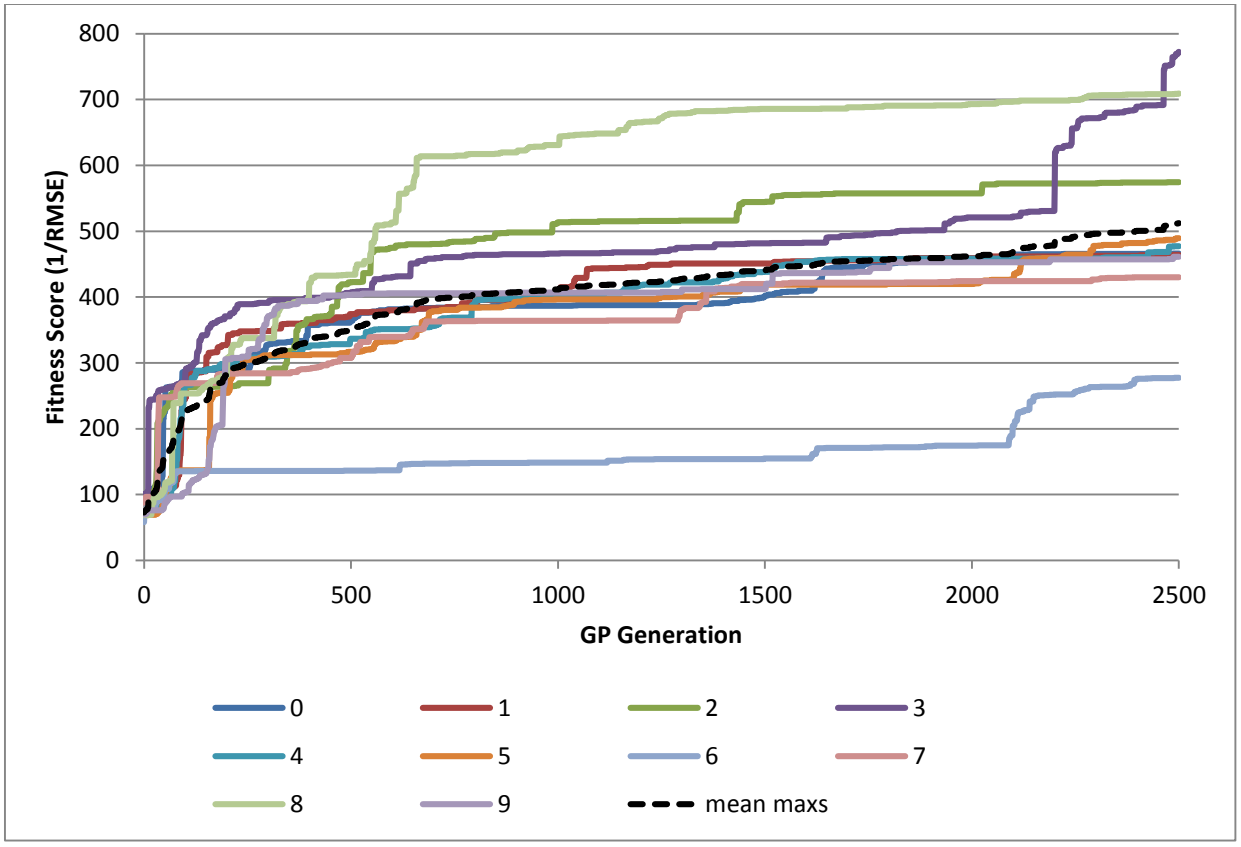


Figure 9.4, Fitness of the fittest individual within each of the 10 populations, trained on hill and pond test case at 50m cell size, and 0.5, 1, and 2 second time steps.

It is clear to see in Figure 9.4, that while the majority of the optimisation is carried out within 500 GP generations, that the system will continue to optimise after this point. Some generation (for example population 3) can even make large jump towards the end of this 2,500 GP generations. However, there are clear example, (for example population 6) which appear to get stuck in a local fitness maxima very early on and remain stuck in this very poor area for a prolonged period, although the majority of populations perform much better, and towards the end of this optimisation does begin to improve.

9.3.4 Conclusions

While a termination criterion of 500 GP generations has been used in the majority of tests within this thesis, this has been done primarily to perform even and fair comparisons between the different optimisation settings with the same length of optimisations. However it is clear from Figure 9.4 results that optimisation will continue at a slower rate after this point, and therefore the

generalisation results in previous sections could be further improved. There maybe be a point during the length of optimisation which is more likely to have the maximum amount of training potency in that it maximises the generalisation properties and minimises the possibility of over training; however, it is also possible that further training continues to increase the likelihood of good generalisation properties as it is a local rule which is trained, further experimentation is required in this area.

This page is intentionally left blank.