



UNIVERSITÀ DELLA CALABRIA



# UNIVERSITÀ DELLA CALABRIA

Dipartimento di Matematica e Informatica

## Dottorato di Ricerca in Matematica e Informatica

*con il contributo del*

Fondo Sociale Europeo - POR Calabria FSE 2007/2013

XXVI CICLO

---

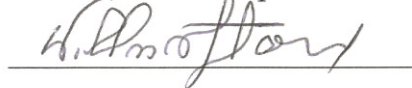
### PARALLEL AND EVOLUTIONARY APPLICATIONS TO CELLULAR AUTOMATA MODELS FOR MITIGATION OF LAVA FLOW INVASIONS

Settore Disciplinare INF/01 – INFORMATICA

**Coordinatore:** Ch.mo Prof. Nicola Leone



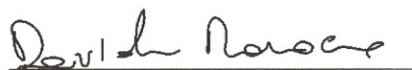
**Supervisor:** Prof. William Spataro



Prof. Donato D'Ambrosio

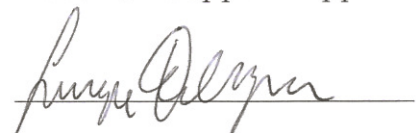


Prof. Davide Marocco





**Dottorando:** Dott. Giuseppe Filippone



This work has been funded with support from the European Commission, European Social Fund (ESF) and from the Regione Calabria (Italy). The author is the only responsible for this thesis and the European Commission and the Regione Calabria have no responsibility for the use that may be made of the information contained therein.

La presente tesi è cofinanziata con il sostegno della Commissione Europea, Fondo Sociale Europeo e della Regione Calabria. L'autore è il solo responsabile di questa tesi e la Commissione Europea e la Regione Calabria declinano ogni responsabilità sull'uso che potrà essere fatto delle informazioni in essa contenute.

*Alla mia famiglia*

# Abstract

In the lava flow mitigation context, the determination of areas exposed to volcanic risk is crucial for diminishing consequences in terms of human casualties and damages of material properties. In order to mitigate the destructive effects of lava flows along volcanic slopes, the building and positioning of artificial barriers is fundamental for controlling and slowing down the lava flow advance.

In this thesis, a decision support system for defining and optimizing volcanic hazard mitigation interventions is proposed. The Cellular Automata numerical model SCIARA-fv2 for simulating lava flows at Mt Etna (Italy) and Parallel Genetic Algorithms (PGA) for optimizing protective measures construction by morphological evolution have been considered.

In particular, the PGA application regarded the optimization of the position, orientation and extension of earth barriers built to protect Rifugio Sapienza, a touristic facility located near the summit of the volcano.

A preliminary release of the algorithm, called single barrier approach (SBA), was initially considered. Subsequently, a second GA strategy, called Evolutionary Greedy Strategy (EGS), was implemented by introducing multi-barrier protection measures in order to improve the efficiency of the final solution. Finally, a Coevolutionary Cooperative Strategy (CCS), has been introduced where all barriers are encoded in the genotype and, because all the constituents parts of the solution interact with the GA environment, a mechanism of cooperation between individuals has been favored. Solutions provided by CCS were extremely efficient and, in particular, the extent of the barriers in terms of volume used to deviate the flow thus avoiding that the lava reaches the inhabited area was less than 72% respect to the EGS

and 284% respect to the SBA. It is also worth to note that the best set of interventions provided by CCS was approximately eighteen times more efficient than the one applied to divert the lava flow away from the facilities during the 2001 Mt.Etna eruption.

Due to the highly intensive computational processes involved, General-Purpose Computation with Graphics Processing Units (GPGPU) is applied to accelerate both single and multiple simultaneous running of SCIARA-fv2 model using CUDA (Compute Unified Device Architecture). Using four different GPGPU devices, the study also illustrates several implementation strategies to speedup the overall process and discusses some numerical results obtained. Carried out experiments show that significant performance improvements are achieved with a parallel speedup of 77.

Finally, to support the analysis phase of the results, an OpenGL and Qt extensible system for the interactive visualization of lava flows simulations was also developed. The System showed that it can run the combined rendering and simulations at interactive frame rate.

The study has produced extremely positive results and represents, to our knowledge, the first application of morphological evolution for lava flow mitigation.

# Sommario

Nel contesto reattivo alla mitigazione del rischio vulcanico, la determinazione di aree esposte al rischio è di fondamentale importanza per la valutazione della pericolosità per l'uomo e le infrastrutture. La costruzione e l'opportuno posizionamento di barriere artificiali è cruciale per il controllo e il rallentamento del fronte lavico.

Nella presente tesi viene definito un sistema di supporto alle decisioni per la realizzazione ed ottimizzazione di opere di protezione da colata lavica. Per la simulazione di flussi lavici è stato utilizzato il modello ad Automi Cellulari SCIARA - fv2 e per l'evoluzione morfologica di opere protettive sono state applicate tecniche di ottimizzazione basate su Algoritmi Genetici Paralleli (AGP). In particolare, gli esperimenti sono stati condotti considerando l'evento lavico del 2001 avvenuto sul Mt.Etna che ha minacciato il rifugio Sapienza, una nota struttura turistica situata ad oltre 1800 metri di quota.

La prima versione del modello implementato, definita come approccio a barriera singola (ABS), fornisce opere di protezione costituite da due nodi. Le limitazioni relative all'approccio basato su singola barriera hanno portato, successivamente, alla realizzazione di una seconda strategia, definita come strategia evolutiva golosa (SEG), che introduce opere multibarriera al fine di rendere più efficienti le soluzioni finali. Il terzo e ultimo approccio è, invece, definito come strategia cooperativa evolutiva (SCC), dove l'insieme di barriere che costituisce la soluzione finale è interamente codificato all'interno del genotipo. Per promuovere l'interazione delle parti costituenti all'interno dell'ambiente dell'AG, inoltre, è stato favorito un meccanismo di cooperazione fra individui. Le soluzioni fornite dall'algoritmo in questione sono state estremamente efficienti, a considerazione del fatto che il volume to-

tale utilizzato dalla migliore soluzione evoluta dall'algoritmo evolutivo per difendere l'Area di Protezione è risultato essere inferiore del 72% rispetto alla tecnica golosa e del 284% rispetto all'approccio a singola barriera. È, inoltre, importante notare come la migliore soluzione fornita dalla strategia SCC sia risultata approssimativamente 18 volte più efficiente rispetto all'insieme di interventi (13 barriere) applicati per deviare i flussi lavici dal Rifugio Sapiaza durante l'eruzione etnea del 2001.

La valutazione della funzione di fitness, durante il processo evolutivo dell'algoritmo genetico, ha richiesto un uso massiccio del simulatore numerico mediante l'esecuzione di migliaia di simulazioni concorrenti. Tale processo, dato l'elevato carico computazionale, ha suggerito l'utilizzo del calcolo ad alte prestazioni. A tal fine, tecniche di GPGPU (General-Purpose Computation with Graphics Processing Units) sono state applicate per accelerare simulazioni singole e simultanee del modello SCIARA-fv2 in ambiente e linguaggio CUDA (Compute Unified Device Architecture). Diverse strategie sono state sviluppate per ridurre il tempo totale di esecuzione. I risultati ottenuti, in riferimento a quattro differenti dispositivi grafici, hanno mostrato dei significativi miglioramenti nelle prestazioni, rispetto alla versione sequenziale del modello, ottenendo uno speedup di 77.

Inoltre, per supportare la fase di analisi dei risultati, in questo lavoro è stato sviluppato un sistema di visualizzazione interattiva di simulazioni di flussi lavici implementato in linguaggio C++ e OpenGL ed integrato in interfaccia Qt. Il framework implementato è stato applicato con successo per la visualizzazione di simulazioni di flussi lavici e l'utilizzo di server multi-GPU dedicati ha, inoltre, offerto la possibilità di accelerare il processo di visualizzazione, garantendo, di fatto, di eseguire visualizzazione e simulazione in real-time.

Lo studio in questione, che rappresenta la prima applicazione di evoluzione morfologica per la mitigazione del rischio indotto da flussi lavici, ha prodotto risultati estremamente positivi.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Genetic Algorithms</b>	<b>5</b>
2.1	Overview . . . . .	6
2.2	A brief history of GAs . . . . .	7
2.3	Elements of GA in the Holland's Model . . . . .	8
2.3.1	Proportional selection . . . . .	10
2.3.2	Crossover . . . . .	11
2.3.3	Mutation . . . . .	11
2.4	Variants of the Holland's Model . . . . .	12
2.4.1	Encoding and genetic operators . . . . .	12
2.4.2	Selection methods and elitism . . . . .	14
2.5	Multiobjective GAs . . . . .	16
2.6	Theoretical foundations of Genetic Algorithms . . . . .	18
2.7	Application of GAs . . . . .	20
2.8	GAs in Geomorphology . . . . .	21
2.9	Conclusions . . . . .	22
<b>3</b>	<b>Cellular Automata</b>	<b>23</b>
3.1	A brief history of Cellular Automata . . . . .	24
3.2	Informal definition of Cellular Automata . . . . .	24
3.2.1	Dimension and geometry of Cellular Automata . . . . .	25
3.2.2	Number of states of a cell . . . . .	25
3.2.3	Relationship of closeness . . . . .	26
3.2.4	State-transition function . . . . .	27
3.3	Formal definition of Cellular Automata . . . . .	27
3.4	Theory of Cellular Automata . . . . .	28
3.4.1	One-dimensional Cellular Automata . . . . .	28
3.4.2	Universality and complexity in Cellular Automata . . . . .	29
3.4.3	Chaos theory . . . . .	31
3.4.4	Other theoretical works on Cellular Automata . . . . .	32



3.5	CA applications . . . . .	33
3.5.1	Artificial life with CA . . . . .	33
3.5.2	Lattice Gas Cellular Automata and Lattice Boltzman Models . . . . .	34
3.5.2.1	Lattice Gas cellular automata . . . . .	34
3.5.2.2	Lattice Boltzmann Models . . . . .	36
<b>4</b>	<b>Modelling Macroscopic phenomenas with Cellular Automata</b>	<b>39</b>
4.1	Complex Cellular Automata . . . . .	40
4.2	Modelling surface flows throug CA . . . . .	42
4.2.1	The Minimization Algorithm of the Differences . . . . .	42
4.3	The Cellular Automata Model SCIARA-fv2 . . . . .	43
4.3.1	The lava flows coumputation (elementary process $\tau_1$ ) . . . . .	46
4.3.2	Temperature variation and lava solidification compu- tations (elementary process $\tau_2$ ) . . . . .	49
4.4	SCIARA-fv2 model applications . . . . .	50
4.5	Discussion . . . . .	53
<b>5</b>	<b>Accelerating Cellular Automata simulations of lava flows</b>	<b>55</b>
5.1	The NVIDIA CUDA programming approach: a quick overview	57
5.1.1	CUDA Threads and Kernels . . . . .	58
5.1.2	Memory hierarchy . . . . .	60
5.1.3	Programming with CUDA C . . . . .	62
5.2	Implementation of the SCIARA-fv2 model . . . . .	62
5.2.1	The Naive implementation: all global memory usage . . . . .	63
5.2.2	Shared Memory Usage . . . . .	66
5.2.3	Dynamic extension of the kernels grid . . . . .	68
5.2.4	Tests and performance results . . . . .	71
5.2.4.1	Numerical verification of experiments . . . . .	72
5.3	Efficient GPGPU application for large number of concurrent lava flow simulations . . . . .	73
5.4	The methodology for defining hazard maps . . . . .	74
5.5	GPGPU-Based Lava Flow Risk Assessment . . . . .	76
5.5.1	A case study and performed simulations . . . . .	78
5.5.2	A Naive implementation: the Whole Cellular Space Implementation . . . . .	78
5.5.3	The Rectangular Bounding Box Strategy: the Dynamic Grid Implementation . . . . .	81
5.6	Conclusions . . . . .	83

---

<b>6</b>	<b>An Interactive Visualization System for Lava Flows Cellular Automata Simulations</b>	<b>85</b>
6.1	The CCAFramework System Architecture . . . . .	86
6.2	Overview of CCAFramework . . . . .	87
6.3	Discussion . . . . .	88
<b>7</b>	<b>A new methodology for mitigation of lava flow invasion hazard</b>	<b>91</b>
7.1	A brief history of mitigation actions in Mt.Etna . . . . .	92
7.2	The Case Study: The 2001 Mt Etna Eruption . . . . .	93
7.3	Morphological Evolution of protective works through Parallel Genetic Algorithms . . . . .	96
7.3.1	Parallel implementation and performance . . . . .	99
7.3.2	Single Barrier Approach: experiments and results . . .	102
7.3.2.1	Consideration on the GA dynamics and emergent behaviors . . . . .	104
7.3.3	Evolving multiple barrier solutions . . . . .	105
7.3.3.1	Evolutionary Greedy Strategy . . . . .	106
7.3.3.2	Cooperative Co-evolutionary Strategy . . . . .	110
7.3.4	Qualitative analysis of the results . . . . .	116
7.3.5	Conclusions . . . . .	118
<b>8</b>	<b>Conclusions</b>	<b>121</b>
	<b>Acknowledgments</b>	<b>125</b>
	<b>Bibliography</b>	<b>126</b>
	<b>List of Figures</b>	<b>144</b>
	<b>List of Tables</b>	<b>148</b>



# Chapter 1

## Introduction

Among several approaches for modelling natural complex phenomena proposed in the literature, Cellular Automata (CA) represent a possible solution when the phenomena to be simulated evolve on the basis of local interactions of their constituent parts. CA are dynamical systems, discrete in space and time. They can be thought of as a lattice of cells, each one embedding an identical finite automaton, interacting only with a small set of neighboring cells. The state of each finite automaton is changed by applying the transition function, which defines local rules of evolution for the cell. The overall CA global dynamic emerges from the simultaneous application of the local rules to each cell. CA have been applied with success to different fields such as pattern recognition [111, 50], cryptography [163, 148, 93] or image processing [142, 143]. However, major interest for CA is regarding their use to model Complex Systems in various fields like Physics, Engineering and Biology [24, 2, 36, 45, 56, 97].

As regards the modeling of natural complex phenomena, Complex Cellular Automata (CCA) can represent a valid methodology to model numerous complex non-linear phenomena [57], such as lava and debris flows. CCA are an extension of classical CA, developed for overcoming some of the limitations affecting conventional CA frames such as the modelling of large scale complex phenomena. Due to their particulate nature and local dynamics, CCA are very powerful in dealing with complex boundaries, incorporating microscopic interactions and parallelization of algorithms.

In the risk assessment of lava flow context, the use of thematic maps of volcanic hazard is of importance in supporting policy managers and administrators in effective land use planning and in taking proper actions that are required during an emergency phase. In particular, hazard maps are a key tool for emergency management since they describe the threat that can be expected at a certain location for future eruptions. At Mt. Etna (Italy), the

most active volcano in Europe, the majority of events that have occurred in the last four centuries report damage to human properties in numerous towns on the volcano flanks [10]. Notwithstanding, the susceptibility of the Etnean area to lava invasion has increased due to continued urbanization [60], the inevitable consequence that new eruptions may involve even greater risks. Current efforts for hazard evaluation and contingency planning in volcanic areas draw heavily on hazard maps and numerical simulations for the purpose of individuating affected areas in advance.

Although many computational modelling methods for lava flow simulation [88, 117, 2, 52] and related techniques for the compilation of susceptibility maps are already known to the international scientific community, the problem of defining a standard methodology for the construction of protection works, in order to mitigate volcanic risk, remains open. Techniques to slow down and divert lava flows, caused by collisions with protective measures such as artificial barriers [6, 29, 109] or dams [7], are now to be considered empirical, exclusively based on past experiences. In order to mitigate the destructive effects of lava flows along volcanic slopes, the building of artificial barriers is fundamental for controlling and slowing down the lava flow advance. Such protective interventions were trialled during a few recent eruptions of Etna: in 1983, 1991-1993, 2001 and in 2002, when earthen barriers were built to control lava flow expansion with different levels of success. The proper positioning of protective measures in the considered area may depend on many factors (viscosity of the magma, output rates, volume erupted, steepness of the slope, topography, economic costs). As a consequence, in this context, one of the major scientific challenges for volcanologists is to provide efficient and effective solutions.

Morphological evolution is a recent development within the field of engineering design, by which evolutionary computation techniques are used to tackle complex design projects. This branch of evolutionary computation is also known as evolutionary design and is a multidisciplinary endeavour that integrates concepts from evolutionary algorithms, engineering and complex systems to solve engineering design problems [12]. Morphological evolution has been largely explored in evolutionary robotics, both for the design of imaginary 3D robotics bodies [149] and for the efficient and autonomous design of adaptive moving robots [17, 108]. Principles of evolutionary design have been also applied in structural engineering at a different level of the design process, from the structural design itself to the logistics involved in the construction [96].

While Genetic Algorithms (GAs) have been applied several times in the past for optimizing CA models [46, 45], this thesis work describes the application of Parallel Genetic Algorithms (PGAs) for optimizing earth barriers

---

construction by morphological evolution to control lava flows. GAs [84] are general-purpose iterative search algorithms inspired by natural selection and genetics. They simulate the evolution of a population of candidate solutions to a specific problem by favouring the reproduction of the best individuals and have been applied, with good results, in many different fields: for the solution of difficult combinatorial problems [76] in the study of the interaction between evolution and learning [81]; in evolutionary robotics [124, 63]. GAs have also been used for improving the performance of CA in resolving difficult computational tasks: cellular-automata based solution of binary classification problem by applying GAs [129]; asynchronous CA evolution to face similar problems [164]. GAs based methods have also been applied to CA for modelling bioremediation of contaminated soils [58] and for the optimisation of lava and debris flow simulation models (e.g., [152, 87, 141, 44, 43]).

For the morphological evolution of protective works by PGAs, the latest release of the lava flows simulation CA model SCIARA was adopted. SCIARA is a family of bi-dimensional CCA lava flow models, successfully applied to the simulation of many real cases such as the 2001 Mt. Etna (Italy) Nicolosi lava flow [36] and the 1991 Valle del Bove (Italy) lava event [8], which occurred on the same volcano and was employed for risk mitigation.

To evaluate the effectiveness and efficiency of the methodology developed in this thesis, the 2001 Mt Etna eruption case study was considered. The 2001 eruption of Mt. Etna, which began on July 17th, caused damage and threatened some important facilities and infrastructure. Despite the construction of thirteen artificial barriers during the eruption, the flow emitted from the lower vent interrupted the SP92 road and invaded a wide parking area located between MTs. Silvestri and the Sapienza zone.

The GA fitness evaluation has implied a massive use of the numerical simulator running thousands of concurrent simulations for every generation computation. Depending on the adopted computer framework, such an operation may require several hours or even months. Due to the high computational complexity of the algorithm, a CPU/GPU library was developed to accelerate the GA running. A *Master-Slave* model was adopted in which the Host-CPU (Master) executes the GA steps (selection, population replacement and mutation), while GPU cores (slaves) evaluate the individuals fitness.

Furthermore, to support the interactive visualization and analysis phase of the results, a Visualization System, based on OpenGL and C++ and integrated into Qt interface was developed.

This thesis is organized as follows: Genetic Algorithms and Cellular Automata are presented in the second and third Chapter, respectively. Their most important theoretical results and some applications are also discussed.

The fourth Chapter concerns the application of Cellular Automata for modelling and simulating some natural complex phenomena. In this same Chapter the algorithm for the minimisation of differences [57] and the latest release of the lava flows simulation model SCIARA are presented. The fifth Chapter focuses on different CUDA approaches to accelerate simultaneous simulation of a large number of lava simulations using GPU. The overview of GPGPU paradigm together with the CUDA framework, implementations and performance analysis, referred to different benchmark simulations of a real event, are reported. An extensible system for the analysis and interactive visualization of Cellular Automata based simulations is discussed in the sixth Chapter. The System Architecture and a framework overview are also described. In the seventh Chapter, the application of Parallel Genetic Algorithms for the morphological evolution of protective works is presented. After a brief description of the case study adopted for the experiments, the main characteristics of three different strategies are discussed and results are presented. For each developed version, a study of GA dynamics, with reference to emergent behaviors and general considerations are also discussed. Final comments and future works are presented in the same Chapter. The closing Chapter concludes with general discussions and directions for future work.

## Chapter 2

# Genetic Algorithms

Among Artificial Intelligence models, such as cellular automata, artificial neural networks, fuzzy systems, multiagent systems, and swarm intelligence, genetic algorithms (GAs) [83] have proved to be an effective and robust support tool for the prediction and modeling of complex phenomena. GAs belong to the family of evolutionary algorithms (EAs) and can be considered as general-purpose search algorithms. GAs have been employed for optimizing a broad variety of problems for which standard optimization techniques require excessive computational resources and time to return the result or, simply, for those problems for which specific optimization procedures do not exist.

GAs are increasingly being considered by the scientific community for their simplicity and effectiveness. In fact, GAs do not require in-depth or specific knowledge in order to be applied, and a wide class of problems can be straightforwardly formulated to fit with the GAs requirements. Moreover, GAs are both employable as so-called embarrassingly parallel algorithms and can be easily adapted for solving problems that involve more than one objective to be optimized simultaneously.

Regarding parallel issues, many examples of parallel GAs (PGAs) have been proposed in the literature for both speeding up and improving the algorithms search ability. For instance, a mere parallelization of a standard panmictic (i.e., single population based) GA can be considered in case the application of the same algorithm would require too much time to converge toward a good solution in a sequential computational environment. Furthermore, even in the case the problem needs to be tractable on sequential architectures, a parallel nonpanmictic GA model can be adopted in order to allow a better exploration of the search space, by favoring the convergence toward solutions of higher quality.

GAs, when used as multiobjective search algorithms, unlike other multi-



objective optimization procedures, are able to provide a set of nondominated Pareto optimal solutions in a single run, which makes them particularly appealing in real world applications. The application of multiobjective GAs (MOGAs) is currently growing and, besides those of single-objective GAs, applications are starting to be used in geomorphology. Actually, applications of GAs in geomorphology are relatively new, starting in the late 1990s, with respect to other fields, for which applications started in the 1980s.

This Chapter is organized as follows. After a general overview, a brief history of GAs introduces the description of the original Holland's model and some of its most used variants in practical applications. After an introduction of the theoretical foundations of GAs, sequentially, a brief presentation of applications in different research branches is presented. General discussion about perspectives and new trends of the field conclude the Chapter.

## 2.1 Overview

Genetic Algorithms [84] are search algorithms based on mechanisms of biological evolution in order to solve problems and to model evolutionary systems. The basic idea behind the GA approach is to simulate the evolution of a population of candidate solutions to a specific search problem, promoting the survival and reproduction of the fittest. Individuals, also called *genotypes* or *chromosomes*, typically are represented by a data structure (e.g., string array or tree) which is conveniently chosen to encode candidate solutions. Elements of the chosen data structure are called *genes*, each one may assume a given number of values, called *alleles*.

The search operates by processing populations of genotypes, generation by generation, promoting the survival and reproduction of the best solutions. In this context, an individual is considered to be better than others if its encoded solution is more efficient than others. In the first generation  $P(t = 0)$ , randomly generated, the members are evaluated through a *fitness function*, which assigns a score (fitness) depending on how well every chromosome solves the problem at hand. Best individuals have higher probabilities to be selected and copied in the so-called mating pool for the reproduction process. The new population  $P(t + 1)$  replaces the old one thanks the introduction of new individuals generated through genetic operators, inspired by sexual reproduction. The phases of evaluation, selection, reproduction and mutation are iterated until a given stopping criteria such as reaching a prefixed threshold value for the fitness function or running a maximum number of GA steps. Algorithm 1 shows the iterative scheme of the GA.

---

**Algorithm 1:** Pseudocode of the iterative scheme of a basic genetic algorithm.

---

```
GA run{
  t = 0;
  initialize population P(t) //random generation
  evaluate population P(t) //fitness evaluation
  while NOT(stopping criteria) do
    t = t + 1;
    create mating pool MP(t) from P(t - 1) //selection
    create population P(t) from MP(t) //crossover, mutation
  }
```

---

## 2.2 A brief history of GAs

The initial studies on GAs go back to the 1960s, when a growing number of researchers began to consider natural systems as a source of inspiration for the development of optimization algorithms for engineering problems. Among these, John Holland, who is universally recognized as the father of GAs, was interested in the principles governing the evolution of adaptive natural systems, speculating that competition and innovation were the key mechanisms through which individuals acquire the ability to adapt themselves to the environment [82]. In the mid-1960s, the first examples of computational algorithms with characteristics similar to GAs were proposed, in which a population of individuals was considered and made to evolve. Furthermore, simple abstractions of genetic operators were employed as derivation mechanisms. However, the official birth of GAs dates back to 1975, when Holland published his book entitled *Adaptation in Natural and Artificial Systems* [83].

Despite the initial perplexities within the artificial intelligence community, Holland continued his work on GAs by introducing the notions of schema, implicit parallelism, and demonstrating the fundamental theorem of GAs [83]. Briefly such theoretical results state that the GA is able to rapidly concentrate the search toward the most-promising region of the search space (i.e., toward the region in which the fittest individuals are found during the first step of the algorithm) due to the mechanism of selection and, at the same time (i.e., in parallel), to continue to explore other regions, due to the mechanisms of sexual reproduction and mutation. Moreover, if a better individual is found outside the above-mentioned, most-promising search region, the algorithm is able to rapidly move the major search effort in the new region where the fittest individual has been found. Differently to what happened for other EAs, such as evolutionary strategies [136] or evolutionary

programming [69], Holland, thus, laid significant theoretical foundations for GAs, which has probably been one of the key factors of their success. The interested reader could refer to Holland [83] or Goldberg [74] for further details on theoretical foundations of GAs. Thereafter, De Jong [48] demonstrated that GAs can be fruitfully employed for the optimization of mathematical functions, highlighting that these artificial models of natural evolution can also be used as powerful search algorithms. The interest in GAs continued to grow slowly until 1989, when David Goldberg published his book entitled *Genetic Algorithms in Search Optimization and Machine Learning* [74]. Goldberg's textbook, which is today considered a classic of GAs literature, obtained the effect of catalyzing the attention of the scientific community, as it presented theory and applications of GAs in a clear, precise, and easily intelligible form [47]. The period from 1990 up to the present has been marked by the tremendous growth of the community of GAs and applications have affected a large number of new areas of research.

### 2.3 Elements of GA in the Holland's Model

The Holland's Model [83] is an iterative algorithm that operates on a population of  $N$  chromosomes encoded as bit strings of length  $l$  ( $l, N \in \mathbb{N}$ ) where each string (genotype) represents a candidate solution (phenotype) of a given research problem. For example, the genotype can encode specific values of a pair of coordinates  $\pi = \{c_i \mid i = 1, 2\}$  where each coordinate  $c_i$  is allowed to vary into a prefixed range  $[\alpha_i, \beta_i] \subset \mathbb{R}$ . The cardinality of the set of binary string of length  $l$  grows exponentially with  $l$  ( $2^l$  elements) and represents the GA search space, that is, the domain of the function to be optimized.

The objective function  $f$ , assigns a fitness value  $f_i = f(g_i)$  to each genotype  $g_i$  ( $i = 1, \dots, N$ ) of the GA. In order to assign a score to each individual of the GA, the fitness function decodes the genotype in the corresponding phenotype and tests it on the specific context (search problem). Usually the value generated by the fitness function represents the ability, for a genotype, to solve the problem. The graph of fitness values plotted against the search space points is called fitness landscape. Figure 2.1 shows a possible fitness landscape for a genetic algorithm with binary genotypes of length  $l = 2$ .

The Holland's model replaces all  $N$  individuals with as many offspring using a proportional selection method to determine the individuals to be reproduced and genetic operators, such as crossover and mutation, are after also applied. The selection operator selects chromosomes for reproduction purpose. The crossover exchanges subsequences of two chromosomes to generate two offspring and the mutation randomly flips some bits in a chromosome.

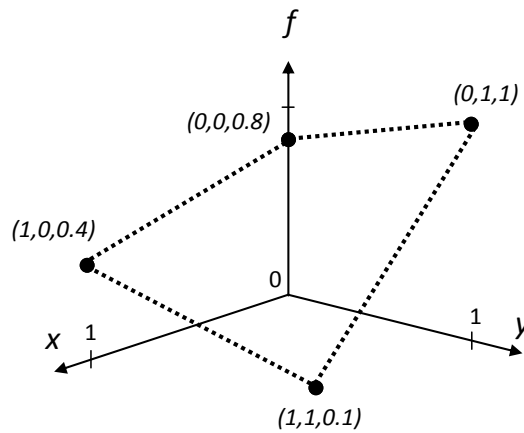


Figure 2.1: Example of fitness landscape for a binary genetic algorithm. In the specific case, the point  $(1,0,0.4)$  represents the fitness value associated to the point  $(1,0)$  if  $f(1,0) = 0.4$

A simple Holland's GA executes the following steps:

1. Start with a randomly generated population of  $N$  chromosomes.
2. Calculate the fitness  $f(x)$  of each chromosome  $x$  in the population.
3. Repeat the steps (4-5-6) until  $N$  offspring have been created.
4. Select a pair of parent chromosomes from the current generation on the base of a probability selection depending of the fitness values.
5. Crossover (with a crossover probability  $p_c$ ) the pair at a randomly chosen cut point to form two offspring.
6. Apply mutation (with a mutation probability  $p_m$ ) to the two offspring and place the resulting chromosomes in the new population.
7. Replace the current population with the new population.
8. Go to step 2.

Each iteration of the algorithm is called *generation* and the entire set of generations is called a *run*.

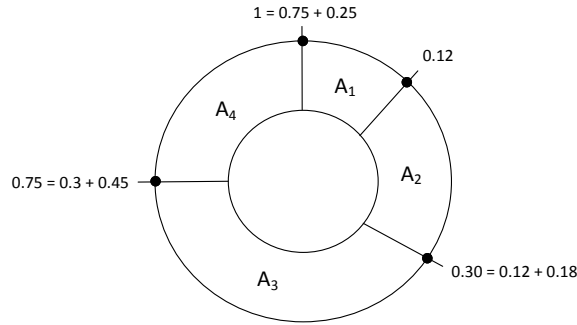


Figure 2.2: Example of proportional selection. The four individuals  $A_1$ ,  $A_2$ ,  $A_3$ , and  $A_4$  hold portions of the roulette proportionally to their selection probabilities, which are set to 0.12, 0.18, 0.45, and 0.25, respectively.

### 2.3.1 Proportional selection

For every genotype  $g_i$ , proportionally to the fitness value  $f_i$ , the probability  $p_{selection,i}$  defined as:

$$p_{selection,i} = \frac{f_i}{\sum_{j=1}^N f_j} \quad (2.1)$$

is associated to it and used to construct a roulette of probability for the selection process. For example, if the GA population is composed by the  $n = 4$  individuals  $A_1$ ,  $A_2$ ,  $A_3$  and  $A_4$  and the probability for each of them is  $p_{selection,1} = 0.12$ ,  $p_{selection,2} = 0.18$ ,  $p_{selection,3} = 0.45$  and  $p_{selection,4} = 0.25$  the roulette will be like the one in Figure 2.2. After the roulette construction, the selection operator takes place. A random number  $c \in [0, 1]$  is generated and the individual associated with the roulette portion containing the value  $c$  is selected to be copied and inserted into the so-called mating pool. For instance, if  $c = 0.58$ , the individual  $A_3$  is selected because the  $c$  falls within the range  $[0.3, 0.75]$ . Once the mating pool reaches a size of  $N$  elements, thanks to the copies of individuals of the population  $P(t)$ , members of the new population  $P(t+1)$  are obtained as their offspring by means the genetic operators application. Following the Darwinian Natural Selection, the GA selection operator favors the selection of individuals with higher fitness and, therefore, determines which individuals of the old population have the chance to generate offspring.

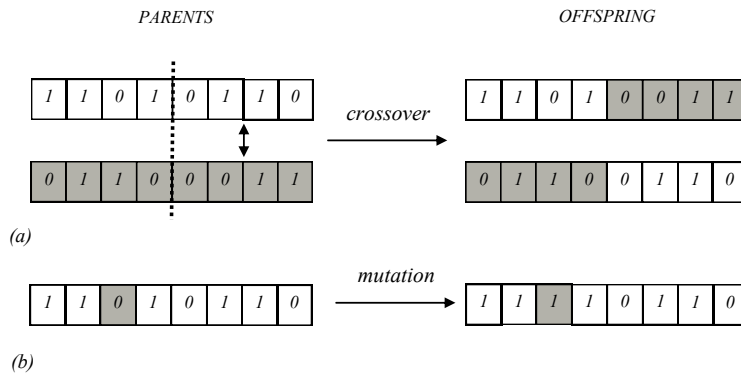


Figure 2.3: (a) Example of single-point crossover for a binary genetic algorithm. (b) Example of mutation for a binary genetic algorithm.

### 2.3.2 Crossover

During the crossover procedure, two parent individuals in the mating pool are randomly chosen and a cutting point selected. Portions, of the right of the cut, of the genotypes are exchanged, generating two offspring. Figure 2.3(a) shows an example of crossover between two binary genotypes.

The crossover is applied  $N/2$  times, according to a prefixed probability,  $p_{crossover}$ , in order to generate  $N$  offspring. If crossover and mutation are not applied, offspring coincide with parents. As the selection operator plays in the GA framework the role of natural selection, crossover is a metaphor of sexual reproduction in which genetic material of offspring results in a recombination of those of the parents. Once  $N$  offspring are obtained by crossover, mutation is applied.

### 2.3.3 Mutation

According to a prefixed and usually small probability,  $p_{mutation}$ , a bit value of each individual is simply changed from 0 to 1, or viceversa (see Figure 2.3(b)). The mutation operator represents the genetic phenomenon of the rare variation of genotypes elements in living beings during evolution.

After crossover and mutation are applied to the individuals of the mating pool, the new GA population,  $P(t + 1)$ , is obtained.

## 2.4 Variants of the Holland's Model

The first theoretical studies and application of GAs were inspired by Holland's model. However, in some cases, depending on the problem to optimize, it is not convenient to use bit string encoding and Holland's genetic operators [116]. An other Holland's Model weakness is that, during the evolution process, the best individuals can be replaced. For this reason, *elitism* principle can improve performance in GAs [144] and from 1980s new models have been proposed in literature which differ from the Holland's model in the genotype encoding, in the adopted genetic operators and selection strategies.

### 2.4.1 Encoding and genetic operators

The way of candidate solutions, for GAs and any other search methods, are encoded is fundamental. Most GA applications use fixed-length bit strings to encode GA individuals but, in recent years, different experiments have been carried out with different encodings.

In theory, it is always possible to encode candidate solutions of a search problem through binary strings. However, for the resolution of some problems, it is more natural to use higher-level representations and define crossover and mutation operators that can properly work on these representations. In practical applications, the most common schemes are the binary and that based on real numbers.

#### Binary encoding

Binary encoding is the most widely used in GA applications, both for historical reasons and because the most important theoretical results were obtained from it [83]. In this case, the employed data structure is a bit vector of length  $l$ , to which corresponds a search space of  $2^l$  possible solutions. The use of binary encoding requires the specification of a function that decodes the genotype in the corresponding phenotype. For example, the following equation decodes a binary genotype  $g$  of length  $l$  in the corresponding floating point value  $x$ :

$$x = x_{min} + \frac{x_{max} - x_{min}}{2^l - 1} \left( \sum_{i=1}^l g[i]^{l-i} \right) \quad (2.2)$$

where  $[x_{min}, x_{max}]$  defines the variation range for  $x$ , while  $g[i]$  is the  $i$ -th binary allele of the genotype  $g$ .

An alternative to the classical binary encoding is represented by gray code

Integer value	binary code	gray code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101

Table 2.1: Comparison between binary code and gray code representations

where adjacent numbers have a single digit differing by 1. Table 2.1 shows a comparison between binary code and gray code.

The most widely used crossover operator with binary encoding is the  $n$  point crossover, which differs from the classical Holland's single point crossover for the use of  $n$  cutting points. Another frequently used operator is the uniform crossover, which randomly exchanges corresponding bits of parents. The mutation operator, for binary encodings, is that proposed by Holland.

### Encoding based on real numbers

Encoding based on real numbers (floating point numbers), as well as natural numbers, is the most natural for optimization problems. The adopted data structure to represent an individual  $g$  is a vector of length  $l$  where each element,  $g[i]$ , is a real number which can vary in a prefixed range  $[\alpha_i, \beta_i] \in \mathbb{R}$ . The representation based on real numbers does not pose particular problems for the crossover operator and the classical one adopted for binary-coded GA can be used.

Regarding the mutation operator, for real-valued genotypes, it alters the genes of the individual by replacing elements with a vector  $M = (m_1, \dots, m_l) : g' = M$  where the elements of  $M$  can be generated in various ways, for instance, through a uniform distribution  $U(\alpha_i, \beta_i)$ . Thus, each  $m_i \in M$  is a value chosen randomly in the interval  $[\alpha_i, \beta_i]$ .

### Tree Encodings

The application of tree encoding can lead to several advantages such as the fact that it allows the search space to be open-ended because crossover and mutation make the tree size limitless. However, this aspect also leads to some potential pitfalls because the resulting trees, being large, can be very



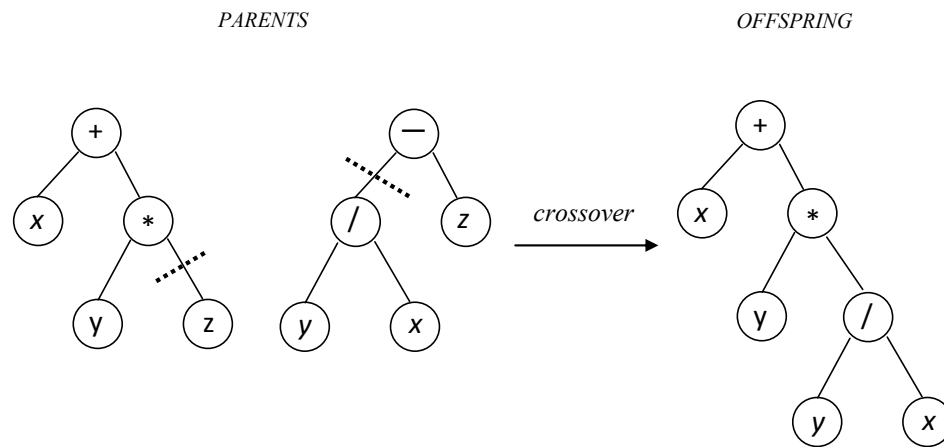


Figure 2.4: Example of single-point crossover between two trees. For each tree, one cut point is selected and two subtrees are exchanged.

difficult to analyze and simplify[101]. Tree encoding is used mainly for evolving programs or expressions in genetic programming and the employed data structure is a hierarchical tree structure, with a root value and subtrees of children, represented as a set of linked nodes. In this case, the crossover and mutation operators are different than the classical ones. Regarding the crossover between two genotypes, in both parent cut point is selected, parents are divided in that point and exchange part below crossover point to produce new offspring. Figure 2.4 shows an example of crossover between two trees. To apply the mutation, selected nodes are changed according to the problem search.

## 2.4.2 Selection methods and elitism

Selection is one of the fundamental processes of a GA because it eliminates individuals with lower fitness and creates one or more copies of individuals with higher fitness from which individuals of the new population are generated. The selection operator has a substantial effect on the dynamics of GA: too much selective pressure may result in an overly rapid convergence, by entrapping the algorithm in a local optimum from which it will be unable to exit; on the other hand, weak selective pressure can lead to an excessive increase in the amount of time required to find an acceptable solution.

Selection operators can replace the entire population or only part of it. Furthermore, a steady-state GA is obtained if at most a few individuals are replaced. In addition, the operator can select an individual once or more

than once. The first case refers to a selection operator without replacement, in the sense that the selected individual is not reinserted back into the old population after mating and, therefore, cannot be selected again. To the contrary, in the second case, the chosen individual is reinserted in the old population and can, therefore, be selected again, by producing more offspring.

In steady-state it may happen that the best individuals are lost in the transition to the subsequent generation. The models that ensure the survival of best individuals are called elitist (or k-elitist, where k is the number of the best individuals that are preserved and copied in the new population).

Besides the proportional selection operator proposed by Holland, the tournament selection is one of the most used in practical applications. The latter, as well as other selection operators (e.g., the Boltzmann and the rank-based ones), was introduced in order to have less selective pressure with respect to the proportional one [116].

### Boltzmann Selection

Boltzmann Selection keeps the selection pressure variable during the GA evolution. An initial low pressure allows less fit individuals to reproduce and a lot of variation in the population can be maintained in the first generations. Later, when the pressure become stronger and the population reached a certain level of diversity, only highly fit individuals are privileged for the reproduction process and this allows the algorithm to find the right part of the search space. In the Boltzmann selection approach the concept of *temperature* is present. Temperature value starts out high (it means that selection pressure is low) and it is gradually lowered.

A typical application of Boltzmann selection is to assign to each individual  $g_i$  an expected value,

$$ExpVal(g_i, t) = \frac{e^{f(i)/T}}{\langle e^{f(i)/T} \rangle_t} \quad (2.3)$$

where T is the temperature and  $\langle \rangle_t$  is the average over the population at time  $t$ . The effects of Boltzmann Selection have been studied by Prügel-Bennett and Shapiro [133, 131].

### Rank Selection

Rank selection purpose, as well as Boltzmann selection, is to prevent quick convergence to local optima. Ranking reduces the selection pressure when the fitness variance is high. In the Rank Based Selection each individual is, first, ranked in increasing order depending of its fitness, from 1 to N. The

users choose the expected value  $Max \geq 0$  on the individual with rank  $N$  and for each individual  $g_i$  in the population the expected value at time  $t$  is:

$$ExpVal(g_i, t) = Min + (Max - Min) \frac{rank(i, t) - 1}{N - 1} \quad (2.4)$$

where  $Min$  is the expected value associated to the individual with rank 1. The effects of Rank Selection have been studied by Blickle [16] and Rogers [138].

### Tournament selection

In the most common type of tournament selection, two individuals are chosen at random from the current population and a number  $c \in [0, 1]$  is randomly generated. If  $c$  is less than a prefixed parameter  $r \in [0, 1]$  the most fit individual wins the tournament and is selected, otherwise the less fit is the winner. In addition, if the scheme with replacement is applied, the two individuals are reintegrated in the old population and maybe selected again.

## 2.5 Multiobjective GAs

Problems characterized by a single objective consist in finding the best-possible solution or, at least, a good approximation of it, and the problem to evolve protective works to divert lava flows, proposed in this thesis, can be fruitfully employed for their optimization. However, problems that have more than one (often-conflicting) objective to be simultaneously optimized are common in real-world applications. When dealing with multiobjective optimization problems, the concept of optimality is generally extended according to the notion originally proposed by Vilfredo Pareto [128]. This notion is called Edge-worth-Pareto optimality, or simply Pareto optimality, and refers to finding good tradeoff solutions among all the objectives, because the latter are commonly in conflict with each other. In fact, multiobjective optimization problems generally do not have one single optimal solution (global optimum) but a set of feasible solutions, each one better with respect to one particular objective and not as good with respect to others. In a multiobjective optimization problem, a set of non-dominated Pareto optimal solutions is, thus, found instead of one single solution.

Informally, a solution  $x$  to the multiobjective search problem dominates another solution  $x$  if and only if it is at least as good as  $x$  with respect to all objectives and better in at least one objective. A solution which is nondominated by no other is said Pareto optimal and represents one of the

best possible trade-off solutions among search objectives. In other words, a solution is said to be Pareto optimal if no other feasible solution can be found which would decrease some criterion without causing a simultaneous increase in at least one other criterion. The image of the Pareto optimal set under the objective functions is called the Pareto front [174].

Due to their flexibility in dealing with a great variety of multiobjective problems and to their simplicity, evolutionary algorithms are the most widely used multiobjective search methods in practice. Also, EAs can be easily adjusted in order to generate several nondominated solutions in a single run. For these reasons, multiobjective EAs (MOEAs) have become popular for the optimization of complex real-world multiobjective problems [28, 20, 9, 62]. It is worth noting, however, that finding a set of Pareto optimal solutions is only a part of the overall multiobjective optimization process, because the choice of one particular solution to use in practice is of fundamental relevance in the subsequent decision-making task. Decision is generally guided by the relevance that users assign to the specific optimization objectives, which help in narrowing down their choice.

The first implementation of a MOEA dates back to the mid-1980s [146]. However, the direct incorporation of the concept of Pareto optimality into an EA was first alluded to by Goldberg [74], who suggested the use of nondominated ranking and selection to move a population toward the Pareto front in a multiobjective optimization problem. The basic idea is to find the set of nondominated solutions, to which is assigned the highest rank. The optimal set is, therefore, momentarily set aside and another set of Pareto optimal solutions is determined from the remaining population. The next highest rank is assigned to the individuals belonging to this newly formed set. This process continues until the entire population is ranked according to Pareto optimality.

Goldberg also suggested the use of some kind of niching technique to keep the GA from converging to a single point on the front. A niching mechanism, such as fitness sharing among individuals belonging to the same front [75], would allow the EA to maintain a good spreading of the individuals along the nondominated frontier. Apart from the best individuals, such kind of feature allows the algorithm to preserve also average and worse individuals in order to sustain diversity within the population. This behavior represents one of the major differences of MOEAs with respect to single-objective EAs where, for instance in the case of GAs, the diversity of the population generally decreases over the generations, until almost all solutions are majorly the same; a small number of individuals are, however, kept different thanks to the action of genetic operators.

The first examples of MOEA [71, 85, 154] suffered from the fact of not

using an elite preservation mechanism, which is able to ensure a monotonically nondegrading performance. On the contrary, all the most important algorithms that were proposed successively implemented an elite preserving operator. Among these, it is possible to find Strength Pareto Evolution Algorithm (SPEA)[176] and SPEA2 [175], Pareto Archived Evolution Strategy (PAES) [99], Pareto Envelope-based Selection Algorithm (PESA) [31] and PESA2 [30], and NSGA-II [51]. Even though new algorithms that optimize directly for the hypervolume (i.e., the area under the Pareto front) have been recently developed within the EA community which outperform the above algorithms in many cases (see, e.g., [13]), NSGA-II nowadays represents the most widely used MOGAs in engineering and scientific fields and is still widely considered the state of the art for practical applications.

## 2.6 Theoretical foundations of Genetic Algorithms

Even if the GA model is simple to define, its behavior can be complicated and very hard to be formalized. Among different works done on the foundation of GAs [74, 135, 169] the theory of GAs [82, 84] is based on the idea that GAs work, in implicit parallel way, by discovering, emphasizing and recombining good candidate solutions called building blocks.

The notion of schema formalizes the abstract concept of building block. A schema is a set of bit strings made up of a set of 1, 0, and \*, where \* represents any possible sequence of 1/0. The schema  $s = 1***0$ , for example, is the set of all 6-bit strings where the first and the last position are fixed. The instances of  $s$  are all the strings that fit on the schema  $s$  (e.g, 111100 or 100000). Furthermore, the schema  $s$  has *order* 2 because it contains exactly two defined bits. The distance, in term of number of bits, between the outermost defined bits is called *defining length*. In a set of length- $L$  bit strings there are  $2^L$  (every bit can be 1 or 0) possible bit strings of length  $L$  and  $3^L$  (every bit can be 1, 0 or \*) possible schemas. In the GA theory, schemas are considered the building blocks that the GA processes during the genetic operators application.

Any length- $L$  bit string is an instance of  $2^L$  different schemas. For example, the string 01 is an instance of \*\*, \*1, 0\*, and 01, thus, any population of  $N$  strings contains instances of between  $2^L$  and  $N \times 2^L$  different schemas. This means that, for every generation, while the GA is evaluating the fitness function of the  $N$  members in the population, it is implicitly estimating the average fitness of all possible instances of a much larger number of schemas.

For example, by considering a population of  $N$  strings, on average  $N/2$  will be instances of  $1 * \dots *$  and  $N/2$  will be instances of  $0 * \dots *$  and the evaluation of the  $N$  strings give an estimate of the average fitness of that schema because the instances evaluated are only example of all possible instances. This consideration is known as *implicit parallelism* [83].

To calculate the approximate dynamics of the example discussed before, it is important to introduce the Schema Theorem. Let  $s$  be a schema with at least one instance present in the population at time  $t$ , let  $N(s, t)$  be the number of instances of  $s$  at time  $t$  and let  $\bar{u}(s, t)$  the average fitness of  $s$  at time  $t$ . The goal of the theorem is to obtain  $E(N(s, t + 1))$ , the expected number of instances of  $s$  at time  $t + 1$ . By considering the selection operator as proportional to fitness, the number of copies of a string  $x$  is  $(F(x))/(\bar{F}(t))$  where  $F(x)$  is the fitness value associated to the string  $x$  and  $\bar{F}(t)$  is the average fitness of the population at time  $t$ . Then,

$$E(N(s, t + 1)) = \sum_{x \in s} F(x)/\bar{F}(t) \quad (2.5)$$

and since  $\bar{u}(s, t) = (\sum_{x \in s} F(x))/N(s, t)$ ,

$$E(N(s, t + 1)) = \frac{\bar{u}(s, t)}{\bar{F}(t)} N(s, t) \quad (2.6)$$

Since Genetic operators (crossover and mutation) can delete and generate instances of  $s$ , it is possible to consider a lower bound of  $E(N(s, t + 1))$ . First, the effect of crossover is taken into account. Let  $p_c$  the probability to apply a single-point crossover to a given string and suppose to choose an instance of  $s$  like a parent for the reproduction process. Schema  $s$  is preserved if, after the crossover apply, one of the offspring belongs to  $s$ . The lower bound on the probability of a schema  $s$  surviving under single-point crossover is

$$S_c(s) \geq 1 - p_c \left( \frac{d(s)}{L - 1} \right) \quad (2.7)$$

where  $d(s)$  is the defining length of  $s$  and  $L$  is the length of bit strings in the search space. The function shows how the probability of survival for the schema  $s$ , under crossover, is higher for shorter schemas.

To quantify the effect of mutation, it is important to introduce  $S_m(s)$ , the probability that schema  $s$  will survive after the mutation process, as following:

$$S_m(s) = (1 - p_m)^{o(s)} \quad (2.8)$$

where  $p_m$  is the probability of any bit being mutated and  $o(s)$  is the order of  $s$  (number of fixed bits in  $s$ ). For each bit in the string, the probability that it will not be modified is  $1 - p_m$  and the probability that no bits of the schema  $s$  will be mutated is this value multiplied by itself  $o(s)$  times. It means that the probability of survival under mutation is higher for lower-order schemas.

Finally, it is possible to introduce the genetic operators effects to the eq. 2.6 in order to evaluate the probability for the schema  $s$  to survive:

$$E(N(s, t + 1)) \geq \frac{\bar{u}(s, t)}{\bar{F}(t)} N(s, t) \left[ 1 - p_c \frac{d(s)}{L - 1} \right] [(1 - p_m)^{o(s)}] \quad (2.9)$$

This is known as the Schema Theorem introduced by Holland [83].

It can be interpreted as follows: low-order schemas whose average fitness remains above the mean will obtain exponentially increasing numbers of instances during the evolution. This is because instances of those schemas that survive remain above average in fitness increases by a factor of  $\bar{u}(s, t)/\bar{F}(t)$  at each generation.

## 2.7 Application of GAs

When considering GAs as search algorithms, it is worth noting that standard optimization techniques, when they exist, generally perform better than GAs. However, GAs can represent the best choice for those problems for which specific techniques require an excessive computational time or where such techniques do not exist. In fact, with the exception of particular cases, these problems are NP-complete and, thus, no deterministic algorithm is known to solve them in polynomial time. This means that the execution time of the fastest algorithm grows exponentially with the problem's dimension, making it often impossible to deal with problems of practical interest. In fact, the success of GAs in combinatorial optimization regards applications to large dimension problems.

GAs have been used in a large number of scientific problems and models: for the solution of difficult combinatorial problems [76]; in the study of the interaction between evolution and learning [81]; in evolutionary robotics [124, 63]. GAs based methods have also been applied to CA for modelling bioremediation of contaminated soils [58] and for the optimisation of lava and debris flow simulation models (e.g., [152, 87, 141, 44, 43]).

Evolutionary optimization is today at its peak, and thus it is difficult to keep track of every single theoretical and applied research. Please refer to international conferences on the topic, for example, the genetic and evolu-

tionary computation conference (GECCO), the evolutionary multi-criterion optimization (EMO), or the more general parallel problem solving from nature (PPSN), in addition to specific scientific journals, for an up-to-date state of the art of the field. Applications of GAs in geomorphology are discussed in the next session.

## 2.8 GAs in Geomorphology

Among EAs, GAs met the wider scientific community's approval, and applications rapidly spread in different fields of science and engineering. This is probably due to their suggestive inspiration from Darwinian evolution and genetics and theoretical foundations. The effectiveness of such algorithms was immediately clear, especially in those fields where no standard optimization techniques existed. This favorable context also stimulated the development of new research trends: at first PGAs were proposed for the improvement of both performance and quality of obtained solutions, whereas MOGAs, of the broader family of MOEAs, were proposed in order to treat problems involving more than one objective to be optimized simultaneously.

Applications of GAs in geomorphology, as well as in similar fields such as hydrology, have established themselves relatively late, starting from the late 1990s. One of the first applications [119] relates to the application of a GA as an alternative to back propagation, for the training of an artificial neural network for flow and soil transport simulations. Further applications exist in rainfall-runoff, erosion and sedimentation models [110, 123, 11, 95], in debris flow prediction [21] and simulation [87, 46, 45], as well as in lava flows simulation [152, 3].

As in other research fields, GAs were initially applied to calibrate simulation models in geomorphology in their simplest form, with the goal of obtaining the best possible match between observed and expected values for single-objective optimization problems. Subsequently, applications of PGAs were proposed, in particular for improving computational efficiency and, as a consequence, for permitting a better exploration of the search space. Already in this first period, besides the undeniable advantages, some early studies indicated the risks associated with the use of GAs in calibrating models in geomorphology [120], whereas others indicated that the search algorithm convergence can be significantly improved by including more objectives in the fitness function [46]. This latter example represents one of the simplest forms of multiobjectivization, because a single-objective EA is in any case applied. The application of MOEAs is the natural evolution of these first attempts to improve search convergence. In fact, MOGAs allow for a more natural



and favorable formulation of problems that involve more than one goal to be achieved, besides providing a set of optimal (also known as nondominated or Pareto optimal) solutions. Furthermore, both theoretical studies and applications to specific optimization problems in other fields demonstrated how the use of more objectives can smooth the fitness landscape by reducing the presence of local optima and, thus, favoring the convergence of the algorithm toward better solutions [98, 90, 20, 78, 89]

The use of MOGAs [51, 28] certainly represents the most-promising trend in the field of application of EAs in geomorphology, even if few examples can today be found in the specific literature [11, 3] and more studies must be performed to assess their effective advantage with respect to single-objective EAs.

## 2.9 Conclusions

The application of GAs today extends to a broad range of scientific and engineering disciplines. Its application in geomorphology, however, is relatively new, having started in the late 1990s, thereby making it a novelty in the field of geophysics. Nevertheless, the first applications in geomorphology demonstrate the potential of GAs, and their usefulness in finding good solutions for the calibration of parameter dependent models of complex natural phenomena.

This Chapter has provided a general overview of both single and multiobjective GAs, focusing on applications in geomorphology through the description of four meaningful works. Specifically, regarding the complex phenomena that depend on a set of parameters GAs showed the ability to find solutions that significantly out-perform those obtained by manual calibration, which is still a widespread optimization practice in the modeling of processes in geomorphology.

Among different GA applications with success, a novel approach in the geomorphology field is represented by the work discussed in this thesis where PGAs have been considered for optimizing the position, the extension and the orientation of earth barriers to control lava flows.

As previously stated, GAs have only recently been discovered in geomorphology and, probably for this reason, their use is not yet widespread. However, it is expected that the field grows significantly in the next few years and the application of GAs to spread to a wide range of problems in geomorphology.

# Chapter 3

## Cellular Automata

Cellular Automata (CA) are computational models whose evolution is regulated by laws which are purely local.

In its essential definition, a CA can be described as a  $d$ -dimensional space composed by regular cells. Each cell can be in a finite number of states and embeds a *finite automaton (fa)*, that is one of the most simple and well known computational model in Computer Science; it can be seen as a system to which is associate a *state* that could change in consequence of an input. At time  $t = 0$ , cells are in an arbitrary state and the CA evolves by changing the states of the cells in discrete steps of time and by applying simultaneously to each of them the same law, or *transition function*. The input for each cell is given by the states of neighboring cells and the *neighborhood* conditions are determined through a geometrical pattern, which is invariant in time and space.

Despite their simple definition, CA may give rise to extremely complex behavior[171] at microscopic level. In fact, even if local laws that regulate the dynamics of the system are known, the global behavior of the system is very hard to be predicted [24]. In other words, the dynamics of the system emerges in a nontrivial manner by the mutual interaction of its basic components.

CA are considered universal computation models because of their equivalence with the Turing Machine [35, 171], as it has been demonstrated by Codd [27] and Thatcher [158].

CA are adapt to model and simulate systems characterized by interaction of numerous elementary constituents and they have been largely employed in several fields of study as pattern recognition [111, 50], image processing [142, 143, 132], cryptography [163, 18] and so on. An example of CA application is the study of the fluids behavior (considered at the microscopic level as particles systems) through Reticular Gas[156]. Other important studies are related to the consideration of CA as parallel computing systems [159, 37,

157]. In this chapter after a brief history of CAs, a quick overview, with the introduction of informal and formal definition of the model, is presented. Some theoretical aspects of CA are also reported and a brief description of some examples of CA applications in different areas conclude the chapter.

### 3.1 A brief history of Cellular Automata

CA developed as the course of a study started in 1947 by Von Neumann, who tried to find the features and the complexity of a self-reproducing system. The Hungarian mathematician prematurely died in 1957 and his work was published later in 1966, edited and completed by A. Burks [167].

The best-known way in which cellular automata were introduced was through work by John von Neumann in trying to develop an abstract model of self-reproduction in biology.

Immediately after the von Neumann's work, two immediate threads emerged. The first, mostly in the 1960s, was increasingly whimsical discussion of building actual self-reproducing automata and the second was an attempt to capture more of the essence of self-reproduction by mathematical studies of detailed properties of cellular automata.

By the end of the 1950s it had been noted that cellular automata could be viewed as parallel computers, and particularly in the 1960s a sequence of increasingly detailed and technical theorems, often analogous to ones about Turing machines, were proved about their formal computational capabilities. At the end of the 1960s there then began to be attempts to connect cellular automata to mathematical discussions of dynamical systems.

However, CA started to be famous in the 70s through one of the easiest CA application, the well-known *Game of Life* defined by the English mathematician John Horton Conway and described by Martin Gardner in his work [73].

### 3.2 Informal definition of Cellular Automata

It is possible to identify an informal definition of cellular automaton by simply listing its main properties:

- it is formed by a  $d$ -dimensional space (*the cellular space*), partitioned into cells of uniform size (triangles, squares, hexagons, cubes) or by a  $d$ -dimensional regular lattice (see Figure 3.1);
- the number of cell states is finite;

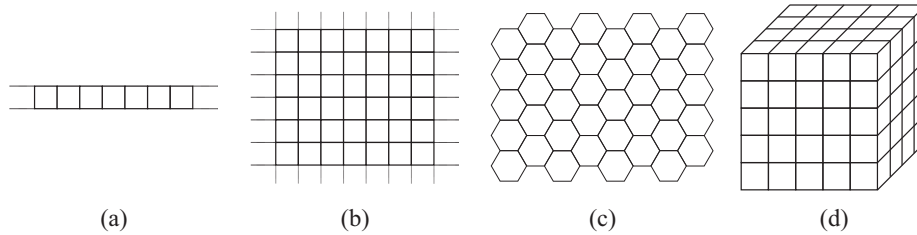


Figure 3.1: Example of cellular spaces: (a) one-dimensional, (b) two-dimensional with square cells, (c) two-dimensional with hexagonal cells, (d) three-dimensional with cubic cells.

- the evolution occurs through discrete steps;
- each cell evolves by simultaneously changing its state by applying the same transition function to the cellular space;
- the transition function depends on the state of the central and neighboring cells;
- the relationship of closeness that defines the neighborhood of a cell is local, uniform and invariant over time.

### 3.2.1 Dimension and geometry of Cellular Automata

The definition of Cellular Automata requires the discretization of the space in cells. In the simplest situation (one-dimensional Cellular Automata), the CA space is one-dimensional and cells are aligned next to each other. Regarding multi-dimensional Cellular Automata the CA space can be discretized in different ways; two-dimensional CA can be represented, for example, with triangle, square or exagonal tessellation while for three-dimensional cellular automata cubic cells are usually chosen. Figure 3.1 shows examples of different cellular spaces.

Even if square tessellation, for two-dimensional cellular automata, can be easily represented by a matrix structure (both for graphics and computation representation), by considering some applications it can lead to anisotropic problems. In these cases, it is preferable to adopt an exagonal tessellation.

### 3.2.2 Number of states of a cell

Number of states of a cell is finite and it is based on the study or application context. In first theoretical studies in which CA were considered as abstract

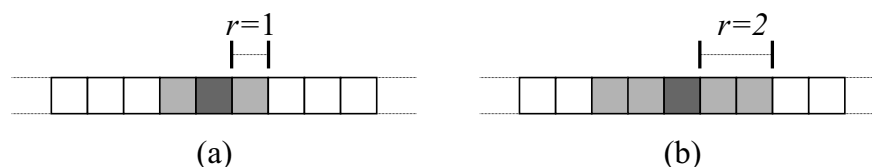


Figure 3.2: Example of neighborhood with radius (a)  $r = 1$  and (b)  $r = 2$  for uni-dimensional cellular automata.

models [27, 158] the number of states of a cell was, usually, quite small.

When the CA is adopted to describe particle systems, it is not necessary a large number of states to model the interactions [155, 168]. In contrast, when studying systems with a continuum of possible states, they may require a large number of states [57].

### 3.2.3 Relationship of closeness

The cell's relationship of closeness depends on the geometry of cells and it has to have the following properties:

1. it must be local because only a limited number of cells near the central one are involved;
2. it must be homogeneous because it is the same for each cell of the cellular space;
3. it must be invariant over time;

For one-dimensional CA, usually, the neighborhood is considered in terms of radius,  $r$ , which defines a neighborhood consisting of  $n = 2r + 1$  cells [170]. For example, a radius  $r = 1$  consists of  $n = 2r + 1 = 3$  cells: the central cell, the aligned left one and the aligned right one. Figure 3.2 shows two different examples of neighborhood for an uni-dimensional CA.

In the case of two-dimensional CA with square tessellation, the neighborhoods most commonly used are those of von Neumann and Moore. the first one comprises the four cells orthogonally surrounding a central cell (north, east, south, west) while the second also contains the north-west, north-east, south-west and south-east cells.

For exagonal two-dimensional cellular automata a typical neighborhood is composed by north, north-east, south-east, south, south-west and north-west cells.

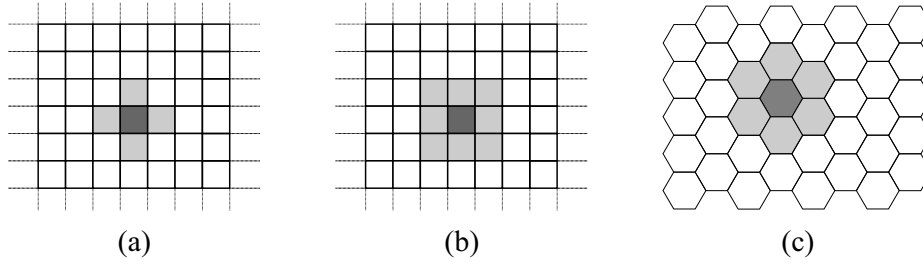


Figure 3.3: von Neumann (a) and Moore (b) neighborhoods for a two-dimensional cellular automata with square cells and with exagonal ones (c).

Figure 3.3 shows (a) the von Neumann neighborhood and (b) the Moore one for square tessellation and (c) the one adopted for exagonal tessellation.

It is worth to note that different relationship of closeness can be applied such as, for example, the Margolus neighborhood widely used in simulations of gas diffusion [160].

### 3.2.4 State-transition function

At each CA step, the transition function is simultaneously applied to all cells of the cellular space, by determining the new state of each cell in function of the state of the neighborhood cells. Parallelism and decentralization are characteristics of the CA computation.

When the number of states is small, usually the transition rules are defined through a look-up table which specifies the new state of the central cell for each possible configuration of the neighborhood [171]. Differently, when the number of CA states is too large, the transition function is usually defined by an algorithm [57].

## 3.3 Formal definition of Cellular Automata

The homogeneous cellular automata is a quadruple:

$$A = \langle \mathbb{Z}^d, Q, X, \sigma \rangle \quad (3.1)$$

where:

- $\mathbb{Z}^d = \{i = (i_1, i_2, \dots, i_d) | i_k \in \mathbb{Z} \forall k = 1, 2, \dots, d\}$  is the d-dimensional cellular space;
- $Q$  is the finite set of states of the cellular automata;

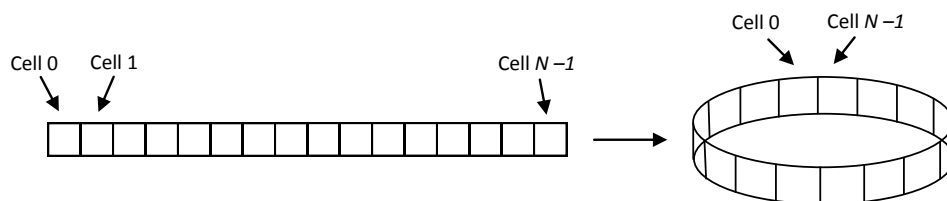


Figure 3.4: Example of one-dimensional cellular automata with periodic boundary conditions. First and last cells are adjacent.

- $X = \{\xi_0, \xi_1 \dots \xi_{m-1}\}$  is the finite set of  $m$   $d$ -dimensional vectors

$$\xi_j = \{\xi_{j1}, \xi_{j2}, \dots, \xi_{jd}\}$$

that define the set

$$V(X, i) = \{i + \xi_0, i + \xi_1, \dots, i + \xi_{m-1}\}$$

of coordinates of cells close to the generic cell  $i$  with coordinates  $(i_1, i_2, \dots, i_d)$ .

$X$  is the geometrical pattern that specifies the neighbourhood relationship;

- $\sigma : Q^m \rightarrow Q$  is the transition function for the CA

## 3.4 Theory of Cellular Automata

In this section, some theoretical aspects of cellular automata are reported.

In particular, results on reversibility, conservation laws, universality and topological dynamics of CA are discussed. Because most of them are related to one-dimensional CA is important to introduce some definitions.

### 3.4.1 One-dimensional Cellular Automata

The simplest CAs are the elementary CAs [170]. They are one-dimensional CAs with  $N$  cells,  $k = 2$  states (0 and 1), neighborhood radius  $r = 1$  and periodic boundary conditions (one-dimensional cellular space is toroidal, seen as a ring where first and last cells are adjacent). Figure 3.4 shows an example of one-dimensional CA with periodic boundary conditions. Representing a CA as a ring allows to define an unlimited space in which the CA can evolve.

The transition function  $\sigma$  is defined through a look-up table. For example, by considering the generic neighborhood's configuration  $\eta$  (the number of neighborhood's configuration is  $k^{2r+1}$  and for one-dimensional CA is  $2^3 = 8$ ), the following transition function determines the central cell new state,  $s = \sigma(\eta)$ :

$\eta$	000	001	010	011	100	101	110	111
$s$	0	0	1	1	0	1	1	0

By adopting this convention, any transition rule for the elementary CA elementary can be defined by listing the central cell new states as follow:

$\eta$	000	001	010	011	100	101	110	111
$\sigma_{00110110} =$								
$s$	0	0	1	1	0	1	1	0

Each possible transition rule can be identified through the decimal number corresponding to the binary number that defines the same rule ( $\sigma_{000000000} = \sigma_0, \sigma_{000000001} = \sigma_1, \dots, \sigma_{000000001} = \sigma_{255}$ ).

If  $r > 1$  the number of possible configuration grows rapidly. For example, by considering  $(k, r) = (2, 2)$  the number of total transition rules is  $2^{32} = 4294967296$  because  $k^{2r+1} = k^n = 2^5 = 32$ , making impossible a comprehensive analysis.

### 3.4.2 Universality and complexity in Cellular Automata

Wolfram [170, 171] proposed a classification of the one-dimensional CA based on their qualitative behaviour, identifying four different complexity classes:

- **class 1** the class 1 CA, nearly all initial patterns, evolve quickly into a uniform final state. Any randomness in the initial pattern disappears;
- **class 2** the class 2 CA evolve into final state with stable or oscillating structures. Some of the randomness in the initial pattern remains;
- **class 3** the class 3 CA are characterized by an extremely pseudo-random or chaotic behaviour. Structures that appear are quickly destroyed;
- **class 4** the class 4 CA are characterized by both uniform and chaotic behaviour. In this class structures can interact with each other in extremely complex ways;



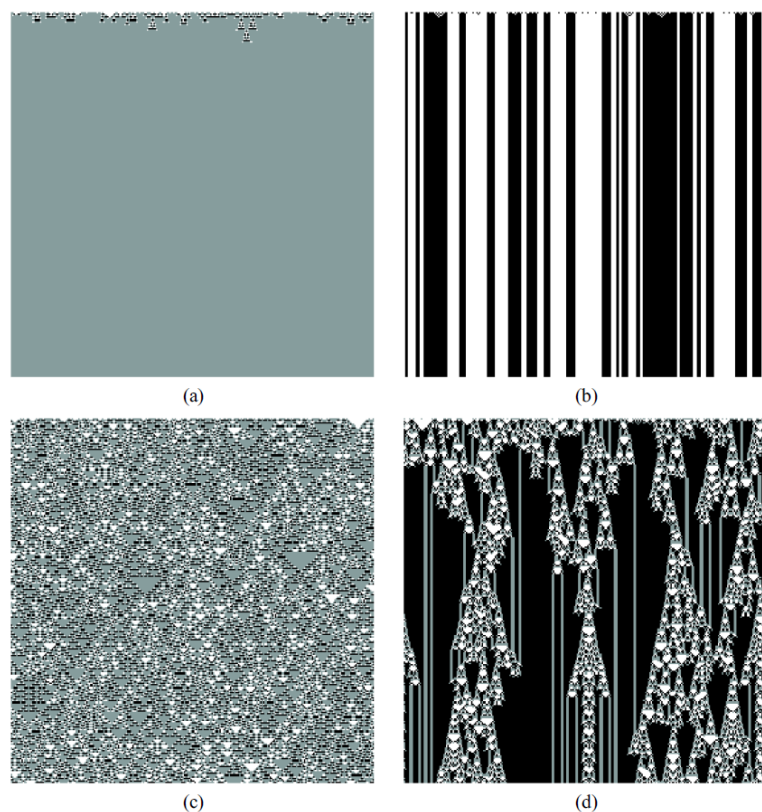


Figure 3.5: First 250 CA calculation steps with  $k = 3$  and  $r = 1$  for (a)  $\sigma_{c=1014}$ , (b)  $\sigma_{c=1008}$ , (c)  $\sigma_{c=1020}$ , (d)  $\sigma_{c=2043}$ . The illustrated CA belong respectively to the complexity classes 1, 2, 3 and 4. The initial configuration consists of 250 cells and it is randomly generated so that each cell can assume state 0 (white), 1 (gray) or 2 (black).

Although other classifications have been proposed [23, 77, 114], the Wolfram one is certainly the most known. Examples of CA belonging to the Wolfram four complexity classes are illustrated in Figure 3.5.

The class 4 proved to be particularly interesting for the presence of structures (glider) able to propagate in space and time. For this reason Wolfram hypothesized that CA belonging to the class 4 can be capable of universal computation. The CA proposed by Wolfram can be seen as calculators; the initial configurations encode data and program, and final configuration (after several calculation steps) encode the computation result. This means that it is possible to represent with a CA and an appropriate transition function every possible computational program. The hypothesis of universality for simple CA comes from the observation that the glider can act as elaborator

of information encoded into the initial configuration. By means of the glider, the state of a cell in a particular position of the cellular space can influence over time the states of cells in arbitrarily distant locations. Furthermore, the glider can interact among themselves in an extremely complex way and, theoretically, can play as demonstrated for the Game of Life by John Horton Conway [73] the logic gates of an universal computer.

### 3.4.3 Chaos theory

Class 4 automata are considered as at the the *edge of chaos* and give a good mathaphor for the idea that the interesting complexity is in equilibrium between stability and chaos. The hypothesis of Wolfram that the simple one-dimensional CA are capable of universal computation was, subsequently, studied by Chris Langton. He has shown that an appropriate parameterization of the space of rules allows to identify both the relationship between the complexity classes and the regions of that space.

Langton in his paper [107] introduced the parameter  $\lambda$  as the fraction of the entries in the transition rule table that are mapped do the quiescent state  $q_s$ . Langton's major finding was that a simple measure such as correlates with the system behavior: as goes from 0 to 1, the average behavior of the systems goes from freezing to periodic patterns to chaos and functions with an avarage  $\lambda \approx 1/2$  (please refer to [107] for a more general discussion) are being on the edge.

In particular, the parameter  $\lambda$  was defined as:

$$\lambda = \frac{k^n - n_q}{k^n} = 1 - \frac{n_q}{k^n} \quad (3.2)$$

where  $k$  is the number of states of the cell,  $n = 2r + 1$  is the number of neighborhood cells and  $n_q$  is the number of transitions that terminate in the quiescent state. If  $n_q = k^n$ , all the transitions of the look-up table go to the quiescent state and  $\lambda = 0$ ; if  $n_q = 0$  there are no transitions that terminate in the quiescent state and  $\lambda = 1$ ; finally,  $\lambda = 1 - 1\lambda$  when in the look-up table when all the states are represented by the same misure.

Langton has analyzed the behaviour of several totalistic CA with  $k = 4$  and  $r = 1$  whith a variation range for  $\lambda$  of  $[0,0.75]$ . The results showed that for small values of  $\lambda$  the CA behaviour is uniform, typical of complexity classes 1 and 2, while for large values the observed behaviours is chaotic, typical of class 3. Between these two zones (order and chaos), however, Langton has observed a very small third zone near to the value  $\lambda = 0.45$ . In this zone, defined as edge of chaos, the CA dynamics is able to generate both static and dynamics structures that can be propagated in space and time, typical

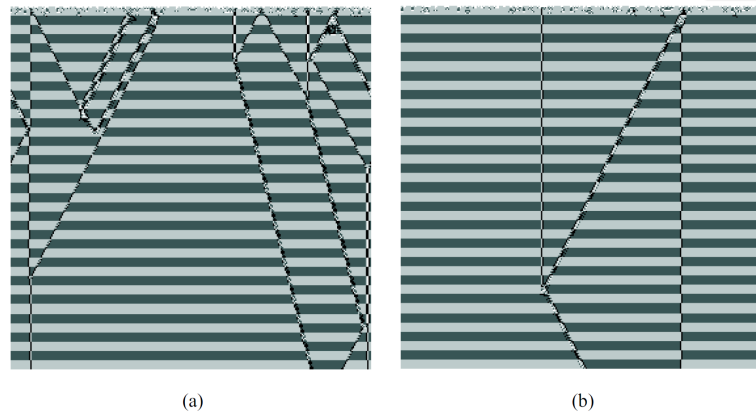


Figure 3.6: Examples of CA at the edge of chaos. Figures (a) and (b) show the evolution of the same CA with  $k = 4$  states and  $r = 1$  with two different initial conditions. The shades of gray represent the four possible states of the cell, from white for the state 0 to black for the state 3.

of the class 4 of Wolfram.

Figure 3.6 shows an example of an CA at the edge of chaos. Only in the edge of chaos zone the encoded information in the CA initial configuration can propagate over long distances, which is a necessary condition for the concept of computation.

### 3.4.4 Other theoretical works on Cellular Automata

The studies discussed in this chapter are only a part of the whole set of theoretical research on CA and many different contributions have come from researchers from all parts of the world. For example, the problem of reversibility of CA has been studied by Moore [118], Myhill [121], Di Gregorio and Trautteur [59], Kari [94], and Toffoli and Margolus [160].

Jiménez-Morales has adopted an evolutionary approach based on GA for the study of non-trivial collective behavior in CA [91].

Other interesting studies regard, the selfreproduction problem in the CA. Among them, Azpeitia and Ibáñez [5] and Bilotta et al. [14] worked in this direction.

Finally, Roli and Zambolli [139] studied the emergence of macro spatial structures in dissipative CA seen as open systems where the environment can influence the dynamics.

## 3.5 CA applications

CA are particularly suited to modeling and simulation of some classes of complex systems characterized by the interaction of a large number of elementary components. The assumption that if a system behaviour is complex, the model that describes it must necessarily be of the same complexity is replaced by the idea that its behavior can be described, at least in some cases, in very simple terms [171].

In some areas, the CA application gave results comparable to those obtained from traditional approaches. A particularly significant example is the CA application to modeling turbulent flows behaviour through lattice gas and lattice Boltzmann models. Another important field of CA application is the Artificial Life, a discipline that deals with the examination of systems related to life, its processes, and its evolution. Moreover, in recent years, CA have been applied with success in the modeling of natural complex phenomena.

A brief description of some examples of CA applications in Artificial Life, Reticular Gas and lattice Boltzmann models is described below. CA application to modeling natural complex phenomena is discussed in the next chapter.

### 3.5.1 Artificial life with CA

Artificial life can be defined as the discipline that deals with the life and the behavior of artificial systems that *live* in an artificial environments. It seeks to study life not out in nature or in the laboratory, but in the computer. Langton suggested that CAs could be an extremely effectiveness model to study artificial life [106]. In fact, John von Neuman since 40's studied the reproduction in living organisms by adopting an artificial approach based on the CA paradigm.

Subsequently, Codd at the end of the 60's and then Langton in the mid-80's, have proposed a simplified model compared to the von Neumann original one for the self-reproduction with self-replicating structures.

von Neumann was convinced that the self-reproduction should incorporate the property of universal computability; for this reason his model was very complex. Codd even if shared the von Neuman hypothesis proposed an alternative model with 8 states [27].

However, Langton proposed a model with self-replicating structures (Langton's loops) not computational equivalent to the Turing Machine, thus he has shown that the universal computability property is a sufficient condition for self-reproduction but not a necessary condition [105].

Chou and Reggia [26] have demonstrated, for the first time, that it is

possible to implement CA with ‘general’ transition functions in order to emerge self-replicating structures with initial configurations completely random. These structures may have different characteristics and shapes and they interact with other structures that simultaneously emerge in the cellular space.

Other interesting works that reflect the original von Neumann study regarding the self-reproduction problem were conducted by Azpeitia and Ibáñez [5] and Bilotta [14].

As it can be seen from the research brach related to the self-reproduction, the Artificial life has producted hypothesis and original results of extreme interest, both from the theoretical point of view and from that of the possible applications and, in this context, CA have played a very important role.

### 3.5.2 Lattice Gas Cellular Automata and Lattice Boltzmann Models

The fluid dynamics is the branch of physics that deals with the behavior of gases and liquids. The classical fluid dynamics is based on the Navier-Stokes equations that formalize the laws of conservation of mass and momentum.

The non-linearity of such equations is the main cause of the difficulty to apply them for not idealized cases [155] and for this reason an alternative approach to the study of fluid dynamics based on CA, namely Lattice Gas, emerged.

#### 3.5.2.1 Lattice Gas cellular automata

The basic idea of lattice gas is to model a fluid through a system of particles that can move, with constant velocity, only along the directions of a discrete lattice. Local laws are defined in order to ensure the invariance of the number of particles (conservation of mass) and the conservation of momentum.

More formally, a Lattice Gas Auotmaton (LGA) is a lattice of cells  $\vec{r}$ . Each cell  $\vec{r}$  contains  $z + 1$  quantities  $n_i(\vec{r}, t)$ ,  $i = 0, \dots, z$  where  $z$  is the coordination number. Neighbors of  $\vec{r}$  are obtained as  $\vec{r} + \vec{v}_i$ , where  $\vec{v}_i$  are given vectors and by convention  $\vec{v}_0 = 0$ . The LGA dynamics consist of two steps. The first one is the interaction step where the quantities  $n_i$  locally collide and new values  $n'_i$  are computed according a predefined collision operator  $\Omega_i(n)$ . The second step regards the propagation by sendind the quantity  $n'_i(\vec{r})$  to the neighboring site along lattice direction  $\vec{v}_i$ .

The first lattice-gas cellular automata (LGCA) was proposed in 1973 [79], for the first time, by Hardy, Pomeau and de Pazzis. It is named HPP and represents the simplest LGCA. In particular, HPP is a two-dimensional

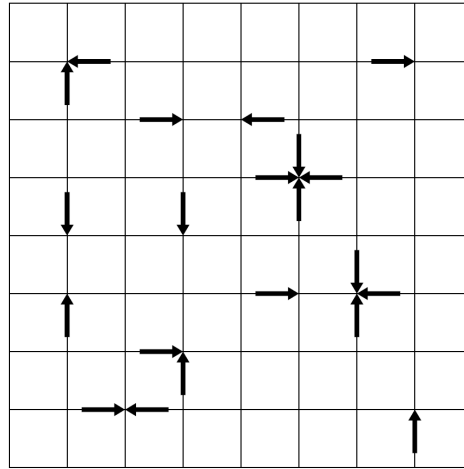


Figure 3.7: Example of LGA on a square lattice. In this example  $n_i \in \{0, 1\}$ . The arrows directions indicate the sites with  $n_i = 1$ . The lattice direction are  $\vec{v}_1 = (1, 0)$ ,  $\vec{v}_2 = (0, 1)$ ,  $\vec{v}_3 = (-1, 0)$ ,  $\vec{v}_4 = (0, -1)$ .

lattice-gas cellular automata model over a square lattice. The vectors  $c_i (i = 1, 2, 3, 4)$  connecting nearest neighbors are called *lattice vectors*. At each node there are four cells (see Figure 3.7) each associated to a link with the nearest neighbor. Cells may be empty or occupied by at most one particle (exclusion principle). The evolution in time is deterministic and proceeds with local collisions and propagation along links to the nearest neighbors. The collision conserves mass and momentum while changing the occupation of the cells and when two particles enter a node from opposite directions and the other two cells are empty a head-on collision takes place which rotates both particles by  $90^\circ$  in the same sense.

The first LGA reproducing a correct hydrodynamic behavior has been introduced in 1986. Frish, Hasslacher and Pomeau showed that an LGCA over a lattice with a larger symmetry group than for the square lattice yields the Navier-Stokes equation in the macroscopic limit. This model, named FHP, presents an hexagonal symmetry.

In particular, the properties of FHP lattice can be described as following:

- The lattice shows hexagonal symmetry (the lattice is composed of triangles).
- Nodes are linked to six nearest neighbors located all at the same distance with respect to the central node.

- $c_i$  is the lattice vectors and it links the neighbor nodes.

$$c_i = \left( \cos \frac{\pi}{3}i, \sin \frac{\pi}{3}i \right), i = 1, \dots, 6 \quad (3.3)$$

- A cell is associated with each link at all nodes.
- Cells can be empty or occupied by at most one particle(exclusion principle)
- All particles have the same mass.
- The evolution proceeds by collisions C and streaming S (propagation) as:

$$\epsilon = S \circ C \quad (3.4)$$

where  $\epsilon$  is the evolution operator.

- The collisions are local.

As for HPP there are 2-particle head-on collisions but in contrast to HPP the FHP model encompasses nondeterministic rules. A pseudo-random choice is used where the rotational sense changes by chance for the whole domain from time step to time step or the sense of rotation changes from node to node but is constant in time.

The basic FHP model defines only two- and three-particle collisions, but it turns out that this is sufficient to yield the desired behavior. If two particles travelling in opposite directions meet at a node, then the particle pair is randomly rotated either clockwise or counterclockwise by sixty degrees. If three particles meet at a node in a symmetric configuration, then they collide in such a way that this configuration is *inverted*.

Please refer to [54] for a more detailed description of the FHP model.

### 3.5.2.2 Lattice Boltzmann Models

Lattice Boltzmann Models (LBM) have been introduced because the LGA are plagued by several diseases for Navier-Stokes equations computation. Lattice Boltzmann equations have been applied by Frisch et al [54] in 1987 to calculate the viscosity of LGCA.

In 1988 LBM have been used by McNamara and Zanetti as numerical method for hydrodynamic simulations [115]. LGCA have been replaced with LBM because the authors decided to completely eliminate the statistical noise that plagues the usual lattice-gas simulations so the boolean fields were replaced by continuous distributions over the FHP lattices.

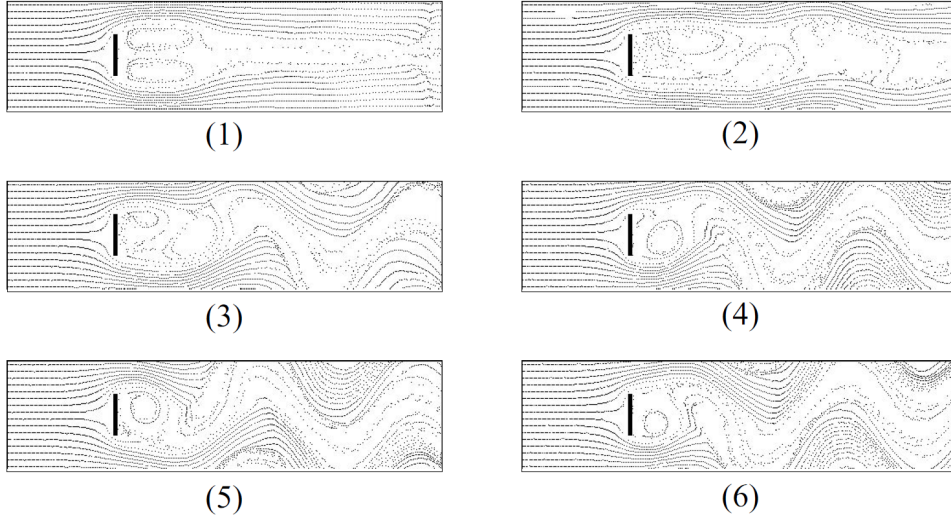


Figure 3.8: Simulation of a flow around a thin plate with a Boltzmann Lattice model. Figures 1 to 6 illustrate the evolution of the system.

Higuera and Jiménez introduced an alternative simulation procedure for lattice hydrodynamics [80], based on the lattice Boltzmann equation instead of on the microdynamical evolution. In particular, in this work the collision operator is expressed in terms of its linearized part with the introduction of few parameters to decrease viscosity.

Koelman [100], Qian et al. [134] and others replaced the collision operator with the Bhatnagar-Gross-Krook (BGK) approximation. This new model, compared to lattice gases, is noise-free and collisions are not anymore defined explicitly.

Lattice Boltzmann models are most popular today. A significant advantage of the LBM, compared to the Reticular Gas, is that only the density of particles is taken into account so the number of components of the system is considerably reduced.

The dynamics of a LBM [134] can be described as follows:

$$f_i(\vec{r} + \tau \vec{v}_i, t + \tau) - f_i(\vec{r}, t) = \Omega_i(f_i(\vec{r}, t)) = \frac{1}{\xi} (f_i^{(eq)}(\vec{r}, t) - (f_i(\vec{r}, t))) \quad (3.5)$$

where  $f_i(\vec{r}, t)$  represents the density of particles that at time  $t$  are in the cell  $\vec{r}$  with velocity  $\vec{v}_i$ ;  $f_i^{(eq)}(\vec{r}, t)$  is the local equilibrium distribution and  $\xi$  is the number of calculation steps to reach the equilibrium at local neighborhood level.

The  $f_i^{(eq)}(\vec{r}, t)$  function specifies the conditions of local equilibrium of the



system in function of density,  $\rho = \sum f_i$  and of momentum,  $\rho\vec{u} = \sum f_i\vec{v}_i$ , of fluid in the cell. The parameter  $\xi$  expresses how the system is dependent of the fluid viscosity  $v = K(\xi - 1/2)$  where  $K$  is a constant that depends on the lattice geometry.

Unlike Lattice Gas Automaton, the viscosity becomes an explicit parameter of the model. Figure 3.8 illustrates the dynamics of the BKG model [134] in the simulation of a flow around a thin plate.

## Chapter 4

# Modelling Macroscopic phenomenas with Cellular Automata

As discussed in the previous chapter, CAs have been applied with success for modelling and simulating complex systems, whose dynamics can be described in terms of local interactions. Among different fields, fluid-dynamics is one of the most important fields of application for CA and, in this research branch, many different CA-based methods were used to simulate fluid flows. Lattice Gas Automata models [53] were introduced for describing the motion and collision of particles on a grid and it was shown that such models can simulate fluid dynamical properties. The continuum limit of these models leads to the Navier-Stokes equations. Lattice Gas models can be regarded as microscopic models, as they describe the motion of fluid *particles* which interact by scattering.

An advantage of Lattice Gas models is that the simplicity of particles, and of their interactions, allow for the simulation of a large number of them, making it therefore possible to observe the emergence of flow patterns. Furthermore, since they are cellular automata systems, it is easy to run simulations with parallel computing. A different approach to LGA is represented by Lattice Boltzmann models [115] in which the state variables can take continuous values, as they are supposed to represent the density of fluid particles, endowed with certain properties, located in each cell (here space and time are discrete, as in lattice gas models). Both Lattice Gas and Lattice Boltzmann Models have been applied for the description of fluid turbulence [25, 156].

Because many complex natural phenomena evolve on very large areas, they are therefore difficult to be modelled at a microscopic level of descrip-

tion. Among these, lava flows can be considered, at the same time, one of the most dangerous and difficult phenomena to be modelled as, for instance, temperature drops along the path by locally modifying the magma dynamical behaviour (because of the effect of the strong temperature-dependence of viscosity). Furthermore, lava flows generally evolve on complex topographies that can change during eruptions, due to lava solidification, and are often characterised by branching and rejoining of the flow. Complex Cellular Automata (CCA) represent a valid alternative to classical CA regarding macroscopic phenomena.

This chapter focuses on CCA approach and its application to SCIARA-fv2, the latest release of the SCIARA family models for simulating lava flows. After a briefly description of CCA, the algorithm to model surface flows is, afterwards, introduced. The SCIARA-fv2 model is also illustrated with results of its application to a real case occurred on Mt Etna. Final discussions conclude the chapter.

## 4.1 Complex Cellular Automata

As regards the modeling of natural complex phenomena, Crisci and co-workers proposed a method based on an extended notion of homogeneous CA, firstly applied to the simulation of basaltic lava flows [35], which makes the modeling of spatially extended systems more straightforward and overcomes some unstated limits of the classical CA, such as having few states and look-up table transition functions [57]. Mainly for this reason, the method is known as Complex Cellular Automata, even though it was also known as Macroscopic Cellular Automata [151] or Multicomponent Cellular Automata [4].

CCA were in fact adopted for the simulation of many macroscopic phenomena, such as lava flows [2], debris flows [45], density currents [145], water flux in unsaturated soils [70], soil erosion/degradation by rainfall [39] as well as pyroclastic flows [2], bioremediation processes [58] and forest fires [165].

Informally, CCA, compared to classical CA, are different because of the following reasons:

- the state of the cell must account for all the characteristics, which are assumed to be relevant to the evolution of the system: these refer to the space portion of the cell. Each characteristic corresponds to a *substate*. The state of the cell is divided into substates and permitted values for a substate must form a finite set. The set of the possible states of a cell represents the global state of the cell and is given by the Cartesian product of the sets of the substates.

- the state of the cell can be decomposed in substates and the transition function may be split into local interactions: the *elementary processes*. Each of them represents a particular aspect that rules the dynamic of the considered phenomenon. Different elementary processes may involve different neighborhoods. The CA neighborhood is given by the union of all the neighborhoods associated to each processes. If the neighborhood of an elementary process is limited to a single cell, such process is considered as an *internal transformation*.
- a set of global parameters to reproduce the several different dynamic behaviors of the considered phenomenon is defined .
- a subset of the cells is also influenced by external influences, represented by a function. External influences are used to model those features that are difficult to describe as local interactions.

Formally, a CCA is a 7-tuple:

$$A = \langle \mathbb{Z}^d, Q, X, P, \tau, E, \gamma \rangle \quad (4.1)$$

where:

- $\mathbb{Z}^d$  is the d-dimensional cellular space;
- $Q = Q_1 \times Q_2 \times \dots \times Q_n$  is the set of states of the cell obtained as the Cartesian product of *substates*  $Q_1 \times Q_2 \times \dots \times Q_n$  each one representing a particular feature of the phenomenon to be modelled;
- $X$  is the geometrical pattern that specifies the neighbourhood relationship;
- $P = p_1, p_2, \dots, p_p$  is the set of CA *parameters*. They allow to tune the model for reproducing different dynamical behaviours of the phenomenon of interest;
- $\tau : Q^m \rightarrow Q$  is the transition function for the CA and it is splitted in *elementary processes*  $\tau_1, \tau_2, \dots, \tau_s$ , each one describing a particular aspect that rules the dynamic of the considered phenomenon.
- $E = E_1 \cup E_2 \cup \dots \cup E_l \subseteq \mathbb{Z}^d$  is the set of cells of  $\mathbb{Z}^d$  that are subject to *external influences*. External influences were introduced in order to model features which are not easy to be described in terms of local interactions;
- $\gamma = \{\gamma_1, \gamma_2, \dots, \gamma_t\}$  is the finite set of functions that define the external input for the CA.

## 4.2 Modelling surface flows through CA

Many geological processes like lava or debris flows can be described in terms of local interactions and thus modelled by CCA. By opportunely discretizing the surface on which the phenomenon evolves, the dynamics of the system can be in fact described in terms of flows of some quantity from one cell to the neighbouring ones. Moreover, as the cell dimension is a constant value throughout the cellular space, it is possible to consider characteristics of the cell (i.e. substates), typically expressed in terms of volume (e.g. lava volume), in terms of thickness. This simple assumption permits to adopt a straightforward but efficacious strategy that computes outflows from the central cell to the neighbouring ones in order to minimize the non-equilibrium conditions.

In the CCA approach, by considering the third dimension (the height) as a property of the cell, outflows can be computed by procedures based on one of *distribution* algorithms as the Minimisation Algorithm of the Differences [57], briefly described in the next Section.

### 4.2.1 The Minimization Algorithm of the Differences

The Minimisation Algorithm of the Differences (MAD), proposed by Di Gregorio and Serra [57], reduces the non-equilibrium conditions by minimizing quantities between central cell and its neighbours. In other words, outflows from the central cell to the other  $n$  neighbouring cells must be determined in order to minimise the differences of a quantity  $q$  in the neighbouring cells.

The MAD is based on the following assumptions:

- two parts of the considered quantity must be identified in the central cell: these are the unmovable part,  $u(0)$ , and the mobile part,  $m$ ;
- only  $m$  can be distributed to the adjacent cells. Let  $f(x, y)$  denote the flow from cell  $x$  to cell  $y$ ;  $m$  can be written as:

$$m = \sum_{i=0}^{\#X} f(0, i) \quad (4.2)$$

where  $f(0, 0)$  is the part which is not distributed, and  $\#X$  is the number of cells belonging to the  $X$  neighbourhood. It is worth to note that this definition preserves the principle of conservation of mass for the distributable quantity  $m$ .

- the quantities in the adjacent cells,  $u(i)$  ( $i = 1, 2, \dots, \#X$ ) are considered unmovable;
- let  $c(i) = u(i) + f(0, i)$  ( $i = 0, 1, \dots, \#X$ ) be the new quantity content in the  $i$ -th neighbouring cell after the distribution and let  $c_{min}$  be the minimum value of  $c(i)$  ( $i = 0, 1, \dots, \#X$ ). The outflows are computed in order to minimise the following expression:

$$m = \sum_{i=0}^{\#X} (c(i) - c_{min}) \quad (4.3)$$

Basically, the MAD operates as follows:

1. the following average is computed:

$$a = \frac{m + \sum_{i \in A} u(i)}{\#A}$$

where  $A$  is the set of not eliminated cells (i.e. those that can receive a flow); note that at the first step  $\#A = \#X$ ;

2. cells for which  $u(i) \geq a$  ( $i = 0, 1, \dots, \#X$ ) are eliminated from the flow distribution and from the subsequent average computation;
3. the first two points are repeated until no cells are eliminated; finally, the flow from the central cell towards the  $i$ -th neighbour is computed as the difference between  $u(i)$  and the last average value  $a$ :

$$f(0, i) = \begin{cases} a - u(i) & i \in A \\ 0 & i \notin A \end{cases}$$

An example of MAD application is reported in Figure 4.1.

Note that the simultaneous application of the minimization principle to each cell gives rise to the global equilibrium of the system. The correctness of the algorithm is stated in [57].

### 4.3 The Cellular Automata Model SCIARA-fv2

As reported in [151], SCIARA is a family of bi-dimensional CCA lava flow models, successfully applied to the simulation of many real cases such as

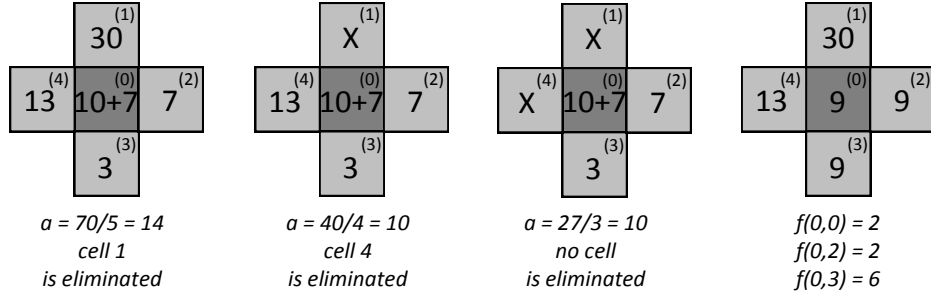


Figure 4.1: Example of application of the Minimization Algorithm of the Differences by considering a bidimensional CA with square cells and Von Neumann neighborhood.

the 1991 Valle del Bove (Italy) lava event [8] and the 2001 Mt. Etna (Italy) Nicolosi lava flow [36], which occurred on the same volcano and was employed as a case study in this thesis.

In the latest model of the SCIARA family, a Bingham-like rheology has been introduced for the first time, in spite of the previous simplified rheological model in which viscosity effects and critical height were modelled in terms of lava adherence. Regarding the previous implementation of the model, depending on temperature, a fixed amount of lava cannot flow out from the cell, while the part that moves is determined by a version of the Minimization Algorithm of the Differences that does not consider the effect of viscosity. Conversely, the rheology here adopted is inspired to the Bingham model and therefore the concepts of critical height and viscosity are explicitly considered. In particular, lava can flow out if and only if its thickness overcomes a critical value (critical height), so that the basal stress exceeds the yield strength. Moreover, viscosity is accounted in terms of flow relaxation rate, being this latter the parameter of the distribution algorithm that influences the amount of lava that actually leaves the cell.

In formal terms, the SCIARA-fv2 model is defined as:

$$SCIARA - fv2 = \langle R, L, X, Q, P, \tau, \gamma \rangle \quad (4.4)$$

where:

- $R$  is the set of square cells covering the bi-dimensional finite region where the phenomenon evolves;
- $L \subset R$  specifies the lava source cells (i.e. craters);

Parameter	Meaning	Unit	Best value
$w$	Cell side	[ $m$ ]	10
$t$	CA clock	[ $s$ ]	120
$T_{sol}$	Temperature of solidification	[ $K$ ]	1143
$T_{vent}$	Temperature of extrusion	[ $K$ ]	1360
$r_{T_{sol}}$	Relaxation rate at the temperature of solidification	-	0.5
$r_{T_{vent}}$	Relaxation rate at the temperature of extrusion	-	0.95
$h_{cT_{sol}}$	Critical height at the temperature of solidification	[ $m$ ]	40
$h_{cT_{vent}}$	Critical height at the temperature of extrusion	[ $m$ ]	1.5
$\delta$	Cooling parameter	-	2.8
$\rho$	Lava density	[ $Kgm^{-3}$ ]	2600
$\epsilon$	Lava emissivity	-	0.9
$\sigma$	Stephan-Boltzmann constant	[ $Jm^{-2}s^{-1}K^{-4}$ ]	$5.68 \cdot 10^{-8}$
$c_v$	Specific heat	[ $Jkg^{-1}K^{-1}$ ]	1150

Table 4.1: List of SCIARA-fv2 parameters with values related to the simulation of the 2006 Etnean lava flows

- $X = \{(0, 0), (0, 1), (-1, 0), (1, 0), (0, -1), (-1, 1), (-1, -1), (1, -1), (1, 1)\}$  is the set that identifies the pattern of 8 cells influencing the cell state change (i.e., the Moore neighborhood);
- $Q = Q_z \times Q_h \times Q_T \times Q_f^8$  is the finite set of states, considered as a Cartesian product of substates. SCIARA-fv2 substates, which describe relevant quantities representing a particular feature of the phenomenon to be modeled are: cell elevation a.s.l., cell lava thickness, cell lava temperature, and lava thickness outflows from the central cell toward neighbours, respectively;
- $P = \{w, t, T_{sol}, T_{vent}, r_{T_{sol}}, r_{T_{vent}}, h_{cT_{sol}}, h_{cT_{vent}}, \delta, \rho, \epsilon, \sigma, c_v\}$  is the finite set of parameters (invariant in time and space) which affect the transition function (refer to Table 4.1 for their specifications);
- $\tau : Q^9 \rightarrow Q$  is the cell deterministic transition function, applied to each cell at each time step, which describes the dynamics of lava flows, such as cooling, solidification and lava outflows from the central cell towards neighbouring ones. The elementary processes are described in the following Sections;
- $\gamma : Q_h \times N \rightarrow Q_h$  specifies the emitted lava thickness,  $h$ , from the source cells at each step  $k \in N$  ( $N$  is the set of natural numbers).



### 4.3.1 The lava flows computation (elementary process $\tau_1$ )

#### Viscosity computation

In the SCIARA-fv2 model the viscosity is modeled in terms of flow relaxation rate,  $r$ , according to the following power law:

$$\log r = a + bT \quad (4.5)$$

where  $T \in Q_T$  (cf. Table 4.1) is the lava temperature and  $a$  and  $b$  are coefficients determined by solving the system:

$$\begin{cases} \log r_{T_{sol}} = a + bT_{sol} \\ \log r_{T_{vent}} = a + bT_{vent} \end{cases}$$

#### Critical height computation

Critical height computation is similar to the viscosity one and mainly depends on lava temperature according to power law of the kind:

$$\log h_c = c + dT \quad (4.6)$$

where  $c$  and  $d$  are obtained by solving the system:

$$\begin{cases} \log hc_{T_{sol}} = c + dT_{sol} \\ \log hc_{T_{vent}} = c + dT_{vent} \end{cases}$$

#### Lava outflows computation

Before applying the Minimisation Algorithm of the Differences to compute lava outflows from the central cell (with index 0) towards the  $i$ -th neighbour, a control is performed to eliminate cells that are not able to receive lava due their state condition. Let  $\bar{z}_i \in Q_z$  be the topographic altitude and  $h_i \in Q_h$  the lava thickness, for the generic cell  $i$  the conditions to participate to the lava distribution are:

1.  $\bar{z}_0 > \bar{z}_i \wedge h_0 \geq h_i$
2.  $(\bar{z}_0 + h_0 > \bar{z}_i + h_i) \wedge \neg(\bar{z}_0 > \bar{z}_i \wedge h_0 \geq h_i)$

In the case 1, the generic cell  $i$  is not eliminated if both the topographic altitude of the central cell is greater than that of the neighbouring cell and

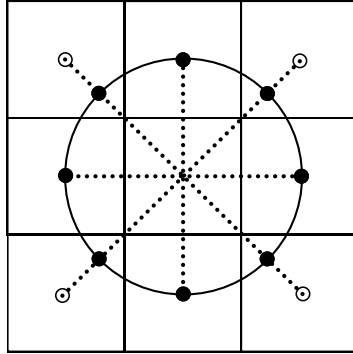


Figure 4.2: Reference schema for cells altitude determination in the Moore neighbourhood. Altitudes of cells along the von Neumann neighbourhood correspond to DEM values. Those along diagonals are taken at the intersection between the diagonal line and the circle with radius  $w$  (cf. Table 4.1), so that the distance with respect to the centre of the central cell is constant for each adjacent neighbour

the debris thickness of the central cell is greater than or equal to that of the neighbouring cell. The second condition (case 2) for the central cell to not be eliminated is that the total height of the central cell overcomes that of the neighbouring cell.

In the latest model of SCIARA family the anisotropic problem was solved by introducing a fictitious topographic alteration along diagonal cells. In a standard situation of non-altered heights, cells along diagonals result in a lower elevation with respect to the remaining ones (which belong to the von Neumann neighbourhood), even in case of constant slope. This is due to the fact that the distance between the central cell and diagonal neighbours is greater than of the distance between the central cell and orthogonal adjacent cells (cf. Figure 4.2). This introduces a side effect in the distribution algorithm, which operates on the basis of height differences. If the algorithm perceives a greater difference along diagonals, it will erroneously privilege them by producing greater outflows. In order to solve this problem, it was considered the height of diagonal neighbours taken at the intersection between the diagonal line and the circle with radius  $w$  (cf. Table 4.1) with center in the central cell. This solution permits to have constant differences in level in correspondence of constant slopes, and the distribution algorithm can work properly even on the Moore neighbourhood. According to this strategy, the topographic altitude conditions is:

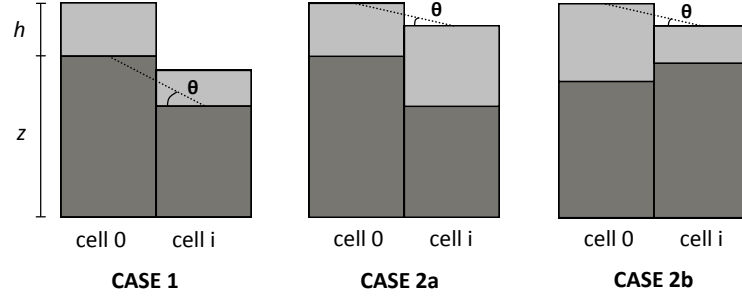


Figure 4.3: Cases in which the generic neighbour (cell  $i$ ) is not eliminated by the Minimisation Algorithm of the Difference and can thus receive a flow by the central cell (cell 0). Note that the slope angle  $\theta$ , considered in the critical height computation, depends on the particular case.

$$\bar{z}_i = \begin{cases} z_i & i = 0, 1, \dots, 4 \\ z_0 - (z_0 - z_i)/\sqrt{2} & i = 5, 6, \dots, \#X \end{cases} \quad (4.7)$$

In Figure 4.3 there are three different cases in which the generic neighbour  $i$  is not eliminated during the MAD computation. The case 1 indicates the situation in which lava moves downslope and a lower amount of lava is found in the neighbouring cell. In this case, lava in the neighbour does not represent an obstacle. Consequently, the distribution algorithm considers only the topographic elevation of the neighbour as unmovable part  $u(i)$  and the slope angle,  $\theta$ , is computed accordingly (cf. Figure 4.3 - CASE 1). The case 2 indicates the situation in which lava moves down or up-slope and lava in the neighbouring cell represents an obstacle. In fact, if  $\bar{z}_0 > \bar{z}_i$  (lava moves downslope),  $h_i > h_0$  must hold, which can indicate a situation where lava motion is slowing down, for instance due to cooling or because a counter-slope begins (cf. Figure 4.3 - CASE 2a). Still, if  $z_0 < z_i$ , lava moves upslope and both neighbouring altitude and lava content oppose to lava motion (cf. Figure 4.3 - CASE 2b). In these cases, the distribution algorithm considers both the topographic elevation and lava content of the neighbours as  $u(i)$  and the slope angle,  $\theta$ , is computed accordingly (cf. Figure 4.3 - CASE 2a and CASE 2b).

The Minimisation Algorithm of the Differences application considers the following quantities:

$$u(0) = \bar{z}_0;$$

$$m = h_0;$$

$$\bar{u}_i = \begin{cases} \bar{z}_i & \text{for case 1} \\ \bar{z}_i + h_i & \text{for case 2} \end{cases}$$

According to the Bingham-like rheology here adopted, actual lava outflows,  $h(0, i)$ , are computed as:

$$h(0, i) = \begin{cases} f(0, i) \cdot r & h_0 > h_c \cdot \cos \theta \\ 0 & h_0 \leq h_c \cdot \cos \theta \end{cases}$$

where  $\theta$  is the slope angle as shown in Figure 4.3. The relaxation rate factor,  $r$ , computed according to equation 4.5, plays the role of the viscosity in the context of the Minimization Algorithm, while the critical height,  $h_c$ , computed according to equation 4.6, has the same meaning as that of a Bingham fluid.

### 4.3.2 Temperature variation and lava solidification computations (elementary process $\tau_2$ )

The temperature, for each CA cell, is determined by two process steps. In the first step, the temperature is calculated as the weighted average of residual lava inside the cell and lava inflows from neighboring cells:

$$T_{avg} = \frac{h_r T_0 + \sum_{i=1}^9 h(i, 0) T_i}{h_r + \sum_{i=1}^9 h(i, 0)} \quad (4.8)$$

where  $h_r \in Q_h$  is the residual lava thickness in the central cell after the outflow distribution,  $T \in Q_T$  is the lava temperature and  $h(i, 0)$  is the outflow lava from the  $i$ -th neighboring cell towards the central one. The second step, to calculate the final temperature, considers the thermal energy loss due to lava surface radiation:

$$T = \frac{T_{avg}}{\sqrt[3]{1 + \frac{3T_{avg}^3 \epsilon \sigma t \delta}{\rho c_v w^2}}} \quad (4.9)$$

where  $\epsilon, \sigma, t, \delta, \rho, c_v$  and  $w$  are specified in Table 4.1.

When the lava temperature drops below the threshold  $T_{sol}$ , lava solidifies and the cell altitude increases by an amount to solidified lava thickness. Consequently, in the same cell the lava thickness is set to zero.

## 4.4 SCIARA-fv2 model applications

In general, deterministic CA for the simulation of macroscopic fluids present a strong dependence on the cell geometry and directions of the cellular space. Due to the discretization of the surface where the phenomenon evolves, diagonal cells can be greatly privileged in flow distribution and thus lava can spread preferentially in these directions. In order to solve the problem, different solutions have been proposed in literature, such as the adoption of hexagonal cells (e.g. [36, 40]) or Monte Carlo approaches (e.g. [166]). The first solution, however, does not solve perfectly the problem on ideal surfaces, while the second one has the disadvantage of giving rise to nondeterministic simulation models.

### Ideal surfaces

This applications of SCIARA shows how the model does not present the anisotropic problem on an ideal surface, represented by an octagonal-base pyramid having faces inclined by an angle  $\alpha = 5^\circ$ . The pyramid is represented by a 10m cell size DEM of 203 columns and 203 rows. By locating the lava source at the top of the structure, both flows along diagonals and orthogonal directions of the square cellular space are observed. Figure 4.5 shows the results of two test cases in which a constant effusion rate, equal to  $1 \text{ m}^3\text{s}^{-1}$ , is emitted for a total of 6 days, and no temperature loss is considered. The first simulation is obtained by considering the actual topographic heights of the cells, while the second by taking into account the topographic corrections. As it can be seen, the anisotropic problem is quite significant in the first case, in which diagonal flows, as expected, reach the base of the pyramid more rapidly with respect to those on the orthogonal directions. In the second case, in which topographic alteration are considered, the problem is practically absent and all flows reach the base of the pyramid at the same moment.

### The 2006 Valle del Bove Lava Flow case study

Etna's July 2006 eruption began during the night of 14 July, when a fissure opened on the east flank of the South- East Crater. Two vents (cf. key 4 of Figure 4.4) fed lava flow towards east into the Valle del Bove. The effusion rate trend here adopted is in agreement with that considered by Neri et al.[122] and is shown in Figure 4.4.

A preliminary calibration allowed to individuate values for models parameters, which are listed in Table 4.1. The corresponding CA simulation,

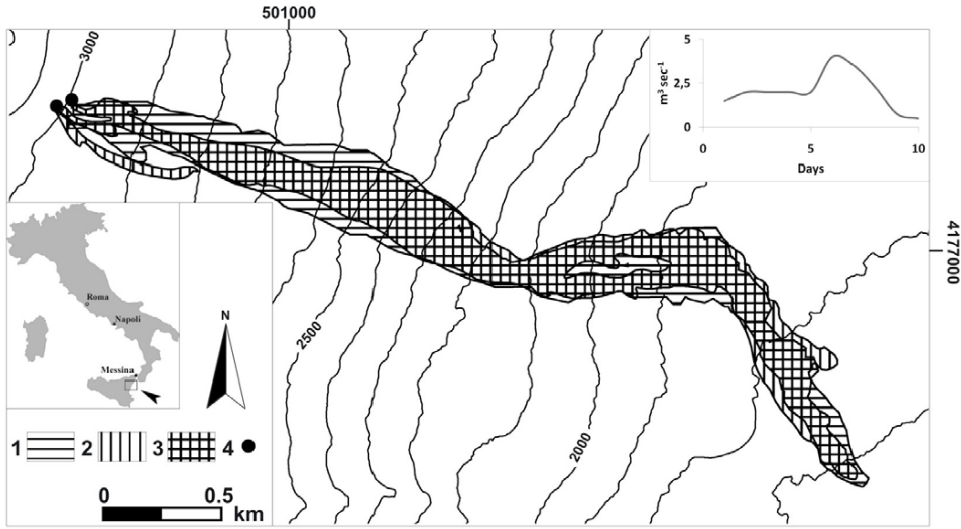


Figure 4.4: Simulation of the 2006 Etnean lava flow by the CA model SCIARA. Key: 1) simulated event; 2) real event; 3) overlapping area between real and simulated events; 4) lava vents sources. The effusion rate is shown in the upper right corner of the figure.

performed on a 10m cell size DEM of 797 columns and 517 rows, is shown in Figure 4.4.

In order to quantitatively evaluate the goodness of the simulation, the  $e_1$  fitness function was adopted, which provides a measure of the overlapping (in terms of areal extent) between the real and simulated event. By considering  $R$  and  $S$  the sets of CA cells affected by the real and simulated event, respectively. Let  $m(R \cap S)$  and  $m(R \cup S)$  be the measure of their intersection and union, respectively. The fitness function  $e_1$  is defined as follows:

$$e_1 = \sqrt{\frac{m(R \cap S)}{m(R \cup S)}} \quad (4.10)$$

Note that the function  $e_1$  gives values belonging to the interval  $[0, 1]$ . Its value is 0 if the actual and simulated events are completely disjoint, being  $m(R \cap S) = 0$ ; it is 1 in case of a perfect overlap, being  $m(R \cap S) = m(R \cup S)$ . As the Figure 4.4 shows, the simulation does not differ significantly from the real case, as confirmed by the more than satisfying value of the fitness function,  $e_1 = 0.8$ . The goodness of the simulation is also confirmed in terms of runout, as the travelled distance from the sources of the simulated event is practically the same as the real one.

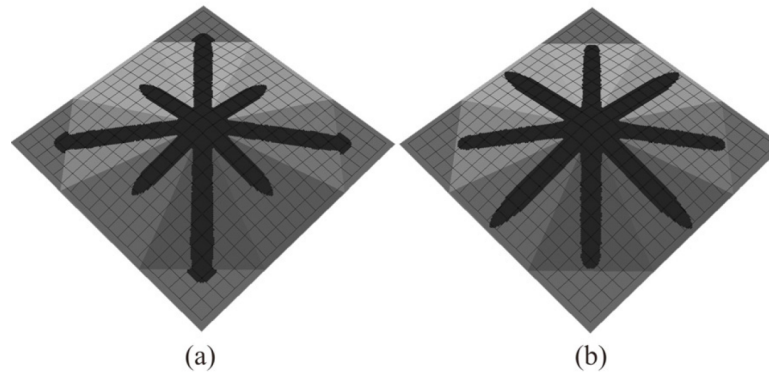


Figure 4.5: SCIARA simulations on an octagonal-base pyramid with faces inclined by an angle  $\theta = 5^\circ$ , performed for evaluating the anisotropic flow direction problem: a) case in which actual cell topographic elevations are considered; b) case in which equation 4.7 is applied and topographic corrections along diagonals considered. Note that the square lattice in both figures is only indicative of the cellular space orientation and does not correspond to the actual cellular space in terms of number of rows and columns.

### The July 2001 Etnean eruption

The SCIARA-fv2 model was also used in the Etnean eruptive crisis of July/August 2001, in particular by focusing on the lava flow emitted by the fracture of Mount Calcarazzi, 2100 m a.s.l. This emission, which started on 18 July, created the main danger for the towns of Nicolosi and Belpasso and it was, in its maximum extension, only 4 km away from Nicolosi. As reported in [36], a grid of hypothetical vents was considered above the towns of Nicolosi and Belpasso and lava flow rates were generated on the documented low rates of the previous Etnean eruptions [33]. Some hypothesised scenarios were proven to be similar to the 2001 real event and in the second day of the crisis simulations are prepared, considering the positions of vents and lava flow rates of the first day.

The simulation with the better areal comparison between the real and simulated ones of the first day is selected. Several scenarios are developed by considering different flow rates. In the successive days the protocol of the previous days is replicated.

When simulations show that lava advance threaten inhabited areas, morphology alterations (embankments or canalisations) are introduced in the simulation in order to evaluate the possibility effects to divert lava flows.

Figure 4.6 illustrates the first 10 days of the real and simulated event, respectively.

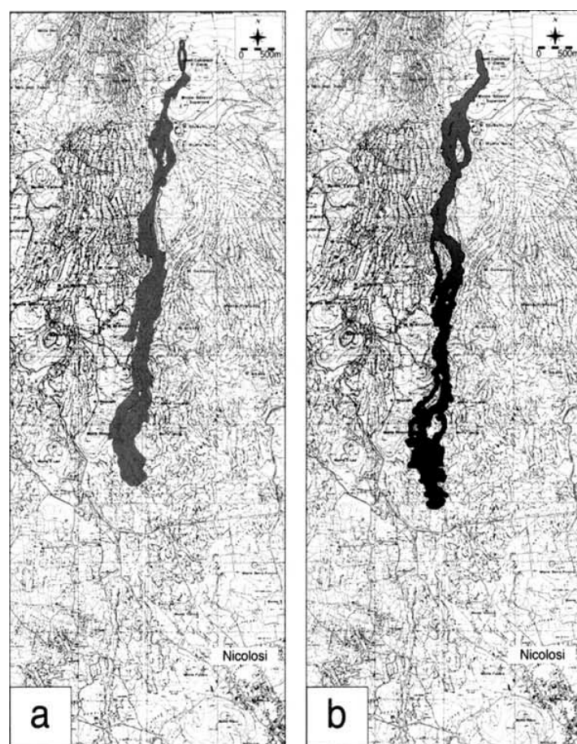


Figure 4.6: The observed (a) and simulated (b) event of the first 10 days lava flow from Mount Calcarazzi fracture (July 2001 Event).

Finally, the obtained simulations (numerous events carried out by using high rates) showed how the town of Belpasso was in a safe situation, while Nicolosi could have been threatened in many events.

## 4.5 Discussion

In this chapter, the CCA approach and its application to SCIARA-fv2, the latest release of the SCIARA family models for simulating lava flows, were presented.

As discussed, when the main features of the phenomena to be simulated can be directly described at a macroscopic level thus disregarding microscopic aspects, CCA represent an alternative to classical CA.

SCIARA-fv2 considers a Bingham-like rheology and introduces a square tessellation of the cellular space instead of the previously adopted hexagonal one, which was considered in the earlier versions to limit the effect of the anisotropic flow direction problem. As shown, the model is able to solve the problem on an ideal inclined surface. This result is particularly significant,



being SCIARA a deterministic model, as all the previously proposed solutions refer to probabilistic CA simulation models. A preliminary calibration also allowed to reproduce two real cases of study, namely the 2001 and 2006 lava flows at Mt Etna (Italy), with a great level of accuracy. In fact, a high degree of overlapping between the real and the simulated event and a perfect fitting in terms of run-out were obtained.

## Chapter 5

# Accelerating Cellular Automata simulations of lava flows

Nowadays, parallel computing is seen as a cost-effective method for the fast and efficient resolution of computationally large and data-intensive problems [104]. The great expansion of High Performance Computing (HPC) in different scientific and engineering fields has permitted the use of numerical simulations as a tool for solving complex equation systems which rule the dynamics of complex real phenomena, through which researchers can study the modelling of, for instance, a lava flow, fire spreading or traffic simulation. Usually, the modeler has to implement proper optimization strategies and when possible, parallelize the program. The type of parallelization needed in this latter phase depends on the kind of available parallel architecture. For instance, in the case of a distributed memory machine (such as Beowulf clusters), this can be accomplished by means of MPI - Message Passing Interface [150]. On the contrary, in the case of a multicore architectures, a shared-memory or multithread implementation based on OpenMP [22] can result in a better and more efficient solution.

In recent years however, parallel computing has undergone a significant revolution with the introduction of GPGPU technology (General-Purpose computing on Graphics Processing Units), a technique that uses the graphics card processor (the GPU) for purposes other than computer graphics. Currently, GPUs outperform CPUs on floating point performance and memory bandwidth, both by a factor of roughly 100. Although the incredible processing power of graphic processors may be used for general purpose computations, a GPU may not be suitable for every computational problem: only a parallel program that results optimized for GPU architectures can fully take advantage of the performance of GPUs. In fact, the performance of a GPGPU program that does not sufficiently exploit a GPU's capabilities

can often be worse than that of a simple sequential one running on a CPU, such as when data transfer from main memory to video memory results crucial. Nevertheless, GPU applications to the important field of Computational Fluid Dynamics (CFD) are increasing both for quantity and quality among the Scientific Community [162, 172, 49].

Among the different methodologies used for modelling geological processes, such as numerical analysis, high order difference approximations and finite differences, Cellular Automata [167] have proven to be particularly suitable when the behaviour of the system to be modelled can be described in terms of local interactions. As seen in Chapter 3, originally introduced by von Neumann in the 1950s to study self-reproduction issues, CA are discrete computational models widely utilized for modeling and simulating complex systems. Well known examples are the Lattice Gas Automata and Lattice Boltzmann methods [156] which are particularly suitable for modelling fluid dynamics at a microscopic level of description. However, many complex phenomena (e.g. landslides or lava flows) are difficult to be modeled at such scale, as they generally evolve on large areas, thus needing a macroscopic level of description. Moreover, as discussed in chapter 4, since it may be difficult to model these phenomena through standard approaches such as differential equations (cf. [113], for the case of lava flows), Complex Cellular Automata [57] can represent a valid alternative.

Several successful attempts have been carried out regarding solutions for parallelizing CCA simulation models. Among others, CAMELot [56] and libAuToti [153] represent valid solutions for implementing and automatically parallelizing CCA models on distributed memory machines while for shared memory architectures, some effective OpenMP parallelizations have been implemented for CA-like models, such as for fire spread simulation [86], Lattice Boltzmann models [64] or lava flow modelling [127]. However, only some examples of GPGPU applications for CA-like models do exist [161, 173, 61] and none regarding the CCA approach.

This chapter illustrates several different implementation strategies, by adopting four different GPGPU devices, to accelerate both single and multiple simultaneous running of SCIARA-fv2 model using CUDA. The first part of the chapter focuses on parallel implementation, in GPGPU environment, of the single lava episode, whereas the second part of the chapter regards a CUDA approach to accelerate simultaneous simulation of a large number of lava simulations using GPU. In the following sections, after a brief overview of GPGPU paradigm together with the CUDA framework, implementations and performance analysis referred to different benchmark simulations of a real event are reported. Conclusions and possible outlooks are shown at the end of the chapter.

## 5.1 The NVIDIA CUDA programming approach: a quick overview

The principle aim of parallel computing, when referred to computational science, is to reduce the processing time of particularly complex simulations, owing to the use of more processing units that collaborate to simultaneously solve the problem. Among others, M.J. Flynn has provided a comprehensive classification of parallel architectures where the category of a specific parallel computer depends on its ability to manage, in parallel, the instruction and/or data stream [68]. Another classification of parallel computers may be based on the kind of memory architecture, whether shared or distributed [104]. In distributed memory architectures, each processor unit has access to a different memory. In this case, it is necessary to have some kind of communication between the various processing units. In shared-memory architectures, all processing units access the same memory, even simultaneously.

As alternative to standard parallel architecture, the term GPGPU (General-Purpose computing on Graphics Processing Units) refers to the use of the card processor (the GPU) as a parallel device for purposes other than graphic elaboration. In recent years, mainly due to the stimulus given by the increasingly demanding performance of gaming and graphics applications in general, graphic cards have undergone a huge technological evolution, giving rise to highly parallel devices, characterized by a multithreaded and multicore architecture and with very fast and large memories.

In general, a GPU consists in a number of SIMD (Single Instruction, Multiple Data) multiprocessors with a limited number of floating-point processors that access a common shared-memory within the multiprocessor. To better understand the enormous potential of GPUs, some comparisons with the CPU are noticeable: a medium-performance GPU (e.g. the NVIDIA Geforce GT200 family) is able to perform nearly 1000 GFLOPS (Giga Floating Point Operations per Second), while a mid-range Intel Core i7 processor has barely 52 GFLOPS. In addition, the most interesting aspect still is the elevated parallelism that a GPU permits. For instance, the NVIDIA GeForce GTX 680 has 8 Streaming Multiprocessor's (SMX) each with 192 processors for a total of 1536 basic cores, while a standard multi-core CPU has few, though highly-functional, cores.

Another motivation of GPUs increasing utilization as parallel architecture regards costs. Until a few years ago, in order to have the corresponding computing power of a medium range GPU of today, it was necessary to spend tens of thousands of Euros. Thus, GPGPU has not only led to a drastic reduction

of computation time, but also to significant cost savings. Summarizing, it is not misleading to affirm that the computational power of GPUs has exceeded that of PC-based CPUs by more than one order of magnitude while being available for a comparable price. The two leading world producers of video cards, ATI and NVIDIA, have developed programming frameworks which permit, by means of an application programming interface (API), the full exploitation of all the processing capabilities of these devices. ATI's CTM (Close-to-Metal) allows the programming and interaction with the GPU in a lower-level manner than what is offered by NVIDIA's CUDA and thus an implementation with the former usually is more difficult to realize. However, recently ATI (acquired in 2006 by AMD, the world's leading microprocessor manufacturer together with Intel) after the development of its Stream SDK high-level programming environment, has completely switched to OpenCL (Open Computing Language), the standard for parallel programming of heterogeneous systems developed by the Khronos consortium [102]. On the other hand, the NVIDIA CUDA technology [126], supported on Windows and Linux Operating systems, permits software development of applications by adopting the standard C language, libraries and drivers. The CUDA programming model provides three key abstractions: the hierarchy with which the threads are organized, the memory organization and the functions that are executed in parallel, called kernels. These abstractions allow the programmer to partition the problem into many sub-problems that can be handled and resolved individually.

### 5.1.1 CUDA Threads and Kernels

A GPU can be seen as a computing device that is capable of executing an elevated number of independent threads in parallel. In addition, it can be thought of as an additional coprocessor of the main CPU (called in the CUDA context Host). In a typical GPU application, data parallel-like portions of the main application are carried out on the device by calling a function (called kernel) that is executed by many threads. Host and device have their own separate DRAM memories, and data is usually copied from one DRAM to the other by means of optimized API calls.

CUDA threads can cooperate together by sharing a common fast shared-memory, implemented using fast DRAM memory similar to first level cache, eventually synchronizing in some points of the kernel, within a so-called thread-block, where each thread is identified by its thread ID as illustrated by Figure 5.1. In order to better exploit the GPU, a thread block usually contains from 64 up to 1024 threads, defined as a three-dimensional array of type dim3 (containing three integers defining each dimension). A thread

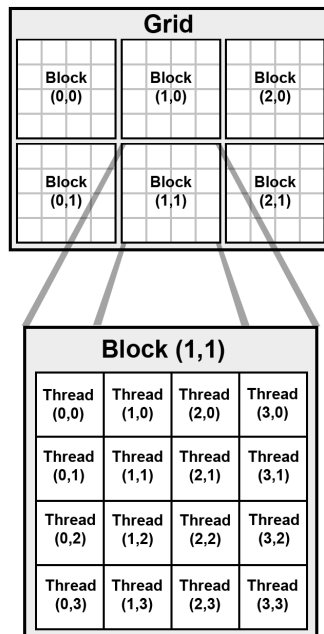


Figure 5.1: Grid of thread blocks

can be referred to within a block by means of the built-in global variable *threadIdx*. While the number of threads within a block is limited, it is possible to launch kernels with a larger total number of threads by batching together blocks of threads by means of a grid of blocks, usually defined as a two-dimensional array, which is also of type `dim3` (with the third component set to 1). In this case, however, thread cooperation is reduced since threads that belong to different blocks do not share the same memory and thus cannot synchronize and communicate with each other. As for threads, a built-in global variable, *blockIdx*, can be used for accessing the block index within the grid. Threads in a block are synchronized by calling the `syncthreads()` function: once all threads have reached this point, execution is resumed normally. As previously reported, one of the fundamental concepts in CUDA is the kernel. This is nothing but a C function, which once invoked is performed in parallel by all threads that the programmer has defined. To define a kernel, the programmer uses the `__global__` qualifier before the definition of the function. This function can be executed only by the device and can be only called by the host. To define the dimension of the grid and blocks on which the kernel will be launched on, the user must specify an expression of the form `<<< Grid_Size, Block_Size >>>`, placed between the kernel name and the argument list, such as in the following simple example:

```

1 | // Kernel definition
2 | __global__ void VectAdd(float* A, float* B, float* C)
3 | {
4 |     int i = threadIdx.x;
5 |     C[i] = A[i] + B[i];
6 | }
7 | int main()
8 | {
9 |     ...
10 |     // Kernel invocation with N threads
11 |     VectAdd<<<1, N>>>(A, B, C);
12 | }

```

The above code first defines a kernel called *VectAdd* which will run on all  $N$  threads, with the aim to compute in the  $i$ -th position of the vector  $C$ , the sum of vectors  $A$  and  $B$ . Assuming that all three vectors have dimension  $N$ , each thread in parallel will be the sum of a position. For example, the thread with  $ID = 2$  will calculate the sum of  $A[2] + B[2]$  and store the result in  $C[2]$ .

### 5.1.2 Memory hierarchy

In CUDA, threads can access different memory locations during execution. Each thread has its own private memory, each block has a (limited) shared memory that is visible to all threads in the same block and finally all threads have access to global memory. In addition to these memory types, two other read-only, fast on-chip memory types can be defined: texture memory and constant memory. In CUDA, memory usage is crucial for the performance. For example, the shared memory is much faster than the global memory and the use of one rather than the other can dramatically increase or decrease performance. By adopting variable type qualifiers, the programmer can define variables that reside in the global memory space of the device (with *\_\_device\_\_*) or variables that reside in the shared memory space (with *\_\_shared\_\_*) that are accessible only from threads within a block. Typical latency for accessing global memory variables is 200-300 clock cycles, compared with only 2-3 clock cycles for shared memory locations. In addition, global memory suffers from coalesced access problems, meaning that access to data should be performed in a particular fashion in order to fetch (or store) the data in the fewest number of transactions [125]. For these reasons, global memory access should be replaced by shared memory access whenever possible. A CUDA C program can allocate global memory of the device in two different ways: through the linear memory or by means of CUDA arrays. CUDA arrays are types of memory optimized for texture management and were not exploited in this work. The more common adopted linear

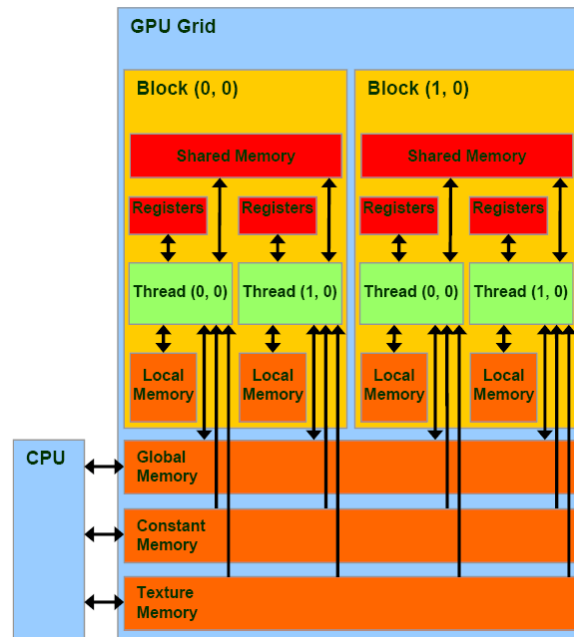


Figure 5.2: Typical memory architecture of a Graphic Processing Unit

memory type is allocated using the *cudaMalloc()* function for allocating and *cudaFree()* function for memory de-allocation. Once allocated, it is possible to transfer data from the Host memory to the global device memory, and vice-versa, by means of a special call to the *cudaMemcpy()* function. Specifically, *cudaMemcpy()* takes as parameters four kinds of memory type transfers: Host to Host, Host to Device, Device to Host and Device to Device. Note that all of the previous functions can only be called on the host. Figure 5.2 illustrates the GPU typical memory architecture. As shown, the fast on-chip shared memory is shared by all threads of a block.

As expected, to improve performance, variable access should be carried out in the shared memory rather than global memory, wherever possible. Unfortunately, as Figure 5.2 shows, each variable or data structure allocated in shared memory must first be initialized in the global memory, and afterwards transferred in the shared one. This means that to copy data in the shared memory, global memory access must be first performed. So, the more this type of data is accessed, the more convenient is to use this type of memory, while for few accesses it is evident that shared memory might be somewhat degrading. As a consequence, a preliminary analysis of data access of the considered algorithm should be performed in order to evaluate the tradeoff and thus, convenience of using shared memory and how. As reported later in this work, the implementation with a hybrid allocation of variables results



in an optimal performance, despite a total shared-memory version as it may be expected.

### 5.1.3 Programming with CUDA C

CUDA C is an extension of C language that permits to write programs for NVIDIA GPUs. With additional constructs and API functions, the programmer is able to allocate and de-allocate memory on the video card (the *device*), transfer the data from the host device (*host*), launch kernels, etc. The CUDA C extension is built on the basis of the CUDA API driver, a low-level library that allows one to perform all the above steps, but which of course is much less user-friendly. On the other hand, the CUDA API driver offers a higher degree of control and is independent of the particular language (e.g., C, Fortran, Java), being written in assembly language. A typical CUDA program can exploit the computing power of both the host (CPU and RAM) and the device (the GPU and memory devices). What follows is a classic pattern of a CUDA application:

1. Allocation and initialization of data structures in RAM memory;
2. Allocation of data structures in the device and transfer of data from RAM to the memory of the device;
3. Definition of the block and thread grids;
4. Performing one or more kernel;
5. Transfer of data from the device memory to Host memory.

In addition, a CUDA application has parts that are normally performed in a serial fashion, and other parts that are performed in parallel.

## 5.2 Implementation of the SCIARA-fv2 model

As previously stated, Cellular Automata models, such as SCIARA, can be straight-forwardly implemented on parallel computers due to their underlying parallel nature. In fact, since Cellular Automata methods require only next neighbor interaction, they are very suitable and can be efficiently implemented even on GPUs. In literature, no examples of Complex Cellular Automata modeling with GPUs can be found. Regarding the considered CA model, three different versions of SCIARA were implemented: a first straight-forward version, called All Global Approach (*AGA*), which uses only global

memory for the entire CA space partitioning, a second called Hybrid Memory Approach (HMA), more performing, which adopts (also) shared memory for CA space substates allocation and, finally, a third strategy called Dynamic Grid Approach (DGA) where the CUDA kernel grid is dynamically adapted to the active cells of the CA space.

### 5.2.1 The Naive implementation: all global memory usage

To develop the first, basic, implementation of the model, the first issue to decide on is what thread mapping should be adopted to better exploit the fine-grain parallelism of the CUDA architecture. For example, one might consider using a thread for each row or each column, as occurs in a typical data-parallel implementation [127]. However, when working in CUDA with arrays, the most widely adopted is the one thread - one cell technique, where each cell of the array is matched with a thread [161]. In addition, in order to achieve maximum performance [125] threads should be executed in groups of multiples of 16 and thus blocks of size  $16 \times 16$  were built in order to obtain square blocks. What follows is an excerpt for defining the grid of blocks that was considered for SCIARA-fv2:

```

1 | #define BLOCK_SIZE 16
2 | ...
3 | ...
4 | int dimX; // CA x dimension
5 | int dimY; // CA y dimension
6 |
7 | dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);
8 | int n_blocks_x = dimX/dimBlock.x;
9 | int n_blocks_y = dimY/dimBlock.y;
10 |
11 | dim3 dimGrid(n_blocks_x, n_blocks_y);
12 | ...
13 | kernel<<<dimGrid, dimBlock>>>(...); // invoke kernel function

```

Once that the grid of blocks (and threads) are defined, the kernels must be managed so that each cell  $(i, j)$  of the cellular automata space is associated to each thread  $(i, j)$ . This is simply done, for each invoked kernel (i.e., *calc\_flows*, *calc\_width*, *calc\_temperature* and *calc\_quote*), by associating each row and column of the CA with the corresponding thread as in this simple scheme:

```

1 | __global__ void kernel(...) {
2 |     int col = blockIdx.x * blockDim.x + threadIdx.x;
3 |     int row = blockIdx.y * blockDim.y + threadIdx.y;
4 |     // decide what memory allocation to use - shared or global
5 |     /** transition function for cell[row][col] **/
6 |     ...
7 | }

```

The basic steps of the CA model CUDA implementation are briefly reported in the following, together with some code excerpts.

- Allocation and initialization the CA space matrix in Host RAM memory:

```

1 | float *h_substates;
2 | size_t size = number_of_substates * NR * NC * sizeof(float);
3 | h_substates = (float*)malloc(size);
4 | updateSubstates(substates);

```

Here, the cellular automata matrixes ( $h\_substates$ ) are represented as a single linearized one-dimensional vector.  $NR$  and  $NC$  represent the dimensions of the cellular space.

- Allocation of arrays in device memory and data transfer from Host to Device:

```

1 | float *d_substates_R; // double matrix CA space...
2 | float *d_substates_W;
3 | cudaMalloc(&d_substates_R, size);
4 | cudaMalloc(&d_substates_W, size);
5 | cudaMemcpy(d_substates_R, h_substates, size, cudaMemcpyHostToDevice);
6 | cudaMemcpy(d_substates_W, h_substates, size, cudaMemcpyHostToDevice);

```

The *cudaMalloc()* function permits to allocate linear memory on the device, while *cudaMemcpy()* transfers data from host memory to the device one, thanks to the *cudaMemcpyHostToDevice* tag.

- Thread grid definition is performed as follows:

```

1 | dim3 dimBlock(BLOCK_SIZER, BLOCK_SIZEEC); //no. of threads per block
2 | // manage also not-divisible dimensions
3 | int n_blocks_x = NC / dimBlocs.x + (NC % dimBlock.x == 0 ? 0:1);
4 | int n_blocks_y = NC / dimBlocs.y + (NC % dimBlock.y == 0 ? 0:1);
5 | dim3 dimGrid(n_blocks_x, n_blocks_y);

```

The block size of threads is determined by the values ( $BLOCK\_SIZER$ ,  $BLOCK\_SIZEEC$ ). The size of the global grid depends, however, from the size of the cellular space and of blocks.

- SCIARA transition function execution is defined as a sequence of kernels, each representing an elementary process (cf. SCIARA model definition presented in chapter 4):

```

1 |   for(int step=0; step< Nstep; step++) {
2 |       // first CA step: add lava at crater cells
3 |       crater <<< 1 , num_craters >>>(d_substates_R,d_substates_W);
4 |       //elem. proc. sigma_1
5 |       calc_flows <<< dimGrid, dimBlock >>> (d_substates_R, d_substates_W);
6 |       //elem. proc. sigma_2
7 |       calc_width <<< dimGrid, dimBlock >>> (d_substates_R, d_substates_W);
8 |       //elem. proc. sigma_3
9 |       calc_temperature <<< dimGrid, dimBlock >>> (d_substates_R, d_substates_W);
10 |      //elem. proc. sigma_4
11 |      calc_quote <<< dimGrid, dimBlock >>> (d_substates_R, d_substates_W);
12 |  }
13 |  // swap matrixes
14 |  copy <<< dimGrid, dimBlock >>> (d_substates_R, d_substates_W);
15 |  }

```

In the time loop, four basic kernels (*calc\_flows*, *calc\_width*, *calc\_temperature* and *calc\_quote*) are launched referred to the elementary processes of SCIARA-fv2,  $\sigma_1$ ,  $\sigma_2$ ,  $\sigma_3$  and  $\sigma_4$ , respectively. The crater kernel refers to the crater cell(s), which is obviously invoked on a smaller grid than the previous ones. The model was implemented by adopting a system of double matrixes for the CA space representation: one (*d\_substates\_R*) for reading cell neighbor substates and a second (*d\_substates\_W*) for writing the new substate value. This choice has proven to be efficient, since it allows separation of the substates reading phase from the update phase after the application of the transition function, thus ensuring data integrity and consistency in a given step of the simulation. After applying the transition function to all the cell space, the main matrix must be updated, replacing values with the corresponding support matrix ones. In order to preserve data consistency, a CA step is simulated by more logical substeps where, after crater cells are updated (by means of the crater kernel), lava outflows are calculated according to the  $\sigma_1$  elementary process. When all outflows are computed, and therefore all outflow substates are consistent, the actual distribution takes place, producing the new value of the quantity of lava in each cell of the CA. Subsequently, each cell reads from a neighbour cell the associated outflow substate corresponding to the quantity of inflowing lava ( $\sigma_2$  elementary process). In this phase, the  $\sigma_3$  and  $\sigma_4$  elementary processes are applied to the new quantity of lava of the cell. Note that this solution respects the CA formal definition where the transition function determines a status change exclusively for the central cell.

- The last step transfers data from device memory to the host memory for displaying the results:

```

1 |   cudaMemcpy(h_substates, d_substates_R, size, cudaMemcpyDeviceToHost);
2 |   printSubstates(h_substates);

```

```

3 |||   cudaFree(d_substates_R);
4 |||   cudaFree(d_substates_W);
5 |||   free(h_substates);

```

### 5.2.2 Shared Memory Usage

Some performance improvements have been reached by implementing a version which adopts a hybrid (both shared and global) memory allocation. As known, access to a location in shared memory of each multiprocessor has a much lower latency than that carried out on the global device memory (cf. Memory hierarchy Section) and can reduce redundant global memory accesses. On the other hand, an access to a shared-memory location necessary needs a first access to global memory to retrieve data to process and a second access to copy the result. For this reason, an accurate analysis was carried out in determining how much memory accesses each thread performs for each CA substate matrix, in order to evaluate the convenience of using shared memory, as described previously. This investigation gave rise to a hybrid memory access pattern, where shared memory allocation was adopted in those kernels accessing CA substate matrixes at least a threshold number of times (cf. Table 5.1).

On this basis, the data matrixes corresponding to *altitude* and *thickness* CA substates in the *calc\_flows*() kernel ( $\sigma 1$ ) and the matrixes corresponding to *temperature* and *flows* CA substates for the *calc\_temperature*() kernel ( $\sigma 3$ ) were allocated in the shared memory, while all of the other matrixes for the remaining elementary processes were allocated in the device global memory. For illustrative purposes, Figure 5.3 shows how shared memory is used in the context of this implementation.

In order to better understand how the shared memory was used, the following is an excerpt of code of the *calc\_flows* kernel for the calculation of the outgoing flows from a generic (central) cell towards neighbor cells, which adopts shared memory. As shown in Table 5.1, shared memory is adopted

Substate/Kernel	<i>calc_flows</i>	<i>calc_width</i>	<i>calc_temperature</i>	<i>calc_quote</i>
altitude	5	0	0	1
thickness	5	1	1	1
temperature	1	0	5	1
flows	1	2	3	0

Table 5.1: Number of memory accesses performed by each SCIARA kernel to substate matrixes

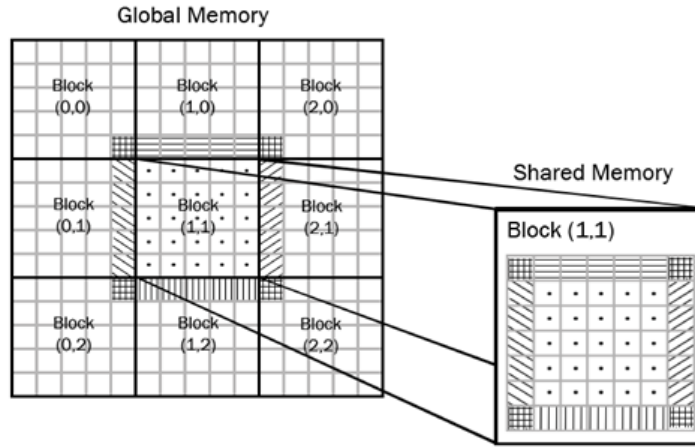


Figure 5.3: Memory mapping of the CA space allocated in global memory with a portion of shared memory. Shaded areas represent portions of neighbouring block areas which need to be swapped at each CA to ensure data consistency.

for the  $Q_z$  and  $Q_t$  substates.

```

1 |   __global__ void calc_flows (float* d_substates_R, float* d_substates_W){
2 |
3 |       int col = blockIdx.x * blockDim.x + threadIdx.x; // global column index
4 |       int row = blockIdx.y * blockDim.y + threadIdx.y; // global row index
5 |       // shared memory in device for Qz and Qt substates
6 |       __shared__ float shared_quote[BLOCK_SIZER+2][BLOCK_SIZE+2];
7 |       __shared__ float shared_width[BLOCK_SIZER+2][BLOCK_SIZE+2];
8 |       int rowShared = row % BLOCK_SIZER + 1; // local row index (shared)
9 |       int colShared = row % BLOCK_SIZE + 1; // local column index (shared)
10 |      // global values copied in shared memory except ghost cells
11 |      shared_quote[rowShared][colShared] = d_substates_R[row*NC + col*NR*NC+
12 |          QUOTE];
13 |      shared_width[rowShared][colShared] = d_substates_R[row*NC + col*NR*NC*
14 |          WIDTH];
15 |      // ghost cells global values copied in shared memory
16 |      // last row of shared block
17 |      if (rowsShared-1 == BLOCK_SIZER-1 && row < NR){
18 |          shared_quote[rowShared+1][colShared] = d_substates_R[(row+1)
19 |              * NC + col + NR*NC*QUOTE];
20 |          shared_width[rowShared+1][colShared] = d_substates_R[(row+1)
21 |              * NC + col + NR*NC*WIDTH];
22 |      }
23 |      // first row of shared block
24 |      if(rowsShared -1 == 0 && row>0) {
25 |          shared_quote[rowsShared-1][colShared] = d_substates_R[(row-1)*NC+
26 |              col
27 |              + NR*NC*QUOTE];
28 |          shared_width[rowsShared-1][colShared] = d_substates_R[(row-1)*NC+
29 |              col
30 |              + NR*NC*WIDTH];
31 |      }
32 |  }

```

```

28 | ...
29 | ...
30 | // synchronize all threads before proceeding ...
31 | _syncthreads();
32 | //
33 | Compute flows according to Minimization algorithm of differences ...
34 | ...
35 | Copy results in Global Memory
36 | }

```

Each thread block is responsible for allocating a shared memory of size  $(BLOCKSIZE_R + 2)(BLOCKSIZE_C + 2)$ . Every thread has the task to copy the cell value to be managed by it to the shared global memory. Note that the copy of ghost cells is performed by threads that manage external cells to each block of the grid (cf. Figure 5.3).

### 5.2.3 Dynamic extension of the kernels grid

A critical aspect of CA implementations that can improve performance, which is also related to CA sequential versions, is that the application of the transition function can be restricted to the only active cells where computation is actually taking place. When considering a phenomenon topologically connected (i.e., a simulation starts from few active cells and evolves by activating neighbour cells), the CA space can be coned within a rectangular bounding box (*RBB*). This optimization drastically reduces execution times, since the sub-rectangle is usually quite smaller than the original CA space. For this reason a GPU optimized CA version called, *DGA*, has been tested that takes into account, at each CA step, the RBB that includes all active cells (i.e., cells containing lava) of the automaton. However, while in the same sequential version of the algorithm the CA space matrix is simply scanned by considering the RBB boundaries instead of the whole CA, the *DGA* must consider the rectangular grid bounding box (*RGBB*) containing active cells, which in general includes the traditional RBB of active cells (cf. Figure 5.5). While still considering a one-thread-one-cell approach, the grid of blocks on which the RBB is activated grows dynamically as the simulation expands, giving rise defacto to a one-thread-one-active-cell strategy. At the subsequent CA step, the grid of threads readapts itself on the basis of the fixed dimensions of each block (e.g., 16 x 16), activating an adjacent block as soon as it's interested by an active cell (cf. Figure 5.4).

As expected, since the number of overall number of launched kernels is reduced, the computational performance of the algorithm improves significantly. The fundamental steps of the CUDA SCIARA-fv2 execution phase is briefly reported hereafter. After the host memory CA space allocation, data

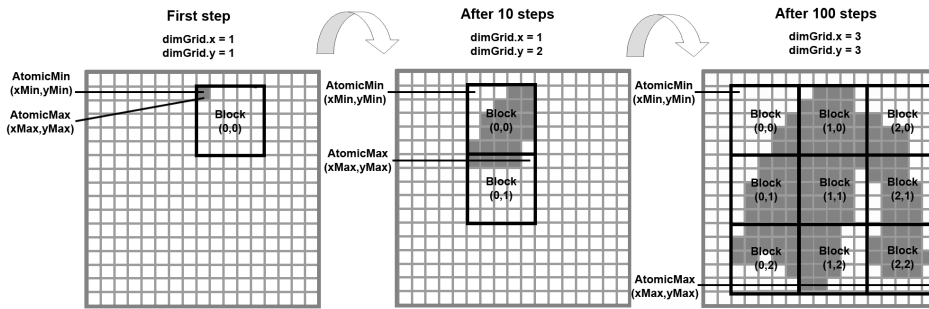


Figure 5.4: An example of dynamic RGBB (rectangular grid bounding box) expansion, referred to a 5 x 5 block size grid. As lava expands, blocks interested by active cells are activated.

structures transfer in the device and definition of the initial block and thread grids, the main CA loop is launched. Firstly, the actual RBB on which the transition function kernels will be launched on, specified by an array of 4 integers (RBB in the following code) representing the 4 coordinates delimiting it, is computed in device memory and transferred in host memory. The new dynamic thread grid (*RGBB*), which includes the standard RBB, is determined as follows:

```

1 | void determineRGBB(dim3 *dynamicGrid, int *RBB)
2 | {
3 |     //Dynamic Grid calculation...
4 |     h_ymax = RBB[3]; h_ymin = RBB[1];
5 |     h_xmax = RBB[2]; h_xmin = RBB[0];
6 |     MRcolsNumber = h_ymax - h_ymin + 1; // RBB columns
7 |     MRrowsNumber = h_xmax - h_xmin + 1; // RBB rows
8 |     // new RGBB dimensions (i.e., the dynamicGrid)
9 |     dynamicGrid.x = (MRcolsNumber + dimBlock.x - 1)/dimBlock.x;
10 |    dynamicGrid.y = (MRrowsNumber + dimBlock.y - 1)/dimBlock.y;
11 | }

```

Subsequently, kernels are launched by considering the newly computed RGBB. The core of the CUDA SCIARA-fv2 algorithm is:

```

1 | // CA loop
2 | for(int currentStep=0; currentStep < NSTEPS; currentStep++){
3 |     // calculate new kernel grid based on RGBB (host memory code)
4 |     updateGrid(dynamicGrid, RBB);
5 |     // add lava at craters
6 |     crater <<< 1, num_craters >>>(updatedCA, currCA);
7 |     // launch kernels on the RGBB
8 |     calc_flows <<< dynamicGrid, dimBlock >>>(updatedCA, currCA);
9 |     calc_width <<< dynamicGrid, dimBlock >>>(updatedCA, currCA);
10 |    calc_temp <<< dynamicGrid, dimBlock >>>(updatedCA, currCA);
11 |    calc_solid <<< dynamicGrid, dimBlock >>>(updatedCA, currCA);
12 |    // determine the RBB atomically
13 |    determineRBB <<< dynamicGrid, dimBlock >>>(currCA);
14 |    // swap matrixes (in device memory)

```



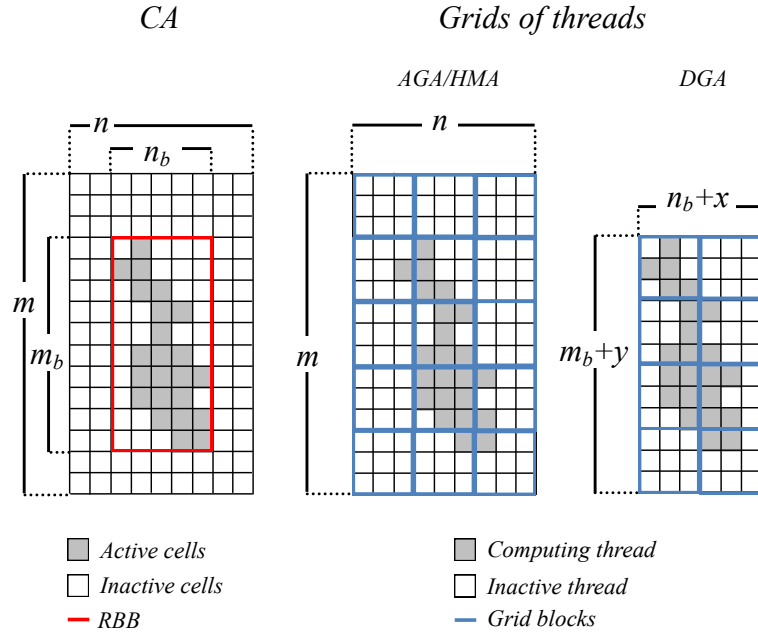


Figure 5.5: Two different ways of mapping the CA transition function into a CUDA grid of threads. In the AGA and HMA cases, a CUDA kernel iterating over the whole automaton. In the DGA approach, kernels operate on each cell belonging to the current RBB.

```

15 |         copyMatrix <<< dynamicGrid, dimBlock >>>(updatedCA,currCA);
16 |         //copy RGBB array from device to host
17 |         cudaMemcpyFromSymbol(RGBB, d_RGBB, size_RGBB,cudaMemcpyDeviceToHost);
18 |     } // CA loop end
19 |     // copy data to Host
20 |     cudaMemcpy(A, currCA, size, cudaMemcpyDeviceToHost);

```

In the time loop, after the determination of the new kernel grid dimensions, four basic kernels, *calc\_flows*, *calc\_width*, *calc\_temperature* and *calc\_solid* are launched which regard the  $\sigma_1, \sigma_2, \sigma_3$  and  $\sigma_4$  SCIARA-fv2 elementary processes, respectively. After applying the transition function to the cell space confined by RBB, the main matrix must be updated, replacing values with the corresponding support matrix ones (i.e., swap matrixes phase). After crater cells are updated (i.e. by the crater kernel), lava outows are calculated according to the  $\sigma_1$  elementary process. When all outows are computed, and thus all outow substates are consistent, the actual distribution takes place, producing the new value of the quantity of lava in each cell of the CA. Subsequently, each cell reads from a neighbour cell the associated outflow substate corresponding to the quantity of inlowing lava ( $\sigma_2$

elementary process). In this phase, the  $\sigma_3$  and  $\sigma_4$  elementary process are applied to the new quantity of lava of the cell in order to calculate the new cell lava temperature and eventual lava solidication, respectively. At the end of each CA step, a specic kernel, *determineRBB*, updates concurrently (and atomically) the values of the smallest RBB containing active cells, which are stored in the RGBB array and copied back to the host memory, by the *cudaMemcpyFromSymbol* function. At the end of the CA loop, data is copied to the Host memory in order to be processed.

### 5.2.4 Tests and performance results

Three parallellisation strategies and different graphic hardware were adopted in all experiments that are reported in the following. In particular, four CUDA devices were used: a nVidia Tesla C2075 and three nVidia Geforce graphic cards, namely the GTX480, GTX 580 and the GTX 680. The latter belongs to the new nVidia’s Kepler GPU architecture while the former are endowed with the older Fermi GPUs. Some relevant characteristics of the used devices are reported in Table 5.2. Also, in order to quantify the achieved parallel speedup, sequential version of the algorithms parallelised for the GPU were run on a workstation equipped with a 2-Quadcore Intel Xeon E5472 (3.00 GHz) CPU. Specifically, the C-language and same optimization approaches used in the corresponding GPGPU versions were adopted in the sequential implementations in all experiments.

Many tests have been performed considering the different implemented strategies and for the verification of the correctness of the results for evaluating the precision of results using GPUs. The simulation of a well known and documented real lava flow event, The Mt Etna Nicolosi Event occurred in July 2001, was considered to test the efficiency of the GPUs algorithms. Table 5.3 reports the results of tests carried out for this experiment where the CA space considered for the real event is a  $819 \times 382$  two-dimensional grid representing the topography of the interested area, considering one crater for

	Tesla C2075	GTX470	GTX580	GTX 680
SM count	14	14	16	8
CUDA cores	448	448	512	1536
Clock rate [MHz]	1150	1215	1544	1006
Bandwidth [GB/s]	144.0	133.9	192.4	102.3
GFLOPs	1030.4	1088.6	1581.1	3090.4

Table 5.2: Major characteristis of adopted GPGPU hardware for all carried out experiments

	16 x 16			32 x 16		
	<i>AGA</i>	<i>HMA</i>	<i>DGA</i>	<i>AGA</i>	<i>HMA</i>	<i>DGA</i>
<b>CPU</b>	1150.00	1150.00	544.60	1150.00	1150.00	544.60
<b>Tesla C2075</b>	45.21	38.92	16.08	38.75	34.75	16.59
<b>GTX 470</b>	37.45	31.50	13.58	33.95	28.70	14.68
<b>GTX 580</b>	24.94	22.16	9.45	23.35	19.30	10.26
<b>GTX 680</b>	25.50	23.21	10.80	22.44	20.16	11.63

Table 5.3: Elapsed times (in seconds) for the computation of 2001 lava flow event by adopting different block dimensions

lava flow emission and 35000 simulating steps.

As a confirm of the vailidity of the assumption regarding the hibrid memory usage approach, results have shown an improving in performance up to 19% in the hybrid memory version, with respect to the implementation that considers only global memory for CA space. This confirms the importance of data access analysis when implementing GPGPU applications. As expected, the best results have regarded versions which adopt the dynamic block optimization. Results show how the use the dynamic block (DGA) can improve performances up to 181% (considering the Tesla C2075 device and blocks of dimension  $16 \times 16$ ), with respect to the GPU implementation that considers the entire CA space allocation(AGA). Nevertheless, all GPU timings outperform, as expected, the corresponding carried out CPU versions.

Timings reported in all experiments for the considered GPU devices (see also Figure 5.6) indicate their full suitability for parallelizing Cellular Automata models. Performance results show the impressive computational power of the considered GPUs in terms of execution time reduction, significantly outperforming CPUs up to 58x by adopting the DGA in conjunction with the GTX580 device.

#### 5.2.4.1 Numerical verification of experiments

As already mentioned, GPUs are specialized for single-precision calculations even though, since NVIDIA Tesla products, double precision floating point operations are fully supported at the expense of an obvious decrease in performance. All versions up to here considered were implemented in single-precision, in order to fully exploit the graphic hardware in their original nature. To test if single-precision data can be considered sufficient for SCIARA simulations, other tests were carried out on the 2001 lava flow event (10000 CA steps) and results produced by the GPU version compared with those of the CPU (sequential) version with single precision data (i.e., C float type

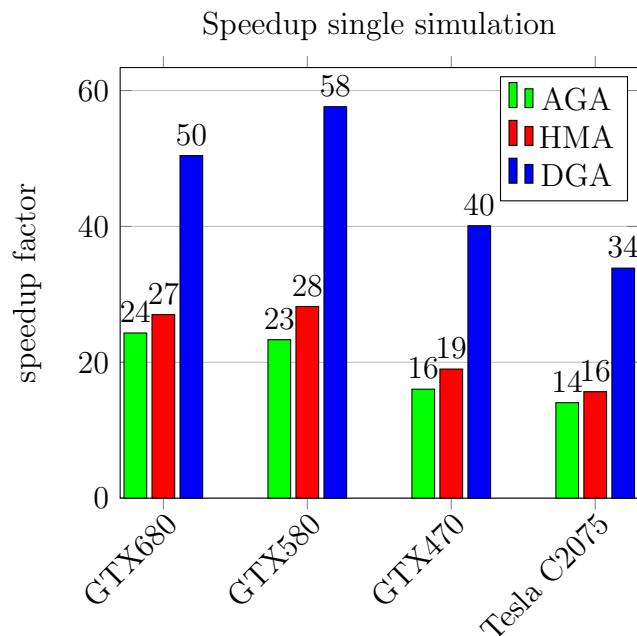


Figure 5.6: Speedup results on different strategy approaches and devices.

variables), and those produced still by the same GPU version against a double precision CPU implementation (i.e., C double type variables). In each case, comparison results were satisfactory, since the areal extensions of simulations resulted the same, except for few errors of approximation in a limited number of cells. In particular, comparing the GPU version with the CPU single-precision version, approximation differences at the third significant digit were only present in 4% of cells, while differences were even less for remaining cells. Differences were even minor compared to the previous case by considering the single precision GPU version and a CPU version which adopts double-precision variables.

### 5.3 Efficient GPGPU application for large number of concurrent lava flow simulations

Results, discussed in the previous sections, regarding the CUDA parallel implementation of the SCIARA model were considered positive and extremely encouraging. However, the most important applications of the SCIARA model require the concurrent simulation of more than a single lava event episode. Implementation of Volcanic Hazard Maps [140], Parallel genetic algorithms for optimizing set of parameters for CA models [46, 45] or evo-

lutionary application for risk mitigation [112, 66] are just a few examples of SCIARA applications that can exploit a CUDA Multisimulator.

The use of thematic maps of volcanic hazard is of fundamental relevance to support policy managers and administrators in taking the most correct land use planning and proper actions that are required during an emergency phase. In particular, hazard maps are a key tool for emergency management, describing the threat that can be expected at a certain location for future eruptions. The methodology for defining high detailed hazard maps is based on the application of the SCIARA lava flows computational model for simulating an elevated number of events on topographic data.

By considering the application of parallel genetic algorithms to the SCIARA-fv2 model, to evaluate a GA individual, an entire CA simulation has to be performed and, depending on the adopted computer framework, such an operation may require several hours (or months). Because of the required high number of explicit lava flow simulations, these applications often results in a highly intensive computational process. For these reasons, in order to develop and test CUDA strategy to run large number of simultaneous simulations, GPGPU is applied for mitigation and assessment purposes, to the process of lava flow simulation model. Different GPGPU strategies, that give rise to different overall execution times, are presented. In the following sections, two different GPGPU approach implementation and performance analysis referred to simulations are reported, while conclusions and outlooks are shown at the end of the chapter.

## 5.4 The methodology for defining hazard maps

Volcanic hazard maps are fundamental for determining locations that are subject to eruptions and their related risk. Typically, a volcanic hazard map divides the volcanic area into a certain number of zones that are differently classified on the basis of the probability of being interested by a specific volcanic event in future. Mapping both the physical threat and the exposure and vulnerability of people and material properties to volcanic hazards can help local authorities to guide decisions about where to locate critical infrastructures (e.g. hospitals, power plants, railroads, etc) and human settlements and to devise mitigation measures that might be appropriate. This could be useful for avoiding the development of inhabited areas in high risk areas, thus controlling land use planning decisions.

A lava flow simulation model can represent an effective instrument for analyzing volcanic risk in an area by simulating possible single episodes with

different characteristics (e.g. vent locations, effusion rates) [33]. However, the methodology for defining high detailed hazard maps here presented is based on the application of the SCIARA lava flows computational model for simulating an elevated number of events on present topographic data. In particular, the methodology requires the analysis of the past behavior of the volcano, for the purpose of classifying the events that historically interested the region. In such a way, a meaningful database of plausible simulated lava flows can be obtained, by characterizing the study area both in terms of areal coverage, and lava flows typologies. Data is subsequently processed by considering a proper criterion of evaluation. A first solution could consist in considering lava flows overlapping, assigning a greater hazard to those areas interested by a higher number of simulations. However, a similar choice could be misleading since, depending on the events particular volcanological characteristics (e.g., location of the main crater, duration and amount of emitted lava, or effusion rate trend), different events can occur with different probabilities, which should be taken into account in evaluating the actual contribution of performed simulations with respect to the definition of the overall hazard of the study area. In most cases, such probabilities can be properly inferred from the statistical analysis of past eruptions, allowing for the definition of a more refined evaluation criterion. Accordingly, in spite of a simple hitting frequency, a measure of lava invasion hazard can be obtained in probabilistic terms.

The methodology, well described in [34] and [140] consists in an elaborate approach in the numerical simulation of Etnean lava flows, based on the results of an elevated number of simulations of flows erupted from a grid of hypothetical vents covering the entire volcano. Firstly, based on documented past behavior of the volcano, the probability of new vents forming was determined, resulting in a characterization (a probability density function map - pdf) of the study region into areas, that represent different probabilities of new vents opening. Then, flank eruptions of Etna since 1600 AD were classified according to duration and lava volume and a representative effusion rate trend considered to characterize lava temporal distribution for the considered representative eruptions, reflecting the effusive mean behavior of Etnean lava flows. An overall probability of occurrence for each lava event,  $p_e$ , was thus defined, by considering the product of the individual probabilities of its main parameters:

$$p_e = p_s \cdot p_c \cdot p_t \quad (5.1)$$

where  $p_s$  denotes the probability of eruption from a given location (i.e., based on the pdf map),  $p_c$  the probability related to the events membership class

(i.e., emitted lava and duration), and  $p_t$  the probability related to its effusion rate trend. Once representative lava flows were devised as above, a set of simulations were planned to be executed in the study area by means of the SCIARA lava flows simulation model. For instance, in [140] a grid composed by 4290 craters, equally spaced by 500m, was defined as a covering for Mt Etna, from where the simulations have been carried out. This choice allowed to both adequately and uniformly cover the study area, besides considering a relatively small number of craters. Specifically, a subset of event classes which define 6 different effusion rates probabilities, derived from historical events considered in [45] were taken into account for each crater, thus resulting in a total of 25740 different simulations to be carried out. Owing to the elevated number of SCIARA simulations to be executed, Parallel Computing was necessary and each scenario simulated for each of the vents of the grid. Lava flow hazard was then punctually (i.e. for each cell) evaluated by considering the contributions of all the simulations which affected a generic cell in terms of their probability of occurrence.

## 5.5 GPGPU-Based Lava Flow Risk Assessment

A natural approach for dealing with the high computational effort related to the construction of hazard maps or multiple simultaneous running of lava flow simulations, due to the elevated number of simulations to be executed, is the use of parallel computing techniques. As stated previously, GPGPU has recently attracted researchers due to its efficacy exploiting the computational power provided by graphic cards through a fine grained data parallel approach, especially when the same computation can be independently carried out on different elements of a dataset, such as in the case of CA models. CA models can be straightforwardly implemented on parallel computers due to their underlying parallel nature. In fact, since CA methods require only next neighbor interaction, they are very suitable and can be efficiently implemented even on GPUs. This results in: (i) computing the next state of all the cells in parallel; (ii) accessing only the current neighbour states during each cells update, thereby giving the chance to increase the efficiency of the memory accesses. As in the sequential case, the typical CA parallel implementation involves two memory regions, which will be called  $CA_{cur}$  and  $CA_{next}$ , representing the current and next states for the cell respectively. For each CA step, the neighbouring values from  $CA_{cur}$  are read by the local transition function, which performs its computation and writes the new state

value into the appropriate element of  $CA_{next}$ .

In several GPGPU parallel implementations of the CA based procedure illustrated above, most of the CA data was stored in the GPU global memory (e.g., [130, 137, 177, 42, 55]). This involves: (i) initialising the current state through a CPU-GPU memory copy operation (i.e. from host to device global memory) before the beginning of the simulation and (ii) retrieving the final state of the CA at the end of the simulation through a GPU-CPU copy operation (i.e. from device global memory to host memory). At the end of each CA step a device-to-device memory copy operation is used to re-initialise the  $CA_{cur}$  values with the  $CA_{next}$  values. In order to speed up the access to memory, the CA data in device global memory should be organised in a way to allow coalescing access. To this purpose, a best practice recommended by nVidia is to use of structures of arrays rather than arrays of structures in organising the memory storage of cells properties [125]. Even if further memory optimizations are possible, a complete contiguous structure access results difficult due to the presence of cell neighbours. As a compromise, an array with the size corresponding to the total number of cells was allocated in the CPU memory for each of the CA substates and, for every simultaneous run, allocated in the GPU memory, together with some additional auxiliary arrays (e.g. for storing the neighbourhood structure and the model parameters). Since all of the neighbouring cells have to access each others substates, a CA step involves repeated accesses to the same portions of the global memory. While the use of faster (but limited) shared memory as in the single simulation strategy could be exploited to efficiently cache the CA substates needed to all the threads of the same block (e.g., [67, 42]), the recent automatic exploitation of caching levels (L1 and L2) for global memory access in recent nVidia hardware was here adopted in spite of shared memory implementation (e.g., [15]).

A key step in the parallelisation of a sequential code for the GPU architecture according to the CUDA approach, consists of identifying all the sets of instructions (e.g., the CA transition function) that can be executed independently of each other on different elements of a dataset (e.g., on the different cells of the CA). As mentioned before, such sequences of instructions are grouped in CUDA kernels, each transparently executed in parallel by the GPU threads. However, a critical aspect of the GPGPU parallelisation object of this study is related to the fact that only the transition function of the CA active cells (i.e., cells containing lava) do actual computation. Hence, launching one thread for each of the CA cells would result in a certain amount of dissipation of the GPU computational power. For this reason, besides the straightforward parallel algorithm in which the above mentioned kernels are mapped to the whole CA, a second, more optimized implementation, has



been developed. In the following, such approaches are described in detail. Using a significant test case (for benchmark purposes) and four recent GPU devices, their computational performances are empirically investigated.

### 5.5.1 A case study and performed simulations

To evaluate the implementation strategies, a landscape benchmark case study was initially considered, modeled through a Digital Elevation Model composed of 200 x 318 square cells with a side of 10 m. Over the area, a regular grid of 38 x 60 craters was superimposed, leading to a total of 2280 simulations to be performed. From each crater of the grid, a typical lava flow with a  $20m^3/s$  emission rate was considered, for a duration of 5000 CA steps. Two parallelisation strategies and different graphic hardware were adopted in all experiments that are reported in the following. In particular, as for the single simulation GPU version, four CUDA devices were used in the experiments: a nVidia Tesla C2075 and three nVidia Geforce graphic cards, namely the GTX480, GTX 580 and the GTX 680. Also, in order to quantify the achieved parallel speedup, sequential versions of the algorithms parallelised for the GPU were run on a workstation equipped with a 2-Quadcore Intel Xeon E5472 (3.00 GHz) CPU. Specifically, the C-language and same optimization approaches used in the corresponding GPGPU versions were adopted in the sequential implementations in all experiments.

### 5.5.2 A Naive implementation: the Whole Cellular Space Implementation

A first, straightforward parallel implementation, labeled as WCSI (Whole Cellular Space Implementation) was considered where the CUDA kernels operate on the whole CA. Since during a lava flow simulation only the transition function of the currently active cells do significant computation, simulating only one simulation at a time would imply a high percentage of uselessly scheduled threads. In addition, given the small size of most simulations (in average, 20% of cells of the entire CA are active during a single simulation), the number of active threads would be too low to allow the GPU to effectively activate the latency-hiding mechanism. For these reasons, in the WCSI approach more than a single lava episode are simultaneously executed. This means that the main CUDA kernel is executed over a number of simulations which are propagated at the same CA step. In particular, each performed simulation is mapped on a different value of  $z$  and on a grid of threads composed of  $16 \times 16$  blocks. That is, the grid of threads used for the CA transition function is three-dimensional, with the base representing the considered CA

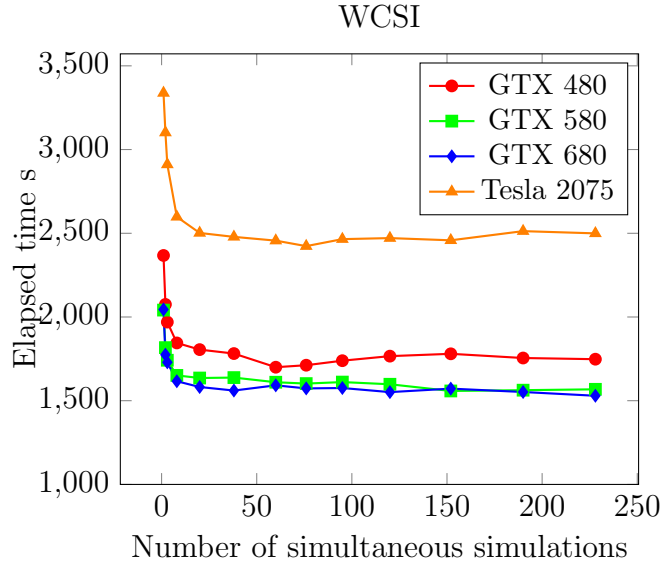


Figure 5.7: Elapsed time as a function of simultaneous lava events using the WCSI approach on different considered GPGPU hardware.

space and the vertical dimension corresponding to the simulations. According to the general lava hazard GPGPU computation Algorithm 2, in the hazard map computation phase, clusters ( $n_s$ ) of simulations are simulated over the entire area under study up to cover the total number of required simulations ( $n_t$ ). Before starting the hazard map construction (from Line 4), a pre-processing sequential phase takes place for compute the thread kernel grid on the basis of the dimensions of blocks and of the CA space.

When the computation begins, *configureCA* resets the  $n_s$  simulations at the beginning of each cluster computation phase and time duration for each simulation initialized. After, the function *transitionFunctionInParallel* (line 10) activates the kernel implementing the simulation propagation mechanism on a 3D grid as described above. Such kernel iterates over the simulations that are the simultaneously propagated. Subsequently, a device-to-device memory copy operation is used to re-initialise the  $CA_{cur}$  values of the sub-states with the values of  $CA_{next}$  (line 11). At the end of the computation, the current CA time step  $CAstep$  is updated.

For a fair comparison, the sequential version of the same algorithm was used and clearly, only one simulation at a time was propagated since the advantages of simulating multiple simulations are not significant. Here, the elapsed time achieved by the CPU was 61312s. Using the adopted GPU devices, the algorithm was solved with the WCSI approach and a variable

---

**Algorithm 2:** The GPGPU procedure for building lava hazard maps.

---

```

1  $n_c \leftarrow 0$ ;
2  $n_t \leftarrow 0$ ;
3  $n_s \leftarrow x$ ;
4 while  $n_c < n_t$  do
5   configureCA( $CA_{cur}, n_s$ );
6    $CA_{time} \leftarrow 0$ ;
7   while notTerminated( $CA_{time}$ ) do
8     grid  $\leftarrow$  updateDynamicGrid( $\Delta CRBB$ );
9     vent  $\lll \lll 1, n_s \ggg \ggg$ ( $CA_{cur}, CA_{next}, emittedLava$ );
10    transitionFunctionInParallel  $\lll \lll grid, n_s \ggg \ggg$ ( $CA_{cur}, CA_{next}$ );
11    updateNextFromCurrent  $\lll \lll grid, n_s \ggg \ggg$ ( $CA_{cur}, CA_{next}$ );
12    updateDeltaCRBB  $\lll \lll grid, n_s \ggg \ggg$ ( $CA_{cur}$ );
13     $CA_{time} \leftarrow CA_{time} + 1$ ;
14   $n_c \leftarrow n_c + n_s$ ;

```

---

number of simultaneous lava simulations. According to the results shown in Figure 5.7, the GTX 680 achieved the lowest elapsed time of 1529s, concurrently simulating 228 lava events. The gain provided by the parallelisation in terms of computing time was significant and corresponded to a parallel speedup of 40 over the used CPU. As can be seen, the simultaneous execution of more lava simulations was beneficial since it allowed a computing time decrement that ranged from about 27% for the GTX 580, 31% for the GTX 680, 36% for the Tesla 2075 and nearly 40% for the GTX 480. However, even if computing times decrease as the number of concurrent simulations increase, for all the GPUs simulating more than about 25 simulations simultaneously did not lead to significant advantages. In fact, once the number of computationally relevant threads is sufficient for enabling latency mitigation, a further increment in the number of simulations is unnecessary. Interestingly, even if characterized by substantial different peak performance in single-precision floating point operations, the best elapsed times of GTX 580 and GTX 680 were quite similar. This is due to the fact that the used kernels are definitely memory bound and that the two GPUs have the same memory bandwidth. In fact, especially considering that most substates are pre-computed once in the pre-processing phase, the CA transition function is characterized by a relatively low computational load. In addition, to apply the lava flow spread mechanism each cell must access its own substates and most of the substates of its neighbouring cells.

### 5.5.3 The Rectangular Bounding Box Strategy: the Dynamic Grid Implementation

The main weakness of the above WCSI approach is due to the difficulty of having a high percentage of computationally active threads in the CUDA grid. For this reasons, an alternative approach was developed in which the grid of threads is dynamically computed during the simulation in order to keep low the number of computationally irrelevant threads. In such an approach, labelled as DGI (Dynamic Grid Implementation), a number of lava flow simulations are simultaneously executed as in the WCSI procedure. In addition, at each CA step the procedure involves the computation of the smallest common rectangular bounding box (CRBB) that includes any active cells in every concurrent simulation. As shown in Figure 5.9, all the kernels required by the CA step in Algorithm 2 are then mapped on such CRBB, reducing the number of useless threads and improving significantly the computational performance of the algorithm. Since the present study exploits relatively recent GPU devices, the CRBB computation was efficiently carried out using the *atomicMin* and *atomicMax* CUDA primitives in the same kernel implementing the transition function.

A further enhancement introduced in the DGI approach concerns the device-to-device memory copy operation used to re-initialise  $CA_{cur}$  with  $CA_{next}$  (line 11 of Algorithm 2). In particular, instead of involving the entire CA, a specific kernel was developed to re-initialise the substates of each concurrent simulation only for the cells lying in the CRBB.

In order to perform a fair comparison, an analogous strategy based on the bounding box has been developed for the sequential version of the DGI algorithm. Using the reference CPU, such sequential procedure required 47521 s for the case study adopted. Figure 5.8 shows the corresponding times taken by the parallel DGI approach as a function of the number of concurrent simulations. As seen, the GTX 680 achieved the lowest elapsed time of 617.348 s, corresponding to a parallel speedup of 77. As expected, the simultaneous execution of many simulations was much more beneficial than in the WCSI case, for all considered hardware. For example, with the GTX 680 card the best elapsed time is less than a half of that obtained by simulating only one simulation at a time. The reason is that, having available a relatively limited number of concurrent simulations, the percentage of active cells is higher in a small CRBB than in the entire CA. However, at the contrary of the WCSI approach, there is a critical value of the number of simultaneous lava episodes to be executed (i.e., 20 for all adopted hardware), after which the overall computing time increases. At the basis of this behavior is the CRBB mechanism, where a certain number of inactive cells are present for

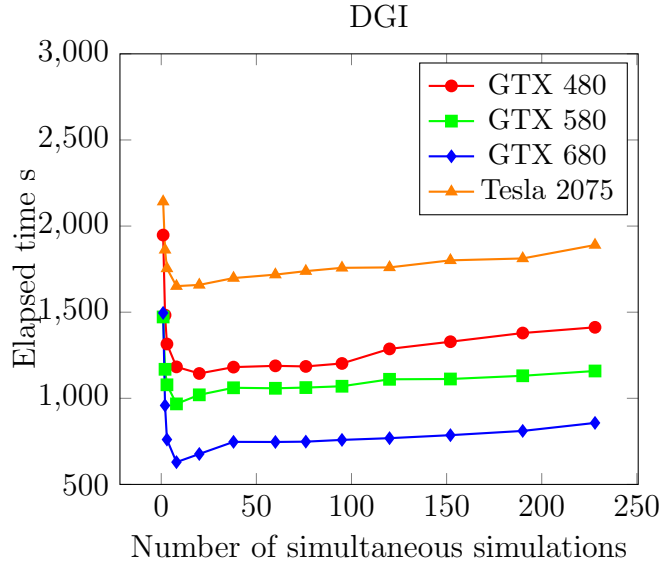


Figure 5.8: Elapsed time as a function of simultaneous lava events using the DGI approach on different considered GPGPU hardware.

each simulation. As a consequence, when the number of computationally active threads is high enough for the latency hiding mechanism, any further increment of simultaneous lava events brings to the unnecessary scheduling of inactive threads, producing an overall decline in computational performance. Furthermore, atomic operations necessary for calculating CRBB that depends on the number of simultaneous simulations are factors that may affect the execution time of the algorithm.

The techniques applied for the DGI approach for building lava flows hazard maps were applied to the area of Mt Etna volcano (Italy) [140]. A subset of the original grid of vents (1006 craters in total) was considered and only one typology of lava effusion rate of  $32 \times 10^6 m^3$  taken into account, emitted in 15 days, corresponding to the most common type of eruption recorded in the past 400 years of activity of the volcano (see [34] for further details). This gave rise to 1006 simulations in spite of the over 25000 carried out in [140]. Figure 5.10 shows the map obtained by the application of the DGI procedure.

In particular the GTX 680 hardware was adopted, by running 20 concurrent simulations (i.e., the  $n_s$  parameter of Algorithm 2). The overall computing time was 805 s, while the map obtained by the same configuration (i.e., same number of grids and simulations), executed as in a previous work [140] with standard MPI Message Passing techniques on a 80-core cluster,

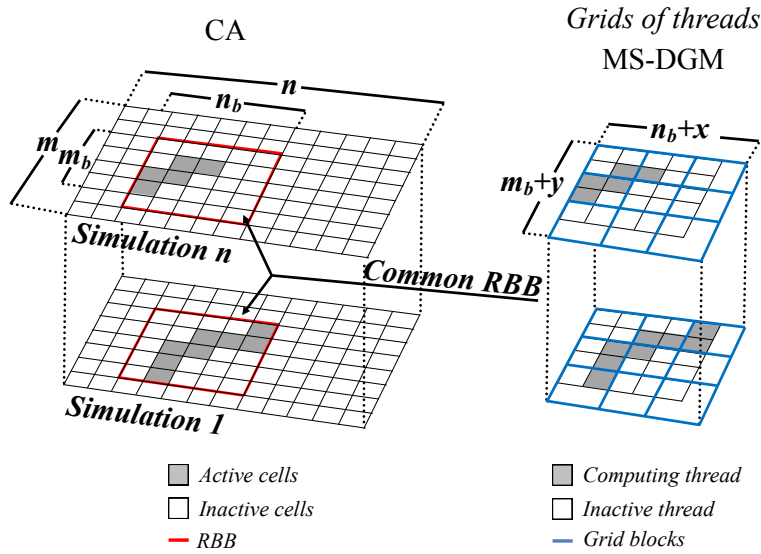


Figure 5.9: Mapping of the CA transition function into a CUDA grid of threads (right) in case of the simultaneous lava flows (left). As it can be seen, computation takes place only for threads that are within the CRBB of active cells of the two lava events.

was processed after 23432 s.

## 5.6 Conclusions

Results reported in this chapter have indeed confirmed the full suitability of GPGPU architectures for implementing CA models. Regarding the implementation of the single lava episode, as seen, the CUDA technology, in combination with an efficient memory management, can produce very efficient version of the SCIARA-fv2 CA lava flow model. The different strategies have demonstrated the advantage of this approach which can have important applications also in the field of risk/hazard mitigation.

In the second part of this chapter, starting from the problem of lava hazard map computation, this study discussed some approaches for executing a large number of concurrent lava simulations using GPGPU. One of the advantages of the presented solutions lies in enabling the building of hazard maps for large areas, which otherwise may not be possible by adopting traditional sequential computation. The parallel speedups attained through the proposed approaches and by considering GPGPU hardware, were indeed significant.

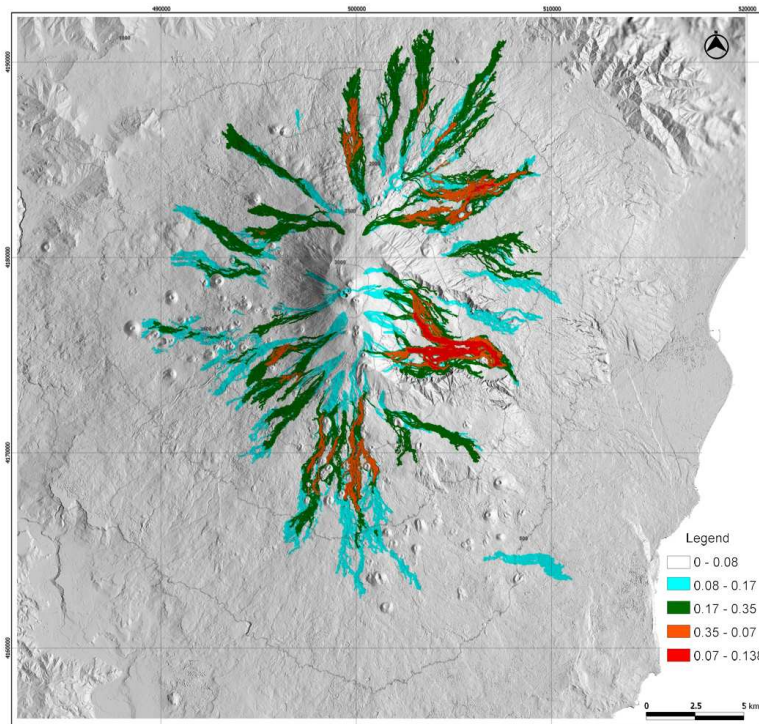


Figure 5.10: Lava-flow invasion scenario at Mount Etna resulting from the simulations of 1006 lava flows originated from a uniform square grid, by considering 1 simulation per vent. Legend indicates different probability classes of areas that could be interested by a lava event in the future.

In addition to the construction of lava risk maps, the presented strategies were initially adopted for carrying out extensive sensitivity analysis to model parameters. Also, the fast simulation of a number of lava simulations gives the opportunity, for the interactive visualization, to run the combined rendering and simulations at interactive frame rates.

Future work will also regard the adoption of multiple kernels, permitted by the recent Fermi architecture, where different streams of kernels can be simultaneously activated which can evidently permit a better management of simulations. Eventually, the same presented techniques are currently being exploited in the application of evolutionary algorithms, which require an extensive simulation phase due to the numerous evaluations of possible candidate solutions, for the morphological evolution of protection works for lava flow stream deviation.

## Chapter 6

# An Interactive Visualization System for Lava Flows Cellular Automata Simulations

The increase in the calculation speed of computers allows to obtain more accurate scientific simulations, through the generation of huge amounts of numerical data: it would be impossible to analyse them or understand their meaning without a valid tool.

Scientific Visualisation can assist scientists by providing processes, techniques and tools that transform data into images that can be analysed without intermediaries. The easiest way to define scientific visualisation is to consider it as the process of transformation of experimental data into images that can be readily analysed, by what is currently the most powerful computer in the world: the human brain.

In this Chapter, CCAFramework, an extensible system for the analysis and interactive visualisation of Cellular Automata Based simulations is presented. Due to the adoption of CCAFramework, a software tool was subsequently developed for the interactive visualisation and the analysis phase of the results for the morphological evolution of protective works by Parallel Genetic Algorithms. The core of the system is SCIARA-fv2. It is the latest release of the SCIARA Cellular Automata family and resides in a remote multi-GPU node which provides a multilayered GPU implementation in order to compute single or multiple simultaneous simulations. Experiment results are interactively visualised in real-time by means of a three-dimensional graphics engine implemented in C++ and VTK and integrated in Qt GUI.

The first Section of this Chapter briefly introduces the visualisation system architecture and its main features are described in Section 2. Finally, Section 3 concludes the Chapter with a general discussion about the per-



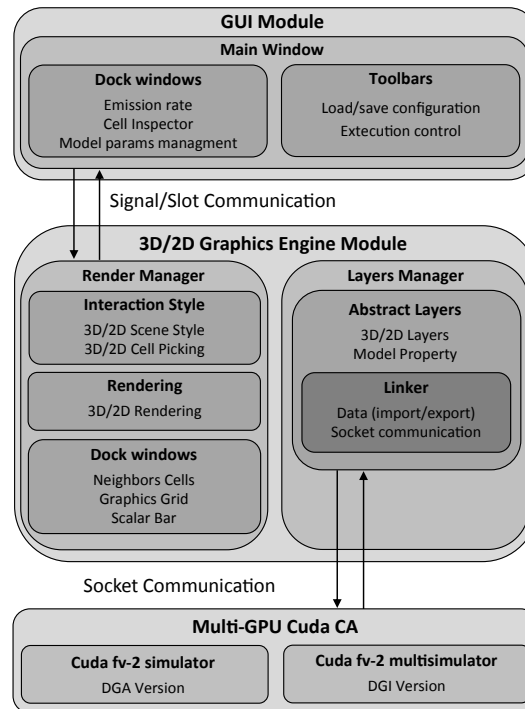


Figure 6.1: CCAFramework System Overview. As shown The CA model implementation resides in a remote high-performance cluster and data transfer takes place using a socket communication.

formed work.

## 6.1 The CCAFramework System Architecture

CCAFramework is a framework for analysis and interactive visualisation of simulations of complex phenomena modelled by Cellular Automata. The visualisation system is based on a three-dimensional graphics engine implemented in C++ and VTK/OpenGL [147, 32] and integrated into Qt Graphical User Interface(Qt GUI). The System architecture is scalable and modular. Interaction between the GUI and the visualisation system is guaranteed by Qt's Signal/Slot communication mechanism and results are accessible in real-time by means of the graphics engine module that establishes a connection with the remote SCIARA model via socket protocol. In Figure 6.2 is showed an example of application of the CCAFramework for lava flows simulations. As presented in Figure 6.1, the architecture is composed of three different layers.

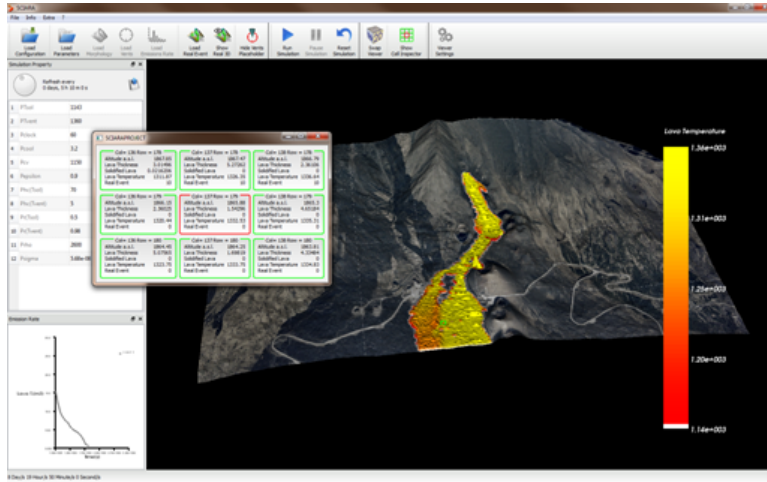


Figure 6.2: SCIARA Software main window. The application adopts the CCAFramework Visualisation System as Graphics Engine.

The CA Model implementation is the layer of the generic numerical model to simulate. It resides in a remote high-performance cluster and the interaction with the visualisation module is possible due to a socket communication. The output generated by the CA Model is the input for the Visualisation System. The CA Model resides in a Multi-GPU cluster and provides a double CUDA SCIARA-fv2 implementation layer: one relative to a CUDA accelerated version for the single simulation (Data Analysis) of lava flow event and the one for a large number of simultaneous simulations (Risk Maps, Genetic Algorithms). The 3D/2D Graphics Engine Module is the visualisation system layer. It is the software core, implemented in VTK/OpenGL. It coordinates the rendering process, manages lights, cameras and tridimensional objects within the scene. The last layer, the GUI module, concerns the System User Interface by integrating the visualisation system. It is Qt based and provides several tools to manage the CA graphical modelling and to interact with the numerical model.

## 6.2 Overview of CCAFramework

The software features can be divided into three categories: Modelling tools, Visualisation tools and Real-Time interaction tools.

**Modeling tools** The software offers the possibility to model the cellular automata substates due to the combination of objects within the scene ren-

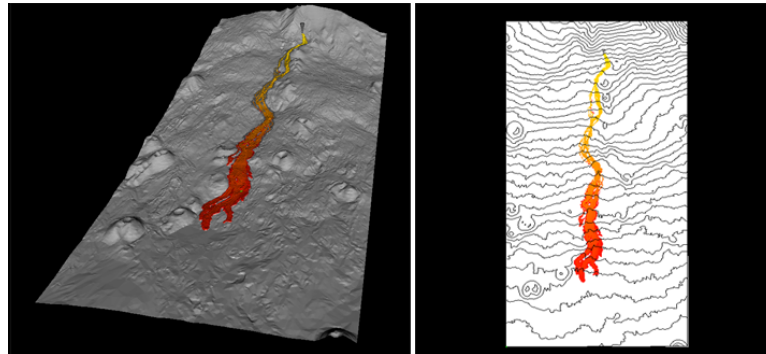


Figure 6.3: Double visualisation system of CCAFramework. The framework provide both prospective (left) and orthogonal (right) views.

dering by creating and customizing 3D/2D typed models. The technique of three-dimensional representation of a substate is based on the use of a structure with topology and geometry property and the meshing model used for this type of object is a triangle strip meshing. The 2D model, however, is represented by a color mapping technique that makes it possible to match a colour point of the three-dimensional structure based on a colorimetric default table. Moreover, The user can customize the scene by introducing the system of rendering contours, textures, models or generic labels(such as water or trees).

**Visualisation tools** CCAFramework offers a double visualisation system (see Figure 6.3) including a 3D view for the prospective view of the simulated phenomena and a 2D view for the orthogonal visualization.

**Real-time interaction tools** Real-time interaction tools provide several scene interaction styles in order to manage the cameras with rotation, zoom and translation operations. The data analysis is allowed by cell picking tool (see Figure 6.4), through wich the user can select, view and edit the set of cell substates values. The user can also control the execution of the simulation with the play,stop and pause buttons or by editing of the simulation parameters.

### 6.3 Discussion

In this Chapter, an efficient Visualization System for Cellular Automata model simulations has been presented. A modular system architecture so-

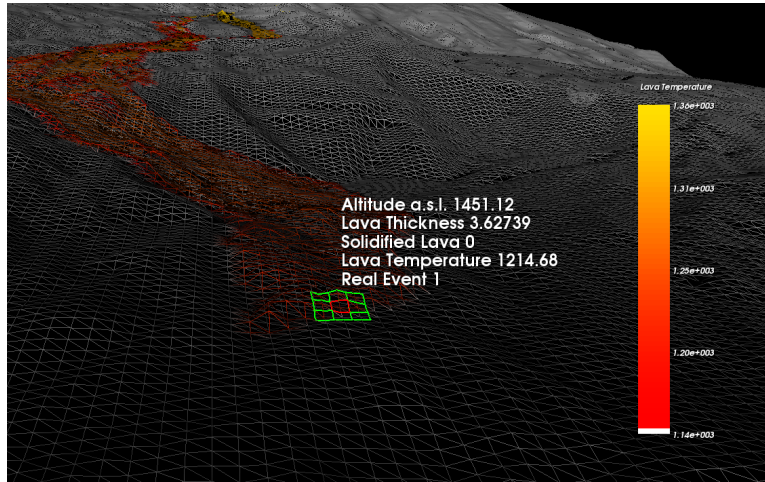


Figure 6.4: Cell Picking tool allows to get the coordinates of the Cellular Automata Cell selected and the its substates values.

lution was adopted to guarantee a clear separation of the interactive GUI process (client) and the computation process (server). Among the different advantages of the considered approach, one of the most interesting consists in the possibility to easily change the computational model without the need to carry out substantial modifications to the overall application.

CCAFramework was successfully applied for the rendering of lava flow simulations and commencing from the issue of accelerating the real-time visualization, the usage of a dedicated server with multiple GPUs was adopted. The System showed that it can run the combined rendering and simulations at interactive frame rates.

CCAFramework has proved to be of extraordinary importance for the analysis of data and emergent behaviors concerning to the experiments discussed in Chapter 7.



# Chapter 7

## A new methodology for mitigation of lava flow invasion hazard

The determination of areas exposed to be interested by new eruptive events in volcanic regions is crucial for diminishing consequences in terms of human casualties and damages of material properties. Nevertheless, urbanized areas, cultural heritage sites or even important infrastructures, such as power plants, hospitals and schools can be protected by diverting the flow towards lower interest regions.

In order to mitigate the destructive effects of lava flows along volcanic slopes, the building of artificial barriers is a fundamental for controlling and slowing down the lava flow advance. Such protective interventions were trialled during a few recent eruptions of Etna: in 1983, 1991-1993, 2001 and in 2002, when earthen barriers were built to control lava flow expansion with different level of success.

The proper positioning of protective measures may depend on many factors and hazard mitigation interventions plans, in this context, are empirical because no standard methodologies exist.

In this Chapter a decision support system for defining and optimising volcanic hazard mitigation interventions is proposed. In particular, this work describes the application of Parallel Genetic Algorithms for optimizing earth barriers construction by morphological evolution, to divert a case study lava flow that is simulated by the numerical Cellular Automata model SCIARA-fv2 at Mt Etna (Sicily, Italy). The GA application regards the optimization of the position, orientation and extension of earth barriers built to protect Rifugio Sapienza, a touristic facility located near the summit of the volcano. The study has produced extremely positive results and represents the first

application of morphological evolution for lava flow mitigation.

Therefore, the GPGPU implementation techniques presented in Chapter 5 were applied to accelerate the GA execution and the visualization system presented in Chapter 6 was extended to allow interactive analysis of the results. A study of GA dynamics, with reference to emergent behaviors, is also discussed.

The Chapter is organized as follows: after a brief description of the case study adopted for the experiments, the main characteristics of three different strategies were discussed and results are presented. For each developed version, a study of GA dynamics, with reference to emergent behaviors, and general considerations are also discussed. Last Section concludes the Chapter with final comments and future works.

## 7.1 A brief history of mitigation actions in Mt.Etna

The technique to divert lava flows by means of artificial barriers or channels is due to the idea that lava encountering natural morphological obstacles can divert its path. These mitigation interventions can represent a valid solution to delay and divert lava flows toward uninhabited areas.

The first documented mitigation action adopted to delay the lava flow advance has been applied during the most destructive Mt. Etna eruption occurred in 1669 [1].

In 1983, explosives were used as an intervention against lava flow invasion by excavating a diversion channel in order to thin a levee. Because of the numerous difficulties and time delays, only 20% of the flow was diverted out of the main channel and several rubble barriers were constructed to protect the main tourist complex on Etna. The barriers prevented lateral spreading of the flow field into developed areas. For the first time, earthen barriers, considered as active direct intervention during a volcanic event, showed their effectiveness to divert lava.

During the 1991-1993 eruption, main interventions to protect Zafferana regarded the building of four earth barriers and attempts to plug the lava tube by throwing blocks, steel hedgehogs and large fragments of solid lava into a skylight close to the vent [7].

The 2001 eruption of Mt.Etna caused damage and lava emitted from the lowermost vents threatened some important facilities of the Rifugio Sapienza area [6]. Thirteen earth barriers were built, during the July-August 2001, to protect the area. The first five upper barriers were almost totally buried by

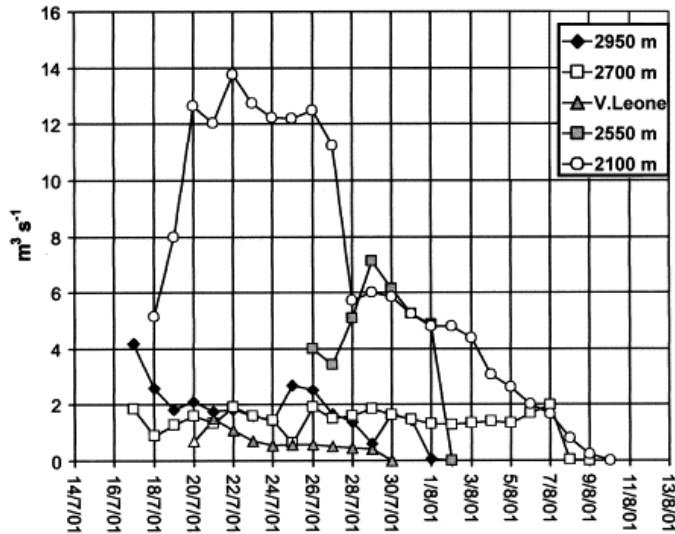


Figure 7.1: Effusion rates at the main vents of the 2001 eruption of Mt Etna [10]

the lava flowing, whereas the four barriers erected close to Rifugio Sapienza for diverting the approaching flow worked properly.

The 2002-2003 eruption produced two lava flows which partially covered the tourist facilities of Piano Provenzana. On the South flank the effusion lasted until 28 January 2003 threatening once again the Rifugio Sapienza area. Six barriers (five on the south and one on the north-east flank), oriented about 30 degree with respect to the main direction of the flow, were erected to contain the flow. Unfortunately, the lava overflowed from the barrier destroying two buildings and cutting the “SP92” road before stopping soon after.

## 7.2 The Case Study: The 2001 Mt Etna Eruption

As reported in [6](see also [10]) the 2001 eruption of Mt. Etna began on July 17 and was characterized by lava emission from several vents on the southern flank of the volcano at elevations of 2100 m, 2550 m, 2600 m, 2700 m, 2950 m, 3050 m (see Figure 7.2). Only lava flows emitted from the lowermost vents (2100 m, 2550 m, 2700 m) caused damage and threatened some important facilities and infrastructure, which were protected by earthen barriers. Effusion rates at the main eruptive vents were estimated daily by



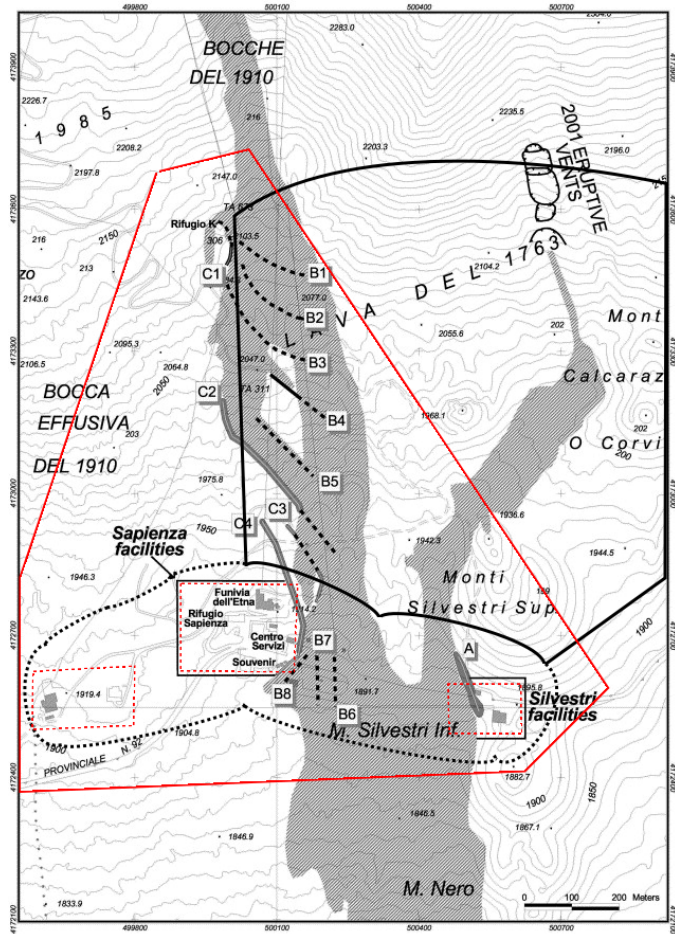


Figure 7.2: Set of interventions carried out during the 2001 eruption event at Mt Etna. Black lines are related to the case study adopted in the SBA(Ms. Silvestri zone), contrariwise, red lines define the input for the second case study(Sapienza zone). Furthermore, in both cases, the dashed perimeters represent the Rifugio (security area), wich delimitates the area that has to be protected by the flow. The solid line perimeters specifies the area in wich the earth barriers can be located.(base figure taken from [6])

[10] from the volume/time ratio and were obtained by careful mapping of the flow area and estimating its mean thickness (Figure 7.1).

The facilities of the Sapienza zone were undoubtedly at risk because of their short distance from the 2700-m and 2550-m effusive vents (respectively 3 and 2.5 km) and thirteen artificial barriers were built during the JulyAugust 2001 Mt. Etna eruption. Their locations are shown in the map of Figure 7.2.

Barrier	Length(m)	Height(m)	Base Width(m)	Volume ( $m^3$ )
A	145	10	25	14000
C2	375	6	12	13300
C3	190	6	10	6800
C4	300	10-12	15	25000

Table 7.1: Dimensions of the four largest barriers built to control lava flows

### Mts. Silvestri Zone

The flow emitted from the lower vent, the 2100-m fissure, interrupted the road SP92 and invaded the adjacent wide parking area located between Mts. Silvestri and the Sapienza zone (1900 m a.s.l.). A large barrier, on the eastern flank of the flow, was built to protect two tourist facilities. This barrier worked effectively and the two buildings were saved (Barrier A in Figure 7.2 and in Table 7.1). Three additional barriers were built (B6-B7-B8 in Figure 7.2) but they were successively buried by the lava descending from 2250-m vent.

### Barriers uphill from Sapienza Zone

The tourist facilities of the Sapienza zone were threatened by the lava because of the flow descending from vent located at 2550 m elevation. To protect the facilities, nine earthen barriers were built (Figure 7.2). Five barriers (B1-B2-B3-B4-B5 in Figure 7.2), 70-150 m in length and 6-8 m in height, were built from the 21st to 23rd July. They were almost totally buried by the lava flow emitted, from 25th July, from the 2550m vent. After the 25th July, the most important intervention was actioned to protect the Sapienza zone by erecting four barriers, facing in a mainly NW–SE direction (C1-C2-C3-C4), to divert the lava flow toward the southeast where there was the park area, thus to minimize the damages. The majority of the Barrier C2 worked effectively and diverted the flow toward the southwest, except the southern part, which was buried by the lava. Barrier C3 partially diverted the flow but, because of the subsequent advance of the flow thickness, lava started to overflow the barrier threatening the facilities located 150 m from the lava front. Some parts of the barrier previously built were removed to obtain a new barrier (C4 in Figure 7.2) placed at a distance of 70 m from the facilities. C4 was the last barrier erected and its northern part represented the last defense for Sapienza. During the evening of the same day, the effusion rate at the 2550m vent decreased and continued to decline during the following days (Figure 7.1) so the lava flow did not further threaten the Sapienza zone.

### 7.3 Morphological Evolution of protective works through Parallel Genetic Algorithms

By considering the 2001 Nicolosi case-study, GAs were adopted in conjunction with the SCIARA-fv2 CA model for the morphological evolution of protective works to control lava flows. The numerical model finite set of states was extended by introducing two substates defined as:

$$Z \subseteq R \quad (7.1)$$

where  $Z$  is the set of cells of the cellular automaton that specifies the Safety Zone, which delimitates the area that has to be protected by the lava flow; and

$$P \subseteq R, P \cap Z = \emptyset \quad (7.2)$$

where  $P$  is the set of cells of the cellular automaton that specifies the Protection Measures Zone that identifies the area in which the protection works can be located. The Protection work  $W = B_1, B_2, \dots, B_n$  was represented as a set of barriers, where every barrier  $B_i = N_{i1}, N_{i2}$  is composed by a pair of nodes  $N_{ij} = x_{ij}, y_{ij}, z_{ij}$ , where  $x_{ij}, y_{ij}$  represent CA coordinates for the generic node  $j$  of the barrier  $i$ , and  $z_{ij}$  the height (expressed in m).

The solutions were encoded into a GA genotype, directly as integer values (Figure 7.3), and a population of 100 individuals, randomly generated inside the Protection Measures Zone, was considered.

The choice of an appropriate fitness function is essential to evaluate the goodness of a given solution. In the present study, two different fitness functions were considered:  $f_1$ , based on the areal comparison between the simulated event and the Safety Zone (in terms of affected area) and  $f_2$ , which considers the total volume of the protection works in order to reduce intervention costs and environmental impact. More formally, the  $f_1$  objective function is defined as:

$$f_1 = \frac{\mu(S \cap Z)}{\mu(S \cup Z)} \quad (7.3)$$

where  $S$  and  $Z$  respectively identify the areal extent of the simulated lava event and the Safety Zone area, with  $\mu(S \cap Z)$  e  $\mu(S \cup Z)$  being the measures of their intersection and union. The function  $f_1$ , assumes values within the range  $[0, 1]$ : it is 0 when simulated event and Safety Zone Area are completely disjointed (best possible simulation); it is 1 where simulated event and Safety Zone Area perfectly overlap (worst possible simulation).



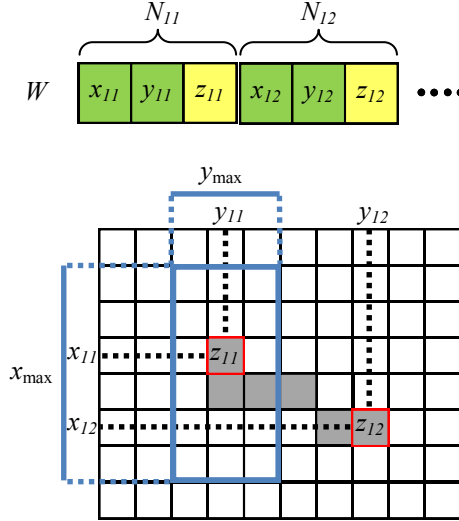


Figure 7.4: Representation of GA mutation phase.

For the genotype fitness evaluation, a composite (aggregate) function  $f_3$  was also introduced as follows:

$$f_3 = f_1 \cdot \omega_1 + f_2 \cdot \omega_2 \quad (7.7)$$

where  $\omega_1, \omega_2 \in R$  and  $(\omega_1 + \omega_2) = 1$ , represent weight parameters associated to  $f_1$  and  $f_2$ . The goal for the GA is to find a solution that minimizes the considered objective function  $f_3 \in [0, 1]$ . In order to classify each genotype in the population, at every generation run, the algorithm executes the following steps:

1. CA cells elevation a.s.l. are increased/decreased in height on the basis of the genotype decoding (i.e, the barrier cells). To complete this step, an extending Bresenham's original algorithm [19] is applied to determine the cells "inside" the segments between the work protections extremes.  $f_2$  is, subsequently, computed.
2. A *SCIARA - fv2* simulation is performed (about 40000 calculation steps) and the impact of the lava thickness on  $Z$  area ( $f_1$  computation) is evaluated. A *SCIARA - fv2* simulation is performed (about 40000 calculation steps) and the impact of the lava thickness on  $Z$  area ( $f_1$  computation) is evaluated.
3.  $f_3$  is computed and individuals are sorted according to their fitness.

GA parameters	Specification	Value
$g_l$	Genotypes length	[2-10]
$p_s$	Population size	100
$n_g$	Number of generations	[100-500]
$p_{mc}$	Coord. gene mutation probability	0.5
$x_{max}$	Gene x position variation radius	10
$y_{max}$	Gene y position variation radius	10
$p_{mh}$	Height gene mutation probability	0.5
$h_{min}$	height min variation range	-5
$h_{max}$	height max variation range	10
$p_c$	Crossover probability	0.05
$c_{h+}$	Cost to build	1
$c_{h-}$	Cost to dig	1
$\omega_{f1}$	$f_1$ weight parameter	0.95
$\omega_{f2}$	$f_2$ weight parameter	0.05
$n_{max-h}$	node max height	20
$n_{max-l}$	node max length	[40-500]

Table 7.2: List of parameters of the adopted GA

### 7.3.1 Parallel implementation and performance

The adopted GA is a rank based and k-elitist model, as at each step only the best genotypes generate off-spring. The 20 individuals which have the highest fitness generate five off-spring each and the  $20 \times 5 = 100$  offspring constitute the next generation. After the rank based selection, the mutation operator is applied with the exception of the first 5 individuals. The complete list of GA characteristics and parameters (with values related to the single-barrier solutions experiments) is reported in Table 7.2. Each gene mutation probability depends on its representation:  $p_{mc}$  for genes corresponding to coordinates value and  $p_{mh}$  for genes corresponding to heights. Therefore, if during the mutation process, a coordinate gene is chosen to be modified, the new value will depend on the parameters  $x_{max}$  and  $y_{max}$  that define the cell radius in which the node position can vary (Figure 7.4). The interval  $[h_{min}, h_{max}]$  is the range within which the values of height nodes are allowed to vary. This strategy ensures the possibility for the GA to provide, as output, either protective barriers or ditches. In this first approach, crossover has not been applied due to the high probability of generating inefficient candidate solutions.

To evaluate a given GA individual, an entire CA simulation has to be performed. Depending on the adopted computer framework, such an oper-

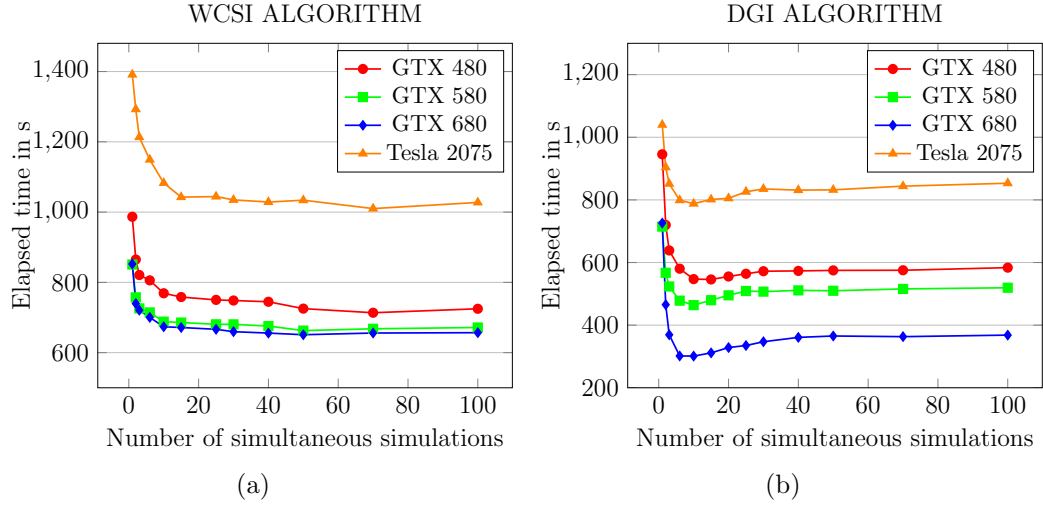


Figure 7.5: Elapsed time as a function of simultaneous lava events using the WCSI (a) and DGI (b) approaches on different considered GPGPU hardware.

ation may require several seconds, or even several hours. For example, on a 2-Quadcore Intel Xeon E5472, 3.00 GHz CPU such evaluation, by considering the first case study (Mts. Silvestri Zone), requires approximately 600 seconds, as at least 40000 CA steps are required for a simulation. Thus, if the GA population is composed of 100 individuals, the time required to run one seed test (100 steps) exceeds 69 days. Moreover, the GA execution can grow, depending on both the extent of the considered area and the number of different tests to run. Due to the high computational complexity of the algorithm, the CPU/GPU library presented in Chapter 5 was adopted to accelerate the GA running. In particular a Master-Slave model was used in which the Host-CPU (Master) executes the GA steps (selection, population replacement and mutation), while GPU cores (slaves) evaluate the individuals fitness. In order to test and evaluate the different implementation strategies with the GA realized, a landscape benchmark case study was considered, modelled through a Digital Elevation Model composed of  $200 \times 318$  square cells with a side of 10 m. A set of 50 hypothetical barriers placed with 2 different inclinations ( $135^\circ$ ,  $225^\circ$ ) to the lava flow direction was considered leading to a total of 100 simulations to be performed.

Two parallelisation strategies and different graphic hardware were adopted in all experiments reported below. In particular, four CUDA devices were used in the experiments: one nVidia Tesla C2075 and three nVidia Geforce graphic cards, namely the GTX480, GTX 580 and the GTX 680. Also, in order to quantify the achieved parallel speedup, sequential versions of the

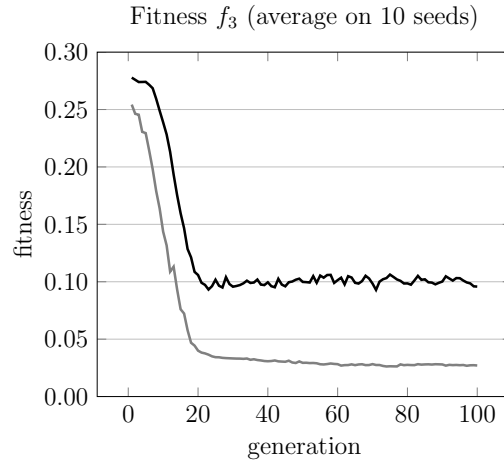


Figure 7.6: Temporal evolution of the composite  $f_3$  fitness of best individual (in black) and of average fitness of whole population (in gray). Fitness values were obtained as an average of 10 GA runs, carried out by adopting different seeds for generation of random numbers.

same GPU strategies were run on a workstation equipped with a 2-Quadcore Intel Xeon E5472 (3.00 GHz) CPU.

As reported in Chapter 5, a first straightforward parallel implementation, labeled as WCSI (Whole Cellular Space Implementation) was initially considered where the CUDA kernels operate on the whole automaton. For a fair comparison, the sequential version of the same algorithm was used and the elapsed time achieved by the CPU was 26039 s. Using the adopted GPU devices, the algorithm was solved with the WCSI approach and a variable number of simultaneous lava simulations. According to the results shown in Figure 7.5(a), the GTX 680 achieved the lowest elapsed time of 650,96 s, concurrently simulating 50 lava events. The gain provided by the parallelisation in terms of computing time was significant and corresponded to a parallel speedup of 40 over the used CPU.

The second strategy used is based on the alternative approach in which the grid of threads is dynamically computed during the simulation in order to keep low the number of computationally irrelevant threads. Using the reference CPU, such sequential procedure required 20180 s for the case study adopted. Figure 7.5(b) shows the corresponding times taken by the parallel DGI approach as a function of the number of concurrent simulations. As seen, the GTX 680 achieved the lowest elapsed time of 301,18 s, corresponding to a parallel speedup of 67.



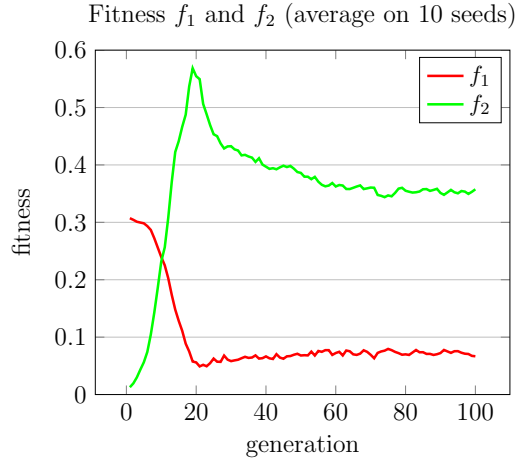


Figure 7.7: Temporal evolution of average fitness  $f_1$  (in red) and  $f_2$  (in green) of whole population. Fitness values were obtained as an average of 10 GA runs, carried out by adopting different seeds for generation of random numbers.

### 7.3.2 Single Barrier Approach: experiments and results

In the Single Barrier Approach (SBA), ten GA runs (based on different random seeds) of 100 generation steps each were carried out, each one with different initial populations. In order to test the effectiveness of this first version of genetic algorithm realized, the Mts. Silvestri case study, discussed in Section 7.2 was adopted.

The elapsed time achieved for the ten GA runs, by considering the first case study, was less than nine hours of computation on a 10 multi-GPU GTX 680 GPU Kepler Devices Cluster (note that the same experiment, on a sequential machine, would had lasted more than seven months).

Furthermore, during the running, a Visualization System Software (presented in Chapter 6), based on OpenGL and C++ and integrated into Qt interface, allowed the interactive visualization and analysis phases of the results. By using the SBA as a preliminary approach, only solutions with two nodes were considered ( $|W| = 1$ ), while Z and P were chosen as in Figure 7.2. The cardinality of W (Protection work) and the gene values in which they are allowed to vary (depending of Z area), define the search space  $S_r$  for the SBA GA:

$$S_{r-SBA} = \{[P_{x_{min}}, P_{x_{max}}] \times [P_{y_{min}}, P_{y_{max}}] \times [(h_{min} \cdot n_g), (h_{max} \cdot n_g)]\}^2 \quad (7.8)$$

Barrier	Length ( <i>m</i> )	Height ( <i>m</i> )	Base Width( <i>m</i> )	Volume ( <i>m</i> <sup>3</sup> )	Inclination (degrees)
[134, 173, 18][114, 158, 35]	250	26,5	10	66250	143
[135, 178, 8][114, 157, 43]	297	25,5	10	75731	135
[133, 172, 19][115, 155, 37]	247	28	10	69324	137
[113, 158, 45][132, 177, 9]	269	27	10	72549	135
[115, 154, 44][133, 171, 14]	248	29	10	71800	137
[133, 172, 12][115, 155, 42]	248	27	10	66848	137
[114, 159, 40][133, 171, 10]	225	25	10	56180	148
[115, 156, 48][134, 173, 9]	255	28,5	10	72661	138
[134, 173, 8][114, 157, 41]	256	24,5	10	62750	141
[115, 152, 38][134, 172, 18]	276	28	10	77241	134

Table 7.3: Dimensions of the ten best barriers carried out by GA run

Regarding the first case study adopted, the temporal evolution of the  $f_3$  fitness is graphically reported in Figure 7.6, in terms of average results over the ten considered experiments. The study, though preliminary, has produced quite satisfying results. Among different best individuals generated by the GA for each seed test (Table 7.3), the best one consists of a barrier with an average height of 25 m and 225 m in length with an inclination angle of 148° with respect to the direction of the lava flow. The barrier (row 7 in Table 7.3) completely deviates the flow avoiding that the lava reaches the inhabited area. The relative elevated height of the barrier is due to the adopted GA experiment parameter values are also listed in Table 7.3. The related CA simulation, obtained by adopting the best individual is shown in Figure 7.9.

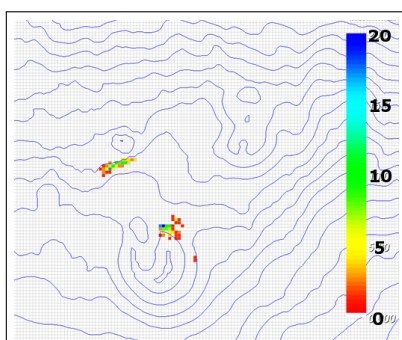


Figure 7.8: Nodes distribution of the best 100 solutions generated by the GA. Scale values indicates occurrence of nodes.

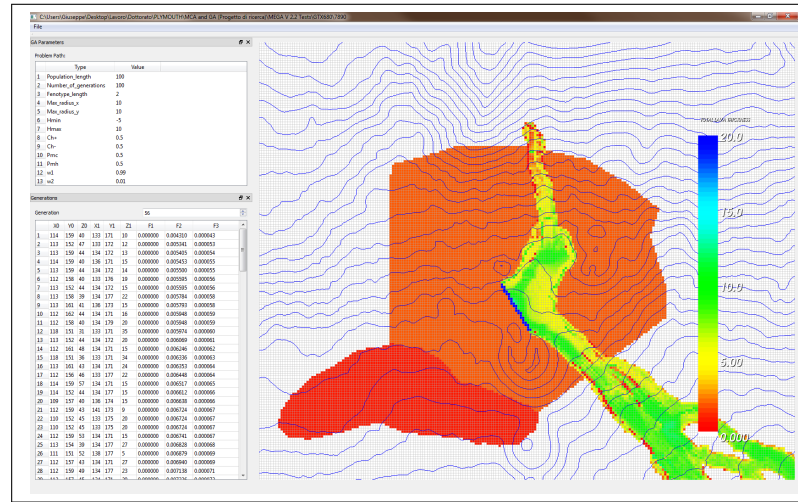


Figure 7.9: SCIARA-fv2 simulation visualization adopting the SBA GA best solution. As seen, the devised barrier (in blue) completely diverts the lava flow from the Safety Area (in red)

### 7.3.2.1 Consideration on the GA dynamics and emergent behaviors

In the GA experiments that have been performed, individuals with high fitness evolved rapidly, even if the initial population was randomly generated and the search space was quite large (Equation 7.8). By analyzing several individuals evolved in ten different GA executions, similar solutions were observed. This behavior is due to the presence of problem constraints (e.g. morphology, lava vent, emission rate, Z and P areas) that lead the GA to search in a region of the solution space characterized by a so called local optimum. In particular,  $f_1$  reaches the minimum value (0) around the twentieth GA generation and the remaining 80 runs are used by GA for the  $f_2$  optimization (cf. Figure 7.7). In any case, the evolutionary process has shown, in accordance with the opinion of the scientific community (e.g., [6, 72]), the ineffectiveness of barriers placed perpendicular to the lava flow direction despite diagonally oriented solutions (130 – 160) (see Table 7.3).

Furthermore, a systematic exploitation of topographical characteristics by GA, during the evolutionary process, has emerged. To better investigate such GA emergence behavior, a study of nodes distribution was conducted (Figure 7.8).

By considering the 100 solutions provided by GA, each node was classified on the basis of the *slope proximity* calculation, as the average of altitude differences between node neighborhood cells (radius = 10) and the central

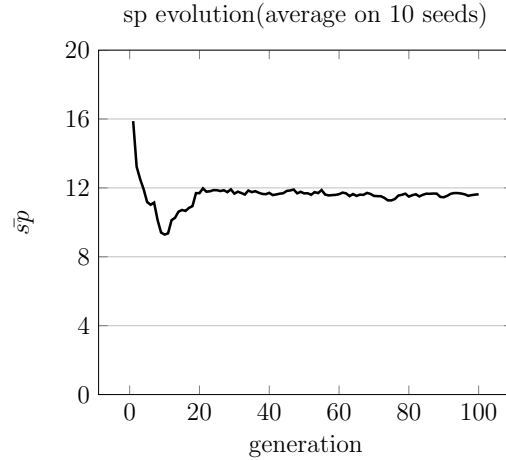


Figure 7.10: Temporal evolution of average slope proximity values for the best individuals.

cell. More formally, the function that assigns to each generic node  $N_{ij}$  a *slope proximity* value is defined as:

$$sp_{ij} = \frac{\sum_{i=1}^{|X|} \bar{z}_i - \bar{z}_0}{|X|} \quad (7.9)$$

where  $X$  is the set of cells that identifies the neighborhood of the cell  $i,j$  and  $\bar{z}_i \in Q_z$  is the topographics altitude (index 0 represents the central cell). As shown in Figure 7.10, starting from the tenth GA generation, the evolutionary process has shown an increase in slope proximity values. Therefore, after the  $f_1$  optimization (cf. Figure 7.7), in order to minimize  $f_2$ , there is a specific evolutionary temporal phase (i.e., up to the 25th generation) where the algorithm generates solutions that are located in the proximity of elevated slopes.

### 7.3.3 Evolving multiple barrier solutions

The SBA, discussed in the previous paragraph, has produced encouraging results. However, considering two-nodes barriers is a strong limitation and a critical aspect of GA implementations that can improve the efficiency of the final solution is the possibility to provide multi-barrier protection measures. For these reasons the GA model has been reengineered and two different strategies have been adopted.

In order to compare the final result provided by GA with the real case, only the case study regarding the barriers in the uphill Sapienza zone, de-

scribed in Section 7.2, was taken into account. The first straightforward implementation is a Evolutionary Greedy Strategy (EGS) based on the iterative execution of the original algorithm (SBA), with the application of dimensional constraints to the barriers, until reaching a stop criteria ( $f_1 = 0$ ). Note that by using EGS, the number of sub-solution that will compose the final solution is not known a priori. In the second adopted strategy, called Coevolutionary Cooperative Strategy (CCS), all barriers are encoded in the genotype and because all the constituents parts of the solution interact with the GA environment, a mechanism of cooperation between individuals has been favored.

After a brief description of the main characteristics of the two implemented strategies, experiments and results are presented. Some considerations of the GA dynamics and emergent behaviors are also discussed. This Chapter terminates with a comparison of efficiency between the two models and with final comments.

### 7.3.3.1 Evolutionary Greedy Strategy

The first implemented strategy to generate multi-barrier solution is an Evolutionary Greedy Technic. The number of barriers belonging to the final solution is implicitly determined on the basis of the genotypes dimensional constraints and on the number of iterations required to reach the stop criteria. The algorithm operates, at each stage of advancement, on the best single protection works generated in the previous step and iterates until Safety Zone Area and simulated event are completely disjointed ( $f_1 = 0$ ).

The strategy approach can be represented by a tree data structure (Figure 7.11), where child nodes denote the best genotypes evolved by using the parameters (morphological changes) of the parent node. In particular, each node  $n_{ij}$  represents the best solution (single barrier) evolved during the run  $i$  with seed  $j$  ( $i$  is the level of the tree and represents  $i$ -th barrier inserted on the morphology). The vertex  $V([i, j], [k, m])$  represents the state of the morphology that depends of the parents nodes for all the previous run. By considering  $f_2^*$  the temporary best value of  $f_2$  (with  $f_1 = 0$ ) reached from the algorithm and  $f_{2tot}^{ij}$  the accumulate fitness value for the node  $ij$ , for each node to process, the algorithm operates as the follows:

$$\bullet \text{ } if(f_1 > 0.0) \left\{ \begin{array}{l} if(f_{2tot} < f_2^*) \rightarrow \text{generates three child nodes} \\ if(f_{2tot} \geq f_2^*) \rightarrow \text{terminates} \end{array} \right\}$$

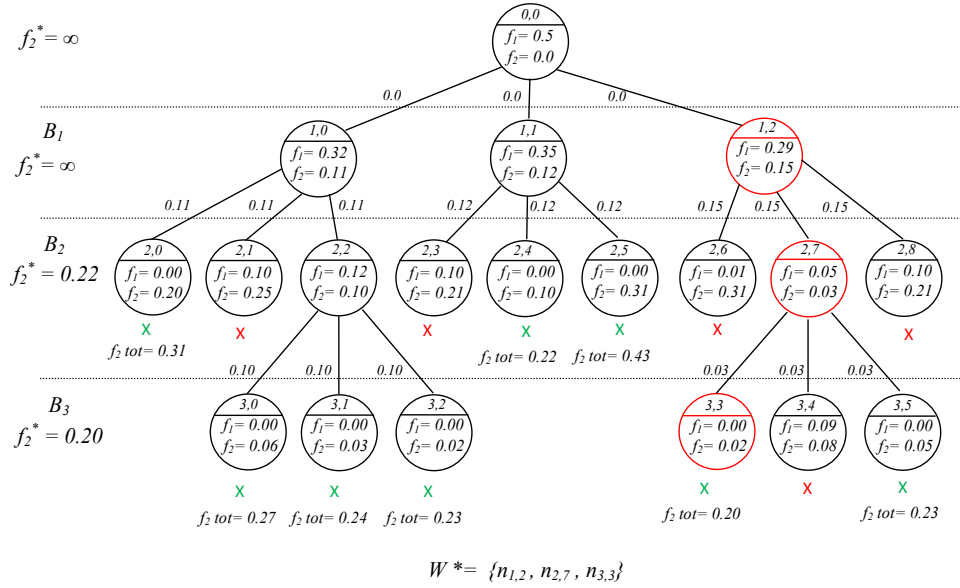


Figure 7.11: Tree representation of an evolutionary greedy strategy algorithm run. Each node represents a GA run and, at the end of the process, it contains and transfers information about the best partial solution evolved to each children node. The process is iterative and each children node operates on an updated morphology according to the best solution evolved by the parent node. When all nodes reach a termination condition (represented by the “x” label in the figure) the best set of barriers is determined by the path from the root node to the one with the best  $f_{2tot}$  value (red nodes in the figure).

$$\bullet \text{ if}(f_1 == 0.0) \left\{ \begin{array}{l} \text{if}(f_{2tot} < f_2^*) \rightarrow f_2^* = f_{2tot}, \text{ terminates} \\ \text{if}(f_{2tot} \geq f_2^*) \rightarrow \text{terminates} \end{array} \right\}$$

The algorithm terminates when all the tree nodes are in a termination condition and the best solution is obtained from the set of partial solutions (tree nodes) of the unique path starting from the root node to the final node with  $f_{2-tot} = f_2^*$ .

**Experiment and results** Six EGS trials based on different dimensional constraints for each barrier were carried out. For each GA run, a population of 100 individuals and 100 generation steps were considered. The set of constraints has been chosen on the basis of the results obtained on the single

Constraints	Barrier	Length (m)	Height (m)	Base Width(m)	Volume (m <sup>3</sup> )	Inclination (degrees)
<i>nolimits</i>	[193,88,-2] [238,122,13]	564	5.5	10	26700	143
140 × 12	[162,93,11] [169,104,7]	130	9	10	10800	122
	[236,110,3] [243,112,5]	73	4	10	3200	164
total					14000	
120 × 12	[173,104,4] [169,93,12]	117	8	10	9600	110
	[175,98,10] [174,107,1]	91	5.5	10	5500	180
	[245,110,2] [238,111,3]	71	2.5	10	2000	180
total					17100	
100 × 12	[160,93,10] [166,101,7]	100	8.5	10	7650	127
	[211,109,12] [209,105,3]	45	7.5	10	3750	117
	[237,111,3] [246,110,2]	91	2.5	10	2500	174
total					13900	
80 × 12	[162,99,8] [157,93,3]	78	5.5	10	3850	130
	[212,108,6] [205,105,3]	76	4.5	10	3600	157
	[172,98,0] [173,97,5]	14	2.5	10	500	135
	[236,110,5] [243,112,5]	73	5.0	10	4000	164
total					11950	
60 × 12	[170,93,4] [173,96,1]	42	2.5	10	1000	135
	[213,104,12] [214,109,11]	51	11.5	10	6900	101
	[242,113,5] [237,110,10]	58	7.5	10	4500	149
total					12400	

Table 7.4: Dimensions of the best barriers evolved after the EGS GA runs

barrier approach in such a way that there exists no feasible solution consisting of a single barrier.

Even if at the end of the computation, a multi-barrier solution is provided by the algorithm, because of its iterative strategy, the search space for EGS is:

$$S_{r-EGS} = \sum_{i=1}^n \{ [P_{x_{min}}, P_{x_{max}}] \times [P_{y_{min}}, P_{y_{max}}] \times [(h_{min} \cdot n_g), (h_{max} \cdot n_g)] \}^2 \quad (7.10)$$

where n is the three depth reached from the EGS algorithm to find the best solution.

Final results are summarized in Table 7.4. The best final solution among six different runs, obtained by using a max length of 80 m and a max height

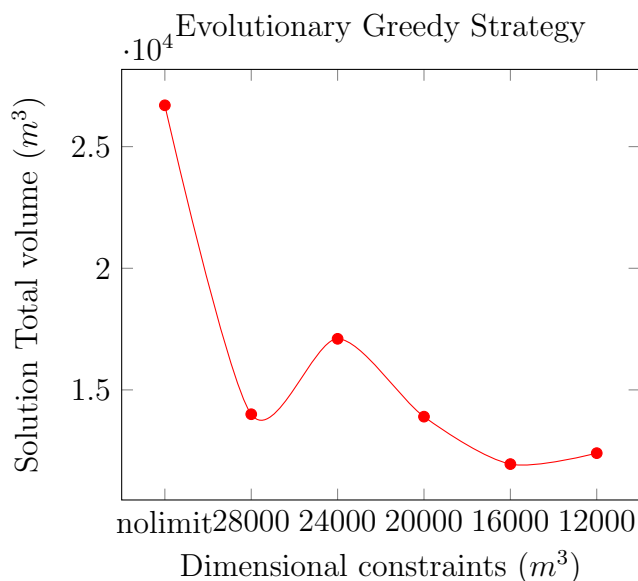


Figure 7.12:  $f_2$  fitness value in the variation of dimensional constraints associated to the single barrier with EGS.

of 12 m for each barrier, consists of four different barriers that completely deviate the flow so that the lava avoids reaching the inhabited area. Figure 7.12 shows different  $f_2$  build cost values, by varying dimensional constraint for each single barrier.

### Considerations

The use of the greedy strategy, in combination with the application of dimensional constraints to genotypes, has allowed the generation of more efficient results than those of the first algorithm version (SBS) based on single barrier strategy. In particular, all the solutions provided by EGS were better than the original single-barrier version and in the best case (see Table 7.4) the build cost of the best solution was 123% lower than in the single-barrier case. As showed in Figure 7.12, in general, the total volume of construction used to divert the lava flows from the area to be protected is inversely proportional to the dimensional constraints associated to the single barrier. On the other hand, the algorithm is very computationally expensive and decreases the dimensional constraints for the single barriers implies that the algorithm to terminate reaches a greater depth in the computation tree. Since to process  $n$  levels with EGS might require  $\sum_{i=1}^n x^n$  different runs, where  $x$  is the number of seed tests, the total execution time grows exponentially with the graph



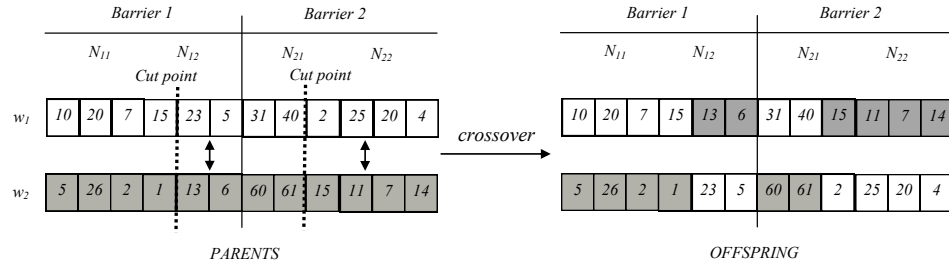


Figure 7.13: Example of n-point crossover between two genotypes. Two cutting points are chosen randomly and corresponding portions of parents recombined in order to obtain two offspring.

depth and the algorithm is applicable only for a limited set of constraints.

Another limitation of this approach is linked with the final solution optimality. In fact, this strategy does not guarantee that at the end of the procedure the optimal solution is found. It is worth to note that greedy algorithms make a series of myopic decisions, each of which, alone, solves some sub-problems optimally, but that together may not be optimal for the problem as a whole. For these reasons, a further strategy was developed, based on genotype multi-barrier encoding.

### 7.3.3.2 Cooperative Co-evolutionary Strategy

One of the weaknesses of the EGS approach lies in the evaluation of each solution separately. Some improvement might be obtained by introducing several barriers directly encoded in the genotype. However, this is made difficult by the fact that each sub-solution encoded in the genotype evolves without promoting an efficient cooperation with the other barriers. In fact, based on the experiments carried out during the first tests, the GA evolution focuses on the expansion of a single substructure making the final result inefficient. For these reasons, an alternative strategy was developed by introducing some modifications to the original approach.

The introduction of constraints related to the size of protective structures has allowed, however, a forced collaboration between the candidate solutions, thus avoiding the competitiveness between them. Moreover, to promote co-evolution and avoid greediness phenomena, a lifecycle value has been associated to each barrier and it represents the number of consecutive generations in which a solution is unable to divert lava flow. Barriers with the lifecycle value greater than the threshold are regenerated. As shown in Figure 7.14, this strategy avoids the survival of members within the genotype, which are

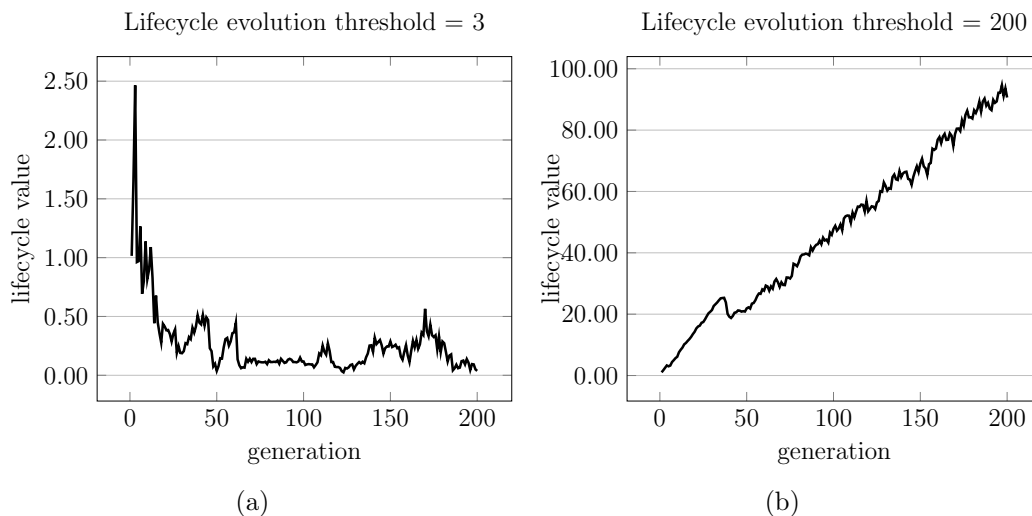


Figure 7.14: Temporal evolution of the average lifecycle value of whole population for best seed test evolved with the CCS strategy by adopting the lifecycle threshold parameter = 3 (a) and the same run by adopting no limit for the lifecycle threshold parameter (b).

quite ineffective for the mitigation lava flow purpose.

To ensure a better exploration of the search space and to avoid a fast convergence of solutions to local optima, with CCS, a  $n$  point crossover operator (see Figure 7.13) has been introduced. Two parent individuals are randomly chosen from the mating pool and two different cutting points for each parents are selected. Cut points always coincide with the first gene of a sub-solution and after the selection portions of the sub-solution chosen in the genotype, they are exchanged. The crossover operator is applied according to a prefixed probability,  $p_c$ , for each sub-solution coded in the genotype. Figure 7.13 shows an example of crossover between two genotypes.

### Experiments and results

Experiments were carried out by varying from the number of barriers adopted to their different dimensional constraints. For each GA run a population of 100 individuals and 500 generation steps were considered. By considering the CCS, the search space grows rapidly and depends on the cardinality of the final solution and the dimensional constraints. Equation 7.11 shows the search space without taking into account barriers length constraints.

$m^3 \times Barrier$	2 Barriers	3 Barriers	4 Barriers	5 Barriers
40000	NA	12100	NA	NA
60000	NA	8650	13320	10725
80000	13750	12030	10411	8150
100000	12283	12977	13689	13164
120000	16250	11742	16042	12372
140000	15849	13241	17526	17187
160000	27162	16887	19400	22065
180000	26354	19553	18423	24314
200000	24354	16639	17416	24388
no limits	NA	66000	82733	117333

Table 7.5: Total volume (in average on 10 seed tests) used to protect the Safety Zone, by varying number of barriers into the final solution and dimensional constraint for each single barrier.

$$S_{r-CCS} = \{[P_{x_{min}}, P_{x_{max}}] \times [P_{y_{min}}, P_{y_{max}}] \times [(n_{min-h}), (n_{max-h})]\}^{2|W|} \quad (7.11)$$

For example, regarding the specific case study adopted, by using 5 barriers encoded into the genotype and a prefixed height limit for each barrier, the search space is as follows:

$$S_{r-CCS} = \{[164, 316] \times [0, 169] \times [(-20), (20)]\}^{2 \times 5} \in N^{60} \quad (7.12)$$

The final results are summarized in the Table 7.5. The Maximum volume available (in  $m^3$ ) for all the protection works belonging to the final solution can vary in the range [40000-200000]. The number of barriers encoded into genotype chosen is from 2 up to 5. The best final solution, among several runs, was obtained by using 4 barriers, each of them with a size limitation of 125m of max length and 20m of max height. The best solution provided by CCS is extremely efficient and, in particular, the total volume used to deviate the flow avoiding that the lava reaches the inhabited area is less than 72% respect on the EGS and 284% respect on the SBS. Regarding the experiment in which the most efficient solution evolved, the temporal evolution of lifecycle values is graphically reported in Figure 7.14. Figure 7.15 shows different results, by only considering the  $f_2$  build cost (in average on 10 seed tests) fitness function obtained by varying dimensional constraint for each single barrier.

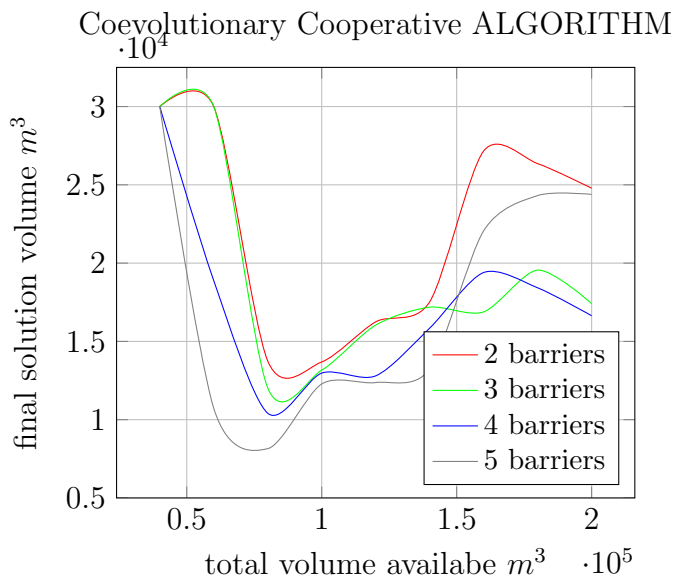


Figure 7.15:  $f_2$  build cost fitness function, by varying dimensional constraint for each single barrier.

### Considerations

By analysing the graph 7.15, it emerges that for every different number of barriers, there is a variation range of total available volume in which the final fitness is efficient and outside this range, the  $f_2$  value rapidly grows. This is due to the fact that in these regions the presence of a local minimum is very high. In fact, for all the values that exceed this range, solutions are inefficient because they have too much freedom to evolve, which means that the cooperation is not applicable because of the lack of co-evolution. For those values above, the mutation range can vary (see the mutation operator 7.4) in a very small radius which is a limitation for the search space exploration.

Although the more barriers used, the more efficient the final solution should be, there is a limit of barriers encoded in the genotype due to the fact that it implies the presence of a larger number of schemas (huge search space) and the approximation provided by the GA becomes inaccurate.

If  $V_{tot}(n), n \in N$  is considered as the total volume used to successfully protect the Protection Area by using exactly  $n$  barriers,  $V_{tot}(n) - V_{tot}(n + 1)$  is the benefit, in terms of final solution efficiency, due to the use of one more barrier than  $n$  with the same available total volume used. So,  $V_{tot}(1) - V(n^*)$  is the efficiency contributor, to the final solution, given by the cooperation of several barriers, where  $n^*$  represents the number of barriers used to reach the best solution. The average total volume saving, by adding a barrier to

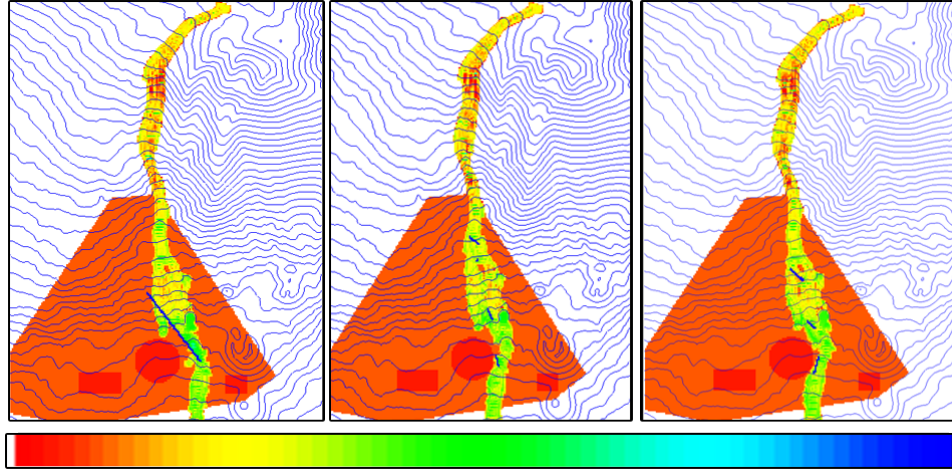


Figure 7.16: SCIARA-fv2 simulations adopting the best solution evolved with three different strategies (Single Barrier Approach (a), Evolutionary Greedy Strategy (b) and Coevolutionary Cooperative Strategy (c)). Lookup table refers to the lava thickness with a range between  $0 \text{ m}^3$  (white) to  $20 \text{ m}^3$  (blue).

the final solution is:

$$\frac{\sum_{i=1}^n V_{tot}(i) - V_{tot}(i+1)}{n} = \frac{V_{tot}(1) - V_{tot}(n)}{n} \quad (7.13)$$

and by considering the results obtained from the experiments it is:

$$\frac{(26700 - 8150)}{5} = 3710 \quad (7.14)$$

This means that, on average, the introduction of a new barrier to the final solution brings an advantage in terms of efficiency of  $3710 \text{ m}^3$  of construction. The total savings volume, by adopting the best multi-barriers algorithm than the single barrier approach is:

$$V_{tot}(1) - V_{tot}(4) = (26700 - 6950) = 19746 \quad (7.15)$$

Following the studies carried out for the first experiments, regarding the exploitation of morphological characteristic by GA, it was investigated when cooperation can play a very important role in leading the simulation towards to efficient solutions.

The possibility to manage the final solution by composing several barriers, essentially offers two advantages: the ability to generate multi-node barriers

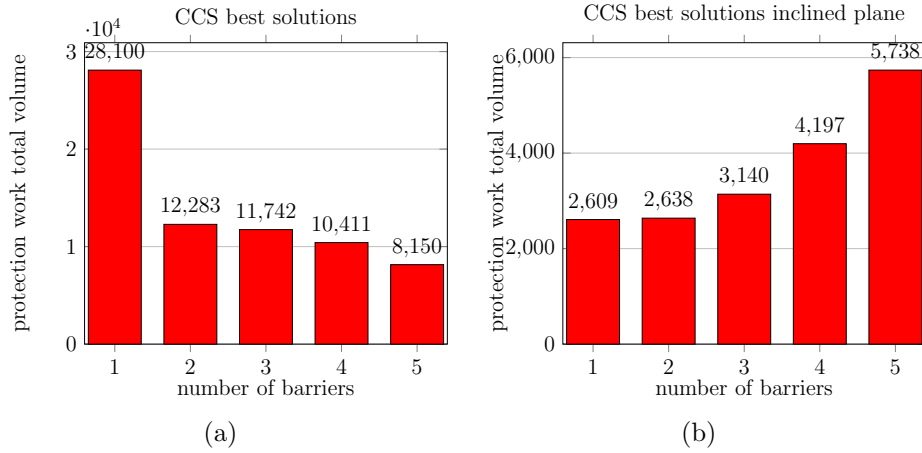


Figure 7.17: Total CCS volume solutions variation on the vary of number of barriers adopted to codify the final solution with the cooperative strategy using Mt. Etna morphology (a) and the inclined plane ideal case (b).

and the ability to create routes to divert the lava. The hypothesis is that cooperation between barriers is more important if the morphology can be exploited, which is possible if within the Protection Measures Zone a set of disjoint clusters of high slope proximity values can be found. This is because they can mark hypothetical lava paths and the barriers can be used to connect them, thus define and complete it.

To test this hypothesis, the numeric model input was modified by introducing a particular case of morphology, where for each CA cell the slope proximity value is null: an inclined plane. By considering the function 7.9 and an inclined plane (any degree of inclination), for each CA cell the average of altitude differences between node neighborhood cells and the central one is zero.

The same set of experiments were carried out for the inclined plane tests. The best final solution, among several runs, was obtained by using 1 barrier without size limitations. The final scenario, in this case, was completely different and results showed that there is no benefit, in terms of final solution efficiency, in using multi-barrier solutions. This is due to the fact that the morphology is not exploitable because of the total absence of clusters of cells with high slope proximity value. In this case, the average total volume saving value by adding a barrier to the final solution confirms the initial hypothesis:

$$\frac{V_{tot}(1) - V_{tot}(n)}{n} = \frac{(2609 - 5738)}{5} = -625 < 0 \quad (7.16)$$

This means, as also confirmed in results reported in Figure 7.17(b) that,

on average, that there is no gain in introducing a new barrier to the final solution when the morphology is not exploitable by the GA. In other words, the principle of cooperation is useless in this case.

### 7.3.4 Qualitative analysis of the results

Results in Figure 7.17(a) shows that instances of CCS algorithm that encode more barriers in the genotype produce better results than those that encode less.

In order to evaluate which of the strategies best fits into the problem to optimize earth barriers construction to divert lava flows for the 2001 Nicolosi case study, extensive experiments have been performed. In particular, results obtained in twenty independent executions (seed tests) for the four variants of the CCS algorithm have been analyzed by focusing on the best fitness value reached by the search process of any GA run. The Kruskal-Wallis non-parametric statistical test [103] with a significance of 95% has been used (p-value < 0.05). The purpose of this test is to verify whether the four independent samples ( $CCS_2, CCS_3, CCS_4, CCS_5$ ) are from the same population. When the Kruskal-Wallis test leads to significant differences in the distributions then at least one of the samples is different from the other samples and, for this reason, multiple comparisons test to determine which pairs of strategies are significantly different and which are not, have performed.

The p-value returned by the Kruskal-Wallis test is 0.0075, for the data obtained with the CCS algorithm with number of barriers  $n = 2, 3, 4$  and 5. So, the null-hypothesis is rejected and it is possible to conclude that the Kruskal-Wallis test confirmed that at least one strategy is significantly different.

Figure 7.18 shows the behaviour of the best fitness for each algorithm instance by varying the adopted number of barriers encoded into the genotype. Algorithms are sorted according to the median of the distribution (horizontal red line). The *SBA* instance obtained the worst results, sign that this version does not exploit the search space sufficiently. This is also due, as discussed before, to the limitation of the solution that the algorithm can generate (only two nodes). On the contrary, the instances of algorithm that use the maximum number of protection work (in this case 5) (*C05*) find the lower values. This is because, even if the search space become huge, the algorithm can better exploit the morphology by using more protection measures.

As it can be seen in the Figure 7.18, the values of the medians decrease inversely proportionally with the number of barriers adopted.

Since significant difference between ranks emerged, the Bonferroni multiple comparison test has been performed, in order to get information about

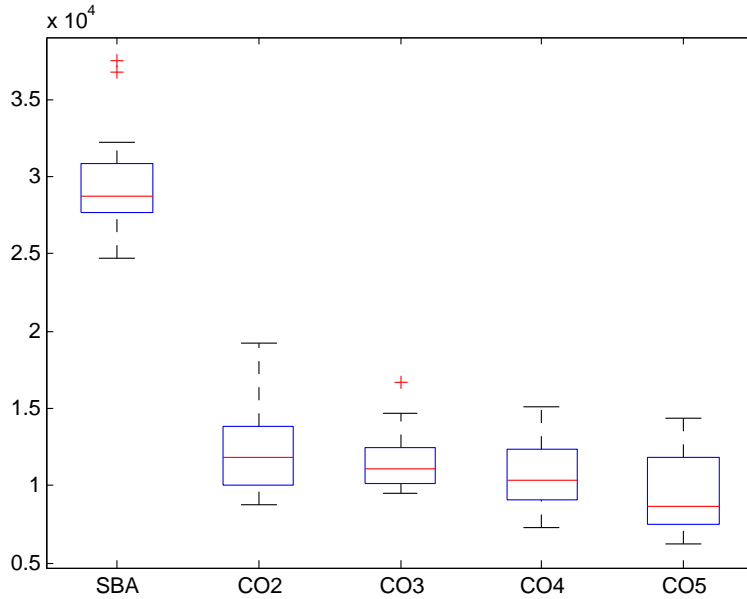


Figure 7.18: Best fitness obtained by each algorithm by varying the number of barriers  $n=1$ (SBA), 2,3,4 and 5.

which pairs of means are significantly different, and which are not.

A conservative familywise significance threshold of  $p_T < 0.005$  (i.e.,  $p_T = 0.05/10$  where 10 is the number of total comparisons by considering all the possible combinations) was applied and all pairwise comparisons were performed. Each comparison is considered statistically significant if the p-value obtained is less than or equal to the  $p_T$  value.

As shown in Table 7.6, in the results of multiple tests, the SBA distribution has statistically significant different with respect to all other algorithms. The results of multicompare test also show a relevant difference between

	SBA	$CCS_2$	$CCS_3$	$CCS_4$	$CCS_5$
SBA	-	-	-	-	-
$CCS_2$	0.0021*	-	-	-	-
$CCS_3$	0.0021*	0.6456	-	-	-
$CCS_4$	0.0021*	0.0850	0.1441	-	-
$CCS_5$	0.0020*	0.0031*	0.0087	0.0935	-

Table 7.6: Table of pairwise comparisons between different strategies adopted. Values with \* represent that the comparison is considered statistically significant



$CCS_2$  and  $CCS_5$ ; the rest of versions have similar behavior.

In particular, high p-values were encountered by comparing algorithm that use a similar number of barriers (e.g.  $CCS_2$ - $CCS_3$ ). In contrast, when the difference between encoded barriers by two different versions increases, the p-value decreases (e.g.  $CCS_2$ - $CCS_5$ ).

### 7.3.5 Conclusions

In this Chapter, a novel approach for devising protective measures to divert lava flows has been presented. Starting from the problem of the high computational complexity of the GA algorithm, a library was developed for executing a large number of concurrent lava simulations using GPGPU. The parallel speedups attained through the proposed approaches and by considering GPGPU hardware, were indeed significant. In fact, the adoption of PGAs permitted to perform, in reasonable times, a greater number of tests shortening the execution by a factor of 67. In addition to the GA algorithm acceleration implementation, an interaction visualization system was also developed for the analysis phases of the results.

A preliminary release of the algorithm called SBA, based on two nodes solutions, was initially considered and evaluated on the basis of two fitness functions. The first fitness function guarantees the goodness of the solution in terms of security, the second one minimizes the environmental impact.

First observations of the GA results permitted to conjecture the presence of local optima in the search space, probably due to problem constraints. To better investigate GA dynamic characteristics, a study of nodes distribution was also conducted and a systematic exploitation of morphological characteristics by GA during the evolutionary process emerged. PGAs experiments, carried out by considering the Nicolosi case-study, demonstrated that artificial barriers can successfully change the direction of lava flow in order to protect predefined point of interests. In particular, by performing extensive experiments, simulations demonstrated that protective works are more effective when placed nearly parallel to the flow direction, while a barrier placed perpendicular to the flow direction can only stop the flux temporarily, ultimately allowing the solidified crust to accumulate, causing the following mass to go over the barrier.

Because of the strong limitation related to single barrier solutions provided by the SBA, a second GA strategy was implemented in order to improve the efficiency of the final solution by introducing multi-barrier protection measures. An Evolutionary Greedy Strategy (EGS) was introduced and evaluated. Based on the iterative execution of the original algorithm (SBA), the use of the greedy strategy, in combination with the application of dimen-

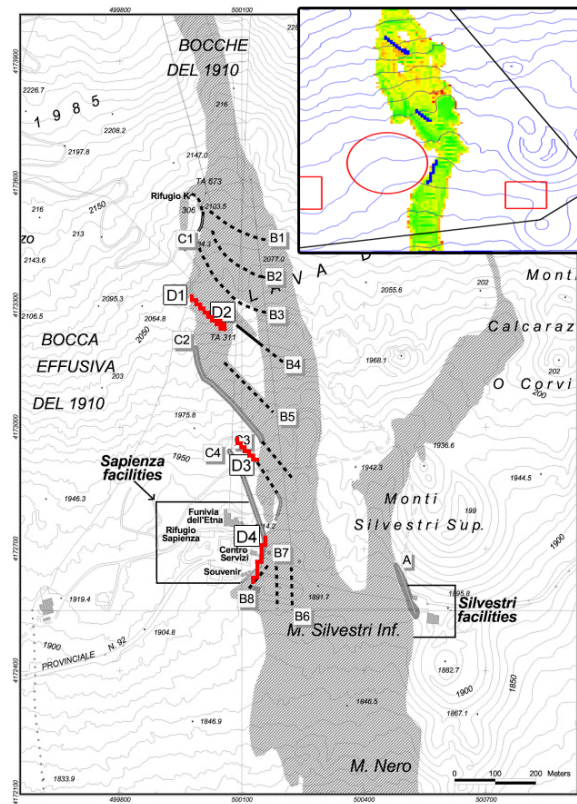


Figure 7.19: Comparison between earthworks built during the 2001 Mt.Etna event (in black) and those provided by the Genetic Algorithms (in red) to divert lava flows.

sional constraints to genotypes, has allowed the generation of more efficient results than those of the first algorithm version (SBS) based on single barrier strategy. However, this strategy does not guarantee that at the end of the procedure the optimal solution is found because of the limitation related to the Greedy Algorithms and because of no presence of coevolution between the GA individuals.

For these reasons, an alternative approach (called Coevolutionary Cooperative Strategy (CCS)) has been introduced where all barriers are encoded in the genotype and because all the constituents parts of the solution interact with the GA environment, a mechanism of cooperation between individuals has been favored.

Solutions provided by CCS were extremely efficient and, in particular, the total volume used to deviate the flow avoiding that the lava reaches the inhabited area was less than 72% respect on the EGS and 284% respect on the SBA. SCIARA-fv2 simulations by adopting the best solutions provided

by the three different strategies are shown in Figure 7.16.

It is also worth to note that the best set of interventions provided by CCS (a set of 5 protection barriers with a maximum length of 80 m, average height of 3,5m, base width of 10 m and total volume of 6250  $m^3$ ) is approximately eighteen times more efficient than the one applied, consisting of thirteen earthen barriers, to divert the lava flow away from the facilities during the 2001 Mt. Etna eruption. Figure 7.19 show the differences and similarities between earthworks built during the real event and those provided by the genetic algorithm.

Furthermore, inspired by the slope proximity study conducted for the SBA, experiments showed also when cooperation between subsolutions can play a very important role in leading the simulation towards to efficient solutions. In particular, a GA emergent behaviour during the evolutionary process, has emerged and results showed that cooperation between barriers is more important if the morphology can be exploited, which is possible if within the Protection Measures Zone a set of disjoint clusters of high slope proximity values can be found.

Finally, the present study has produced extremely positive results and simulations have demonstrated that GAs can represent a valid tool to determine protection works construction in order to mitigate the lava flows risk.

# Chapter 8

## Conclusions

This thesis has contributed to defining a novel approach for the volcanic risk mitigation, by devising protective measures to divert lava flows. The context in which this work is located is multidisciplinary because the implemented decision support system has required the application of techniques belonging to different branches such as Computer Science, Artificial Intelligence, Physics and Geology. However, the main contribution in these contexts, was particularly significant during the phase of *parallelization*, real-time *visualization* and *optimization*, through genetic algorithms, of the adopted CA model.

The model adopted to evaluate the intermediate choices of the genetic algorithm was the latest release fv2 of the SCIARA CA model for simulating lava flows. SCIARA-fv2 (presented in Chapter 4) is a reliable CCA lava flow model, successfully applied to the simulation of many real events that have occurred on Mt.Etna. This release considers a Bingham-like rheology and introduces a square tessellation of the cellular space instead of the previously adopted hexagonal one, which was considered in the earlier versions to limit the effect of the anisotropic flow direction problem. A preliminary calibration also allowed for the reproduction of a real case of study, namely the 2001 and 2006 lava flows at Mt Etna (Italy), with a great level of accuracy. In effect, a high degree of overlapping between the real and the simulated events and a perfect fitting, in terms of run-out, were obtained.

The reduction of the overall GA execution time, due to parallel computing adoption, allowed dynamics and emergent behaviours of the GA to be better understood by running several seed tests. To evaluate a given GA individual an entire CA simulation has to be performed and, depending on the adopted computer framework, such an operation may require several hours or even months. Starting from the problem of the high computational complexity of the GA algorithm fitness evaluation, a CPU/GPU library was developed

for executing a large number of concurrent lava simulations. As discussed in Chapter 5, different strategies were implemented and evaluated. The most efficient one, based on a dynamical CUDA kernel grid, enabled the use of a high percentage of computationally active threads and for this reason was adopted to implement a *master-slave* parallel genetic algorithm in which the CPU (Master) executes the GA steps, while GPU cores evaluate the individuals fitness. The parallel speedups, attained through the proposed approaches and considering GPGPU hardware, were indeed significant. The adoption of PGAs permitted a greater number of tests to be performed in reasonable times, shortening the execution by a factor of 67. The parallel acceleration of both single and multiple CA based simulations was the object of several publications in proceedings of international conferences [67, 38] and international journals [42, 41, 55].

Understanding the meaning of huge amounts of numerical data produced by the GA without a valid tool would be impossible. In addition to the GA algorithm acceleration implementation, an interaction visualisation system was also developed for the analysis phases of the results and to better understand GA dynamics and emergent behaviors. Furthermore, the importance of this tool lies in the possibility for the user to interact with the GA model during the evolution process in order to eventually modify candidate solutions by promoting the fittest ones. The Interactive Visualization System for Lava Flows Cellular Automata Simulations using CUDA, discussed in Chapter 6, was presented at the GPGPU Technology Conference [65], the NVIDIA's annual research event.

Since GAs, as discussed in Chapter 2, have been applied with success in Geomorphology and in particular in the recent past for optimizing CA models, in this work GAs were adopted as an optimization method, in conjunction with the SCIARA-fv2 CA model for the morphological evolution of protective works to control lava flows. In order to model the GA, two fitness functions were considered. The first one,  $f_1$ , considers the *effectiveness* of the protection measure and is based on the areal comparison between the simulated event and the zone which delimitates the area that has to be protected by the lava flow. The second one,  $f_2$ , considers the *efficiency* of the solution provided by the GA because it minimizes the total volume of the protection works in order to reduce intervention costs and environmental impact.

A preliminary release of the algorithm called SBA, based on two nodes solutions, was initially considered. First observations of the GA results permitted to conjecture the presence of local optima in the search space, probably due to problem constraints. To better investigate GA dynamic characteristics, a study of nodes distribution was also conducted and a systematic exploitation of morphological characteristics by GA during the evolutionary

---

process emerged. PGAs experiments, carried out by considering the 2001 Nicolosi case-study, demonstrated that artificial barriers can successfully change the direction of lava flow in order to protect predefined point of interests. In particular, by performing extensive experiments, simulations demonstrated that protective works are more effective when placed nearly parallel to the flow direction, while a barrier placed perpendicular to the flow direction can only stop the flux temporarily, ultimately allowing for the solidified crust to accumulate, causing the following mass to go over the barrier.

Due to the strong limitation related to single barrier solutions provided by the SBA, a second GA strategy was implemented by introducing multi-barrier protection measures in order to improve the efficiency of the final solution. An Evolutionary Greedy Strategy (GES) was introduced and evaluated. Based on the iterative execution of the original algorithm (SBA), in combination with the application of dimensional constraints to the barriers until reaching a stop criteria, the use of the greedy strategy has allowed the generation of more efficient results than those of the first algorithm version (SBA). However, this strategy does not guarantee that at the end of the procedure the optimal solution is found because of the limitation related to the Greedy Algorithms and because of no presence of coevolution between the GA individuals.

For these reasons, an alternative approach, defined as Coevolutionary Cooperative Strategy (CCS), has been introduced where all barriers are encoded in the genotype and, because all the constituents parts of the solution interact with the GA environment, a mechanism of cooperation between individuals has been favored. Solutions provided by CCS were extremely efficient and, in particular, the total volume used to deviate the flow thus avoiding that the lava reaches the inhabited area was less than 72% respect to the EGS and 284% respect to the SBA. It is also worth to note that the best set of interventions provided by CCS was approximately eighteen times more efficient than the one applied to divert the lava flow away from the facilities during the 2001 Mt.Etna eruption.

Furthermore, inspired by the slope proximity study conducted for the SBA, experiments showed also when cooperation between subsolutions can play a very important role in leading the simulation towards to efficient solutions. In particular, a GA emergent behaviour during the evolutionary process, has emerged and results showed that cooperation between barriers is more important if the morphology can be exploited, which is possible if within the Protection Measures Zone a set of disjoint clusters of high slope proximity values can be found.

The present study has produced extremely positive results and simulations have demonstrated that GAs can represent a valid tool to determine

protection works construction in order to mitigate the lava flows risk.

The validity of the obtained results was also confirmed by the presentation of the methodology for mitigation of lava flow invasion hazard proposed in this thesis as refereed publication in International Conferences [66] and abstracts [112].

Because no standard optimization methodology to slow down and divert lava flows exist today, this methodology, if it is possible to determine the effectiveness on other case studies, can be adopted as a decision support system by applying interventions both during a lava episode and after a volcano risk assessment. Future work regarding the possible extensions of model parameters will consider the introduction of lava cooling by water jets.

By considering the hazard evaluation context, an important application of the methodology could be to consider as an event to be mitigated a grid of hypothetical vents defined as the source for the simulations to be carried out. In this case, protection measures provided by the GA can represent a preventive solution to assess the effect of possible human interventions.

Furthermore, it will be very important to evaluate the extension of this method to other different complex natural phenomena such as a debris flow models.

# Acknowledgments

First of all I would like to express my gratitude to my supervisors **Dr. Donato D'Ambrosio** and **Dr. William Spataro** for giving me guidance, encouragement and motivation throughout my research path.

I also wish to express my sincere appreciation to **Dr. Davide Marocco** for his advice and inspiration for this work.

I would like to thank **Prof. Angelo Cangelosi** and all the research group members of the Centre for Robotics and Neural Systems at Plymouth University.

Finally, I wish to extend my deepest gratitude to my family for their support and love.





# Bibliography

- [1] L. Abersten. Diversion of a lava flow from its natural bed to an artificial channel with the aid of explosives: Etna, 1983. *Bulletin Volcanologique*, 47(4):1165–1177, 1984.
- [2] M. V. Avolio, G. M. Crisci, S. Di Gregorio, R. Rongo, W. Spataro, and D. D’Ambrosio. Pyroclastic flows modelling using Cellular Automata. *Computers & Geosciences*, 32:897–911, 2006.
- [3] M. V. Avolio, D. D’Ambrosio, S. Gregorio, V. Lupiano, R. Rongo, W. Spataro, and G. Trunfio. Evaluating cellular automata models by evolutionary multiobjective calibration. In H. Umeo, S. Morishita, K. Nishinari, T. Komatsuzaki, and S. Bandini, editors, *Cellular Automata*, volume 5191 of *Lecture Notes in Computer Science*, pages 114–119. Springer Berlin Heidelberg, 2008.
- [4] M. V. Avolio, S. Di Gregorio, W. Spataro, and G. A. Trunfio. A theorem about the algorithm of minimization of differences for multicomponent cellular automata. In G. C. Sirakoulis and S. Bandini, editors, *ACRI*, volume 7495 of *Lecture Notes in Computer Science*, pages 289–298. Springer, 2012.
- [5] I. Azpeitia and J. Ibáñez. Spontaneous emergence of robust cellular replicators. In S. Bandini, B. Chopard, and M. Tomassini, editors, *Cellular Automata*, volume 2493 of *Lecture Notes in Computer Science*, pages 132–143. Springer Berlin Heidelberg, 2002.
- [6] F. Barberi, F. Brondi, M. Carapezza, L. Cavarra, and C. Murgia. Earthen barriers to control lava flows in the 2001 eruption of Mt. Etna. *Journal of Volcanology and Geothermal Research*, 123:231–243, 2003.
- [7] F. Barberi, M. Carapezza, M. Valenza, and L. Villari. The control of lava flow during the 1991-1992 eruption of Mt. Etna. *Journal of Volcanology and Geothermal Research*, 56:1–34, 1993.

- [8] D. Barca, G. M. Crisci, S. Di Gregorio, and F. Nicoletta. Cellular Automata for simulating lava flows: a method and examples of the etnean eruptions. *Transport Theory and Statistical Physics*, 23(1-3):195–232, 1994.
- [9] M. Bashiri, A. Amiri, M. H. Doroudyan, and A. Asgari. Multi-objective genetic algorithm for economic statistical design of control chart. *Scientia Iranica*, 20(3):909 – 918, 2013.
- [10] B. Behncke and M. Neri. The July-August 2001 eruption of Mt. Etna (Sicily). *Bulletin of Volcanology*, 65(7):461–476, 2003.
- [11] E. G. Bekele and J. W. Nicklow. Multi-objective automatic calibration of SWAT using NSGA-II. *Journal of Hydrology*, 341(34):165 – 176, 2007.
- [12] P. Bentley. An introduction to evolutionary design by computers. In P. J. Bentley, editor, *Evolutionary Design by Computers*, chapter 1, pages 1–73. Morgan Kaufman, San Francisco, USA, 1999.
- [13] N. Beume, B. Naujoks, and M. Emmerich. Sms-emoa: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181(3):1653 – 1669, 2007.
- [14] E. Bilotta, A. Lafusa, and P. Pantano. Is self-replication an embedded characteristic of artificial/living matter? In *Proceedings of the eighth international conference on Artificial life*, ICAL 2003, pages 38–48, Cambridge, MA, USA, 2003. MIT Press.
- [15] G. Bilotta, E. Rustico, A. Hrault, A. Vicari, G. Russo, C. D. Negro, and G. Gallo. Porting and optimizing MAGFLOW on CUDA. *Annals of Geophysics*, 54(5):580–591, 2011.
- [16] T. Blickle and L. Thiele. A mathematical analysis of tournament selection. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 9–16. Morgan Kaufmann, 1995.
- [17] J. Bongard. Morphological change in machines accelerates the evolution of robust behavior. In *Proceedings of the National Academy of Sciences*, volume 108, pages 1234–1239, Jan. 2011.
- [18] P. Bouvry, F. Seredyński, and A. Zomaya. Application of cellular automata for cryptography. In R. Wyrzykowski, J. Dongarra, M. Paprzycki, and J. Waniewski, editors, *Parallel Processing and Applied Math-*

- ematics*, volume 3019 of *Lecture Notes in Computer Science*, pages 447–454. Springer Berlin Heidelberg, 2004.
- [19] J. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.
- [20] D. Brockhoff and E. Zitzler. Objective Reduction in Evolutionary Multiobjective Optimization: Theory and Applications. *Evolutionary Computation*, 17(2):135–166, 2009.
- [21] T. C. Chang and Y. H. Chien. The application of genetic algorithm in debris flows prediction. *Environmental Geology*, 53(2):339–347, 2007.
- [22] B. Chapman, G. Jost, and R. van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming*. The MIT Press, Oct. 2007.
- [23] H. Chaté and P. Manneville. Criticality in cellular automata. *Physica D: Nonlinear Phenomena*, 45(13):122 – 135, 1990.
- [24] B. Chopard. Cellular automata modeling of physical systems. In R. A. Meyers, editor, *Computational Complexity*, pages 407–433. Springer New York, 2012.
- [25] B. Chopard and M. Droz. *Cellular Automata Modeling of Physical Systems*. Cambridge University Press, 1998.
- [26] H. H. Chou and J. A. Reggia. Emergence of self-replicating structures in a cellular automata space. *Physica D*, 110:252–276, 1997.
- [27] E. F. Codd. *Cellular Automata*. Academic Press, Inc., Orlando, FL, USA, 1968.
- [28] C. Coello Coello, G. Lamont, and D. van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Genetic and Evolutionary Computation. Springer, Berlin, Heidelberg, 2nd edition, 2007.
- [29] R. Colombrita. Methodology for the construction of earth barriers to divert lava flows: The Mt. Etna 1983 eruption. *Bulletin Volcanologique*, 47(4):1009–1038, 1984.
- [30] D. W. Corne, N. R. Jerram, J. D. Knowles, M. J. Oates, and M. J. PESA-II: Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2001)*, pages 283–290. Morgan Kaufmann Publishers, 2001.

- [31] D. W. Corne, J. D. Knowles, and M. J. Oates. The pareto envelope-based selection algorithm for multiobjective optimization. In *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 839–848. Springer, 2000.
- [32] P. Cozzi and C. Riccio. *OpenGL Insights*. CRC Press, 2012. url-<http://www.openglinsights.com/>.
- [33] G. Crisci, S. Di Gregorio, R. Rongo, W. Spataro, and F. Nicoletta. Analysing lava risk for the etnean area: Simulation by cellular automata methods. *Natural Hazards*, 20(2-3):215–229, 1999.
- [34] G. M. Crisci, M. V. Avolio, B. Behncke, D. D’Ambrosio, S. Di Gregorio, V. Lupiano, M. Neri, R. Rongo, and W. Spataro. Predicting the impact of lava flows at Mount Etna, Italy. *Journal of Geophysical Research: Solid Earth*, 115(B4):n/a–n/a, 2010.
- [35] G. M. Crisci and S. R. G. Di Gregorio. A cellular space model of basaltic lava flow. In *International AMSE Conference Modelling & Simulation*, HPG ’10, pages 65–67, Aire-la-Ville, Switzerland, 1982. Group 11 ”Terrestrial resources and phenomena”.
- [36] G. M. Crisci, R. Rongo, S. Di Gregorio, and W. Spataro. The simulation model SCIARA: the 1991 and 2001 lava flows at Mount Etna. *Journal of Volcanology and Geothermal Research*, 132(23):253 – 267, 2004.
- [37] J. P. Crutchfield, M. Mitchell, and R. Das. The evolutionary design of collective computation in cellular automata. In *MACHINE LEARNING JOURNAL*, 1998.
- [38] D. D’Ambrosio, S. Di Gregorio, G. Filippone, R. Rongo, W. Spataro, and G. A. Trunfio. Fast assessment of wildfire spatial hazard with GPGPU. In *SIMULTECH’12*, pages 260–269, 2012.
- [39] D. D’Ambrosio, S. Di Gregorio, S. Gabriele, and R. Gaudio. A cellular automata model for soil erosion by water. *Physics and Chemistry of the Earth, Part B: Hydrology, Oceans and Atmosphere*, 26(1):33 – 39, 2001.
- [40] D. D’Ambrosio, S. Di Gregorio, and G. Iovine. Simulating debris flows through a hexagonal cellular automata model: SCIDDICA  $s_{3hex}$ . *Natural Hazards and Earth System Science*, 3(6):545–559, 2003.

- [41] D. D'Ambrosio, G. Filippone, D. Marocco, R. Rongo, and W. Spataro. Efficient application of GPGPU for lava flow hazard mapping. *The Journal of Supercomputing*, 65(2):630–644, 2013.
- [42] D. D'Ambrosio, G. Filippone, R. Rongo, W. Spataro, and G. A. Trunfio. Cellular automata and GPGPU: an application to lava flow modeling. *International Journal of Grid and High Performance Computing*, 4(3):30–47, July 2012.
- [43] D. D'Ambrosio, R. Rongo, W. Spataro, and G. Trunfio. Optimizing cellular automata through a meta-model assisted memetic algorithm. In *Proceedings of Parallel Problem Solving from Nature - PPSN XII*, volume 7492 of *Lecture Notes in Computer Science*, pages 317–326. Springer Berlin Heidelberg, 2012.
- [44] D. D'Ambrosio, R. Rongo, W. Spataro, and G. A. Trunfio. Meta-model assisted evolutionary optimization of cellular automata: an application to the SCIARA model. In *Proceedings of the 9th international conference on Parallel Processing and Applied Mathematics - Volume Part II, PPAM'11*, pages 533 – 542, Berlin, Heidelberg, 2012. Springer-Verlag.
- [45] D. D'Ambrosio and W. Spataro. Parallel evolutionary modelling of geological processes. *Journal of Parallel Computing*, 33(3):186–212, 2007.
- [46] D. D'Ambrosio, W. Spataro, and G. Iovine. Parallel genetic algorithms for optimising cellular automata models of natural complex phenomena: An application to debris flows. *Computers & Geosciences*, 32(7):861–875, 2006.
- [47] K. De Jong. Evolutionary computation: a unified approach. In *Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, GECCO '08, pages 2245–2258, New York, NY, USA, 2008.
- [48] K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, Ann Arbor, MI, USA, 1975.
- [49] M. De La Asunción, J. M. Mantas, M. J. Castro, and E. D. Fernández-Nieto. An MPI-CUDA implementation of an improved Roe method for two-layer shallow water systems. *Journal of Parallel and Distributed Computing*, 72(9):1065–1072, 2012.

- [50] M. de Menezes, E. Brigatti, and V. Schwämmle. Evolving cellular automata for diversity generation and pattern recognition: deterministic versus random strategy. *Journal of Statistical Mechanics: Theory and Experiment*, 2013(08):P08006, 2013.
- [51] K. Deb and D. Kalyanmoy. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc., New York, NY, USA, 2001.
- [52] C. Del Negro, L. Fortuna, A. Herault, and A. Vicari. Simulations of the 2004 lava flow at Etna volcano using the MAGFLOW cellular automata model. *Bulletin of Volcanology*, 70(7):805–812, 2008.
- [53] D. D’Humières, A. Clouqueur, and P. Lallemand. Lattice gases and parallel processors. *CALCOLO*, 25(1-2):129–151, 1988.
- [54] D. D’Humières, P. Lallemand, and U. Frisch. Lattice Gas Models for 3D Hydrodynamics. *EPL (Europhysics Letters)*, 2(4):291+, Aug. 1986.
- [55] S. Di Gregorio, G. Filippone, W. Spataro, and G. A. Trunfio. Accelerating wildfire susceptibility mapping through GPGPU. *Journal of Parallel and Distributed Computing*, 73(8):1183 – 1194, 2013.
- [56] S. Di Gregorio, R. Rongo, W. Spataro, G. Spezzano, and D. Talia. High performance scientific computing by a parallel cellular environment. *Future Generation Computer Systems*, 12(5):357–369, 1997.
- [57] S. Di Gregorio and R. Serra. An empirical method for modelling and simulating some complex macroscopic phenomena by cellular automata. *Future Generation Computer Systems*, 16(2-3):259–271, 1999.
- [58] S. Di Gregorio, R. Serra, and M. Villani. Applying cellular automata to complex environmental problems: The simulation of the bioremediation of contaminated soils. *Theoretical Computer Science*, 217(1):131 – 156, 1999.
- [59] S. Di Gregorio and G. Trautteur. On reversibility in cellular automata. *Journal of Computer and System Sciences*, 11(3):382 – 391, 1975.
- [60] C. Dibben. Leaving the city for the suburbs - The dominance of ‘ordinary’ decision making over volcanic risk perception in the production of volcanic risk on Mt. Etna, Sicily. *Journal of Volcanology and Geothermal Research*, 172(7):288–299, 2008.

- [61] 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*, pages 216–222, 2012.
- [62] J. Du, C. Wang, and F. Zhang. Multi-objective optimization of bus dispatching based on improved genetic algorithm. In *Proceedings of the 2011 Seventh International Conference on Computational Intelligence and Security*, CIS '11, pages 106–109, Washington, DC, USA, 2011.
- [63] A. ElSayed, E. Kongar, S. Gupta, and T. Sobh. A Robotic-Driven Disassembly Sequence Generator for End-Of-Life Electronic Products. *Journal of Intelligent & Robotic Systems*, 68(1):43–52, 2012.
- [64] J. Fietz, M. J. Krause, C. Schulz, P. Sanders, and V. Heuveline. Optimized hybrid parallel lattice boltzmann fluid flow simulations on complex geometries. In Kaklamani et al. [92], pages 818–829.
- [65] G. Filippone, W. Spataro, D. D’Ambrosio, and D. Marocco. An interactive Visualization System for Lava Flows Cellular Automata Simulations using CUDA, 2013. GPU Technology Conference (Poster), San Jose, California.
- [66] G. Filippone, W. Spataro, D. D’Ambrosio, and D. Marocco. A new methodology for mitigation of lava flow invasion hazard: Morphological evolution of protective works by parallel genetic algorithms. In *Proceedings of the 5th International Conference on Evolutionary Computation Theory and Applications (ECTA)*, 2013.
- [67] G. Filippone, W. Spataro, G. Spingola, D. D’Ambrosio, R. Rongo, G. Perna, and S. Di Gregorio. GPGPU programming and cellular automata: Implementation of the SCIARA lava flow simulation code. In *23rd European Modeling and simulation Symposium (WMSS)*, Rome, Italy, 12-14 September 2011.
- [68] M. Flynn. Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers*, 21:948–960, 1972.
- [69] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley, 1966.
- [70] G. Folino, G. Mendicino, A. Senatore, G. Spezzano, and S. Straface. A model based on cellular automata for the parallel simulation of 3D unsaturated flow. *Parallel Computing*, 32(5-6):357–376, 2006.



- [71] C. M. Fonseca and P. J. Fleming. Genetic algorithms for multiobjective optimization: Formulation discussion and generalization. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 416–423, San Francisco, CA, USA, 1993.
- [72] E. Fujita, M. Hidaka, A. Goto, and S. Umino. Simulations of measures to control lava flows. *Bulletin of Volcanology*, 71:401–408, 2009.
- [73] M. Gardner. Mathematical games: The fantastic combinations of John Conway’s new solitaire game ‘Life’. *Scientific American*, 223(4):120–123, 1970.
- [74] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.
- [75] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 41–49, Hillsdale, NJ, USA, 1987.
- [76] J. F. Gonçalves and M. G. Resende. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17(5):487–525, 2011.
- [77] H. A. Gutowitz. A hierarchical classification of cellular automata. *Physica D: Nonlinear Phenomena*, 45(13):136 – 156, 1990.
- [78] J. Handl and J. Knowles. An evolutionary approach to multiobjective clustering. *IEEE Transactions on Evolutionary Computation*, 11(1):56–76, 2007.
- [79] J. Hardy, Y. Pomeau, and G. de Pazzis. Thermodynamics and hydrodynamics for a modeled fluid. *Journal of Mathematical Physics*, 13(5):1949–1961, 1976.
- [80] F. J. Higuera and J. Jiménez. Boltzmann approach to lattice gas simulations. *Europhysics Letters*, 9(7):663–668, 1989.
- [81] G. E. Hinton and S. J. Nowlan. How learning can guide evolution. *Complex Systems*, pages 495–502, 1987.
- [82] J. H. Holland. *Nonlinear environments permitting efficient adaptation*. Computer and Information Sciences II. New York: Academic, 1967.

- [83] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, USA, 1975.
- [84] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, 1992.
- [85] J. Horn, N. Nafpliotis, N. Nafpliotis, D. E. Goldberg, and D. E. Goldberg. A niched Pareto genetic algorithm for multiobjective optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, pages 82–87, 1994.
- [86] E. Innocenti, X. Silvani, A. Muzy, and D. R. C. Hill. A software framework for fine grain parallelization of cellular models with OpenMP: Application to fire spread. *Environmental Modelling and Software*, 24(7):819–831, 2009.
- [87] G. Iovine, D. D’Ambrosio, and S. Di Gregorio. Applying genetic algorithms for calibrating a hexagonal cellular automata model for the simulation of debris flows characterised by strong inertial effects. *Geomorphology*, 66(14):287 – 303, 2005.
- [88] K. Ishihara, M. Iguchi, and K. Kamo. Numerical Simulation of Lava Flows on Some Volcanoes in Japan. In J. H. Fink, editor, *Lava Flows and Domes*, volume 2 of *IAVCEI Proceedings in Volcanology*, pages 174–207. Springer Berlin Heidelberg, 1990.
- [89] M. Jähne, X. Li, and J. Branke. Evolutionary algorithms and multi-objectivization for the travelling salesman problem. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation, GECCO ’09*, pages 595–602, New York, NY, USA, 2009.
- [90] M. Jensen. Helper-objectives: Using multi-objective evolutionary algorithms for single-objective optimisation. *Journal of Mathematical Modelling and Algorithms*, 3(4):323–347, 2004.
- [91] F. Jiménez-Morales. An evolutionary approach to the study of non-trivial collective behavior in cellular automata. In S. Bandini, B. Chopard, and M. Tomassini, editors, *Cellular Automata*, volume 2493 of *Lecture Notes in Computer Science*, pages 32–43. Springer Berlin Heidelberg, 2002.

- [92] C. Kaklamanis, T. S. Papatheodorou, and P. G. Spirakis, editors. *Euro-Par 2012 Parallel Processing - 18th International Conference, Euro-Par 2012, Rhodes Island, Greece, August 27-31, 2012. Proceedings*, volume 7484 of *Lecture Notes in Computer Science*. Springer, 2012.
- [93] B. Kar, D. C. Rao, and A. K. Rath. Generating PNS for Secret Key Cryptography Using Cellular Automaton. *International Journal of Advanced Computer Science and Applications*, 2(5):101–105, 2011.
- [94] J. Kari. Reversibility of 2d cellular automata is undecidable. *Physica D: Nonlinear Phenomena*, 45(13):379 – 385, 1990.
- [95] S.-T. Khu, H. Madsen, and F. di Pierro. Incorporating multiple observations for distributed hydrologic model calibration: An approach using a multi-objective evolutionary algorithm and clustering. *Advances in Water Resources*, 31(10):1387 – 1398, 2008.
- [96] R. Kicinger, T. Arciszewski, and K. D. De Jong. Evolutionary computation and structural design: A survey of the state-of-the-art. *Comput. Struct.*, 83(23-24):1943–1978, 2005.
- [97] L. Kier and P. Seybold. Cellular automata modeling of complex biochemical systems. In R. A. Meyers, editor, *Encyclopedia of Complexity and Systems Science*, pages 848–865. Springer New York, 2009.
- [98] J. Knowles, R. Watson, and D. Corne. Reducing local optima in single-objective problems by multi-objectivization. In E. Zitzler, L. Thiele, K. Deb, C. Coello Coello, and D. Corne, editors, *Evolutionary Multi-Criterion Optimization*, volume 1993 of *Lecture Notes in Computer Science*, pages 269–283. Springer Berlin Heidelberg, 2001.
- [99] J. D. Knowles and D. W. Corne. Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172, 2000.
- [100] J. M. V. A. Koelman. A Simple Lattice Boltzmann Scheme for Navier-Stokes Fluid Flow. *EPL (Europhysics Letters)*, 15(6):603, 1991.
- [101] J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, USA, 1992.
- [102] Kronos. Kronos opencl website (<http://www.kronos.org/opencl/>), 2013.

- [103] W. H. Kruskal and W. A. Wallis. Use of Ranks in One-Criterion Variance Analysis. *Journal of the American Statistical Association*, 47(260):583–621, 1952.
- [104] V. Kumar. *Introduction to Parallel Computing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.
- [105] C. G. Langton. Self-reproduction in cellular automata. *Physica D*, 10D(1-2):135–44, 1984.
- [106] C. G. Langton. Studying artificial life with cellular automata. *Physica D*, 2(1-3):120–149, Oct. 1986.
- [107] C. G. Langton. Computation at the edge of chaos: phase transitions and emergent computation. *Physica D*, 42(1-3):12–37, 1990.
- [108] H. Lipson and J. B. Pollack. Automatic Design and Manufacture of Artificial Lifeforms. *Nature*, 406:974–978, 2000.
- [109] G. A. MacDonald. The 1959 and 1960 eruptions of Kilauea volcano, Hawaii, and the construction of walls to restrict the spread of the lava flows. *Bulletin Volcanologique*, 24(1):249–294, 1962.
- [110] H. Madsen. Parameter estimation in distributed hydrological catchment modelling using automatic calibration with multiple objectives. *Advances in Water Resources*, 26(2):205–216, 2003.
- [111] P. Maji, N. Ganguly, S. Saha, A. Roy, and P. Chaudhuri. Cellular automata machine for pattern recognition. In S. Bandini, B. Chopard, and M. Tomassini, editors, *Cellular Automata*, volume 2493 of *Lecture Notes in Computer Science*, pages 270–281. Springer Berlin Heidelberg, 2002.
- [112] D. Marocco, W. Spataro, D. D’Ambrosio, G. Filippone, R. Rongo, G. Iovine, and M. Neri. Morphological evolution of protective works by Genetic Algorithms: An application to Mt Etna. In *EGU General Assembly Conference Abstracts*, volume 15, page 11227, 2013.
- [113] A. R. McBirney and T. Murase. Rheological properties of magmas. *Annual Review of Earth and Planetary Sciences*, 12(1):337–357, 1984.
- [114] H. V. McIntosh. Wolfram’s class IV automata and a good life. *Physica D: Nonlinear Phenomena*, 45(13):105–121, 1990.

- [115] G. R. McNamara and G. Zanetti. Use of the Boltzmann equation to simulate lattice-gas automata. *Physical Review Letters*, 61(20):2332–2335, 1988.
- [116] M. Mitchell. *An introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1996.
- [117] H. Miyamoto and S. Sasaki. Simulating lava flows by an improved cellular automata method. *Computers & Geosciences*, 23(3):283–292, 1997.
- [118] E. F. Moore. Machine models of self-reproduction. In *Mathematical Problems in Biological Sciences (Proceedings of Symposia in Applied Mathematics)*. American Mathematical Society, 1962.
- [119] J. Morshed and J. J. Kaluarachchi. Application of artificial neural network and genetic algorithm in flow and transport simulations. *Advances in Water Resources*, 22(2):145–158, 1998.
- [120] M. K. Muleta and J. W. Nicklow. Evolutionary algorithms for multiobjective evaluation of watershed management decisions. *Journal of Hydroinformatics*, 4(2)(5):83–97, 2011.
- [121] J. Myhill. The converse of Moores Garden-of-Eden theorem. *Proceedings of The American Mathematical Society*, 14:685–685, 1963.
- [122] M. Neri, B. Behncke, M. Burton, G. Galli, S. Giammanco, E. Pecora, E. Privitera, and D. Reitano. Continuous soil radon monitoring during the july 2006 etna eruption. *Geophysical Research Letters*, 33(24), 2006.
- [123] J. Nicklow, O. Ozkurt, and J. Bringer, JohnA. Control of channel bed morphology in large-scale river networks using a genetic algorithm. *Water Resources Management*, 17(2):113–132, 2003.
- [124] S. Nolfi and D. Marocco. Evolving robots able to integrate sensory-motor information over time. *Theory in Biosciences*, 120:287–310, 2001.
- [125] NVIDIA Corporation. *CUDA C Best Practices Guide*. NVIDIA Corporation, 2701 San Tomas Expressway, Santa Clara 95050, USA, 5.0 edition, October 2012.
- [126] NVIDIA Corporation. *CUDA C Programming Guide*. NVIDIA Corporation, 2701 San Tomas Expressway, Santa Clara 95050, USA, 5.0 edition, October 2012.

- [127] M. Oliverio, W. Spataro, D. D'Ambrosio, R. Rongo, G. Spingola, and G. A. Trunfio. OpenMP parallelization of the SCIARA Cellular Automata lava flow model: performance analysis on shared-memory computers. *Procedia CS*, 4:271–280, 2011.
- [128] V. Pareto. *Cours d'Economie Politique*. Droz, Genève, 1896.
- [129] A. Piwonska, F. Sereczynski, and M. Szaban. Learning cellular automata rules for binary classification problem. *The Journal of Supercomputing*, 63(3):800–815, 2013.
- [130] T. Preis. Gpu-computing in econophysics and statistical physics. *The European Physical Journal Special Topics*, 194(1):87–119, 2011.
- [131] A. Prugel-Bennett and J. L. Shapiro. The dynamics of a genetic algorithm for simple random ising systems. *Physica D*, 104:75–114, 1996.
- [132] B. Priego, D. Souto, F. Bellas, and R. J. Duro. Hyperspectral image segmentation through evolved cellular automata. *Pattern Recognition Letters*, 34(14):1648–1658, 2013.
- [133] A. Prugel-Bennett and J. L. Shapiro. An analysis of genetic algorithms using statistical mechanics. *Physica D*, 104:75–114, 1997.
- [134] Y. H. Qian, D. D'Humières, and P. Lallemand. Lattice BGK models for Navier-Stokes equation. *EPL (Europhysics Letters)*, 17:479, 1992.
- [135] G. J. E. Rawlins. Foundations of genetic algorithms. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*. San Francisco, CA: Morgan Kaufmann, 1991.
- [136] I. Rechenberg. *Evolutionsstrategie : Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Number 15 in Problematika. Frommann-Holzboog, Stuttgart-Bad Cannstatt, 1973.
- [137] M. Roberts, J. Packer, M. C. Sousa, and J. R. Mitchell. A work-efficient GPU algorithm for level set segmentation. In *Proceedings of the Conference on High Performance Graphics, HPG '10*, pages 123–132, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.
- [138] A. Rogers and A. Prugel-Bennett. The dynamics of a genetic algorithm on a model hard optimization problem. *Complex Systems*, 11:437–464, 1997.

- [139] A. Roli and F. Zambonelli. Emergence of macro spatial structures in dissipative cellular automata. In *In Proceedings of Cellular Copyright 2007*, pages 144–155. John Wiley & Sons, 2002.
- [140] R. Rongo, M. V. Avolio, B. Behncke, D. D’Ambrosio, S. Di Gregorio, V. Lupiano, M. Neri, W. Spataro, and G. M. Crisci. Defining High Detailed Hazard Maps by a Cellular Automata approach: Application to Mt. Etna (Italy). *Annals of Geophysics*, 54(5):568–578, 2011.
- [141] R. Rongo, W. Spataro, D. D’Ambrosio, M. V. Avolio, G. A. Trunfio, and S. Di Gregorio. Lava flow hazard evaluation through cellular automata and genetic algorithms: an application to Mt Etna volcano. *Fundamenta Informaticae*, 87:247–267, 2008.
- [142] P. L. Rosin. Training Cellular Automata for Image Processing. In H. Kalviainen, J. Parkkinen, and A. Kaarna, editors, *Image Analysis*, volume 3540 of *Lecture Notes in Computer Science*, pages 195–204. Springer Berlin Heidelberg, 2005.
- [143] P. L. Rosin. Image processing using 3-state cellular automata. *Computer Vision and Image Understanding*, 114(7):790 – 802, 2010.
- [144] G. Rudolph and A. Agapie. Convergence properties of some multi-objective evolutionary algorithms. In *Congress on Evolutionary Computation (CEC 2000)*, pages 1010–1016. IEEE Press, 2000.
- [145] T. Salles, S. Lopez, M. Cacas, and T. Mulder. Cellular automata model of density currents. *Geomorphology*, 88(12):1 – 20, 2007.
- [146] J. D. Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 93–100, Hillsdale, NJ, USA, 1985.
- [147] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit, 4th Edition*. Kitware Inc., 2006.
- [148] F. Serebinski, P. Bouvry, and A. Y. Zomaya. Cellular automata computations and secret key cryptography. *Parallel Computing*, 30(5):753–766, 2004.
- [149] K. Sims. Evolving 3D morphology and behavior by competition. In *Proceedings of Artificial Life IV*, pages 28–39. MIT Press, 1994.

- [150] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI: The complete reference*. MIT Press, Cambridge, MA, 1996.
- [151] W. Spataro, M. V. Avolio, V. Lupiano, G. A. Trunfio, R. Rongo, and D. D'Ambrosio. The latest release of the lava flows simulation model SCIARA: First application to Mt Etna (Italy) and solution of the anisotropic flow direction problem on an ideal surface. In *International Conference on Computational Science*, pages 17–26, 2010.
- [152] W. Spataro, D. D'Ambrosio, R. Rongo, and G. Trunfio. An Evolutionary Approach for Modelling Lava Flows through Cellular Automata. In *ACRI 2004*, volume 3305 of *Lecture Notes in Computer Science*, pages 725–734. Springer Berlin Heidelberg, 2004.
- [153] G. Spingola, D. D'Ambrosio, W. Spataro, R. Rongo, and G. Zito. Modeling complex natural phenomena with the libautoti cellular automata library: An example of application to lava flows simulation. In *PDPTA'08*, pages 277–283, 2008.
- [154] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2:221–248, 1994.
- [155] S. Succi. *Automati cellulari. Una nuova frontiera del calcolo scientifico*. Collana informatica domani / IBM SEMEA. Franco Angeli, 1991.
- [156] S. Succi. *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*. Clarendon Press, Oxford, 2001.
- [157] D. Talia. The role of parallel cellular programming in computational science. In J. Palma, J. Dongarra, and V. Hernandez, editors, *Vector and Parallel Processing VECPAR 2000*, volume 1981 of *Lecture Notes in Computer Science*, pages 207–220. Springer Berlin Heidelberg, 2001.
- [158] J. Thatcher. *Universality in the Von Neumann Cellular Model*. Ethic-sSA research report. IBM Watson Research Center, 1964.
- [159] T. Toffoli and N. Margolus. *Cellular automata machines: a new environment for modeling*. MIT Press, Cambridge, MA, USA, 1987.
- [160] T. Toffoli and N. Margolus. Invertible cellular automata: A review. *Physica D*, 45:229–253, 1990.



- [161] J. Tölke. Implementation of a Lattice Boltzmann kernel using the Compute Unified Device Architecture developed by NVIDIA. *Computing and Visualization in Science*, 13(1):29–39, 2010.
- [162] J. Tölke and M. Krafczyk. TeraFLOP computing on a desktop PC with GPUs for 3D CFD. *International Journal of Computational Fluid Dynamics*, 22(7):443–456, 2008.
- [163] M. Tomassini and M. Perrenoud. Cryptography with cellular automata. *Applied Soft Computing*, 1(2):151–160, 2001.
- [164] M. Tomassini and M. Venzi. Artificially evolved asynchronous cellular automata for the density task. In *Proceedings of the fifth International Conference on Cellular Automata for Research and Industry.*, volume 2493 of *Lecture Notes in Computer Science*, pages 44–55. Springer Berlin Heidelberg, 2002.
- [165] G. A. Trunfio, D. D’Ambrosio, R. Rongo, W. Spataro, and S. Di Gregorio. A new algorithm for simulating wildfire spread through cellular automata. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 22(1):6, 2011.
- [166] A. Vicari, H. Alexis, C. Del Negro, M. Coltelli, M. Marsella, and C. Proietti. Modeling of the 2001 lava flow at etna volcano by a cellular automata approach. *Environmental Modelling & Software*, 22(10):1465–1471, Oct. 2007.
- [167] J. Von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.
- [168] J. Weimar. *Simulation with Cellular Automata*. Logos-Verlag, Berlin, 1998.
- [169] L. Whitley. *Foundations of Genetic Algorithms 2*. Morgan Kaufmann, San Francisco, 1993.
- [170] S. Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55(3):601–644, 1983.
- [171] S. Wolfram. *A New Kind of Science*. Wolfram Media, 2002.
- [172] C. Xu, X. Deng, L. Zhang, Y. Jiang, W. Cao, J. Fang, Y. Che, Y. Wang, and W. Liu. Parallelizing a High-Order CFD Software for 3D, Multi-block, Structural Grids on the TianHe-1A Supercomputer.

- In J. Kunkel, T. Ludwig, and H. Meuer, editors, *Supercomputing*, volume 7905 of *Lecture Notes in Computer Science*, pages 26–39. Springer Berlin Heidelberg, 2013.
- [173] M. Zamith, J. R. da Silva Jr, M. Joselli, E. Clua, and E. Soluri. An Approach for Traffic Forecast with GPU Computing & Cellular Automata Model. In *Proceedings of the GPU Forum 2012 (CSBC)*. SBC, 2012.
- [174] E. Zitzler, D. Brockhoff, and L. Thiele. The Hypervolume Indicator Revisited: On the Design of Pareto-compliant Indicators Via Weighted Integration. In S. Obayashi et al., editors, *Conference on Evolutionary Multi-Criterion Optimization (EMO 2007)*, volume 4403 of *LNCS*, pages 862–876, Berlin, 2007. Springer.
- [175] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In K. C. Giannakoglou, D. T. Tsahalis, J. Périaux, K. D. Papailiou, and T. Fogarty, editors, *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, pages 95–100, Athens, Greece, 2001.
- [176] E. Zitzler and L. Thiele. An Evolutionary Algorithm for Multiobjective Optimization: The Strength Pareto Approach, 1998. TIK-Report.
- [177] W. Zuo and Q. Chen. Da rivedere. *Building and Environment*, 45(5-6):747–757, 2010.



# List of Figures

2.1	Example of fitness landscape for a binary genetic algorithm. . . . .	9
2.2	Example of proportional selection. . . . .	10
2.3	Example of crossover and mutation. . . . .	11
2.4	Example of single-point crossover between two trees. . . . .	14
3.1	Example of cellular spaces . . . . .	25
3.2	Example of one dimensional neighborhood with square tessellation	26
3.3	Example of two dimensional neighborhood with square and exagonal tessellations . . . . .	27
3.4	Example of one-dimensional CA with periodic boundary con- ditions . . . . .	28
3.5	Examples of CA belonging to the Wolfram four complexity classes . . . . .	30
3.6	Examples of CA at the edge of chaos . . . . .	32
3.7	Example of LGA on a square lattice . . . . .	35
3.8	Simulation of a flow around a thin plate with the BGK model	37
4.1	Example of application of the Minimization Algorithm of the Differences. . . . .	44
4.2	Reference schema for cells altitude determination in the Moore neighbourhood . . . . .	47
4.3	Cases in which the generic neighbour is not eliminated by the Minimisation Algorithm of the Difference . . . . .	48
4.4	Simulation of the 2006 Etnean lava flow by the CA model SCIARA . . . . .	51
4.5	SCIARA simulations on an octagonal-base pyramid . . . . .	52
4.6	Comparison between observed and simulated event of the 2001 Mt. Etna eruption . . . . .	53
5.1	Grid of thread blocks . . . . .	59
5.2	Memory architecture of a GPU . . . . .	61

5.3	Memory mapping of the CA space allocated in global memory with a portion of shared memory . . . . .	67
5.4	Example of dynamic RGBB expansion . . . . .	69
5.5	Example of RGBB mapping into a CUDA grid . . . . .	70
5.6	Speed up results on different strategy approaches and devices .	73
5.7	Elapsed time as a function of simultaneous lava events using the WCSI approach . . . . .	79
5.8	Elapsed time as a function of simultaneous lava events using the DGI approach . . . . .	82
5.9	Mapping of the CA transition function into a CUDA grid of threads . . . . .	83
5.10	Lava-flow invasion scenario at Mount Etna . . . . .	84
6.1	CCAFramework System Overview . . . . .	86
6.2	SCIARA Software screenshot . . . . .	87
6.3	Double visualisation system of CCAFramework . . . . .	88
6.4	Cell Picking tool . . . . .	89
7.1	Effusion rates at the main vents of the 2001 eruption of Mt Etna	93
7.2	Set of interventions carried out during the 2001 eruption event at Mt Etna . . . . .	94
7.3	Barriers encoding into a GA genotype . . . . .	97
7.4	Representation of GA mutation phase . . . . .	98
7.5	Elapsed time as a function of simultaneous lava events using the WCSI and DGI . . . . .	100
7.6	Temporal evolution of the composite $f_3$ fitness . . . . .	101
7.7	Temporal evolution of average fitness $f_1$ and $f_2$ of whole population . . . . .	102
7.8	Nodes distribution of the best 100 solutions generated by the GA . . . . .	103
7.9	SCIARA-fv2 simulation visualization adopting the SBA GA best solution . . . . .	104
7.10	Temporal evolution of average slope proximity values for the best individuals . . . . .	105
7.11	Tree representation of an evolutionary greedy strategy algorithm run . . . . .	107
7.12	$f_2$ fitness value in the variation of dimensional constraints associated to the single barrier with EGS . . . . .	109
7.13	Example of n-point crossover between two genotypes . . . . .	110
7.14	Temporal evolution of lifecycle value . . . . .	111
7.15	$f_2$ build cost fitness function evolution . . . . .	113

---

7.16	SCIARA-fv2 simulations adopting the best solution evolved with three different strategies . . . . .	114
7.17	Total CCS volume solutions variation on the vary of number of barriers adopted to codify the final solution . . . . .	115
7.18	Best fitness obtained by each algorithm by varying the number of barriers . . . . .	117
7.19	Comparison between solutions adopted in 2001 Mt.Etna lava and those provided by GA . . . . .	119



# List of Tables

2.1	Comparison between binary code and gray code representations	13
4.1	List of SCIARA-fv2 parameters with values related to the simulation of the 2006 Etnean lava flows . . . . .	45
5.1	Number of memory accesses performed by each SCIARA kernel to substate matrixes . . . . .	66
5.2	Major characteristics of adopted GPGPU hardware for all carried out experiments . . . . .	71
5.3	Elapsed times (in seconds) for the computation of 2001 lava flow event by adopting different block dimensions . . . . .	72
7.1	Dimensions of the four largest barriers built to control lava flows	95
7.2	List of parameters of the adopted GA . . . . .	99
7.3	Dimensions of the ten best barriers carried out by GA run . .	103
7.4	Dimensions of the best barriers evolved after the EGS GA runs	108
7.5	Total volume used to protect the Safety Zone, by varying number of barriers into the final solution and dimensional constraint for each single barrier . . . . .	112
7.6	Table of pairwise comparisons between different strategies adopted . . . . .	117