Parallel complex systems simulation

Schoneveld, A.

Citation for published version (APA):
Chapter 1

Introduction

“No one, it has been said, is better at taking a puzzle apart than the person who put it together and no one is better at putting a puzzle together than the one who took it apart”


1.1 Complex Systems simulation

Until now much experimental simulation work has been done in physics and chemistry on parallel computing systems. However, the development of formal models, suitable for modeling complex systems from nature and for efficiently mapping a complex system onto a parallel computing system, is still in its infancy. The development of techniques, suitable for modeling complex systems stemming from physics, chemistry, and biology, and mapping these onto parallel platforms, is one of the grand challenges of future research.

Many fundamental problems from natural sciences deal with complex systems. We define a complex system as a population of unique elements with well defined attributes. In the case that these elements have non-linear interactions in the temporal and spatial evolution of the system, complex macroscopic behavior can emerge from these microscopic interactions. This emergent behavior can, in general, not be predicted from the individual elements and their interaction. A typical example of emergent behavior is self-organization, e.g. Turing patterns in reaction-diffusion systems. These problems are often irreducible* and cannot be solved by analytical means. The only available option to obtain more insight in these systems is by explicit simulation. Moreover many of these problems are intractable: in order to obtain the required macroscopic information, extensive, computationally expensive simulation is necessary. Simulating models of complex systems requires an enormous computational effort, currently, the only feasible way is to apply massively parallel computing techniques to these models.

*Irreducible problems can only be solved by direct simulation
A major future challenge is to apply High Performance Computing in research on complex systems and, in addition, to offer a parallel computing environment easily accessible for applications [143, 142].

Traditionally, science has studied the properties of large systems composed of basic entities that obey simple microscopic equations reflecting the fundamental laws of nature. These entities could be fluid fields in fluid dynamics, gravitating galaxies in astrophysics, or gluons in lattice gauge theory. These natural systems may be studied by computer simulation in a variety of ways. Generally, the first step in any computer simulation is to develop some mathematical model consisting of a finite number of discrete parts. The correspondence between the discrete components and the natural system is completely arbitrary. Often, the discretization realized indirectly, by means of partial differential equations. An alternative, less widely used, approach is to develop solvers that conserve the characteristic intrinsic parallel properties of the applications and that allow for optimal mapping to a massive parallel computing system. These solvers have the properties that they map the parallelism in the application via a simple transformation to the parallelism in the machine. With these simple transformations the necessity to express the application into complex mathematical formulations becomes obsolete. One example would be the modeling of a fluid flow. Traditionally this problem is simulated through mathematical descriptions of the phenomena via Navier Stokes equations and discretization of these equations into numerical constructs for algorithmic presentation on a computer. This process of simulation involves a number of approximations and abstractions to the real fluid flow problem. Moreover, while defining the simulation model, intrinsic properties and explicit information of the physical phenomena are obscured. Even worse, the possible implicit parallelism of the problem becomes completely indistinct in the abstraction process. Also, in several cases, traditional mathematical methods are not able to model (or solve) complex phenomena (e.g., turbulence, road traffic, etc.). An alternative approach would be to model the microscopic properties of the fluid flow with cellular automata. Next, the macroscopical processes of interest can be explored through computer simulation. This approach has the advantage that the physical characteristics of the fluid flow problem remain visible in the solving method and that this method conserves the parallelism in the problem. Although such simulation methods are not yet completely understood and certainly not fully exploited, they are of crucial importance when massive parallel computers are concerned. We define these type of solvers as “natural solvers” these techniques have in common that they are inspired by processes from nature. Four important examples of natural solvers are Genetic Algorithms (inspired by the process of natural selection) [72], Simulated Annealing (inspired by the process of cooling heated material which converges to a state of minimal energy) [83], the Lattice Boltzmann method (a many particle system with a macroscopic behavior that corresponds to the hydrodynamic equations) [151], and artificial Neural Networks. In parallel computing especially the class of natural solvers is a very promising approach, since the characteristics of the original physi-
1.1 Complex Systems simulation

In Fig. 1.1 the different steps in the mapping process from application onto parallel machines is presented. As can be seen, an application is first transformed into a solver method. Detailed knowledge of the problem domain is obligatory. A solver can have a different nature and one particular problem can be represented by different solver methods (like the example of the fluid flow). Next, the intrinsic parallelism in the solver is passed through the decomposition layer that captures the parallelism and dependencies into objects and communication relationships. Finally, these two classes are mapped onto a Virtual Parallel Machine model that allows for implementation on a large suite of parallel systems.

In a nutshell we have just described the main context of this thesis. The main emphasis is on the definition, characteristics and applicability of the so called decomposition layer, the resulting interacting objects and their mapping onto a virtual parallel machine.

To be able to capture the generic aspects of parallel solvers and to express the basic properties of the natural system we will define our own abstract solver model indicated as the Virtual Particle model. The Virtual Particle (ViP) can be
defined as the basic element in the simulation model. The ViP can be defined on several levels of abstraction. For example in a simulation model of a biological system, the ViP can correspond to a certain level of organization and aggregation in the system (e.g., molecule-organelle-cell -tissue-organ- organism-population). The choice of the abstraction level is determined by a combination of the desired refinedness of the model and the computational requirements.

In the ViP model the microscopic, temporal or spatial, rules have to be specified in such a way that they approximate the microscopic rules as observed in the actual system. In the ViP model, the ViPs may correspond to the individual particles in the natural solver, as for example in lattice gases. In the ViP model the particles can also be organized hierarchically, where ViPs can be an individual particle or clusters of ViPs. In such a hierarchical model interactions can occur between individual ViPs, clusters of ViPs, and individuals and clusters. An example where clustering of ViPs is applied is in $N$-body problems, where the $O(N^2)$ number of long-range interactions between the particles can be approximated by $O(N\log N)$ interactions by using hierarchical tree methods [6, 11, 66].

In the hierarchical algorithm the overall complexity of the problem is reduced, which allows for the simulation of relatively larger problems, while the information about the error introduced by this approximation is preserved. By allowing to cluster ViPs into a new ViP, it becomes possible to develop an efficient abstraction and consequent mapping of the application model onto the Virtual Parallel Machine model.

The natural system is represented by a Dynamic Complex System (DCS), the application model, where the individual elements are the ViPs. The application model is mapped onto the Virtual Parallel Machine Model (see Fig. 1.2), which can be another instance of a DCS consisting of a population of processors. The concept of interconnected Virtual Particles will be used to define and build a computational framework to allow an efficient mapping of the ViPs to a parallel machine model. By means of this framework we are effectively merging the application DCS and the parallel machine DCS, each with their specific characteristics. By viewing a parallel machine as yet another DCS we are enabled to study it with common techniques from the field of natural DCS, i.e., physics. An important aspect of complex systems theory is the genericity encountered in the statics and the dynamics of the system. Different microscopic interaction details are often irrelevant for triggering specific macroscopic phenomena like for example pattern formation [162, 86, 115], strange attractors [64, 69] and phase transitions [140, 128, 146, 102, 41].

On the other hand, physical and biological DCSs may be characterized using methodology from the field of computer science. In the remaining part of this chapter we will talk about the computational aspects of a DCS and discuss a number of examples where the computational properties of physical and biological systems are studied.
1.2 Computation in Dynamic Complex Systems

In this section we will discuss the concepts of Universal Computing, Cellular Automata, Undecidability and Intractability. The underlying motivation is to integrate our current knowledge of formal computation with the notion that physical (and biological) processes can be viewed as "computations."

1.2.1 Universal Computing

The intuitive notion of an algorithm can be formalized by a Turing Machine (TM) (Alan Turing, 1912-1954). It is conjectured that any way of formalizing an algorithm is equivalent to formulating a Turing Machine. Formally, the basic model consists of a finite control, an input tape and a head that can be used for reading or writing on that tape. A Turing Machine can execute an algorithm defined by a string on the tape. The string is constructed from a finite alphabet. A TM can decide whether a given string is an element of a language. A language is defined as a set of strings. A Turing Machine can both be used for deciding membership and for computing functions (basically the two operations are identical). A language $L(M)$ that is accepted by a TM $M$, is said to be recursively enumerable (r.e.). The class of r.e. languages includes some languages $L(M)$ for which there does not exist an algorithm to determine the membership of a string, i.e., we cannot tell whether the machine $M$ will eventually accept the string if we let it run long enough, or whether $M$ will run forever [73]. A subclass of the r.e. languages are the recursive sets, which are those languages accepted by at least one TM that halts on all inputs.

We can also view a Turing Machine as a device that computes functions from integers to integers. Again we can identify two different classes. First, the so
called total recursive functions, which are those functions that are computed by Turing Machines that halt on all inputs. For example all common arithmetic functions, like multiplication, addition, etc, are total recursive functions. The analogon of r.e. languages in the computation of functions are called partial recursive functions, i.e., those functions that are computed by Turing Machines that may or may not halt on a given input.

With this background information we can now define the Church-Turing Hypothesis, which states that the intuitive notion of an algorithm can be identified with the class of partial recursive functions and consequently computed by a Turing Machine.

Currently we cannot prove this hypothesis because of the informal notion of "computable." If we place no bound on the resources (time and storage), the partial recursive functions are intuitively computable. It is difficult to tell whether the class of partial recursive functions includes all "computable" functions. Many other formalisms have been presented, like λ-calculus and Post systems, but all are equivalent to the class of partial recursive functions [73].

A TM that can simulate an arbitrary TM is called a Universal Turing Machine (UTM) and as a consequence has the ability to do universal computation. The rules of a particular TM are encoded on the tape of the UTM. An UTM consists of the following parts: the part where the TM is encoded, the part where the input is encoded, and a workspace where information representing the current state of the TM head, the current character of the TM tape and the movement command are encoded. In the section on Cellular Automata, which are a special kind of computing devices, we will show that these devices are also capable of universal computing.

In the remainder we will talk about TMs solving problems instead of TMs recognizing words or computing functions. We deliberately choose this nomenclature because it is more related to the spirit of this thesis.

1.2.2 Cellular Automata

Even in systems with a very simple structure, it is possible to observe complex behavior. Examples are systems with only a few degrees of freedom, e.g., the logistic map, and systems with many, but a finite degree of freedom, e.g., Cellular Automata (CA) [166].

The research on CA started in 1948, when von Neumann embarked on an ambitious project: to show that phenomena as complex as life - the survival, reproduction and evolution of complex forms of organization - can be reduced to the dynamics of many identical primitives capable of interacting and maintaining their identity. Following a suggestion by Ulam he adopted a fully discrete approach: space, time and the dynamical variables were defined to be discrete. From these first investigations by von Neumann resulted the concept of a cellular automaton. The nice thing about CA, is that they are easy to construct and are explicitly parallel, which make them good model candidates for implementation on massively parallel computers. On the one hand, CA can be seen as discrete approximations to partial differential equations [153] and can be used
for modeling a wide variety of Dynamic Complex Systems [100]. Another approach based on so called “intuitive” physics, where basic microscopic interaction rules are constructed using a minimal set of physical principles. Lattice Gas Automata (LGA) are a good example of this approach [39]. LGA are a special kind of Cellular Automata to model fluid flow. The microscopic interaction rules are constructed such that they conserve basic fluid properties like conservation of mass and momentum. In Chapter 5 we will discuss these LGA in more detail.

Cellular Automata are mathematical idealizations of physical systems in which space and time are discrete and physical quantities take on a finite set of discrete values or states. A cellular automaton (CA) consists of a regular uniform lattice of cells. A CA evolves in discrete time steps, with the value of the state at one cell being affected by the states of the cells in its neighborhood at the previous time step. The neighborhood of a cell consists of the cell itself and its immediate neighbors. By immediate neighbors we mean those cells that can be reached in \( r \) steps. For example if we have \( r = 1 \) and a 1D lattice, the neighborhood consists of the cell itself and its left and right neighbors, which results in a neighborhood of size 3. Let \( a_i(t) \) denote the value of cell \( i \) in a one dimensional cellular automaton at time step \( t \). Each cell value is specified as an integer in the range 0 through \( k-1 \). The cell values evolve according to:

\[
a_i(t) = F[a_{i-r}(t-1), a_{i-r+1}(t-1), \ldots, a_i(t-1), \ldots, a_{i+r}(t-1)]
\]

where \( F \) is called the transition function of the CA. The parameter \( r \) determines the range of the rule, i.e., the neighborhood. The size of a neighborhood is \( 2r+1 \). Propagating features generated in CA evolution travel at most \( r \) cells per time step. After \( t \) time steps a region of at most \( 1 + 2rt \) cells may therefore be affected by a given initial rule. The total number of different neighborhood states, is \( k^{2r+1} \) which implies that the total number of different CA rules is \( k^{2k+1} \).

It is possible to code a specific transition rule by a numerical code. This code is determined from the binary value that is associated with the resulting states found after applying the rule to the specific neighborhood. Let’s try to clarify this by an example.

Consider a 1D CA, with \( k = 2 \) and \( r = 1 \). The total number of possible neighborhoods equals 8:

- 111
- 110
- 101
- 100
- 011
- 010
- 001
- 000

Under each possible neighborhood the new state, to which the middle cell evolves, is depicted. The binary number 01101110, which is read from reading the new states from left to right, defines the rule. The corresponding decimal number is 110, and by definition the update rule is classified as CA 110. Note that the transition rule can also be identified with a boolean function. The “space-time” diagram of CA 110 is shown in Fig. 1.3 as it evolves from a random initial configuration.
Universal computing in 2D CA

In 1971 Alvy Ray Smith constructed a series of progressively simpler CA capable of universal computation [77]. He started by constructing a 2D CA that directly simulates a Turing Machine, i.e., with a one-to-one correspondence between the steps of the CA and the steps of the TM. The CA states represent both tape symbols as well as the states of the TM. If the TM has $m$ tape symbols and $n$ internal states, the CA has $1 + \max(m, n)$ states.

The construction continues as follows, one row of the CA is used to represent the tape. The row of cells above the tape contain one cell, $h$, that simulates the head. The cells immediately to the left and right of $h$ are labeled $a$ and $b$ respectively. The cell just below the head is called $s$, and is used to simulate the tape symbol that is currently being scanned. All cells, except the tape head and the input tape are in the “quiescent” state, denoted by “0.” In Fig. 1.5, the labeling of the different cells is schematically depicted in a 2D CA grid. Subsequently, the following actions of the TM have to be encoded into a CA transition rule:

- scan the current symbol $u$ in state $v$.
- write a new tape symbol $p$ and go into state $q$.
- move either left or right.

Using the neighborhood template of Fig. 1.4 and the schematic representation of different types of cells in Fig. 1.5, the actions can be encoded into a CA transition rule:

One of the four CA rules is applied depending on the cell label of the shaded cell (see Fig. 1.4). In all other cases nothing happens with the state of the shaded cell.

In this way, the CA can simulate any given TM in real time. As a consequence, a CA can be constructed that simulates a Universal Turing Machine in real time. In the remaining part of Smith’s paper he gives several variations of the construction described above, using different neighborhood templates and one dimensional architectures.
1.2 Computation in Dynamic Complex Systems

<table>
<thead>
<tr>
<th>Cell $c$</th>
<th>Neighborhood $c$</th>
<th>Next state of $c$</th>
<th>conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$</td>
<td>$p$</td>
<td>$p$</td>
<td>$S_0 = u$</td>
</tr>
<tr>
<td></td>
<td>$S_{-1}S_0S_1$</td>
<td>$000$</td>
<td>$P = v$</td>
</tr>
<tr>
<td>$h$</td>
<td>$0$</td>
<td>$0$</td>
<td>always</td>
</tr>
<tr>
<td></td>
<td>$0^r0^l$</td>
<td>$0$</td>
<td>if right move</td>
</tr>
<tr>
<td></td>
<td>$S_{-1}S_0S_a$</td>
<td>$q$</td>
<td>if left move</td>
</tr>
<tr>
<td>$a$</td>
<td>$0$</td>
<td>$0$</td>
<td>if right move</td>
</tr>
<tr>
<td></td>
<td>$0^r0^l$</td>
<td>$q$</td>
<td>if left move</td>
</tr>
<tr>
<td></td>
<td>$S_{-2}S_0^rS_0$</td>
<td>$0$</td>
<td></td>
</tr>
<tr>
<td>$b$</td>
<td>$0$</td>
<td>$q$</td>
<td>if right move</td>
</tr>
<tr>
<td></td>
<td>$P0^r0^l$</td>
<td>$0$</td>
<td>if left move</td>
</tr>
<tr>
<td></td>
<td>$S_0S_1S_2$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1.1: Part of the transition rule for Smith’s universal CA.

![Figure 1.4: The neighborhood template used in White's universal computing CA](image)

![Figure 1.5: Schematic representation of the operation of White's CA. The tape head is labeled $h$ with current state $P$. The cell to the left of $h$ is labeled $a$ and the cell to the right is labeled $b$. Cell $s$, containing symbol $S_0$, is currently being read by the head $h$. The other symbols on the tape are represented by $S$.](image)

**Classification of CA dynamics**

It is possible to classify CA evolution by their spatial/temporal dynamics. A possible classification was given in [164]. Globally four different classes are identified:

I. **Fixed point behavior**, eventually the evolution of a CA ends in a fixed point of the state space.

II. **Periodic behavior**, the evolution $f$ is periodic with a period $T$, so $f(t) = f(t + T)$.

III. **Chaotic behavior**, evolution leads to chaotic patterns.
IV. Complex behavior, complicated localized patterns occur, sometimes they die out sometimes they don't.

The first three types of behavior are commonly found in dynamical systems, only until recently, behavior of the complex-type has been identified in dynamical systems [113, 127, 76, 126].

In Figs. 1.6 and 1.7, the four possible types of dynamical behavior are shown for 1 dimensional CAs. Dynamical behavior of the CAs in Fig. 1.7 are examples of computational irreducibility. The problem of determining the configuration of a CA after $t$ time steps, given an initial seed and the CA-rule, can be solved in $O(\log t)$ time for the CAs in Figure 1. The evolution for the two other rules can only be found by explicit simulation with length $O(t)$ [165]. The difference is that the former two are reducible and the latter two are irreducible, i.e., their ultimate behavior can only be predicted by explicit simulation. For infinite class IV CA it is effectively undecidable whether a particular rule operating on a particular initial seed will ultimately lead to a frozen state or not [89, 165]. Obviously this decision problem is not undecidable for finite CA, because it is always possible to determine its fate in at most $k^N$ steps, for $k$ states and $N$ cells.

In Section 1.2.3 we will discuss the concepts of irreducibility and undecidability in more detail. Summarizing CA behavior:

- **Computational reducible behavior**: fixed points and periodic. Final state of the system may be predicted analytically.

- **Computational irreducible behavior**: chaotic dynamics are unpredictable and complex dynamics can even be undecidable. The evolution of these
systems is its own simplest description and its own fastest computation [43].

As a consequence of these observations, it is assumed that dynamical systems which display complex behavior are capable of universal computation [164, 89]. If this hypothesis is true than CA 110, as described above, is a good candidate for universal computation [91].

1.2.3 Computing and Simulation

In this Section we will review the most important concepts from the field of computational complexity. The notions of undecidability, intractability and irreducibility will be introduced in an informal way. The position of simulation in the world of computational complexity will be discussed in the section on irreducibility, which was already loosely discussed in Section 1.2.2.

Undecidability

In 1931 the logician Kurt Gödel proved that a certain axiomatic system of number theory was incomplete. His theorem informally states that there are certain statements about numbers that cannot be proved within the axiomatic formalism. Gödel's original proof of the incompleteness theorem is based on the paradox of the liar: "This statement is false." By changing this statement to: "This statement is unprovable," he obtains a theorem instead of a paradox. If this assertion is unprovable, than it is true and the formal system is incomplete. If this assertion is provable, then it is false and the formal system is inconsistent. The original proof of Gödel was based on a specific formalization of number theory and was followed by a paper showing that the same methods apply to a much broader class of formal axiomatic systems. At that time he could not yet prove it for all axiomatic systems, due to the lack of a mathematical definition of an algorithm. Only after the formal definition of an algorithm by Turing, it became possible to proceed in a more general fashion. In turn Turing showed that there is no algorithm for deciding whether or not a program ever halts, i.e., the halting problem is unsolvable. One can derive Gödel's theorem from the halting problem [22]. It is a fact that any nontrivial property about TMs, is undecidable. The latter fact is known as Rice's theorem [73]. Using Rice's theorem, we know that if we observe a physical phenomenon that is able to mimic a universal TM then most questions, concerning this phenomenon, in idealized limits of infinity, require arbitrary long computations and are formally undecidable.

Intractability

Computational complexity describes the amount of time, space or other resources needed to solve a problem. The space complexity \( S(n) \) denotes the maximum number of cells that a TM \( M \) will scan for an input of length \( n \). The problem solved by \( M \) is said to be of
space complexity $S(n)$. The time complexity $T(n)$ denotes the maximum number of moves that $M$ will make for every input word of length $n$. $M$ is said to be of time complexity $T(n)$.

In the time complexity regime two important classes can be defined: $P$ and $NP$. The class $P$ is informally defined as the class of those problems for which there is a polynomial time deterministic TM to solve them. The class $NP$ is informally defined as the class of those problems for which there is a polynomial time non-deterministic TM to solve them. Furthermore, it is defined that a problem belongs to $NP$ if and only if a given solution can be verified in polynomial deterministic time (see also [18]).

A special class of problems are known as $NP$-complete. A problem $L \subseteq NP$ is $NP$-completer if every other problem $L \subseteq NP$ can be reduced to it in polynomial time. A problem $L$ can be polynomially reduced to another problem $L'$ if there exists a polynomial time transformation from an instance in $L$ to an instance in $L'$. Next, we have to find the first $NP$-complete problem. Therefore we cannot use the above mentioned approach, simply because we do not have a known problem to transform from. As we shall see a common trick to prove $C$-completeness is just to mimic a specific machine that accepts an arbitrary language $L\in C$, where $C$ stands for a specific complexity class, i.e., $P$ or $NP$. We shall see that this is also the approach that can be used in deciding completeness results for certain physical phenomena. Thus in order to prove $NP$-completeness for our first language, we must show that this language is able to mimic the computation of an arbitrary language in $NP$ on a non-deterministic TM in polynomial time. The Satisfiability problem was the first problem for which this was done [18, 58]. Mutatis mutandis, we can define the class of $P$-complete problems. These problems possess yet another characteristic, irreducibility, which will be discussed in the next section.

Another important class of problems is called $NP$-hard. A problem $L$ is called $NP$-hard if all problems in $NP$ can be reduced to it, but $L$ is not necessarily in $NP$. A major open problem in the field of computational complexity is the question whether $P=NP$. It is suspected that $P$ is a proper subclass of $NP$, which implies that $NP$-complete problems are provably more difficult than problems in $P$.

In the space complexity regime we can define the classes PSPACE and NPSPACE. The class PSPACE is informally defined as the class of those problems that can be solved by a polynomial space bounded deterministic TM that halts on all inputs. NPSPACE is defined as PSPACE, but now for non-deterministic TMs. As opposed to the time complexity classes $P$ and $NP$, it is possible to prove that $PSPACE = NPSPACE$ [73, 58]. Hence deterministic polynomial space is as powerful as non-deterministic polynomial space. We can now define the class of PSPACE-complete problems: a problem $L'$ is PSPACE-complete, if $L' \in PSPACE$ and for all $L \in PSPACE$, $L$ can be reduced to $L'$ in polynomial time. It follows that if $L$ is PSPACE-complete, then $L \in P$ if and only if $P = PSPACE$ and $L \in NP$ if and only if $NP = PSPACE$. Note that we could have $P = NP$ even if $P \neq PSPACE$, indicating that if a problem is PSPACE-complete, it is an even stronger indication that it is intractable, than if it is NP-complete [58]. Note that every
known NP-complete problem is also in PSPACE. Note that all problems solvable in polynomial time can be solved in polynomial space. Or in other words, \( L \) must be able to simulate a universal polynomial-space bounded TM, i.e., a TM which can compute every polynomial-space bounded function. The first problem that was proven PSPACE-complete is called Quantified Boolean Formulas (QBF)[58].

Analogous to the NP-hard problems, we can define a PSPACE-hard problem \( L \) as a problem to which every other problem in PSPACE can be reduced, but \( L \) is not necessarily in PSPACE.

**Irreducibility**

Now that we have indicated the hardness of certain kind of problems. We are confronted with situations where it is fundamentally impossible to find shortcuts for solving problems, or even worse, solving those problems can be computationally intractable. In this section we will position simulation in the world of computation theory. For the moment let us define a system as some dynamical entity which evolves through “time.” In the contexts of CA we already observed that a complex system can be called irreducible whenever the system is capable of mimicking a Universal TM. Problems regarding the system’s ultimate fate are often computationally intractable. Irreducibility of a problem implies a simulation approach, by the fact that every computational step must be explicitly simulated, i.e., we can also conclude that the system suffers from history dependence. In other words we want to solve the problem in what state a system will be after “running” for \( t \) time steps.

The problem of irreducibility is closely related to the computational theoretical notion of efficient parallelization. Computational complexity theory offers different formal models for parallel computers, e.g., P-RAM and uniform circuit families [18]. A separate complexity class, uniform \( NC \), has been defined for problems that admit efficient parallelization [18]. The definition is as follows: Let \( L \) be a problem that is solved on a standard TM. If the time complexity of \( L \) is \( O(N^k) \), \( L \) is in uniform \( NC_k \) iff there exists a parallel algorithm which can solve the problem is poly-logarithmic time, \( O(\log^k N) \), using only \( O(N^k) \) processors. The class of \( P \)-complete problems now contains those problems which are “hardest” to solve in parallel, i.e., for which no efficient parallel algorithm exists (unless uniform \( NC = P \), which is also unknown). A well known \( P \)-complete problem is the Circuit Value Problem (CVP). The problem is, given an input string, to compute the truth value of a so called Boolean circuit, which consists of a hierarchical connected topology of boolean gates. Another interesting \( P \)-complete problem is Generic Machine Simulation (GMS): Given an input \( x \), an encoding of a TM \( T \), and an integer \( t \) coded in unary, decide whether \( T \) accepts \( x \) within \( t \) steps. The \( P \)-completeness result of GMS implies that the computation of any system that is able to perform arbitrary computations is a \( P \)-complete problem. As an example consider an \( N \)-body system consisting of a set of interacting particles. A question could be: “Will a particular particle ever reach a certain part of the embedding space?” It can be shown that this problem is actually
PSPACE-hard, hence the solution cannot be found by a “fast” procedure (see Section 1.3.2). The system can perform “computations” that are as powerful as those performed by universal computers requiring polynomial space. This result implies that the dynamics of the system, i.e., the state of the system \( t \) time steps away, is a P-hard problem, hence irreducible.

The definition of computational irreducibility, stating that a system can only be explicitly simulated, is the same as stating that a P-hard problem must be solved. This implies that efficient parallel simulation is not possible according to the given definition. Besides the fact that those systems can not be solved by analytical means, they can also not be simulated efficiently in parallel. This sounds like a devastating result, especially since this thesis concerns the parallel simulation of complex systems.

However, the definition of “efficiently parallel” used by the people from the field of computational science differs from the previous definition used in complexity theory. Parallel computers are successfully applied to solve large problems from the field of computational science. Generally, the speedup in computation time is at most linear in the number of cooperating processors.

Consider for example a parallel method for finding the largest integer among a set of \( n \) integers. The problem can be solved by decomposing the problem in \( \sqrt{n} \) sub problems of \( \sqrt{n} \) elements each. In the first step a sequential algorithm is applied in parallel to determine the largest value of each sub problem. Subsequently the same algorithm is applied to determine the largest value among the \( \sqrt{n} \) resulting elements. The parallel time complexity of this method is \( O(\sqrt{n}) \), which is not efficiently parallel, but in practice we can still be satisfied (Note that an efficient algorithm of \( O(\log(n)) \) exists [18]).

When simulating a system, certain phenomena or properties can emerge during the simulation, phenomena that are not explicitly coded in the simulation model. By analyzing this emergent behavior, we can answer questions about those systems. For every DCS we can define a set of properties we are interested in. As an alternative to computing the property we can simulate the system and “watch” whether a specific property emerges.

Even non-computable problems can be “solved” by simulation, just by watching the system evolve. An example [125] is the problem of determining whether a certain point is a member of the Julia set, which is the closure of the unstable equilibrium set of the following complex mapping:

\[
z(n+1) = z(n)^2 + c
\]  

(1.2)

It is undecidable whether a given point in the complex plane is an element of the Julia set [17]. However, it is possible to just simulate the mapping and observe the membership of a certain point.

P-completeness has been proved for a variety of simulation models. For example, Moore and Nordhal have shown that LGA (see Section 1.2.2) prediction is P-complete, implying that there is no shortcut possible to calculate the future state of an arbitrary LGA configuration [114].

In the next section we will show how the discussed concepts can be used in a
1.3 Intractability and undecidability in some Physical Systems

Physical processes can be viewed as computations [165], where the difficulty of answering questions about these processes is equivalent to performing the corresponding computations. In principle the behavior of many complex system can be calculated by explicit simulation. However, the theoretical sciences are concerned with devising shorter calculations to reproduce the outcome of such systems. Assume a class of systems that can indeed be predicted by using a formula describing its future state. Then essentially, this means that the calculations performed by using the formula must be more sophisticated than the calculations performed by the physical system itself. This implies that the formal system must be more powerful than the corresponding physical system. However, for a physical system that is able to perform calculations that are as powerful as those performed by a Universal Turing Machine, no shortcuts are possible. In contrast some physical processes behave chaotic [33] and therefore cannot be viewed as computations. Here we are confronted with a kind of unpredictability: it is not possible to predict the future accurately, just because slight discrepancies in the initial configuration will grow unbounded in time [40]. This implies that, to accurately predict the system \( t \) steps in the future, we need approximately \( t \) digits in the specification of the initial configuration. So, the amount of information necessary to predict the future grows exponentially with time.

A trivial example of a physical process that can be viewed as a computation, is the logic performed by a general purpose computer. There is no way of finding a general procedure for predicting future states. Using computation- and complexity theory it can be shown that such systems have a stronger kind of unpredictability than chaotic systems [165, 113]. Of course everything that is said here is based on the Church-Turing Thesis and in particular on a physical version of it:

"Universal computers are as powerful in their computational capabilities as any physically realizable system can be, so that they can simulate any physical system [165]."

If this hypothesis is true, no physical system can shortcut a computationally irreducible process. In this case simulation is the only way out. A system with bounded storage capacity that solves PSPACE-complete problems can be called universal. Can we find any physical system, other than an artificial computer, that is capable of universal computation? It is conjectured there are many of such systems [123, 165, 113]. A complex dynamical system, that can simulate any universal computation is PSPACE-complete. The question whether a given initial state \( x \) will ever reach a particular set \( A \) is equal to the question whether a Universal Turing Machine ever halts for a given initial input. Even if \( x \) is known
exactly, its basin of attraction is not a recursive set, i.e., there is no algorithm to test whether or not a point is in it [113]. In the section on CA we have seen that even in structurally very simple computational systems, very complex behavior can emerge, so that questions about their long time behavior are formally undecidable.

Summarizing so far, we can say that a dynamic complex system is formally intractable if it can simulate any polynomial space bounded TM and thus questions regarding its behavior can be PSPACE-hard. In addition it is possible that some systems can simulate any polynomial time bounded non-deterministic Turing Machine. Imagine for example a system in which certain questions about it can be non-deterministically guessed and checked in polynomial time. Questions regarding such systems can be called NP-complete. The last and most difficult type of behavior, is displayed by those systems, which are capable of imitating any Turing Machine, with no resource bounds. Answering questions about these kind of systems can be formally undecidable, we can simulate for ever, without being certain of its behavior. Note that only systems having a continuum of states are potentially Turing universal, while in space bounded systems having a finite number of states, the limited precision will cause that after some period, the system will return exactly to some previous state.

In this section we will discuss models of three different complex systems, namely an optical system [127], an N-body system [126] and non-equilibrium growth [80]. It can be shown that these systems exhibit undecidable, intractable and irreducible behavior. We will shortly describe these systems and their related hardness proofs. For a thorough description we refer to [127, 126, 80].

### 1.3.1 Optical Beam Tracing

First the optical system, which also forms a basis for the hardness proofs of the N-body system. In [127] the “Computability and Complexity of Optical Beam Tracing” is explored. The Ray Tracing Problem (RTB) is stated as:

\[
\text{Given an optical system (a finite set of reflective or refractive surfaces) and an initial position and direction of a light ray and some fixed point } p, \text{ does the light ray eventually reach the point } p?\]

We summarize the computability results for different optical models, for more detail see [127]:

1. In a three dimensional system which consists of a finite set of mirrors, half-silvered mirrors and quadratic lenses, RTB is undecidable. The proof remains valid for rational coordinates.

2. In a three dimensional system which consists of just a finite set of mirrors with irrational endpoint coordinates, RTB is undecidable. When the endpoints are restricted to rational coordinates the problem is PSPACE-hard.
3. For any system with dimension \( \geq 2 \), which consists of a finite set of mirrors with rational endpoints which lie perpendicular to each other, RTB is in PSPACE. For dimensions \( \geq 3 \) the problem is PSPACE-complete.

By showing that the total path of the light ray can be divided into sub paths and their corresponding rational equations, all models can be shown to be recursively enumerable. Hence the total path can be traced. In order to prove undecidability, an optical model must be able to mimic an unrestricted Turing Machine. This can be achieved by coding the contents of the tape by \((x, y)\) coordinates of the position of the light ray. The states of the TM are “coded” by basic optical boxes (a specific set of mirrors and lenses), which implement the transition function \( \delta \) for a particular state. The mapping of \( \delta \) onto the operation of an optical model is done by simulating the two basic different types of transitions:

\[
\delta(q, c) = (q', w, L) \quad (1.3)
\]
\[
\delta(q, c) = (q', w, R) \quad (1.4)
\]

Where \( L \) and \( R \) represent a left and right move respectively, \( q \) and \( q' \) correspond to TM states, \( c \) to the read symbol, and \( w \) to the written symbol. The basic boxes implement the various transition functions, by manipulating the light ray through the use of mirrors and lenses, which in turn implement the functioning of an entire TM. Undecidability of RTB using some optical model can be proven by showing that an unrestricted TM can be mimicked. PSPACE-hardness can be proven by showing that some space bounded TM can be mimicked.

### 1.3.2 N-body Simulation

A second physical model to which a hardness proof is applied, is N-body simulations [126]. The N-body simulation problem is stated as follows:

*Given initial positions and velocities of \( n \) particles that have pair-wise force interactions, simulate the movement of these particles so as to determine the positions of the particles at a future time.*

The problem of determining whether a specific particle will reach a certain region at some specified target time is called the N-body reachability problem. The equations of motion for each body is given by Newton’s laws of motion which constitute \( n \) ordinary differential equations. The corresponding solutions can be approximated by time stepping. The initial positions and velocities of the bodies are given by \( n^2 \)-bit rational. The destination position is given by a ball, which is an \( n \)-bit rational (the ratio of two \( n \)-bit integers). A related result can be found in [53], where the “billiard ball computer” is introduced and which is proved to be PSPACE-hard. However this model depends on non-movable obstacles as does the optical model stated above and hence is not applicable to N-body simulation.
The hardness proof is done by constructing an analog of optical beam tracing in the N-body system. It is assumed that initial positions, velocities and position of the destination ball are rational. The global sketch of the proof is by reducing several problems to each other, starting with a known PSPACE-hard problem, namely that of optical ray tracing with reflective (only mirrors) surfaces and ending with the n-body simulation problem by subsequently reducing subproblems. For a detailed overview we refer to [126].

1.3.3 Non Equilibrium Growth Processes

In equilibrium growth processes, as for example found in a perfect crystal where the growth process is near or in equilibrium, molecules are exploring various sites of the crystal and are added to the crystal until the most stable configuration is found. In this type of growth process a continuous rearrangement of particles takes place, the process is relatively slow and the resulting objects are very regular [134]. In some cases the growth form which emerges is a normal object from Euclidean geometry whereas in other cases objects are formed that resemble regular fractal objects.

Many growth processes in nature are not in equilibrium. An extreme example is an aggregation process of particles where as soon as a particle is added to the growth form, it stops trying other sites and no further rearrangement takes place. The local growth probabilities are not equal everywhere on the aggregate and an unstable situation emerges. The growth process in non-equilibrium is relatively fast and often irregular objects, characterized by a fractal dimension, are formed [133, 134, 132]. An example of a growth process, in non-equilibrium from physics, is viscous fingering. The phenomenon can be demonstrated in an experiment where air displaces a high-viscosity fluid between two glass plates. In Fig. 1.8 a diagram is shown of an experiment where air is injected between two glass plates at $y = 0$ and displaces a high viscosity fluid, which is only removed at the top of the plates (both sides are closed). The pressure $P$ will be the highest at $y = 0$ and the lowest at $y = L$, where $L$ represents the length of the glass plates. In the fluid the pressure is given by the Laplace equation [47]:

$$- \nabla^2 P = 0$$

(1.5)

In the air the pressure is everywhere equal, since its viscosity can be ignored. The pressure in the air equals to the input pressure $P(y = 0)$ and the consequences is that the largest pressure gradients occur at the tips of the fingers in Fig. 1.8, while the lowest gradients occur below the tips. The probability that the fingers continue to grow will be the highest at the tips and in a next growth stage the pressure gradients in the tips are still more amplified, resulting in an unstable situation. In Fig. 1.9 an example of the resulting growth pattern is shown, it is an irregular shaped object, known in the literature as viscous fingering.

Another example of growth in non-equilibrium is growth of a bacteria colony (for example *Bacillus subtilis*) on a petri-dish [56]. The colony consumes nutrients
from its immediate environment and the distribution of nutrients is determined by diffusion. When it is assumed that the concentration \( c \) is zero at the colony and that the diffusion process is fast compared to the growth process, the concentration field will attain a steady state in which the diffusion equation

\[
\frac{dc}{dt} = D \nabla^2 c \tag{1.6}
\]

equals zero. In this equation \( D \) is the diffusion coefficient. The nutrient source may be, for example, a circle around the colony or a linear source where the concentration is maximal. The local nutrient concentration at sites between the colony and the source can be described with the Laplace equation:

\[
\nabla^2 C = 0 \tag{1.7}
\]

The growth process of a bacteria colony, viscous fingering, and various other growth patterns from physics as for example electric discharge patterns and growth forms of electro deposits, can be simulated with one model: the Diffusion Limited Aggregation (DLA) model [162, 163, 133]. At the heart of all these growth patterns there is one Partial Differential Equation, the Laplace equation, which describes the distribution of the concentration (Eq. 1.7), pressure
(Eq. 1.5), electric potential, etc., in the environment of the growth pattern. The underlying Laplace equation can be solved numerically and a DLA cluster can be constructed using the nutrient distribution over the lattice. The cluster is initialized with a seed and the following boundary conditions are applied: \( c = 0 \) on the cluster itself and \( c = 1 \) at the nutrient source, which may be circular, linear etc. The probability \( p \) that a perimeter site with index \( k \) will be added to the DLA-cluster is determined by

\[
p(k \in \text{perimeter sites} \rightarrow k \in \text{cluster sites}) = \frac{(c_k)^\eta}{\sum_{j \in \text{perimeter sites}} (c_j)^\eta}
\]

where \( c_k = \text{concentration at position } k \)

The exponent \( \eta \) applied in Eq. 1.8 describes the relation between the local field and the probability. This exponent usually ranges in experiments from 0.0 to 2.0. The sum in the denominator represents the sum of all local concentrations of the possible growth candidates. The probability that new sites will be added to the cluster will be the highest at the tips of the cluster, where the steepest nutrient gradients occur, and the lowest in the bays between the branches. In successive growth steps the nutrient gradients at the tips will even become steeper and a comparable instable situation is encountered as in the viscous fingering example. The effect of changing the exponent \( \eta \) in Eq. 1.8 is that the overall shape of the cluster changes. For the value \( \eta = 0 \) the shape changes in a compact cluster and it can be demonstrated that the DLA-model for this special case transforms into the Eden model [44]. This model is one of the earliest probabilistic cellular automata to simulate growth. In the Eden model each possible growth candidate has the same probability to become occupied. For the value \( \eta = 1 \) the normal DLA-cluster is obtained, while for higher values more dendritic shapes are generated [105]. With the parameter \( \eta \) the effect of nutrient gradients on the growth process can be controlled, where the Eden model is an extreme example in which gradients have no effect on the local probability that a new site will be added to the growth form.

In order to model and simulate different natural growth processes, a variety of growth models have been suggested, each with different characteristics. Most of these models display scaling or fractal behavior, analogous to characteristics observed in second order phase transitions. Besides DLA and Eden, other examples of these models are: invasion percolation, and ballistic deposition [106]. From a computational point of view these models can be divided in two classes known from computational complexity theory: uniform \( \text{NC} \) and \( \text{P} \) (see Section 1.2.3).

As usual complexity classes are defined for decision problems, so in order to talk about the computational complexity of growth models, they have to be formulated as such. Consider a cluster growing on a lattice of \( N \) sites. The decision problem can be divided into \( N \) separate decision problems, corresponding to whether the \( N \) sites of a lattice will eventually be occupied or not by the spreading cluster. Through the equivalence of a hydrodynamic model [85] it can be
shown that DLA is P-complete\cite{95}, hence DLA can be regarded as a universal computer for problems in P. On the other hand the Eden model and invasion percolation can be formulated as \textit{waiting time models} \cite{131} and as such can be proven to be in the class uniform NC \cite{96}.

Let us first consider the equivalence between DLA and the Chamber-Tube (CT) model \cite{85}. An approach to model two-fluid flow (an inviscid fluid driving a viscous one) in a porous medium is to regard the porous medium as a system of chambers connected by tubes. The pressure of the viscous fluid driven by the invading fluid, satisfies a Laplace equation: \( V^2 p = 0 \). Basically this is the CT model, which can be shown to be equivalent to DLA. In \cite{95} it is shown that this CT model is P-complete, from which a restricted planar version of this CT model is also shown to be P-complete. Intuitively the DLA model is inherently history dependent (hence irreducible), i.e., the growth at a given time step depends in the prior history of the system (Markovian). Because of this property, the DLA model does not lend itself for \textit{exact} efficient parallelization in the computational complexity sense.

Apparently also Eden models suffer from history dependence, at each time step a particle is added randomly to the perimeter of the cluster. In \cite{96} it is shown that this history dependence can be overcome. In \cite{131} a mapping is made between the Eden model and the growth of directed polymers in random media. The \textit{directed polymer problem} is defined in a random medium, at zero temperature. To each site \( i \) of a lattice, a random number \( x_i \) from some probability distribution is assigned. This random number corresponds to the local interaction energy between the polymer and the embedding medium at this site. A directed polymer is a directed path \( P \) that spans the lattice. The energy of the polymer is the sum of the \( x \)'s along the path \( P \). At zero temperature the polymer will be in the lowest energy configuration, such that \( E \) will be:

\[
E = \min_P \left\{ \sum_{i \in P} x_i \right\} \tag{1.9}
\]

Subsequently an equivalence with Eden growth can be made. Set a clock on each site of the lattice and record the time during which the site has been a potential growth site, without being actually part of the cluster. Note that eventually all sites will be part of the Eden cluster. Call this delay \( \tau_i \) for site \( i \). From these delay times, the real time \( t_i \) at which the site became part of the cluster can be computed: it is equal to the time at which \( i \) became a neighbor to the cluster plus the time \( \tau_i \). The time \( t_i' \) at which the neighbor becomes part of the cluster is the minimum of the sum of delays to this neighbor:

\[
t_i = \min_{P_i} \left\{ \sum_{j \in P_i} \tau_j \right\} \tag{1.10}
\]

The resemblance between Eq. 1.10 and Eq. 1.9 is obvious. The time should be compared with the minimum energy of a polymer. This so called waiting time model can in turn be mapped onto a Minimum-Weight Path algorithm (MWP), which is known to be efficiently parallelisable. MWP is defined as:
An undirected graph $G = (V,E)$, where $V$ is a set of sites and $E$ is a set of bonds connecting pairs of sites. Weights $w(i,j)$ are assigned to each bond $\{i,j\} \in E$. The problem is to find a matrix containing weights of the minimum-weight paths between every pair of sites in $V$.

We have seen that it is possible to describe both Eden and DLA with the same equation [122], only differing in one parameter $\eta$. When $\eta = 1$ DLA is recovered, and when $\eta = 0$, the Eden model is obtained. It follows that, though both DLA and Eden can be described by the same equations, they are not equal in computational complexity. We now speculate that this transition to another complexity class can be established by changing this $\eta$ parameter. It is interesting to know whether general properties and conditions for this difference in time complexity between growth models can be found.

### 1.4 Speculation: Computing at The Edge of Chaos

So far we have identified systems which display all kinds of complex behavior, i.e., they were found to be computationally irreducible or intractable or even formally undecidable. The other side of the metaphorical DCS medal are the physical aspects of computation. Questions we can ask at this point is: “When does a system display complex behavior?” or “Where are the most difficult problems instances located?” The first question tries to answer the physical requirements and consequences for computations, complementing the theoretical computational aspects discussed in the previous sections. The second question deals with the location of computationally difficult problem instances, instead of only classifying them by their worst case scenarios. This thesis will not be that ambitious and overconfident to answer these long standing open problems, however we will shortly discuss current hypotheses regarding this subject. In Chapter 3 we will discuss this matter in the context of a famous problem in parallel computing: the Task Allocation Problem.

In 1949, Shannon wrote a book on “The mathematical theory of communication.” In this work Shannon defines a mathematical framework for the quantification of and the reasoning about information. The most important quantity that he defines is called information entropy, which is quite similar to the thermo-dynamical entropy used in statistical physics. Instead of quantifying disorder by the probability of each state in which a physical system can be found, information entropy quantifies uncertainty of a data stream by using the probabilities of the occurrence of each symbol. We will go in more detail to this subject in Section 3.4.1.

Some authors [164, 88, 31] believe that when a system displays “complex” behavior, universal computations can be performed. Mechanically speaking a computational system requires transmission, storage and modification of information. So, whenever we identify those three components in some dynamical system, the system could be computationally universal. But then the question remains when does this happen? Broadly said, using information theo-
In a number of papers, Christopher Langton and co-workers have tried to answer this question by considering Cellular Automata as a theoretical model for a physical system [88, 89, 92, 167, 67]. The hypothesis “Computation at the Edge of Chaos” resulted from this research. Briefly it says that universal computations can take place at the border between order and disorder (i.e., chaos). This statement partly resulted from the field of statistical physics: temporal and spatial correlations can become infinite during or at a second order phase transition between for example a solid and a liquid phase. Recall that a discontinuous change in an order parameter of the system corresponds to a first order transition. A sudden, but continuous, change corresponds to a second order transition. At such a transition, the system is in a critical state. By some authors [88, 82] it is believed that these infinite correlations can be interpreted as long-term memory needed to store information.

Some non-equilibrium systems can display critical behavior without external parameter tuning. This critical behavior is analogous to the behavior of equilibrium system at second order phase transitions, i.e., no characteristic length scales can be found. Systems with critical behavior as an attractor, are denoted by the term Self Organized Criticality (SOC), coined by Bak and co-workers [9]. In Chapter 6 SOC is studied in the dynamics of parallel discrete event simulation.

### 1.4.1 Phase transition in CA

In this paragraph we will shortly review the work on phase transitions in CA, conducted by Langton motivated by the work of Packard [120]. If we have a k-state CA with a neighborhood size \( r \), the total number of possible transition rules is \( k^r \), which can become very large, even for a moderate number of states and/or a small neighborhood. If a structure is present in this enormous space, it should be possible to identify areas of equal complexity (Wolfram classes (see Section 1.2.2) and how these areas are connected to each other. Using this ordering scheme, one can locate those areas which support the transmission, storage and modification of information. Langton suggested the \( \lambda \) parameter, to structure the CA rule-space. An arbitrary state \( s \in \Sigma \) is assigned to the quiescent state \( s_q \). Let there be \( n \) transitions to this quiescent state in an arbitrary transition rule. The remaining \( k^r - n \) transitions are filled randomly by picking uniformly over the other \( k - 1 \) states:

\[
\lambda = \frac{k^r - n}{k^r}
\]

If \( \lambda = 0.0 \) then all transitions in the rule will be to the quiescent state \( s_q \). If \( \lambda = 1.0 \) there will be no transitions to \( s_q \). All states are represented equally in the rule if \( \lambda = 1.0 - 1/k \). With the aid of the \( \lambda \)-parameter it should be possible to examine the conjecture that complex behavior is located at intermediate regime be-
Between ordered and disordered behavior. The spectrum of dynamical behavior can be explored with the so-called *table-walk-through-method* which increases the $\lambda$-parameter at successive time steps. At each new time step a transition table is incrementally updated using the transition table at the previous time step. Because the described method is actually a "random walk" through a coarse-grained version of the CA state-space, each table-walk displays quantitatively different behavior. Several measures can be used to characterize the dynamical behavior of the CA at each new value of the $\lambda$-parameter. These measures include the numerical determination of entropies and both temporal and spatial mutual information statistics, which is a kind of correlation measure for information entropy. In Section 3.4.1 we will discuss these measures in more detail. At intermediate values of $\lambda$, i.e. at the edge between ordered and disordered dynamics several remarkable events occur:

- Transient lengths grow rapidly, analogously to the physical event of *critical slowing down*.
- Transient lengths depend exponentially on the size of the CA.
- Mutual information measures have their maximum values (both spatial and temporal mutual information) (see Fig. 1.10) at the entropy transition (see Fig. 1.11)

![Figure 1.10: Temporal mutual information between two sites separated by one time step for a 4-state 2-neighbor CA, as $\lambda$ is varied.](image)

The exponential dependence of transient lengths on the size of the CA could be compared to the exponential dependence on problem size in the NP and
PSPACE complexity classes. As for the halting-computations, it will be formally undecidable for an arbitrary CA in the vicinity of a phase transition, whether transients will ever die out. The increase in mutual information indicates that the correlation length is growing, which implies further evidence for a phase transition in that region. Of course we cannot observe a real phase transition other than in the thermodynamic limit.

Other evidence for the “Edge of chaos” hypothesis can be found in the work of Crutchfield on continuous dynamical systems [30] and the resulting ϵ-machine reconstruction. In [32] the so called intrinsic computation abilities of a continuous dynamical system are investigated. The output of the system (an iterative map: \( x_{n+1} = f(x_n) \)) in time is coarse grained into a sequence of zeros and ones. In other words the output domain \( x_n \) is divided into two regions, \( P_0 = \{ x_n < x_c \} \) and \( P_1 = \{ x_n \geq x_c \} \), where \( x_c \) is an arbitrary chosen division point. The complexity of the dynamical system is quantified by construction of the minimal regular language which accepts the generated sequence. The complexity and entropy (see Eq. 3.34) for the logistic map was examined in [32] using the method of regular language complexity (size of the corresponding finite automaton). It was found that the lowest values of complexity correspond to the periodic and fully chaotic regimes of the map. The highest value of the complexity occurs where the period doubling cascade of the map meets the band-merging cascade, i.e., at the border between order and chaos.

In [42] the work of Langton and Crutchfield is complemented by examining the dynamical behavior of Turing Machines. A class of 7-state 4-symbol Turing machines, which also includes Minsky's universal Turing machine [108], was used to address the question whether universal computation is found be-
between order and chaos. A large number of randomly created TMs was used to generate three different sequences: a sequence of symbols read, a sequence of states and a sequence of moves made by the TM head. For all these sequences the corresponding regular language complexity was calculated using the technique of e-machine reconstruction and plotted against its block-entropy (see Section 3.4.1). They found that the most complex TMs are indeed located at intermediate values of the entropy, including Minsky's universal TM.

Mitchell et al. reviewed this idea of computation at the “edge of chaos” and reported on experiments producing very different results from the original “edge of chaos” experiment by Packard [120]. They suggest that the interpretation of his results is not correct [110]. Those negative results did not disprove the hypothesis that computational capability can be correlated with phase transitions in CA rule space; they only showed that Packard’s results did not prove the hypothesis [109]. All in all this is still an open research question that might have a large impact on the understanding of computation in CAs

1.5 Conclusions

The use of simulation in the natural sciences is on the one hand being pushed by the advent of powerful computing systems and on the other hand being pulled by the theoretical limitations of conventional methods. The complex systems approach to understand natural phenomena is probably a result of these forces that promote computer simulation. Computer simulation and complex systems are very tightly connected.

Simulating large complex systems requires high performance parallel computer platforms. Hence, there is also a strong push for even more powerful computing systems. From a computer science point of view one would like to exploit the generality of these complex systems models for efficient simulation. This generality can be extended to include a parallel computer itself as a complex system. Thus, in order to simplify the simulation of a complex system on a parallel computer, an efficient mapping of a general complex systems model onto a specific complex system instance, i.e., a parallel computer, has to be realized.

Cellular Automata can be regarded as both a model for parallel computers as well as a general model for complex systems. With CA we can implement microscopical rules to implement a corresponding physical process. Lattice Gas Automata are a well known example where CA are successfully used to do computational fluid dynamics. On the other hand an arbitrary transition rule can be viewed as an artificial physical system with its own characteristics. It has been shown that CA are very powerful computational systems, even capable of universal computation. Wolfram has tried to classify CA dynamical behavior into 4 different classes, in close analogy with conventional dynamical systems. It has been speculated that so called class IV CA, which display “complex” dynamics, are capable of universal computation.

It is often the case that understanding the dynamics of a system composed of many interacting particles can often only be accomplished by simulation. Ir-
1.6 Outline of this thesis

In this chapter we have sketched the context of the thesis and the main problems that it will try to tackle. In the remainder of this thesis the results of an academical journey will be discussed in detail. The major philosophy behind the research approach practiced in this thesis is to stick as close as possible to the abstract notion of parallel task graphs as a mathematical framework for describing virtual particles.

Based on the abstract view of interacting virtual particles, a parallel computation framework is introduced. This framework embodies the basic research tool as well as a subject of the research in this thesis. In Chapter 2 we will discuss the design and implementation of the framework, called Parallel Cellular Automata Modeling environment (P-CAM).

An important problem in parallel computing research, static task allocation is studied. In Chapter 3 we will study the task allocation problem (TAP) from a complex systems point of view. That is by using analogies from from complex systems from physics. Various analytical and experimental tools from the field of statistical physics can be used to study the TAP. A statistical approach is taken to study theoretical and empirical aspects of a constrained class of task graphs: random task graphs.

Because optimal parallel task allocation is an intractable problem, we are confronted with a fundamental difficulty. In order to find the optimal allocation a exploration of all the possible task allocations is required. Therefore a heuristic approach is taken. Parallel task graphs are optimized by applying parallel evolutionary optimization algorithms implemented using the ViP framework of
Chapter 2. A uniform parallel optimization framework is defined in Chapter 4 and applied to both a real world application from parallel finite element simulation and the theoretical task graph model of Chapter 3. The parallel optimization framework can also be used to (sub-optimally) solve arbitrary computationally intractable problems.

To show the general applicability of the parallel simulation framework, two case studies are presented in Chapter 5. The first case is a natural solver approach to a combined aggregation and computational fluid dynamics problem. Using advanced parallelization techniques offered by P-CAM a parallel lattice Boltzmann solver is implemented. For the second case, a fully functional dynamic load balancing module is presented which is plugged into an existing parallel finite element program. Several simulation studies are presented to demonstrate the usability of the dynamic load balancing approach to parallel finite element simulations by exploiting the general applicability of the ViP framework.

An important issue in the simulation of complex systems is the order of updating individual virtual particles. Basically, we can discern two methods: synchronous and asynchronous updating. In the previous chapters we assumed synchronous updating for all simulation models, that is every virtual particle gets updated simultaneously. In Chapter 6, asynchronous updating will be considered. Parallel execution of asynchronous virtual particle models imposes a considerable amount of additional bookkeeping effort in order to assure correct dynamical behavior. Several so called parallel discrete event protocols have been defined to accomplish this task. In this chapter, the complex behavior of one of these protocols, Time Warp, is studied. Most of the results are still preliminary, but very interesting with respect to the identification of complex behavior in parallel computation.