Computing with Solitons in Bulk Media

Mariusz Hieronim Jakubowski

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
By the Department of
Computer Science

January 1999

Abstract

We propose a new physical medium for computation: colliding solitons. As particle-like nonlinear waves, solitons can carry and exchange information, thus acting as computing agents. Optical solitons have pulse widths measured in picoseconds or femtoseconds, and this approach could ultimately offer an alternative to electronics.

This work grew from an abstract computational model called the *particle machine (PM)*, which uses discrete propagating and colliding particles to compute. We show that this model is universal, and describe several efficient PM algorithms, such as convolution, certain systolic-array computations, and linear-time, arbitrary-precision arithmetic, including an iterative algorithm for division. This leads to the question of physical instantiation of PMs that could do useful computation, and optical solitons provide a natural example.

Propagating solitons carry information in their amplitudes, velocities, and phases. We show how colliding solitons can transfer such information in nontrivial ways, thus making computation possible. We characterize the state transformations caused by collisions in a particular system of coupled optical solitons — the Manakov system — and show how to implement several computations, such as a NOT processor and certain other reversible operators.

Our results resolve some questions about computational power in different soliton systems, and serve as first steps towards realizing a practical soliton-based computer.

Acknowledgments

I thank Ken Steiglitz, my advisor, for teaching me valuable and profound lessons on many fronts and for making this thesis possible. Thanks also to Rich Squier for working with us and adding to the fun. I am grateful to Moti Segev and Andy Yao for serving as readers, and to Doug Clark and Perry Cook for being on my thesis committee. Overall, Princeton University and the computer science department have been a great environment for learning and working, and I thank all faculty, staff and fellow students for making it that way. Finally, I am indebted to my family and to many people for their help and friendship — too many to list, but Dan Boneh, Pei Cao, YuQun Chen, Stef Damianakis, Chris Dunworth, Dannie Durand, Peter Frey, Liviu Iftode, Yefim Shuf, Alex Shum, Wim Sweldens, and Venkie and his family are a few.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Nature offers practically limitless means of building computers. The properties of matter and energy — for example, frequency and intensity of waves — can encode information, and the state changes resulting from mechanical, chemical, electromagnetic, and subatomic phenomena can do computation. One may say that physical systems quickly and effortlessly solve the complicated differential equations we pose to describe physical behavior; on the other hand, the ability to tap into such computational power may not immediately apply to solving "practical" problems, such as sorting a database or factoring an integer. However, transforming the inherent computing capabilities of physics into a form usable by humans — perhaps not unlike converting various forms of energy into electricity — can give us benefits and capabilities far beyond our former reach.

Modern digital processors are based on essentially one physical phenomenon, namely the propagation of electricity through wires and transistors, and have been tremendously successful owing to the miniaturization of computing elements. Such success, however, cannot continue for much longer — perhaps another few decades [48] — because media that conduct electricity and light are subject to fundamental limitations of signal speed and component size. It is thus worthwhile to search for alternate methods of harnessing physics for computation, both to overcome the quickly approaching limits of traditional approaches, and to discover means of implementing specific operations orders of magnitude faster. This dissertation studies a method of nonstandard computation, namely soliton interaction in uniform media, that may offer such benefits.

1.1 Standard and nonstandard computation

The dominant methodology for constructing modern computer processors is the "lithographic" paradigm of laying out transistors and wires in several planar layers. While this approach has worked extremely well, the abstract operation of gates is rather far removed from the underlying physics, and components that function in a more physically "natural" manner may extract more computing power out of physical behavior. For example, whereas physics is essentially reversible (at least microscopically), the operation of typical gates, such as AND and OR, is not, and energy is dissipated in the process. This places restrictions on the size and speed of components, leads to waste heat, and often necessitates that processors be equipped with cooling devices. In addition, today's gates are macroscopic devices, far removed from the atomic level; for example, an AND gate can be built today out of no fewer than 10^9 to 10^{12} atoms [101]. A current practical limit on the minimum number of electrons required to store a single bit in memory is about 6×10^5 [48].

One way bring gates and wires closer to fundamental physics is to use reversible logic. Bennett [11] proved that any Turing machine can be made reversible, thus showing that general computation can take place with far less energy dissipation than in traditional machines. Subsequent work [12, 72] has explored this approach, and the general paradigm of conservative (reversible) logic was treated in [26]. However, this approach incurs overhead in circuit speed and size, and has not found its way into mainstream electronic processor design.

Rather than attempting to improve designs based on gates and wires, researchers have recently broken away from this paradigm entirely, and explored more novel ways to exploit the computational power of physics. A number of new ways to use physical phenomena for computation have been suggested, with the hope that such approaches could lead to faster, more efficient solution of certain problems. We briefly mention several of the better known ideas.

- *Quantum computing* [17, 18, 85, 84, 95] exploits the parallelism of quantum superposition and interference to solve certain problems, most notably factoring [84], asymptotically faster than what is possible with the best known classical algorithms. In fact, problems exist which provably take exponential time on a classical machine in a certain model, but which require only polynomial time on a quantum computer [85]. This idea is still largely a thought experiment rather than a practical method of computing, but recent progress in overcoming the challenges of building a quantum computer offers some promise for future realization of such machines.

- *DNA computing* [5, 54, 107] solves certain combinatorial problems by making use of the minute size and fine-grain parallelism of interacting DNA molecules. One application of DNA to computing essentially adds a large constant factor of speed to the solution of some NP-complete problems, such as Hamiltonian path and Boolean satisfiability [5, 54]. Other approaches use DNA to simulate cellular automata [107].

- *Chemical computing* (for example, [62, 57]) is a general approach which uses reactions of various compounds to do computation. Such reactions can simulate restoring digital logic [57].

- In addition to the above, various other *analog* computing schemes and devices have been proposed (for example, [19, 103, 92]), including mechanical gadgets and electrical circuits for solving "hard" (NP-complete) problems. Although classical analog machines in general are hypothesized not to operate asymptotically faster than digital computers [103], in some cases it is a puzzling open problem to determine the exact reason for the presumed failure of a machine to solve an NP-complete problem.

- The general paradigm of *programmable matter* [100] models computation done by large numbers of microscopic agents interacting in a uniform medium. The topic studied in this dissertation — solitons propagating and colliding in a bulk medium — is one instance of this idea. Other examples include neurons communicating in the human brain, and DNA molecules interacting in an aqueous solution.

1.2 Outline of the thesis

This dissertation presents an abstract particle-based model of computation, and shows how this model can be realized with physical solitons. To make this work self-contained, we also cover some necessary background in physics and numerical methods.

In chapter 2, we introduce the abstract particle machine (PM) model of computation — a general framework, based on cellular automata (CA), for computing with particle-based phenomena. We discuss properties of the PM, and show several algorithms which demonstrate some efficient particle-based computations, including a linear-time, arbitrary-precision PM division algorithm. Much of this work was published in [89, 42].

Since prior work has shown that traditional methods (VLSI) can be used to implement CA efficiently, in chapter 3 we present and evaluate a design for building PMs with VLSI. We find that such a design offer may offer some benefits over today's microprocessors. However, the remainder of the thesis concentrates on more promising nonstandard means of realizing PMs.

Before we discuss how to use solitons to implement PMs in a nontraditional manner, in chapter 4 we present an overview of solitons and their properties. Soliton theory is a rich field with an extensive history, and we do not attempt to give a full summary; rather, we present only necessary material upon which we draw throughout the thesis. Although much of this material is well known, we present our own derivations of many facts, and give additional details and refinements.

In chapter 5 we describe numerical methods for solving nonlinear Schrödinger equations, which are physical models for solitons in various media, and which are our main platform for studying soliton-based computation. The numerical methods we use have appeared previously in the literature, but we present our own derivations and adaptations of the methods for our purposes. In addition, we describe our implementation of these methods in an interactive graphical program, `nls`.

Chapter 6 covers general issues in using solitons to do computation. We discuss the notion of information transfer in soliton systems, and introduce a restricted version of the PM called a *soliton machine (SM)*, which models soliton systems more accurately than a general PM. These results were published in [43, 44, 45].

Finally, in chapter 7 we study a specific soliton system, the Manakov system, in which we demonstrate simple examples of actual computation. This work appears in [41, 46].

We conclude with a summary, including a discussion of open problems and future work.

Chapter 2

The Particle-Machine Model of Computation

The particle machine (PM) is a model intended to capture the notion of computation by propagating and colliding particles. Like the Turing machine (TM), the PM can do general computation [89], and operates in discrete time and space. However, while the TM's tape, read-write head, and uniprocessing operations hint at mechanical, man-made origins, the PM's particle interactions and fine-grain parallelism are reminiscent of physical systems. Indeed, the PM is meant as a practical computational model of physics based on particles and particle-like phenomena.

This chapter gives an introduction to the abstract PM model, and shows several applications. Formally, we define the PM as a cellular automaton (CA) with states that represent idealized particles, and with an update rule that encodes the propagation and collision of such particles. While PMs can have any number of dimensions, here we concentrate on 1-d PMs. We discuss the issue of PM rule conflict, a problem that arises during the design of particle machines for specific computations. Finally, we present several PM algorithms, including a detailed description of an implementation of Leighton's Newtonian division algorithm [53] in the model.

As we describe later, 1-d PMs could be realized with solitons in $1+1$ (one spatial and one temporal dimension) media such as optical fibers, which have been studied extensively and are used widely in practice [38]. Later we briefly discuss the implementation of higher-dimensional PMs in $2+1$ and $3+1$ media such as glasses and crystals.

2.1 Characteristics of PMs

Particle machines are a method of incorporating the parallelism of systolic arrays [50] in hardware that is not application-specific and is easy to fabricate. The PM model, introduced and studied in [93, 89], uses colliding particles to encode computation. A PM can be realized in VLSI as a cellular automaton, and the resultant chips are locally connected, very regular (being CA), and can be concatenated with a minimum of glue logic. Thus, many VLSI chips can be strung together to provide a very long PM, which can then support many computations in parallel. What computation takes place is determined entirely by the stream of injected particles: There are no multipliers or other fixed calculating units in the machine, and the logic supports only particle propagation and collisions. While many algorithms for a PM mimic systolic arrays and achieve their parallelism, these algorithms are not hard-wired, but are "soft" in the sense that they do not determine any fixed hardware structures.

An interesting consequence of this flexibility is that the precision of fixed-point arithmetic is completely arbitrary and determined at run time by the user. A recent paper [89] shows that FIR filtering (convolution) of a continuous input stream, and arbitrarily nested combinations of fixed-point addition, subtraction, and multiplication, can all be performed in one fixed CA-based PM in time linear in the number of input bits, all with arbitrary precision. Later in this chapter we complete this suite of parallel arithmetic operations with an implementation of division that exploits the PM's flexibility by changing precision during computation.

2.1.1 Cellular automata

We briefly review the notion of the *cellular automaton (CA)* [92, 108], an abstract model used for both computation and simulation of physics. The CA was invented by von Neumann and Ulam [104] to study systems capable of universal computation and self-reproduction. Von Neumann's original CA was two-dimensional and contained 29 states per cell, but this construction has been simplified over the years. One of the simplest yet most interesting CA is the 2-d Game of Life [13], which has only 2 states per cell and yet is computation-universal.

A 1-d CA consists of a sequence of *sites*, or *cells*, which evolve in discrete time steps according to an *update rule*. The state of cell i of a 1-d CA at time $t+1$ is determined by the states of cells in the *neighborhood* of cell i at time t, the neighborhood being defined to be those cells at a distance, or *radius*, r or less of cell i. Thus, the neighborhood of a CA with radius r contains $k = 2r + 1$ cells and includes cell i itself. The operation of the CA is determined by the update rule, a function that maps states of the cells in the neighborhood of cell i at time t to the state of cell i at time $t + 1$. When the CA evolves from time t to $t + 1$, the update rule is applied simultaneously to all cells. Higher-dimensional CA operate similarly.

2.1.2 The PM model

We define the PM formally as follows:

Definition 1 A *Particle Machine (PM)* is a CA with an update rule designed to support the propagation and collision of logical *particles* in a one-dimensional homogeneous medium. Each particle has a distinct identity, which includes the particle's velocity. We think of each cell's state in a PM as a *binary occupancy vector*, in which each bit represents the presence or absence of one of n particle types (the same idea is used in lattice gasses; see, for example, [28]). The state of cell i at time *t+1* is determined by the states of cells in the *neighborhood* of cell i, where the neighborhood includes the *2r+1* cells within a distance, or *radius*, r of cell i, including cell i. In a PM, the radius is equal to the maximum velocity of any particle, plus the maximum displacement that any particle can undergo during collision.

The medium of a PM supports particles propagating with constant velocities. Two or more particles may collide; a set of *collision rules*, which are encoded by the CA update rule, specifies which particles are created, which are destroyed, and which are unaffected in collisions. A PM begins with a finite *initial configuration* of particles and evolves in discrete time steps.

For clarity when reasoning about PM operations, we can logically break up each time step of a PM's evolution into two separate phases:

- During the *propagation* phase, each particle moves from one cell to another (zero-velocity particles stay in the same cell).
- During the *collision* phase, collision rules are applied to the particle group present in each cell.

These steps are incorporated into the update rule of the CA; we do not extend the concept of CA in any way, but introduce this abstraction to simplify thinking about PM evolution.

Note that the definition of a PM precludes the presence of two or more identical particles in the same cell at a given time. While the notion of a PM could be extended to remove this limitation, the more complicated PM model does not appear to offer many additional possibilities for computation. Also note that multiple collision rules can apply to a particle group present in a given cell at a given time, but the order of rule application is undefined, leading to the *rule-compatibility* problem studied in section 2.2.

A PM, like a CA, can have a *periodic background*; that is, an infinite, periodic sequence of nonzero state values in the medium of the CA. Periodic backgrounds are sometimes used to add computational power to CA, as in [10]. To make this theoretical abstraction physically realizable in a PM, we can choose a specific cell to be a PM's *terminus*, or logical end, located away from the region in which computation occurs. We can then inject a regular, periodic sequence of particles at this cell, thus simulating a periodic background.

2.1.3 Simple computation with PMs

Fig. 2.1 shows the general arrangement of a 1-d PM. Particles are injected at one end of the one-dimensional CA, and these particles move through the medium provided by

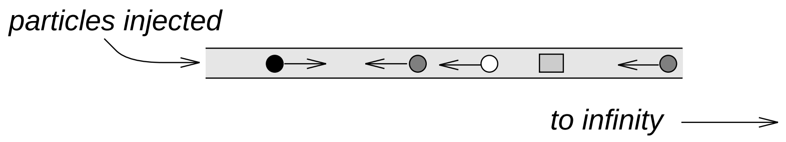

Figure 2.1: The basic conception of a particle machine

the cells. When two or more particles collide, new particles may be created, existing particles may be annihilated, or no interaction may occur, depending on the types of particles involved in the collision.

Fig. 2.2 illustrates some typical collisions when binary addition is implemented by particle collisions. This particular method of addition will be one of two described later when we develop arithmetic algorithms. The basic idea is that each addend is represented by a stream of particles containing one particle for each bit in the addend, one stream moving left and the other moving right. The two addend streams collide with a ripple-carry adder particle where the addition operation takes place. The ripple-carry particle keeps track of the current value of the carry between collisions of subsequent addend-bit particles as the streams collide least-significant-bit first. As each collision occurs, a new right-moving result-bit particle is created and the two addend particles are annihilated. Finally, a trailing "reset" particle moving right resets the ripple-carry to zero and creates an additional result-bit particle moving right.

We need to be careful to avoid confusion between the bits of the arithmetic operation and the bits in the state vector. The ripple-carry adder is represented by two particle types; the bits of the right-moving addend and the right-moving result are represented by two more particle types; the left-moving addend bits are represented by another two types; and the reset particle is represented by one additional type. Thus, the operations shown in fig. 2.2 use seven bits of the state vector. We'll denote by C_i the Boolean state vector variable for cell i. The individual bits in the state vector will be denoted by bracket notation: For instance, the state vector bit corresponding to a right-moving zero particle in cell i is denoted $C_i[0_R]$. The seven Boolean variables representing the seven particles are:

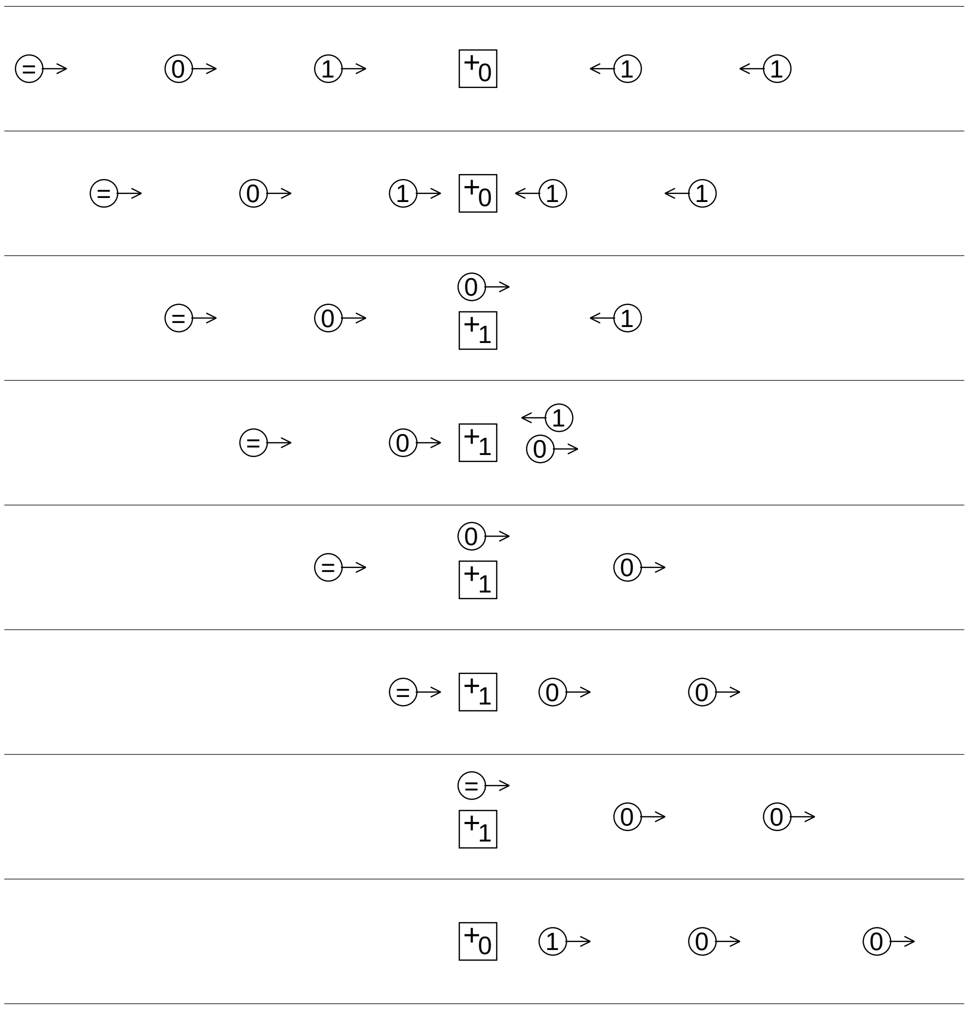

Figure 2.2: An example illustrating some typical particle collisions, and one way to perform addition in a particle machine. What is shown is actually the calculation $01_2 + 11_2 = 100_2$, implemented by having the two operands, one moving left and the other moving right, collide at a stationary "ripple-carry" particle. When the leading, least-significant bits collide (in the third row from the top of the figure), the ripple-carry particle changes its identity so that it encodes a carry bit of 1, and a right-moving sum particle representing a bit of 0 is created. The final answer emerges as the right-moving stream 100_2, and the ripple-carry particle is reset by the "equals" particle to encode a carry of 0. The bits of the two addends are annihilated when the sum and carry bits are formed. Notice that the particles are originally separated by empty cells, and that all operations can be effected by a CA with a neighborhood size of 3 (a radius of 1).

$C_i[0_R]$	right-moving zero
$C_i[0_L]$	left-moving zero
$C_i[1_R]$	right-moving one
$C_i[1_L]$	left-moving one
$C_i[+_0]$	ripple-carry adder with zero carry
$C_i[+_1]$	ripple-carry adder with one carry
$C_i[=_R]$	right-moving adder reset

All the particle interactions and transformations shown in the example can be effected in a CA with radius one; that is, using only the states of cells $i-1$, i, and $i+1$ to update the state of cell i. A typical next-state rule (as illustrated in the first collision in Fig. 2.2) therefore looks like

$$C_i[0_R]^{(t+1)} \Leftarrow (C_{i-1}[1_R] \wedge C_i[+_0] \wedge C_{i+1}[1_L]\ )^{(t)} \tag{2.1}$$

which says simply that if the colliding addends are 1 and 1, and the carry is 0, then the result bit is a right-moving 0.

Notice that using two state-vector bits to represent one data bit allows us to encode the situation when the particular data bit is simply not present. (Theoretically, it also gives us the opportunity to encode the situation when it is both 0 and 1 simultaneously, although the rules are usually such that this never occurs.) It can be very useful to know a data bit isn't present.

2.1.4 Simulation of PMs

To study the PM model and algorithms, we developed a PM simulator — a program which reads a description of a PM, and outputs the state of the PM's medium after each propagation and collision. The description is a text file which specifies the

```
% particles
C0 0 0
C1 1 0
R0 r 1
R1 R 1
L0 l -1
L1 L -1
Eq = 1

% rules
R0 L0 C0 -> ~L0
R0 L0 C1 -> ~R0 ~L0 ~C1 R1 C0
R0 L1 C0 -> ~R0 ~L1 R1
R0 L1 C1 -> ~L1
R1 L0 C0 -> ~L0
R1 L0 C1 -> ~R1 ~L0 R0
R1 L1 C0 -> ~R1 ~L1 ~C0 R0 C1
R1 L1 C1 -> ~L1
Eq C0 -> ~Eq R0
Eq C1 -> ~Eq ~C1 R1 C0

% initial configuration
Eq R0 R1 _ C0 _ L1 L1
```

Figure 2.3: Input file for simulation of a PM executing a ripple-carry addition algorithm. The file's three sections specify a PM and its input. Each line in the "particles" section lists a particle's symbolic name, the symbol printed for the particle in the simulator's output, and the particle's velocity, in order from left to right. Each line in the "rules" section specifies a collision rule, using the symbolic particle names given earlier, and with the negation symbol (˜) used to specify that a particle be destroyed; for example, the second rule in the list states that when particles $R0$, $L0$ and $C1$ collide, these particles are to be destroyed, and new particles $R1$ and $C0$ are to be created in their place. The "initial configuration" section gives the initial state of the PM's medium, using symbolic particle names to specify the addition problem $01_2 + 11_2$. Underscores (_) represent empty PM cells.

```
.=  .r  .R  .   .0  .   .L  .L  .   .

.   .=  .r  .R  .0  .L  .L  .   .   .
.   .=  .r  .R  .0  .L  .L  .   .   .

.   .   .=  .r  .0RL.L  .   .   .   .
.   .   .=  .r  .1r .L  .   .   .   .

.   .   .   .=  .1rL.r  .   .   .   .
.   .   .   .=  .1r .r  .   .   .   .

.   .   .   .   .1= .r  .r  .   .   .
.   .   .   .   .0R .r  .r  .   .   .

.   .   .   .   .0  .R  .r  .r  .   .
.   .   .   .   .0  .R  .r  .r  .   .

.   .   .   .   .0  .   .R  .r  .r  .
.   .   .   .   .0  .   .R  .r  .r  .
```

Figure 2.4: Simulator output for PM ripple-carry addition. The top line in the figure gives the initial state of the PM's medium, representing the addition problem $01_2 + 11_2$, as described in the text. Each successive pair of lines depicts the state of the medium after the propagation and collision phases of each time step. The bottom line in the picture shows the stationary 0-carry particle followed by the answer, 100_2, moving right.

PM's particles, collision rules, and the initial state of the PM's medium. All our PM algorithms have been implemented and verified using this simulator.

Fig. 2.3 shows the simulator input file for the ripple-carry addition algorithm described above, and fig. 2.4 is a copy of the simulator's output for this file. For the addition algorithm, there are seven particles, listed in the "particles" section of fig. 2.3:

- $C0$ and $C1$ — stationary carry bits (0 and 1)
- $L0$ and $L1$ — left-moving operand bits (0 and 1)
- $R0$ and $R1$ — right-moving operand bits (0 and 1)
- Eq — particle used to convert the final carry bit into the last answer bit

The algorithm includes 10 collision rules, listed in the "rules" section of fig. 2.3. The first eight of these handle the eight possibilities for adding two operand bits and a carry bit; the final two rules convert the final carry particle into the last answer particle.

2.2 Compatible collision rules

The description of a particular PM includes its collision-rule set, which determines the results of collisions of particles. These rules only partially specify their input, and more than one rule can be applicable to a collision of a set of particles. Thus, there are rule sets whose rules conflict on the outcome of some collisions; that is, one rule can state that some particle is present in the outcome, and another rule can state the opposite. If, given a particular set of inputs to the PM, one of these conflicting collisions occurs, we say the rule set is not *compatible* with respect to that set of inputs. The PM model is Turing-equivalent [89], and the general question of compatibility is undecidable, as we prove in this section. However, we also show that if the set of inputs is sufficiently constrained, as is usually the case, the constraints can be used to test compatibility in time polynomial in the number of particles and rules.

2.2.1 Collision rules

A set of rules in a PM is a relation between *preconditions* that determine the rule's applicability to collisions and *effects* that give the outcome of collisions. The domain is a set of pairs of the form (P_D, A_D), where P_D is a set of particles that must be *present* in the neighborhood in order for the rule to apply, and A_D is a set of particles that must be *absent*. We refer to particles in A_D as *particle negations*, and we consider particle negations to collide, even though the actual particles are not present. The range is a set of pairs (P_R, A_R), where the sets P_R and A_R give the particles that are created and destroyed, respectively.

2.2.2 Compatibility

When a set of particles collide in a cell, two or more collision rules may apply simultaneously to determine the results. For example, let A, B, and C denote three particles, and consider the collision rules

$$A + B \rightarrow \bar{A} + C, \tag{2.2}$$

$$A + C \rightarrow \bar{B} + \bar{C}. \tag{2.3}$$

Rule 2.2 states that when A and B collide, A is destroyed, B is unaffected, and C is created if it is not already present in the cell; rule 2.3 states that when A and C collide, A is unaffected, C is destroyed, and B is destroyed if it is present in the cell. When A, B, and C collide with one another, and possibly with additional particles, both of these rules are invoked.

For our purposes the effects of collision rules should not conflict; that is, one such rule should not destroy any particle created by another. If this condition is satisfied, the results of the collision depend only on the colliding particles, not on the order of rule application. We call a rule set of a PM *compatible with respect to a set of inputs* (or simply *compatible*) if every collision that occurs is resolved without conflicts. We use the term *input* to include the initial state of the PM's medium, and all particles subsequently injected. Note that the two collision rules given above are incompatible for the collision of A, B, and C, since rule 2.2 creates C, whereas rule 2.3 destroys C.

Given a PM, we want to be able to determine whether or not its rule set is compatible. The general problem of determining compatibility turns out to be undecidable. However, as we will see, if the PM designer provides certain additional information about the particles, the problem is solvable in time polynomial in the number of particles and rules.

Rule compatibility is undecidable

Theorem 1 *The rule-compatibility problem is undecidable.*

Proof: A straightforward reduction from the halting problem. Given a Turing machine M with an initial configuration of its tape, we transform M into a PM, and M's tape into this PM's input, in such a way that M halts if and only if the PM's rule set is not compatible.

Let S, Γ, δ, and h denote M's set of states, tape alphabet, transition function, and halt state, respectively. We begin constructing the PM P from M as follows. For each symbol $x \in \Gamma$, introduce a new stationary particle x_p. The tape of the Turing machine then maps directly into the medium of the PM; in particular, the initial configuration of M's tape corresponds to the initial state of P's medium.

We simulate the transition function δ with particles and collision rules. For each state $s \in S$, create a stationary particle, s_N. This particle is designed to perform the function of M's tape head. Assume that the transition function is defined as δ: $S \times \Gamma \rightarrow S \times \Gamma \cup \{L, R\})$, where L and R are special symbols that indicate left and right movement of the tape head. For all states s and tape symbols x such that $\delta(s, x)$ is defined:

- If $\delta(s, x) = (t, y)$, introduce a rule that transforms the stationary state particle s_N to the stationary state particle t_N, and transforms the symbol particle x_p into the symbol particle y_p. This rule simulates M's changing state and writing a symbol on its tape.

- If $\delta(s,x) = (t, L)$, introduce rules that move the state particle s_N one cell to the left and transforms s_N into t_N. These rules simulate M's changing state and moving its tape head to the left.
- If $\delta(s,x) = (t, R)$, introduce analogous rules to simulate tape head movement to the right.

To complete the construction, add the stationary particle corresponding to M's initial state to the cell corresponding to M's initial head position. The above rules must be compatible, because the medium behaves exactly like M's tape and the rules operate according to M's transition function. Finally, choose an arbitrary particle x_p and introduce two conflicting rules. One rule transforms the particle representing the halt state of M into x_p; the other rule transforms it into x_p's negation. The complete set of rules is compatible if and only if M never halts. ∎

Additional information makes compatibility decidable

Although deciding rule compatibility from only the rule set and the input is not possible in general, all is not lost. If the PM designer provides complete information about which pairs of particles can collide, we can determine compatibility with a simple polynomial-time algorithm. The PM designer usually has a good idea of which particles can collide and which cannot, even though computing this information is in general an undecidable problem. For example, a binary arithmetic algorithm most likely uses two particles representing a 0 and a 1 that never coexist in the same cell.

The information which the PM designer should provide is an exhaustive list $\mathcal{L}$ of pairs of the form (α, β), where α and β are particles or negations of particles, and α and β can collide. We assume that the designer is willing to guarantee the correctness and completeness of this information, so that if the pair (α, β) is not in the list, then α and β can never collide.

An easy way to check for rule compatibility is to ensure that each pair of rules in the rule set satisfies the following condition: If the rules apply to any collision simultaneously, then the effects of the rules do not conflict. The rule effects (P_1, A_1)

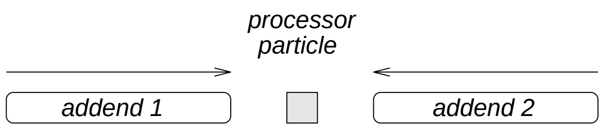

Figure 2.5: The particle configuration for adding by having the addends collide bit by bit at a single processor particle.

and (P_2, A_2) conflict if and only if one rule destroys a particle created by the other; that is, if $P_1 \cap A_2 \neq \emptyset$ or $P_2 \cap A_1 \neq \emptyset$. Two rules with preconditions $(\hat{P}_1, \hat{A}_1)$ and $(\hat{P}_2, \hat{A}_2)$ can be applicable simultaneously only if the following conditions hold:

- The rules do not conflict in their preconditions, that is, $\hat{P}_1 \cap \hat{A}_2 = \emptyset$ and $\hat{P}_2 \cap \hat{A}_1 = \emptyset$.
- The combined preconditions of the rules contain only pairs (α, β) that can collide; that is, $\alpha \in \hat{P}_1 \cup \hat{A}_1$ and $\beta \in \hat{P}_2 \cup \hat{A}_2$ only if $(\alpha, \beta) \in \mathcal{L}$.

These conditions can be checked in $O(p^2 r^2)$ time, where p and r are the numbers of particles and rules, respectively. An algorithm to do this can examine a pair of rules in $O(p^2)$ time for simultaneous applicability and conflicts, and needs to perform no more than r^2 such checks (one for each pair of rules).

2.3 Linear-time arithmetic

We conclude this chapter with a description of a linear-time PM implementation of Leighton's division algorithm [53]. Before we discuss division, however, we briefly review the implementations of addition and multiplication given in [89].

Note that in all of these implementations, we can consider velocities as relative to an arbitrary frame of reference. We can always change the frame of reference by appropriate changes in the update rules.

Fig. 2.5 shows in diagrammatical form the scheme already described in detail in fig. 2.2. Fig. 2.6 shows an alternate way to add, in which the addends are stationary, and a ripple-carry particle travels through them, taking with it the bit representing

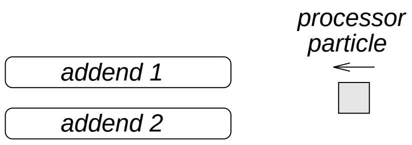

Figure 2.6: An alternate addition scheme, in which a processor travels through the addends.

the carry. We can use either scheme to add simply by injecting the appropriate stream of particles. The choice will depend on the form in which the addends happen to be available in any particular circumstance, and on the form desired for the sum. Note also that negation can be performed easily by sending a particle through a number to complement its bits, and then adding one — assuming we use two's-complement arithmetic.

Fig. 2.6 also illustrates the use of "tracks." In this case two different kinds of particles are used to store data at the same cell position, at the cost of enlarging the particle set. This turns out to be a very useful idea for implementing multiply-accumulators for FIR filtering, and feedback for IIR filtering [89]. The idea is used in the next section for implementing division.

Fig. 2.7 shows the particle arrangement for fixed-point multiplication. This mirrors a well known systolic array for the same purpose [50], but of course the structure is "soft" in the sense that it represents only the input stream of the PM that accomplishes the operation. Fig. 2.8 shows a simulation of this multiplication scheme for the product $11_2 * 11_2$. In that figure, the particles depicted by R and r represent right-moving 0 and 1 operand bits, respectively, and L and l similarly represent left-moving operand bits; p represents stationary "processor" particles in the computation region where the product is formed; c represents "carry" particles propagated during computation; and 0 and 1 represent stationary bits of the product. The top of the figure shows two operands (11_2 and 11_2) on their way towards collisions in the central computation region containing stationary "processor" particles; the bottom of the figure shows the same operands emerging unchanged from the computation, with the answer (1001_2) remaining in the central computation region.

This particular multiplication scheme uses 8 particle types and 23 collision rules;

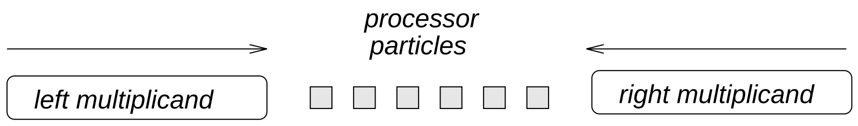

Figure 2.7: Multiplication scheme, based on a systolic array. The processor particles are stationary and the data particles collide. Product bits are stored in the identity of the processor particles, and carry bits are stored in the identity of the data particles, and thereby transported to neighbor bits.

we omit the simulator input file for the algorithm. The reader is referred to [90, 89] for more detailed descriptions and a discussion of nested operations and digital filtering.

2.3.1 Systolic arrays and PMs

Systolic arrays are regular grids, usually one- or two-dimensional, of simple, interconnected processors that work in parallel to speed up solutions to many problems, such as arithmetic and digital filtering. PMs are similar to systolic arrays in that both use fine-grain parallelism and only local communication; however, as we have explained, fine-grain systolic arrays are "fixed" in the sense that the computing elements have pre-assigned purposes, and the arrays are built to execute specific algorithms. On the other hand, PMs are "soft" in that the PM medium can support general computation, which is determined by the input stream of particles.

The multiplication scheme described above is mapped onto the PM directly from a systolic array [50]. Some types of systolic arrays can be mapped just as directly onto PMs; for example, we produced a straightforward PM simulation of the dynamic-programming algorithm for the problem of determining the minimum edit distance between two strings [73, 56]. These multiplication and minimum-edit-distance algorithms are similar in that computation using both can be expressed on a two-dimensional (space-time) grid with a linear computational wavefront [51]; moreover, it appears that any such algorithm can be mapped directly onto a PM. However, we leave the general question of mapping arbitrary systolic arrays onto PMs for further work.

```
.R   .    .R   .    .p   .p   .p   .p   .    .    .L   .    .L   .

.    .R   .    .R   .p   .p   .p   .p   .    .L   .    .L   .    .
.    .R   .    .R   .p   .p   .p   .p   .    .L   .    .L   .    .

.    .    .R   .    .Rp  .p   .p   .p   .L   .    .L   .    .    .
.    .    .R   .    .Rp  .p   .p   .p   .L   .    .L   .    .    .

.    .    .    .R   .p   .Rp  .p   .Lp  .    .L   .    .    .    .
.    .    .    .R   .p   .Rp  .p   .Lp  .    .L   .    .    .    .

.    .    .    .    .Rp  .p   .RLp .p   .L   .    .    .    .    .
.    .    .    .    .Rp  .p   .RL1 .p   .L   .    .    .    .    .

.    .    .    .    .p   .RLp .1   .RLp .    .    .    .    .    .
.    .    .    .    .p   .RL1 .1   .RL1 .    .    .    .    .    .

.    .    .    .    .Lp  .1   .RL1 .1   .R   .    .    .    .    .
.    .    .    .    .Lp  .1   .RL0c.1   .R   .    .    .    .    .

.    .    .    .L   .p   .L1c .0   .R1  .    .R   .    .    .    .
.    .    .    .L   .p   .L0c .0   .R1  .    .R   .    .    .    .

.    .    .L   .    .Lpc .0   .0   .1   .R   .    .R   .    .    .
.    .    .L   .    .L1  .0   .0   .1   .R   .    .R   .    .    .

.    .L   .    .L   .1   .0   .0   .1   .    .R   .    .R   .    .
.    .L   .    .L   .1   .0   .0   .1   .    .R   .    .R   .    .

.L   .    .L   .    .1   .0   .0   .1   .    .    .R   .    .R   .
.L   .    .L   .    .1   .0   .0   .1   .    .    .R   .    .R   .
```

Figure 2.8: Simulator output for PM multiplication. The top line in the figure gives the initial state of the PM's medium, representing the multiplication problem $11_2 * 11_2$, as described in the text. Each successive pair of lines depicts the state of the medium after the propagation and collision phases of each time step. The bottom line in the picture shows the stationary answer, 1001_2, in the central computation region, along with the unchanged operands moving away from the region.

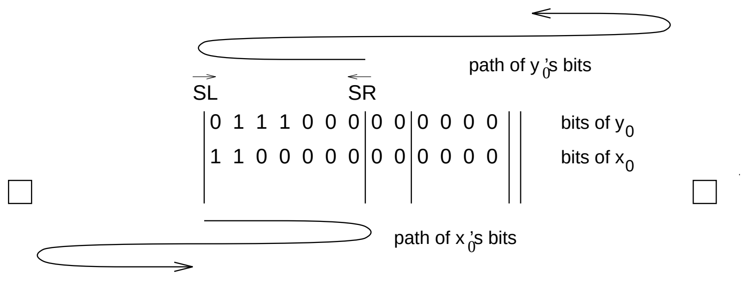

Figure 2.9: Initial configuration for division.

2.4 Linear-time division

Although division is much more complicated than the other elementary arithmetic operations, a linear-time, arbitrary-precision algorithm is possible using the particle model. The algorithm we present here, based on Newtonian iteration and described by Leighton [53],[1] calculates the reciprocal of a number x. We assume for simplicity that x is scaled so that $\frac{1}{2} \leq x < 1$. For an arbitrary division problem, we rescale the divisor by shifting its binary point left or right, calculate its reciprocal, multiply by the dividend, and finally scale the result back. Since each of these steps takes only linear time, the entire division uses only linear time.

The algorithm works as follows. Let N denote the number of bits desired in the reciprocal. For simplicity, assume N is a power of 2. Let x_i represent the i-th approximation of the reciprocal and y_i the divisor, both rounded down to $2^{i+1} + 4$ places. Beginning with $x_0 = 1.1$ in binary, the method iterates using $x_{i+1} = x_i(2 - y_i x_i)$, for $0 \leq i < \lg N$. At the end of the i-th iteration, the error is less than $2^{-2^{i+1}}$.

The particle implementation of this algorithm carefully coordinates stationary and moving particles to create a loop. The i-th iteration of the loop performs linear-time, fixed-point additions and multiplications using $2^{i+1} + 4$ bits of precision. *Marker* particles delimit the input number and indicate the bit positions that define the successive precisions required by the algorithm.

Fig. 2.9 illustrates this setup, giving the initial template for calculating the recip-

[1]The algorithm described here is actually slightly different, but it is not hard to verify its complexity and correctness.

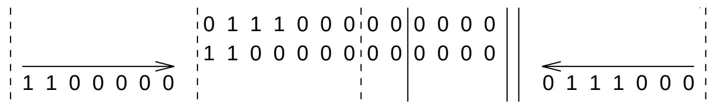

Figure 2.10: Configuration just before the first multiplication.

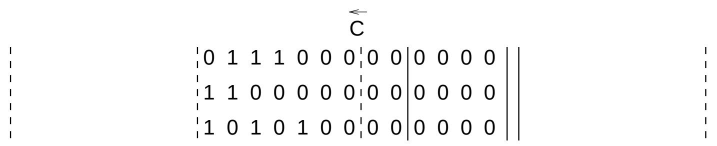

Figure 2.11: Configuration just before subtraction.

rocal of the number 7 to 8 bits of precision, that is, with error less than $\frac{1}{256}$. The outer markers enclose the binary numbers $y_0 = 0.111$, which is 7 rescaled to fit the algorithm, and $x_0 = 1.1$, the first approximation to the reciprocal. Additional markers are at bit places 6, 8, and 12 after the binary point, indicating that three iterations are required, at precisions of 6, 8, and 12 bits. Two consecutive markers terminate the number.

The only moving particles in fig. 2.9 are *sender* particles, denoted by SL and SR, whose job is to travel through the medium and send data bits to the left and right to begin an iteration of the loop. (The SL and SR particles are created by a chain reaction of previous collisions which will not be described here.) Also present in fig. 2.9 are *mirror* particles, denoted by squares, which “reflect” moving data bits; that is, a collision of a moving bit with a mirror generates a bit moving in the opposite direction and destroys the original bit.

An iteration of the loop proceeds as follows. First, the sender particles SL and SR collide with the two markers, as shown in fig. 2.9, to generate two mirrors in place of the markers. Next, SL moves right, sending bits of y_0 to the left; at the same time, SR moves left, sending bits of x_0 to the right. The moving bits of the two numbers bounce off the first mirrors, pass through the second mirrors, and finally bounce off the mirrors represented by squares in fig. 2.9. This prepares the medium for the multiplication y_0x_0. At this point all four mirrors are transformed to stationary markers. Fig. 2.10 shows the configuration just before multiplication occurs. When

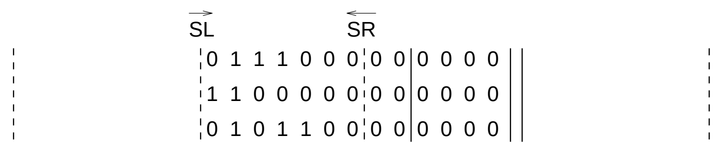

Figure 2.12: Configuration just before the second multiplication.

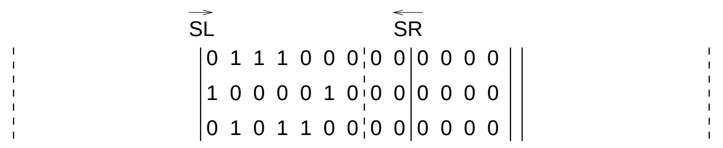

Figure 2.13: Configuration after the first iteration.

the last bits of x_0 and y_0 collide with markers, the multiplication is finished, as shown in fig. 2.11.

The next task is to calculate $2 - y_0x_0$. We do this simply by taking the 2's complement of y_0x_0, since $0 < y_ix_i < 2$. For this purpose we generate a special particle C when the last bit of y_0 collides with a marker (see fig. 2.11). The particle C is similar to a ripple-carry particle (see Figs. 2.2, 2.5, and 2.6). It first adds 1 to the complement of x_0y_0's first bit, then moves left through the rest of the bits, flipping them and adding the carry from the previous bits. Fig. 2.12 shows the result.

All that remains is to multiply this result by x_0. This proceeds in the same manner as the multiplication of x_0 by y_0. Once this is done, an iteration of the loop is complete, and the bits of x_1 replace the bits of x_0. Fig. 2.13 illustrates the configuration after completion of the first iteration.

The remaining iterations proceed exactly as described above, only with higher precisions, as determined by the markers in the template. When two consecutive markers are encountered, the division is finished, and we send a special particle through the template to restore the markers and mirrors for the next division.

Fig. 2.14 shows a picture generated by a simulation of the division example just described. The simulated PM uses 38 types of particles and 79 rules, and is capable of realizing all the applications mentioned in this chapter, including FIR and IIR filtering.

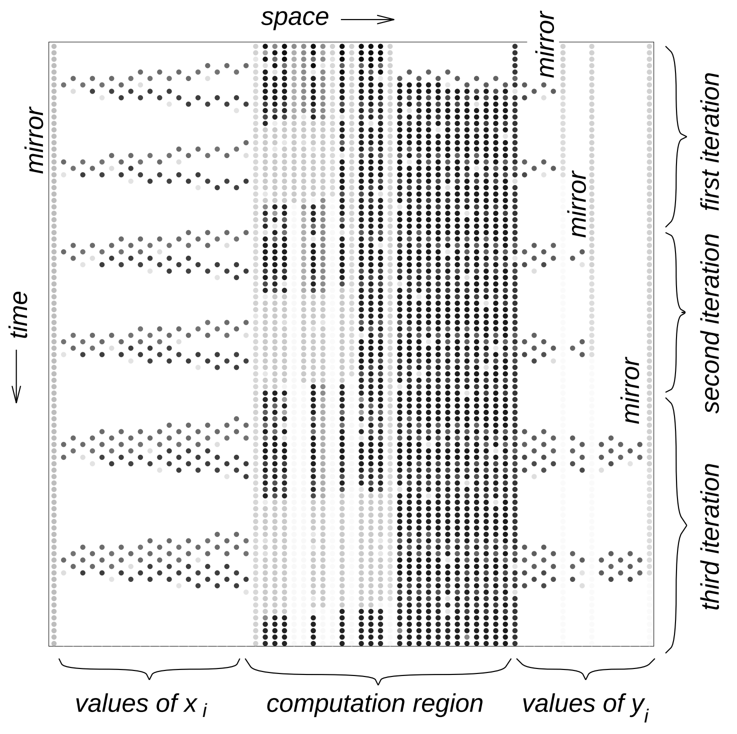

Figure 2.14: Output generated by a simulation of the division implementation. Each cell is represented by a small circle whose shading depends on which particles are present in that cell. For clarity, only every seventh generation is shown. The example is the one described in the text, 1/7.

2.5 Discussion

In this chapter we introduced the PM model of computation and described some of its important properties. Then we showed that the general problem of determining whether rules can ever conflict is undecidable, but also that this is not a serious problem in practice. Finally, we showed how to implement a linear-time division algorithm in a PM, complementing a suite of linear-time implementations of basic arithmetic: addition/subtraction and multiplication/division. As pointed out in [89], these operations can be nested and combined for parallel execution in the cells of a PM.

In a way, a PM is a programmable parallel computer without an explicit instruction set: What happens in the machine is determined by the stream of input particles. We found some methods for translating systolic arrays and other structures into particle streams, but general problems of programming a PM, such as designing higher-level languages and building compilers, are unexplored.

The three main advantages of PMs for doing parallel arithmetic are the ease of design and construction, the high degree of parallelism available through simple concatenation, and the flexibility of word length — which depends, of course, on only the particle groups entering the machine. In the next chapter we show that the simplicity of PMs allows us to use traditional VLSI technology to design PM implementations more easily than traditional processors. In addition, we argue that for certain computations, the features of PMs could make them competitive with such processors.

Chapter 3

Implementing Particle Machines with VLSI

In this chapter we take the first steps towards building physical machines based on the PM model. The PM is intended for implementation in any physical system that supports the requisite particles and collisions, but was not originally expected to lend itself well to fast and efficient realization using traditional means. Still, the tremendous success of VLSI technology, as well as previous work in using VLSI to implement CA and systolic arrays [102, 50, 56], suggest that an evaluation of VLSI for building PMs is in order.

Much like systolic arrays, CA can be realized in hardware as regular arrays, usually one- or two-dimensional, of locally interconnected, uniform processors. Each such processor needs only to implement the CA's update rule and to synchronize operation with all other processors. The simplicity of such processors, combined with their parallel operation, can lead to CA machines, such as the MIT Cellular-Automaton Machine (CAM) computer [102, 99], that simulate CA much faster than conventional uni- or multiprocessors.

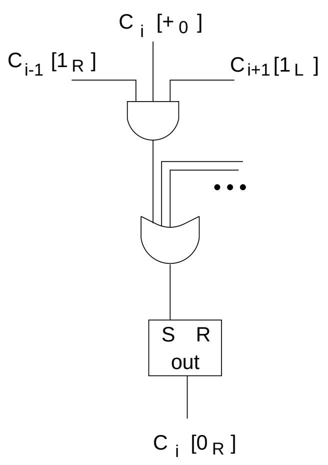

Figure 3.1: The logic fragment of an implementation of the rule in eq. 2.1. The conditions create a right-moving 0.

3.1 VLSI size and performance estimate

We base our estimate of VLSI size and performance on a straightforward realization of the PM rule set as suggested by fig. 3.1. In that figure, the rule in eq. 2.1 is implemented directly in random logic feeding the inputs to an S-R flip-flop that stores one bit of a cell's state vector. This makes it easy to map the rules into logic generally; each rule that creates or destroys a particle sets or resets its corresponding bit. Fig. 3.2 shows the general logic fanning into a state bit.

In a practical VLSI implementation the rules would probably be combined and realized in a programmable logic array (PLA). The following area estimate assumes this is done. If PMs prove practically useful, it may be worth investing considerable effort in optimizing the layout of a PM for a rule set that is sufficiently powerful to implement a wide variety of computations. After all, this design need to be done only once, and the resulting chip would be useful in many applications. As with memory cells, only a single cell needs to be designed, which can then be replicated. The chips themselves can be concatenated to form very long — and hence highly parallel — machines.

Our layout of a row of cells is shown in fig. 3.3. Here each cell has p bits in its state vector, thus supporting p particle types, and contains a logic block, a bank of flip-

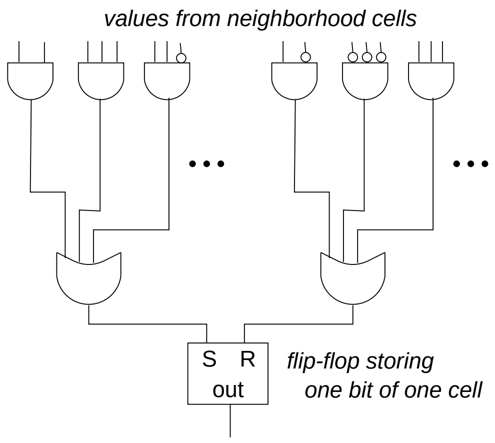
values from neighborhood cells
S R
out
flip-flop storing one bit of one cell

Figure 3.2: A possible layout plan for a PM.

flops, and wiring connecting logic inputs and outputs to flip-flop inputs and outputs. Our estimate of the area required for these elements uses a rough approximation to the space required to route a single signal wire in modern VLSI [106]: about $\alpha = 6\lambda$, where $\lambda = 0.2\mu$. Thus, a wiring channel containing n wires we estimate to be $n6\lambda$ across. We allow four times as much space per signal wire for PLA signal wires, or $\beta = 24\lambda$ per PLA wire.

As laid out in fig. 3.3, a cell requires vertical space for two wiring channels. In addition, we must fit in the larger of either a bank of p flip-flops or the vertical span of the PLA. Since we assume a simple layout, the PLA is the larger structure and we ignore the flip-flops in the vertical direction. The PLA contains p input wires, $p/3$ from the cell's own flip-flops and $p/3$ each from the two neighbor cells, and p output wires, giving $2p$ total PLA wires, counting vertically. The total height of a cell is then $(2/3)p\alpha + 2p\beta$.

Horizontally, a cell must accommodate the PLA, two wiring channels, and the flip-flops. The PLA requires roughly as many wires as minterms. Estimating an average of four minterms per output wire, we get $4p$ horizontal PLA wires. The width of the flip-flops is about 10 PLA wires. A cell's width is then $(2/3)p\alpha + 4p\beta + 10\beta$.

We establish the number of particles required, p, for a general PM. Consider a single track of data. In general for a single track we need data bits that travel in either direction or remain stationary. This requires six particles: 0 and 1 moving left, 0 and

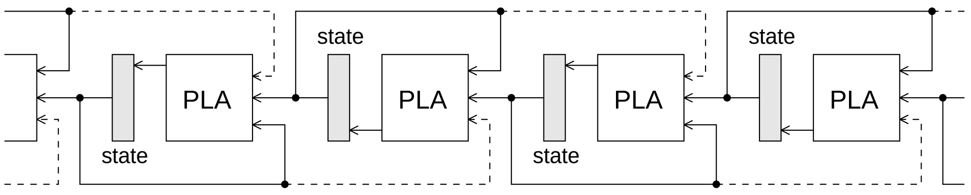

Figure 3.3: The general layout of cells in the CA for a PM. The bits in the state vector are shown as shaded registers. Each such register supplies data to the update logic for its own cell, and those of its nearest neighbor cells. Connections to right neighbors are shown as dashed lines.

1 moving right, and stationary 0 and 1. Three data tracks suffice for all applications we have tried. For each data track we assume about six operator particles are needed. This gives us $p = 36$ total operator and data particles. Using the above area estimates and given a chip one centimeter on a side, we find there is room for about 300 cells on a single chip for a PM supporting 36 particles.

Now we estimate the potential parallel performance. Using the multiplication scheme shown earlier in fig. 2.7 we need about $2n$ cells to hold a single n-bit operand moving either left or right. The processor particles require $2n$ cells. This gives $6n$ total cells between the beginning of one multiplication and the beginning of the next multiplication. Supposing we have 32-bit operands, this means we can fit about 1.56 multiplications operating concurrently on a single chip. The cell logic is very simple, so a conservative estimate of clock speed is 500 Mhz. A multiplication completes in $2n$ clock ticks. This gives us about 12 million 32-bit integer multiplies per second per chip. Using logic optimization and other layout and performance refinements in the chip design, we might expect to get a factor of 5 to 10 improvement over this estimate.

3.2 Discussion

We conclude that a PM implemented in VLSI does not offer benefits in terms of speed or circuit size, although such a machine's performance can be competitive with that of today's processors in special applications. As an example, for 32-bit multiplications the Intel Pentium Pro processor has a latency of 4 cycles and a throughput of one operation per cycle [1]. Running at 200 MHz, such a processor is capable of executing 200 million 32-bit multiplications per second, and we have verified that this is true in practice if all operands are in registers; however, if operands are in cache, our tests revealed that this number falls to about 80 million multiplications per second, and to about 50 million if the operands are in memory but not in cache. Pentium processors with the MMX extensions, as well as specialized digital signal processors (DSPs), can achieve speeds several times these numbers [79]. This compares with our estimated maximum of about 120 million multiplications per second on a single PM chip; however, we consider such chips as bulk computing elements which, if we ignore interconnection and scalability issues, can simply be concatenated to yield more powerful PMs.

As we noted earlier, the PM was not originally intended for implementation with traditional technology, but rather with particle-like physical phenomena. The rest of this dissertation is devoted to a study of one such phenomenon — the soliton — and to its use in realizing PMs. In the next chapter, we introduce solitons, describe their characteristics and phenomenology, and explain methods we use later to study solitons as means of implementing PMs.

Chapter 4

Solitons

Certain nonlinear partial differential equations (PDEs) give rise to *solitons* [55, 76, 20] — particle-like solitary waves that propagate without decay in homogeneous materials and survive collisions with shape and velocity intact. Soliton-supporting media include shallow water, electrical transmission lines, chains of masses connected by nonlinear springs, optical fibers, and plasmas [76, 20]. In addition, discrete solitons exist in certain CA [68, 93], and behave much like the continuous solitons described by PDEs.[1]

This chapter provides an overview of solitons and their phenomenology. Most of this material is well known, but serves as background for later chapters, in which we will use solitons as particles in PMs. Our main platform for studying soliton properties is the nonlinear Schrödinger (NLS) equation, a model for soliton behavior in many physical systems. After a general introduction to soliton systems, we describe specific NLS systems in some detail.

4.1 Soliton systems

Probably the earliest recorded observation of solitons was published by J. Scott-Russell, a Scottish scientist, in 1844. Russell's first encounter with the "great wave of translation," as he named the phenomenon he witnessed, is perhaps best recounted

[1]To our knowledge, the precise nature of this connection between CA and PDEs is unexplained.

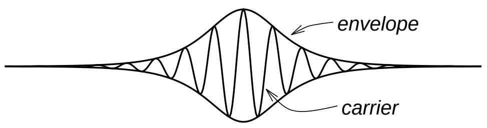

Figure 4.1: An envelope soliton.

in his own words:

I was observing the motion of a boat which was rapidly drawn along a narrow channel by a pair of horses, when the boat suddenly stopped — not so the mass of water in the channel which it had put in motion; it accumulated round the prow of the vessel in a state of violent agitation, then suddenly leaving it behind, rolled forward with great velocity, assuming the form of a large solitary elevation, a rounded, smooth and well-defined heap of water, which continued its course along the channel apparently without change of form or diminution of speed. I followed it on horseback, and overtook it still rolling at a rate of some eight or nine miles an hour, preserving its original figure some thirty feet long and a foot to a foot and a half in height. Its height gradually diminished, and after a chase of one or two miles I lost it in the windings of the channel. Such, in the month of August 1834, was my first chance interview with that singular and beautiful phenomenon....[77]

Scott-Russell did extensive empirical studies of solitons, but the mathematics describing these nonlinear waves was developed some decades later. In 1895, Korteweg and de Vries proposed their famous equation for the propagation of long waves in shallow water [49],

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{\partial^3 u}{\partial x^3} = 0, \tag{4.1}$$

where u is the amplitude of a wave, x is space, and t is time. This equation admits single-soliton solutions, which Scott-Russell observed empirically, and which have the form

$$u(x,t) = 3c\ \mathrm{sech}^2[\sqrt{c}(x-ct)/2], \tag{4.2}$$

where c is the soliton's velocity, a positive real number [55].

The KdV equation (eq. 4.1) states that the rate of change of a wave's amplitude is determined by two physical effects:

- Dispersion, modeled by the term $\frac{\partial^3 u}{\partial x^3}$, causes separation of the wave into elementary components having velocities proportional to amplitudes.

- Nonlinearity, modeled by the term $u\frac{\partial u}{\partial x}$, slows down high-amplitude wave components, which are dragged along the shallow bottom. This phenomenon is commonly observed at an ocean beach when waves break as they approach land.

When these two effects cancel out, as stated by eq. 4.1, the result is a soliton — a stable wave packet whose components all travel at the same velocity. In general, solitons described by any nonlinear PDE arise when a dispersive effect and a nonlinearity balance each other.

The term "soliton" was coined in 1965 by Zabusky and Kruskal [110], who performed numerical studies of the KdV equation, and found particle-like waves which retained their shapes and velocities after collisions. This work also explained a remarkable observation made in 1955 by Fermi, Pasta, and Ulam [21], who studied the flow of energy in a chain of equal masses connected by nonlinear springs, and discovered that energy injected into this system was not eventually shared equally by all modes of the system. Instead, as Zabusky and Kruskal found, such energy decomposes into a number of solitons which propagate and collide along the chain, and this behavior is modeled by the KdV equation. The same equation also describes other physical systems, including electrical transmission lines [40, 69] and mechanical systems such as the Toda lattice [98], an infinite version of the Fermi-Pasta-Ulam chain of masses.

4.1.1 Integrable soliton systems

Soliton equations can be *integrable*, meaning that the solution spaces of such equations contain an infinite number of conserved quantities, such as energy. The term *integrable*, or *completely integrable*, has also been used in the literature to mean *solvable by the inverse scattering transform (IST)* [30, 4, 2, 80, 20], a remarkable method which identifies soliton and dispersive components in the initial-value problem for certain nonlinear PDEs. It is unclear whether or not these two meanings are equivalent; we use *integrable* to refer to systems for which exact multisoliton solutions can be obtained with the IST.

Solitons arising from real-valued integrable PDEs, such as the KdV and sine-Gordon equations [76, 75], are uniquely identified by their constant velocities, which are determined by their amplitudes. Complex-valued integrable PDEs, such as the cubic nonlinear Schrödinger (cubic-NLS) equation [76, 33], support *envelope* solitons, or wave packets consisting of sinusoidal *carrier waves* modulated by surrounding *envelopes* (see fig. 4.1). In the cubic-NLS system, a carrier wave is characterized by an amplitude, a frequency, and a *phase* (a periodic value that changes according to the wave's *phase velocity*, a function of the wave's amplitude). An envelope soliton travels at its *group velocity*, which is determined by its amplitude and by the frequency of its carrier wave.

Solitons in integrable systems can be single wave packets, as with the KdV and cubic-NLS equations; we refer to such solitons as *single-mode*. In addition, integrable solitons can consist of coupled components in two or more modes of a system, as with the coupled Manakov equations we study in chapter 7. Compared with single-mode solitons, *multi-mode* (or *vector*) solitons can behave in more complex ways, and can be more suitable for implementing computation, as we explain later in detail.

4.1.2 Nonintegrable soliton systems

Certain nonintegrable PDEs support soliton-like waves[2] with behavior more complex than that of integrable solitons. Examples include the Klein-Gordon [3], the logarithmically nonlinear Schrödinger (log-NLS) [14, 15, 66], and the saturable nonlinear Schrödinger (sat-NLS) [59] equations. Systems described by such equations exhibit behavior more complex, and potentially better suited for implementing computation, than the behavior of integrable systems; however, nonintegrable systems are less analytically tractable, and require the usage of numerical methods to study even simple collision properties. In addition, collisions can generate varying amounts of *radiation*, or dissipation of energy from solitons.

4.1.3 Soliton behavior

Collisions

Multisoliton solutions of various integrable systems [34, 35, 36, 4] describe the behavior of two or more integrable solitons in collision. Such solutions imply many interesting properties of solitons, such as their particle-like nature: Integrable solitons do not decay with time, and collide *elastically.*[3] Single-mode integrable solitons retain their respective energies and identities after collisions, undergoing only a phase shift and a displacement in space (see fig. 6.13). Both phase shift and spatial displacement are simple functions of amplitudes and frequencies of soliton carrier waves. Arbitrary numbers of single-mode solitons may collide simultaneously, but the cumulative phase shift and displacement of each soliton are obtained by summing the shifts and displacements that result from the soliton's pairwise collisions with all

[2]We refer to such waves as *solitons*, as is often done in the literature.

[3]We use the term *elastic* to mean *non-radiating*, or *conserving total energy.*

others; that is, phase shifts and displacements are additive. These phase shifts and displacements can be calculated easily from multisoliton solutions.

For nonintegrable systems, numerical results show that collisions can change any soliton parameter, including amplitude, velocity, and phase (see figs. 6.14, 6.4, 6.5, and 6.6). Such collisions may be inelastic or *near-elastic*; that is, colliding solitons can dissipate their energy by producing varying amounts of radiation (see fig. 6.7), which erodes other solitons and may eventually lead to complete decay of useful information in the system. To our knowledge, it is an open question whether or not there exists a nonintegrable system with perfectly elastic, or non-radiating, collisions. The nonintegrable systems we consider later, described by the sat-NLS and log-NLS equations, support near-elastic collisions with negligible radiation, and may support perfectly elastic, non-radiating collisions as well.

In the integrable multi-mode Manakov system, which we study in chapter 7, two-soliton collisions generate no radiation, and soliton velocities do not change, as exact two-soliton solutions [71] imply. However, the amplitudes of soliton components in each mode can change as a result of collisions, much like in nonintegrable systems. This makes the Manakov system an especially promising candidate for implementing PMs, a matter we consider fully in chapter 7.

Breathers

Integrable PDEs such as the cubic-NLS and sine-Gordon equations [9, 36, 75], as well as nonintegrable PDEs, support *bound-state* solitons, or *breathers*, which often consist of two or more equal-velocity solitons moving close together in perpetual collision. In general, breathers are solitons with pulsating amplitudes (see the two solitons at the bottom of fig 6.4). In the cubic-NLS equation, two equal-velocity solitons attract each other with a force that weakens exponentially as the distance between the solitons; these solitons form a breather by alternately colliding and separating, with the time between successive collisions, or *oscillation period* of the breather, increasing exponentially as the initial separation between the solitons. According to [96], "the interaction [between equal-velocity solitons] can effectively be avoided if the solitons are widely separated."

Our models ignore attraction between equal-velocity solitons. We leave it for future work to determine exactly what happens in collisions of breathers with solitons and with other breathers. However, numerical experiments in the cubic-NLS system suggest that such collisions result only in additive phase shifts and displacements of all colliding solitons. If this is true in general, then all results in this thesis hold also for soliton systems in which breathers are allowed.

Fusion

In some nonintegrable systems, such as the sat-NLS equation, two or more colliding solitons sometimes "fuse" into a single particle (see fig. 6.16). Such solitons typically begin with low relative speeds, and collide to generate a breather accompanied by radiation. Our models ignore fusion, because our numerical experiments have shown that fused breathers are not stable enough for computation. However, further work is required to investigate potential computational applications of fusion.

Birth

The phenomenon of birth involves extra solitons that form as a result of some collisions in nonintegrable systems. For example, in the log-NLS system, a collision of two gaussons can result in three gaussons after impact (see fig. 6.8). We do not account for birth in our models, because this phenomenon is not regular and predictable enough to be modeled with PM collisions, as our numerical studies have revealed. Further study is needed to determine how useful birth of new solitons is for implementing computation.

4.2 The NLS equation and its solutions

The 1-d nonlinear Schrödinger (NLS) equation [76, 96, 65, 38] is a model for soliton behavior in various media, such as optical fibers, waveguides, and crystals. Various forms of the NLS equation, both integrable and nonintegrable, describe physical systems commonly built and studied in laboratories, and the NLS system offers promise for realization of PM models. The NLS equation is our primary system for studying soliton-based computation.

4.2.1 General form of NLS

The 1-d NLS equation can be written as [59, 78]

$$-i\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + N(|u|)u. \tag{4.3}$$

Here x is space, t is time, u is the complex amplitude of waves described by the equation, D is a real constant, and $N(|u|)$ is a nonlinear function of $|u|$. For example, for cubic-NLS, $N(|u|) = |u|^2$, and for sat-NLS, $N(|u|) = m + k|u|^2/(1+|u|^2)$, where k and m are real constants.

4.2.2 Single-soliton solutions of NLS

The nonlinearity $N(|u|)$ determines the integrability of eq. 4.3, and the existence of closed-form solitary-wave solutions. To find solitary waves, either analytically or numerically, we assume that each such wave consists of an envelope modulating a sinusoidal carrier wave. Following [76], we make the ansatz

$$u(x,t) = \Phi(x - u_e t)e^{i\theta(x-u_c t)}, \tag{4.4}$$

where $\Phi(x-u_e t)$ is the envelope, $e^{i\theta(x-u_c t)}$ is the carrier, and u_e and u_c are the envelope and carrier velocities, respectively. We separate the real and imaginary parts of the result, obtaining

$$\begin{aligned} D\Phi_{xx} - D\Phi\theta_x^2 + u_c\Phi\theta_x + N(|\Phi|)\Phi &= 0, & (4.5)\\ D\Phi\theta_{xx} + 2D\Phi_x\theta_x - u_e\Phi_x &= 0. & (4.6)\end{aligned}$$

We solve eq. 4.6 for the carrier function θ, which is given by

$$\theta(x - u_c t) = \frac{u_e}{2D}(x - u_c t) + \phi_0, \tag{4.7}$$

where ϕ_0 is an arbitrary constant. Substituting this value of θ into eq. 4.5 allows to express the envelope function Φ as

$$\xi = \int_{\Phi(0)}^{\Phi(\xi)} \frac{d\Phi}{\sqrt{\frac{\alpha\Phi^2}{D} - \frac{2}{D}\int N(|\Phi|)\Phi d\Phi}}, \tag{4.8}$$

where $\xi = x - u_e t$ and $\alpha = (u_e^2 - 2u_e u_c)/(4D)$.

If the integral in eq. 4.8 can be evaluated analytically and used to solve eq. 4.8 for the envelope function $\Phi(\xi)$, then eq. 4.4 gives an exact expression for a solitary wave, as is the case with the cubic-NLS equation. Otherwise the integral and $\Phi(\xi)$ can be computed numerically, using boundary conditions chosen to yield solitary waves, as we will later show by example.

4.2.3 Multisoliton solutions of NLS

When eq. 4.3 is integrable, exact solutions describing the behavior of multiple solitons can be found using the inverse scattering transform [30, 4, 2, 80, 20]. For both integrable and nonintegrable variants of eq. 4.3, multisoliton solutions can be constructed numerically by plotting two or more single solitons, well separated in space, in a discrete 1-d grid. The *weak nonlinear superposition principle* for the NLS equation [14] states that a superposition of solitons in continuous space is a solution of the equation, provided that the solitons are sufficiently distant from one another. If the solitons have different velocities, applying a numerical method to propagate the system forward in time allows us to observe and study multisoliton collisions. We study such numerical methods in detail in chapter 5.

4.2.4 Some NLS systems and solitons

In this section we present the mathematical forms of several NLS systems and show how to obtain single-soliton solutions analytically (when possible) and numerically. These equations have been proposed as models for the behavior of light and other forms of energy in media such as waveguides, glasses, and crystals. In the next section we describe some applications of these equations to simulation of physical systems.

k-NLS

The one-dimensional k-NLS equation has the following form [59]:

$$-i\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + b|u|^n u. \tag{4.9}$$

Here $k = n+1$, and b is real and positive. When $k = 3$, $D = 1/2$, and $b = 1$, eq. 4.9 is a frequently used form of the well known integrable cubic-NLS equation [38].

To find single-soliton solutions of this equation, we use the procedure from section 4.2.1. Soliton solutions have the form of eq. 4.4, where the carrier θ is immediately given by eq. 4.7. The envelope Φ can be found from eq. 4.8, which in this case gives

$$\xi = \int_{\Phi(0)}^{\Phi(\xi)} \frac{d\Phi}{\sqrt{\frac{\alpha}{D}\Phi^2 - \frac{2b}{D}\int \Phi^{n+1} d\Phi}}. \tag{4.10}$$

Simplifying, we get

$$\begin{aligned} i\xi &= \int_{\Phi(0)}^{\Phi(\xi)} \frac{d\Phi}{\Phi\sqrt{\frac{2b}{D(n+2)}\Phi^n - \frac{\alpha}{D}}} &(4.11)\\ &= \frac{2}{n\sqrt{\alpha/D}} \ \sec^{-1}\sqrt{\frac{2b}{\alpha(n+2)}\Phi^n}. &(4.12)\end{aligned}$$

We solve this for Φ to obtain

$$\Phi = \left(\frac{\alpha(n+2)}{2b}\right)^{1/n} \operatorname{sech}^{2/n}\left(\frac{n\sqrt{\alpha/D}}{2}\xi\right). \tag{4.13}$$

Multiplying the envelope by the carrier, we obtain an expression for the k-NLS soliton,

$$u(x,t) = e^{i(\frac{u_e}{2D}(x-u_c t)+\phi_0)} \left(\frac{\alpha(n+2)}{2b}\right)^{1/n} \operatorname{sech}^{2/n}\left(\frac{n\sqrt{\alpha/D}}{2}\xi\right). \tag{4.14}$$

Note that each k-NLS soliton is specified by three parameters: envelope velocity (u_e), carrier velocity (u_c), and initial phase (ϕ_0).

Sat-NLS

We consider the following form of the sat-NLS equation [59]:

$$-i\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + (m + \frac{k|u|^2}{1+|u|^2})u. \tag{4.15}$$

Here m and k are real constants. Systems of two or more sat-NLS equations can be coupled by their nonlinear terms, as in

$$\begin{aligned} -i\frac{\partial u}{\partial t} &= \frac{\partial^2 u}{\partial x^2} + (m + \frac{k(|u|^2+|v|^2)}{1+|u|^2+|v|^2})u, \\ -i\frac{\partial v}{\partial t} &= \frac{\partial^2 v}{\partial x^2} + (m + \frac{k(|u|^2+|v|^2)}{1+|u|^2+|v|^2})v. \end{aligned} \tag{4.16}$$

This example of coupling extends to any number of equations.

We explain how to use the procedure from section 4.2.1 to obtain sat-NLS solitons, which can be used in numerical simulations of both single and coupled sat-NLS equations. For sat-NLS, the integral given by eq. 4.17 does not seem to admit a closed-form expression, and no exact expressions have been found for these solitons, to our knowledge. Thus, we use numerical integration in the procedure described below.

As before, a soliton solution consists of an envelope modulating a carrier, as given by eq. 4.4. The envelope Φ can be found from eq. 4.8, which simplifies to

$$\xi = \int_{\Phi(0)}^{\Phi(\xi)} \frac{d\Phi}{\sqrt{c\Phi^2 + k\log(1+\Phi^2)}}, \tag{4.17}$$

where $c = \alpha - m - k$. We evaluate the above integral numerically using the boundary conditions $\Phi(0) = A$ and $\Phi(\pm\infty) = 0$, where A is the maximum amplitude of the envelope and is determined by u_c and u_e; note that any soliton must satisfy these

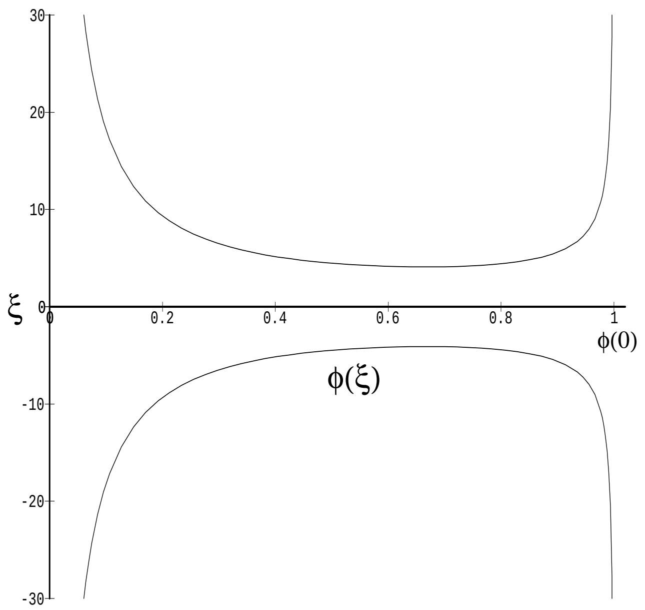

Figure 4.2: Integrand for computing a sat-NLS envelope with $A = u_e = 1$.

conditions. The integration yields ξ as a function of the envelope $\Phi(\xi)$. We invert the result of the integration to compute the envelope $\Phi(\xi)$ as a function of ξ.

As an example of this procedure, we show how to generate a sat-NLS soliton with amplitude $A = 1$ and velocity $u_e = 1$, assuming $m = -1$ and $k = 1$ in eq. 4.15. We obtain the soliton's carrier immediately from eq. 4.7. To find the soliton's envelope, we consider the integral given by eq. 4.17. We need to evaluate this integral only in the first quadrant, since the integration limits, which correspond to the envelope's amplitude at various points, are always nonnegative. Observe that if

$$\sqrt{c\Phi^2 + k\log(1+\Phi^2)} = 0 \tag{4.18}$$

for $\Phi = 0$ and $\Phi = A$, then in the first quadrant, the integral is defined only for $0 < \Phi < A$, and this allows us to satisfy the boundary conditions necessary for soliton formation, as we will see. Straightforward algebra shows that eq. 4.18 is

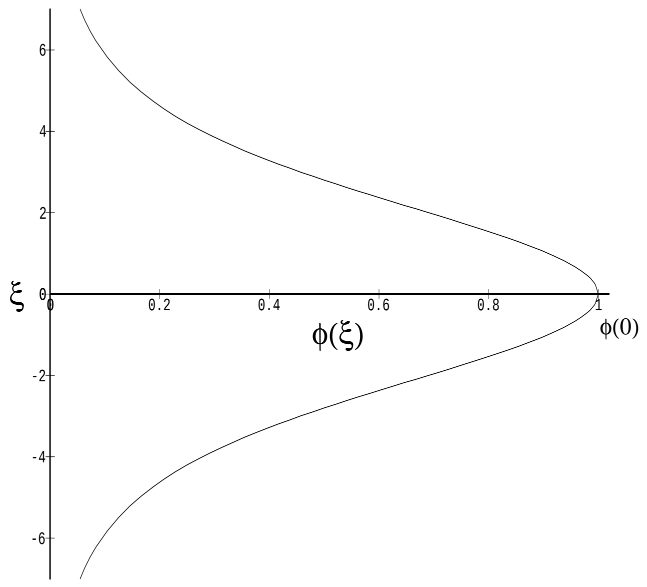

Figure 4.3: The graph of fig. 4.2 after numerical integration: a sat-NLS soliton rotated by 90 degrees.

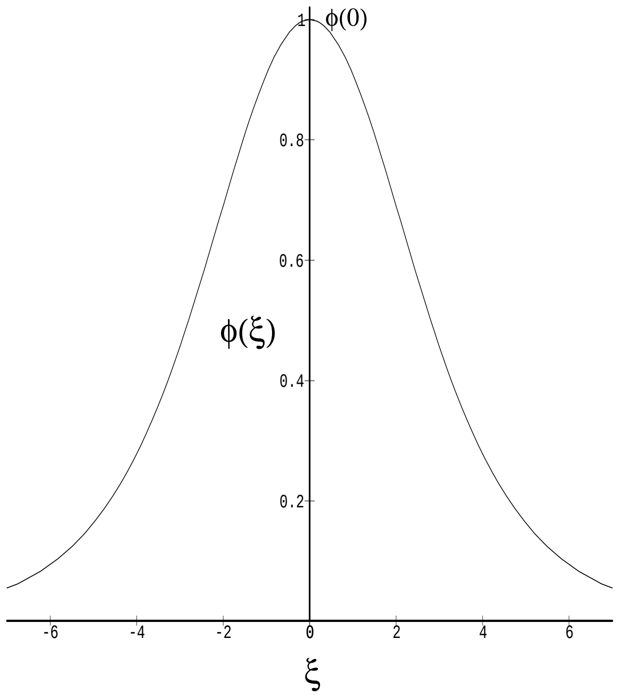

Figure 4.4: The graph of fig. 4.3 after inversion (rotation by 90 degrees): a sat-NLS soliton.

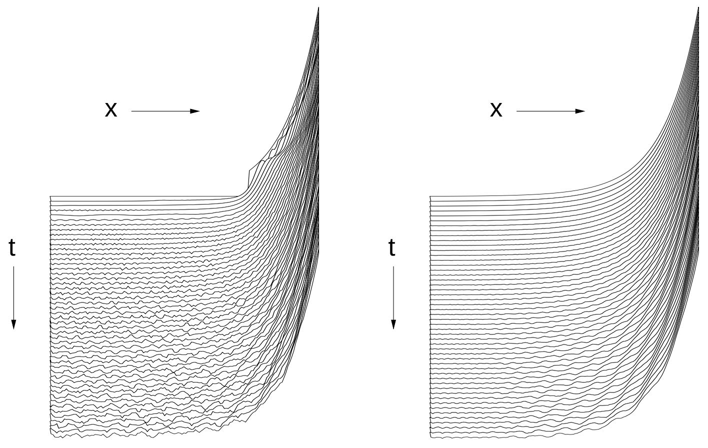

Figure 4.5: Highly magnified left edges of propagating sat-NLS solitons. The left graph shows a soliton obtained by numerical integration, as described in the text, without interpolation of the soliton's tails. Note the amplitude discontinuity in the initial conditions at the top of the graph. The right graph shows the same soliton, but with its tails interpolated using an exponentially decaying function. Note the lesser amount of radiation as compared with the left graph.

always true for $\Phi = 0$, and is true for $\Phi = A$ when

$$u_c = 2\frac{\frac{k}{A^2}\log(1+A^2) - k(m+1) + \frac{u_e^2}{4}}{u_e}. \tag{4.19}$$

When $A = 1$ and $u_e = 1$, this gives $u_c = 2\log(2) + 1/2$, and fig. 4.2 shows a first-quadrant graph of the integrand from eq. 4.17. We now integrate this numerically from $\Phi(0)$ to $\Phi(\xi)$, where $\Phi(\xi)$ ranges from A to 0 in small samples along the Φ axis. This integration computes the proper ξ coordinate for each amplitude sample $\Phi(\xi)$; fig. 4.3 shows the result. Note that the integrand contains a square root; the positive and negative values of this root produce the respective symmetric "halves" of the soliton below and above the Φ axis. Finally, we numerically invert, or rotate by 90 degrees, the soliton in fig. 4.3 to obtain the properly oriented soliton depicted in fig. 4.4.

As $\Phi(\xi)$ approaches 0, ξ grows very quickly; that is, a soliton's amplitude decays slowly as the distance from the soliton's peak increases. This is because the integrand given by eq. 4.17 has a singularity at $\Phi(\xi) = 0$, as fig. 4.2 shows. This singularity introduces a numerical-accuracy issue when the integral is computed with $\Phi(\xi)$ close to zero; if we subdivide the interval $0..A$ on the Φ axis into N intervals for numerical integration, then for

$$\Phi(\xi) = A/N, 2A/N, ..., kA/N, \tag{4.20}$$

the integrals from $\Phi(0)$ to $\Phi(\xi)$ will be large and widely separated values, which are the ξ coordinates of the envelope locations where the amplitude is $A/N, 2A/N, ...$ This means that while the amplitude decreases very slowly from $2A/N$ to A/N, the amplitude drops instantly from A/N to 0, since A/N is the minimum interval for numerical integration; the left half of fig 4.5 shows an example of this. Note the discontinuity in amplitude at the top of the figure, and the radiation generated as a result of this problem, which occurs even when N is large ($A = 1$ and $N = 65536$ in this example). One solution is to interpolate the amplitude from N/A to 0 using an exponential function; the right half of fig. 4.5 shows the effect when the amplitude is interpolated with the function $\Phi = a(1.7^{\xi} - 1)$, where a is computed so that $\Phi = 0$ at the left edge of the grid, and $\Phi = A/N$ at the grid location corresponding to the amplitude A/N. Note that the interpolation results in less radiation generated as a result of numerical simulation, and subsequently leads to greater accuracy of numerical results.

We remark that sat-NLS solitary waves are characterized by four parameters: amplitude (A), envelope velocity (u_e), carrier velocity (u_c), and phase (ϕ_0). Using eq. 4.17, it can be shown that any two of A, u_e, and u_c determine the third, which can be computed with eq. 4.19. We may choose ϕ_0 freely, so that there are three degrees of freedom in choosing the initial state of a sat-NLS solitary wave.

Log-NLS

We consider the following form of the log-NLS equation [14, 15, 66]:

$$-i\frac{\partial u}{\partial t} = \frac{h}{2m}\frac{\partial^2 u}{\partial x^2} + \frac{b}{h}\ln{(a|u|^2)}u. \tag{4.21}$$

Here a and b are constants computed from two real parameters called *gausson width* (l) and *gausson rest energy* (E_0), according to

$$b = \frac{h^2}{2ml^2}, \tag{4.22}$$

$$a = e^{1-E_0/b}. \tag{4.23}$$

Soliton solutions of this equation have been called *gaussons*, and are given by

$$u(x,t) = \left(h\sqrt{\frac{\pi}{2mb}}\right)^{-\frac{1}{2}} e^{-\frac{iu_e^2}{2mh}t+\frac{iu_e}{h}x-\frac{mb}{h^2}(x-u_e t/m)^2+i\phi_0+\frac{E_0}{2b}}. \tag{4.24}$$

This single-soliton expression, which has been verified by direct substitution into eq. 4.21, appears to correct several typos in the solution from [66].

4.3 Applications of the NLS equation

The standard model for solitons in optical fibers is the integrable cubic-NLS equation, which describes the dispersion and nonlinearity effects whose balance leads to solitons. In this case, the dispersion effect is the tendency for high frequencies of light to travel faster than low frequencies, a phenomenon referred to as *negative*, or *anomalous*, *group velocity dispersion (GVD)*. The nonlinearity is the *Kerr effect*, a change in refractive index related to the intensity of the optical field. When negative GVD and the Kerr effect counteract each other so that all frequencies travel at the same velocity, the result is an optical soliton [96, 33].

In general, the cubic-NLS equation describes soliton propagation in so-called *Kerr materials* — materials in which the operative nonlinearity is due to the Kerr effect. This is the case for centrosymmetric and isotropic materials [65], and includes optical fibers, in which soliton transmission has been demonstrated over long distances [63, 64]. In such media, the most important physical effects are dispersion and nonlinearity, but propagation is also influenced by other effects, including fiber loss, higher-order dispersion and nonlinearity, and the Raman effect, which leads to a self-frequency shift [96]. To account for these effects, the cubic-NLS equation can be extended into the nonintegrable version

$$-i\frac{\partial u}{\partial t} = \frac{1}{2}\frac{\partial^2 u}{\partial x^2} + |u|^2 u + i\Gamma u - i\beta\frac{\partial^3 u}{\partial x^3} + \alpha_1\frac{\partial}{\partial x}(|u|^2 u) - \alpha_2 u\frac{\partial |u|^2}{\partial x}, \tag{4.25}$$

where Γ, β, α_1, and α_2 represent the fiber loss, higher-order dispersion, higher-order nonlinearity, and the Raman effect, respectively [96].

The nonintegrable sat-NLS and k-NLS equations are applicable to simulating various physical phenomena, including the nonlinear effects of laser beam propagation in various media [59]. Sat-NLS also describes the recently discovered $1+1$-dimension (one space and one time dimension) photorefractive optical spatial solitons in steady state [78, 83, 81], and the optical spatial solitons in atomic media in the proximity of an electronic resonance [97]. Coupled systems of two or more sat-NLS equations describe the interaction of incoherent light pulses, with one such pulse modeled by each equation [78, 83, 81].

Both cubic-NLS and sat-NLS describe temporal solitons; with the transformation

$t \to z$, both equations also describe spatial solitons, with x and z being the transverse and longitudinal directions [16, 8]. In practice, spatial solitons appear better suited for computation, because temporal solitons require long distances to propagate. In addition, spatial solitons also exist in dimension $2+1$ (two space and one time dimensions) [83, 97], offering an additional degree of freedom and suggesting the possibility of implementing two-dimensional universal systems such as the billiard-ball model of computation [38, 61].

The log-NLS equation, which supports solitons called *gaussons*, was proposed as a nonlinear model of wave mechanics [14, 15, 66], but was later found unlikely to describe any physical system [29]. Gaussons are wave packets with gaussian-shaped envelopes and sinusoidal carrier waves. They are analogous to the wavefunctions of linear wave (quantum) mechanics; that is, the square of the amplitude of a gausson at a given point x can be interpreted as the probability that the particle described by the gausson is at x.

4.4 Discussion

In this chapter we introduced physical solitons and described a handful of 1-d soliton systems. Myriads of such systems exist, both classical and quantum mechanical; here we have not addressed quantum mechanical [105] or higher-dimensional solitons, because those are separate topics worthy of independent research. We have concentrated on using 1-d classical solitons, particularly the electromagnetic solitons described by NLS equations. In the next chapter we explain how we simulate and study such solitons.

Chapter 5

Numerical Methods

To study the behavior of solitons in various NLS systems, we often make use of numerical methods. Although exact multisoliton solutions of some integrable NLS systems are known, such solutions are quite complicated for more than two or three solitons, and numerical solution can be faster and easier to implement. The integrable Manakov system, for example, admits a complex two-soliton solution found only recently [71], and exact expressions for three or more colliding solitons appear as yet undiscovered. In addition, for arbitrary initial conditions, even integrable NLS systems must be solved numerically. To our knowledge, no multisoliton solutions have been found for any nonintegrable NLS system, and the existence of such solutions is an open problem. However, numerical methods can be used to study the propagation and collision of multiple solitons efficiently in both integrable and nonintegrable systems.

In this chapter we present two types of numerical methods: an implicit finite-difference scheme, originally devised for the linear Schrödinger equation [31], which we adapted to solve the NLS; and several variants of the split-step Fourier method [22, 94, 6]. These methods solve the general single-mode NLS equation, as well as systems of coupled NLS equations. In addition, we describe the implementation of these methods in the computer program `nls` — an interactive graphical tool for simulation and analysis of NLS systems.

5.1 Background and notation

In a numerical simulation of an NLS system, we first plot two or more well separated single-soliton solutions, obtained by the methods in chapter 4, on a discrete spatial grid at time 0. As we explained earlier, such a superposition of solitons is a solution of the NLS equation, provided the solitons are sufficiently separated in space [14]. We then apply a numerical method to compute the state of the system at successive time steps, plotting the results in a space-time graph, as in figs. 6.4 through 6.12.

In this chapter we use the following notation. Let $u[j, n]$ denote a discrete grid of complex numbers, mapped from the continuous space $u(x, t)$ by discretizing space and time in small intervals of Δx and Δt, respectively, so that $x = j\Delta x$ and $t = n\Delta t$ for $j = 0, 1, 2, ..., J$ and $n = 0, 1, 2, ...$ We rewrite eq. 4.3 as

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + N(|u|)u, \tag{5.1}$$

where D is a complex number, and N an arbitrary operator on $|u|$.

5.2 Finite-difference method

The implicit finite-difference method in [31] solves the linear Schrödinger equation, which can be written as eq. 5.1 with $D = 1$ and with the time-dependent nonlinearity $N(|u|)$ replaced by a time-independent potential. In adapting the method to solve eq. 5.1, we ignore the time dependence of $N(|u|)$ for small time steps Δt. This allows us to follow the derivation in [31] with only one change — in the Schrödinger equation, we replace the $\partial^2/\partial x^2$ operator with $D\partial^2/\partial x^2$, where D is an arbitrary complex number. We present the resulting finite-difference algorithm, and we refer the reader to [31] for details of the derivation.

The algorithm begins with the initial conditions $u[j, 0]$ for $j = 0, 1, 2, ..., J$, and requires the boundary conditions $u[0, n] = u[J, n] = 0$ for all n. To propagate the solution from time step t to $t + \Delta t$ (i.e., from grid time n to $n+1$), let $e[1..J-1]$ and $f[1..J-1]$ denote two arrays of complex numbers, and compute

$$\begin{aligned} e[1] &= 2 + (\Delta x)^2 N(|u[1, n]|)/D - i\lambda, & (5.2) \\ e[j] &= 2 + (\Delta x)^2 N(|u[j, n]|)/D - i\lambda - 1/e[j-1], & (5.3) \end{aligned}$$

for $1 < j < J$, where $\lambda = 2(\Delta x)^2/(D\Delta t)$. Next, compute

$$\begin{aligned} f[1] &= \Omega_1^n, & (5.4) \\ f[j] &= \Omega_j^n + f[j-1]/e[j-1], & (5.5) \end{aligned}$$

for $1 < j < J$, where

$$\Omega_j^n = -u[j+1, n] + (i\lambda + (\Delta x)^2 N(|u[j, n]|)/D + 2)u[j, n] - u[j-1, n]. \tag{5.6}$$

Finally, compute

$$u[j, n+1] = (u[j+1, n+1] - f[j])/e[j], \tag{5.7}$$

for $1 < j < J$. Recall that the boundary conditions specify $u[J,\ n\ +\ 1] = u[0,\ n\ +\ 1] = 0$. This completes the prescription for propagating the solution from one time step to the next.

The boundary conditions for the finite-difference method have meanings that depend on the interpretation of the Schrödinger equation. In the quantum-mechanical interpretation, the equation describes a physical system enclosed in a 1-d "box" such that the wavefunction vanishes at the "walls" of the box. In the classical interpretation, waves such as solitons "bounce" off these endpoints in the same way as waves on a vibrating string tied down at both ends.

5.3 Split-step Fourier method

The split-step Fourier method [94, 6] decomposes eq. 5.1 into two equations that describe the dispersive and nonlinear effects in a soliton-supporting medium:

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2}, \tag{5.8}$$

$$\frac{\partial u}{\partial t} = N(|u|)u. \tag{5.9}$$

Dispersion (eq. 5.8) and nonlinearity (eq. 5.9) normally act simultaneously on the optical field u. The split-step method approximates a solution of eq. 5.1 by assuming that for small time steps Δt, the two effects can be considered independent. Thus, to propagate the optical field from time t to $t + \Delta t$, eqs. 5.8 and 5.9 are solved separately as described below. It can be shown that this procedure gives results accurate to second order in Δt [6].

5.3.1 Derivation of the basic scheme

To derive a method for solving eq. 5.8, fix t and let $U[f, t]$ represent the discrete Fourier transform (DFT) of $u[j, t]$, where $j = f = 0, 1, 2, ..., J-1$, and $x = j\Delta x$.

That is, by definition of the inverse DFT,

$$u[j,t] = \frac{1}{J}\sum_{f=0}^{J-1} U[f,t]e^{2\pi i f j/J}. \tag{5.10}$$

Substituting eq. 5.10 in eq. 5.8, we obtain

$$\frac{1}{J}\sum_{f=0}^{J-1} U'[f,t]e^{2\pi i f j/J} = \frac{D}{J}\sum_{f=0}^{J-1}\left(\frac{2\pi i f}{J\Delta x}\right)^2 U[f,t]e^{2\pi i f j/J}, \tag{5.11}$$

where the prime ($'$) denotes partial differentiation with respect to t. Matching terms on both sides of this equation, we obtain the linear differential equation

$$U'[f,t] = D\left(\frac{2\pi i f}{J\Delta x}\right)^2 U[f,t], \tag{5.12}$$

which yields

$$U[f,t] = e^{-D\left(\frac{2\pi f}{J\Delta x}\right)^2 t}U[f,0]. \tag{5.13}$$

To compute the solution at time $t+\Delta t$ from the solution at time t in frequency space, the above equation gives

$$U[f,t+\Delta t] = e^{-D\left(\frac{2\pi f}{J\Delta x}\right)^2 \Delta t}U[f,t]. \tag{5.14}$$

To approximate the solution of eq. 5.9, we treat $N(|u[k,t]|)$ as a constant, disregarding its time dependence for small steps Δt. Thus, eq. 5.9 yields

$$u[j,t] = e^{N(|u[j,t]|)t}u[j,0], \tag{5.15}$$

which gives

$$u[j,t+\Delta t] = e^{N(|u[j,t]|)\Delta t}u[j,t]. \tag{5.16}$$

Combining the above steps, we obtain the basic split-step Fourier scheme for solving eq. 5.1:

- Linear step:
  - Let U = FFT(u).
  - Perform the computation specified by eq. 5.14.
  - Let u = inverse FFT(U).
- Nonlinear step:
  - Perform the computation specified by eq. 5.16.

To compute DFT and inverse DFT, we use the fast Fourier transform (FFT) algorithm for speed. The DFT operates on a circular domain, which implies that the boundary conditions for the split-step Fourier method are cylindrical; waves in the numerical grid "wrap around" from one grid edge to the other, rather than bouncing off edges, as with the finite-difference method described earlier.

5.3.2 Improving accuracy of the basic scheme

The basic split-step scheme propagates the optical field twice over the time interval Δt — first with dispersion only, and then with nonlinearity only. A more accurate variant of the scheme, called the *symmetrized* split-step Fourier method [94, 6], first propagates the field with dispersion over the interval $\Delta t/2$, then applies the effect of nonlinearity, and finally propagates the field over the remaining interval $\Delta t/2$. That is, the method computes half of the linear step, the full nonlinear step, and finally another half of the linear step.

The symmetrized split-step method is simple to implement. For the scheme which includes the full nonlinear step between two linear half-steps, we perform the computation specified by eq. 5.14 over the interval $\Delta t/2$ instead of Δt; to effect this, we take the square root of the right side of eq. 5.14. Then we compute the nonlinear step as specified by eq. 5.16, followed by another calculation of the initial linear half-step. This scheme is accurate to third order in Δt [6], but is slower because of the additional linear step.

For faster computation, we can compute the linear step between two halves of the nonlinear step, but this variant gives somewhat less accurate results. To gain accuracy at the expense of computation speed, we can split up both the linear and nonlinear steps in more complicated ways than above. In addition, we can solve eq. 5.9 more accurately. Note that a formal solution of this equation is

$$u(x, t + \Delta t) = e^{\int_t^{t+\Delta t} N(x,t)dt} u(x, t), \tag{5.17}$$

and we can approximate the above integral using the trapezoidal rule or other techniques [6]. Later in this chapter we present results on the speed and accuracy of the various numerical schemes we implemented.

5.3.3 Application of the method to systems of NLS equations

For solving coupled NLS equations, such as the coupled sat-NLS system (eq. 4.16) and the Manakov system (eq. 7.1), we can extend the split-step Fourier method in a straightforward fashion. For each equation in the system, we use a separate numerical grid. We then split up each equation into its linear and nonlinear steps, as with a single NLS equation, and perform the computations specified by eqs. 5.14 and 5.16 on each grid. To achieve higher accuracy of solutions, for each equation we can apply the same symmetrization technique as explained above.

5.4 Software for solution of NLS equations

We implemented the finite-difference and several split-step Fourier methods in the computer program `nls`, a package for studying soliton behavior in NLS systems. In this program we also include functionality for presenting and analyzing numerical solutions; all soliton figures and numerical results in this dissertation were generated with `nls`. The software, written in C and C++, runs on several platforms, including Unix and Windows. In this section we briefly describe the program's functionality and features, and we refer the reader to the WWW site

`http://www.cs.princeton.edu/~mj/nls`

for more detailed documentation.

5.4.1 Overview

`Nls` reads text input files specifying parameters for numerical simulation of a 1-d NLS soliton system; generates initial conditions by plotting solitons on a 1-d numerical grid; and applies a numerical method to propagate the initial conditions over time. The input files give all necessary simulation parameters, such as the numerical-solution method, type of NLS equation, grid size, Δx, and Δt. `Nls` generates both 2-d and 3-d PostScript output that shows space-time graphs of the NLS system's evolution, displaying variables such as magnitude, carrier, and phase. The program also supports direct numerical output that can be easily processed by graphing and visualization software. In addition, `nls` can show the output in a graphical window that allows the user to study the simulation interactively.

The input files for `nls` contain lines of the form `param-name = param-value`, each of which indicates that the parameter called `param-name` is to be set to the value `param-value`. In addition, the files include lines indicating solitons to be plotted in the initial conditions for the simulation. Each such line specifies soliton parameters such as amplitude, velocity, and initial position in the numerical grid.

As an example, fig. 5.1 shows a sample `nls` input file for a two-soliton collision in the cubic-NLS system. The file specifies that the split-step Fourier method be used for numerical solution using a grid of size 1024 points, with $\Delta x = 0.025$, and $\Delta t = 0.004$. The file also states that the simulation is to be run for 9000 time steps, and that 3000 time steps are to be displayed per page of PostScript output, resulting in 3 total pages of output. Each page displays 100 lines of output, and each line displays the magnitude squared of every second of the 1024 grid points, as specified by the `display` and `space-skip` parameters; in addition, three angle parameters specify the 3-d rotation of the display plane, which is fit into a PostScript bounding box of size 600 by 500. The last two lines of the input file specify the solitons to be plotted, which have amplitudes 1, velocities 1 and -1, initial phases 0 and 0.5π, and initial grid positions 256 and 768. Fig. 5.2 shows the first page of the PostScript output generated by running `nls` with the sample input file. Fig. 5.3 shows the `nls` PostScript output from the same file, but with the parameters `display = phase`, `space-skip = 8`, `axes = 0`, and `shaded-plot = 1`. This figure is a visualization of the phase shift due to a cubic-NLS soliton collision.

5.4.2 NLS systems and solution methods

`Nls` currently supports solution of several NLS equations, which are specified by the input parameter `equation`, as in fig. 5.1. Table 5.1 gives a list of these equations as values that can be used in `nls` input files for the `equation` parameter. The table also lists the `nls` names of parameters that can be set for each equation. These parameters correspond directly to the variable names used in the NLS equations

```
#-------------------------------------------------------
# a sample input file for nls

# parameter values
equation = cubic-nls
solver = split-fourier
gridsize = 1024
deltaX = .025
deltaT = .004
timesteps = 9000
steps-per-page = 3000
lines-per-page = 100
PS-height = 600
PS-width = 500
display = magsqr
space-skip = 2
XYangle = 190
XZangle = 185
YZangle = 315
axes = 1

# solitons to be plotted in initial conditions
soliton: v = 1, A = 1, phi0 = 0.5, x0 = 256
soliton: v = -1, A = 1, phi0 = 0, x0 = 768
#-------------------------------------------------------
```

Figure 5.1: Sample `nls` input file for a two-soliton collision in the cubic-NLS system.

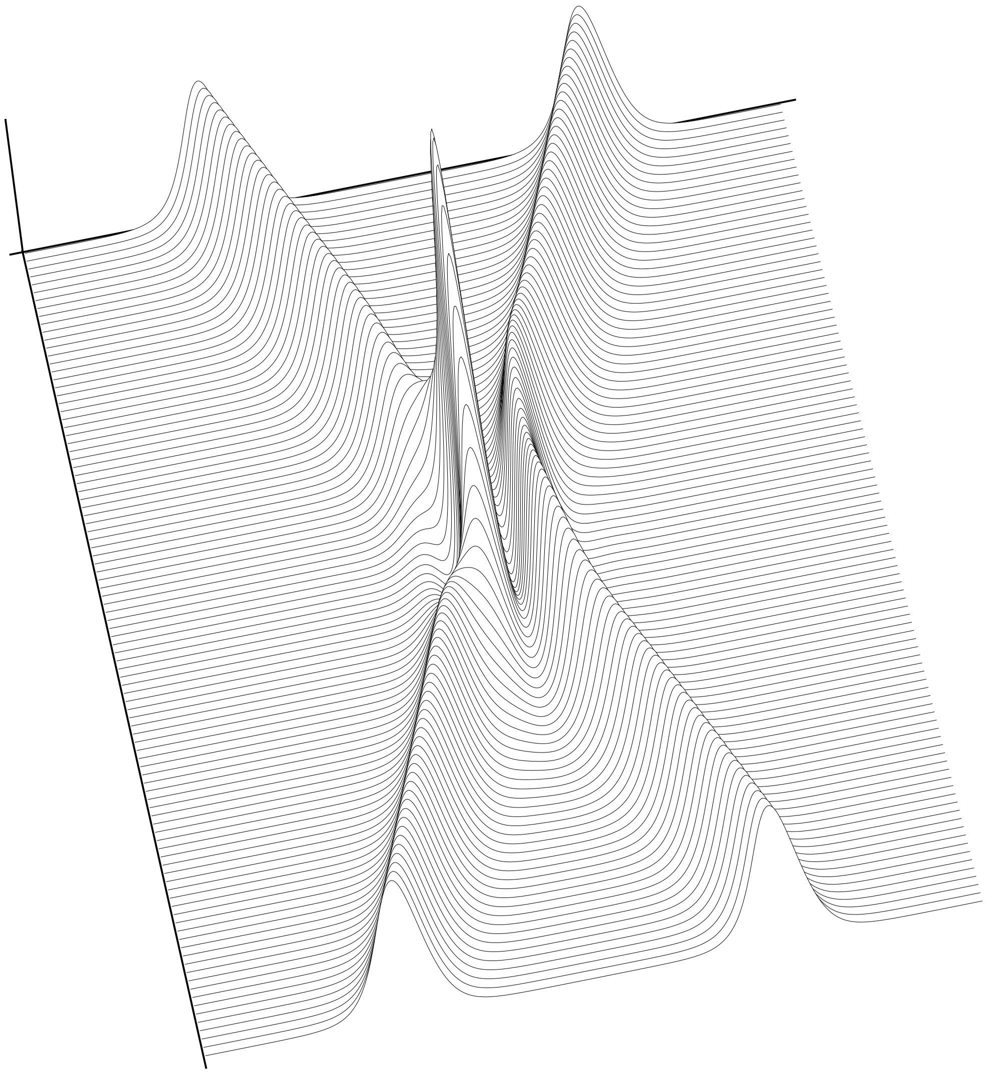

Figure 5.2: First page of the graph generated by nls with the sample input file from fig. 5.1. The graph shows a two-soliton collision in the cubic-NLS system.

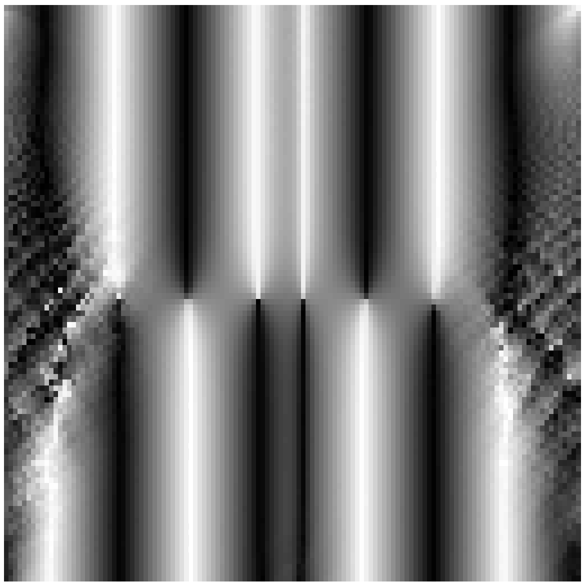

Figure 5.3: First page of the graph generated by `nls` with the sample input file from fig. 5.1, modified as described in the text. The graph shows a two-soliton collision in the cubic-NLS system by plotting shades of gray for the phase values of grid points. The phase shift due to the collision is apparent as a sudden change in the orientation of the phase contours approximately halfway down from the top of the graph.

in this dissertation; see the corresponding descriptions of the equations in chapters 4 and 7. Additionally, the parameter `numgrids` for the coupled sat-NLS system specifies the number of coupled equations, and thus the number of numerical grids used in solution.

The solution methods implemented in `nls` include the finite-difference method from section 5.2 and several variants of the split-step Fourier method described in section 5.3. The split-step methods are specified in `nls` input files by the `solver` parameter, which can have the following values:

- `finite-diff` — finite-difference method;
- `split-fourier` — basic split-step Fourier method;
- `split-fourier-NLN` — symmetrized split-step method which includes the lin-

`Nls` equation name	Equation in text	`Nls` parameter names
`log-nls`	eq. 4.21	`h, m, l, E0`
`cubic-nls`	eq. 4.9 ($D = 0.5$, $b = 1$, $n = 2$)	
`ext-cubic-NLS`	eq. 4.25	`alpha1, alpha2, beta, gamma`
`k-nls`	eq. 4.9	`D, b, n`
`sat-nls`	eq. 4.15	`D, m, k`
`coupled-sat-nls`	eq. 4.16	`D, m, k, numgrids`
`Manakov`	eq. 7.1	`mu`
`k-Manakov`	eq. 7.1	`mu, numgrids`

Table 5.1: Equations and systems solved by `nls`. The `numgrids` parameter specifies the number of simultaneous equations in coupled systems.

ear steps between two nonlinear half-steps;

- `split-fourier-LNL` — symmetrized split-step method which includes the nonlinear step between two linear half-steps;
- `split-fourier-LNLtrap` — symmetrized split-step method with trapezoidal-rule approximation of the nonlinear integral [6].

5.4.3 Specifying initial conditions

Analytical single-soliton solutions of NLS equations have the general form

$$f(x) = A\Phi(x - u_e t)\theta(x - u_c t), \tag{5.18}$$

where A, u_e (also called v) and u_c denote a soliton's amplitude, envelope velocity, and carrier velocity, respectively. `Nls` allows specification of A and u_e, which together determine u_c. Furthermore, each soliton has an initial phase ϕ_0 and location x_0. The `nls` soliton parameters that represent these variables are the following:

- `A` — amplitude;

- `v` — envelope velocity;
- `phi0` — initial phase (in multiples of π radians);
- `x0` — initial grid position (a number from 0 to `gridsize`).

In `nls` input files, each single soliton is specified with a line such as

```
soliton:  A = 1, v = -1, phi0 = 0.5, x0 = 512
```

placed immediately after the list of equation and display parameters.

For coupled systems, single-component solitons are specified similarly, with the additional parameter `grid` to indicate the grid in which the soliton component is to be plotted; for example,

```
soliton:  grid = 0, A = 1, v = -1, phi0 = 0.5, x0 = 512.
```

Two-component solitons for the `Manakov` system are specified with lines of the form

```
soliton:  k = 1|1, alpha = 2|0.5, beta = 1, x0 = 512,
```

where the parameters `k`, `alpha` and `beta` are complex numbers described above. The vertical bar (`|`), with no spaces, is used to separate real and imaginary parts of numbers.

Multi-component solitons for the `k-Manakov` and `coupled-sat-nls` systems are specified with lines such as

```
soliton:  v = 2, A = 1, kappa = 0:0:.5, w = .5:1.2:2, x0 = 512,
```

where `v` is the velocity of all components, and `A` an amplitude. The real vectors `kappa` $= \kappa_0, \kappa_1, ..., \kappa_n$ and `w` $= w_0, w_1, ..., w_n$, where $n =$ `numgrids - 1`, specify phases and amplitudes of individual components, with the colon (`:`) used to separate scalars in the vectors (no spaces).

Such multi-component solitons for coupled systems are plotted as follows. For `k-Manakov`, the soliton component in grid i is generated by plotting a k-NLS soliton with amplitude `A`, velocity `v`, and phase κ_i.[1] The component is then multiplied by w_i/W, where $W = \sqrt{\sum_i w_i^2}$. For `coupled-sat-nls`, components are generated similarly, except each component is initially plotted as a single, numerically computed `sat-nls` soliton. It can be verified that this produces exact multi-component solitons for these systems.

5.4.4 Parameter reference

This section gives a complete list of `nls` parameter names followed by brief descriptions. In each description, values in parentheses indicate defaults.

General parameters

- `gridsize` — number of grid points in each numerical grid (1024)
- `timesteps` — number of propagations to compute (10000)
- `deltax` — Δx (0.03)
- `deltat` — Δt (0.01)
- `startpt` — grid index at which to begin output (0)
- `endpt` — grid index at which to end output (1023)
- `space-skip` — grid-point subsampling amount for output (1)
- `equation` — NLS system to solve (see table 5.1)

[1]Note that the k in `k-Manakov` refers to the number of coupled equations, whereas the k in `k-NLS` refers to the exponent in the NLS equation's nonlinear term, and is equal to 2 in this case.

- `solver` — solution method (see section 5.4.2)
- `numgrids` — number of equations for coupled systems (1)
- `dispgrid` — number of grid to output (0), between 0 and `numgrids` (When `dispgrid` = `numgrids`, the value output is the sum of corresponding values in all grids.)

PostScript 3-d output parameters

Values indicated below as *boolean* can be set to `1` for *true* and `0` for *false*.

- `XYangle, XZangle, YZangle` — orientation of coordinate planes with respect to viewer (180.0, 180.0, 315.0)
- `viewx, viewy, viewz` — absolute position of viewer (0, 0, 900)
- `foreshortening-x, foreshortening-y` — perspective- foreshortening amounts (5000, 5000)
- `ps-width, ps-height` — PostScript width and height (611, 791) [2]
- `ps-origin-x, ps-origin-y` — PostScript coordinates of corner of output box (20, 20)
- `x-scale, y-scale, z-scale` — coordinate scaling (1.0, 3.0, 6.0)
- `hidden-lines` — boolean: remove hidden lines (1)
- `scale-to-box` — boolean: make graph fit within output box (1)
- `center` — boolean: center graph within output box (1)

[2]Standard PostScript 8.5”x11” page dimensions are 612x792.

- `axes` — boolean: draw coordinate axes (0)
- `box` — boolean: draw PostScript box around graph (0)
- `ps-line-width` — PostScript line width (0.1)
- `lines-per-page` — number of contour lines per page of output (140)

Parameters for 3-d PostScript and 2-d window output

- `steps-per-page` — number of propagation steps to summarize in each page of output (5000)
- `display` — display variable, which can be one of the following functions of grid values (default `magsqr`):
  - `real` — real part
  - `imag` — imaginary part
  - `mag` — magnitude
  - `magsqr` — magnitude squared
  - `phase` — phase angle
  - `color-phase` — color representation of phase angle
- `displaysrc` — source of variables to display, which can be one of the following (default `single-grid`):
  - `single-grid` — a single grid
  - `coupled-grids` — sum of values in all coupled grids
  - `manakov` — quotient of corresponding complex numbers in two grids (i.e., the Manakov state ρ for two grids, as described in chapter 7)

- `shaded-plot` — boolean: display gray shades instead of contours (0)
- `color-shaded-plot` — boolean: display color shades instead of contours (0)

Equation-specific parameters

See table 5.1 and the equation descriptions referenced therein.

5.4.5 Performance

In our implementation, the `finite-diff` method is generally the fastest and the least accurate, whereas the `split-fourier-LNL` method is the slowest and most accurate. Our tests revealed that the `split-fourier-LNLtrap` method, described in in [6] and potentially very accurate, was by far the slowest, but had accuracy little better than the finite-difference method. This leaves some doubts about our implementation.

Tables 5.2 and 5.3 list some speed and accuracy results for numerical methods implemented in `nls`. Our tests are based on solving the cubic-NLS equation by running `nls` on a Sun UltraSPARC machine. We measure speed in propagations (time steps) per second. To estimate accuracy, we plot an exact two-soliton cubic-NLS solution [38] at time 0, and use numerical methods to propagate the system forward in time. After a given number n of time steps, we compare the numerical solution with the exact solution by computing the root-mean-squared (RMS) error.

5.4.6 The `nls` interactive interface

The window-based graphical interface of `nls` offers the user a number of functions for visualizing and analyzing soliton behavior in NLS systems. For example, one can use an input device such as a mouse to select 1-d regions of the space-time solution graph, "copy" such regions into a file, "paste" them anywhere in the graph, and let the

Method	Speed	Error at $n = 3000$	Error at $n = 6000$
`finite-diff`	256	1.212958×10^{-2}	4.066236×10^{-2}
`split-fourier`	113	1.337030×10^{-3}	1.354952×10^{-3}
`split-fourier-NLN`	83	3.860094×10^{-5}	7.965682×10^{-5}
`split-fourier-LNL`	70	2.730716×10^{-5}	5.536547×10^{-5}

Table 5.2: Speed and accuracy measurements for numerical methods solving cubic-NLS on a grid of size 2048, with $\Delta x = 0.03$ and $\Delta t = 0.003$. Speed is in propagations per second, and error is the RMS error between exact and numerical solutions, as described in the text.

Method	Speed	Error at $n = 1500$	Error at $n = 3000$
`finite-diff`	514	2.665434×10^{-2}	8.344766×10^{-2}
`split-fourier`	272	2.734989×10^{-3}	2.863168×10^{-3}
`split-fourier-NLN`	192	1.543877×10^{-4}	3.185947×10^{-4}
`split-fourier-LNL`	182	1.092079×10^{-4}	2.214205×10^{-4}

Table 5.3: Speed and accuracy measurements for numerical methods solving cubic-NLS on a grid of size 1024, with $\Delta x = 0.06$ and $\Delta t = 0.006$. Speed is in propagations per second, and error is the RMS error between exact and numerical solutions, as described in the text.

numerical solution proceed from that point in time. The user can also plot additional solitons anywhere in the graph, measure soliton parameters, such as amplitude and velocity, and change solution parameters. Additionally, one can select regions of the graph, and zoom in on them or magnify them; this can be useful for close examination of radiation and artifacts of numerical solution. `Nls` can generate PostScript output of any graph displayed in the window. Figs. 5.4 and 5.5 show window shots of `nls` running in interactive mode.

5.5 Discussion

We now have a set of theoretical and practical tools with which we can study soliton behavior. In the next chapter we present results on the types of PMs that arise from various soliton systems. We examine some general issues about mapping soliton sys-

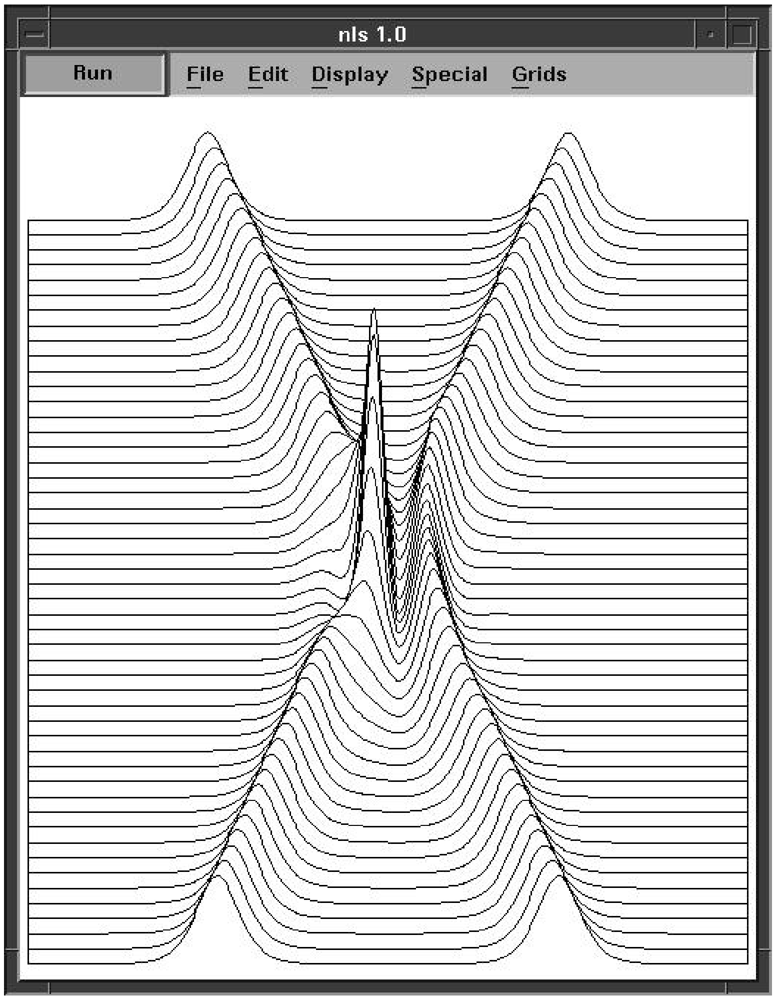

Figure 5.4: Window shot of `nls` running under the X Windows system. The graph was produced by running `nls` with the input file from fig. 5.1 and changing a scaling parameter from an `nls` menu.

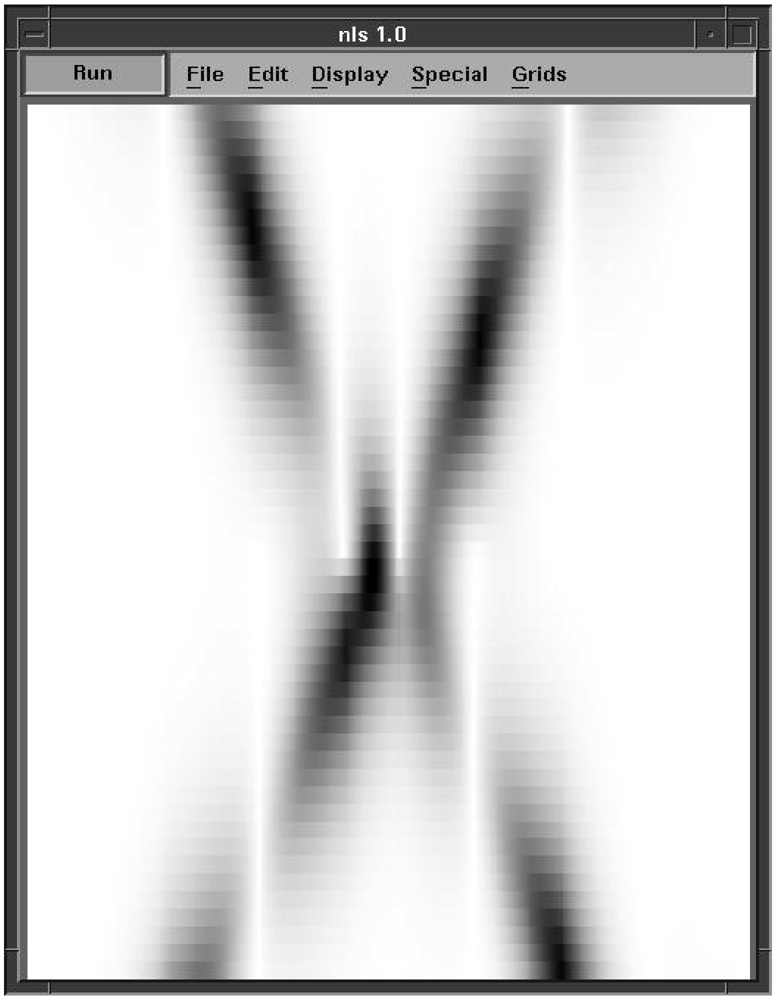

Figure 5.5: Window shot of `nls` running under the X Windows system. The graph was produced by running `nls` with the input file from fig. 5.1 and changing a display parameter from an `nls` menu. For each grid-point value, this graph uses a shade of gray to show phase weighted by magnitude.

tems onto PMs, including information-bearing and information-transfer capabilities of solitons. In a later chapter we demonstrate examples of PM computation in a specific physical system, namely the Manakov system of optical solitons.

Chapter 6

Implementing Particle Machines with Solitons

The PM is a natural computational model for soliton systems. By discretizing time and space in physical soliton-supporting media, we model solitons as particles in PMs, which we can try using for practical computation. However, different physical systems give rise to PMs of varying computational capabilities. In this chapter we take steps to characterize the power of PMs arising from various soliton systems.

The idea of using abstract solitons in a homogeneous medium for "gateless" computation goes back at least to [93], where solitons in a CA[1] are used to build a carry-ripple adder. This chapter moves from the abstraction of CA and PMs to the physical realm represented by PDEs such as the nonlinear Schrödinger (NLS), Korteweg-de Vries (KdV), and sine-Gordon equations [76]. Our goal is to study the general issue of information storage and transfer among solitons in various physical

[1]These solitons arise in the mathematical framework of a CA [23, 24, 32, 68], and have an entirely different origin than the physically based solitons we consider here. However, CA-based and PDE-based solitons display remarkably similar behavior. As far as we know, the connection between CA solitons and PDE solitons is unexplained, though some authors [4, 67] have juxtaposed discussions of both systems.

systems, and to determine how this could — and could not — be used for general computation.

6.1 Solitons and computation

In recent years much effort has been expended on analyzing the properties of solitons for purposes such as high-speed communications and optical computing gates [39, 38, 96]. To our knowledge, however, our work is the first attempt to do practical computation via collisions of physical solitons in a bulk medium. The resulting computational system would fulfill the promise of Toffoli's "programmable matter" [100] — offering computation that is very close to the underlying physics, and therefore potentially providing ultra-scale parallel processing.

The most immediate physical realization of PM computation may be provided by solitons in an optical fiber [33, 38, 96], described by the cubic nonlinear Schrödinger equation. Other media are also possible, including Josephson junctions [80] and electrical transmission lines [40, 69], which support solitons governed by the sine-Gordon and Korteweg-de Vries equations, respectively.

We should emphasize that using optical solitons in this way is quite different from what is commonly termed "optical computing" [39, 38], which uses optical solitons to construct gates that could replace electronic gates, but which remains within the "lithographic" paradigm of laying out gates and wires. The idea presented here uses a completely homogeneous medium for computation – the entire computation is determined by an input stream of particles. A general version of the structure proposed is shown in fig. 6.1.

To use physical solitons for computation, we define restricted versions of the PM called *soliton machines (SMs).* Both PMs and SMs are one-dimensional cellular

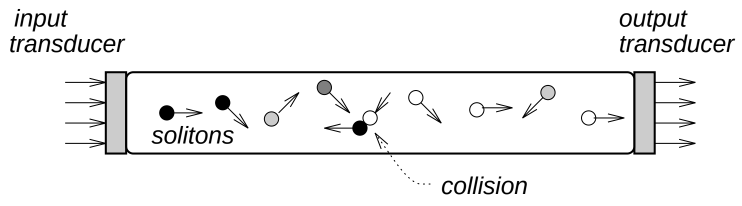

Figure 6.1: Computing with solitons in a bulk medium. Solitons are injected at the left of the diagram, computation takes place within the medium via the interaction of the pseudoparticles, and the results exit from the right of the diagram. The actual medium can be linear, planar, or 3-d.

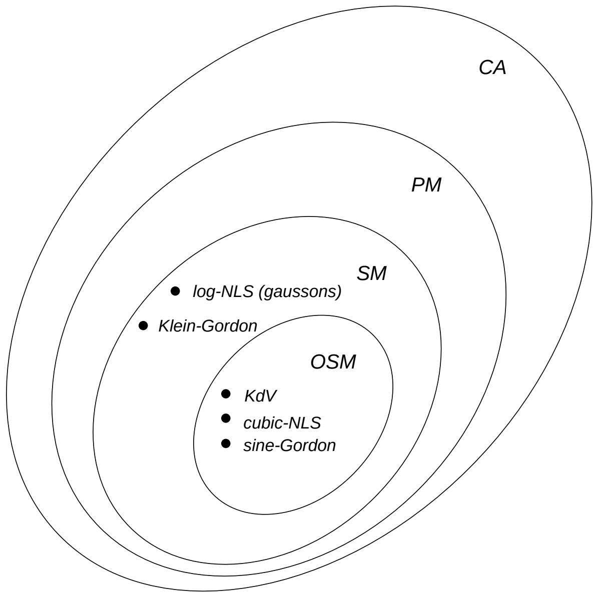

Figure 6.2: Hierarchy of computational systems in the world of cellular automata (CA). Particle machines (PMs) are CA designed to model particle-supporting physical media. Soliton machines (SMs) are restricted PMs that model general soliton systems, including PDEs such as the Klein-Gordon and log-NLS equations. Oblivious soliton machines (OSMs) are SMs that model integrable soliton systems, such as the KdV, cubic-NLS, and sine-Gordon equations.

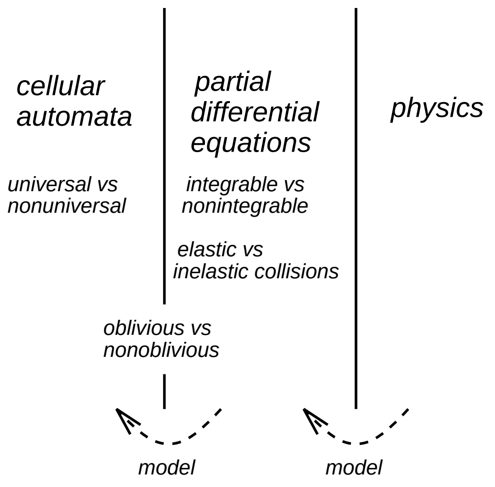
cellular automata
partial differential equations
physics
universal vs nonuniversal
integrable vs nonintegrable
elastic vs inelastic collisions
oblivious vs nonoblivious
model
model

Figure 6.3: The three worlds considered in this chapter. Notice that the property *oblivious* applies to both CA and soliton solutions of PDEs, whereas the properties *integrable* and *having elastic collisions* apply only to soliton systems.

automata that model motion and collision of particles in a uniform medium. *Oblivious soliton machines* (OSMs) are SMs further restricted to model a class of integrable soliton systems. The hierarchy of the computational systems we consider is shown in fig. 6.2. In general, we abstract a physical system by modeling it first with PDEs, and then with CA, namely PMs and SMs, as shown in fig. 6.3.

We will discuss the computational power of the ideal machines we use to model physical systems. Being able to simulate a Turing machine, or another universal model, is neither necessary nor sufficient for being able to perform useful computation. For example, certain PMs can perform some very practical regular numerical computations, such as digital filtering, quite efficiently, and yet such PMs are not necessarily universal [89, 42]. Conversely, simulating a Turing machine is a very cumbersome and inefficient way to compute, and any practical application of physical phenomena to computing would require a more flexible computational environment. Nevertheless, universality serves as a guide to the inherent power of a particular machine model.

6.2 Information transfer and computation

A soliton can carry information in its envelope amplitude, width, and position; its group and phase velocities; and its carrier phase; and this information can be exchanged in collisions with other solitons. This section is devoted to the question of whether such information transfer can occur in a way that is useful as a basis for computation, while still preserving particle identities. If this is possible, it suggests that general computation can be performed via interacting waves in a uniform medium, such as a nonlinear optical material.

In the usual conception of optical computing, one builds discrete gates based on the propagation of light, and then essentially mimics the construction of a conventional computer. We describe here an alternative approach to building an all-optical computer, using only solitons in a homogeneous nonlinear optical medium. In our approach, programs and data are encoded as streams of solitons, which are injected into the medium at a boundary, and which compute via the information transfer effected by solitary-wave collisions.

A general Turing-equivalent model for such "gateless" computation is the PM. By exploiting the fine-grain parallelism of particle systems, this model supports fast and efficient execution of many operations, including arithmetic and convolution. Briefly, particle machines treat solitons as particles whose collisions can change particle states, thus performing computation. Such computation requires that if solitons A and B collide, then (1) some part of the resulting state of A depends on the state of B; and (2) – this is essential — the state of B is changed by collisions. In a word, information should be transferred from one wave to the other in "interesting" ways.

There is much already known about the phenomenology of collisions in nonintegrable versions of NLS equations [66, 59, 87, 86, 25, 81]. Generally, solitons in these systems, including the saturable NLS, can change amplitude and velocity after collisions. We emphasize that this is not in itself sufficient to meet our criterion of computationally "interesting."

The properties that are useful for our computational purposes are the opposite of those usually considered useful in communication optics: At least some of the collision products must effect a nontrivial transformation of information in the colliding waves. The reason for this is that general computation requires a transformation of information in basic logic operations. Unfortunately, many commonly studied systems that support waves do not have this behavior. For instance, because of linear superposition, colliding plane waves in a linear medium do not interact, i.e., do not undergo any state changes, and therefore cannot support information interaction among colliding waves.

An example of a system in which collisions do cause change of state but nevertheless cannot transform information in a nontrivial manner is the cubic nonlinear Schrödinger equation (3-NLS). In order to do computation, solitons must carry information from one collision to the next; such information must be coded in parameters that are not constant. However, in the 3-NLS system, the state parameters that cause the information transfer are themselves invariant: The only change of state occurs in spatial (or temporal) position and carrier phase, and this change depends only on the amplitudes and velocities of the envelopes of the incoming solitons. We conjecture in [43, 44] that all solitary-wave collisions in integrable systems have this property,[2] and we show that particle machines based on such systems are very limited in computational power. In particular, these particle machines are not Turing-equivalent. We must therefore look to solitons in nonintegrable or coupled integrable systems for

[2]As we will see in chapter 7, this conjecture is false for multi-mode integrable systems, but remains unresolved for single-mode systems such as the ones considered in the current chapter.

computationally useful collisions.

For solitons to carry information, they must also preserve their integrity after a sequence of collisions, and lose negligible energy through radiation. These requirements are apparently antagonistic to the information-transform capability necessary for computation, but our goal is to find systems which meet all these requirements. The results shown here suggest that the saturable nonlinear Schrödinger equation (sat-NLS) describes such a system.

6.2.1 Information transfer

To be more precise about the definition of *information transfer*, suppose that a medium supports a set of solitons. Then a selected set of properties that can change during a collision define a *state* $S(A)$ of a wave A, whereas a set of constant wave properties that are unaffected by collisions define an *identity* $I(A)$ of A. Note that we may define different types of states S and identities I for the same type of wave. Denote by A' the soliton A after a collision with wave B. Then a collision of A with B supports transfer of information if $S(A')$ depends on $S(B)$, for some $S(A)$ and $S(B)$; otherwise, the collision transfers only trivial information (if $S(A')$ depends on only $I(B)$) or no information (if $S(A') = S(A)$).

We illustrate the above definition using the the cubic-NLS and the saturable-NLS systems. The cubic-NLS equation supports solitons whose variable states are phases, and whose constant identities are amplitudes and velocities. Collisions of such solitons transfer only trivial information, since the phase shifts due to soliton collisions are a function of only the amplitudes and velocities, i.e., the identities, of the colliding solitons. On the other hand, the saturable-NLS system gives rise to solitons whose variable state includes phases, amplitudes, and velocities. This system supports collisions that transfer nontrivial information, since the state changes due to collisions are a function of the states of the colliding waves.

6.2.2 Computational power

To examine how information transfer relates to computational power, we briefly review the notion of Turing equivalence, or computational universality. Informally, a Turing machine is a computational model in which programs and data are stored on an infinite tape of discrete cells. A read-write head processes information by reading cell contents, writing new cell contents, and moving back and forth along the tape, all according to a transition function that considers both the state of the head and the symbol read from underneath the head. The machine can enter a special "halt" state, which signals the end of computation and the presence of the machine's final output on the tape.

It is generally accepted (by virtue of Church's thesis [37]) that given enough time and space, a Turing machine can implement any algorithm; that is, in terms of the results that can be computed, a Turing machine is as powerful as any computer. A computational system is Turing-equivalent, or computation-universal, if it can simulate a Turing machine. While this property is not absolutely necessary for a system to do useful computation, universality nevertheless serves as a good measure of a system's computational potential.

Intuitively, in order for computation to take place in a solitary-wave system, colliding waves should interact and transfer information that is necessary to execute steps of an algorithm. In [44] we show that only at most cubic-time computation can be done on a particle machine that models a system in which collisions transfer at most trivial information. This upper bound on such a system's computation time proves that this system cannot be Turing-equivalent, since universal computation can take an arbitrarily long time. Moreover, solitary-wave interactions in this system are computationally very limited, and designing algorithms based on these interactions appears tedious and impractical. It is unclear whether or not collisions supporting only trivial information transfer can encode any useful computation at all.

Solitary-wave systems in which collisions transfer nontrivial information are more readily applicable to encoding computation. We have shown in [44] that such a system can be Turing-equivalent, provided that the solitary-wave state changes are sufficiently complex.

6.3 Oblivious soliton machines

The PM model is a convenient abstraction for computing with solitons. In practice, however, the single-mode integrable soliton systems we have described are not suitable for implementing general PMs. Specifically, these systems do not support the creation of new solitons or the destruction of existing solitons, and soliton state changes due to collisions are restricted. Thus, we adopt as our model a restricted PM called an *oblivious soliton machine (OSM)*. Like a PM, an OSM is a CA designed to support particles propagating through a homogeneous medium, but an OSM more closely models the integrable soliton systems under consideration.

6.3.1 The OSM model

An OSM is a PM in which each particle has a constant identity and a variable state that are both vectors of real numbers. The velocity of a particle is part of its identity. A typical state may consist of a *phase* and a *position relative to a Galilean frame of reference*, whereas a typical identity may include an *amplitude* in addition to a velocity. No particles can be created or destroyed in collisions, and the identities of particles are preserved, much like the constant amplitudes and velocities of colliding solitons. A function of the *identities* (not states) of the colliding particles determines particle state changes.

Immediately after a collision, particles are *displaced*, much like the colliding solitons discussed earlier. Let P_{slow} and P_{fast} denote two particles such that the velocity of P_{slow} is (algebraically) less than the velocity of P_{fast}; that is, if V_{slow} and V_{fast} are signed integers representing the velocities of P_{slow} and P_{fast}, respectively, then $V_{slow} < V_{fast}$. In a two-way collision of P_{slow} and P_{fast}, P_{fast} is displaced by a positive integer amount, and P_{slow} by a negative integer amount. In collisions involving three or more particles, displacements are such that the relative order of the particles after the collision is the reverse of their order before the collision, and all particles are displaced into separate cells. Displacement amounts are functions of the identities of the colliding particles. In addition, we require that once two particles collide, the same particles can never collide again; this can be accomplished by spacing the particles properly, or by choosing particle velocities and displacements appropriately. This scheme models particle interaction in the integrable soliton systems described earlier.

Definition 2 An *oblivious soliton machine (OSM)* consists of the following elements:

- A two-way infinite one-dimensional *medium M*.
- A finite set P of *particles*, each with one of a finite set of *velocities*.
- A finite set S of real-number vectors called *particle states*.
- A *collision function C*.
- A *post-collision displacement function D*.

- A finite *initial configuration* or *input I*.

M contains discrete *cells*, each of which can hold from *0* to |P| particles. At most one particle of a given identity can occupy a cell. Each particle travels at a constant velocity and has a variable state which may change as a result of collisions with other particles in the medium. The initial configuration *I* is a finite section of *M*, and includes the *input* particles present at the beginning of the OSM computation. Up to |P| particles can occupy each cell of *I*. The *input size* is defined as the length of *I* plus the number of input particles in *I*. During collisions, no particles are created or destroyed, and the identities of particles remain constant, but the states of particles change according to the function *C*. For each possible pair of identities of colliding particles, *C* specifies two new particle states. After two or more particles collide, they are displaced by amounts determined by the function *D*, as described previously. Any pair of particles can collide at most once. The machine begins with its initial configuration on either a quiescent or a periodic background, and evolves in discrete time steps.

6.3.2 Non-universality of OSMs

We refer to OSMs as *oblivious* because the state changes in an OSM do not depend on the variable states of colliding particles, but only on their constant identities. *Oblivious* collisions in the OSM model correspond to *elastic* collisions in the integrable PDEs discussed here; however, it is an open question to the authors whether or not all elastic soliton collisions in all integrable systems are oblivious. The spatial displacements of OSM particles after collisions occur only in the constrained fashion described previously. The result of these properties is that OSMs cannot compute universally.

Theorem 1 *OSMs are not computation-universal, either with or without a periodic background. The maximum time that an OSM can spend performing useful computation is cubic in the size of the input.*

Proof: We show that the halting problem for OSMs is decidable. More specifically, we calculate a cubic upper bound (in terms of input size) on the amount of time taken by an OSM to do any computation.

To execute any algorithm using an OSM, we must encode the algorithm and its input as a finite sequence of particles in a finite-length initial configuration (input) I of an infinite homogeneous medium. We must also be able to decode the OSM state when the results of the algorithm are ready. Let N denote the number of particles in I, not counting the particles in a possibly periodic background (PB), and L the length of I. The input size $|I|$ of the OSM is then $N + L$.

We first examine the case of an OSM with a quiescent background. For such an OSM we can calculate upper bounds on the maximum number of particle collisions, and on the maximum time before each collision occurs. The product of these two values will give an upper bound on the maximum time that the OSM can spend performing useful computation.

- An upper bound on the number of particle collisions is $\binom{N}{2}$, since each particle can collide at most once with any other particle, by definition.
- An upper bound on the time before the collision of any two particles that do collide is

$$\frac{L + N|D_b| + N|D_f|}{|v_f| - |v_s|} \tag{6.1}$$

where D_b and D_f are the largest negative (backward) and positive (forward) displacements possible among the input particles, and $|v_f|$ and $|v_s|$ are the largest and smallest speeds, respectively, among the input particles.

Thus, an upper bound on the time that an OSM can perform useful computation is

$$\binom{N}{2}\frac{L + N|D_b| + N|D_f|}{|v_f| - |v_s|}. \tag{6.2}$$

Since $|I| = N + L$, $|v_f|$ and $|v_s|$ are nonnegative integers, and D_b and D_f are constants in a particular OSM, expression (6.2) is $O(|I|^3)$; that is, cubic in the input size.

In an OSM with a PB (PB-OSM), the PB particles can displace the input particles both left and right at regular intervals. Note that these periodic displacements cause the velocities of the input particles to change by constant amounts, which depend on the specific periodic configuration of PB particles. Thus, we can recalculate the velocities of the input particles, using the displacements effected by the PB particles. To find an upper bound on the time taken by a PB-OSM to do useful computation, we apply a similar argument as for the quiescent background, but with the newly calculated effective velocities.

Note that collisions in a PB-OSM can occur forever, but the collisions useful for computation can occur only within the time bound that we can calculate. To see this, observe that after the time given by this bound, the input particles of a PB-OSM will stay in a fixed relative order, unable to collide again with one another. Thus, each input particle either breaks away from the rest, as in the case with a quiescent background, or stays close to the others in a periodic configuration. In neither situation can the input particles do useful computation. ∎

Corollary 1 *OSM-based computational systems governed by the KdV and sG equations are not universal, given that positions are used as state. OSM-based systems governed by the cubic-NLS equation are not universal, given that positions and phases are used as state.*

Conjecture 1 *All single-mode integrable systems using any choice of state are non-universal using the OSM model.*

6.4 Soliton machines

Intuitively, OSMs cannot compute universally because particles in an OSM do not transfer enough state information during collisions. We can make a simple modification to the OSM model so that universal computation becomes possible: We make the results of collisions depend on both the identities and *states* of colliding particles. In addition, we allow particle identities to change. We call the resulting model a *soliton machine (SM)*. In the final section of this chapter, we will describe non-integrable equations that support soliton-like waves which we believe may be capable of realizing the SM model.

6.4.1 The SM model

Definition 3 A *soliton machine (SM)* is defined in the same way as an OSM, with the following differences. Both the collision and displacement functions can depend on the identities and states of particles. The identities of particles can change during collisions, so that the collision function returns both the new states and the new identities of colliding particles.

Like an OSM, an SM is also a CA and a PM (see fig. 6.2). The only difference between an SM and a PM is that no particles can be created or destroyed in an SM. However, we can use a periodic background of particles in special *inert* or *blank* states, and simulate creation and destruction of particles by choosing collision rules so that particles go into, and out of, these states.

6.4.2 Universality of SMs

SMs with a quiescent background have at least the computational power of Turing machines (TMs) with finite tapes, as we will prove. The question of whether such SMs are universal is open, however. Still, these SMs are more powerful than any OSM, since OSMs can only do computation that requires at most cubic time, while problems exist that require more than cubic time on bounded-tape Turing machines.

The class of algorithms that a finite-tape TM can implement depends on the specific function that bounds the size of the TM's tape; for instance, TMs with tapes of length polynomial in the input size can do any problem in $PSPACE$. Although not universal, such TMs can do almost any problem of practical significance.

Theorem 2 *SMs with a quiescent background are at least as powerful as Turing machines with bounded tapes.*

Proof: We describe how SMs can simulate any finite-tape Turing machine M. Let $B(N)$ denote the function that bounds the size of M's tape, given an input of size N.

We construct an SM equivalent to M as follows. For each possible state of a cell of M, including the blank state, we introduce a distinct, stationary *state* particle. M's finite tape then maps directly to a length-$B(N)$ section T of the SM's medium.

To simulate the action of M's tape head, we introduce two *head* particles, h_f and h_b, of velocities 1 and -1, corresponding to the right and left motions of the head, respectively. Before computation, we place h_f in the cell immediately to the left of T.

The SM begins in the initial configuration described above, and operates as follows. The head particle h_f moves through T, colliding with the state particles in T. We choose collision rules such that collisions between state particles and either h_f or h_b simulate M's transition function. Thus, a collision between h_f or h_b and a state particle s can change s to another state particle; in addition, h_f can change into h_b, and vice versa. This simulates writing on M's tape and changing the direction of M's head. $\blacksquare$

SMs with a periodic background are universal, since we can use such SMs to simulate a Turing machine as described previously, but with a periodic background of blank-state particles. This background maps directly to the infinite blank portion of the Turing machine's tape. Thus, we have proved the following theorem:

Theorem 3 *SMs with a periodic background are computation-universal.*

6.4.3 Discussion

Theorems 1 through 3 suggest that we should look to nonintegrable or coupled integrable systems for solitons that may support universal computation. It is an open question whether or not there exists such a soliton system. In what follows, we recall some soliton equations and explain the features that could enable them to encode SMs. We also describe numerical experiments with two particular nonintegrable PDEs, the logarithmically nonlinear Schrödinger equation (log-NLS) and the saturable Schrödinger equation (sat-NLS).

6.5 Information transfer in collisions of NLS solitary waves

6.5.1 The cubic-NLS equation

In the integrable 3-NLS system, solitary waves are true solitons whose collisions can change only their envelope position and carrier phase; envelope amplitude and velocities are conserved. In addition, the spatial and phase shifts of colliding solitons depend only on their constant amplitudes and velocities. Thus, such collisions transmit only trivial information, and are computationally very limited, as we demonstrated earlier in this dissertation and in [44]. (See fig. 6.13.)

6.5.2 Gaussons and the log-NLS equation

Our numerical simulations of gausson collisions verify a published report [66] that they range from deeply inelastic to near-elastic, and perhaps perfectly elastic, depending on the velocities of the colliding gaussons. In [66] an approximate range of velocities (the *resonance region*) is given for which collisions are apparently inelastic; outside this region, collisions are reportedly elastic. We confirmed these results, and investigated in more detail to find the following three distinct velocity regions in which gaussons behave very differently:

1. When $0 < |v| < 0.5$, gausson collisions are near-elastic, and possibly perfectly elastic, and clearly nonoblivious. We observed marked post-collision changes in both amplitude and velocity, which strongly depend on the phases of the colliding gaussons. These phenomena appear to be newly observed here. (See

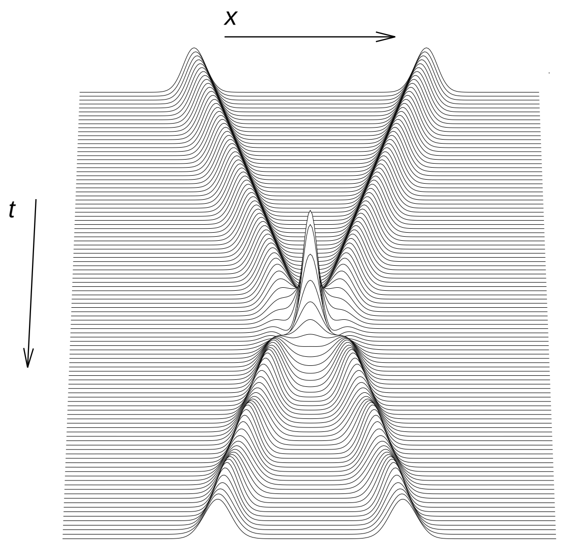

Figure 6.4: Gausson collisions in region 1. From left to right, velocities are 0.4 and -0.4; phases are both 0.

figs. 6.4, 6.5, and 6.6.) [3]

2. When $0.5 \leq |v| < 10$, collisions are non-elastic, and possibly near-elastic. The amount of radiation generated in collisions varies with the phases of the colliding gaussons, and in general decreases as v increases. (See Figs. 6.7, 6.8, 6.11, 6.12.)

3. When $|v| \geq 10$, collisions are near-elastic, and possibly elastic, but apparently oblivious. They are similar to the collisions found in integrable soliton systems, such as the cubic-NLS equation. (See fig. 6.9.)

These ranges are approximate, and gausson behavior changes gradually from one to the next. The differences between our results and those in [66] (written ca. 1978) are likely due to our more extensive numerical experimentation, given the faster computers available to us. In our calculations we used the split-step Fourier method [22, 94] with a cylindrical (wrap-around) one-dimensional coordinate system;

[3]All gausson figures are graphs of space versus time, with time increasing from top to bottom. The variable graphed is $|u|^2$, that is, the square of the gausson envelope.

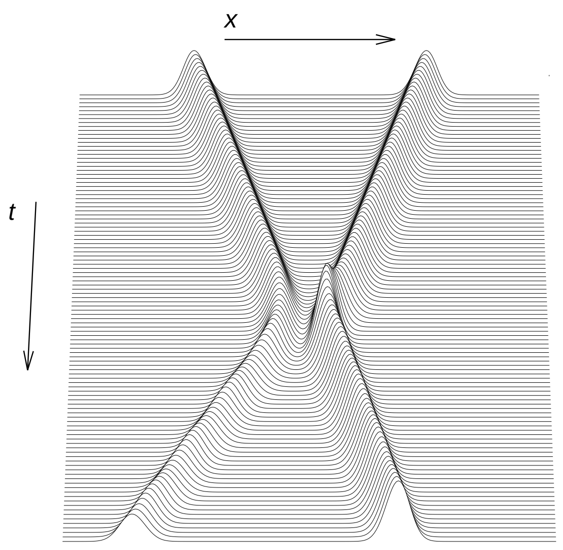

Figure 6.5: Gausson collisions in region 1. From left to right, velocities are 0.4 and -0.4; phases are 0 and 0.5π.

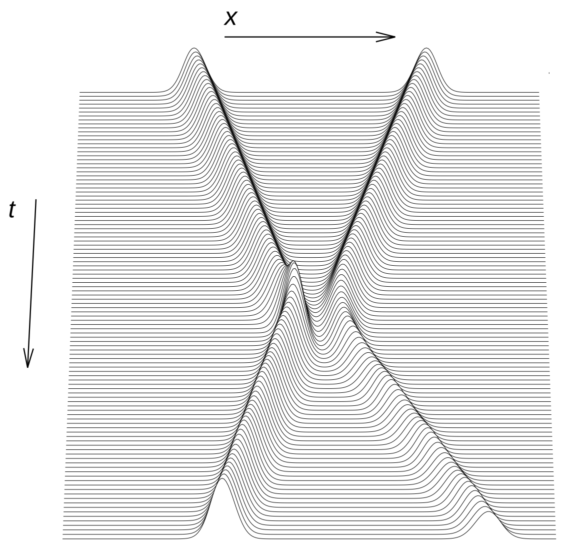

Figure 6.6: Gausson collisions in region 1. From left to right, velocities are 0.4 and -0.4; phases are 0.5π and 0.

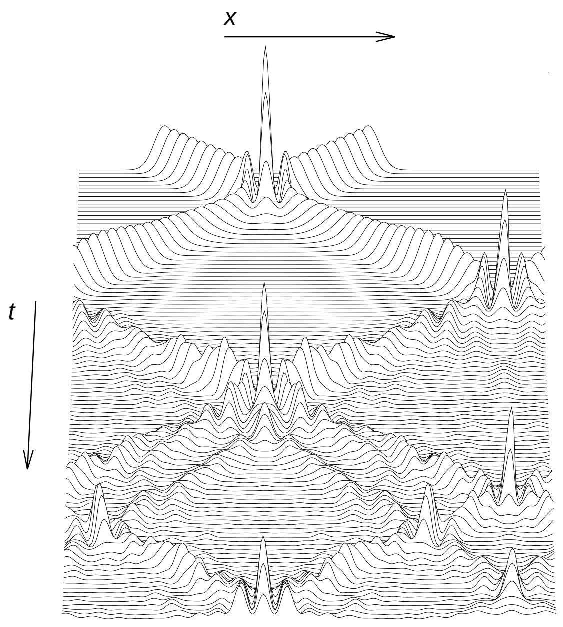

Figure 6.7: Gausson collisions in region 2. From left to right, velocities are 3.0 and -3.0; phases are both 0. A cylindrical coordinate system is used here, so that there is wrap around from the right to the left edge, and vice versa.

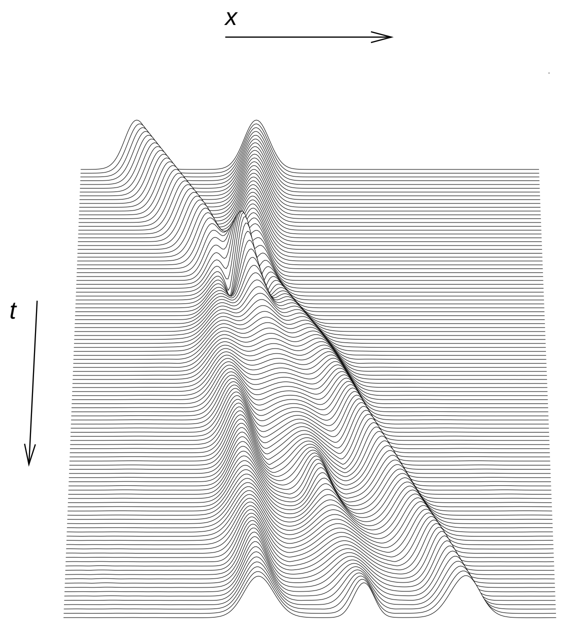

Figure 6.8: Gausson collisions in region 2. From left to right, velocities are 2.0 and -0.1; phases are both 0. A cylindrical coordinate system is used here, so that there is wrap around from the right to the left edge, and vice versa.

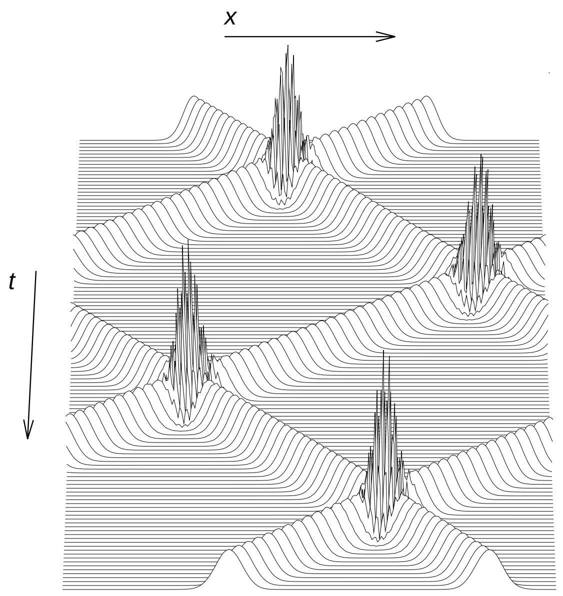

Figure 6.9: Gausson collisions in region 3. From left to right, velocities are 10.0 and −15.0; phases are both 0. A cylindrical coordinate system is used here, so that gaussons wrap around from right to left, and vice versa.

the authors of [66] used a finite difference scheme [31], which we also implemented, and which confirms the results of the split Fourier method. However, this finite difference method's treatment of boundary conditions makes a cylindrical coordinate system difficult to use.

Nonobliviousness in region 2 is not as easily determined by visual inspection as it is in region 1. Next, we describe a numerical experiment which demonstrates that there is a near-elastic gausson collision that is nonoblivious in region 2. Fig. 6.10 shows the setup of this experiment. To show that the collision between two gaussons, A and B, is nonoblivious, we begin at time 0 with three gaussons, A, B, and C, in that order, on the x-axis. The velocities and initial distances among the three gaussons are set so that A and B collide first, followed by a collision of A and C. We observe the results of the AC collision for various initial phases of B's carrier, keeping constant the initial phases of A and C. If we find two initial phases for B that lead to

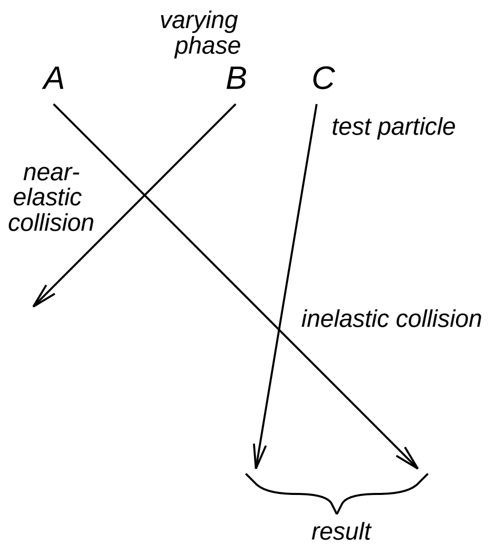

Figure 6.10: Testing for nonobliviousness using a near-elastic collision (AB) followed by an inelastic collision (AC). If the result of the AC collision depends on the initial phase of B, then the AB collision is nonoblivious.

two different results of the AC collision, then we can conclude that the AB collisions were nonoblivious. Note that we require only that the results of the AC collisions be *different*; the AC collisions can be strongly inelastic, for the C particle is used only to probe the state of the A particle.

Figs. 6.11 and 6.12 show an example of such an experiment. In both figures, the two leftmost gaussons move at a velocity of ± 3.25, the rightmost gausson has velocity -1, and all but the center gaussons have initial phase 0. The center gaussons in figs. 6.11 and 6.12 have phases 0.05π and 0.55π, respectively, which cause the different results after collisions. We conclude that the collision between A and B is nonoblivious. Note that we cannot determine nonobliviousness merely from the visual appearance of the AB collision, because what happens *during* a collision can depend on the phases of the colliding solitons, whether the collision is oblivious or not; it is the *post*-collision results that determine nonobliviousness.

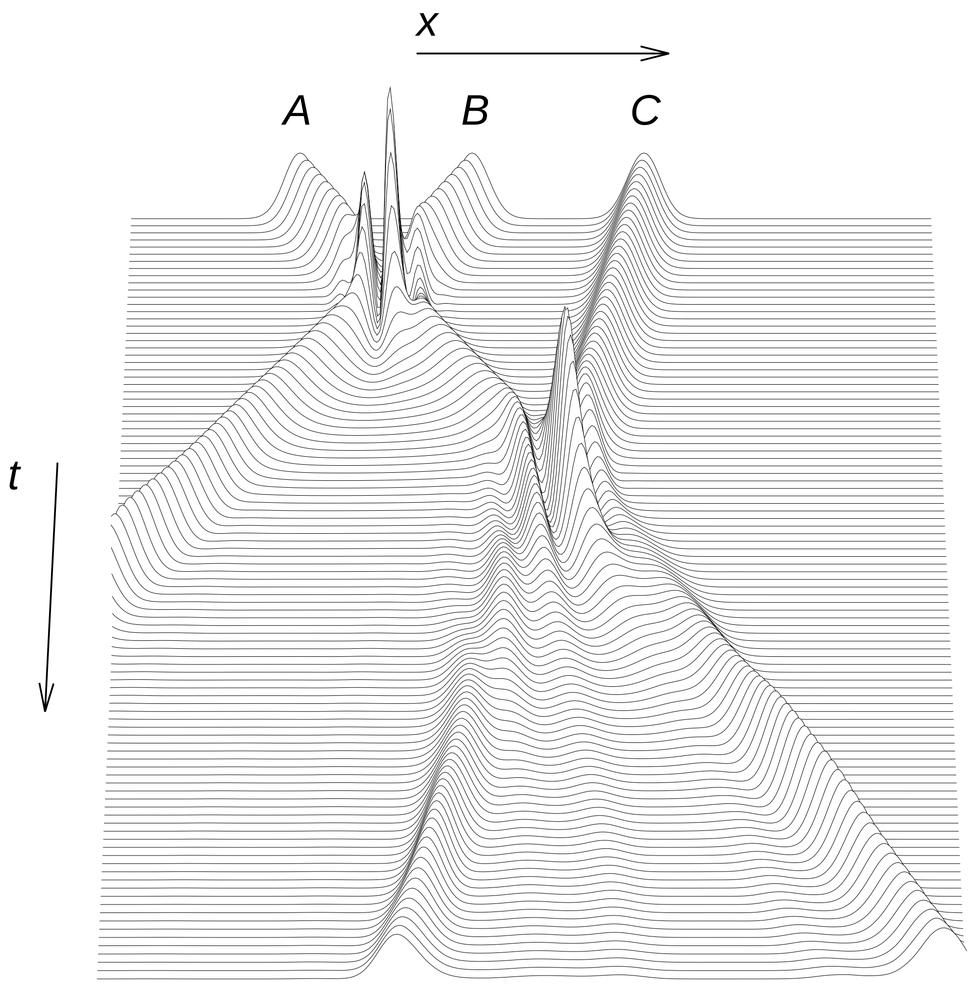

Figure 6.11: Testing for obliviousness of the collision between the leftmost (A) and center (B) gaussons in region 2. The center gausson's phase is 0.05π; the other two gaussons' phases are both 0. The collision is nonoblivious, since the results of the test collision between the leftmost and rightmost gaussons (A and C) differ from those in the next figure.

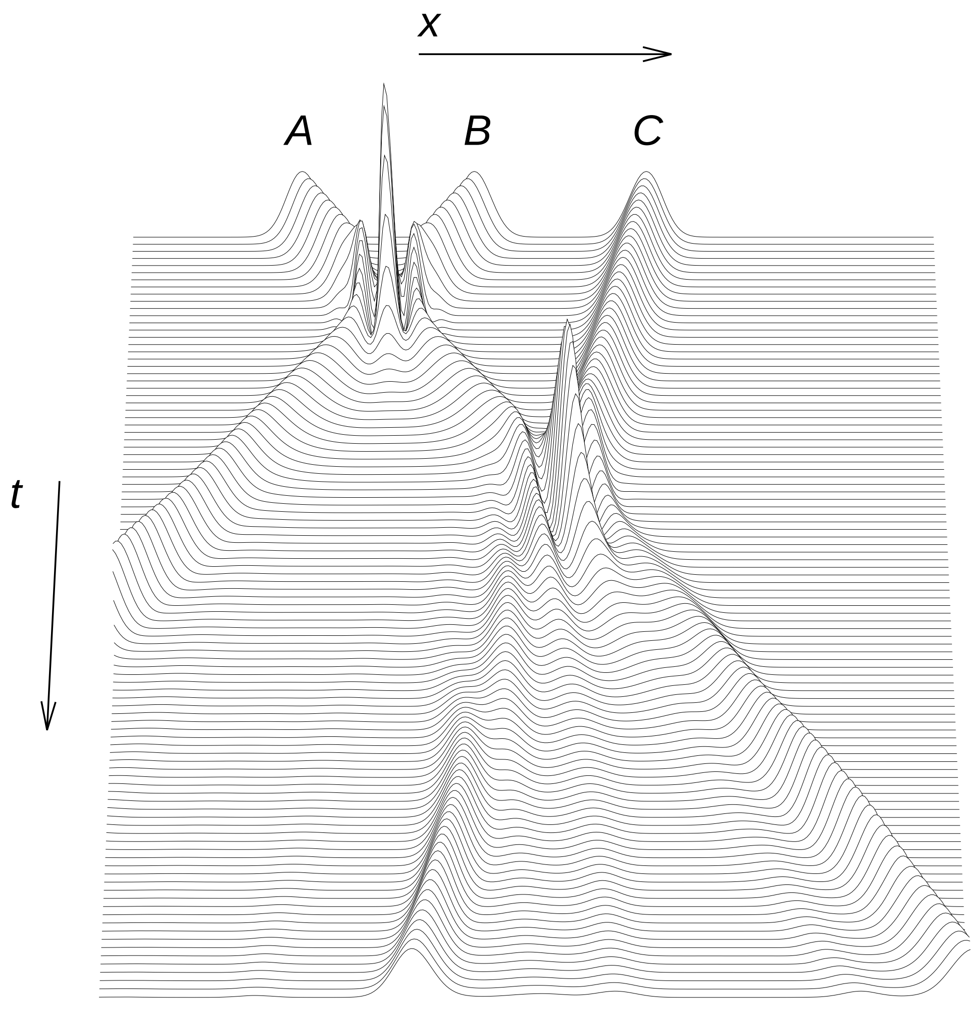

Figure 6.12: Testing for obliviousness of the collision between the leftmost (A) and center (B) gaussons in region 2. The center gausson's phase is 0.55π; the other two gaussons' phases are both 0. The collision is nonoblivious, since the results of the test collision between the leftmost and rightmost gaussons (A and C) differ from those in the previous figure.

6.5.3 Soliton stability and elasticity

The inelastic and near-elastic soliton collisions we observed in regions 1 and 2 are nonoblivious, thus leaving open the possibility of using them for computation in SMs. For example, to compute using gaussons, we can use an approach similar to the techniques in [93]. As with the CA solitons in [93], we might first create a database of pairwise collisions of gaussons by running a series of numerical experiments; we would then search the database for useful collisions to encode a specific computation. This approach was used in [93] to implement a solitonic ripple-carry adder.

One problem with such an approach is the potential connection between soliton stability and collision elasticity. We observed that inelastic collisions often resulted in *radiation ripples* emanating from collisions (fig. 6.7) and eventual disintegration of gaussons in a cylindrical one-dimensional system. In region 2, these ripples and the resulting instability may make the system unsuitable for sustained computation. The more inelastic the collisions, the quicker the system decayed. However, we do not know if stability and elasticity are necessarily correlated in general, nor do we know if elasticity and obliviousness (and thus lack of computation universality) are related. In fact, collisions of region-1 solitons in the log-NLS equation appear to be both elastic and strongly nonoblivious.

6.5.4 The sat-NLS equation

The nonintegrable sat-NLS equation gives rise to solitary waves whose collisions support nontrivial information transfer via transactive collisions. (See fig. 6.14.) In particular, phases, amplitudes, and velocities can all change as a function of the parameters of the colliding waves. We have observed that the most computationally useful collisions occur when the solitary waves have a low relative speed (approxi-

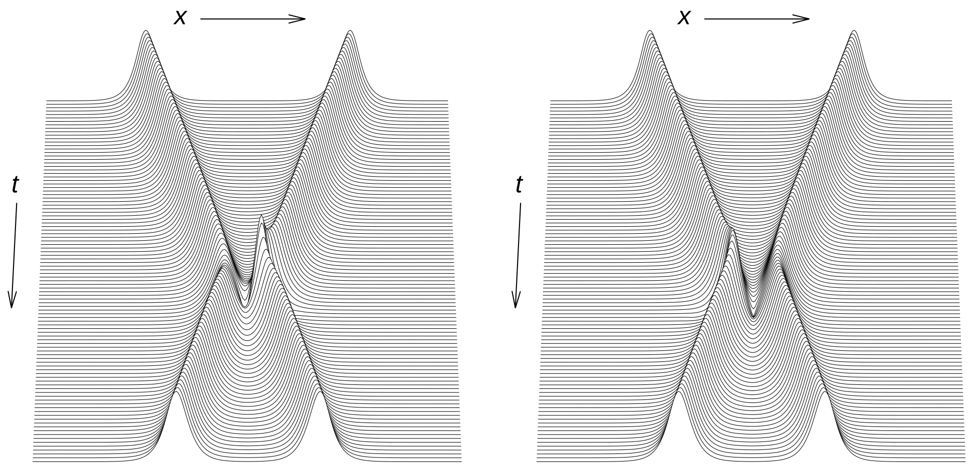

Figure 6.13: Trivial information transfer in collisions of 3-NLS solitons. The initial relative phases of the solitons in the left and right graphs are 0.25π and -0.45π, respectively; velocities are ± 0.2. Phase and spatial shifts, though not apparent from these graphs, are a function of only the constant amplitudes and velocities.

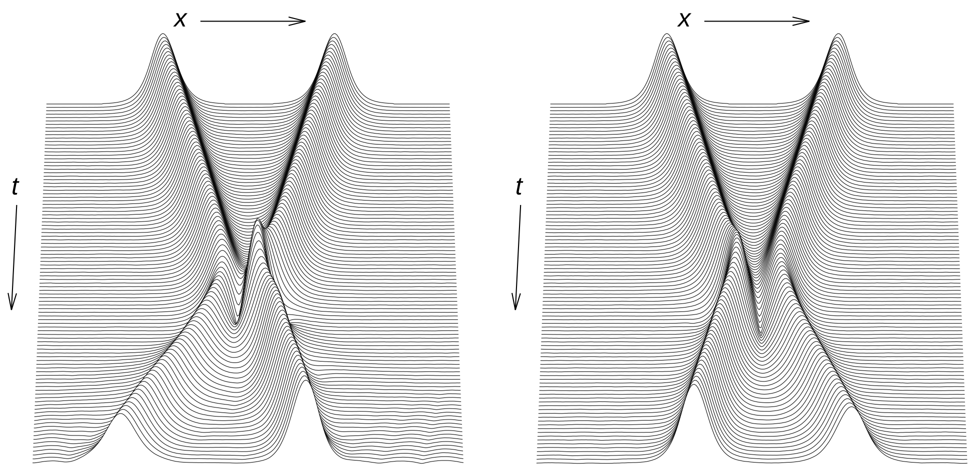

Figure 6.14: Nontrivial information transfer in collisions of sat-NLS solitary waves. The initial relative phases of the waves in the left and right graphs are 0.25π and -0.45π, respectively; velocities are ± 0.2.

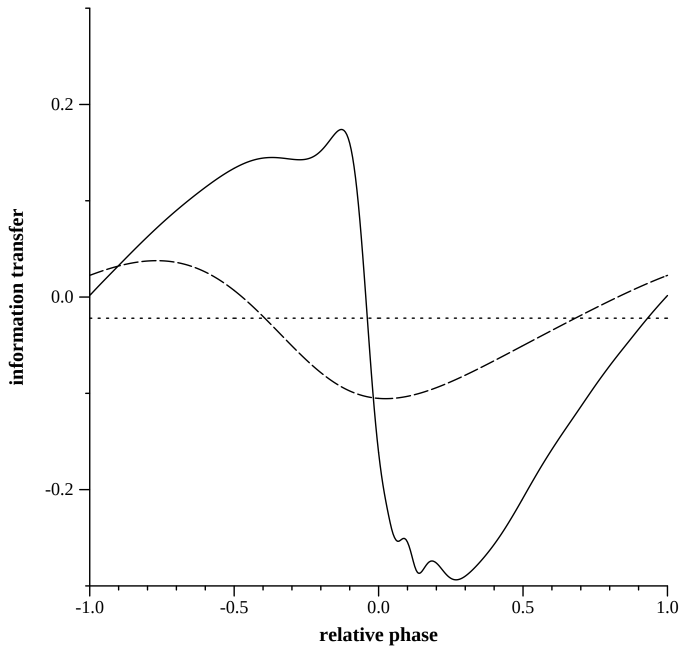

Figure 6.15: Information transfer for collisions of two sat-NLS solitary waves. Here information transfer is defined as the fractional change in the amplitude of one solitary wave; that is, the transfer is equal to $\Delta A_1/A_1$, where A_1 is the initial amplitude of the right-moving wave, and ΔA_1 is the amplitude change due to collisions. The solid, dashed, and dotted curves show information transfer for collisions of two waves with amplitudes 1.0 and velocities ± 0.5, ± 1.5, and ± 10.0, respectively. Relative phase is in multiples of π. Note that in the low-velocity case (solid line) near zero relative phase there is significant radiation and breathing in the collision products, making the amplitude poorly-defined. What is shown is the result of measuring the amplitude peak at a fixed time.

mately 4.0 and below). The magnitude of information transfer decreases gradually as the relative speed of the waves increases. To estimate this magnitude, we measured the amplitude and velocity changes following collisions of low-velocity waves at various initial phases. In fig. 6.15, normalized amplitude change is plotted as a function of the relative phase of two colliding solitary waves.

In practice it is reasonable to expect that the amplitudes of two colliding solitons cannot be made precisely equal. To test the robustness of the results in fig. 6.15 we ran experiments with unequal amplitudes (amplitude ratios of 1.1, 1.3 and 2.0) and found the results to be quite similar, except that the magnitude of the effect was even greater.

It might appear that in the perfectly symmetric case — when relative phase is zero — there should be no amplitude change. That is, that a nonzero value of amplitude change at zero relative phase shift would imply that energy is transferred from one wave to another, thereby spontaneously breaking symmetry. To explain this apparent problem, we note first that what is plotted is change in amplitude, not energy. A nonzero value of amplitude change at zero relative phase (which is indeed a symmetric situation) then means that the amplitudes of both solitons change. There are three ways that this can happen: (1) Radiation can decrease the energy of both solitons, (2) the amplitudes can change, but a change in width can compensate to preserve energy, and (3) the collision products can breathe, which makes the amplitude poorly defined.

6.6 Radiation and reusability

In general, computation encoded in an NLS system must reuse solitons after they have been involved in multiple collisions. To behave like the particles of a particle machine, these solitary waves should be stable; more specifically, collisions should preserve the identities of solitary waves and generate negligible radiation.

Numerical results reveal that information transfer and radiation often go hand in hand. Soliton collisions in the 3-NLS system are perfectly elastic and generate no radiation, but such collisions support only trivial information transfer, as we have seen. In the sat-NLS system, large amounts of radiation tend to accompany large magnitudes of information transfer. However, much like other known nonintegrable NLS systems [59], the sat-NLS system does support collisions that transfer information and yet generate only small amounts of radiation. More specifically, our numerical studies have revealed the following:

- When at least one of the solitary waves is moving at a high speed (approximately 4.0 and above), their collision generates negligible radiation and supports no measurable information transfer. (This phenomenon in generalized NLS systems was mentioned by Snyder and Sheppard [87].)
- When the relative phase $\phi_0 = \pi$, the collision is the same as in the above case, no matter what the value of the relative speed v_0.
- When both waves have low speeds (below 4.0) and $0 \leq \phi_0 < \pi$, the collision is accompanied by larger amounts of radiation and information transfer. However, as ϕ_0 tends towards π, both radiation and the magnitude of information transfer decrease. For $\phi_0 > \pi/2$, very little or no measurable radiation is generated.

The solitary waves that emerge from collisions in the sat-NLS system may or may not be of the form given by eq. 4.4, depending on the initial wave parameters. As observed early on in a variety of nonintegrable systems [59, 66], and predicted theoretically for a wide range of non-Kerr materials by Snyder and Sheppard [87], certain regimes of operation can lead to breathers, and more dramatically, to the fusion of colliding waves and the birth of new waves. We show examples of fusion in the saturable NLS in fig. 6.16.

Breathers, fusion, and the birth of new particles may be useful for computation in our context, but are more difficult to study and characterize than collisions that conserve the shape and number of particles, especially because these phenomena often seem to be accompanied by more radiation. In fact, our definition of information transfer is not applicable to these situations. However, the idea of information transfer may be generalizable to all interactions in a wide class of nonintegrable systems.[4]

[4]We thank an anonymous reviewer for this observation.

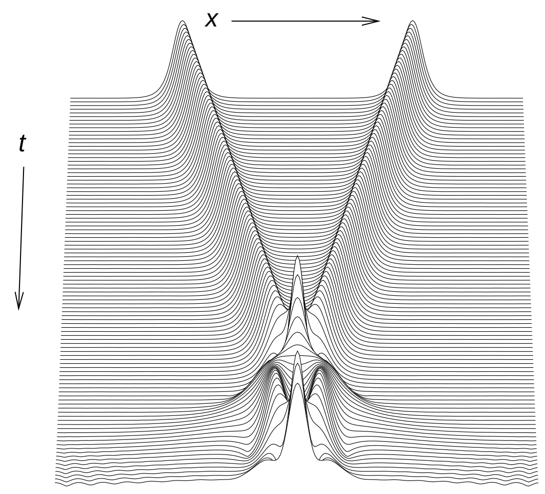

Figure 6.16: Fusion of two solitons after collision. Here the two solitons approach each other with velocities ± 0.2 and amplitudes 1.0.

When wave velocities are very low (< 1.0) and relative phases are approximately in the range 0.0 to 0.3, collisions produce "breathers," or waves whose amplitude pulsates regularly, which cannot arise from eq. 4.4. However, we observed that other collisions result in waves that can be specified by eq. 4.4.

To test the hypothesis that collision products are of the same form as the original waves, we measured the amplitudes, envelope velocities, and phases at the peaks of waves after collisions; we then used these parameters to plot "fresh" waves and to compare their characteristics with those of the post-collision waves. In particular, we compared the carrier velocities of the "fresh" and post-collision waves, and observed what happens in collisions between two "fresh" waves and between two post-collision waves. The results do suggest that the post-collision waves have the form of eq. 4.4.

We estimated radiation for the collisions of fig. 6.15 by finding the fixed-size section of the numerical-solution grid with the lowest root-mean-squared (RMS) norm of the grid points.[5] Ideally, this RMS norm should be very close to zero for solitary waves.

[5]We use circular boundary conditions in our numerical simulations, so that any radiation generated by collisions remains in the system.

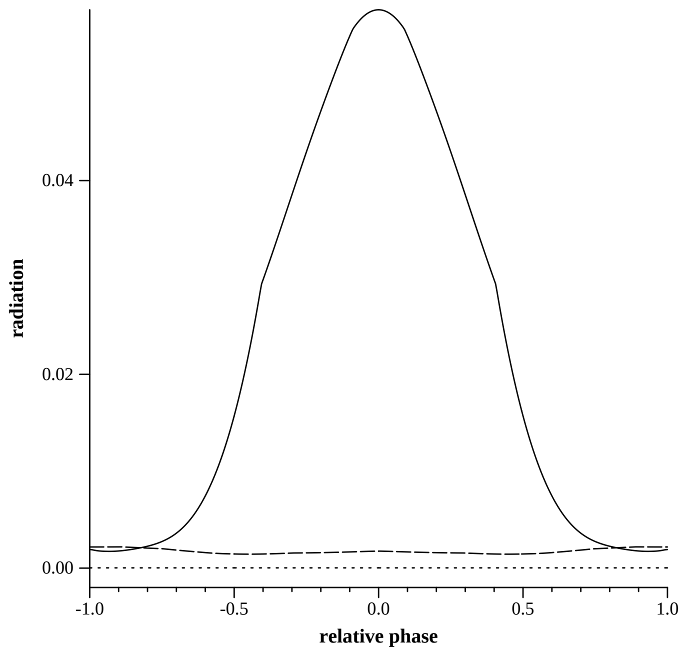

Figure 6.17: Radiation due to collisions in the sat-NLS system. Radiation is computed by finding the section of the numerical-solution grid with the lowest RMS norm of grid points, using sections of size $N/10$, where N is the the size of the entire grid; radiation is taken to be this lowest RMS norm. The solid, dashed, and dotted curves show radiation for collisions of two solitary waves with velocities ± 0.5, ± 1.5, and ± 10.0, respectively. Relative phase is in multiples of π.

Numerical error caused by the discrete nature of time and space in the grid contributes some noise, which we measured for the analytically solvable case of the 3-NLS by comparing numerical results with exact solutions. Based on these investigations, it appears that our simple measure of radiation gives a good general idea of the usefulness of various collisions for computation. In fig. 6.17, radiation is plotted as a function of the relative phase of two colliding waves.

The sat-NLS solitary waves that appear to hold promise for encoding computation have relative speeds from about 0.2 to 4.0, and relative phases whose absolute values range from about 0.2π to 0.8π. Frauenkron et al. report [25] numerical studies of a quintic perturbation of the cubic NLS, and show that radiation in that system is $O(\epsilon^2)$ while energy exchange is first order — a general indication that the phenomena involved in information transfer can dominate radiation in nonintegrable variations of the NLS.

6.7 Summary and questions

We have explored several soliton systems, with the goal of using them for computation in a one-dimensional homogeneous bulk medium. We defined soliton machines (SMs) to model integrable and nonintegrable soliton systems, and found that a class of integrable PDEs cannot support universal computation under the OSM model. In addition, we proved that the SM model is universal in general, and suggested that log-NLS and sat-NLS solitons may be capable of realizing universal SMs, since collisions of such solitons transfer information in nontrivial ways.

Many open problems remain. Foremost among these is determining whether or not gaussons and sat-NLS solitons have behavior sufficiently complex and stable to implement a universal SM. We found three velocity regions in which gaussons have different behavior. Gaussons with low velocities (region 1) offer the most promise for realizing useful computation, since their collisions appear both elastic and nonoblivious. We may be able to use the phase-coding approach in [93] to implement useful computation with gausson interactions. Collisions of gaussons with higher velocities (regions 2 and 3) appear in general to be either oblivious or radiating, though for some combinations of velocities and phases, these collisions are nonoblivious and very near-elastic. The search for answers is complicated by the necessity of numerical solution of the log-NLS equation.

Although we know of no physical realization of the log-NLS equation, the sat-NLS equation models physical systems commonly studied in experiments. Other nonintegrable nonlinear PDEs that model physical behavior also offer possibilities for implementing SMs. For example, the Klein-Gordon equation [3], the NLS equation with additional terms to model optical fiber loss and dispersion, and the coupled NLS equation for birefringent optical fibers [38, 96] all support soliton collisions with complex behavior potentially useful for encoding SMs. Optical solitons that arise from these more complicated equations exhibit gausson-like behavior, and are easily realizable in physical fibers; thus, such optical solitons may be particularly useful as practical means of computing using SMs. *Near-integrable* equations [58], or slightly altered versions of integrable equations, could also offer possibilities for implementing general SMs.

In addition, we may consider using solitons in two or three dimensions [27, 74]. Gaussons, for example, exist in any number of dimensions, and display behavior similar to that in one dimension. The added degrees of freedom of movement in two or more dimensions may enable implementation of universal systems such as the billiard ball computation model [61] or lattice gas models [88].

Although we conjecture that single-mode integrable systems support only trivial information transfer, in the next chapter we consider the coupled integrable Manakov system as a basis for PM computation. We find that the coupled solitons in this system support nontrivial information transfer, and give examples of computation using collisions of these solitons. We leave as an open problem the possibility of useful computation in nonintegrable soliton systems.

Chapter 7

Computation in the Manakov System

The Manakov system [60], which consists of two coupled NLS equations, is a model for light behavior in certain nonlinear media. Radhakrishnan et al. [71] have recently given a general bright two-soliton solution for this system, and derived explicit asymptotic results for collisions in the anomalous dispersion region. Those results are remarkable because they show large energy switching between modes in an integrable system. Surprisingly, as we show in this chapter, the parameters controlling this switching exhibit non-trivial information transformations [45], contradicting an earlier conjecture that this was not possible in integrable systems [44]. Furthermore, these transformations can be used to implement PM computations, or logic operations in a self-restoring discrete domain, suggesting exciting possibilities for all-soliton digital information processing in nonlinear optical media without radiative losses.

In this chapter we use the explicit two-soliton solutions in [71] to show that in the coupled Manakov system:

- An appropriately defined polarization state [109] which is a single complex number can be used to characterize a soliton in collisions. Thus, two degrees of freedom per soliton suffice to describe state transformations in collisions, instead of the six degrees of freedom in a complete description of a Manakov soliton.

- The transformations of this state caused by collisions are given by explicit linear fractional transformations of the extended complex plane. These transformations depend on the total energies and velocities of the solitons (the complex k parameters), which are invariant in collisions, but which can be used to tailor desired transformations.

In order to make use of the basic state transformations, we will derive some of their features and limitations. We view a particle in state ρ_1 as an operator T_{ρ_1} that transforms the state of any other particle by colliding with it. Then we show, among other results, that every such operator has a simply determined inverse; that the only fixed points of such an operator are ρ_1 and its inverse; that no such operator effects a pure rotation of the complex state for all operands; that by concatenating such operators a pure rotation operator can be achieved; that certain sequences of such operators map the unit circle to itself; and so on.

Finally, we discuss the application of these ideas to implementing PM-based all-optical computation without employing physically discrete components. Such a computing machine would be based on the propagation and collision of solitons, and could use conservative logic operations [26], since the collisions we consider preserve the total energy and number of solitons. Finding a sufficiently powerful set of operators and reusable particles in this regime would open the way for integrated computation in homogeneous nonlinear optical media [89, 42], quite a different scheme from using soliton-dragging gates [38] as discrete components to build a computer.

7.1 Informational state in the Manakov system

7.1.1 The Manakov system and its solutions

We review the integrable one-dimensional Manakov system [60, 47, 70] and its analytical solutions from [71]. The system consists of two coupled NLS equations,

$$\begin{aligned} iq_{1t} + q_{1xx} + 2\mu(|q_1|^2 + |q_2|^2)q_1 &= 0, \\ iq_{2t} + q_{2xx} + 2\mu(|q_1|^2 + |q_2|^2)q_2 &= 0, \end{aligned} \tag{7.1}$$

where $q_1 = q_1(x,t)$ and $q_2 = q_2(x,t)$ are the complex amplitudes of two interacting optical modes, μ is a positive parameter, and x and t are normalized space and time. Note that in order for t to represent the propagation variable, as in Manakov's original paper [60], our variables x and t are interchanged with those of [71]. The system admits single-soliton solutions consisting of two components,

$$\begin{aligned} q_1 &= \frac{\alpha}{2} e^{-\frac{R}{2}+i\eta_I} \operatorname{sech}(\eta_R + \frac{R}{2}), \\ q_2 &= \frac{\beta}{2} e^{-\frac{R}{2}+i\eta_I} \operatorname{sech}(\eta_R + \frac{R}{2}), \end{aligned} \tag{7.2}$$

where

$$\eta = k(x + ikt), \tag{7.3}$$

$$e^R = \frac{\mu(|\alpha|^2 + |\beta^2|)}{k + k^*}, \tag{7.4}$$

and α, β, and k are arbitrary complex parameters. Subscripts R and I on η and k indicate real and imaginary parts. Note that $k_R \neq 0$.

To study the effects of collisions on soliton states in this system, we consider the analytical two-soliton solution given in [71]. By taking limits of this solution as $t \to \pm\infty$ and $x \to \pm\infty$, we find asymptotic formulas for the widely separated solitons before and after a collision. Let $q^{(1)}, q^{(2)}, q^{(L)}$ and $q^{(R)}$ denote these four solitons:

$$q_1^{(1)} = \frac{\alpha_1}{2} e^{\frac{-R_1}{2}+i\eta_{1I}} \operatorname{sech}(\eta_{1R} + \frac{R_1}{2}), \tag{7.5}$$

$$q_2^{(1)} = \frac{\beta_1}{2} e^{\frac{-R_1}{2}+i\eta_{1I}} \operatorname{sech}(\eta_{1R} + \frac{R_1}{2}), \tag{7.6}$$

$$q_1^{(2)} = \frac{\alpha_2}{2} e^{\frac{-R_2}{2}+i\eta_{2I}} \operatorname{sech}(\eta_{2R} + \frac{R_2}{2}), \tag{7.7}$$

$$q_2^{(2)} = \frac{\beta_2}{2} e^{\frac{-R_2}{2}+i\eta_{2I}} \operatorname{sech}(\eta_{2R} + \frac{R_2}{2}), \tag{7.8}$$

$$q_1^{(L)} = \frac{1}{2} e^{\delta_1-\frac{R_1+R_3}{2}+i\eta_{2I}} \operatorname{sech}(\eta_{2R} + \frac{R_3-R_1}{2}), \tag{7.9}$$

$$q_2^{(L)} = \frac{1}{2} e^{\delta_1'-\frac{R_1+R_3}{2}+i\eta_{2I}} \operatorname{sech}(\eta_{2R} + \frac{R_3-R_1}{2}), \tag{7.10}$$

$$q_1^{(R)} = \frac{1}{2} e^{\delta_2-\frac{R_2+R_3}{2}+i\eta_{1I}} \operatorname{sech}(\eta_{1R} + \frac{R_3-R_2}{2}), \tag{7.11}$$

$$q_2^{(R)} = \frac{1}{2} e^{\delta_2'-\frac{R_2+R_3}{2}+i\eta_{1I}} \operatorname{sech}(\eta_{1R} + \frac{R_3-R_2}{2}), \tag{7.12}$$

where

$$e^{R_1} = \frac{\kappa_{11}}{k_1+k_1^*}, \tag{7.13}$$

$$e^{R_2} = \frac{\kappa_{22}}{k_2+k_2^*}, \tag{7.14}$$

$$e^{R_3} = \frac{|k_1-k_2|^2}{(k_1+k_1^*)(k_2+k_2^*)|k_1+k_2^*|^2}(\kappa_{11}\kappa_{22}-\kappa_{12}\kappa_{21}), \tag{7.15}$$

$$e^{\delta_1} = \frac{k_1-k_2}{(k_1+k_1^*)(k_1^*+k_2)}(\alpha_1\kappa_{21}-\alpha_2\kappa_{11}), \tag{7.16}$$

$$e^{\delta_2} = \frac{k_2-k_1}{(k_2+k_2^*)(k_1+k_2^*)}(\alpha_1\kappa_{21}-\alpha_2\kappa_{11}), \tag{7.17}$$

$$e^{\delta_1'} = \frac{k_1-k_2}{(k_1+k_1^*)(k_1^*+k_2)}(\beta_1\kappa_{21}-\beta_2\kappa_{11}), \tag{7.18}$$

$$e^{\delta_2'} = \frac{k_2-k_1}{(k_2+k_2^*)(k_1+k_2^*)}(\beta_1\kappa_{21}-\beta_2\kappa_{11}), \tag{7.19}$$

Sign of k_{1R}	Sign of k_{2R}	Solitons before collision	Solitons after collision
$+$	$+$	$q^{(1)}, q^{(L)}$	$q^{(2)}, q^{(R)}$
$-$	$+$	$q^{(1)}, q^{(2)}$	$q^{(L)}, q^{(R)}$
$+$	$-$	$q^{(R)}, q^{(L)}$	$q^{(2)}, q^{(1)}$
$-$	$-$	$q^{(R)}, q^{(2)}$	$q^{(L)}, q^{(1)}$

Table 7.1: Asymptotic forms of solitons before and after collisions, as determined by the signs of the soliton parameters k_R.

and

$$\kappa_{ij} = \frac{\mu(\alpha_i \alpha_j^* + \beta_i \beta_j^*)}{k_i + k_j^*}. \tag{7.20}$$

To determine the effects of collisions on solitons, eqs. 7.5–7.12 can be compared to the single-soliton solution (eq. 7.2). The interpretation of these asymptotic equations in terms of the actual motions of the solitons depends on the signs of the parameters k_{1R} and k_{2R}; there are four possible cases, listed in table 7.1. Note that $k_{jR} \neq 0$.

In the next subsection, we determine the effects of collisions on solitons by comparing these formulas with eq. 7.2.

7.1.2 State in the Manakov system

The three complex numbers α, β, and k (with six degrees of freedom) in eq. 7.2 characterize bright solitons in the Manakov system. Since k is unchanged by collisions, two degrees of freedom can be removed immediately from an informational state characterization. We note that Manakov [60] removed an additional degree of freedom by normalizing the polarization vector determined by α and β by the total magnitude $(\alpha^2 + \beta^2)^{1/2}$. However, we show that the single complex-valued polarization state $\rho = \alpha/\beta$, with only two degrees of freedom [109], suffices to characterize two-soliton collisions when the constants k of both solitons are given.

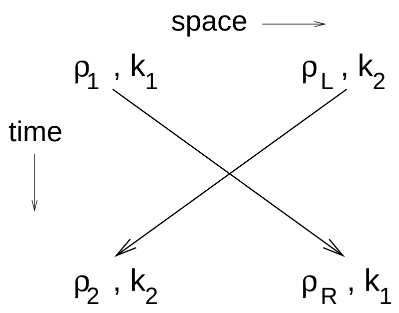

Figure 7.1: A general two-soliton collision in the Manakov system. The complex numbers ρ_1, ρ_L, ρ_2, and ρ_R indicate the variable soliton states; k_1 and k_2 indicate the constant soliton parameters.

We use the tuple (ρ, k) to refer to a soliton with variable state ρ and constant parameter k:

- $\rho = q_1(x,t)/q_2(x,t) = \alpha/\beta$: a complex number[1], constant between collisions;
- $k = k_R + ik_I$: a complex number, with $k_R \neq 0$.

Consider a two-soliton collision, and let k_1 and k_2 represent the constant soliton parameters. Let ρ_1 and ρ_L denote the respective soliton states before impact. Suppose the collision transforms ρ_1 into ρ_R, and ρ_L into ρ_2 (see fig. 7.1). In the rest of this chapter we always associate k_1 and ρ_1 with the right-moving particle, and k_2 and ρ_L with the left-moving particle. To specify these state transformations, we write

$$T_{\rho_1}(\rho_L) = \rho_2, \tag{7.21}$$

$$T_{\rho_L}(\rho_1) = \rho_R. \tag{7.22}$$

The soliton velocities are determined by k_{1I} and k_{2I}, and are therefore constant.

To determine the state changes undergone by the colliding solitons, we take the limits $x \to \pm\infty$ and $t \to \pm\infty$ in the two-soliton expression from [71], as described

[1]Throughout this chapter we use the complex plane extended to include the point at infinity.

earlier. These limits depend on the signs of k_{1R} and k_{2R}; there are four cases, each of which yields asymptotic formulas for both components of each soliton before and after the collision, as given by eqs. 7.5 through 7.12 and table 7.1.

We find each soliton's state by computing the quotient of the soliton's two components. When $k_{1R} > 0$ and $k_{2R} > 0$, we obtain

$$\rho_2 = \frac{[(1-g)/\rho_1^* + \rho_1]\rho_L + g\rho_1/\rho_1^*}{g\rho_L + (1-g)\rho_1 + 1/\rho_1^*}, \tag{7.23}$$

where

$$g = \frac{k_1 + k_1^*}{k_2 + k_1^*}. \tag{7.24}$$

By a symmetry argument, we obtain

$$\rho_R = \frac{[(1-h^*)/\rho_L^* + \rho_L]\rho_1 + h^*\rho_L/\rho_L^*}{h^*\rho_1 + (1-h^*)\rho_L + 1/\rho_L^*}, \tag{7.25}$$

where

$$h = \frac{k_2 + k_2^*}{k_1 + k_2^*}. \tag{7.26}$$

Note that the state changes given by eqs. 7.23 and 7.25 (and by eqs. 7.21 and 7.22) depend on the constants k_1 and k_2. We often omit these from expressions, as in eqs. 7.21 and 7.22. However, when we need to specify the values of k_1 and k_2 explicitly, we write

$$T_{\rho_1,k_1}(\rho_L, k_2) = \rho_2. \tag{7.27}$$

For the remaining three cases of signs of k_{1R} and k_{2R}, we used similar methods to find six additional state-change expressions:

- Case 2: $k_{1R} < 0, k_{2R} > 0$

$$\rho_2 = \frac{[(g-1)\rho_1 - 1/\rho_1^*]\rho_L + g\rho_1/\rho_1^*}{g\rho_L + (g-1)/\rho_1^* - \rho_1} \tag{7.28}$$

$$\rho_R = \frac{[(h-1)\rho_L| - 1/\rho_L^*]\rho_1 + h\rho_L/\rho_L^*}{h\rho_1 + (h-1)/\rho_L^* - \rho_L} \tag{7.29}$$

- Case 3: $k_{1R} > 0, k_{2R} < 0$

$$\rho_2 = \frac{[(g^*-1)\rho_1 - 1/\rho_1^*]\rho_L + g^*\rho_1/\rho_1^*}{g^*\rho_L + (g^*-1)/\rho_1^* - \rho_1} \tag{7.30}$$

$$\rho_R = \frac{[(h^*-1)\rho_L - 1/\rho_L^*]\rho_1 + h^*\rho_L/\rho_L^*}{h^*\rho_1 + (h^*-1)/\rho_L^* - \rho_L} \tag{7.31}$$

- Case 4: $k_{1R} < 0, k_{2R} < 0$

$$\rho_2 = \frac{[(1-g^*)/\rho_1^* + \rho_1]\rho_L + g^*\rho_1/\rho_1^*}{g^*\rho_L + (1-g^*)\rho_1 + 1/\rho_1^*} \tag{7.32}$$

$$\rho_R = \frac{[(1-h)/\rho_L^* + \rho_L]\rho_1 + h\rho_L/\rho_L^*}{h\rho_1 + (1-h)\rho_L + 1/\rho_L^*} \tag{7.33}$$

It is a matter of algebra to verify that these six formulas can be obtained from eqs. 7.23 and 7.25 by using the following relations:

$$T_{\rho_1,k_{1R}+ik_{1I}}(\rho_L, k_{2R}+ik_{2I}) \rightarrow T_{-1/\rho_1^*,-k_{1R}+ik_{1I}}(\rho_L, k_{2R}+ik_{2I}), \tag{7.34}$$

$$T_{\rho_L,k_{2R}+ik_{2I}}(\rho_1, k_{1R}+ik_{1I}), \rightarrow T_{-1/\rho_L^*,k_{2R}-ik_{2I}}(\rho_1, -k_{1R}-ik_{1I}). \tag{7.35}$$

Relation 7.34 states that when the sign of k_{1R} is changed in eq. 7.23, we must also replace ρ_1 with $-1/\rho_1^*$ in the same equation in order to obtain the correct formula for the state change. The particle $-1/\rho_1^*$ has a special significance — it acts as inverse operator to ρ_1 (see section 7.2). The same relation can also be used to obtain the

Sign of k_{1R}	Sign of k_{2R}	T_{0,k_1}	T_{0,k_2}	T_{∞,k_1}	T_{∞,k_2}
$+$	$+$	$1-g$	$1-h^*$	$1/(1-g)$	$1/(1-h^*)$
$-$	$+$	$1/(1-g)$	$1/(1-h)$	$1-g$	$1-h$
$+$	$-$	$1/(1-g^*)$	$1/(1-h^*)$	$1-g^*$	$1-h^*$
$-$	$-$	$1-g^*$	$1-h$	$1/(1-g^*)$	$1/(1-h)$

Table 7.2: State-change factors for T_0 and T_∞ transformations. The columns for T_{0,k_1} and T_{∞,k_1} list the factors by which ρ_1 is multiplied to get ρ_R, and the columns for T_{0,k_2} and T_{∞,k_2} list the factors by which ρ_L is multiplied to get ρ_2.

proper state-change formula when the sign of k_{2R} is changed in eq. 7.25. Relation 7.35 can be used to obtain the state-change expressions when the sign of k_{2R}, is changed in eq. 7.23, or when the sign of k_{1R} is changed in eq. 7.25. Taken together, relations 7.34 and 7.35 imply

$$T_{\rho_1,k_{1R}+ik_{1I}}(\rho_L,k_{2R}+ik_{2I}) \;\rightarrow\; T_{\rho_1,-k_{1R}-ik_{1I}}(\rho_L,-k_{2R}-ik_{2I}); \qquad (7.36)$$

that is, when the signs of both k_{1R} and k_{2R} are switched in either eq. 7.23 or eq. 7.25, we must also conjugate k_1 and k_2 in the equations in order to obtain the correct state-change expressions. It can also be verified that all four cases above collapse to the first if we simply use $|k_{1R}|$ and $|k_{2R}|$ in eqs. 7.23— 7.26, although this does not seem initially obvious.

We assume the soliton velocities are such that a collision occurs; that is, $k_{1I} > k_{2I}$. Also, note that $g,h \neq 0$, since $k_{1R},k_{2R} \neq 0$, and that g and h cannot be pure real numbers.

In each of the four cases mentioned above, the soliton states after collision are completely determined by the soliton states before collision, which shows that our definition of state is complete. The k parameter of a soliton remains constant, but is in general different for different solitons.

A special class of state transformations

The class of transformations given by $T_{0,k}$ and $T_{\infty,k}$, where $k = k_1$ (right-moving) or k_2 (left-moving), will be useful later. They specify state changes caused by collisions with solitons whose entire energy is contained in only one mode, and are functions of k_1 and k_2. Table 7.1.2 shows the state-change factors due to these transformations. It is not hard to verify from this table that

$$T_{0,k_R+ik_I} \quad \rightarrow \quad T_{\infty,-k_R+ik_I}, \tag{7.37}$$

which is a special case of the state-change relations described earlier.

7.2 Properties of the collision state transformation

For concreteness we will restrict attention in this section to the case $k_{1R}, k_{2R} > 0$, and the transformation eq. 7.23 of the left-moving particle in state ρ_L to the left-moving particle in state ρ_2. All the results hold for other signs of k_{1R} and k_{2R} and the other collisions with appropriate changes in the particle names and the parameter that plays the role of g. When the signs of k_{1R} and k_{2R} are an issue, we will mention that explicitly.

A state transformation can be viewed either as a mapping $T_{\rho_1}(\rho_L)$ from the complex ρ_L-plane to the complex ρ_2-plane, or in general as a mapping from the complex plane to itself, depending on the context. The state transformation is in fact a *linear fractional transformation* (LFT) (or *bilinear* or *Möbius*) of the form

$$\rho_2 = \frac{a\rho_L + b}{c\rho_L + d}, \tag{7.38}$$

where the coefficients are functions of the right-moving particle in state ρ_1,

$$\begin{aligned} a &= (1-g)/\rho_1^* + \rho_1 \\ b &= g\rho_1/\rho_1^* \\ c &= g \\ d &= (1-g)\rho_1 + 1/\rho_1^* \end{aligned}$$

The choice of a, b, c, and d is not unique, since we can multiply numerator and denominator by an arbitrary nonzero number, but we will use these throughout this chapter. The limiting versions of eq. 7.38 then give $T_\infty(\rho) = (1/(1-g))\rho$ and $T_0(\rho) = (1-g)\rho$.

When there is no danger of confusion we will refer to particles and their states interchangeably, so for example we can speak of "transforming the particle ρ_L." By eq. 7.22, the collision above also results in the transformation $T_{\rho_L}(\rho_1)$ of ρ_1 caused by collision with ρ_L, and each result we give about the properties of the transformations of left-moving particles has its symmetrical counterpart about transformations of right-moving particles.

It is usually assumed that an LFT must have a nonzero determinant $ac - bd$, which ensures that it is nonconstant. This is always true for our class of LFTs and a straightforward calculation shows

Property 1 (Determinant) *The LFT given by eq. 7.38 has determinant*

$$(1-g)(\rho_1 + 1/\rho_1^*), \tag{7.39}$$

which cannot vanish, since $g \neq 1$.

Property 2 (Inverse) *Every operator T_{ρ_1} has a unique inverse T_σ, where $\sigma = -1/\rho_1^*$ and g is the same for ρ_1 and its inverse.*

Proof: Replacing ρ_1 by $-1/\rho_1^*$ in the expressions above for $a,\ b,\ c,\ d$ above results in $-d,\ b,\ c, -a$, which are the coefficients in the inverse of eq. 7.38. Uniqueness follows because the set of all LFTs forms a group. ∎

We refer to a particle ρ followed by its inverse $-1/\rho^*$ as an *inverse pair.* It follows from the next result that collision with an inverse pair leaves any sequence of particles unchanged.

Property 3 (Preservation of inverse pairs) *If an inverse pair collides with any particle, the two resulting particles also form an inverse pair.*

Proof: Replacing $\partial/\partial t$ by $-\partial/\partial t$ in the original Manakov system (eq. 7.1) shows that if the system is run backwards in time, the same collision rules apply if solutions are replaced by their conjugates. Thus if an inverse pair leaves a particle σ invariant, the conjugates of the collision products of the inverse pair do also, and hence the collision products must themselves be an inverse pair. ∎

Property 4 (Fixed points) *Every operator T_{ρ_1} has exactly two distinct fixed points, ρ_1 and $-1/\rho_1^*$. It follows that a particle is transparent to itself and the particle corresponding to its inverse operator, and to no other particles.*

Proof: The fixed-point condition $T_{\rho_1}(\rho_L) = \rho_L$ using eq. 7.38 leads to a quadratic equation, since $c = g \neq 0$. There are therefore at most two fixed points. The stated fixed points are always distinct and it is easy to verify that they satisfy eq. 7.38 by direct substitution. ∎

The following property is expected from the fact that the two components of the Manakov system are incoherently coupled.

Property 5 (Rotational invariance of collisions) *If ρ_1 and ρ_L are both rotated by θ, then ρ_2 and ρ_R are also rotated by θ.*

Proof: This is easily verified using the transformation eqs. 7.23 et seq. ∎

Property 6 (Absence of pure rotations or scalars) *There is no (single-collision) operator of the form*

$$T_{\rho_1}(\rho_L) = e^{i\theta}\rho_L \tag{7.40}$$

for any angle θ, or

$$T_{\rho_1}(\rho_L) = K\rho_L \tag{7.41}$$

for any real K. In particular, there is no single-collision identity operator.

Proof: First consider the possibility of pure rotations. The case $\theta = 0$ corresponds to the identity operator, for which every point is a fixed point, contradicting property 4. When $\theta \neq 0$, the fixed points of a pure rotation are 0 and ∞, so any pure rotation must be a T_0 or T_∞. Since every T_∞ is the inverse of a T_0, it suffices to consider the case of a T_0, which has the operator $T_0(\rho_L) = (1-g)\rho_L$. We can write

$$1 - g = \frac{k_{2R} - k_{1R} + i}{k_{2R} + k_{1R} + i}, \tag{7.42}$$

where we normalize by setting

$$\Delta = k_{2I} - k_{1I} = 1.$$

This normalization is allowed because the difference in k_I's represents the difference in envelope velocities, and so must be nonzero if there is to be a collision. The magnitude of $1 - g$ in eq. 7.42 cannot be one unless either $k_{1R} = 0$ or $k_{2R} = 0$, which is not allowed.

For the possibility of a scalar multiplication, we can again restrict attention to T_0's and eq. 7.42, by the same reasoning as above. The right-hand side of that equation cannot be real unless $k_{1R} = 0$. ∎

The composition of any number of LFTs can be written as an LFT $w = L(z)$, and if it has exactly two distinct fixed points z_1 and z_2, it can be written in the implicit form

$$\frac{w - z_1}{w - z_2} = K \cdot \frac{z - z_1}{z - z_2}. \tag{7.43}$$

In the single-collision case this becomes

Property 7 (Implicit form) *The single-collision transformation $T_{\rho_1}(\rho_L)$ can be written in the implicit form*

$$\frac{\rho_2 - \rho_1}{\rho_2 + 1/\rho_1^*} = K \cdot \frac{\rho_L - \rho_1}{\rho_L + 1/\rho_1^*}, \tag{7.44}$$

where $K = (1 - g)$ is called the "invariant" of the LFT. By symmetry, the inverse transformation is the same except K is replaced by $1/K$.

The next result is standard [7].

Property 8 (Invariant circles) *The fixed points of a LFT with two distinct fixed points determine two orthogonal families of circles in the w plane: (1) $\mathcal{C}_1$, which are the circles that pass through the fixed points; and (2) $\mathcal{C}_2$, which are the circles determined by the condition that the distances to the fixed points have a constant ratio, the* circles of Apollonius. *These are images, respectively, of points through the* *origin and concentric circles about the origin in the w plane. The circles $\mathcal{C}_1$ are mapped into themselves as a set, and similarly for the circles $\mathcal{C}_2$. For each circle in $\mathcal{C}_1$ to be mapped onto itself, we require that invariant $K = 1 - g$ be real, which is not possible for a single collision. (The mapping in this case is called* hyperbolic.*) For each circle in $\mathcal{C}_2$ to be mapped onto itself, we require that $|K| = 1$, which is also not possible in the single-collision case. (The mapping in this case is called* elliptic.*)*

One consequence of this last result is that if we can design a group of particles with a real or modulus-one invariant K, and arrange for the unit circle to be in $\mathcal{C}_1$ or $\mathcal{C}_2$, then the net effect of collisions with these particles will be to map the unit circle to itself. That is, the modulus-one property of particles will be preserved on collision with these "operator" groups, and we will effectively have a state variable with one degree of freedom in the "processed" particles.

Property 9 (Invariant of multiple collision) *Consider a composite collision with a set of particles, each of which is either a given ρ or its inverse, and each having a possibly different invariant K_j. The fixed points are the same as those of ρ (ρ and $-1/\rho^*$) and the invariant is $\prod(K_j^{\pm 1})$, where we use K_j^{+1} for the ρ's and K_j^{-1} for the inverses.*

Proof: The fact that the fixed points are those of ρ is an immediate consequence of Property 4. We need to consider only the two cases of collision with two copies of ρ, and with ρ and its inverse; the general result then follows by induction on the number of collisions. The invariant for a transformation with fixed points ρ and $-1/\rho^*$ can be written [52]

$$K = \frac{a - c\rho}{a + c/\rho^*}, \tag{7.45}$$

using the coefficients a and c in eq. 7.38. The result for these two cases follows by straightforward algebra. ∎

The impossibility of achieving a pure rotation with a single collision suggests looking for multiple collisions that do have that effect. We do so in the next section, where we exhibit pure rotation operators realized by composite particles composed of T_0's and T_∞'s. This is achieved by carefully designing the particles' k parameters.

7.3 Particle design

We conclude with some examples which illustrate the design of groups of particles that effect certain transformations that have potential application to embedded logic. Future work will explore the limits of this approach, and especially the question of the extent to which arithmetic and possibly general computation can be encoded in this system.

7.3.1 An i operator

A simple nontrivial operator is pure rotation by $\pi/2$, or multiplication by i. This changes linearly polarized solitons to circularly polarized solitons, and vice versa. A numerical search yielded the useful transformations (see table 7.2)

$$T_{\rho_L}(\rho) = T_{0,1-i}(\rho, 1+i) = (1 - h^*(1+i, 1-i))\rho = \frac{1}{\sqrt{2}} e^{-\frac{\pi}{4} i} \rho, \tag{7.46}$$

$$T_{\rho_L}(\rho) = T_{\infty,5-i}(\rho, 1+i) = \frac{\rho}{1 - h^*(1+i, 5-i)} = \sqrt{2} e^{\frac{3\pi}{4} i} \rho, \tag{7.47}$$

which, when composed, result in the transformation

$$U(\rho, 1+i) = i\rho. \tag{7.48}$$

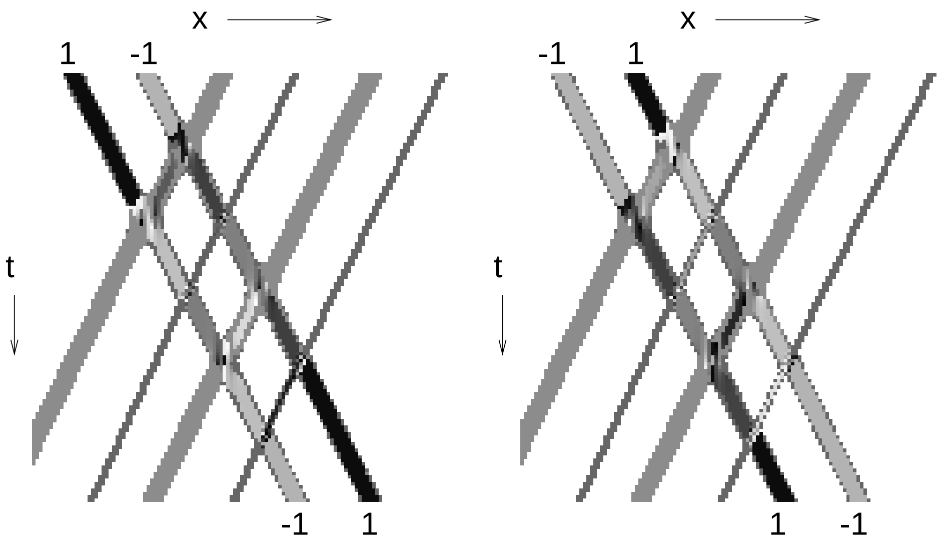

Figure 7.2: Numerical simulation of a NOT processor implemented in the Manakov system. These graphs display the color-coded phase of ρ for solitons that encode data and processors for two cases. In the initial conditions (top of graphs), the two leftmost (data) solitons are an inverse pair that can represent a 0 in the left graph, and a 1 in the right graph. In each graph, these solitons collide with the four rightmost (processor) solitons, resulting in a soliton pair representing a 1 and a 0, respectively. The processor solitons are unchanged. These graphs were obtained by numerical simulation of eq. 7.1 with $\mu = 1$.

(Here we think of the data as right-moving and the operator as left-moving.) We refer to U as an i operator. Its effect is achieved by first colliding a soliton $(\rho, 1+i)$ with $(0, 1-i)$, and then colliding the result with $(\infty, 5-i)$, which yields $(i\rho, 1+i)$.

7.3.2 A -1 operator (NOT processor)

Composing two i operators results in the -1 operator, which with appropriate encoding of information can be used as a logical NOT processor. Figs. 7.2 and 7.3 show a NOT processor with reusable data and operator solitons. The two right-moving particles represent data and are an inverse pair, and thus leave the operator unchanged; the left-moving group comprise the four components of the -1 operator. This figure was obtained by direct numerical simulation of the Manakov system, with initial state

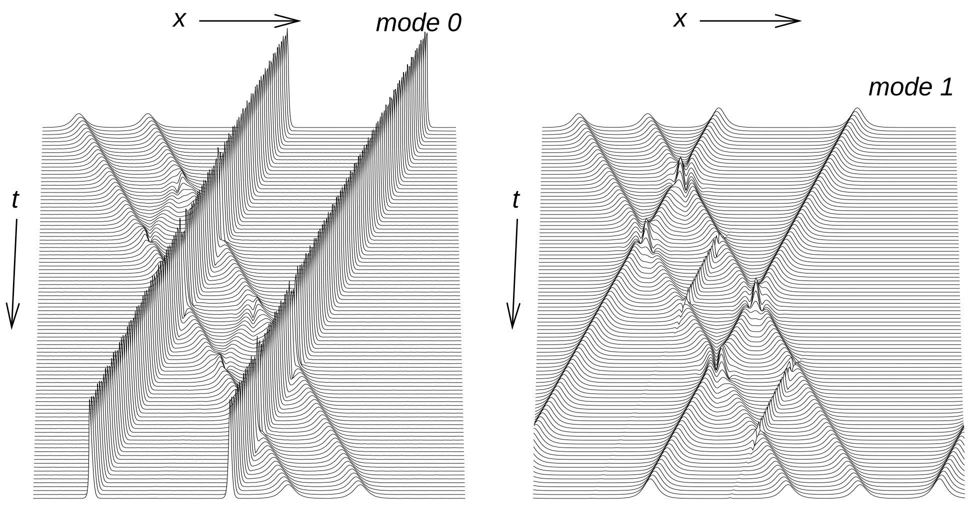

Figure 7.3: Another visualization of numerical simulation of a NOT processor implemented in the Manakov system. These two graphs present the same information as either single graph of fig. 7.2, but display the magnitude of the soliton components in both modes (0 and 1). Each single graph of fig. 7.2 corresponds to two magnitude profiles (one for each mode), but these magnitude graphs are the same for both graphs of fig. 7.2. This is because the data solitons for the two NOT operations differ only in phase, whereas the operator solitons are the same.

that contains the appropriate data and processor solitons.

We may treat this NOT processor as a controlled NOT (or exclusive-or) by observing that the processor solitons can be selected so that both the data and processor solitons are unaffected by collisions with one another. Such processor solitons encode a “zero” control signal, which specifies that no operation be done.

This NOT processor switches the phase of the (right-moving ± 1) data particles, using the energy partition of the (left-moving 0 and ∞) operator particles. A kind of dual NOT gate exists, with the same composite $K = -1$, which uses only the phase of the operator particles to switch the energy of the data particles. In particular, if we use the same k's as in the phase-switching NOT gate, code data as 0 and ∞, and use a sequence of four ± 1 operator particles, the effect is to switch 0 to ∞ and ∞ to 0 — that is, to switch all energy from one component of the data particles to the

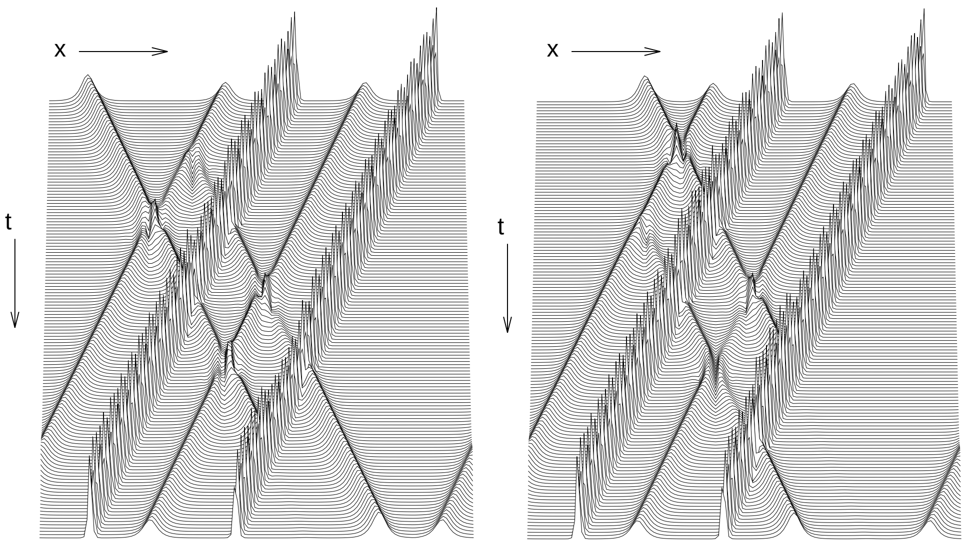

Figure 7.4: Numerical simulation of an energy-switching NOT processor implemented in the Manakov system. These graphs display the magnitude of one soliton component for the same two operations as in the previous figures. Here the right-moving (data) particles are the inverse pair with states $\infty, 0$ in the left graph, and $0, \infty$ in the right graph. As before, the left-moving (operator) particles emerge unchanged, and here have initial and final states ± 1.

other. This can be checked easily by setting $K = -1$ and $\rho_1 = 1$ in the implicit form, eq. 7.44, using Property 9 for this composite collision.

7.3.3 A "move" operator

Fig. 7.5 depicts a simple example of information transfer from one particle to another, reminiscent of an assembly-language MOVE instruction. In the initial conditions of each graph, a "carrier" particle C collides with the middle particle; this collision transfers information from the middle particle to C. The carrier particle then transfers its information to another particle via a collision. The appropriate particles A, B, and C for this operation were found through a numerical search, as with the particles for our NOT gate.

Note that "garbage" particles arise as a result of this "move" operation. In general,

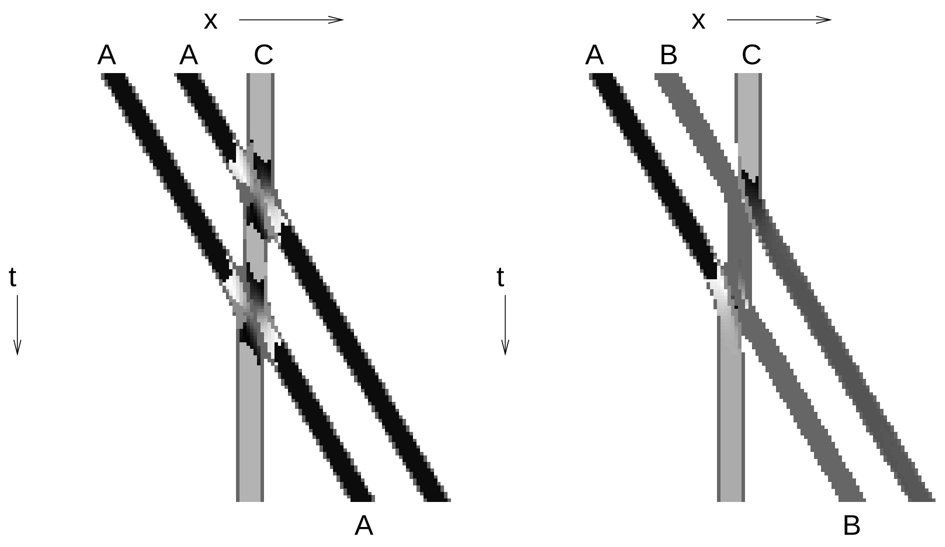

Figure 7.5: Numerical simulation of a "move" operation implemented in the Manakov system. These graphs display the color-coded phase of ρ. In each graph, the information contained in the middle particle in the initial conditions (top of graphs) is moved to the middle particle in the final conditions (bottom of graphs). The information transfer is effected by the "carrier" particle C. These graphs were obtained by numerical simulation of eq. 7.1 with $\mu = 1$.

because the Manakov system is reversible, such "garbage" often appears in computations, and needs to be managed explicitly or used as part of computation, as with conservative logic [26].

7.3.4 Particles that map the unit circle to itself

By Property 8 any composite (multi-particle) operator with an invariant K that is real will map the unit circle to itself if the operator particles are all the same and themselves have state on the unit circle. Let the operator particles have state $\rho = e^{i\theta}$. Then by eq. 7.44, the transformation from z to w is

$$\frac{w - e^{i\theta}}{w + e^{i\theta}} = K \cdot \frac{z - e^{i\theta}}{z + e^{i\theta}}, \tag{7.49}$$

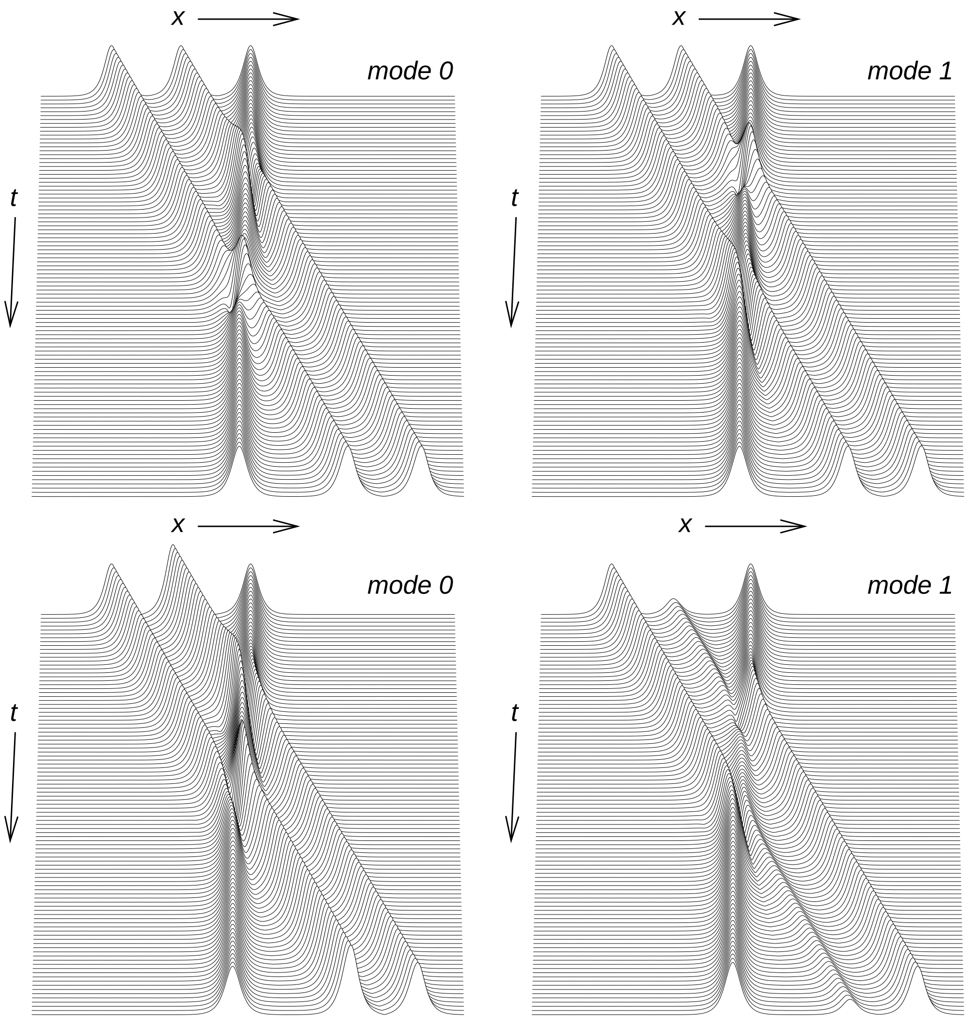

Figure 7.6: Another visualization of numerical simulation of a "move" operation implemented in the Manakov system. These graphs show the same information as fig. 7.5, but display the magnitude of the soliton components in both modes. The top two graphs correspond to the left graph of fig. 7.5, and show the magnitude of the soliton components in modes 0 and 1. The bottom two graphs correspond to the right graph of fig. 7.5, and show analogous information.

and therefore, if $z = e^{i\phi}$ and $w = e^{i\psi}$,

$$\psi = \theta + 2\arctan(K \tan \frac{\phi - \theta}{2}). \tag{7.50}$$

Thus, if we restrict all "data" particles to the unit circle, collision ("operator") particles of this type will preserve that property.

For an example yielding real K, consider the composition of eight identical operator particles with $k_2 = 1 - i$, colliding with a particle having $k_1 = 1 + i$. The resulting invariant is, using $1 - h^* = 2^{-1/2}e^{-\pi i/4}$ from eq. 7.46, $K = (2^{-1/2}e^{-\pi i/4})^8 = 1/16$. We can also mix copies of an operator particle and its inverse and use Property 9 to get a wider variety of invariant values.

7.3.5 Group structure of state LFTs

Our numerical experiments with the state LFTs given by eqs. 7.23 and 7.25 revealed that these LFTs have an interesting algebraic structure. For example, repeated collisions of two particles sometimes return the particles to their pre-collision states. It is possible to adjust the states ρ_1 and ρ_2 of a particle pair so that the particles require a given number of collisions with each other to revert back to these states; we refer to this number as the *period* of the particle pair. Each particle goes through a sequence of states during the collisions, and we can use these intermediate states to create particle sequences whose compound collisions with each other leave the sequences unchanged; we call such particle sequences *braids*. Fig. 7.7 shows an example collision of two braids of six particles each; note that the braids emerge unchanged after all collisions. We found these braids through a numerical search, but algebraic methods could probably be used to find braids and other structures. We leave this as an interesting area for further study.

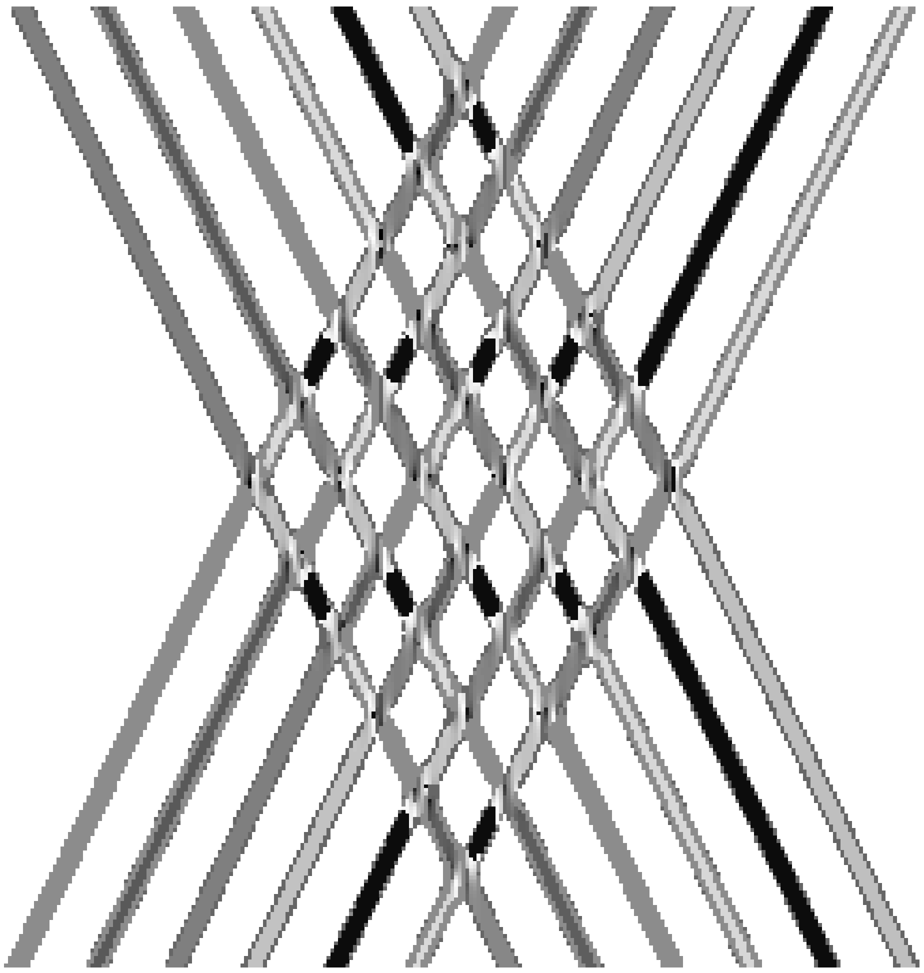

Figure 7.7: Numerical simulation of the compound collision of two soliton braids in the Manakov system. This graph displays the color coded phase of ρ, and was obtained by numerical simulation of eq. 7.1 with $\mu = 1$.

7.4 Discussion

The line of inquiry followed in this chapter suggests that it may be possible to perform useful computation in bulk media by using colliding solitons alone, and leads to many open questions. First, it would be useful to obtain a complete mathematical characterization of the state LFTs obtainable by composing either a finite number — or an infinite number — of the special ones induced by collisions in the Manakov system. Second, we should like to know whether the complex-valued polarization state used here for the Manakov system is also useful in other multimode systems, especially those that are near-integrable and support spatial solitons [45, 8, 25, 78, 81, 83, 87, 86, 97, 82, 91]. Finally, we need to study the computational power of this and related systems from the point of view of implementing logic of some generality. In particular, which systems in $1+1$ or $2+1$ dimensions, integrable or nonintegrable, are Turing-equivalent and therefore universal?

Chapter 8

Summary and Future Work

We have studied a nonstandard method of computation based on solitons in uniform media. Such a computing design is one way to exploit the fine-grain parallelism of physical systems, and potentially to build machines orders of magnitude faster and more efficient than existing computers, at least for certain computational tasks.

This dissertation leaves many interesting directions of study to follow, both experimental and theoretical. It would be instructive to implement one of the computational examples from chapter 7 in an actual laboratory setup, with lasers and special optical fibers, glasses, or crystals. This would provide a proof of concept and encourage further theoretical study to encode more complicated operations in the Manakov system. Perhaps such an experiment could be done in the more easily realizable saturable NLS system, provided that state and state transformations could be characterized for this nonintegrable equation.

Many unanswered questions are left about the Manakov system. Our numerical experiments with the LFTs that describe collisions in this system reveal that the group of these LFTs has a rich and interesting structure. Fully characterizing this structure may open the door to easier design of arbitrary computation in the Manakov system.

The question of Turing universality of this system is an interesting open problem. A less ambitious but currently more useful task is to search for particles to do other computations in the system; since we have exact formulas for the state changes of Manakov solitons, such a search could perhaps be automated, or novel ways could be found to look for various computations.

Finally, there are a great number of other soliton systems, both integrable and nonintegrable, to explore as means of implementing PMs. A question that appears to be open is whether or not a method other than numerical simulation exists to determine state changes of nonintegrable solitons after collision. In general, finding analytical ways to study such systems appears to be an area with many unsolved problems, and solutions of such problems could lead to useful computation in nonintegrable systems.

Bibliography

[1] *Intel Architecture Optimization Manual.* Intel Corp., http://www.intel.com/, 1997.

[2] M. J. Ablowitz and P. A. Clarkson. *Solitons, Nonlinear Evolution Equations, and Inverse Scattering.* Cambridge University Press, Cambridge, UK, 1991.

[3] M. J. Ablowitz, M. D. Kruskal, and J. F. Ladik. Solitary wave collisions. *SIAM Journal of Applied Mathematics*, 36(3):429–437, 1979.

[4] M. J. Ablowitz and H. Segur. *Solitons and the Inverse Scattering Transform.* SIAM, Philadelphia, PA, 1981.

[5] L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.

[6] G. P. Agrawal. *Nonlinear Fiber Optics.* Academic Press, Inc. (Harcourt, Brace and Co.), San Diego, CA, 1995.

[7] L. V. Ahlfors. *Complex Analysis.* McGraw-Hill, New York, third edition, 1979.

[8] J. S. Aitchison, A. M. Weiner, Y. Silberberg, M. K. Oliver, J. L. Jackel, D. E. Leaird, E. M. Vogel, and P. W. Smith. Observation of spatial optical solitons in a nonlinear glass waveguide. *Optics Letters*, 15:471, 1990.

[9] S. A. Akhmanov, V. A. Vysloukh, and A. S. Chirkin. *Optics of Femtosecond Laser Pulses.* American Institute of Physics, New York, 1992.

[10] J. Albert and K. Culik II. A simple universal cellular automaton and its one-way and totalistic version. *Complex Systems*, 1:1–16, 1987.

[11] C. H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17:525–532, 1973.

[12] C. H. Bennett. Notes on the history of reversible computation. *IBM Journal of Research and Development*, 32(1):16–23, 1988.

[13] E. R. Berlekamp, J. H. Conway, and R. K. Guy. *Winning Ways for Your Mathematical Plays.* Academic Press, New York, NY, 1982.

[14] I. Białynicki-Birula and J. Mycielski. Nonlinear wave mechanics. *Annals of Physics*, 100:62–93, 1976.

[15] I. Białynicki-Birula and J. Mycielski. Gaussons: Solitons of the logarithmic Schrödinger equation. *Physica Scripta*, 20:539–544, 1979.

[16] R. Y. Chiao, E. Garmire, and C. H. Townes. Self-trapping of optical beams. *Physical Review Letters*, 13, 1964.

[17] D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London*, A400:97–117, 1985.

[18] D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London*, A439:553–558, 1992.

[19] A. K. Dewdney. On the spaghetti computer and other analog gadgets for problem solving. *Scientific American*, 250(6):19–26, June 1984.

[20] P. G. Drazin and R. S. Johnson. *Solitons: An Introduction.* Cambridge University Press, Cambridge, UK, 1989.

[21] E. Fermi, J. R. Pasta, and S. M. Ulam. Studies of nonlinear problems. In *Collected Papers of Enrico Fermi*, volume 2, pages 978–988. Univ. of Chicago Press, Chicago, IL, 1965.

[22] J. A. Fleck, J. R. Morris, and M. D. Feit. Time-dependent propagation of high energy laser beams through the atmosphere. *Applied Physics*, 10:129–160, 1976.

[23] A. S. Fokas, E. Papadopoulou, and Y. Saridakis. Particles in soliton cellular automata. *Complex Systems*, 3:615–633, 1989.

[24] A. S. Fokas, E. Papadopoulou, and Y. Saridakis. Coherent structures in cellular automata. *Physics Letters*, 147A(7):369–379, 1990.

[25] H. Frauenkron, Y. S. Kivshar, and B. A. Malomed. Multisoliton collisions in nearly integrable systems. *Physical Review E*, 54:2244R–2247R, 1996.

[26] E. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3/4):219, 1982.

[27] N. C. Freeman. Soliton interaction in two dimensions. *Advances in Applied Mechanics*, 20:1–37, 1980.

[28] U. Frisch, D. d'Humie'res, B. Hasslacher, P. L. Y. Pomeau, and J. P. Rivet. Lattice gas hydrodynamics in two and three dimensions. *Complex Systems*, 1:649–707, 1987.

[29] R. Gähler, A. G. Klein, and A. Zeilinger. Neutron optical tests of nonlinear wave mechanics. *Physical Review A*, 23(4):1611–1617, 1981.

[30] C. S. Gardner, J. M. Greene, and M. D. Kruskal. Method for solving the Korteweg-de Vries equation. *Physical Review Letters*, 19:1095–1097, 1967.

[31] A. Goldberg, H. M. Schey, and J. L. Schwartz. Computer-generated motion pictures of one-dimensional quantum- mechanical transmission and reflection phenomena. *American Journal of Physics*, 35(3):177–187, 1967.

[32] C. H. Goldberg. Parity filter automata. *Complex Systems*, 2:91–141, 1988.

[33] H. A. Haus. Optical fiber solitons, their properties and uses. *Proceedings of the IEEE*, 81(7):970–983, 1993.

[34] R. Hirota. Exact solution of the Korteweg-de Vries equation for multiple collisions of solitons. *Physical Review Letters*, 27:1192, 1971.

[35] R. Hirota. Exact solution of the modified Korteweg-de Vries equation for multiple collisions of solitons. *Journal of the Physical Society of Japan*, 33(5):1456–1458, 1972.

[36] R. Hirota. Exact solution of the sine-Gordon equation for multiple collisions of solitons. *Journal of the Physical Society of Japan*, 33(5):1459–1463, 1972.

[37] J. E. Hopcroft and J. D. Ullman. *Introducton to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.

[38] M. N. Islam. *Ultrafast Fiber Switching Devices and Systems.* Cambridge University Press, Cambridge, UK, 1992.

[39] M. N. Islam and C. E. Soccolich. Billiard-ball soliton interaction gates. *Optics Letters*, 16(19):1490–1492, 1991.

[40] N. Islam, J. Singh, , and K. Steiglitz. Soliton phase shifts in a dissipative lattice. *Journal of Applied Physics*, 62(2):689–693, 1987.

[41] M. H. Jakubowski, K. Steiglitz, and R. Squier. State transformations of colliding optical solitons and possible application to computation in bulk media. *Physical Review E*, 58:6752–6758, 1998.

[42] M. H. Jakubowski, K. Steiglitz, and R. K. Squier. Implementation of parallel arithmetic in a cellular automaton. In *1995 Int. Conf. on Application Specific Array Processors, Strasbourg, France (P. Cappello et al., ed.)*, Los Alamitos, CA, July 24–26, 1995. IEEE Computer Society Press.

[43] M. H. Jakubowski, K. Steiglitz, and R. K. Squier. Relative computational power of integrable and nonintegrable soliton systems. In *Fourth Workshop in Physics and Computation: PhysComp '96*, Cambridge, MA, Nov. 1996. New England Complex Systems Institute.

[44] M. H. Jakubowski, K. Steiglitz, and R. K. Squier. When can solitons compute? *Complex Systems*, 10(1):1, 1996.

[45] M. H. Jakubowski, K. Steiglitz, and R. K. Squier. Information transfer between solitary waves in the saturable Schrödinger equation. *Physical Review E*, 56:7267, 1997.

[46] M. H. Jakubowski, K. Steiglitz, and R. K. Squier. Embedding computation in nonlinear optical media using collisions of Manakov solitons. In *Int. Conf. on Complex Systems: ICCS '98*, Cambridge, MA, Oct. 1998. New England Complex Systems Institute.

[47] D. J. Kaup and B. A. Malomed. Soliton trapping and daughter waves in the Manakov model. *Physical Review A*, 48(1):599, 1993.

[48] R. W. Keyes. Miniaturization of electronics and its limits. *IBM Journal of Research and Development*, 32(1):24–28, Jan. 1988.

[49] D. J. Korteweg and G. de Vries. On the change of form of long waves advancing in a rectangular canal, and on a new type of long stationary waves. *Philosophical Magazine*, 39:422–443, 1895.

[50] H. T. Kung. Why systolic architectures? *IEEE Computer*, 15(1):37–46, Jan. 1982.

[51] S. Y. Kung, S. C. Lo, S. N. Jean, and J. N. Hwang. Wavefront array processors: Concept to implementation. *IEEE Computer*, 20(2):18–32, 1987.

[52] A. Kyrala. *Applied Functions of a Complex Variable.* Wiley-Interscience, New York, 1972.

[53] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures.* Morgan Kaufman Publishers, San Mateo, CA, 1992.

[54] R. J. Lipton. DNA solution of hard computational problems. *Science*, 268:542–548, 1995.

[55] P. S. Lomdahl. What is a soliton? *Los Alamos Science*, 10:27–31, 1984.

[56] D. Lopresti. P-NAC: A systolic array for comparing nucleic acid sequences. *IEEE Computer*, 20(2):98–99, 1987.

[57] M. O. Magnasco. Chemical kinetics is Turing universal. *Physical Review E*, 87(6):1190–1193, Feb. 1997.

[58] V. G. Makhankov. Computer and solitons. *Physica Scripta*, 20:558–562, 1979.

[59] V. G. Makhankov. *Soliton Phenomenology.* Kluwer Academic Publishers, Norwell, MA, 1990.

[60] S. V. Manakov. On the theory of two-dimensional stationary self-focusing of electromagnetic waves. *Soviet Physics: JETP*, 38(2):248, Feb. 1974.

[61] N. Margolus. Physics-like models of computation. *Physica*, 10D:81–95, 1984.

[62] Y. V. Matiyasevich. Possible nontraditional methods of establishing unsatisfiability of propositional formulas. *American Mathematical Society Translations*, 178(2):75–77, 1996.

[63] L. F. Mollenauer and K. Smith. Demonstration of soliton transmission over more than 4,000 km in fiber with loss periodically compensated by Raman gain. *Optics Letters*, 13:675–677, 1989.

[64] L. F. Mollenauer, R. H. Stolen, and J. P. Gordon. Experimental observation of picosecond pulse narrowing and solitons in optical fibers. *Physical Review Letters*, 45, 1980.

[65] A. C. Newell and J. V. Moloney. *Nonlinear Optics.* Addison-Wesley, Redwood City, CA, 1992.

[66] J. Oficjalski and I. Białynicki-Birula. Collisions of gaussons. *Acta Physica Polonica*, B9:759–775, 1978.

[67] P. J. Olver and D. H. Sattinger (editors). *Solitons in Physics, Mathematics, and Nonlinear Optics.* Springer-Verlag, New York, NY, 1990.

[68] J. K. Park, K. Steiglitz, and W. P. Thurston. Soliton-like behavior in automata. *Physica*, 19D:423–432, 1986.

[69] G. E. Peterson. Electrical transmission lines as models for soliton propagation in materials: Elementary aspects of video solitons. *AT&T Bell Laboratories Technical Journal*, 63(6):901–919, 1984.

[70] R. Radhakrishnan and M. Lakshmanan. Bright and dark soliton solutions to coupled nonlinear Schrödinger equations. *Journal of Physics A*, 28:2683, 1995.

[71] R. Radhakrishnan, M. Lakshmanan, and J. Hietarinta. Inelastic collision and switching of coupled bright solitons in optical fibers. *Physical Review E*, 56:2213, 1997.

[72] A. L. Robinson. Computing without dissipating energy. *Science*, 223:1164–1166, 1984.

[73] D. Sankoff and J. B. Kruskal. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley, Reading, MA, 1983.

[74] J. Satsuma. N-soliton solution of the two-dimensional Korteweg-de Vries equation. *Journal of the Physical Society of Japan*, 40(1):286–290, 1976.

[75] A. C. Scott. The Sine-Gordon equation. *AMS Lectures in Applied Mathematics*, 15:211–213, 1974.

[76] A. C. Scott, F. Y. F. Chu, and D. W. McLaughlin. The soliton: A new concept in applied science. *Proceedings of the IEEE*, 61(10):1443–1483, 1973.

[77] J. Scott-Russell. Report on waves. In *Proceedings of the Royal Society of Edinburgh*, pages 319–320, 1844.

[78] M. Segev, G. C. Valley, B. Crosignani, P. DiPorto, and A. Yariv. Steady-state spatial screening solitons in photorefractive materials with external applied field. *Physical Review Letters*, 73:3211, 1994.

[79] T. Shanley. *Pentium Pro Processor System Architecture*. MindShare, Inc. (Addison-Wesley), Reading, MA, 1997.

[80] S. S. Shen. *A Course on Nonlinear Waves*. Kluwer Academic Publishers, Norwell, MA, 1993.

[81] M. Shih and M. Segev. Incoherent collisions between two-dimensional bright steady-state photorefractive spatial screening solitons. *Optics Letters*, 21(19):1538, 1996.

[82] M. Shih, M. Segev, and G. Salamo. Three-dimensional spiraling of interacting spatial solitons. *Physical Review Letters*, 78:2551, 1997.

[83] M. Shih, M. Segev, G. C. Valley, G. Salamo, B. Crosignani, and P. DiPorto. Observation of two-dimensional steady-state photorefractive screening solitons. *Electronics Letters*, 31(10):826, 1995.

[84] P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 20–22. IEEE Press, 1994.

[85] D. Simon. On the power of quantum computation. In *35th Annual Symposium on Foundations of Computer Science*, pages 116–123. IEEE Press, 1994.

[86] A. W. Snyder, D. J. Mitchell, and Y. S. Kivshar. Unification of linear and nonlinear wave optics. *Modern Physics Letters B*, 9(23):1479–1506, 1995.

[87] A. W. Snyder and A. P. Sheppard. Collisions, steering, and guidance with spatial solitons. *Optics Letters*, 18(7):482–484, 1993.

[88] R. Squier and K. Steiglitz. 2-d FHP lattice gasses are computation universal. *Complex Systems*, 7:297–307, 1993.

[89] R. Squier and K. Steiglitz. Programmable parallel arithmetic in cellular automata using a particle model. *Complex Systems*, 8:311, 1994.

[90] R. K. Squier and K. Steiglitz. Subatomic particle machines: Parallel processing in bulk material. *Submitted to Signal Processing Letters.*

[91] V. V. Steblina, Y. S. Kivshar, and A. V. Buryak. Scattering and spiraling of solitons in a bulk quadratic medium. *Optics Letters*, 23:156, 1998.

[92] K. Steiglitz. Two non-standard paradigms for computation: Analog machines and cellular automata. In *Performance Limits in Communication Ttheory and Practice (NATO Advanced Study Institutes Series E)*, pages 173–192, Dordrecht, The Netherlands, 1988. Kluwer Academic Publishers.

[93] K. Steiglitz, I. Kamal, and A. Watson. Embedding computation in one-dimensional automata by phase coding solitons. *IEEE Transactions on Computers*, 37(2):138–145, 1988.

[94] T. R. Taha and M. J. Ablowitz. Analytical and numerical aspects of certain nonlinear evolution equations. *Journal of Computational Physics*, 55:192–202, 1984.

[95] S. Takeuchi. A simple quantum computer: experimental realization of the Deutsch-Jozsa algorithm with linear optics. In *Fourth Workshop in Physics and Computation: PhysComp '96*, pages 299–302, Cambridge, MA, Nov. 1996. New England Complex Systems Institute.

[96] J. R. Taylor (editor). *Optical Solitons: Theory and Experiment.* Cambridge University Press, Cambridge, UK, 1992.

[97] V. Tikhonenko, J. Christou, and B. Luther-Davies. Three-dimensional bright spatial soliton collision and fusion in a saturable nonlinear medium. *Physical Review Letters*, 76:2698, 1996.

[98] M. Toda. *Nonlinear Waves and Solitons*. Kluwer Academic Publishers, Boston, MA, 1989.

[99] T. Toffoli. CAM: A high-performance cellular-automaton machine. *Physica*, 10D:195–204, 1984.

[100] T. Toffoli. Fine-grained models and massively parallel architectures: The case for programmable matter. In *1995 SIAM Conference on Parallel Processing for Scientific Computing*, Feb. 1995.

[101] T. Toffoli, M. Biafore, and J. Leão (editors). *Fourth Workshop on Physics and Computation: PhysComp '96*. New England Complex Systems Institute, Cambridge, MA, 1996.

[102] T. Toffoli and N. Margolus. *Cellular Automata Machines*. MIT Press, Cambridge, MA, 1987.

[103] A. Vergis, K. Steiglitz, and B. Dickinson. The complexity of analog computation. *Mathematics and Computers in Simulation*, 28:91–113, 1986.

[104] J. von Neumann (edited by A. W. Burks). *Theory of Self-reproducing Automata*. Univ. of Illinois Press, 1966.

[105] M. J. Werner and S. R. Friberg. Phase transitions and the internal noise structure of nonlinear Schrödinger equation solitons. *Physical Review Letters*, 79:4143–4146, 1997.

[106] N. Weste and K. Eshraghian. *Principles of CMOS VLSI Design*. Addison-Wesley, Reading, MA, 1985.

[107] E. Winfree, X. Yang, and N. C. Seeman. Universal computation via self-assembly of DNA: Some theory and experiments. In *Second DIMACS Meeting on DNA Based Computers, Princeton, NJ*, 1996.

[108] S. Wolfram. Universality and complexity in cellular automata. *Physica*, 10D:1–35, 1984.

[109] A. Yariv and P. Yeh. *Nonlinear Waves in Crystals*. Wiley, New York, 1984.

[110] N. J. Zabusky and M. D. Kruskal. Interaction of solitons in a collisionless plasma and the recurrence of initial states. *Physical Review Letters*, 15(6):240–243, Aug. 1965.