Okinawa Institute of Science and Technology
Graduate University

Thesis submitted for the degree

Doctor of Philosophy

Massively parallel split-step Fourier techniques for simulating quantum systems on graphics processing units

by

James Schloss

Supervisor: **Thomas Busch**

July, 2019

Declaration of Original and Sole Authorship

I, James Schloss, declare that this thesis entitled *Massively parallel split-step Fourier techniques for simulating quantum systems on graphics processing units* and the data presented in it are original and my own work.

I confirm that:

- No part of this work has previously been submitted for a degree at this or any other university.
- References to the work of others have been clearly acknowledged. Quotations from the work of others have been clearly indicated, and attributed to them.
- In cases where others have contributed to part of this work, such contribution has been clearly acknowledged and distinguished from my own work.
- None of this work has been previously published elsewhere, with the exception of the following: (provide list of publications or presentations, or delete this part). (If the work of any co-authors appears in this thesis, authorization such as a release or signed waiver from all affected co-authors must be obtained prior to publishing the thesis. If so, attach copies of this authorization to your initial and final submitted versions, as a separate document for retention by the Graduate School, and indicate on this page that such authorization has been obtained).

Date: July, 2019
Signature:

Abstract

Massively parallel split-step Fourier techniques for simulating quantum systems on graphics processing units

The split-step Fourier method is a powerful technique for solving partial differential equations and simulating quantum systems of various forms. In this body of work, we focus on several variations of this method to allow for simulations of one, two, and three-dimensional quantum systems, along with several notable methods for controlling these systems. In particular, we use quantum optimal control and shortcuts to adiabaticity to study the non-adiabatic generation of superposition states in strongly correlated one-dimensional systems, analyze chaotic vortex trajectories in two dimensions by using rotation and phase imprinting methods, and create stable, three-dimensional vortex structures in Bose–Einstein condensates through artificial magnetic fields generated by the evanescent field of an optical nanofiber. We also discuss algorithmic optimizations for implementing the compressed split-step Fourier method for graphics processing units and multicomponent simulations. All variations present in this work are justified with physical systems where such techniques have been applied and have been incorporated into a state-of-the-art and open-source software suite known as GPUE, which is currently the fastest quantum simulator of its kind.

Acknowledgment

Theses must acknowledge assistance received in any of the following areas:

- Designing the research
- Executing the research
- Analyzing the data
- Interpreting the data/research
- Writing, proofing, or copyediting the manuscript

Abbreviations

All abbreviations used in the thesis should be listed here, with their definitions, in alphabetical order. This includes trivial and commonly used abbreviations (at your own discretion), but not words that have entered into general English usage (such as laser or DNA). In particular, non-standard abbreviations should be presented here. This is an aid to the reader who may not read all sections of the thesis.

BEC	Bose–Einstein Condensate
GPGPU	General Purpose computation on Graphics Processing Units
GPU	Graphics Processing Unit
NLSE	Non-linear Schrödinger Equation

Glossary

Dipole Blockade	Phenomenon in which the simultaneous excitation of two atoms is inhibited by their dipolar interaction.
Cavity Induced Transparency	Phenomenon in which a cavity containing two atoms excited with light at a frequency halfway between the atomic frequencies contains the number of photons an empty cavity would contain.

Nomenclature

c	Speed of light ($2.997\ 924\ 58 \times 10^{8}$ ms^{-1})
$\hbar$	Planck constant ($1.054\ 572\ 66 \times 10^{-34}$ Js)
k_B	Boltzmann constant ($1.380\ 658 \times 10^{-23}$ JK^{-1})
Z_0	Impedance of free space ($376.730\ 313\ 461\ \Omega$)
μ_0	Permeability of free-space ($4\pi \times 10^{-7}$ Hm^{-1})

If desired, an optional and short dedication may be included here.

Contents

Declaration of Original and Sole Authorship iii

Abstract v

Acknowledgment vii

Abbreviations ix

Glossary xi

Nomenclature xiii

Contents xvii

List of Tables xxi

Introduction 1

1 Introduction to the SSFM for vortex simulations 5
- 1.1 The SSFM . . . 6
- 1.2 Introduction to ultracold quantum systems . . . 10
  - 1.2.1 Bose–Einstein condensation and the Gross–Pitaevskii Equation 10
- 1.3 Superfluid systems and vortex dynamics . . . 12
  - 1.3.1 Rotation . . . 16
  - 1.3.2 Phase imprinting . . . 17
  - 1.3.3 Artificial magnetic fields . . . 17
- 1.4 Modifications to the SSFM for superfluid vortex simulations . . . 20
- 1.5 Multi-component simulations . . . 21

2 Quantum engineering for one-dimensional quantum systems 23
- 2.1 Optimization methods . . . 23
  - 2.1.1 Nelder–Mead . . . 24
  - 2.1.2 Chopped random basis . . . 26
- 2.2 Shortcuts to adiabaticity . . . 26
- 2.3 Non-adiabatic generation of NOON states in a Tonks–Girardeau gas . . . 28
  - 2.3.1 TG gas . . . 28

2.3.2 NOON states in TG gas . . . 29
2.3.3 Results with optimal control . . . 31
2.3.4 Results with STA protocols . . . 33
2.4 Outlook . . . 39

3 General Purpose computing with Graphics Processing Units and the GPUE codebase 41
3.1 Types of parallelism . . . 42
3.2 General purpose computing with graphics processing units . . . 43
3.2.1 Limitations of GPU computing . . . 43
3.2.2 GPU hardware architecture . . . 44
3.2.3 Comparison between various languages for GPGPU computation 50
3.3 Introduction to the GPUE codebase for n-dimensional simulations of quantum systems on the GPU . . . 52
3.3.1 FFT optimization . . . 52
3.3.2 Dynamic field input and output in GPUE with expression trees 53
3.3.3 Vortex tracking and highlighting . . . 54
3.3.4 Energy calculation for superfluid simulations . . . 57
3.3.5 Future direction and multi-GPU development . . . 58
3.3.6 Similar software packages . . . 59
3.4 DistributedTranspose.jl . . . 59

4 Vortex analysis of two-dimensional superfluid systems 61
4.1 Chaotic few-body vortex dynamics in rotating Bose–Einstein condensates 61
4.2 Model . . . 62
4.3 Regular and irregular vortex dynamics . . . 63
4.4 Characterizing chaotic vortex dynamics . . . 66
4.5 Outlook . . . 69

5 Generation, control and detection of 3D vortex structures in superfluid systems 71
5.1 Vortex ring dynamics in BEC systems . . . 71
5.2 Controlled creation of three-dimensional vortex structures in Bose–Einstein condensates using artificial magnetic fields . . . 74
5.2.1 Bose–Einstein condensate dynamics in the presence of an optical nanofiber . . . 74
5.2.2 Ground state vortex configurations . . . 79
5.2.3 Dynamic vortex detection and scissor modes . . . 82
5.3 Conclusion . . . 84

Conclusion 85
5.4 Overall conclusions . . . 85
5.5 Further development of GPUE . . . 85
5.5.1 Vortex tracking in n dimensions . . . 86
5.5.2 General purpose Hamiltonian solver . . . 86
5.6 Future simulations of quantum systems . . . 86

A Simple vector additions in CUDA, OpenCL, and JuliaGPU **87**
A.1 Vector addition with C++ 87
A.2 Vector addition with CUDA 88
A.3 Vector addition with OpenCL 89
A.4 Julia 92

Bibliography **93**

List of Tables

Introduction

[CN for everything]

Massively parallel methods have become commonplace in High-Performance Computing (HPC) environments, as they often rely on large networks of distributed computing nodes for performing simulations of various forms. In recent years, it has been found that the Graphics Processing Unit (GPU) can provide a higher bandwidth for highly parallelizable computation because all components are available in a single device that has been developed specifically for the computation of many small actions in parallel. As such, many supercomputers have been transitioning to GPU-based computation, including Summit, currently the fastest supercomputer in the world [1]. For these reasons, General-Purpose GPU (GPGPU) programming methods have become more relevant than ever and many frameworks are beginning to cater to the demand [2–5], with the current state-of-the art platform being NVIDIA's CUDA (Compute Unified Device Architecture) [6].

Though GPU devices are often faster than their CPU counterparts for simple tasks, there are plenty of drawbacks to using the GPU. For example, each GPU card typically has less available memory than the CPU, and inter-GPU or GPU-CPU communication is an incredibly slow process, thereby encouraging developers to restrict communication between devices as much as possible. In comparison to CPU software, developers need to be more aware of how GPU memory is being used to write optimized code for their specific purpose. In addition, individual GPU computing cores are weaker than those found on the CPU, so iterative tasks are even less optimal and should be avoided when programming for GPUs.

Spectral methods are an interesting subset of problems that are not often used for large-scale, distributed computation on HPC environments due to their reliance on FFTs (Fast Fourier Transforms), which are global operations, often requiring memory manipulation on multiple nodes simultaneously and requiring communication between them. Thought there are robust FFT libraries, like FFTW [7] to perform distributed FFTs, spectral methods are still avoided at scale and replaced by either finite-element or finite-difference counterparts. There is an interesting and open question about whether spectral methods could be faster on distributed networks of GPU devices, as the bulk of the parallelization occurs in a single device and should be faster than a distributed CPU network. Though special care needs to be taken to ensure that GPU memory is properly aligned, several pseudo-spectral methods have been shown to be both easier to implement and faster than other force integration schemes like Runge-Kutta, and one such method is the Split-Step Fourier Method (SSFM) [8].

The SSFM is known as the primary workhorse for computation of wavepackets in single and multi-mode fiber optic systems and is primarily intended to solve the Nonlinear Schrödinger equation, which has obvious applications to many areas of quantum simulation. In particular, the SSFM can be used to solve the Gross-Pitaevskii equation, which is the governing formula for all dynamics of superfluid Bose–Einstein condensates in the mean-field limit. Superfluid systems behave fundamentally differently than classical fluids and there is significant interest in many areas of superfluid research, including methods of vortex generation and their interaction. This work has developed a massively parallel, GPU-based library for the simulation of various quantum phe-

nomenon with the SSFM, with a focus on solving the Gross-Pitaevskii equation for the simulation of superfluid vortex dynamics. I will also discuss and motivate several methods for simulation of quantum engineering on GPU devices. This work was carried out during my time as a PhD student in the Quantum Systems Unit at the Okinawa Institute of Science and Technology Graduate University (OIST).

Introduction to the SSFM for vortex simulations

This work will begin with an introduction to the SSFM, itself, along with the physical target for most of this work: superfluid vortex simulations. As such, it will also discuss methods of vortex generation, including rotation, phase imprinting, and gauge fields, along with modifications to the Gross–Pitaevskii equation for simulating rotating and multi-component systems. This chapter lays the groundwork for all subsequent chapters.

Quantum engineering for one-dimensional quantum systems

This chapter will motivate several methods that are difficult to simulate on GPU architecture, focusing on methods of quantum engineering for a one-dimensional example where macroscopic superposition states are generated in a Tonks–Girardeau gas. In addition, this chapter will highlight methods in quantum optimal control and shortcuts to adiabaticity that will serve as examples of quantum engineering methods physicists might wish to implement for various quantum phenomena. This work has been published in the New Journal of Physics [9].

Introduction to GPGPU and the GPUE codebase

This chapter will introduce the concept of General-Purpose GPU computing and the GPUE codebase for superfluid vortex simulation. It will also cover GPU architecture in-depth and discuss several optimizations performed in GPUE to enable certain features which could not be done before on other GPU libraries with similar purposes. This chapter will conclude with a discussion on a notoriously difficult problem with GPU hardware that could make spectral and pseudo=spectral methods even more efficient on GPU hardware: an n-dimensional distributed transpose. The GPUE codebase has been published in the Journal of Open Source Software [10].

Vortex analysis of two-dimensional superfluid systems

This chapter will be related to an example of superfluid simulations with GPUE in two-dimensions, where vortices essentially follow the dynamics of point-vortex models. Here, the system is shown to exhibit chaotic dynamics with only a few vortices present. This system highlights the necessity of good post-processing methods for the simulations performed with GPUE, as the Lyapunov exponents are used on the tracked vortex positions to ascertain the degree of chaotic motion. This work has been published in Phys. Rev. Fluids [11].

Generation, control, and detection of 3D vortex structures in superfluid systems

This chapter is another example of superfluid simulations performed with GPUE, this time in three-dimensions. For this system, a novel device is proposed that can generate, control, and detect vortex ring-like structures by coupling the BEC to the light of an optical nanofiber. This system highlights the need for many of the features suggested during GPUE development for minimizing the memory footprint and ensuring fast, dynamic simulations. This work has been submitted to Phys. Rev. Fluids [12].

Outlook

Though there will be features developed for GPUE that are not used, specifically in physical examples used here, there is currently ongoing work to use these features at this time. In addition, it is worth discussing several features with GPUE that could use more development in the future.

Chapter 1

Introduction to the SSFM for vortex simulations

The Split-Step Fourier Method (SSFM) is an essential technique for simulating a variety of physical systems and is particularly useful for simulating the propagation of wave packets in single and multimode fibers [13–16] and in various quantum systems [17–19]. Though other methods, such as explicit and implicit Euler [20], Crank-Nicholson [21], and Runge-Kutta [20], can solve similar differential equations, the SSFM has distinct advantages over these methods. For example, the SSFM is often much easier to parallelize than Runge-Kutta [8], as it primarily relies on embarrassingly parallel element-wise matrix multiplications and Fast Fourier Transform (FFT) routines that have been optimized for parallel and distributed systems. The SSFM also provides a lower error bound than either the Euler or Crank-Nicholson methods, and does not require an implicit or tridiagonal solver [22, 23] which are also not easily parallelizable [24–26]. Though much of this work focuses on using the SSFM to simulate superfluid vortex states, I will not be discussing alternative methods, such as vortex-point or vortex-filament simulations in rigorous detail, as this work focuses primarily on engineering appropriate quantum states, while vortex-point and vortex filament methods focus primarily on vortex structures, themselves.

In the work presented here, I will be focusing on the application of the SSFM to superfluid vortex simulations and will use primarily physical arguments to understand the details of the method itself. I will also discuss several numerical techniques for optimally simulating quantum systems on massively parallel Graphics Processing Units (GPUs), along with software developed for this purpose: GPUE, the Graphics Processing Unit Gross-Pitaevskii Equation Solver. More details about General Purpose computing with Graphics Processing Units (GPGPU) and the GPUE simulation software can be found in Chapter 3. There, I will discuss several additional areas of interest for implementing similar solvers on GPUs, including distributed transposes and important considerations for traditional FFT routines for simulating quantum systems on multiple GPU devices.

This chapter will assume familiarity with basic principles of quantum mechanics and focus on specific performance considerations for simulating quantum systems with the SSFM. As such, I will also introduce important physical insights for understanding ultracold atomic systems. In particular, I will focus on understanding superfluid sys-

tems created by Bose–Einstein condensation and methods by which we can generate and control vortex dynamics in a Bose–Einstein Condensate (BEC).

1.1 The SSFM

Let us begin with the one-dimensional Schrödinger equation,

$$i\hbar\frac{\partial\Psi(r,t)}{\partial t} = \left(\frac{p^2}{2m} + V_0(\mathbf{r})\right)\Psi(\mathbf{r},t) \tag{1.1}$$

where $\hat{p} = -i\hbar\frac{\partial}{\partial x}$ is the canonical momentum operator, m is the mass, $V_0(\mathbf{r})$ is the trapping potential, and $\Psi(r,t)$ is the single-particle wavefunction. In this case, we often replace most of the right-hand side of the equation with a Hamiltonian operator, which for this case would be $\hat{\mathcal{H}} = \frac{p^2}{2m} + V_0(\mathbf{r})$. This noticeably has two components, one acting in position-space, $\hat{\mathcal{H}}_v = V_0(\mathbf{r})$ and another in momentum-space, $\hat{\mathcal{H}}_p = \frac{\hat{p}^2}{2m}$. For consistency, I will denote all variables in momentum-space with a p, and real-space with a v. Additionally, the wavefunction can be expanded into a complete set of eigenkets of the Hamiltonian, with $\hat{\mathcal{H}}\left|\Psi_n\right\rangle = E_n\left|\Psi_n\right\rangle$. One can then write

$$\left|\Psi(\mathbf{r})\right\rangle = \sum_{n=0}^{N} c_n e^{-iE_n t}\left|\psi_n(\mathbf{r})\right\rangle, \tag{1.2}$$

where N is the total number of states in the system, c_n is a constant for each constituent eigenfunction $\psi_n(\mathbf{r})$.

Simply stated, the SSFM splits the Hamiltonian into separate operators and uses a Fourier transform on the wavefunction to ensure that these operators are applied in the natural space. In order to apply the Hamiltonian to the system, we first use a formal solution to the Schrödinger equation,

$$\Psi(\mathbf{r},t+dt) = \left[e^{-\frac{i\hat{\mathcal{H}}dt}{\hbar}}\right]\Psi(\mathbf{r},t) = \left[e^{-\frac{i(\hat{\mathcal{H}}_v+\hat{\mathcal{H}}_p)dt}{\hbar}}\right]\Psi(\mathbf{r},t), \tag{1.3}$$

where dt is a small timestep. If we assume we are simulating our system by a series of small timesteps, we can split this operation by using the Baker-Campbell-Hausdorff formula,

$$\Psi(\mathbf{r},t+dt) = \left[e^{-\frac{i\hat{\mathcal{H}}_v dt}{\hbar}}e^{-\frac{i\hat{\mathcal{H}}_p dt}{\hbar}}e^{-\frac{[i\hat{H}_v, i\hat{H}_p]dt^2}{2}}\right]\Psi(\mathbf{r},t). \tag{1.4}$$

If neglected, the commutation of the real and momentum-space components of the Hamiltonian will accrue an error on the order of dt^2. This is a noticeably high; however, we can decrease the dt^2 error to dt^3 by performing a half-step in position space before doing a full-step in momentum space, through a process called Strang splitting [27],

$$\Psi(\mathbf{r},t+dt) = \left[e^{-\frac{i\hat{\mathcal{H}}_v dt}{2\hbar}}e^{-\frac{i\hat{\mathcal{H}}_p dt}{\hbar}}e^{-\frac{i\hat{\mathcal{H}}_v dt}{2\hbar}}\right]\Psi(\mathbf{r},t) + \mathcal{O}(dt^3). \tag{1.5}$$

Strang splitting can be best understood by performing a Taylor series expansion on both $e^{h(\mathbf{A}+\mathbf{B})}$ and $e^{h\mathbf{A}}e^{h\mathbf{B}}$, where $\mathbf{A}$ and $\mathbf{B}$ are matrices and h is a defined step size [28]. When expanded,

$$e^{h(A+B)} \approx \mathbf{I} + h(\mathbf{A}+\mathbf{B}) + \frac{1}{2}h^2(\mathbf{A}+\mathbf{B})^2. \tag{1.6}$$

where $\mathbf{I}$ is the identity matrix. To the second order terms, the expansion is identical to the expansion of $e^{h\mathbf{A}}e^{h\mathbf{B}}$; however, when expanding the last term, we find,

$$\frac{1}{2}h^2(\mathbf{A}+\mathbf{B})^2 = \frac{1}{2}\left(\mathbf{A}^2 + \mathbf{AB} + \mathbf{BA} + \mathbf{B}^2\right). \tag{1.7}$$

Here, the $\mathbf{BA}$ term with the expansion of $e^{h\mathbf{A}}e^{h\mathbf{B}}$ because $\mathbf{A}$ always comes before $\mathbf{B}$. If a symmetric splitting is used instead, it is clear that all the terms up to the second order are the same, such that $e^{h(\mathbf{A}+\mathbf{B})} \approx e^{h\mathbf{A}/2}e^{h\mathbf{B}}e^{h\mathbf{A}/2}$.

Because position and momentum are conjugate domains, after Strang splitting we address each part of this solution in chunks, first in position space, then in momentum space, then in position space again by using Fourier transforms. Which looks something like this:

$$\Psi(\mathbf{r}, t+dt) = \left[\hat{U}_r(dt)\mathcal{F}^{-1}\left[\hat{U}_p(dt)\mathcal{F}\left[\hat{U}_r(dt)\Psi(\mathbf{r},t)\right]\right]\right] + \mathcal{O}(dt^3) \tag{1.8}$$

where $\hat{U}_r = e^{-\frac{i\hat{\mathcal{H}}_v dt}{2\hbar}}$, $\hat{U}_p = e^{-\frac{i\hat{\mathcal{H}}_p dt}{\hbar}}$, and $\mathcal{F}$ and $\mathcal{F}^{-1}$ indicate forward and inverse Fourier transforms. In practice, these Fourier transforms are performed with Fast Fourier Transforms (FFTs), which typically use a variation on the Cooley-Tukey method, which was first discovered by Gauss and later contemporized by Cooley and Tukey when they independently discovered it [29]. This method is not straightforwardly parallelizable; however, FFTs have become so fundamental to signal processing, that they have been incredibly well-optimized with several libraries, including FFTW [7] and CuFFT [3] for distributed and GPU calculations, respectively. We will discuss optimal techniques for using FFTs with the SSFM method in Chapter 3.

Each timestep of the SSFM is essentially composed of the following steps:

1. Multiply the wavefunction in real space with the real-space operator by using a half-step in position space.
2. Flip to momentum space with an FFT operation on the wavefunction.
3. Multiply the momentum-space wavefuntion by the momentum-space operator.
4. Flip to position space with an inverse FFT on the wavefunction.
5. Repeat 1-4 until satisfied.

With the method described so far, we can simulate simple quantum systems. For example, if we use a Gaussian wavefunction which is offset from the center of a simple harmonic oscillator, we can simulate the wavefunction oscillation, as shown in Figure 1.1(a).

In addition to this, we can find the lowest energy state of our system with the SSFM by performing a Wick rotation and using $\tau = it$ for the simulation instead of

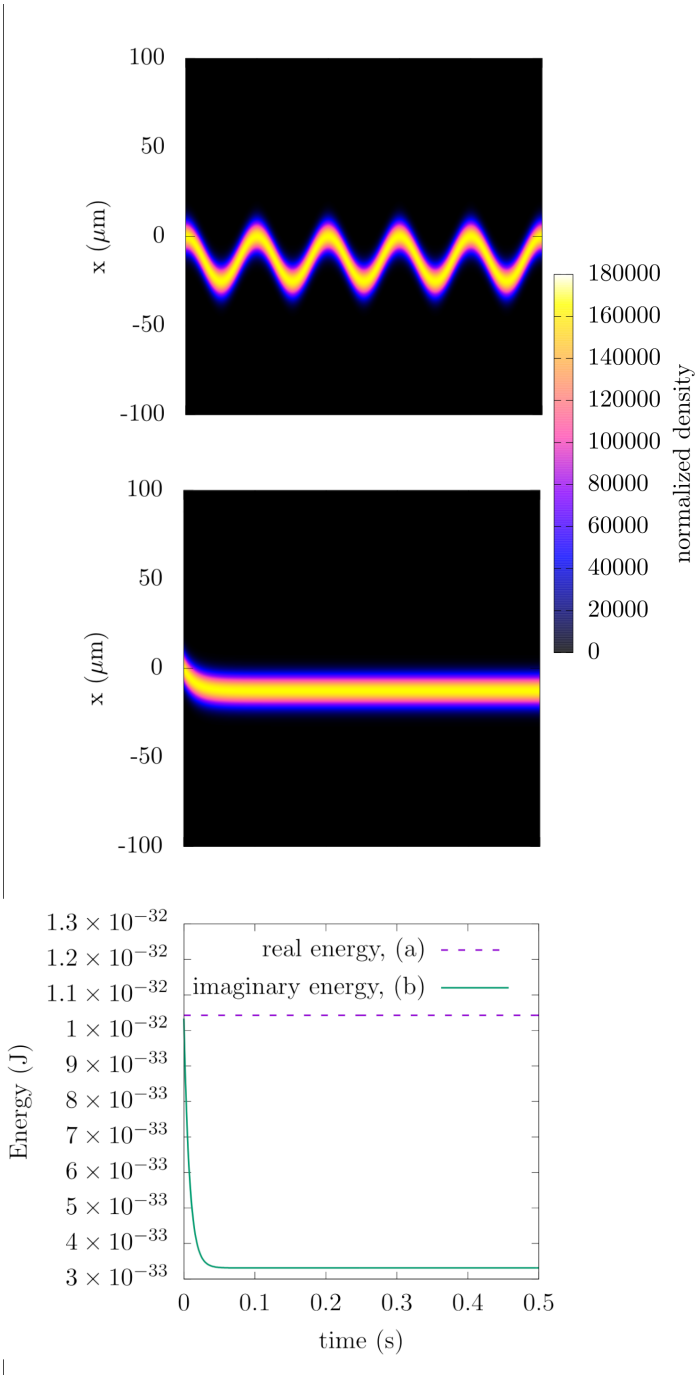

Figure 1.1: Evolution of a simple harmonic oscillator in (a) real, and (b) imaginary time after slightly shifting the trapping potential in the $\hat{x}$ direction. In (c), we see the energy as a function of time and show that the energy of the system when evolving in real time remains constant, but in imaginary time it will decay to the known ground-state energy of the simple harmonic oscillator. The simulated results are from evolution with the SSFM after 10,000 steps. Here, we use a $^{8}7$ Rb atom with $\omega_x = 10$Hz on a 256-point grid of size 200 μm, where the trap has been shifted by 5 μm. The wavefunction has been normalized such that $\int_{-\infty}^{\infty} |\Psi|^2 dx = 1$. This simulation was performed with the GPUE codebase [10]. [WIP]

traditional units of time [30]. This changes the solution from the complex sinusoid shown in Equation (1.4) to an exponential decay,

$$\Psi(\mathbf{r}, \tau + d\tau) = \left[e^{-\frac{\hat{\mathcal{H}} d\tau}{\hbar}}\right] \Psi(\mathbf{r}, \tau) = \sum_{n=0}^{N} \left[e^{-\frac{(E_n d\tau)}{\hbar}}\right] \Psi_n(\mathbf{r}, \tau). \tag{1.9}$$

This has two notable effects:

1. There will be an exponential decay of all higher-energy states, leaving only the ground state.

2. We see an exponential decay in the wavefunction density and therefore require renormalization regularly during the imaginary-time evolution.

Let me start with a discussion on the first point, by showing a simulation of the same system as in Figure 1.1(a), but in imaginary time, shown in Figure 1.1(b). Here, we see that the wavefunction density is shifting to the center of the trap, and in Figure 1.1(c), we see the energy decaying to the known ground-state energy of a quantum harmonic oscillator of $\frac{1}{2}\hbar\omega = 3.31 \times 10^{-33}$. Here, the energy is computed as,

$$E = \langle \Psi(\mathbf{r}) | \hat{H} | \Psi(\mathbf{r}) \rangle \tag{1.10}$$

This is because all eigenstates of the wavefunction are affected by the exponential decay, with the ground state decaying the slowest, and because renormalization occurs every timestep, only the ground state will survive imaginary time propagation.

For many systems, we can assume the simulation is in the ground state when the energy converges to a fixed value. More quantitatively, this means that the simulation can be completed when the change in energy every step in imaginary time is below some provided threshold; however, this is not always the best course-of-action. Further discussion on energy calculations performed in this work will be shown in Chapter 3.

Now to discuss the second point, which is more problematic from a software perspective. In practice, every step in imaginary time requires a relatively costly renormalization step with

$$\int_{\infty}^{\infty} |\Psi(\mathbf{r}, t)|^2 d\mathbf{r} = 1, \tag{1.11}$$

for a single-particle wavefunction. Computationally, this operation requires a summation, which is not well-optimized for GPU hardware; however, it is possible to perform a parallel reduction (summation), which allows for a considerable improvement on massively parallel systems [31]. Even so, the normalization is still a slow operation and should be used sparingly.

The implementation of the SSFM provided here assumes large-scale element-wise matrix multiplications in position and momentum-space; however, even though this implementation lends itself to parallelization, we will see other methods in Chapter 3 when we discuss the implementation of the SSFM on graphics processing units. Now we will turn our focus to systems to be simulated via the SSFM throughout this work: ultracold atoms.

1.2 Introduction to ultracold quantum systems

When atomic systems are cooled to temperatures near zero Kelvin, it becomes easier to discern their quantum properties which vary drastically depending on whether the particles are bosonic or fermionic. Because fermions have half-integer spin, they must obey the Pauli exclusion principle and are constrained to Fermi–Dirac statistics at zero temperature. This creates a *Fermi sea*, where the particles fill defined energy levels from the bottom-up with two particles of opposite spin per level. On the other hand, bosons have integer spin and follow Bose–Einstein statistics. They will condense into a single, macroscopic ground state when cooled [32, 33], and this state of matter is known as a Bose–Einstein Condensate (BEC). The BEC has the properties of a superfluid, which will be discussed more completely in the following section.

There are notable exceptions to these rules, such as the highly correlated Tonks–Girardeau gas where bosons may act as spinless, non-interacting fermions [9, 34]. Though there also exists a regime where interacting fermions can condense into a BEC-like system by forming molecules with integer spin [35, 36], we will not discuss fermionic systems further in this work. For now, we will focus on BEC systems, but will also discuss Tonks–Girardeau gas systems later in Chapter 2.

1.2.1 Bose–Einstein condensation and the Gross–Pitaevskii Equation

For this section, we will follow a straightforward derivation using the second quantization [37]. As mentioned in Section 1.2, bosons in a BEC condense into a single ground state, meaning we must introduce a many-body Hamiltonian for the system and take inter-particle interactions into account. Because experimental syustems are dilute, we will only consider two-body interactions and assume any interactions between three or more atoms to be unlikely and negligible. We can write the Hamiltonian with two body interactions in the second quantized form as

$$\hat{\mathcal{H}} = \int \hat{\Psi}^\dagger(\mathbf{r}) \left[-\frac{\hbar^2}{2m}\nabla^2 + V_0(\mathbf{r}) \right] \hat{\Psi}(\mathbf{r}) d\mathbf{r} + \frac{1}{2} \int \hat{\Psi}^\dagger(\mathbf{r}) \hat{\Psi}^\dagger(\mathbf{r}') V(\mathbf{r}-\mathbf{r}') \hat{\Psi}(\mathbf{r}') \hat{\Psi}(\mathbf{r}) d\mathbf{r} d\mathbf{r}' \tag{1.12}$$

where $\mathbf{r}$ and $\mathbf{r}'$ are the positions of the two colliding particles, $V(\mathbf{r}-\mathbf{r}')$ is the interaction potential, and $\hat{\Psi}^\dagger(\mathbf{r})$ and $\hat{\Psi}(\mathbf{r})$ are the creation and annihilation operators that follow the bosonic commutation relations,

$$[\hat{\Psi}(\mathbf{r}), \hat{\Psi}^\dagger(\mathbf{r})] = \delta(\mathbf{r}-\mathbf{r}') \tag{1.13}$$

$$[\hat{\Psi}^\dagger(\mathbf{r}), \hat{\Psi}^\dagger(\mathbf{r})] = 0 \tag{1.14}$$

$$[\hat{\Psi}(\mathbf{r}), \hat{\Psi}(\mathbf{r})] = 0. \tag{1.15}$$

In the case of a BEC at $T \approx 0$, we may perform a Bogoliubov expansion [38, 39]

$$\hat{\Psi}(\mathbf{r}, t) = \Phi(\mathbf{r}, t) + \delta\hat{\Phi}(\mathbf{r}, t), \tag{1.16}$$

where $\Phi(\mathbf{r}, t) \equiv \langle \hat{\Psi}(\mathbf{r}, t) \rangle$ is the wavefunction of the condensate known as the order parameter and $\delta\hat{\Phi}(\mathbf{r}, t)$ represents fluctuations of the BEC system. For Equation (**??**)

I have used the Heisenberg representation for field operators [39]. In a BEC, the condensate density is defined as

$$n(\mathbf{r},t) = |\Phi(\mathbf{r},t)|^2. \tag{1.17}$$

Now we may use the Heisenberg equation of motion,

$$i\hbar\frac{\partial}{\partial t}\hat{\Psi}(\mathbf{r},t) = [\hat{\Psi},\hat{H}], \tag{1.18}$$

to determine the time evolution of the field operator $\hat{\Psi}(\mathbf{r},t)$ as

$$\frac{\partial}{\partial t}\hat{\Psi}(\mathbf{r},t) = \frac{1}{i\hbar}\left[-\frac{\hbar^2}{2m}\nabla^2 + V_0(\mathbf{r}) + \int d\mathbf{r}'\hat{\Psi}^\dagger(\mathbf{r}',t)V(\mathbf{r}'-\mathbf{r})\hat{\Psi}(\mathbf{r}',t)\right]\hat{\Psi}(\mathbf{r},t), \tag{1.19}$$

which follows from Equation (1.12) after integrating over $\mathbf{r}$. If we assume that two bosons will only interact with a contact potential of the form

$$V(\mathbf{r}'-\mathbf{r}) = g\delta(\mathbf{r}'-\mathbf{r}), \tag{1.20}$$

which has a strength given by

$$g \equiv \frac{4\pi\hbar^2 a_s}{m}, \tag{1.21}$$

where a_s is the species and state-dependent s-wave scattering length, we may write the time-dependent Schrödinger equation as

$$i\hbar\frac{\partial}{\partial t}\Phi(\mathbf{r},t) = \left(-\frac{\hbar^2}{2m}\nabla^2 + V_0(\mathbf{r}) + g|\Phi(\mathbf{r},t)|^2\right)\Phi(\mathbf{r},t). \tag{1.22}$$

This is the Gross–Pitaevskii Equation (GPE), which is known as a governing equation for the physics of BEC systems. When written in the time-independent form it determines the chemical potential μ of the condensate system [40, 41]

$$\mu\Phi(\mathbf{r}) = \left(-\frac{\hbar^2}{2m}\nabla^2 + V_0(\mathbf{r}) + g|\Phi(\mathbf{r})|^2\right)\Phi(\mathbf{r}). \tag{1.23}$$

This equation allows us to determine the full dynamics of a BEC system and the numerical solutions will be discussed in subsequent chapters. Similar derivations of the GPE can be found in many introductory texts on BEC physics [33, 42, 43]. The main difference between this equation and the Schrödinger equation is the non-linear interaction term of $g|\Psi|^2$, that accounts for the fact that BECs typically consist of 10^3 to 10^6 particles.. When solving this system in a simple harmonic oscillator in the limit where interactions are significant, we will find a slightly different distribution for the wavefunction density, known as a Thomas–Fermi distribution, shown in Figure 1.2 for a single dimension.

The Thomas–Fermi distribution in this case can be derived from the time-independent GPE in the limit where there are a large number of bosons in the condensate ($N \gg 1$) as [44]

$$\Psi_{\mathrm{TF}}(\mathbf{r}) = \sqrt{\frac{[\mu - V(\mathbf{r})\Theta(\mu - V(\mathbf{r})}{g}} \tag{1.24}$$

where Θ is the Heaviside step function that ensures the condensate stays positive and $\Psi(\mathbf{r})_{\mathrm{TF}}$ is the wavefunction in the Thomas–Fermi limit. The shape of this function is similar to that of an inverted parabola for a harmonic trap, and the radius in any direction from the center can be found to be

$$R_{\mathrm{TF}} = \sqrt{\frac{2\mu}{m\omega_i^2}} \tag{1.25}$$

where R_{TF} is the Thomas–Fermi radius and ω_i is the trapping frequency in the ith direction. In the case of multiple particles, the normalization condition becomes $\int_{-\infty}^{\infty} |\Psi(\mathbf{r},t)|^2 = N$, where N is the total number of particles. By using this normalization condition, we find that the Thomas–Fermi chemical potential is

$$\mu_{\mathrm{TF}} = \frac{h\omega_i}{2}\left(\frac{15Na_s}{a}\right)^{2/5}, \tag{1.26}$$

where $a = \sqrt{\hbar/m\bar{\omega}}$ and $\bar{o}mega$ is the average of all the harmonic trapping frequencies. Using Equations (1.26) and (1.25) we find that

$$R_{\mathrm{TF}} = a\left(\frac{15Na_s}{a}\right)^{1/5}\frac{\bar{\omega}}{\omega_i} \tag{1.27}$$

This approximation is valid for stationary condensate solutions in simple harmonic oscillator trapping geometries.

It is important to note that a BEC acts like a superfluid, which is a state of matter that is similar to a classical fluid without viscosity. This means that once a superfluid is set in motion, there is no retarding force to keep it from flowing. There are a few known systems in which superfluidity can exist, such as ^{4}He (sometimes called Helium II when in its superfluid phase) [45], neutron stars [46], or BEC systems [32, 47]. BEC systems are generally cleaner experimental systems to create, as they do not have a classical fluid fraction, like ^{4}He. As stated, we will focus on BEC systems in this work, but it is important to note that any results shown in this body of work for BEC systems may have applications beyond cold, atomic physics.

1.3 Superfluid systems and vortex dynamics

Though there are many interesting differences between classical fluids and superfluids, such as sound wave propagation and turbulent dynamics, we will focus here on vortex dynamics before continuing to discuss three methods of vortex generation in superfluid systems: rotation, phase imprinting and artificial magnetic fields. By rotating a fluid, it is possible to create a vortex around the axis of rotation; however, because of the viscosity of a classical fluid, the vortex will eventually disappear without constant driving. In a superfluid, this is not necessarily the case. To properly describe the differences in fluid and superfluid dynamics, it is worthwhile to discuss the hydrodynamic description of a BEC, following the text of Pethick and Smith [42], To start, I will treat the condensate wavefunction as

$$\Psi(\mathbf{r},t) = \sqrt{\rho(\mathbf{r},t)}e^{i\phi(\mathbf{r},t)}, \tag{1.28}$$

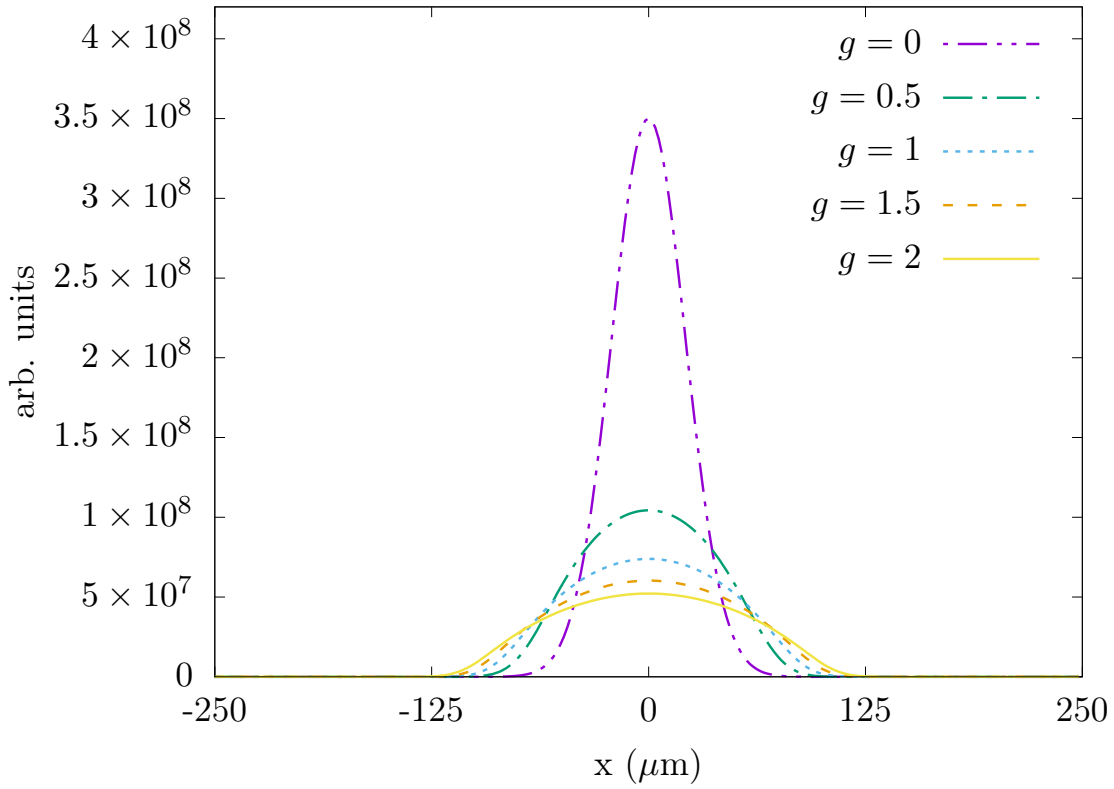

Figure 1.2: Ground state of simple harmonic oscillator for the Schrödinger equation (purple, dashed) and the GPE with varying interaction strengths (all other lines). Here, we see that the GPE solution follows a Thomas-Fermi distribution, while the Schrödinger equation is Gaussian. This simulation was performed with GPUE [10] for a two-dimensional grid of 256^2 elements of size $250\mu m$ with a trapping potential of $\omega_x = \omega_y = 1$, and 5×10^4 particles.

with

$$\rho(\mathbf{r},t) = \Psi(\mathbf{r},t)^*\Psi(\mathbf{r},t) = |\Psi(\mathbf{r},t)|^2. \tag{1.29}$$

where $\phi(\mathbf{r},t)$ is the BEC phase. By multiplying the GPE by $\Psi^*(\mathbf{r},t)$ and subtracting out the complex conjugate of the result, one can obtain the continuity equation [42],

$$\frac{\partial}{\partial t}\rho(\mathbf{r},t) + \nabla\cdot\mathbf{J}(\mathbf{r},t) = 0, \tag{1.30}$$

where $\mathbf{j}$ is the current density of the condensate, defined as,

$$\mathbf{j}(\mathbf{r},t) = \frac{-i\hbar}{2m}\left(\Psi^*(\mathbf{r},t)\nabla\Psi(\mathbf{r},t) - \Psi(\mathbf{r},t)\nabla\Psi^*(\mathbf{r},t)\right). \tag{1.31}$$

By substituting Equation (1.28) into Equation (1.31), we find the form of the current density for the GPE to be,

$$\mathbf{j}(\mathbf{r},t) = |\Psi(\mathbf{r},t)|^2\frac{\hbar}{m}\nabla\phi(\mathbf{r},t). \tag{1.32}$$

The velocity of the superfluid is defined as a ratio of the current density to the density, itself, which is

$$\mathbf{v}(\mathbf{r},t) = \frac{\mathbf{j}(\mathbf{r},t)}{\rho(\mathbf{r},t)} = \frac{\hbar}{m}\nabla\phi(\mathbf{r},t). \tag{1.33}$$

this can be interpreted to mean that the gradient of the phase determines the velocity of atoms in the BEC, indicating that the system is irrotational ($\nabla\times\mathbf{v} = 0$). Because the condensate is single-valued,

$$\oint \mathbf{v}\cdot d\ell = \frac{\hbar}{m}2\pi\ell. \tag{1.34}$$

Here, ℓ is an integer charge of circulation, and Equation (1.34) shows the quantized nature of circulation in a superfluid with each vortex hosting multiples of 2π charge. Every singly-charged vortex in a BEC will have a 2π phase winding, and an example simulation with one vortex and its corresponding phase can be seen in Figure 1.3 (a and c). Equation (1.34) also indicates that the phase is not defined at the center of the vortes; however, at this point, the density has also dropped to zero. The density dip from the normal condensate density to zero happens over the scale of the healing length. For repulsive interactions, the healing length is

$$\xi = \frac{1}{\sqrt{8\pi\rho_b a_s}} \tag{1.35}$$

where ρ_b is the bulk density of the condensate.

In a cyllindrically symmetric condensate with a single vortex at its center, the wavefunction can be written as

$$\Psi(\mathbf{r}) = |\Psi(\mathbf{r})|e^{i\ell\phi} \tag{1.36}$$

And when substituting this in to Equation (1.10) for the GPE, we find

$$E = \int_{-\infty}^{\infty}\frac{\hbar^2}{2m}\left(|\nabla\Psi(\mathbf{r})|^2 + \frac{|\Psi(\mathbf{r})|^2\ell^2 m\mathbf{v}^2}{2}\right) + V_0(\mathbf{r})|\Psi(\mathbf{r})|^2 + \frac{g}{2}|\Psi(\mathbf{r})|^4. \tag{1.37}$$

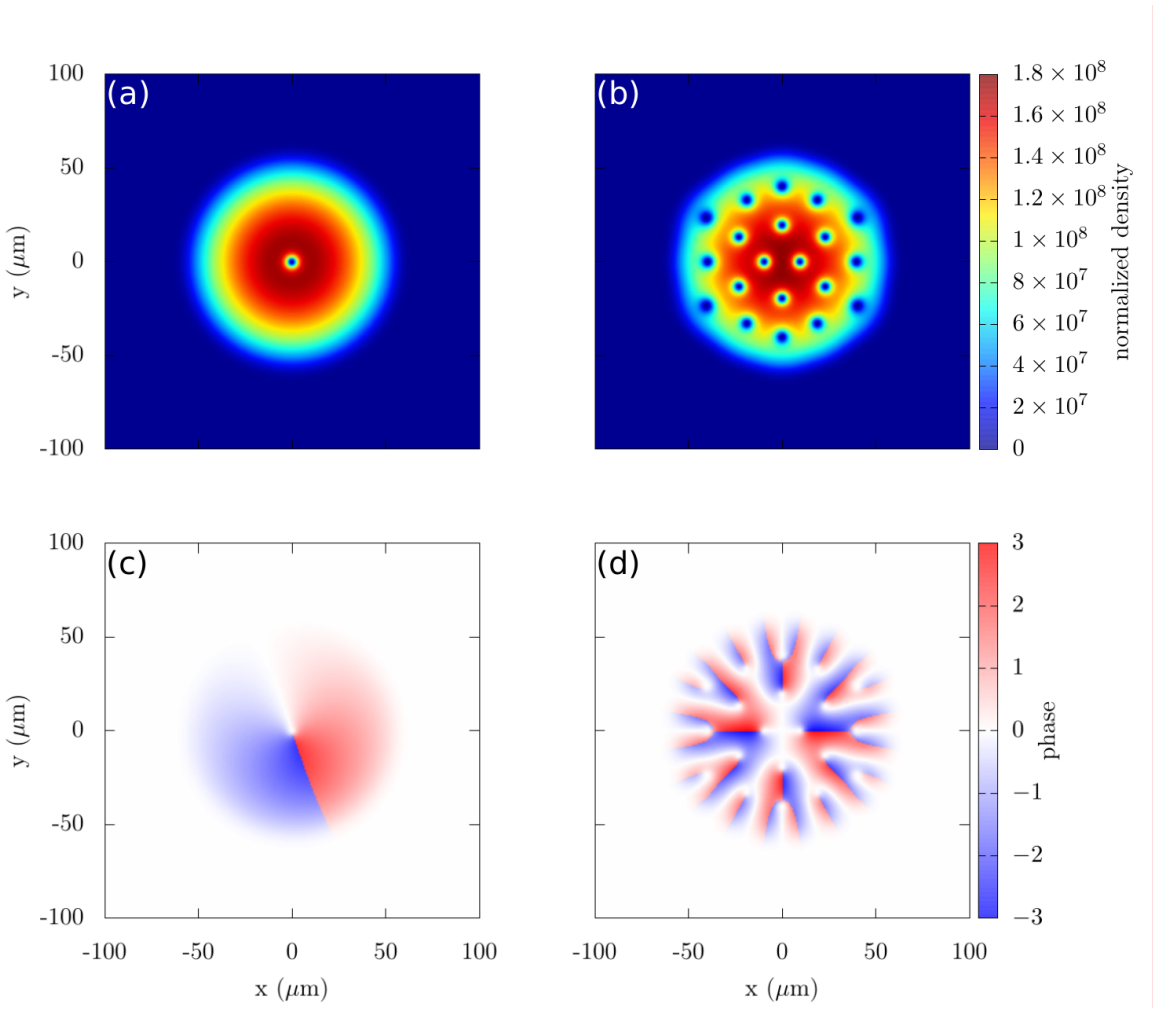

Figure 1.3: Simulation of rotation with a single vortex (a, b) and a vortex lattice (c, d). The wavefunction density is shown in a and c, while the corresponding phase is shown in b and d. A rotation of $\Omega = 0.35\omega_x$ was used for a and b, while a rotation of $\Omega = 0.99\omega_x$ was used for c and d. Here, we use a $^{8}7$ Rb atom with $\omega_x = 2\pi$Hz on a 512-point grid of size 200 μm. This simulation was performed with the GPUE codebase [10], and the phase plots are created by multiplying the phase by the wavefunction density to remove anomalous noise beyond the BEC boundary. [WIP]

Because $E \propto \ell^2$[CN], as a superfluid is spun faster beyond a critical angular velocity, a vortex will not grow or shrink in angular momentum, but multiple vortices will spawn instead [42]. In other words, it is energetically favorable for two vortices of smaller angular momentum to form instead of a single vortex with a large amount of angular momentum. As angular momentum increases and more vortices are introduced into the system, they will eventually arrange themselves in a triangular lattice structure known as an Abrikosov lattice [48, 49]. This behavior is identical to that of type II superconductors under the effects of a magnetic field, and the size of each vortex is described by a physical scale known as the healing length. An example of a vortex lattice and its phase can be seen in Figure 1.3 (b and d).

The three-dimensional properties of vortices in superfluid systems are also peculiar when compared to their classical counterparts. Here, vortex lines are formed that must either end at the end of the condensate [50] or reconnect in the form of vortex rings or other, more complicated vortex structures [51, 52]. Because the circulation around superfluid vortices is quantized, when two vortices approach each other with different velocity fields, they may reconnect into smaller, more energetically favorable vortex structures. During this reconnection, the abrupt change in energy will create a sound wave at the reconnection site [53].

Three dimensional vortex structures of non-trivial topology have been notoriously difficult to controllably generate experimentally, but we will discuss an experimentally viable method to generate, control, and detect vortex ring-like structures in superfluid BEC systems in Chapter 5. In that chapter, I will also further discuss three-dimensional vortex motion. For now, we will begin discussing the three primary processes to generate vortex structures in superfluid systems: rotation, phase imprinting, and artificial magnetic fields.

1.3.1 Rotation

Rotation of a BEC system will provide vortex lines that follow the axis of rotation and start and end on the BEC boundary. To simulate the effects of rotation, we simply need to append the angular momentum operator $L_z = -i\hbar(xp_y - yp_x)$ to the GPE,

$$i\hbar\frac{\partial\Psi(\mathbf{r},t)}{\partial t} = \left(\frac{p^2}{2m} + V_0 + g|\Psi(\mathbf{r},t)|^2 - \Omega L_z\right)\Psi(\mathbf{r},t), \tag{1.38}$$

where Ω is the rotation frequency.

In order to generate a vortex via rotation in a harmonic trap, we must rotate faster than the critical velocity of $\Omega_c \approx 0.3\omega_\perp$, where $\omega_\perp$ is the trapping frequency [CN]. In addition, if the rotation frequency is greater than the trapping frequency, the atoms will no longer be bound by the trap due to centripetal forces. As such, finding the appropriate rotation frequency for creating vortex lattices in BEC systems is a precarious balancing act. Even so, large scale vortex lattices have been generated both experimentally and theoretically, with the largest vortex lattices being generated with $\Omega \approx \omega_\perp$ [54–57].

Experimentally, rotation for a small number of vortices can be generated in a number of ways, such as a "rotating bucket" approach that has been extended to bosonic systems [58]. For this method, bosons are confined to a magnetic trap and an anisotropic

potential is superimposed that rotate with the desired angular velocity [50, 56, 59, 60]. For rotation velocities near the harmonic trapping frequency, additional methods must be used to ensure the atoms remain in-place. These methods include adding an extra confining potential [61], or the evaporative spin-up technique, where atoms with less angular momentum are evaporated such that the remaining atoms have a higher rotation speed [57].

In this work, we use rotation in a qualitatively similar way to Equation (1.38); however, we will later introduce artificial magnetic fields, which is a broader framework that encompass rotation and will be used in all our simulations moving forward. We will discuss this in more detail in Section 1.4.

1.3.2 Phase imprinting

Phase imprinting is a powerful tool to allow for the generation of various structures in atomic systems, including vortices [62–66]. To generate vortices in a BEC, this technique relies on imprinting a 2π phase winding onto a ground state condensate wavefunction. Experimentally, phase imprinting can be done in a number of ways. As an example, the phase could be imprinted with a Raman two-photon process to transfer orbital angular momentum to atoms from a Laguerre–Gaussian beam [63, 67]. Another method is through pulsing a spatially dependent potential for a short time when compared to the trapping frequency, which will imprint the potential energy onto the phase of the system [68] and can be used to generate solitons [65], vortex [69], or other states with quantized circulation [62]. Phase imprinting has also been used in theoretical studies to create a defect in a large vortex lattice by flipping the phase (and therefore rotation direction) of a selected vortex by imprinting a -4π phase at the vortex's location [55]. Note that if a phase greater than $|2\pi|$ is imprinted onto the system, the vortices are likely to decay into multiple vortices of $|2\pi|$ phase [70].

For the purposes of this work, we will only consider imprinting vortex lines along the transverse plane by apply phase imprinting operations to our condensate, such that

$$\Psi_{\mathrm{IMP}}(x, y, t) = |\Psi(x, y, t)| e^{i(\theta(x,y,t)+\theta_{\mathrm{IMP}}(x,y))}, \tag{1.39}$$

where $\Psi_{\mathrm{IMP}}(x, y, t)$ is the condensate wavefunction after phase imprinting. This method allows us to apply a phase mask to any location in the transverse plane. This technique will be applied further in Chapter 4.

Phase imprinting has allowed for the generation of many interesting vortex topologies in theoretical and experimental studies [71, 72]; however, it is a dynamical process that does not create eigenstates of the system. As such, it is not as useful for engineering stable vortex structures, but is instead useful for dynamical studies, such as those found in Chapter 4.

1.3.3 Artificial magnetic fields

Magnetic fields are capable of generating rotational effects in charged systems through the Lorentz force, $F_l = q(\mathbf{E} + \mathbf{v} \times \mathbf{B})$, where q is the charge of the system, $\mathbf{E}$ is the electric field, $\mathbf{B}$ is the magnetic field, and $\mathbf{v}$ is the velocity of the particle. This effect

allows for the creation of vortices in type II superconductors; however, it is not directly applicable to BEC systems, because BECs are composed of neutral atoms. Even so, it is possible to generate artificial magnetic fields with similar effects, and these artificial magnetic fields have been shown to create vortices experimentally [73]. In addition, artificial magnetic fields create a broader framework that encompasses rotational effects previously shown in Section 1.3.1, and we will use this framework instead of rotation for remainder of this work. As such, a detailed introduction, similar to that given by Dalibard in [74], will be presented below.

If written in the Hamiltonian formalism, the Lorentz force law becomes

$$\hat{\mathcal{H}} = \frac{(\hat{\mathbf{p}} - q\mathbf{A}(\mathbf{r}))^2}{2m} \tag{1.40}$$

where $\mathbf{A}$ is a vector potential such that $\mathbf{B} = \nabla \times \mathbf{A}$ and q is the charge of the particle. Because cold atoms are neutral, we must find ways to simulate the effects of magnetic fields instead of using magnetic fields, themselves.

Firstly, let us describe how rotation can be considered to be an artificial vector potential and thus generate vortex structures in BEC systems. Imagine a plane rotating with an angular velocity Ω around the z-axis ($\mathbf{\Omega} = \Omega\hat{z}$). In this case, the Coriolis force is defined as

$$\mathbf{F}_{\text{Coriolis}} = 2m\mathbf{v} \times \mathbf{\Omega}, \tag{1.41}$$

which is formally similar to the Lorentz force law. By applying the transformation $\hat{\mathcal{H}} = \hat{H}_0 - \Omega\hat{L}_z$, where $\hat{L}_z = x\partial_y - y\partial_x$, we find [75]

$$\begin{aligned}
\hat{\mathcal{H}} &= -\frac{\hbar^2}{2m}\nabla^2 + \frac{1}{2}m\omega^2(x^2+y^2) - \frac{\hbar\Omega}{i}(x\partial_y - y\partial_x) \\
&= \frac{1}{2m}\left(\frac{\hbar}{i}\nabla - m(\mathbf{\Omega}\times\mathbf{r})\right)^2 + \frac{m}{2}\left(\omega^2 - \Omega^2\right)r^2 \\
&= \frac{(\hat{\mathbf{p}} - m\mathbf{A}(\mathbf{r}))^2}{2m} + V_0(\mathbf{r}),
\end{aligned} \tag{1.42}$$

where ω is the trapping frequency for a symmetric two-dimensional harmonic trap, $\mathbf{A} \equiv \mathbf{\Omega}\times\mathbf{r}$, and $V_0 = m/2\left(\omega^2 - \Omega^2\right)r^2$. The final form is similar to that of the Lorentz force law and coincides with an effective magnetic field of $2\mathbf{\Omega} \propto \mathbf{B}$. In this way, we may recreate the rotation expected from the Lorentz force law in a cold atomic system with an artificial magnetic field [50, 56, 76].

Artificial magnetic fields provide a powerful tool to researchers who wish to generate and control complex vortex structures, and because of this, it is worth discussing them in further detail. Important implementation details for how to use artificial magnetic fields with the SSFM will be discussed in Section 1.4.

Geometric Gauge Fields

As we have already described how rotation can act as an artificial Lorentz force, we now turn our attention towards methods that might allow us to generate more general rotational effects and vortex structures. In particular, we will discuss the adiabatic

motion of free atoms undergoing geometric phase transformations through the Berry phase. For this, we assume that our system has an external parameter λ such that

$$\hat{H}(\lambda)\left|\psi_n(\lambda)\right\rangle = E_n(\lambda)\left|\psi_n(\lambda)\right\rangle, \tag{1.43}$$

where the set of eigenstates $\{|\psi_n(\lambda)\rangle\}$ allows us to define the time evolution of our system such that

$$\left|\psi(t)\right\rangle = \sum_n c_n(t)\left|\psi_n(\lambda(t))\right\rangle, \tag{1.44}$$

and we consider λ to evolve slowly with time. If we consider the system to begin with

$$c_l(0) = 1, \qquad c_n(0) = 0, \qquad \text{for all } n \neq l \tag{1.45}$$

the state of the system is proportional to $|\psi_l(\lambda(t))\rangle$. In this case, $c_l(t)$ is determined by the equation

$$i\hbar\dot{c}_l = [E_l(t) - \dot{\lambda}\cdot\mathbf{A}_l(\lambda)]c_l, \tag{1.46}$$

where

$$\mathbf{A}_l(\lambda) = i\hbar\left\langle\psi_l|\nabla\psi_l\right\rangle. \tag{1.47}$$

This quantity is called the Berry connection, which is considered a vector potential, such that we can define a new artificial magnetic field, the Berry curvature as

$$\mathbf{B}_l = \nabla\times\mathbf{A}_l. \tag{1.48}$$

Now imagine that the λ parameter follows the closed contour C such that $\lambda(T) = \lambda(0)$. By integrating Equation (1.46) above, we find

$$c_l(t) = e^{i\Phi_{\mathrm{dyn}}(t)}e^{i\Phi_{\mathrm{B}}(T)}c_l(0), \tag{1.49}$$

where

$$\begin{aligned}\Phi_{\mathrm{dyn}}(T) &= -\frac{1}{\hbar}\int_0^T E_l(t)dt \\ \Phi_{\mathrm{Berry}}(T) &= \frac{1}{\hbar}\int_0^T \dot{\lambda}\cdot\mathbf{A}_l(\lambda)dt = \frac{1}{\hbar}\oint\mathbf{A}_l(\lambda)\cdot d\lambda\end{aligned} \tag{1.50}$$

In this case Φ_{Berry} is called the Berry phase and it only depends on the motion path of λ. It should be mentioned that both of the exponential terms in Equation (1.49) are gauge invariant and thus remain unchanged when $|\psi_n(\lambda)\rangle$ is multiplied by a phase factor. This phase allows us to transfer angular momentum into our BEC and generate a vortex geometry like those formed in the 2009 experiment by Lin *et al.* [73]. As a note, the vortex structures generated in this way follow the magnetic fields lines, thus providing the capability to generate complex vortex structures beyond vortex lines. Outside of rotation, another method to generate these gauge fields in an experimentally realizable way with the evanesent field of a dielectric system undergoing total internal reflection will be described in Chapter 5, but for now we should discuss how these fields can be implemented in the SSFM, with an emphasis on their effects for superfluid simulations.

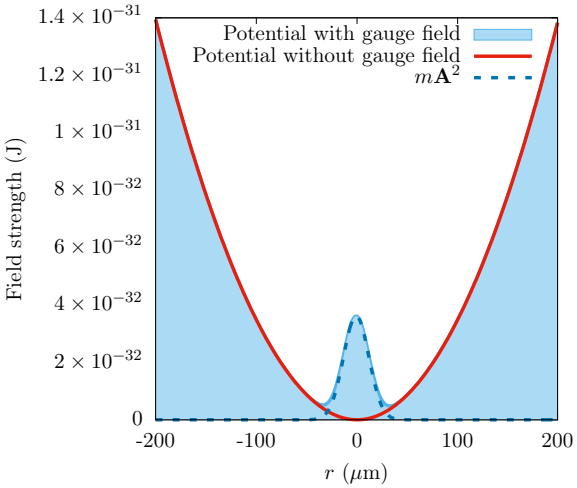

Figure 1.4: Simulation of GPE with an arbitrarily chosen Gaussian artificial vector potential. For this simulation, we have removed the pA term to focus on the warped potential, itself.

1.4 Modifications to the SSFM for superfluid vortex simulations

As shown above, in order to simulate the effects of artificial magnetic fields on BEC systems, we must slightly modify the Hamiltonian, such that

$$\hat{\mathcal{H}} = \frac{(p - m\mathbf{A})^2}{2m} + V_0 + g|\Psi(\mathbf{r}, t)|^2, \tag{1.51}$$

where $\mathbf{A}$ is an artificial vector potential. As a note, some texts absorb the mass term into the artificial vector potential, but here we are stating it explicitly for clarity. When expanded, we note that the gauge field has a component in position space, $\frac{m\mathbf{A}^2}{2}$, which couples with the trapping potential, and another component that is partially in both position and momentum space, $p\mathbf{A}$.

Firstly, let us discuss the component purely in position space. Here, we see that it is important to properly balance the trapping potential and the artificial vector potential when simulating BEC systems with artificial vector potentials, otherwise, the trapping geometry will become warped. In the case of rotation, this is effectively balanced by the centripetal force; however, in the case of arbitrarily chosen vector potentials, this can create rather unusual potential geometries, as shown in Figure 5.7 for a Gaussian $\mathbf{A_x}$ and $\mathbf{A_y}$. Though we might be able to balance this warping with a centripetal force tailored to the gauge fields we introduce, we did not consider this for the purposes of this work.

The components of the artificial vector potential that are partially in position and momentum space are somewhat difficult to consider numerically. As they reside in both spaces, it is required to perform a one-dimensional FFT across n-dimensional data. This means that if we have three operators, $p\mathbf{A}_x\hat{x}$, $p\mathbf{A}_y\hat{y}$, and $p\mathbf{A}_z\hat{z}$, we will need to perform an FFT on our wavefunction in the $\hat{x}$, $\hat{y}$, and $\hat{z}$ dimensions, respectively, before applying the operators through element-wise multiplication. As I will discuss in

Chapter 3, this has significant performance penalties if we do not consider the massively parallel GPU architecture. This also requires the usage of more intricate FFT plans for the FFTW or CuFFT libraries, which are non-trivial for three-dimensional simulations.

1.5 Multi-component simulations

Until now, the discussion has been oriented around single-component condensates; however, in recent years, there has been a significant amount of theoretical and experimental interest in multiple BEC components in the same system [CN]. When simulating such systems, the GPE must be modified to take interactions between components into account and each wavefunction must be evolved separately, such that

$$i\hbar\frac{\partial\Psi_i(\mathbf{r},t)}{\partial t} = \left(\frac{(p-mA)^2}{2m} + V_0 + \sum_{j=1}^{C} g_{ij}|\Psi_j(\mathbf{r},t)|^2\right)\Psi_i(\mathbf{r},t) \tag{1.52}$$

where C is the total number of components being simulated, $\Psi_i(\mathbf{r},t)$ for $i \in 0,...,C$ is the wavefunction for a unique component being evolved, and g_{ij} is the interaction strength between components i and j. The interaction matrix is symmetric, such that $g_{ji} = g_{ij}$, and there are generally three mixing regimes in which multi-component condensates can be found:

No interaction In this regime, $g_{ij} = 0$, while $g_{ii} = 1$. This creates an interaction matrix that is identical to the identity matrix and can be understood as a simulation of C GPE components without interaction. It is rare that these simulations will be performed, simply because the additional components will produce identical results unless the atomic species is modified, and there is no reason to perform a non-interacting multi-component simulation rather than multiple single-component simulations.

Miscible In this regime, $0 < g_{ij} < 1$ and the C components mix in some fashion. This means that there will be some differentiation between components, but they can still exist in the same physical space.

Immiscible In this regime, $g_{ij} \geq 1$ and the C components no longer mix and therefore cannot exist in the same physical locations as other components.

ADD FIGURE

In Figure **??**, I show simulations for a three component condensate in each of these three regimes for a rotating system. These simulations are memory-intensive, but we will discuss several methods to minimize the memory footprint of single-component simulations in Chapter 3. These methods also apply to multicomponent simulations and allow for simulations with a higher component number.

At this point, we have discussed the SSFM, itself, along with the primary system we will be simulating in this work; however, we have yet to discuss key techniques in quantum engineering that are necessary to motivate several design decisions. As such, we will consider two dynamic methods for quantum state engineering in the following chapter: shortcuts to adiabaticity and quantum optimal control.

Chapter 2

Quantum engineering for one-dimensional quantum systems

Until recently, methods involving dynamic control protocols to engineer specific quantum states have been difficult to perform on GPU devices without full control of the software, itself. This meant that researchers wishing to engineer specific quantum states by using GPU simulations would be required to have domain-specific knowledge in both software design and quantum mechanics, neither of which are trivial to understand. This chapter serves as motivation for several methods to be discussed in Chapter 3 to allow for the simulation of quantum control methods on GPU devices, with a particular focus on quantum optimal control [77] and Shortcuts to Adiabaticity (STA) [78]. Both of these control protocols will be used in a physical example for the non-adiabatic generation of superposition states in a one-dimensional Tonks-Girardeau gas [9]. To start, I will discuss the field of optimization algorithms before moving to STA protocols and a physical system using both in practice.

2.1 Optimization methods

Optimization algorithms have become essential to many areas of modern computing and focus on either minimizing or maximizing a cost function by modifying several control parameters [79]. The number of control parameters create an n-dimensional space to traverse, and optimization algorithms are tasked at finding the global minimum or maximum of this domain. For certain domains, it is difficult to find a global optimization strategy and many methods instead get caught by local minima while attempting to find the appropriate solution. Because generalized optimization is such a fundamental problem, there are many known optimal control methods, such as gradient descent [80], the Nelder-Mead or simplex method [81], genetic algorithms [82], and many more. Of these, gradient descent is often considered to be one of the easiest to implement with favorable complexity and convergence guarantees, and because of this it has become ubiquitous in many areas such as machine learning, which is of particular interest for GPU engineering. Even so, there are limitations to gradient descent, such as its dependence on calculating the gradient of the cost function's solution domain along with lengthy convergence times for high-precision solutions.

For this work, I am primarily interested in the area of quantum optimal control, which is a method typically used to determine the optimal laser pulses necessary to transform an initial state to a final desired state [77]. This means that one will often be maximizing the fidelity between states, defined as $\mathcal{F} = |\langle\psi|\phi\rangle|^2$, where ψ is the engineered quantum state and ϕ is the state we attempting to replicate. This problem can be re-framed as an attempt to find the maximum of a fidelity *landscape*, which is difficult to find the gradient of without resolving the Schrödinger equation. For this reason, we will be introducing a gradient-less (derivative-free) optimization algorithm, the Nelder Mead (simplex) method, which is often used as a heuristic approach to this problem and there are several known optimizations for this method [81, 83, 84]. This method is also the recommended optimization algorithm for the chosen quantum optimal control method of the physical example later discussed in this chapter, Chopped RAndom Basis (CRAB) optimal control. This will be described in Section 2.1.2.

2.1.1 Nelder–Mead

The Nelder–Mead method is one of the most commonly implemented gradient-less optimization algorithms to-date and relies heavily on the concept of a simplex, which is a generalization of the three-dimensional polyhedron to n-dimensions. For example, a 0-simplex is a point, a 1-simplex is a line, a 2-simplex is a triangle, a 3-simplex is a tetrahedron, and so on. If the Nelder–Mead method is attempting to optimize a cost function with n control parameters, an $n+1$ point simplex will be created, and the points of this simplex will be manipulated until they have converged to a minimum or maximum in the domain. For this section, we will focus on the method introduced in the original work by Nelder and Mead in 1965 while working at the National Vegetable Research Station in Warwick, England [81].

For $P_i \in i = \{0, ..., n\}$ simplex points with heights of $y_i \in i = \{0, ..., n\}$, the Nelder–Mead method is tasked at minimizing all points such that $\sqrt{\sum(y_i - \bar{y})^2/n} < \eta$, where $\bar{y}$ denotes the height of the centroid location of the simplex, and η is some pre-defined threshold value. This convergence criteria assumes that if all points have converged with this method, the final location must be the minimum of the optimization domain; however, as mentioned in the previous section, this method may become trapped in a local minima. At every step in the Nelder–Mead method, the points with the highest and lowest values are determined and denoted as P_h and P_l, respectively. In addition, the centroid location is found as $\bar{P}$. This method then performs up to three basic operations on the simplex, itself:

Reflection For this operation, P_h is flipped across $\bar{P}$, such that the new location,

$$P^* = (1+\alpha)\bar{P} - \alpha P_h. \tag{2.1}$$

Here, $\alpha > 0$ is a constant known as the reflection coefficient. After this operation, the new height is compared to y_l. If it is lower, the method proceeds to an expansion step. Otherwise, the method checks whether the new height is lower than some other $y_i \in \{0, ..., n\}, n \neq \{l, h\}$, and if it is, the point is kept and the method continues to find the new simplex ordering. If it is found that the reflected point is higher in value than all other points, the method then keeps whichever

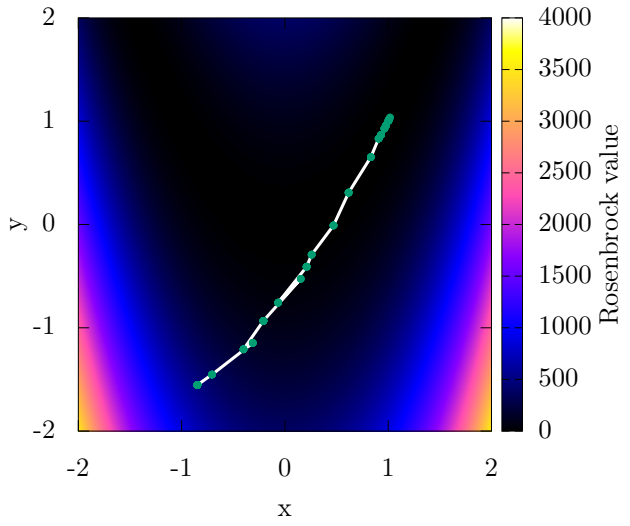

Figure 2.1: Plot of the centroid locations (green dots connected with white line) for the Nelder–Mead method while optimizing the Rosenbrock banana function, $f(x, y) = (a - x)^2 + b(y - x^2)^2$, with $a = 1$ and $b = 100$. Here, we see the centroid locations starting at $(-0.851, -1.553)$ and end at the known minimum of $(1, 1)$ in 19 iterations.

point corresponds to the lowest height between the reflected and previous highest point and proceeds to the contraction step.

Expansion For this operation, an expansion is performed, such that the new location,

$$P^{**} = \gamma P^* + (1 - \gamma)\bar{P}. \tag{2.2}$$

Here, $\gamma > 1$ is a constant known as the expansion coefficient. If the new height is less than the previously lowest point, the expanded point is kept, otherwise the reflected point is kept.

Contraction For this operation, a contraction is performed, such that the new location,

$$P^{**} = \beta P_h + (1 - \beta)\bar{P}. \tag{2.3}$$

Here, $0 < \beta < 1$ is a constant known as the contraction coefficient. If the contracted point is lower than the previous highest point, we keep the contracted point, otherwise, we contract the entire simplex closer to the lowest point with $P_i = (P_i + P_l)/2$.

Each step in the Nelder–Mead method begins with a proposed reflection of the least optimal point about its centroid position. From there, the method follows the protocol described above. The choice of α, β, and γ is somewhat arbitrary and should be optimized by-hand. An example of the centroid locations for minimization using this method with the Rosenbrock banana function [85] can be seen in Figure 2.1.

It is clear that this method is not intended to be optimal, but is sufficient for this discussion. Ultimately, any gradient-less optimization algorithm can be used to traverse the fidelity landscape for quantum optimal control, and in the next section, I will discuss a common method used in the field: the Chopped RAndom Basis (CRAB) optimal control method.

2.1.2 Chopped random basis

The CRAB technique works by modifying a control parameter for a given system, Γ, with a multiplicative term as

$$\Gamma^{\mathrm{CRAB}}(t) = \Gamma^0(t)\gamma(t), \tag{2.4}$$

where $\Gamma^0(t)$ is an initial guess, and the function $\gamma(t)$ is written as a sum of $2J$ sinusoidal functions,

$$\gamma(t) = 1 + \frac{1}{\lambda(t)} \sum_{j=1}^{J} (A_j \sin(\nu_j t) + B_j \cos(\nu_j t)). \tag{2.5}$$

Here, $\lambda(t)$ is usually defined by the system, such that Γ^{CRAB} and Γ^0 coincide at initial and final times. This means that $\lim_{t\to 0} \lambda(t) = \lim_{t\to T} \lambda(t) = \infty$, where T is the final time of evolution. As such, any smooth function may be chosen with these constraints. For example, we might use

$$\lambda(t) = \frac{T^2}{4t(t-T)}, \tag{2.6}$$

which satisfies the provided conditions. This then transforms the operation to an optimization of the a space spanning $\{A_j, B_j, \nu_j\}$, which can be done by using Nelder–Mead with a simplex of random initial points. As an example, if $J = 10$, a thirty-dimensional space would be created and a thirty-one simplex would be formed to traverse this space It is important to remember that each new simplex and simplex-operation might require re-solving the Schrödinger equation for those values, and as such, this is a computational costly technique. Even though higher J values will produce a more accurate result, this is why lower values should be chosen, if possible.

This method is a general-purpose computational tool for determining the optimal pulse to ensure the generated state is as close to the desired state as possible, and we will see an example of it being used later in this chapter. As such, it is sometimes worthwhile to attempt to devise analytical frameworks that serve a similar purpose, and for certain systems, this can be done with Shortcuts To Adiabaticity (STA).

2.2 Shortcuts to adiabaticity

STA protocols are semi-analytical methods that allow for quantum state generation while retaining the effects of adiabatic movement. Here, adiabatic processes are defined as actions by which slow changes in the control parameters leave particular properties invariant, such as the quantum number [78]. The ultimate goal of STA protocols is to achieve adiabatic motion in sub-adiabatic time, and this can be done in a number

of ways; however, in this section, we will introduce only the invariant-based inverse-engineering approach using Lewis-Riesenfeld invariants [86]. In particular, I will focus on the specific methods necessary for the example to be introduced later in this chapter and much of this section will follow traditional derivations from various sources [9, 78, 86].

With the method of Lewis-Riesenfeld invariants, we find that the theory for relating different eigenstates of a time-dependent, Hermitian invariant to the solutions to the Schrödinger equation [87] can be applied to systems with time-dependent Hamiltonians, such that

$$\frac{d}{dt} = i\hbar\frac{\partial I(t)}{\partial t} - \frac{i}{\hbar}\left[\hat{\mathcal{H}}, I(t)\right] = 0, \tag{2.7}$$

where, $I(t)$ is the invariant. This ensures that the expectation values for states driven by $\hat{\mathcal{H}}$ are constant in time. It is possible to expand the state of the system $|\Psi(t)\rangle$ into the orthonormal basis of the invariant with, $I(t)$ can also be used to expand $|\Psi(t)\rangle$ into a superposition of dynamical modes $|\psi_n(t)\rangle$,

$$|\Psi(t)\rangle = \sum_n^\infty c_n |\psi_n(t)\rangle \tag{2.8}$$

$$|\psi_n(t)\rangle = e^{i\alpha_n(t)} |\phi_n(t)\rangle , \tag{2.9}$$

where c_n are time-independent amplitudes for each state, and $|\phi_n(t)\rangle$ are orthonormal eigenvectors of the invariant, such that

$$I(t) = \sum_n^\infty |\phi_n(t)\rangle \lambda_n \langle\phi_n(t)| . \tag{2.10}$$

Here, the λ_n are real constants, and the phase is defined as [87]

$$\alpha_n(t) = \frac{1}{\hbar}\int_0^t \langle\phi_n(t')|\hbar\frac{\partial}{\partial t'} - \hat{\mathcal{H}}(t')|\phi_n(t')\rangle\, dt'. \tag{2.11}$$

From here, inverse engineering can be used to create the desired time-dependent Hamiltonian. The phases, $\alpha_n(t)$ may be chosen as arbitrary functions to create a time-dependent unitary operator,

$$U = \sum_n^\infty e^{i\alpha_n(t)} |\phi_n(t)\rangle \langle\phi_n(0)| , \tag{2.12}$$

obeys $i\hbar\dot{U} = \hat{\mathcal{H}}(t)U$ and the dot is a time-derivative. If one considers Hamiltonians of the Lewis and Leach variety [88],

$$\hat{\mathcal{H}} = \frac{p^2}{2m} - F(t)q + \frac{m}{2}\omega^2(t)q^2 + \frac{1}{\rho(t)^2}U\left[\frac{q-q_c}{\rho(t)}\right] + f(t) \tag{2.13}$$

there will be an invariant that is quadratic in momentum,

$$I = \frac{1}{2m}[\rho(p - m\dot{q}_c) - m\dot{\rho}(q-q_c)]2 + \frac{1}{2}m\omega_0^2\left(\frac{q-q_c}{\rho}\right)^2 + U\left(\frac{q-q_c}{\rho}\right). \tag{2.14}$$

These equations are valid so long as ρ, q_c, ω, and F satisfy

$$\ddot{\rho} + \omega^2(t)\rho = \frac{\omega_0^2}{\rho^3} \tag{2.15}$$

$$\ddot{q}_c + \omega^2(t)q_c = F(t)/m \tag{2.16}$$

with ω_0 as a defined constant whose physical interpretation depends on the system. As in the case of quantum optimal control, additional constraints must be considered to ensure the Hamiltonian and its invariant commute at initial and final times t_0 and T.

The obvious drawbacks to STA methods are the strengths of quantum optimal control. Where shortcuts can only be used on a specific subset of problems we wish to evolve adiabatically and that are amenable to the analytical methods used, quantum optimal control is a more general tool for a wider variety of systems. On the flip-side, STA protocols are semi-analytical and if such protocols can be found, they greatly reduce the computational cost to engineering particular quantum states. Now that I have provided specific examples of methods used in quantum engineering, it is time to put them into practice with an example of creating large-scale superposition states non-adiabatically in the highly-correlated Tonks–Girardeau (TG) gas regime.

2.3 Non-adiabatic generation of NOON states in a Tonks–Girardeau gas

For this example application of quantum optimal control and STA protocols, we are interested in generating the maximally entangled $|N, 0\rangle + |0, N\rangle$ (NOON) state, which is composed of two modes where all particles can be found exclusively in one or the other. Such behavior has been created experimentally in photonic states generated by mixing classical states with down-converted photon pairs [89] In this system it is possible to create NOON states with around 5 photons [90]. Recently, Hallwood *et al.* proposed an experimentally realistic method to generate NOON states in a gas of strongly interacting, neutral bosons on a one-dimensional ring. In this system, different rotational states can be coupled by breaking the rotational symmetry and it is possible to create superposition states with rotating and non-rotating components. Because the atoms are considered to be in the strongly correlated TG gas regime, this process results in a macroscopically-entangled state. It is worth discussing the TG gas in further detail before discussing the precise method of NOON state generation for this example.

2.3.1 TG gas

As discussed in Chapter 1, the TG gas consists of a number of bosons that have the properties of spinless, non-interacting fermions. This is a particular case of the one-dimensional Schrödinger equation where $g \to \infty$. In this case, the bosons cannot pass each other and cannot be at the same location, which acts formally similar to the Pauli-exclusion principle for fermionic systems. In this case, the Hamiltonian can be solved

by the Bose–Fermi mapping theorem [91, 92], which replaces the interaction terms in the Hamiltonian with a boundary condition on the many-body bosonic wavefunction.

$$\Psi_B(x_1, x_2, \ldots, x_N) = 0, \qquad \text{if} \qquad x_i - x_j = 0 \quad \text{with} \quad i \neq j. \tag{2.17}$$

The Bose–Fermi mapping theorem allows us to replace strongly interacting bosons by spinless, non-interacting fermions, on which we can use the Slater determinant [93],

$$\Psi_F(x_1, x_2, \ldots, x_N) = \frac{1}{\sqrt{N}} \det\left[\psi_n(x_j)\right]_{n,j=1}^{N}. \tag{2.18}$$

where $\psi_n(x_j)$ are the single-particle eigenstates of the trapping potential V_0. Because the Fermionic many-body wavefunction is anti-symmetric, this needs to be symmetrized for bosonic states as,

$$\Psi_B(x_1, x_2, \ldots, x_N) = \prod_{i<j} \mathrm{sgn}(x_i - x_j) \Psi_F(x_1, x_2, \ldots, x_N) \tag{2.19}$$

which means that calculating the time evolution of a TG gas requires evolving single-particle states, governed by a much simpler Hamiltonian.

2.3.2 NOON states in TG gas

Similar to other ring systems introduced in the literature [94, 95], the system suggested by Hallwood *et al.* considers a gas of N interacting bosons of mass m on a one-dimensional ring with circumference L [96]. In addition, this system includes a potential barrier, modeled by a Dirac δ function that rotates with an angular frequency Ω, as shown in Figure 2.2. In the rotating frame, the scaled Hamiltonian of the system is given by [96]

$$H^{(N)} = \sum_{i=1}^{N} \left[\frac{1}{2} \left(-i \frac{\partial}{\partial x_i} - \Omega \right)^2 + b\delta(x_i) + g \sum_{i<j}^{N} \delta(x_i - x_j) \right], \tag{2.20}$$

where b is the height of the barrier (in units of $\hbar^2/mL^2$), $x_i \in [-1/2, 1/2]$ is the position of the i–th particle (in units of L) and g (in units of $\hbar^2/mL^2$) is the effective interaction strength between the atoms. As discussed in the previous section, the evolution of a TG gas requires the evolution of single-particle states, and in the case of this system, the Hamiltonian becomes,

$$H = -\frac{1}{2} \frac{\partial^2}{\partial x} + b\delta\left[x - x_0(t)\right], \tag{2.21}$$

where x_0 is the position of the barrier at time t.

The energy spectrum of this system is shown in Figure 2.3 as a function of the rotational frequency $\Omega \equiv \dot{x}_0/L$ of the system. In Figure 2.3(a) it is shown that in the absence of a barrier, the eigenstates of $\hat{\mathcal{H}}$ are plane waves with quantized angular momentum in integer multiples of 2π; however when $b > 0$ (Figure 2.3(b)), the symmetry is broken and avoided crossings appear in the energy spectrum. This makes transitions between different manifolds possible [97].

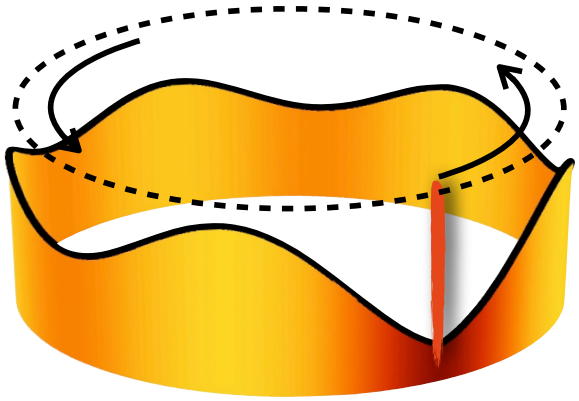

Figure 2.2: Schematic of the system described by Hallwood *et al.* [96]. Here, the density profile for five atoms in a TG gas is shown being stirred by a highly localized potential, indicated by the vertical line.

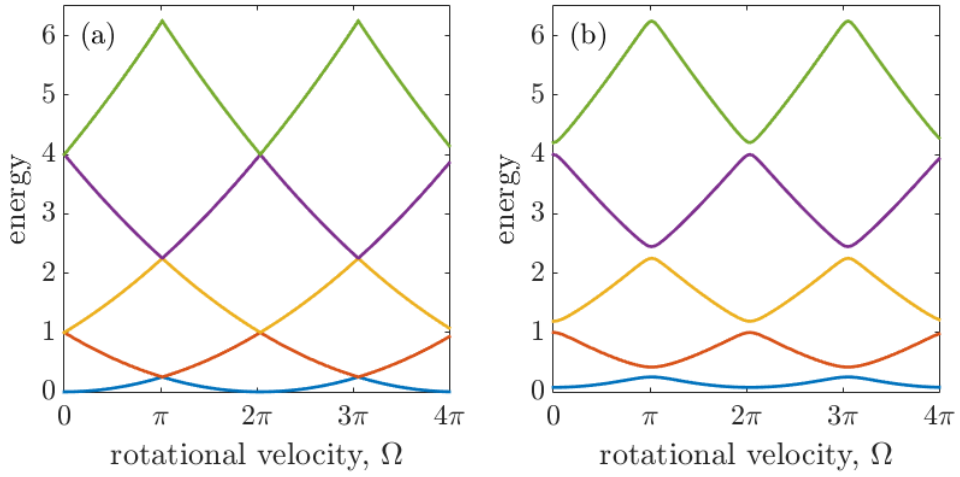

Figure 2.3: Single-particle energy spectrum as a function of Ω for a barrier height of (a) $b = 0$ and (b) $b = 2$. When a barrier is present in the system, avoided crossings appear in the energy spectrum which grow as the barrier strength increases.

By adiabatically accelerating the barrier's rotational frequency from 0 to π, a particle will enter a superposition between two rotational states, and in the case of the TG gas, this will create a macroscopic NOON superposition state between successive values of angular momentum [96]. Any non-adiabatic behavior around the rotational frequencies of the avoided crossing can lead to a transition to a higher energy state and destroy the NOON state. For this reason, the condition for adiabaticity must depend on the gap size, which is dictated by the barrier strength [98]; however, for a delta barrier, the gap size stays relatively constant to first-order approximation [99].

Because this system requires adiabatic movement to properly generate the NOON state, it is difficult to efficiently generate it experimentally. For this reason, it is a perfect example of a system where quantum optimal control and STA protocols can be used to rapidly engineer the appropriate states. For quantum optimal control in this system, a non-adiabatic rotational frequency $\Omega(t)$ must be found, for which we will use the CRAB technique with an initial condition of $\Omega = 0$ and final condition of $\Omega = \pi$. For each simulation in the fidelity landscape, we will be modifying this pulse with procedurally generated sinusoidal functions, calculating the fidelity, and then using the Nelder–Mead method to optimize the result. This will allow us to determine an optimal pulse that maximizes the fidelity of our generated state when compared to the expected NOON state in a pre-set amount of time. For this system, we will show the optimal pulse for the rotational velocity, the barrier height, and both.

For STA protocols, we will split the acceleration process into two, one that breaks the rotational symmetry and another that accelerates the atoms. At the end of the protocol, the potential is lowered to restore rotational symmetry. Here, it is worth mentioning that a FAst, QUasi-ADiabatic (FAQUAD) shortcut for the creation of superposition stated in a TG gas has also been created with some similarities [100].

For both of these methods, instead of calculating the fidelity, itself, we calculate the *infidelity*, which is simply $1 - \mathcal{F} = 1 - |\langle\Psi|\Phi\rangle|^2$. It is also worth mentioning that the fidelity between two many-particle states in a TG gas can be calculated by using the method of mode projections [101, 102],

$$\begin{aligned}\langle\Psi|\Phi\rangle &= \frac{1}{N!}\sum_{\eta,\mu\in P}\epsilon_\eta\epsilon_\mu\langle\psi_{\eta_1}(x_1)|\phi_{\mu_1}(x_1)\rangle\cdots\langle\psi_{\eta_N}(x_N)|\phi_{\mu_N}(x_N)\rangle \\ &= \det\Big[\langle\psi_i|\phi_j\rangle\Big]_{i,j=1}^{N}\end{aligned} \tag{2.22}$$

which follows directly from the form of the TG state [34]

$$\Psi(x_1,x_2,\ldots,x_N) = \frac{1}{\sqrt{N!}}\prod_{i<j}\mathrm{sign}(x_i-x_j)\sum_{\eta\in P}\epsilon_\eta\psi_{\eta_1}(x_1)\cdots\psi_{\eta_N}(x_N). \tag{2.23}$$

Here P represents the set of all permutations of N elements, ϵ_η represents the anti-symmetric tensor of the permutation η, and ψ_i represent the orbitals. Now I will discuss our findings with both optimal control and STA protocols.

2.3.3 Results with optimal control

First, I will focus on the acceleration of a single particle, initially in the ground-state of the system. Figure 2.4 (a) shows the results of this simulation if the barrier height is

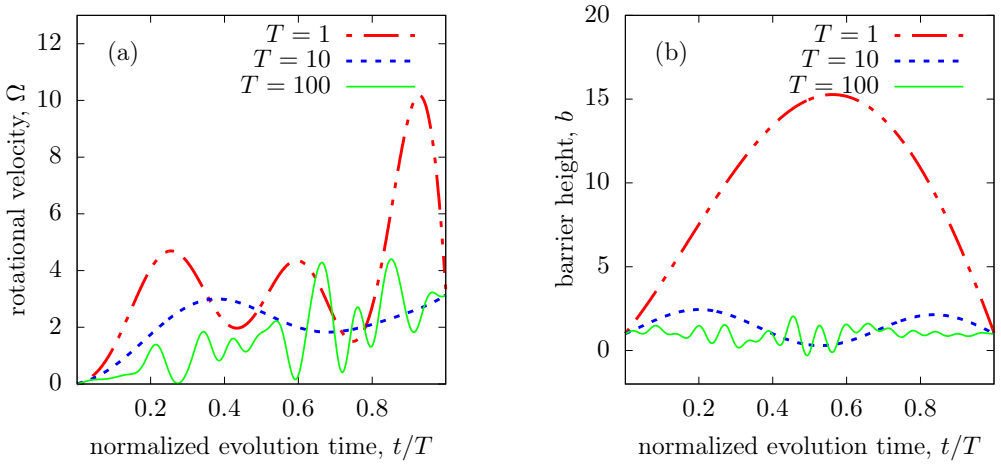

Figure 2.4: Accelerating a single particle from the ground state with $J = 15$. (a) Optimal rotational velocity pulses for $T = 1$, 10, and 100 for fixed barrier height $b = 1$. (b) Optimal barrier height for a linearly increasing rotational velocity, $\Omega = \pi t/T$ for $T = 1$, 10, and 100.

kept constant and we assume an initial unmodified pulse that corresponds to a linear ramp from $\Omega = 0$ to π for a preset total time T. For longer evolution times, there are many local maxima for the fidelity, and as such, longer evolution times effectively produce noisy signals, as the Nelder–Mead method converges on one of many local minima. For shorter evolution times, the pulse greatly affects the system and the shapes vary greatly from the initial linear ramp. The infidelities for the linear guess pulse and its corresponding optimized pulse can be found in Figure 2.6, and one can see an improvement of several orders of magnitude. Here, we see that for longer evolution times, the initial linear pulse is a reasonable method to generate NOON states with an infidelity of 10^{-2} because it is close to adiabaticity by definition; however, even in this case, the NOON state generation fidelity is better with optimization. For all optimal control results in this Chapter, the CRAB method was run 100 times and the data with the highest fidelity was kept.

For optimizations of the barrier strength, we chose a simple linear ramp for Ω and an initial final height for the barrier at $b = 1$. The optimal pulses for the barrier height for $T = 1$, 10, and 100 are shown in Figure 2.4(b) and shorter evolution times similarly produce larger deviations from the initial pulse. Again these pulses lead to significant improvements in the fidelity shown in Figure 2.6.

With the CRAB method, it is possible to optimize over as many control parameters as one would like, and as such, it is possible to optimize over both the barrier strength and rotational frequency, and the results can be seen in Figure 2.5. When comparing to the previous cases, similar trends emerge. In particular, shorter evolution times result in less noisy optimizations when compared to longer evolution. Even so, all sets of pulses are radically different when compared to optimizations over a variable. When comparing the fidelities in Figure 2.6, it is clear that optimizations over rotation, alone

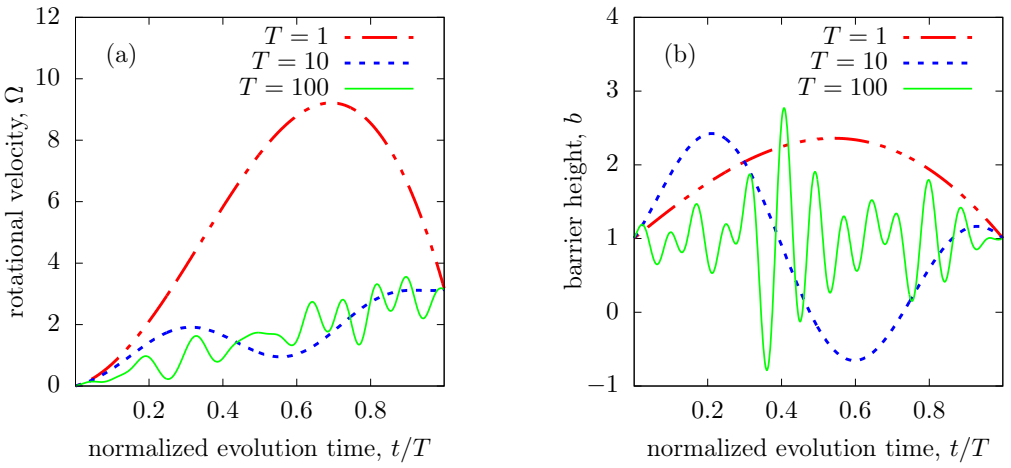

Figure 2.5: Optimal pulses to accelerate a single particle initially in the ground state of the trap for $T = 1$, 10, and 100 for the (a) rotational velocity and (b) barrier height when optimizing over both simultaneously.

provides the simplest method to optimize the fidelity with this method. It is likely that each run of the CRAB method became stuck in a local minima in the fidelity landscape at some point and that optimization over both variables can provide the same optimization as rotating, alone.

Finally, it is worth discussing the dynamics of a TG gas with 3 and 5 particles. Because of the Bose–Fermi mapping theorem, the evolution of an N-particle TG gas can be calculated by evolving a gas of N spinless fermions. In the zero-temperature limit, the fermions in the initial and target state create a Fermi sea by filling the lowest N energy levels. In this case, only atoms near the Fermi edge can transition into empty states and it is thus crucial to optimize the dynamics of the overall gas with respect to the particle with highest energy [100]. In Figure 2.7, we show the fidelity for the particle closest to the Fermi edge and the entire TG gas for $N = 3$ and 5. In this figure, we see that by performing the optimization for the atoms near the fermi edge, we can increase the fidelity of the entire gas for certain regimes; however, in contrast to Figure 2.6, there seems to be no fidelity increase from a linear pulse for short evolution times. There also seems to be a crossover regime where optimizations of the particle at the fermi edge seem to fail, but evolution of the entire gas is still slightly more optimal than the linear pulse. It is clear that the CRAB method creates highly effective pulses; however, for very short and long evolution times, the fidelity increase from a linear pulse is not as drastic.

2.3.4 Results with STA protocols

In this section, we will describe an STA protocol to generate NOON states in this system non-adiabatically, and though this was mentioned briefly in Section 2.3.2, it will be described more rigorously here. For this method, the rotational symmetry

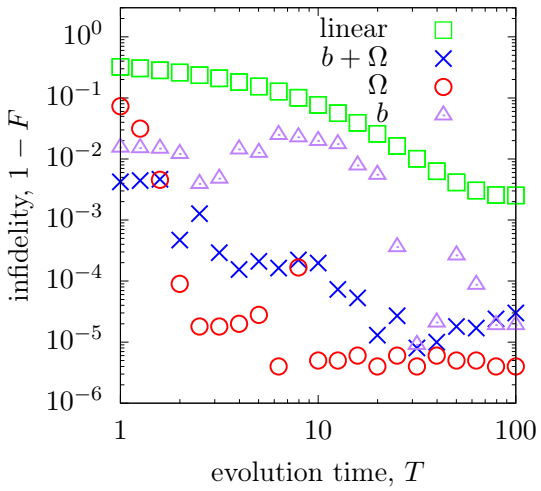

Figure 2.6: Infidelities as a function of the overall process time for optimally controlled rotational acceleration, barrier height, or both. Here, 'linear' refers to an unoptimized linear acceleration from $\Omega = 0$ to π while keeping the barrier height fixed at $b = 1$.

must be broken by introducing a time-dependent external potential that is removed in the end. This requires a slight modification of the system proposed by Hallwood *et al.* to allow for a barrier that is not a δ function, but instead a harmonic or sinusoidal potential along the ring.

The protocol consists of five steps:

1. Adiabatic raising of a weak harmonic or sinusoidal potential around the ring.
2. Fast tightening of this potential to localize the particles.
3. Accelerating the particles by moving the center of the potential.
4. Lowering the potential by reversing step 2.
5. Adiabatic lowering of the harmonic or sinusoidal potential.

A schematic of this process is shown in Figure 2.8. For steps 2-4, pre-existing STA protocols can be used, and these will be discussed in this section. The full protocol for the TG ring example will follow the STA methods outlined above with Lewis-Riesenfeld invariants and in our case, we need to fulfil boundary conditions such that $\hat{\mathcal{H}}(t_0) = \hat{\mathcal{H}}(T) = p^2/2m$. To be clear, the NOON states created with the STA protocol are slightly different those generated with quantum optimal control, as the STA variant does not rely on a δ barrier.

One of the two known shortcuts being used for this system involves raising and lowering a harmonic potential [103, 104]. For this shortcut, we need a stationary harmonic potential and can set F, q_c, and U from Equation (2.13) to zero, leading to

$$\hat{\mathcal{H}} = -\frac{1}{2}\frac{\partial^2}{\partial x} + \frac{1}{2}\omega^2(t)q^2. \tag{2.24}$$

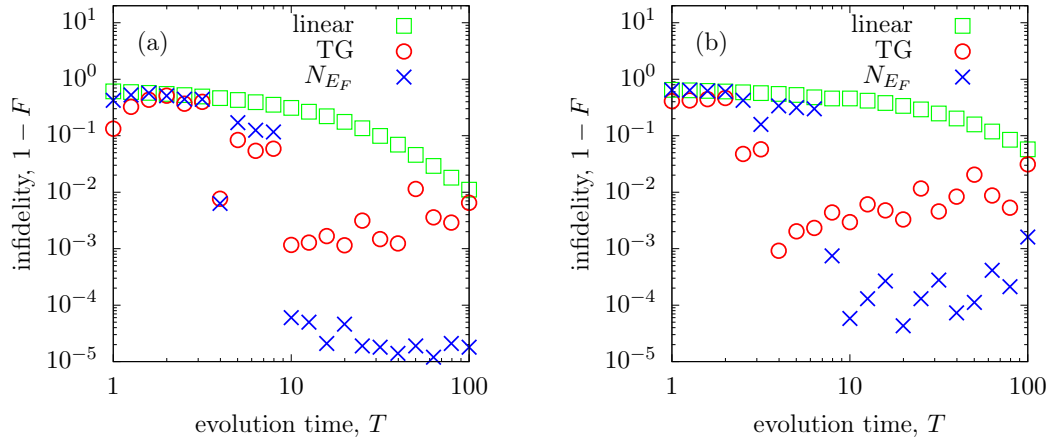

Figure 2.7: Infidelities for the evolution of a TG gas with (a) $N = 3$ and (b) $N = 5$ particles using the CRAB optimal control technique. The optimal pulses were determined for the particle at the Fermi edge (blue crosses), and applied to the whole gas (red circles). Here, the green squares show the fidelity of a linear pulse for the atom closest to the Fermi edge. A clear range where the CRAB algorithm is effective for generating NOON states with multiple particles can be clearly identified.

To change the frequency while keeping the commutation relations and $\omega(t)$ continuous the following conditions must be imposed:

$$\begin{aligned} \rho(t_0) &= 1, \quad & \rho(t_f) &= \gamma = \sqrt{\omega_0/\omega_f}, \\ \dot{\rho}(t_0) &= 0, \quad & \dot{\rho}(t_f) &= 0, \\ \ddot{\rho}(t_0) &= 0, \quad & \ddot{\rho}(t_f) &= 0. \end{aligned} \tag{2.25}$$

As long as the conditions, Equations (2.15) and (2.25) are obeyed, any form of ρ can be chosen, and a good choice is the polynomial,

$$\rho(s) = 6\,(\gamma - 1)\,s^5 - 15\,(\gamma - 1)\,s^4 + 10\,(\gamma - 1)\,s^3 + 1, \tag{2.26}$$

where $s = (t - t_0)/(t_f - t_0)$ allows us to numerically find a solution for $\omega(t)$ that leads to the squeezing or expansion of the particle wavefunction with high fidelity in a short time. As an important note, small values for $omega_0$ can provide purely imaginary values for $\omega(t)$ from Equation (2.15), corresponding to repulsice potentials. Because our final states require the external potential to be absent, the first and final steps in the protocol for this system involves adiabatically raising and lowering a potential to a suitable ω_0 value.

Once the potential has been raised, it is time to accelerate the particles to the chosen frequency, and a shortcut for this process with a harmonic trap exists [105–107]. In the rotational shortcut, the trapping frequency is held constant and the position of the potential is modified. This means that $U = 0$, $F = \omega_0^2 x_0(t)$, and

$$H = -\frac{1}{2}\frac{\partial^2}{\partial x} + \frac{1}{2}\omega_0^2(q - x_0(t))^2. \tag{2.27}$$

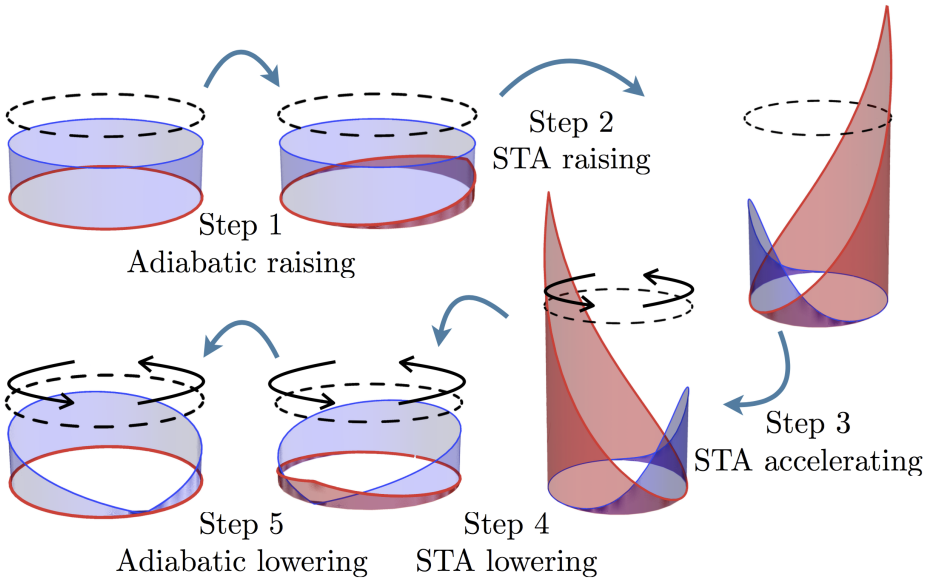

Figure 2.8: Scheme for the acceleration of a single atom using STA. In this example, the ground state of free space gets localized, accelerated and released at the angular velocity of $\Omega = \pi$ into the state $(\exp(i2\pi x) + 1)/\sqrt{2}$. The atomic density is indicated in blue and the potential is in red.

Here, Equation (2.16) becomes the only relevant auxiliary equation,

$$\ddot{q}_c + \omega_0^2(q_c - x_0) = 0. \tag{2.28}$$

Conditions are then imposed on q_c, such that

$$\begin{aligned} q_c(t_0) &= x_0(t_0), & q_c(t_f) &= d, \\ \dot{q}_c(t_0) &= 0, & \dot{q}_c(t_f) &= \Omega_f, \\ \ddot{q}_c(t_0) &= 0, & \ddot{q}_c(t_f) &= 0, \end{aligned} \tag{2.29}$$

where d is the final position of the potential minimum and Ω_f is its final velocity. For most applications of this shortcut, d is important, and Ω is set to zero; however, our case is the opposite.

Like the shortcut for raising the potential, the exact form of q_c can be chosen somewhat arbitrarily, and a convenient choice is

$$q_c(s) = (6d - 3\Omega_f)s^5 - (15d - 7\Omega_f)s^4 + (10d - 4\Omega_f)s^3 + x_0(t_0), \tag{2.30}$$

where, as above, s is the normalized time. The value of Ω_f can then be chosen to be odd multiples of π to generate the desired NOON states.

Unlike the shortcut to raise the potential, this shortcut is only approximate and works best when ω is large so that the particles are highly localized. Both of these shortcuts rely on the presence of a harmonic potential of the form

$$V_H(x,t) = \frac{1}{2}\omega^2(t)\left(x - x_0(t)\right)^2, \tag{2.31}$$

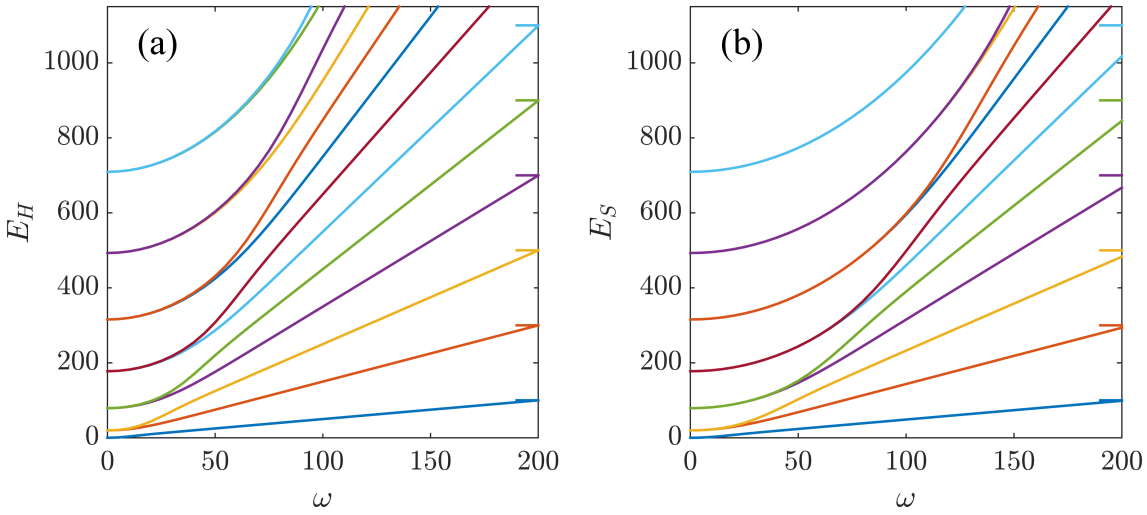

Figure 2.9: Energy eigenspectrum of the system with (a) a harmonic or (b) a sinusoidal potential as a function of ω. The eigenstates continuously change from angular momentum states of energy $E_k = 2\pi^2 k^2$ (with $k = 0, \pm 1, \ldots$) at $\omega = 0$, towards harmonic-oscillator states of energy $E_n = \omega(n + 1/2)$ (with $n = 0, 1, \ldots$) for large ω. For comparison, the horizontal lines on the right vertical axis give the energy levels in a harmonic potential with $\omega = 200$.

where ω is the frequency of the trap (in units of $\hbar/mL^2$) and x_0 the position of its minimum. In the case of the TG ring, the potential must be symmetric around x_0, such that it is continuous at $x = \pm 1/2$; therefore, the real form of $(x - x_0)$ must be $(x - x_0 + 1/2)(\text{mod } 1) - 1/2$. The potential V_H is then continuous everywhere on the ring, but its derivative is discontinuous at $x = x_0 + 1/2$ because this position is diametrically opposite to x_0. Though V_H is ideal theoretically, it is not necessarily experimentally realistic, so a sinusoidal potential is also considered [108, 109],

$$V_S(x,t) = \frac{\omega^2(t)}{2\pi^2} \sin^2\left(\pi\left(x - x_0(t)\right)\right), \tag{2.32}$$

where the notation is the same as before. Here, prefactors are chosen such that V_H is an approximation of V_S around x_0.

In Figure 2.9 we show the difference between the two potentials by computing the energy spectra of both Hamiltonians. Here, the eigenstates at $\omega = 0$ are the angular momentum states $e^{i2\pi kx}$, with degenerate clockwise and counterclockwise momentum states of opposite quantum number k. As ω is increased, the degeneracy ceases and the spectrum asymptotically approaches that of a harmonic oscillator. For the sinusoidal case, the difference with the harmonic spectrum increases with the quantum number n.

Like the case of quantum optimal control, we will first show the single-particle results. In Figure 2.10(a), the values for $\omega(t)$ and $\Omega(t)$ are shown, and in Figure 2.10(b), the infidelities for the state preparation of plane waves $e^{i\Omega_f x}$ with $\Omega_f = 1 \ldots 10 \times 2\pi$ are shown. Here, it is clear that even for a large amount of angular momentum, the fidelities remain high for both the harmonic and sinusoidal potentials.

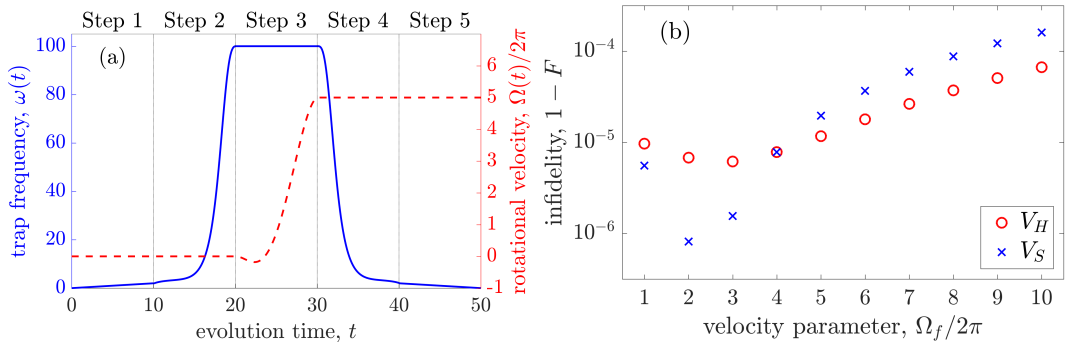

Figure 2.10: (a) Plot of the parameters $\omega(t)$ and the angular velocity $\Omega(t)$ for the entire protocol. The parameters are $\omega_0 = 2$, $\omega_f = 100$, $d = 100$, each step is executed in $t_f - t_0 = 10$, and Ω_f is picked depending on the desired output state (here, $\Omega_f = 5\times 2\pi$). (b) Final infidelities for $\Omega_f = 1, 2, \ldots, 10 \times 2\pi$ for V_H (dotted blue line) and V_S (solid red line). The rest of parameters are as shown in (a).

For a multi-particle case in the TG regime, the initial states for the particles will be eigenstates of free space, which are simply plane waves $e^{i2\pi kx}$ with integer k. Because the states with $\pm k$ are degenerate, it is equally valid to consider the initial eigenstates

$$\phi_0^i(x) = 1, \tag{2.33}$$

$$\phi_{2l-1}^i(x) = \frac{1}{\sqrt{2}}\left(e^{i2\pi lx} - e^{-i2\pi lx}\right) = i\sqrt{2}\sin(2l\pi x), \tag{2.34}$$

$$\phi_{2l}^i(x) = \frac{1}{\sqrt{2}}\left(e^{i2\pi lx} + e^{-i2\pi lx}\right) = \sqrt{2}\cos(2l\pi x), \tag{2.35}$$

for $l = 1, 2, \ldots$. These states have a total angular momentum of zero and are well-suited for the provided STA protocol because when an odd number of particles occupies the lower eigenstates, the sin/cos pairs are guaranteed to be populated.

For $\Omega_f = \pi$, the plane wave of quantum numbers $k+1$ and $-k$ are degenerate and we can construct the target states

$$\phi_{2l}^t(x) = \frac{1}{\sqrt{2}}\left(e^{i2\pi(l+1)x} + e^{-i2\pi lx}\right) = \sqrt{2}\cos[(2l+1)\pi x]e^{i\pi x}, \tag{2.36}$$

$$\phi_{2l+1}^t(x) = \frac{1}{\sqrt{2}}\left(e^{i2\pi(l+1)x} - e^{-i2\pi lx}\right) = i\sqrt{2}\sin[(2l+1)\pi x]e^{i\pi x}, \tag{2.37}$$

for $l = 0, 1, 2, \ldots$. The states with total angular momentum π are similar to NOON states.

Any initial state $|\phi_l^i\rangle$ can be brought to the target state $|\phi_l^t\rangle$ with high fidelity with the proposed protocol, and the process also works for TG gases. In Figure 2.11(a), the harmonic potential fidelities are shown to remain high for $N \leq 11$ after which they decrease due to a finite maximum height of the potential enforced by periodic boundary conditions. The fidelities can be improved by increasing the maximum trapping frequency ω_f as we demonstrate in Figure 2.11(b) where the value of ω_f is doubled and the fidelities remain high until $N \leq 21$. When using the sinusoidal potential, the fidelity drops for smaller particle numbers compared to the harmonic potential (although it also increases with ω_f), due to the lower height V_S has compared to V_H.

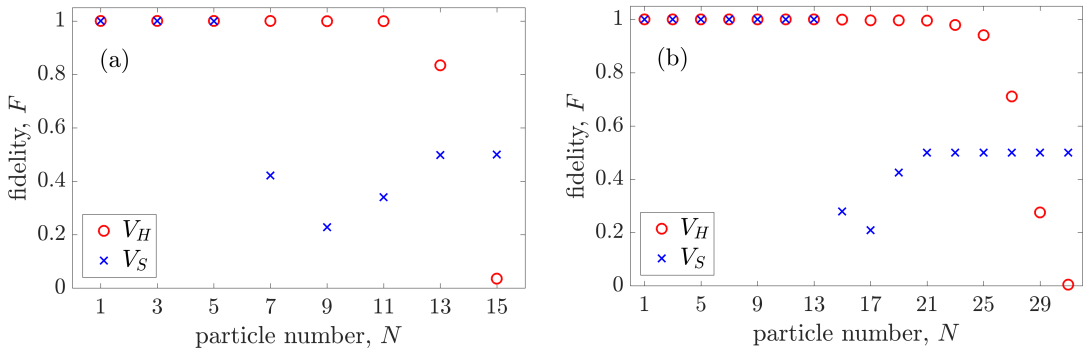

Figure 2.11: Final fidelities F of TG states of increasing particle number for the protocol shown in Figure 2.10(a) with $\Omega_f = \pi$ for V_H (red circle) and V_S (blue cross). Plot (a) shows the fidelity of the protocol with $\omega_f = 100$ and (b) with $\omega_f = 200$.

2.4 Outlook

In this chapter, a physical example was introduced to simulate a quantum engineering. Here, processes to generate NOON states in a TG gas non-adiabatically with both quantum optimal control and STA protocols. Both methods were shown to be highly effective. Such dynamical evolution techniques require time-dependent variables, such as rotation frequency or barrier height, and allowing for these dynamic operations has hitherto been a difficult task on GPU hardware. In the following chapter, we will discuss GPU hardware in-depth and also tackle this issue, along with several others noted in Chapter 1.

Chapter 3

General Purpose computing with Graphics Processing Units and the GPUE codebase

The Graphics Processing Unit (GPU) is a computing card that typically connects to the motherboard through a Peripheral Component Interconnect (PCI) slot. As the name implies, the GPU is designed to rapidly manipulate memory to create images or graphics that are sent to a display device, such as a monitor. Because individual pixels in images are independent of each other and modern computers require updating all pixels on the display device quickly, the GPU has been developed as a massively parallel computing device, capable of efficiently performing simple tasks (such as pixel generation or manipulation) rapidly by distributing the computation among many computing cores. This design methodology starkly contrasts the few, powerful cores on the Central Processing Unit (CPU), which is the default computing device on modern desktop systems. Due to this difference in hardware design, there are also several optimizations to consider when programming for massively parallel GPU devices, and several of these techniques will be covered in this chapter.

As GPU technology grew, other areas of computational science became increasingly hungry for computing power, specifically in the area of scientific computing on High-Performance Computing (HPC) systems. Historically, HPC systems were often developed as large, distributed networks of computing nodes intended for CPU-based computation. As such, these systems facilitated the development of highly parallel and distributed numerical methods to perform scientific computation.

With new, parallel algorithms being developed for HPC systems and GPU technology advancing rapidly to perform more computation in parallel to satiate the consumer demands for high-quality videos and graphics for video games and other media, it became possible to use the GPU as a scientific computing device with a new technique called General Purpose computing on Graphics Processing Units (GPGPU). Modern HPC design incorporates the GPU into each computing node, thereby increasing the throughput of the system, overall, and the fastest known supercomputer today (Summit, ORNL [1]), is almost entirely composed of GPU nodes with NVIDIA Tesla V100 cards (32 GB of available RAM), connected with NVlink and IBM's power architecture. In addition to the utility of GPGPU for scientific computing, GPU technology

has also been rapidly developed for AI and related fields.

In this chapter, I will discuss the design methodology for the hardware and software related to GPGPU before proceeding to the development of GPUE, the GPU-based Gross-Pitaevskii Equation solver, which will be used for the remainder of this work. To start, we must first look into different types of parallelism and how these affect different hardware and software practices.

3.1 Types of parallelism

Older CPU architecture with a single core was designed as SISD (Single Instruction, Single Data) according to Flynn's taxonomy [110]. This simply means that no parallelism exists in the instructions or data. Even now, most code is naïvely written as if it is to be executed on SISD architecture, even though it is rare to find such a system in a modern environment. For capable devices, there are two separate methods to parallelize computation: *Task parallelism* and *data parallelism*.

Task parallelism allows programmers to split their computation along as separate, non-interacting *tasks* or instructions, where each core performs its designated computation before moving on. On the other hand, data parallelism allows programmers to perform the same, repetitive task along a large data set by distributing threads across the data. Task parallelism is often better for dealing with a large number of specific actors, while data parallelism is often better for dealing with a large number of similar tasks on the same data, such as a large matrix. If a computing architecture allows for multiple instructions, but only a single data stream, it is considered to be MISD (Multiple Instruction, Single Data) by Flynn's taxonomy; meanwhile, if the architecture allows for multiple data streams with only a single instruction, it is considered to be SIMD (Single Instruction, Multiple Data). Most modern HPC systems are designed to be MIMD (Multiple Instruction, Multiple Data), and both task and data parallelism is exploited by developers; however, for GPU computation, data parallelism is often used more frequently.

In the realm of data parallelism, there is an extreme case where the data is *embarrassingly parallel*. Here, there could be a large matrix of data to manipulate, but no single element depends on any other. This means that when distributing computation along this matrix, we can simply assign tasks to each core without considering interactions with the rest of the data set. In this way, it is embarrassingly easy to parallelize, and hence the term *embarrassingly parallel*. As a note, matrix multiplications are embarrassingly parallel operations; however, FFTs are not [111]. As such, the SSFM is not overall embarrassingly parallel; however, because the FFTs are handled by the CuFFT library, programmers do not often need to consider task parallelism at all when developing SSFM code. Even so, understanding all the features of GPUE and its future directions requires a strong understanding of GPGPU, and this will be discussed in the following section.

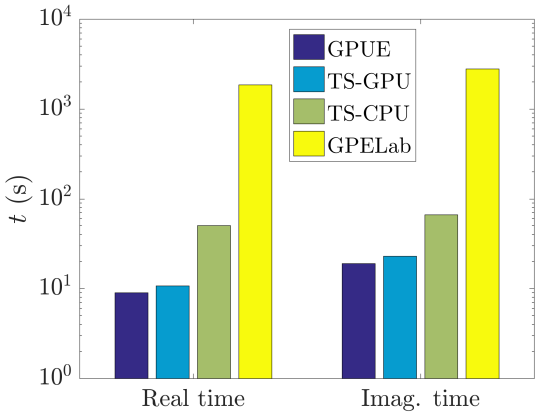

Figure 3.1: Comparison between GPUE (CUDA), Trotter-Suzuki on both GPU (CUDA) and CPU (C++), and GPElab (Matlab). Here, we see that GPUE is marginally faster than Trotter-Suzuki, but both GPU implementations are faster than the CPU-based variants. Both software packages are much faster than GPElab.

3.2 General purpose computing with graphics processing units

GPGPU programming is a relatively new development to the computing world and is generally much faster than CPU-based computation for tasks that can be easily parallelized in a SIMD fashion. Though benchmarks vary greatly depending programming languages, code quality, and intent of the software being benchmarked, our GPUE codebase is often 5 to 10 times faster than well-optimized C/C++ code and 100-200 times faster than Matlab code that is simulating the same system [112]. This is shown in Figure 3.1, where a comparison between GPUE, GPElab [113], and Trotter-Suzuki [114] are shown. These benchmarks are consistent with other GPGPU programs [115–117].

[More explicit citations]

As it is possible to massively increase the performance of certain programs by using the GPU hardware, it is important to discuss the differences between GPGPU and CPU-based computation, along with important optimizations for GPU computing that will be used throughout this work. For the remainder of this work, I will use the term *host* interchangeably with CPU and *device* with GPU.

3.2.1 Limitations of GPU computing

GPGPU and massively parallel computation are best suited for embarrassingly parallel systems and there are several problems that are poorly suited to parallelization. For example, any task that is inherently iterative (such as summation) or recursive (such as tree traversal) is not suited for parallel computation. Even so, there are methods to re-frame these problems such that they are better optimized for massively parallel devices, and these will be covered when relevant to the development of GPUE.

In addition to these algorithmic limitations, GPU cards have several notable drawbacks in terms of memory available on individual cards, which is often much less than

Listing 3.1: An example of vector addition performed in C or C++ for a, b, and c, all of size n

```
1 for (int i = 0; i < n; ++i){
2     c[i] = a[i] + b[i];
3 }
```

the amount available on the host. As such, when simulating a large system on the GPU, we often limit the resolution to what can fit onto the GPU memory. In addition the data transfer between GPUs and between the GPU and CPU through the PCI bandwidth is a slow process. Until recently, these limited the size of our simulated wavefunction with GPUE to roughly 512^3 on a single Tesla K80 card. Higher resolution simulations could be performed with more recent cards (such as the Tesla V100) or by using multiple cards; however, because it takes time to transfer data between GPUs, we preferred to use a single card where possible. At this point, it is worthwhile to fully discuss GPU hardware and software ideologies, with particular focus on areas relevant to the development of GPUE. We will discuss important methods used in the development of GPUE to overcome these shortcomings afterward in Section 3.3.

3.2.2 GPU hardware architecture

Even though several programming frameworks exist with the capability of running code on the GPU, most of these hide necessary optimizations from the user. As such, we have chosen to focus exclusively on programming frameworks that expose the hardware for software developers, such as CUDA, OpenCL, and Julia. Though the following discussion will primarily focus on CUDA, a brief discussion of OpenCL and Julia can be found in Section 3.2.3, and example code for both languages can be found in the Appendix A. For the purposes of this discussion, we will cover only the GPU memory architecture of NVIDIA GPU devices as these are the most common computing devices for HPC systems.

This topic is easiest to describe by dividing it into two parts: an introduction to the software interface as defined by the CUDA API, followed by a discussion of the memory and thread hierarchy of GPU devices. Throughout these sections, we will discuss performance tips to ensure maximum GPU utilization, memory throughput, and instruction throughput.

Introduction to CUDA software interface

The CUDA parallel computing platform bares the hardware of the GPU to software developers. This means that important elements of this programming interface will appear in subsequent sections regarding hardware limitations and performance guidelines. Much of this discussion can be found in the *CUDA C Programming Guide* [6], while other sources will be cited as necessary. Full code for this discussion can be found in the Appendix A.

Listing 3.2: An example of a vector addition kernel in CUDA

```
1 __global__ void vecAdd(double *a, double *b, double *c){
2
3     // Global Thread ID
4     int id = threadIdx.x;
5
6     c[id] = a[id] + b[id];
7 }
```

To start, let us assume a simple example where we would like to add two vectors such that $\mathbf{a} + \mathbf{b} = \mathbf{c}$. This can be done with a simple `for` loop in C, shown in Listing 3.1. In this case, we take each element with a specified ID in **a** and **b** and add them to the appropriate ID **c**. In some parallel programming models (OpenACC [118], OpenMP [119], GPUifyLoops.jl, and many others), parallelization of this method is possible by adding a macro to the start of the loop to specify that this operation is to be performed in parallel; however, this obscures GPU hardware for the user and does not always have the same performance guarantees [2]. As such, CUDA takes a slightly different approach by encouraging software developers to write *kernels*, specific to the computation at hand. An example CUDA kernel for vector addition is shown in Listing 3.2, which has a number of notable differences to the `for` loop in C in Listing 3.2. This kernel is already remarkably different than an expected function on the CPU, and it is worth comparing Listings 3.1 and **??** in detail for a better understanding of GPU hardware.

The first peculiarity appears in line 1 with the `__global__` function specifier. This is a necessary element of all CUDA kernels that specifies where and when this kernel is capable of being called. A `__global__` kernel can be called by either host (with a standard CPU function) or the device (with a GPU kernel). A `__host__` kernel is exactly the same as a CPU function and can only be called by other CPU functions. Finally, a `__device__` kernel can only be called by other `__device__` or `__global__` kernels. As a note `__global__` kernels are incapable of returning vectors or other variables, and must instead mutate the variables, themselves. This is why the `__global__ vecAdd(...)` kernel does not return c, but instead assumes it is a pre-allocated variable.

Another peculiarity appears on line 6, where the addition, itself, occurs. Though there was a necessity for a `for` loop in Listing 3.1, there does not seem to be one at all in Listing 3.2. This is because the GPU is handling the parallelism behind the scenes on line 4 with the `int id = threadIdx.x` command. In this line, we are identifying which element of the array we are operating on with the CUDA-specific `threadIdx.x` variable.

Here, each *thread* is an individual instructional element acted on in parallel with other threads in the same *block*, which is further subdivided into *grids*. All threads in the same block have a *shared memory* resource, while all three have access to *global memory*. In general, it is important to use shared memory when possible, as it has a lower latency than global memory, and this will be discussed further in subsequent

Listing 3.3: An example of a vector addition kernel in CUDA using blocks and threads, and ensuring no computation happens beyond the size of the array, n.

```
__global__ void vecAdd(double *a, double *b, double *c, int n){

    // Global Thread ID
    int id = blockDim.x * blockIdx.x + threadIdx.x;

    if (id < n){
        c[id] = a[id] + b[id];
    }
}
```

sections. As a note, threads often work without feedback from other threads; moreover, it may be necessary to stop thread execution until all other threads have caught up. This can be done with the `__syncthreads()` function in CUDA.

Threads, blocks, and grids are all `dim3` variables with x, y, and z attributes. The way in which these threads and blocks are allocated are defined by the user before kernel execution. Often times, a thread number of 1024 is chosen, and the number of blocks is decided based on the size of the input array. If there are more elements to compute than threads in a block, we then need to use the `blockDim.x` and `blockIdx.x` variables to access the appropriate threads for computation. All threads are indexed as one-dimensional vectors even in a multidimensional space, as shown in Figure 3.2. Even though threads are rarely acted on sequentially, the thread ID has important ramifications that will be discussed further with other performance tips.

As another note, CUDA will execute code on unallocated memory if one does not tell it to otherwise. As such, if we had failed to set the number of threads in our block to the number of elements in our array in Listing 3.2, the kernel would exhibit undefined behavior. For this reason, we need to take into account potential out-of-bounds computation. If we take the above example of vector addition, assuming that the thread count is higher than can fit in one block and take into consideration potential out-of-bounds behavior, the kernel would instead look like Listing 3.3.

Finally, we need to discuss how software developers call these CUDA kernels in host code. This requires the developer to allocate space on the device for use in the CUDA kernel via a `cudaMalloc(...)` command. Often times, arrays on the host must also be established and transferred to the device with `cudaMemcpy(...)` functions as well. In addition, the kernel must be configured before running with `<<<grid, threads,...>>>`. When everything is considered, the host code might look like what is shown in Listing 3.4

All said, vector addition is often known as the "Hello World!" of GPU programming as it is the first application that shows the parallelism of GPU devices; however, even here, we begin to see a distinction between users and developers. As more complex software is developed, it becomes more important to write software in such a way that users do not directly interface with CUDA code, and the full ramifications of this will be discussed in later in this chapter. In addition, this discussion highlights the important

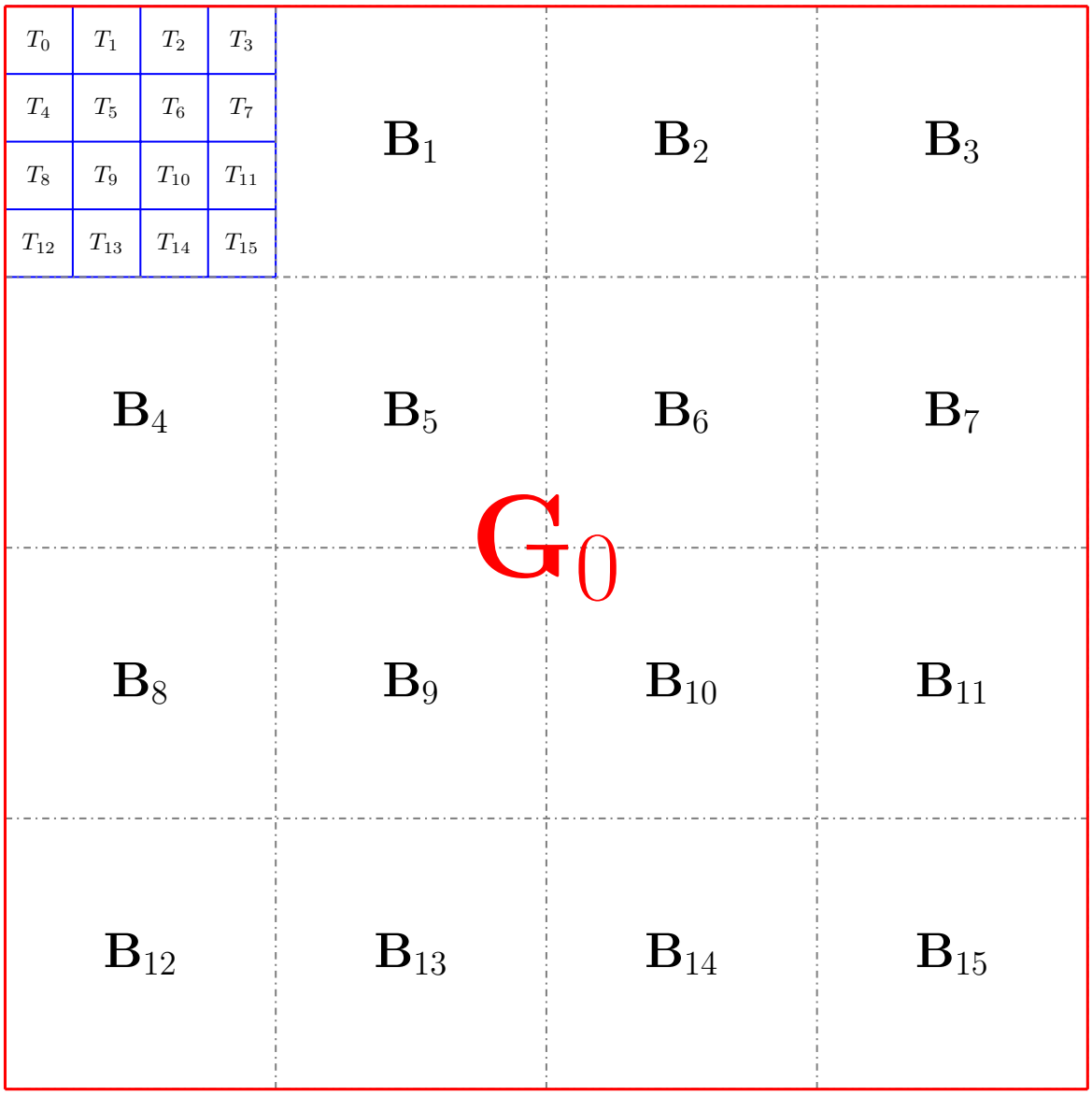

Figure 3.2: Each grid is subdivided into multiple blocks, which is further subdivided into threads for computation. Each thread has a specified ID, which acts as a one-dimensional array, even in a two or three-dimensional system. Here, all areas outlined in red have access to global memory, and any area outlined in blue has access to shared memory. [slight modification necessary to show 1D indexing!]

Listing 3.4: An example of host code to run Listing 3.3.

```
int main(){

    int n = 1024;

    // Initializing host vectors
    double *a, *b, *c;
    a = (double*)malloc(sizeof(double)*n);
    b = (double*)malloc(sizeof(double)*n);
    c = (double*)malloc(sizeof(double)*n);

    // Initializing all device vectors
    double *d_a, *d_b, *d_c;

    cudaMalloc(&d_a, sizeof(double)*n);
    cudaMalloc(&d_b, sizeof(double)*n);
    cudaMalloc(&d_c, sizeof(double)*n);

    // Initializing a and b
    for (size_t i = 0; i < n; ++i){
        a[i] = i;
        b[i] = i;
        c[i] = 0;
    }

    cudaMemcpy(d_a, a, sizeof(double)*n, cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, b, sizeof(double)*n, cudaMemcpyHostToDevice);

    dim3 threads, grid;

    // threads are arbitrarily chosen
    threads = {100, 1, 1};
    grid = {(unsigned int)ceil((float)n/threads.x), 1, 1};
    vecAdd<<<grid, threads>>>(d_a, d_b, d_c, n);

    // Copying back to host
    cudaMemcpy(c, d_c, sizeof(double)*n, cudaMemcpyDeviceToHost);

    // Check to make sure everything works
    for (size_t i = 0; i < n; ++i){
        if (c[i] != a[i] + b[i]){
            std::cout << "Yo. You failed. What a loser! Ha\n";
            exit(1);
        }
    }

    std::cout << "You passed the test, congratulations!\n";

    free(a);
    free(b);
    free(c);

    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);
}
```

distinction between host and device code, including the concept of threads and blocks, shared memory, and the ability to transfer data from the host to the device. For now, we will continue this discussion by moving to GPU hardware, focusing on thread and memory hierarchy.

Discussion of GPU thread and memory hierarchy

Every time host code invokes a CUDA kernel call (as shown in Listing **??**), the data is mapped to a scalable array of multiprocessors on GPU hardware. Multiple blocks might be distributed to the same multiprocessor, but blocks are always distributed contiguously. Each multiprocessor is designed to execute hundreds of threads in parallel by using a unique SIMT (Single Instruction, Multiple Threads) architecture, which is similar to SIMD and can be used as such for most cases; however, there are performance benefits to optimizing instruction-level parallelism at the thread level. Each multiprocessor distributes its parallel processes into *warps*, which are units of 32 threads that execute a single common operation at a time. Notably, the way a block is distributed into warps is always the same, so it is important to ensure that the input data is in powers of 32 to avoid wasting unnecessary computation. Outside of this, developers can often ignore SIMT behavior as long as they do not allow threads in a warp to have separate operations.

Now we will turn our focus to memory within GPU devices. There are three forms of GPU memory that are useful for most applications of GPGPU for scientific computation: Global memory, Shared memory, and texture memory. As described above, global memory is a shared between all grids, blocks, and threads and is considered to be the slowest memory bank. As such, whenever a warp accesses global memory, it tries to perform as few accessing operations as possible, which is made easier if the warp needs to access contiguous memory blocks. If the warp is required to access non-contiguous blocks, more accesses will be necessary and thus performance will take a relatively large hit. For this reason, it is important to make sure all data accesses are *coalesced*, which ensures that the warp will access consecutive elements as depicted in Figure 3.2.

For optimal memory throughput, shared memory is an essential tool to understand and use appropriately. As described, shared memory is on-chip memory that is shared between all threads in a block. The amount of shared memory available is hardware-dependent and configurable on kernel execution. In general, it is worthwhile to transfer data with a large number of accesses to shared memory for performance. Shared memory is split into several memory banks which can be accessed simultaneously. If two memory accesses are required of the same bank, there will be a conflict and the operation can no longer be performed in parallel. It is sometimes necessary to pad variables to prevent bank conflicts from occurring [120].

Of the three types of memory mentioned, texture memory is the least-often used and is primarily on the GPU for graphics computation and focuses on performance for two-dimensional structures. Texture memory has a relatively long write time, but is quick to read. It is also faster than global memory for non-coalesced access patterns and therefore can be useful for certain stencil-based calculations. Unfortunately, it uses single-precision values and thus will not be used for the remainder of this work.

In addition to appropriate usage of memory on GPU architecture, it is also essential to minimize data transfers between the device and host and even between devices in multi-GPU setups. The data transfer between the host and device or between devices must send data through the PCI slot on the motherboard, which is a slow operation. For data transfer between devices, this transfer time can be slightly alleviated on Power architecture where NVlink technology can directly transfer data from device to device [121], but the data transfer between devices will still likely be the slowest part of the computation. In addition, CUDA-aware MPI for multi-GPU setups may add an huge burden on software development time [122, 123]. As such, developers often try to keep all of their computation on a single card, if possible, and several optimization strategies are used when multiple GPU cards are needed. These strategies will be covered on a case-by-case basis as they arise in this work.

As a final note, one optimization strategy for CUDA code that will not be discussed in-depth in this work is the maximization of instruction throughput. This simply means that programmers can increase the number of instructions performed over a specified period by trading precision for speed and minimizing thread synchronization. Because we are performing high-precision superfluid simulations, we cannot perform this trade-off. An important caveat here comes from conditionals, like `if` and `switch` statements. Here, programmers need to be careful not to accidentally cause the operation executed on threads in a warp to diverge.

3.2.3 Comparison between various languages for GPGPU computation

As one might expect, specialized programming languages are necessary to write code that compiles and runs on GPU architecture. There are several known libraries to extend modern programming languages such as Matlab, python, and C++ to GPU devices; however, we will limit this discussion to common programming methods that allow fine-grained control of GPU memory and could be used for the development of GPUE. We will briefly discuss the advantages and disadvantages of three competing languages here: CUDA, OpenCL, and Julia, and as a simple example, vector addition in these languages is shown in Appendix A.

CUDA

CUDA is a computing API provided by NVIDIA for interfacing with NVIDIA GPUs and is the industry standard for GPGPU programming. CUDA is primarily limited by the NVIDIA-specific hardware it runs on, and although NVIDIA currently produces the most common GPUs for GPGPU programming, AMD GPU devices are also available and often cheaper for a similar level of computational power. In addition, CUDA support has recently ceased for Mac OS systems as NVIDIA cards are no longer bundled with current generation Mac computers, so CUDA code can only be used on Windows and Linux devices.

GPUE was written entirely in CUDA; however, due to the aforementioned limitations, there has been some consideration to re-writing the software in OpenCL or Julia.

OpenCL

Though CUDA is the industry-standard for GPGPU programming, OpenCL (Open Compute Language) is competitive in terms of performance and has the benefit of being compatible with NVIDIA and AMD GPU devices. OpenCL is also completely open-source and works as additional libraries to C or C++, which allows developers to compile OpenCL code with traditional compilers like `gcc` or `clang`. OpenCL has nearly identical structure to CUDA with slightly more verbose syntax, and thus provides all necessary functionality to develop and maintain scientific software. In addition, compute kernels are compiled at run-time, meaning that users can potentially modify kernels without recompiling the code. This could be a huge boon for developers writing software for users who may need to quickly simulate a slightly modified system. Unfortunately, OpenCL has a rather cumbersome interface and has less peripheral support than CUDA. As such, it is rarely used for scientific computing software.

In the end, although OpenCL does provide the ability to more easily construct dynamic kernels, the increased engineering time necessary to write software in OpenCL is often not worth the cost; however, further advances in compiler design for heterogeneous architecture has been made in the past few years [124], which has provided the unique opportunity for computer scientists to write maintainable and fast code in new languages, like Julia.

Julia

Julia is a new language to scientific computing, but boasts promising results and claims to be as usable as python, but as performant as C [125]. This is a huge boon for maintainability. In addition, Julia's runtime is comparable to CUDA C for GPGPU computation and allows for similar hardware optimizations [4, 126], while also allowing users to edit the compiler implementations at will. This is an important point that will be discussed in more detail in Section 3.3.2.

In addition, because Julia is much easier to write than C for new programmers, GPU-based Julia code could allow developers to provide fast, efficient code with a usable interface for scientists and engineers. The trade off between performance and readability in programming has been described as the "two-language" problem, as most scientific computing solutions to this point have required using two languages: a fast language for the back-end and a readable language for the user interface. Julia succeeds in bridging the gap between the languages, effectively solving the two-language problem and allowing scientists and engineers to write efficient code that is even compilable on the GPU. For these reasons, we have begun porting our CUDA code to Julia, as it will lead to simpler and more maintainable code in the future. This will be further discussed in the conclusion of this work, Chapter 5.4.

3.3 Introduction to the GPUE codebase for n-dimensional simulations of quantum systems on the GPU

At this point, all the motivation and background necessary has been provided to discuss GPUE, the GPU-based Gross-Pitaevskii Equation solver. This codebase will be used for all remaining simulations performed in this work and its development has also inspired the development of other computational libraries such as the `DistributedTranspose.jl` package, which will also be discussed in Section 3.4. Some additional information on prior development of GPUE can be found in other sources [127]. For this section, I will first describe the FFT optimizations used in GPUE, followed by additional features necessary for dynamic simulations on GPU architecture.

3.3.1 FFT optimization

As mentioned in Chapter 1 and previously in this chapter, the SSFM is primarily limited by the complexity of the FFT operations. For a three dimensional simulation with gauge fields in the $\hat{x}$, $\hat{y}$, and $\hat{z}$ directions, one set of global FFTs and three sets of one-dimensional FFTs must be performed. This is equivalent to two three-dimensional FFT operations, which become much more complex when scaling to multiple GPU devices [111]. The CuFFT API provides an option for computing FFT's on separate batches of an array in GPU memory with the `cufftPlanMany(...)` command; however, if this command is repurposed for one-dimensional FFT operations, it does not provide the necessary functionality for FFTs across the $\hat{y}$ or $\hat{z}$ dimensions. With the CuFFT library, all FFT operations performed with this plan must follow an indexing pattern, such that

```
input[b*idist + x*istride]
output[b*odist + x*ostride]
```

where `b` is the batch ID, `x` is the element ID, `idist` and `odist` are the distances between batches for the input and output array, respectively, and `istride` and `ostride` are the strides between consecutive elements for computation with the input and output array, respectively. If data is transferred to the GPU, it must be re-indexed as a one-dimensional array, such that

```
array[i,j,k] = array[i + j*xDim + k*xDim*yDim]
```

where `i`, `j`, and `k` are iterable variables in the $\hat{x}$, $\hat{y}$, and $\hat{z}$ directions, and `xDim`, `yDim`, and `zDim` are the dimensions in $\hat{x}$, $\hat{y}$, and $\hat{z}$, respectively. As such, it is not possible to use the `cufftPlanMany` functionality to perform one-dimensional FFTs in $\hat{y}$ and $\hat{z}$; however, if we increase the number of batches to `xDim*yDim*zDim`, set the distance between each stride to 1, and assume the stride between each element is `xDim*yDim`, we can recreate the functionality of the FFT in the $\hat{z}$ direction. For the $\hat{y}$ FFT operations, we need an external loop that iterates over each xy slab, performing `xDim` operations on each slab, and this greatly hampers performance.

[Not sure if images will help here]

With this considered, only one-third of the necessary FFT operations are appropriately coalesced in memory for three dimensional simulations. Because FFTs are global

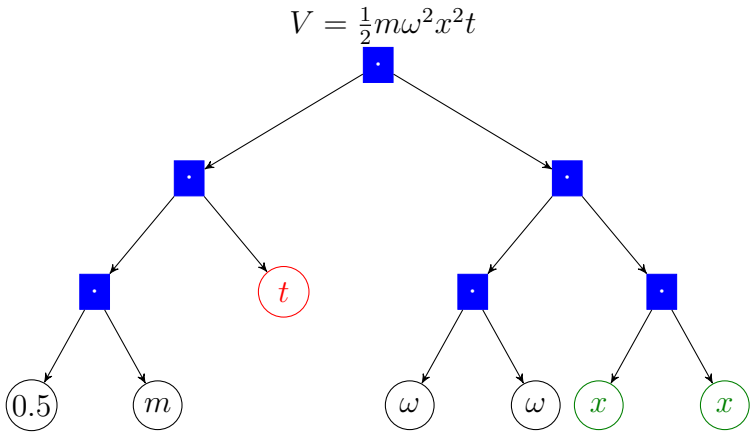

Figure 3.3: Example of expression tree for $V = \frac{1}{2}m\omega^2x^2t$. Blue, filled nodes are operations, leaf nodes are variables, time has been highlighted in red, and spatially-dependent variables are in green. This visualization was modified from a form provided by Xadisten during a Twitch livestream.

operations that are best performed on contiguous chunks of memory, multi-GPU simulations with the SSFM are even less optimal. This has motivated the development of other packages to allow for memory coalescence with FFT operations, such as the distributed transpose, which will be described in Section 3.4. Even though the three-dimensional FFT operations are the biggest bottleneck in the GPUE codebase, it is not easy to avoid usage of the `cufftPlanMany(...)` operation while still using CUDA. Next, we will focus on another feature that was inhibited by the CUDA framework, but is nevertheless possible: methods to enable dynamic quantum engineering with expression trees.

3.3.2 Dynamic field input and output in GPUE with expression trees

As mentioned in Chapter 2, quantum engineering typically requires some form of time-dependent variables, along with evolution in real time. This means that the user must be able to input a time-dependent equation to GPUE. Because we chose to write GPUE in CUDA, there is no straightforward method for the user to input time-dependent fields without recompiling the source code and modifying CUDA kernels at will, which is unnecessarily cumbersome for the user. As such, we have provided a method for users to input the fields of their choosing as strings, which will be transpiled into an array of operations to perform on the GPU through expression trees, which are similar to Abstract Syntax Trees (ASTs) in compiler design [128, 129].

An example of an expression tree can be seen in Figure 3.3. These are evaluated depth-first to follow the traditional order of operations. With this method, a user can type in a string, like `"V = m*omega*omega*x*x*t"`, and this will be parsed into a set of operations to be performed on-the-fly by the GPU. After parsing user-provided equations, we designate certain leaf nodes that are either spatially or temporally dynamic. In the case of spatially dynamic variables (`x`, `y`, and `z`), we pull values from constituent

vectors based on their `threadIdx.xyz` values, and for any equation that is dependent on `t`, we pull upon a stored `time` variable. This operation necessitates the usage of a dictionary data structure to hold all variables in some fashion, which inhibits host performance; however, because the bulk of the computation is performed on the GPU, this does not significantly impact GPUE performance, overall. On the GPU, each necessary variable can be stored in a shared memory buffer, and because we are replacing the embarrassingly parallel element-wise matrix multiplications with these expression trees, the performance is not severely impacted; however, because parsing expression trees is an inherently iterative process, the longer the expression, the less optimal using this method is. Ultimately, more work could be done in the future to maximize instruction throughput with our implementation of GPU-accelerated expression trees.

Not only do expression trees allow for STA and quantum optimal control methods to be used with GPUE, but they also eliminate the need to store any operators in GPU memory, effectively increasing the available memory by a factor of 5 for each three-dimensional simulation, as V, K, A_x, A_y, and A_z no longer need to be stored. This allowed us to perform higher-resolution simulations and could allow for dynamical turbulence studies in the future; however, in order to scale beyond this limit, either multiple GPUs must be used or we must find some way to compress the wavefuntion. This will be discussed further in Section 3.3.5. As a final note, this feature can be implemented easier in other GPU frameworks, such as OpenCL and Julia.

The implementation of expression trees in GPUE effectively decreases the memory footprint of our simulations and allows for dynamical studies on the GPU; however, dynamical studies also require a large amount of fileIO. Though we had initially considered using the Compressed Split-Step Fourier Method (CSSFM) **??** to compress the size of our wavefunction. The CSSFM attempts to compress the wavefunction into a basis where it is sparse and then performs the SSFM on this compressed wavefunction with operators that have been transformed into the appropriate spaces. In the original work by Bayindir [17], one-dimensional simulations of soliton dynamis were performed on a downsampled grid and reconstructed to higher resolution with compressed sensing [**?**]. This provided a considerable improvement in both performance and memory usage for a wide range of potential resolutions. After attempting to use this method with GPUE, we ultimately found this method to be unsuitable for n-dimensional vortex simulations, because compressive sensing does not provide adequate compression for simulations of this nature.

As such, we instead reworked GPUE fileIO with HDF5 to limit the file size. That said, for two-dimensional vortex simulations, we often do not need to output the entire condensate wavefunction, but can instead output only the vortex locations. Methods of vortex tracking and highlighting will be discussed in the following section.

3.3.3 Vortex tracking and highlighting

In order to analyze the motion of vortices in a superfluid system, some form of vortex tracking must be implemented, and the current vortex tracking methods used in GPUE for two dimensions can be found in prior work [127] or the GPUE documentation [130]. It is important to describe two dimensional vortex tracking first before continuing to three dimensional vortex analysis, which is a much more complicated process.

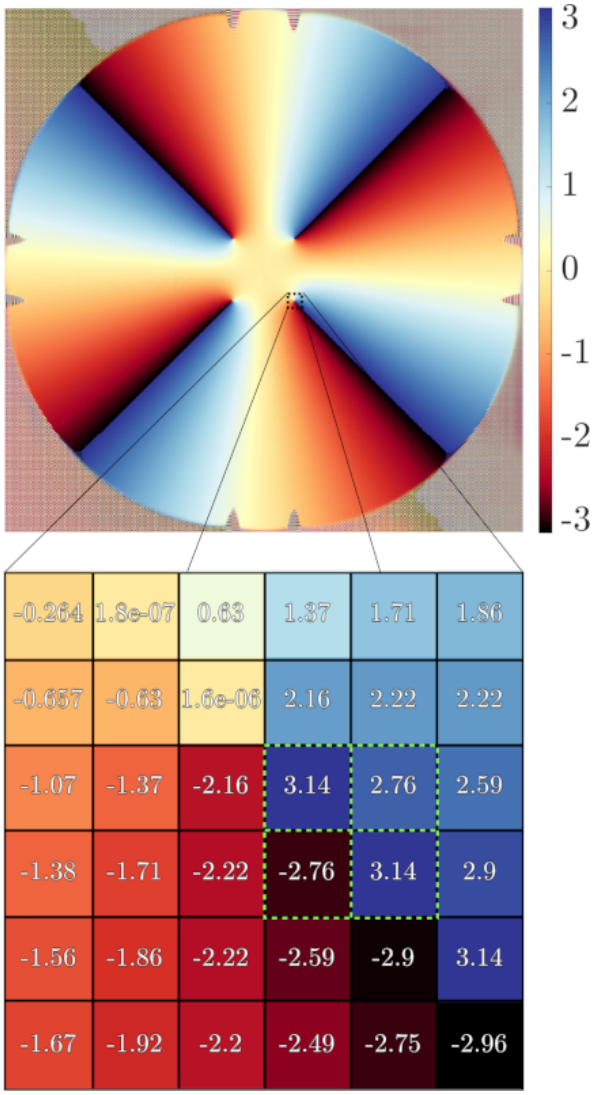

Figure 3.4: An example phase plot of a condensate with four vortices. The inset shows the values of the grid around each vortex location and highlights where the sum is 2π for vortex tracking.

At a first glance, one might assume that vortices are located at areas of low density in a superfluid system; however, this is not always the case. Because our condensate is often inhomogeneously trapped and does not necessarily extend to the edges of the simulated domain, there will be large areas of zero density outside our condensate. In addition, sound waves and other perturbations with minimal density can occur. As such, locations of low density should only be used as educated guesses as to where actual vortices are located, but should not be used as the final predictor.

Instead, the phase can be used to uniquely identify vortex locations, as shown in Figure 4.1. In the highlighted region, all elements sum to a value of 2π. In this way, vortex tracking essentially becomes a straightforward task of locating all the 2π phase windings in the simulated domain via minimization routines where we attempt to find any four grid elements whose sum is 2π. This process also necessitates a mask

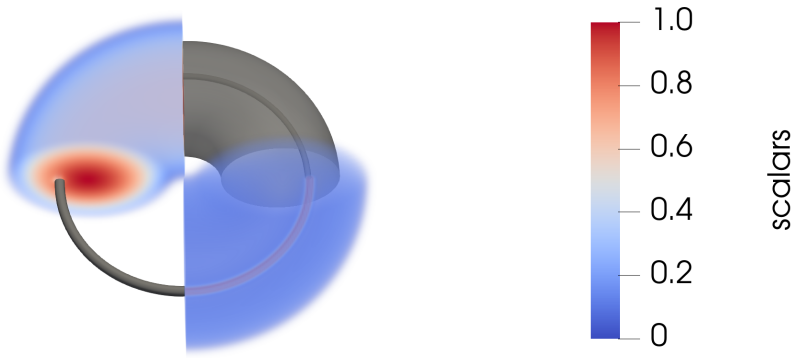

Figure 3.5: An example of vortex highlighting with a Sobel filter. The upper left quadrant is the superfluid density with no modifications and the upper right quadrant is an isosurface of the density with an opacity of 0.6. Note here that if there is no opacity set, it is not possible to see the vortex because it is obscured by the outside boundary of the BEC. The lower right quadrant is the superfluid density after Sobel filtering and the bottom left quadrant is an isosuface on the Sobel filtered density. Here, we can easily create isosurfaces of vortices that would be occluded when using the density, alone. The scale varies depending on whether it is coloring the normalized wavefunction density or the filtered density.

for regions outside of the BEC domain, and further discussion on how to refine this position can be found in previous work [127, 130].

In three dimensions, vortices are no longer confined to a plane and can extend in any direction, so long as the vortex lines either end at the end of the superfluid or reconnect in the form of vortex rings or more complicated vortex structures. This is a much more difficult problem which does not have many solutions in superfluid simulations where the superfluid does not fill the simulation domain. The current state-of-the-art solution has been proposed by Villois *et al* [131], and requires finding density dips in the superfluid as initial guesses as to where a vortices might exist. From there, a vorticity plane is determined and the entire vortex is discovered by moving perpendicularly to the vorticity plane at each gridpoint. This is a tedious and time-consuming process that does not lend itself well to GPGPU computation without communication between the host and device. As such, we are currently seeking a more computationally efficient method for tracking vortices in three dimensions.

As our system do not necessarily fill the contents of our simulation domain, the proposed method will not work without some modification. We could still use the method if we have some understanding of the trapping geometry; however, as we discussed in Chapter 1, this is not always the case with gauge fields.

As such, instead of focusing on vortex *tracking*, we have instead implemented a simple vortex *highlighting* scheme for three dimensions. This can be done with a Sobel filter on the condensate density, and can easily create crisp visualizations like those found in the computer graphics literature [132]. An example of a vortex highlighted wavefunction density, along with an isosurface of both the density and the highlighted

density can be seen in Figure 3.5. In this figure, we show that the density after being Sobel filtered can be more easily used to isolate vortex structures without the background BEC. Though it would be possible to use further edge detection methods, such as the Canny edge detector [133], this would add a significant computational overhead and thus was not implemented in the current work. The problem of efficiently tracking vortex skeletons in three-dimensions is a difficult problem that requires further study; however, vortex highlighting is enough for most three dimensional vortex simulations. In Chapter 5, we show and example simulation where vortex highlighting has been used to determine the vortex isosurfaces.

3.3.4 Energy calculation for superfluid simulations

As discussed in Chapter 1, energy calculations can play an essential role in SSFM simulations and can be used to help understand vortex dynamics in certain simulations. More importantly, energy calculations lie at the heart of convergence criteria for imaginary time propagation. Essentially, in order to avoid unnecessary computation, many SSFM implementations will cease simulating the system in imaginary time when the change in energy every timestep drops below a certain threshold value. Though we have this option available in GPUE, it is not a native feature for at least three important reasons:

1. Certain systems, such as large vortex lattices with high rotation, have a seemingly degenerate ground state with different vortex configurations [54, 55, 127].

2. GPUE is often run on a computing cluster where the maximum simulation time is set before-hand. For this, the user must be able to estimate the duration of their simulation, and this is not straightforward if imaginary-time propagation finishes at an unknown time.

3. The energy calculation is memory and operation-intensive and requires at least one additional object of the size of the wavefunction to be created and stored on GPU memory.

The first and second of these are somewhat self-explanatory, but the third requires further elucidation.

Energy calculations in GPUE are essentially composed of the following operation,

$$E = \langle \Psi | \hat{\mathcal{H}} | \Psi \rangle \tag{3.1}$$

The first problem with this operation is that it requires a summation for the final energy value, and as discussed, this is a poorly-suited problem for GPU hardware. Even though we have a robust implementation of parallel reduction, this is still a slow process. The next problem comes from the nature of the Hamiltonian, itself. As described in Chapter 1, the Hamiltonian is essentially composed of three separate components for vortex simulations:

$$\hat{\mathcal{H}}_v = V_0 + g|\Psi|^2 + \frac{m\mathbf{A}^2}{2} \tag{3.2}$$

$$\hat{\mathcal{H}}_p = \frac{p^2}{2m} \tag{3.3}$$

$$\hat{\mathcal{H}}_{pv} = p\mathbf{A} \tag{3.4}$$

where $\hat{\mathcal{H}}_v$, $\hat{\mathcal{H}}_p$, and $\hat{\mathcal{H}}_{pv}$ are the Hamiltonians in position-space, momentum-space, and mixed-space, respectively. These operations can be considered with expression trees; however, for three-dimensional simulations they still require either a set of forward and inverse FFT's or a derivative function with fixed stride along with the parallel reduction operation. This ultimately amounts to the same number of operations required for a single step of imaginary-time evolution; however, because we do not want to influence the simulated wavefunction, it requires at least one additional allocation of a wavefunction-sized array. Due to the computational time required for each energy calculation, we request users to input the set of timesteps they would like to compute the energy for before-hand. In addition, at certain points, it is impossible to run GPUE with the energy calculation, simply because there is not enough memory available on the device.

Though finding the energy of the wavefunction is a useful feature for certain simulations, it should not be used regularly for memory-limited tasks or tasks that should be performed quickly. Even so, for most applications of GPUE on HPC environments, there should be no problem running the energy-calculation alongside the simulation, itself.

3.3.5 Future direction and multi-GPU development

At this point, I would consider GPUE to be close to feature-complete. It is capable of simulating a wide-variety of quantum systems and can even perform dynamic quantum engineering studies with minimal fileIO. Though more work can be done to maximize instruction throughput, this will not significantly improve the performance of the code because we rely heavily on double-precision.

The next logical step for GPUE development is scaling to larger simulations. This means that we either need to increase the number of GPUs used for the simulation or decrease the size of the wavefunction, itself. Though the CSSFM method **??** should allow for the latter, we ultimately found it unsuitable for our purposes. As such, we are now attempting to scale GPUE to multiple GPU devices; however, as I hope to have impressed by now, this is not a trivial task. Even though the CuFFT library can support multiple GPU devices, this comes with a huge performance penalty, especially for the `cufftPlanMany(...)` functionality.

For this reason, we have begun the development of GPUE.jl, which has similar performance to GPUE, but is currently lacking the expression tree functionality. Once GPUE.jl is at feature parity with GPUE, we will then focus on it as the primary future direction of GPUE. Ultimately, the Julia language allows us to develop GPUE in a much more maintainable fashion, and also allows for us to access GPU hardware in

a more convenient way. Because of this, we have already begun development on the tool that should allow for multiple GPU simulations with GPUE to be possible: the DistributedTranspose.jl package.

3.3.6 Similar software packages

[Discuss other software here, like XMDS]

3.4 DistributedTranspose.jl

At its heart, the two-dimensional transpose is a straightforward operation consisting of a simple swapping of all row and column elements. Unsurprisingly, this is a rather difficult task to ensure memory coalescence, and the current recommended method for transposes on massively parallel devices requires heavy use of shared memory tiles to speed up the process [120]. In this case, it is possible to perform a two-dimensional transpose at the same performance as a simple copy, so long as the operation is out-of-place in memory. The transpose becomes even more difficult to create when we wish to transpose large three-dimensional matrices, potentially spanning across multiple GPU devices, while also ensuring the operation is in-place in memory.

In principle, there are three types of three-dimensional transposes:

Simple Copy A benchmark for other transpositions,

$$A_{xyz} \to A_{xyz} \tag{3.5}$$

Involution A transpose where a two-dimensional transpose is operated on a three-dimensional data structure,

$$A_{x\mathbf{yz}} \to A_{x\mathbf{zy}} \tag{3.6}$$
$$A_{\mathbf{xy}z} \to A_{\mathbf{xy}z} \tag{3.7}$$
$$A_{\mathbf{x}y\mathbf{z}} \to A_{\mathbf{z}y\mathbf{x}} \tag{3.8}$$

Rotation A fully three-dimensional transpose,

$$A_{\mathbf{xyz}} \to A_{\mathbf{yzx}} \tag{3.9}$$
$$A_{\mathbf{xyz}} \to A_{\mathbf{zxy}} \tag{3.10}$$

It has been shown that for out-of-place transpositions, it is possible to perform all of these operations as efficiently as a a simple copy; however, in-place rotational transposes can only attain 60% of the performance based on currently known methods [134]. In addition, distributed transposes of this nature have not been discussed except for an out-of-place array [135].

At its current state, the DistributedTranspose.jl package is able to do out-of-place, distributed transposes; however, when feature-complete, it should allow for the implementation of new distributed methods for such computation. It should be mentioned

that this package has potential to be used by many other software packages that require using spectral methods on multiple GPU devices. Often, HPC engineers prefer finite-element or difference methods when computing largescale fluid flow because these methods scale much better across distributed systems; however, with an appropriate distributed transpose methods, it might be more optimal to perform spectral simulations in certain regimes over other methods.

[I will probably move this section and GPUE future dev to conclusions / future work... But the research for this area should be completed by the time of the thesis defense. I am not sure whether to leave it in or not.]

Chapter 4

Vortex analysis of two-dimensional superfluid systems

Here, we show an application of the GPUE codebase by simulating a two-dimensional chaotic system with few vortices. In addition, this chapter intends to display the dependence of post-processing metrics such as the Lyapunov exponent to dynamical studies of superfluid systems.

As a note, the physics simulated in this section is two-dimensional, so I would like to begin this chapter with a disclaimer about the dimensionality of the system we will be simulating. In principle, all real-world physics is three-dimensional, but just as a one-dimensional cigar-shaped BEC can be created, a pancake-like geometry can also be constructed by increasing the trapping frequency in the $\hat{z}$ (perpendicular) direction with respect to the $\hat{x}$ and $\hat{y}$ (transverse) directions. With this geometry, we can assume that the condensate is in the ground state along the $\hat{z}$ dimension and rewrite the wavefunction as $\Psi(\mathbf{r},t) = \Psi(x,y,t)\phi(z)$, where $\Psi(x,y,t)$ is the wavefunction in the transverse plane and $\phi(z) = (m\omega_z/(\pi\hbar))\exp(z^2 m\omega_z/(2\hbar))$ is the ground state along the $\hat{z}$ dimension. By integrating over $\hat{z}$, we also find that the interaction strength is modified for a two-dimensional condensate to be

$$g_{2D} = g\sqrt{\frac{m\omega_z}{2\pi\hbar}}. \tag{4.1}$$

With these changes, we can simulate two-dimensional settings with the GPUE codebase [11, 54, 55, 127]. This chapter will apply several of the techniques mentioned in Chapters 1 and 3 to a rotating two-dimensional BEC system for a small number of vortices and was published in *Phys. Rev. Fluids* 4(5):054701, 2019 **??**.

4.1 Chaotic few-body vortex dynamics in rotating Bose–Einstein condensates

Chaotic evolution is typically identified by a significant divergence in trajectory based on a small change in the initial conditions [136], and it is possible to find such an environment in turbulent flow [137, 138]. For classically turbulent flow, the degree of

chaos depends on the Reynolds number [139]; however, the nature of quantum turbulent effects is still an active area of research [71]. Because superfluid vortices are have quantized winding numbers, they can be considered less complex when compared to their classical counterparts, there has also been significant interest in the differences between classical and quantum turbulence [140–143]. In spite of the differences between the fluid models, vortex dynamics in superfluid systems are remarkably similar to classical point-vortex models and key features of classical turbulence, such as the Kolmogorov spectrum have been shown to exist for large, turbulent, quantum systems [144–147].

It is known that it is not possible to excite quantum chaos in large vortex lattices, as these systems have been proven to be stable to external perturbations [127]. For this reason, we wish to probe quantum chaos with a small number of vortices, such that quantum turbulent effects are not excited. Chaotic, few-vortex systems have been studied previously by Aref and Pomphrey [148–150], who showed that quantum chaos can be excited in systems with as few as four vortices in an infinite plane [148]. Unlike classical chaos, the onset of quantum chaos seems to appear with fewer vortices present, and few-vortex systems have been explored experimentally for two, three, and four vortices in harmonically trapped BECs [143]. When analyzed with a reduced Hamiltonian approach, harmonically trapped BECs seem to exhibit chaotic effects with as few as three vortices, two co-rotating vortices and an anti-vortex rotating in the other direction [141, 142].

Experimentally, it is now possible to detect vortex circulation [151] and image vortices in-situ [152]. It is also possible to probe vortex dynamics at different times within a single experiment [153, 154]. Because quantum vortices are simple and BECs are highly controllable experimental systems in two-dimensions, there has been significant interest in two-dimensional quantum turbulent systems as well [70, 155]. Additional effects, such as the Kármán vortex street [156] and Onsager vortex clusters [157, 158] have already been shown to exist experimentally.

Because chaotic events require very small changes in the initial conditions of the system, we require a system with well-controlled initial condistions. For this, we will start with a small vortex lattice of four vortices, and then create a defect in this lattice using phase imprinting, as described in Chapter 1. This process will controllably induce chaotic vortex dynamics in an experimentally feasible way. We also show that the chaotic dynamics are enhanced by the close approach of vortices when oscillating in the trap. By using phase imprinting in this way on a larger number of vortices in a vortex lattice, it might be possible to induce chaotic events there as well, which might enable studying a set of vortex trajectories that is chaotic at specific points, but stable overall.

4.2 Model

For this study, we use a condensate with $N = 10^6$ ^{87}Rb atoms with an s-wave scattering length of $a_s = 4.76 \times 10^{-9}$ in a pancake geometry with typical trapping frequencies of $(\omega_\perp, \omega_z) = (2\pi, 32\pi)$Hz. Here, the effective two-dimensional interaction strength is $g = 6.8 \times 10^{-40}$ m^{4}kg/s^2. These simulations were performed on a grid of $2^{10} \times 2^{10}$

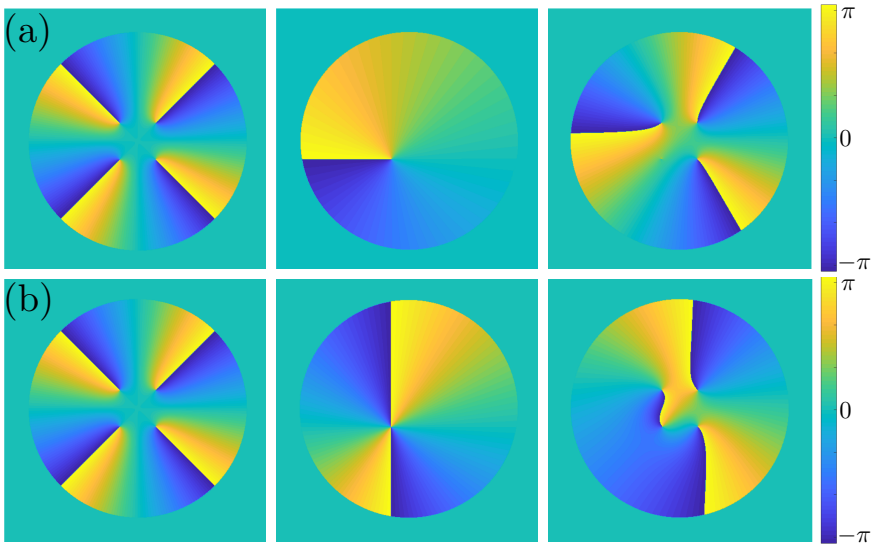

Figure 4.1: Phase distribution of the initial four co-rotating vortex system is shown at the left. In the center is the applied phase mask of -2π for (a) and -4π for (b), and the resulting phase distribution is shown on the right. Here we see that by applying a -2π phase winding, we erase a vortex from the system, and by applying a -4π phase winding, we flip the vortex, creating an anti-vortex.

points and covering an extent of $700\mu\text{m} \times 700\mu\text{m}$

First, ground-state evolution was performed with a low rotation frequency of $\Omega = 0.3 \times 2\pi$ Hz via GPUE [10]. Though large rotational frequencies will create a triangular lattice, for smaller frequencies, other configurations are known [159], and we will focus on the regime where the ground state is composed of four vortices in a square configuration [160]. Once this configuration is achieved, we then manipulate a vortex via phase imprinting, such that three co-rotating vortices and one anti-vortex exist in the system. Examples of phase imprinting on this system can be seen in Figure 4.1, where the top row shows a simple vortex annihilation and the bottom row shows a vortex flip.

4.3 Regular and irregular vortex dynamics

It is known that a lattice of vortices with the same direction of rotation will exhibit regular dynamics [56], and in Figure 4.2(a), we confirm this for this system. In this figure, we show a histogram of the vortex trajectory over 20 seconds of evolution when removing and then re-imprinting a vortex of the same rotational direction at the same location. Even though a small residual movement appears, potentially due to phonon excitations that were not fully removed from the imaginary time evolution, the vortices remain stationary. In Figure 4.2(b), we also show that if the re-imprinted rotation is of the opposite direction, the vortex dynamics become more disordered, with vortices traversing a larger width of the condensate. It is worth mentioning that similar histograms can be constructed experimentally with available imaging techniques [152, 153].

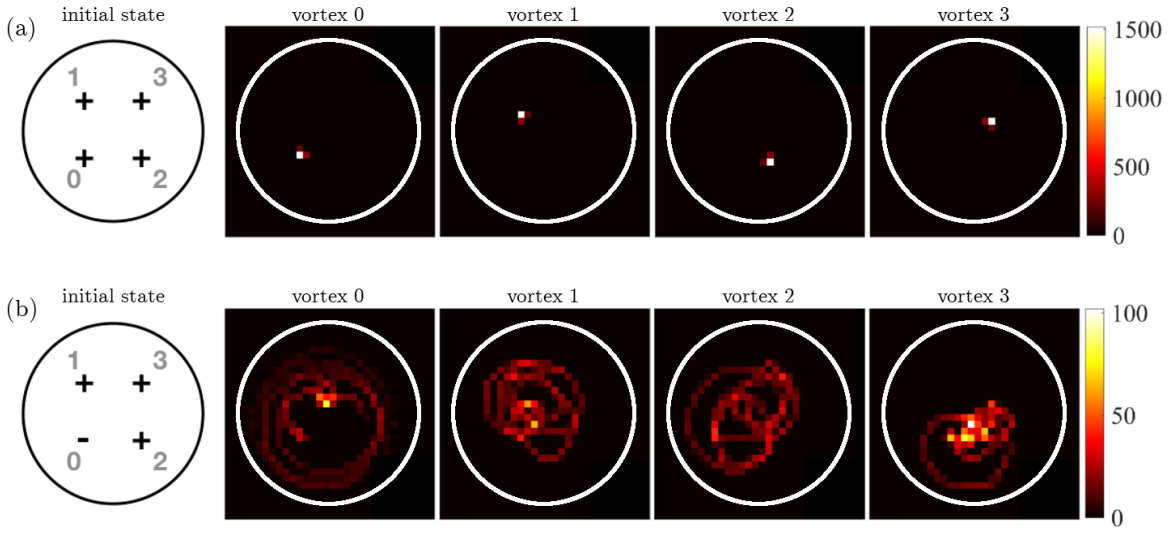

Figure 4.2: Histograms of the positions of each vortex in the transversal plane for 20 seconds in the co-rotating frame. The lower left vortex has been annihilated and re-imprinted with (a) the same and (b) the opposite direction of rotation, exactly on the location of the previous vortex. The area of each plot is $400\mu(m) \times 400\mu$m, and the white circles correspond to iso-lines at 40% the maximum density to highlight the extent of the vortex motion. In (a), we see that if all four vortices are co-rotating, regular trajectories appear, but in (b), we see that flipping the rotation direction of a single vortex creates disordered trajectories.

Even though the introduction of the anti-vortex creates a disordered trajectory, it could be entirely possible that this trajectory is still stable. To uniquely identify chaotic behavior, we need to show that any small perturbation in the vortex location will also provide a significantly different trajectory. To check this, we compare two sets of vortex trajectories with slightly different shifts in the initial position of the anti-vortex, $\mathbf{r}_0$ and $\mathbf{r}'_0$, where $\mathbf{r}_0 - \mathbf{r}'_0 = \xi/3$ and ξ is the healing length. In Figure 4.3, we show the differences in trajectories, defined as $\Delta\mathbf{r}_i(t) = \mathbf{r}_i(t) - \mathbf{r}'_i(t)$, where $\mathbf{r}_i$ refers to the position of the ith vortex from the center of the condensate and $i \in \{1, 2, 3\}$ corresponds to the vortex number. The unique ID for each vortex is shown in Figure 4.2. Here, we see that the difference in trajectory is initially small, but diverges significantly at around $t \approx 10$ seconds, which is a strong indication of chaotic behavior.

After closely inspecting the vortex dynamics (shown in the supplementary movie [161]), we see that this strong divergence in vortex trajectories seems to be accelerated when all four vortices come in close proximity. Because the velocity fields of each vortex decays as $1/\mathbf{r}$, where $\mathbf{r}$ is the distance from the vortex's core, the vortices experience stronger velocity fields when they are closer; therefore, the point of minimal separation can be seen as a highly nonlinear multi-vortex scattering event that accelerates the divergence shown in Figure 4.3. In Figure 4.4 we study this further by showing snapshots of the condensate density before ($t = 6$s), at ($t = 10$s), and after ($t = 15$s) the scattering in (a) and (b) for the non-shifted and shifted anti-vortex locations. The differences between position of each vortex and the anti-vortex is also shown (c) for the case where the anti-vortex is shifted by $\xi/3$. Here, we see that there is a clear

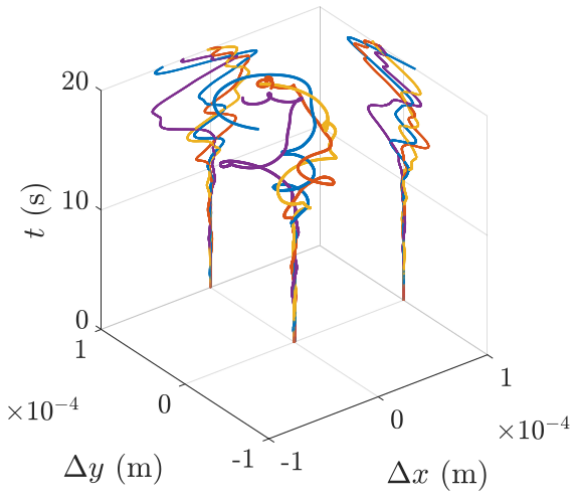

Figure 4.3: Evolution of the difference in trajectories $\triangle \mathbf{r}_i = \mathbf{r}_i - \mathbf{r}'_i$ with $\mathbf{r}$ corresponding to the position of the ith vortex from the center of the condensate and $i \in \{0,\ 1,\ 2,\ 3\}$ labelling each individual vortex for the four-vortex system as shown in Fig. 4.2. A small change in the initial position of the anti-vortex arises from a phase-imprint at (x_0, y_0) and $(x_0 - \xi/3, y_0)$ where (x_0, y_0) denotes the pre-existing co-rotating vortex core position. The curves show that even though the onset of disorder is immediate, a strong divergence of trajectories is observed at about $t \approx 10$s (see projections onto the x-t and y-t planes). The difference in trajectory of the anti-vortex, $\Delta \mathbf{r}_0$, is shown in blue, while yellow, orange, and purple lines depict the three co-rotating vortices.

minimum at $t \approx 10$ seconds, which is the same time at which the trajectories begin to diverge in Figure 4.3. In order to characterize this divergence in trajectory, we must analyze the vortex dynamics in more detail and calculate the Lyapunov exponent.

4.4 Characterizing chaotic vortex dynamics

To characterize the degree of chaos for the shown vortex dynamics, we have chosen to use Lyapunov exponents, which give the rates of divergence for nearby orbits in phase space [162]. With this measure, we track two trajectories in phase space and assume that the divergence between the two trajectories will either exponentially converge or diverge, which can be modelled with

$$|\delta \mathbf{Z}(t)| \approx e^{\lambda t} |\delta \mathbf{Z}_0|. \tag{4.2}$$

Here, $\delta \mathbf{Z}_0$ is the initial separation between the trajectories and λ is a quantity known as the Lyapunov exponent. If the exponent is negative, this means that the trajectories tend to converge, but if it is positive, the trajectories will diverge, thus indicating chaotic motion. The rate of divergence is determined by the value of the exponent.

For our simulation, we model the trajectories in four-dimensional phase-space with $\mathbf{P}(\mathbf{t}) = (x(t), y(t), v_x(t), v_y(t))$ and $\mathbf{P}'(t) = (x'(t), y'(t), v'_x(t), v'_y(t))$. The separation is then defined to be $\delta\mathbf{P}(\mathbf{t}) = (\delta x(t), \delta y(t), \delta v_x(t), \delta v_y(t))$ where $\delta x(t) = x(t) - x'(t)$, etc. The exponent can then be calculated as

$$\lambda = \lim_{t \to \infty} \frac{1}{t} \ln \frac{||\delta \mathbf{P}(t)||}{||\delta \mathbf{P}(0)||} \tag{4.3}$$

where $|| \cdot ||$ denotes the Euclidean norm.

In this case, we track each vortex with the methods outlined in Chapter 3, and use both the position and velocity of the vortex. Because we wish to determine whether the total system is chaotic, beyond its constituent vortices, we use a center of mass (COM) variable, defined as $\mathbf{R}_M = \frac{1}{n+1} \sum_{i=0}^{n} r_i$, where $n+1$ is the number of vortices. Similarly, the center of velocity is defined as $\mathbf{v}_M = \frac{1}{n+1} \sum_{i=0}^{n} v_i$. These values are then used with Equation (4.3), and the results are shown in Figure 4.5. The insets in Figure 4.5 show the histograms of the COM trajectories for the case where an anti-vortex is and is not present.

As expected, the exponent spectrum calculated in Figure 4.5 shows that the regular, co-rotating system always shows a negative (converging) exponential value, but the system with the anti-vortex is largely positive (diverging). During the scattering event shown in Figures 4.4 and 4.3, we see that the exponent becomes positive for the remaining duration of the simulation. It is worth noting that other global measurements could be used instead of the COM, such as the center of charge [141], but we found these to provide similar qualitative results to those shown here.

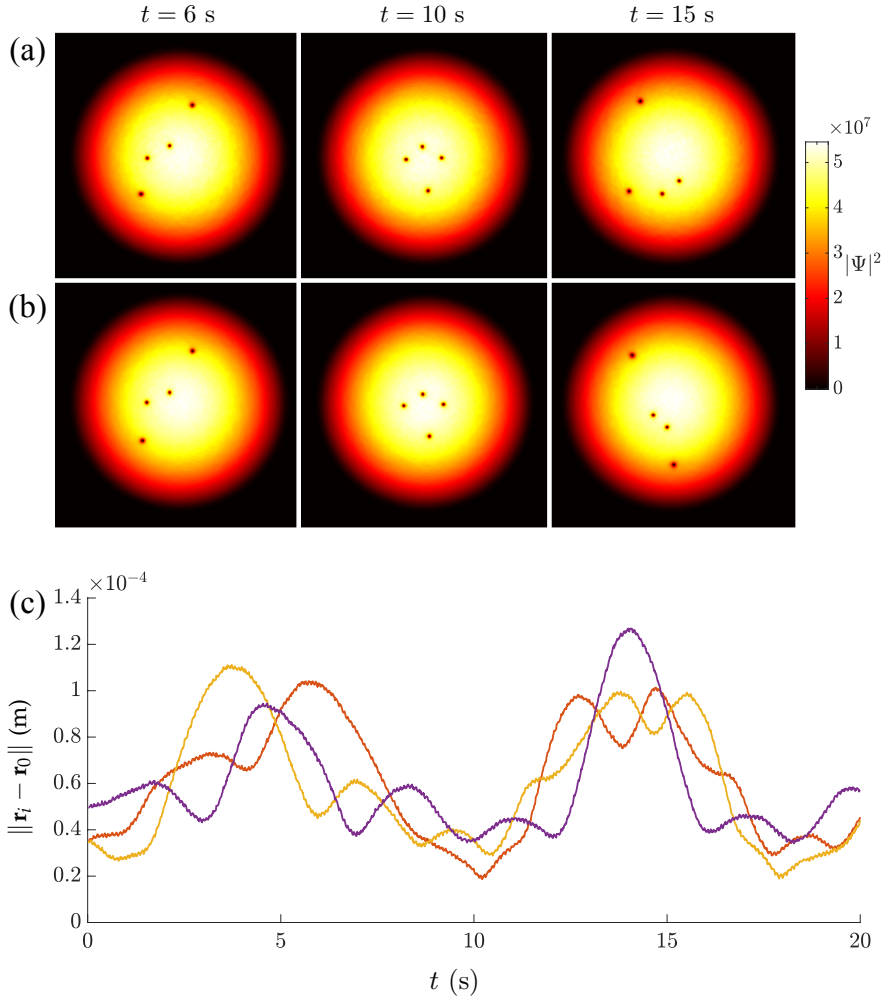

Figure 4.4: Density plots of condensate for (a) $\Delta x = 0$ and (b)$\Delta x = \xi/3$ at times $t = \{6, 10, 15\}$ s. The densities before the scattering event differ only on small scales (see $t = 6s$), whereas for times after the event large deviations are visible (see $t = 15s$). At $t = 10$ seconds the vortices make their closest approach. The area plotted is $500\mu m \times 500\mu m$. (c) Distances between the vortices at positions $\mathbf{r}_i$ with $\mathbf{r}$ corresponding to the position of the ith vortex from the center of the condensate and $i \in \{1, 2, 3\}$ corresponding to the vortex number as shown in Fig. 4.2 and anti-vortex at $\mathbf{r}_0$ for $\Delta x = 0$. A minimum around $t = 10$ seconds is clearly visible.

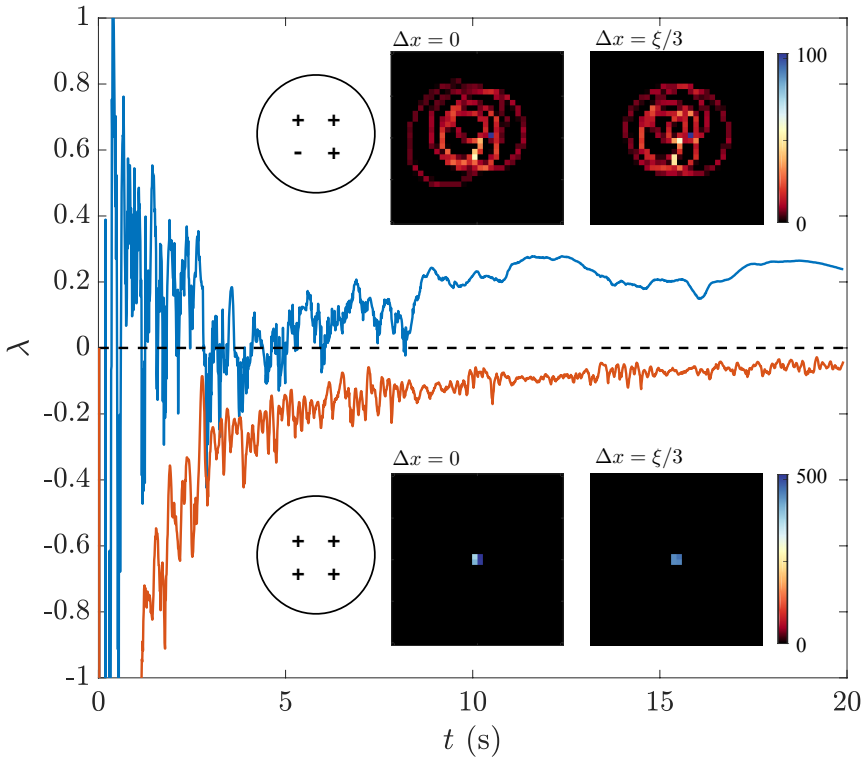

Figure 4.5: The insets show the histograms of the COM trajectories calculated over 20 seconds of evolution for the system of four vortices when the position of a single vortex has been shifted by $\Delta x = 0\xi$ and $\Delta x = \xi/3$. The upper two panels depict the corresponding trajectories after the direction of rotation of a single vortex has been reversed, whereas the lower row displays the trajectories for the case where all vortices co-rotate. The main curve plots the corresponding Lyapunov exponents, calculated from the shown COM trajectories. The negative Lyapunov exponents (orange) indicate that shifting the vortex about the initial position still ensures the stability of vortex trajectories. Reversing the direction of circulation of a single vortex (blue) however leads to fluctuations about zero, eventually leading to a fully positive exponent.

4.5 Outlook

With this study, we have shown that it is possible to induce chaotic vortex dynamics in few-vortex systems by using phase imprinting to flip the rotational direction of a vortex in two dimensions. We also show that a scattering event seems to be correlated to a positive Lyapunov exponent and an acceleration of chaotic behavior. This behavior is radically different than the behavior of large scale vortex lattices, as similar techniques for these systems have been shown to only cause local disturbances [55]. Further exploration of the crossover from regular to turbulent dynamics, and the crossover from chaotic to stable dynamics for large-scale vortex lattices remains an interesting extension for future work.

This study shows that there is strong utility in simulating two-dimensional quantum gases and highlights dynamic measures, such as the Lyapunov exponent. Here, it is obvious that fast vortex tracking methods are essential to dynamical turbulence and chaos modelling, and a major limitation to performing similar studies in three dimensions is the computational hurdle of vortex tracking in this area. As such, most three-dimensional studies of quantum chaos rely on other methods, such as vortex filament methods which provide vortex skeletons during the simulation, itself. As mentioned in Chapter 1, these methods cannot simulate the underlying dynamics of the condensate, and are thus removed from experimental application. Further extensions of this work in three dimensions would also allow for studies on the movement of the vortex lines, themselves, which were projected onto two-dimensional point-vortices in this model.

It is important to note the utility of the GPUE codebase in this study. Throughout the process, highly-resolved (up to 2048×2048) wavefunction densities were generated in the ground state via imaginary time propagation and dynamics were determined with real-time evolution. For a typical run of 1024×1024 with 1×10^6 steps, with a time resolution of 1×10^{-5} and outputting every 1000 steps, the simulation can be completed within an hour. This metric includes vortex tracking on every output step, which is a computational intensive task, as discussed in Chapter 3. Because of the computational speed of GPUE, this study performed hundreds of individual runs with slightly varying initial conditions to provide a strong indication of chaotic behaviour. Because this study very heavily relied on post-processing metrics, such as vortex tracking and the calculation of the Lyapunov exponent, it is apparant that it is ideal to provide an interface to the GPUE codebase in a language that more easily provides the functionality expected by researchers for data manipulation. Such an interface is possible with GPUE.jl and further motivates this development.

For the next study, we will transition into a discussion of three-dimensional vortex dynamics and show an experimentally realistic system to allow for the generation, control, and detection of vortex ring-like systems with artificial magnetic fields.

Chapter 5

Generation, control and detection of 3D vortex structures in superfluid systems

In this chapter, I will discuss another application of the GPUE codebase, this time to the controlled creation of vortex structures in three dimensions with artificial magnetic fields generated by an optical nanofiber. To the best of my knowledge, this is the first time an experimentally realizable device to generate vortex ring-like structures in this manner has been suggested and investigated, and we also provide a method to detect whether a vortex ring is present in an elliptic-toroidal condensate. This project encompasses three-dimensional vortex dynamics and coupled light-matter systems, so to begin, I will briefly discuss the dynamics of vortices in three-dimensional systems used in this system, followed by the model used for this project, where I will describe how the light from an optical nanofiber can generate and control vortex structures in BEC systems. This chapter will largely follow the work of Schloss *et al.* [12].

5.1 Vortex ring dynamics in BEC systems

As mentioned in Chapter 1, in BEC systems with large amounts of angular momentum and a single axis of rotation, the vortices will create a triangular, Abrikosov lattice [48, 56]. This regular structure is a direct consequence of the quantization of angular momentum in quantum mechanics, and in Chapter 4, I discussed a small vortex lattice in two-dimensions by integrating out the $\hat{z}$ direction. In this chapter, I will discuss the full three-dimensional behavior of vortex structures in BEC systems. In three dimensions, BEC systems have been shown to support a large variety of flow-related excitations, such as vortex lines and rings [36, 50, 56, 163–167]. There are many interesting features to superfluid vortices in three dimensions, many of which follow from classical fluid dynamic theory [43], which is a well-studied field and covered in many texts [168–171]. To start, I will discuss the dynamics of vortex rings in BEC systems.

In a three-dimensional BEC and other superfluid systems, vortices can be described as systems of freely-moving, current-carrying filaments located along the axis of rota-

tion. In this way, it is possible to model such systems using the Biot-Savart law [172]

$$\boldsymbol{v}_s(\boldsymbol{r},t) = \frac{\kappa}{4\pi}\int \frac{|s_1 - \boldsymbol{r}| \times ds_1}{|s_1 - \boldsymbol{r}|^3}, \tag{5.1}$$

where $\kappa = \hbar/m$ is the circulation. Modelling a superfluid system in this way is the heart of the vortex filament model, and allows us to attain an intuitive understanding for the motion of vortices in a superfluid system; however, it falls short in several areas, such as sound wave propagation and vortex reconnections [173]. Even so, many of the dynamics discussed in this section can be understood with the Biot-Savart law. This model is traditionally taught alongside electrodynamics, where current can only flow within a closed loop or between two conductors. Similarly, when a vortex filament is generated by injecting angular momentum into a superfluid system, it must either end at the edges of the superfluid or reconnect with other vortices.

By modifying our axis of rotation or inducing a vortex structure with either artificial magnetic fields or phase imprinting, we may create three-dimensional topologies, like the vortex ring. This structure is also common in large, three dimensional modelling of superfluid systems and is a direct consequence of the required connections of vortex lines. The stability of vortex rings is ensured by Kelvin's theorem [174], which means that unstable excitations may decay into vortex rings or objects with ring-like topology [164].

We will begin our discussion of vortex rings with the thin-core approximation in inviscid fluids, as described by Barenghi and Donnelly [175]. If we imagine a thin vortex ring of radius $R \gg a$, where a is the core radius, and with circulation Γ moving in an inviscid fluid with density ρ, the kinetic energy of the ring is given by

$$E = \frac{1}{2}\rho\Gamma^2 R\left[\ln\left(\frac{8R}{a}\right) - \alpha\right], \tag{5.2}$$

where α is a parameter determined by the system, and different values of α correspond to different core models for vortex motion. For example, it is 1.615 for the GPE [176]. The momentum of the system is given by

$$p = \rho\Gamma\pi R^2, \tag{5.3}$$

and the self-induced velocity is [177]

$$v = \frac{\partial E}{\partial p} = \left(\frac{\Gamma}{4\pi R}\left[\ln(\frac{8R}{a}) - \beta\right]\right), \tag{5.4}$$

with $\beta = \alpha - 1$ under constant pressure, and $\beta = \alpha - 3/2$ under constant volume. This method of vortex ring analysis has been extended to rings with finite, but small values for the core radius [178, 179]. These equations come from the Biot-Savart law (Equation (5.1)) above. If we consider a filament that is infinitesimally thin, we find an energy $E = \rho\Gamma^2 R/2$ and a velocity of $v = \Gamma/4\pi R$.

In the case of multiple interacting vortex rings, we can expect to find many similar features in superfluids to what has been found previously in classical, viscous fluids. If two vortex rings are generated in the same plane and in close proximity, it could

be possible for the two velocity fields to interact, causing one ring to expand and slow down while the other contracts and speeds up. Under the right conditions, the lagging ring can pass the forward ring through a process known as *leapfrogging* [180, 181].

In addition to leapfrogging, vortex rings can interact through direct collisions [182]. In superfluid ^{4}He, some of the earliest experiments on vortex collisions with vortex rings were performed by Schwarz in 1968 [183]. In the case of a head-on collision, two identical, moving vortex rings will first grow in size before dispersing into a series of smaller vortex rings around their common circumference [184]. These smaller rings are created by vortex reconnections, which can occur any time vortex lines are facing anti-parallel directions and it is energetically favorable to do so.

Finally, we will briefly discuss vortex reconnections, themselves. As predicted by Feynman in 1955, vortex reconnections in a dissipative superfluid systems lead to larger vortices continually reconnecting into smaller ones until the loops become small enough to decay from dissipation or from interactions with boundaries. [53]. These reconnections produce sound waves when vortices directly interact and Kelvin waves when vortices indirectly interact [185]. When a vortex ring structure is not pinned by either gauge fields or rotation, it will evolve naturally by reconnecting into smaller and smaller vortex rings when in a turbulent system [186]. This means that we expect to see vortex reconnections in any sufficiently complicated vortex tangle [52]. In addition, when simulations of vortex rings were performed by Jones and Roberts, they found that by computing vortex rings towards the limit of $R \to a$, they eventually found a solitary wave [187, 188].

Though these dynamics are expected in superfluid systems, it is difficult to devise experimental systems systematically generate the desired behavior. In practice, complex three-dimensional structures cannot be easily created by stirring or rotating a BEC because vortex lines generated in this way must follow the axis of rotation, thus even vortex rings can be a challenge to create, control, and detect experimentally. In most cases, including in most theoretical proposals, vortex ring generation in BEC systems relies on dynamic processes that do not create eigenstates of the system, such as the decay of dark solitons in multicomponent condensates [164] with the snake instability [189], the collision of symmetric defects [190], or direct density engineering [191, 192]. There are other theoretical proposals that consider interfering two BEC systems [186], using Feschbach resonances [193], or phase imprinting methods [189]. As a note, in inhomogeniously trapped BEC systems, vortex ring structures are known to be unstable, which has led to difficulties in their experimental observation [194]. In addition, imaging techniques employed for BECs are not suited to identify whether three-dimensional vortex structures are present.

To consistently control and generate more complex three-dimensional structures, methods beyond rotation must be used, and there are only a few known experimental systems that can do so [164, 167]. There is also a large amount of interest in generating more complicated vortex structures, such as vortex knots [72, 195, 196]. Artificial magnetic fields seem to be a promising method for the generation of complex three-dimensional vortex structures in BEC systems [197], and in this chapter, we will present a method to generate vortex rings, ring-lattices, and other vortex structures in three dimensions by using the artificial magnetic field generated by an optical nanofiber.

5.2 Controlled creation of three-dimensional vortex structures in Bose–Einstein condensates using artificial magnetic fields

One method to create artificial magnetic fields involves the interaction between an atomic system in a dressed state and an electric field that is tuned near an atomic resonance frequency [198]. In practice, this means that we can create a configurable artificial magnetic field with an appropriately tuned electric field that varies strongly over short distances, such as those found in the near-field regime on the surface of a dielectric system when light undergoes total internal reflection [199]. One such system that suits this purpose and can be used to generate vortex ring structures in BEC systems is the optical nanofiber, which has several propagation modes to facilitate the generation of configurable artificial magnetic fields.

Optical nanofiber systems can be created by heating and stretching optical fibers until their thinnest region is roughly hundreds of nanometers in diameter [200, 201]. At this scale, the wavelength of light is larger than the diameter of the fiber and the strength of the evanescent field is significantly enhanced [202]. The form of the evanescent field varies significantly depending on the optical modes propagating through the nanofiber, and we will show that this can be used to generate interesting and tunable artificial magnetic fields.

Optical nanofibers are already used in many different experiments with ultracold atoms [203–208], and trapping potentials around 200nm from the fiber surface can be created with two differently detuned input fields [108, 209]. Our proposed device will allow for the creation of vortex rings in BEC systems that are trapped toroidally around the nanofiber at roughly the same distance as nanofiber systems by coupling the BEC to the evanescent field created by different modes propagating through the nanofiber [210]. A schematic of this system is depicted in Figure 5.1

With this same device, it is also possible to detect whether a vortex ring is present in the system by exciting the scissors mode in an elliptic-toroidal trapping geometry [211–213]. This can be done by tilting the trap radially from the center of the torus, which will cause the BEC to oscillate in and out in the new potential, similar to the oscillation shown in Chapter 1 for a simple harmonic oscillator. Without a vortex present, this oscillation possesses a single frequency, whereas in the presence of a vortex ring, it will contain two frequencies that average to the vortex-less oscillation frequency, similar to scissors mode oscillations in a two-dimensional, elliptically-trapped BEC [214–216].

5.2.1 Bose–Einstein condensate dynamics in the presence of an optical nanofiber

As discussed in Chapter 1, in the presence of an artificial magnetic field, the GPE becomes,

$$i\hbar\frac{\partial\Psi}{\partial t} = \left[\frac{(p - m\mathbf{A}(\mathbf{r}))^2}{2m} + V_{\text{trap}}(\mathbf{r}) + g|\Psi|^2\right]\Psi. \tag{5.5}$$

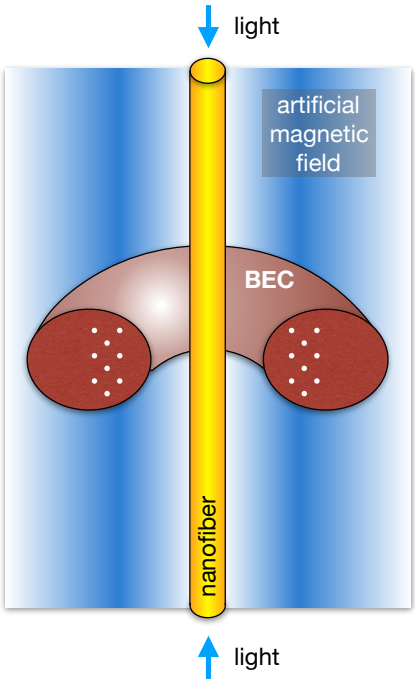

Figure 5.1: Schematic of the system. Blue or red-detuned light is sent into the nanofiber (yellow), creating an evanescent field and artificial magnetic field (blue) that influences the BEC (maroon) held by a toroidal trapping potential. If the artificial magnetic field strength is greater than a threshold value, vortex rings (white) will appear and begin to arrange themselves into a triangular lattice.

Here, all values are defined as before. The artificial vector potential can take many forms, but for here, we will again choose a description based on Berry's connection [198],

$$\mathbf{A} = i\hbar \left\langle \Psi_l | \nabla \Psi_l \right\rangle , \tag{5.6}$$

where Ψ_l is the atomic wavefunction in some dressed state l.

Considering a dressed, two-state atoms in the presence of an optical field, we can write these states within the rotating wave approximation as [199],

$$|\Psi_1(\mathbf{r})\rangle = \begin{pmatrix} \cos[\Phi(\mathbf{r})/2] \\ \sin[\Phi(\mathbf{r})/2] e^{i\phi(z)} \end{pmatrix} , \tag{5.7}$$

$$|\Psi_2(\mathbf{r})\rangle = \begin{pmatrix} -\sin[\Phi(\mathbf{r})/2] e^{-i\phi(z)} \\ \cos[\Phi(\mathbf{r})/2] \end{pmatrix} , \tag{5.8}$$

where $\phi(z)$ is the phase of the optical field and $\Phi(\mathbf{r}) = \arctan(|\kappa(\mathbf{r})|/\Delta)$, with $\Delta = \omega_0 - \omega$ being the detuning and $\kappa(\mathbf{r}) = \mathbf{d} \cdot \mathbf{E}(\mathbf{r})/\hbar$ being the Rabi frequency. The atomic dipole moment is given by $\mathbf{d}$ and $\mathbf{E}(\mathbf{r})$ is the electric field. Here we see that the form of $\mathbf{A}$ follows the form of the optical fields, and the artificial magnetic field is given by $\mathbf{B} = \nabla \times \mathbf{A}$; therefore, it is possible to influence the magnetic field and vortex structures generated with this system by modifying the optical profile of the nanofiber.

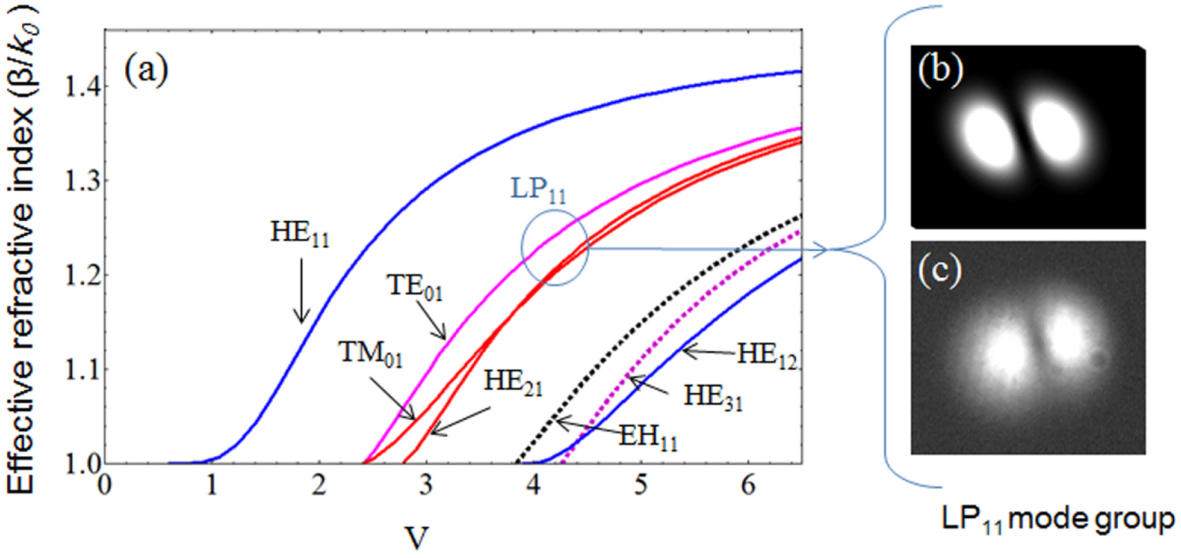

Figure 5.2: (a) Plot of effective refractive index and V number of an optical fiber. The circle indicates the LP_{11} group, which is the first higher-order group composed of the HE_{21even}, HE_{21odd}, TE_{01} and the TM_{01} modes. (b) Simulated and (c) experimental images of the output from the LP_{11} group are also shown. Reproduced from [205, 208].

For optical nanofibers, one can determine which modes will propagate in the system by calculating the V-number, with $V = k_0 a\sqrt{n_1^2 - n_2^2}$, where a is the fiber radius, n_1 is the refractive index of the fiber, n_2 is the refractive index of the cladding, and $k_0 = \omega/c$ with ω being the frequency of the input light beam. The V number can be easily chosen by modifying the radius of the fiber, and the way in which light propagates through the fiber for each mode is characterized by the modal propagation constant, β. In addition, the effective refractive index of the fiber is $n_{\text{eff}} = \beta/\kappa_0$. In figure 5.2, a plot of n_{eff} vs V number is shown in (a) and it can be seen that certain modes cannot propagate until certain threshold values are reached. In (b) and (c), the simulated and experimental images of the fiber output are shown as the LP_{11} group, which is a combination of the HE_{21even}, HE_{21odd}, TE_{01} and the TM_{01} modes. For the system considered in this chapter, the fiber has been tapered such that the cladding is the vacuum, itself, with $n_2 = 1$; therefore, higher order modes can only be sustained if $V > V_c \simeq 2.405$. Below this value, only the fundamental mode can propagate in the system.

Using cylindrical coordinates, the evanescent field of the $HE_{\ell m}$ mode with circular polarization is [217],

$$E_r = iC[(1-s)K_{\ell-1}(qr) + (1+s)K_{\ell+1}(qr)]e^{i(\omega t - \beta z)}, \tag{5.9}$$

$$E_\phi = -C[(1-s)K_{\ell-1}(qr) - (1+s)K_{\ell+1}(qr)]e^{i(\omega t - \beta z)}, \tag{5.10}$$

$$E_z = 2C(q/\beta)K_\ell(qr)e^{i(\omega t - \beta z)}, \tag{5.11}$$

where

$$s = \frac{1/h^2a^2 + 1/q^2a^2}{J'_\ell(ha)/[haJ_\ell(ha)] + K'_\ell(qa)/[qaK_\ell(qa)]}, \tag{5.12}$$

$$C = \frac{\beta}{2q}\frac{J_\ell(ha)/K_\ell(qa)}{\sqrt{2\pi a^2(n_1^2N_1 + n_2^2N_2)}}, \tag{5.13}$$

and

$$\begin{aligned} N_1 =& \frac{\beta^2}{4h^2}\Big[(1-s)^2\left[J_{\ell-1}^2(ha) + J_\ell^2(ha)\right] \\ &+ (1+s)^2\left[J_{\ell+1}^2(ha) - J_\ell(ha)J_{\ell+2}(ha)\right]\Big] \\ &+ \frac{1}{2}\left[J_\ell^2(ha) - J_{\ell-1}(ha)J_{\ell+1}(ha)\right], \end{aligned} \tag{5.14}$$

$$\begin{aligned} N_2 =& \frac{J_\ell^2(ha)}{2K_\ell^2(qa)}\Bigg(\frac{\beta^2}{4q^2}\Big[(1-s)^2\left[K_{\ell-1}^2(qa) - K_\ell^2(qa)\right] \\ &- (1+s)^2\left[K_{\ell+1}^2(qa) - K_\ell(qa)K_{\ell+2}(qa)\right]\Big] \\ &\left[K_\ell^2(qa) + K_{\ell-1}(qa)K_{\ell+1}(qa)\right]\Bigg). \end{aligned} \tag{5.15}$$

The mode geometry is given by $J_n(x)$, the Bessel function of the first kind, $K_n(x)$, the modified Bessel function of the second kind, and β, the propagation constant of the fiber. The scaling factors are given by $q = \sqrt{\beta^2 - n_2^2k_0^2}$ and $h = \sqrt{n_1^2k_0^2 - \beta^2}$, the normalization constant is C and s is a dimensionless parameter.

When the input light field is linearly polarized, it is convenient to write the Cartesian components of the evanescent electric field as

$$E_x = \sqrt{2}C\Big[(1-s)K_{\ell-1}(qr)\cos(\phi_0) + (1+s)K_{\ell+1}(qr)\cos(2\phi - \phi_0)\Big]e^{i(\omega t - \beta z)}, \tag{5.16}$$

$$E_y = \sqrt{2}C\Big[(1-s)K_{\ell-1}(qr)\sin(\phi_0) + (1+s)K_{\ell+1}(qr)\sin(2\phi - \phi_0)\Big]e^{i(\omega t - \beta z)}, \tag{5.17}$$

$$E_z = 2\sqrt{2}iC(q/\beta)K_\ell(qr)\cos(\phi - \phi_0)e^{i(\omega t - \beta z)}. \tag{5.18}$$

Here ϕ_0 determines the orientation of polarization, with $\phi_0 = 0$ being along the x axis and $\pi/2$ being along the y axis. The artificial vector potential produced by such evanescent fields around an optical nanofiber is then given by [210]

$$\mathbf{A} = \hat{z}\hbar\kappa_0(n_1+1)\tilde{s}\left[\frac{|d_rE_r + d_\phi E_\phi + d_zE_z|^2}{1 + \tilde{s}^2|d_rE_r + d_\phi E_\phi + d_zE_z|^2}\right], \tag{5.19}$$

where $\tilde{s} = \frac{|\mathbf{d}\cdot\mathbf{E}|}{\hbar|\Delta|}$ and the corresponding magnetic field $\mathbf{B} = \nabla \times \mathbf{A}$ can be calculated to be

$$\begin{aligned} \mathbf{B} =& \frac{\hbar\kappa_0 s^2(n_1+1)}{(1 + \tilde{s}^2|d_rE_r + d_\phi E_\phi + d_zE_z|^2)^2} \\ &\times \Bigg[\hat{\phi}\frac{\partial}{\partial r}|d_rE_r + d_\phi E_\phi + d_zE_z|^2 \\ &- \hat{r}\frac{1}{r}\frac{\partial}{\partial\phi}|d_rE_r + d_\phi E_\phi + d_zE_z|^2\Bigg]. \end{aligned} \tag{5.20}$$

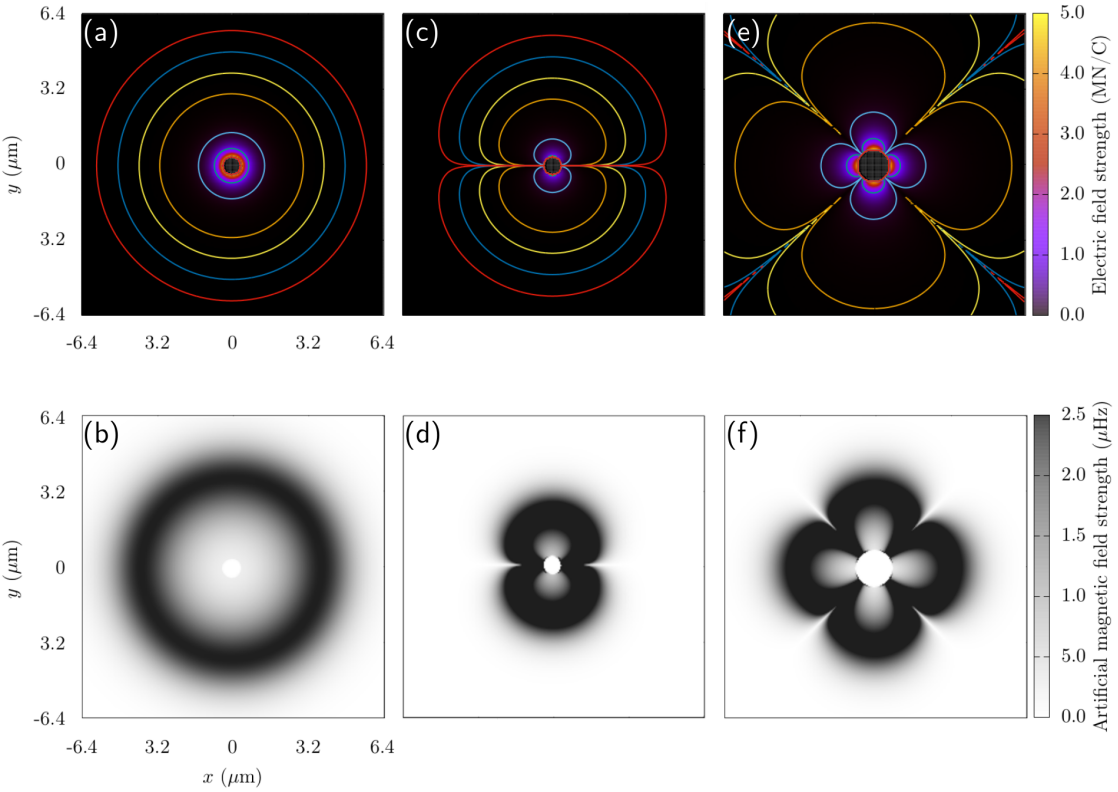

Figure 5.3: Images of electric and artificial magnetic field profiles for [(a) and (b)] the fundamental HE_{11} mode with circular polarization, [(c) and (d)] the HE_{11} mode with linear polarization, and [(e) and (f)] the HE_{21} mode with linear polarization. For these calculations, the input power is 372 nW in (a) and (b) , 16 nW in (c) and (d), and 418 nW in (e) and (f). For the HE_{11} mode, the nanofiber radius is 200 nm with blue-detuned light of 700 nm, and for the HE_{21} mode, the nanofiber radius is 400 nm with red-detuned light of 980 nm

This shows that the **B** field has only components in the $\hat{\phi}$ and $\hat{r}$ directions, which means that all field lines lie in the horizontal plane if the fiber is aligned along the vertical $\hat{z}$ direction.

This also means that a BEC trapped toroidally around the nanofiber would facilitate vortex structures that wrap around the nanofiber and potentially close on themselves in the form of vortex rings; however, other vortex structures are possible as well. In addition, the value of $\tilde{s}$ governs the amplitude and range of the magnetic field, and as such, it is possible to manipulate the size and shape of the generated vortex rings by changing the detuning and intensity of the electric field [210].

For the purposes of this project, we will consider the fundamental HE_{11} mode with circular polarization, the HE_{11} mode with linear polarization, and the HE_{21} mode with linear polarization. Though even higher-order modes may be generated by the optical nanofiber, increasing the V-number to facilitate these modes also requires increasing the fiber radius beyond what is experimentally achievable. It is also possible to create even more complex field configurations by interfering different modes; however, the chosen modes will still demonstrate a range of possible magnetic field profiles. The electric field

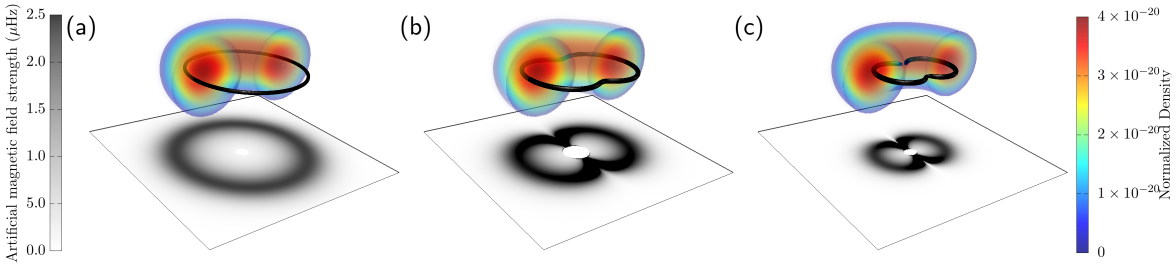

Figure 5.4: Vortex configurations for different magnetic field profiles from the nanofiber for the fundamental HE_{11} mode with (a) circular polarization, (b) elliptical polarization, and (c) linear polarization along the $\hat{y}$ direction. The vortex distributions have been found via an isosurface on the Sobel filtered wavefunction density for a ^{87}Rb BEC and all optical fiber fields are normalized and for a nanofiber of 200 nm in radius with blue-detuned light of 700nm. The magnetic field profiles shown in the shaded region beneath wavefunction density are similar to those in Figure 5.3(b) and (d).

configurations and their corresponding magnetic field profiles can be seen in Figure 5.3. Here, we see in that the circularly polarized HE_{11} mode will create a cylindrically symmetric electric (a) and magnetic (b) field profiles; however, linearly polarized light will create a lobed structure for both (c and d). When using the linearly-polarized HE_{21} mode, we see four petals appear in the electric and magnetic field profiles, which suggest unusual vortex structures. Now I will discuss what types of vortex structures can be generated with this system, including the possibility of generating vortex ring lattices.

5.2.2 Ground state vortex configurations

We will use GPUE [10] to describe a ^{87}Rb condensate with 1×10^5 atoms with a scattering length of $a_s = 4.76 \times 10^{-9}$ m on a three-dimensional grid of 256^3 points with a spatial resolution of 50 nm, and we assume a toroidal trapping potential around the fiber given by,

$$V_{\text{trap}} = m(\omega_r^2(r-\eta)^2 + \omega_z^2 z^2), \tag{5.21}$$

where we choose the frequencies in the $\hat{r}$ and $\hat{z}$ directions to be $\omega_r = \omega_z = 7071$Hz to match typical experimental conditions in fiber trapping [203]. Here, η describes the distance of the center of the torus from the center of the fiber and is chosen such that the atoms are trapped beyond the van-der-Waals potential of the fiber. For the HE_{11} mode, the fiber radius is 200 nm and $\eta = 3.20\,\mu$m, creating a toroidal BEC with an inner radius of roughly 300 nm from the fiber surface. For the HE_{21} mode, we increase the fiber radius to 400 nm, but keep all other parameters the same, creating a toroidal BEC with an inner radius of roughly 150 nm.

As shown in Figure 5.4(a), when simulating the HE_{11} mode with these parameters, we see a vortex line that wraps around the fiber and reconnect in the form of a single vortex ring as the ground state solution. In contrast, the linearly polarized HE_{11} mode in Figure 5.4(c) shows a ground state where the vortex lines bend toward the center of the torus, creating two vortex lobes. As a note, the vortex lines do not follow the magnetic field lines exactly, but instead reconnect to the neighboring lobe

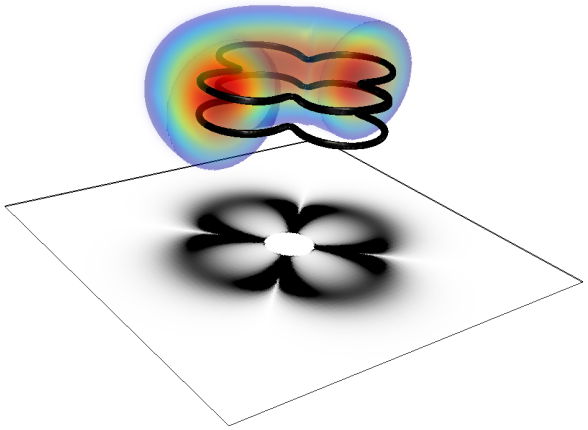

Figure 5.5: Vortex configuration for the HE_{21} mode with linear polarization along the $\hat{y}$ direction. The magnetic field profile is similar to the one shown in Figure 5.3(f), and has been calculated for a nanofiber of 400 nm in radius with red-detuned light of 980 nm.

when approaching each other within a healing length. If elliptically polarized light is considered, we see a hybrid ground state between the circularly and linearly polarized modes, shown in Figure 5.4(b). Finally, we show a four-petal ground-state solution when using the HE_{21} linearly polarized mode in Figure 5.5. Here, we also show that it is possible to generate multiple vortex structures in-line with themselves by increasing the intensity of the artificial magnetic field.

This indicates that it is possible to generate interesting vortex ring lattice structures in three-dimensions by sufficiently increasing the artificial magnetic field. To study the control of multiple vortex structures with this system, we first simulated a system with low artificial magnetic field strength and showed that this simulation will cause all vortex rings to line up at peaks in the magnetic field in Figure 5.6(a and b). As the magnetic field is increased from this point, we see that the vortex rings begin to pack together and form an Abrikosov-like lattice in-line with peaks in the artificial magnetic field, shown in Figure 5.6(a and c). If other magnetic field profiles are used, it could be possible to generate different Abrikosov-like ring lattice configurations. This system could allow for studies of bulk vortex-ring movement, thereby potentially creating a vortex ring lattice that rotates in a leapfrogging manner and moves based on individual vortices self-induced velocity.

The optical nanofiber seems to provide unprecedented control over the vortex geometries generated in a toroidally-trapped BEC system, and one can control the shape of each vortex structure by manipulating the optical fields input into the nanofiber. In addition, because the optical fields could be time-dependent, this system could be used in the future to probe dynamical effects of vortices. In this case, one must consider the effects of high artificial magnetic fields on the distribution of atoms, themselves, because (as described in Chapter 1), the external potential V_{trap} will be modified by a term proportional to $\mathbf{A}^2$. In Figure 5.7, we show this modification for the artificial magnetic fields shown in Figure 5.6(c). Dynamical studies in which the gauge field is switched off in a finite time will thus lead to phonon excitations in the condensate,

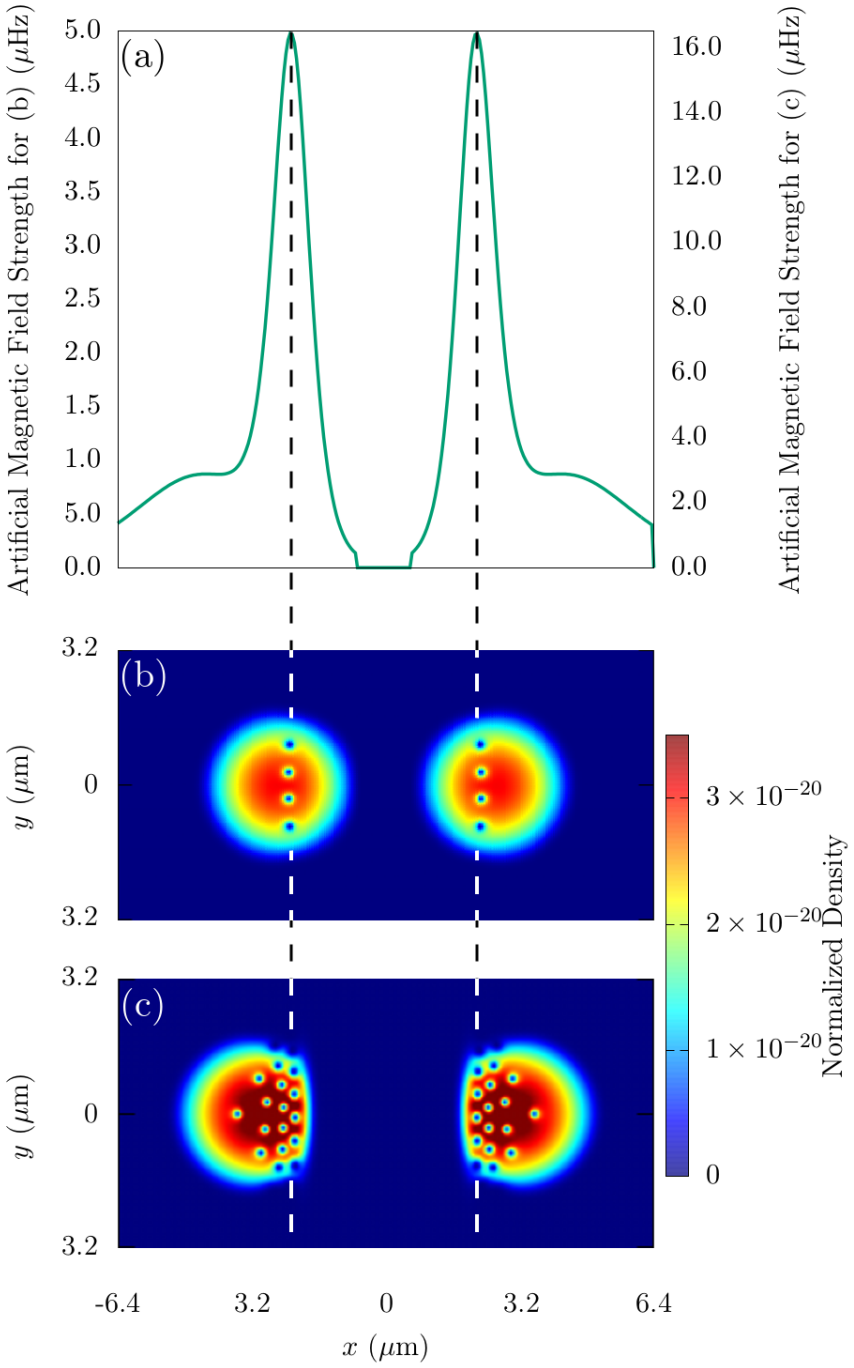

Figure 5.6: (a) The magnetic field profile along the x-direction for the fundamental HE_{11} mode with circular polarization outside a fiber of 200 nm radius. Note that for this mode and polarization the whole system is azimuthally symmetric. For weak fields (see (b)) this leads to a small number of vortices that align along the line at which the magnetic field is maximal and for larger fields (see (c)) more vortex rings appear that form the beginning of an Abrikosov lattice. The optical fiber field and wavefunction density have been normalized and are for a nanofiber of 200 nm in diameter with blue-detuned light of 700nm and a ^{87}Rb BEC respectively.

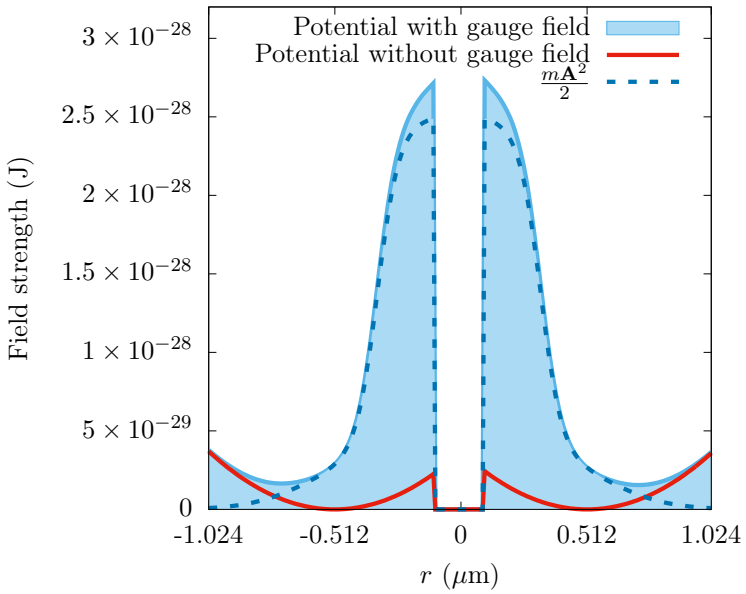

Figure 5.7: Visualization of the change in potential due to the effects of the artificial magnetic field for Figure 5.6. Here, the experienced potential is shown in the blue shaded region, the scaled artificial vector potential is shown as a dotted, dark blue line, and the original trap is shown in red.

which will also influence the vortex lines, themselves, and this is an area of future work.

5.2.3 Dynamic vortex detection and scissor modes

Observing the presence of vortex rings in a three dimensional BEC is a difficult problem, as absorption spectroscopy usually only provides a picture of an integrated two-dimensional density. However, due to the unique geometry of this system, we can identify whether vortex rings are present by exciting the scissors mode of the condensate [211–213]. For an elliptic-toroidal geometry ($\omega_z < \omega_r$), the scissors mode can be excited by modifying the external potential with a rotation in the rz-plane,

$$V = V_{\text{trap}}(r, \theta, z) - m\omega_0^2 \alpha r z, \tag{5.22}$$

where $\alpha = 2\epsilon\theta$ is a coefficient related to the tilting angle, ϵ is the deformation of the trap in the rz plane and θ is the angle at which the original trap was aligned at. For a small initial angle of θ_0 this change in the potential causes a BEC without rotation to oscillate back and forth in the trap with a frequency given by [216]

$$\omega_{\text{scissors}} = \sqrt{\omega_r^2 + \omega_z^2}. \tag{5.23}$$

If, however, this mode is excited for a BEC that contains a vortex line, the oscillation will be strongly influenced by the currents inside the condensate and two different frequencies (ω_+ and ω_-) appear in the oscillation [214–216]. When the splitting frequency is small compared to the scissors mode frequency, it can be written as [215]

$$\omega_+ - \omega_- = \frac{\langle l_z \rangle}{m\langle r^2 + z^2 \rangle}, \tag{5.24}$$

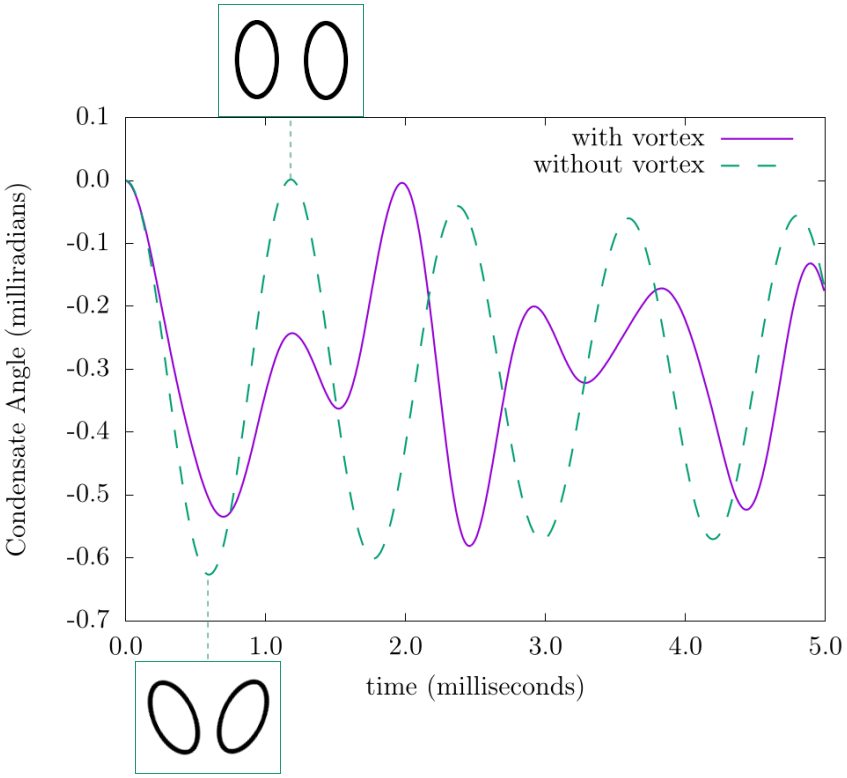

Figure 5.8: Angle of the condensate axis after excitation of the toroidal scissors mode for an elliptic-toroidal BEC with a single vortex ring (purple, solid) and without a vortex present (cyan, dashed). Depictions of two dimensional slices for the scissors mode without a vortex are shown in the insets. Here we see that the scissors mode causes oscillation in and out towards the center of the system, and that two distinct frequencies are present in the curve for the condensate carrying a vortex ring.

where $\langle l_z \rangle$ is the average angular momentum per particle. Calculating these values for our system for $\omega_r = 4242$Hz, and $\omega_z = 2828$Hz then leads to $\omega_{\text{scissors}} = 5090$Hz, $\omega_- = 3765$Hz, and $\omega_+ = 6415$Hz, which are very close to the values observed in the numerical simulations shown in Fig. 5.8. However, one can also see from this figure that the oscillation is not perfect and seems to decay over time. This is due to the above mentioned modification of the trapping potential by the artificial vector potential, which leads to a deviation from the perfect elliptical toroidal shape used in the derivation of Equation (5.24).

It is worth noting that this method cannot be used to detect a vortex ring inside a simply connected condensate, as in this situation the flow around the vortex line has no preferred direction. However, in the toroidal shape, each radial slice can be seen as a two-dimensional elliptical BEC with a single vortex, and the system will therefore exhibit the scissors mode frequency as expected. While in principle the excitation of the scissors mode can also be used to detect Abrikosov vortex-ring lattice, the fact that the inhomogeneous artificial magnetic field leads to an inhomogeneous vortex ring distribution will have an effect on the expected oscillation frequencies.

5.3 Conclusion

In this application of the GPUE codebase, we have shown that it is possible to create and control vortex rings and more complicated vortex structures by using the artificial magnetic field generated by the optical nanofiber. There is currently no other known method to generate the structures created by the linearly polarized modes, shown in Figure 5.4(b,c) and 5.5. We have also shown that the scissors mode can be used to detect whether a vortex ring is present in an elliptic toroidal trap. These fiber-generated structures could allow for experimental systems to study complicated superfluid mechanisms, like the kelvin-mode cascade, superfluid turbulence, or reconnection events between superfluid vortex lines. These structures might also be the first step in creating knotted vortex lines around an optical nanofiber; however, to generate these structures, the magnetic field must have a dependence on $\hat{z}$, which is not present in this model.

This project leaves several open fields of study for future work and future simulations with GPUE, including the study of dynamic field effects on vortex structures, the generation of vortex knots in superfluid systems with this device, and studies of vortex ring lattice movement in BEC systems. For both of these cases, significant work must be performed both theoretically and computationally.

Conclusion

5.4 Overall conclusions

This thesis has presented my efforts to create a massively parallel SSFM codebase for the simulation of superfluid vortex dynamics in Bose–Einstein Condensates (BECs). Starting from the SSFM and a discussion on the dynamics of ultracold atomic systems, I motivated the dynamic quantum engineering by introducing quantum optimal control and shortcuts to adiabaticity. Both of these methods were used to show that it is possible to generate macroscopic superposition states in a Tonks–Girardeau gas when on a ring with a barrier to break rotational symmetry. From there, I introduced the GPGPU and the GPUE codebase, emphasizing existing challenges in the field and the methods I used to overcome them. In the process, I also briefly mentioned the challenging problem of memory coaliescence when using spectral methods on GPU hardware and proposed an additional software package as method to further optimize the FFT operations with a distributed, multi-GPU transpose. After this, I introduced an example physical system that showed it is possible to generate chaotic vortex dynamics in few-vortex systems and emphasized the need for vortex tracking methods and post-processing methods, by calculating the Lyapunov exponent on the vortex trajectories. Finally, I introduced a three-dimensional example system that generates, controls, and detects vortex ring-like geometries in a toroidally-trapped BEC coupled to the artificial magnetic field generated by an optical nanofiber.

Throughout this work, there are several physical and computational areas that require further development in the future and these will be further discussed in this section.

[Not really sure which future directions to highlight here...]

5.5 Further development of GPUE

Though the GPUE codebase is roughly feature complete, there are several directions for future development, many of which were described in Chapter 3. In particular, a re-write of GPUE in julia along with developing an n-dimensional, distributed, GPU transpose are currently being worked on. The former will allow for GPUE to be more maintainable and require less development time in the future. The latter is applicable to a wide range of spectral methods and might allow for several methods to become relevant again on HPC environment. As both of these were discussed at length in Chapter 3, I will not discuss them further here; however, there are future directions

where proper development has not begun, such as new vortex tracking methods and potentially using expression trees for general-purpose Hamiltonian solutions.

5.5.1 Vortex tracking in n dimensions

The GPUE codebase currently has the capability of tracking vortices in two dimensions and highlighting vortices in three; however, vortex tracking has yet to be implemented as there are no reliable and general methods for tracking three dimensional vortex structures in superfluid simulations. In addition, the vortex tracking method in GPUE is currently unstable for non-harmonic traps in two dimensions, and in many cases, the user does not know the precise geometry of the trapping system. As such, a generalized vortex tracking methods for n-dimensional simulations is desired.

In 2016, a method for three dimensional vortex tracking was proposed by Villois *et. al.* [131]; however, this method has no computational complexity bound, assumes periodic boundary conditions, and required a large amount of communication between the device and host. We have considered development of a similar method that leverages our vortex highlighting scheme to create n-dimensional vortex skeletons for vortex tracking in two and three dimensions. [ADD MORE IF KEEPING]

5.5.2 General purpose Hamiltonian solver

GPUE.jl

After considering our options, we have begun development of GPUE.jl, the a julia re-write of GPUE.

5.6 Future simulations of quantum systems

In addition to further developments of GPUE, there are also several new simulations that can be performed now on GPU hardware, such are multicomponent simulations with gauge fields and dynamic studies of the system introduced in Chapter 5. Because GPUE allows for the simulation of dynamic variables through expression trees, further three-dimensional STA studies can also be performed.

Ultimately, this work has provided a toolbox for the simulation of various quantum phenomenon that were computational inctractable before now, including the three-dimensional simulations of superfluid turbulence without relying on vortex-filament mthods, and simulations of multicomponent systems with gauge fields. It has also developed novel methods for maximizing the size of the simulated domain with the SSFM, along with tools like the DistributedTranspose.jl that allow for spectral methods to be more widely used in HPC environments.

Appendix A

Simple vector additions in CUDA, OpenCL, and JuliaGPU

This is a compilation of an introductory GPGPU example in three languages (CUDA, OpenCL, and Julia) to show the differences in different approaches to GPGPU computation and the necessary abstractions required by users in order to perform basic tasks using GPGPU

A.1 Vector addition with C++

Firstly, a simple vector addition without GPGPU as a baseline:

```
#include <iostream>

int main(){

    int n = 1024;

    // Initializing host vectors
    double *a, *b, *c;
    a = (double*)malloc(sizeof(double)*n);
    b = (double*)malloc(sizeof(double)*n);
    c = (double*)malloc(sizeof(double)*n);

    // Initializing a and b
    for (size_t i = 0; i < n; ++i){
        a[i] = i;
        b[i] = i;
        c[i] = 0;
    }

    // Vector Addition
    for (size_t i = 0; i < n; ++i){
        c[i] = a[i] + b[i];
    }

    // Check to make sure everything works
    for (size_t i = 0; i < n; ++i){
        if (c[i] != a[i] + b[i]){
```

```
            std::cout << "Yo. You failed. What a loser! Ha\n";
            exit(1);
        }
    }

    std::cout << "You passed the test, congratulations!\n";

    free(a);
    free(b);
    free(c);
}
```

A.2 Vector addition with CUDA

Now for vector addition with CUDA. Here, it is important to note that it is not practical to ask users to write CUDA kernels, as they are not skilled in GPGPU. In addition, direct kernel manipulation would require a recompilation every run, which is cumbersome for most users along with dedicated directories for differently built binaries if running multiple GPUE simulations simultaneously. As such additional techniques are used in GPUE to allow users to write their own functions without recompiling the GPUE codebase every run, and these methods are discussed in Chapter **??**

```
#include <iostream>
#include <math.h>
#include <chrono>

__global__ void vecAdd(double *a, double *b, double *c, int n){

    // Global Thread ID
    int id = blockIdx.x*blockDim.x + threadIdx.x;

    // Check to make sure we are in range
    if (id < n){
        c[id] = a[id] + b[id];
    }
}

int main(){

    int n = 1024;

    // Initializing host vectors
    double *a, *b, *c;
    a = (double*)malloc(sizeof(double)*n);
    b = (double*)malloc(sizeof(double)*n);
    c = (double*)malloc(sizeof(double)*n);

    // Initializing all device vectors
    double *d_a, *d_b, *d_c;

    cudaMalloc(&d_a, sizeof(double)*n);
    cudaMalloc(&d_b, sizeof(double)*n);
```

```
    cudaMalloc(&d_c, sizeof(double)*n);

    // Initializing a and b
    for (size_t i = 0; i < n; ++i){
        a[i] = i;
        b[i] = i;
        c[i] = 0;
    }

    cudaMemcpy(d_a, a, sizeof(double)*n, cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, b, sizeof(double)*n, cudaMemcpyHostToDevice);

    dim3 threads, grid;

    // threads are arbitrarily chosen
    threads = {100, 1, 1};
    grid = {(unsigned int)ceil((float)n/threads.x), 1, 1};
    vecAdd<<<grid, threads>>>(d_a, d_b, d_c, n);

    // Copying back to host
    cudaMemcpy(c, d_c, sizeof(double)*n, cudaMemcpyDeviceToHost);

    // Check to make sure everything works
    for (size_t i = 0; i < n; ++i){
        if (c[i] != a[i] + b[i]){
            std::cout << "Yo. You failed. What a loser! Ha\n";
            exit(1);
        }
    }

    std::cout << "You passed the test, congratulations!\n";

    free(a);
    free(b);
    free(c);

    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);
}
```

A.3 Vector addition with OpenCL

Vector addition with OpenCL has notable advantages to CUDA, namely that users can update the kernels without recompilation. Though it is tempting to use OpenCL due to this, it is notably more cumbersome to write OpenCL code than CUDA, and it we lack the engineering resources to develop an OpenCL variant of GPUE. In addition, Julia provides similar benefits to OpenCL with much higher maintainability.

```
#define __CL_ENABLE_EXCEPTIONS

#include <CL/cl.hpp>
#include <iostream>
```

```
#include <vector>
#include <math.h>

// OpenCL kernel
const char *kernelSource =                                      "\n" \
"#pragma OPENCL EXTENSION cl_khr_fp64 : enable                  \n" \
"__kernel void vecAdd( __global double *a,                      \n" \
"                      __global double *b,                      \n" \
"                      __global double *c,                      \n" \
"                      const unsigned int n){                   \n" \
"                                                               \n" \
"    // Global Tread ID                                         \n" \
"    int id = get_global_id(0);                                 \n" \
"                                                               \n" \
"    // Remain in boundaries                                    \n" \
"    if (id < n){                                               \n" \
"        c[id] = a[id] + b[id];                                 \n" \
"    }                                                          \n" \
"}                                                              \n";

int main(){
    unsigned int n = 1024;

    double *h_a, *h_b, *h_c;

    h_a = new double[n];
    h_b = new double[n];
    h_c = new double[n];

    for (size_t i = 0; i < n; ++i){
        h_a[i] = 1;
        h_b[i] = 1;
    }

    cl::Buffer d_a, d_b, d_c;

    cl_int err = CL_SUCCESS;
    try{
        std::vector<cl::Platform> platforms;
        cl::Platform::get(&platforms);
        if(platforms.size() == 0){
            std::cout << "Platforms size is 0\n";
            return -1;
        }

        cl_context_properties properties[] =
            { CL_CONTEXT_PLATFORM, (cl_context_properties)(platforms[0])(), 0 };

        cl::Context context(CL_DEVICE_TYPE_GPU, properties);
        std::vector<cl::Device> devices = context.getInfo<CL_CONTEXT_DEVICES>();

        cl::CommandQueue queue(context, devices[0], 0, &err);

        d_a = cl::Buffer(context, CL_MEM_READ_ONLY, n*sizeof(double));
        d_b = cl::Buffer(context, CL_MEM_READ_ONLY, n*sizeof(double));
```

```
        d_c = cl::Buffer(context, CL_MEM_WRITE_ONLY, n*sizeof(double));

        queue.enqueueWriteBuffer(d_a, CL_TRUE, 0, n*sizeof(double), h_a);
        queue.enqueueWriteBuffer(d_b, CL_TRUE, 0, n*sizeof(double), h_b);

        cl::Program::Sources source(1,
            std::make_pair(kernelSource,strlen(kernelSource)));
        cl::Program program_ = cl::Program(context, source);
        program_.build(devices);

        cl::Kernel kernel(program_, "vecAdd", &err);

        kernel.setArg(0, d_a);
        kernel.setArg(1, d_b);
        kernel.setArg(2, d_c);
        kernel.setArg(3, n);

        cl::NDRange localSize(64);

        cl::NDRange globalSize((int)(ceil(n/(float)64)*64));

        cl::Event event;
        queue.enqueueNDRangeKernel(
            kernel,
            cl::NullRange,
            globalSize,
            localSize,
            NULL,
            &event
        );

        event.wait();
        queue.enqueueReadBuffer(d_c, CL_TRUE, 0, n*sizeof(double), h_c);
    }
    catch(cl::Error err){
        std::cerr << "ERROR: " << err.what() << "(" << err.err() << ")\n";
    }

    // Check to make sure everything works
    for (size_t i = 0; i < n; ++i){
        if (h_c[i] != h_a[i] + h_b[i]){
            std::cout << "Yo. You failed. What a loser! Ha\n";
            exit(1);
        }
    }

    std::cout << "You passed the test, congratulations!\n";

    delete(h_a);
    delete(h_b);
    delete(h_c);
}
```

A.4 Julia

Julia is a relatively new language, but boasts the performance of CUDA without as much bulk. In addition, Julia allows users to more easily look at the AST for compilation, thus rendering GPUE's AST process obsolete. Here is vector addition in Julia.

```
using CUDAnative, CUDAdrv, CuArrays, Test

function kernel_vadd(a, b, c)
    i = (blockIdx().x-1) * blockDim().x + threadIdx().x
    j = (blockIdx().y-1) * blockDim().y + threadIdx().y
    @inbounds c[i,j] = a[i,j] + b[i,j]
    return nothing
end

function main()

    res =1024

    # CUDAdrv functionality: generate and upload data
    a = round.(rand(Float32, (1024, 1024)) * 100)
    b = round.(rand(Float32, (1024, 1024)) * 100)
    d_a = CuArray(a)
    d_b = CuArray(b)
    d_c = similar(d_a)  # output array

    # run the kernel and fetch results
    # syntax: @cuda [kwargs...] kernel(args...)
    @cuda threads = (128, 1, 1) blocks = (div(res,128),res,1)
       kernel_vadd(d_a, d_b, d_c)

    # CUDAdrv functionality: download data
    # this synchronizes the device
    c = Array(d_c)
    a = Array(d_a)
    b = Array(d_b)

    @test isapprox(a+b, c)
end
```

Bibliography

[1] J. A. Kahle, J. Moreno, and D. Dreps. 2.1 summit and sierra: Designing ai/hpc supercomputers. In *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 42–43, Feb 2019.

[2] Ruymán Reyes, Iván López-Rodríguez, Juan J Fumero, and Francisco De Sande. accull: an openacc implementation with cuda and opencl support. In *European Conference on Parallel Processing*, pages 871–882. Springer, 2012.

[3] Massimiliano Fatica, P LeGresley, I Buck, J Stone, J Phillips, S Morton, and P Micikevicius. High performance computing with cuda. *SC08*, 2008.

[4] Tim Besard, Pieter Verstraete, and Bjorn De Sutter. High-level gpu programming in julia. *arXiv preprint arXiv:1604.03410*, 2016.

[5] Aaftab Munshi, Benedict Gaster, Timothy G Mattson, and Dan Ginsburg. *OpenCL programming guide*. Pearson Education, 2011.

[6] John Cheng, Max Grossman, and Ty McKercher. *Professional Cuda C Programming*. John Wiley & Sons, 2014.

[7] Matteo Frigo and Steven G Johnson. Fftw: An adaptive software architecture for the fft. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181)*, volume 3, pages 1381–1384. IEEE, 1998.

[8] Marius Brehler, Malte Schirwon, Dominik Göddeke, and Peter M Krummrich. A gpu-accelerated fourth-order runge–kutta in the interaction picture method for the simulation of nonlinear signal propagation in multimode fibers. *Journal of Lightwave Technology*, 35(17):3622–3628, 2017.

[9] James Schloss, Albert Benseny, Jérémie Gillet, Jacob Swain, and Thomas Busch. Non-adiabatic generation of noon states in a tonks–girardeau gas. *New Journal of Physics*, 18(3):035012, 2016.

[10] James R Schloss and Lee J Riordan. Gpue: Graphics processing unit gross-pitaevskii equation solver. *J. Open Source Software*, 3(32):1037, 2018.

[11] Tiantian Zhang, James Schloss, Andreas Thomasen, Lee James O'Riordan, Thomas Busch, and Angela White. Chaotic few-body vortex dynamics in rotating bose-einstein condensates. *Physical Review Fluids*, 4(5):054701, 2019.

[12] James Schloss, Peter Barnett, Rashi Sachdeva, and Thomas Busch. Controlled creation of three-dimensional vortex structures in bose–einstein condensates using artificial magnetic fields, 2019.

[13] Govind P Agrawal. Nonlinear fiber optics. In *Nonlinear Science at the Dawn of the 21st Century*, pages 195–211. Springer, 2000.

[14] O. V. Sinkin, R. Holzlohner, J. Zweck, and C. R. Menyuk. Optimization of the split-step fourier method in modeling optical-fiber communications systems. *Journal of Lightwave Technology*, 21(1):61–68, Jan 2003.

[15] T. T. Meirelles, A. A. Rieznik, and H. L. Fragnito. Study on a new split-step fourier algorithm for optical fiber transmission systems simulations. In *SBMO/IEEE MTT-S International Conference on Microwave and Optoelectronics, 2005.*, pages 100–102, July 2005.

[16] Rao Min, Sun Xiao-Han, and Zhang Ming-De. A modified split-step fourier method for optical pulse propagation with polarization mode dispersion. *Chinese Physics*, 12(5):502–506, apr 2003.

[17] Cihan Bayindir. Compressive split-step fourier method. *arXiv preprint arXiv:1512.03932*, 2015.

[18] JAC Weideman and BM Herbst. Split-step methods for the solution of the nonlinear schrödinger equation. *SIAM Journal on Numerical Analysis*, 23(3):485–507, 1986.

[19] Hanquan Wang. Numerical studies on the split-step finite difference method for nonlinear schrödinger equations. *Applied Mathematics and Computation*, 170(1):17–35, 2005.

[20] John Charles Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016.

[21] John Crank and Phyllis Nicolson. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 43, pages 50–67. Cambridge University Press, 1947.

[22] Samuel Daniel Conte and Carl De Boor. *Elementary numerical analysis: an algorithmic approach*, volume 78. SIAM, 2017.

[23] Llewellyn Thomas. Elliptic problems in linear differential equations over a network: Watson scientific computing laboratory. *Columbia Univ., NY*, 1949.

[24] Dominik Goddeke and Robert Strzodka. Cyclic reduction tridiagonal solvers on gpus applied to mixed-precision multigrid. *IEEE Transactions on Parallel and Distributed Systems*, 22(1):22–32, 2010.

[25] HH Wang. A parallel method for tridiagonal equations. *ACM Transactions on Mathematical Software (TOMS)*, 7(2):170–183, 1981.

[26] Roland A Sweet. A cyclic reduction algorithm for solving block tridiagonal systems of arbitrary dimension. *SIAM Journal on Numerical Analysis*, 14(4):706–720, 1977.

[27] Gilbert Strang. On the construction and comparison of difference schemes. *SIAM journal on numerical analysis*, 5(3):506–517, 1968.

[28] Shev MacNamara and Gilbert Strang. Operator splitting. In *Splitting Methods in Communication, Imaging, Science, and Engineering*, pages 95–114. Springer, 2016.

[29] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[30] Gian Carlo Wick. Properties of bethe-salpeter wave functions. *Physical Review*, 96(4):1124, 1954.

[31] Mark Harris et al. Optimizing parallel reduction in cuda. *Nvidia developer technology*, 2(4):70, 2007.

[32] A. B. Migdal. Quantum theory of the monatomic ideal gas, part II. *Physikalisch-mathematische Klasse*, 1:3, 1925.

[33] Alexander L. Fetter and John Dirk Walecka. *Quantum Theory of Many-Particle Systems*. Dover Publications, 2003.

[34] Marvin Girardeau. Relationship between systems of impenetrable bosons and fermions in one dimension. *Journal of Mathematical Physics*, 1(6):516–523, 1960.

[35] P. Nozières and S. Schmitt-Rink. Bose condensation in an attractive fermion gas: From weak to strong coupling superconductivity. *Journal of Low Temperature Physics*, 59(3):195–211, 1985.

[36] Aurel Bulgac, Michael McNeil Forbes, Michelle M Kelley, Kenneth J Roche, and Gabriel Wlazłowski. Quantized superfluid vortex rings in the unitary fermi gas. *Physical review letters*, 112(2):025301, 2014.

[37] Alan Aversa. The gross-pitaevskii equation: A non-linear schrödinger equation, 2008.

[38] N. N. Bogoliubov. On the theory of superfluidity. *J. Phys.(USSR)*, 11:23–32, 1947.

[39] Franco Dalfovo, Stefano Giorgini, Lev P. Pitaevskii, and Sandro Stringari. Theory of Bose-Einstein condensation in trapped gases. *Rev. Mod. Phys.*, 71:463–512, Apr 1999.

[40] E. P. Gross. Structure of a quantized vortex in boson systems. *Il Nuovo Cimento (1955-1965)*, 20(3):454–477, 1961.

[41] L. Pitaevskii. Vortex lines in an imperfect bose gas. *Zh. Eksp. Teor. Fiz*, 40:646, 1961.

[42] C. J. Pethick and H. Smith. *Bose–Einstein Condensation in Dilute Gases*. Cambridge University Press, 2002.

[43] Alexander L. Fetter. Rotating trapped bose-einstein condensates. *Rev. Mod. Phys.*, 81:647–691, May 2009.

[44] Masahito Ueda. *Fundamentals and new frontiers of Bose-Einstein condensation*. World Scientific, 2010.

[45] J. Allen and A.Misener. Flow of liquid Helium II. *Nature*, 142:643, 1938.

[46] A. B. Migdal. A phenomenological approach to the theory of the nucleus. *Soviet Physics JETP*, 10:176, 1960.

[47] M. H. Anderson, J. R. Ensher, M. R. Matthews, C. E. Wieman, and E. A. Cornell. Observation of Bose–Einstein condensation in a dilute atomic vapor. *Science*, 269(5221):198–201, 1995.

[48] AA Abrikosov. The magnetic properties of superconducting alloys. *Journal of Physics and Chemistry of Solids*, 2(3):199–208, 1957.

[49] Alexander L Fetter and Anatoly A Svidzinsky. Vortices in a trapped dilute bose-einstein condensate. *Journal of Physics: Condensed Matter*, 13(12):R135, 2001.

[50] KW Madison, F Chevy, W Wohlleben, and Jl Dalibard. Vortex formation in a stirred bose-einstein condensate. *Physical review letters*, 84(5):806, 2000.

[51] Matthew D. Reichl and Erich J. Mueller. Vortex ring dynamics in trapped Bose–Einstein condensates. *Phys. Rev. A*, 88:053626, Nov 2013.

[52] Carlo F. Barenghi, Ladislav Skrbek, and Katepalli R. Sreenivasan. Introduction to quantum turbulence. *PNAS*, 111:4647, 2014.

[53] R. P. Feynman. *Progress in Low Temperature Physics: Chapter II, Application of Quantum Mechanics to Liquid Helium*, volume 1. Interscience Publishers Inc, 1955.

[54] LJ O'Riordan, AC White, and Th Busch. Moiré superlattice structures in kicked bose-einstein condensates. *Physical Review A*, 93(2):023609, 2016.

[55] Lee James O'Riordan and Thomas Busch. Topological defect dynamics of vortex lattices in bose-einstein condensates. *Physical Review A*, 94(5):053603, 2016.

[56] JR Abo-Shaeer, C Raman, JM Vogels, and Wolfgang Ketterle. Observation of vortex lattices in bose-einstein condensates. *Science*, 292(5516):476–479, 2001.

[57] V. Schweikhard, I. Coddington, P. Engels, V. P. Mogendorff, and E. A. Cornell. Rapidly rotating Bose–Einstein condensates in and near the lowest landau level. *Phys. Rev. Lett.*, 92:040404, Jan 2004.

[58] F Chevy and J Dalibard. Rotating bose-einstein condensates. *Europhysics News*, 37(1):12–16, 2006.

[59] E Hodby, G Hechenblaikner, SA Hopkins, OM Marago, and CJ Foot. Vortex nucleation in bose-einstein condensates in an oblate, purely magnetic potential. *Physical review letters*, 88(1):010405, 2001.

[60] PC Haljan, I Coddington, Peter Engels, and Eric A Cornell. Driving bose-einstein-condensate vorticity with a rotating normal cloud. *Physical review letters*, 87(21):210403, 2001.

[61] Vincent Bretin, Sabine Stock, Yannick Seurin, and Jean Dalibard. Fast rotation of a bose-einstein condensate. *Physical review letters*, 92(5):050403, 2004.

[62] Avinash Kumar, Romain Dubessy, Thomas Badr, Camilla De Rossi, Mathieu de Goër de Herve, Laurent Longchambon, and Hélène Perrin. Producing superfluid circulation states using phase imprinting. *Physical Review A*, 97(4):043615, 2018.

[63] Stuart Moulder, Scott Beattie, Robert P. Smith, Naaman Tammuz, and Zoran Hadzibabic. Quantized supercurrent decay in an annular bose-einstein condensate. *Phys. Rev. A*, 86:013629, Jul 2012.

[64] S. Burger, K. Bongs, S. Dettmer, W. Ertmer, K. Sengstock, A. Sanpera, G. V. Shlyapnikov, and M. Lewenstein. Dark solitons in bose-einstein condensates. *Phys. Rev. Lett.*, 83:5198–5201, Dec 1999.

[65] J Denschlag, Je E Simsarian, Dl L Feder, Charles W Clark, La A Collins, J Cubizolles, Lu Deng, Edward W Hagley, Kristian Helmerson, William P Reinhardt, et al. Generating solitons by phase engineering of a bose-einstein condensate. *Science*, 287(5450):97–101, 2000.

[66] Biao Wu, Jie Liu, and Qian Niu. Controlled generation of dark solitons with phase imprinting. *Physical review letters*, 88(3):034101, 2002.

[67] C. Ryu, M. F. Andersen, P. Cladé, Vasant Natarajan, K. Helmerson, and W. D. Phillips. Observation of persistent flow of a bose-einstein condensate in a toroidal trap. *Phys. Rev. Lett.*, 99:260401, Dec 2007.

[68] Mark Kasevich and Steven Chu. Atomic interferometry using stimulated raman transitions. *Phys. Rev. Lett.*, 67:181–184, Jul 1991.

[69] M Gajda, M Lewenstein, K Sengstock, G Birkl, W Ertmer, et al. Optical generation of vortices in trapped bose-einstein condensates. *Physical Review A*, 60(5):R3381, 1999.

[70] Yong-il Shin, Michele Saba, Mukund Vengalattore, TA Pasquini, Christian Sanner, AE Leanhardt, Mara Prentiss, DE Pritchard, and Wolfgang Ketterle. Dynamical instability of a doubly quantized vortex in a bose-einstein condensate. *Physical review letters*, 93(16):160406, 2004.

[71] Angela C White, Brian P Anderson, and Vanderlei S Bagnato. Vortices and turbulence in trapped atomic condensates. *Proceedings of the National Academy of Sciences*, 111(Supplement 1):4719–4726, 2014.

[72] F Maucher, SA Gardiner, and IG Hughes. Excitation of knotted vortex lines in matter waves. *New Journal of Physics*, 18(6):063016, 2016.

[73] Y.-J. Lin, R. L. Compton, K. Jimenez-Garcia, J. V. Porto, and I. B. Spielman. Synthetic magnetic fields for ultracold neutral atoms. *Nature*, 462(7273):628–632, Dec 2009.

[74] J. Dalibard. Introduction to the physics of artificial gauge fields. *ArXiv e-prints*, April 2015.

[75] Rajiv Bhat. *Bosons in rotating optical lattices*. PhD thesis, Indian Institute of Technology Kanpur, 2001.

[76] A. Tonomura M. Peshkin. *The Aharonov–Bohm Effect*, volume 340. Springer-Verlag, 1989.

[77] J Werschnik and EKU Gross. Quantum optimal control theory. *Journal of Physics B: Atomic, Molecular and Optical Physics*, 40(18):R175, 2007.

[78] D Guéry-Odelin, A Ruschhaupt, A Kiely, E Torrontegui, S Martínez-Garaot, and JG Muga. Shortcuts to adiabaticity: concepts, methods, and applications. *arXiv preprint arXiv:1904.08448*, 2019.

[79] Frank L Lewis, Draguna Vrabie, and Vassilis L Syrmos. *Optimal control*. John Wiley & Sons, 2012.

[80] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[81] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 01 1965.

[82] John R Koza. Genetic programming. 1997.

[83] Tamara G Kolda, Robert Michael Lewis, and Virginia Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM review*, 45(3):385–482, 2003.

[84] Robert Michael Lewis, Anne Shepherd, and Virginia Torczon. Implementing generating set search methods for linearly constrained minimization. *SIAM Journal on Scientific Computing*, 29(6):2507–2530, 2007.

[85] Hartmut Pohlheim. Examples of objective functions. *Retrieved*, 4(10):2012, 2007.

[86] Erik Torrontegui, Sara Ibánez, Sofia Martínez-Garaot, Michele Modugno, Adolfo del Campo, David Guéry-Odelin, Andreas Ruschhaupt, Xi Chen, and Juan Gonzalo Muga. Shortcuts to adiabaticity. In *Advances in atomic, molecular, and optical physics*, volume 62, pages 117–169. Elsevier, 2013.

[87] H Ralph Lewis Jr and WB Riesenfeld. An exact quantum theory of the time-dependent harmonic oscillator and of a charged particle in a time-dependent electromagnetic field. *Journal of Mathematical Physics*, 10(8):1458–1473, 1969.

[88] H Ralph Lewis and PGL Leach. A direct approach to finding exact invariants for one-dimensional time-dependent classical hamiltonians. *Journal of Mathematical Physics*, 23(12):2371–2374, 1982.

[89] Y. Israel, I. Afek, S. Rosen, O. Ambar, and Y. Silberberg. Experimental tomography of noon states with large photon numbers. *Phys. Rev. A*, 85:022115, Feb 2012.

[90] Itai Afek, Oron Ambar, and Yaron Silberberg. High-noon states by mixing quantum and classical light. *Science*, 328(5980):879–881, 2010.

[91] M. D. Girardeau, E. M. Wright, and J. M. Triscari. Ground-state properties of a one-dimensional system of hard-core bosons in a harmonic trap. *Phys. Rev. A*, 63:033601, Feb 2001.

[92] M. D. Girardeau and E. M. Wright. Measurement of one-particle correlations and momentum distributions for trapped 1d gases. *Phys. Rev. Lett.*, 87:050403, Jul 2001.

[93] John C Slater. The theory of complex spectra. *Physical Review*, 34(10):1293, 1929.

[94] Kunal K. Das, M. D. Girardeau, and E. M. Wright. Interference of a thermal tonks gas on a ring. *Phys. Rev. Lett.*, 89:170404, Oct 2002.

[95] M. D. Girardeau and A. Minguzzi. Motion of an impurity particle in an ultracold quasi-one-dimensional gas of hard-core bosons. *Phys. Rev. A*, 79:033610, Mar 2009.

[96] David W. Hallwood, Thomas Ernst, and Joachim Brand. Robust mesoscopic superposition of strongly correlated ultracold atoms. *Phys. Rev. A*, 82:063623, Dec 2010.

[97] C Schenke, A Minguzzi, and FWJ Hekking. Probing superfluidity of a mesoscopic tonks-girardeau gas. *Physical Review A*, 85(5):053627, 2012.

[98] Andreas Nunnenkamp, Ana Maria Rey, and Keith Burnett. Generation of macroscopic superposition states in ring superlattices. *Phys. Rev. A*, 77:023622, Feb 2008.

[99] David W Hallwood, Keith Burnett, and Jacob Dunningham. The barriers to producing multiparticle superposition states in rotating bose–einstein condensates. *Journal of Modern Optics*, 54(13-15):2129–2148, 2007.

[100] S. Martínez-Garaot, A. Ruschhaupt, J. Gillet, Th. Busch, and J. G. Muga. Fast quasiadiabatic dynamics. *Phys. Rev. A*, 92:043406, Oct 2015.

[101] A. del Campo. Long-time behavior of many-particle quantum decay. *Phys. Rev. A*, 84:012113, Jul 2011.

[102] K. Lelas, T. Ševa, and H. Buljan. Loschmidt echo in one-dimensional interacting bose gases. *Phys. Rev. A*, 84:063601, Dec 2011.

[103] Xi Chen and J. G. Muga. Transient energy excitation in shortcuts to adiabaticity for the time-dependent harmonic oscillator. *Phys. Rev. A*, 82:053403, Nov 2010.

[104] Xi Chen, A. Ruschhaupt, S. Schmidt, A. del Campo, D. Guéry-Odelin, and J. G. Muga. Fast optimal frictionless atom cooling in harmonic traps: Shortcut to adiabaticity. *Phys. Rev. Lett.*, 104:063002, Feb 2010.

[105] Shumpei Masuda and Katsuhiro Nakamura. Fast-forward of adiabatic dynamics in quantum mechanics. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 466(2116):1135–1154, 2009.

[106] E. Torrontegui, S. Ibáñez, Xi Chen, A. Ruschhaupt, D. Guéry-Odelin, and J. G. Muga. Fast atomic transport without vibrational heating. *Phys. Rev. A*, 83:013415, Jan 2011.

[107] Shumpei Masuda. Acceleration of adiabatic transport of interacting particles and rapid manipulations of a dilute bose gas in the ground state. *Phys. Rev. A*, 86:063624, Dec 2012.

[108] C.F. Phelan, T. Hennessy, and Th. Busch. Shaping the evanescent field of optical nanofibers for cold atom trapping. *Opt. Express*, 21(22):27093–27101, Nov 2013.

[109] Shumpei Masuda, Katsuhiro Nakamura, and Adolfo del Campo. High-fidelity rapid ground-state loading of an ultracold gas into an optical lattice. *Phys. Rev. Lett.*, 113:063003, Aug 2014.

[110] JR Gurd. A taxonomy of parallel computer architectures. In *1988 International Specialist Seminar on the Design and Application of Parallel Digital Processors*, pages 57–61. IET, 1988.

[111] Kenneth Czechowski, Casey Battaglino, Chris McClanahan, Kartik Iyer, P-K Yeung, and Richard Vuduc. On the communication complexity of 3d ffts and its implications for exascale. In *Proceedings of the 26th ACM international conference on Supercomputing*, pages 205–214. ACM, 2012.

[112] P Wittek. Comparing three numerical solvers of the gross-pitaevskii equation, 2016.

[113] Xavier Antoine and Romain Duboscq. Gpelab, a matlab toolbox to solve gross–pitaevskii equations i: Computation of stationary solutions. *Computer Physics Communications*, 185(11):2969–2991, 2014.

[114] Peter Wittek and Fernando M Cucchietti. A second-order distributed trotter–suzuki solver with a hybrid cpu–gpu kernel. *Computer Physics Communications*, 184(4):1165–1171, 2013.

[115] Michael Garland, Scott Le Grand, John Nickolls, Joshua Anderson, Jim Hardwick, Scott Morton, Everett Phillips, Yao Zhang, and Vasily Volkov. Parallel computing experiences with cuda. *IEEE micro*, 28(4):13–27, 2008.

[116] Victor W Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, et al. Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu. *ACM SIGARCH computer architecture news*, 38(3):451–460, 2010.

[117] John Nickolls and William J Dally. The gpu computing era. *IEEE micro*, 30(2):56–69, 2010.

[118] Sandra Wienke, Paul Springer, Christian Terboven, and Dieter an Mey. Openacc—first experiences with real-world applications. In *European Conference on Parallel Processing*, pages 859–870. Springer, 2012.

[119] Rohit Chandra, Leo Dagum, David Kohr, Ramesh Menon, Dror Maydan, and Jeff McDonald. *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[120] Mark Harris. An efficient matrix transpose in cuda c/c++. *Retrieved July*, 26:2018, 2013.

[121] Denis Foley and John Danskin. Ultra-performance pascal gpu and nvlink interconnect. *IEEE Micro*, 37(2):7–17, 2017.

[122] Vladimir Lončar, Luis E Young-S, Srdjan Škrbić, Paulsamy Muruganandam, Sadhan K Adhikari, and Antun Balaž. Openmp, openmp/mpi, and cuda/mpi c programs for solving the time-dependent dipolar gross–pitaevskii equation. *Computer Physics Communications*, 209:190–196, 2016.

[123] Hao Wang, Sreeram Potluri, Devendar Bureddy, Carlos Rosales, and Dhabaleswar K Panda. Gpu-aware mpi on rdma-enabled clusters: Design, implementation and evaluation. *IEEE Transactions on Parallel and Distributed Systems*, 25(10):2595–2605, 2013.

[124] Tim Besard, Valentin Churavy, Alan Edelman, and Bjorn De Sutter. Rapid software prototyping for heterogeneous and distributed platforms. *Advances in Engineering Software*, 132:29–46, 2019.

[125] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[126] Tim Besard, Christophe Foket, and Bjorn De Sutter. Effective extensible programming: unleashing julia on gpus. *IEEE Transactions on Parallel and Distributed Systems*, 30(4):827–841, 2018.

[127] Lee James O'Riordan. *Non-equilibrium vortex dynamics in rapidly rotating Bose-Einstein condensates*. PhD thesis, Okinawa Institute of Science and Technology Graduate University, 2017.

[128] Robert F Cohen and Roberto Tamassia. Dynamic expression trees and their applications. In *SODA*, pages 52–61, 1991.

[129] Ruyman Reyes and Francisco de Sande. Automatic code generation for gpus in llc. *The Journal of Supercomputing*, 58(3):349–356, 2011.

[130] James Schloss and Lee O'riordan.

[131] Alberto Villois, Giorgio Krstulovic, Davide Proment, and Hayder Salman. A vortex filament tracking method for the gross–pitaevskii model of a superfluid. *Journal of Physics A: Mathematical and Theoretical*, 49(41):415502, 2016.

[132] Yulong Guo, Xiaopei Liu, Chi Xiong, Xuemiao Xu, and Chi-Wing Fu. Towards high-quality visualization of superfluid vortices. *IEEE transactions on visualization and computer graphics*, 24(8):2440–2455, 2018.

[133] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[134] Jose L Jodra, Ibai Gurrutxaga, and Javier Muguerza. Efficient 3d transpositions in graphics processing units. *International Journal of Parallel Programming*, 43(5):876–891, 2015.

[135] Gregory Ruetsch and Massimiliano Fatica. *CUDA Fortran for scientists and engineers: best practices for efficient CUDA Fortran programming*. Elsevier, 2013.

[136] Steven H Strogatz. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC Press, 2018.

[137] EA Spiegel. Chaos: a mixed metaphor for turbulence. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 413(1844):87–95, 1987.

[138] Luca Biferale, Guido Boffetta, A Celani, BJ Devenish, A Lanotte, and Federico Toschi. Lagrangian statistics of particle pairs in homogeneous isotropic turbulence. *Physics of Fluids*, 17(11):115101, 2005.

[139] Arjun Berera and Richard DJG Ho. Chaotic properties of a turbulent isotropic fluid. *Physical review letters*, 120(2):024101, 2018.

[140] SK Nemirovskii and W Fiszdon. Chaotic quantized vortices and hydrodynamic processes in superfluid helium. *Reviews of Modern Physics*, 67(1):37, 1995.

[141] Nikos Kyriakopoulos, Vassilis Koukouloyannis, Charalampos Skokos, and Panayotis G Kevrekidis. Chaotic behavior of three interacting vortices in a confined bose-einstein condensate. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 24(2):024410, 2014.

[142] Vassilis Koukouloyannis, George Voyatzis, and Panayotis G Kevrekidis. Dynamics of three noncorotating vortices in bose-einstein condensates. *Physical Review E*, 89(4):042905, 2014.

[143] R Navarro, R Carretero-González, PJ Torres, PG Kevrekidis, DJ Frantzeskakis, MW Ray, E Altuntaş, and DS Hall. Dynamics of a few corotating vortices in bose-einstein condensates. *Physical review letters*, 110(22):225301, 2013.

[144] Caroline Nore, Malek Abid, and ME Brachet. Kolmogorov turbulence in low-temperature superflows. *Physical review letters*, 78(20):3896, 1997.

[145] Steven R Stalp, L Skrbek, and Russell J Donnelly. Decay of grid turbulence in a finite channel. *Physical review letters*, 82(24):4831, 1999.

[146] Tsunehiko Araki, Makoto Tsubota, and Sergey K Nemirovskii. Energy spectrum of superfluid turbulence with no normal-fluid component. *Physical review letters*, 89(14):145301, 2002.

[147] Julien Salort, Christophe Baudet, Bernard Castaing, Benoît Chabaud, François Daviaud, Thomas Didelot, Pantxo Diribarne, Bérengère Dubrulle, Yves Gagne, Frédéric Gauthier, et al. Turbulent velocity spectra in superfluid flows. *Physics of Fluids*, 22(12):125102, 2010.

[148] Hassan Aref and N Pomphrey. Integrable and chaotic motions of four vortices. i. the case of identical vortices. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 380(1779):359–387, 1982.

[149] Hassan Aref and Neil Pomphrey. Integrable and chaotic motions of four vortices. *Physics Letters A*, 78(4):297–300, 1980.

[150] Hassan Aref. Integrable, chaotic, and turbulent vortex motion in two-dimensional flows. *Annual Review of Fluid Mechanics*, 15(1):345–389, 1983.

[151] Sang Won Seo, Bumsuk Ko, Joon Hyun Kim, and Yong-il Shin. Observation of vortex-antivortex pairing in decaying 2d turbulence of a superfluid gas. *Scientific reports*, 7(1):4587, 2017.

[152] Kali E Wilson, Zachary L Newman, Joseph D Lowney, and Brian P Anderson. In situ imaging of vortices in bose-einstein condensates. *Physical Review A*, 91(2):023621, 2015.

[153] DV Freilich, DM Bianchi, AM Kaufman, TK Langin, and DS Hall. Real-time dynamics of single vortex lines and vortex dipoles in a bose-einstein condensate. *Science*, 329(5996):1182–1185, 2010.

[154] Simone Serafini, Luca Galantucci, Elena Iseni, Tom Bienaimé, Russell N Bisset, Carlo F Barenghi, Franco Dalfovo, Giacomo Lamporesi, and Gabriele Ferrari. Vortex reconnections and rebounds in trapped atomic bose-einstein condensates. *Physical Review X*, 7(2):021031, 2017.

[155] TW Neely, AS Bradley, EC Samson, SJ Rooney, Ewan M Wright, KJH Law, R Carretero-González, PG Kevrekidis, MJ Davis, and Brian P Anderson. Characteristics of two-dimensional quantum turbulence in a compressible superfluid. *Physical review letters*, 111(23):235301, 2013.

[156] Woo Jin Kwon, Geol Moon, Jae-yoon Choi, Sang Won Seo, and Yong-il Shin. Relaxation of superfluid turbulence in highly oblate bose-einstein condensates. *Physical Review A*, 90(6):063627, 2014.

[157] Guillaume Gauthier, Matthew T Reeves, Xiaoquan Yu, Ashton S Bradley, Mark Baker, Thomas A Bell, Halina Rubinsztein-Dunlop, Matthew J Davis, and Tyler W Neely. Negative-temperature onsager vortex clusters in a quantum fluid. *arXiv preprint arXiv:1801.06951*, 2018.

[158] Shaun P Johnstone, Andrew J Groszek, Philip T Starkey, Christopher J Billington, Tapio P Simula, and Kristian Helmerson. Order from chaos: Observation of large-scale flow from turbulence in a two-dimensional superfluid. *arXiv preprint arXiv:1801.06952*, 2018.

[159] Amandine Aftalion and Qiang Du. Vortices in a rotating bose-einstein condensate: Critical angular velocities and energy diagrams in the thomas-fermi regime. *Physical Review A*, 64(6):063603, 2001.

[160] Alexandra V Zampetaki, R Carretero-González, PG Kevrekidis, FK Diakonos, and DJ Frantzeskakis. Exploring rigidly rotating vortex configurations and their bifurcations in atomic bose-einstein condensates. *Physical Review E*, 88(4):042914, 2013.

[161] Tiantian Zhang, James Schloss, Andreas Thomasen, Lee James O'Riordan, Thomas Busch, and Angela White.

[162] Alan Wolf, Jack B Swift, Harry L Swinney, and John A Vastano. Determining lyapunov exponents from a time series. *Physica D: Nonlinear Phenomena*, 16(3):285–317, 1985.

[163] Daniel H Wacks, Andrew W Baggaley, and Carlo F Barenghi. Large-scale superfluid vortex rings at nonzero temperatures. *Physical Review B*, 90(22):224514, 2014.

[164] B. P. Anderson, P. C. Haljan, C. A. Regal, D. L. Feder, L. A. Collins, C. W. Clark, and E. A. Cornell. Watching dark solitons decay into vortex rings in a bose-einstein condensate. *Phys. Rev. Lett.*, 86:2926–2929, Apr 2001.

[165] Mark JH Ku, Biswaroop Mukherjee, Tarik Yefsah, and Martin W Zwierlein. Cascade of solitonic excitations in a superfluid fermi gas: From planar solitons to vortex rings and lines. *Physical review letters*, 116(4):045304, 2016.

[166] Michael Robin Matthews, Brian P Anderson, PC Haljan, DS Hall, CE Wieman, and Eric A Cornell. Vortices in a bose-einstein condensate. *Physical Review Letters*, 83(13):2498, 1999.

[167] Tarik Yefsah, Ariel T Sommer, Mark JH Ku, Lawrence W Cheuk, Wenjie Ji, Waseem S Bakr, and Martin W Zwierlein. Heavy solitons in a fermionic superfluid. *Nature*, 499(7459):426, 2013.

[168] T. E. Faber. *Fluid Dynamics for Physicists*. Cambridge University Press, 1995.

[169] Ira M. Cohen David R. Dowling, Pijush K. Kundu. *Fluid Mechanics*. Academic Press, Boston, fifth edition edition, 2012.

[170] D. J. Tritton. *Physical Fluid Dynamics*. Oxford University Press, 1988.

[171] L. D. Landau and E. M. Lifshitz. *Fluid Mechanics*. Butterworth-Heinemann, 1987.

[172] K. W. Schwarz. Three-dimensional vortex dynamics in superfluid ^{4}He: Line-line and line-boundary interactions. *Phys. Rev. B*, 31:5782–5804, May 1985.

[173] S. Zuccher, M. Caliari, A. W. Baggaley, and C. F. Barenghi. Quantum vortex reconnections. *Physics of Fluids*, 24(12), 2012.

[174] R. Donnelly. *Quantized Vortices in Helium II*. Cambridge University Press, 1991.

[175] C F Barenghi and R J Donnelly. Vortex rings in classical and quantum systems. *Fluid Dynamics Research*, 41(5):051401, 2009.

[176] P. H. Roberts and J. Grant. Motions in a bose condensate. I: The structure of a large circular vortex. *J. Phys.*, 4:55, 1971.

[177] P. H. Roberts and R. J. Donnelly. Dynamics of vortex rings. *Phys. Lett.*, 31A:137, 1970.

[178] L. E. Fraenkel. On steady vortex rings of small cross-section in an ideal fluid. *Proceedings of the Royal Society of London A: Mathematical , Physical and Engineering Sciences*, 316(1524):29–62, 1970.

[179] L. E. Fraenkel. Examples of steady vortex rings of small cross-section in an ideal fluid. *Journal of Fluid Mechanics*, 51:119–135, 1 1972.

[180] A. Sommerfield. *Mechanics of Deformable Bodies: Lectures of Theoretical Physics*, volume 2. Academic Press, 1960.

[181] R. M. Caplan, J. D. Talley, and P. G. Carretero-González, R. a nd Kevrekidis. Scattering and leapfrogging of vortex rings in a superfluid. *Physics of Fluids*, 26(9), 2014.

[182] K Shariff and A Leonard. Vortex rings. *Annual Review of Fluid Mechanics*, 24(1):235–279, 1992.

[183] K. W. Schwarz. Interaction of quantized vortex rings with quantized vortex lines in rotating he II. *Phys. Rev.*, 165:323–334, Jan 1968.

[184] T. T. Lim and T. B. Nickels. *Fluid Vortices*. Springer Netherlands, Dordrecht, 1995.

[185] Matthew S. Paoletti and Daniel P. Lathrop. Quantum turbulence. *Annual Review of Condensed Matter Physics*, 2(1):213–234, 2011.

[186] B. Jackson, J. F. McCann, and C. S. Adams. Vortex line and ring dynamics in trapped Bose–Einstein condensates. *Phys. Rev. A*, 61:013604, Dec 1999.

[187] C A Jones and P H Roberts. Motions in a bose condensate. iv. axisymmetric solitary waves. *Journal of Physics A: Mathematical and General*, 15(8):2599, 1982.

[188] Natalia G. Berloff. Solitary waves on vortex lines in ginzburg-landau models for the example of bose-einstein condensates. *Phys. Rev. Lett.*, 94:010403, Jan 2005.

[189] J. Ruostekoski and J. R. Anglin. Creating vortex rings and three-dimensional skyrmions in Bose–Einstein condensates. *Phys. Rev. Lett.*, 86:3934–3937, Apr 2001.

[190] Naomi S Ginsberg, Joachim Brand, and Lene Vestergaard Hau. Observation of hybrid soliton vortex-ring structures in bose-einstein condensates. *Physical review letters*, 94(4):040403, 2005.

[191] I Shomroni, E Lahoud, S Levy, and J Steinhauer. Evidence for an oscillating soliton/vortex ring by density engineering of a bose–einstein condensate. *Nature Physics*, 5(3):193, 2009.

[192] Janne Ruostekoski and Zachary Dutton. Engineering vortex rings and systems for controlled studies of vortex interactions in bose-einstein condensates. *Physical Review A*, 72(6):063626, 2005.

[193] Florian Pinsker, Natalia G Berloff, and Víctor M Pérez-García. Nonlinear quantum piston for the controlled generation of vortex rings and soliton trains. *Physical Review A*, 87(5):053624, 2013.

[194] M. Abad, M. Guilleumas, R. Mayol, and M. Pi. Vortex rings in toroidal Bose–Einstein condensates. *Laser Physics*, 18(5):648–652, 2008.

[195] Dustin Kleckner, Louis H Kauffman, and William TM Irvine. How superfluid vortex knots untie. *Nature Physics*, 12(7):650, 2016.

[196] Renzo L Ricca, David C Samuels, and Carlo F Barenghi. Evolution of vortex knots. *Journal of Fluid Mechanics*, 391:29–44, 1999.

[197] Callum W Duncan, Calum Ross, Niclas Westerberg, Manuel Valiente, Bernd J Schroers, and Patrik Öhberg. Linked and knotted synthetic magnetic fields. *Physical Review A*, 99(6):063613, 2019.

[198] J Dalibard and F Gerbier. Juzeliunas, and p. öhberg. *Rev. Mod. Phys*, 83:1523, 2011.

[199] M. Mochol and K. Sacha. Artificial magnetic field induced by an evanescent wave. *Scientific Reports*, 5, Jan 2015. Article.

[200] Jonathan M Ward, Danny G O'Shea, Brian J Shortt, Michael J Morrissey, Kieran Deasy, and Síle G Nic Chormaic. Heat-and-pull rig for fiber taper fabrication. *Review of scientific instruments*, 77(8):083105, 2006.

[201] Limin Tong, Rafael R Gattass, Jonathan B Ashcom, Sailing He, Jingyi Lou, Mengyan Shen, Iva Maxwell, and Eric Mazur. Subwavelength-diameter silica wires for low-loss optical wave guiding. *Nature*, 426(6968):816, 2003.

[202] Amnon Yariv et al. *Optical electronics in modern communications*, volume 1. Oxford University Press, USA, 1997.

[203] E Vetsch, D Reitz, G Sagué, R Schmidt, ST Dawkins, and A Rauschenbeutel. Optical interface created by laser-cooled atoms trapped in the evanescent field surrounding an optical nanofiber. *Physical review letters*, 104(20):203603, 2010.

[204] C Lacroûte, KS Choi, A Goban, DJ Alton, D Ding, NP Stern, and HJ Kimble. A state-insensitive, compensated nanofiber trap. *New Journal of Physics*, 14(2):023056, 2012.

[205] Thomas Nieddu, Vandna Gokhroo, and Síle Nic Chormaic. Optical nanofibres and neutral atoms. *Journal of Optics*, 18(5):053001, 2016.

[206] G Sagué, E Vetsch, W Alt, D Meschede, and A Rauschenbeutel. Cold-atom physics using ultrathin optical fibers: Light-induced dipole forces and surface interactions. *Physical review letters*, 99(16):163602, 2007.

[207] Laura Russell, Kieran Deasy, Mark J Daly, Michael J Morrissey, and Síle Nic Chormaic. Sub-doppler temperature measurements of laser-cooled atoms using optical nanofibres. *Measurement Science and Technology*, 23(1):015201, 2011.

[208] Ravi Kumar, Vandna Gokhroo, Kieran Deasy, Aili Maimaiti, Mary C Frawley, Ciarán Phelan, and Síle Nic Chormaic. Interaction of laser-cooled 87rb atoms with higher order modes of an optical nanofibre. *New Journal of Physics*, 17(1):013026, 2015.

[209] Fam Le Kien, V. I. Balykin, and K. Hakuta. Atom trap and waveguide using a two-color evanescent light field around a subwavelength-diameter optical fiber. *Phys. Rev. A*, 70:063403, Dec 2004.

[210] Rashi Sachdeva and Thomas Busch. Creating superfluid vortex rings in artificial magnetic fields. *Physical Review A*, 95(3):033615, 2017.

[211] Marco Cozzini, Sandro Stringari, Vincent Bretin, Peter Rosenbusch, and Jean Dalibard. Scissors mode of a rotating bose-einstein condensate. *Physical Review A*, 67(2):021602, 2003.

[212] D Guéry-Odelin and S Stringari. Scissors mode and superfluidity of a trapped bose-einstein condensed gas. *Physical review letters*, 83(22):4452, 1999.

[213] OM Marago, SA Hopkins, J Arlt, E Hodby, G Hechenblaikner, and CJ Foot. Observation of the scissors mode and evidence for superfluidity of a trapped bose-einstein condensed gas. *Physical review letters*, 84(10):2056, 2000.

[214] NL Smith, WH Heathcote, JM Krueger, and CJ Foot. Experimental observation of the tilting mode of an array of vortices in a dilute bose-einstein condensate. *Physical review letters*, 93(8):080406, 2004.

[215] Francesca Zambelli and Sandro Stringari. Quantized vortices and collective oscillations of a trapped bose-einstein condensate. *Physical review letters*, 81(9):1754, 1998.

[216] S Stringari. Superfluid gyroscope with cold atomic gases. *Physical review letters*, 86(21):4725, 2001.

[217] Vladimir Georgievich Minogin and Síle Nic Chormaic. Manifestation of the van der waals surface interaction in the spontaneous emission of atoms into an optical nanofiber. *Laser Physics*, 20(1):32–37, 2010.