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**

October, 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: October, 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 ultracold atomic 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 demonstrated on physical systems 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.

Contents

List of Tables

Introduction

Massively parallel methods have become commonplace in High-Performance Computing (HPC) environments that 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 and pseudo-spectral methods are interesting subsets of problems that are used for large-scale, distributed computation on HPC environments that rely on FFTs (Fast Fourier Transforms) to solve partial differential equations of various forms. Though there are robust FFT libraries, like FFTW [7] to perform distributed FFTs [8], FFTs are still global operations, often requiring memory manipulation on multiple nodes simultaneously and requiring communication between them. For this reason, the FFT is often the computational bottleneck for spectral and pseudo-spectral methods. It is difficult to directly benchmark GPU and CPU software, but it is generally accepted that one-dimensional GPU-based FFTs perform more optimally as the grid size increases; however, for higher-dimensional FFT operations, this performance increase is not as drastic [9]. This leads to an interesting 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, a few pseudo-spectral methods have been shown to be both easier to implement and faster than other force integration schemes like Runge-Kutta. One such method is the Split-Step Fourier Method (SSFM) [10].

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 in 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; however, three-dimensional simulations can quickly become computational infeasible. For this reason, it is worth exploring and developing GPU-based libraries for the computation of superfluid dynamics. I will discuss this work and also motivate several methods for simulation of quantum engineering on GPU devices that I have developed. The structure of my thesis is as follows:

Introduction to the SSFM for vortex simulations

I will begin with an introduction to the SSFM, 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 discuss several settings 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 that are difficult to perform on GPU architecture. This work has been published in the New Journal of Physics **18**(3):035012, 2016 [11].

Introduction to GPGPU and the GPUE codebase

This chapter will introduce the concept of General-Purpose GPU computing and the GPUE (GPU-based Gross-Piteavskii Equation solver) 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 with other GPU libraries with similar purposes. This chapter will conclude with a discussion of a notoriously difficult problem 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 **3**(32):1037, 2018 [12].

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 **4**(5):054701, 2019 [13].

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 (arXiv:1910.02364) [14].

Outlook

Throughout this text, I will try to motivate future directions at the end of each chapter; however, a global outlook, including new simulations possible with GPUE and other areas of development that could be focused on will be discussed in the end.

Chapter 1

Introduction to the SSFM for simulating superfluid vortex systems

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 [15–18] and in various quantum systems [19–21]. Though other methods, such as explicit and implicit Euler [22], Crank-Nicholson [23], and Runge-Kutta [22], can solve similar differential equations, the SSFM has distinct advantages over these methods. For example, because the SSFM primarily relies on embarrassingly parallel element-wise matrix multiplications and Fast Fourier Transform (FFT) routines that have been optimized for parallel and distributed systems, is often much easier to parallelize than Runge-Kutta [10]. 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 [24, 25] which are also not easily parallelizable [26–28]. Though much of this work focuses on using the SSFM to simulate superfluid vortex states, I will not be discussing alternative methods, such as point-vortex [29] or vortex-filament [30] 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 in this chapter, 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 considerations for simulating quantum systems with the SSFM. As such, I will also introduce important physical insights for understanding ultracold atomic systems that will be used throughout the rest of this work. In particular, I will fo-

cus on understanding superfluid systems created by Bose–Einstein condensation and methods by which vortex structures can be generated and controlled in a Bose–Einstein Condensate (BEC).

1.1 The SSFM

Let us begin with the one-dimensional Schrödinger equation,

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

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, one often replace most of the right-hand side of the equation with a Hamiltonian operator, which for this case would be $\hat{\mathcal{H}} = \frac{\hat{p}^2}{2m} + V_0(\mathbf{r})$. 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, any wavefunction can be expanded into a complete set of eigenkets of the Hamiltonian, with $\hat{\mathcal{H}}\left|\phi_n\right\rangle = E_n\left|\phi_n\right\rangle$. Which allows one to write

$$\left|\Psi(\mathbf{r})\right\rangle = \sum_{n=0}^{\infty} c_n\left|\phi_n(\mathbf{r})\right\rangle, \tag{1.2}$$

where 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, one first assumes 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 system is being simulated in a timestepping mannor with a series of small timesteps, one 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, the dt^2 error can be decreased 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 [31],

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

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 [32]. 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. Here, the first-order terms for the expansion in Equation (1.6) is identical to the expansion of $e^{h\mathbf{A}}e^{h\mathbf{B}}$; however, when expanding the last term, one finds,

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

Here, the $\mathbf{BA}$ term is missing in 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 one can 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.

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

where $\hat{U}_r = e^{-\frac{i\hat{\mathcal{H}}_r dt}{2\hbar}}$, $\hat{U}_p = e^{-\frac{i\hat{\mathcal{H}}_p dt}{\hbar}}$, 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), typically using 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 [33]. 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. I 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, one can simulate simple, one-dimensional quantum systems. For example, if one uses a Gaussian wavefunction which is offset from the center of a simple harmonic oscillator, one can simulate the wavefunction oscillation, as shown in Figure 1.1(a).

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

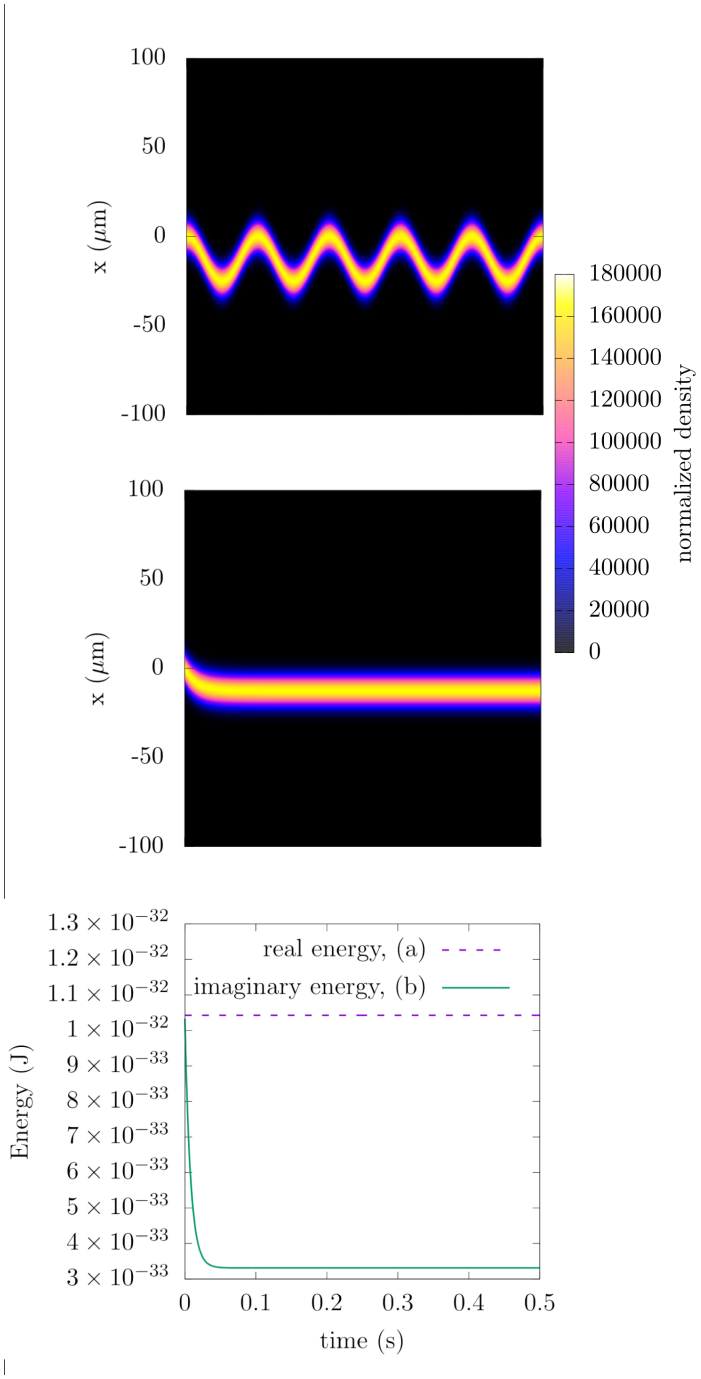

Figure 1.1: Evolution of a one-dimensional simple harmonic oscillator in (a) real, and (b) imaginary time after slightly shifting the trapping potential in the $\hat{x}$ direction. In (c), the energy as a function of time and show that the energy of the system when evolving in real time remains constant, 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, I use a ^{87} Rb atom with $\omega_x = 10$Hz on a 256-point grid of size 200 μm, at $t = 0$, 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 [12].

time [34]. This changes the solution from the complex sinusoid shown in Equation (1.4) to an exponential decay,

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

Note that it is not necessary to include the c_n coefficients into the expansion as this form of time stepping is no longer unitary. Overall, imaginary time evolution has two notable effects:

1. There will be an exponential decay of all energy states, with higher-energy states decaying faster than the ground-state.

2. The non-unitary evolution requires renormalization at every time step in 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, the wavefunction density is shifting to the center of the trap, and in Figure 1.1(c), the energy decaying to the known ground-state energy of a quantum harmonic oscillator of $\frac{1}{2}\hbar\omega = 3.31 \times 10^{-33}$J for the chosen parameters. 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.

The energy is computed as,

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

For many systems, one can assume the simulation has reached the ground state when the energy converges to a fixed value. More quantitatively, this means that the simulation can be stopped 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 problematic from a software perspective. In practice, every step in imaginary time requires a relatively costly renormalization step with

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

Computationally, this operation is 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 [35]. 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, I will show other methods in Chapter 3 when I discuss the implementation of the SSFM on graphics processing units.

It is important to note that the complexity of the SSFM is similar to the complexity of a convolution via the convolutional theorem,

$$f * g = (F)^{-1} \left(\mathcal{F}(f) \cdot \mathcal{F}(g) \right). \tag{1.12}$$

Here, f and g are arbitrarily chosen functions to be convolved. In fact, interpreting the method as a series of convolutions can lead to important insight as to how the method operates. For example, in imaginary time, the momentum-space operator becomes a Gaussian and the position-space operator becomes a more tightly-confining potential. In this way, every time-step in imaginary time corresponds to a blurring operation with the momentum-space step and strong confinement with the position-space steps. Real-time propagation can be interpreted in a similar way, as the momentum-space operator is similar to the Fourier transform of two Sobel filters, thereby performing two spatial derivatives. In Chapter 3, I will discuss this further in combination with how derivatives can be performed on GPU architecture, but for now I will turn the 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 the single particle 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 [36, 37], and this state of matter is known as a Bose–Einstein Condensate (BEC). A 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 [11, 38]. It is also possible for interacting fermions to condense into a BEC-like state by forming molecules with integer spin [39, 40]; however, I will not discuss fermionic systems further in this work. For now, I will focus on BEC systems, but will also discuss Tonks–Girardeau gases later in Chapter 2.

1.2.1 Bose–Einstein condensation and the Gross–Pitaevskii Equation

To motivate the formalism regularly used to describe BEC systems, I will follow a straightforward derivation using the second quantization [41]. As mentioned in Section 1.2, bosons in a BEC occupy a single ground state, meaning one must introduce a many-body Hamiltonian for the system and take inter-particle interactions into account. Because experimental systems are dilute, I 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

$$\hat{\mathcal{H}} = \int \hat{\Psi}^\dagger(\mathbf{r}) \left[-\frac{\hbar^2}{2m}\nabla^2 + V_0(\mathbf{r}) \right] \hat{\Psi}(\mathbf{r}) d\mathbf{r} + \frac{1}{2} \int \hat{\Psi}^\dagger(\mathbf{r}) \hat{\Psi}^\dagger(\mathbf{r}') V(\mathbf{r}-\mathbf{r}') \hat{\Psi}(\mathbf{r}') \hat{\Psi}(\mathbf{r}) d\mathbf{r} d\mathbf{r}', \tag{1.13}$$

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 for the atomic field that follow the bosonic commutation relations,

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

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

$$[\hat{\Psi}(\mathbf{r}), \hat{\Psi}(\mathbf{r})] = 0. \tag{1.16}$$

In the case of a BEC at $T \approx 0$, one can perform a Bogoliubov expansion [42, 43]

$$\hat{\Psi}(\mathbf{r}, t) = \Phi(\mathbf{r}, t) + \delta\hat{\Phi}(\mathbf{r}, t), \tag{1.17}$$

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 quantum fluctuations of the BEC system; therefore, the condensate density is defined as

$$n(\mathbf{r}, t) = |\Phi(\mathbf{r}, t)|^2. \tag{1.18}$$

Now one may use the Heisenberg equation of motion,

$$i\hbar \frac{\partial}{\partial t} \hat{\Psi}(\mathbf{r}, t) = [\hat{\Psi}, \hat{H}], \tag{1.19}$$

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

$$\frac{\partial}{\partial t} \hat{\Psi}(\mathbf{r}, t) = \frac{1}{i\hbar} \left[-\frac{\hbar^2}{2m}\nabla^2 + V_0(\mathbf{r}) + \int d\mathbf{r}' \hat{\Psi}^\dagger(\mathbf{r}', t) V(\mathbf{r}'-\mathbf{r}) \hat{\Psi}(\mathbf{r}', t) \right] \hat{\Psi}(\mathbf{r}, t), \tag{1.20}$$

which follows from Equation (1.13) after integrating over $\mathbf{r}$. If it is assumed that two bosons will only interact with a contact potential of the form

$$V(\mathbf{r}'-\mathbf{r}) = g\delta(\mathbf{r}'-\mathbf{r}), \tag{1.21}$$

which has a strength given by

$$g \equiv \frac{4\pi\hbar^2 a_s}{m}, \tag{1.22}$$

where a_s is the species and state-dependent s-wave scattering length, we may write the evolution for the wavefunction as

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

This is celebrated the Gross–Pitaevskii Equation (GPE), which is known as a governing equation for the physics of dilute BEC systems. When written in the time-independent form it determines the chemical potential μ of the condensate as [44, 45]

$$\mu\Phi(\mathbf{r}) = \left(-\frac{\hbar^2}{2m}\nabla^2 + V_0(\mathbf{r}) + g|\Phi(\mathbf{r})|^2\right)\Phi(\mathbf{r}). \tag{1.24}$$

The time-dependent GPE allows one 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 [37, 46, 47]. The main difference between this equation and the Schrödinger equation is the nonlinear interaction term $g|\Psi|^2$, 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, one can find an analytical solution for the wavefunction density, known as a Thomas–Fermi distribution, shown in Figure 1.2 for a single dimension and varying interaction strengths.

The Thomas–Fermi distribution 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 [48], so that the interaction energy exceeds the kinetic energy and it reads,

$$\Psi_{\mathrm{TF}}(\mathbf{r}) = \sqrt{\frac{[\mu - V(\mathbf{r})]\Theta(\mu - V(\mathbf{r}))}{g}} \tag{1.25}$$

where Θ is the Heaviside step function that ensures the density stays positive. The shape of this function is that of an inverted parabola for a harmonic trap, and the radius in any direction from the center can be found to be

$$R_{\mathrm{TF}} = \sqrt{\frac{2\mu}{m\omega_i^2}}, \tag{1.26}$$

where ω_i is the trapping frequency in the ith direction. Using the normalization condition, $\int_{-\infty}^{\infty}|\Psi(\mathbf{r},t)|^2 = N$, the chemical potential in the Thomas–Fermi limit becomes

$$\mu_{\mathrm{TF}} = \frac{h\omega_i}{2}\left(\frac{15Na_s}{a}\right)^{2/5}. \tag{1.27}$$

where $a = \sqrt{\hbar/m\bar{\omega}}$ and $\bar{\omega}$ is the average of all the harmonic trapping frequencies. Using Equations (1.27) and (1.26) one finds that

$$R_{\mathrm{TF}} = a\left(\frac{15Na_s}{a}\right)^{1/5}\frac{\bar{\omega}}{\omega_i} \tag{1.28}$$

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

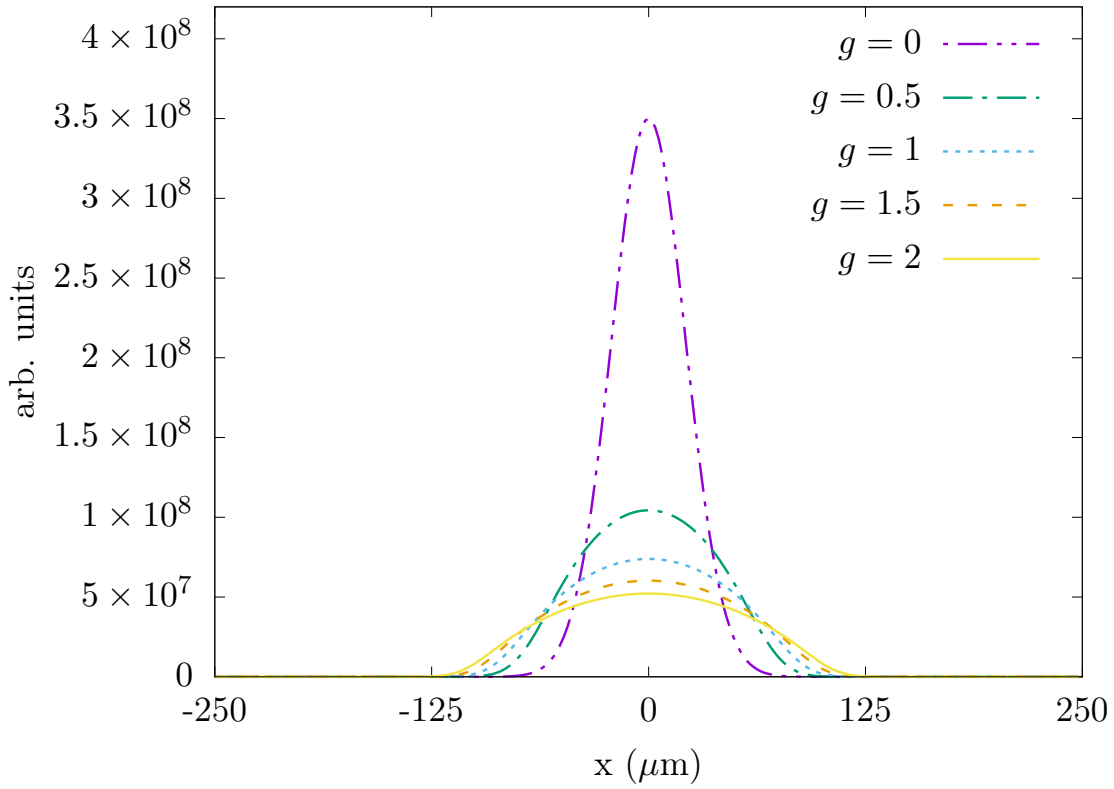

Figure 1.2: Ground state of simple harmonic oscillator for the Schrödinger equation (purple, dashed) and the GPE with varying interaction strengths (all other lines). Here, the GPE solution follows a Thomas-Fermi distribution, while the Schrödinger equation is Gaussian. This simulation was performed with GPUE [12] 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.

systems in which superfluidity can exist, such as ^{4}He (sometimes called Helium II when in its superfluid phase) [49], neutron stars [50], or BEC systems [36, 51]. BEC systems are generally cleaner experimental systems to create, as they do not have a classical fluid fraction, like ^{4}He. They are therefore well-suited systems to study excitations related to superfluid flow.

1.3 Superfluid systems and vortex dynamics

Next, I will focus on the differences between vortex dynamics in classical and superfluid systems 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. For this discussion, it is worthwhile to discuss the hydrodynamic description of a BEC, following the text of Pethick and Smith [46]. To start, I will rewrite the condensate wavefunction as

$$\Psi(\mathbf{r},t) = \sqrt{\rho(\mathbf{r},t)}e^{i\psi(\mathbf{r},t)}, \tag{1.29}$$

with

$$\rho(\mathbf{r},t) = \Psi(\mathbf{r},t)^*\Psi(\mathbf{r},t) = |\Psi(\mathbf{r},t)|^2, \tag{1.30}$$

where $\psi(\mathbf{r},t)$ is the BEC phase. By multiplying the GPE by $\Psi^*(\mathbf{r},t)$ and subtracting the complex conjugate, one can obtain the continuity equation [46],

$$\frac{\partial}{\partial t}\rho(\mathbf{r},t) + \nabla\cdot\mathbf{J}(\mathbf{r},t) = 0, \tag{1.31}$$

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

$$\mathbf{j}(\mathbf{r},t) = \frac{-i\hbar}{2m}\left(\Psi^*(\mathbf{r},t)\nabla\Psi(\mathbf{r},t) - \Psi(\mathbf{r},t)\nabla\Psi^*(\mathbf{r},t)\right). \tag{1.32}$$

By substituting Equation (1.29) into Equation (1.32), the form of the current density for the GPE will be,

$$\mathbf{j}(\mathbf{r},t) = |\Psi(\mathbf{r},t)|^2\frac{\hbar}{m}\nabla\phi(\mathbf{r},t). \tag{1.33}$$

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

$$\mathbf{v}(\mathbf{r},t) = \frac{\mathbf{j}(\mathbf{r},t)}{\rho(\mathbf{r},t)} = \frac{\hbar}{m}\nabla\phi(\mathbf{r},t). \tag{1.34}$$

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 wavefunction has to be single-valued, any rotation in a finite system has to be quantized as,

$$\oint \mathbf{v}\cdot d\ell = \frac{\hbar}{m}2\pi\ell, \tag{1.35}$$

where, ℓ is the integer charge of the circulation. Equation (1.35) shows the quantized nature of circulation in a superfluid with each vortex hosting multiples of 2π charges. This means that every singly-charged vortex in a BEC will have a 2π phase winding, and an example of one vortex in a two-dimensional condensate can be seen in Figure 1.3 (a and c). Equation (1.35) also indicates that the phase is not defined at the center of the vortex; however, the condensate circumvents this problem by requiring the density at this point to be zero. The density dip from the normal condensate density to zero happens over the scale of the healing length, which is

$$\xi = \frac{1}{\sqrt{8\pi\rho_b a_s}} \tag{1.36}$$

for repulsive interactions, 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 then be written as

$$\Psi(\mathbf{r}) = |\Psi(\mathbf{r})|e^{i\ell\phi}, \tag{1.37}$$

which provides and energy (Equation (1.10)) for the GPE of ,

$$E = \int_{-\infty}^{\infty} \frac{\hbar^2}{2m}\left(|\nabla\Psi(\mathbf{r})|^2 + \frac{|\Psi(\mathbf{r})|^2\ell^2 m\mathbf{v}^2}{2}\right) + V_0(\mathbf{r})|\Psi(\mathbf{r})|^2 + \frac{g}{2}|\Psi(\mathbf{r})|^4. \tag{1.38}$$

Because $E \propto \ell^2$, as a superfluid is spun faster, a vortex will not grow in angular momentum, but multiple vortices with $\ell = 1$ will spawn instead [46]. 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; therefore, 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 [52, 53]. This behavior is identical to that of type II superconductors under the effects of a magnetic field. An example of a vortex lattice and its phase can be seen in Figure 1.3 (b and d).

Until now, I have focused primarily on vortex structures in two-dimensions; however, 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 surface of the condensate [54] or reconnect in the form of vortex rings or other, more complicated vortex structures [55, 56]. 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 sound waves at the reconnection site [57].

Three dimensional vortex structures in BECs are difficult to controllably generate experimentally, but I 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. In Chapter 4, I will also discuss important aspects of simulating two-dimensional condensate systems. For now, I will begin discussing the three primary processes to generate vortex structures in superfluid systems: rotation, phase imprinting, and artificial magnetic fields.

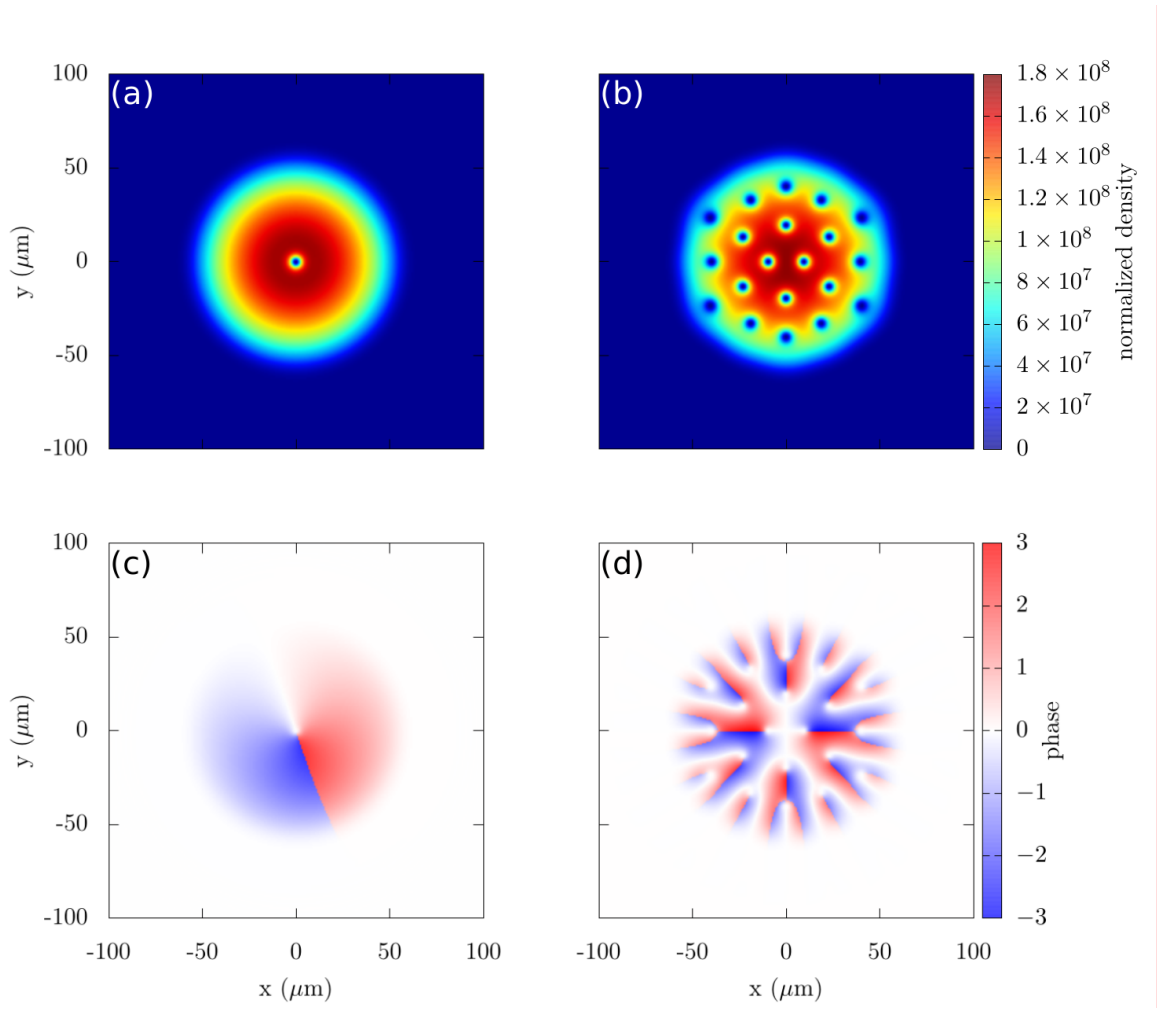

Figure 1.3: Simulation of rotation with a single vortex (a, b) and a vortex lattice (c, d). The wavefunction density is shown in a and c, while the corresponding phase is shown in b and d. A rotation of $\Omega = 0.35\omega_x$ was used for a and b, while a rotation of $\Omega = 0.99\omega_x$ was used for c and d. Here, a ^{87} Rb atom is used with $\omega_x = 2\pi$Hz on a 512-point grid of size 200 μm. This simulation was performed with the GPUE codebase [12], and the phase plots are created by multiplying the phase by the wavefunction density to remove anomalous noise beyond the BEC boundary. [WIP]

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, one simply need to append the angular momentum operator $L_z = -i\hbar(xp_y - yp_x)$ to the GPE,

$$i\hbar\frac{\partial\Psi(\mathbf{r},t)}{\partial t} = \left(\frac{p^2}{2m} + V_0 + g|\Psi(\mathbf{r},t)|^2 - \Omega L_z\right)\Psi(\mathbf{r},t), \tag{1.39}$$

where Ω is the rotation frequency.

In order to generate a vortex via rotation in a harmonic trap, one must rotate faster than the critical velocity of $\Omega_c \approx 0.7\omega_\perp$, where $\omega_\perp$ is the trapping frequency perpendicular to the axis of rotation [47]. 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 [58–61].

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 ultracold atomic systems [62]. For this method, bosons are confined to a magnetic trap and an anisotropic potential is superimposed that rotates with the desired angular velocity [54, 60, 63, 64]. For rotation velocities close to the harmonic trapping frequency, additional methods must be used to ensure the atoms remain in-place. These methods include adding an extra confining potential [65], or the evaporative spin-up technique, where atoms with less angular momentum are evaporated such that the remaining atoms have a higher rotation speed [61, 66].

In this work, I use rotation in a qualitatively similar way to Equation (1.39); however, I will later introduce artificial magnetic fields, which is a broader framework that encompass rotation and will be used in all 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 [67–71]. To generate vortices in a BEC, this technique relies on imprinting a 2π phase winding onto a ground state condensate wavefunction, after which the density adjusts to zero at regions near the phase singularity. Experimentally, phase imprinting can be done in a number of ways. As an example, the phase could be imprinted with a two-photon Raman process to transfer orbital angular momentum to atoms from a Laguerre–Gaussian beam [68, 72]. Another method is through pulsing a spatially dependent potential for a short time when compared to the trapping frequency, which will imprint its potential energy onto the phase of the system [73] and can be used to generate solitons [70], vortices [74], or other states with quantized circulation [67]. 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 [59]. 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 [75].

For the purposes of this work, I will only consider imprinting vortices in two-dimensional settings by applying phase imprinting operations, such that

$$\Psi_{\mathrm{IMP}}(x,y,t) = |\Psi(x,y,t)|e^{i(\theta(x,y,t)+\theta_{\mathrm{IMP}}(x,y))}, \tag{1.40}$$

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 [76, 77]; 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 [78]. In addition, artificial magnetic fields create a broader framework that encompasses rotational effects previously shown in Section 1.3.1, and I will use this framework instead of rotation for simulations in this work. For this, a detailed introduction, similar to that given by Dalibard in [79], will be presented below.

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

$$\hat{\mathcal{H}} = \frac{(\hat{\mathbf{p}} - q\mathbf{A}(\mathbf{r}))^2}{2m} \tag{1.41}$$

where $\mathbf{A}$ is a vector potential such that the magnetic field is given by $\mathbf{B} = \nabla \times \mathbf{A}$ and q is the charge of the particle. Because cold atoms are neutral, one 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

$$\mathbf{F}_{\mathrm{Coriolis}} = 2m\mathbf{v} \times \mathbf{\Omega}, \tag{1.42}$$

which is formally similar to the Lorentz force law. By applying the transformation

$\hat{\mathcal{H}} = \hat{\mathcal{H}}_0 - \Omega\hat{L}_z$, where $\hat{L}_z = x\partial_y - y\partial_x$, one finds [80]

$$\begin{aligned}\hat{\mathcal{H}} &= -\frac{\hbar^2}{2m}\nabla^2 + \frac{1}{2}m\omega^2(x^2+y^2) - \frac{\hbar\Omega}{i}(x\partial_y - y\partial_x) \\ &= \frac{1}{2m}\left(\frac{\hbar}{i}\nabla - m(\boldsymbol{\Omega}\times\mathbf{r})\right)^2 + \frac{m}{2}\left(\omega^2 - \Omega^2\right)r^2 \\ &= \frac{(\hat{\mathbf{p}} - m\mathbf{A}(\mathbf{r}))^2}{2m} + V_0(\mathbf{r}),\end{aligned} \tag{1.43}$$

where ω is the trapping frequency for a symmetric two-dimensional harmonic trap, $\mathbf{A} \equiv \boldsymbol{\Omega}\times\mathbf{r}$, 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 $2\boldsymbol{\Omega} \propto \mathbf{B}$. In this way, one can recreate the rotation expected from the Lorentz force law in a cold atomic system with an artificial magnetic field [54, 60, 81]. 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 and further discussions of how these modifications can be applied on GPU architecture can be found in Chapter 3.

Geometric Gauge Fields

As I have already described how rotation can act as an artificial Lorentz force, I will now turn the attention towards methods that might allow us to generate more general rotational effects and vortex structures. In particular, I will discuss the adiabatic motion of free atoms undergoing geometric phase transformations through a Berry phase. For this, one can assume that the system has an external parameter λ such that

$$\hat{H}(\lambda)\left|\psi_n(\lambda)\right\rangle = E_n(\lambda)\left|\psi_n(\lambda)\right\rangle, \tag{1.44}$$

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

$$\left|\psi(t)\right\rangle = \sum_n c_n(t)\left|\psi_n(\lambda(t))\right\rangle, \tag{1.45}$$

where λ evolves slowly with time. If one considers the initial state of the system to be

$$c_l(0) = 1, \qquad c_n(0) = 0, \qquad \text{for all } n \neq l, \tag{1.46}$$

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

$$i\hbar\dot{c}_l = [E_l(t) - \dot{\lambda}\cdot\mathbf{A}_l(\lambda)]c_l, \tag{1.47}$$

where

$$\mathbf{A}_l(\lambda) = i\hbar\left\langle\psi_l|\nabla\psi_l\right\rangle. \tag{1.48}$$

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

$$\mathbf{B}_l = \nabla\times\mathbf{A}_l. \tag{1.49}$$

Now imagine that the λ parameter follows the closed contour C such that $\lambda(T) = \lambda(0)$. By integrating Equation (1.47), one finds

$$c_l(t) = e^{i\Phi_{\mathrm{dyn}}(t)} e^{i\Phi_{\mathrm{B}}(T)} c_l(0), \tag{1.50}$$

where

$$\begin{aligned} \Phi_{\mathrm{dyn}}(T) &= -\frac{1}{\hbar}\int_0^T E_l(t)dt \\ \Phi_{\mathrm{Berry}}(T) &= \frac{1}{\hbar}\int_0^T \dot{\lambda}\cdot\mathbf{A}_l(\lambda)dt = \frac{1}{\hbar}\oint \mathbf{A}_l(\lambda)\cdot d\lambda. \end{aligned} \tag{1.51}$$

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 (1.50) are gauge invariant and thus remain unchanged when $|\psi_n(\lambda)\rangle$ is multiplied by a phase factor. The Berry phase allows us to transfer angular momentum into a BEC and generate a vortex geometry. This has been experimentally demonstrated like those formed in the 2009 by Lin *et al.* [78]. 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 simple, straight lines. A method to generate these gauge fields in an experimentally realizable way with the evanescent field of a dielectric system undergoing total internal reflection will be described in Chapter 5, but for now I will discuss how these fields can be implemented in the SSFM, with an emphasis on their effects for superfluid simulations.

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, one must modify the Hamiltonian, such that

$$\hat{\mathcal{H}} = \frac{(p - m\mathbf{A})^2}{2m} + V_0 + g|\Psi(\mathbf{r},t)|^2, \tag{1.52}$$

where $\mathbf{A}$ is an artificial vector potential. As a note, some texts absorb the mass term into the artificial vector potential, but here I am stating it explicitly for clarity. When expanded, 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, $-\left(\frac{p\mathbf{A}+\mathbf{A}p}{2}\right)$.

Firstly, let us discuss the component purely in position space. Here, 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 one might be able to balance this warping with a centripetal force

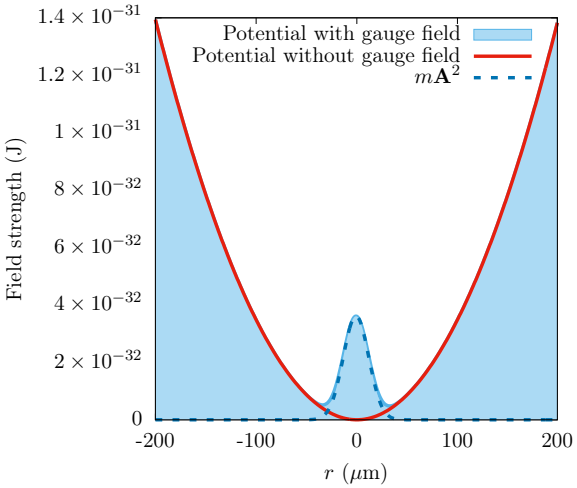

Figure 1.4: Simulation of GPE with an arbitrarily chosen Gaussian artificial vector potential. For this simulation, I have removed the pA term to focus on the warped potential, itself.

tailored to the gauge fields introduced, I did not consider this for the simulations within this work.

The components of the artificial vector potential that are partially in position and momentum space are somewhat difficult to consider numerically. For many physical examples, such as rotation, $\frac{\partial \mathbf{A}_i}{\partial i} = 0$ for $i \in x, y, z$, which means the only relevant term is $-\frac{\mathbf{A}_i p_i}{2}$. This will effectively create a variable that resides in both position and momentum space, and multiplication with this operator can be performed with a one-dimensional FFT across n-dimensional data on the wavefunction first. This means that if there are three operators, $\mathbf{A}_x p_x \hat{x}$, $\mathbf{A}_y p_y \hat{y}$, and $\mathbf{A}_z p_z \hat{z}$, one needs to perform an FFT on the wavefunction in the $\hat{x}$, $\hat{y}$, and $\hat{z}$ dimensions, respectively, before performing an element-wise multiplication. As I will discuss in Chapter 3, this has significant performance penalties if one does 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.

If $\frac{\partial \mathbf{A}_i}{\partial i} \neq 0$, then the other operator, $-\frac{p_i \mathbf{A}_i}{2}$, must also be considered. To perform this operation, a derivative must be performed on $\mathbf{A}$ before application to the wavefunction. This operation doubles the complexity of the application of artificial magnetic fields to the system.

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

Chapter 2

Engineering NOON states in for one-dimensional quantum gases

Until recently, simulations 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 [85] and Shortcuts To Adiabaticity (STA) [86]. 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 (TG) gas [11]. To start, I will discuss the field of optimization algorithms before moving to STA protocols and a physical system using both in practice.

The work in this chapter has been published in New Journal of Physics [11], and in this publication, I performed all calculations for all the figures generated and focused on optimal control methods. The STA protocol was devised by Jérémie Gillet, and the research was supervised by Albert Benseny and Thomas Busch.

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 [87]. 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 in 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 [88], the Nelder–Mead or simplex method [89], genetic algorithms [90], 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 time-dependent control parameters necessary to transform an initial state to a final, desired state [85]. 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 one is attempting to replicate. This problem can be re-framed as an attempt to find the maximum of a fidelity *landscape*, where each point in the domain is a calculation of the fidelity, itself. This means that each point in the fidelity landscape necessarily involves solving the Schrödinger equation for chosen control parameters. For this reason, I 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 [89, 91, 92]. 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 further discussed 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 tetrahedron 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, I 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 [89].

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 convergence 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 minimum. 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, P^*, is higher in value than all other points, the method then keeps whichever 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, P^{**}, 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, P^{**}, is lower than the previous highest point, the contracted point is kept, otherwise, the entire simplex is contracted 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 [93] can be seen in Figure 2.1.

Even though the Nelder–Mead method is a heuristic approach and can become stuck in a local minimum, as long as a sufficient number of random simplexes are chosen at the start of the simulation, it can be used to find an adequately optimal solution. 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 optimal control

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

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

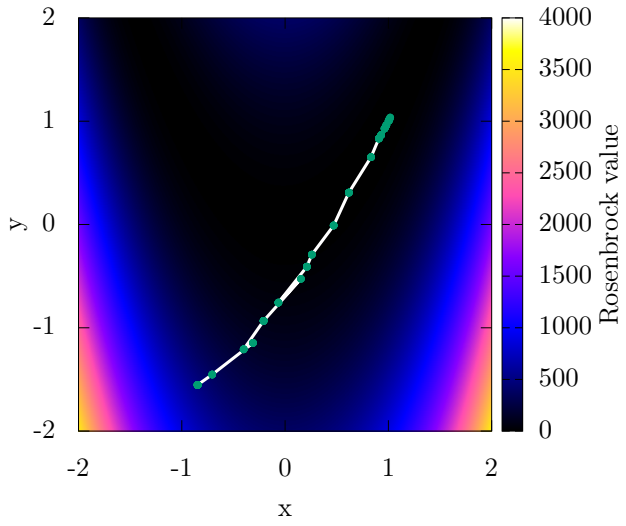

Figure 2.1: Plot of the centroid locations (green dots connected with white line) for the Nelder–Mead method while optimizing the Rosenbrock banana function, $f(x, y) = (a - x)^2 + b(y - x^2)^2$, with $a = 1$ and $b = 100$. Here, centroid locations start at $(-0.851, -1.553)$ and end at the known minimum of $(1, 1)$ in 19 iterations. Here, $\alpha = 1$, $\beta = 0.5$, and $\gamma = 1.5$.

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, one might use

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

which satisfies the provided conditions. This then transforms the optimal control problem into an optimization of the 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 requires 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, 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 I will show 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 [86]. 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, I will introduce only the invariant-based inverse-engineering approach using Lewis–Riesenfeld invariants [94]. 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 [11, 86, 94].

With the method of Lewis-Riesenfeld invariants, the theory for relating different eigenstates of a time-dependent, Hermitian invariant to the solutions to the Schrödinger equation [95] can be applied to systems with time-dependent Hamiltonians, such that

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

where $I(t)$ is the invariant. This ensures that the expectation values for the 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,

$$|\Psi(t)\rangle = \sum_{n=1}^{\infty} c_n \, |\psi_n(t)\rangle \tag{2.8}$$

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

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 [95]

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

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

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

that 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 [96],

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

there will be an invariant that is quadratic in momentum,

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

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

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

$$\ddot{x}_c + \omega^2(t)x_c = F(t)/m, \tag{2.16}$$

with ω_0 as a 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 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 other hand, 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, I am 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. 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 moving to the precise method of NOON state generation for this example.

2.3.1 Tonks–Girardeau gas

As mentioned 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 the repulsive interaction strength $g \to \infty$. In this case, the bosons cannot be at the same location, which acts formally similar to the Pauli-exclusion principle for fermionic systems. In this case, the bosonic Hamiltonian can be solved by the Bose–Fermi mapping theorem [97, 98], 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 one can use the Slater determinant [99],

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

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 [100, 101], 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 [102]. 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 [102]

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

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,

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

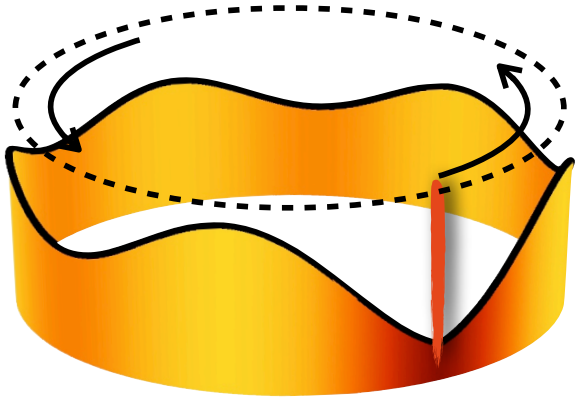

Figure 2.2: Schematic of the system described by Hallwood *et al.* [102]. Here, the density profile for five atoms in a TG gas is shown being stirred by a highly localized potential, indicated by the vertical line.

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 rotational symmetry is broken and avoided crossings appear in the energy spectrum. This makes transitions between different manifolds possible [103].

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 [102]. 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 [104]; however, for a delta barrier, the gap size stays relatively constant to first-order approximation [105].

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 I 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, this pulse will be modified with procedurally generated sinusoidal functions, calculating the fidelity, and then using the Nelder–Mead method to optimize the result. This will allow one 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, I will show the optimal pulse for the cases where I manipulate the rotational velocity, the barrier height, and both.

For STA protocols, the acceleration process will be split into two, one that breaks the rotational symmetry and another that accelerates the atoms. At the end of the

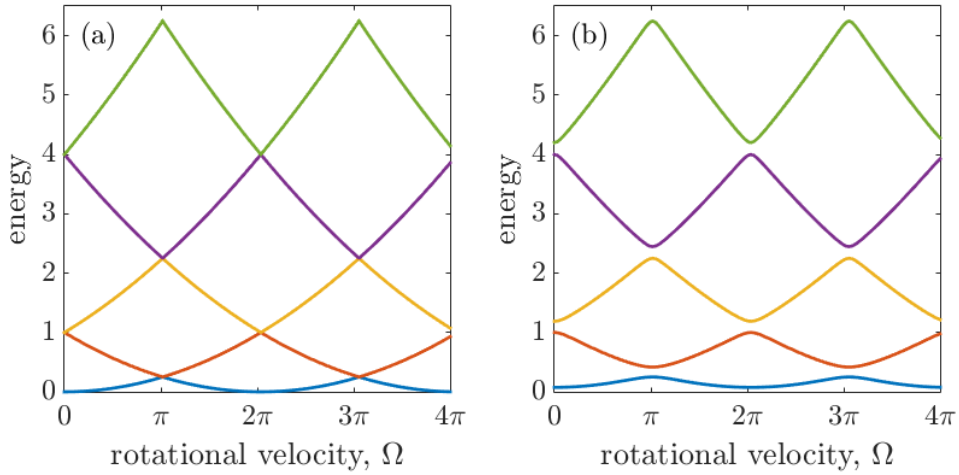

Figure 2.3: Single-particle energy spectrum as a function of Ω for a barrier height of (a) $b = 0$ and (b) $b = 2$. When a barrier is present in the system, avoided crossings appear in the energy spectrum which grow as the barrier strength increases.

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 states in a TG gas has also been created with some similarities [106].

For both of these methods, instead of calculating the fidelity, itself, I calculate the *infidelity*, which is simply $1 - \mathcal{F} = 1 - |\langle\Psi|\Phi\rangle|^2$, as the function to minimize. 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 [107, 108],

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

which follows directly from the form of the TG state [38]

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

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 kept constant and one assumes 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

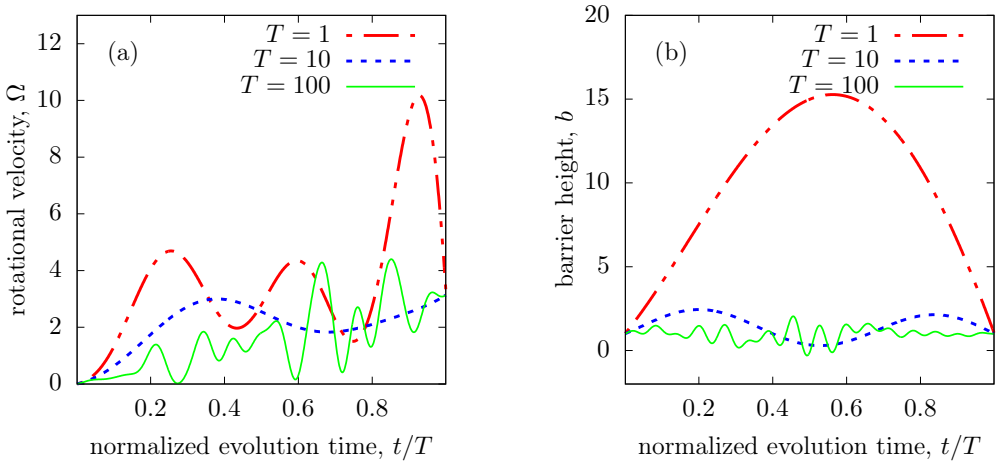

Figure 2.4: Accelerating a single particle from the ground state with $J = 15$. (a) Optimal rotational velocity pulses for $T = 1$, 10, and 100 for fixed barrier height $b = 1$. (b) Optimal barrier height for a linearly increasing rotational velocity, $\Omega = \pi t/T$ for $T = 1$, 10, and 100.

are many local maxima for the fidelity, and as such, longer evolution times effectively produce noisy signals and 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, 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, simple linear ramp for Ω and an initial final height for the barrier at $b = 1$ were chosen. 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, where (a) is the modified rotational frequency and (b) is the barrier height. 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 single variable. When comparing the fidelities in Figure 2.6, it is clear that optimizations over rotation, alone provide the simplest method to optimize the fidelity. It is likely that each run of the

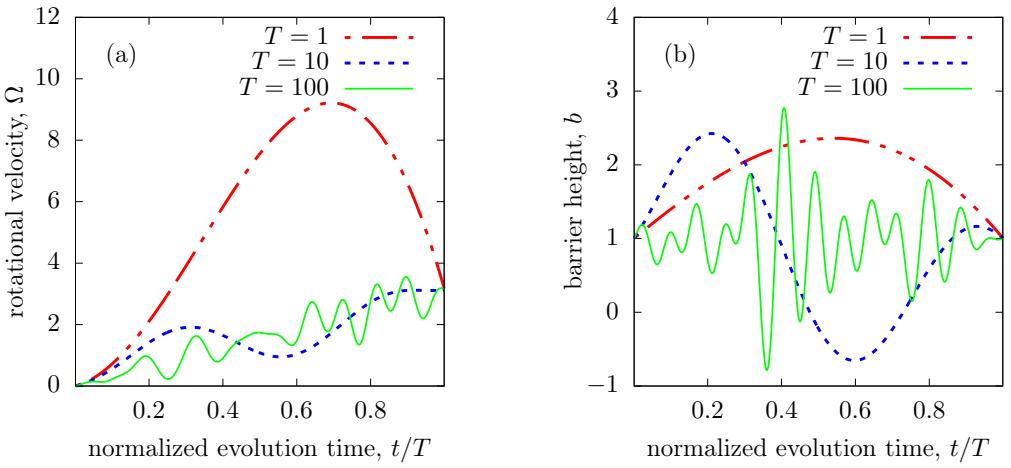

Figure 2.5: Optimal pulses to accelerate a single particle initially in the ground state of the trap for $T = 1$, 10, and 100 for the (a) rotational velocity and (b) barrier height when optimizing over both simultaneously.

CRAB method is 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 [106]. In Figure 2.7, I show the fidelity for the particle closest to the Fermi edge and the entire TG gas for $N = 3$ and 5. In this figure, one can see that by performing the optimization for the atoms near the Fermi edge, one 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 better 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, I 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 must be broken by introducing a time-dependent external potential that is removed in the end.

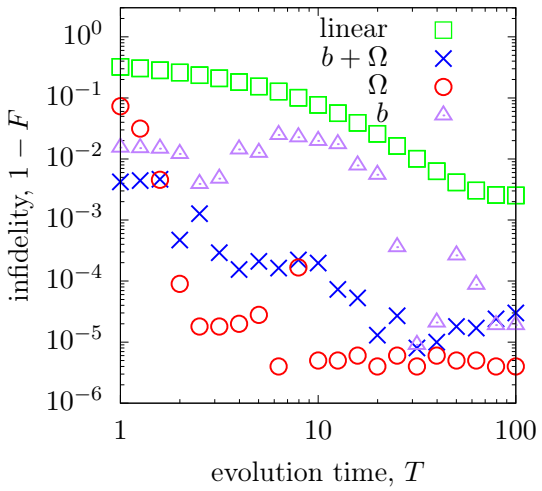

Figure 2.6: Infidelities as a function of the overall process time for optimally controlled rotational acceleration, barrier height, or both. Here, 'linear' refers to an unoptimized linear acceleration from $\Omega = 0$ to π while keeping the barrier height fixed at $b = 1$.

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, one needs to fulfill 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 shortcuts being used for this system involves raising and lowering a harmonic potential [109, 110]. For this shortcut, a stationary harmonic potential is required and F, x_c, and U from Equation (2.13) can all be set to zero, leading to

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

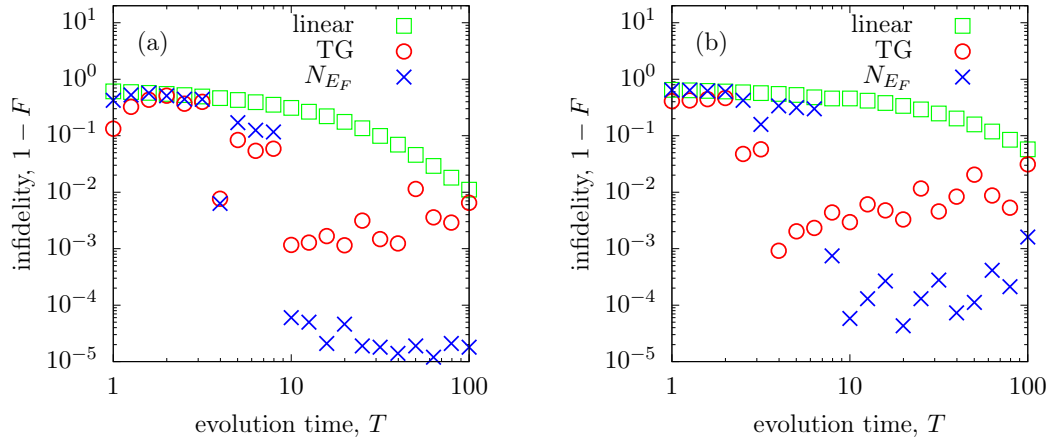

Figure 2.7: Infidelities for the evolution of a TG gas with (a) $N = 3$ and (b) $N = 5$ particles using the CRAB optimal control technique. The optimal pulses were determined for the particle at the Fermi edge (blue crosses), and applied to the whole gas (red circles). Here, the green squares show the 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 one 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 $omega_0$, Equation (2.15) can provide purely imaginary values for $\omega(t)$, corresponding to repulsive potentials. In order to avoid this and because the final states of this protocol 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 [111–113]. 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

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

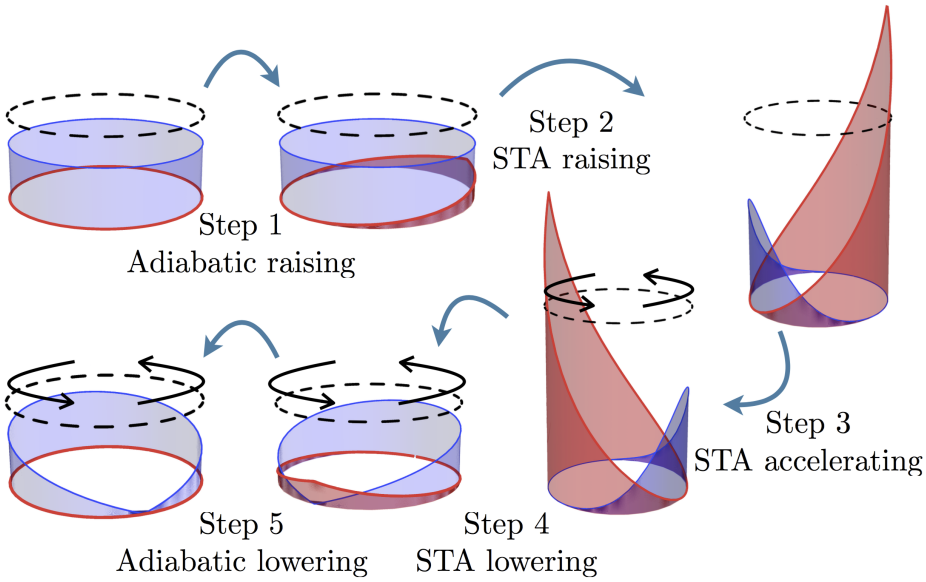

Figure 2.8: Scheme for the acceleration of a single atom using STA. In this example, the ground state of free space gets localized, accelerated and released at the angular velocity of $\Omega = \pi$ into the state $(\exp(i2\pi x) + 1)/\sqrt{2}$. The atomic density is indicated in blue and the potential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

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

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

Conditions are then imposed on x_c, such that

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

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_f is set to zero; however, this case is the opposite.

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

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

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

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

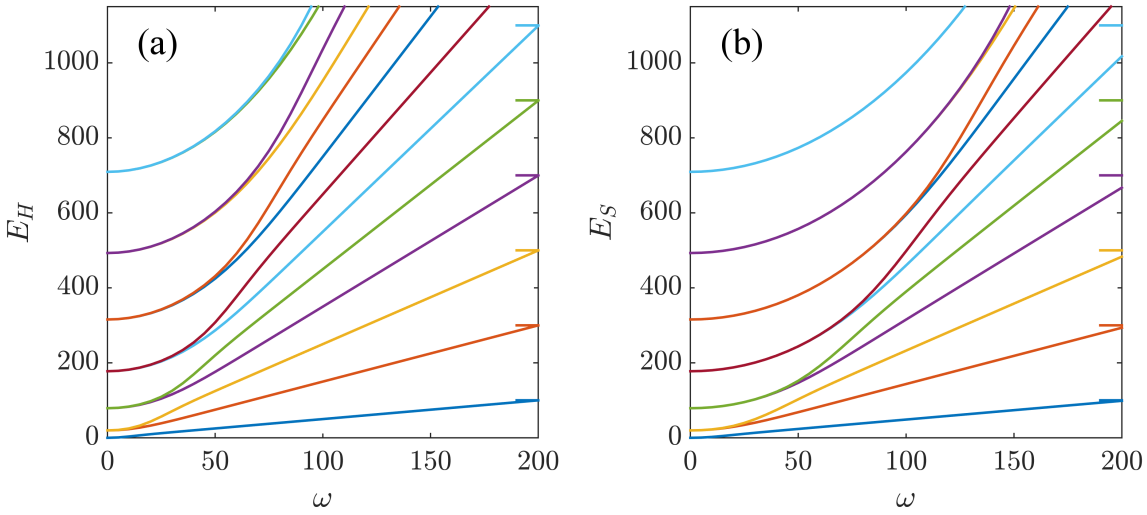

Figure 2.9: Energy eigenspectrum of the system with (a) a harmonic or (b) a sinusoidal potential as a function of ω. The eigenstates continuously change from angular momentum states of energy $E_k = 2\pi^2 k^2$ (with $k = 0, \pm 1, \ldots$) at $\omega = 0$, towards harmonic-oscillator states of energy $E_n = \omega(n + 1/2)$ (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 [114, 115],

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

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 I 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, I 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.

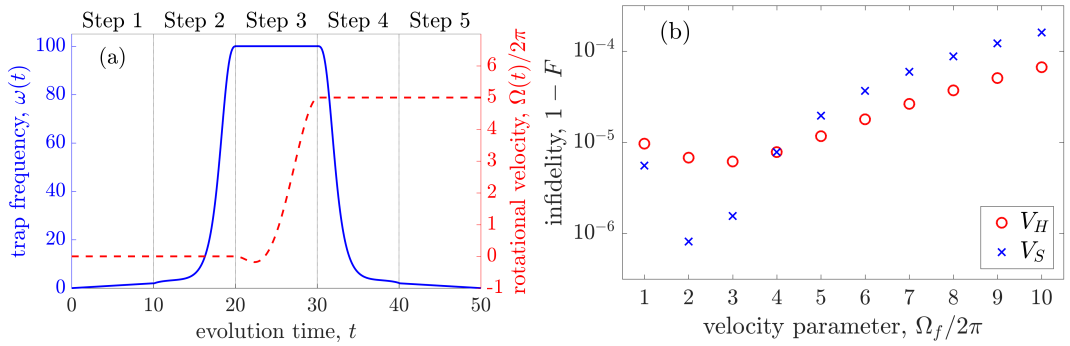

Figure 2.10: (a) Plot of the parameters $\omega(t)$ and the angular velocity $\Omega(t)$ for the entire protocol. The parameters are $\omega_0 = 2$, $\omega_f = 100$, $d = 100$, each step is executed in $t_f - t_0 = 10$, and Ω_f is picked depending on the desired output state (here, $\Omega_f = 5\times 2\pi$). (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 one 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 was demonstrated 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.

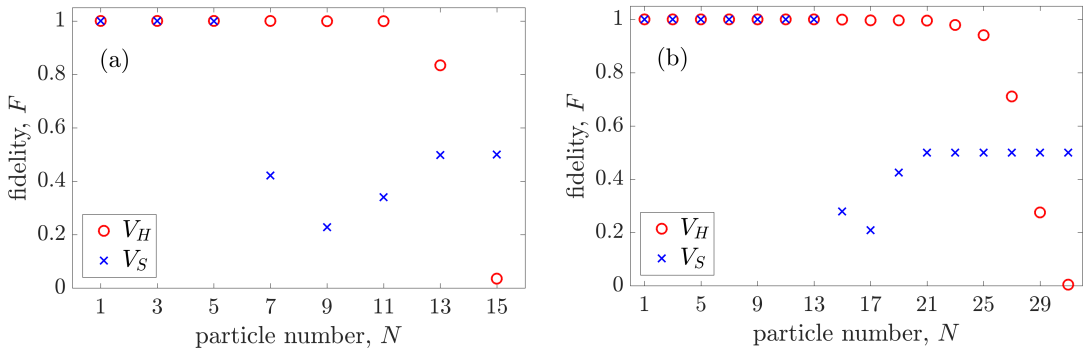

Figure 2.11: Final fidelities F of TG states of increasing particle number for the protocol shown in Figure 2.10(a) with $\Omega_f = \pi$ for V_H (red circle) and V_S (blue cross). Plot (a) shows the fidelity of the protocol with $\omega_f = 100$ and (b) with $\omega_f = 200$.

2.4 Outlook

In this chapter, quantum optimal control and STA protocols were introduced to optimize quantum engineering tasks. I introduced a physical system to generate NOON states in a TG gas non-adiabatically with both methods, and they were shown to be highly effective. Such dynamical evolution techniques require time-dependent control parameters, 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, I 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 connects to the motherboard through a Peripheral Component Interconnect Express (PCIE) slot. As the name implies, the GPU is designed to rapidly manipulate data 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 computing 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 as well with a new technique called General Purpose computing on Graphics Processing Units (GPGPU). Modern HPC design often incorporates the GPU into each computing node, thereby increasing the throughput of the system, overall. The fastest known supercomputer today (Summit, ORNL [1]), is 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, I will 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 [116]. 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 modern environments. 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 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 worker 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 operations on 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, element-wise matrix multiplications are embarrassingly parallel operations; however, FFTs are not [117]. 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.

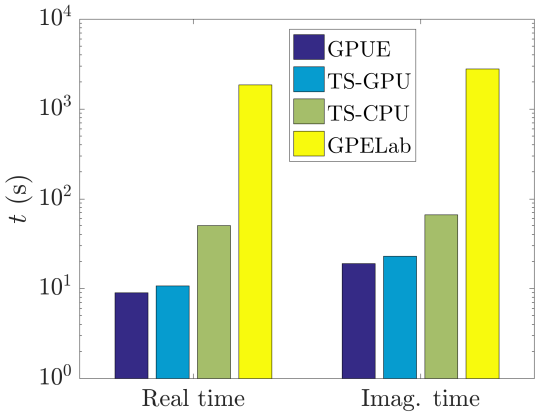

Figure 3.1: Comparison between GPUE (CUDA), Trotter-Suzuki on both GPU (CUDA) and CPU (C++), and GPElab (Matlab). Here, it is shown that GPUE is marginally faster than Trotter-Suzuki, but both GPU implementations are faster than the CPU-based variants. Both software packages are much faster than GPElab [118].

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-like 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 CPU C/C++ code and 100-200 times faster than Matlab code that is simulating the same system [118]. This is shown in Figure 3.1, where a comparison between GPUE, GPElab [119], and Trotter-Suzuki [120] are shown. These benchmarks are consistent with other GPGPU programs [121–123].

As it is possible to massively increase the performance of certain programs by using 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 available memory 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, one often limits the resolution to what can fit onto GPU memory. In addition the data transfer between GPUs and between the GPU and CPU through the PCIE bus is a slow process. Until recently, these limited the size of the 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, it is 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 shortcomings in GPGPU 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, I have chosen to focus exclusively on programming frameworks that expose the hardware for software developers, such as CUDA, OpenCL, and Julia-GPU. 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, I will cover only the GPU memory architecture of NVIDIA GPU accelerators 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, I 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 }
```

I will show 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 loop in C, shown in Listing 3.1. In this case, we take each element with a specified index in **a** and **b** and add them to the appropriate index **c**. In some parallel programming models (OpenACC [124], OpenMP [125], GPUifyLoops.jl, and many others), parallelization of this method is possible by adding a pragma to the start of the loop to specify that the 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 requires 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 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 3.2 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__` function is exactly the same as a CPU function and can only be called by other CPU functions. Finally, a `__device__` function can only be called by other `__device__` functions 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. All kernels are performed asynchronously, ans special care must be taken for iterative tasks.

Another peculiarity appears on line 6, where the addition, itself, occurs. Though there was a necessity for a loop in Listing 3.1, there does not seem to be one at all in Listing 3.2. This is because the GPU is performing this task in parallel and 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*, multiple blocks are organized into *grids*. All threads in the same block have a *shared memory* resource, while all three structures have access to *global memory*. In general, it is important to use shared memory when possible, as it

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.

```
1 __global__ void vecAdd(double *a, double *b, double *c, int n){
2
3     // Global Thread ID
4     int id = blockDim.x * blockIdx.x + threadIdx.x;
5
6     if (id < n){
7         c[id] = a[id] + b[id];
8     }
9 }
```

has a lower latency than global memory, and this will be discussed further in subsequent 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 256, 512, or 1024 is chosen based on register usage, 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, one then need to use the `blockDim.x` and `blockIdx.x` variables to access the appropriate threads for computation. Though threads may be three-dimensional in indexing, data is often 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 and thus one needs to check the bounds of every computation. As such, if one had failed to set the number of threads in the block to the number of elements in the array in Listing 3.2, the kernel would exhibit undefined behavior. For this reason, one needs to take into account potential out-of-bounds computation. If one takes 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, I 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, a distinction between users and developers can be seen. As more complex software

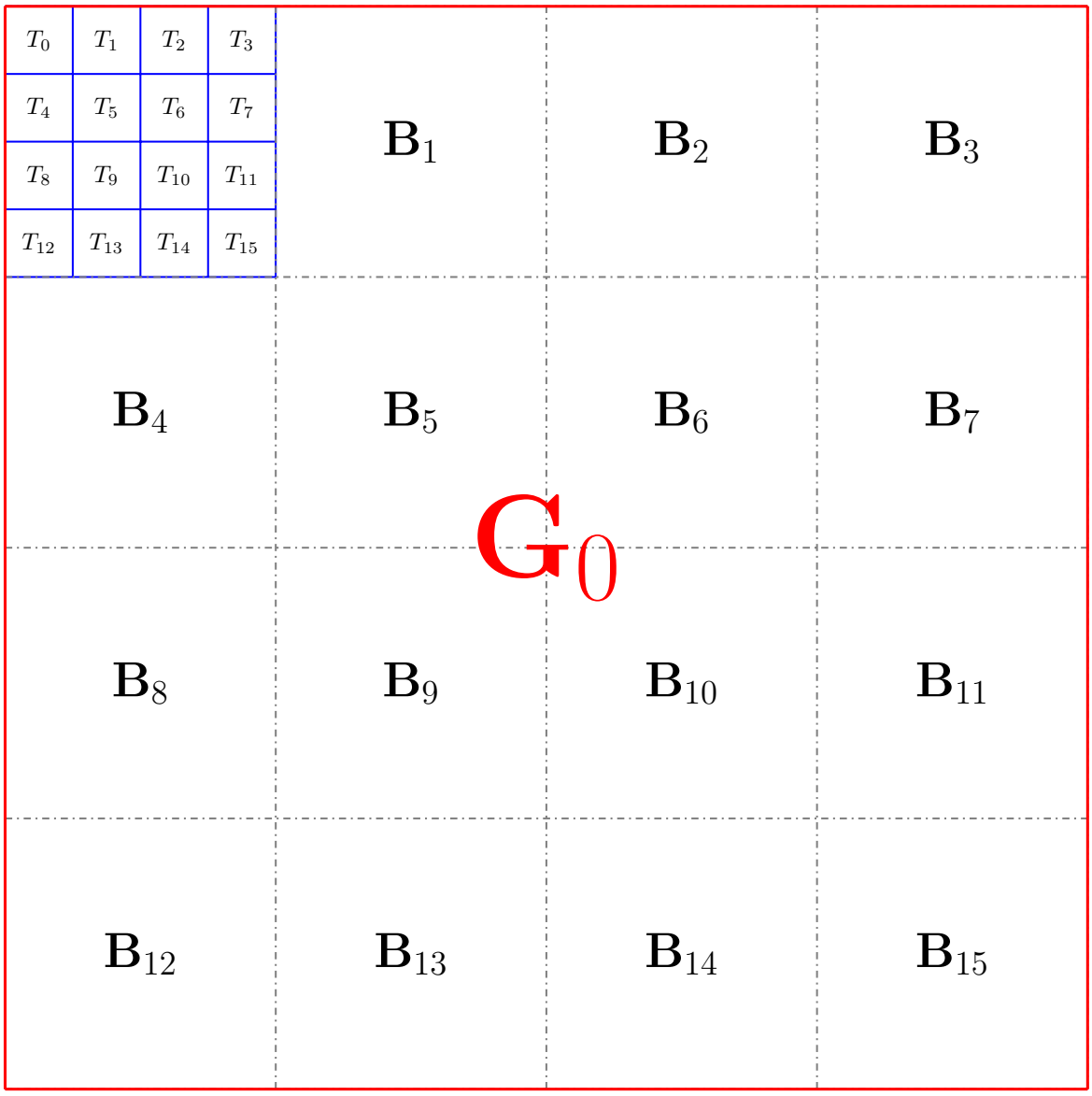

Figure 3.2: Each grid is subdivided into multiple blocks, which is further subdivided into threads for computation. Each thread has a specified ID, which acts as a one-dimensional array, even in a two or three-dimensional system. Here, all areas outlined in red have access to global memory, and any area outlined in blue has access to shared memory. This figure has been slightly modified from [126]

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);
}
```

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 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, all of which are essential to understanding particular design decisions of GPUE. For now, I 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 3.4), 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.

Understanding GPU memory architecture is essential to properly using GPU hardware, and as such, it is worth discussing in-detail. There are three forms of GPU memory that are useful for most applications of GPGPU for scientific computation: global memory, shared memory, local memory, and texture memory. Of these memory types, local memory has the smallest scope and is only available on each individual thread. On the other hand, global memory is shared between all grids, blocks, and threads and is considered to be the slowest memory. 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. Essentially, each time a warp needs to access global memory, it tries to read a word of 1, 2, 4, 8, or 16 bytes, and if the warp is required to access non-contiguous blocks, more accesses will be necessary and thus performance will take a relatively large hit. This is shown in Figure 3.3, where the size of each word is depicted as a grey box on global memory. If each element is 1 byte, a single word is considered to be 4 bytes, and a transfer of 4 elements is required from global memory, a coalesced memory transfer corresponds to 4 consecutive elements in memory, while a strided memory transfer will not work on consecutive elements. In this example, if the stride is 2, an additional access to global memory must be used to transfer the memory, and thus the operation will be half as optimal. 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

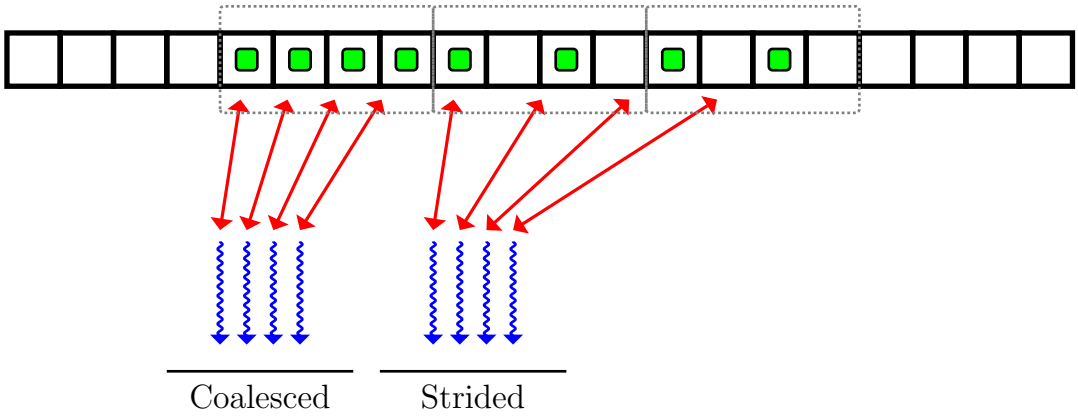

Figure 3.3: An example of a coalesced and strided memory transfer. In this figure, each element is 1 byte and a single word consists of 4 bytes (grey box). If 4 elements are required to be transferred (green boxes), a coalesced accessing patters will access 4 consecutive elements, while a strided one will not. In this case, the strided memory access pattern requires an additional transfer and will thus be less optimal. In this figure, worker threads are seen as blue arrows. This figure was modified from the tikz source from [127].

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 (known as a bank conflict) and the operation can no longer be performed in parallel. It is sometimes necessary to pad variables to prevent bank conflicts from occurring [128].

Of the four 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 tasks with slowly varying operators. In principle, this is the case for the momentum-space operator in the SSFM; however, because texture memory uses single-precision, it will not be used further in 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 or perform these transfers asynchronously. The data transfer between the host and device or between devices must send data through the PCIE 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 [129], 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 greatly increase software development time [130, 131]. 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. The simplest optimizations here involve increasing 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 a trade-off between precision and computational speed. When optimizing the instruction throughput for GPU devices, it is important to discuss 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 considered 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 provide a similar level of computational power. In addition, CUDA support has recently ceased for MacOS 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 with NVIDIA cards. 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 largely competitive in terms of performance and has the benefit of being compatible with both NVIDIA and AMD GPU devices [5, 132]. 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 similar structure to CUDA with slightly more verbose syntax, and thus provides all necessary functionality to develop and maintain scientific software. It should be mentioned that OpenCL can also run on a large variety of other computing

architectures, such as Field-Programmable Gate Arrays (FPGA) and is a more general-purpose computing library than CUDA. 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. Because OpenCL is defined as a general-purpose API with higher access to low-level functionality, it is often more cumbersome for developers than CUDA for GPGPU [133]. As such, it is not as widely used for scientific computing software.

In the end, although OpenCL does provide the ability to more easily construct kernels that can be compiled at runtime, 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 [134], which has provided the unique opportunity for computer scientists to write maintainable and fast code in new languages, like Julia.

Julia-GPU

Julia is a new language to scientific computing, but boasts promising results. It claims to be as usable as Python, but as performant as C [135], which is beneficial for maintainability of HPC software. In addition, Julia's runtime is comparable to CUDA C for GPGPU computation and allows for similar hardware optimizations [4, 136], 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 also 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 outlook of this work, Chapter 5.4. In Chapters 4 and 5, I will also introduce systems that could benefit from a Julia interface.

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 [126]. The GPUE codebase

was published in the Journal of Open Source Software in 2018 [12] and was originally designed by Lee J. O'Riordan with the capability of simulating large-scale Abrikosov lattices in superfluid BECs **??**. My focus with GPUE development has been to simulate three-dimensional systems and optimize the software for dynamic studies on GPU hardware. For this section, I will first describe the FFT optimizations used in GPUE for three-dimensional simulations, 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 undesirable when scaling to multiple GPU devices [117]. 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. On the other hand, if data is transferred to the GPU, it is often 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 `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. This is depicted in Figure 3.4 for a $2 \times 2 \times 2$ grid.

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

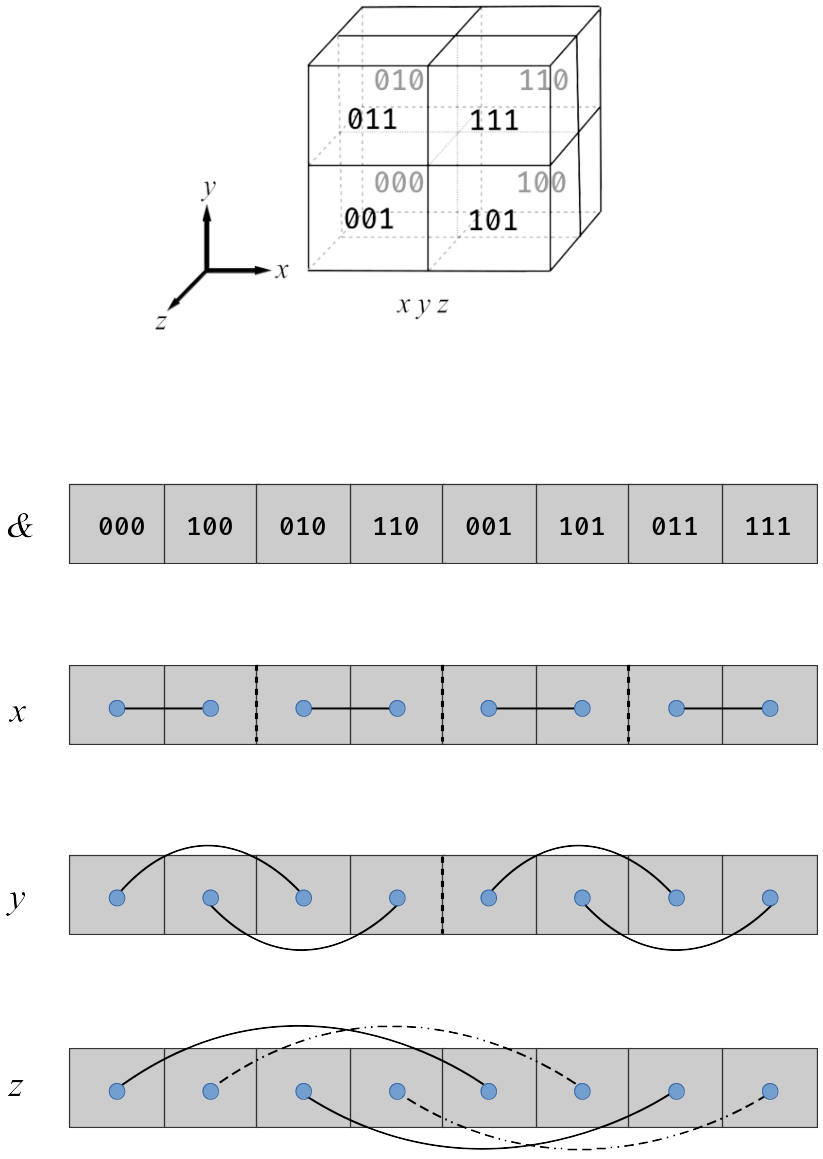

Figure 3.4: An example 2^3 data cube with indices 000→111 with stride and batches shown for x, y, and z FFTs. For the x FFT, the stride is 1, the batch number is 4, and the distance between each batch is 2. For the z FFT, the stride is 4, the batch number is 4, and the distance between each batch is 1. Here, every other line is dashed for visualization. These two transforms can be performed with the `cufftPlanMany` functionality. For the y FFT, the stride should be 2, and the number of batches should be 4; however, no matter what one specifies the distance between each batch to be, it is not possible to perform the operation. This figure can be found in the GPUE documentation [137].

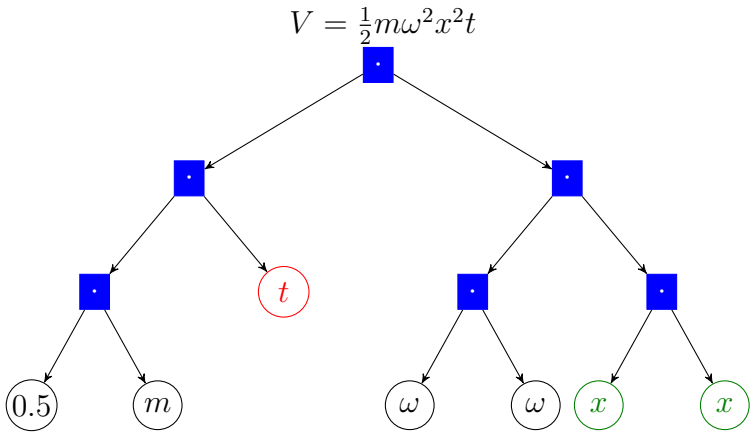

Figure 3.5: Example of expression tree for $V = \frac{1}{2}m\omega^2x^2t$. Blue, filled nodes are operations, leaf nodes are variables, time has been highlighted in red, and spatially-dependent variables are in green. This visualization was modified from a form provided by Xadisten during a Twitch livestream.

not easy to avoid usage of the `cufftPlanMany(...)` operation while still using CUDA. This is why optimizations to GPUE FFT operations will be performed exclusively in GPUE.jl. Next, I will focus on another feature that was inhibited by the CUDA framework, but is nevertheless possible: methods to enable dynamic quantum state engineering with expression trees.

3.3.2 Dynamic field input and output in GPUE with expression trees

As mentioned in Chapter 2, quantum state 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, often for $V(t)$, $g(t)$, or $\mathbf{A}(t)$. Because GPUE is written in C/C++ and 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 compiled into a set of operations to perform on the GPU through expression trees, which are similar to Abstract Syntax Trees (ASTs) in compiler design [138, 139].

An example of an expression tree can be seen in Figure 3.5. 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, certain leaf nodes are designated as either spatially or temporally dynamic. In the case of spatially dynamic variables (`x`, `y`, and `z`), values are taken from constituent vectors based on their `threadIdx.xyz` values, and for any equation that is dependent on `t`, a stored `time` variable is used. This operation necessitates the usage of a dictionary data structure to hold all variables on the host in some fashion, which inhibits host per-

formance; 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 even though 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 naïvely is an iterative process, the longer the expression, the less optimal using this method is. There are ways to use task parallelism when parsing expression trees to allow for greater efficiency; however, because a new expression tree must be parsed for every element in the wavefunction array, adding an additional layer of task parallelism is not straightforward. 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 allows one 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 3.3.7. 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 file input and output. Next, I will briefly discuss efforts to curtail the memory and storage footprint of GPUE simulations.

3.3.3 GPUE memory footprint

For the simulations to be shown in Chapter 4, roughly 50 TB of storage was used for relatively simple two-dimensional, dynamic simulations; however, at the time of that study, there was no compression performed on the output data. It is clear that some level of compression is necessary for three-dimensional, dynamic studies and that this level of compression is likely beyond what can be performed with higher-dimensional, compressed data formats like HDF5. For many vortex studies, it is possible to output only the vortex locations with proper vortex tracking methods, and these methods will be discussed in Section 3.3.4. Though HDF5 is now fully supported by GPUE, this section will focus on other methods that could allow for compressed SSFM simulations.

Initially, the Compressed Split-Step Fourier Method (CSSFM) [19] was considered 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. After CSSFM operation, the wavefunction is then reconstructed to an effectively higher resolution with compressed sensing [140], and in the original work by Bayindir [19], one-dimensional simulations of soliton dynamics were performed. 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, it was found to be unsuitable for two and three-dimensional vortex simulations, be-

cause compressed sensing does not provide adequate compression for simulations of this nature.

In addition to this, an octree-based grid reduction scheme has been considered for two and three-dimensional simulations with the SSFM. This method creates an octree grid based on the Sobel filter of the density with the intent of creating a higher-resolution simulation at locations where the condensate density fluctuates. Further discussion of this system can be found in the Conclusion 5.4.

3.3.4 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 [126] or the GPUE documentation [137]. It is important to describe two dimensional vortex-tracking first before continuing to three-dimensional vortex analysis, which is a much more complicated process.

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 a condensate simulated with GPUE 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 transforms into the 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 for regions outside of the BEC domain, as these regions create spurious 2π phase windings because of the density is roughly zero outside of the condensate region. Further discussion on how to refine this position can be found in previous work [126, 137].

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 superfluid boundaries or reconnect in the form of vortex rings or more complicated vortex structures. Tracking three-dimensional vortices 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* [141], 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. Because some systems simulated with GPUE do not fill the contents of our simulation domain, the proposed method will not work without some modification. This method could still be used if one has some understanding of the trapping geometry to mask out regions beyond the condensate density; however, as we discussed in Chapter 1, this is not always the case with gauge fields. As such, we are currently seeking a more computationally efficient

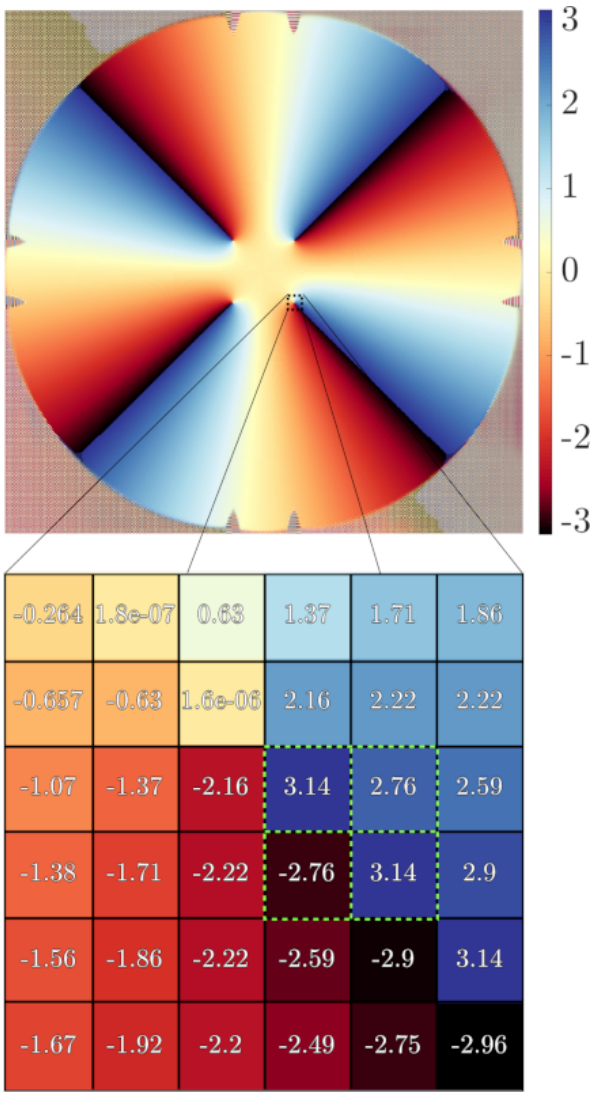

Figure 3.6: An example phase plot of a condensate with four vortices. The inset shows the values of the grid around each vortex location and highlights where the sum is 2π for vortex tracking. This image can be found in the GPUE documentation [137].

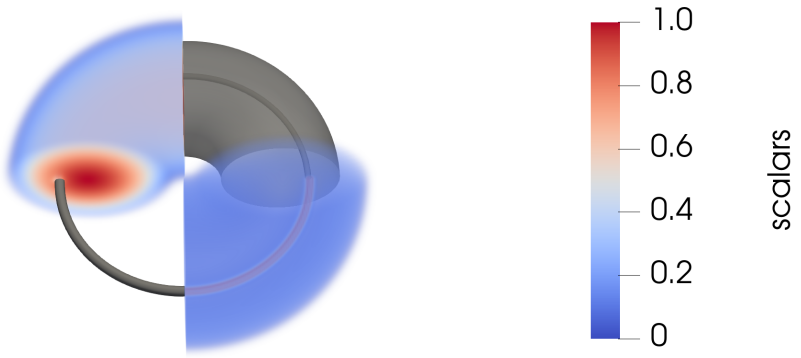

Figure 3.7: An example of vortex highlighting with a Sobel filter. The upper left quadrant is the superfluid density with no modifications and the upper right quadrant is an isosurface of the density with an opacity of 0.6. Note here that if there is no opacity set, it is not possible to see the vortex 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.

method for tracking vortices in three dimensions, and some thoughts on how this could be done can be found in the Conclusion 5.4.

For these reasons, instead of focusing on vortex *tracking*, I 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 [142]. 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 3.7. 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 [143], 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.

3.3.5 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 this option is 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 [58, 59, 126].

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 a robust implementation of parallel reduction has been implemented in GPUE, 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 the simulated wavefunction cannot be influenced by the energy calculation, the operation requires at least one additional allocation of a wavefunction-sized array. Due to the computational time required for each energy calculation, users are requested 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.

3.3.6 Similar software packages

[Discuss other software here, like XMDS]

3.3.7 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 file input and output. Though more work can be done to maximize instruction throughput, this will not significantly improve the performance of the code because it relies 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 [19] should allow for the latter, it was ultimately found unsuitable for general-purpose simulations. As such, the next logical progression is 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 because it lacks an in-built, distributed transpose. For the `cufftPlanMany(...)` functionality, this becomes an even bigger hurdle. To scale to multiple GPU devices efficiently, while still using the SSFM, a large-scale, in-place, multi-GPU transpose is required to ensure proper memory coalescence for FFT routines, which eliminated the need for `cufftPlanMany(...)` in GPUE.

For this reason, development of GPUE.jl has begun, which has similar performance to GPUE, but is currently lacking the expression tree functionality. Once GPUE.jl is at feature parity with GPUE, it will become the primary software package for future development. Ultimately, the Julia language allows for the development of GPUE in a much more maintainable fashion, and also allows for the accessing of GPU hardware in a more convenient way. In addition to this, Julia allows for more modular development of certain features, such as the DistributedTranspose.jl package that should allow for multi-GPU transposes when fully developed.

3.4 DistributedTranspose.jl

At its heart, the two-dimensional transpose is a straightforward operation consisting of a swapping of all row and column elements. Unsurprisingly, this is a difficult task to ensure memory coalescence, but 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 operation is performed on shared memory tiles, and bank conflicts

are avoided by padding the data structure being transposed [128] 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,

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

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 60-70% of the performance based on currently known methods [144, 145]. In addition, distributed transposes of this nature have not been discussed except for on an out-of-place array [146].

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 large-scale 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 and pseudo-spectral simulations in certain regimes over other methods.

3.5 Outlook

In this chapter, I presented the fundamentals of GPGPU, along with the GPUE codebase for simulating superfluid vortex systems. It is important to note that GPU architecture is best at embarrassingly parallel tasks, and as such, the SSFM is severely limited by its FFT routines; however, because one-dimensional FFT operations on GPU devices are often faster for larger matrix sizes than their CPU-based counter-parts [9], it seems that the SSFM is better suited for GPU architecture. In this chapter, I also

discussed important optimizations done in GPUE to ensure proper utilization of GPU architecture for dynamic simulations of dynamic superfluid vortex simulations in two and three dimensions, including expression trees, FFT optimizations, vortex tracking and highlighting, methods used to decrease GPUE's storage footprint, and GPUE.jl. Finally, I discussed the future development of the DistributedTranspose.jl package, which should allow for large-scale spectral methods to be suitable for distributed GPU systems, often found in HPC environments.

For Chapters 4 and 5, I will discuss two physical examples that were enabled by the GPUE codebase, also highlighting future physical directions and re-enforcing the future directions discussed here.

Chapter 4

Chaotic vortex analysis of two-dimensional superfluid systems for few-vortex 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.

Chaotic evolution is typically identified by a significant divergence in trajectory based on a small change in the initial conditions [147], and it is possible to find such an environment in turbulent flow [148, 149]. For classically turbulent flow, the degree of chaos depends on the Reynolds number [150]; however, the nature of quantum chaos for superfluid flow is still an active area of research [76]. Because superfluid vortices have well defined strength and quantized winding numbers, they can be considered less complex when compared to classical vortices where circulation is continuous, there has also been significant interest in the differences between classical and quantum turbulence [151–154]. 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 [155–158].

It is known that it is not possible to excite chaotic behavior in large vortex lattices, as these systems have been proven to be stable to external perturbations [126]. For this reason, it is interesting to probe classical 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 [159–161], who showed that chaos can be excited in systems with as few as four vortices in an infinite plane [159]. Unlike classical chaos, the onset of chaos in quantum systems 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 [154]. 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 [152, 153].

Experimentally, it is now possible to detect vortex circulation [162] and image vor-

tices in-situ [163]. It is also possible to probe vortex dynamics at different times within a single experiment [164, 165]. 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 [75, 166]. Additional effects, such as the Kármán vortex street [167] and Onsager vortex clusters [168, 169] have already been shown to exist experimentally.

Because chaotic events require very small changes in the initial conditions of the system, it is important to create a system with well-controlled initial conditions. For this, I 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. 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.

4.1 Model

The physics simulated in this Chapter is purely two-dimensional, so it is appropriate to begin this section with a disclaimer about the dimensionality of the system I 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, one can assume that the condensate is in the ground state along the $\hat{z}$ dimension so long as there are no energy excitations in the $\hat{z}$ direction that compare with the chemical potential. One can then 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 $\hat{z}$, the interaction strength is modified for a two-dimensional condensate to be

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

For the GPUE codebase, $g_{2\mathrm{D}} = 6.8\times10^{-40}$. With these changes, one can simulate two-dimensional settings with the GPUE codebase [13, 58, 59, 126]. 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 [13].

For this study, a condensate with $N = 10^6$ ^{87}Rb atoms and an s-wave scattering length of $a_s = 4.76\times10^{-9}$m in a pancake geometry with typical trapping frequencies of $(\omega_\perp,\omega_z) = (2\pi, 32\pi)$Hz will be considered. Here, the effective two-dimensional interaction strength is $g = 6.8\times10^{-40}$ m^{4}kg/s^2. These simulations were performed on a grid of $2^{10}\times2^{10}$ points and covering an extent of 700μm $\times$ 700μm

First, ground-state evolution was performed with a low rotation frequency of $\Omega = 0.3\times2\pi$ Hz via GPUE [12]. Although large rotational frequencies will create a triangular lattice, for smaller frequencies, other configurations are known [170], and I will

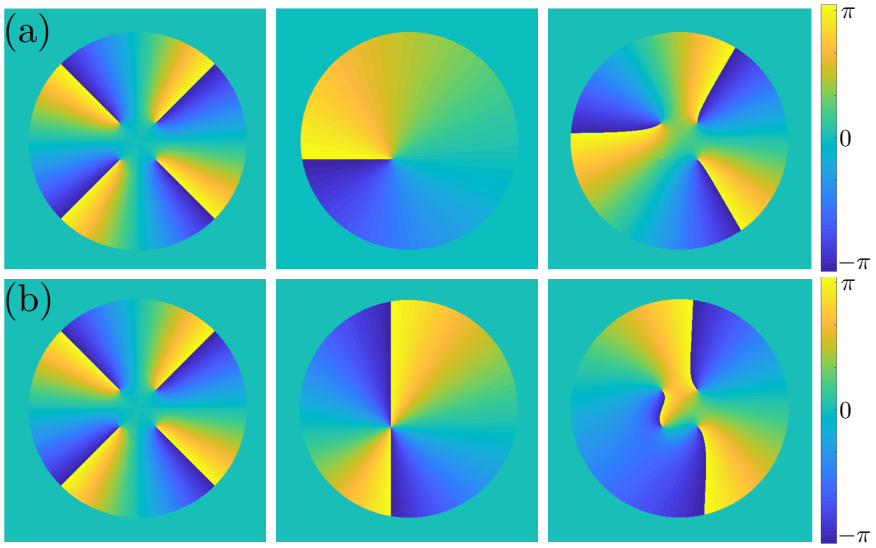

Figure 4.1: Phase distribution of the initial four co-rotating vortex system is shown at the left. In the center is the applied phase mask of -2π for (a) and -4π for (b), and the resulting phase distribution is shown on the right. By applying a -2π phase winding, a vortex is erased from the system, and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

focus on the regime where the ground state is composed of four vortices in a square configuration [171]. Once this configuration is achieved, a vortex is manipulated 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.

4.2 Regular and irregular vortex dynamics

It is known that a lattice of vortices with the same direction of rotation will exhibit regular dynamics [60], and in Figure 4.2(a), we confirm this for this system. In this figure, we show a histogram of the vortex trajectories 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 [163, 164].

Even though the introduction of the anti-vortex creates disordered trajectories, it could be entirely possible that these trajectories are still stable. To uniquely identify chaotic behavior, one needs to show that any small perturbation in the vortex location will also provide a significantly different trajectory. To check this, I compare two sets of

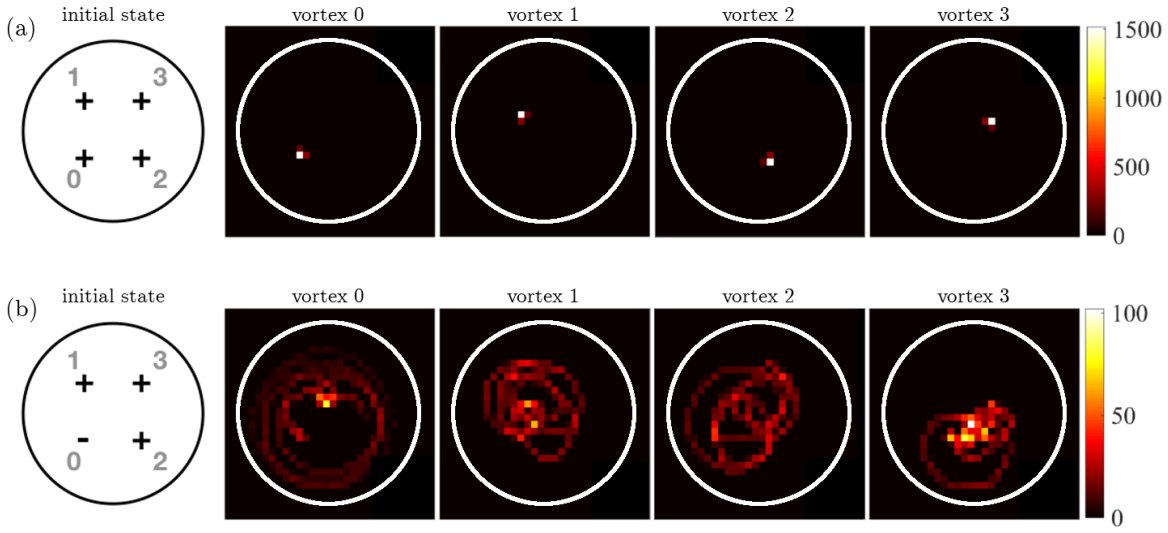

Figure 4.2: Histograms of the positions of each vortex in the transversal plane for 20 seconds in the co-rotating frame. The lower left vortex has been annihilated and re-imprinted with (a) the same and (b) the opposite direction of rotation, exactly on the location of the previous vortex. The area of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

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, I 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, one sees that the difference in trajectories 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 [172]), one sees 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 event in (a) and (b) for the non-shifted and shifted initial 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, there is a clear 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, one must analyze the vortex dynamics in more detail and calculate the Lyapunov exponent.

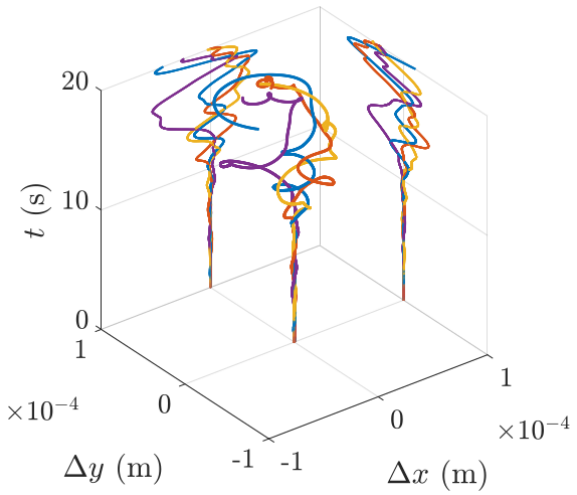

Figure 4.3: Evolution of the difference in trajectories $\triangle \mathbf{r}_i = \mathbf{r}_i - \mathbf{r}'_i$ with $\mathbf{r}$ corresponding to the position of the ith vortex from the center of the condensate and $i \in \{0,\ 1,\ 2,\ 3\}$ labelling each individual vortex for the four-vortex system 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.

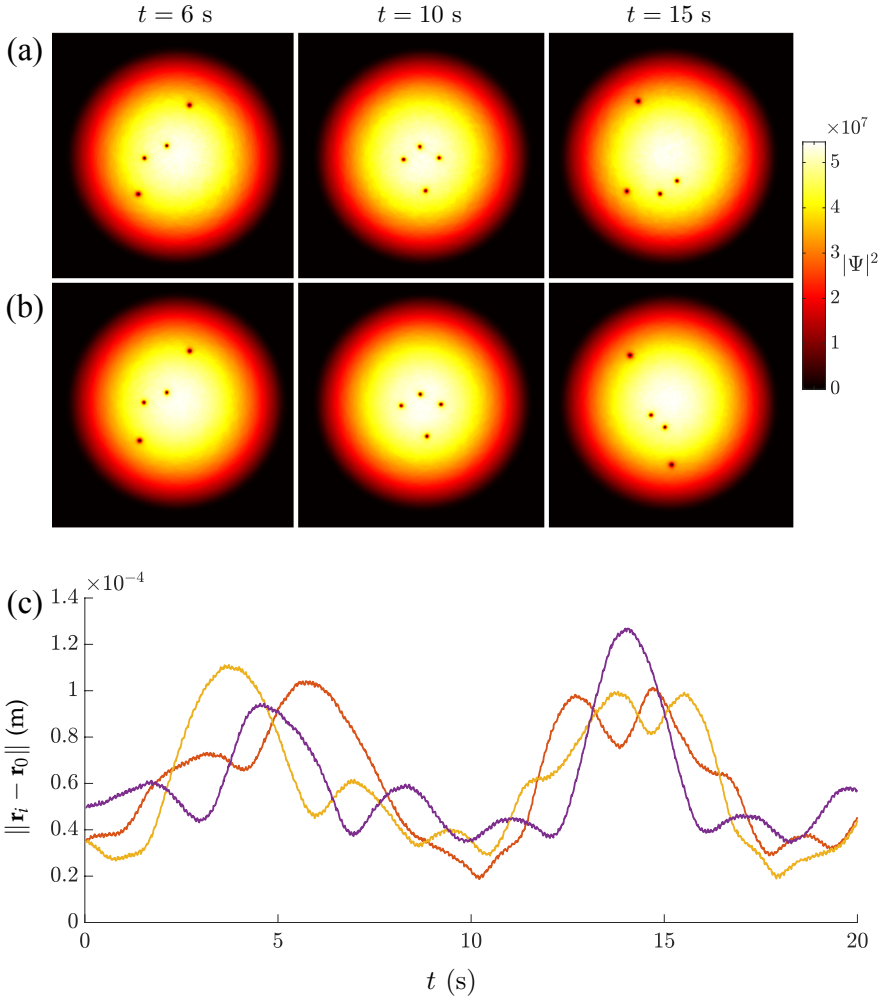

Figure 4.4: Density plots of condensate for (a) $\Delta x = 0$ and (b)$\Delta x = \xi/3$ at times $t = \{6, 10, 15\}$ s. The densities before the scattering event differ only on small scales (see $t = 6s$), whereas for times after the event, large deviations are visible (see $t = 15s$). At $t = 10$ seconds 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.

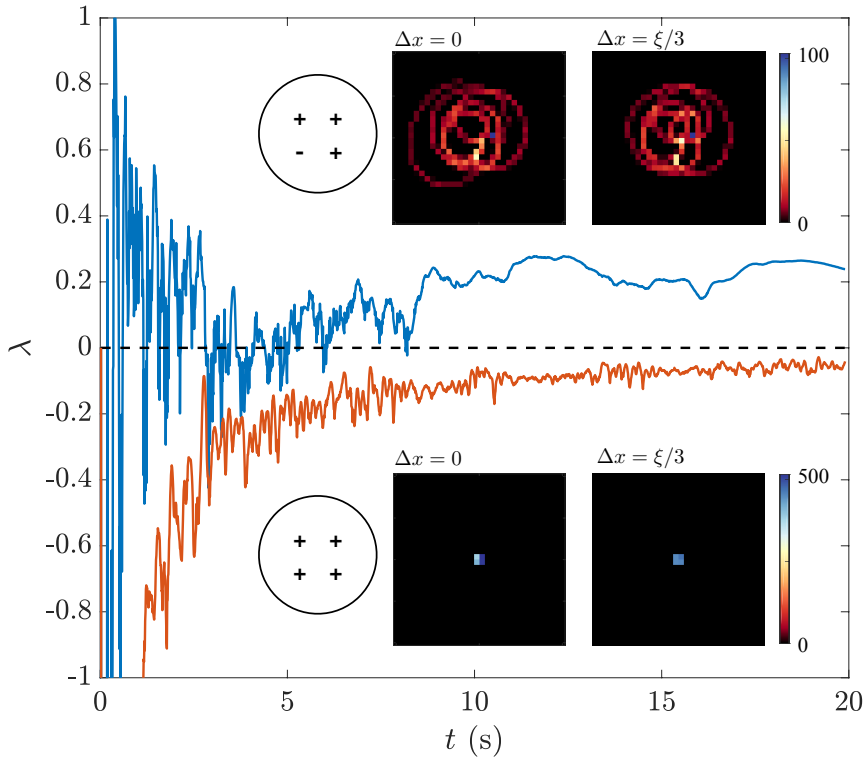

Figure 4.5: The insets show the histograms of the COM trajectories calculated over 20 seconds of evolution for the system of four vortices when the position of a single vortex has been shifted by $\Delta x = 0\xi$ and $\Delta x = \xi/3$. The upper two panels depict the corresponding trajectories after 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.

4.3 Characterizing chaotic vortex dynamics

To characterize the degree of chaos for the shown vortex dynamics, I have chosen to use Lyapunov exponents, which give the rates of divergence for nearby orbits in phase space [173]. With this measure, one can track two trajectories in phase space and assume that the divergence between the two trajectories will either exponentially converge or diverge, which can be modeled 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 this simulation, I 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, each vortex is tracked with the methods outlined in Chapter 3, and use both the position and velocity of the vortex. To determine whether the total system is chaotic, beyond its constituent vortices, I 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, 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 [152], but these were found to provide similar qualitative results to those shown here.

4.4 Outlook

With this study, I 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 to the behavior of large scale vortex lattices, as similar techniques for these systems have been shown to only cause local disturbances [59]. 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 behavior. Because this study very heavily relied on post-processing metrics, such as vortex tracking and the calculation of the Lyapunov exponent, it is apparent 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, I 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 with a dielectric system has been suggested and investigated, and I also provide a method to detect whether a vortex ring is present in an elliptic-toroidal condensate. This project encompasses three-dimensional vortices and coupled light-matter systems, so to begin, I will briefly discuss vortices in three-dimensional systems, 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. As a note, even though vortex dynamics are not shown in this study, GPUE is more than capable of simulating these effects, and potential applications of vortex ring dynamics will be discussed in the outlook of this chapter. The contents of this chapter have recently been submitted to Phys. Rev. Fluids **??**. In this study, I performed all simulations, with exception of vortex states generated in Figures **??** and **??**, which were performed under my supervision by Peter Barnett. I also designed the GPUE codebase to allow for these simulations and visualized all data. This project was supervised overall by Thomas Busch and Rashi Sachdeva.

5.1 Three-dimensional vortex structures

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 [52, 60]. 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 fully three-dimensional 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 [40, 54, 60, 174–178]. There are many interesting features to superfluid vortices in three dimensions, many of which follow from classical fluid dynamic theory [47], which is a well-studied field and covered in many texts [179–182].

By modifying the axis of rotation or inducing a vortex structure with either artificial magnetic fields or phase imprinting, one 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 [183], which means that unstable excitations may decay into vortex rings or objects with ring-like topology [175].

In the case of multiple, interacting vortex rings, one 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* [184, 185]. This behavior can be extended to vortex ring bundles, in such systems the entire bundle will turn in on itself while moving in its self-induced velocity field [174].

In addition to leapfrogging, vortex rings can interact through direct collisions [186]. In superfluid ^{4}He, some of the earliest experiments on vortex collisions with vortex rings were performed by Schwarz in 1968 [187]. 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 [188]. 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, I 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. [57]. These reconnections produce sound waves when vortices directly interact and Kelvin waves when vortices indirectly interact [189]. 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 [190]. This means that one would expect to see vortex reconnections in any sufficiently complicated vortex tangle [56].

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 [175] with the snake instability [191], the collision of symmetric defects [192], or direct density engineering [193, 194]. There are other theoretical proposals that consider interfering two BEC systems [190], using Feschbach resonances [195], or phase imprinting methods [191]. 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 [196]. 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 [175, 178]. There is also a large amount of interest in generating more complicated vortex structures, such as vortex knots [77, 197, 198]. Artificial magnetic fields seem to be a promising method for the generation of complex three-dimensional vortex structures in BEC systems [199], and in this chapter, I 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 [200]. In practice, this means that one 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 [201]. 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 [202, 203]. 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 [204]. The form of the evanescent field varies significantly depending on the optical modes propagating through the nanofiber, and I 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 [205–210], and trapping potentials around 200nm from the fiber surface can be created with two differently detuned input fields [114, 211]. 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 [212]. 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 [213–215]. 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

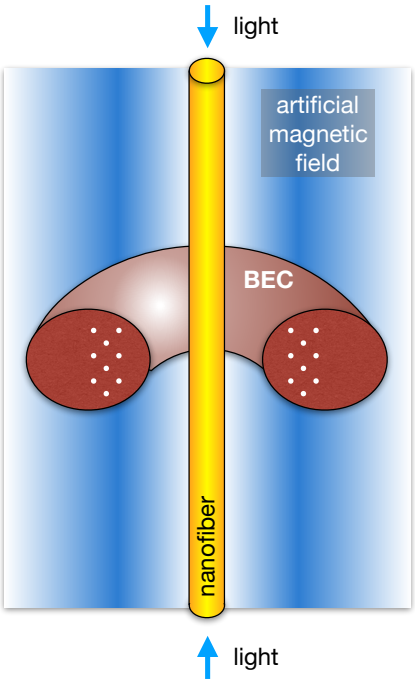

Figure 5.1: Schematic of the system. Blue or red-detuned light is sent into the nanofiber (yellow), creating an evanescent field and artificial magnetic field (blue) that influences the BEC (maroon) held by a toroidal trapping potential. If the artificial magnetic field strength is greater than a threshold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

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 [216–218].

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,

$$i\hbar\frac{\partial\Psi}{\partial t} = \left[\frac{(p - m\mathbf{A}(\mathbf{r}))^2}{2m} + V_{\text{trap}}(\mathbf{r}) + g|\Psi|^2\right]\Psi. \tag{5.1}$$

Here, all values are defined as before. The artificial vector potential can take many forms, but for here, I will again choose a description based on Berry's connection [200],

$$\mathbf{A} = i\hbar\left\langle\Psi_l|\nabla\Psi_l\right\rangle, \tag{5.2}$$

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, these states can be written within the rotating wave approximation as [201],

$$|\Psi_1(\mathbf{r})\rangle = \begin{pmatrix} \cos[\Phi(\mathbf{r})/2] \\ \sin[\Phi(\mathbf{r})/2]e^{i\phi(z)} \end{pmatrix}, \tag{5.3}$$

$$|\Psi_2(\mathbf{r})\rangle = \begin{pmatrix} -\sin[\Phi(\mathbf{r})/2]e^{-i\phi(z)} \\ \cos[\Phi(\mathbf{r})/2] \end{pmatrix}, \tag{5.4}$$

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 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.

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 $\text{HE}_{21\text{even}}$, $\text{HE}_{21\text{odd}}$, 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 $\text{HE}_{\ell m}$ mode with circular polarization is [219],

$$E_r = iC[(1-s)K_{\ell-1}(qr) + (1+s)K_{\ell+1}(qr)]e^{i(\omega t - \beta z)}, \tag{5.5}$$

$$E_\phi = -C[(1-s)K_{\ell-1}(qr) - (1+s)K_{\ell+1}(qr)]e^{i(\omega t - \beta z)}, \tag{5.6}$$

$$E_z = 2C(q/\beta)K_\ell(qr)e^{i(\omega t - \beta z)}, \tag{5.7}$$

where

$$s = \frac{1/h^2a^2 + 1/q^2a^2}{J'_\ell(ha)/[haJ_\ell(ha)] + K'_\ell(qa)/[qaK_\ell(qa)]}, \tag{5.8}$$

$$C = \frac{\beta}{2q}\frac{J_\ell(ha)/K_\ell(qa)}{\sqrt{2\pi a^2(n_1^2N_1 + n_2^2N_2)}}, \tag{5.9}$$

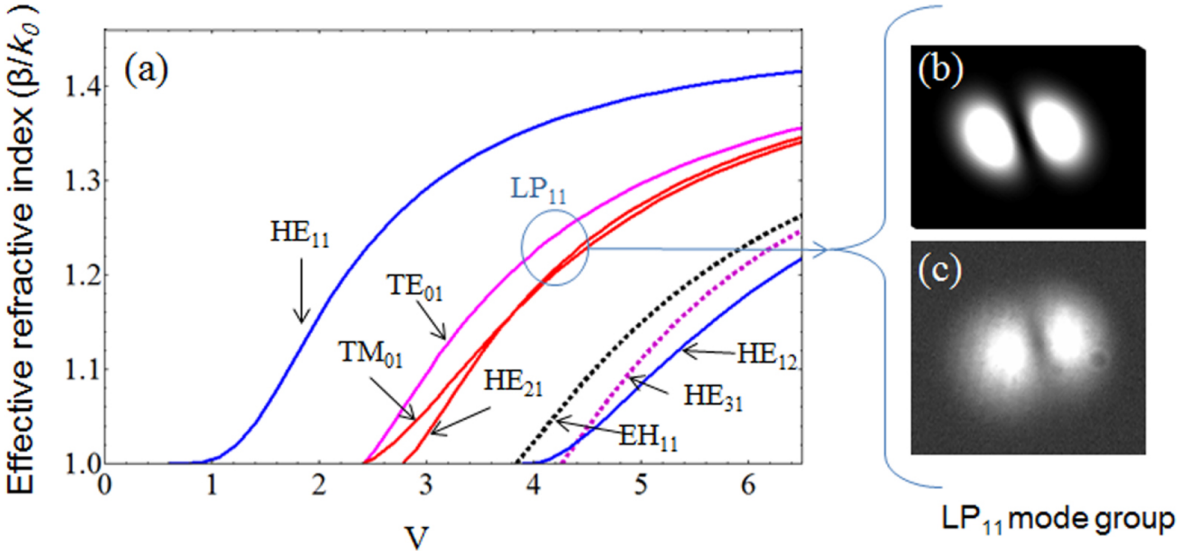

Figure 5.2: (a) Plot of effective refractive index and V number of an optical fiber. The circle indicates the LP_{11} group, which is the first higher-order group composed of the HE_{21even}, HE_{21odd}, TE_{01} and the TM_{01} modes. (b) Simulated and (c) experimental images of the output from the LP_{11} group are also shown. Reproduced from [207, 210].

and

$$\begin{aligned} N_1 =& \frac{\beta^2}{4h^2}\Big[(1-s)^2\left[J_{\ell-1}^2(ha)+J_\ell^2(ha)\right] \\ &+(1+s)^2\left[J_{\ell+1}^2(ha)-J_\ell(ha)J_{\ell+2}(ha)\right]\Big] \\ &+\frac{1}{2}\left[J_\ell^2(ha)-J_{\ell-1}(ha)J_{\ell+1}(ha)\right], \end{aligned} \tag{5.10}$$

$$\begin{aligned} N_2 =& \frac{J_\ell^2(ha)}{2K_\ell^2(qa)}\Bigg(\frac{\beta^2}{4q^2}\Big[(1-s)^2\left[K_{\ell-1}^2(qa)-K_\ell^2(qa)\right] \\ &-(1+s)^2\left[K_{\ell+1}^2(qa)-K_\ell(qa)K_{\ell+2}(qa)\right]\Big] \\ &\left[K_\ell^2(qa)+K_{\ell-1}(qa)K_{\ell+1}(qa)\right]\Bigg). \end{aligned} \tag{5.11}$$

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

$$E_x = \sqrt{2}C\Big[(1-s)K_{\ell-1}(qr)\cos(\phi_0)+(1+s)K_{\ell+1}(qr)\cos(2\phi-\phi_0)\Big]e^{i(\omega t-\beta z)}, \tag{5.12}$$

$$E_y = \sqrt{2}C\Big[(1-s)K_{\ell-1}(qr)\sin(\phi_0)+(1+s)K_{\ell+1}(qr)\sin(2\phi-\phi_0)\Big]e^{i(\omega t-\beta z)}, \tag{5.13}$$

$$E_z = 2\sqrt{2}iC(q/\beta)K_\ell(qr)\cos(\phi-\phi_0)e^{i(\omega t-\beta z)}. \tag{5.14}$$

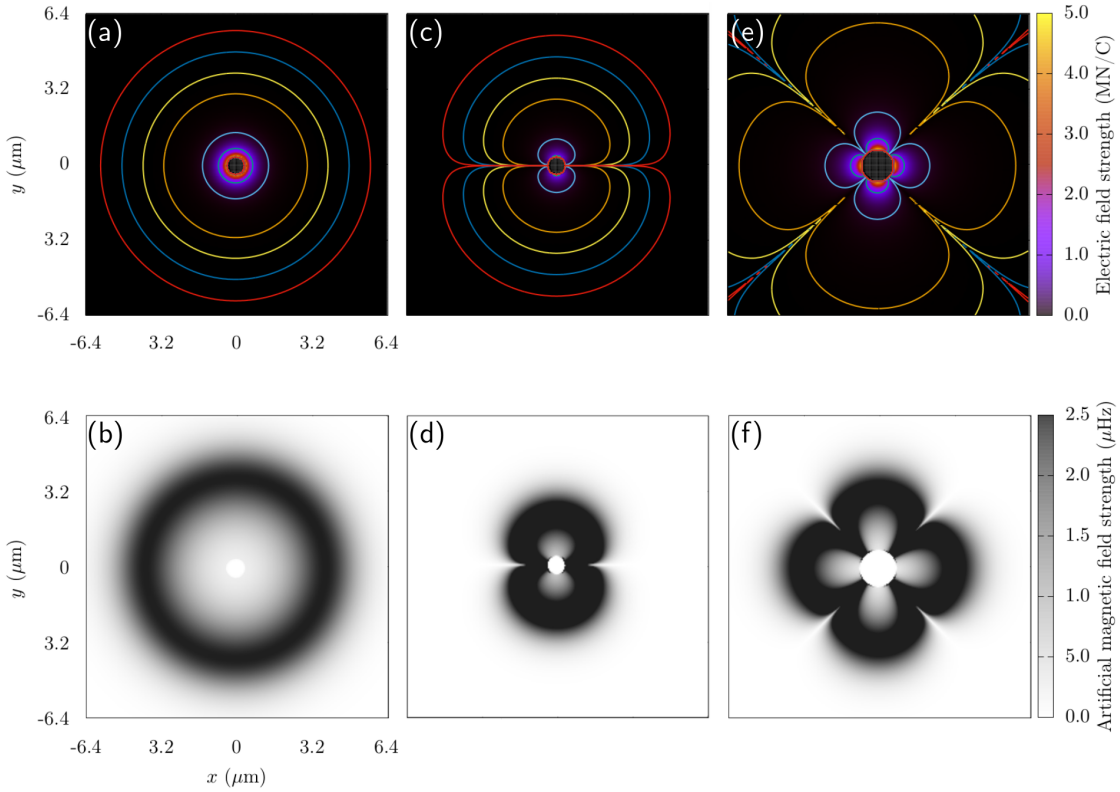

Figure 5.3: Images of electric and artificial magnetic field profiles for [(a) and (b)] the fundamental HE_{11} mode with circular polarization, [(c) and (d)] the HE_{11} mode with linear polarization, and [(e) and (f)] the HE_{21} mode with linear polarization. For these calculations, the input power 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

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 [212]

$$\mathbf{A} = \hat{z}\hbar\kappa_0(n_1+1)\tilde{s}\left[\frac{|d_rE_r + d_\phi E_\phi + d_zE_z|^2}{1+\tilde{s}^2|d_rE_r + d_\phi E_\phi + d_zE_z|^2}\right], \tag{5.15}$$

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

$$\begin{aligned}\mathbf{B} =& \frac{\hbar\kappa_0 s^2(n_1+1)}{(1+\tilde{s}^2|d_rE_r + d_\phi E_\phi + d_zE_z|^2)^2} \\ &\times\left[\hat{\phi}\frac{\partial}{\partial r}|d_rE_r + d_\phi E_\phi + d_zE_z|^2\right. \\ &\left. - \hat{r}\frac{1}{r}\frac{\partial}{\partial\phi}|d_rE_r + d_\phi E_\phi + d_zE_z|^2\right].\end{aligned} \tag{5.16}$$

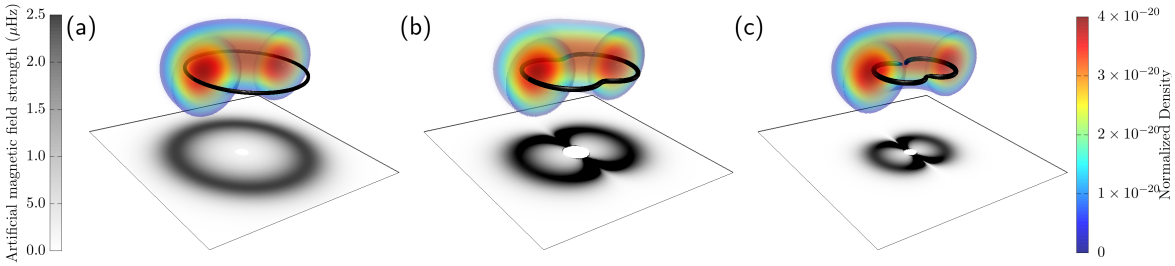

Figure 5.4: Vortex configurations for different magnetic field profiles from the nanofiber for the fundamental HE_{11} mode with (a) circular polarization, (b) elliptical polarization, and (c) linear polarization along the $\hat{y}$ direction. The vortex distributions have been found via an isosurface 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).

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 [212].

For the purposes of this project, I 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 configurations and their corresponding magnetic field profiles can be seen in Figure 5.3. Here, the circularly polarized HE_{11} mode will create a cyllindrically 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, 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 [12] 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, with a toroidal trapping potential around the fiber given

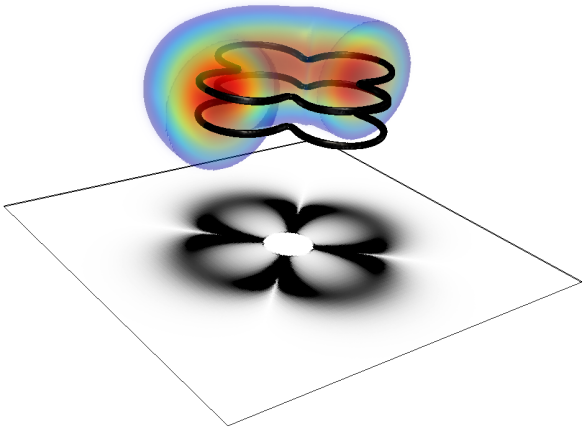

Figure 5.5: Vortex configuration for the HE_{21} mode with linear polarization along the $\hat{y}$ direction. The magnetic field profile is similar to the one shown in Figure 5.3(f), and has been calculated for a nanofiber of 400 nm in radius with red-detuned light of 980 nm.

by,

$$V_{\text{trap}} = m(\omega_r^2(r-\eta)^2 + \omega_z^2 z^2), \tag{5.17}$$

where the frequencies in the $\hat{r}$ and $\hat{z}$ directions are chosen to be $\omega_r = \omega_z = 7071\text{Hz}$ to match typical experimental conditions in fiber trapping [205]. 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, the fiber radius is increased 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, one can 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 when approaching each other within a healing length. If elliptically polarized light is considered, one can see a hybrid ground state between the circularly and linearly polarized modes, shown in Figure 5.4(b). Finally, I show a four-petal ground-state solution when using the HE_{21} linearly polarized mode in Figure 5.5. Here, I 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, I 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, the vortex rings begin to pack together and

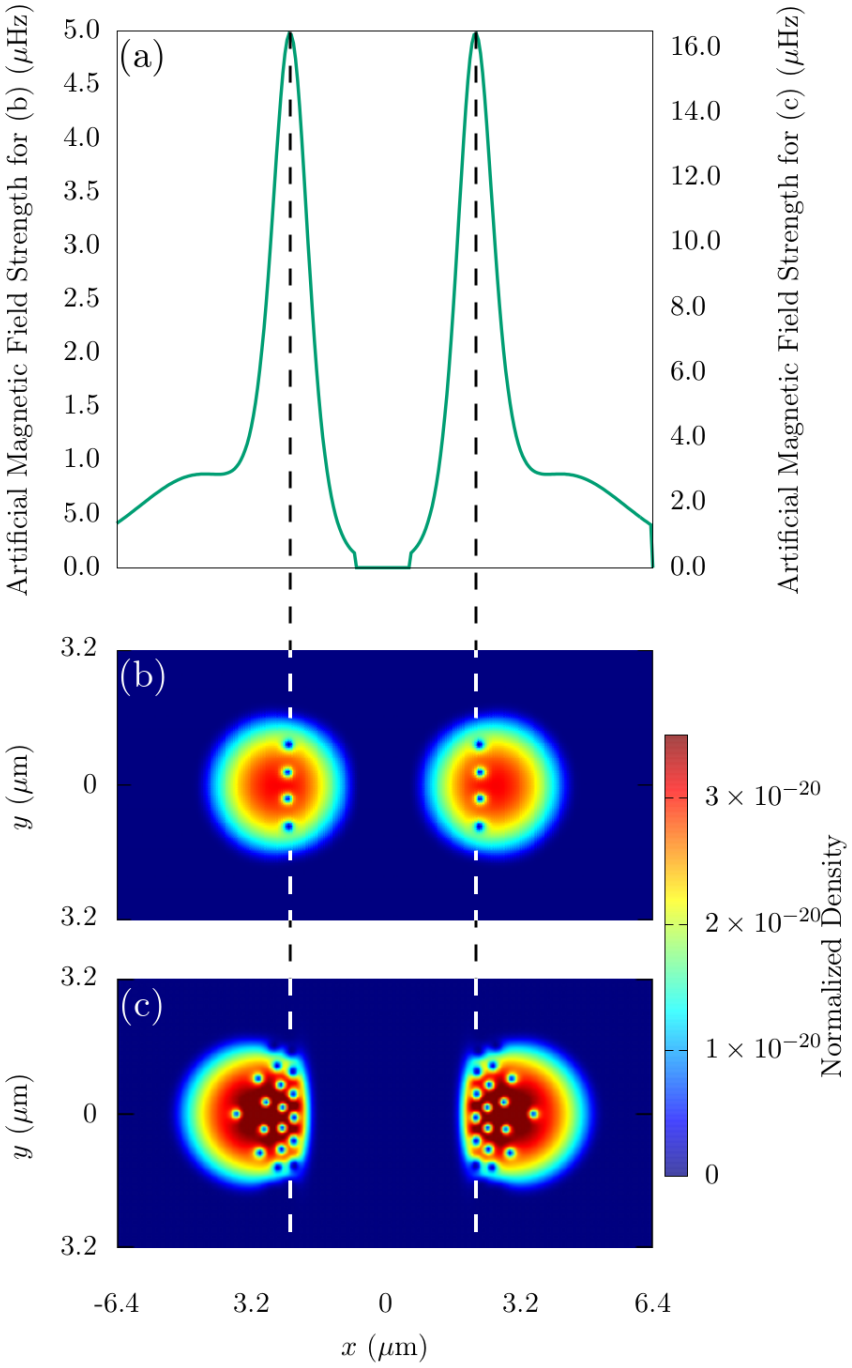

Figure 5.6: (a) The magnetic field profile along the x-direction for the fundamental HE_{11} mode with circular polarization outside a fiber of 200 nm radius. Note that for this mode and polarization the whole system is azimuthally symmetric. For weak fields (see (b)) this leads to a small number of 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.

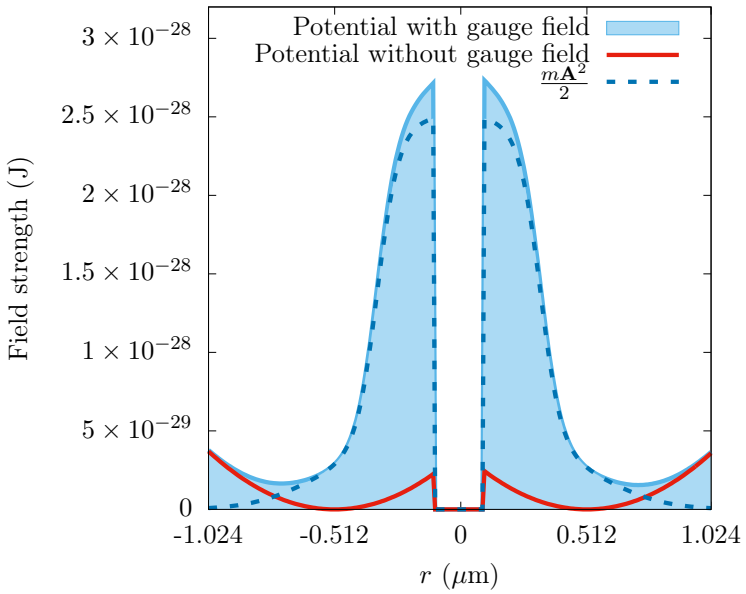

Figure 5.7: Visualization of the change in potential due to the effects of the artificial magnetic field for Figure 5.6. Here, the experienced potential is shown in the blue shaded region, the scaled artificial vector potential is shown as a dotted, dark blue line, and the original trap is shown in red.

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, I 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, 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, one can identify whether vortex rings are present by exciting the scissors mode of the conden-

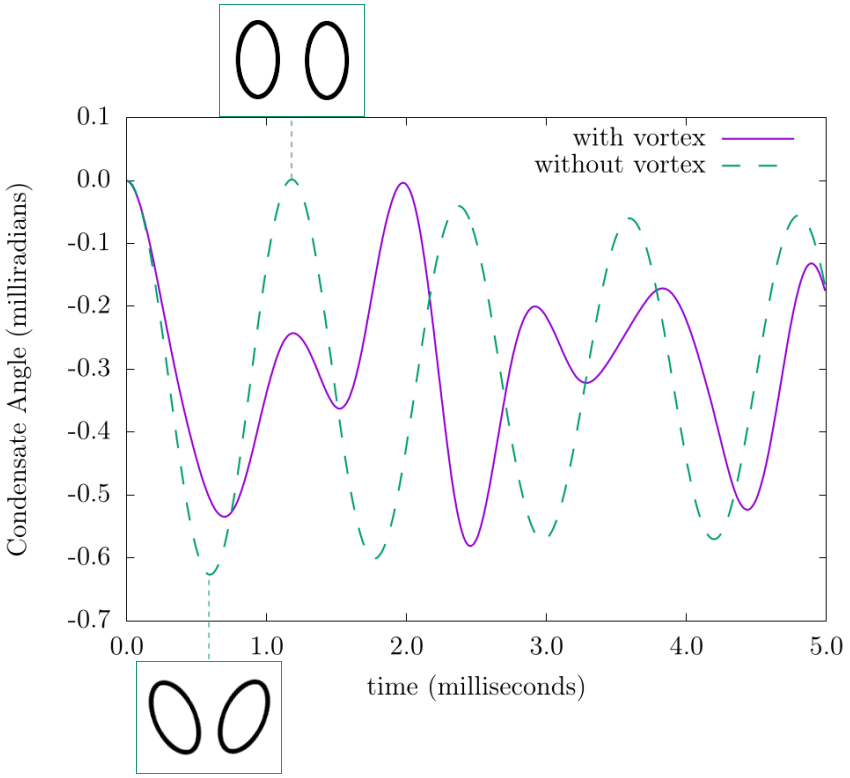

Figure 5.8: Angle of the condensate axis after excitation of the toroidal scissors mode for an elliptic-toroidal BEC with a single vortex ring (purple, solid) and without a vortex present (cyan, dashed). Depictions of two dimensional slices for the scissors mode without a vortex are shown in the insets. Here, 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.

sate [213–215]. 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,

$$V = V_{\text{trap}}(r, \theta, z) - m\omega_0^2 \alpha r z, \tag{5.18}$$

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 [218]

$$\omega_{\text{scissors}} = \sqrt{\omega_r^2 + \omega_z^2}. \tag{5.19}$$

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 [216–218]. When the splitting frequency is small compared to the scissors mode frequency, it can be written as [217]

$$\omega_+ - \omega_- = \frac{\langle l_z \rangle}{m\langle r^2 + z^2 \rangle}, \tag{5.20}$$

where $\langle l_z \rangle$ is the average angular momentum per particle. Calculating these values for the 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 (5.20).

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, I have shown that it is possible to create and control vortex rings and ring-like 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 superfluid mechanisms, like the kelvin-mode cascade, superfluid turbulence, or reconnection events between superfluid vortex

lines. This 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.

To generate vortex knots with this system, a magnetic field with $\hat{z}$ must be created. Such fields might be possible with fiber Bragg gratings [220] or other methods to change the evanescent profile of the fiber. Such simulations would require FEM or FDTD simulations of the optical fiber, itself, which makes the problem an intricate engineering process of generating the right field with an optical nanofiber. In addition, a z-dependent gauge field requires analysis of the other gauge field term, $\frac{p_z \mathbf{A}_z}{2}$, which is not considered in this work because the derivative along the $\hat{z}$ direction for the fiber field is 0. This leads to interesting questions about how BEC systems behave in the presence of such artificial magnetic fields.

For the movement of dynamic vortex ring tangles generated with this system, vortex tracking methods in three-dimensions should be employed. As described in Chapter 3, this is not a trivial task and I will discuss methods that could be used to do this in the Conclusion 5.4. It is also interesting to see what happens if the HE_{21} vortex structures are evolved in real-time and whether scissors modes can be used to detect such vortex structures as well. With a sharp magnetic field, it is also possible to create a phase separation along a vortex line for multicomponent condensate simulations.

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 coalescence 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, I attempted to highlight all future directions when relevant to the chapter. I will continue the discussion on future research pursuits here, while also presenting new directions not yet considered in this work.

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 on new 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 two and three 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 and thus cannot mask out the phase outside the condensate surface. Though it is possible to create mask out areas without a vortex by multiplying the phase domain by the condensate density, this has not yet been implemented in GPUE. As such, a generalized vortex tracking methods for two and three-dimensional simulations is desirable.

In 2016, a method for three dimensional vortex tracking was proposed by Villois *et. al.* [141]; however, this method has no computational complexity bound, assumes periodic boundary conditions, and required a large amount of communication between the device and host. This method begins to search for vortex locations by creating a Boolean domain of locations to search through by locating where the density is below a threshold. After the Boolean matrix is defined, the search is refined by locating phase plaquettes within this domain and iteratively locating a vortex skeleton. For the case where the condensate does not extend to the edges of the simulated domain, this method needs further refinement, and I have considered developing a similar method that leverages the vortex highlighting scheme to create three-dimensional vortex skeletons for vortex tracking in two and three dimensions. In this case, rather than simply searching for locations where the density is low, we could search for locations between peaks in the Sobel-filtered density, which correspond to likely vortex locations. This method could be easily re-worked for two-dimensional simulations as well, creating a dynamic mask every timestep for vortex tracking, thereby eliminated the current, unstable masking procedure.

5.5.2 General purpose Hamiltonian solver

As GPUE can perform multi-component simulations and can also parse expression trees, it is possible to simulate a wide variety of physical systems beyond the GPE. One such extension involves using the expression tree parser to solve arbitrarily provided Hamiltonians. This project would require significant software engineering time apart from GPUE-specific development, as the SSFM is not always the most optimal choice for solving certain quantum systems. In addition, expression trees can be much more easily created and used in other software frameworks, such as Julia.

5.5.3 Octree grid

Though the CSSFM method [19] does not provide adequate compression for vortex-based simulations, there is still an interesting open question about whether any grid compression schemes might work with the SSFM. One such method that has been

suggested is an octree-based grid, where each grid element is determined in a finite-volume fashion based on the Sobel filter of the density. This would allow for higher resolution in areas that have the greatest change in condensate density, which would be around the condensate edges, vortices, and perturbations, such as sound waves. One advantage of using an octree for this purpose is that FFTs may be performed with small modifications on the grid, thereby allowing for the SSFM with dynamic grids; however, the regridding operation is still computationally intensive and even though it is entirely possible to perform an FFT on this system, such functionality is beyond the scope of CuFFT. Therefore, significant engineering time must be devoted to either developing a CuFFT competitor for this case or for finding some method to allow CuFFT to be used in this way. Because the primary advantage of SSFM over other methods is the speed of FFT-based operations, there is likely little reason to use this method over FEM solutions.

GPUE.jl

In Chapter 3, I discussed the advantages of re-writing GPUE in Julia, and in Chapters 4 and 5, I discussed applications of GPUE and suggested areas where Julia could improve the software. In these chapters, I mentioned that post-processing operations, such as vortex tracking, calculating of the Lyapunov exponent spectrum, and calculation of the scissors mode oscillation angle, require processing output data from GPUE, and though vortex tracking and highlighting is already in-built to GPUE, it is still computationally complex operation. For this reason, it would be useful for both the timestepping method and all necessary post-processing tools to be available in the same language, and in Julia, this is possible with littler performance loss. This would also allow for a modular development and maintenance of GPUE in the future, where the timestepping methods remain unchanged as users develop post-processing methods when required.

In addition to these, there are many computer science researchers who are using Julia as a testing-bed for new ideas for software engineering and certain operations, such as GPU-enabled automatic differentiation [221] could be a useful tool for future quantum simulations like those presented in this work. At the current data, the GPUE.jl package competes with GPUE (CUDA) in GPU performance; however, expression trees have not been completely implemented. Even so, as the development of the DistributedTranspose.jl package is in Julia, future development of GPUE.jl should quickly surpass the functionality of its CUDA-based variant.

5.6 Future simulations of quantum systems

In addition to further developments of GPUE and related software packages, 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 control processes through expression trees, further three-dimensional STA studies can also be performed.

Ultimately, this work has provided a toolbox for the simulation of various quan-

tum phenomenon that were computational intractable before now, including the three-dimensional simulations of superfluid turbulence without relying on vortex-filament methods, 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.

[Add other physics futures]

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 3

```
#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. doi: 10.1109/ISSCC.2019.8662426.

[2] R. Reyes, I. López-Rodríguez, J. J. Fumero, and F. De Sande. accull: an OpenACC implementation with CUDA and OpenCL support. In *European Conference on Parallel Processing*, pages 871–882. Springer, 2012.

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

[4] T. Besard, P. Verstraete, and B. De Sutter. High-level gpu programming in julia. *arXiv preprint arXiv:1604.03410*, 2016.

[5] A. Munshi, B. Gaster, T. G. Mattson, and D. Ginsburg. *OpenCL programming guide*. Pearson Education, 2011.

[6] J. Cheng, M. Grossman, and T. McKercher. *Professional CUDA C Programming*. John Wiley & Sons, 2014.

[7] M. Frigo and S. 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] D. T. Popovici, T. M. Low, and F. Franchetti. Large bandwidth-efficient FFTs on multicore and multi-socket systems. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 379–388. IEEE, 2018.

[9] H. Merz. CUFFT 1.1/2.0 vs FFTW 3.1. 2 (x86_64) vs FFTW 3.2 (cell) comparison, 2016.

[10] M. Brehler, M. Schirwon, D. Göddeke, and P. 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.

[11] J. Schloss, A. Benseny, J. Gillet, J. Swain, and Th. Busch. Non-adiabatic generation of NOON states in a Tonks–Girardeau gas. *New Journal of Physics*, 18 (3):035012, 2016.

[12] J. R Schloss and L. J. Riordan. GPUE: Graphics processing unit Gross-Pitaevskii equation solver. *J. Open Source Software*, 3(32):1037, 2018.

[13] T. Zhang, J. Schloss, A. Thomasen, L. J. O'Riordan, Th. Busch, and A. White. Chaotic few-body vortex dynamics in rotating Bose-Einstein condensates. *Physical Review Fluids*, 4(5):054701, 2019.

[14] J. Schloss, P. Barnett, R. Sachdeva, and Th. Busch. Controlled creation of three-dimensional vortex structures in Bose–Einstein condensates using artificial magnetic fields, 2019.

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

[16] 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. ISSN 0733-8724. doi: 10.1109/JLT.2003.808628.

[17] 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. doi: 10.1109/IMOC.2005.1580091.

[18] Rao M., Sun X., and Zhang M. A modified split-step Fourier method for optical pulse propagation with polarization mode dispersion. *Chinese Physics*, 12(5): 502–506, apr 2003. doi: 10.1088/1009-1963/12/5/307. URL `https://doi.org/10.1088%2F1009-1963%2F12%2F5%2F307`.

[19] C. Bayindir. Compressive split-step Fourier method. *arXiv preprint arXiv:1512.03932*, 2015.

[20] J. A. C. Weideman and B. M. Herbst. Split-step methods for the solution of the nonlinear Schrödinger equation. *SIAM Journal on Numerical Analysis*, 23(3): 485–507, 1986.

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

[22] J. C. Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016.

[23] J. Crank and P. 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.

[24] S. D. Conte and C. De Boor. *Elementary numerical analysis: an algorithmic approach*, volume 78. SIAM, 2017.

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

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

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

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

[29] R. Benzi, M. Colella, M. Briscolini, and P. Santangelo. A simple point vortex model for two-dimensional decaying turbulence. *Physics of Fluids A: Fluid Dynamics*, 4(5):1036–1039, 1992.

[30] K. W. Schwarz. Three-dimensional vortex dynamics in superfluid he 4: Homogeneous superfluid turbulence. *Physical Review B*, 38(4):2398, 1988.

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

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

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

[34] G. C. Wick. Properties of Bethe–Salpeter wave functions. *Physical Review*, 96 (4):1124, 1954.

[35] M. Harris et al. Optimizing parallel reduction in CUDA. *Nvidia developer technology*, 2(4):70, 2007.

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

[37] A. L. Fetter and J. D. Walecka. *Quantum Theory of Many-Particle Systems*. Dover Publications, 2003.

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

[39] 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. ISSN 1573-7357. doi: 10.1007/BF00683774. URL `http://dx.doi.org/10.1007/BF00683774`.

[40] A. Bulgac, M. M. Forbes, M. M. Kelley, K. J. Roche, and G. Wlazłowski. Quantized superfluid vortex rings in the unitary Fermi gas. *Physical review letters*, 112(2):025301, 2014.

[41] A. Aversa. The Gross-Pitaevskii equation: A non-linear Schrödinger equation, 2008.

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

[43] F. Dalfovo, S. Giorgini, L. P. Pitaevskii, and S. Stringari. Theory of Bose-Einstein condensation in trapped gases. *Rev. Mod. Phys.*, 71:463–512, Apr 1999. doi: 10.1103/RevModPhys.71.463. URL `http://link.aps.org/doi/10.1103/RevModPhys.71.463`.

[44] E. P. Gross. Structure of a quantized vortex in boson systems. *Il Nuovo Cimento (1955-1965)*, 20(3):454–477, 1961. ISSN 1827-6121. doi: 10.1007/BF02731494. URL `http://dx.doi.org/10.1007/BF02731494`.

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

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

[47] A. L. Fetter. Rotating trapped Bose-Einstein condensates. *Rev. Mod. Phys.*, 81: 647–691, May 2009. doi: 10.1103/RevModPhys.81.647. URL `http://link.aps.org/doi/10.1103/RevModPhys.81.647`.

[48] M. Ueda. *Fundamentals and new frontiers of Bose–Einstein condensation*. World Scientific, 2010.

[49] J. Allen and A. Misener. Flow of liquid Helium II. *Nature*, 142:643, 1938. doi: 10.1038/141075a0. URL `http://www.nature.com/nature/journal/v141/n3558/abs/141075a0.html`.

[50] A. B. Migdal. A phenomenological approach to the theory of the nucleus. *Soviet Physics JETP*, 10:176, 1960. doi: 10.1016/0029-5582(64)90294-9. URL `http://www.sciencedirect.com/science/article/pii/0029558264902949`.

[51] 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. ISSN 0036-8075. doi: 10.1126/science.269.5221.198. URL `http://science.sciencemag.org/content/269/5221/198`.

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

[53] A. L. Fetter and A. A. Svidzinsky. Vortices in a trapped dilute Bose-Einstein condensate. *Journal of Physics: Condensed Matter*, 13(12):R135, 2001. URL `http://stacks.iop.org/0953-8984/13/i=12/a=201`.

[54] K. W. Madison, F. Chevy, W. Wohlleben, and Jl. Dalibard. Vortex formation in a stirred Bose–Einstein condensate. *Physical review letters*, 84(5):806, 2000.

[55] M. D. Reichl and E. J. Mueller. Vortex ring dynamics in trapped Bose–Einstein condensates. *Phys. Rev. A*, 88:053626, Nov 2013. doi: 10.1103/PhysRevA.88.053626. URL `http://link.aps.org/doi/10.1103/PhysRevA.88.053626`.

[56] C. F. Barenghi, L. Skrbek, and K. R. Sreenivasan. Introduction to quantum turbulence. *PNAS*, 111:4647, 2014.

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

[58] L. J. O'Riordan, A. C. White, and Th. Busch. Moiré superlattice structures in kicked Bose–Einstein condensates. *Physical Review A*, 93(2):023609, 2016.

[59] L. J. O'Riordan and Th. Busch. Topological defect dynamics of vortex lattices in Bose–Einstein condensates. *Physical Review A*, 94(5):053603, 2016.

[60] J. R. Abo-Shaeer, C. Raman, J. M. Vogels, and W. Ketterle. Observation of vortex lattices in Bose-Einstein condensates. *Science*, 292(5516):476–479, 2001.

[61] 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. doi: 10.1103/PhysRevLett.92.040404. URL `http://link.aps.org/doi/10.1103/PhysRevLett.92.040404`.

[62] F. Chevy and J. Dalibard. Rotating Bose–Einstein condensates. *Europhysics News*, 37(1):12–16, 2006.

[63] E. Hodby, G. Hechenblaikner, S. A. Hopkins, O. M. Marago, and C. J. Foot. Vortex nucleation in Bose–Einstein condensates in an oblate, purely magnetic potential. *Physical review letters*, 88(1):010405, 2001.

[64] P. C. Haljan, I. Coddington, P. Engels, and E. A. Cornell. Driving Bose–Einstein-condensate vorticity with a rotating normal cloud. *Physical review letters*, 87(21): 210403, 2001.

[65] V. Bretin, S. Stock, Y. Seurin, and J. Dalibard. Fast rotation of a Bose–Einstein condensate. *Physical review letters*, 92(5):050403, 2004.

[66] P. Engels, I Coddington, P. C. Haljan, V. Schweikhard, and E. A. Cornell. Observation of long-lived vortex aggregates in rapidly rotating Bose–Einstein condensates. *Physical review letters*, 90(17):170405, 2003.

[67] A. Kumar, R. Dubessy, T. Badr, C. De Rossi, M.G. de Herve, L Longchambon, and H. Perrin. Producing superfluid circulation states using phase imprinting. *Physical Review A*, 97(4):043615, 2018.

[68] S. Moulder, S. Beattie, R. P. Smith, N. Tammuz, and Z. Hadzibabic. Quantized supercurrent decay in an annular Bose-Einstein condensate. *Phys. Rev. A*, 86: 013629, Jul 2012. doi: 10.1103/PhysRevA.86.013629. URL `https://link.aps.org/doi/10.1103/PhysRevA.86.013629`.

[69] 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. doi: 10.1103/PhysRevLett.83.5198. URL `https://link.aps.org/doi/10.1103/PhysRevLett.83.5198`.

[70] J. Denschlag, Je. E. Simsarian, Dl. L. Feder, C. W. Clark, La. A. Collins, J. Cubizolles, L. Deng, E. W. Hagley, K. Helmerson, W. P. Reinhardt, et al. Generating solitons by phase engineering of a Bose-Einstein condensate. *Science*, 287(5450): 97–101, 2000.

[71] B. Wu, J. Liu, and Q. Niu. Controlled generation of dark solitons with phase imprinting. *Physical review letters*, 88(3):034101, 2002.

[72] 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. doi: 10.1103/PhysRevLett.99.260401. URL `https://link.aps.org/doi/10.1103/PhysRevLett.99.260401`.

[73] M. Kasevich and S. Chu. Atomic interferometry using stimulated Raman transitions. *Phys. Rev. Lett.*, 67:181–184, Jul 1991. doi: 10.1103/PhysRevLett.67.181. URL `https://link.aps.org/doi/10.1103/PhysRevLett.67.181`.

[74] 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.

[75] Y. Shin, M. Saba, M. Vengalattore, T. A. Pasquini, C. Sanner, A. E. Leanhardt, M. Prentiss, D. E. Pritchard, and W. Ketterle. Dynamical instability of a doubly quantized vortex in a Bose–Einstein condensate. *Physical review letters*, 93(16): 160406, 2004.

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

[77] .F Maucher, S. A. Gardiner, and I. G. Hughes. Excitation of knotted vortex lines in matter waves. *New Journal of Physics*, 18(6):063016, 2016.

[78] 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. ISSN 0028-0836. doi: 10.1038/nature08609. URL `http://dx.doi.org/10.1038/nature08609`.

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

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

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

[82] K. Kasamatsu, M. Tsubota, and M. Ueda. Vortices in multicomponent bose–einstein condensates. *International Journal of Modern Physics B*, 19(11):1835–1904, 2005.

[83] W. Bao. Ground states and dynamics of multicomponent bose–einstein condensates. *Multiscale Modeling & Simulation*, 2(2):210–236, 2004.

[84] P. Mason. Ground states of two-component condensates in a harmonic plus gaussian trap. *The European Physical Journal B*, 86(11):453, 2013.

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

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

[87] F. L. Lewis, D. Vrabie, and V. L. Syrmos. *Optimal control*. John Wiley & Sons, 2012.

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

[89] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 01 1965. ISSN 0010-4620. doi: 10.1093/comjnl/7.4.308. URL `https://doi.org/10.1093/comjnl/7.4.308`.

[90] J. R. Koza. Genetic programming. 1997.

[91] T. G. Kolda, R. M. Lewis, and V. Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM review*, 45(3):385–482, 2003.

[92] R. M. Lewis, A. Shepherd, and V. Torczon. Implementing generating set search methods for linearly constrained minimization. *SIAM Journal on Scientific Computing*, 29(6):2507–2530, 2007.

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

[94] E. Torrontegui, S. Ibánez, S. Martínez-Garaot, M. Modugno, A. del Campo, D. Guéry-Odelin, A. Ruschhaupt, X. Chen, and J. G. Muga. Shortcuts to adiabaticity. In *Advances in atomic, molecular, and optical physics*, volume 62, pages 117–169. Elsevier, 2013.

[95] H. R. Lewis Jr and W. B. 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.

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

[97] 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. doi: 10.1103/PhysRevA.63.033601. URL `https://link.aps.org/doi/10.1103/PhysRevA.63.033601`.

[98] 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. doi: 10.1103/PhysRevLett.87.050403. URL `https://link.aps.org/doi/10.1103/PhysRevLett.87.050403`.

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

[100] K. 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. doi: 10.1103/PhysRevLett.89.170404. URL `https://link.aps.org/doi/10.1103/PhysRevLett.89.170404`.

[101] 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. doi: 10.1103/PhysRevA.79.033610. URL `https://link.aps.org/doi/10.1103/PhysRevA.79.033610`.

[102] D. W. Hallwood, T. Ernst, and J. Brand. Robust mesoscopic superposition of strongly correlated ultracold atoms. *Phys. Rev. A*, 82:063623, Dec 2010. doi: 10.1103/PhysRevA.82.063623. URL `http://link.aps.org/doi/10.1103/PhysRevA.82.063623`.

[103] C. Schenke, A. Minguzzi, and F. W. J. Hekking. Probing superfluidity of a mesoscopic Tonks–Girardeau gas. *Physical Review A*, 85(5):053627, 2012.

[104] A. Nunnenkamp, A. M. Rey, and K. Burnett. Generation of macroscopic superposition states in ring superlattices. *Phys. Rev. A*, 77:023622, Feb 2008. doi: 10.1103/PhysRevA.77.023622. URL `https://link.aps.org/doi/10.1103/PhysRevA.77.023622`.

[105] D. W. Hallwood, K. Burnett, and J. Dunningham. The barriers to producing multiparticle superposition states in rotating Bose–Einstein condensates. *Journal of Modern Optics*, 54(13-15):2129–2148, 2007.

[106] S. Martínez-Garaot, A. Ruschhaupt, J. Gillet, Th. Busch, and J. G. Muga. Fast quasiadiabatic dynamics. *Phys. Rev. A*, 92:043406, Oct 2015. doi: 10.1103/

PhysRevA.92.043406. URL https://link.aps.org/doi/10.1103/PhysRevA.92.043406.

[107] A. del Campo. Long-time behavior of many-particle quantum decay. *Phys. Rev. A*, 84:012113, Jul 2011. doi: 10.1103/PhysRevA.84.012113. URL https://link.aps.org/doi/10.1103/PhysRevA.84.012113.

[108] K. Lelas, T. Ševa, and H. Buljan. Loschmidt echo in one-dimensional interacting Bose gases. *Phys. Rev. A*, 84:063601, Dec 2011. doi: 10.1103/PhysRevA.84.063601. URL https://link.aps.org/doi/10.1103/PhysRevA.84.063601.

[109] 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. doi: 10.1103/PhysRevA.82.053403. URL https://link.aps.org/doi/10.1103/PhysRevA.82.053403.

[110] X. 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. doi: 10.1103/PhysRevLett.104.063002. URL https://link.aps.org/doi/10.1103/PhysRevLett.104.063002.

[111] S. Masuda and K. 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.

[112] 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. doi: 10.1103/PhysRevA.83.013415. URL https://link.aps.org/doi/10.1103/PhysRevA.83.013415.

[113] S. 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. doi: 10.1103/PhysRevA.86.063624. URL https://link.aps.org/doi/10.1103/PhysRevA.86.063624.

[114] 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. doi: 10.1364/OE.21.027093. URL http://www.opticsexpress.org/abstract.cfm?URI=oe-21-22-27093.

[115] S. Masuda, K. Nakamura, and A. del Campo. High-fidelity rapid ground-state loading of an ultracold gas into an optical lattice. *Phys. Rev. Lett.*, 113:063003, Aug 2014. doi: 10.1103/PhysRevLett.113.063003. URL https://link.aps.org/doi/10.1103/PhysRevLett.113.063003.

[116] J. R. 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.

[117] K. Czechowski, C. Battaglino, C. McClanahan, K. Iyer, P. Yeung, and R. 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.

[118] P. Wittek. Comparing three numerical solvers of the Gross-Pitaevskii equation, 2016. URL `https://web.archive.org/web/20171120181431/https://peterwittek.com/gpe-comparison.html`.

[119] X. Antoine and R. Duboscq. GPELab, a matlab toolbox to solve Gross–Pitaevskii equations i: Computation of stationary solutions. *Computer Physics Communications*, 185(11):2969–2991, 2014.

[120] P. Wittek and F. M. Cucchietti. A second-order distributed Trotter–Suzuki solver with a hybrid cpu–gpu kernel. *Computer Physics Communications*, 184 (4):1165–1171, 2013.

[121] M. Garland, S. Le Grand, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang, and V. Volkov. Parallel computing experiences with CUDA. *IEEE micro*, 28(4):13–27, 2008.

[122] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. 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.

[123] J. Nickolls and W. J. Dally. The GPU computing era. *IEEE micro*, 30(2):56–69, 2010.

[124] S. Wienke, P. Springer, C. Terboven, and D. an Mey. OpenACC—first experiences with real-world applications. In *European Conference on Parallel Processing*, pages 859–870. Springer, 2012.

[125] R. Chandra, L. Dagum, D. Kohr, R. Menon, D. Maydan, and J. McDonald. *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

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

[127] R. Slaw, 2013. URL `https://gist.github.com/realazthat/1eba733ab5cf3ae5fe2a#file-memory-access-coalescing-tex`.

[128] M. Harris. An efficient matrix transpose in CUDA C/C++. *Retrieved July*, 26: 2018, 2013.

[129] D. Foley and J. Danskin. Ultra-performance pascal GPU and NVLink interconnect. *IEEE Micro*, 37(2):7–17, 2017.

[130] V. Lončar, L. E. Young-S, S. Škrbić, P. Muruganandam, S. K. Adhikari, and A. 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.

[131] H. Wang, S. Potluri, D. Bureddy, C. Rosales, and D. 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.

[132] J. Fang, A. L. Varbanescu, and H. Sips. A comprehensive performance comparison of cuda and opencl. In *2011 International Conference on Parallel Processing*, pages 216–225. IEEE, 2011.

[133] K. Komatsu, K. Sato, Y. Arai, K. Koyama, H. Takizawa, and H. Kobayashi. Evaluating performance and portability of opencl programs. In *The fifth international workshop on automatic performance tuning*, volume 66, page 1, 2010.

[134] T. Besard, V. Churavy, A. Edelman, and B. De Sutter. Rapid software prototyping for heterogeneous and distributed platforms. *Advances in Engineering Software*, 132:29–46, 2019.

[135] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[136] T. Besard, C. Foket, and B. De Sutter. Effective extensible programming: unleashing julia on gpus. *IEEE Transactions on Parallel and Distributed Systems*, 30(4):827–841, 2018.

[137] J. Schloss and L. O'riordan. URL `https://gpue-group.github.io/`.

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

[139] R. Reyes and F. de Sande. Automatic code generation for GPUs in llc. *The Journal of Supercomputing*, 58(3):349–356, 2011.

[140] M. A. Davenport, M. F. Duarte, Y. C. Eldar, and G. Kutyniok. Introduction to compressed sensing. *Compressed sensing: theory and applications*, 105:106, 2012.

[141] A. Villois, G. Krstulovic, D. Proment, and H. 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.

[142] Y. Guo, X. Liu, C. Xiong, X. Xu, and C. Fu. Towards high-quality visualization of superfluid vortices. *IEEE transactions on visualization and computer graphics*, 24(8):2440–2455, 2018.

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

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

[145] A. El-Moursy, A. El-Mahdy, and H. El-Shishiny. An efficient in-place 3d transpose for multicore processors with software managed memory hierarchy. In *Proceedings of the 1st international forum on Next-generation multicore/manycore technologies*, page 10. ACM, 2008.

[146] G. Ruetsch and M. Fatica. *CUDA Fortran for scientists and engineers: best practices for efficient CUDA Fortran programming*. Elsevier, 2013.

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

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

[149] L. Biferale, G. Boffetta, A. Celani, B. J. Devenish, A. Lanotte, and F. Toschi. Lagrangian statistics of particle pairs in homogeneous isotropic turbulence. *Physics of Fluids*, 17(11):115101, 2005.

[150] A. Berera and R. D. J. G. Ho. Chaotic properties of a turbulent isotropic fluid. *Physical review letters*, 120(2):024101, 2018.

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

[152] N. Kyriakopoulos, V. Koukouloyannis, C. Skokos, and P. 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.

[153] V. Koukouloyannis, G. Voyatzis, and P. G. Kevrekidis. Dynamics of three non-corotating vortices in Bose–Einstein condensates. *Physical Review E*, 89(4): 042905, 2014.

[154] R. Navarro, R. Carretero-González, P. J. Torres, P. G. Kevrekidis, D. J. Frantzeskakis, M. W. Ray, E. Altuntaş, and D. S. Hall. Dynamics of a few corotating vortices in Bose–Einstein condensates. *Physical review letters*, 110 (22):225301, 2013.

[155] C. Nore, M. Abid, and M. E. Brachet. Kolmogorov turbulence in low-temperature superflows. *Physical review letters*, 78(20):3896, 1997.

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

[157] T. Araki, M. Tsubota, and S. K. Nemirovskii. Energy spectrum of superfluid turbulence with no normal-fluid component. *Physical review letters*, 89(14):145301, 2002.

[158] J. Salort, C. Baudet, B. Castaing, B. Chabaud, F. Daviaud, T. Didelot, P. Diribarne, B. Dubrulle, Y. Gagne, F. Gauthier, et al. Turbulent velocity spectra in superfluid flows. *Physics of Fluids*, 22(12):125102, 2010.

[159] H. 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.

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

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

[162] S. W. Seo, B. Ko, J. H. Kim, and Y. Shin. Observation of vortex-antivortex pairing in decaying 2d turbulence of a superfluid gas. *Scientific reports*, 7(1): 4587, 2017.

[163] K. E. Wilson, Z. L. Newman, J. D. Lowney, and B. P. Anderson. In situ imaging of vortices in Bose–Einstein condensates. *Physical Review A*, 91(2):023621, 2015.

[164] D. V. Freilich, D. M. Bianchi, A. M. Kaufman, T. K. Langin, and D. S. Hall. Real-time dynamics of single vortex lines and vortex dipoles in a Bose–Einstein condensate. *Science*, 329(5996):1182–1185, 2010.

[165] S. Serafini, L. Galantucci, E. Iseni, T. Bienaimé, R. N. Bisset, C. F. Barenghi, F. Dalfovo, G. Lamporesi, and G. Ferrari. Vortex reconnections and rebounds in trapped atomic Bose–Einstein condensates. *Physical Review X*, 7(2):021031, 2017.

[166] T. W. Neely, A. S. Bradley, E. C. Samson, S. J. Rooney, E. M. Wright, K. J. H. Law, R. Carretero-González, P. G. Kevrekidis, M. J. Davis, and B. P. Anderson. Characteristics of two-dimensional quantum turbulence in a compressible superfluid. *Physical review letters*, 111(23):235301, 2013.

[167] W. J. Kwon, G. Moon, J. Choi, S. W. Seo, and Y. Shin. Relaxation of superfluid turbulence in highly oblate Bose–Einstein condensates. *Physical Review A*, 90 (6):063627, 2014.

[168] G. Gauthier, M. T. Reeves, X. Yu, A. S. Bradley, M. Baker, T. A. Bell, H. Rubinsztein-Dunlop, M. J. Davis, and T. W. Neely. Negative-temperature Onsager vortex clusters in a quantum fluid. *arXiv preprint arXiv:1801.06951*, 2018.

[169] S. P. Johnstone, A. J. Groszek, P. T. Starkey, C. J. Billington, T. P. Simula, and K. Helmerson. Order from chaos: Observation of large-scale flow from turbulence in a two-dimensional superfluid. *arXiv preprint arXiv:1801.06952*, 2018.

[170] A. Aftalion and Q. 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.

[171] A. V. Zampetaki, R. Carretero-González, P. G. Kevrekidis, F. K. Diakonos, and D. J. Frantzeskakis. Exploring rigidly rotating vortex configurations and their bifurcations in atomic Bose–Einstein condensates. *Physical Review E*, 88(4): 042914, 2013.

[172] T. Zhang, J. Schloss, A. Thomasen, L. J. O'Riordan, Th. Busch, and A. White. URL `https://journals.aps.org/prfluids/abstract/10.1103/PhysRevFluids.4.054701#supplemental`.

[173] A. Wolf, J. B. Swift, H. L. Swinney, and J. A. Vastano. Determining Lyapunov exponents from a time series. *Physica D: Nonlinear Phenomena*, 16(3):285–317, 1985.

[174] D. H. Wacks, A. W. Baggaley, and C. F. Barenghi. Large-scale superfluid vortex rings at nonzero temperatures. *Physical Review B*, 90(22):224514, 2014.

[175] 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. doi: 10.1103/PhysRevLett.86.2926. URL `http://link.aps.org/doi/10.1103/PhysRevLett.86.2926`.

[176] M. J. H. Ku, B. Mukherjee, T. 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.

[177] M. Robin Matthews, B. P. Anderson, P. C. Haljan, D. S. Hall, C. E. Wieman, and E. A. Cornell. Vortices in a Bose–Einstein condensate. *Physical Review Letters*, 83(13):2498, 1999.

[178] T. Yefsah, A. T. Sommer, M. J. H. Ku, L. W. Cheuk, W. Ji, W. S. Bakr, and M. W. Zwierlein. Heavy solitons in a fermionic superfluid. *Nature*, 499(7459): 426, 2013.

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

[180] I. M. Cohen D. R. Dowling, P. K. Kundu. *Fluid Mechanics*. Academic Press, Boston, fifth edition edition, 2012. ISBN 978-0-12-382100-3. doi: http://dx.doi.org/10.1016/B978-0-12-382100-3.10017-4. URL `http://www.sciencedirect.com/science/article/pii/B9780123821003100174`.

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

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

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

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

[185] 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):097101, 2014. doi: http://dx.doi.org/10.1063/1.4894745. URL `http://scitation.aip.org/content/aip/journal/pof2/26/9/10.1063/1.4894745`.

[186] K. Shariff and A. Leonard. Vortex rings. *Annual Review of Fluid Mechanics*, 24 (1):235–279, 1992. doi: 10.1146/annurev.fl.24.010192.001315.

[187] K. W. Schwarz. Interaction of quantized vortex rings with quantized vortex lines in rotating he II. *Phys. Rev.*, 165:323–334, Jan 1968. doi: 10.1103/PhysRev.165.323. URL `http://link.aps.org/doi/10.1103/PhysRev.165.323`.

[188] T. T. Lim and T. B. Nickels. *Fluid Vortices*. Springer Netherlands, Dordrecht, 1995. ISBN 978-94-011-0249-0. doi: 10.1007/978-94-011-0249-0_4. URL `http://dx.doi.org/10.1007/978-94-011-0249-0_4`.

[189] M. S. Paoletti and D. P. Lathrop. Quantum turbulence. *Annual Review of Condensed Matter Physics*, 2(1):213–234, 2011. doi: 10.1146/annurev-conmatphys-062910-140533. URL `http://dx.doi.org/10.1146/annurev-conmatphys-062910-140533`.

[190] 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. doi: 10.1103/PhysRevA.61.013604. URL `http://link.aps.org/doi/10.1103/PhysRevA.61.013604`.

[191] 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. doi: 10.1103/PhysRevLett.86.3934. URL `http://link.aps.org/doi/10.1103/PhysRevLett.86.3934`.

[192] N. S. Ginsberg, J. Brand, and L. V. Hau. Observation of hybrid soliton vortex-ring structures in Bose–Einstein condensates. *Physical review letters*, 94(4): 040403, 2005.

[193] 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.

[194] J. Ruostekoski and Z. Dutton. Engineering vortex rings and systems for controlled studies of vortex interactions in Bose–Einstein condensates. *Physical Review A*, 72(6):063626, 2005.

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

[196] M. Abad, M. Guilleumas, R. Mayol, and M. Pi. Vortex rings in toroidal Bose–Einstein condensates. *Laser Physics*, 18(5):648–652, 2008. ISSN 1555-6611. doi: 10.1134/S1054660X08050162. URL `http://dx.doi.org/10.1134/S1054660X08050162`.

[197] D. Kleckner, L. H. Kauffman, and W. T. M. Irvine. How superfluid vortex knots untie. *Nature Physics*, 12(7):650, 2016.

[198] R. L. Ricca, D. C. Samuels, and C. F. Barenghi. Evolution of vortex knots. *Journal of Fluid Mechanics*, 391:29–44, 1999.

[199] C. W. Duncan, C. Ross, N. Westerberg, M. Valiente, B. J. Schroers, and P. Öhberg. Linked and knotted synthetic magnetic fields. *Physical Review A*, 99(6): 063613, 2019.

[200] J. Dalibard, F. Gerbier, G. Juzeliūnas, and P. Öhberg. Colloquium: Artificial gauge potentials for neutral atoms. *Reviews of Modern Physics*, 83(4):1523, 2011.

[201] M. Mochol and K. Sacha. Artificial magnetic field induced by an evanescent wave. *Scientific Reports*, 5, Jan 2015. URL `http://dx.doi.org/10.1038/srep07672`. Article.

[202] J. M. Ward, D. G. O'Shea, B. J. Shortt, M. J. Morrissey, K. Deasy, and S. Nic Chormaic. Heat-and-pull rig for fiber taper fabrication. *Review of scientific instruments*, 77(8):083105, 2006.

[203] L. Tong, R. R. Gattass, J. B. Ashcom, S. He, J. Lou, M. Shen, I. Maxwell, and E. Mazur. Subwavelength-diameter silica wires for low-loss optical wave guiding. *Nature*, 426(6968):816, 2003.

[204] A. Yariv et al. *Optical electronics in modern communications*, volume 1. Oxford University Press, USA, 1997.

[205] E. Vetsch, D. Reitz, G. Sagué, R. Schmidt, S. T. 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.

[206] C. Lacroûte, K. S. Choi, A. Goban, D. J. Alton, D. Ding, N. P. Stern, and H. J. Kimble. A state-insensitive, compensated nanofiber trap. *New Journal of Physics*, 14(2):023056, 2012.

[207] T. Nieddu, V. Gokhroo, and S. Nic Chormaic. Optical nanofibres and neutral atoms. *Journal of Optics*, 18(5):053001, 2016.

[208] 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.

[209] L. Russell, K. Deasy, M. J. Daly, M. J. Morrissey, and S. Nic Chormaic. Sub-doppler temperature measurements of laser-cooled atoms using optical nanofibres. *Measurement Science and Technology*, 23(1):015201, 2011.

[210] R. Kumar, V. Gokhroo, K. Deasy, A. Maimaiti, M C. Frawley, C Phelan, and S. Nic Chormaic. Interaction of laser-cooled 87rb atoms with higher order modes of an optical nanofibre. *New Journal of Physics*, 17(1):013026, 2015.

[211] F. 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. doi: 10.1103/PhysRevA.70.063403. URL `http://link.aps.org/doi/10.1103/PhysRevA.70.063403`.

[212] R. Sachdeva and Th. Busch. Creating superfluid vortex rings in artificial magnetic fields. *Physical Review A*, 95(3):033615, 2017.

[213] M. Cozzini, S. Stringari, V. Bretin, P. Rosenbusch, and J. Dalibard. Scissors mode of a rotating Bose–Einstein condensate. *Physical Review A*, 67(2):021602, 2003.

[214] D. Guéry-Odelin and S. Stringari. Scissors mode and superfluidity of a trapped Bose–Einstein condensed gas. *Physical review letters*, 83(22):4452, 1999.

[215] O. M. Marago, S. A. Hopkins, J. Arlt, E. Hodby, G. Hechenblaikner, and C. J. Foot. Observation of the scissors mode and evidence for superfluidity of a trapped Bose–Einstein condensed gas. *Physical review letters*, 84(10):2056, 2000.

[216] N. L. Smith, W. H. Heathcote, J. M. Krueger, and C. J. 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.

[217] F. Zambelli and S. Stringari. Quantized vortices and collective oscillations of a trapped Bose–Einstein condensate. *Physical review letters*, 81(9):1754, 1998.

[218] S Stringari. Superfluid gyroscope with cold atomic gases. *Physical review letters*, 86(21):4725, 2001.

[219] V. G. Minogin and S. 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.

[220] K. O. Hill and G. Meltz. Fiber Bragg grating technology fundamentals and overview. *Journal of lightwave technology*, 15(8):1263–1276, 1997.

[221] J Revels, T. Besard, V. Churavy, B. De Sutter, and J. P. Vielma. Dynamic automatic differentiation of GPU broadcast kernels. *arXiv preprint arXiv:1810.08297*, 2018.