

The System: 2D Pseudospin-1/2 Spinor BECs

Coupled Gross-Pitaevskii Equation (GPE):

- Bose-Einstein condensates (BECs) can be well understood through mean-field theory, as described by the Gross-Pitaevskii equation (GPE) [1].
- For spinor BECs with two spin components j, k , we use the coupled form [2, 3],

$$i\hbar\frac{\partial\psi_j}{\partial t} = \left[-\frac{\hbar^2}{2m}\nabla^2 + \mathcal{V}(\mathbf{r},t) + g_j|\psi_j|^2 + g_{jk}|\psi_k|^2 \right] \psi_j$$

Kinetic Energy Trapping Potential Intra-species s-wave scattering Inter-species s-wave scattering

- In 2D, we solve for the time-evolution of the coupled spins from the above GPE. We assume a Gaussian density profile along the third dimension.

Numerical Simulation Techniques:

- Wavefunctions are discretized onto Python `complex128` arrays; time-evolution is approximated to be unitary over small discrete time-scales [3,4]:

$$\psi(\mathbf{r}, t + \delta t) \simeq e^{-i\hat{\mathcal{H}}(\mathbf{r})\delta t/\hbar}\psi(\mathbf{r}, t)$$

- Evolution operators are also represented as 2D arrays; a single step forward in time is thus computed as a Hadamard (elementwise) product.

- We avoid taking numerical derivatives for the kinetic energy by transforming via FFT into and from Fourier space, where the Laplacian is simply a 2D array.

- The Issue:** This algorithm is extremely computationally expensive!

For a single time step:

# FFT	24
# Hadamard	>60

The Solution: Python GPU Acceleration

- It is well known that a graphics processing unit (GPU) can be used to dramatically speed up certain computations [5, 6].
- The heavy reliance of our algorithm on FFTs and Hadamard products makes GPUs as a computational tool an excellent choice.

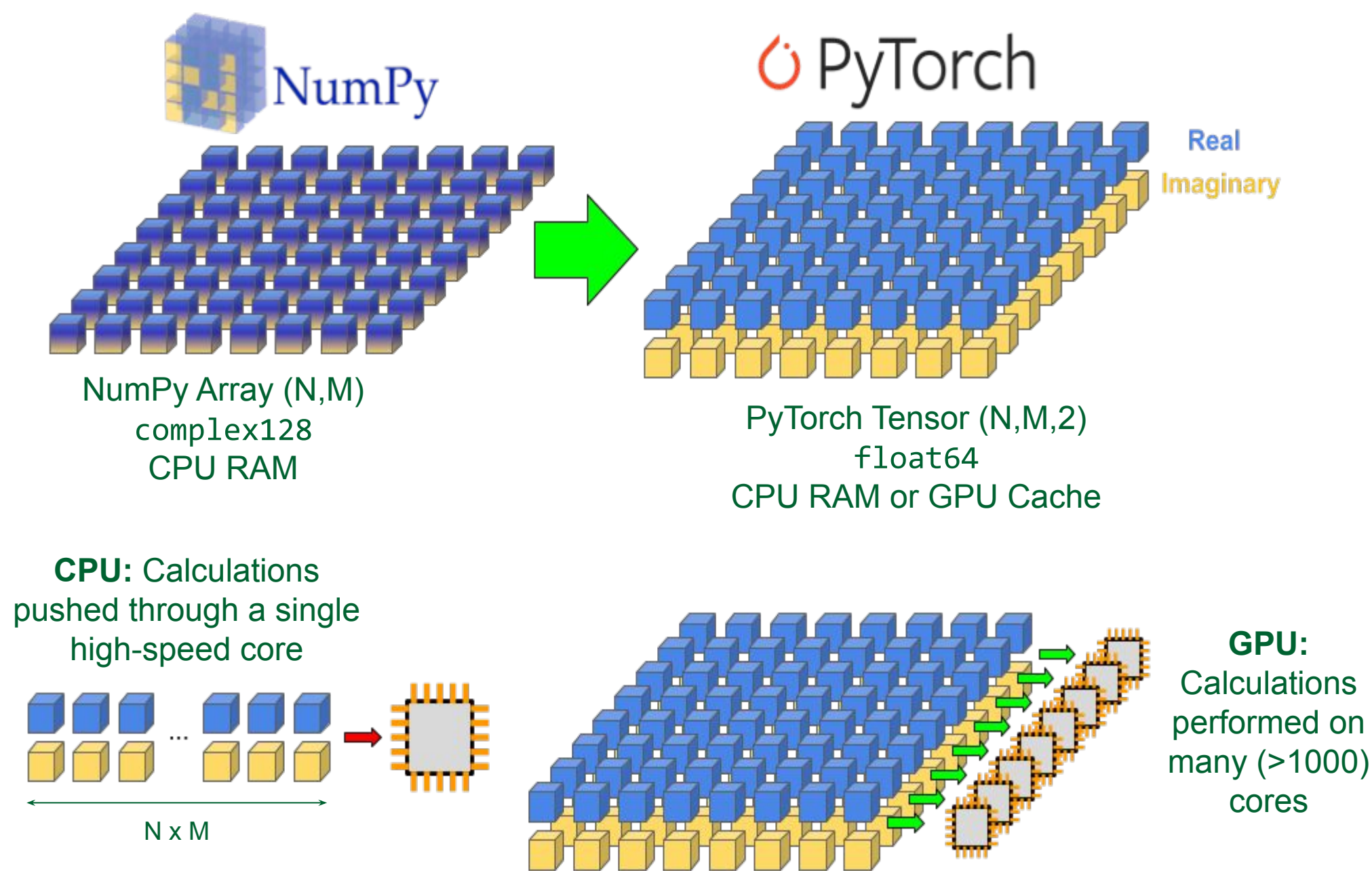
GPU Computing in a Nutshell:

	CPU	GPU
Mantra	Minimize latency; instructions are executed sequentially with minimal time between.	Maximize throughput; most instructions are executed simultaneously, but at individually slower rates.
Method	Divide problems temporally; schedules computations sequentially, devoting all of its resources to each.	Divide problems spatially; sends data to devices/resources best suited for that computation, all simultaneously.
Advantages	- Performing sequential operations - Performing complicated arithmetic with any data type	- Performing <i>similar calculations</i> on multiple elements of data which are <i>independent</i> of each other - Computing special functions on arrays (sin, cos, exp, ...)
Drawbacks	- All operations are done sequentially, no parallelism - Very general-purpose	- Bus times (time spent going to/from GPU) - Limited performance with certain data types
Technology	Diminishing returns in higher transistor count/density	- Performance curves beyond Moore's law - Large benefit to higher transistor count/density

- Previous work accelerated their GPE problem with [CUDA](#) [7], but without an interface to common languages like Python. Open-source tools can accelerate the GPE with a GPU, with Python and MATLAB interfaces [8], however, all these tools remain very problem-specific and not universally-applicable.

Our Implementation

- Use [PyTorch](#), a “native-Python”, open-source machine-learning library.
- PyTorch syntax is purposefully close to NumPy, but with substantially more control of data storage and computing resources.



The Upshot: Adapting our previous GPE code to PyTorch, with the above structure [9], was remarkably easy, and the gains are substantial.
→ This approach can be generally applied to accelerate **existing** Python GPE code without requiring a major overhaul.

Acceleration Results

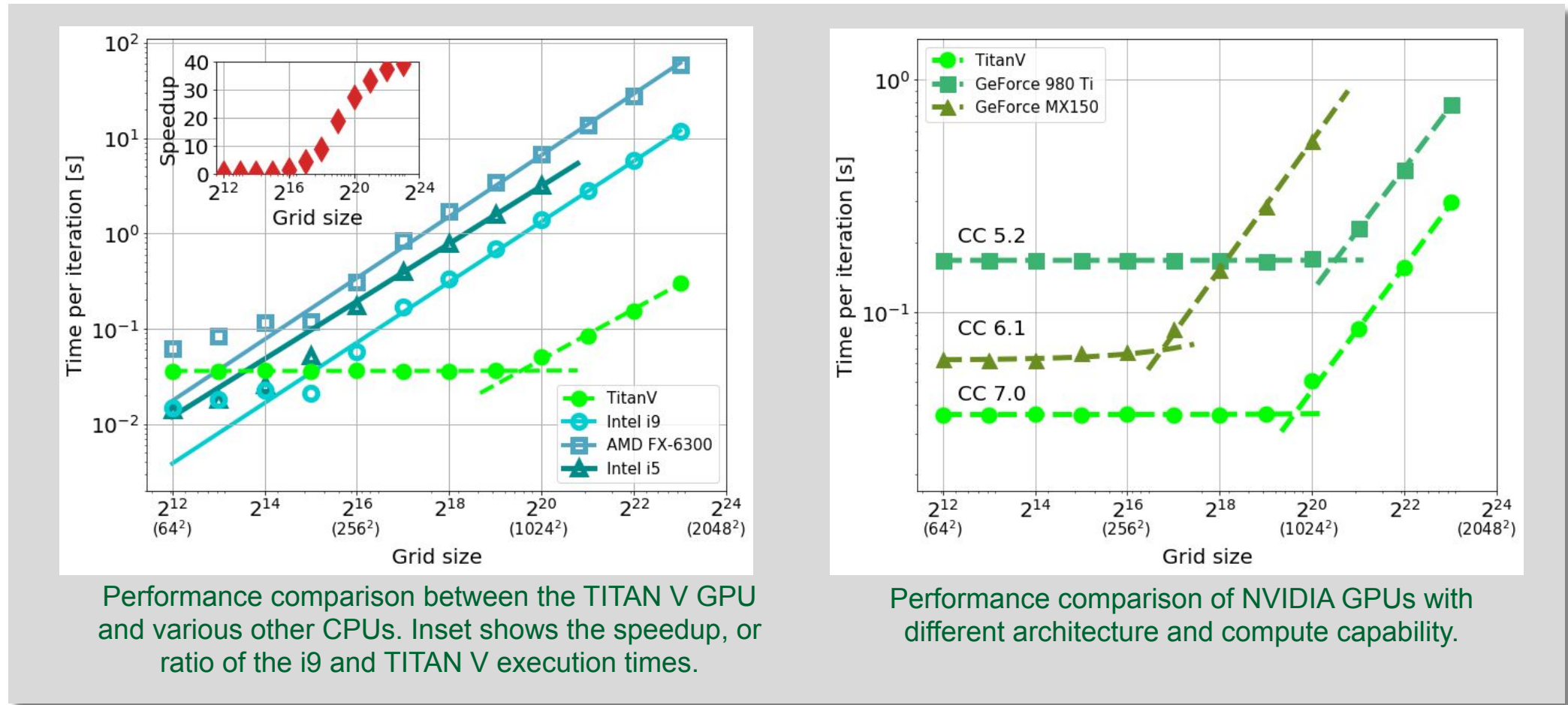
Summary of Hardware

Our Pytorch implementation can execute on any of our CPUs (left column) or CUDA-enabled NVIDIA GPUs (right column; CC is the “[compute capability](#)”). The hardware in the first two rows are installed on custom-built workstations; those in the third row are packaged together in a commercial laptop.

 Intel i9-9900K 3.7 GHz (Boost to 5 GHz) 32 GB RAM 8 Cores	 NVIDIA TITAN V 5120 CUDA cores 12 GB Cache CC 7.0 - Volta Architecture
 AMD FX-6300 3.5 GHz (Boost to 4.1 GHz) 16 GB RAM 6 Cores	 NVIDIA GeForce GTX 980 Ti 2816 CUDA cores 6 GB Cache CC 5.2 - Maxwell Architecture
 Intel i5-7200U 2.5 GHz (Boost to 3.1 GHz) 8 GB RAM 2 Cores	 NVIDIA GeForce MX150 384 CUDA cores 2 GB Cache CC 6.1 - Pascal Architecture

Performance Benchmarks

Our PyTorch code contains a function to time-evolve both spinor components through an individual discrete time step. This function is typically called ~1000 times during a given simulation. For various grid sizes (N x M) of double-precision values, we timed multiple evaluations of this function and averaged the results:



Discussion

The GPU times initially do not scale with increasing grid sizes, compared to the $O(n^{1.04})$ for the CPUs. The kinks we observe in the GPU times are due the device being unable to parallelize such large grid operations at once. Instead, it begins to separate and compute batched parallel operations sequentially.

Each GPU has a computational overhead - even at small grid sizes - which depends on its architecture and CC. This sets a lower limit on iteration times that can be achieved.

Intel i9 [min]	23.17
TITAN V [min]	0.84

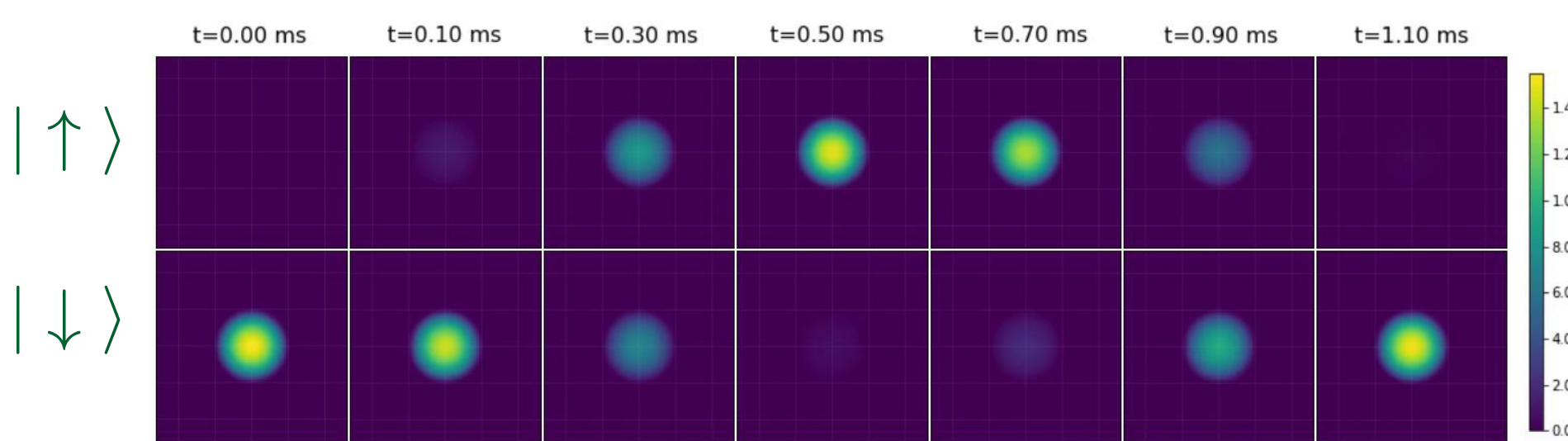
Comparison of total execution time for 1000 time steps with a typical 1024 x 1024 (2²⁰) grid size for our best CPU and GPU.

	FFT		Hadamard	
# Bits	32	64	32	64
i9 [ms]	30.9	25.4	7.75	15.5
TITAN V [ms]	0.805	0.922	1.9	3.62

Time per function call with a grid size of 1024 x 1024, for single-precision (32 bits) and double-precision (64 bits) numbers.

Example Spinor Simulations

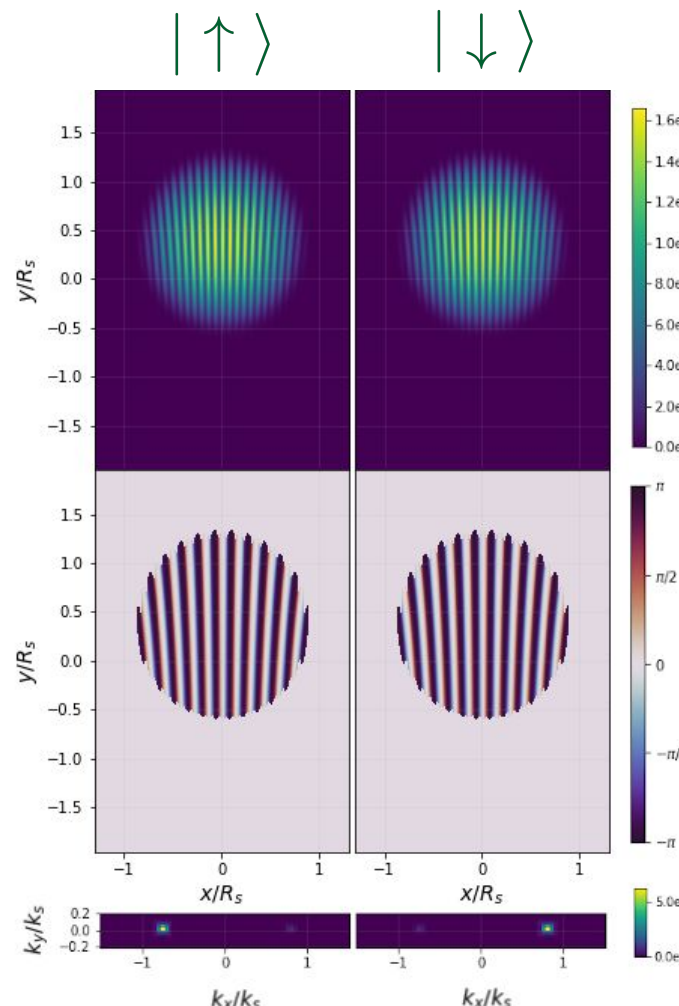
Real-time Dynamics: Rabi Flopping



Ground State Solutions:

In addition to studying real-time dynamics, we can propagate in imaginary-time to find ground state solutions of the GPE.

The figure to the right shows (from top to bottom) the real-space density, phase, and k-space density of the ground state of a Raman-coupled spinor BEC with a Rabi frequency that increases linearly from the bottom of the window to the top ($\Omega \in [0, 4.5] E_L$).



References

- [1] F. Dalfovo, S. Giorgini, L. P. Pitaevskii, and S. Stringari, *Rev. Mod. Phys.* **71**, 463 (1998).
- [2] J. Radic, T. A. Sedrakyan, I. B. Spielman, V. Galitski, *Phys. Rev. A* **84**, 063604 (2011).
- [3] W. Bao, *Multiscale Modeling & Sim.*, **2(2)**, 210-236 (2004).
- [4] W. Bao, and Y. Cai, *Kin. & Rel. Mods.* **6**, 1 (2013).
- [5] J. D. Owens, et. al. *Computer graphics forum*, vol. **26(1)** 80-113 (2007).
- [6] J. D. Owens, et. al. *Proceedings of the IEEE* **96(5)** 879-899 (2008).
- [7] V. Loncara, et al., *Comput. Phys. Commun.* **200** (2016).
- [8] J.R. Schloss and L. J. O’Riordan, *J. Open Source Softw.*, **3(32)**, 1037 (2018).
- [9] O. L. Berman, et al., *Phys. Rev. E* **91**, 062901 (2015).