TensorFlow Data Backend for QuTiP

Google Summer of Code 2021 - QuTiP

Asier Galicia*

April 13, 2021

Abstract

Qutip is a python package that presents a comprehensive toolbox to simulate the dynamics of open quantum systems. Its core class is `Qobj` and it is used to represent a variety of Quantum objects such as kets or Hamiltonians, which has been improved in the last Google Summer of Code to extend its flexibility. It can now handle multiple representations of its data efficiently, for instance dense or sparse matrices. Here we propose to extend the implementation of this class allowing it to handle TensorFlow's data type which will store and manipulate the data of `Qobj` using a Graphical Processing Unit (GPU). In this way, QuTiP will be able to take advantage of the computing capabilities that the GPU offers.

I. Technical overview

Quantum mechanics is one of the most successful theories in physics that has lead to many great discoveries in many fields of science in the last decades. Unfortunately only the most simple systems described by quantum mechanics can be solved analytically. This means that numerical methods are essential to understand most of the complex phenomena that can be described by using quantum mechanics. QuTiP is a powerful tool in python that can be used to simulate and understand such complex phenomena.

Recently, the QuTiP project has invested a lot of effort to provide a new version of the package, QuTiP 5.0. One of the achievements of this new version consist on improving the flexibility of its core class Qobj which is used to represent most of the desired quantum objects, such as a ket or a Hamiltonian. This was achieved in the last Google Summer of Code with Jake Lishman's proposal and lead to a Qobj class that can now seamlessly store and operate multiple data formats (such as dense or sparse matrices) internally.

In this proposal we would like to continue extending the capabilities of the Qobj class to represent and operate with data stored in a Graphical Processing Unit (GPU). Nowadays GPUs have become a very powerful and widespread available hardware. These processing units have the advantage of handling a higher number of floating point operations per second (FLOPS) than the CPU, which makes them a promising candidate to carry the numerically heavy computation task that are required to simulate quantum dynamics. However, the computing paradigm of the GPU differs from the CPU one in that it is heavily based on parallel operations. It is not clear only from a TFLOPS comparison whether and algorithm will work faster in a GPU than in a CPU since it heavily depends on the nature of the algorithm.

When solving problems in quantum mechanics, we are mostly interested in matrix multiplication, eigenvalue solver and ordinary differential equation (ODE) solver using in all these cases complex numbers. Regarding to matrix multiplication, we expect to see a improvement in the performance of QuTiP when using a GPU only for large system sizes[1]. In the case of the ODE

*a.galiciamartinez@student.tudelft.nl

[1]Based on the comparison done in the cupy

solvers, there are already a few implementations in the GPU for other programming languages (for instance, the DiffEqGPU library in Julia and Boost library in C++). These packages report that solving an ODE in a GPU instead of in a CPU is desirable in two situations: 1) when the system size is very large (odeint library specifically claims a system size greater than 10^6 which is roughly the system size of 10 qubits or 1000 energy levels) or 2) when the ODE is of a small size but needs to be solved for a large number of parameters. This last case is particularly interesting for the Monte Carlo solvers that simulate the dynamics of the system using a smaller representation of the system (a ket instead of a density matrix) and a probabilistic approach. In this case, the solution is obtained by averaging over multiple independent runs of the same ODE that performs some stochastic decisions based on the problem to be solved.

During this project we will first follow an of the shelf approach to include the data types of TensorFlow as part of the data types that the class Qobj is able to handle. The reason to chose TensorFlow over other options, such as cupy, is its ability to scale in the future to multi GPU systems and that it already implements most of the algorithms we need (including the ODE solver). We will also benchmark both the performance of the CPU based data type against the GPU based ones to obtain an approximate range of parameters for when using a GPU outperforms the CPU. These benchmarks will be meant to help the user choose the appropriate backend for the specific problem they are trying to solve.

II. Timeline

The project starts on June 7th (although there is a community bonding period before) and consist of 10 weeks working for an average of 18 hours a week. I would like to add here that this is my only activity planned for this period of time.

The project will be divided in 4 phases, including the boning period, and it will be hosted in its own repository for organization purposes.

Community bonding period

- *Familiarize with the core class Qobj and the data layer*. Jake Lishman has written documentation[2] that extensively addresses the creation of new data types in QuTiP. For this, I will also familiarize with Cython since it is used to define the Qobj data types.
- *Familiarize with TensorFlow and CUDA programming*. Even thought the goal it to use TensorFlow, it would be advisable to also familiarize with CUDA programming for future upgrades or efficiency improvements. This should also lead to a better understanding on how the GPU could improve numerical simulations of open quantum systems.
- *Interact with the community and begin with the creation of the blog*. I will continue contributing to the QuTiP project during this period of time, figuring out the details necessary for the project. I will also begin setting up the blog that will be used to keep track of the progress in the project.

Week 1-2. ***Design and implement the benchmark structure.***

During this phase I will design and implement a set of benchmarks that will test the performance of the currently implemented data types. This benchmark will be implemented using the package `pytests-benchmark` which will allow to include in the future these benchmarks for regression test purposes if we find them suitable. The operations that will be first benchmarked are:

[2] Link to documentation page.

- Matrix multiplication.
- Eigenvalue decomposition.
- ODE solver (`mesolve`).

These are in most of the cases the operations that take most of the computational time and hence are subjected to optimization. We will design the benchmarks to depend on the size of the system for later comparison with the GPU implementation of the Qobj class. Finally we will explore the possibility to run the benchmarks automatically.

Week 3-5. ***Include TensorFlow data type and basic operations.***

- During these weeks I will include the TensorFlow data type for the Qobj class. This will most likely be the most complex phase since it will require me to familiarize with the core of QuTiP and to make sure that all the Qobj methods work properly with the new data type. Most of these methods will only require to call the corresponding method in the TensorFlow library, although some others will require a little bit more thinking (such as the partial trace operation), since there is no equivalent in the TensorFlow library.
- I will also compare the performance of each method in different case scenarios (large and small system size) to conclude when the GPU can provide an improvement in the computation time.
- I will also ensure that the rest of QuTiP's functionality works well with the new data type, including the necessary tests for this.

Week 6-8. ***Inlude ODE solver for new data type.***

- During this period I will focus on implementing an ODE solver to solve the Linbald equation. Currently, the main ode solver, `qutip.solver.mesolve._generic_ode_solve` uses scipy's `integrate` module to solve the required ODEs. With the new TensorFlow data type it will be necessary to include a solver that makes use of the GPU. For this we will make use of TensorFlow's ODE solver. The challenge here will be to include all the functionality that the current ODE solver already has in the new solver.
- If I have enough time, we would like to adapt `qutip/solve/mcsolve.py` to also make use of the GPU since the Monte Carlo approach to solve quantum dynamics seems to be specially well suited to make use of the computing power that the GPU offers.

Week 9-10 ***Documentation, tests and wraping up.***

I will leave the last two weeks free to accommodate any delays that may happen during the project. During this time I will also ensure that the code is properly documented and tested. In particular I will include documentation that shows the use cases were the implemented TensorFlow backend provides an advantage over the other implemented data types.

III. Development experience

I am a Master student at the Delft University of Technology and I am currently doing my Master's thesis project on the "*Hyperpolarisation of nuclear spins using a Nitrogen-Vacancy centre in diamond*" under the supervision of Tim Taminiau and Conor Bradley. During the master's thesis I had the opportunity to a attend a course in computational physics. This was a project based course (where we used python as programming language) that required us to make use of GitLab to

track our progress. The last of these projects can be found in my GitHub repository. During this course I familiarized myself with some of the scientific libraries in python such as Numpy, Scipy, MatplotLib and Pandas. Furthermore, during my master's thesis project I was required to perform simulations of the dynamics of Nitrogen-Vacancy systems for which I made use of the QuTiP library. Unfortunately, I can not share my most recent code from the master's thesis project due to intellectual property issues.

i. QuTiP contributions

I have one pull request merged to the QuTiP master branch already:

- #1478: Fix issue #1460 The function mesolve was not including the complex part of the expectation value when the initial state was a not Hermitian. This led to an error in correlation_2op_1t() which uses an initial state that is not Hermitian to compute the correlation. I also included tests for the relevant changes made in this PR.

IV. Why this project

Ever since I started using Python I was fascinated by the amount of high quality open source libraries that were available for scientific computing. I have been using these libraries extensively during my Bachelor and Master's degree and I always felt that I should at some point contribute to these projects. The Google Summer of Code seems the best way to start with this and I chose QuTiP as the project to contribute to due to my background in physics.

Regarding to why I chose this particular project that involves working with a GPU, is that, even though I do not have any relevant prior experience working with GPUs, I find them quite interesting pieces of hardware. I am amused by the incredible results that the computing capabilities of the GPUs have enabled in the Machine Learning field and I am eager to see how much of this computational power we can borrow in physics to speed up numerically heavy simulations.