MatSuite Writeup

George Saussy

October 18, 2016

In this project we attempt to build a GPU optimized implementation of Krylov wave propagation for quantum states. In the process we, implement a small suite of functions designed to handle matrices. This system will be used to train an neural network that will speed up the calculation of the propagation of quantum states. The neural network can then be used to aid in the design of molecules which would be computationally infeasible otherwise.

1 Motivation

Physicist must frequently calculate the propagation of a quantum state. This calculation is often extremely computationally expensive, even for simple systems. This problem is only exasperated when many bodies are introduced or the Hamiltonian is not well behaved.

However, as physical methods play a greater role in chemistry, more efficient means of calculating wave propagation in many body systems must be explored. The aim of this research should be to bring down the cost of understanding the the behaviour of particles and molecules in generic settings, so that the methods found can be generally applied without needing to be customized for a given problem.

In the interest of generality the most generic form of a quantum system is given by the Schrodinger equation, $i\hbar d_t|\phi\rangle = H|\phi\rangle$, where $|\phi\rangle$ is the quantum state and H is the Hamiltonian under which the system is evolving. The generic solution to this equation is then

$$|\phi(t)\rangle = \exp(-iHt/\hbar)|\phi_0\rangle$$

However, this computation is practice is very expensive because of the exponential. In particular the generic definition of $\exp(A)$ for matrix A is given by

$$\exp(A) = \sum_{k=0}^{\infty} A^k/k!$$

which does not converge quickly for $\|A\| >> 1$. (In fact, the error term given only a constant number of terms are used in the above equation will grow exponentially with $\|A\|$.) Bringing the computation time for matrix exponentiation down would allow theorists to consider systems with a greater set of pure quantum states, and potentially to numerically consider the behaviour of systems that were previously infeasible to treat.

In this project, I present the design and implementation of a suite of GPU optimized software to speed up these computations.

2 Mathematical background

3 Project design

(More detailed documentation for the suite can be found in the code. Run `doxygen` in the repository's home directory and a directory containing technical documentation in HTML will be generated.)

This project has two overall goals: 1) to create code that will perform wave propagation as efficiently as possible, and 2) to make this implementation as portable and easy to use as possible to use as possible. To that end, there will be four functions implemented, each representing different approximation methods.

The primary computational structure used in the program is a `struct SqMat`. This struct is a square matrix of real numbers, and the current implementation of the the library only supports real matrices.

```
double* expv(struct SqMat mat, double t, double * v, double tau, double minerr);
```

A function to perform the the Krylov estimation for $\exp(t * mat)v$ where `tau` is the time step used and `minerr` is the minimum error allowed (must be greater than machine error on double operations). This algorithm is optimized for the case `mat` is large and sparse. (It is the user's responsibility ensure that $Dim(v) = \sqrt{Dim(mat)}$.)

```
struct SqMat expPade(struct SqMat mat, int p, int q);
```

An implementation of an older algorithm to implement the p,q-Pade approximation for $\exp(mat)$. By default, $p = q = 14$ should be used.

```
struct SqMat  struct SqMat expTaylor(struct SqMat mat, int k);
```

An implementation of the Taylor approximation of

$$\exp(mat) \approx \sum_{n=0}^{k} (mat)^n / n!$$

This should not be used in practice but is included for completion.

A small set of example programs demonstrating how the functions should be implemented in the near future.

4 TODO

The project now forks into two sub-project: 1) begin the GPU optimization process and 2) implement more features.

While GPU optimization is simple enough to understand, the practical steps are more finely detailed. As of now, we are tentatively considering using SPIR-V from the Vulkan API. The advantaged of this implementation over something like CUDA or OpenCL is that SPIR-V does not require the programmer to configure the the code for any possible computer architecture. SPIR-V allows the programmer to focus on higher level program design. One draw back on SPIR-V,

especially in comparison to CUDA, is that the Vulkan API has not been generally accepted, so one older systems, the library may not run. (Vulkan is an API slotted to replace OpenGL, and SPIR-V is its GPU coding paradigm.)

The other direction, implementing more features, is more strait forward. First the structs `struct Complex` and `struct SqMatCplx` for complex numbers and complex matrices respectively. These will be used in a new implementation of the above functions. These new functions are absolutely necessary, as nearly all Hamiltonians are complex. (The decision to focus on real matrices first was to get a functioning implementation ready as soon as possible, even with limited functionality.) Of course once the functions are implemented, they will need to be GPU optimized as well, but this should be more strait forward as

5 Results

FORTHCOMING

6 Discussion

FORTHCOMING