Parallel **Si**mulation of **Q**uantum **Co**mputation by **P**ractically **A**diabatic **T**ime-dependent **H**amiltonians (PSiQCoPATH) Progress Report

Eric Fellheimer and Mark Rudner

April 14, 2005

Since the time of the project proposals, Eric and I have done a lot of work on hammering out the details of our projet and beginning its implementation. The first task was to figure out exactly what we wanted PSiQCoPATH to calculate. To this end, we consulted with a few experts in the field of quantum computation and adiabatic evolution.

Although the current direction of research-level work in this field is to look for "efficient" ways to simulate systems of a large number of qubits, we chose to make our code "exact." Just how exact it is will be discussed below, but because our main goal is to simulate physics which inherently scales exponentially in memory we do not feel compelled to find ways of cheating nature. In fact, the whole idea behind the power of a quantum computer is that it *cannot* be simulated efficiently on a classical computer, so finding a polynomial simulation algorithm either a) is impossible, or b) would deal a crushing blow to a major area of current research in physics, math, and computer science.

There are three types of systems that we are interested simulating, which we call PSiQCoPATH's MO (modes of operation):

1. Modes of Operation

   i) Single pure state evolution: $|\,\psi_0\,\rangle \longrightarrow |\,\psi_f\,\rangle = \mathcal{T}\left[e^{-i/\hbar\int_{t_0}^{t_f}\hat{H}(t)\,dt}\right]|\,\psi_0\,\rangle$

   ii) Mixed state evolution: $\hat{\rho}_0 \longrightarrow \hat{\rho}_f = \hat{U}(t_f;\,t_0)\,\hat{\rho}_0\,\hat{U}^\dagger(t_f;\,t_0)$

   iii) General time evolution operator calculation: $\hat{U}(t_f;\,t_0) = \mathcal{T}\left[e^{-i/\hbar\int_{t_0}^{t_f}\hat{H}(t)\,dt}\right]$

In these equations, $\hat{H}(t)$ is the system's (time-dependent) Hamiltonian, and $\mathcal{T}$ denotes the time-ordered product. Within these MOs are two sub-categories, depending on whether we wish to simulate explicit Hamiltonian dynamics or a series of unitary quantum gate operations defined by a sequence of evolution operators $\hat{U}$ without specific reference to time. This second case would be interesting for the purpose of simulating quantum algorithms in the Quantum Gate Array (QGA) model of quantum computation.

For the the last two modes of operation, it is necessary to calculate the full time evolution operator of the quantum system. As discussed in the project proposal, the time evolution operator can be divided up into a product of intermediates:

$$\hat{U}(t_f;\,t_0) = \hat{U}(t_f;\,t_f-\Delta t)\hat{U}(t_f-\Delta t;\,t_f-2\Delta t)\cdots\hat{U}(t_0+\Delta t;\,t_0).$$

This product of intermediates, in turn, can be evaluated efficiently in parallel using the parallel prefix algorithm discussed in class.

For a pure state evolution where we are only interested in the evolution of a particular pure state, it is not necessary to calculate the full time evolution operator. Rather than do the N^3 operations to multiply the evolution matrices for successive time steps, one can turn the problem into a series of N^2 matrix-vector products. The trade-off, of course, is losing the rest of the information contained in the time evolution operator.

To accomplish this type of pure state evolution efficiently, we plan to write a row-distributed matrix-vector product implementation. What we lose in evolution information by performing pure state evolution we will gain in maximum problem size. The dimensionality grows as 2^n where n is the number of qubits in the system. Our maximum problem size will be determined by the size of matrix that we can fit into a single processor. By splitting the matrices over 16 processors, for example, we will be able to add two more qubits.

We also worked out the detailed physics of the calculation we are trying to perform. By using the time evolution operator approach described above, discretizing the time evolution into finite time steps *does not* itself introduce any inherent approximation. The only approximation we introduce in the simulation is in the calculation of each time step's evolution operator.

For a k-differentiable Hamiltonian, the associated time evolution operator (solution to Schrodinger's equation) will be $k+1$-differentiable. If we allow only smooth (infinitely differentiable) Hamiltonians, then we can expand the time evolution operator in a Taylor series to as many orders in Δt as we like. In this way, we derived an explicit expression for the single time step evolution operator up to third order in Δt (involving $\hat{H}$, $\frac{d}{dt}\hat{H}$, and $\frac{d^2}{dt^2}\hat{H}$).

Additionally, we decided on a suitable format for inputting/specifying the time dependent Hamiltonians. Taking experimental physics as a guide, we decided to allow users to specify Hamiltonians which are arbitrary time-dependent linear combinations of up to three pieces:

$$\hat{H}(t) = \alpha_1(t)\,\hat{H}_1 + \alpha_2(t)\,\hat{H}_2 + \alpha_3(t)\,\hat{H}_3 + \cdots$$

Physically, a Hamiltonian of this form would arise in an experiment where one has control over up to three "knobs" that allow him to tune some external fields that influence the particles in the system. We felt that three parameters would be sufficient to explore most types of interesting possibilities, including sweeping through or around a quantum critical point.

PSiQCoPATH is fed this information by explicitly supplying $\hat{H}_1$ to $\hat{H}_3$ in a text file, and a list of $\alpha_i(t)$, $\dot{\alpha}_i(t)$, $\ddot{\alpha}_i(t)$ at the desired time steps to be calculated. Note that this also gives us the freedom to have an adjustable time step throughout the calculation. The text file input format is a list with rows in the following form:

$$t_k,\ \alpha_1(t_k),\ \alpha_2(t_k),\ \alpha_3(t_k),\ \left.\frac{d\alpha_1}{dt}\right|_{t_k},\ \left.\frac{d\alpha_2}{dt}\right|_{t_k},\ \left.\frac{d\alpha_3}{dt}\right|_{t_k},\cdots,\ \left.\frac{d^{m-1}\alpha_1}{dt^{m-1}}\right|_{t_k},\ \left.\frac{d^{m-1}\alpha_2}{dt^{m-1}}\right|_{t_k},\ \left.\frac{d^{m-1}\alpha_3}{dt^{m-1}}\right|_{t_k}$$

We are currently in the progress of implementing the data read-in and optimizing the (complex) matrix/vector classes. Once this is complete, we will integrate these with our parallel prefix and row-distributed matrix-vector product codes.