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

Eric Fellheimer and Mark Rudner

March 27, 2005

1. Program Purpose — PSiQCoPATH is designed to compute the "exact" time evolution of a wide range of quantum systems with time-dependent Hamiltonians in a discrete time simulation to accuracy $\mathcal{O}\left[(\Delta t)^n\right]$

   A. 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: $\hat{U}(t_f;\,t_0) = \mathcal{T}\left[e^{-i/\hbar\int_{t_0}^{t_f}\hat{H}(t)\,dt}\right]$

   B. Results

      i) For pure state evolution, the result is a sequence of $N \times 1$ column vectors containing the coefficients of $|\,\psi(t)\,\rangle$ in the basis used in the input. Here, N is the dimensionality of the Hilbert space, which for a quantum computing system is typically 2^n. Runs of this type can use Eric's row-distributed matrix-vector algorithm.

      ii) For mixed state evolution or simply a time-evolution operator calculation, a sequence of matrices corresponding to the time evolution operators $\hat{U}(t;\,t_0)$ in the same basis as the input will be saved. Additionally, the final mixed state density matrix will be saved if an initial density matrix is supplied. Runs of these types use the matrix-matrix parallel prefix algorithm.

      iii) In all cases, output should be saved in a format that can be read back in as the starting point for a further calculation

   C. Applications

      i) Quantum Gate Array (QGA) Simulation: This corresponds to a run with a piecewise constant Hamiltonian. For such cases, a special mode should be available where the user can directly supply the sequence of unitary operators corresponding to the desired gates. For an individual pure state QGA simulation, the matrices should be read in directly in a row-distributed manner in preparation for use in Eric's algorithm.

      ii) Quantum Adiabatic Evolution (QAE): An initial and final Hamiltonian are specified with smooth interpolation in between. More details on this discussed in the input section below.

      iii) General Time-dependent Quantum Evolution: PSiQCoPATH can be used to study the dynamics of any quantum system with a finite-dimensional Hilbert space (e.g. spin chains, spin

lattices, etc). Of particular interest may be the behavior in the vicinity of quantum critical points.

2. Program Details

A. Supported Types of Evolution

i) Ideally, any arbitray time-dependent Hamiltonian would be supported. However, in practice this is very difficult to implement

ii) QGA simulations are the easiest, and are a straightforward application of the parallel prefix algorithm or Eric's algorithm as the case may be. All that is required is a way for the user to supply the desired gates in the form of a sequence of large unitary matrices

iii) Quite often, it is convenient to study QAE on systems where the Hamiltonian linearly scales from an initial Hamiltonian $\hat{H}_A$ to a final Hamiltonian $\hat{H}_P$ such that

$$\hat{H}(t) = \left(\frac{t}{T}\right)\hat{H}_A + \left(1 - \frac{t}{T}\right)\hat{H}_P.$$

If we feel this is a sufficiently common type of system, we may want to code such linear interpolation directly in as an option in the program.

iv) More generally, we may think of time-dependent evolution as arising in some system from some external agent adjusting "knobs" on an apparatus, which in turn changes some fields or other parameters of the system. In this model, the time dependent Hamiltonian of the system can be written as

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

where $\{\hat{H}_i\}$ are constant operators.

Allowing up to 3 adjustable parameters $\alpha_i(t),\ i \in \{1, 2, 3\}$ would probably be sufficient to take care of the needs of most things one would want to use this for. An example of when you might want more than 2 parameters would be if a quantum critical point exists in a system of interest for adiabatic quantum computation; by tuning extra parameters, one may be able to avoid this point where the energy gap to the first excited state vanishes.

B. The evolution operator for a single time step

i) The fundamental physics in PSiQCoPATH is contained in the calculation of the evolution operator for each discrete time step. For QGA simulations, this step is not performed by the program as the individual unitary steps are supplied directly. In all other circumstances, evolution is governed by the Schrödinger equation

$$i\frac{d}{dt}|\,\psi(t)\,\rangle = \hat{H}(t)|\,\psi(t)\,\rangle$$

in units units where $\hbar \equiv 1$. By writing

$$|\,\psi(t)\,\rangle = \hat{U}(t;\,t_0)|\,\psi(t_0)\,\rangle$$

we find that the time evolution operator $\hat{U}(t;\,t_0)$ also obeys the Schrödinger equation

$$i\frac{d}{dt}\hat{U}(t;\,t_0) = \hat{H}(t)\,\hat{U}(t;\,t_0).$$

Note that the derivative operates only on the first argument; the argument after the semicolon acts as a constant parameter.

For a finite time step Δt starting at time t_k, we wish to find

$$|\,\psi(t_k+\Delta t)\,\rangle = \hat{U}(t_k+\Delta t;\,t_k)|\,\psi(t_k)\,\rangle.$$

For a Hamiltonian with general nontrivial time dependence, we cannot find a closed form analytic solution to the Schrödinger equation. From the fact that as $\Delta t \to 0$ no evolution should occur, however, we do have that $\hat{U}(t_k+\Delta t;\,t_k) \to \mathbb{1}$ as $\Delta t \to 0$.

If $\hat{H}(t)$ is infinitely differentiable in time, then $\hat{U}(t)$ will also have analytic time dependence. For less well-behaved $\hat{H}$ continuously differentiable only $m-1$ times, $\hat{U}(t) \in C^m$. Thus the expansion is only valid up to m^{th} order in such cases. If we restrict ourselves to smooth time dependence of $\hat{H}$, then there is no problem. For situations in which one would like to simulate evolution under discontinuous changes of $\hat{H}$, PSiQCoPATH can be run multiple times in succession on the different pieces, and the results combined at the end without any loss of information or generatlity.
Assuming $\hat{H} \in \mathcal{C}^\infty$ and using the fact that $\hat{U}(t_k+\Delta t;\,t_k) \to \mathbb{1}$ as $\Delta t \to 0$, we can write $\hat{U}(t_k+\Delta t;\,t_k)$ as the Taylor series

$$\hat{U}(t_k+\Delta t;\,t_k) = \mathbb{1} + \left.\frac{d\hat{U}}{dt}\right|_{t_k}\Delta t + \frac{1}{2!}\left.\frac{d^2\hat{U}}{dt^2}\right|_{t_k}(\Delta t)^2 + \frac{1}{3!}\left.\frac{d^3\hat{U}}{dt^3}\right|_{t_k}(\Delta t)^3 + \cdots.$$

The only approximation made by PSiQCoPATH is the termination of this series to finitely many terms. In principle we can go to arbitrary order, but I feel like 3rd order should be high enough (and it will only get messier and messier to continue). For this reason, I compute here explicitly the terms of this series up to order $\mathcal{O}(\Delta t^3)$.

$\mathcal{O}(1):\ \mathbb{1}$

$\mathcal{O}(\Delta t):\ -i\hat{H}(t_k)$

This comes from a straight application of the Schrödinger equation $\frac{d\hat{U}}{dt} = -i\hat{H}\hat{U}$:

$$\left.\frac{d\hat{U}}{dt}\right|_{t_k} = -i\hat{H}(t_k)\hat{U}(t_k;\,t_k) = -i\hat{H}(t_k)$$

For our general form for a tunable Hamiltonian $\hat{H}(t) = \alpha_1(t)\hat{H}_1 + \cdots$ this gives

$$\left.\frac{d\hat{U}}{dt}\right|_{t_k} = -i\left[\alpha_1(t_k)\hat{H}_1 + \alpha_2(t_k)\hat{H}_2 + \alpha_3(t_k)\hat{H}_3\right].$$

In the simpler case of linear interpolation, $\alpha_1(t) = \frac{t}{T}$, $\alpha_2(t) = \left(1-\frac{t}{T}\right)$, and $\alpha_3 = 0$. Then the first order derivative term is

$$\left.\frac{d\hat{U}}{dt}\right|_{t_k} = -i\left[\left(\frac{t_k}{T}\right)\hat{H}_1 + \left(1-\frac{t_k}{T}\right)\hat{H}_2\right].$$

$\mathcal{O}(\Delta t^2):\ -\frac{1}{2}\hat{H}^2(t_k) - \frac{i}{2}\left.\frac{d\hat{H}}{dt}\right|_{t_k}$

Again this result is obtained by applying the Schrödinger equation twice:

$$\frac{d^2\hat{U}}{dt^2} = \frac{d}{dt}\left[-i\hat{H}(t)\hat{U}(t;\,t_k)\right].$$

Because both $\hat{H}(t)$ and $\hat{U}(t;\, t_k)$ depend on t, the product rule produces two terms on the RHS

$$\frac{d^2\hat{U}}{dt^2} = -i\frac{d\hat{H}}{dt}\hat{U} - i\hat{H}\frac{d\hat{U}}{dt}.$$

In the second term we recognize $\frac{d\hat{U}}{dt}$ as $-i\hat{H}\hat{U}$. Evaluating the terms at $t = t_k$ (i.e. $\Delta t = 0$) and putting in $\hat{U}(t_k;\, t_k) = \mathbb{1}$ gives

$$\left.\frac{d^2\hat{U}}{dt^2}\right|_{t_k} = -i\left.\frac{d\hat{H}}{dt}\right|_{t_k} - \hat{H}^2(t_k).$$

For the general 3-parameter form, this gives

$$\left.\frac{d^2\hat{U}}{dt^2}\right|_{t_k} = -i\left[\dot{\alpha}_1(t_k)\hat{H}_1 + \dot{\alpha}_2(t_k)\hat{H}_2 + \dot{\alpha}_3(t_k)\hat{H}_3\right] - \left[\alpha_1(t_k)\hat{H}_1 + \alpha_2(t_k)\hat{H}_2 + \alpha_3(t_k)\hat{H}_3\right]^2,$$

and for the case of linear interpolation we get

$$\left.\frac{d^2\hat{U}}{dt^2}\right|_{t_k} = -\frac{i}{T}\left(\hat{H}_1 - \hat{H}_2\right) - \left(\frac{t_k}{T}\right)^2 {\hat{H}_1}^2 - \frac{t_k}{T}\left(1 - \frac{t_k}{T}\right)\left(\hat{H}_1\hat{H}_2 + \hat{H}_2\hat{H}_1\right) - \left(1 - \frac{t_k}{T}\right)^2 {\hat{H}_2}^2$$

$$\mathcal{O}(\Delta t^3):\ -\tfrac{1}{3!}\left(2\left.\tfrac{d\hat{H}}{dt}\right|_{t_k}\hat{H}(t_k) + \hat{H}(t_k)\left.\tfrac{d\hat{H}}{dt}\right|_{t_k} - i\hat{H}^3(t_k) + i\left.\tfrac{d^2\hat{H}}{dt^2}\right|_{t_k}\right)$$

This was obtained by expanding

$$\frac{d^3\hat{U}}{dt^3} = -\frac{d}{dt}\left[\hat{H}(t)\,\hat{H}(t)\,\hat{U}(t;\, t_k)\right] - i\frac{d}{dt}\left[\frac{d\hat{H}}{dt}\hat{U}(t;\, t_k)\right]$$

using the chain rule to get

$$-\frac{d}{dt}\left[\hat{H}(t)\,\hat{H}(t)\,\hat{U}(t;\, t_k)\right] = -\frac{d\hat{H}}{dt}\hat{H}(t)\,\hat{U}(t;\, t_k) - \hat{H}(t)\,\frac{d\hat{H}}{dt}\,\hat{U}(t;\, t_k) + i\hat{H}^3(t)\,\hat{U}(t;\, t_k)$$

and

$$-i\frac{d}{dt}\left[\frac{d\hat{H}}{dt}\,\hat{U}(t;\, t_k)\right] = -i\frac{d^2\hat{H}}{dt^2}\,\hat{U}(t;\, t_k) - \frac{d\hat{H}}{dt}\hat{H}(t)\,\hat{U}(t;\, t_k).$$

For the 3-parameter type Hamiltonians, we just need to insert

$$\hat{H}(t_k) = \sum_i \alpha_i(t_k)\hat{H}_i,$$

$$\left.\frac{d\hat{H}}{dt}\right|_{t_k} = \sum_i \dot{\alpha}_i(t_k)\hat{H}_i,$$

and

$$\left.\frac{d^2\hat{H}}{dt^2}\right|_{t_k} = \sum_i \ddot{\alpha}_i(t_k)\hat{H}_i.$$

For the linear interpolation case, the $\left.\frac{d^2\hat{H}}{dt^2}\right|_{t_k}$ term drops out, and we only need to insert

$$\hat{H}(t_k) = \left(\frac{t_k}{T}\right)\hat{H}_1 + \left(1 - \frac{t_k}{T}\right)\hat{H}_2$$

and

$$\left.\frac{d\hat{H}}{dt}\right|_{t_k} = \frac{1}{T}\left(\hat{H}_1 - \hat{H}_2\right).$$

The third order expression is a little messy, but not too bad I guess... We should decide whether or not we want to code it up, and if there's any reason to carry the expansion to one more order. It seems pretty costly to keep calculating higher and higher order terms, and at some point it would probably be better to just decrease the time step than to go higher.

ii) Once the evolution operators are calculated for each indidual time step, the discretized evolution operator of the system is given by the series

$$\mathbb{1},\ \hat{U}(\Delta t;\, 0),\ \hat{U}(2\Delta t;\, \Delta t)\hat{U}(\Delta t;\, 0),\ \hat{U}(3\Delta t;\, 2\Delta t)\hat{U}(2\Delta t;\, \Delta t)\hat{U}(\Delta t;\, 0),\ \ldots$$

which can be calculated efficiently in parallel using the standard parallel prefix (scan) algorithm.

iii) Unitarity: By terminating the series for $\hat{U}(t_k + \Delta t;\, t_k)$ after finitely many terms, the result will be a non-unitary evolution operator. However, if the simulation parameters (Δt, order to which $\hat{U}$ is calculated) are good enough, then this non-unitarity will be small. Put another way, the closeness of the final result to unitarity gives us a way to track the convergence of the simulation, and to determine if we need to decrease the time step/go to higher order. Two quantitative ways to check this are through $\|\hat{U}^\dagger\hat{U} - \mathbb{1}\|$ and by looking at the sum of the squares of each individual row or column (these are the diagonal elements of $\hat{U}^\dagger\hat{U}$).

iv) Code-wise my idea is that each $\hat{U}$ should be calculated on a single processor (the same one on which it will need to be for the parallel prefix operation). This is the most logical for the matrix-matrix type of run, but I'm not sure what the best way is for the matrix-vector method. Somehow all of these $\hat{U}$'s need to be calculated and then distributed row-wise. My guess is that it's still best to generate them in the same way (each processor generating $T/p\Delta t$ of them) and then to redistribute the data, but I've only just though about it in the last 30 seconds.

v) I'm a little worried about the matrix-vector algorithm right now. I'm not sure what the slowest step will be here. Calculating the $\hat{U}$'s still involves matrix multiplications, which are $\mathcal{O}(N^3)$ operations. Thus even though the matrix-vector approach speeds up the prefixing operation, the benefit may not be as large as we had hoped. On the other hand, it will still have speedup over the fully matrix-matrix approach for the cases where it applies, so for that reason it's probably good to proceed with it.

vi) If we hard-code in linear interpolation, it's easy to just code up what was derived above. For the general 3-parameter Hamiltonians my thought is that the user can specify $\hat{H}_1$, $\hat{H}_2$, $\hat{H}_3$, and an $\mathcal{N} \times (1 + 3m)$ matrix of values, the k^{th} row of which contains

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

where $\mathcal{N}$ is the number of time steps to be used, t_k is the total time elapsed up to the k^{th} time step, m is the order to which the exponential is to be calculated at each time step, and the derivates $\frac{d\alpha_i}{dt}$ up to $\frac{d^{m-1}\alpha_i}{dt^{m-1}}$ are included in the row.

Note that this method allows for arbitrarily varying time steps throughout the run, and any smooth functional dependence one can dream up for the three parameters. It should be simple

to get Matlab to output an array of values in this format for whatever we feel like throwing at it.

C. Implementation Issues

i) Matrix Storage and Operations: We still need to do some more research here. My friend at Los Alamos said that if we're going to use BLAS we should check out the ATLAS implementation which is some sort of self-optimizing thing. I finally read about Strassen's algorithm yesterday. The issue there seems to be memory, since it might really rack up the usage with all the recursive calls. When working with $2^n \times 2^n$ matrices, this could become a problem as we try to scale up n.

On the other hand, we can't reap the benefit of BLAS-3 for matrices larger than what will fit in the fast cache memory of whatever machine we're running on. The optimal thing to do may be to block the matrices using Strassen down to the size where they can fit in fast memory, then terminate the recursion by doing a BLAS-3 style matrix multiplication instead of going all the way down to 1×1. You probably have a lot better idea about these types of things than me, so I'll be interested to see your take on this.

ii) Input Format