High performance computing and quantum trajectory method in CPU and GPU systems

Joanna Wiśniewska¹, Marek Sawerwain², Wiesław Leoński³

¹ Institute of Information Systems, Faculty of Cybernetics, Military University of Technology, Kaliskiego 2, 00-908 Warsaw, Poland
² Institute of Control and Computation Engineering, University of Zielona Góra, Lichealna 9, 65-417 Zielona Góra, Poland
³ Quantum Optics and Engineering Division, Institute of Physics, University of Zielona Góra, Lichealna 9, Zielona Góra 65-417, Poland

E-mail: jwisniewska@wat.edu.pl

Abstract. Nowadays, a dynamic progress in computational techniques allows for development of various methods, which offer significant speed-up of computations, especially those related to the problems of quantum optics and quantum computing. In this work, we propose computational solutions which re-implement the quantum trajectory method (QTM) algorithm in modern parallel computation environments in which multi-core CPUs and modern many-core GPUs can be used. In consequence, new computational routines are developed in more effective way than those applied in other commonly used packages, such as Quantum Optics Toolbox (QOT) for Matlab or QuTIP for Python.

1. Introduction

The Quantum Trajectory Method (QTM) is still very popular method applied for investigation in the field of open quantum systems. However, its application requires a special attention in its reimplementation in modern hardware systems, especially those built with use of GPUs and modern multi-core CPUs.

The proposed implementation of QTM is also a hybrid method, in which power of traditional CPU and modern GPU is combined to obtain more effective routines applicable in simulations of open quantum systems’ dynamics. It should be pointed out that discussed here solution which is based on GPU, is several times faster than other implementations of QTM prepared for traditional serial CPU systems. The proposed implementation allows for considerable reduction of calculation time and this speed-up depends on numerical method chosen for solving ordinary differential equations (ODE) appearing in considered problem. We will shortly discuss the fourth order Runge-Kutta method (RK4) and the fourth order backward differentiation formula (BDF4) which can be simultaneously applied in proposed method.

2. Quantum Trajectory Method

To describe the system’s evolution with use of QTM we need an effective Hamiltonian \( H_{\text{eff}} \) defined with use of collapse operators \( C_n \) and system’s Hamiltonian \( H_{\text{eff}} = H_{\text{sys}} - \frac{i\hbar}{2} \sum_n C_n^* C_n \).
The operator $C_n$ allows to determine the probability of appearance of so-called quantum jump. Such jump is modelled by the collapse operator acting on a current quantum state of the system, whereas the probability can be defined as: 

$$\delta p = \delta t \sum_{n} \langle \psi(t)|C_n^\dagger C_n|\psi(t) \rangle.$$  

If such quantum jump occurs, the system’s state for the moment of time $(t + \delta t)$ just after the collapse operation can be written as:

$$|\psi(t + \delta t)\rangle = C_n|\psi(t)\rangle/\sqrt{\langle \psi(t)|C_n^\dagger C_n|\psi(t) \rangle}.$$  

Moreover, if more than one collapse operator are applied in considered model, the probability of using $i$-th operator can be written as:

$$P_i(t) = \langle \psi(t)|C_i^\dagger C_i|\psi(t) \rangle/\delta p.$$  

To ensure the probabilistic choice of adequate application of collapse operator in the process of simulation, a proper random numbers generator should be applied. Moreover all considered calculations correspond to operations performed on matrices and vectors. Applied matrices are usually band matrices, so they may be treated as sparse matrices to economize memory usage and to speed-up the calculations. In consequence, the compressed sparse row format will be used because of many matrix-vector multiplication – it also gives an additional speed-up.

The above essential remarks constitute the idea of a following algorithm for a single quantum trajectory simulation which can be presented in a form of four computational steps:

(A) a random value $r \in (0, 1)$ is computed, where $r$ denotes the probability of quantum jump occurrence,

(B) the Schrödinger equation is integrated with application of the Hamiltonian $H_{\text{eff}}$ to get state’s vector at for the time $t$ in such way to make the state’s vector norm equal or greater to $r$: 

$$\langle \psi(t)|\psi(t) \rangle \geq r,$$

(C) a quantum jump occurrence causes the system’s state projection at the moment $t$, to one of the states given by the equation (1). The operator $C_n$ is selected to meet the following relation for adequate $n$: 

$$\sum_{i=1}^{n} P_i(t) \geq r$$

and $P_i(t)$ is specified by the equation (2),

(D) the projected state of obtained wave-function is a new initial value corresponding to the moment of time $t$; next, the new value of $r$ is randomly selected and then, the procedure repeats the process of quantum trajectory simulation starting from the step (B) – more precisely: the simulation is performed again but starts from previously given value of $t$.

The presented approach refers to solutions described in the other publications [1, 2].

3. Some implementation notes and performance of numerical experiments

To implement QTM, the procedures defining pseudo-random numbers generator (PRNG) and some methods for solving ODEs are required. The presented solution uses the PRNG procedure...
defined in cuRand package [5] which is a part of CUDA Toolkit. Solving of ODEs will be realized by numerical methods for solving systems of differential equations – two methods will be used: RK4 and BDF4. Both methods are implemented in two versions – for CPUs and GPUs systems.

After performing a single QTM algorithm only one trajectory is obtained which is not sufficient to describe the state of quantum system. The QTM is a Monte Carlo method, so it necessary to generate more trajectories. Such trajectories have to be averaged to obtain the final trajectory which describes the dynamics of analysed system. Since the trajectories can be calculated without any correlations to one another with use of CPUs and GPUs, as well, this tactics leads to increasing the accuracy of calculations and reduction of computation time.

To ensure the high efficiency of numerical procedures, C++ and CUDA C++ programming languages were applied. Moreover, to simplify the use of proposed toolbox, special Python package was prepared. As an example, below we present a fragment of a script which allows for simulation of quantum system’s dynamics with use of QTM (e.g. for 100 trajectories):

```python
import qcs
import matplotlib

N0=N1=N2=6
A0=qcs.TensorProduct(qcs.DestroyOperator(N0),qcs.Eye(N1),qcs.Eye(N2))
...
T = qcs.linspace(0, 10, 100) ; Psi0 = # initial state
H = 1.0j*(A0*a1.Adj()*A2.Adj()-A0.Adj()*A1*A2)
EO = # expectation operator
CO1 = # collapse operator 1 ; CO2 = # collapse operator 2
expectdata = qcs.QuantumTrj(Psi0, T, "const", H, 100, EO, [CO1, CO2])
matplotlib.plot(T, expectdata)
```

Figure 2. Generating and averaging trajectories with use of CPUs and GPUs. Moreover, using GPUs for averaging trajectories reduces the time of calculations.

To compare the efficiency of proposed procedures with those already implemented in QuTIP package, we have performed simulations of time-evolution of optical parametric amplifier, described by the following trilinear Hamiltonian [4]:

\[
\dot{H} = K(\hat{a}\hat{b}^\dagger\hat{c}^\dagger - \hat{a}^\dagger\hat{b}\hat{c}) \tag{3}
\]

The operators \( \hat{a} \), \( \hat{b} \) and \( \hat{c} \) are usual boson annihilation operators and correspond to the pump, signal and idler fields, respectively, whereas \( K \) denotes coupling constant. To perform simulations we assume that \( K = 1 \) and the initial state of considered system is \( |\psi_0\rangle = |\alpha\rangle_a|0\rangle_b|0\rangle_c \) (coherent state for the pump mode (a) and vacuum states for signal (b) and idler (c) modes).

Thus, Tab.(1) shows the computation times for simulations performed with use of QuTIP package, and our procedures using RK4 and BDF4 methods, for \( (6 \times 6 \times 6) \)-dimensional initial state. We see that if we use RK4 method, our procedures are \(~10 \text{ times faster}~\) than those implemented in QuTIP. For BDF4 method the speed-up is naturally smaller, but our package is still \(~3.5 \text{ times faster}~\) than QuTIP. Tab.(1) also shows the calculations’ efficiency for time-dependent Hamiltonian:

\[
H = H_0 - f(t) \cdot H_1 \quad \text{where} \quad f(t) = \alpha \cdot \exp\left(-\left(\frac{t}{\sigma}\right)^2\right) \tag{4}
\]
Table 1. The duration of simulations (in seconds) for exemplary Hamiltonians. The data refer to simulations with use of RK4 and BDF4 methods implemented in our toolbox (parallel CUDA technology NVIDIA graphic adapter GeForce 680 was used). Results are compared to those obtained with use of QuTIP package (computing system with Intel Core i7 950 3.00 Ghz with only one core was used – using two cores for given number of trajectories will shorten the simulation’s time twice).

| No. of trj. | QuTIP     | GPU 680 RK4 | GPU BDF4 680 |
|-------------|-----------|-------------|--------------|
|             | Timings for Hamiltonian given by eq. (3) |             |              |
| 100         | 5.57      | 1.10        | 5.24         |
| 200         | 10.99     | 2.32        | 7.13         |
| 500         | 27.56     | 3.20        | 9.48         |
|             | Timings for Hamiltonian given by eq. (4) |             |              |
| 100         | 12.33     | 1.42        | 7.91         |
| 200         | 37.02     | 3.02        | 11.10        |
| 500         | 60.08     | 3.85        | 13.22        |

where \( H_0 \) represents a constant part of the Hamiltonian and \( H_1 \) is time-dependent. The Hamiltonian \( H \) is defined with use of function \( f(t) \) where \( \alpha \) and \( \sigma \) are scalar values. In the presented solution the time-dependent Hamiltonian is calculated before the trajectories are computed for the points given by time variable – such approach speeds-up the calculations of trajectories (the missing Hamiltonians are computed during the simulation and built-in mechanism of caching of the Hamiltonians avoids recalculating previously obtained \( H \)). The duration of simulations, showed in Tab.(1), convict that application of GPUs and multi-core systems allows calculating more trajectories in shorter time. Furthermore, increasing the number of trajectories do not results in significant increase of computations’ time as it was for the case of CPU systems. It should also be noticed that when time-dependent Hamiltonian was applied, the time of simulation with use of QuTIP package last about 50% longer.

4. Conclusions
We presented solution that allows to reduce the time of calculations as compared with other known toolkits. One should remember that discussed exemplary problem was represented by the Hamiltonian described by matrices with leading dimension \( 6 \times 6 \times 6 \). For simpler unitary Hamiltonians which matrix representations are sized: \( 5 \times 5 \), \( 15 \times 15 \), GPU based computations can be performed faster than in CPU systems: twenty, fifty times, respectively. In cases of bigger quantum systems such speed-up will be three or even five times greater. Moreover, additional integration of presented solution with Python environment allows for more efficient application of our routines to simulate QTM, e.g. we can combine these routines with QuTIP package or use them in Matlab environment together with QOT package.

References
[1] J R Johansson P D Nation F Nori 2013 Comp. Phys. Comm. 184(4) 234
[2] B M Garraway P L Knight 1994 Phys. Rev. A 50, 25482563
[3] Sze M Tan 1999 J. Opt. B: Quantum Semiclass. Opt. 1 424
[4] F Dell’Anno S De Siena F. Illuminati 2006 Phys. Rep. 428 53
[5] NVIDIA, 2014, CURAND Toolkit Documentation. http://docs.nvidia.com/cuda/curand/index.html