Increasing Performance of the Quantum Trajectory Method by Grouping Trajectories

Alexey Liniov[1], Valentin Volokitin[1], Iosif Meyerov[1], Mikhail Ivanchenko[1], and Sergey Denisov[1,2]

[1] Lobachevsky State University of Nizhni Novgorod, Nizhny Novgorod, Russia
[2] Institut für Physik, Universität Augsburg, D-86135 Augsburg, Germany
alin@unn.ru✉, valyav95@mail.ru, meerov@vmk.unn.ru,
ivanchenko.mv@gmail.com, sergey.denisov@physik.uni-augsburg.de

Abstract. The quantum trajectory method is the most popular and widely used algorithm to simulate the evolution of an open N-dimensional quantum system. The key idea is to unravel Markovian equation describing evolution of the system density operator (a $N \times N$ Hermitian matrix) into a set of independent stochastic realizations obtained by propagating system wave function (a complex N vector). Since the method decreases the scaling of the computational problem from N^2 to N, it is especially efficient for the systems of large dimensions. Intrinsic parallelism that is characteristic to all Monte Carlo schemes allows for efficient implementations of quantum trajectories on a high-performance computational cluster. One of the core mathematical operations involved into the method is the matrix-vector multiplication. We propose to improve the algorithm by grouping trajectories into matrices and substituting a set of matrix-vector multiplications with a single matrix-matrix multiplication. By using a testbed model with 1024 states, we demonstrate that, even in the presence of intrinsic asynchrony between different trajectories, this step leads to a 17-fold acceleration on the 4-socket 96-core Intel Broadwell CPU.

Keywords: Open quantum systems · Quantum trajectory method · High-performance computing · Supercomputing technologies · Parallel computing · Performance analysis and optimization

1 Introduction

Physics of open quantum systems attracts a lot of attention during the last decade. This is because it considers quantum systems in their natural habitats, i.e., when the former interact with their environments [1]. The growing interest to open systems was initiated by the rise of quantum technologies and is maintained by ever-increasing number of real-life applications of quantum systems that a decade ago existed on paper only. It is evident that in order to blueprint a realistic quantum device, effects of its interaction with environment should be taken into account.

The most elaborated, both from mathematical and physical point views, approach to model open quantum system is the Lindblad formalism, which is based on the idea of quantum dynamical semi-groups and culminates into the Lindblad equation [2]. This approach is very popular in such fields as quantum optics, optomechanics, cavity quantum electrodynamics, and cold atom physics [3]. Straightforward numerical solution of the Lindblad equation (and thus obtaining the asymptotic state of the model of interest) is not feasible when the model dimension N – that is the dimension of the Hilbert space the model lives in – is larger than 500. When the model Hamiltonian is explicitly time-periodic, i.e., the system is additionally modulated in time [4], evaluation of the system non-equilibrium asymptotic state involves numerical integration of the Lindblad equation in time. It is hardly doable even when $N \backsimeq 400$.

Model with $N = 400$ states may still be too small to describe real-life quantum systems. It is possible to go beyond this limit by unraveling the Lindblad equation into a set of stochastic realizations, called "quantum trajectories" [3,5]. This method allows transform the problem of the numerical solution of the Lindblad equation into a task of statistical sampling over quantum trajectories, with every trajectory specified by a complex vector of the size N. The price to pay for the reduction from N^2 to N is that one now has to sample over many realizations.

In our work [6] we presented an implementation of the quantum trajectory method that allowed us to resolve non-equilibrium asymptotic states (which we called "quantum attractors') of a periodically modulated quantum model. We demonstrated that a regular high-performance cluster (with up to 512 computational cores) is enough to sample such attractors with high accuracy for the model of the dimension $N \approx 2000$. The aim of this paper is to investigate the potential for the further optimization of the implementation and improvement of its performance.

Like in a number of other numerical software, a substantial advance can be potentially reached by increasing parallelism on each level of computing (processes, threads, SIMD, instruction level parallelism), as well as by improving memory usage efficiency. Since the quantum trajectory method belongs to the Monte Carlo family, it should possess high intrinsic parallelism. Note that due to the nature of the method, intrinsic stochastic steps – quantum jumps – can occur at different random times for different trajectories, and their number (for a fixed time interval) can vary too. Nevertheless, numerical experiments indicate the absence of a substantial variation, which together with the opportunity of merging a set of trajectories into a single computational task for parallel computing, leads to a small imbalance of computational load, order 5% only. At the same time, an empiric choice of the ratio between the number of the employed processes and threads of the hybrid MPI + OpenMP parallelization scheme, on the contrary, significantly affects the total computation time. The next level of parallelism is related with vectorization of computing and effective usage of wide vector registries and AVX2-instructions in modern processors. It is essential that the main computational core of the method that consumes $> 95\%$ of the total time is the dense matrix-vector multiplication, which can be vectorized.

In that respect, our code makes use of the high performance implementation of the matrix-vector multiplication from the Intel Math Kernel Library (MKL). Summarizing the above, there is a considerable room for exploiting parallelism in quantum trajectory method.

Improving performance of computations here could be achieved by reducing the number of calls to the main memory and by a more efficient usage of the cache. The most straightforward idea along these lines would be a substitution of the groups of matrix-vector multiplications to single matrix-matrix multiplications. In [7] it was demonstrated, that such optimization can substantially reduce computational cost, at least for the addressed class of problems. There, merging was achieved by substituting propagation of separate vectors with propagation of a matrix composed of them. In the case of quantum trajectories, there is an asynchrony between different trajectories in times of jumps and such merging is not so straightforward. Here we propose a solution to the problem that allows to attain the same results but at a smaller time, as a rule. We will demonstrate the way to organize computing, focusing on a matrix multiplication, where appropriate.

The paper is organized as follows. In section 2 we give a mathematical model – a system of indistinguishable interacting bosons hopping between the sites of a periodically rocked dimer. In section 3 the description of the quantum trajectory method is given. In section 4 we present the optimized method. Numerical results and performance analysis are given in section 5. Section 6 concludes the paper.

2 Model

The Lindblad equation is described by its generator $\mathcal{L}$, which has a universal structure [2]:

$$\dot{\varrho} = \mathcal{L}(\varrho) = -i[H(t), \varrho] + \sum_{k=1}^{K} \gamma_k(t) \cdot \mathcal{D}_k(\varrho),$$
$$\mathcal{D}_k(\varrho) = A_k \varrho V_k^\dagger - \frac{1}{2}\{A_k^\dagger A_k, \varrho\}. \tag{1}$$

Here ϱ is the system density matrix, while the set of quantum jump operators, A_k, $k = 1, ..., K$, captures the action of the environment on the system. Namely, it acts through K 'channels' with time-dependent (in general) rates γ_k. Finally, $[.,.]$ and $\{.,.\}$ denote commutator and anti-commutator, respectively.

As a testbed model we use a system of $N-1$ indistinguishable interacting bosons hopping over a periodically rocked dimer [8]. The system Hamiltonian is

$$H(t) = -J\left(b_1^\dagger b_2 + b_2^\dagger b_1\right) + \frac{U}{2(N-1)} \sum_{g=1,2} n_g\,(n_g - 1)$$
$$+ \varepsilon(t)\,(n_2 - n_1) \tag{2}$$

where J is the tunneling amplitude, U is the interaction strength, and $\varepsilon(t)$ presents the modulation of the on-site potential difference. In particular, we

choose $\varepsilon(t) = \varepsilon(t+T) = \mu_0 + \mu_1\theta(\omega t)$, where μ_0 and μ_1 are static and dynamic energy offsets between the two sites, respectively, and $\theta(x)$ is a step-like periodic function of period one. Here, b_g and $b_g^\dagger$ are the annihilation and creation operators on site $g \in \{1, 2\}$, while $n_g = b_g^\dagger b_g$ is the particle number operator. The system Hilbert space has dimension N and can be spanned with the Fock basis vectors, labeled by the number of boson on the first site t, $\{|t+1\rangle\}$, $t = 0, ..., N-1$. So, the model has N states and its size is controlled by the total number of bosons. Hamiltonian (2) has been used for theoretical studies and was already implemented in experiments [8]. On top, this is a nicely scalable model; its dimension N can be incremented by simply adding one boson.

We use a single jump operator [9,10],

$$A = (b_1^\dagger + b_2^\dagger)(b_1 - b_2), \tag{3}$$

which tries to 'synchronize' the dynamics on the sites by constantly recycling anti-symmetric out-phase modes into the symmetric in-phase ones. The coupling constant $\gamma = (N-1)\gamma_0$ is assumed to be time-independent.

3 Quantum Trajectory Method

3.1 Base Algorithm

Solution to the Lindblad equation (1) for the density matrix of an open system can be unraveled into an ensemble of quantum trajectories, which are governed by the equation

$$|\dot{\psi}\rangle = -i\tilde{H}|\psi\rangle, \tag{4}$$

where $|\psi\rangle$ is the state vector of dimension N, and $\tilde{H} = H - \frac{i}{2}\sum_k \gamma_k A_k^* A_k$ is the non-Hermitian Hamiltonian, constructed from the original system Hamiltonian and jump operators.

The method is implemented as follows (see Algorithm 1). Initially, the code loads the model and method parameters: system size N, number of quantum trajectories for sampling L, end time T_{max}, the matrices for exponential operators *expM* (lines 12). Initial conditions $|\psi_0\rangle$ are chosen such that the norm of the vector equals 1; this corresponds to the initial condition $\varrho_0 = |\psi_0\rangle\langle\psi_0|$ for the original Lindblad equation (1). The main computational cycle (lines 312) contains numerical propagation of L vectors $\{|\psi_l(t)\rangle; l = 1, .., L\}$ in time within the interval $[0; T_{max}]$.

The details of propagation follow (lines 411).

1. Choose a random number η from the uniform distribution on $[0, 1]$ (line 5).
2. Perform propagation in time τ (lines 69), until the following is satisfied: $\||\psi(t)\rangle\|^2 = \eta$. Reaching it is ensured by the special form of the Hamiltonian $\tilde{H}$ that monotonously decreases the norm.
3. Make quantum jump (line 10).

   3.1. Normalize the state vector again $|\psi(t)\rangle = \frac{|\psi(t)\rangle}{\||\psi(t)\rangle\|}$.

Algorithm 1 : Quantum trajectory method

```
1: load the system and method parameters (N, L, T_max, ExpM)
2: for l = 1 to L do
3:     for t = 0 to T_max do
4:         generate η = U[0, 1]
5:         while |||ψ(t)⟩||^2 > η do
6:             calculate t_next
7:             propagate |ψ(t)⟩ on [t_cur, t_next]
8:         end while
9:         makes quantum jump
10:    end for
11: end for
12: calculate the final density matrix
13: release memory
```

3.2. Calculate probabilities of selecting quantum jump channels $[p_1, .., p_K]$, $p_k = \frac{\gamma_k \|A_k|\psi(t)\rangle\|^2}{\sum_{i=1}^{K} \gamma_i \|A_k|\psi(t)\rangle\|^2}$. Therefore, we split a unit interval in K parts of the lengths $p_1, .., p_K$, respectively.

3.3. Choose random ξ from a uniform distribution on $[0, 1]$. Determine the corresponding m-th quantum channel such that $\xi \in p_m$, and complete the quantum jump according to: $|\psi(t)\rangle = \frac{A_m|\psi(t)\rangle}{\|A_m|\psi(t)\rangle\|}$.

As a result, we obtain an ensemble of quantum trajectories $\{|\psi_l(t)\rangle; t \in [0, T_{max}]; l = 1, .., L\}$. The density matrix, approximating an exact solution to Eq.(1) at an arbitrary time $t \in [0; T_{max}]$, can be unraveled by averaging over the trajectories:

$$\tilde{\rho}_L(t) = \frac{1}{L} \sum_{l=1}^{L} \frac{|\psi_l(t)\rangle\langle\psi_l(t)|}{\||\psi_l(t)\rangle\|^2} \tag{5}$$

It is proved that $\lim_{L\to\infty} \tilde{\rho}_L(t) = \rho(t)$ [2].

3.2 Exponential Operators

The most computationally intensive part of the described algorithm is propagating a vector $|\psi(t)\rangle$ until the condition of a jump is met (line 8). This step can be substantially accelerated, taking into account that $\tilde{H}$ is constant between switching of $\theta(t)$. There, propagation is explicitly described by an exponential operator [6]:

$$B(\Delta t) = e^{-i\tilde{H}\Delta t},$$

which does not depend on a particular state $|\psi(t)\rangle$, and therefore allows for calculating evolution of an arbitrary vector over time Δt. To implement a high-precision approach to resoltion of the jump moments, one can pre-calculate a set of exponential operators, for different time steps. The maximal time step should

be chosen to be in an integer ratio to the time interval T, where the Hamiltonian $\tilde{H}$ is constant: $T = k\Delta t_1$, $k \in Z$. For finer timescales one defines:

$$\{B(\Delta t_i)|\Delta t_i = 2\Delta t_{i+1} : i = \overline{1, m-1}\}$$

The particular value of Δt_1 should be chosen as to decrease the computation time between quantum jumps, and so it is model and parameter dependent. In our experience, a "rule of thumb" defines the best value of Δt_1 as that makes $\||\psi(t)\rangle\|$ decreasing by $30-60\%$ under action of $B(\Delta t_1)$. The depth of timescale hierarchy, m, should be chosen with regard to the required precision for the resolution of jump moments, which is $\leq \Delta t_m$.

Propagation of the trajectory untill the next jump implements bisection method (Algorithm 2). It is initialized with the a current time t, state vector $|\psi(t)\rangle$ and random η. The first step takes $s = 1$, $\delta t = \Delta t_1$ (line 1), next steps take values of s and δt, obtained in the end of the preceding step. The moment of jump is found as follows:

1. The main cycle of the algorithm (lines 2-13) implements propagation with a given time step until quantum jump conditions are fulfilled, $\||\psi(t)\rangle\|^2 \leq \eta$.
2. The value of the state vector at $t + \delta t$ is calculated (line 3).
3. If $|\psi(t+\delta t)\rangle$ fulfills the jump condition, the time of the jump is rsolved with higher precision. This is achieved by taking a smaller time step (line 5), if the minimal one, t_m, has not been reached yet. Otherwise, the time of the jump is determined with the maximal possible precision.
4. If the jump condition is not fulfilled, or its moment is found with maximal precision, then the current time and state vector values are renewed (lines 7-8). Then, the maximal time step, Δt_s, to be used for the next iteration of the algorithm (lines 9-11), is chosen such that the time to the next switch of the Hamiltonian is the multiple of Δt_s.

4 Optimized Algorithm

Algorithm 2 that propagates the state vector $|\psi(t)\rangle$ to the next time moment $t + \delta t$, involves the multiplication of the vector with the matrix of the exponential operator. Our idea of accelerating the algorithm is to cluster (group) the trajectories so that multiple independent matrix-vector multiplications are substituted with a single matrix-matrix multiplication, which, for example, for matrix dimension of the order of 10^3 reduces computational time by several-fold. The main challenge lies in clustering of the vectors in groups; that is because moments of next quantum jumps for the trajectories are different and independent. It should also be noted that while Hamiltonian matrices for quantum dynamics are often sparse, the matrices for exponential operators are not.

Below we present the detailed description of our solution. The task is formulated as the propagation of the group of vectors $|\psi_l(t)\rangle, l = 1, .., L'$ over the time interval $[t_{\tilde{H}}, t_{\tilde{H}} + T_{\tilde{H}}]$, where the Hamiltonian $\tilde{H}$ is constant. As a result, all vectors have to be propagated to the end of the specified time interval.

Algorithm 2 : Finding the moment of a quantum jump with the set of exponential operators

$\delta t = \Delta t_1$
$s = 1$
while $\||\psi(t)\rangle\|^2 > \eta$ **do**
  $|\mu\rangle = B(t_s)|\psi(t)\rangle$
  if $\||\mu\rangle\|^2 \leq \eta$ & $s < m$ **then**
    $s = s + 1$
  else
    $|\psi(t)\rangle = |\mu\rangle$
    $t = t + \delta t$
    while $s > 1$ & $\delta t = k * \Delta t_{s-1}, k \in \mathbb{Z}^+$ **do**
      $s = s - 1$
      $\delta t = \Delta t_s$
    end while
  end if
end while

For simplicity of description and without loss of generality, we consider the set of operators $\{B(\Delta t_i)|k\Delta t_1 = T_{\tilde{H}}, k \in Z; \Delta t_i = 2\Delta t_{i+1} : i = \overline{1, m-1}\}$. The group of vectors is stored in the matrix form $V \in C^{L' \times N}$, where each vector is given by a separate row. Here L' is the number of vectors in a group, N is the dimension (number of states) of the model quantum system, $V[l]$ is the l-th state vector, and $\|V[l]\|$ is its norm.

Each l-th vector is given additional characteristics.

- $\eta[l]$, a random number from a uniform distribution, $U[0, 1]$, which determines the value of the norm, when a quantum jump occurs.
- $d[l]$ is an integer number that sets a current time within the propagation interval, when Hamiltonian remains constant, $[t_{\tilde{H}}, t_{\tilde{H}} + T_{\tilde{H}}]$. $d[l] = 0$ corresponds to the beginning of the interval. Increasing $d[l]$ by 1 corresponds to increasing current time by Δt_m. $d[l] = k * 2^{m-1}$ corresponds to the end of the time interval, $t_{\tilde{H}} + T_{\tilde{H}}$.

Matrix V has a specific structure and contains 4 blocks of rows of distinct classes. In course of the run, vectors are moved from one block to another (change their class), and the sizes of blocks change.

Start of the propagation from the left boundary of the interval $[t_{\tilde{H}}, t_{\tilde{H}} + T_{\tilde{H}}]$ is accompanied by the following initialization.

- All vectors belong to class A.
- If $t_{\tilde{H}} = 0$, then for all vectors the values $\eta[l] = U[0, 1]$ are calculated. Otherwise $\eta[l]$ is brought forward from the previous time interval.
- For all vectors $d[l] = 0$ is set.

Propagation is performed in the forward step (Algorithm 3) and backward step (Algorithm 4). The forward step is repeated until there remain trajectories from VA class. Otherwise, the backward step is performed once.

Table 1. The structure of the matrix

Class/Block	**Description**
VA, rows $(1)-(A)$	Vectors, which could experience a jump until the right boundary of the interval is reached $[t_{\tilde{H}}, t_{\tilde{H}}+T_{\tilde{H}}]$, but an estimate of the time of jump is missing.
VB, rows $(A+1)-(A+B)$	Vectors, which will experience a jump during the time interval Δt_1.
VC, rows $(A+B+1)-$ $(A+B+C)$	Vectors, which cannot experience a jump before the right boundary of the time interval is reached $[t_{\tilde{H}}, t_{\tilde{H}}+T_{\tilde{H}}]$.
VD, rows $(A+B+C+1)-$ $(A+B+C+D=L')$	Vectors, propagated to the right boundary of the time interval $[t_{\tilde{H}}, t_{\tilde{H}}+T_{\tilde{H}}]$.

The forward step of the algorithm finds the next moment for the quantum jump for all vectors of class A. It is specified in Algorithm 3.

1. The first part of the algorithm (lines 1-17) propagates each vector VA by a maximally possible number of steps Δt_1, which does not lead to the quantum jump yet. For that a vector is sequentially multiplied by B_1, but before it is renewed, one of the following is checked.
   1.1. Next product produces the quantum jump condition. Then, the vector is moved to block VB (lines 4-6).
   1.2. Next product does not lead to quantum jump and will not allow the current time to reach the right boundary of the interval $[t_{\tilde{H}}, t_{\tilde{H}}+T_{\tilde{H}}]$. Then, the value of the vector should be renewed, and propagation continued (rows 7-9).
   1.3. Next product does not lead to quantum jumps condition, but the current time becomes equal to the right boundary of the interval $[t_{\tilde{H}}, t_{\tilde{H}}+T_{\tilde{H}}]$. In that case, the vector should be renewed and moved to block VD (lines 10-12).
   1.4. Next product does not lead to quantum jump, but the current time becomes greater then the right boundary of the interval $[t_{\tilde{H}}, t_{\tilde{H}}+T_{\tilde{H}}]$. Then, the vector should be moved to block VC (rows 13-15).

   On completion of the first part of the algorithm block A becomes empty.
2. The second part of the algorithm (rows 18-34) searches for the quantum jump times for vectors from block VB, provided that it occurs within $[t_{\tilde{H}}, t_{\tilde{H}}+T_{\tilde{H}}]$. Namely, each vector from VB is multiplied by the matrices of the exponential operators, B_i, $i = 2..m$, one by one. Processing the result $VBnext[l]$ of the multiplication of each vector by each matrix is determined by the following.
   2.1. If a quantum jump occurs for $VBnext[l]$, the result is not saved (lines 21-23). In this case, we can either find a moment for the quantum jump to a higher precision or make the last move towards time Δt_m in the third part of the algorithm.

Algorithm 3 : Forward step of the algorithm

while $A > 0$ **do**
  $VAnext = B_1 \times VA$
  for $l = 1; l < A; l = l + 1$ **do**
    if $\|VAnext[l]\|^2 < \eta[l]$ **then** move $VA[l]$ to VB
    end if
    if $\|VAnext[l]\|^2 \geq \eta[l]$ & $d[l] + 2^{m-1} < k * 2^{m-1}$ **then**
      $VA[l] = VAnext[l] d[l] = d[l] + 2^{m-1}$;
    end if
    if $\|VAnext[l]\|^2 \geq \eta[l]$ & $d[l] + 2^{m-1} == k * 2^{m-1}$ **then**
      $VA[l] = VAnext[l]$; $d[l] = 0$;
      move $VA[l]$ to VD
    end if
    if $\|VAnext[l]\|^2 \geq \eta[l]$ & $d[l] + 2^{m-1} > k * 2^{m-1}$ **then**
      move $VA[l]$ to VC
    end if
  end for
end while
for $i = 2; i \leq m; i = i + 1$ **do**
  $VBnext = B_i \times VB$
  for $l = 1; l \leq B; l = l + 1$ **do**
    if $\|VBnext[l]\|^2 < \eta[l]$ & $d[l] + 2^{m-i} < k * 2^{m-1}$ **then**
      $VB[l] = VB[l]$;
    end if
    if $\|VBnext[l]\|^2 \geq \eta[l]$ & $d[l] + 2^{m-i} < k * 2^{m-1}$ **then**
      $VB[l] = VBnext[l]; d[l] = d[l] + 2^{m-i}$
    end if
    if $\|VBnext[l]\|^2 \geq \eta[l]$ & $d[l] + 2^{m-i} == k * 2^{m-1}$ **then**
      $VB[l] = VBnext[l]; d[l] = 0$; move $VB[l]$ to VD
    end if
    if $\|VBnext[l]\|^2 \geq \eta[l]$ & $d[l] + 2^{m-i} > k * 2^{m-1}$ **then**
      move $VB[l]$ to VC
    end if
  end for
end for
$VB = B_m \times VB$
Make quantum jump for all vectors in VB
for $l = 1; l \leq B; l = l + 1$ **do**
  generate $\eta[l] = U[0, 1]$
  $d[l] = d[l] + 1$
  if $d[l] == k * 2^{m-1}$ **then**
    $d[l] = 0$; move $VB[l]$ to VD
  else
    move $VB[l]$ to VA
  end if
end for

2.2. If the quantum jump condition is not fulfilled for $VBnext[l]$, and the time has not reached the end of the interval $[t_{\tilde{H}}, t_{\tilde{H}} + T_{\tilde{H}}]$, then the vector $VB[l]$ is renewed, together with its current time $d[l]$ (lines 24-26).
2.3. If the quantum jump condition is not fulfilled for $VBnext[l]$, but the right boundary of the interval $[t_{\tilde{H}}, t_{\tilde{H}} + T_{\tilde{H}}]$ is reached, then the result is saved, the vector is moved to block VD (lines 27-29). Propagation step is finished.
2.4. In the quantum jump condition is not fulfilled for $VBnext[l]$, and the current time has gone beyond the interval $[t_{\tilde{H}}, t_{\tilde{H}} + T_{\tilde{H}}]$, then the result is not saved, and the vector is moved to block VC (lines 30-32).

On completion of the second part of the algorithm, block VB contains only those vectors, for which quantum jump occurs only after propagation to the time Δt_m.

3. In the third part of the algorithm (lines 35-45), vectors from block VB are multiplied by matrix B_m, and undergo quantum jumps (lines 35-36). Then for each vectors from block VB, there is a new value of $\eta[l]$ is generated, the current time $d[l]$ is renewed, and the vector is moved to VA or VD (lines 37-45).

The backward step of the algorithm brings all vectors from VC to the right end of the interval $[t_{\tilde{H}}, t_{\tilde{H}} + T_{\tilde{H}}]$. It is implemented in the case, when the class VA becomes empty after an iteration of the forward step. The backward step is organized as follows (Algorithm 4).

1. Vectors from block VC are propagated to the right boundary of the time interval $[t_{\tilde{H}}, t_{\tilde{H}} + T_{\tilde{H}}]$. Block VC is multiplied by exponential operator matrices $B_i, i = \overline{m, .., 2}$, one by one (lines 1-11). Processing of the result of the multiplication of each vector of the block by a matrix, $VCnext[l]$, is determined by the following.
   – In propagation over Δt_i is required to reach an exact boundary of the time interval, $[t_{\tilde{H}}, t_{\tilde{H}} + T_{\tilde{H}}]$, then both the current time and vector are renewed (lines 4-6).
   – If after propagation over time Δt_i we reach the right end of the interval $[t_{\tilde{H}}, t_{\tilde{H}} + T_{\tilde{H}}]$, then the current time and vector are renewed, the vector is moved in block VD (lines 7-9).
2. All vectors from VD move to VA (line 12).

5 Numerical Results

5.1 Computational Infrastructure

We used a node of the Intel Endeavor cluster with 4 high-end 24-core Intel Xeon E7-8890v4 CPUs (2.2 GHz, codename Broadwell). We employed the Intel Math Kernel Library, Intel MPI, and Intel C++ Compiler from the Intel Parallel Studio XE Cluster Edition 2017.

Algorithm 4 : Backward step of the algorithm

for $i = m; i \geq 2$ & $C > 0; i = i - 1$ **do**
  $VCnext = B_i \times VC$
  for $l = 1; l \leq C; l = l + 1$ **do**
    if $d[l]$ & $2^{m-i} = 1$ /* & means bitwise AND here */ **then**
      $VC[l] = VCnext[l]; d[l] = d[l] + 2^{m-i}$
    end if
    if $d[l] == k * 2^{m-1}$ **then**
      $d[l] = 0$; move $VC[l]$ to VD
    end if
  end for
end for
move all vectors from VD to VA

5.2 Methodology

The goal of the experiments and the model problem The main scientific contribution of this paper is the algorithmic optimization of the quantum trajectory method described in the previous section. In this section we empirically show the advantages of the optimized algorithm compared to the baseline implementation. We also identify the most promising run modes by means of trying different combinations of MPI processes, OpenMP threads, and MKL threads. The quantum dimer with $N = 1024$ states described in section 2 is chosen as a testbed problem. For this number of states the run time of the baseline algorithm is acceptable and so we can run and analyze extensive performance tests. Besides, this system size is big enough to highlight the advantage of the optimized algorithm.

Correctness tests First, we check the correctness of the optimized implementation. Note that since in both cases we are dealing with stochastic algorithms, the results will not be exactly the same. Given that the correctness of the baseline implementation has been verified in our earlier work [6], we take its results as a basis for further comparison. Correctness evaluation of the optimized version consisted of two stages. At the first stage, we generated a sequence of pseudo-random numbers and used it for both algorithms. Then we compared the resulting density matrices. For the time periods considered, the relative difference did not exceed 10^{-14}, which can be explained by the different order of floating-point operations. At the second stage we used the time intervals and numbers of trajectories sufficient to reach the attractor, which is of great interest for researchers of this kind of problems. We found that the results of the optimized algorithm were in the 95% confidence interval computed for the baseline version. Thus, our experiments demonstrate that the results of the optimized code match the expectations.

Performance evaluation A straightforward choice of the performance metric is to compare the run time to reach the attractor with the given accuracy. How-

ever, the problem considered is computationally intensive and testing many configurations is somewhat wasteful in terms of CPU-hours consumed. Therefore, we employ the number of trajectories processed per second, while propagating the system for one time period, as the performance metric. Since our performance evaluation experiments are done for a single period, the metric corresponds to simply trajectories per second.

The plan for performance evaluation takes into account the following features. The hotspot of the baseline implementation is the dense matrix-vector multiplication routine. On the contrary, the optimized version spends most time on dense matrix-matrix multiplication. Workload imbalance is under 5% even for small numbers of trajectories.

The baseline implementation employs MPI + OpenMP parallelism on the level of trajectories with sequential MKL matrix-vector multiplication. Parallel matrix-vector multiplication is not beneficial because of a small workload per invocation combined with good balancing on MPI + OpenMP level. The optimized version uses the same MPI + OpenMP scheme, but we additionally study efficiency of internal parallelism in MKL matrix-matrix multiplication.

Based on the above-mentioned considerations, we fixed the integration time to be equal to one period and varied the number of processes P, the number of threads T and, for the optimized version, the number of MKL threads M. For both versions the total number of threads in each configuration was equal to the number of cores (96). Previously we have checked that using all 96 cores indeed yields better performance (in terms of the metric used) compared to smaller numbers of cores. All experiments were performed on a single node of the Endeavor system, since the scaling efficiency on distributed memory is close to linear due to a small workload imbalance and virtually absent communications between nodes.

5.3 Results and Discussion

First, we found the empirically best combination of processes and threads for the baseline version (Table 2). The value of the performance metric varies between 0.8 and 1.69 trajectories per second, with the optimal configuration being 4 processes with 24 threads per process. In this mode each process is run on a separate 24-core CPU with an affinity mask used to pin OpenMP threads to cores.

The next series of experiments concerns the optimized version with varying numbers of MPI processes, OpenMP threads and MKL threads. The results are presented at Table 3. Same as for the baseline version, the configurations with external parallelism and sequential MKL routines are superior. The best configuration is again 4 processes with 24 threads per process, scoring 29.09 trajectories per second. In this configuration the optimized version outperforms the baseline version by a factor of 17.21, which proves efficacy of the proposed approach to optimization. Increasing the problem size will likely further increase the speedup due to the growing advantage of matrix-matrix multiplication over a set of matrix-vector multiplications.

Table 2. Comparison of MPI + OpenMP configurations for the baseline version.

# processes	# threads	trajectories per second
1	96	0.80
2	48	0.63
4	**24**	**1.69**
8	12	1.49
12	8	1.44
24	4	1.39
48	2	1.36
96	1	1.35

Table 3. Comparison of MPI, OpenMP and MKL configurations for the optimized version.

# processes	# OpenMP threads	# MKL threads	trajectories per second
1	1	96	5.57
1	96	1	27.60
2	1	48	11.69
2	48	1	28.07
4	1	24	21.58
4	**24**	**1**	**29.09**
8	1	12	16.32
8	12	1	25.75
12	1	8	27.03
12	8	1	21.02
24	1	4	27.33
24	4	1	18.48
48	1	2	13.99
48	2	1	16.87
96	1	1	13.78

To assess the hardware usage efficiency we apply the Roofline model [11]. Presented several years ago, this method of analysis is widely used to compare the achieved and theoretically attainable performance on specific computing systems. The main advantage of this model is a visual representation of the achieved performance and its theoretical upper bounds. In our experiments we collected the data traffic through L1 cache and arithmetic intensity (AI) using the Roofline Analysis of Intel Advisor. The baseline and optimized implementations were run using the best combination of MPI processes and OpenMP threads. The resulted Roofline model is presented below as a log-log plot (Fig. 1). The arithmetic intensity, computed as the number of floating point operations related to the data traffic through L1 cache, is shown on the horizontal axis. The vertical axis corresponds to the achieved and attainable floating-point performance in double precision. Four dotted sloping lines show the peak performance as a function of arithmetic intensity with fixed memory bandwidth of L1, L2, L3, and DRAM. Three horizontal dotted lines show peak performance for double precision float-

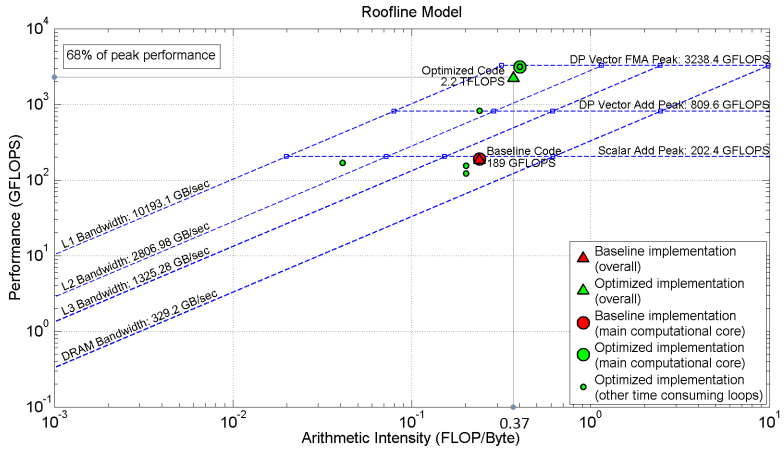

Fig. 1. Roofline model for the baseline and optimized algorithms

ing point computations in the scalar, vector and vector fused multiply-add (vector FMA) modes, respectively. The collected performance data is presented as follows. The red triangle represents the overall baseline version performance 184.69 GFLOPS with $AI = 0.239$ FLOP/Byte. The green triangle represents the overall optimized version performance 2211.92 GFLOPS with $AI = 0.37$ FLOP/Byte. The red and green circles correspond to the main hotspots of both implementations. Thus, the largest red circle corresponds to the MKL dense matrix multiplication routine. It achieved 3128.504 GFLOPS that is very close to the peak hardware performance. Overall, the optimized version achieved 68% of the 4-socket 96-core Intel Broadwell CPU peak performance which is quite well for state-of-the-art scientific applications.

6 Conclusions

We proposed and validated the optimized version of the quantum trajectory method, which allows to find asymptotic states of open quantum systems. The central idea is the clustering (grouping) of trajectories into matrices and substitution of multiple matrix-vector multiplication operations with a single matrix-matrix multiplication. This modification significantly increases efficiency of the multi-level hierarchic memory usage due to the potential of re-using the data, previously loaded in the different level cache memory. The original algorithm [6] did not allow for an automatic merging of trajectories due to the different times of quantum jumps on every trajectory. It required a substantial rewriting of the code, which proved to be completely justified. Computational results showed more than 17-fold acceleration with the testbed quantum model of the dimension $N = 1024$, which demonstrated a possibility of substantial economy of

computational resources or/and time of calculations. The obtained results open the door to studying systems of even greater dimension. We expect our approach to be applicable to many models, actual and timely in different fields of modern quantum physics.

Acknowledgements

We are grateful to Intel Corporation and Lobachevsky University of Nizhni Novgorod for access to the systems used for performing computational experiments presented in this paper. This work was supported by the Russian Science Foundation grant No. 15-12-20029.

References

1. Breuer, H.-P., and Petruccione, F.: The Theory of Open Quantum Systems. Oxford University Press, Oxford (2002)
2. Lindblad, G.: On the generators of quantum dynamical semigroups. Commun. Math. Phys. 48(2), 119–130 (1976)
3. Daley, A.: Quantum trajectories and open many-body quantum systems. J. Adv. Phys. 63(2), 77–149 (2014)
4. Bukov, M., D'Alessio, L., Polkovnikov, A.: Universal high-frequency behavior of periodically driven systems: from dynamical stabilization to Floquet engineering. Adv. Phys. 64(2), 139–226 (2015)
5. Carmichael, H.J.: An Open Systems Approach to Quantum Optics. Springer, Berlin (1993)
6. Volokitin, V., Liniov, A., Meyerov, I., Hartmann, M., Ivanchenko, M., Hänggi, P., Denisov, S.: Towards quantum attractors: Sampling asymptotic states of modulated open systems with quantum trajectories. arxiv:1612.03848 (2016)
7. Laptyeva, T.V., Kozinov, E.A., Meyerov, I.B., Ivanchenko, M.V., Denisov, S., Hänggi, P.: Calculating Floquet states of large quantum systems: A parallelization strategy and its cluster implementation. Comp. Phys. Comm. 201, 85–94 (2016)
8. Hartmann, M., Poletti, D., Ivanchenko, M., Denisov, S., Hänggi, P.: Asymptotic state of many-body open quantum systems under time-periodic modulations. arxiv:1606.03896 (2016)
9. Diehl, S., Micheli, A., Kantian, A., Kraus, B., Büchler, H.P., Zoller, P.: Quantum states and phases in driven open quantum systems with cold atoms. Nature Physics 4(11), 878–883 (2008)
10. Diehl, S., Tomadin, A., Micheli, A., Fazio, R., Zoller, P.: Dynamical Phase Transitions and Instabilities in Open Atomic Many-Body Systems. Phys. Rev. Lett. 105, 015702 (2010)
11. Williams, S., Waterman, A., Patterson, D.: Roofline: an insightful visual performance model for multicore architectures. Commun. ACM 52(4), 65-76 (2009)