Symulacja w Badaniach i Rozwoju
Vol. 6, No. 1/2015

Joanna WIŚNIEWSKA
Wojskowa Akademia Techniczna, 00-908 Warszawa, ul. Kaliskiego 2
E-mail: jwisniewska@wat.edu.pl

Marek SAWERWAIN
Uniwersytet Zielonogórski, 65-256 Zielona Góra, ul. Licealna 9
E-mail: M.Sawerwain@issi.uz.zgora.pl

Symulacja metody kwantowych trajektorii dla problemów optyki kwantowej oraz informatyki kwantowej

1 Wstęp

Metody Monte Carlo, począwszy od pierwszych prac Johna von Neumanna, Stanisława Ulama oraz Nicholasa Metropolisa [6, 7], są szeroko wykorzystywane w obliczeniach dotyczących obszaru chemii, astrofizyki, fizyki kwantowej oraz obszaru odnoszącego się do obliczeń kwantowych.

W przypadku problemów fizyki kwantowej, czy też obliczeń kwantowych wykorzystujących metody Monte Carlo można wskazać wiele otwartych projektów obliczeniowych (lista oprogramowania dostępna pod adresem: http://www.quantiki.org/wiki/List_of_QC_simulators).

Na wskazanej liście projektów można wskazać kilka projektów odnoszących się do tzw. kwantowych trajektorii: a są to między innymi pakiet autorstwa Sze M. Tan [10], pakiet QuTIP [5], a także C++QED przedstawiony w pracy [12]. Dostępne jest też nieco starsze rozwiązanie [9], również wykorzystujące technikę trajektorii kwantowych.

Jednakże wymienione rozwiązania nie wykorzystują wielordzeniowych systemów opartych na CPU (ang. *Central Processing Unit*), jak i GPU (ang. *Graphics Processing Unit*). Wyjątkiem jest pakiet QuTIP, gdzie podczas obliczenia poszczególnych trajektorii kwantowych wykorzystuje się dostępne rdzenie obliczeniowe. Przy czym obliczenia są realizowane tylko za pomocą CPU. W pracy [5] pokazano, iż w przypadku równoległego przetwarzania trajektorii, w stosunku do starszych rozwiązań jak na przykład: pakiet [10], uzyskuje się liniowe przyspieszenie. Zastosowanie GPU w przypadku trajektorii pozwala na dalsze zwiększenie wydajności, a także pozwala na osiągnięcie lepszej dokładności obliczeń w krótszym czasie, gdyż większa liczba obliczonych, a później uśrednionych trajektorii, statystycznie polepsza dokładność uzyskanych danych.

W tym artykule zostanie zaprezentowana implementacja metody kwantowych trajektorii wykonana za pomocą technologii CUDA (ang. *Compute Unified Device Architecture*). Celem takiego podejścia jest umożliwienie wspomnianym symulatorom programowym jak najszybszego i najdokładniejszego odwzorowania zjawisk z zakresu optyki i informatyki kwantowej. Badania tego rodzaju mogą przyczynić się do rozwoju wiedzy na temat powyższych zjawisk bez inwestowania środków w nieporównywalnie

droższe eksperymenty fizyczne. Przedstawione zostaną także testy wydajności zaproponowanej implementacji. Algorytm kwantowych trajektorii wymaga także odpowiednich metod do generowania liczb pseudolosowych (pakiet CUDA oferuje odpowiednie generatory liczb pseudolosowych o dobrej wydajności oraz w pełni dostosowane do pracy w środowisku równoległym), a także podania odpowiednich metod do rozwiązywania zagadnienia początkowego – odpowiednie metody zostały zaimplementowane w proponowanym rozwiązaniu i krótko omówione w dalszej części artykułu.

2 Metoda kwantowych trajektorii

Opis dynamiki stanów kwantowych można podzielić na dwa podstawowe przypadki, tzw. ewolucję zamkniętą, która jest dość często stosowana, na przykład: w przypadku obwodów kwantowych, tj. obliczeń kwantowych, oraz ewolucję w środowisku otwartym, gdzie również otoczenie wpływa na postać stanu kwantowego. Nie jest naszym celem w tym miejscu opisywać, jak przedstawia się matematyczny opis dynamiki wspominanych dwóch przypadków. Szczegółowy opis można odszukać na przykład w [8] oraz [11].

Nadmienimy tylko, iż w przypadku zamkniętym (tzw. ewolucja unitarna) stosuje się równanie Schrödingera [1, 3]:

$$\text{(A) } i\hbar\frac{\partial}{\partial t}\Psi = \hat{H}\Psi,\ \text{(B) } i\hbar\frac{d}{dt}\left|\psi\right\rangle = H\left|\psi\right\rangle\ , \tag{1}$$

gdzie równanie (A) to cząstkowe równanie różniczkowe, natomiast w przypadku symulacji numerycznych stosuje się postać (B). W postaci (B) wielkość H to tzw. Hamiltonian opisujący dynamikę, a stan $\left|\psi\right\rangle$ reprezentuje stan badanego systemu w chwili t.

Jeśli natomiast należy uwzględnić wpływ otoczenia, to stosowane jest równanie von Neumanna, które opisuje uogólniony stan układu za pomocą operatora gęstości:

$$\dot{\rho}_{tot}(t) = -\frac{i}{\hbar}\left[H_{tot}, \rho_{tot}(t)\right],\quad H_{tot} = H_{sys} + H_{env} + H_{\text{int}}\ , \tag{2}$$

gdzie H_{sys} opisuje dynamikę systemu zamkniętego, H_{env} dynamikę otoczenia, a H_{int} dynamikę interakcji otoczenia z systemem. Dość często stosuje się tzw. operację śladu częściowego, aby usunąć wpływ otoczenia, i w ten sposób uzyskuje się tzw. równanie główne Lindblada o postaci:

$$\dot{\rho}(t) = -\frac{i}{\hbar}\left[H(t), \rho(t)\right] + \sum_n \frac{1}{2}\left[2C_n\rho(t)C_n^+ - \rho(t)C_n^+C_n - C_n^+C_n\rho(t)\right]\ , \tag{3}$$

gdzie C_n to tzw. operatory collapsu, które reprezentują wpływ otoczenia na symulowany układ. Zastosowanie takiego operatora naturalnie zmienia postać stanu. Jednakże nadal mamy do czynienia z wykładniczą złożonością obliczeń w trakcie symulacji zachowania układu kwantowego.

Obecnie uznaną metodą zmniejszenia wymagań pamięciowych jest stosowanie tzw. metody kwantowych trajektorii. Jednakże, aby poprawnie przybliżyć zachowanie się układu kwantowego, wymagana jest symulacja wielu pojedynczych trajektorii (co jest naturalną konsekwencją losowego charakteru metod Monte Carlo). Poszczególne trajektorie można symulować niezależnie od siebie, co ułatwia równoległą implementację metody kwantowych trajektorii.

Podsumowując, symulacja ewolucji systemu w przypadku metod kwantowych trajektorii jest opisana przez Hamiltonian o postaci:

$$H_{eff} = H_{sys} - \frac{i\hbar}{2}\sum_n C_n^+ C_n \,. \tag{4}$$

Prawdopodobieństwo tzw. kwantowego skoku, czyli zastosowania operatora collapsu C_n do stanu układu $|\psi\rangle$ w chwili t, jest opisane jako:

$$\delta p = \delta t \sum_n \langle \psi(t) | C_n^+ C_n | \psi(t) \rangle \,. \tag{5}$$

Natomiast stan systemu kwantowego po zastosowaniu operatora collapsu przedstawia się następująco:

$$|\psi(t+\delta t)\rangle = \frac{C_n|\psi(t)\rangle}{\sqrt{\langle \psi(t)|C_n^+ C_n|\psi(t)\rangle}} \,. \tag{6}$$

W przypadku, gdy mamy więcej niż jeden operator collapsu, prawdopodobieństwo zastosowania i-tego operatora C jest opisane w następujący sposób:

$$P_i(t) = \frac{\langle \psi(t)|C_i^+ C_i|\psi(t)\rangle}{\delta p} \,. \tag{7}$$

W procesie symulacji należy naturalnie skorzystać z generatora liczb pseudolosowych, aby zasymulować probabilistyczny wybór odpowiedniego C_i. Wszystkie powyższe obliczenia sprowadzają się do operacji na macierzach oraz wektorach, przy czym postać wspomnianych macierzy zazwyczaj jest wstęgowa, tj. rzadka, toteż dla oszczędności pamięci, a także znacznego przyspieszenia obliczeń należy stosować macierze typu sparse. Ze względu na wiele operacji typu iloczyn macierz-wektor, zastosowany został format CSR (ang. *Compressed Sparse Row*), który oferuje w tym przypadku dalsze zwiększenie efektywności.

Powyższe niezbędne uwagi, pozwalają na poniższy słowny zapis algorytmu symulacji pojedynczej trajektorii, w postaci następujących czterech głównych kroków obliczeniowych (analogicznie do podejścia pokazanego np. w pracy [5]):

1. Wybierana jest wartość losowa $r \in (0,1)$ z rozkładem jednostajnym ciągłym, gdzie wartość r opisuje prawdopodobieństwo wystąpienia kwantowego skoku.
2. Rozwiązywane jest równanie Schrödingera (postać (B) ze wzoru (1)) przy pomocy

Hamiltonianu H_{eff} do czasu t, tak aby spełniona była nierówność: $\langle\psi(t)|\psi(t)\rangle \geq r$, co oznacza wystąpienie kwantowego skoku.

3. Wystąpienie kwantowego skoku powoduje operację projekcji systemu przy czasie t do jednego ze stanów opisanych w równaniu (6). Odpowiedni operator C_i jest wybierany tak, aby dla odpowiedniego n spełniona była relacja:

$$\sum_{i=1}^{n} P_i(t) \geq r; \tag{8}$$

poszczególne prawdopodobieństwa P_i są określone przez równanie (7).

4. Otrzymany, w wyniku projekcji, stan funkcji falowej stanowi nową wartość początkową dla czasu t. Losowana jest nowa wartość r i procedura symulacji trajektorii rozpoczyna się od kroku pierwszego. Ogólnie symulacja trajektorii jest przeprowadzana do wcześniej wskazanej wartości czasu t.

3 Szczegóły implementacji w architekturze CUDA C/C++

Algorytm kwantowych trajektorii (QTM – ang. *Quantum Trajectories Method*) ze względu na swoją podstawową własność, czyli obliczanie wielu niezależnych trajektorii, można dość łatwo zapisać jako algorytm równoległy. W algorytmie wyróżnia się dwa główne etapy:

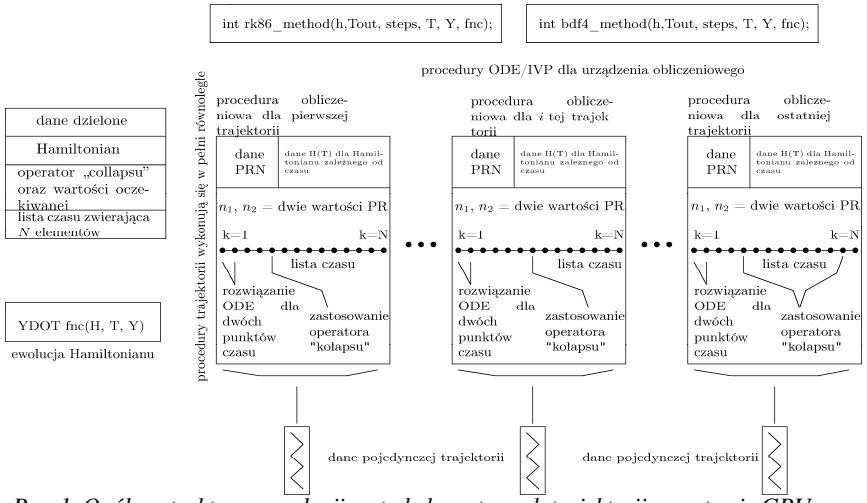
1. Symulację kwantowych trajektorii według metody opisanej w powyższej sekcji.
2. Uśrednienie otrzymanych trajektorii i utworzenie ostatecznej trajektorii.

Główne zadanie, które w całym procesie obliczeniowym zabiera najwięcej czasu, to naturalnie sam proces wyznaczania trajektorii. Podstawowa strategia implementacji algorytmu QTM wykorzystuje wspominaną własność, iż indywidualne trajektorie są obliczane niezależnie od pozostałych, więc na etapie wyznaczania trajektorii istnieje pełna niezależność danych pomiędzy poszczególnymi trajektoriami. Łatwo także wskazać dane wspólne, jak np.: definicja Hamiltonianu, definicja operatorów collapsu, czy zmienna czasu w postaci listy lub tablicy, zawierająca poszczególne wartości zmiennej czasowej. Wymienione rodzaje danych mogą pozostawać niezmienne podczas wyznaczania poszczególnych trajektorii.

W przypadku implementacji algorytmu QTM należy też wskazać duży obszar danych lokalnych w każdym wątku przetwarzającym pojedynczą trajektorię. Do zmiennych lokalnych przynależą także zmienne opisujące stany generatorów liczb pseudolosowych, które dla metod Monte Carlo muszą wytwarzać różne sekwencje liczb w poszczególnych wątkach, aby symulacja była poprawna. Niestety, duża ilość danych lokalnych, związanych też z wieloma instancjami procedury do rozwiązywania zwyczajnych równań różniczkowych (ODE – ang. *Ordinary Differential Equation*), oznacza, iż warto rozpatrywać dwa przypadki. Pierwszy przypadek stanowią symulacje małych układów, gdzie ograniczone zasoby szybkiej pamięci lokalnej dostępne w jednostkach obliczeniowych GPU są w takim przypadku wystarczające. Drugi przypadek to duże systemy, gdzie niezbędne jest korzystanie z pamięci głównej GPU.

Schemat zadań, jaki jest wykonywany w jądrze obliczeniowym podczas wyznaczania wartości trajektorii kwantowych, został przedstawiony na rysunku 1.

Dodajmy, iż drugi etap QTM, polegający na uśrednieniu otrzymanego zestawu trajektorii w jedną finalną postać, można efektywnie zrealizować, wykorzystując operację redukcji równoległej.



Rys. 1. Ogólna struktura symulacji metody kwantowych trajektorii w systemie GPU (wartość PR jest pozyskiwana z generatora liczb pseudolosowych)

Fig. 1. The general structure of quantum trajectories method simulation for GPU (value PR is obtained with use of pseudorandom number generator)

Realizacja algorytmu w postaci jądra obliczeniowego dla GPU wymaga zwrócenia szczególnej uwagi na postać danych wykorzystywanych w metodzie kwantowych trajektorii. Wynikiem algorytmu QTM jest ciąg tzw. wartości oczekiwanych, będących liczbami rzeczywistymi (wartości te otrzymuje się poprzez zastosowanie dodatkowego operatora unitarnego na stanie układu po zakończeniu obliczeń dotyczących określonej trajektorii). Wektor stanu oraz macierze poszczególnych operatorów wymagają, aby elementy tych struktur były wartościami zespolonymi. Pakiet CUDA oferuje odpowiedni typ danych cuComplex (jest to typ szablonowy języka C++). Jednakże w implementacji użyto mniej zaawansowaną implementację liczb zespolonych, ponieważ dodatkowy kod odpowiedzialny za stabilność numeryczną podczas realizacji operacji, takich jak dzielenie czy obliczenia modułu, nie jest potrzebny. Wszystkie wartości są znormalizowane, więc zastosowanie bardziej podstawowej implementacji ma znaczenie dla wydajności całego systemu i nie powoduje kłopotów ze stabilnością numeryczną.

Definicja wzorca simpleComplex, jaka jest używana w dyskutowanej implementacji, to typowa definicja oparta na strukturze:

```
template <typename T> struct simpleComplex {
    T re;
    T im;
};
```

Naturalnie wraz z typem simpleComplex zostały przeciążone podstawowe operatory sumy, iloczynu i etc., co pozwala zarówno dla kodu gospodarza (ang. *host*) jak też kodu wykonywanego po stronie urządzenia (ang. *device*), na czytelny zapis operacji arytmetycznych. W podobny sposób zdefiniowany został typ uVector. Typ ten można stosować zarówno dla kodu wykonywanego po stronie hosta, jak i po stronie urządzenia GPU. Dla wygody użytkownika zdefiniowano także operator dostępu do elementów wektora:

```
    template <typename T, size_t v_size>
    struct uVector {
       unsigned int size;
       T m[ v_size ];
       __host__ __device__  T& operator[]
                           (const size_t idx) { return m[idx]; };
       __host__ __device__  const T& operator[]
                           (const size_t idx) const { return
m[idx]; };
    };
```

Pozostałe pomocnicze definicje, jak typ reprezentujący pełną macierz uMatrix oraz macierz typu sparse uCSRMatrix, zostały oparte na wzorcu uVector.

Istotnym elementem są generatory liczb losowych. Zastosowano rozwiązanie CURAND dostępne w pakiecie CUDA, oferujące wystarczająco wysoką wydajność oraz dobre własności statystyczne. Dostępne w pakiecie cuRand generatory liczb pseudolosowych to XORWOW oraz MRG32k3A.

Jednak przy rozpatrywaniu określonej symulacji dynamiki systemu kwantowego za pomocą QTM może się okazać iż zamiast stosować generator liczby pseudolosowych w każdym wątku warto użyć wcześniej przygotowanego zbioru wartości losowych (czy to za pomocą generatora sekwencji pseudolosowych, czy też w oparciu o źródła fizyczne [2] oraz [4]). Nie wpłynie to istotnie na wielkość transferu danych, lecz pozwoli zwolnić część zasobów pamięci lokalnej. W efekcie może to pozwolić na zwiększenie liczby wyznaczanych trajektorii w obrębie jednego bloku.

Kolejnym niezbędnym elementem do otrzymania poprawnej implementacji metody kwantowych trajektorii są metody numeryczne do rozwiązywania zwyczajnych równań różniczkowych lub dokładniej – problem zagadnienia początkowego. W omawianym symulatorze dokonano implementacji metody Runge Kutta czwartego rzędu (RK4) oraz metody BDF (ang. *Backward Differentiation Formula*).

Implementacja metod została wykonana w postaci wzorca, co umożliwia łatwą konwersję między typami danych liczbowych float i double. Pozwala także na określenie wielkości struktur, jakie będą przetwarzane w tej procedurze obliczeniowej. Jest to konieczne, bowiem metoda QTM wylicza rozwiązanie układu równań, gdzie liczba równań jest wielkością stanu kwantowego podlegającego badaniu.

Zwarta postać metody RK4 jest konsekwencją obecności definicji typów simpleComplex czy uVector wraz z bogatym zestawem przeciążonych operatorów, co pozwala na klarowną implementację operacji arytmetycznych na wektorach i skalarach. Należy jednak ponownie podkreślić, iż w trakcie wyznaczania trajektorii

równolegle funkcjonuje wiele instancji tej metody, a do poprawnego funkcjonowania metoda ta wymaga dodatkowych wektorów, które naturalnie muszą alokować lokalne zasoby dostępne dla każdego wątku.

Zestawy parametrów przekazywanych do metod RK4 i BDF są identyczne i wymagają podania: szerokości przedziału integracji (h), tzw. czasu wyjścia (Tout), maksymalnej liczby iteracji, aktualnej wartości zmiennej czasowej, aktualnego stanu układu Y. Ostatni parametr to funkcja odpowiedzialna za ewolucję wartości stanu za pomocą wcześniej określonego Hamiltonianu.

4 Podsumowanie

Porównajmy zaproponowaną implementację z bardzo popularnym w ostatnim czasie pakietem QuTIP, wspierającym metodę trajektorii kwantowych. Niech przykładowa symulacja dotyczy tzw. unitarnego Hamiltonianu:

$$H = \frac{2\pi}{10}\sigma_x, \quad \sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \tag{9}$$

gdzie σ_x reprezentuje tzw. operator Pauliego *X*, w obszarze informatyki kwantowej nazywany także operatorem negacji. Stan początkowy jest określony następująco:

$$|\psi_0\rangle = |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{10}$$

Operator collapsu użyty w symulacji został określony w następujący sposób:

$$C_0 = \frac{5}{100}\sigma_x, \quad E_0 = \sigma_z, \quad \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \tag{11}$$

gdzie σ_z to kolejny operator Pauliego *Z*, nazywany także operatorem zmiany znaku.

Choć powyższe struktury są niewielkie, to proces symulacji 50 trajektorii za pomocą pakietu QuTIP na komputerze klasy PC, wyposażonego w procesor Intel Core 2 Duo 8400 3.0 Ghz, przy wykorzystaniu jednego rdzenia zabiera około trzech, czterech sekund. W przypadku użycia dwóch rdzeni obliczeniowych czas będzie podobny ze względu na to, iż pakiet QuTIP generuje zbyt duży narzut związany z obsługą dwóch wątków. Naturalnie zwiększenie liczby trajektorii pozwoli zauważyć zysk związany ze zwiększoną ilością rdzeni obliczeniowych.

Wyniki dla karty klasy GeForce 460, wyposażonej 1GB RAM, w przypadku gdy stosowana jest metoda RK4, otrzymujemy czas działania około 0.08 sekundy, co daje około pięćdziesięciokrotne przyspieszenie w stosunku do czasu działania wersji jednoprocesorowej. A należy dodać, iż tylko jeden wątek był wykorzystywany w ramach jednego bloku. Przy użyciu metody BDF4 naturalnie czas działania jest dłuży i wynosi około 0.2 sekundy, więc otrzymane przyspieszenie jest już mniejsze, bo dwudziestokrotne. Naturalnie obliczenia za pomocą CPU oraz GPU były przeprowadzane na liczbach typu double.

Zaproponowane rozwiązanie pozwala na osiągnięcie przyspieszenia w obliczeniach opartych na metodzie kwantowych trajektorii w stosunku do istniejących rozwiązań. Dokładność proponowanej implementacji, dotyczy to szczególnie przypadku, gdy używana jest metoda RK4, jest naturalnie porównywalna do obliczeń przeprowadzanych za pomocą tradycyjnych szeregowych implementacji, czy też równoległych, opartych na tradycyjnych uniwersalnych procesorach. Należy mieć jednak świadomość, że otrzymywane przyspieszenie jest w dużej mierze zależne od sprzętu, którym dysponujemy. Zjawiska kwantowe posiadają wykładniczą naturę i symulowanie ich na „klasycznych" maszynach zawsze wiąże się z pewnymi ograniczeniami, np. w metodzie kwantowych trajektorii należy liczyć się z tym, że otrzymamy pewien uśredniony wynik, ponieważ jest to metoda typu Monte Carlo.

W dalszych planach rozwoju opisanych tu procedur obliczeniowych, przewidziane jest dołączenie innych metod, jak np.: Block BDF (BBDF) czy metody Adamsa-Bashforda, do dwóch już zaimplementowanych metod przeznaczonych do rozwiązywania równań różniczkowych, tj. metody Runge-Kutta oraz BDF. Można oczekiwać, iż metoda BBDF dodatkowo skróci czas obliczeń ze względu na mniejszą liczbę potrzebnych iteracji przy zachowaniu stabilności oferowanej przez metody BDF. Również wykorzystanie nowych możliwości związanych z technologią Dynamic Parallelism (obecna w najnowszych urządzeniach NVIDIA oraz dostępna w ramach CUDA C/C++ od wersji 5.0) może przyczynić się do uzyskania wydajniejszej implementacji opisywanych metod.

Ważnym zagadnieniem jest także dodanie innego rozwiązania związanego z generacją liczb pseudolosowych. Zamiast wykorzystania zestawu generatorów liczb pseudolosowych wartości pseudolosowe będą czerpane ze zbioru wcześniej przygotowanych danych. Pozwoli to na zwolnienie części lokalnych zasobów, a także umożliwi korzystanie z wartości wygenerowanych za pomocą urządzeń fizycznych.

Literatura

1. Chudy M.: *Wprowadzenie do informatyki kwantowej*. EXIT, Warszawa 2011
2. Frauchiger D., Renner R., Troyer M.: *True randomness from realistic quantum devices*. arXiv:1311.4547, 2013
3. Hirvensalo M.: *Algorytmy kwantowe*. WsiP, Warszawa 2004
4. ID Quantique SA., Quantis, product web page http://www.idquantique.com/random-number-generators/products.html, 2013
5. Johansson J.R., Nation P.D. and Nori F.: QuTiP 2: A Python framework for the dynamics of open quantum systems. *Comp. Phys. Comm.*, vol. 184, Issue 4, pp. 1234-1240, 2013
6. Metropolis N., Ulam S.: The Monte Carlo Method. *Journal of the American Statistical Association*, vol. 44, no. 247, pp. 335-341, 1949
7. Metropolis N., Rosenbluth A.W., Rosenbluth M.N., Teller A.H., Teller E.: Equation of state calculation by fast computing machines. *Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087-1092, 1953
8. Nielsen, M.A., Chuang, I.L.: *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010
9. Schacka R., Brun T.A.: A C++ library using quantum trajectories to solve quantum master equations. *Comp. Phys. Comm.*, vol. 102, pp. 210-228, 1997

10. Sze M. Tan: A Computational Toolbox for Quantum and Atomic Optics. *J. Opt. B: Quantum Semiclass. Opt.*, vol. 1, no. 4, pp. 424, 1999
11. Wyatt R.E.: *Quantum Dynamics with Trajectories*. Springer New York, 2005
12. Vukics A.: C++QEDv2: The multi-array concept and compile-time algorithms in the definition of composite quantum systems. *Computer Physics Communications*, vol. 183, pp. 1381-1396, 2012

Streszczenie

W artykule została przedstawiona równoległa implementacja odmiany metody Monte Carlo do symulacji dynamiki kwantowych systemów otwartych – jest to tzw. metoda kwantowych trajektorii (QTM). Implementacja została wykonana za pomocą technologii CUDA i obejmuje ona realizację procedury numerycznej odpowiedzialnej za algorytm QTM. W artykule została też pokazana wydajność otrzymanych metod numerycznych dla QTM w stosunku do innych znanych implementacji.

Słowa kluczowe: metoda kwantowych trajektorii, symulacja kwantowych systemów otwartych, obliczenia numeryczne, technologia CUDA, obliczenia GPU

Simulating the quantum trajectories method for problems related to quantum optics and quantum computing

Summary

The chapter contains a parallel implementation of Monte Carlo method for simulating the open quantum systems' dynamics. The mentioned approach is the Quantum Trajectories Method (QTM). The implementation is carried out with use of CUDA technology and it is based on a numerical procedure realizing QTM algorithm. The chapter presents also a comparison of elaborated numerical methods' performance in comparison to other existing implementations.

Keywords: quantum trajectories method, simulation of open quantum systems, numerical computations, CUDA technology, GPU computation