Computer Physics Communications ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Magnus integrators on multicore CPUs and GPUs

N. Auer, L. Einkemmer *, P. Kandolf, A. Ostermann

Department of Mathematics, University of Innsbruck, Austria

ARTICLE INFO

Article history:
Received 20 September 2017
Received in revised form 16 February 2018
Accepted 20 February 2018
Available online xxxx

Keywords:
Magnus integrators
Graphic processing unit
Parallelization
Commutator-free Magnus integrators
Performance comparison
Heisenberg model

ABSTRACT

In the present paper we consider numerical methods to solve the discrete Schrödinger equation with a time dependent Hamiltonian (motivated by problems encountered in the study of spin systems). We will consider both short-range interactions, which lead to evolution equations involving sparse matrices, and long-range interactions, which lead to dense matrices. Both of these settings show very different computational characteristics. We use Magnus integrators for time integration and employ a framework based on Leja interpolation to compute the resulting action of the matrix exponential. We consider both traditional Magnus integrators (which are extensively used for these types of problems in the literature) as well as the recently developed commutator-free Magnus integrators and implement them on modern CPU and GPU (graphics processing unit) based systems.

We find that GPUs can yield a significant speed-up (up to a factor of 10 in the dense case) for these types of problems. In the sparse case GPUs are only advantageous for large problem sizes and the achieved speed-ups are more modest. In most cases the commutator-free variant is superior but especially on the GPU this advantage is rather small. In fact, none of the advantage of commutator-free methods on GPUs (and on multi-core CPUs) is due to the elimination of commutators. This has important consequences for the design of more efficient numerical methods.

© 2018 Elsevier B.V. All rights reserved.

1. Introduction

To numerically solve the Schrödinger equation with a time-dependent Hamiltonian

$$\mathrm{i}\frac{\mathrm{d}\psi}{\mathrm{d}t} = H(t)\psi(t), \quad \psi(0) = \psi_0, \tag{1}$$

is a problem of significant interest in various fields of quantum mechanics. Applications range from discrete spin systems to (continuous) models of atom–laser interaction. Therefore, it is important to have both good numerical algorithms as well as an efficient implementation on state of the art computer hardware of these algorithms at one's disposal.

Magnus integrators are used in many such applications (see, for example, [1–5]). The implementation of these Magnus integrators (which constitute a subclass of exponential integrators; for more details see [2,6,7]) requires the computation of the action of matrix exponentials in an efficient and stable manner. For some problems, e.g. if the continuous Schrödinger equation is used to model atom-laser interaction, this can be done using fast Fourier techniques. However, for many other interesting problems this is not possible. For the latter case a number of approaches have been proposed in the literature (see, for example, [6,8–12]). Most of them are based on polynomial interpolation. In [11,13] it was shown that interpolation at Leja points is a very efficient way of performing such an approximation for the Schrödinger equation. This algorithm interpolates the exponential function and thus reduces the task of computing the action of a matrix exponential to the task of computing a sequence of matrix–vector products. Let us also note that, in addition to the Schrödinger equation considered in this paper, Magnus integrators have been successfully applied to many related problems.

In addition to the matrix exponential, traditional Magnus integrators of higher order require the computation of matrix commutators (see, for example, [2,6,14]). In time dependent problems (as those considered here) these matrix commutators have to be computed once every time step. Thus, especially for large problem sizes the corresponding cost can outweigh the cost of the matrix–vector products. Recently, commutator-free Magnus integrators have been developed [15,16]. They eliminate commutators but usually require additional matrix–vector products.

Due to the trend towards CPUs with more and more cores as well as the trend towards GPUs, providing an efficient implementation of numerical algorithms on modern multi-core CPUs and GPUs is of great practical importance. Some preliminary work on implementing exponential integrators [17] and matrix functions [12] has been conducted on GPUs (with generally promising results). The purpose of the present work is to investigate the

* Corresponding author.
E-mail address: lukas.einkemmer@uibk.ac.at (L. Einkemmer).

https://doi.org/10.1016/j.cpc.2018.02.019
0010-4655/© 2018 Elsevier B.V. All rights reserved.

performance of both commutator-free and traditional Magnus integrators. This is done in the context of multi-core CPUs and GPUs. Although from a computational complexity point of view, one might conjecture that computing the commutators will dominate the total computational cost, this is not necessarily true in an actual implementation. In particular, on GPUs matrix–matrix products (necessary for computing the commutators) can operate close to peak efficiency while this is usually not the case for matrix–vector products. The comparison will be performed in the context of both short-range interactions (which lead to sparse Hamiltonians $H(t)$) and long-range interactions (which lead to dense Hamiltonians $H(t)$) to ascertain in which situations GPUs result in a significant gain in performance.

This paper is based, in part, on the thesis [18] and is structured as follows. In Section 2 we provide an introduction to Magnus integrators and specify the numerical methods used in the subsequent sections. Section 3 then details the numerical approximation and the implementation. The numerical results are presented and discussed in Section 4. Finally, we conclude in Section 5.

2. Magnus integrators

The solution of the linear differential equation

$$Y'(t) = A(t)Y(t), \quad Y(0) = Y_0, \tag{2}$$

can be expressed as

$$Y(t) = \exp\left(\Omega(t)\right) Y_0, \tag{3}$$

where the difficulty lies in finding a suitable matrix $\Omega(t)$. In [19] Magnus used the ansatz of differentiating (3) to find an expression for $\Omega(t)$. This results in

$$Y'(t) = \frac{\mathrm{d}}{\mathrm{d}t}\exp(\Omega(t))Y_0 = \mathrm{dexp}_{\Omega(t)}(\Omega'(t))\exp(\Omega(t))Y_0, \tag{4}$$

where the operator dexp is defined as

$$\mathrm{dexp}_{\Omega}(C) = \sum_{k=0}^{\infty} \frac{1}{(k+1)!}\,\mathrm{ad}_{\Omega}^{k}(C) \tag{5}$$

see [7]. Here the operator $\mathrm{ad}_{\Omega}^{k}(C)$ is the iterated commutator and recursively defined as

$$\mathrm{ad}_{\Omega}^{j}(C) = \left[\Omega, \mathrm{ad}_{\Omega}^{j-1}(C)\right], \quad j \geq 1,$$

and $\mathrm{ad}_{\Omega}^{0}(C) = C$. Comparing (2) and (4) leads to

$$A(t) = \mathrm{dexp}_{\Omega(t)}(\Omega'(t)), \quad \Omega(0) = 0. \tag{6}$$

By applying the inverse of the derivative of the matrix exponential we obtain a differential equation for Ω. In fact, when $\|\Omega(t)\| < \pi$ the operator $\mathrm{dexp}_{\Omega(t)}$ is invertible and has the convergent series representation

$$\mathrm{dexp}^{-1}_{\Omega(t)}(A(t)) = \sum_{k=0}^{\infty} \frac{\beta_k}{k!}\mathrm{ad}^{k}_{\Omega(t)}(A(t)),$$

where the coefficients β_k denote the Bernoulli numbers. As a result we obtain an explicit differential equation for $\Omega(t)$ as

$$\begin{aligned}\Omega'(t) &= \mathrm{dexp}^{-1}_{\Omega(t)}(A(t)) \\ &= A(t) - \frac{1}{2}\left[\Omega(t), A(t)\right] + \frac{1}{12}\left[\Omega(t), \left[\Omega(t), A(t)\right]\right] + \cdots.\end{aligned} \tag{7}$$

Eq. (7) can be integrated by Picard iteration and this leads to the so-called *Magnus expansion*,

$$\begin{aligned}\Omega(t) = &\int_0^t A(t_1)\mathrm{d}t_1 - \frac{1}{2}\int_0^t \left[\int_0^{t_1} A(t_2)\mathrm{d}t_2, A(t_1)\right]\mathrm{d}t_1 \\ &+ \frac{1}{4}\int_0^t \left[\int_0^{t_1}\left[\int_0^{t_2} A(t_3)\mathrm{d}t_3, A(t_2)\right]\mathrm{d}t_2, A(t_1)\right]\mathrm{d}t_1 \\ &+ \frac{1}{12}\int_0^t \left[\int_0^{t_1} A(t_2)\mathrm{d}t_2, \left[\int_0^{t_1} A(t_2)\mathrm{d}t_2, A(t_1)\right]\right]\mathrm{d}t_1 + \cdots.\end{aligned} \tag{8}$$

To derive numerical methods from the Magnus expansion we assume a constant time step size τ and thus the solution after one time step is

$$Y(t_n + \tau) = \exp(\Omega(t_n + \tau))Y(t_n), \tag{9}$$

resulting in the numerical scheme

$$Y_{n+1} = \exp(\Omega_n)Y_n, \tag{10}$$

for a suitable approximation Ω_n of $\Omega(t_n+\tau)$. One way of deriving a formula for Ω_n is to approximate the integrals in (8) by quadrature rules.

In the following we will introduce the three traditional Magnus integrators that are used for the numerical experiments in Section 4.

Method 1 (*M2*)**.** The first example is the simplest method, which is obtained by truncating the series (8) after the first term and approximating the integral by the midpoint rule. This yields

$$\Omega_n(\tau) = \tau A\left(t_n + \frac{\tau}{2}\right)$$

as an approximation of $\Omega(t_n + \tau)$. The corresponding numerical scheme is the exponential midpoint rule

$$Y_{n+1} = \exp\left(\tau A\left(t_n + \frac{\tau}{2}\right)\right) Y_n,$$

which is of order two.

Method 2 (*M4*)**.** The second example is a scheme of order four. The Magnus series (8) is truncated after the second term and the integrals are approximated by the two-stage Gauss quadrature rule with weights $b_1 = b_2 = \frac{1}{2}$ and nodes $c_1 = \frac{1}{2} - \frac{\sqrt{3}}{6}$, $c_2 = \frac{1}{2} + \frac{\sqrt{3}}{6}$. We obtain

$$Y_{n+1} = \exp\left(\frac{\tau}{2}\left(A_1 + A_2\right) + \frac{\sqrt{3}\tau^2}{12}\left[A_2, A_1\right]\right) Y_n,$$

where $A_1 = A(t_n + c_1\tau)$ and $A_2 = A(t_n + c_2\tau)$.

Method 3 (*M6*)**.** As a third example, we consider the following scheme of order six:

$$\begin{aligned}Y_{n+1} = \exp\Bigg(&B_1 + \frac{1}{2}B_3 \\ &+ \frac{1}{240}\Big[-20B_1 - B_3 + [B_1, B_2], B_2 \\ &- \frac{1}{60}\Big[B_1, 2B_3 + [B_1, B_2]\Big]\Big]\Bigg) Y_n,\end{aligned}$$

where $A_i = A\left(t_n + c_i\tau\right)$ for

$$c_1 = \frac{1}{2} - \frac{\sqrt{15}}{10}, \quad c_2 = \frac{1}{2}, \quad c_3 = \frac{1}{2} + \frac{\sqrt{15}}{10},$$

and

$$B_1 = \tau A_2, \quad B_2 = \frac{\sqrt{15}}{3}\tau(A_3 - A_1), \quad B_3 = \frac{10}{3}\tau(A_3 - 2A_2 + A_1).$$

Please cite this article in press as: N. Auer, et al., Magnus integrators on multicore CPUs and GPUs, Computer Physics Communications (2018), https://doi.org/10.1016/j.cpc.2018.02.019.

Download English Version:

https://daneshyari.com/en/article/6919049

Download Persian Version:

https://daneshyari.com/article/6919049

Daneshyari.com