A μ-mode-based integrator for solving evolution equations in Kronecker form

Marco Caliari

(joint work with F. Cassini, L. Einkemmer, A. Ostermann and F. Zivcovich)

University of Verona (Italy)

June 21, 2021 - 8ECM

Motivating example

- Consider the PDE

$$\begin{aligned}\partial_t u(t,\mathbf{x}) &= \Delta u(t,\mathbf{x}) = \left(\partial_1^2 + \partial_2^2\right) u(t,\mathbf{x}),\\ u(0,\mathbf{x}) &= u_0(\mathbf{x}),\end{aligned}$$

where $\mathbf{x} \in \Omega \subset \mathbb{R}^2$, Ω rectangular domain and $t > 0$

Motivating example

- Consider the PDE

$$\begin{aligned}\partial_t u(t,\mathbf{x}) &= \Delta u(t,\mathbf{x}) = \left(\partial_1^2 + \partial_2^2\right) u(t,\mathbf{x}),\\ u(0,\mathbf{x}) &= u_0(\mathbf{x}),\end{aligned}$$

  where $\mathbf{x} \in \Omega \subset \mathbb{R}^2$, Ω rectangular domain and $t > 0$
- Analytic solution:

$$u(t,\cdot) = \mathrm{e}^{t\Delta} u_0 = \mathrm{e}^{t\partial_1^2}\mathrm{e}^{t\partial_2^2} u_0 = \mathrm{e}^{t\partial_2^2}\mathrm{e}^{t\partial_1^2} u_0$$

Motivating example

- Consider the PDE

$$\begin{aligned}\partial_t u(t,\mathbf{x}) &= \Delta u(t,\mathbf{x}) = \left(\partial_1^2 + \partial_2^2\right) u(t,\mathbf{x}),\\ u(0,\mathbf{x}) &= u_0(\mathbf{x}),\end{aligned}$$

where $\mathbf{x} \in \Omega \subset \mathbb{R}^2$, Ω rectangular domain and $t > 0$

- Analytic solution:

$$u(t,\cdot) = \mathrm{e}^{t\Delta} u_0 = \mathrm{e}^{t\partial_1^2}\mathrm{e}^{t\partial_2^2} u_0 = \mathrm{e}^{t\partial_2^2}\mathrm{e}^{t\partial_1^2} u_0$$

- Space discretization of PDE $\Rightarrow$ ODE system (vector form)

$$\mathbf{u}'(t) = (I_2 \otimes A_1 + A_2 \otimes I_1)\,\mathbf{u}(t), \quad \mathbf{u}(0) = \mathbf{u}_0,$$

where $A_1 \in \mathbb{R}^{n_1 \times n_1}$ and $A_2 \in \mathbb{R}^{n_2 \times n_2}$.

Motivating example: matrix formulation

- Vector solution of ODE system

$$\mathbf{u}(t) = \mathrm{e}^{t(I_2 \otimes A_1 + A_2 \otimes I_1)}\mathbf{u}_0 = \mathrm{e}^{tI_2 \otimes A_1}\mathrm{e}^{tA_2 \otimes I_1}\mathbf{u}_0 = \mathrm{e}^{tA_2 \otimes I_1}\mathrm{e}^{tI_2 \otimes A_1}\mathbf{u}_0$$

but $\mathrm{e}^{tA_2 \otimes I_1}$ and $\mathrm{e}^{tI_2 \otimes A_1}$ are large.

Motivating example: matrix formulation

- Vector solution of ODE system

$$\mathbf{u}(t) = \mathrm{e}^{t(I_2 \otimes A_1 + A_2 \otimes I_1)}\mathbf{u}_0 = \mathrm{e}^{tI_2 \otimes A_1}\mathrm{e}^{tA_2 \otimes I_1}\mathbf{u}_0 = \mathrm{e}^{tA_2 \otimes I_1}\mathrm{e}^{tI_2 \otimes A_1}\mathbf{u}_0$$

  but $\mathrm{e}^{tA_2 \otimes I_1}$ and $\mathrm{e}^{tI_2 \otimes A_1}$ are large.
- ODE system (matrix form)

$$\mathbf{U}'(t) = A_1\,\mathbf{U}(t) + \mathbf{U}(t)A_2^{\mathsf{T}}, \quad \mathbf{U}(0) = \mathbf{U}_0,$$

  where

$$\mathbf{U}(t)(i_1, i_2) \approx u(t, x_1^{i_1}, x_2^{i_2}), \quad i_1 = 1, \ldots, n_1, \quad i_2 = 1, \ldots, n_2.$$

Motivating example: matrix formulation

- Vector solution of ODE system

$$\mathbf{u}(t) = \mathrm{e}^{t(I_2\otimes A_1 + A_2\otimes I_1)}\mathbf{u}_0 = \mathrm{e}^{tI_2\otimes A_1}\mathrm{e}^{tA_2\otimes I_1}\mathbf{u}_0 = \mathrm{e}^{tA_2\otimes I_1}\mathrm{e}^{tI_2\otimes A_1}\mathbf{u}_0$$

  but $\mathrm{e}^{tA_2\otimes I_1}$ and $\mathrm{e}^{tI_2\otimes A_1}$ are large.

- ODE system (matrix form)

$$\mathbf{U}'(t) = A_1\,\mathbf{U}(t) + \mathbf{U}(t)A_2^{\mathsf{T}}, \quad \mathbf{U}(0) = \mathbf{U}_0,$$

  where

$$\mathbf{U}(t)(i_1, i_2) \approx u(t, x_1^{i_1}, x_2^{i_2}), \quad i_1 = 1, \ldots, n_1, \quad i_2 = 1, \ldots, n_2.$$

- Matrix solution of ODE system

$$\mathbf{U}(t) = \mathrm{e}^{tA_1}\mathbf{U}_0\,\mathrm{e}^{tA_2^{\mathsf{T}}}$$

Motivating example: matrix formulation

- Vector solution of ODE system

$$\mathbf{u}(t) = \mathrm{e}^{t(I_2\otimes A_1 + A_2\otimes I_1)}\mathbf{u}_0 = \mathrm{e}^{tI_2\otimes A_1}\mathrm{e}^{tA_2\otimes I_1}\mathbf{u}_0 = \mathrm{e}^{tA_2\otimes I_1}\mathrm{e}^{tI_2\otimes A_1}\mathbf{u}_0$$

  but $\mathrm{e}^{tA_2\otimes I_1}$ and $\mathrm{e}^{tI_2\otimes A_1}$ are large.

- ODE system (matrix form)

$$\mathbf{U}'(t) = A_1\,\mathbf{U}(t) + \mathbf{U}(t)A_2^{\mathsf{T}}, \quad \mathbf{U}(0) = \mathbf{U}_0,$$

  where

$$\mathbf{U}(t)(i_1, i_2) \approx u(t, x_1^{i_1}, x_2^{i_2}), \quad i_1 = 1, \ldots, n_1, \quad i_2 = 1, \ldots, n_2.$$

- Matrix solution of ODE system

$$\mathbf{U}(t) = \mathrm{e}^{tA_1}\mathbf{U}_0\,\mathrm{e}^{tA_2^{\mathsf{T}}} \quad \text{Now } \mathrm{e}^{tA_1} \text{ and } \mathrm{e}^{tA_2^{\mathsf{T}}} \text{ are small.}$$

Motivating example: algorithm

- Algorithm to compute $\mathbf{U}(t) = \mathrm{e}^{tA_1}\mathbf{U}_0\,\mathrm{e}^{tA_2^{\mathsf{T}}}$:

$$
\begin{aligned}
\mathbf{U}^{(0)} &= \mathbf{U}_0, \\
\mathbf{U}^{(1)}(\cdot, i_2) &= \mathrm{e}^{tA_1}\mathbf{U}^{(0)}(\cdot, i_2), \qquad && i_2 = 1, \ldots, n_2, \\
\mathbf{U}^{(2)}(i_1, \cdot) &= \mathrm{e}^{tA_2}\mathbf{U}^{(1)}(i_1, \cdot), \qquad && i_1 = 1, \ldots, n_1, \\
\mathbf{U}(t) &= \mathbf{U}^{(2)}.
\end{aligned}
$$

Motivating example: algorithm

- Algorithm to compute $\mathbf{U}(t) = \mathrm{e}^{tA_1}\mathbf{U}_0\,\mathrm{e}^{tA_2^\mathsf{T}}$:

$$\begin{aligned}
\mathbf{U}^{(0)} &= \mathbf{U}_0, \\
\mathbf{U}^{(1)}(\cdot, i_2) &= \mathrm{e}^{tA_1}\mathbf{U}^{(0)}(\cdot, i_2), \qquad && i_2 = 1, \ldots, n_2, \\
\mathbf{U}^{(2)}(i_1, \cdot) &= \mathrm{e}^{tA_2}\mathbf{U}^{(1)}(i_1, \cdot), \qquad && i_1 = 1, \ldots, n_1, \\
\mathbf{U}(t) &= \mathbf{U}^{(2)}.
\end{aligned}$$

- Idea: generalize this approach to arbitrary discretizations and dimensions.

Linear problem in Kronecker form

- Consider the differential equation

$$\mathbf{u}'(t) = M\mathbf{u}(t) = \left(\sum_{\mu=1}^{d} A_{\otimes\mu}\right)\mathbf{u}(t), \quad \mathbf{u}(0) = \mathbf{u}_0,$$

where

$$A_{\otimes\mu} = I_d \otimes \cdots \otimes I_{\mu+1} \otimes A_\mu \otimes I_{\mu-1} \otimes \cdots \otimes I_1,$$

A_μ is an arbitrary small $n_\mu \times n_\mu$ matrix and I_μ is the identity matrix of size n_μ

Linear problem in Kronecker form

- Consider the differential equation

$$\mathbf{u}'(t) = M\mathbf{u}(t) = \left(\sum_{\mu=1}^{d} A_{\otimes\mu}\right)\mathbf{u}(t), \quad \mathbf{u}(0) = \mathbf{u}_0,$$

  where

$$A_{\otimes\mu} = I_d \otimes \cdots \otimes I_{\mu+1} \otimes A_\mu \otimes I_{\mu-1} \otimes \cdots \otimes I_1,$$

  A_μ is an arbitrary small $n_\mu \times n_\mu$ matrix and I_μ is the identity matrix of size n_μ
- We call it linear problem in *Kronecker form*.

Linear problem in Kronecker form: tensor formulation

- Vector solution of a linear problem in Kronecker form:

$$\mathbf{u}(t) = \mathrm{e}^{tA_{\otimes 1}} \cdots \mathrm{e}^{tA_{\otimes d}} \mathbf{u}_0$$

Linear problem in Kronecker form: tensor formulation

- Vector solution of a linear problem in Kronecker form:

$$\mathbf{u}(t) = \mathrm{e}^{tA_{\otimes 1}} \cdots \mathrm{e}^{tA_{\otimes d}} \mathbf{u}_0$$

- Let $\mathbf{U}(t)$ be an order d tensor such that

$$\mathbf{U}(t)(i_1, \ldots, i_d) \approx u(t, x_1^{i_1}, \ldots, x_d^{i_d}),$$

where $1 \leq i_\mu \leq n_\mu$, $1 \leq \mu \leq d$

Linear problem in Kronecker form: tensor formulation

- Vector solution of a linear problem in Kronecker form:

$$\mathbf{u}(t) = \mathrm{e}^{tA_{\otimes 1}} \cdots \mathrm{e}^{tA_{\otimes d}} \mathbf{u}_0$$

- Let $\mathbf{U}(t)$ be an order d tensor such that

$$\mathbf{U}(t)(i_1, \ldots, i_d) \approx u(t, x_1^{i_1}, \ldots, x_d^{i_d}),$$

  where $1 \leq i_\mu \leq n_\mu$, $1 \leq \mu \leq d$
- As in the two-dimensional case, the computation of $\mathbf{u}(t)$ just requires the actions of the matrices e^{tA_μ} on properly chosen “parts” of $\mathbf{U}(t)$.

Linear problem in Kronecker form

- Algorithm to compute $\mathbf{U}(t)$

$$\mathbf{U}^{(0)} = \mathbf{U}_0,$$

$$\mathbf{U}^{(1)}(\cdot, i_2, \ldots, i_d) = \mathrm{e}^{tA_1}\mathbf{U}^{(0)}(\cdot, i_2, \ldots, i_d),$$
$$\mathbf{U}^{(2)}(i_1, \cdot, i_3, \ldots, i_d) = \mathrm{e}^{tA_2}\mathbf{U}^{(1)}(i_1, \cdot, i_3, \ldots, i_d),$$
$$\ldots$$
$$\mathbf{U}^{(d)}(i_1, \ldots, i_{d-1}, \cdot) = \mathrm{e}^{tA_d}\mathbf{U}^{(d-1)}(i_1, \ldots, i_{d-1}, \cdot),$$

$$\mathbf{U}(t) = \mathbf{U}^{(d)}$$

where $1 \leq i_\mu \leq n_\mu$.

Tensor algebra formulation

- Let $\mathbf{U} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ be an order d tensor. A μ-fiber of $\mathbf{U}$ is a vector in $\mathbb{C}^{n_\mu}$ obtained by fixing every index of the tensor but the μth, that is

$$\mathbf{U}(i_1, \ldots, i_{\mu-1}, \cdot, i_{\mu+1}, \ldots, i_d)$$

Tensor algebra formulation

- Let $\mathbf{U} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ be an order d tensor. A μ-fiber of $\mathbf{U}$ is a vector in $\mathbb{C}^{n_\mu}$ obtained by fixing every index of the tensor but the μth, that is

$$\mathbf{U}(i_1, \ldots, i_{\mu-1}, \cdot, i_{\mu+1}, \ldots, i_d)$$

- In the previous algorithm

$$\mathbf{U}^{(\mu)}(i_1, \ldots, i_{\mu-1}, \cdot, i_{\mu+1}, \ldots, i_d) = \mathrm{e}^{tA_\mu} \mathbf{U}^{(\mu-1)}(i_1, \ldots, i_{\mu-1}, \cdot, i_{\mu+1}, \ldots, i_d)$$

is the action of e^{tA_μ} on the μ-fibers of $\mathbf{U}^{(\mu-1)}$

Tensor algebra formulation: μ-mode product

Let $L \in \mathbb{C}^{m \times n_\mu}$ be a matrix. Then the μ-mode product of L with $\mathbf{U}$, denoted by $\mathbf{S} = \mathbf{U} \times_\mu L$, is the tensor $\mathbf{S} \in \mathbb{C}^{n_1 \times \cdots \times n_{\mu-1} \times m \times n_{\mu+1} \times \cdots \times n_d}$ obtained by multiplying the matrix L onto the μ-fibers of $\mathbf{U}$, that is

$$\mathbf{S}(i_1, \cdots, i_{\mu-1}, i, i_{\mu+1}, \cdots, i_d) = \sum_{j=1}^{n_\mu} L_{ij} \mathbf{U}(i_1, \cdots, i_{\mu-1}, j, i_{\mu+1}, \cdots, i_d),$$

where $1 \leq i \leq m$

Tensor algebra formulation

- According to this definition, $\mathbf{U}(t) = \mathbf{U}^{(d)}$ is the result of d consecutive μ-mode products with the matrices e^{tA_μ}, $1 \le \mu \le d$, starting on $\mathbf{U}^{(0)} = \mathbf{U}_0$

Tensor algebra formulation

- According to this definition, $\mathbf{U}(t) = \mathbf{U}^{(d)}$ is the result of d consecutive μ-mode products with the matrices e^{tA_μ}, $1 \leq \mu \leq d$, starting on $\mathbf{U}^{(0)} = \mathbf{U}_0$
- Therefore, we can rewrite the integrator as

$$\mathbf{U}(t) = \mathbf{U}_0 \times_1 \mathrm{e}^{tA_1} \times_2 \cdots \times_d \mathrm{e}^{tA_d}$$

Tensor algebra formulation

- According to this definition, $\mathbf{U}(t) = \mathbf{U}^{(d)}$ is the result of d consecutive μ-mode products with the matrices e^{tA_μ}, $1 \le \mu \le d$, starting on $\mathbf{U}^{(0)} = \mathbf{U}_0$
- Therefore, we can rewrite the integrator as

$$\mathbf{U}(t) = \mathbf{U}_0 \times_1 \mathrm{e}^{tA_1} \times_2 \cdots \times_d \mathrm{e}^{tA_d}$$

- This is the reason why we call the proposed method the μ-mode integrator.

Computational cost

- The computation of $\mathbf{U}_0 \times_1 \mathrm{e}^{tA_1} \times_2 \cdots \times_d \mathrm{e}^{tA_d}$ requires the computation of d small matrix exponentials of sizes $n_1 \times n_1, \ldots, n_d \times n_d$

Computational cost

- The computation of $\mathbf{U}_0 \times_1 e^{tA_1} \times_2 \cdots \times_d e^{tA_d}$ requires the computation of d small matrix exponentials of sizes $n_1 \times n_1, \ldots, n_d \times n_d$
- Then, the main component of the final cost is the computation of matrix-matrix products of size $n_\mu \times n_\mu$ times $n_\mu \times (n_1 \cdots n_{\mu-1} n_{\mu+1} \cdots n_d)$, which is $\mathcal{O}(Nn_\mu)$, with $N = n_1 \cdots n_d$ the total number of degrees of freedom

Computational cost

- The computation of $\mathbf{U}_0 \times_1 \mathrm{e}^{tA_1} \times_2 \cdots \times_d \mathrm{e}^{tA_d}$ requires the computation of d small matrix exponentials of sizes $n_1 \times n_1, \ldots, n_d \times n_d$
- Then, the main component of the final cost is the computation of matrix-matrix products of size $n_\mu \times n_\mu$ times $n_\mu \times (n_1 \cdots n_{\mu-1} n_{\mu+1} \cdots n_d)$, which is $\mathcal{O}(N n_\mu)$, with $N = n_1 \cdots n_d$ the total number of degrees of freedom
- Matrix-matrix products as required for the μ-mode integrator can be performed very efficiently on modern computer systems (level-3 BLAS operation)

Alternative approaches

- Different space discretizazion (diagonal spectral matrices)
- Solution of $\mathbf{u}'(t) = M\mathbf{u}(t)$ by directly computing the action of the matrix exponential e^{tM} on the vector $\mathbf{u}_0$
- This is possible using iterative schemes such as Krylov projection, Taylor series, or polynomial interpolation techniques, as the involved matrix M is large and sparse
- It is difficult to predict the cost of such algorithms, as the number (or the cost) of iterations highly depends on the norm and other properties of the matrix
- Efficient/parallel sparse matrix-vector products may depend on the storage format

μ-mode integrator as building block

- The μ-mode integrator is exact for linear problems with time-invariant coefficients in Kronecker form: linear diffusion-advection-absorption equations or linear Schrödinger equations with a potential in Kronecker form

μ-mode integrator as building block

- The μ-mode integrator is exact for linear problems with time-invariant coefficients in Kronecker form: linear diffusion-advection-absorption equations or linear Schrödinger equations with a potential in Kronecker form
- The scheme can also be used as a building block for solving nonlinear PDEs. For example in the context of
  - Exponential integrators
  - Splitting methods

Numerical experiments (MATLAB)

- The μ-mode integrator is implemented in MATLAB in any dimension d, using a clever combination of `reshape`, `mtimes`, and `permute` functions

Numerical experiments (MATLAB)

- The μ-mode integrator is implemented in MATLAB in any dimension d, using a clever combination of `reshape`, `mtimes`, and `permute` functions
- We test the proposed integrator either against the following iterative schemes:
  - `expmv`: polynomial method based on Taylor approximation [Al-Mohy–Higham 2011]
  - `phipm`: Krylov method with full orthogonalization [Niesen–Wright 2012]
  - `kiops`: Krylov method with incomplete orthogonalization [Gaudrealt–Rainwater–Tokman 2018]

  or against FFT based techniques, depending on the problem

Numerical experiments (MATLAB)

- The μ-mode integrator is implemented in MATLAB in any dimension d, using a clever combination of reshape, mtimes, and permute functions
- We test the proposed integrator either against the following iterative schemes:
  - expmv: polynomial method based on Taylor approximation [Al-Mohy–Higham 2011]
  - phipm: Krylov method with full orthogonalization [Niesen–Wright 2012]
  - kiops: Krylov method with incomplete orthogonalization [Gaudrealt–Rainwater–Tokman 2018]

  or against FFT based techniques, depending on the problem
- As a measure of cost, we consider the computational time

Introductory test: liner heat equation

- We consider

$$\begin{cases} \partial_t u(t,\mathbf{x}) = \Delta u(t,\mathbf{x}), & \mathbf{x} \in [0,2\pi)^3, \quad t \in [0,T], \\ u(0,\mathbf{x}) = \cos x_1 + \cos x_2 + \cos x_3, \end{cases}$$

with periodic boundary conditions

Introductory test: liner heat equation

- We consider

$$\begin{cases} \partial_t u(t,\mathbf{x}) = \Delta u(t,\mathbf{x}), & \mathbf{x} \in [0,2\pi)^3, \quad t \in [0,T], \\ u(0,\mathbf{x}) = \cos x_1 + \cos x_2 + \cos x_3, \end{cases}$$

  with periodic boundary conditions
- After space discretization (finite differences) we have

$$\mathbf{u}'(t) = M\mathbf{u}(t),$$

  with $M = I_3 \otimes I_2 \otimes A_1 + I_3 \otimes A_2 \otimes I_1 + A_3 \otimes I_2 \otimes I_1$, $A_\mu \in \mathbb{R}^{n\times n}$

Introductory test: liner heat equation

- We consider

$$\begin{cases} \partial_t u(t,\mathbf{x}) = \Delta u(t,\mathbf{x}), & \mathbf{x} \in [0,2\pi)^3, \quad t \in [0,T], \\ u(0,\mathbf{x}) = \cos x_1 + \cos x_2 + \cos x_3, & \end{cases}$$

  with periodic boundary conditions
- After space discretization (finite differences) we have

$$\mathbf{u}'(t) = M\mathbf{u}(t),$$

  with $M = I_3 \otimes I_2 \otimes A_1 + I_3 \otimes A_2 \otimes I_1 + A_3 \otimes I_2 \otimes I_1$, $A_\mu \in \mathbb{R}^{n \times n}$
- Solution:

$$\mathbf{u}(t) = \mathrm{e}^{tM}\mathbf{u}(0) \iff \mathbf{U}(t) = \mathbf{U}(0) \times_1 \mathrm{e}^{tA_1} \times_2 \mathrm{e}^{tA_2} \times_3 \mathrm{e}^{tA_3}$$

Heat equation: results

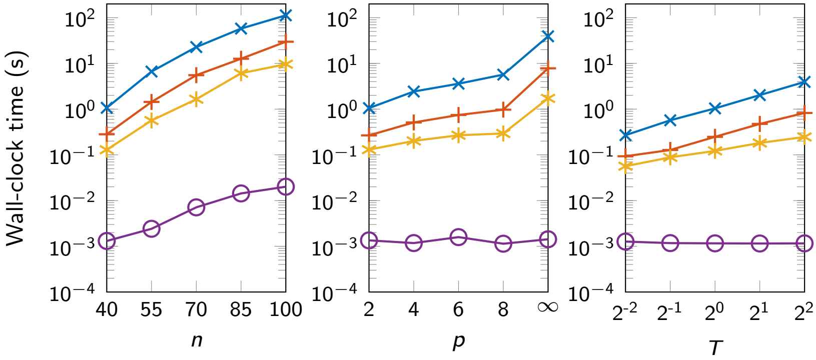

Figure: Wall-clock time as a function of n (left), of the order of the finite difference scheme p (middle), and of the final time T (right). Blue line `expmv`, red line `phipm`, orange line `kiops` and purple line μ-mode integrator.

Heat equation: partial CPU times

n	40	55	70	85	100
`expm`*	0.52	0.71	1.37	3.15	3.54
μ-mode products	0.79	1.71	5.74	10.92	16.89
Total	1.31	2.42	7.11	14.07	20.43

Table: Breakdown of wall-clock time (in ms) for the μ-mode integrator for different values of n.

* We used `MATLAB` `expm` (Padé with scaling and squaring), but other methods are possibile.

Schrödinger equation with time dependent potential

- We consider

$$\begin{cases} \partial_t \psi(t,\mathbf{x}) = H(t,\mathbf{x})\psi(t,\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^3, \quad t \in [0,1] \\ \psi(0,\mathbf{x}) = 2^{-\frac{5}{2}}\pi^{-\frac{3}{4}}(x_1 + \mathrm{i}x_2)\exp\left(-x_1^2/4 - x_2^2/4 - x_3^2/4\right), \\ \psi(t,\infty) = 0 \end{cases}$$

  where the Hamiltonian is given by

$$H(\mathbf{x},t) = \frac{\mathrm{i}}{2}\left(\Delta - x_1^2 - x_2^2 - x_3^2 - 2x_3\sin^2 t\right)$$

- Comparison between:
  - TSFMP: Domain truncation, period b.c., splitting in time, FFT for the Laplacian part, Magnus (order 2) for the potential part
  - HKMP: Hermite pseudospectral approach, Magnus (order 2) in time.

Schrödinger equation with time dependent potential

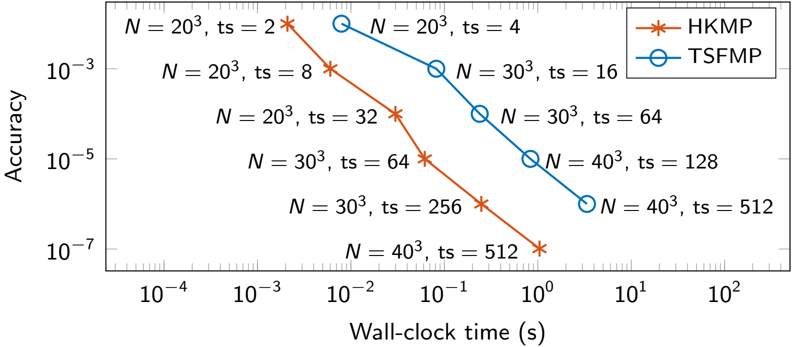

Figure: Integration of the Schrödinger equation up to $T = 1$. The ref. solution has been computed by the HKMP method with d.o.f. $N = 100^3$ and ts $= 2048$.

Nonlinear Schrödinger/Gross–Pitaevskii equation

- We consider

$$\partial_t \psi(t,\mathbf{x}) = \frac{\mathrm{i}}{2}\Delta\psi(t,\mathbf{x}) + \frac{\mathrm{i}}{2}\left(1 - |\psi(t,\mathbf{x})|^2\right)\psi(t,\mathbf{x}),$$

with $\mathbf{x} \in \mathbb{R}^3$, $t \in [0, 25]$, and initial condition constituted by the superimposition of two straight vortices in a background density $|\psi_\infty|^2 = 1$

Nonlinear Schrödinger/Gross–Pitaevskii equation

- We consider

$$\partial_t \psi(t,\mathbf{x}) = \frac{\mathrm{i}}{2}\Delta\psi(t,\mathbf{x}) + \frac{\mathrm{i}}{2}\left(1 - |\psi(t,\mathbf{x})|^2\right)\psi(t,\mathbf{x}),$$

  with $\mathbf{x} \in \mathbb{R}^3$, $t \in [0,25]$, and initial condition constituted by the superimposition of two straight vortices in a background density $|\psi_\infty|^2 = 1$
- Discretization with TSFD method, truncating the unbounded domain to $\mathbf{x} \in [-20,20]^3$ and using nonuniform finite differences with homogeneous Neumann boundary conditions

Nonlinear Schrödinger/Gross–Pitaevskii equation

- We consider

$$\partial_t \psi(t, \mathbf{x}) = \frac{\mathrm{i}}{2}\Delta\psi(t, \mathbf{x}) + \frac{\mathrm{i}}{2}\left(1 - |\psi(t, \mathbf{x})|^2\right)\psi(t, \mathbf{x}),$$

  with $\mathbf{x} \in \mathbb{R}^3$, $t \in [0, 25]$, and initial condition constituted by the superimposition of two straight vortices in a background density $|\psi_\infty|^2 = 1$
- Discretization with TSFD method, truncating the unbounded domain to $\mathbf{x} \in [-20, 20]^3$ and using nonuniform finite differences with homogeneous Neumann boundary conditions
- Comparison between:
  - Iterative methods for the matrix exponential
  - μ-mode integrator

Nonlinear Schrödinger/Gross–Pitaevskii equation

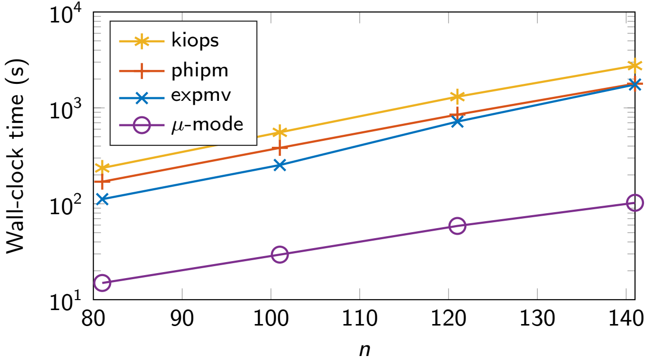

Figure: Wall-clock time for the integration of the Schrödinger equation up to $T = 25$ as a function of n. A constant time step size $\tau = 0.1$ is employed.

Conclusions and further research

- The proposed μ-mode integrator can make use of level-3 BLAS operations: this is good in MATLAB and modern hardware (thanks to batched GEMM routines)

Conclusions and further research

- The proposed μ-mode integrator can make use of level-3 BLAS operations: this is good in MATLAB and modern hardware (thanks to batched GEMM routines)
- For problems from quantum mechanics the approach can outperform well-established integrators in the literature by a significant margin

Conclusions and further research

- The proposed μ-mode integrator can make use of level-3 BLAS operations: this is good in MATLAB and modern hardware (thanks to batched GEMM routines)
- For problems from quantum mechanics the approach can outperform well-established integrators in the literature by a significant margin
- μ-mode products can be used to efficiently compute arbitrary d-dimensional spectral transforms, when no fast transform is available (not shown)

Conclusions and further research

- The proposed μ-mode integrator can make use of level-3 BLAS operations: this is good in MATLAB and modern hardware (thanks to batched GEMM routines)
- For problems from quantum mechanics the approach can outperform well-established integrators in the literature by a significant margin
- μ-mode products can be used to efficiently compute arbitrary d-dimensional spectral transforms, when no fast transform is available (not shown)
- Extension to φ functions for exponential integrators

μ-mode products to compute spectral transforms

If a function $f(\mathbf{x})$ can be expanded into a series $\sum_{\mathbf{i}} f_{\mathbf{i}}\phi_{\mathbf{i}}(\mathbf{x})$, then

$$f_{\mathbf{i}} = \int_{R_1\times\cdots\times R_d} f(\mathbf{x})\overline{\phi_{\mathbf{i}}}(\mathbf{x})d\mathbf{x} \Rightarrow f_{\mathbf{i}} \approx \hat{f}_{\mathbf{i}} = \sum_{\boldsymbol{\ell}<\mathbf{m}} f(\mathbf{x}_{\boldsymbol{\ell}})\overline{\phi_{\mathbf{i}}}(\mathbf{x}_{\boldsymbol{\ell}})w_{\boldsymbol{\ell}}, \quad \mathbf{i}<\mathbf{k}.$$

Define the matrices $\Phi_\mu \in \mathbb{C}^{k_\mu\times m_\mu}$, $1\le\mu\le d$, with components

$$(\Phi_\mu)_{i\ell} = \overline{\phi_i^\mu}(X_\ell^\mu), \quad \mathbf{x}_{\boldsymbol{\ell}} = (X_{\ell_1}^1,\cdots,X_{\ell_d}^d)\in\mathbb{R}^d$$

and denote by $\mathbf{F_W}\in\mathbb{C}^{m_1\times\cdots\times m_d}$ the tensor with elements $f(\mathbf{x}_{\boldsymbol{\ell}})w_{\boldsymbol{\ell}}$ and by $\hat{\mathbf{F}}\in\mathbb{C}^{k_1\times\cdots\times k_d}$ the tensor with elements $\hat{f}_{\mathbf{i}}$. Then

$$\hat{\mathbf{F}} = \mathbf{F_W}\times_1\Phi_1\times_2\cdots\times_d\Phi_d. \tag{1}$$

Schrödinger equation with time independent potential

- We consider

$$\begin{cases} \mathrm{i}\partial_t\psi(t,\mathbf{x}) = -\dfrac{1}{2}\Delta\psi(t,\mathbf{x}) + V(\mathbf{x})\psi(t,\mathbf{x}), & \mathbf{x}\in\mathbb{R}^3, t\in[0,1] \\ \psi(0,\mathbf{x}) = 2^{-\frac{5}{2}}\pi^{-\frac{3}{4}}(x_1+\mathrm{i}x_2)\exp\left(-x_1^2/4 - x_2^2/4 - x_3^2/4\right) \end{cases}$$

  with potential $V(\mathbf{x}) = V_1(x_1) + V_2(x_2) + V_3(x_3)$, where

$$V_1(x_1) = \cos(2\pi x_1), \quad V_2(x_2) = x_2^2/2, \quad V_3(x_3) = x_3^2/2$$

- Comparison between:
  - TSFP: Splitting in time, FFT for the Laplacian part
  - HKP: Hermite pseudospectral approach, exactness in time

Schrödinger equation with time independent potential

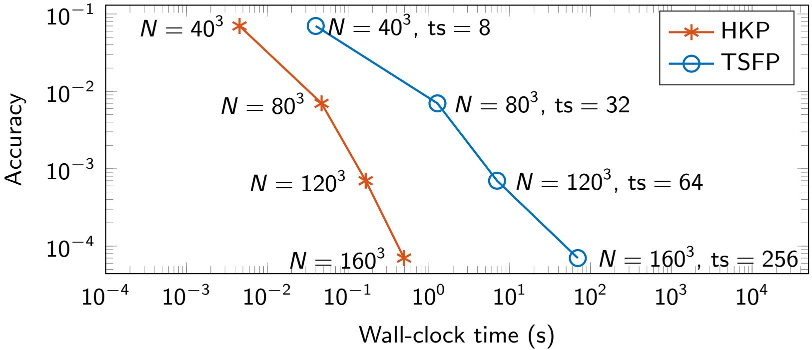

Figure: Integration of the Schrödinger equation up to $T = 1$. The ref. solution has been computed by the HKP method with d.o.f. $N = 300^3$.

Numerical experiments (CPU/GPU)

- The μ-mode integrator is implemented in C++ for the CPU and uses CUDA for the GPU
- In both cases, μ-mode products are computed directly on the multi-dimensional arrays stored in memory using appropriate batched gemm routines
- We measure the performance improvements that we obtain from performing computations on the GPU with different precisions and problem sizes
- As for the numerical experiments in `MATLAB`, the measure of cost is the wall-clock time needed to solve numerically the differential equation under consideration up to a fixed final time.

Heat equation

- We consider

$$\begin{cases} \partial_t u(t,\mathbf{x}) = \Delta u(t,\mathbf{x}), & \mathbf{x} \in [0,2\pi)^3, \quad t \in [0,1], \\ u(0,\mathbf{x}) = \cos x_1 + \cos x_2 + \cos x_3 \end{cases}$$

  with periodic boundary conditions;
- Space discretization: second order centered finite differences with n^3 d.o.f.;
- Time discretization: μ-mode integrator.

Heat equation

n	exp	double			single			half
		MKL	GPU	speedup	MKL	GPU	speedup	GPU
200	2.92	38.39	2.66	14.4x	19.48	1.33	14.6x	0.39
300	4.88	136.17	8.90	15.3x	81.65	5.27	15.5x	2.73
400	10.14	310.11	29.88	10.4x	161.97	16.89	9.6x	6.68
500	17.74	711.07	52.86	13.5x	373.36	30.51	12.2x	15.43

Table: Wall-clock time (in ms) for the heat equation. The speedup is the ratio between the single step performed in MKL and GPU, in double and single precision.

Heat equation (denormal)

- We consider

$$\begin{cases} \partial_t u(t,\mathbf{x}) = \Delta u(t,\mathbf{x}), \quad \mathbf{x} \in \left[-\frac{11}{4}, \frac{11}{4}\right]^3, \quad t \in [0,1], \\ u(0,\mathbf{x}) = \left(x_1^4 + x_2^4 + x_3^4\right) \exp\left(-x_1^4 - x_2^4 - x_3^4\right) \end{cases}$$

  with (artificial) Dirichlet boundary conditions;
- Space discretization: second order centered finite differences with n^3 d.o.f.;
- Time discretization: μ-mode integrator.

Heat equation (denormal)

n	exp	double		single		scaled single	half
		MKL	GPU	MKL	GPU	MKL	GPU
200	2.92	38.80	2.64	92.19	1.34	19.98	0.38
300	6.01	157.41	8.87	385.84	5.22	71.24	2.71
400	13.40	314.96	29.85	1059.78	16.86	154.84	6.67
500	30.19	702.48	52.92	2567.56	30.42	367.34	13.44

Table: Wall-clock time for the heat equation (denormal). The performance degradation of Intel MKL due to denormal numbers disappears when using the scaling workaround (scaled single).

Schrödinger equation with time independent potential

- We consider

$$\begin{cases} \mathrm{i}\partial_t\psi(t,\mathbf{x}) = -\dfrac{1}{2}\Delta\psi(t,\mathbf{x}) + V(\mathbf{x})\psi(t,\mathbf{x}), & \mathbf{x}\in\mathbb{R}^3, t\in[0,1] \\ \psi(0,\mathbf{x}) = 2^{-\frac{5}{2}}\pi^{-\frac{3}{4}}(x_1+\mathrm{i}x_2)\exp\left(-x_1^2/4 - x_2^2/4 - x_3^2/4\right) \end{cases}$$

with potential $V(\mathbf{x}) = V_1(x_1) + V_2(x_2) + V_3(x_3)$, where

$$V_1(x_1) = \cos(2\pi x_1), \quad V_2(x_2) = x_2^2/2, \quad V_3(x_3) = x_3^2/2$$

- Space discretization: Hermite pseudospectral method;
- Time discretization: μ-mode integrator.

Schrödinger equation with time independent potential

n	double				single			
	exp	MKL	GPU	speedup	exp	MKL	GPU	speedup
127	5.56	20.89	1.27	16.4x	4.71	13.71	0.64	21.4x
255	8.31	224.13	16.02	13.9x	5.16	134.21	8.11	16.5x
511	50.79	3121.42	219.13	14.2x	28.01	1824.93	119.46	15.2x

Table: Wall-clock time (in ms) for the Schrödinger equation. The speedup is the ratio between the single step performed in MKL and GPU, in double and single precision.

Schrödinger equation with time dependent potential

- We consider

$$\begin{cases} \partial_t \psi(t,\mathbf{x}) = H(t,\mathbf{x})\psi(t,\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^3, \quad t \in [0,1] \\ \psi(0,\mathbf{x}) = 2^{-\frac{5}{2}}\pi^{-\frac{3}{4}}(x_1 + \mathrm{i}\, x_2)\exp\left(-x_1^2/4 - x_2^2/4 - x_3^2/4\right), \end{cases}$$

where the Hamiltonian is given by

$$H(\mathbf{x},t) = \frac{\mathrm{i}}{2}\left(\Delta - x_1^2 - x_2^2 - x_3^2 - 2x_3\sin^2 t\right)$$

- Space discretization: Hermite pseudospectral method;
- Time discretization: Order 2 Magnus integrator with μ-mode approach.

Schrödinger equation with time dependent potential

n	double					
	exp (ext)	MKL		GPU		speedup
		exp (int)	μ-mode	exp (int)	μ-mode	
127	0.02	2.56	19.38	0.37	1.05	15.3x
255	0.05	4.52	200.46	0.66	13.79	14.2x
511	0.07	29.71	3043.88	2.38	213.21	14.3x

n	single					
	exp (ext)	MKL		GPU		speedup
		exp (int)	μ-mode	exp (int)	μ-mode	
127	0.01	2.16	12.51	0.25	0.54	18.9x
255	0.03	2.88	100.35	0.34	7.01	13.9x
511	0.05	14.25	1600.86	1.09	108.31	14.8x

Table: Wall-clock time (in ms) for the Schrödinger equation. The speedup is the ratio between the single step performed in MKL and GPU, in double precision (top) and single precision (bottom).

References I

- A.H. Al-Mohy and N.J. Higham, *Computing the action of the matrix exponential with an application to exponential integrators*, SIAM J. Sci. Comput. **33** (2011), no. 2, 488–511.
- M. Caliari, F. Cassini, L. Einkemmer, A. Ostermann, and F. Zivcovich, *A μ-mode integrator for solving evolution equations in Kronecker form*, Submitted to Journal of Computational Physics (2021).
- M. Caliari and S. Zuccher, *Reliability of the time splitting Fourier method for singular solutions in quantum fluids*, Comput. Phys. Commun. **222** (2018), 46–58.

References II

- S. Gaudreault, G. Rainwater, and M. Tokman, *KIOPS: A fast adaptive Krylov subspace solver for exponential integrators*, J. Comput. Phys. **372** (2018), no. 1, 236–255.
- T.G. Kolda and B.W. Bader, *Tensor decompositions and applications*, SIAM Rev. **51** (2009), no. 3, 455–500.
- J. Niesen and W. M. Wright, *Algorithm 919: A Krylov subspace algorithm for evaluating the φ-functions appearing in exponential integrators*, ACM Trans. Math. Software **38** (2012), no. 3, 1–19.

Thank you for your attention