JASS 2008

St. Petersburg, March 2008

A quantum control algorithm: numerical aspects

Philipp Klenze

Technische Universität München, Germany

This presentation contains slides adapted from the presentation “Numerical Linear Algebra Tasks in a Quantum Control Problem” by Konrad Waldherr.

Gradient Flow Algorithm

One iteration step in the Gradient Flow Algorithm

- Calculate the forward-propagation for all $t_1, t_2, ..., t_k$:

$$\mathbf{U}(t_k) = e^{-i\Delta t \mathbf{H}_k} \cdot e^{-i\Delta t \mathbf{H}_{k-1}} \dots e^{-i\Delta t \mathbf{H}_1}$$

- Compute the backward-propagation for all $t_M, t_{M-1}, \dots, t_k$

$$\mathbf{\Lambda}(t_k) = e^{-i\Delta t \mathbf{H}_k} \cdot e^{-i\Delta t \mathbf{H}_{k+1}} \dots e^{-i\Delta t \mathbf{H}_M}$$

- Calculate the update

$$\frac{\partial h(\mathbf{U}(t_k))}{\partial u_j} = \mathrm{Re}\left\{\mathrm{tr}\left[\mathbf{\Lambda}^{\dagger}(t_k)(-i\mathbf{H}_j)\mathbf{U}(t_k)\right]\right\}$$

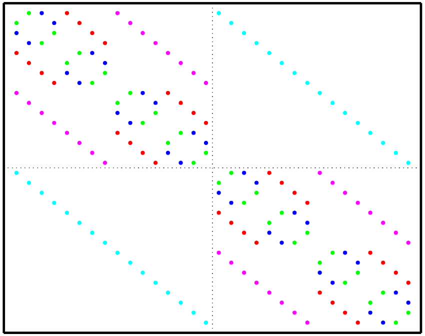
Numerical tasks

- Computation of the matrix exponentials
- Computation of all intermediate products

$$\begin{array}{l}
\mathbf{U}_0 \\
\mathbf{U}_0 \cdot \mathbf{U}_1 \\
\mathbf{U}_0 \cdot \mathbf{U}_1 \cdot \mathbf{U}_2 \\
\vdots \\
\mathbf{U}_0 \cdot \mathbf{U}_1 \cdot \mathbf{U}_2 \cdots \mathbf{U}_M
\end{array}$$

Properties of $\mathbf{H}$

- $\mathbf{H}$ is *sparse*, most entries are zero
- $\mathbf{H}$ is *hermitian*, $\mathbf{H}^\dagger = \mathbf{H}$
- $\mathbf{H}$ is *persymmetric*, symmetric with respect to the north-west-south-east diagonal, $\mathbf{HJ} = \mathbf{JH}^\dagger$
- $\mathbf{H}$ has the following sparsity pattern



Simplifying the problem

- $\mathbf{H}$ can be transformed into a real matrix
- Then, $\mathbf{H}$ can be transformed to two real blocks of half size:

$$\begin{pmatrix} \mathbf{I} & \mathbf{J} \\ \mathbf{I} & -\mathbf{J} \end{pmatrix} \cdot \mathbf{H} \cdot \begin{pmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{J} & -\mathbf{J} \end{pmatrix} = \begin{pmatrix} \mathbf{A}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_2 \end{pmatrix}$$

Ideas to calculate the matrix exponential

- The problem: compute $e^{\mathbf{A}} := \sum_{k=0}^{\infty} \frac{\mathbf{A}^k}{k!}$

Ideas to calculate the matrix exponential

- The problem: compute $e^{\mathbf{A}} := \sum_{k=0}^{\infty} \frac{\mathbf{A}^k}{k!}$
- Diagonalise $\mathbf{A}$ (eigendecomposition)

Ideas to calculate the matrix exponential

- The problem: compute $e^{\mathbf{A}} := \sum_{k=0}^{\infty} \frac{\mathbf{A}^k}{k!}$
- Diagonalise $\mathbf{A}$ (eigendecomposition)
- Approximate the exponential function

Ideas to calculate the matrix exponential

- The problem: compute $e^{\mathbf{A}} := \sum_{k=0}^{\infty} \frac{\mathbf{A}^k}{k!}$
- Diagonalise $\mathbf{A}$ (eigendecomposition)
- Approximate the exponential function
  - with polynomials

Ideas to calculate the matrix exponential

- The problem: compute $e^{\mathbf{A}} := \sum_{k=0}^{\infty} \frac{\mathbf{A}^k}{k!}$
- Diagonalise $\mathbf{A}$ (eigendecomposition)
- Approximate the exponential function
    - with polynomials
        - TAYLOR series

Ideas to calculate the matrix exponential

- The problem: compute $e^{\mathbf{A}} := \sum_{k=0}^{\infty} \frac{\mathbf{A}^k}{k!}$
- Diagonalise $\mathbf{A}$ (eigendecomposition)
- Approximate the exponential function
  - with polynomials
    - TAYLOR series
    - CHEBYSHEV series expansion

Ideas to calculate the matrix exponential

- The problem: compute $e^{\mathbf{A}} := \sum_{k=0}^{\infty} \frac{\mathbf{A}^k}{k!}$
- Diagonalise $\mathbf{A}$ (eigendecomposition)
- Approximate the exponential function
  - with polynomials
    - TAYLOR series
    - CHEBYSHEV series expansion
  - with rational functions
    - PADÉ approximation

Eigendecomposition

- In the case of a diagonal matrix

$$\mathbf{A} = \operatorname{diag}(d_1, \ldots, d_n) = \begin{pmatrix} d_1 & & \\ & \ddots & \\ & & d_n \end{pmatrix}$$

it holds

$$e^{\mathbf{A}} = \operatorname{diag}(e^{d_1}, \ldots, e^{d_n}) = \begin{pmatrix} e^{d_1} & & \\ & \ddots & \\ & & e^{d_n} \end{pmatrix}$$

- If $\mathbf{A} = \mathbf{SDS}^{-1} = \mathbf{S}\left(\operatorname{diag}(d_1, \ldots, d_n)\right)\mathbf{S}^{-1}$ it follows

$$e^{\mathbf{A}} = S\left(\operatorname{diag}(e^{d_1}, \ldots, e^{d_n})\right)S^{-1}$$

- **Expensive part:** Computation of the eigendecomposition

Scaling and Squaring

- Some approximations work much better if the norm of the matrix $\mathbf{A}$ is not to big
- $\left(e^{\mathbf{A}/m}\right)^m = e^{\mathbf{A}}$

Scaling and Squaring

- Some approximations work much better if the norm of the matrix $\mathbf{A}$ is not to big
- $\left(e^{\mathbf{A}/m}\right)^m = e^{\mathbf{A}}$
- **Scaling & Squaring**

$$e^{\mathbf{A}} = \left(e^{\mathbf{A}/2^k}\right)^{2^k}$$

- We scale our matrix by a factor of $\frac{1}{2^k}$
- Then, we compute the approximation
- In the end, we square the approximation k times
- This is not very expensive
- Additional error

Taylor series

- Idea: use a partial sum of the Taylor series

$$e^{\mathbf{A}} \approx S_m(\mathbf{A}) := \sum_{k=0}^{m} \frac{\mathbf{A}^k}{k!}$$

Taylor series

- Idea: use a partial sum of the Taylor series

$$e^{\mathbf{A}} \approx S_m(\mathbf{A}) := \sum_{k=0}^{m} \frac{\mathbf{A}^k}{k!}$$

- The error estimate depends on the norm of $\mathbf{A}$
  $\rightarrow$ Scaling & Squaring

Taylor series

- Idea: use a partial sum of the Taylor series

$$e^{\mathbf{A}} \approx S_m(\mathbf{A}) := \sum_{k=0}^{m} \frac{\mathbf{A}^k}{k!}$$

- The error estimate depends on the norm of $\mathbf{A}$
  $\longrightarrow$ Scaling & Squaring
- Convergence is slow

Taylor series

- Idea: use a partial sum of the Taylor series

$$e^{\mathbf{A}} \approx S_m(\mathbf{A}) := \sum_{k=0}^{m} \frac{\mathbf{A}^k}{k!}$$

- The error estimate depends on the norm of $\mathbf{A}$
  $\rightarrow$ Scaling & Squaring
- Convergence is slow
- **Not numerically stable**

Chebyshev series expansion

- A well-behaved function $f : [-1,1] \to \mathbb{C}$ can be approximated by Chebychev polynomials $T_k(x)$:

$$f(x) \approx \frac{a_0}{2} + \sum_{k=1}^{m} a_k T_k(x)$$

with

$$a_k := \frac{2}{\pi} \int_{-1}^{1} f(x) T_k(x) \frac{dx}{\sqrt{1-x^2}}$$

Chebyshev series expansion

- A well-behaved function $f : [-1, 1] \to \mathbb{C}$ can be approximated by Chebychev polynomials $T_k(x)$:

$$f(x) \approx \frac{a_0}{2} + \sum_{k=1}^{m} a_k T_k(x)$$

with

$$a_k := \frac{2}{\pi} \int_{-1}^{1} f(x) T_k(x) \frac{dx}{\sqrt{1 - x^2}}$$

- For the exponential function, a_k decreases as $\frac{1}{2^k k!}$

Chebyshev series expansion

- A well-behaved function $f : [-1,1] \to \mathbb{C}$ can be approximated by Chebychev polynomials $T_k(x)$:

$$f(x) \approx \frac{a_0}{2} + \sum_{k=1}^{m} a_k T_k(x)$$

with

$$a_k := \frac{2}{\pi} \int_{-1}^{1} f(x) T_k(x) \frac{dx}{\sqrt{1-x^2}}$$

- For the exponential function, a_k decreases as $\frac{1}{2^k k!}$
- This works also for matrices if the norm is smaller than one

Chebyshev series expansion

- A well-behaved function $f : [-1, 1] \to \mathbb{C}$ can be approximated by Chebychev polynomials $T_k(x)$:

$$f(x) \approx \frac{a_0}{2} + \sum_{k=1}^{m} a_k T_k(x)$$

with

$$a_k := \frac{2}{\pi} \int_{-1}^{1} f(x) T_k(x) \frac{dx}{\sqrt{1-x^2}}$$

- For the exponential function, a_k decreases as $\frac{1}{2^k k!}$
- This works also for matrices if the norm is smaller than one
- Arbitrary norm
  $\to$ Scaling & Squaring

Padé approximation

- Padé approximation works like Taylor, but using a rational function instead of a polynomial
- For $x \in \mathbb{C}$ the Padé approximation $r_m(x)$ of e^x is given by

$$r_m(x) = \frac{p_m(x)}{q_m(x)}$$

with $p_m(x) = \sum_{j=0}^{m} \frac{(2m-j)!m!}{(2m)!(m-j)!j!} x^j$, $q_m(x) = \sum_{j=0}^{m} \frac{(2m-j)!m!(-1)^j}{(2m)!(m-j)!j!} x^j$

Padé approximation

- Padé approximation works like Taylor, but using a rational function instead of a polynomial
- For $x \in \mathbb{C}$ the Padé approximation $r_m(x)$ of e^x is given by

$$r_m(x) = \frac{p_m(x)}{q_m(x)}$$

with $p_m(x) = \sum_{j=0}^{m} \frac{(2m-j)!m!}{(2m)!(m-j)!j!} x^j$, $q_m(x) = \sum_{j=0}^{m} \frac{(2m-j)!m!(-1)^j}{(2m)!(m-j)!j!} x^j$

- **Generalization to matrices:**

$$e^{\mathbf{A}} \approx r_m(\mathbf{A}) = (q_m(\mathbf{A}))^{-1} p_m(\mathbf{A})$$

- Approximation is only good around 0
  $\rightarrow$ Scaling & Squaring

Padé approximation

- Padé approximation works like Taylor, but using a rational function instead of a polynomial
- For $x \in \mathbb{C}$ the Padé approximation $r_m(x)$ of e^x is given by

$$r_m(x) = \frac{p_m(x)}{q_m(x)}$$

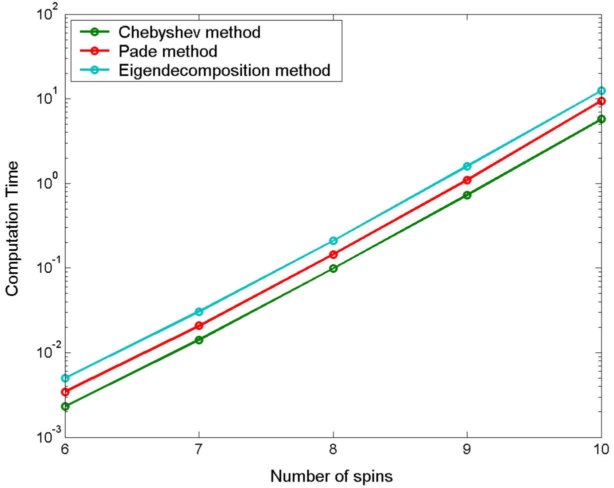
with $p_m(x) = \sum_{j=0}^{m} \frac{(2m-j)!m!}{(2m)!(m-j)!j!} x^j$, $q_m(x) = \sum_{j=0}^{m} \frac{(2m-j)!m!(-1)^j}{(2m)!(m-j)!j!} x^j$

- **Generalization to matrices:**

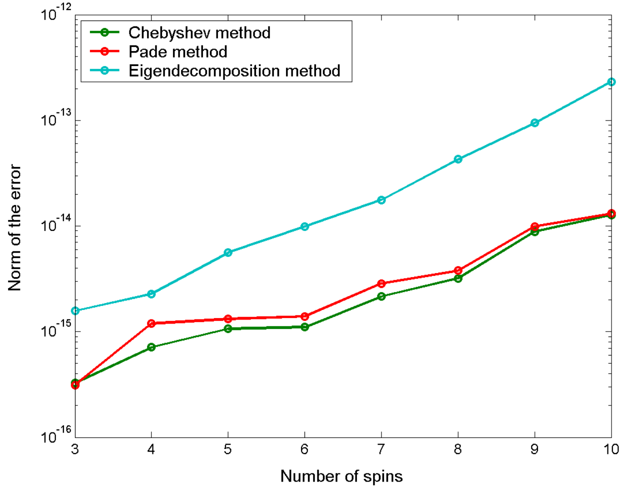
$$e^{\mathbf{A}} \approx r_m(\mathbf{A}) = (q_m(\mathbf{A}))^{-1} p_m(\mathbf{A})$$

- Approximation is only good around 0
  $\rightarrow$ Scaling & Squaring
- **Expensive part:** Computation of the matrix inverse

Comparison of the methods: Computation time



Comparison of the methods: accuracy



Advantages of the Chebyshev series method

- Only the evaluation of a matrix polynomial required:
  $\Longrightarrow$ BLAS-Routines
- Only products of the form dense * sparse appear
- Good convergence properties
- Matrix polynomials of order k can be evaluated with only $O(\sqrt{k})$ matrix-matrix-products
- Theoretically nice approach

Parallel matrix-matrix-multiplication

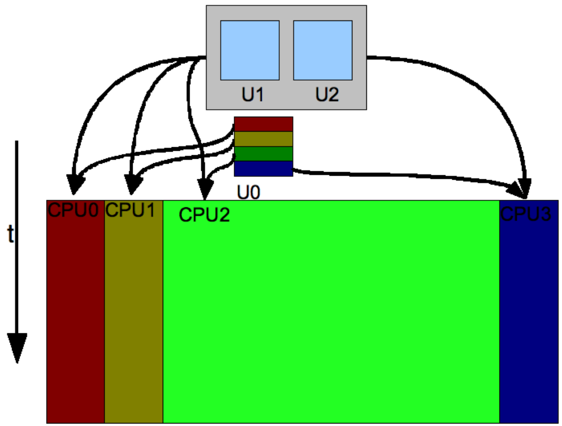
- Numerical task: Compute all intermediate products

$$\begin{array}{l} U_0 \\ U_0 \cdot U_1 \\ U_0 \cdot U_1 \cdot U_2 \\ \vdots \\ U_0 \cdot U_1 \cdot U_2 \cdots U_M \end{array}$$

- Two approaches for a parallel algorithm:
  - slice-wise method
  - tree-like method

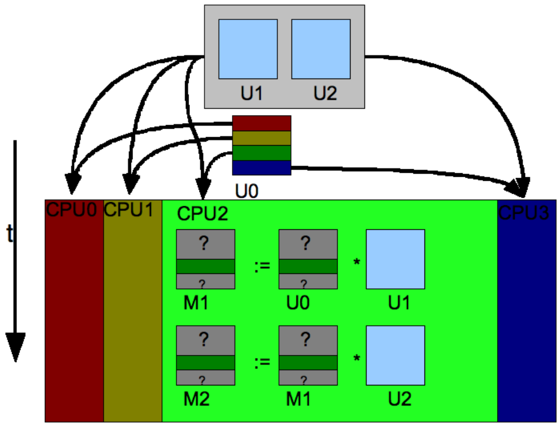
The slice-wise approach

Part 1



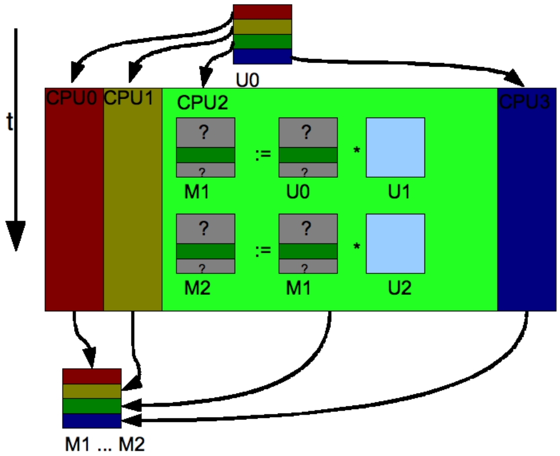
The slice-wise approach

Part 2



The slice-wise approach

Part 3



The slice-wise approach: conclusions

- Broadcast of all matrices $\mathbf{U}_k$ to all processors
- Each processor is responsible for ”its” rows
- No communication during the algorithm required
- Optimal in terms of scalar multiplications
- Broadcasting costs most of the time
- Memory becomes an issue

The tree-like approach

Please look at the whiteboard.

The tree-like approach: conclusions

- “Expanded” binary tree
- Complicated algorithm
- No broadcasting required
- Still much communication
- More multiplications than strictly needed
- “Super Nodes” do quasi-broadcast

a new approach

Please look at the whiteboard.

The best of both worlds: a new approach

- Pipeline
- Litte communication required
- Simple algorithm
- Optimal in terms of total scalar multiplications
- Possibility to parallelize the entire GRAPE algorithm
- Some idle time until the pipeline is filled