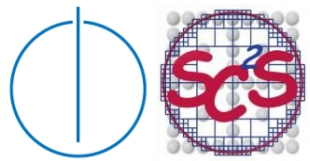
Fast 3D Block Parallelisation for the Matrix Multiplication Prefix Problem

Application in Quantum Control

Garching, December 9th, 2009

Konrad Waldherr

Technische Universität München, Germany



Optimal Control: A mathematical point of view

- **Given:**
  - $W \in \mathbb{C}^{N \times N}$: unitary target matrix,
  - $H_0 \in \mathbb{C}^{N \times N}$: constant and time-independent drift term,
  - $H_j \in \mathbb{C}^{N \times N}$: control matrices,
  - M: number of discretisation points.

Optimal Control: A mathematical point of view

- **Given:**
  - $W \in \mathbb{C}^{N \times N}$: unitary target matrix,
  - $H_0 \in \mathbb{C}^{N \times N}$: constant and time-independent drift term,
  - $H_j \in \mathbb{C}^{N \times N}$: control matrices,
  - M: number of discretisation points.
- **Problem:**
  Find optimal controls, i.e. find $u_j(t_k)$, such that

$$U(t_M) = \underbrace{e^{-i(H_0+\sum_j u_j(t_M)H_j)}}_{e^{-iH_M}} \underbrace{e^{-i(H_0+\sum_j u_j(t_{M-1})H_j)}}_{e^{-iH_{M-1}}} \dots \underbrace{e^{-i(H_0+\sum_j u_j(t_1)H_j)}}_{e^{-iH_1}}$$

  approximates the target W in an optimal way.

Optimal Control: A mathematical point of view

- **Given:**
  - $W \in \mathbb{C}^{N \times N}$: unitary target matrix,
  - $H_0 \in \mathbb{C}^{N \times N}$: constant and time-independent drift term,
  - $H_j \in \mathbb{C}^{N \times N}$: control matrices,
  - M: number of discretisation points.
- **Problem:**
  Find optimal controls, i.e. find $u_j(t_k)$, such that

$$U(t_M) = \underbrace{e^{-i(H_0+\sum_j u_j(t_M)H_j)}}_{e^{-iH_M}} \underbrace{e^{-i(H_0+\sum_j u_j(t_{M-1})H_j)}}_{e^{-iH_{M-1}}} \cdots \underbrace{e^{-i(H_0+\sum_j u_j(t_1)H_j)}}_{e^{-iH_1}}$$

approximates the target W in an optimal way.

Solution of an optimization problem

Optimal Control: A mathematical point of view

Optimization problem:

$$\min_{u_j(t_k)} \|U(t_M) - W\|_F^2 = \min_{u_j(t_k)} \operatorname{tr}\left((U(t_M) - W)^\dagger (U(t_M) - W) \right)$$

$$= \min_{u_j(t_k)} \left(2N - 2\,\mathrm{Re}\left(\operatorname{tr}\left(W^\dagger U(t_M) \right) \right) \right)$$

Optimal Control: A mathematical point of view

Optimization problem:

$$\min_{u_j(t_k)} \|U(t_M) - W\|_F^2 = \min_{u_j(t_k)} \operatorname{tr}\left((U(t_M) - W)^\dagger (U(t_M) - W)\right)$$

$$= \min_{u_j(t_k)} \left(2N - 2\,\mathrm{Re}\left(\operatorname{tr}\left(W^\dagger U(t_M)\right)\right)\right)$$

Maximization problem of the function

$$f(u_j(t_k)) := \mathrm{Re}\left(\operatorname{tr}\left(W^\dagger \overbrace{\underbrace{e^{-i(H_0+\sum_j u_j(t_M)H_j)}}_{U_M} \cdots \underbrace{e^{-i(H_0+\sum_j u_j(t_1)H_j)}}_{U_1}}^{U(t_M)}\right)\right)$$

Optimal Control: A mathematical point of view

Optimization problem:

$$\min_{u_j(t_k)} \|U(t_M) - W\|_F^2 = \min_{u_j(t_k)} \operatorname{tr}\left((U(t_M) - W)^\dagger (U(t_M) - W)\right)$$
$$= \min_{u_j(t_k)} \left(2N - 2\,\mathrm{Re}\left(\operatorname{tr}\left(W^\dagger U(t_M)\right)\right)\right)$$

Maximization problem of the function

$$f(u_j(t_k)) := \mathrm{Re}\left(\operatorname{tr}\left(W^\dagger \overbrace{\underbrace{e^{-i(H_0+\sum_j u_j(t_M)H_j)}}_{U_M} \ldots \underbrace{e^{-i(H_0+\sum_j u_j(t_1)H_j)}}_{U_1}}^{U(t_M)}\right)\right)$$

Partial derivatives:

$$\frac{\partial f(u_j(t_k))}{\partial u_j(t_k)} = \mathrm{Re}\left(\operatorname{tr}\left(\underbrace{W^\dagger U_M^\dagger \ldots U_{k+1}^\dagger}_{W^\dagger(t_{k+1})}(-iH_j)\underbrace{U_k \ldots U_1}_{U(t_k)}\right)\right)$$

The GRAPE algorithm:

1: Define the **quality function** $f(u_j(t_k)) := \mathrm{Re}\ \mathrm{tr}\{W^\dagger U(t_M)\}$

2: Set **initial control amplitudes** $u_j^{(0)}(t_k)$ for all times t_k

3: **for** $\ell = 1, 2, \ldots$

a: Calculate the **forward-propagation** $U(t_k)$ for all $t_1, \ldots, t_M$

$$U(t_k) = e^{-iH_k} e^{-iH_{k-1}} \cdots e^{-iH_1}$$

b: Compute the **back-propagation** $W(t_{k+1})$for all $t_M, t_{M-1}, \ldots, t_1$

$$W(t_{k+1}) = e^{-iH_{k+1}} \cdots e^{-iH_M} W$$

c: Calculate the **gradient**

$$\frac{\partial f}{\partial u_j(t_k)} = \mathrm{Re}\ \mathrm{tr}\left\{W^\dagger(t_{k+1})\,(-iH_j)U(t_k)\right\}$$

d: **Update** the controls $u_j(t_k)$

$$u_j^{(\ell+1)}(t_k) = u_j^{(\ell)}(t_k) + \epsilon \frac{\partial f}{\partial u_j(t_k)}$$

The GRAPE algorithm

Computational tasks:

1. Computation of the matrix exponentials $U_k = e^{H_0 + \sum_j u_j(t_k) H_j}$
2. Computation of the forward propagation $U(t_k) = U_k \ldots U_1$
3. Computation of the back propagation $W(t_k) = U_k \ldots\ldots U_M W$
4. For all j, evaluate the traces $\mathrm{Re}\ \mathrm{tr}\left\{W^\dagger(t_{k+1})\,(-i\,H_j)U(t_k)\right\}$

The GRAPE algorithm

Computational tasks:

1. Computation of the matrix exponentials $U_k = e^{H_0 + \sum_j u_j(t_k) H_j}$
2. Computation of the forward propagation $U(t_k) = U_k \dots U_1$
3. Computation of the back propagation $W(t_k) = U_k \dots\dots U_M W$
4. For all j, evaluate the traces $\mathrm{Re}\ \mathrm{tr}\left\{W^\dagger(t_{k+1})\,(-i\,H_j)U(t_k)\right\}$

Computation and parallelisation of step 1

- Distribute Hamiltonians $H_k = H_0 + \sum_j u_j(t_k) H_j$ uniformly
- Each process computes "its" exponentials $U_k = e^{-iH_k}$
- Computation via Chebyshev series expansion

The GRAPE algorithm

Computational tasks:

1. Computation of the matrix exponentials $U_k = e^{H_0 + \sum_j u_j(t_k) H_j}$
2. Computation of the forward propagation $U(t_k) = U_k \dots U_1$
3. Computation of the back propagation $W(t_k) = U_k \dots\dots U_M W$
4. For all j, evaluate the traces $\mathrm{Re}\ \mathrm{tr}\left\{W^\dagger(t_{k+1})\,(-i\,H_j)U(t_k)\right\}$

Computation and parallelisation of steps 2 and 3

- Enables different strategies for parallelisation
- The computation offers a coarse-grain and a fine-grain approach
- Specifies the distribution of the problems 1. and 4.

The GRAPE algorithm

Computational tasks:

1. Computation of the matrix exponentials $U_k = e^{H_0 + \sum_j u_j(t_k) H_j}$
2. Computation of the forward propagation $U(t_k) = U_k \dots U_1$
3. Computation of the back propagation $W(t_k) = U_k \dots\dots U_M W$
4. For all j, evaluate the traces $\mathrm{Re}\ \mathrm{tr}\left\{W^\dagger(t_{k+1})\,(-i\,H_j)U(t_k)\right\}$

Computation and parallelisation of step 4

- Each process computes “its” part of the gradient
- No possibility for algorithmic improvement

The matrix multiplication prefix problem

- compute all the products

$$U_{1:k} = U_1 \dots U_k$$

The matrix multiplication prefix problem

- compute all the products

$$U_{1:k} = U_1 \dots U_k$$

- Fine-grain approach
  - Parallelise the individual matrix multiplications
  - Compute the matrix products $U_{1:k}$ sequentially
  - Offers 1D, 2D, and 3D approaches

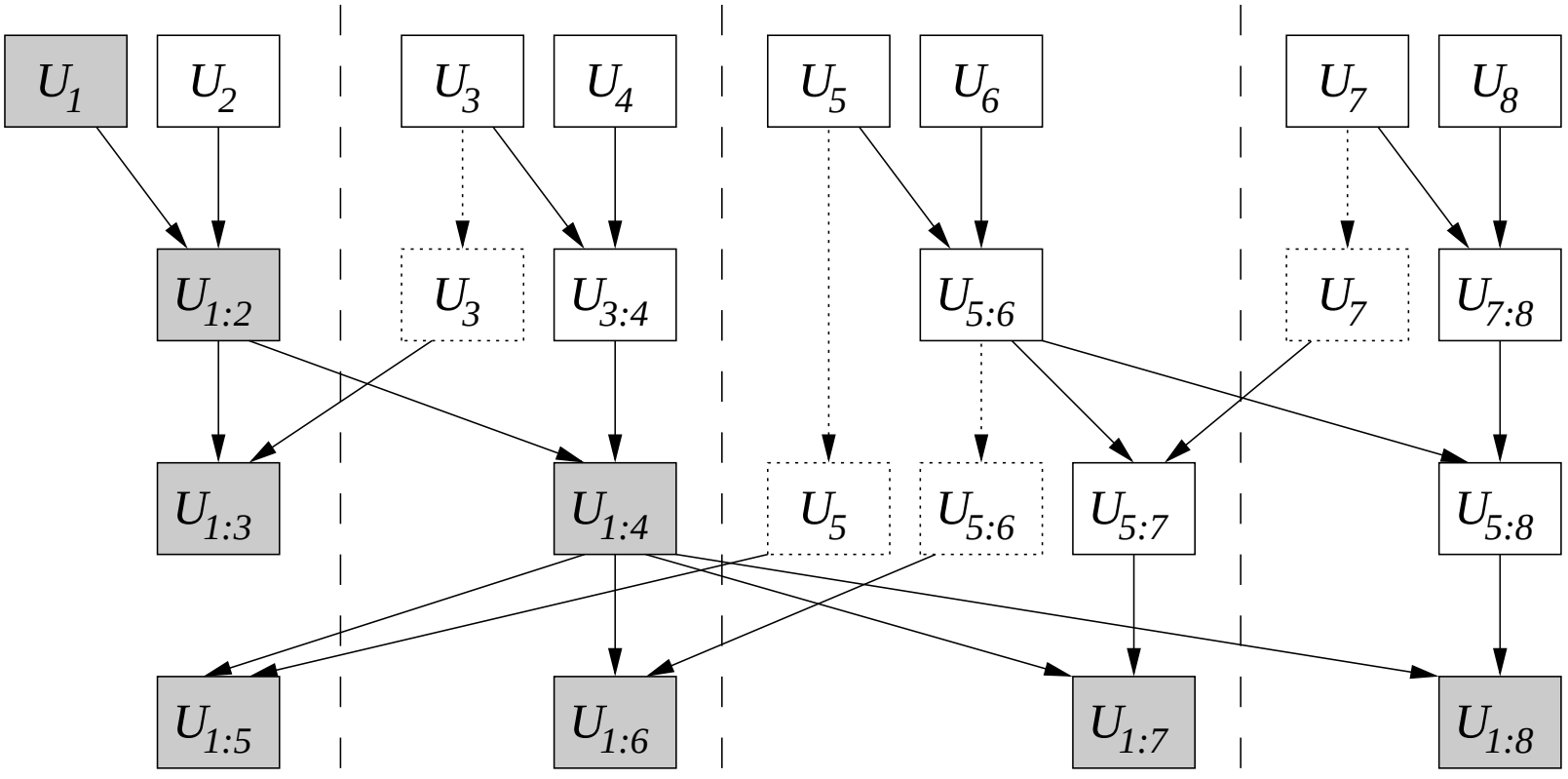
The matrix multiplication prefix problem

- compute all the products

$$U_{1:k} = U_1 \dots U_k$$

- Fine-grain approach
  - Parallelise the individual matrix multiplications
  - Compute the matrix products $U_{1:k}$ sequentially
  - Offers 1D, 2D, and 3D approaches
- Coarse-grain approach
  - Apply a divide-and-conquer approach
  - Leads to a tree-like multiplication scheme
  - The individual matrix multiplications are strictly sequential

Coarse-grain tree-like approach



Coarse-grain tree-like approach

- Advantages:
  - Simple communication pattern
  - Good speedup properties are to be expected

Coarse-grain tree-like approach

- Advantages:
  - Simple communication pattern
  - Good speedup properties are to be expected
- Disadvantages:
  - Increase of the computational work by a logarithmic factor
  - Memory demand is not balanced among the processors
  - Result matrices are not balanced among the processors
  - Limitation to the number of processors $p \leq M/2$

Fine-grain approach

- Parallelise the individual matrix multiplication

1: **for all** $(i,j,k) \in \{1,\ldots,N\} \times \{1,\ldots,N\} \times \{1,\ldots,N\}$ **do**
2: $\quad C_{ik} \leftarrow C_{ik} + A_{ij}B_{jk}$
3: **end for**

Fine-grain approach

- Parallelise the individual matrix multiplication

  1: **for all** $(i,j,k) \in \{1,\ldots,N\} \times \{1,\ldots,N\} \times \{1,\ldots,N\}$ **do**
  2: $\quad C_{ik} \leftarrow C_{ik} + A_{ij}B_{jk}$
  3: **end for**

- $1D$ **algorithm:** Column-wise computation of the result matrix
- $2D$ **algorithm:** “Owner computes”
- $3D$ **algorithm:** Blocking on the three nested main loops; operand and result matrices may be distributed over several processors

Fine-grain approach

- Parallelise the individual matrix multiplication

  1: **for all** $(i,j,k) \in \{1,\ldots,N\} \times \{1,\ldots,N\} \times \{1,\ldots,N\}$ **do**
  2: $\quad C_{ik} \leftarrow C_{ik} + A_{ij}B_{jk}$
  3: **end for**

- $1D$ **algorithm:** Column-wise computation of the result matrix
- $2D$ **algorithm:** "Owner computes"
- $3D$ **algorithm:** Blocking on the three nested main loops; operand and result matrices may be distributed over several processors
- Performance properties:

Algorithm	Communication	Memory	Computational costs
$1D$	$O(N^2 p)$	$O(N^2)$	$O(N^3)$
$2D$	$O(N^2 p^{\frac{1}{2}})$	$O(N^2)$	$O(N^3)$
$3D$	$O(N^2 p^{\frac{1}{3}})$	$O(N^2 p^{\frac{1}{3}})$	$O(N^3)$

$3D$ block matrix multiplication

- Choose D as smallest power of 2, such that $D^3 \geq p$
- Distribute result matrix C into D^2 subblocks $C_{i,k}$
- Each of the D^2 blocks $C_{i,k}$ is computed by p/D^2 processors
- Each processor performs D^3/p block operations for $C_{i,k}$

$3D$ block matrix multiplication

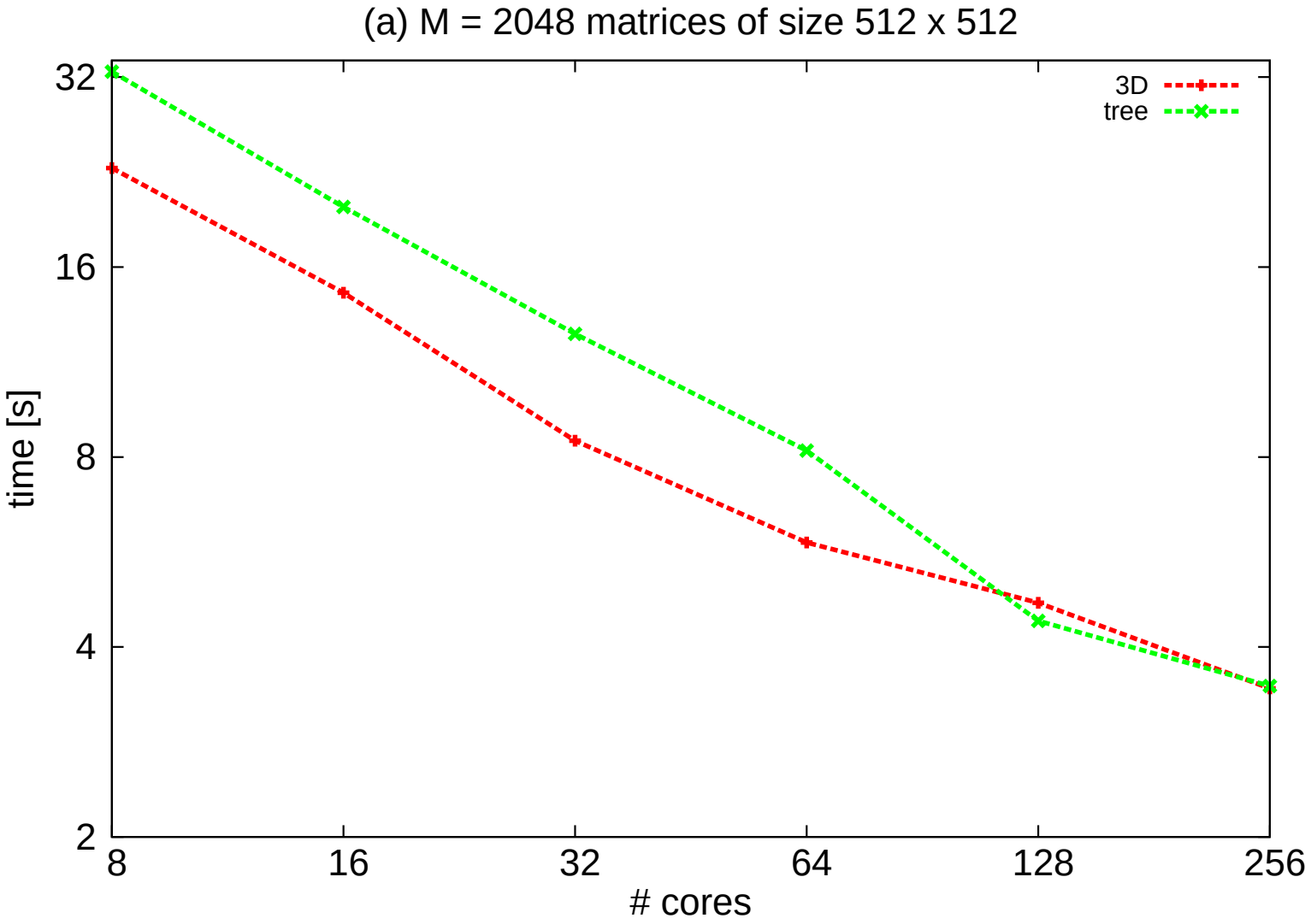
- Choose D as smallest power of 2, such that $D^3 \geq p$
- Distribute result matrix C into D^2 subblocks $C_{i,k}$
- Each of the D^2 blocks $C_{i,k}$ is computed by p/D^2 processors
- Each processor performs D^3/p block operations for $C_{i,k}$
- Each process executes the following two steps:

```
1: for all local block operations (i, j, k) do
2:     fetch A_ij and B_jk from remote processes
3:     C_ik ← C_ik + A_ij B_jk
4: end for
5: accumulate all results for block C_ik
```

- Accumulation is a group-collective operation, organised as pairwise accumulation of data
- Computation and communication within the for loop may be overlapped

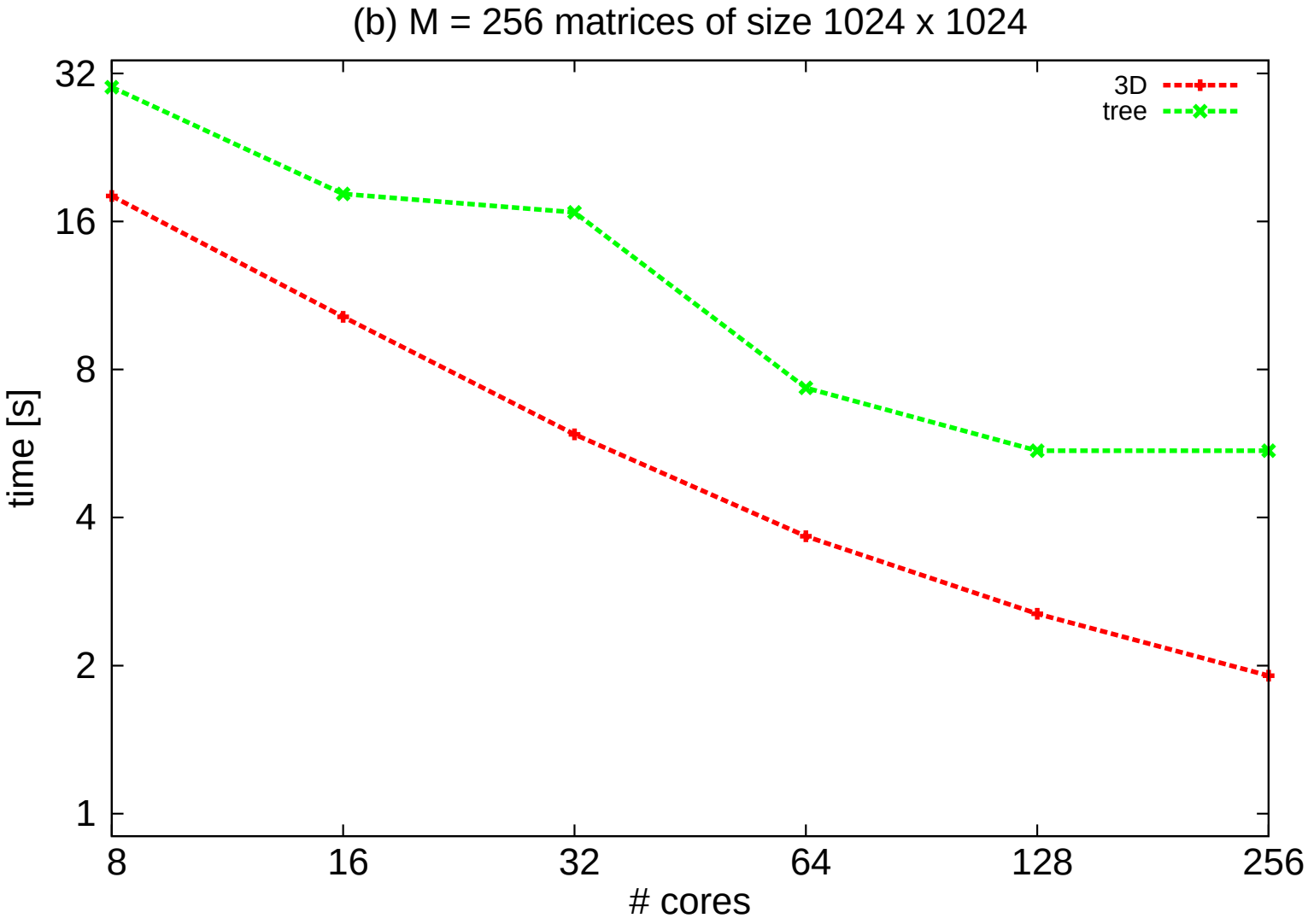
Computation time

Tree-like versus 3D block algorithm ($M = 2048, N = 512$)



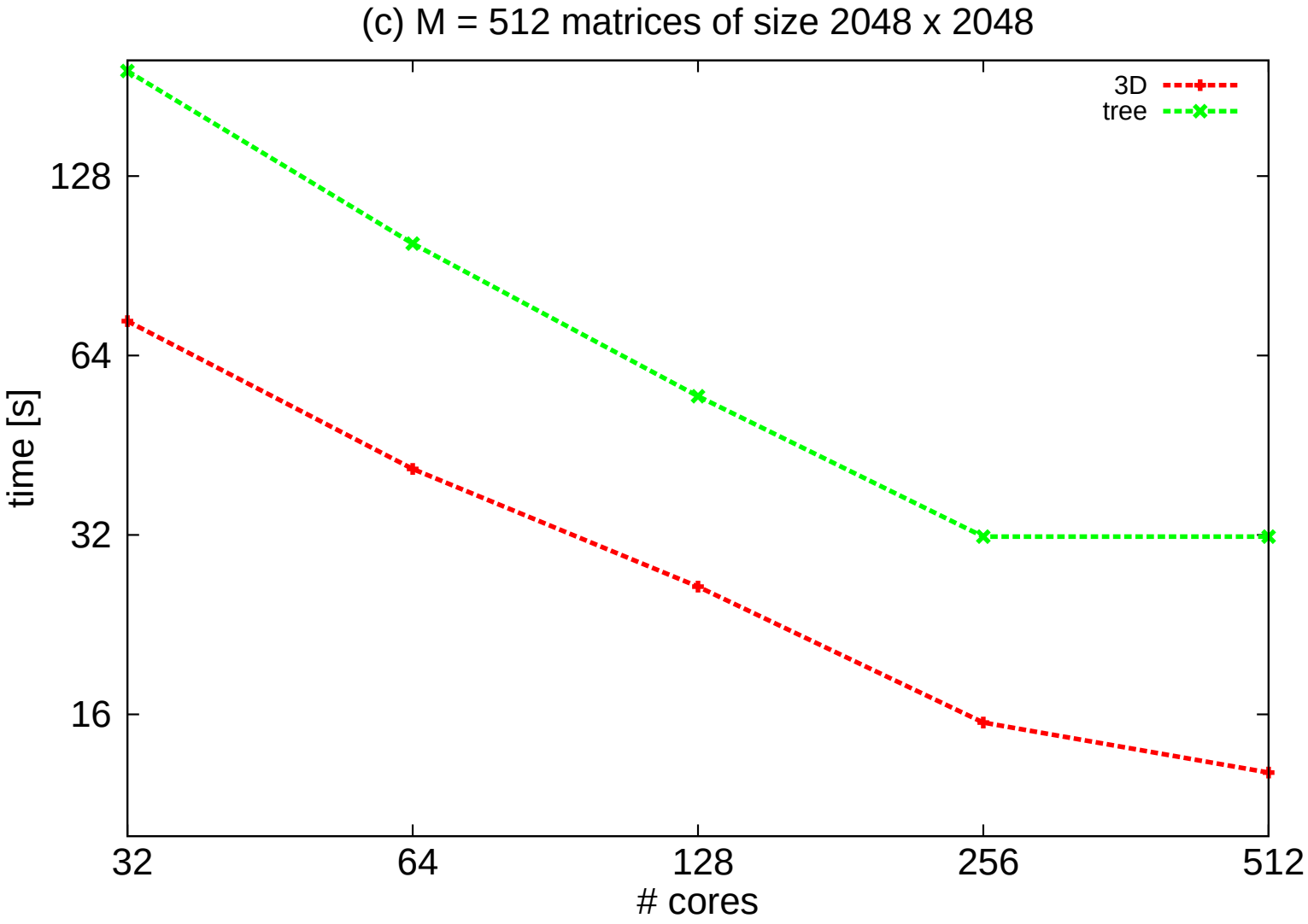
Computation time

Tree-like versus 3D block algorithm ($M = 256, N = 1024$)



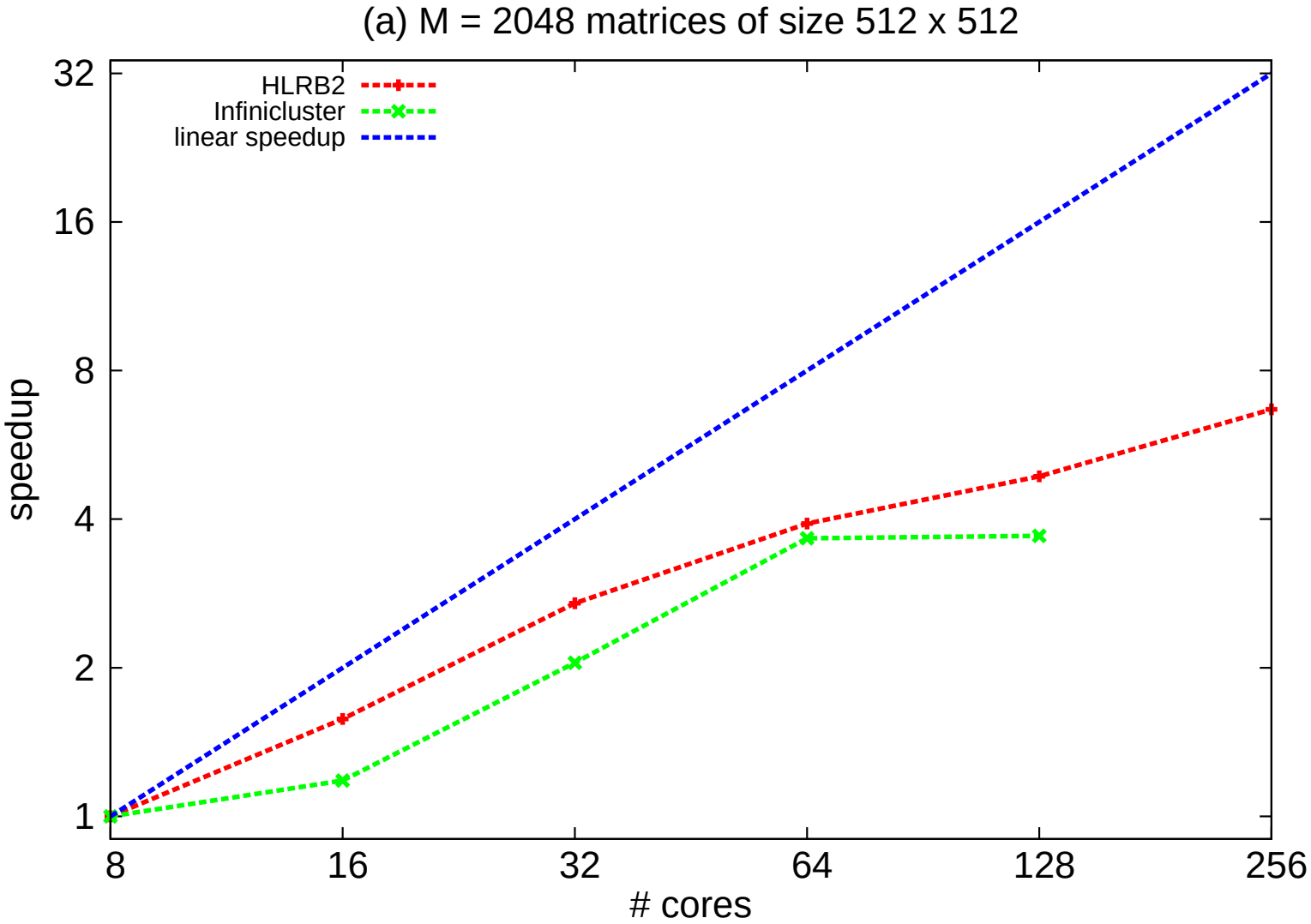
Computation time

Tree-like versus 3D block algorithm ($M = 512, N = 2048$)



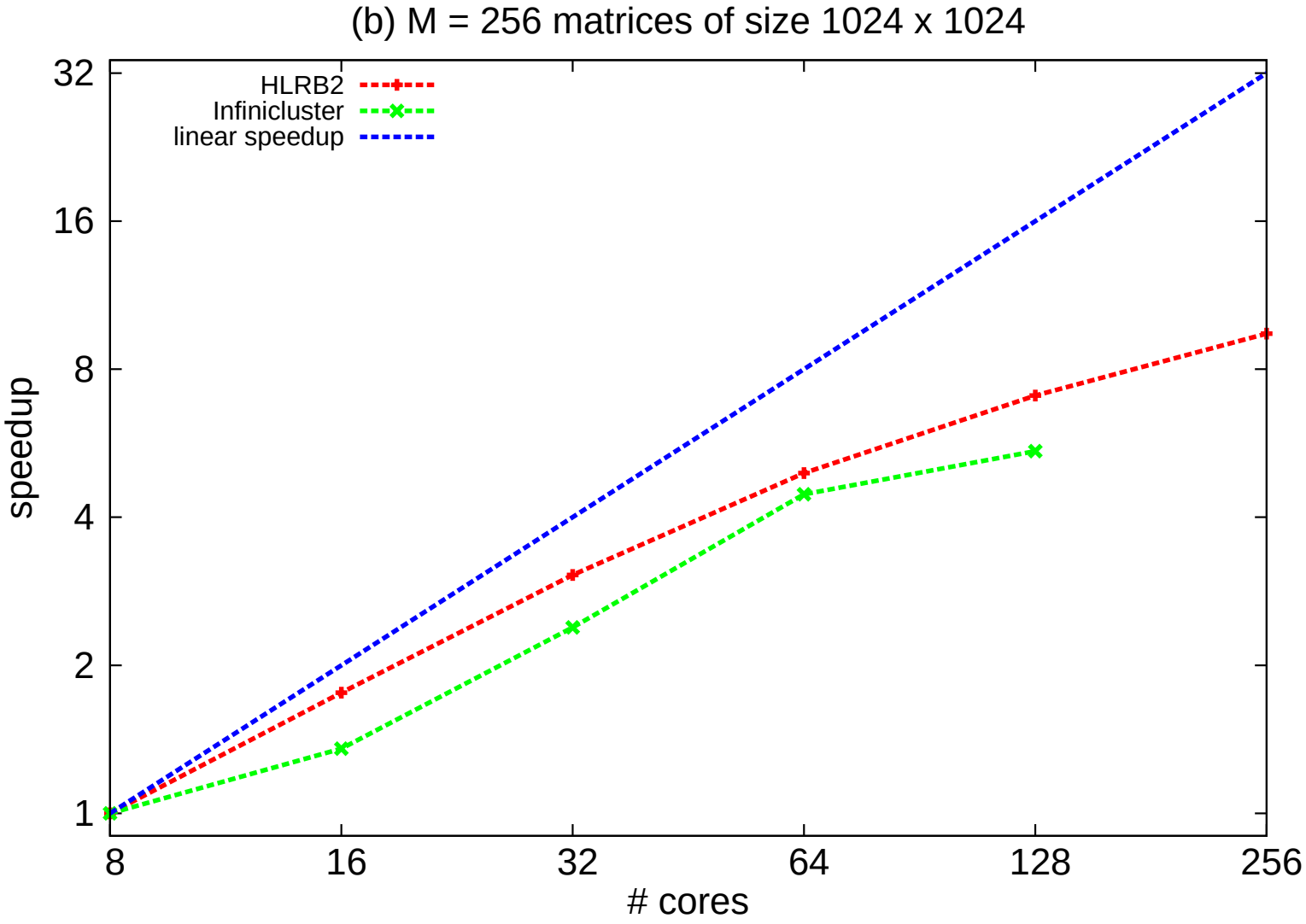
Speedup of the 3D algorithm

Different architectures ($M = 2048, N = 512$)

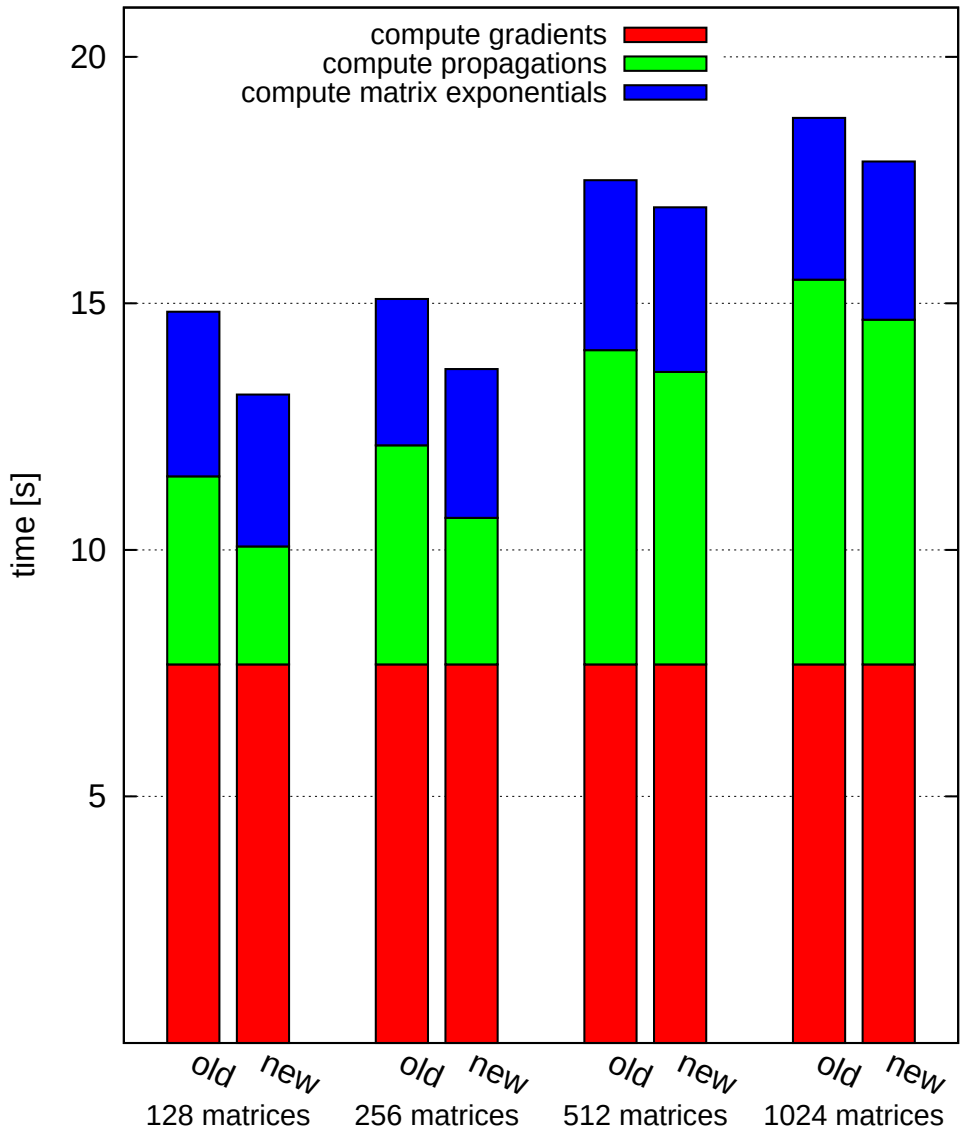


Speedup of the 3D algorithm

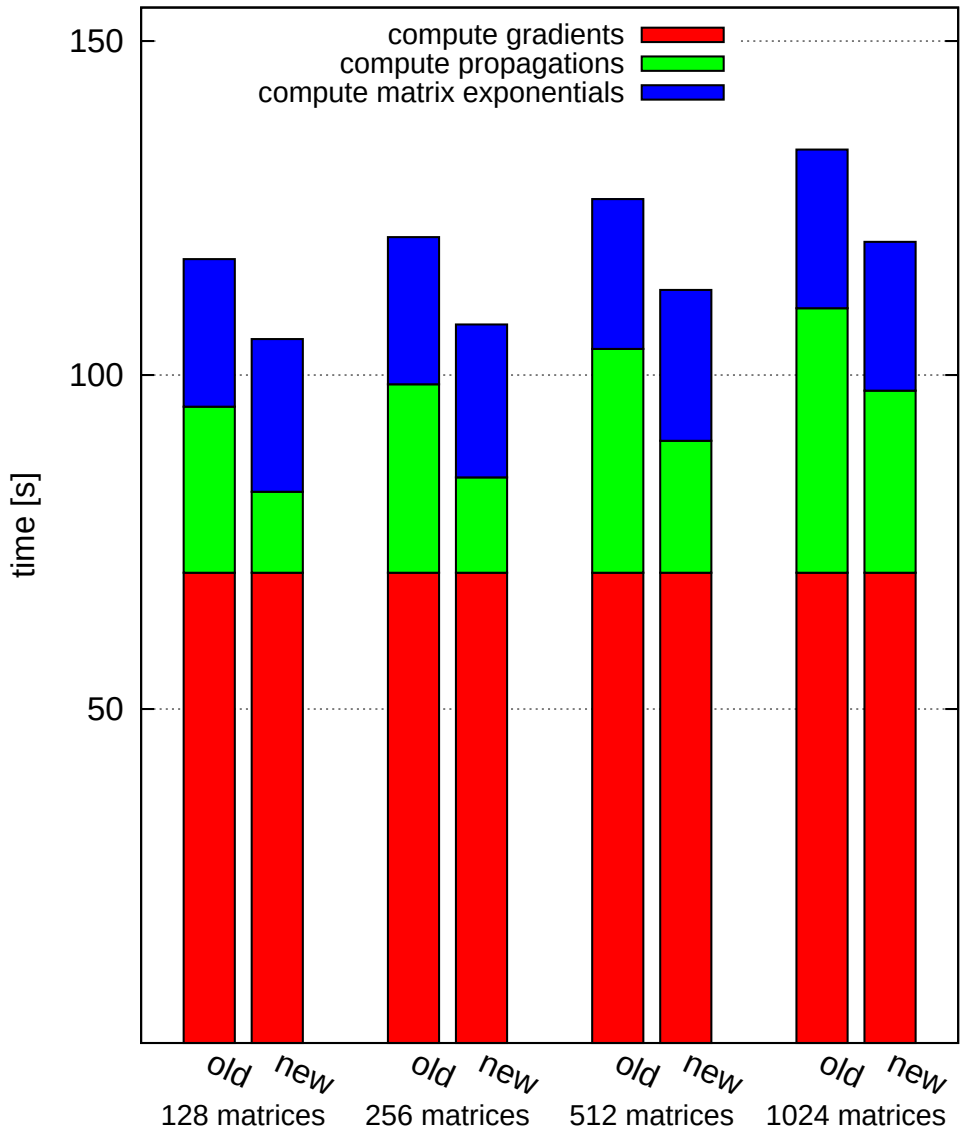
Different architectures ($M = 256, N = 1024$)



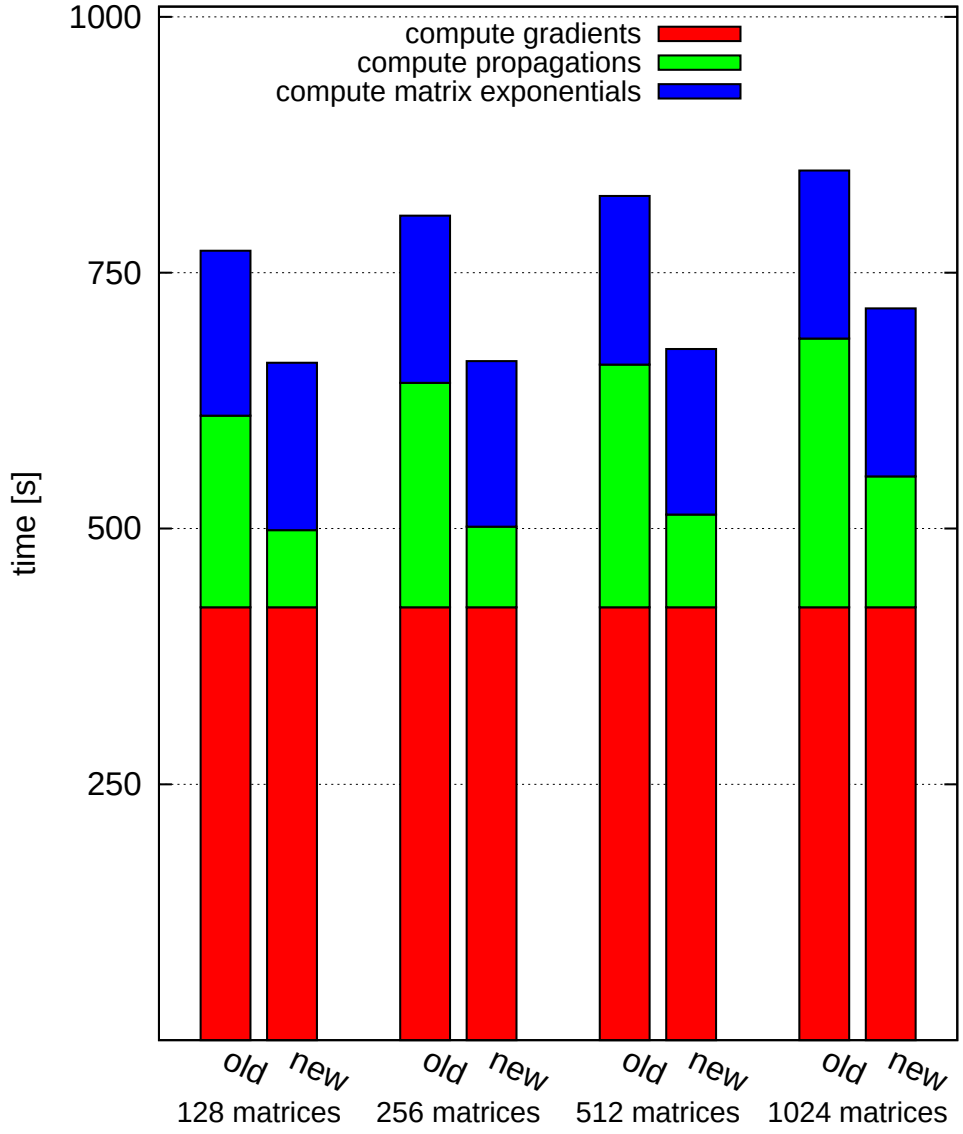
Runtime of the GRAPE algorithm ($N = 512$)



Runtime of the GRAPE algorithm ($N = 1024$)



Runtime of the GRAPE algorithm ($N = 2048$)



Conclusions and Outlook

Conclusions:

- The Grape algorithm for optimal control
- 3D Block Multiplication scheme offers several advantages:
  - Saves logarithmic overhead of the tree-like approach
  - Matrices are balanced among the processors
  - Allows more options for parallelisation
  - Shows good parallel speedup behavior

Conclusions and Outlook

Conclusions:

- The Grape algorithm for optimal control
- 3D Block Multiplication scheme offers several advantages:
  - Saves logarithmic overhead of the tree-like approach
  - Matrices are balanced among the processors
  - Allows more options for parallelisation
  - Shows good parallel speedup behavior

Outlook:

- Quasi-Newton Updates for better convergence (Parallelisation?)
- Sequential versus simultaneous updates (Parallelisation?)