Para 2012, Helsinki
June 12, 2012

A Shared Memory Parallel Implementation of the IRKA Algorithm for $\mathcal{H}_2$ Model Order Reduction

Martin Köhler and Jens Saak

Max Planck Institute for Dynamics of Complex Technical Systems
Computational Methods in Systems and Control Theory

MAX PLANCK INSTITUTE
FOR DYNAMICS OF COMPLEX
TECHNICAL SYSTEMS
MAGDEBURG

Outline

Mathematical Background

Mathematical Background

Linear Time Invariant Systems

Linear Time Invariant System

$$\dot{x}(t) = A\,x(t) + B\,u(t)$$

$$y(t) = C\,x(t)$$

Mathematical Background

Linear Time Invariant Systems

Linear Time Invariant System

$$\dot{x}(t) = A\,x(t) + B\,u(t)$$

$$y(t) = C\,x(t)$$

Matrices:

- *System-Matrix* $A \in \mathbb{R}^{n\times n}$,
- *Input-Matrix* $B \in \mathbb{R}^{n\times m}$,
- *Output-Matrix* $C \in \mathbb{R}^{p\times n}$

Mathematical Background

Linear Time Invariant Systems

Linear Time Invariant System

$$\dot{x}(t) = A\,x(t) + B\,u(t)$$

$$y(t) = C\,x(t)$$

Matrices:

- *System-Matrix* $A \in \mathbb{R}^{n\times n}$,
- *Input-Matrix* $B \in \mathbb{R}^{n\times m}$,
- *Output-Matrix* $C \in \mathbb{R}^{p\times n}$

$p = m = 1 \rightarrow$ System is Single-Input Single-Output (SISO)

Mathematical Background

Linear Time Invariant Systems

Linear Time Invariant System

$$\dot{x}(t) = A\,x(t) + B\,u(t)$$

$$y(t) = C\,x(t)$$

Matrices:

- *System-Matrix* $A \in \mathbb{R}^{n\times n}$,
- *Input-Matrix* $B \in \mathbb{R}^{n\times m}$,
- *Output-Matrix* $C \in \mathbb{R}^{p\times n}$

$p = m = 1 \rightarrow$ System is Single-Input Single-Output (SISO)

Applications

- Development of Integrated Circuits (IC)
- Vibration analysis of mechanical structures
- Biological Processes
- Discretized PDEs
- ...

Mathematical Background

Model Order Reduction

Full Order Model (FOM)

$$\dot{x}(t) = A\,x(t) + B\,u(t)$$

$$y(t) = C\,x(t)$$

Problems

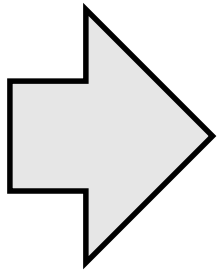
- Current applications lead to large systems orders n, e.g. $n \approx 10^5$ and higher.
- Efficient simulation impossible, computational costs increasing drastically.

Mathematical Background

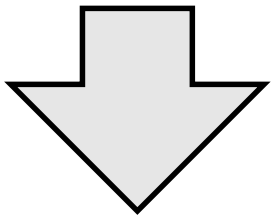
Model Order Reduction

Full Order Model (FOM)

$$\dot{x}(t) = A\,x(t) + B\,u(t)$$
$$y(t) = C\,x(t)$$



Model
Order
Reduction



Reduced Order Model (ROM)

$$\dot{\tilde{x}}(t) = \tilde{A}\,\tilde{x}(t) + \tilde{B}\,u(t)$$
$$\tilde{y}(t) = \tilde{C}\,\tilde{x}(t)$$

Solution

Projection on a reduced subspace of dimension k:

- $\tilde{A} = W^T A V \in \mathbb{R}^{k\times k}$
- $\tilde{B} = W^T B \in \mathbb{R}^{k\times 1}$
- $\tilde{C} = CV \in \mathbb{R}^{1\times k}$

Mathematical Background

Model Order Reduction

Full Order Model (FOM)

$$\dot{x}(t) = A\,x(t) + B\,u(t)$$
$$y(t) = C\,x(t)$$

Model Order Reduction

Question

How to choose $V \in \mathbb{R}^{n\times k}$ and $W \in \mathbb{R}^{n\times k}$?

Goal

FOM and ROM close in a suitable norm, such that $\tilde{y} \approx y$.

Solution

Projection onto reduced subspace of dimension k:

- $\tilde{A} = W^T A V \in \mathbb{R}^{k\times k}$
- $\tilde{B} = W^T B \in \mathbb{R}^{k\times 1}$
- $\tilde{C} = CV \in \mathbb{R}^{1\times k}$

Reduced Order Model (ROM)

$$\dot{\tilde{x}}(t) = \tilde{A}\,\tilde{x}(t) + \tilde{B}\,u(t)$$
$$\tilde{y}(t) = \tilde{C}\,\tilde{x}(t)$$

Mathematical Background

Transfer Function and $\mathcal{H}_2$-Norm

Considering a Linear Time Invariant (LTI) System

$$\Sigma : \begin{cases} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) \end{cases},$$

its **transfer function** representing the input-output mapping in frequency domain is given as:

$$H(s) = C(sI - A)^{-1}B. \tag{1}$$

Mathematical Background

Transfer Function and $\mathcal{H}_2$-Norm

Considering a Linear Time Invariant (LTI) System

$$\Sigma : \begin{cases} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) \end{cases},$$

its **transfer function** representing the input-output mapping in frequency domain is given as:

$$H(s) = C(sI - A)^{-1}B. \tag{1}$$

Employing (1) we define the $\mathcal{H}_2$-**norm** by

$$||H||_{\mathcal{H}_2}^2 =< H, H >_{\mathcal{H}_2} = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \operatorname{tr}\left(H(i\omega)^H H(i\omega)\right)\, \mathrm{d}\omega.$$

Mathematical Background

Transfer Function and $\mathcal{H}_2$-Norm

Considering a Linear Time Invariant (LTI) System

$$\Sigma : \begin{cases} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) \end{cases},$$

its **transfer function** representing the input-output mapping in frequency domain is given as

$$H(s) = C(sI - A)^{-1}B. \tag{1}$$

$\mathcal{H}_2$ MOR:

Solve MOR problem w.r.t. the $\mathcal{H}_2$-norm, i.e.:

$$\max_{t>0} |y(t) - \tilde{y}(t)| \leq \|H - \tilde{H}\|_{\mathcal{H}_2}$$

Employing (1) we define the $\mathcal{H}_2$-**norm** by

$$\|H\|_{\mathcal{H}_2}^2 = < H, H >_{\mathcal{H}_2} = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \operatorname{tr}\left(H(i\omega)^H H(i\omega)\right) \, \mathrm{d}\omega.$$

The IRKA Algorithm

The IRKA Algorithm

Construction of V and W

Idea behind the Iterative Rational Krylov Algorithm (IRKA)

Choose $V \in \mathbb{R}^{n \times k}$ and $W \in \mathbb{R}^{n \times k}$ to interpolate $H(s)$ and its first derivative at $k \ll n$ interpolation points. $\rightarrow$ *Hermite Interpolation*

The IRKA Algorithm

Construction of V and W

Idea behind the Iterative Rational Krylov Algorithm (IRKA)

Choose $V \in \mathbb{R}^{n \times k}$ and $W \in \mathbb{R}^{n \times k}$ to interpolate $H(s)$ and its first derivative at $k \ll n$ interpolation points. $\rightarrow$ *Hermite Interpolation*

Lemma

[Gugercin/Antoulas/Beattie '08]

Suppose $\sigma \in \mathbb{C}$ is not an eigenvalue of either A or $\tilde{A}$:

- If $(\sigma I - A)^{-1} B \in \text{span}\,(V)$ then $H(\sigma) = \tilde{H}(\sigma)$.
- If $(\sigma I - A)^{-H} C^T \in \text{span}\,(W)$ then $H(\sigma) = \tilde{H}(\sigma)$.
- If $(\sigma I - A)^{-H} C^T \in \text{span}\,(W)$ and $(\sigma I - A)^{-1} B \in \text{span}\,(V)$ then $H(\sigma) = \tilde{H}(\sigma)$ and $H'(\sigma) = \tilde{H}'(\sigma)$.

The IRKA Algorithm

Construction of V and W

Corollary [Gugercin/Antoulas/Beattie '08]

If the set $\{\sigma_r\}_{r=1}^{k}$ of interpolation points is closed under complex conjugation the matrices V and W can be chosen to be real.

The IRKA Algorithm

Construction of V and W

Corollary [Gugercin/Antoulas/Beattie '08]

If the set $\{\sigma_r\}_{r=1}^k$ of interpolation points is closed under complex conjugation the matrices V and W can be chosen to be real.

For a complex pair $\sigma_r = \overline{\sigma_{r+1}}$, instead of

$$\left\{(\sigma_r I - A)^{-1} B, (\sigma_{r+1} I - A)^{-1} B\right\} \in \operatorname{span}(V)$$

and

$$\left\{(\sigma_r I - A)^{-H} C^T, (\sigma_{r+1} I - A)^{-H} C^T\right\} \in \operatorname{span}(W),$$

we choose

$$\left\{\operatorname{Re}\left((\sigma_r I - A)^{-1} B\right), \operatorname{Im}\left((\sigma_r I - A)^{-1} B\right)\right\} \in \operatorname{span}(V)$$

and

$$\left\{\operatorname{Re}\left((\sigma_r I - A)^{-H} C^T\right), \operatorname{Im}\left((\sigma_r I - A)^{-H} C^T\right)\right\} \in \operatorname{span}(W).$$

The IRKA Algorithm

Construction of V and W

Corollary [Gugercin/Antoulas/Beattie '08]

If the set $\{\sigma_r\}_{r=1}^k$ of interpolation points is closed under complex conjugation the matrices V and W can be chosen to be real.

For a complex pair $\sigma_r = \overline{\sigma_{r+1}}$, instead of

$$\{(\sigma_r I - A)^{-1}B, (\sigma_{r+1} I - A)^{-1}B\} \in \overline{\text{span}\,(V)}$$

and

$$\{(\sigma_r I - A)^{-H}C^T, (\sigma_{r+1} I - A)^{-H}C^T\} \in \overline{\text{span}\,(W)},$$

We should guarantee $\{\sigma_r\}_{r=1}^k = \overline{\{\sigma_r\}_{r=1}^k}$

$\rightarrow$ Save the solution of two complex linear systems if $\sigma_r \in \mathbb{C}$.

we choose

$$\left\{\text{Re}\left((\sigma_r I - A)^{-1}B\right), \text{Im}\left((\sigma_r I - A)^{-1}B\right)\right\} \in \text{span}\,(V)$$

and

$$\left\{\text{Re}\left((\sigma_r I - A)^{-H}C^T\right), \text{Im}\left((\sigma_r I - A)^{-H}C^T\right)\right\} \in \text{span}\,(W)\,.$$

The IRKA Algorithm

Optimality

The set of stable dynamical systems of order k is not convex.

The IRKA Algorithm

Optimality

The set of stable dynamical systems of order k is not convex.

We employ a *local minimizer* for which it holds:

$$\forall \varepsilon > 0 : \quad ||H - \tilde{H}||_{\mathcal{H}_2} \leq ||H - \tilde{H}^{(\varepsilon)}||_{\mathcal{H}_2}$$

for a transfer function $\tilde{H}^{(\varepsilon)}$ of a stable k-th order LTI system fulfilling

$$||\tilde{H} - \tilde{H}^{(\varepsilon)}||_{\mathcal{H}_2} \leq K\varepsilon.$$

The IRKA Algorithm

Optimality

The set of stable dynamical systems of order k is not convex.

We employ a *local minimizer* for which it holds:

$$\forall \varepsilon > 0: \quad ||H - \tilde{H}||_{\mathcal{H}_2} \leq ||H - \tilde{H}^{(\varepsilon)}||_{\mathcal{H}_2}$$

for a transfer function $\tilde{H}^{(\varepsilon)}$ of a stable k-th order LTI system fulfilling

$$||\tilde{H} - \tilde{H}^{(\varepsilon)}||_{\mathcal{H}_2} \leq K\varepsilon.$$

If $\tilde{H}$ is a local minimizer to H and $\tilde{H}$ has simple poles then

$$< H - \tilde{H}, \tilde{H} \cdot G_1 + G_2 >= 0$$

holds for all realsystems G_1 and G_2 with the same poles as $\tilde{H}$.

The IRKA Algorithm

Optimality

The set of stable dynamical systems of order k is not convex.

We employ a *local minimizer* for which it holds:

$$\forall \varepsilon > 0: \quad ||H - \tilde{H}||_{\mathcal{H}_2} \leq ||H - \tilde{H}^{(\varepsilon)}||_{\mathcal{H}_2}$$

for a transfer function $\tilde{H}^{(\varepsilon)}$ of a stable k-th order LTI system fulfilling

$$||\tilde{H} - \tilde{H}^{(\varepsilon)}||_{\mathcal{H}_2} \leq K\varepsilon.$$

If $\tilde{H}$ is a local minimizer and $< H - \tilde{H}, \tilde{H} \cdot G_1 + G_2 >= 0$ holds, then

$$H(-\sigma_r) = \tilde{H}(-\sigma_r) \text{ and } H'(-\sigma_r) = \tilde{H}'(-\sigma_r)$$

holds for all poles σ_r of $\tilde{H}$.

The IRKA Algorithm

Optimality

The set of stable dynamical systems of order k is not convex.

We employ a *local minimizer* for which it holds:

$$\forall \varepsilon > 0: \quad ||H - \tilde{H}||_{\mathcal{H}_2} \leq ||H - \tilde{H}^{(\varepsilon)}||_{\mathcal{H}_2}$$

for a transfer function $\tilde{H}^{(\varepsilon)}$ of a stable k-th order LTI system fulfilling

$$||\tilde{H} - \tilde{H}^{(\varepsilon)}||_{\mathcal{H}_2} \leq K\varepsilon.$$

If $\tilde{H}$ is a local minimizer and $< H - \tilde{H}, \tilde{H} \cdot G_1 + G_2 >= 0$ holds, then

$$H(-\sigma_r) = \tilde{H}(-\sigma_r) \text{ and } H'(-\sigma_r) = \tilde{H}'(-\sigma_r)$$

holds for all poles σ_r of $\tilde{H}$.

$\tilde{H}$ interpolates H and its first derivative in $-\sigma_1, \ldots, -\sigma_k$.

Parallel Implementation of IRKA

Parallel Implementation of IRKA

Basic Algorithm

Idea: Use a fixed-point like iteration scheme to compute a local minimizer from an initial set of interpolation points.

[Gugercin/Antoulas/Beattie '08]

Parallel Implementation of IRKA

Basic Algorithm

Idea: Use a fixed-point like iteration scheme to compute a local minimizer from an initial set of interpolation points.

[Gugercin/Antoulas/Beattie '08]

Input: $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times 1}$, $C \in \mathbb{R}^{1 \times n}$, admissible $\sigma = \{\sigma_r\}_{r=1}^{k}$
Output: $\tilde{A} \in \mathbb{R}^{k \times k}$, $\tilde{B} \in \mathbb{R}^{k \times 1}$, $\tilde{C} \in \mathbb{R}^{1 \times k}$ as local minimizer.

1: **while** not converged **do**
2: Compute $\operatorname{span}(V) = \operatorname{span}\left((\sigma_1 I - A)^{-1} B, \ldots, (\sigma_r I - A)^{-1} B\right)$
Compute $\operatorname{span}(W) = \operatorname{span}\left((\sigma_1 I - A)^{-H} C^T, \ldots, (\sigma_r I - A)^{-H} C^T\right)$
3: Find biorthonormal bases of V and W, i.e. $W^T V = I$
4: $\tilde{A} := W^T A V$
5: Compute the Eigenvalues $\lambda := \{\lambda_i\}_{i=1}^{r}$ of $\tilde{A}$
6: $\sigma := -\lambda$
7: **end while**
8: $\tilde{A} := W^T A V$, $\tilde{B} := W^T B$ and $\tilde{C} := C V$

Parallel Implementation of IRKA

Basic Algorithm

Idea: Use a fixed-point like iteration scheme to compute a local minimizer from an initial set of interpolation points.

[Gugercin/Antoulas/Beattie '08]

Input: $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times 1}$, $C \in \mathbb{R}^{1 \times n}$, admissible $\sigma = \{\sigma_r\}_{r=1}^{k}$

Output: $\tilde{A} \in \mathbb{R}^{k \times k}$, $\tilde{B} \in \mathbb{R}^{k \times 1}$, $\tilde{C} \in \mathbb{R}^{1 \times k}$ as local minimizer.

1: **while** not converged **do**

2: Compute span $(V) = \text{span}\left((\sigma_1 I - A)^{-1} B, \ldots, (\sigma_r I - A)^{-1} B\right)$
Compute span $(W) = \text{span}\left((\sigma_1 I - A)^{-H} C^T, \ldots, (\sigma_r I - A)^{-H} C^T\right)$

3: Find biorthonormal bases of V and W, i.e. $W^T V = I$

4: $\tilde{A} := W^T A V$

5: Compute the Eigenvalues $\lambda := \{\lambda_i\}_{i=1}^{r}$ of $\tilde{A}$

6: $\sigma := -\lambda$

7: **end while**

8: $\tilde{A}$ ... $\tilde{C} := CV$

Most expensive operations

Process them in parallel ensuring V and W are real.

Parallel Implementation of IRKA

Basic Algorithm

[K./Saak '09-'12]

Step 2 is computed in two stages:

Parallel Implementation of IRKA

Basic Algorithm

[K./Saak '09-'12]

Step 2 is computed in two stages:

1. Compute

$$L_r U_r = (\sigma_r I - A) \quad \forall r \in S,$$

with $S = \{s \in \{1, \ldots, k\} | \sigma_s \in \mathbb{R} \text{ or } \sigma_{s-1} \neq \overline{\sigma_s}\}$

Parallel Implementation of IRKA

Basic Algorithm

[K./Saak '09-'12]

Step 2 is computed in two stages:

1. Compute

$$L_r U_r = (\sigma_r I - A) \quad \forall r \in S,$$

with $S = \{s \in \{1, \ldots, k\} | \sigma_s \in \mathbb{R} \text{ or } \sigma_{s-1} \neq \overline{\sigma_s}\}$

$\rightarrow$ Memory efficient parallelization available.

Parallel Implementation of IRKA

Basic Algorithm

[K./Saak '09-'12]

Step 2 is computed in two stages:

1. Compute
$$L_r U_r = (\sigma_r I - A) \quad \forall r \in S,$$
with $S = \{s \in \{1, \ldots, k\} | \sigma_s \in \mathbb{R} \text{ or } \sigma_{s-1} \neq \overline{\sigma_s}\}$
$\rightarrow$ Memory efficient parallelization available.
2. If $\sigma_r \in \mathbb{R}$ solve
$$L_r U_r V(:, r) = B \text{ and } U_r^T L_r^T W(:, r) = C^T,$$
otherwise compute
$$L_r U_r \tilde{V} = B \text{ and } U_r^H L_r^H \tilde{W} = C^T,$$
$$V(:, r : r+1) = \left[\mathrm{Re}\,(\tilde{V}), \mathrm{Im}\,(\tilde{V})\right],$$
$$W(:, r : r+1) = \left[\mathrm{Re}\,(\tilde{W}), \mathrm{Im}\,(\tilde{W})\right].$$
and skip σ_{r+1}.

Parallel Implementation of IRKA

Basic Algorithm

[K./Saak '09-'12]

Step 2 is computed in two stages:

1. Compute
$$L_r U_r = (\sigma_r I - A) \quad \forall r \in S,$$
with $S = \{s \in \{1, \ldots, k\} | \sigma_s \in \mathbb{R} \text{ or } \sigma_{s-1} \neq \overline{\sigma_s}\}$
$\rightarrow$ Memory efficient parallelization available.
2. If $\sigma_r \in \mathbb{R}$ solve

Skipping an index in a `for`-loop is not allowed in OpenMP `parallel for`.

otherwise compute
$$L_r U_r \tilde{V} = B \text{ and } U_r^H L_r^H \tilde{W} = C^T,$$
$$V(:, r:r+1) = \left[\mathrm{Re}(\tilde{V}), \mathrm{Im}(\tilde{V})\right],$$
$$W(:, r:r+1) = \left[\mathrm{Re}(\tilde{W}), \mathrm{Im}(\tilde{W})\right].$$
and skip σ_{r+1}.

Parallel Implementation of IRKA

Using OpenMP Sections

Idea 1: Iterate over $r = 1, \ldots, k$, check if $\sigma_r \in \mathbb{R}$ and compute $V(:, r)$ and $W(:, r)$ in parallel. $\rightarrow$ Determine the set S implicitly.

Parallel Implementation of IRKA

Using OpenMP Sections

Idea 1: Iterate over $r = 1, \ldots, k$, check if $\sigma_r \in \mathbb{R}$ and compute $V(:, r)$ and $W(:, r)$ in parallel. $\rightarrow$ Determine the set S implicitly.

1: Compute $L_r U_r = (\sigma_r I - A)$,e.g., Using [K., Saak '09]
2: **for** $r = 1, \ldots, k$ **do**
3: **if** $\sigma_r \in \mathbb{R}$ **then**
4: `#pragma omp section`
5: $L_r U_r V(:, r) = B$
6: `#pragma omp section`
7: $U_r^T L_r^T W(:, r) = C^T$
8: **else**
9: `#pragma omp section`
10: $L_r U_r \tilde{V} = B, \quad V(:, r : r+1) = \left[\mathrm{Re}\,(\tilde{V}), \mathrm{Im}\,(\tilde{V})\right]$
11: `#pragma omp section`
12: $U_r^H L_r^H \tilde{W} = C^T, \quad W(:, r : r+1) = \left[\mathrm{Re}\,(\tilde{W}), \mathrm{Im}\,(\tilde{W})\right]$ and skip $r + 1$
13: **end if**
14: **end for**

Parallel Implementation of IRKA

Using OpenMP with Preprocessing

Idea 2: Compute S and loop over $r \in S$. $\rightarrow$ remove index skip
$\rightarrow$ Parallelize the `for`-loop the OpenMP way.

Parallel Implementation of IRKA

Using OpenMP with Preprocessing

Idea 2: Compute S and loop over $r \in S$. $\rightarrow$ remove index skip
$\rightarrow$ Parallelize the `for`-loop the OpenMP way.

1: Compute $L_r U_r = (\sigma_r I - A)$ Using [K., Saak '09]
2: Compute $S = \{s \in \{1, \ldots, k\} | \sigma_s \in \mathbb{R} \text{ or } \sigma_{s-1} \neq \overline{\sigma_s}\}$
3: `#pragma omp parallel for`
4: **for** $r \in S$ **do**
5: **if** $\sigma_r \in \mathbb{R}$ **then**
6: $L_r U_r V(:, r) = B$
7: $U_r^T L_r^T W(:, r) = C^T$
8: **else**
9: $L_r U_r \tilde{V} = B, \quad V(:, r : r+1) = \left[\mathrm{Re}\,(\tilde{V}), \mathrm{Im}\,(\tilde{V})\right]$
10: $U_r^H L_r^H \tilde{W} = C^T, \quad W(:, r : r+1) = \left[\mathrm{Re}\,(\tilde{W}), \mathrm{Im}\,(\tilde{W})\right]$
11: **end if**
12: **end for**

Parallel Implementation of IRKA

Using a PThreads Threadpool

Idea 3: Compute the set S and insert all computations for $V(:,\cdot)$ and $W(:,\cdot)$ in a PThreads based thread pool.

Parallel Implementation of IRKA

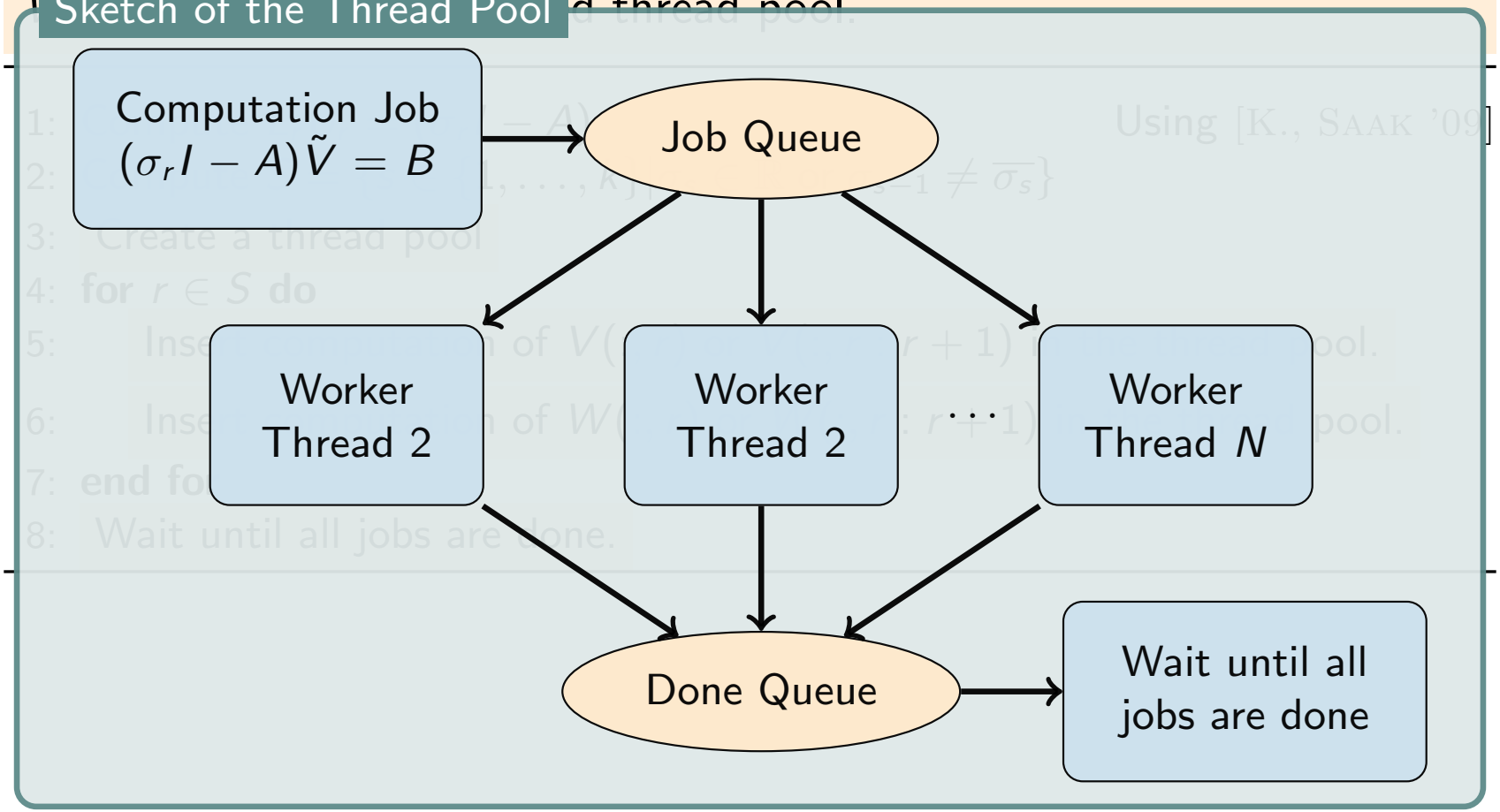
Using a PThreads Threadpool

Idea 3: Compute the set S and insert all computations for $V(:,\cdot)$ and $W(:,\cdot)$ in a PThreads based thread pool.

1: Compute $L_r U_r = (\sigma_r I - A)$ Using [K., Saak '09]
2: Compute $S = \{s \in \{1,\ldots,k\} | \sigma_s \in \mathbb{R} \text{ or } \sigma_{s-1} \neq \overline{\sigma_s}\}$
3: Create a thread pool
4: **for** $r \in S$ **do**
5: Insert computation of $V(:,r)$ or $V(:,r:r+1)$ in the thread pool.
6: Insert computation of $W(:,r)$ or $W(:,r:r+1)$ in the thread pool.
7: **end for**
8: Wait until all jobs are done.

Parallel Implementation of IRKA

Using a PThreads Threadpool



Parallel Implementation of IRKA

Using a PThreads Threadpool

Idea 3: Compute the set S and insert all computations for $V(:,\cdot)$ and $W(:,\cdot)$ in a PThreads based thread pool.

1: Compute $L_r U_r = (\sigma_r I - A)$ Using [K., Saak '09]
2: Compute $S = \{s \in \{1,\ldots,k\} | \sigma_s \in \mathbb{R} \text{ or } \sigma_{s-1} \neq \overline{\sigma_s}\}$
3: Create a thread pool
4: **for** $r \in S$ **do**
5: Insert computation of $V(:,r)$ or $V(:,r:r+1)$ in the thread pool.
6: Insert computation of $W(:,r)$ or $W(:,r:r+1)$ in the thread pool.
7: **end for**
8: Wait until all jobs are done.

Implemented as one job in the thread pool.

Numerical Results

Numerical Results

Hardware and Model Problem

All computations performed on a single node of `otto`[1]:

- 2x Intel Xeon X5650 CPUs (6 Cores each)
- 48 GB DDR3 RAM

[1] HPC Linux Cluster at MPI Magdeburg

Numerical Results

Hardware and Model Problem

All computations performed on a single node of `otto`[1]:

- 2x Intel Xeon X5650 CPUs (6 Cores each)
- 48 GB DDR3 RAM

As model problem we use an Finite-Difference space-discretization of:

$$\Delta u - 10x\frac{\partial u}{\partial x} - 100y\frac{\partial u}{\partial y} = \frac{\partial u}{\partial t} \text{ on } \Omega,$$

with $\Omega = (0,1) \times (0,1)$. The input acts on $\Omega_B = (0.1, 0.4) \times (0,1)$ and the output measures the function values on $\Omega_C = (0.7, 0.9) \times (0,1)$.

[1] HPC Linux Cluster at MPI Magdeburg

Numerical Results

Hardware and Model Problem

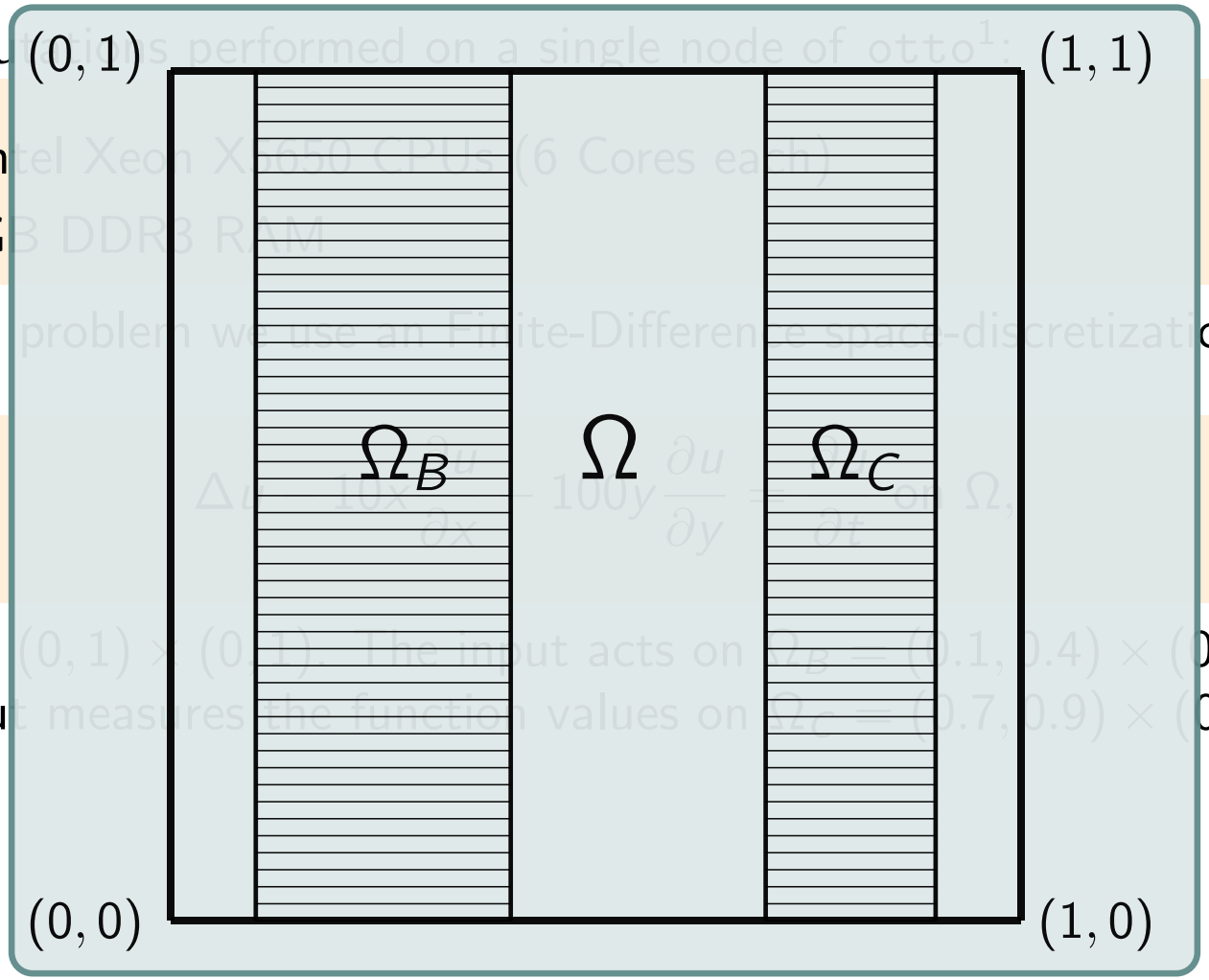
All computations performed on a single node of otto[1]:

- 2x Intel Xeon X5650 CPUs (6 Cores each)
- 48 GB DDR3 RAM

As model problem we use an Finite-Differences space-discretization of:

$$\Delta u - 10x\frac{\partial u}{\partial x} - 100y\frac{\partial u}{\partial y} = \frac{\partial u}{\partial t} \quad \text{on } \Omega,$$

with $\Omega = (0,1) \times (0,1)$. The input acts on $\Omega_B = (0.1, 0.4) \times (0,1)$ and the output measures the function values on $\Omega_C = (0.7, 0.9) \times (0,1)$.



[1] HPC Linux Cluster at MPI Magdeburg

Numerical Results

Overall Process

Runtime of IRKA:

15 steps, fixed $\dim(FOM) = 90\,000$ and $\dim(ROM) = 15$

Number of Threads	Sequential	OpenMP Sections	OpenMP Preproc.	PThreads Thread Pool
1	2.690 e+02	2.698 e+02	2.698 e+02	2.708 e+02
2	1.614 e+02	1.570 e+02	1.570 e+02	**1.561 e+02**
4	1.155 e+02	1.113 e+02	1.110 e+02	**1.082 e+02**
6	8.503 e+01	8.101 e+01	8.126 e+01	**7.709 e+01**
8	8.756 e+01	8.289 e+01	8.345 e+01	**8.003 e+01**
12	5.704 e+01	5.325 e+01	5.349 e+01	**5.018 e+01**

Numerical Results

Overall Process

Runtime of IRKA:

15 steps, fixed $\dim(FOM) = 90\,000$ and $\dim(ROM) = 15$

Number of Threads	Sequential	OpenMP Sections	OpenMP Preproc.	PThreads Thread Pool
1	2.690 e+02	2.698 e+02	2.698 e+02	2.708 e+02
2	1.614 e+02	1.570 e+02	1.570 e+02	**1.561 e+02**
4	1.155 e+02	1.113 e+02	1.110 e+02	**1.082 e+02**
6	8.503 e+01	8.101 e+01	8.126 e+01	**7.709 e+01**
8	8.756 e+01	8.289 e+01	8.345 e+01	**8.003 e+01**
12	5.704 e+01	5.325 e+01	5.349 e+01	**5.018 e+01**

Sequential: Only V, W construction performed sequentially.

Numerical Results

Overall Process

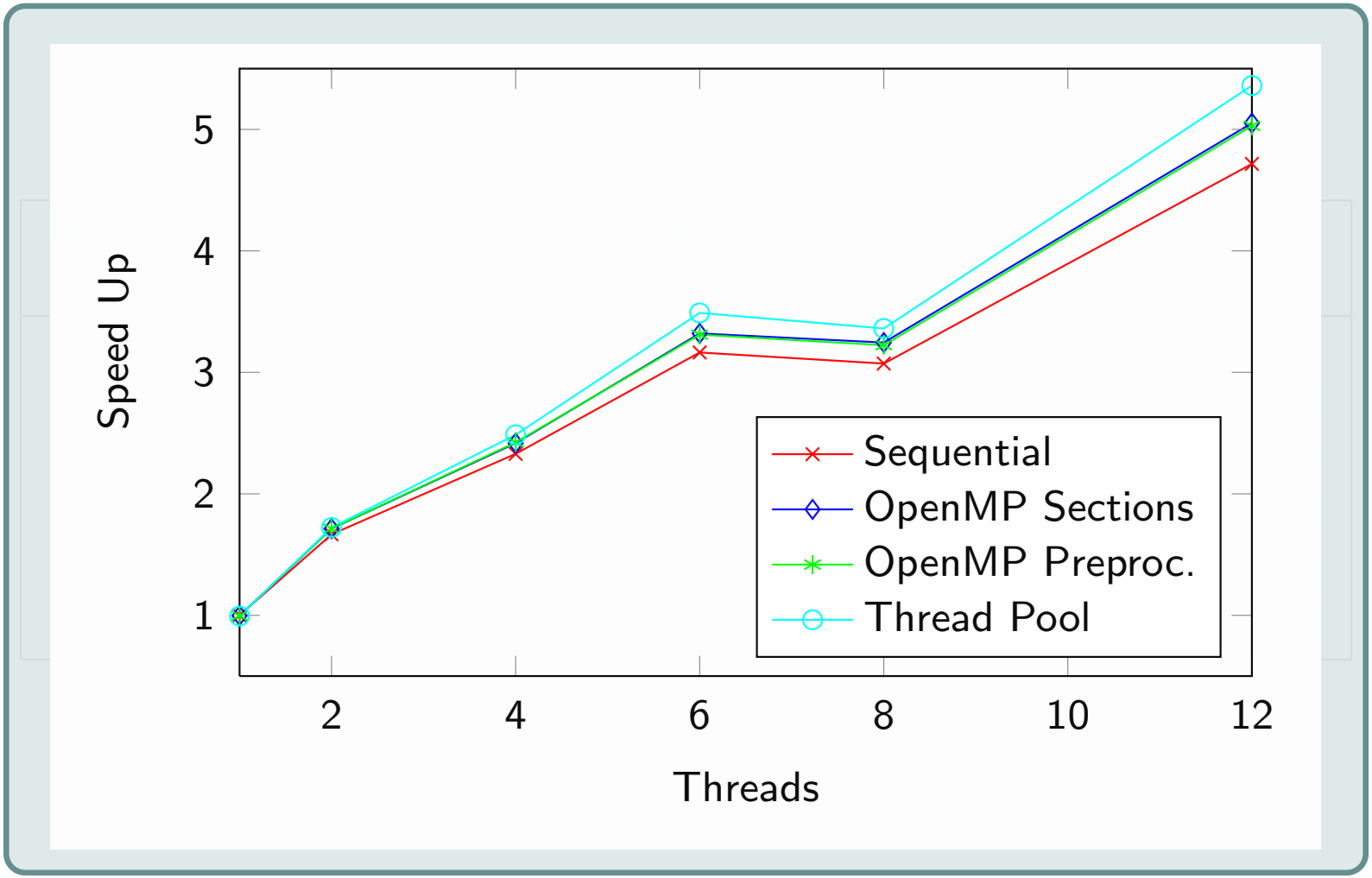
Speed up of IRKA:

15 steps, fixed $\dim(FOM) = 90\,000$ and $\dim(ROM) = 15$

Number of Threads	Sequential	OpenMP Sections	OpenMP Preproc.	PThreads Thread Pool
1	1.00e+00	9.97e−01	9.97e−01	9.93e−01
2	1.66e+00	1.71e+00	1.71e+00	**1.72e+00**
4	2.33e+00	2.416+00	2.42e+00	**2.49e+00**
6	3.16e+00	3.32e+00	3.31e+00	**3.49e+00**
8	3.07e+00	3.25e+00	3.22e+00	**3.36e+00**
12	4.72e+00	5.05e+00	5.03e+00	**5.36e+00**

Numerical Results

Overall Process



Numerical Results

Construction of V and W

Runtime of IRKA:

15 steps, fixed number of threads=12 and dim(ROM) = 15

dim(FOM)	Sequential	OpenMP Sections	OpenMP Preproc.	PThreads Thread Pool
100	4.167 e−02	**3.420 e−02**	3.432 e−02	2.967 e−01
625	9.238 e−02	**8.381 e−02**	8.465 e−02	4.912 e−01
2 500	4.261 e−01	3.701 e−01	**3.352 e−01**	9.342 e−01
10 000	2.426 e+00	**2.184 e+00**	2.194 e+00	2.652 e+00
40 000	1.542 e+01	1.397 e+01	1.393 e+01	**1.328 e+01**
90 000	5.704 e+01	5.325 e+01	5.349 e+01	**5.018 e+01**
160 000	1.506 e+02	1.440 e+02	1.444 e+02	**1.351 e+02**
250 000	2.875 e+02	2.740 e+02	2.746 e+02	**2.588 e+02**
562 500	1.124 e+03	1.087 e+03	1.090 e+03	**1.043 e+03**

Numerical Results

Construction of V and W

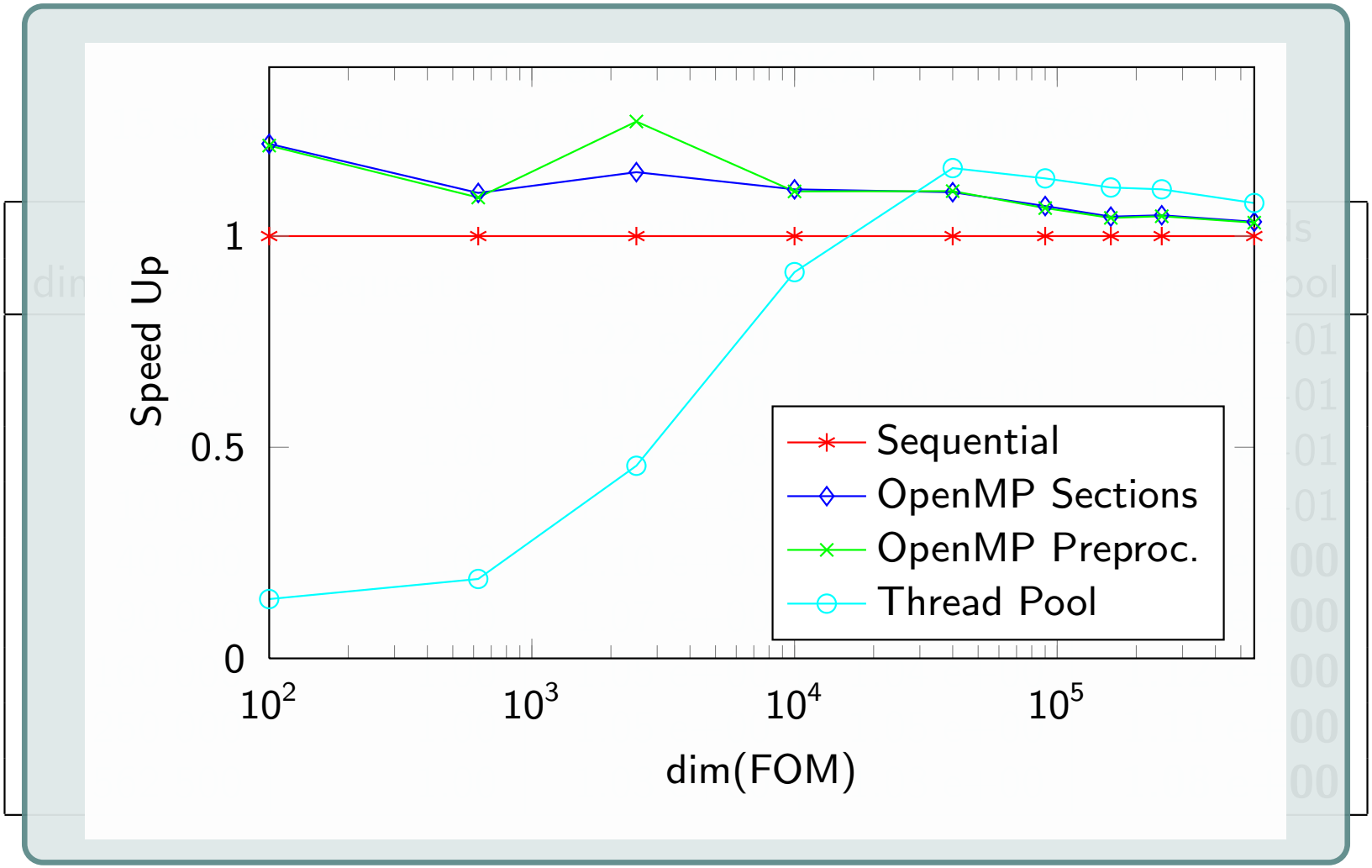
Speed up of IRKA:

15 steps, fixed number of threads=12 and dim(ROM) = 15

dim(FOM)	Sequential	OpenMP Sections	OpenMP Preproc.	PThreads Thread Pool
100	1.00	**1.22 e+00**	1.21 e+00	1.40 e−01
625	1.00	**1.10 e+00**	1.09 e+00	1.88 e−01
2 500	1.00	1.15 e+00	**1.27 e+00**	4.56 e−01
10 000	1.00	1.11 e+00	**1.11 e+00**	9.15 e−01
40 000	1.00	1.10 e+00	1.11 e+00	**1.16 e+00**
90 000	1.00	1.07 e+00	1.07 e+00	**1.14 e+00**
160 000	1.00	1.05 e+00	1.04 e+00	**1.12 e+00**
250 000	1.00	1.05 e+00	1.05 e+00	**1.11 e+00**
562 500	1.00	1.03 e+00	1.03 e+00	**1.08 e+00**

Numerical Results

Construction of V and W



Conclusions

- PThreads can beat OpenMP even for easy computations.
- Easy mathematical constructs require an extra preprocessing.
- Similar performance of the OpenMP sections and the OpenMP preprocessing code. (2 and 12 threads)
- Memory bandwidth limits more than the CPU speed.

Conclusions

- PThreads can beat OpenMP even for easy computations.
- Easy mathematical constructs require an extra preprocessing.
- Similar performance of the OpenMP sections and the OpenMP preprocessing code. (2 and 12 threads)
- Memory bandwidth limits more than the CPU speed.

Implemented in M.E.S.S.:
`http://www.mpi-magdeburg.mpg.de/mess`

Conclusions

- PThreads can beat OpenMP even for easy computations.
- Easy mathematical constructs require an extra preprocessing.
- Similar performance of the OpenMP sections and the OpenMP preprocessing code. (2 and 12 threads)
- Memory bandwidth limits more than the CPU speed.

Implemented in M.E.S.S.:
`http://www.mpi-magdeburg.mpg.de/mess`

S. Gugercin, A. C. Antoulas, and C. Beattie, *$\mathcal{H}_2$ Model Reduction for Large-Scale Dynamical Systems*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 609–638.

M. Köhler and J. Saak, *Efficiency Improving Implementation Techniques for Large Scale Matrix Equation Solvers*, Chemnitz Scientific Computing Prep. CSC 09-10, TU Chemnitz, 2009.

Conclusions

- PThreads can beat OpenMP even for easy computations.
- Easy mathematical constructs require an extra preprocessing.
- Similar performance of the OpenMP sections and the OpenMP preprocessing code. (2 and 12 threads)
- Memory bandwidth limits more than the CPU speed.

Thank you for your attention.

S. Gugercin, A. C. Antoulas, and C. Beattie, *$\mathcal{H}_2$ Model Reduction for Large-Scale Dynamical Systems*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 609–638.

M. Köhler and J. Saak, *Efficiency Improving Implementation Techniques for Large Scale Matrix Equation Solvers*, Chemnitz Scientific Computing Prep. CSC 09-10, TU Chemnitz, 2009.

Appendix

Single Pattern Multi-Value LU

Compute $L_i U_i = (A + p_i I)$ with knowledge of $LU = A$.

Appendix

Single Pattern Multi-Value LU

Compute $L_i U_i = (A + p_i I)$ with knowledge of $LU = A$.

Definition

Let $A \in \mathbb{R}^{n \times m}$ be a matrix. We call the set

$$\mathcal{P}(A) = \{(i,j) \mid A_{i,j} \neq 0\}$$

the **pattern** of A. Furthermore we define

$$\mathcal{P}_R(A,i) = \{j \mid A_{i,j} \neq 0\}$$

as the **pattern of the i-th row** of A.

Appendix

Single Pattern Multi-Value LU

Compute $L_i U_i = (A + p_i I)$ with knowledge of $LU = A$.

Key oberservation

If $\mathcal{P}(A + p_i I) = \mathcal{P}(A)$ holds:

- $\mathcal{P}(L_i) = \mathcal{P}(L)$ and $\mathcal{P}(U_i) = \mathcal{P}(U)$. [Gilbert '94]

Appendix

Single Pattern Multi-Value LU

Compute $L_i U_i = (A + p_i I)$ with knowledge of $LU = A$.

Key oberservation

If $\mathcal{P}(A + p_i I) = \mathcal{P}(A)$ holds:

- $\mathcal{P}(L_i) = \mathcal{P}(L)$ and $\mathcal{P}(U_i) = \mathcal{P}(U)$. [Gilbert '94]

- Symbolic analysis can be skipped.
- $\mathcal{P}(L)$ and $\mathcal{P}(U)$ are invariant w.r.t. to the shift parameter.
- All memory allocations can be done in one step, i.e. no reallocations needed.

$\rightarrow$ Use $\mathcal{P}(L)$ and $\mathcal{P}(U)$ to compute $L_i U_i = (A + p_i I)$.
$\rightarrow$ Only hold $\mathcal{P}(L)$ and $\mathcal{P}(U)$ one time in memory.

Appendix

Single Pattern Multi-Value LU

Observation:

- $\mathcal{P}(L)$ and $\mathcal{P}(U)$ are the same for any p_i.
- $\mathcal{P}(L)$ and $\mathcal{P}(U)$ accessed read only $\rightarrow$ no race conditions.

Appendix

Single Pattern Multi-Value LU

Observation:

- $\mathcal{P}(L)$ and $\mathcal{P}(U)$ are the same for any p_i.
- $\mathcal{P}(L)$ and $\mathcal{P}(U)$ accessed read only $\rightarrow$ no race conditions.

Realization: Use OpenMP to parallelize the loop over all p_i.

Appendix

Single Pattern Multi-Value LU

Observation:

- $\mathcal{P}(L)$ and $\mathcal{P}(U)$ are the same for any p_i.
- $\mathcal{P}(L)$ and $\mathcal{P}(U)$ accessed read only $\rightarrow$ no race conditions.

Realization: Use OpenMP to parallelize the loop over all p_i.

Observation: If the set $\{p_i\}_{i=1}^{r}$ is closed w.r.t to complex conjugation, we can save LU decompositions. For $p_{i+1} = \bar{p}_i$ use $L_{i+1} = \bar{L}_i$ and $U_{i+1} = \bar{U}_i$.
$\rightarrow$ Save up to the half of the decompositions.
$\rightarrow$ But destroys easy OpenMP parallelization.

Appendix

Single Pattern Multi-Value LU

Observation:

- $\mathcal{P}(L)$ and $\mathcal{P}(U)$ are the same for any p_i.
- $\mathcal{P}(L)$ and $\mathcal{P}(U)$ accessed read only $\rightarrow$ no race conditions.

Realization: Use OpenMP to parallelize the loop over all p_i.

Observation: If the set $\{p_i\}_{i=1}^r$ is closed w.r.t to complex conjugation, we can save LU decompositions. For $p_{i+1} = \bar{p}_i$ use $L_{i+1} = \bar{L}_i$ and $U_{i+1} = \bar{U}_i$.
$\rightarrow$ Save up to the half of the decompositions.
$\rightarrow$ But destroys easy OpenMP parallelization.

Solution:
$\rightarrow$ Add a preprocessing to the parallelized loop.
$\rightarrow$ Or use a thread pool (built with PThreads,...).

Appendix

Single Pattern Multi-Value LU

Observation:

- $\mathcal{P}(L)$ and $\mathcal{P}(U)$ are the same for any p_i.
- $\mathcal{P}(L)$ and $\mathcal{P}(U)$ accessed read only $\rightarrow$ no race conditions.

Realization: Use OpenMP to parallelize the loop over all p_i.

Observation: If the set $\{p_i\}_{i=1}^{r}$ is closed w.r.t to complex conjugation, we can save LU decompositions. For $p_{i+1} = \bar{p}_i$ use $L_{i+1} = \bar{L}_i$ and $U_{i+1} = \bar{U}_i$.
$\rightarrow$ Save up to the half of the decompositions.
$\rightarrow$ But destroys easy OpenMP parallelization.

Same problems as in the construction of V and W.

Solution:
$\rightarrow$ Add a preprocessing to the parallelized loop.
$\rightarrow$ Or use a thread pool (built with PThreads,...).

Appendix

Transfer Function Sampling

We what to evaluate $H(s) = C(sI - A)^{-1}B$ for a large number of points s.

Appendix

Transfer Function Sampling

We what to evaluate $H(s) = C(sI - A)^{-1}B$ for a large number of points s.

$\rightarrow$ each LU decomposition used only once
$\rightarrow$ too many in parallel $\rightarrow$ memory can run short
$\rightarrow$ swapping slows down the computation

Appendix

Transfer Function Sampling

We what to evaluate $H(s) = C(sI - A)^{-1}B$ for a large number of points s.

$\rightarrow$ each LU decomposition used only once
$\rightarrow$ too many in parallel $\rightarrow$ memory can run short
$\rightarrow$ swapping slows down the computation

Solution: Estimate the maximal number of threads, that can run without swapping the memory:

$$\#threads = \max\left\{\left\lfloor \frac{s \cdot (M - \text{sizeof}(\mathcal{P}(L)) - \text{sizeof}(\mathcal{P}(U)))}{\text{sizeof}(datatype) \cdot (\text{nnz}(L) + \text{nnz}(U))} \right\rfloor, 1\right\},$$

where M is the size of the physical memory and s is a security factor.