MAX PLANCK INSTITUTE
FOR DYNAMICS OF COMPLEX
TECHNICAL SYSTEMS
MAGDEBURG

CSC

COMPUTATIONAL METHODS IN
SYSTEMS AND CONTROL THEORY

Efficient Implementation of BLAS Level-3 solvers for Sylvester-type Matrix Equations

Martin Köhler

February 14, 2017

7th Workshop on Matrix Equations and Tensor Techniques

Introduction

Generalized Sylvester Equation (GSYLV)

$$AXB \pm CXD = Y$$

Introduction

Standard Sylvester Equation (SYLV)

$$AX \pm XB = Y$$

Standard Lyapunov Equation (LYAP)

$$AX + AX^T = Y$$

Standard Sylvester Equation 2 (SYLV2)

$$AXB \pm X = Y$$

Standard Stein Equation (STEIN)

$$AXA^T - X = Y$$

Generalized Sylvester Equation (GSYLV)

$$AXB \pm CXD = Y$$

Introduction

Standard Sylvester Equation (SYLV)

$$AX \pm XB = Y$$

Standard Lyapunov Equation (LYAP)

$$AX + AX^T = Y$$

Standard Sylvester Equation 2 (SYLV2)

$$AXB \pm X = Y$$

Standard Stein Equation (STEIN)

$$AXA^T - X = Y$$

Generalized Sylvester Equation (GSYLV)

$$AXB \pm CXD = Y$$

Generalized Stein Equation (GSTEIN)

$$AXA^T - BXB^T = Y$$

Generalized Lyapunov Equation (GLYAP)

$$AXB^T + BXA^T = Y$$

Introduction

Standard Sylvester Equation (SYLV)

$$AX \pm XB = Y$$

Standard Lyapunov Equation (LYAP)

$$AX + AX^T = Y$$

Standard Sylvester Equation 2 (SYLV2)

$$AXB \pm X = Y$$

Standard Stein Equation (STEIN)

$$AXA^T - X = Y$$

Generalized Sylvester Equation (GSYLV)

$$AXB \pm CXD = Y$$

Coupled Sylvester Equation (CSYLV)

$$AR \pm LB = E$$

$$CR \pm LD = F$$

Generalized Stein Equation (GSTEIN)

$$AXA^T - BXB^T = Y$$

Generalized Lyapunov Equation (GLYAP)

$$AXB^T + BXA^T = Y$$

Introduction

Implemented Direct Solvers on Shared Memory Architectures:

Equation	Software Packages				
	RECSY[1]	SLICOT[2]	LAPACK	Alg. 432[3]	Alg. 705[4]
SYLV	RECSYCT	SB04PD	xTRSYL	AXPXB	-
SYLV2	RECSYDT	SB04PD	-	-	-
LYAP	RECLYCT	SB03TD	-	ATXPXA	-
STEIN	RECLYDT	SB03UD	-	-	-
GSYLV	RECGSYL	-	-	-	SYLG
CSYLV	RECGCSY	SB04OD	xTGSYL	-	-
GLYAP	RECGLYCT	SG03AD	-	-	SYLGC
GSTEIN	RECGLYDT	SG03AD	-	-	SYLGD

[1] http://www8.cs.umu.se/~isak/recsy/

[2] http://www.slicot.org/

[3] http://www.netlib.org/toms/432.gz

[4] http://www.netlib.org/toms/705.gz

Introduction

Implemented Direct Solvers on Shared Memory Architectures:

Remarks:

→ **SLICOT** implementations only use Level-2 BLAS.

→ **RECSY** implementations are recursive and Level-3 BLAS, but optimized for 15 years old architectures, bad license.

→ **Algorithm 432** – original code by Bartels and Stewart, no BLAS at all, FORTRAN IV. ↯

→ **Algorithm 705** – LINPACK style code by Gardiner et. al., few Level-1 BLAS calls. ↯

→ **GLYAP-3** – Level-3 BLAS block implementation only for GLYAP/GSTEIN.

	Software Packages				
Equation	**RECSY**[1]	**SLICOT**[2]	**LAPACK**	**Alg. 432**[3]	**Alg. 705**[4]
SYLV	RECSYCT	SB04MD	XTRSYL	AXPXB	-
SYLV2	RECSYDT	SB04PD	-	-	-
LYAP	RECLYCT	SB03TD	-	ATXPXA	-
STEIN	RECLYDT	SB03UD	-	-	-
GSYLV	RECGSYL	-	-	-	SYLG
CSYLV	RECGCSY	SB04OD	XTGSYL	-	-
GLYAP	RECGLYCT	SG03AD	-	-	SYLGC
GSTEIN	RECGLYDT	SG03AD	-	-	SYLGD

[1] `http://www8.cs.umu.se/~isak/recsy/`

[2] `http://www.slicot.org/`

[3] `http://www.netlib.org/toms/432.gz`

[4] `http://www.netlib.org/toms/705.gz`

Introduction

Implemented Direct Solvers on Shared Memory Architectures:

	Software Packages				
Equation	RECSY[1]	SLICOT[2]	LAPACK	Alg. 432[3]	Alg. 705[4]
SYLV	RECSYCT	SB04MD	XTRSYL	AXPXB	-
SYLV2	RECSYDT	SB04PD	-	-	-
LYAP	RECLYCT	SB03MD	-	ATXPXA	-
STEIN	RECLYDT	SB03MD	-	-	-
GSYLV	RECGSYL	-	-	-	SYLG
CSYLV	RECGCSY	SB04OD	XTGSYL	-	-
GLYAP	RECGLYCT	SG03AD	-	-	SYLGC
GSTEIN	RECGLYDT	SG03AD	-	-	SYLGD

Remarks:

→ **SLICOT** implementations only use Level-2 BLAS.

→ **RECSY** implementations are recursive and Level-3 BLAS, but optimized for 15 years old architectures, bad license.

→ **Algorithm 432** – original code by Bartels and Stewart, no BLAS at all, FORTRAN IV. ↯

→ **Algorithm 705** – LINPACK style code by Gardiner et. al., few Level-1 BLAS calls. ↯

→ **GLYAP-3** – Level-3 BLAS block implementation only for GLYAP/GSTEIN.

All packages are not feature complete and mostly old.

[1] http://www8.cs.umu.se/~isak/recsy/

[2] http://www.slicot.org/

[3] http://www.netlib.org/toms/432.gz

[4] http://www.netlib.org/toms/705.gz

Introduction

General Workflow in Direct Solvers:

Compute real Schur forms:

$$A = QA_sQ^T \text{ and } B = ZB_sZ^T.$$

Transform the right hand side:

$$\tilde{Y} = Q^TYZ.$$

Solve the (quasi-) triangular equation:

$$A_s\tilde{X} \pm \tilde{X}B_s = \tilde{Y}.$$

Transform the solution:

$$X = Q\tilde{X}Z^T.$$

Compute generalized real Schur forms:

$$(A, C) = (Q_1A_sZ_1^T, Q_1C_sZ_1^T)$$

$$(B, D) = (Q_2B_sZ_2^T, Q_2D_sZ_2^T).$$

Transform the right hand side:

$$\tilde{Y} = Q_1^TYZ_2.$$

Solve the (quasi-) triangular equation

$$A_s\tilde{X}B_s \pm C_s\tilde{X}D_s = \tilde{Y}.$$

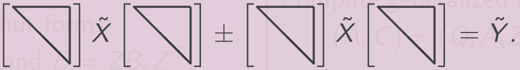
Transform the solution:

$$X = Z_1\tilde{X}Q_2^T.$$

Introduction

Our focus:

Fast solution of (quasi-) triangular Sylvester-type Matrix Equations:



General Workflow in Direct Solvers:

Compute real Schur forms:

$$A = QA_sQ^T \text{ and } B = ZB_sZ^T.$$

Transform the right hand side:

$$\tilde{Y} = Q^T Y Z.$$

Solve the (quasi-) triangular equation:

$$A_s\tilde{X} \pm \tilde{X}B_s = \tilde{Y}.$$

Transform the solution:

$$X = Q\tilde{X}Z^T.$$

Compute generalized real Schur forms:

$$(A,C) = (Q_1A_sZ_1^T, Q_1C_sZ_1^T)$$
$$(B,D) = (Q_2B_sZ_2^T, Q_2D_sZ_2^T).$$

Transform the right hand side:

$$\tilde{Y} = Q_1^T Y Z_2.$$

Solve the (quasi-) triangular equation

$$A_s\tilde{X}B_s \pm C_s\tilde{X}D_s = \tilde{Y}.$$

Transform the solution:

$$X = Z_1\tilde{X}Q_2^T.$$

Introduction

General Workflow in Direct Solvers:

Compute real Schur forms:

$$A = QA_sQ^T \text{ and } B = ZB_sZ^T.$$

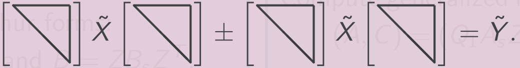
Compute generalized real Schur forms:

$$(A, C) = (Q_1 A_s Z_1^T, Q_1 C_s Z_1^T)$$

$$(B, D) = (Q_2 B_s Z_2^T, Q_2 D_s Z_2^T).$$

Our focus:

Fast solution of (quasi-) triangular Sylvester-type Matrix Equations:



We use the Generalized Sylvester Equation as most complex variant for the optimizations.

Transform the right hand side:

$$\tilde{Y} = Q^T Y Z.$$

Transform the right hand side:

$$\tilde{Y} = Q_1^T Y Z_2.$$

Solve the (quasi-) triangular equation:

$$A_s\tilde{X} \pm \tilde{X}B_s = \tilde{Y}.$$

Solve the (quasi-) triangular equation

$$A_s\tilde{X}B_s \pm C_s\tilde{X}D_s = \tilde{Y}.$$

Transform the solution:

$$X = Q\tilde{X}Z^T.$$

Transform the solution:

$$X = Z_1\tilde{X}Q_2^T.$$

Block-Algorithm

The real Generalized Schur Decomposition of (A, C) and (B, D) yields:

$$A_s = \begin{bmatrix} A_{11} & \cdots & A_{1p} \\ & \ddots & \vdots \\ 0 & & A_{pp} \end{bmatrix}, \quad C_s = \begin{bmatrix} C_{11} & \cdots & C_{1p} \\ & \ddots & \vdots \\ 0 & & C_{pp} \end{bmatrix}, \quad \tilde{X} = \begin{bmatrix} \tilde{X}_{11} & \cdots & \tilde{X}_{1q} \\ \vdots & \ddots & \vdots \\ \tilde{X}_{p1} & \cdots & \tilde{X}_{pq} \end{bmatrix},$$

$$B_s = \begin{bmatrix} B_{11} & \cdots & B_{1q} \\ & \ddots & \vdots \\ 0 & & B_{qq} \end{bmatrix}, \quad D_s = \begin{bmatrix} D_{11} & \cdots & C_{1q} \\ & \ddots & \vdots \\ 0 & & D_{qq} \end{bmatrix}, \quad \tilde{Y} = \begin{bmatrix} \tilde{Y}_{11} & \cdots & \tilde{Y}_{1q} \\ \vdots & \ddots & \vdots \\ \tilde{Y}_{p1} & \cdots & \tilde{Y}_{qq} \end{bmatrix},$$

where (A_{ii}, C_{ii}) and (B_{ii}, D_{ii}) are 1×1 or 2×2 according to the eigenvalues of (A, C), or (B, D) respectively.

Block-Algorithm

The real Generalized Schur Decomposition of (A, C) and (B, D) yields:

$$A_s = \begin{bmatrix} A_{11} & \cdots & A_{1p} \\ & \ddots & \vdots \\ 0 & & A_{pp} \end{bmatrix}, \quad C_s = \begin{bmatrix} C_{11} & \cdots & C_{1p} \\ & \ddots & \vdots \\ 0 & & C_{pp} \end{bmatrix}, \quad \tilde{X} = \begin{bmatrix} \tilde{X}_{11} & \cdots & \tilde{X}_{1q} \\ \vdots & \ddots & \vdots \\ \tilde{X}_{p1} & \cdots & \tilde{X}_{pq} \end{bmatrix},$$

$$B_s = \begin{bmatrix} B_{11} & \cdots & B_{1q} \\ & \ddots & \vdots \\ 0 & & B_{qq} \end{bmatrix}, \quad D_s = \begin{bmatrix} D_{11} & \cdots & C_{1q} \\ & \ddots & \vdots \\ 0 & & D_{qq} \end{bmatrix}, \quad \tilde{Y} = \begin{bmatrix} \tilde{Y}_{11} & \cdots & \tilde{Y}_{1q} \\ \vdots & \ddots & \vdots \\ \tilde{Y}_{p1} & \cdots & \tilde{Y}_{qq} \end{bmatrix},$$

where (A_{ii}, C_{ii}) and (B_{ii}, D_{ii}) are 1×1 or 2×2 according to the eigenvalues of (A, C), or (B, D) respectively.

We need to solve $p \cdot q$ small Sylvester equations:

$$A_{kk}\tilde{X}_{kl}B_{ll} + C_{kk}\tilde{X}_{k\ell}D_{ll} = \tilde{Y}_{kl} - \sum_{\substack{i=k,\dots,p \\ j=1,\dots,l \\ (i,j)\neq(k,l)}} \left(A_{ki}\tilde{X}_{ij}B_{jl} \pm C_{ki}\tilde{X}_{ij}D_{jl} \right).$$

Block-Algorithm

Our Question:

Are the matrices A_{kk}, A_{ll}, $\ldots$ restricted to be 1×1 or 2×2 matrices?

Block-Algorithm

Our Question:

Are the matrices $A_{kk},\ A_{ll},\ \dots$ restricted to be 1×1 or 2×2 matrices?

Answer – No!

As long as no complex eigenvalue-pairs in (A, C) and (B, D) is split by the blocking and $\tilde{X}$ and $\tilde{Y}$ are partitioned accordingly everything works for larger blocks.

Block-Algorithm

Our Question:

Are the matrices $A_{kk},\ A_{ll},\ \ldots$ restricted to be 1×1 or 2×2 matrices?

Answer – No!

As long as no complex eigenvalue-pairs in (A, C) and (B, D) is split by the blocking and $\tilde{X}$ and $\tilde{Y}$ are partitioned accordingly everything works for larger blocks.
Arbitrary block sizes m_b for (A, C) and n_b for (B, D) are possible but adjustments by ± 1 to fit the eigenvalue structure are necessary.

Block-Algorithm

Our Question:

Are the matrices $A_{kk},\ A_{ll},\ \dots$ restricted to be 1×1 or 2×2 matrices?

Answer – No!

As long as no complex eigenvalue-pairs in (A, C) and (B, D) is split by the blocking and $\tilde{X}$ and $\tilde{Y}$ are partitioned accordingly everything works for larger blocks.

Arbitrary block sizes m_b for (A, C) and n_b for (B, D) are possible but adjustments by ± 1 to fit the eigenvalue structure are necessary.

Special cases: $(A_s,\ C_s \in \mathbb{R}^{m \times m},\ B_s,\ D_s \in \mathbb{R}^{n \times n}$ and $\tilde{X},\ \tilde{Y} \in \mathbb{R}^{m \times n})$

- $m_b = 1$ and $n_b = 1$ – Level-2 Bartels-Stewart implementation in SLICOT.

Block-Algorithm

Our Question:

Are the matrices $A_{kk},\ A_{ll},\ \dots$ restricted to be 1×1 or 2×2 matrices?

Answer – No!

As long as no complex eigenvalue-pairs in (A, C) and (B, D) is split by the blocking and $\tilde{X}$ and $\tilde{Y}$ are partitioned accordingly everything works for larger blocks.
Arbitrary block sizes m_b for (A, C) and n_b for (B, D) are possible but adjustments by ± 1 to fit the eigenvalue structure are necessary.

Special cases: $(A_s,\ C_s \in \mathbb{R}^{m\times m},\ B_s,\ D_s \in \mathbb{R}^{n\times n}$ and $\tilde{X},\ \tilde{Y} \in \mathbb{R}^{m\times n})$

- $m_b = 1$ and $n_b = 1$ – Level-2 Bartels-Stewart implementation in SLICOT.
- $m_b = \frac{m}{2}$ and $n_b = \frac{n}{2}$ – Recursive Blocking (RECSY)

Block-Algorithm

Our Question:

Are the matrices $A_{kk}, A_{ll}, \dots$ restricted to be 1×1 or 2×2 matrices?

Answer – No!

As long as no complex eigenvalue-pairs in (A, C) and (B, D) is split by the blocking and $\tilde{X}$ and $\tilde{Y}$ are partitioned accordingly everything works for larger blocks.

Arbitrary block sizes m_b for (A, C) and n_b for (B, D) are possible but adjustments by ± 1 to fit the eigenvalue structure are necessary.

Special cases: $(A_s, C_s \in \mathbb{R}^{m \times m}, B_s, D_s \in \mathbb{R}^{n \times n}$ and $\tilde{X}, \tilde{Y} \in \mathbb{R}^{m \times n})$

- $m_b = 1$ and $n_b = 1$ – Level-2 Bartels-Stewart implementation in SLICOT.
- $m_b = \frac{m}{2}$ and $n_b = \frac{n}{2}$ – Recursive Blocking (RECSY)
- $m_b = m$ and $n_b = 1$ – Algorithm 705 by Gardiner et. al.

Block-Algorithm

Our Question:

Are the matrices $A_{kk},\ A_{ll},\ \ldots$ restricted to be 1×1 or 2×2 matrices?

Answer – No!

As long as no complex eigenvalue-pairs in (A, C) and (B, D) is split by the blocking and $\tilde{X}$ and $\tilde{Y}$ are partitioned accordingly everything works for larger blocks.
Arbitrary block sizes m_b for (A, C) and n_b for (B, D) are possible but adjustments by ± 1 to fit the eigenvalue structure are necessary.

Special cases: $(A_s,\ C_s \in \mathbb{R}^{m\times m},\ B_s,\ D_s \in \mathbb{R}^{n\times n}$ and $\tilde{X},\ \tilde{Y} \in \mathbb{R}^{m\times n})$

- $m_b = 1$ and $n_b = 1$ – Level-2 Bartels-Stewart implementation in SLICOT.
- $m_b = \frac{m}{2}$ and $n_b = \frac{n}{2}$ – Recursive Blocking (RECSY)
- $m_b = m$ and $n_b = 1$ – Algorithm 705 by Gardiner et. al.

$\rightarrow$ Why are m_b and n_b not chosen such that $A_{kk},\ B_{ll},\ C_{kk},\ D_{ll},\ \tilde{X}_{kl}$ and $\tilde{Y}_{kl}$ fit into the CPU's cache(s)?

Efficient Implementation

General Aspects:

- The updates on the right hand sides $\tilde{Y}_{kl}$

$$\tilde{Y}_{kl} - \sum_{\substack{i=k,\dots,p\\ j=1,\dots,l\\ (i,j)\neq(k,l)}} \left(A_{ki}\tilde{X}_{ij}B_{jl} \pm C_{ki}\tilde{X}_{ij}D_{jl} \right)$$

can be rewritten and unified into few GEMM/TRMM operations.
→ Assumed to be as efficient as possible for sufficiently large matrices.

Efficient Implementation

General Aspects:

- The updates on the right hand sides $\tilde{Y}_{kl}$

$$\tilde{Y}_{kl} - \sum_{\substack{i=k,\ldots,p\\ j=1,\ldots,l\\ (i,j)\neq(k,l)}} \left(A_{ki}\tilde{X}_{ij}B_{jl} \pm C_{ki}\tilde{X}_{ij}D_{jl}\right)$$

  can be rewritten and unified into few GEMM/TRMM operations.
  → Assumed to be as efficient as possible for sufficiently large matrices.
- Block sizes m_b and n_b are a freely selectable parameter.

Efficient Implementation

General Aspects:

- The updates on the right hand sides $\tilde{Y}_{kl}$

$$\tilde{Y}_{kl} - \sum_{\substack{i=k,\ldots,p \\ j=1,\ldots,l \\ (i,j)\neq(k,l)}} \left(A_{ki}\tilde{X}_{ij}B_{jl} \pm C_{ki}\tilde{X}_{ij}D_{jl} \right)$$

  can be rewritten and unified into few GEMM/TRMM operations.
  → Assumed to be as efficient as possible for sufficiently large matrices.
- Block sizes m_b and n_b are a freely selectable parameter.
- The solution $\tilde{X}$ overwrites the right hand side $\tilde{Y}$.

Efficient Implementation

General Aspects:

- The updates on the right hand sides $\tilde{Y}_{kl}$

  $$\tilde{Y}_{kl} - \sum_{\substack{i=k,\ldots,p \\ j=1,\ldots,l \\ (i,j)\neq(k,l)}} \left(A_{ki}\tilde{X}_{ij}B_{jl} \pm C_{ki}\tilde{X}_{ij}D_{jl} \right)$$

  can be rewritten and unified into few GEMM/TRMM operations.
  → Assumed to be as efficient as possible for sufficiently large matrices.
- Block sizes m_b and n_b are a freely selectable parameter.
- The solution $\tilde{X}$ overwrites the right hand side $\tilde{Y}$.
- All matrices are stored in the Fortran Column-Major-Scheme.

Efficient Implementation

Algorithm 1 Block Bartels-Stewart Algorithm for Generalized Sylvester Equations

```
Input: $A_s,\ C_s \in \mathbb{R}^{m\times m},\ B_s,\ D_s \in \mathbb{R}^{n\times n}$ and $\tilde{Y} \in \mathbb{R}^{m\times n},\ m_b,\ n_b \in \mathbb{N}$
Output: $\tilde{X} \in \mathbb{R}^{m\times n}$ overwriting $\tilde{Y}$
if  $m \leq m_b$ and $n \leq n_b$ then
    Solve $A_s\tilde{X}B_s \pm C_s\tilde{X}D_s = \tilde{Y}$.
else
    for $k = m, \ldots, 1$ step by $m_b$ do
        for $l = 1, \ldots, n$ step by $n_b$ do
            Solve $A_{kk}\tilde{X}_{kl}B_{ll} \pm C_{kk}\tilde{X}_{kl}D_{ll} = \tilde{Y}_{kl}$.
            $\tilde{Y}_{k,l+1:n} = \tilde{Y}_{k,l+1:n} - A_{kk}\tilde{X}_{kl}B_{kl} \mp C_{kk}\tilde{X}_{kl}D_{kl}$
        end for
        $\tilde{Y}_{1:k-1,1:n} = \tilde{Y}_{1:k-1,1:n} - A_{1:k-1,k:k+m_b-1}\tilde{X}_{k:k+m_b-1,1:l}B_s$
                    $\mp C_{1:k-1,k:k+m_b-1}\tilde{X}_{k:k+m_b-1,1:l}D_s$
    end for
end if
```

Efficient Implementation

Algorithm 1 Block Bartels-Stewart Algorithm for Generalized Sylvester Equations

```
Input: $A_s,\ C_s \in \mathbb{R}^{m\times m},\ B_s,\ D_s \in \mathbb{R}^{n\times n}$ and $\tilde{Y} \in \mathbb{R}^{m\times n},\ m_b,\ n_b \in \mathbb{N}$
Output: $\tilde{X} \in \mathbb{R}^{m\times n}$ overwriting $\tilde{Y}$
if  $m \leq m_b$ and $n \leq n_b$ then
    $\boxed{\text{Solve } A_s\tilde{X}B_s \pm C_s\tilde{X}D_s = \tilde{Y}.}$
else
    for $k = m, \dots, 1$ step by $m_b$ do
        for $l = 1, \dots, n$ step by $n_b$ do
            $\boxed{\text{Solve } A_{kk}\tilde{X}_{kl}B_{ll} \pm C_{kk}\tilde{X}_{kl}D_{ll} = \tilde{Y}_{kl}.}$
            $\tilde{Y}_{k,l+1:n} = \tilde{Y}_{k,l+1:n} - A_{kk}\tilde{X}_{kl}B_{kl} \mp C_{kk}\tilde{X}_{kl}D_{kl}$
        end for
        $\tilde{Y}_{1:k-1,1:n} = \tilde{Y}_{1:k-1,1:n} - A_{1:k-1,k:k+m_b-1}\tilde{X}_{k:k+m_b-1,1:l}B_s$
                                  $\mp C_{1:k-1,k:k+m_b-1}\tilde{X}_{k:k+m_b-1,1:l}D_s$
    end for
end if
```

Critical operation for a fast solution scheme.

Efficient Implementation

Algorithm 1 Block Bartels-Stewart Algorithm for Generalized Sylvester Equations

```
Input: $A_s, C_s \in \mathbb{R}^{m \times m}$, $B_s, D_s \in \mathbb{R}^{n \times n}$ and $\tilde{Y} \in \mathbb{R}^{m \times n}$, $m_b, n_b \in \mathbb{N}$
Output: $\tilde{X} \in \mathbb{R}^{m \times n}$ overwriting $\tilde{Y}$
if $m \leq m_b$ and $n \leq n_b$ then
    Solve $A_s \tilde{X} B_s \pm C_s \tilde{X} D_s = \tilde{Y}$.
else
    for $k = m, \ldots, 1$ step by $m_b$ do
        for $l = 1, \ldots, n$ step by $n_b$ do
            Solve $A_{kk} \tilde{X}_{kl} B_{ll} \pm C_{kk} \tilde{X}_{kl} D_{ll} = \tilde{Y}_{kl}$.
            $\tilde{Y}_{k,l+1:n} = \tilde{Y}_{k,l+1:n} - A_{kk} \tilde{X}_{kl} B_{kl} \mp C_{kk} \tilde{X}_{kl} D_{kl}$
        end for
        $\tilde{Y}_{1:k-1,1:n} = \tilde{Y}_{1:k-1,1:n} - A_{1:k-1,k:k+m_b-1} \tilde{X}_{k:k+m_b-1,1:l} B_s$
                    $\mp C_{1:k-1,k:k+m_b-1} \tilde{X}_{k:k+m_b-1,1:l} D_s$
    end for
end if
```

Critical operation for a fast solution scheme.

Goal: Develop an efficient solver for small Sylvester Equations.

Efficient Inner Solvers

Hardware and Software Setup

We use a test driven development scheme for the kernels on random matrices.

Hardware

- Intel Xeon E5-2640v3 (Haswell), 2x8 Cores, 16x256kB L2 Cache, 2x20MB L3 Cache, AVX vector unit
- 64GB DDR3 RAM

Software

- CentOS 7.3 – x86_64
- Intel Parallel Studio XE 2017 C/Fortran Compiler + Intel MKL 2017.0.1
- Compiler-Flags: `-O3 -xHost -qopenmp`
- Computational routines are written in Fortran 90/95.
- Reference results with Algorithm 705 and RECSY, double precision.
- Residual and Forward error of all results are comparable.

Efficient Inner Solvers

Naive Approach

- Sylvester equations with $m \leq 2$ and $n \leq 2$ are trivial to solve via their Kronecker representation:

$$AXB \pm CXD = Y \quad \Longleftrightarrow \quad (B^T \otimes A \pm D^T \otimes C) \operatorname{vec} X = \operatorname{vec} Y$$

- Larger problems are solved using Algorithm 1 with $m_b = 1$ and $n_b = 1$.

Efficient Inner Solvers

Naive Approach

- Sylvester equations with $m \leq 2$ and $n \leq 2$ are trivial to solve via their Kronecker representation:

$$AXB \pm CXD = Y \quad \Longleftrightarrow \quad (B^T \otimes A \pm D^T \otimes C) \operatorname{vec} X = \operatorname{vec} Y$$

- Larger problems are solved using Algorithm 1 with $m_b = 1$ and $n_b = 1$.

Usual LAPACK-like blocking scheme.

Efficient Inner Solvers

Naive Approach

- Sylvester equations with $m \leq 2$ and $n \leq 2$ are trivial to solve via their Kronecker representation:

$$AXB \pm CXD = Y \quad \Longleftrightarrow \quad (B^T \otimes A \pm D^T \otimes C) \operatorname{vec} X = \operatorname{vec} Y$$

- Larger problems are solved using Algorithm 1 with $m_b = 1$ and $n_b = 1$.

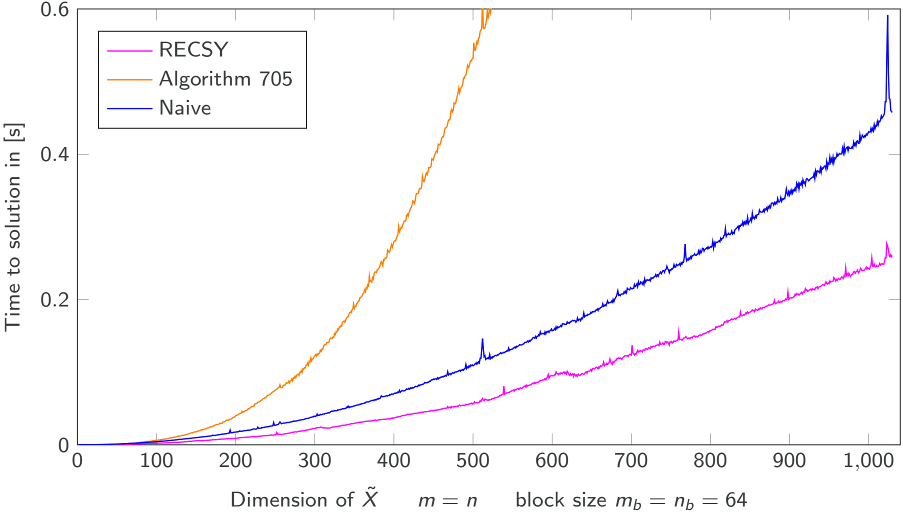
Usual LAPACK-like blocking scheme.

Test Procedure

- Algorithm 1 with random matrices $A, B, C, D \in \mathbb{R}^{m \times m}$, $m = 1, \ldots, 1030$.
- Eigenvalues sorted such that $A_{64k+1,64k} = 0$ and $B_{64k+1,64k} = 0$, $\forall k \in \mathbb{N}$.
- Only inner solver exchanged/optimized.
- 16 threads for multi-threaded BLAS calls.

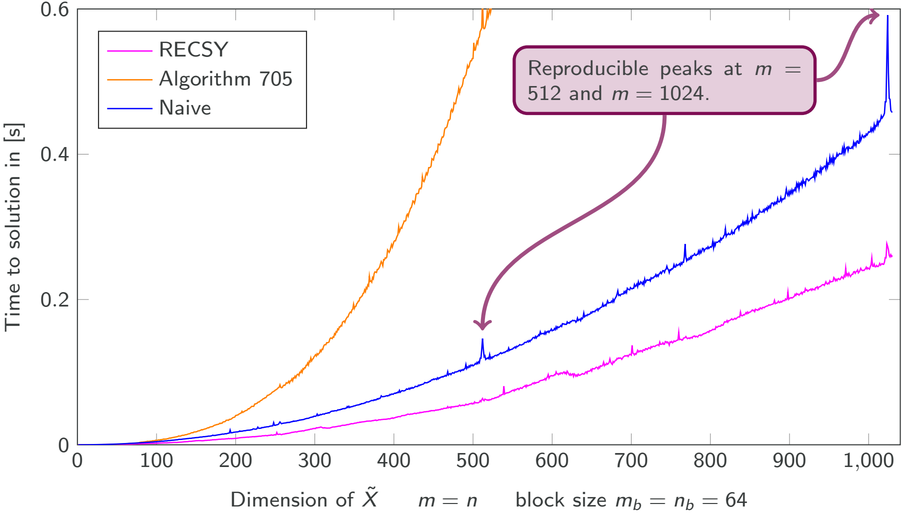
Efficient Inner Solvers

Naive Approach



Efficient Inner Solvers

Naive Approach



Reproducible peaks at $m = 512$ and $m = 1024$.

Efficient Inner Solvers

Level-2 BLAS Calls

- Replace Level-3 BLAS calls with the corresponding Level-2 operations:
  - xGEMM → xGEMV, xGER, and xAXPY,
  - xTRMM → xTRMV.

Efficient Inner Solvers

Level-2 BLAS Calls

- Replace Level-3 BLAS calls with the corresponding Level-2 operations:
  - xGEMM → xGEMV, xGER, and xAXPY,
  - xTRMM → xTRMV.
- GEMM operations up to 2×2 are directly computed.

Efficient Inner Solvers

Level-2 BLAS Calls

- Replace Level-3 BLAS calls with the corresponding Level-2 operations:
  - xGEMM → xGEMV, xGER, and xAXPY,
  - xTRMM → xTRMV.
- GEMM operations up to 2×2 are directly computed.
- Appearing xAXPY operations, caused by the GEMM replacement, are performed by Fortran intrinsics:

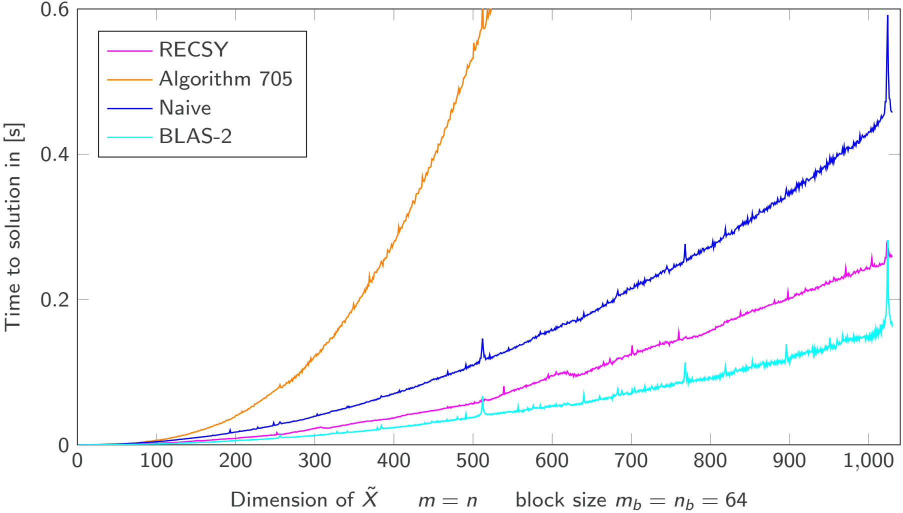
```
CALL DAXPY(N-LH, -MAT(1,1), B(L,LH+1), LDB, X(K,LH+1), LDX)
CALL DAXPY(N-LH, -MAT(3,1), B(LH,LH+1), LDB,X(K,LH+1), LDX)
CALL DAXPY(N-LH, -MAT(2,1), B(L,LH+1), LDB, X(KH,LH+1), LDX)
CALL DAXPY(N-LH, -MAT(4,1), B(LH,LH+1), LDB,X(KH,LH+1), LDX)
```

  is transformed into

```
X(K,LH+1:N) = X(K,LH+1:N) - MAT(1,1) * B(L,LH+1:N) -
  MAT(3,1) * B(LH,LH+1:N) - SGN * MAT(1,2) * D(L,LH+1:N)
  - SGN * MAT(3,2) * D(LH, LH+1:N)
```

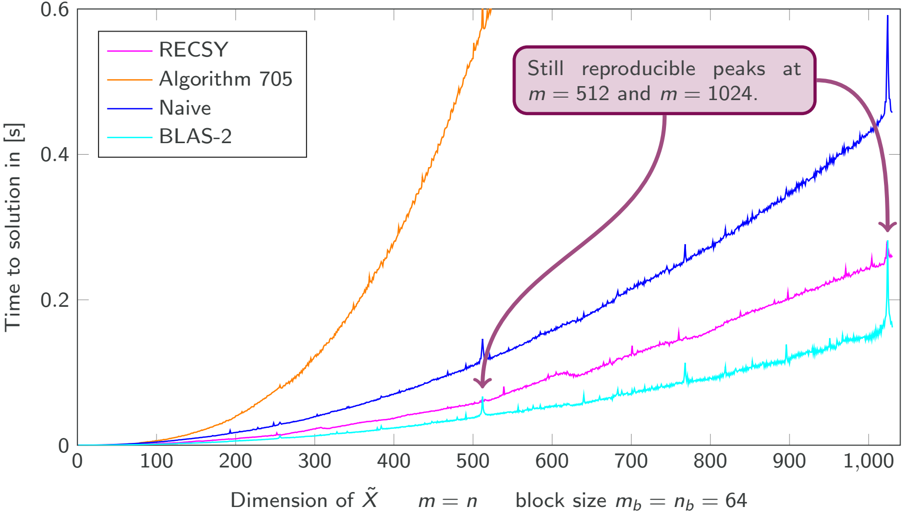
Efficient Inner Solvers

Level-2 BLAS Calls



Efficient Inner Solvers

Level-2 BLAS Calls



Still reproducible peaks at $m = 512$ and $m = 1024$.

Efficient Inner Solvers

Reorder the data access

The peaks at $m = 512$ and $m = 1024$ (and the next ones at 1536 and 2048) are caused by cache-misses/unused prefetching.
Reason: The leading dimension of the matrix (times 8 Byte per value) is a multiple of the pagesize (4096 Bytes).

The access to `X(K,LH+1:N)`, `B(L,LH+1:N)`, and `D(L,LH+1:N)` is not well suited for the matrix storage. (column-major-storage vs. row-wise access).

Efficient Inner Solvers

Reorder the data access

The peaks at $m = 512$ and $m = 1024$ (and the next ones at 1536 and 2048) are caused by cache-misses/unused prefetching.
Reason: The leading dimension of the matrix (times 8 Byte per value) is a multiple of the pagesize (4096 Bytes).

The access to `X(K,LH+1:N)`, `B(L,LH+1:N)`, and `D(L,LH+1:N)` is not well suited for the matrix storage. (column-major-storage vs. row-wise access).

- Same optimizations as before but...

Efficient Inner Solvers

Reorder the data access

The peaks at $m = 512$ and $m = 1024$ (and the next ones at 1536 and 2048) are caused by cache-misses/unused prefetching.
Reason: The leading dimension of the matrix (times 8 Byte per value) is a multiple of the pagesize (4096 Bytes).

The access to `X(K,LH+1:N)`, `B(L,LH+1:N)`, and `D(L,LH+1:N)` is not well suited for the matrix storage. (column-major-storage vs. row-wise access).

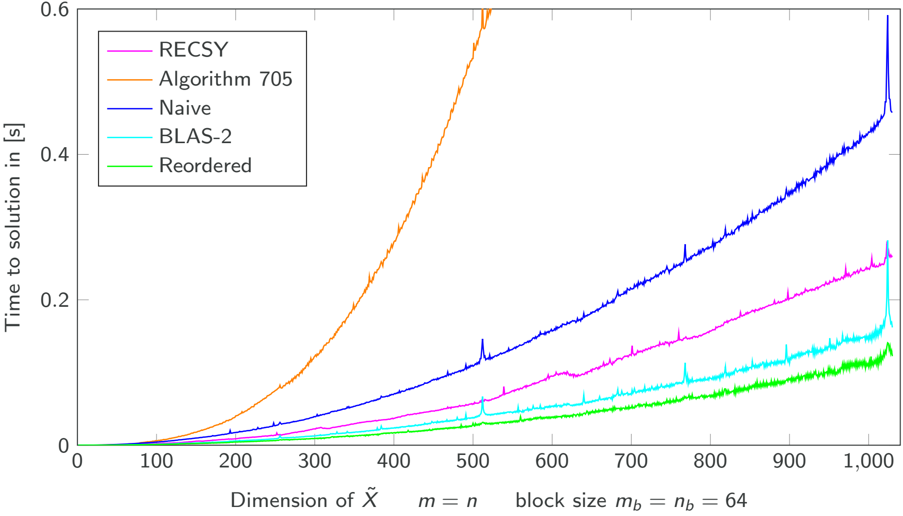
- Same optimizations as before but…
- solve the small equation column-wise instead of row-wise.
  The crucial operations change to:

```
X(1:K-1,L) = X(1:K-1,L) - MAT(1,1) * A(1:K-1,K)
   - MAT(2,1) * A(1:K-1,KH) - SGN * MAT(1,2) * C(1:K-1,K)
   - SGN * MAT(2,2) * C(1:K-1,KH)
```

→ Access on X_{kl}, A_{kk} and C_{kk} fits the storage scheme.

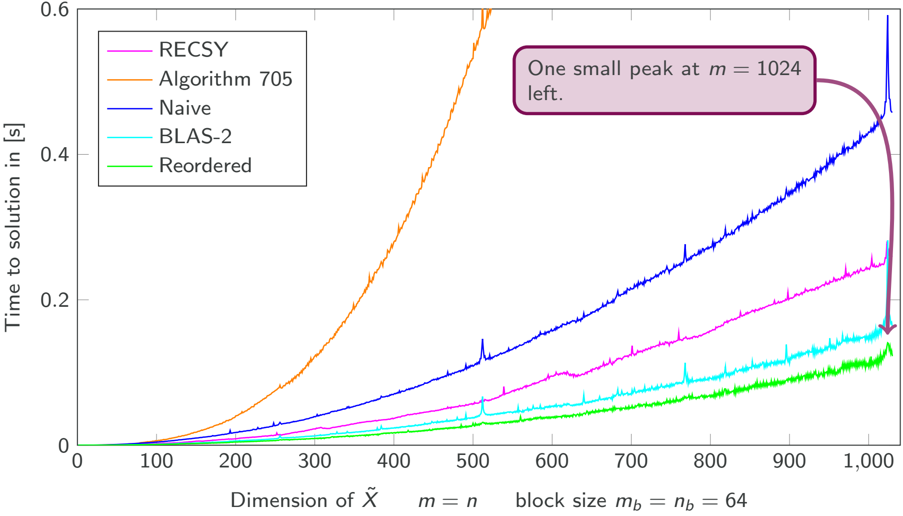
Efficient Inner Solvers

Reorder the data access



Efficient Inner Solvers

Reorder the data access



One small peak at $m = 1024$ left.

Efficient Inner Solvers

Use local copies of A_{kk}, B_{ll}, C_{kk}, D_{ll}, and X_{kl}

The leading dimension of A_s, B_s, C_s and D_s still yields cache misses and unnecessary prefetching.

- Keep all previous optimizations and

Efficient Inner Solvers

Use local copies of A_{kk}, B_{ll}, C_{kk}, D_{ll}, and X_{kl}

The leading dimension of A_s, B_s, C_s and D_s still yields cache misses and unnecessary prefetching.

- Keep all previous optimizations and
- create a local copy of A_{kk} and C_{kk} with leading dimension m_b,

Efficient Inner Solvers

Use local copies of A_{kk}, B_{ll}, C_{kk}, D_{ll}, and X_{kl}

The leading dimension of A_s, B_s, C_s and D_s still yields cache misses and unnecessary prefetching.

- Keep all previous optimizations and
- create a local copy of A_{kk} and C_{kk} with leading dimension m_b,
- create a local copy of B_{ll} and C_{ll} with leading dimension n_b,

Efficient Inner Solvers

Use local copies of A_{kk}, B_{ll}, C_{kk}, D_{ll}, and X_{kl}

The leading dimension of A_s, B_s, C_s and D_s still yields cache misses and unnecessary prefetching.

- Keep all previous optimizations and
- create a local copy of A_{kk} and C_{kk} with leading dimension m_b,
- create a local copy of B_{ll} and C_{ll} with leading dimension n_b,
- create a local copy of X_{kl} with leading dimension m_b and copy it to the original location afterwards.

Efficient Inner Solvers

Use local copies of A_{kk}, B_{ll}, C_{kk}, D_{ll}, and X_{kl}

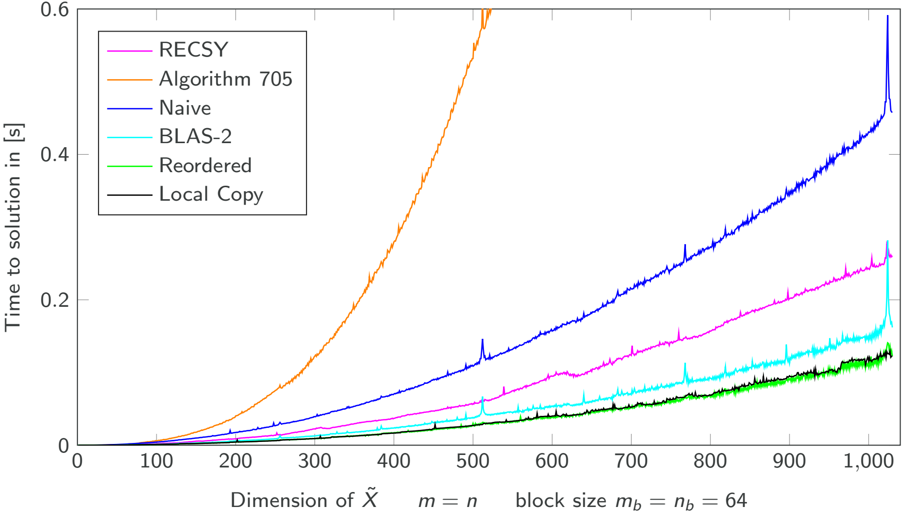
The leading dimension of A_s, B_s, C_s and D_s still yields cache misses and unnecessary prefetching.

- Keep all previous optimizations and
- create a local copy of A_{kk} and C_{kk} with leading dimension m_b,
- create a local copy of B_{ll} and C_{ll} with leading dimension n_b,
- create a local copy of X_{kl} with leading dimension m_b and copy it to the original location afterwards.

All data can be copied to the (L2) cache before the solution of the inner Sylvester equation starts.

Efficient Inner Solvers

Use local copies of A_{kk}, B_{ll}, C_{kk}, D_{ll}, and X_{kl}



Efficient Inner Solvers

Alignment of the local copies

Vector units (AVX, VSX,. . .) of modern CPUs need a special data alignment to work really fast.
→ Without additional help compilers cannot produce efficient code for them.

- Keep all previous optimizations, especially the local copies.

Efficient Inner Solvers

Alignment of the local copies

Vector units (AVX, VSX,. . .) of modern CPUs need a special data alignment to work really fast.
→ Without additional help compilers cannot produce efficient code for them.

- Keep all previous optimizations, especially the local copies.
- Annotate the declaration of the local data such that they are 64-byte aligned:

  ```
  !dir$ attributes align: 64:: AL, BL, CL, DL, XL
  ```

  or

  ```
  !IBM* ALIGN(64,AL,BL,CL,DL,XL)
  ```

  depending on the Fortran compiler.

Efficient Inner Solvers

Alignment of the local copies

Vector units (AVX, VSX,...) of modern CPUs need a special data alignment to work really fast.
→ Without additional help compilers cannot produce efficient code for them.

- Keep all previous optimizations, especially the local copies.
- Annotate the declaration of the local data such that they are 64-byte aligned:

  ```
  !dir$ attributes align: 64:: AL, BL, CL, DL, XL
  ```

  or

  ```
  !IBM* ALIGN(64,AL,BL,CL,DL,XL)
  ```

  depending on the Fortran compiler.
- Replace all BLAS calls except of TRMV with Fortran vector intrinsics.

Efficient Inner Solvers

Alignment of the local copies

Vector units (AVX, VSX,. . .) of modern CPUs need a special data alignment to work really fast.
→ Without additional help compilers cannot produce efficient code for them.

- Keep all previous optimizations, especially the local copies.
- Annotate the declaration of the local data such that they are 64-byte aligned:

  ```
  !dir$ attributes align: 64:: AL, BL, CL, DL, XL
  ```

  or

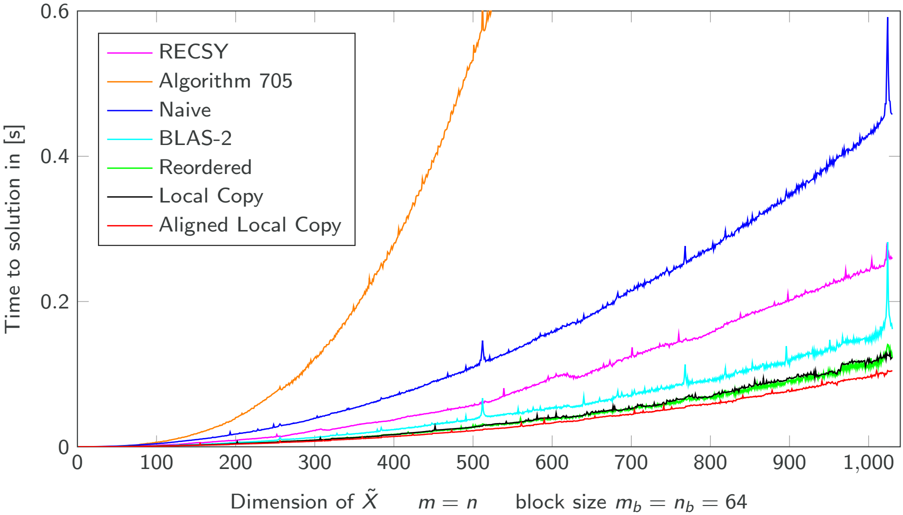
  ```
  !IBM* ALIGN(64,AL,BL,CL,DL,XL)
  ```

  depending on the Fortran compiler.
- Replace all BLAS calls except of TRMV with Fortran vector intrinsics.

→ The compiler should be able to optimize our code.

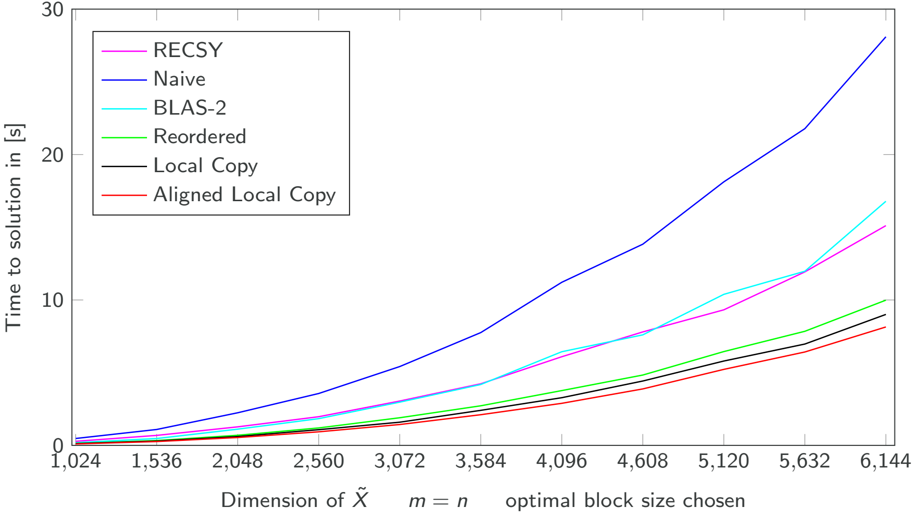
Efficient Inner Solvers

Alignment of the local copies



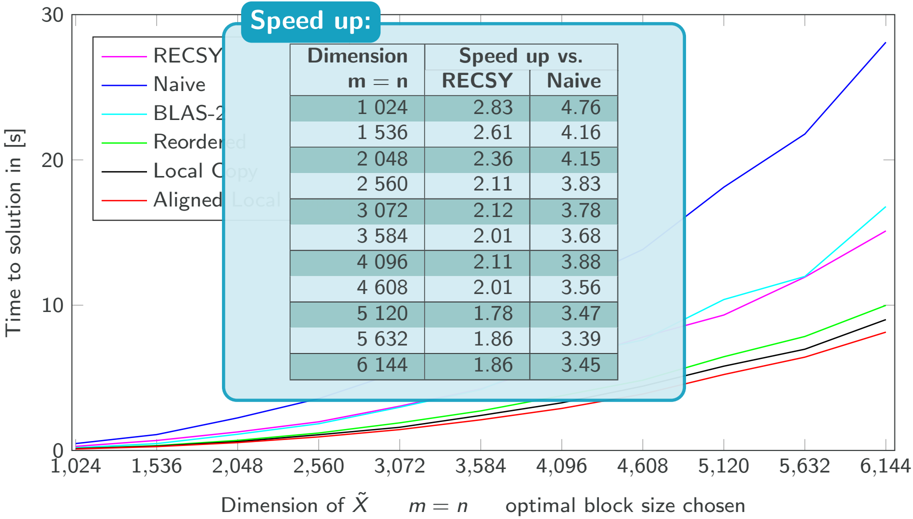
Efficient Inner Solvers

Large Scale Test



Efficient Inner Solvers

Large Scale Test



Speed up:

Dimension	Speed up vs.	
m = n	RECSY	Naive
1 024	2.83	4.76
1 536	2.61	4.16
2 048	2.36	4.15
2 560	2.11	3.83
3 072	2.12	3.78
3 584	2.01	3.68
4 096	2.11	3.88
4 608	2.01	3.56
5 120	1.78	3.47
5 632	1.86	3.39
6 144	1.86	3.45

Further Optimization

Optimal Block Sizes m_b and n_b

The optimal block sizes m_b and n_b must be chosen such that:

- small enough that the inner solvers working nearly inside the (L2) CPU cache
- and large enough that the matrix-matrix products in Algorithm 1 are making use of the multi-threading capabilities of the BLAS library.

Further Optimization

Optimal Block Sizes m_b and n_b

The optimal block sizes m_b and n_b must be chosen such that:

- small enough that the inner solvers working nearly inside the (L2) CPU cache
- and large enough that the matrix-matrix products in Algorithm 1 are making use of the multi-threading capabilities of the BLAS library.

For large m and n and increasing number of CPU cores it is no longer possible to satisfy both conditions properly.

Further Optimization

Optimal Block Sizes m_b and n_b

The optimal block sizes m_b and n_b must be chosen such that:

- small enough that the inner solvers working nearly inside the (L2) CPU cache
- and large enough that the matrix-matrix products in Algorithm 1 are making use of the multi-threading capabilities of the BLAS library.

For large m and n and increasing number of CPU cores it is no longer possible to satisfy both conditions properly.

Idea

Move the parallelization from the BLAS library to the Algorithm.
Use the data dependencies between the $p \cdot q$ blocks of the solution matrix $\tilde{X}$:

- $\tilde{X}_{p1}$ depends on nothing.
- $\tilde{X}_{pj}$, $j = 2 \dots q$ depend on $\tilde{X}_{p,j-1}$.
- $\tilde{X}_{i1}$, $i = p - 1 \dots 1$ depend on $\tilde{X}_{i+1,1}$.
- $\tilde{X}_{ij}$, $i = p - 1 \dots 1$, $j = 2 \dots q$ depend on $\tilde{X}_{i,j-1}$ and $\tilde{X}_{i+1,j}$.

Further Optimization

Direct-Acyclic-Graph (DAG) Scheduling

The data dependencies in the solution lead to the following DAG:

$$\begin{bmatrix}
\vdots & \dots & \vdots & \dots & \vdots & \dots \\
\uparrow & & \uparrow & & \uparrow & \\
\tilde{X}_{p-2,1} & \rightarrow & \tilde{X}_{p-2,2} & \rightarrow & \tilde{X}_{p-2,3} & \dots \\
\uparrow & & \uparrow & & \uparrow & \\
\tilde{X}_{p-1,1} & \rightarrow & \tilde{X}_{p-1,2} & \rightarrow & \tilde{X}_{p-1,3} & \dots \\
\uparrow & & \uparrow & & \uparrow & \\
\tilde{X}_{p,1} & \rightarrow & \tilde{X}_{p,2} & \rightarrow & \tilde{X}_{p,3} & \dots
\end{bmatrix}$$

The blocks $\tilde{X}_{ij}$ on the same anti-diagonal can be solved independently from each other (in parallel).

Further Optimization

Direct-Acyclic-Graph (DAG) Scheduling

The data dependencies in the solution lead to the following DAG:

$$\begin{bmatrix}
\vdots & \cdots & \vdots & \cdots & \vdots & \cdots \\
\uparrow & & \uparrow & & \uparrow & \\
\tilde{X}_{p-2,1} & \rightarrow & \tilde{X}_{p-2,2} & \rightarrow & \tilde{X}_{p-2,3} & \cdots \\
\uparrow & & \uparrow & & \uparrow & \\
\tilde{X}_{p-1,1} & \rightarrow & \tilde{X}_{p-1,2} & \rightarrow & \tilde{X}_{p-1,3} & \cdots \\
\uparrow & & \uparrow & & \uparrow & \\
\tilde{X}_{p,1} & \rightarrow & \tilde{X}_{p,2} & \rightarrow & \tilde{X}_{p,3} & \cdots
\end{bmatrix}$$

The blocks $\tilde{X}_{ij}$ on the same anti-diagonal can be solved independently from each other (in parallel).

OpenMP 4.0 – `task depend`

Since OpenMP 4.0 such data dependencies can be attached to the `omp task` directive and the OpenMP runtime system does the scheduling with respect to the DAG.

Further Optimization

Direct-Acyclic-Graph (DAG) Scheduling

The data dependencies in the solution lead to the following DAG:

$$\begin{tikzcd}
\vdots & \vdots & \vdots & \\
\tilde{X}_{p-2,1} \arrow[u] \arrow[r] & \tilde{X}_{p-2,2} \arrow[u] \arrow[r] & \tilde{X}_{p-2,3} \arrow[u] & \cdots \\
\tilde{X}_{p-1,1} \arrow[u] \arrow[r] & \tilde{X}_{p-1,2} \arrow[u] \arrow[r] & \tilde{X}_{p-1,3} \arrow[u] & \cdots \\
\tilde{X}_{p,1} \arrow[u] \arrow[r] & \tilde{X}_{p,2} \arrow[u] \arrow[r] & \tilde{X}_{p,3} \arrow[u] & \cdots
\end{tikzcd}$$

The blocks $\tilde{X}_{ij}$ on the same anti-diagonal can be solved independently from each other (in parallel).

OpenMP 4.0 – task depend

Since OpenMP 4.0 such data dependencies can be attached to the `omp task` directive and the OpenMP runtime system does the scheduling with respect to the DAG.

Sketch of the Implementation:

OpenMP parallelization is done using annotations in the source code:

```
CALL TGSYLV_SOLVE_BLOCK(TRANSA, TRANSB, M,N,K,KH,KB,L,LH,LB, &
 & A,LDA,B,LDB,C,LDC,D,LDD,X,LDX,SGN,SCAL,WORK,INFO1,ARCH )
```

Further Optimization

Direct-Acyclic-Graph (DAG) Scheduling

The data dependencies in the solution lead to the following DAG:

Sketch of the Implementation:

OpenMP parallelization is done using annotations in the source code:

```
!$omp task firstprivate(K,KH,KB,L,LH,LB,SCAL,INFO1)
!$omp& depend(in:  X(KOLD,L), X(K,LOLD)) depend(out:  X(K,L))
!$omp& default(shared)
CALL TGSYLV_SOLVE_BLOCK(TRANSA, TRANSB, M,N,K,KH,KB,L,LH,LB, &
 & A,LDA,B,LDB,C,LDC,D,LDD,X,LDX,SGN,SCAL,WORK,INFO1,ARCH )
!$omp end task
```

The blocks $\tilde{X}_{ij}$ on the same anti-diagonal can be solved independently from each other (in parallel).

OpenMP 4.0 – `task depend`

Since OpenMP 4.0 such data dependencies can be attached to the `omp task` directive and the OpenMP runtime system does the scheduling with respect to the DAG.

Further Optimization

Direct-Acyclic-Graph (DAG) Scheduling

The data dependencies in the solution lead to the following DAG:

Sketch of the Implementation:

OpenMP parallelization is done using annotations in the source code:

```
!$omp task firstprivate(K,KH,KB,L,LH,LB,SCAL,INFO1)
!$omp& depend(in:  X(KOLD,L), X(K,LOLD)) depend(out:  X(K,L))
!$omp& default(shared)
CALL TGSYLV_SOLVE_BLOCK(TRANSA, TRANSB, M,N,K,KH,KB,L,LH,LB, &
 & A,LDA,B,LDB,C,LDC,D,LDD,X,LDX,SGN,SCAL,WORK,INFO1,ARCH )
!$omp end task
```

→ Annotations are ignored by non OpenMP compatible compilers.

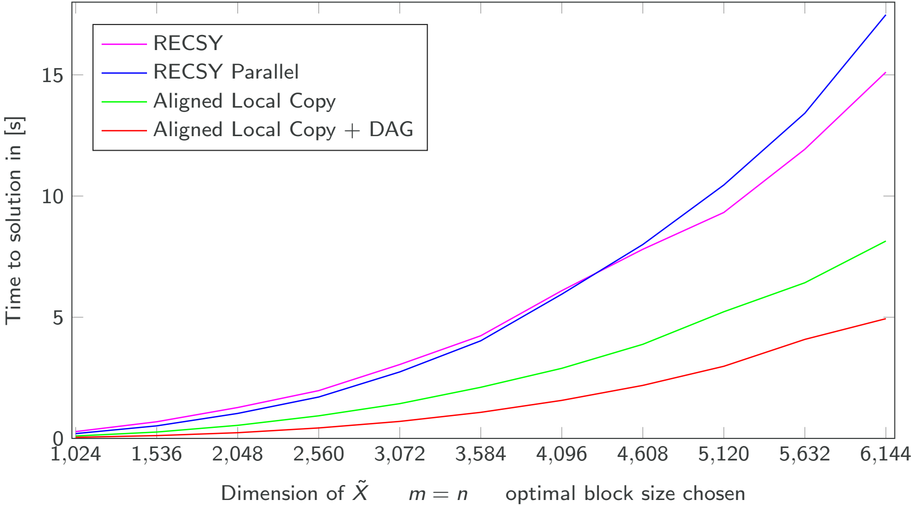
The blocks $\tilde{X}_{ij}$ on the same anti-diagonal can be solved independently from each other (in parallel).

OpenMP 4.0 – task depend

Since OpenMP 4.0 such data dependencies can be attached to the `omp task` directive and the OpenMP runtime system does the scheduling with respect to the DAG.

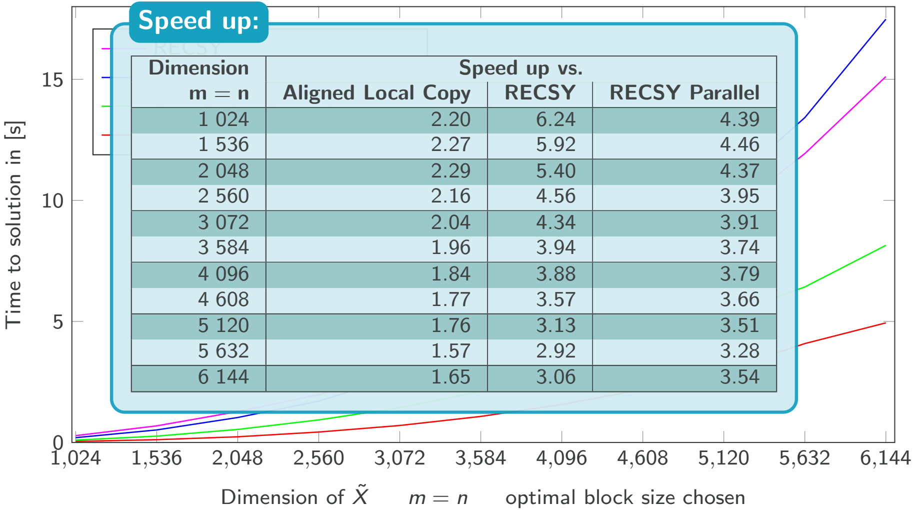
Further Optimization

Direct-Acyclic-Graph (DAG) Scheduling



Further Optimization

Direct-Acyclic-Graph (DAG) Scheduling



Speed up:

Dimension	Speed up vs.		
m = n	Aligned Local Copy	RECSY	RECSY Parallel
1 024	2.20	6.24	4.39
1 536	2.27	5.92	4.46
2 048	2.29	5.40	4.37
2 560	2.16	4.56	3.95
3 072	2.04	4.34	3.91
3 584	1.96	3.94	3.74
4 096	1.84	3.88	3.79
4 608	1.77	3.57	3.66
5 120	1.76	3.13	3.51
5 632	1.57	2.92	3.28
6 144	1.65	3.06	3.54

Conclusions

- Block Bartels-Stewart algorithms can beat the RECSY approach by a factor of 2.8 or 6.2.
- Performance gain of a factor 4.7 if the code is written in a way such that the compiler can optimize the code properly.
- DAG-Scheduling in OpenMP 4 allows easy high level parallelization on top of the data dependencies. (GCC since 4.9.1, Intel since 15.0, LLVM since 3.9)
- Same ideas work for LYAP, STEIN, SYLV, SYLV2, GLYAP, GSTEIN and CSYLV.

Conclusions

- Block Bartels-Stewart algorithms can beat the RECSY approach by a factor of 2.8 or 6.2.
- Performance gain of a factor 4.7 if the code is written in a way such that the compiler can optimize the code properly.
- DAG-Scheduling in OpenMP 4 allows easy high level parallelization on top of the data dependencies. (GCC since 4.9.1, Intel since 15.0, LLVM since 3.9)
- Same ideas work for LYAP, STEIN, SYLV, SYLV2, GLYAP, GSTEIN and CSYLV.

Outlook

- GPU/Accelerator enabled implementations.
- Compile-time tuning by benchmarks of xGEMM and xTRMM.
- DAG-scheduling like algorithms if OpenMP 4 features are not available.

Conclusions

- Block Bartels-Stewart algorithms can beat the RECSY approach by a factor of 2.8 or 6.2.
- Performance gain of a factor 4.7 if the code is written in a way such that the compiler can optimize the code properly.
- DAG-Scheduling in OpenMP 4 allows easy high level parallelization on top of the data dependencies. (GCC since 4.9.1, Intel since 15.0, LLVM since 3.9)
- Same ideas work for LYAP, STEIN, SYLV, SYLV2, GLYAP, GSTEIN and CSYLV.

Thank you for your attention.

Outlook

- GPU/Accelerator enabled implementations.
- Compile-time tuning by benchmarks of xGEMM and xTRMM.
- DAG-scheduling like algorithms if OpenMP 4 features are not available.