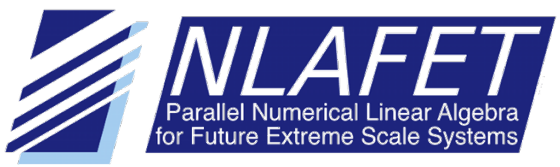
Parallel robust solution of triangular equations

Carl Christian Kjelgaard Mikkelsen

Department of Computing Science
Umeå University

Nordic Numerical Linear Algebra Meeting
KTH, Stockholm
October 21-22th, 2019



What is a robust algorithm?

We say that an algorithm is robust iff

all intermediate and final results are in the representable range

What is a robust algorithm?

We say that an algorithm is robust iff

all intermediate and final results are in the representable range

A robust algorithm returns a result which

is free of `Inf` and `NaN`

and can be evaluated in the context of the user's application.

Example 1: $\boldsymbol{Ax} = \boldsymbol{b}$

$$\boldsymbol{Ax} := \begin{bmatrix} 1 & & & & \\ -1 & 1 & & & \\ -1 & -1 & 1 & & \\ -1 & -1 & -1 & 1 & \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 4 \\ 8 \\ 16 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} =: \boldsymbol{b}$$

Example 2: $\boldsymbol{SX} = \boldsymbol{X}\Lambda$

Matrix:

$$\boldsymbol{S} = \begin{bmatrix} 1 & & & & \\ -5 & 2 & & & \\ -5 & -5 & 3 & & \\ -5 & -5 & -5 & 4 & \\ -5 & -5 & -5 & -5 & 5 \end{bmatrix}$$

Eigenvectors:

$$\boldsymbol{X} = \begin{bmatrix} 1 & & & & \\ 5 & 1 & & & \\ 15 & 5 & 1 & & \\ 35 & 15 & 5 & 1 & \\ 70 & 35 & 15 & 5 & 1 \end{bmatrix}$$

Example 3: $\boldsymbol{AX} + \boldsymbol{XA}^T = \boldsymbol{bb}^T$

Matrices:

$$\boldsymbol{A} = \begin{bmatrix} \frac{1}{2} & & & & \\ -1 & \frac{1}{2} & & & \\ -1 & -1 & \frac{1}{2} & & \\ -1 & -1 & -1 & \frac{1}{2} & \\ -1 & -1 & -1 & -1 & \frac{1}{2} \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Solution:

$$\boldsymbol{X} = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 2 & 5 & 12 & 28 & 64 \\ 4 & 12 & 33 & 86 & 216 \\ 8 & 28 & 86 & 245 & 664 \\ 16 & 64 & 216 & 664 & 1921 \end{bmatrix}$$

Robust triangular solvers in LAPACK

Routines are scalar, sequential codes descended from `xlatrs`.

Robust triangular solvers in LAPACK

Routines are scalar, sequential codes descended from `xlatrs`.

Subroutine	Problem	Comment
`xlatrs`	$\boldsymbol{A}\boldsymbol{x} = \alpha\boldsymbol{b}$	written 1991
`xtrevc`	$\boldsymbol{A}\boldsymbol{X} = \boldsymbol{X}\boldsymbol{\Lambda}$	
`xtgevc`	$\boldsymbol{A}\boldsymbol{X} = \boldsymbol{B}\boldsymbol{X}\boldsymbol{\Lambda}$	
`xtrsyl`	$\boldsymbol{A}\boldsymbol{X} + \boldsymbol{X}\boldsymbol{B} = \alpha\boldsymbol{C}$	

Robust triangular solvers in ScaLAPACK?

From the source code of `pztrevc`[1], lines 186:190

The algorithm used in this program is basically backward (forward) substitution. It is the hope that scaling would be used to make the the code robust against possible overflow. But scaling has not yet been implemented in PZLATTRS which is called by this routine to solve the triangular systems. PZLATTRS just calls PZTRSV.

[1] `http://www.netlib.org/scalapack/explore-html/d5/dbc/pztrevc_8f_source.html` Retrieved: October 19th 2019.

Robust backward substitution, $\boldsymbol{T}\boldsymbol{x} = \alpha \boldsymbol{b}$

Algorithm 1: $[\alpha, \boldsymbol{x}] = \texttt{RobustBackSub}(\boldsymbol{T}, \boldsymbol{b})$

$\alpha \leftarrow 1$;
for $j \leftarrow n, n-1, \ldots, 1$ **do**
  $\beta = \texttt{ProtectDivision}(x_j, t_{jj})$;
  if $\beta \neq 1$ **then**
    $\boldsymbol{x} \leftarrow \beta \boldsymbol{x}, \alpha \leftarrow \beta\alpha$;
  $x_j \leftarrow x_j / t_{jj}$;
  if $j > 1$ **then**
    $\beta = \texttt{ProtectUpdate}(x_{\max}, \|\boldsymbol{T}_{1:j-1,j}\|_\infty, |x_j|)$;
    if $\beta \neq 1$ **then**
      $\boldsymbol{x} \leftarrow \beta \boldsymbol{x}, \alpha \leftarrow \beta\alpha$;
    $\boldsymbol{x}_{1:j-1} \leftarrow \boldsymbol{x}_{1:j-1} - \boldsymbol{T}_{1:j-1,j} x_j$;
    $x_{\max} \leftarrow \|\boldsymbol{x}_{1:j-1}\|_\infty$;
return $[\alpha, \boldsymbol{x}]$;

ProtectDivision

The division $y \leftarrow b/t$ is safe if

$$|b| \leq |t|\Omega.$$

ProtectDivision

The division $y \leftarrow b/t$ is safe if

$$|b| \leq |t|\Omega.$$

Problem: Find scaling $\alpha \in (0, 1]$ such that

$$|\alpha b| \leq |t|\Omega.$$

ProtectDivision

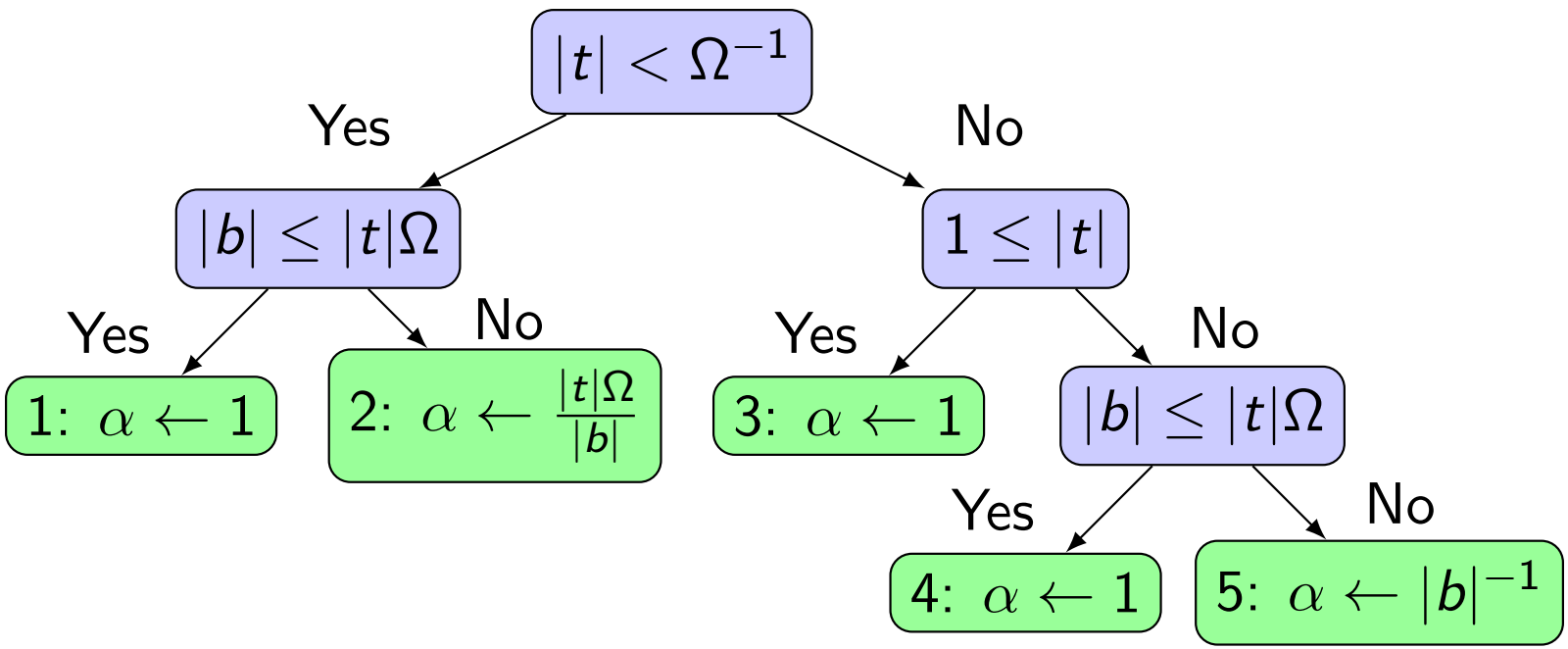
The division $y \leftarrow b/t$ is safe if

$$|b| \leq |t|\Omega.$$

Problem: Find scaling $\alpha \in (0, 1]$ such that

$$|\alpha b| \leq |t|\Omega.$$

Solution:



ProtectUpdate

The linear update $\boldsymbol{y} \leftarrow \boldsymbol{y} - \boldsymbol{T}\boldsymbol{x}$ is safe if

$$\|\boldsymbol{y}\|_\infty + \|\boldsymbol{T}\|_\infty \|\boldsymbol{x}\|_\infty \leq \Omega.$$

ProtectUpdate

The linear update $\boldsymbol{y} \leftarrow \boldsymbol{y} - \boldsymbol{T}\boldsymbol{x}$ is safe if

$$\|\boldsymbol{y}\|_\infty + \|\boldsymbol{T}\|_\infty \|\boldsymbol{x}\|_\infty \leq \Omega.$$

Problem: Find $\zeta \in (0, 1]$ such that

$$\|\zeta\boldsymbol{y}\|_\infty + \|\boldsymbol{T}\|_\infty \|\zeta\boldsymbol{x}\|_\infty \leq \Omega.$$

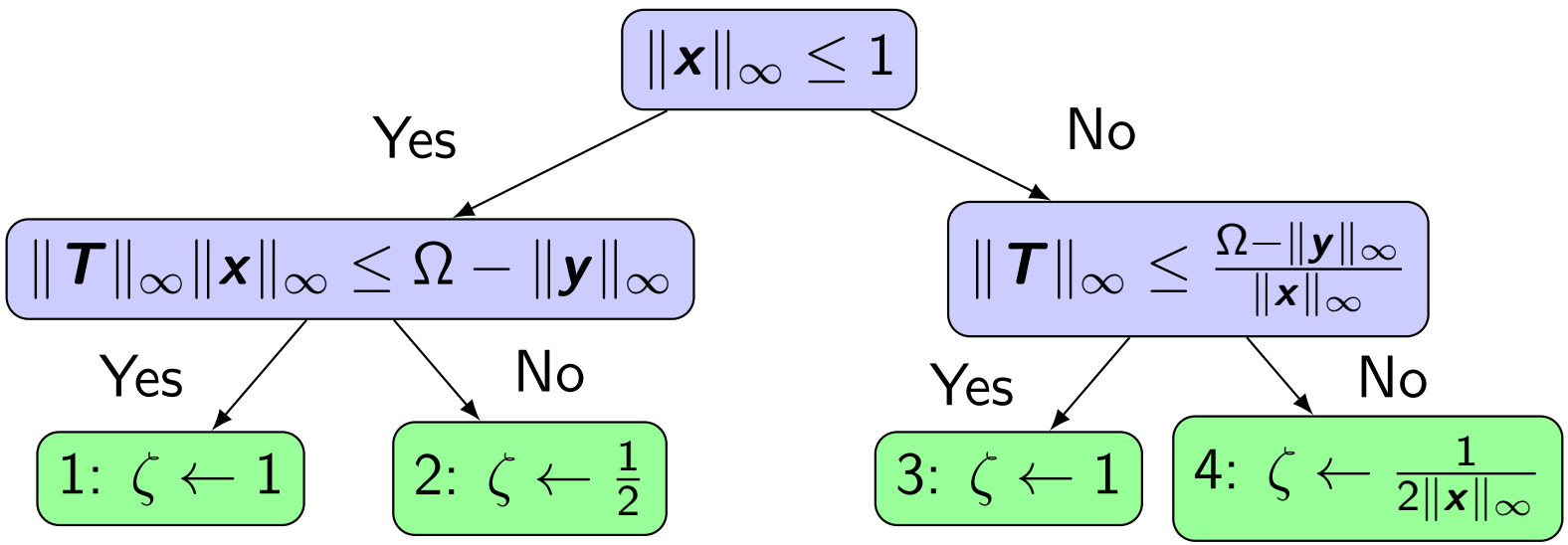
ProtectUpdate

The linear update $\boldsymbol{y} \leftarrow \boldsymbol{y} - \boldsymbol{T}\boldsymbol{x}$ is safe if

$$\|\boldsymbol{y}\|_\infty + \|\boldsymbol{T}\|_\infty \|\boldsymbol{x}\|_\infty \leq \Omega.$$

Problem: Find $\zeta \in (0, 1]$ such that

$$\|\zeta\boldsymbol{y}\|_\infty + \|\boldsymbol{T}\|_\infty \|\zeta\boldsymbol{x}\|_\infty \leq \Omega.$$



How to develop blocked, parallel and robust codes?

The main idea:

Different processing units can use different units of measurement ...

Step 1: Partition the problem

$$\begin{bmatrix} T_{11} & T_{12} & \dots & T_{1M} \\ & T_{22} & \dots & T_{2m} \\ & & \ddots & \vdots \\ & & & T_{MM} \end{bmatrix} \begin{bmatrix} X_{11} & X_{12} & \dots & X_{1N} \\ X_{21} & \ddots & \ddots & X_{2N} \\ \vdots & \ddots & \ddots & \vdots \\ X_{M1} & \dots & \dots & X_{MN} \end{bmatrix}$$

$$= \begin{bmatrix} B_{11} & B_{12} & \dots & B_{1N} \\ B_{21} & \ddots & \ddots & B_{2N} \\ \vdots & \ddots & \ddots & \vdots \\ B_{M1} & \dots & \dots & B_{MN} \end{bmatrix}$$

Step 2: Augment each block of data

A separate scaling factor α_{ijk} for each column of every block $\boldsymbol{X}_{ij}$.

Step 2: Augment each block of data

A separate scaling factor α_{ijk} for each column of every block $\boldsymbol{X}_{ij}$.

An augmented block

$$\langle \boldsymbol{\alpha}, \boldsymbol{X} \rangle$$

where

$$\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n), \quad \alpha_j \in (0, 1], \quad \boldsymbol{X} \in \mathbb{R}^{m \times n}$$

is used to represents the matrix

$$\boldsymbol{Y} \in \mathbb{R}^{m \times n}$$

where

$$\boldsymbol{y}_j = \alpha_j^{-1} \boldsymbol{x}_j, \quad j = 1, 2, \ldots, n.$$

Step 3: Replace kernels with robust kernels ...

Problem: Given $\langle \boldsymbol{\alpha}, \boldsymbol{X} \rangle$ and $\langle \boldsymbol{\beta}, \boldsymbol{Y} \rangle$ find $\langle \boldsymbol{\nu}, \boldsymbol{Z} \rangle$ such that

$$\nu_j^{-1} \boldsymbol{z}_j = \beta_j^{-1} \boldsymbol{y}_j - \alpha_j^{-1} \boldsymbol{T} \boldsymbol{x}_j, \quad j = 1, 2, \ldots, n$$

using BLAS level-3 operations.

Step 3: Replace kernels with robust kernels ...

Problem: Given $\langle \boldsymbol{\alpha}, \boldsymbol{X} \rangle$ and $\langle \boldsymbol{\beta}, \boldsymbol{Y} \rangle$ find $\langle \boldsymbol{\nu}, \boldsymbol{Z} \rangle$ such that

$$\nu_j^{-1} \boldsymbol{z}_j = \beta_j^{-1} \boldsymbol{y}_j - \alpha_j^{-1} \boldsymbol{T} \boldsymbol{x}_j, \quad j = 1, 2, \ldots, n$$

using BLAS level-3 operations.

Algorithm 3: $\langle \boldsymbol{\nu}, \mathbf{Z} \rangle \leftarrow$ `RobustGEMM`$(\langle \boldsymbol{\beta}, \mathbf{Y} \rangle, \mathbf{T}, \langle \boldsymbol{\alpha}, \mathbf{X} \rangle)$

1 **for** $j \leftarrow 1, 2, \ldots, n$ **do**
2 $\quad \gamma_j \leftarrow \min\{\alpha_j, \beta_j\}$;
3 $\quad \zeta_j \leftarrow$ `ProtectUpdate`$((\gamma_j/\beta_j)\|\mathbf{y}_j\|_\infty, \|\mathbf{T}\|_\infty, (\gamma_j/\alpha_j)\|\mathbf{x}_j\|_\infty)$;
4 $\quad \nu_j \leftarrow \zeta_j \gamma_j$;
5 $\mathbf{Z} \leftarrow \Big[\mathbf{Y}\mathrm{diag}(\boldsymbol{\nu} \oslash \boldsymbol{\beta})\Big] - \mathbf{T}\Big[\mathbf{X}\mathrm{diag}(\boldsymbol{\nu} \oslash \boldsymbol{\alpha})\Big]$;
6 **Return** $\langle \boldsymbol{\nu}, \mathbf{Z} \rangle$;

Robust solution of partitioned system $\boldsymbol{TX} = \boldsymbol{B}$.

Algorithm 4: $\langle \alpha, \mathbf{X} \rangle \leftarrow$ `RobustBlockedSolveMRHS`$(\mathbf{T}, \mathbf{B})$

for $(i, j) \in \{1, 2, \ldots, M\} \times \{1, 2, \ldots, N\}$ **do**
  $\langle \boldsymbol{\alpha}_{ij}, \mathbf{X}_{ij} \rangle \leftarrow \langle \mathbf{e}, \mathbf{B}_{ij} \rangle$;
for $j \leftarrow M, M-1, \ldots, 1$ **do**
  for $k \leftarrow 1, 2, \ldots, N$ **do**
    $\langle \boldsymbol{\alpha}_{jk}, \mathbf{X}_{jk} \rangle \leftarrow$ `RobustTRSM`$(\mathbf{T}_{jj}, \langle \boldsymbol{\alpha}_{jk}, \mathbf{X}_{jk} \rangle)$;
  for $i \leftarrow 1, 2, \ldots, j-1$ **do**
    for $k \leftarrow 1, 2, \ldots, N$ **do**
      $\langle \boldsymbol{\alpha}_{ik}, \mathbf{X}_{ik} \rangle \leftarrow$ `RobustGEMM`$(\langle \boldsymbol{\alpha}_{ik}, \mathbf{X}_{ik} \rangle, \mathbf{T}_{ij}, \langle \boldsymbol{\alpha}_{jk}, \mathbf{X}_{jk} \rangle)$;
for $k \leftarrow 1, 2, \ldots, N$ **do**
  $\boldsymbol{\alpha}_k \leftarrow \min_i \{\boldsymbol{\alpha}_{ik}\}$;
  for $i \leftarrow 1, 2, \ldots, M$ **do**
    $\mathbf{X}_{ik} \leftarrow \mathbf{X}_{ik} \mathrm{diag}(\boldsymbol{\alpha}_k \oslash \boldsymbol{\alpha}_{ik})$;
$\boldsymbol{\alpha} \leftarrow \begin{bmatrix} \boldsymbol{\alpha}_1 & \boldsymbol{\alpha}_2 & \cdots & \boldsymbol{\alpha}_N \end{bmatrix}$;
Return $\langle \alpha, \mathbf{X} \rangle$;

Performance degradation of robust solvers?

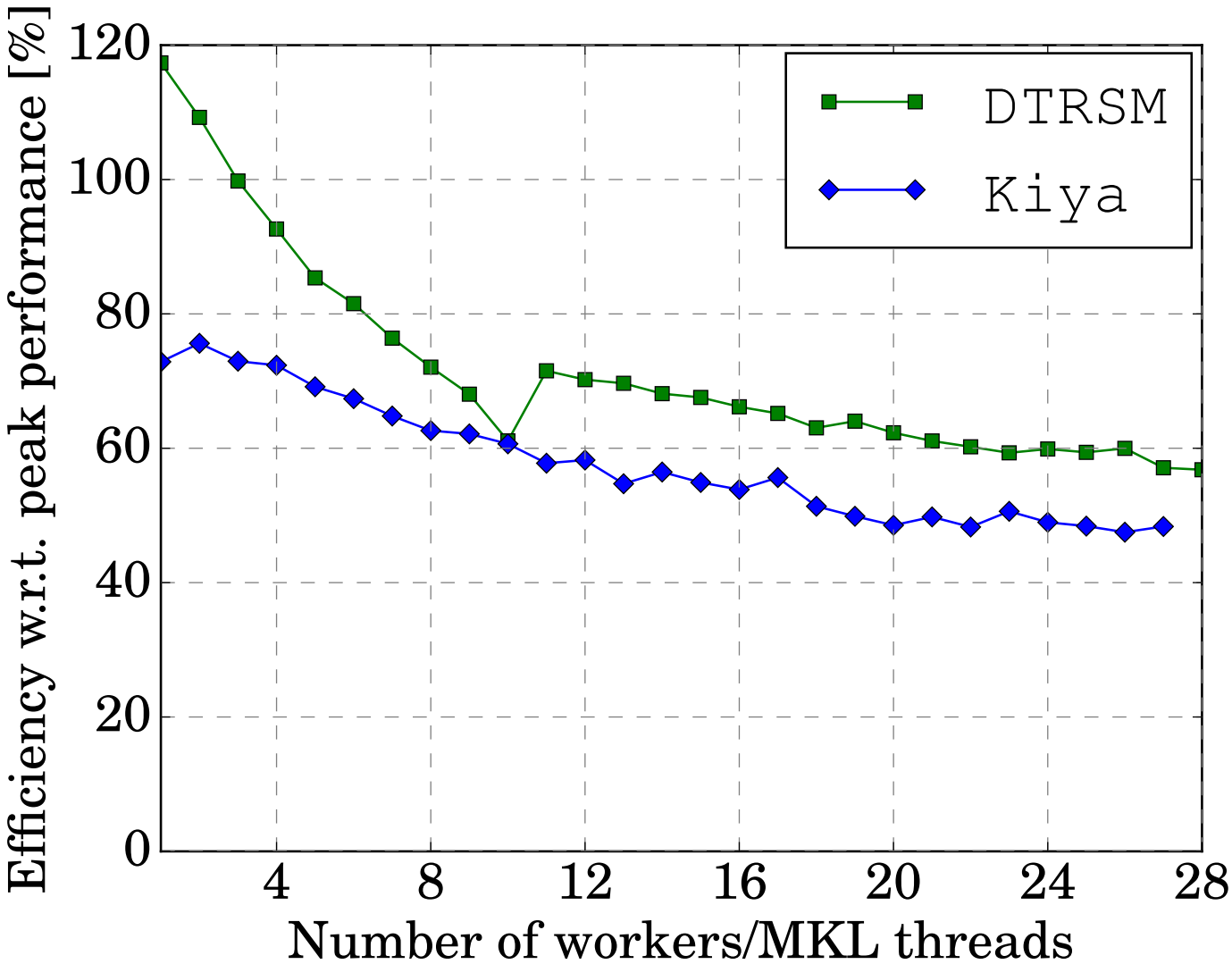


Figure: Triangular linear system $m = 40000$ and $n = 2000$

Caution: Underflow can still destroy the result!

$$\left[\begin{array}{c|ccccc} \frac{1}{2} & & & & & \\ \hline & M & & & & \\ & -2 & 1 & & & \\ & & \ddots & \ddots & & \\ & & & & -2 & 1 \end{array}\right] \left[\begin{array}{c} 2\Omega \\ \hline \frac{\epsilon}{M} \\ \frac{2^1\epsilon}{M} \\ \vdots \\ \frac{2^{m-1}\epsilon}{M} \end{array}\right] = \left[\begin{array}{c} \Omega \\ \hline \epsilon \\ 0 \\ \vdots \\ 0 \end{array}\right].$$

New robust solvers

During the NLAFET project, UMU has made robust solvers for

1. Standard linear systems, SM, StarPU:
2. Eigenvectors
   - 2.1 Standard case, real arithmetic, SM: StarNeig
   - 2.2 Generalized case, real arithmetic, SM: StarNeig
3. Sylvester matrix equations, SM, OpenMP.

New robust solvers

During the NLAFET project, UMU has made robust solvers for

1. Standard linear systems, SM, StarPU:
2. Eigenvectors
   - 2.1 Standard case, real arithmetic, SM: StarNeig
   - 2.2 Generalized case, real arithmetic, SM: StarNeig
3. Sylvester matrix equations, SM, OpenMP.

NLAFET at UMU:

Mirko Myllykoski
Carl Christian Kjelgaard Mikkelsen
Angelika Schwartz
Lars Karlsson
Bo Kågström
Björn Adlerborn
Mahmoud Eljammally

Interesting problems which can now be solved ...

1. Catastrophic underflow in triangular equations can be avoided.
2. Proof of correctness of `ztgevc` which solves

$$\boldsymbol{SZ} = \boldsymbol{TZ}\boldsymbol{\Lambda}.$$

3. Proof of correctness of `dladiv` which does complex division.

$$p + iq = \frac{a + ib}{c + id}.$$