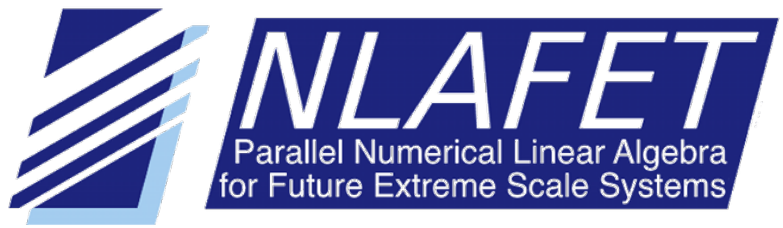
Blocked Algorithms for Robust Solution of Triangular Linear Systems

Carl Christian Kjelgaard Mikkelsen and Lars Karlsson

Department of Computing Science and HPC2N
Umeå University

PPAM-2017



My coauthor

Figure: Lars Karlsson

Main problem: Parallel Overflow Protection

$$\begin{bmatrix} \frac{1}{2} & & & & & & & \\ -1 & \frac{1}{2} & & & & & & \\ -1 & -1 & \frac{1}{2} & & & & & \\ -1 & -1 & -1 & \frac{1}{2} & & & & \\ -1 & -1 & -1 & -1 & \frac{1}{2} & & & \\ -1 & -1 & -1 & -1 & -1 & \frac{1}{2} & & \\ -1 & -1 & -1 & -1 & -1 & -1 & \frac{1}{2} & \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 12 \\ 36 \\ 108 \\ 324 \\ 972 \\ 2916 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$x_1 = 2, \quad x_j = 4 \cdot 3^{j-2}, \quad j \geqslant 2$$

Main problem: Parallel Overflow Protection

$$\begin{bmatrix} a & & & & & & & \\ -b & a & & & & & & \\ -b & -b & a & & & & & \\ -b & -b & -b & a & & & & \\ -b & -b & -b & -b & a & & & \\ -b & -b & -b & -b & -b & a & & \\ -b & -b & -b & -b & -b & -b & a & \\ -b & -b & -b & -b & -b & -b & -b & a \end{bmatrix} \begin{bmatrix} \gamma \\ \gamma^2 \\ \gamma^2(1+\gamma) \\ \gamma^2(1+\gamma)^2 \\ \gamma^2(1+\gamma)^3 \\ \gamma^2(1+\gamma)^4 \\ \gamma^2(1+\gamma)^5 \\ \gamma^2(1+\gamma)^6 \end{bmatrix} = \begin{bmatrix} b \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\gamma = \frac{b}{a}, \quad x_1 = \gamma, \quad x_j = \gamma^2 \cdot (1+\gamma)^{j-2}, \quad j \geqslant 2$$

Why should you bother?

Forward/backward substitution is as good as it gets, i.e.,

a small relative componentwise backward error,

so why bother writing a "robust" algorithm?

Overflow is

- ... possible even for small systems
- ... likely when computing eigenvectors for clustered eigenvalues

Previous work: Sequential solution

LAPACK

- xLATRS (Ed Anderson) and it descendants solve

$$Tx = \alpha b$$

  with respect to vector x and scalar α.
- Scaling α protects against overflow in computation of x

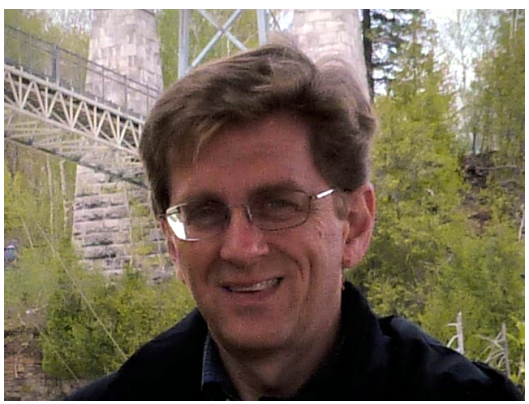


Figure: Edward Anderson

Previous work: Sequential solution

LAPACK

1. A scaling β is computed so that

$$x \leftarrow (\beta b)/t$$

   can not overflow.

2. Similarly, a scaling ξ is computed so that

$$\mathbf{y} \leftarrow (\xi \mathbf{y}) - \mathbf{t}(\xi x)$$

   can not overflow.

3. All scalings are applied to the RHS and accumulated in α.

Exposing the parallel bottleneck

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

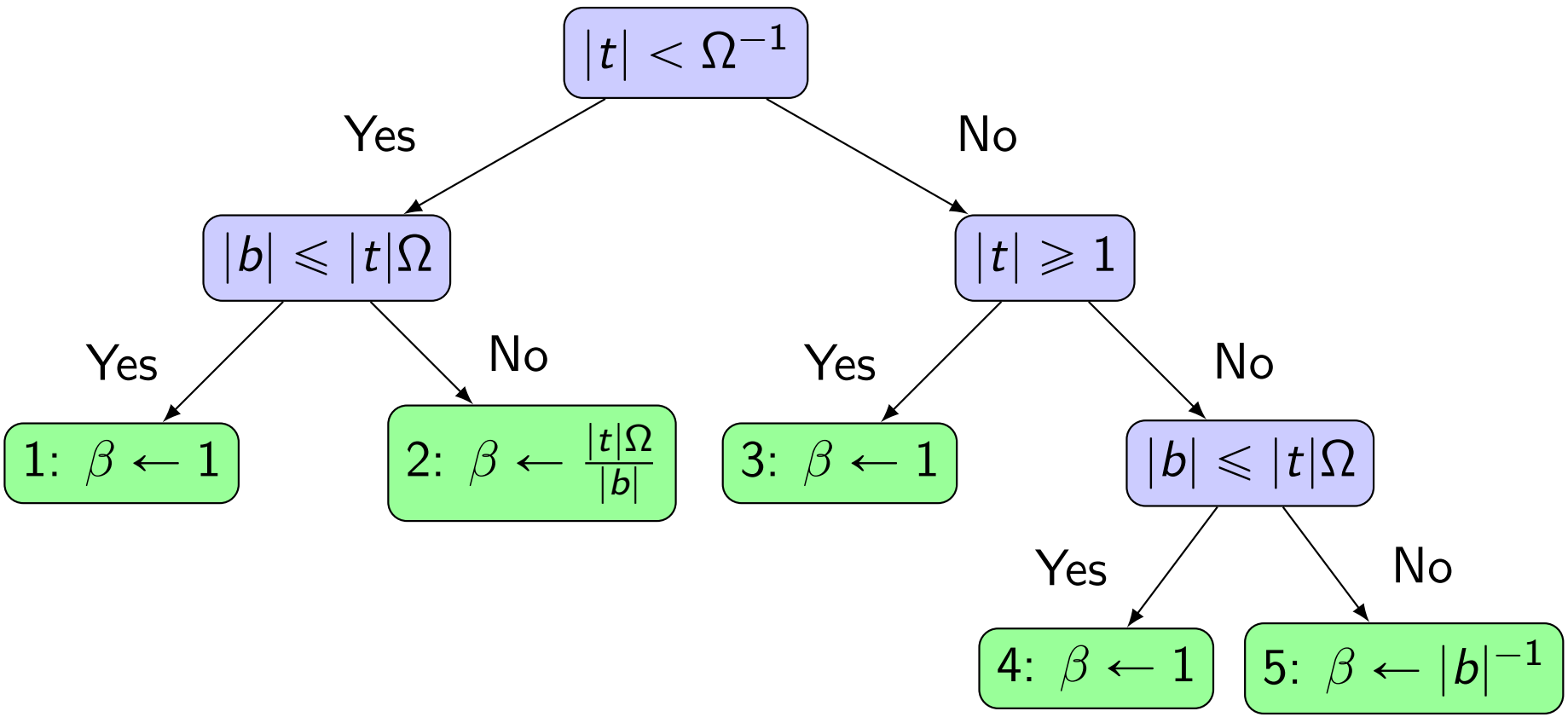
Find c_j such that $\|T_{1:j-1,j}\|_\infty \leqslant c_j \leqslant \Omega$ for $j = 2, 3, \ldots, n$.;
$\alpha \leftarrow 1$, $\mathbf{x} \leftarrow \mathbf{b}$, $x_{\max} \leftarrow \|\mathbf{x}\|_\infty$;
for $j \leftarrow n, n-1, \ldots, 1$ **do**
    $\beta = \texttt{ProtectDivision}(x_j, t_{jj})$;
    if $\beta \neq 1$ **then**
        $\mathbf{x} \leftarrow \beta\mathbf{x}$, $\alpha \leftarrow \beta\alpha$;
    $x_j \leftarrow x_j / t_{jj}$;
    if $j > 1$ **then**
        $\beta = \texttt{ProtectUpdate}(x_{\max}, c_j, |x_j|)$;
        if $\beta \neq 1$ **then**
            $\mathbf{x} \leftarrow \beta\mathbf{x}, \alpha \leftarrow \beta\alpha$;
        $\mathbf{x_{1:j-1}} \leftarrow \mathbf{x_{1:j-1}} - \mathbf{T_{1:j-1,j}}\mathbf{x_j}$;
        $x_{\max} \leftarrow \|\mathbf{x}_{1:j-1}\|_\infty$;
return $[\alpha, \mathbf{x}]$;

ProtectDivision

Compute $\beta \in (0, 1]$ such that

$$x \leftarrow \frac{(\beta b)}{t}$$

cannot exceed Ω.

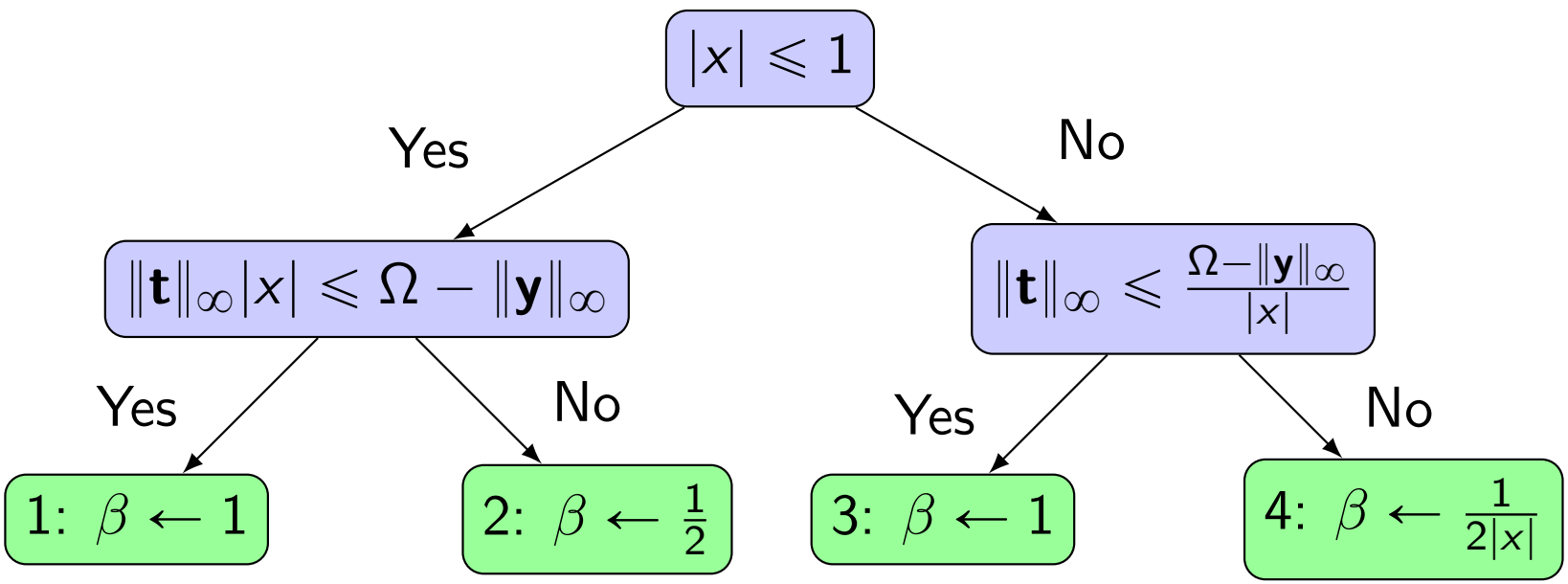


ProtectUpdate

Compute $\beta \in (0, 1]$ such that the components of

$$\mathbf{y} \leftarrow \mathbf{y} - \mathbf{t}x$$

cannot exceed Ω.



Partitioned linear systems

A partitioned linear system

$$\begin{bmatrix} \mathbf{T}_{11} & \mathbf{T}_{12} & \dots & \mathbf{T}_{1N} \\ & \mathbf{T}_{22} & \dots & \mathbf{T}_{2N} \\ & & \ddots & \vdots \\ & & & \mathbf{T}_{NN} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_N \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_N \end{bmatrix}$$

can be solved in parallel using a task based runtime system.

Non robust blocked solvers

Algorithm 2: $\mathbf{x} = \texttt{BlockedSolve}(\mathbf{T}, \mathbf{b})$

```
1 x ← b;
2 for j = N, N − 1, . . . , 1 do
3 |   x_j ← f(T_jj, b_j);
4 |   for i = 1, 2, . . . , j − 1 do
5 |   |   x_i ← g(x_i, T_ij, x_j);
6 return x;
```

The standard kernels

$$f(\mathbf{T}, \mathbf{b}) = \mathbf{T}^{-1}\mathbf{b}, \quad g(\mathbf{y}, \mathbf{T}, \mathbf{x}) = \mathbf{y} - \mathbf{T}\mathbf{x}.$$

are not robust.

Our contribution: Robust kernels

Our kernels operate on "augmented vectors"

$$\langle \alpha, \mathbf{x} \rangle, \quad \alpha \in (0, 1], \quad \mathbf{x} \in \mathbb{R}^n$$

An augmented vector $\langle \alpha, \mathbf{x} \rangle$ represents the real vector

$$\alpha^{-1}\mathbf{x}$$

The scaling α prevents overflow in $\mathbf{x}$.

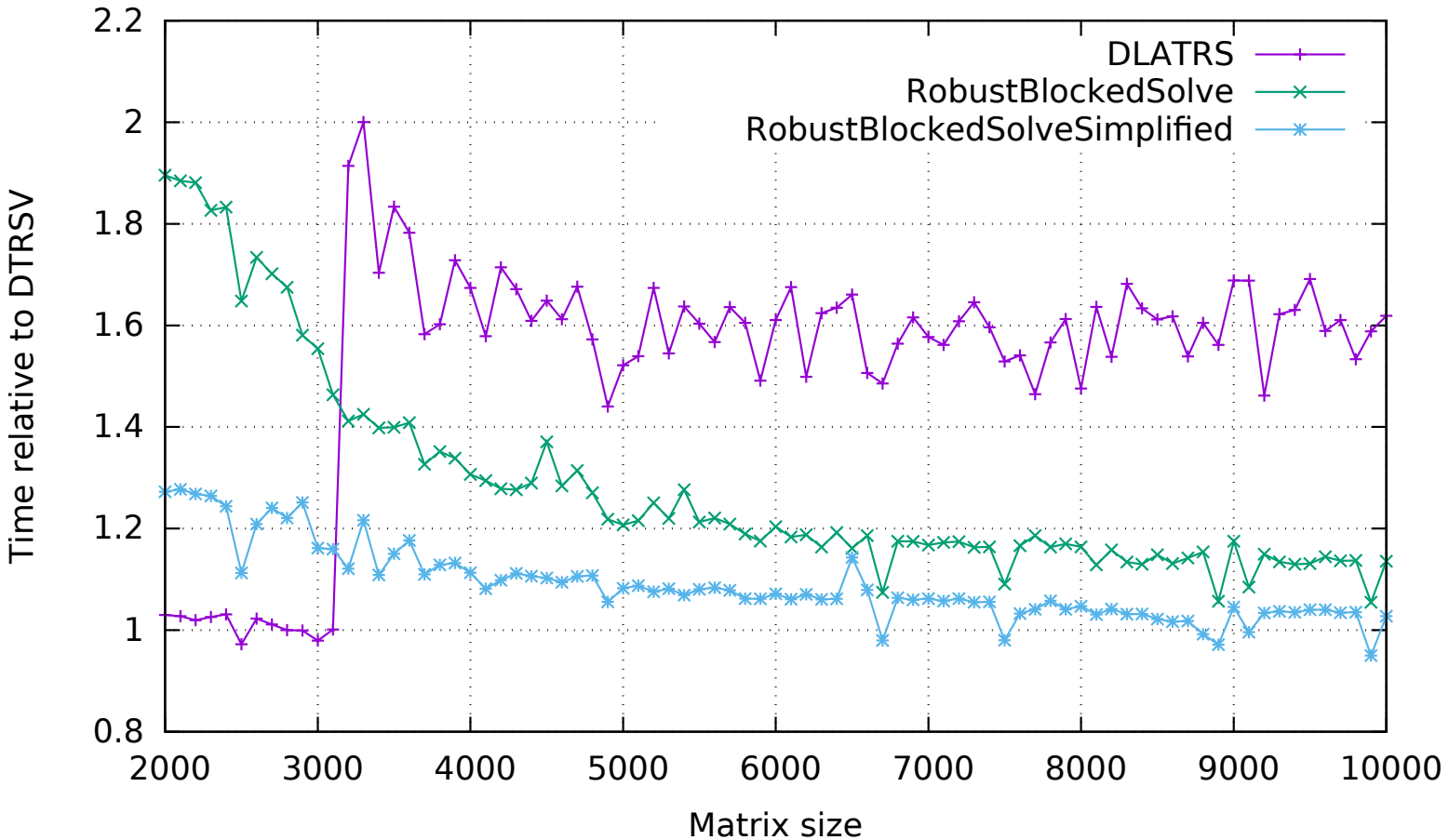
$\langle \alpha, \mathbf{x} \rangle$ contains exactly the same information as $\alpha^{-1}x$.

Robust blocked solvers

Algorithm 3: $[\alpha, \mathbf{x}] = \texttt{RobustBlockedSolve}(\mathbf{T}, \mathbf{b})$

1 **for** $i = 1, 2, \ldots, N$ **do**
2 $\quad \langle \alpha_i, \mathbf{x}_i \rangle \leftarrow \langle 1, \mathbf{b}_i \rangle$;
3 **for** $j = N, N-1, \ldots, 1$ **do**
4 $\quad \langle \alpha_j, \mathbf{x}_j \rangle \leftarrow \texttt{RobustBlockSolve}(\mathbf{T}_{jj}, \langle \alpha_j, \mathbf{x}_j \rangle)$;
5 $\quad$ **for** $i = 1, 2, \ldots, j-1$ **do**
6 $\quad\quad \langle \alpha_i, \mathbf{x}_i \rangle \leftarrow \texttt{RobustBlockUpdate}(\langle \alpha_i, \mathbf{x}_i \rangle, \mathbf{T}_{ij}, \langle \alpha_j, \mathbf{x}_j \rangle)$
7 $\alpha \leftarrow \min\{\alpha_1, \alpha_2, \ldots, \alpha_N\}$;
8 **for** $i = 1, 2, \ldots, N$ **do**
9 $\quad$ **if** $\alpha \neq \alpha_i$ **then**
10 $\quad\quad \mathbf{x}_i \leftarrow (\alpha / \alpha_i)\mathbf{x}_i$;
11 **return** $[\alpha, \mathbf{x}]$;

The price of robustness



How far can you go?

With

$$\alpha = 2^{-k}, \quad \text{where } k \text{ is a 64 bit integer,}$$

the smallest scaling factor is

$$\alpha_{\min} = 2^{-(2^{64}-1)} \approx 10^{-5.5530\times 10^{18}}.$$

In double precision

$$\gamma_{\max} = \frac{(2 - 2^{-52}) \times 2^{1023}}{2^{-1074}} \approx 2^{2098}$$

In the worst case

$$x_n = \gamma_{\max}^2 (1 + \gamma_{max})^{n-2} \approx \gamma_{\max}^n$$

overwhelms $\alpha_{\min}$ only when

$$n \approx 8.7925 \times 10^{15}, \quad 4n^2 \approx 3.0932 \times 10^{32}, \quad M_\oplus = 5.972 \times 10^{24} kg$$

Questions?

Thank you for your attention!