# Solving tridiagonal systems in parallel

Alan Edelman, Andreas Jensen, **Eka Palamadai**

# Outline

- Serial problem
  - Results
- Parallel problem
  - Results
  - Algorithm
- Future work

# Serial problem

Solve $Ax = b$, where $A$ is a symmetric diagonally-dominant tridiagonal matrix, and $b$, $x$ are column vectors.

# Code snippet to solve Ax = b

```
# a and c are diagonal and sub-diagonal of A
# d is the diagonal of U
# b is the rhs and x is the solution
function solve(a, c, d, b, x)
    # factorize A = LU and solve Lx = b
    n = size(a,1) ; d [1] = a [1] ; x [1] = b [1] ;
    for i = 2:n
        l = c [i - 1] / d [i - 1] ;
        d [i] = a [i] - c [i - 1] * l ;
        x [i] = b [i] - l * x [i - 1] ;
    end
```

# Code snippet continued…

```
# a and c are diagonal and sub-diagonal of A
# d is the diagonal of U
# b is the rhs and x is the solution
function solve(a, c, d, b, x)
    ….
    # solve Ux = x
    x [n] = x [n] / d [n]
    for i = n - 1 : -1 : 1
        x [i] = (x [i] - c [i] * x [i + 1]) / d [i]
    end
end
```

# Serial results for a matrix of size 1 billion

| | |
|---|---|
| solve | : 32 s |
| lapack | : 64 s |

Why does solve run faster than lapack?

- solve doesn't pivot off the diagonal
- solve doesn't store the sub-diagonal of L

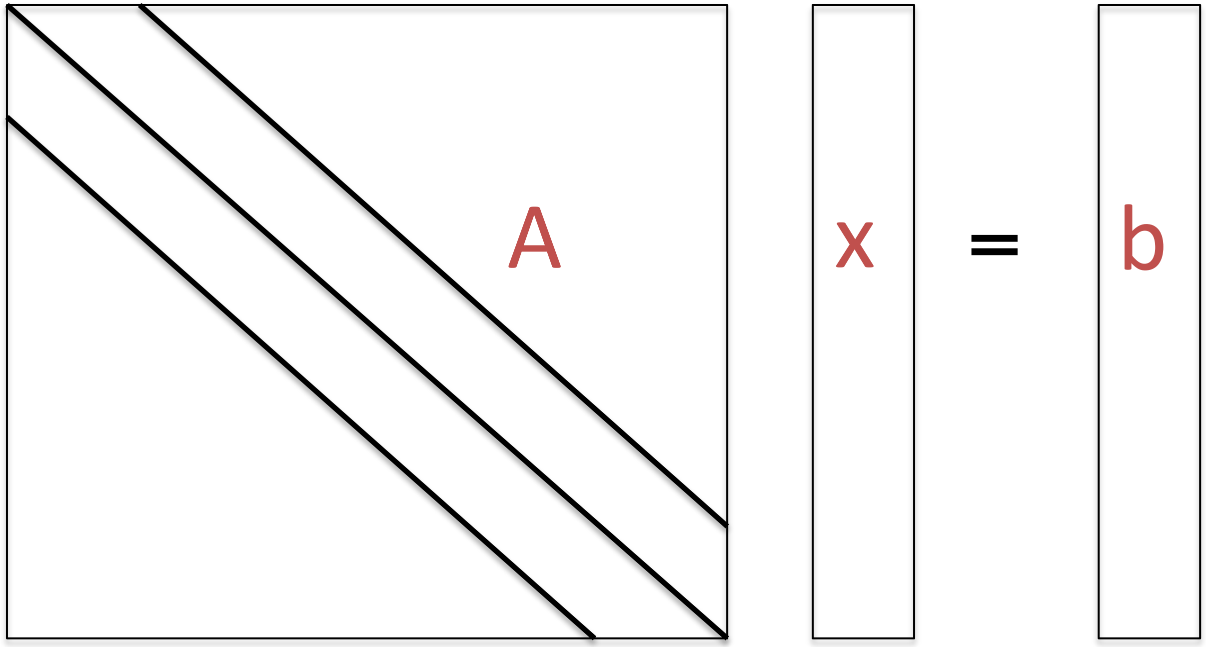
# Parallel problem

Given $p$ processors, solve $Ax = b$, where $A$ is a symmetric diagonally-dominant tridiagonal matrix, and $b$, $x$ are column vectors.

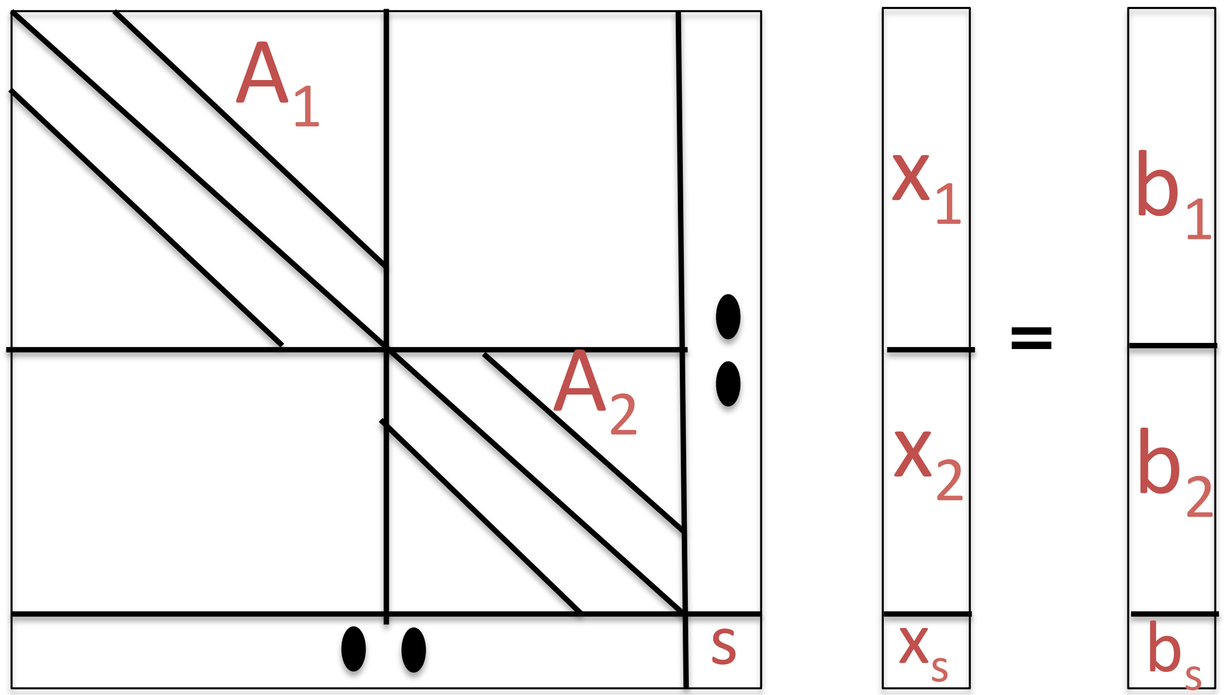
# Parallel results for a matrix of size 1 billion

| # of processors | Runtime, s |
|---:|---:|
| 1 | 46.28 |
| 2 | 27.88 |
| 4 | 13.93 |
| 8 | 7.31 |
| 16 | 4.51 |

# A parallel algorithm for 2 processors



# A parallel algorithm for 2 processors

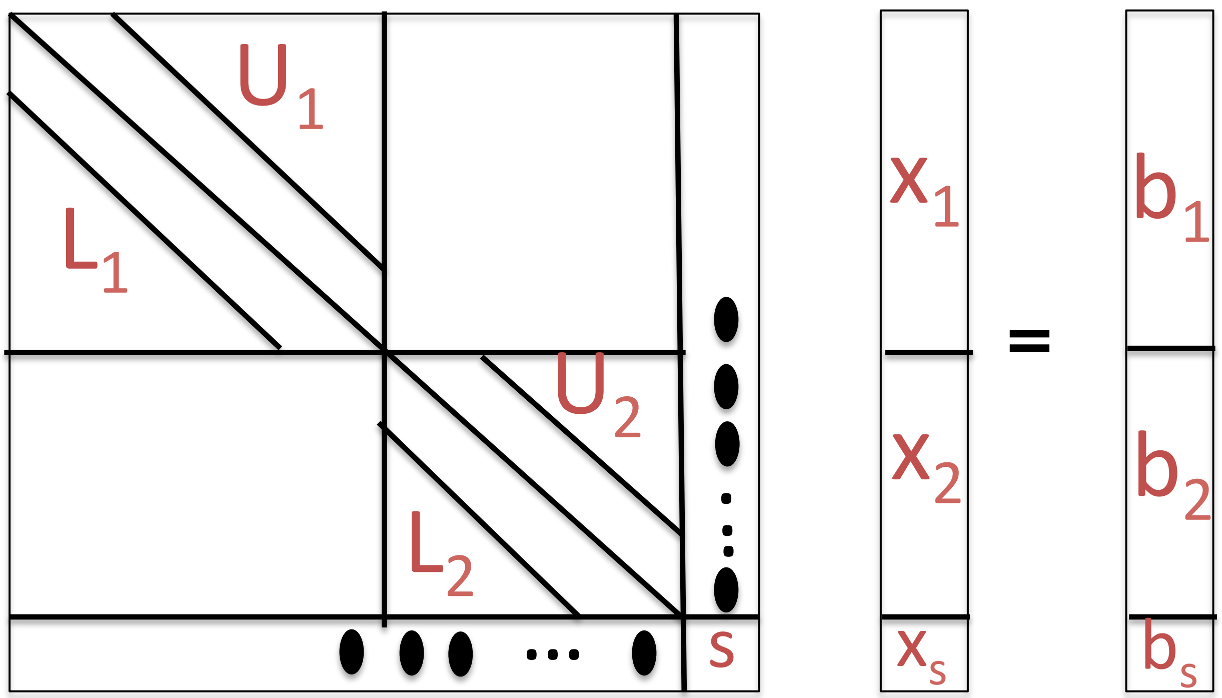



function 2-solve (A, b, x)

Trisect A into tridiagonal matrices $A_1$ , $A_2$, and a separator s

Trisect x into $x_1$, $x_2$, and $x_s$ ; Trisect b into $b_1$, $b_2$, and $b_s$

# A parallel algorithm for 2 processors

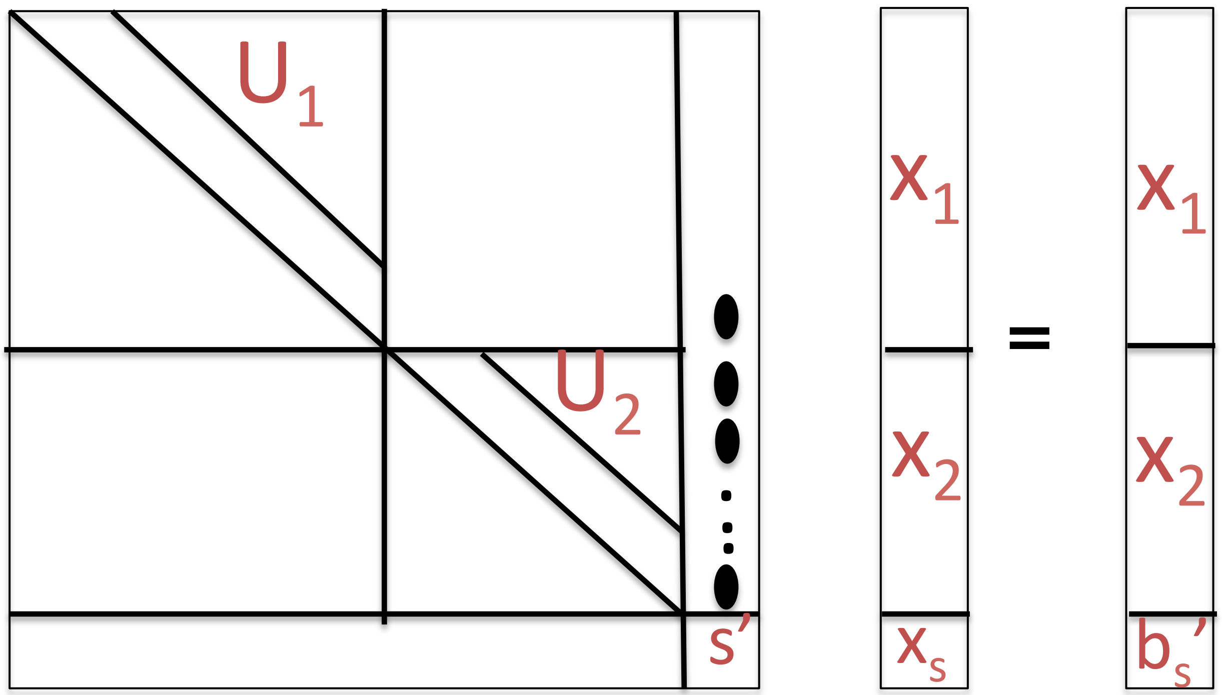



function 2-solve (A, b, x)

Parallel : { Factorize $A_1 = L_1U_1$ and forward solve $L_1x_1 = b_1$ ;

Factorize $A_2 = L_2U_2$ and forward solve $L_2x_2 = b_2$ ;}

# A parallel algorithm for 2 processors




function 2-solve (A, b, x)

Solve the reduced system $s' x_s = b_s'$

Parallel : { Back solve $U_1 x_1 = x_1$ ; Back solve $U_2 x_2 = x_2$ ;}

# A parallel algorithm for p processors

function psolve (A, b, x)

1. n = size (A)
2. Partition A into p tridiagonal matrices $A_1$ , $A_{2,}$ …, $A_p$ of size roughly (n – p + 1) / p, and p - 1 separators $s_1$, $s_2$, …, $s_{p-1}$ of size 1.
3. Partition b into 2p-1 subvectors $b_1$, $b_2$, …, $b_p$, $bs_1$, $bs_2$, …, $bs_{p-1}$
4. Partition x into 2p-1 subvectors $x_1$, $x_2$, …, $x_p$, $xs_1$, $xs_2$, …, $xs_{p-1}$
5. Parallel : {Factorize $A_i = L_iU_i$ and forward solve $L_ix_i = b_i$} for $i \in \{1, . . . , p\}$
6. Update separators $s_1, . . . , s_{p-1}$ with the schur complements from submatrices $A_1$, …, $A_p$
7. Solve for the unknowns $xs_1$, $xs_2$, …, $xs_{p-1}$ corresponding to the separators.
8. Parallel : {Back solve $U_ix_i = x_i$} for $i \in \{1,…,p\}$

# Analysis

1 processor :

Number of floating point operations : $8n$

p-processors :

Number of floating point operations ~ $17\,(n-p+1)/p + 8\,(p - 1)$

$\leq 17\,(n+1)\,/\,p + 8p$

# Future work

Higher dimensional problems

# Questions?

# Thank you