CS 240A: Solving Ax = b in parallel

- Dense A: Gaussian elimination with partial pivoting (LU)
  - Same flavor as matrix * matrix, but more complicated
- Sparse A: Gaussian elimination – Cholesky, LU, etc.
  - Graph algorithms
- Sparse A: Iterative methods – Conjugate gradient, etc.
  - Sparse matrix times dense vector
- Sparse A: Preconditioned iterative methods and multigrid
  - Mixture of lots of things

CS 240A: Solving Ax = b in parallel

- Dense A: Gaussian elimination with partial pivoting (LU)
  - Same flavor as matrix * matrix, but more complicated

- Sparse A: Gaussian elimination – Cholesky, LU, etc.
  - Graph algorithms

- Sparse A: Iterative methods – Conjugate gradient, etc.
  - Sparse matrix times dense vector

- Sparse A: Preconditioned iterative methods and multigrid
  - Mixture of lots of things

Dense Linear Algebra (Excerpts)

James Demmel

http://www.cs.berkeley.edu/~demmel/cs267_221001.ppt

Motivation

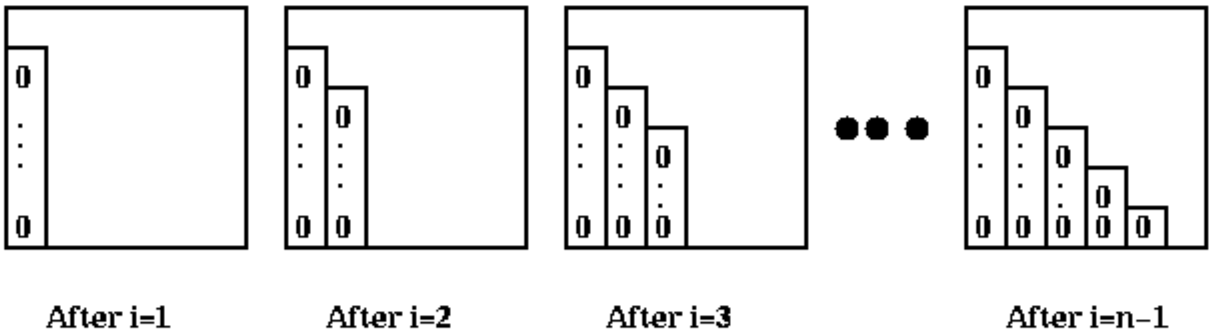
- **3 Basic Linear Algebra Problems**
  - Linear Equations: Solve Ax=b for x
  - Least Squares: Find x that minimizes $\Sigma\, r_i^2$ where r=Ax-b
  - Eigenvalues: Find λ and x where Ax = λ x
  - Lots of variations depending on structure of A (eg symmetry)
- **Why dense A, as opposed to sparse A?**
  - Aren't "most" large matrices sparse?
  - Dense algorithms easier to understand
  - Some applications yields large dense matrices
    - Ax=b: Computational Electromagnetics
    - Ax = λx: Quantum Chemistry
  - Benchmarking
    - "How fast is your computer?" =
      "How fast can you solve dense Ax=b?"
  - Large sparse matrix algorithms often yield smaller (but still large) dense problems

Review of Gaussian Elimination (GE) for solving Ax=b

- Add multiples of each row to later rows to make A upper triangular
- Solve resulting triangular system Ux = c by substitution

```
… for each column i
… zero it out below the diagonal by adding multiples of row i to later rows
for i = 1 to n-1
   … for each row j below row i
   for j = i+1 to n
       … add a multiple of row i to row j
       for k = i to n
           A(j,k) = A(j,k) - (A(j,i)/A(i,i)) * A(i,k)
```

Structure of Matrix during simple version of Gaussian Elimination



Refine GE Algorithm (1)

° **Initial Version**

```
… for each column i
… zero it out below the diagonal by adding multiples of row i to later rows
for i = 1 to n-1
    … for each row j below row i
    for j = i+1 to n
        … add a multiple of row i to row j
        for k = i to n
            A(j,k) = A(j,k) - (A(j,i)/A(i,i)) * A(i,k)
```

° **Remove computation of constant A(j,i)/A(i,i) from inner loop**

```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i)/A(i,i)
        for k = i to n
            A(j,k) = A(j,k) - m * A(i,k)
```

Refine GE Algorithm (2)

- **Last version**

```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i)/A(i,i)
        for k = i to n
            A(j,k) = A(j,k) - m * A(i,k)
```

- **Don't compute what we already know: zeros below diagonal in column i**

```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i)/A(i,i)
        for k = i+1 to n
            A(j,k) = A(j,k) - m * A(i,k)
```

Refine GE Algorithm (3)

- Last version

```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i)/A(i,i)
        for k = i+1 to n
            A(j,k) = A(j,k) - m * A(i,k)
```

- Store multipliers m below diagonal in zeroed entries for later use

```
for i = 1 to n-1
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```

Refine GE Algorithm (4)

- **Last version**

```
for i = 1 to n-1
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```

  - **Split Loop**

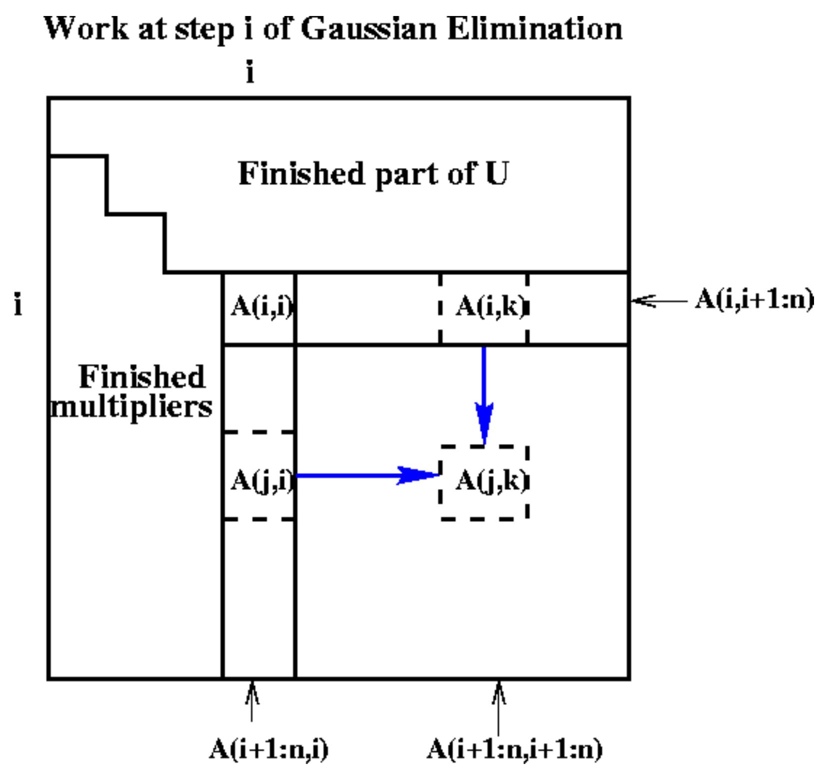
```
for i = 1 to n-1
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
    for j = i+1 to n
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```

Refine GE Algorithm (5)

- **Last version**

```
for i = 1 to n-1
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
    for j = i+1 to n
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```

- **Express using matrix operations (BLAS)**




```
for i = 1 to n-1
    A(i+1:n,i) = A(i+1:n,i) * ( 1 / A(i,i) )
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n )
          - A(i+1:n , i) * A(i , i+1:n)
```

What GE really computes

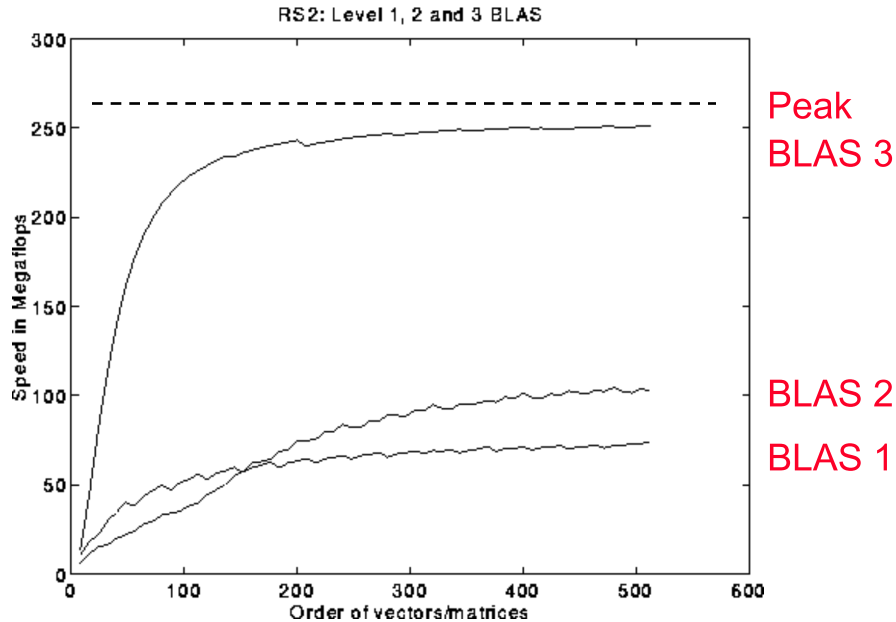
```
for i = 1 to n-1
    A(i+1:n,i) = A(i+1:n,i) / A(i,i)
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n ) - A(i+1:n , i) * A(i , i+1:n)
```

- **Call the strictly lower triangular matrix of multipliers M, and let L = I+M**
- **Call the upper triangle of the final matrix U**
- ***Lemma (LU Factorization):* If the above algorithm terminates (does not divide by zero) then A = L*U**
- **Solving A*x=b using GE**
  - Factorize A = L*U using GE (cost = 2/3 n^3 flops)
  - Solve L*y = b for y, using substitution (cost = n^2 flops)
  - Solve U*x = y for x, using substitution (cost = n^2 flops)
- **Thus A*x = (L*U)*x = L*(U*x) = L*y = b as desired**

Problems with basic GE algorithm

- **What if some A(i,i) is zero? Or very small?**
  - Result may not exist, or be "unstable", so need to **pivot**
- **Current computation all BLAS 1 or BLAS 2, but we know that BLAS 3 (matrix multiply) is fastest (earlier lectures…)**

```
for i = 1 to n-1
    A(i+1:n,i) = A(i+1:n,i) / A(i,i)          … BLAS 1 (scale a vector)
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n )   … BLAS 2 (rank-1 update)
          - A(i+1:n , i) * A(i , i+1:n)
```



Pivoting in Gaussian Elimination

- $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ fails completely, even though A is "easy"

- Illustrate problems in 3-decimal digit arithmetic:

  $A = \begin{bmatrix} 1e\text{-}4 & 1 \\ 1 & 1 \end{bmatrix}$ and $b = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, correct answer to 3 places is $x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

- Result of LU decomposition is

  $L = \begin{bmatrix} 1 & 0 \\ fl(1/1e\text{-}4) & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1e4 & 1 \end{bmatrix}$ … No roundoff error yet

  $U = \begin{bmatrix} 1e\text{-}4 & 1 \\ 0 & fl(1-1e4*1) \end{bmatrix} = \begin{bmatrix} 1e\text{-}4 & 1 \\ 0 & -1e4 \end{bmatrix}$ … Error in 4th decimal place

  Check if $A = L*U = \begin{bmatrix} 1e\text{-}4 & 1 \\ 1 & 0 \end{bmatrix}$ … (2,2) entry entirely wrong

- Algorithm "forgets" (2,2) entry, gets same L and U for all |A(2,2)|<5
  - Numerical instability
  - Computed solution x totally inaccurate
- Cure: Pivot (swap rows of A) so entries of L and U bounded

Gaussian Elimination with Partial Pivoting (GEPP)

° Partial Pivoting: swap rows so that each multiplier
$|L(i,j)| = |A(j,i)/A(i,i)| <= 1$

```
for i = 1 to n-1
    find and record k where |A(k,i)| = max{i <= j <= n} |A(j,i)|
         … i.e. largest entry in rest of column i
    if |A(k,i)| = 0
        exit with a warning that A is singular, or nearly so
    elseif  k != i
        swap rows i and k of A
    end if
    A(i+1:n,i) = A(i+1:n,i) / A(i,i)       … each quotient lies in [-1,1]
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n ) - A(i+1:n , i) * A(i , i+1:n)
```

° *Lemma*: This algorithm computes A = P*L*U, where P is a permutation matrix

° Since each entry of $|L(i,j)| <= 1$, this algorithm is considered numerically stable

° For details see LAPACK code at www.netlib.org/lapack/single/sgetf2.f

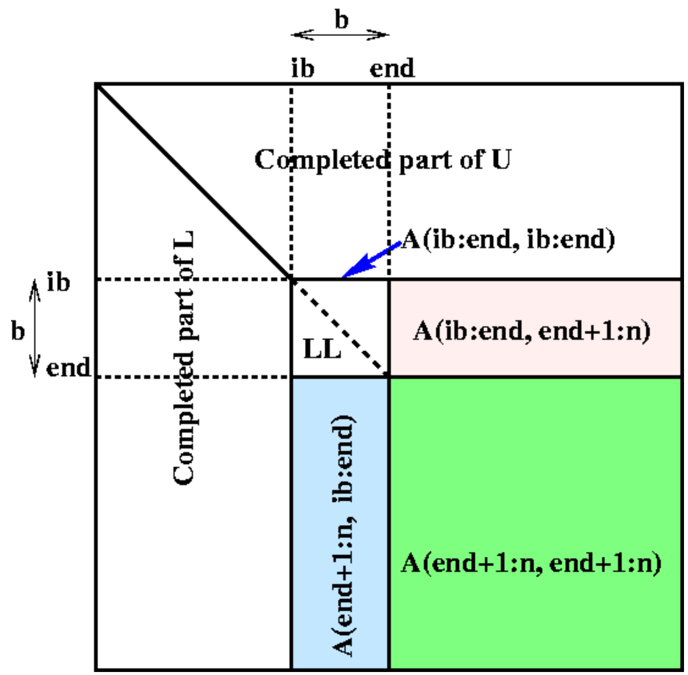
Converting BLAS2 to BLAS3 in GEPP

- **Blocking**
  - Used to optimize matrix-multiplication
  - Harder here because of data dependencies in GEPP
- **Delayed Updates**
  - Save updates to "trailing matrix" from several consecutive BLAS2 updates
  - Apply many saved updates simultaneously in one BLAS3 operation
- **Same idea works for much of dense linear algebra**
  - Open questions remain
- **Need to choose a block size b**
  - Algorithm will save and apply b updates
  - b must be small enough so that active submatrix consisting of b columns of A fits in cache
  - b must be large enough to make BLAS3 fast

Blocked GEPP (www.netlib.org/lapack/single/sgetrf.f)

for ib = 1 to n-1 step b … Process matrix b columns at a time
  end = ib + b-1 … Point to end of block of b columns
  apply BLAS2 version of GEPP to get A(ib:n , ib:end) = P' * L' * U'
  … let LL denote the strict lower triangular part of A(ib:end , ib:end) + I
  A(ib:end , end+1:n) = LL^{-1} * A(ib:end , end+1:n) … update next b rows of U
  A(end+1:n , end+1:n) = A(end+1:n , end+1:n)
    - A(end+1:n , ib:end) * A(ib:end , end+1:n)
      … apply delayed updates with single matrix-multiply
      … with inner dimension b

Gaussian Elimination using BLAS 3



Overview of LAPACK

- **Standard library for dense/banded linear algebra**
  - Linear systems: $A*x=b$
  - Least squares problems: $\min_x \| A*x-b \|_2$
  - Eigenvalue problems: $Ax = \lambda x$, $Ax = \lambda Bx$
  - Singular value decomposition (SVD): $A = U\Sigma V^T$
- **Algorithms reorganized to use BLAS3 as much as possible**
- **Basis of math libraries on many computers, Matlab 6**
- **Many algorithmic innovations remain**
  - Automatic optimization
  - Quadtree matrix data structures for locality
  - New eigenvalue algorithms

Parallelizing Gaussian Elimination

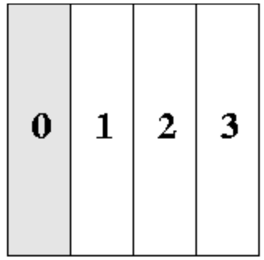
- **Recall parallelization steps from earlier lecture**
  - **Decomposition:** identify enough parallel work, but not too much
  - **Assignment:** load balance work among threads
  - **Orchestrate:** communication and synchronization
  - **Mapping:** which processors execute which threads
- **Decomposition**
  - In BLAS 2 algorithm nearly each flop in inner loop can be done in parallel, so with n^2 processors, need 3n parallel steps

```
for i = 1 to n-1
    A(i+1:n,i) = A(i+1:n,i) / A(i,i)        … BLAS 1 (scale a vector)
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n )  … BLAS 2 (rank-1 update)
            - A(i+1:n , i) * A(i , i+1:n)
```

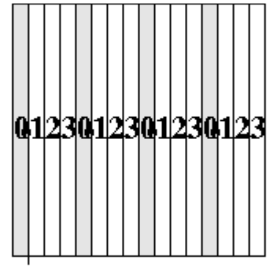
  - This is too fine-grained, prefer calls to local matmuls instead
  - Need to discuss parallel matrix multiplication
- **Assignment**
  - Which processors are responsible for which submatrices?

Different Data Layouts for Parallel GE (on 4 procs)

Bad load balance: P0 idle after first n/4 steps



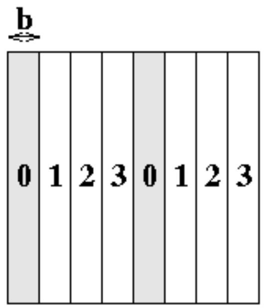
1) Column Blocked Layout



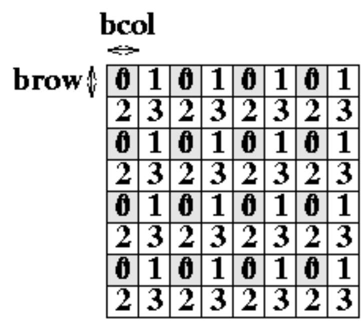
2) Column Cyclic Layout

Load balanced, but can't easily use BLAS2 or BLAS3

Can trade load balance and BLAS2/3 performance by choosing b, but factorization of block column is a bottleneck

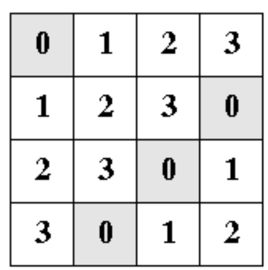


3) Column Block Cyclic Layout



4) Row and Column Block Cyclic Layout

The winner!



Complicated addressing

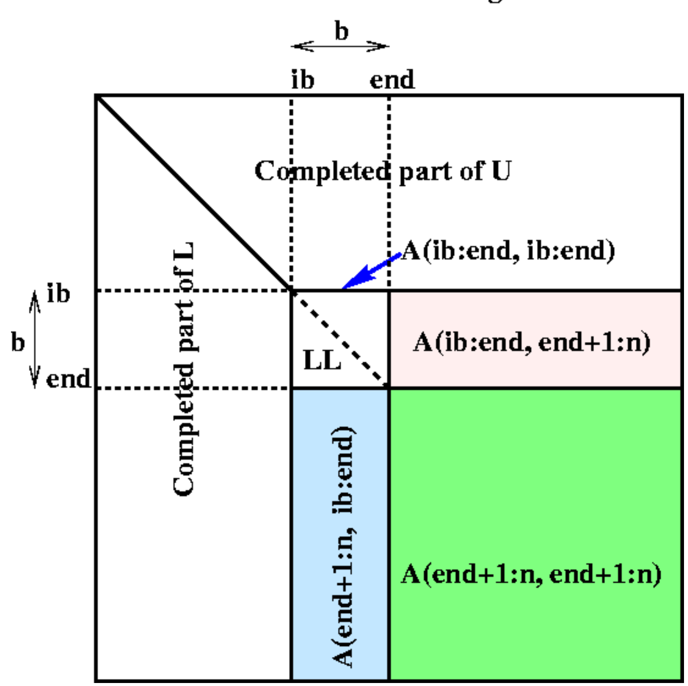
5) Block Skewed Layout

Review: BLAS 3 (Blocked) GEPP

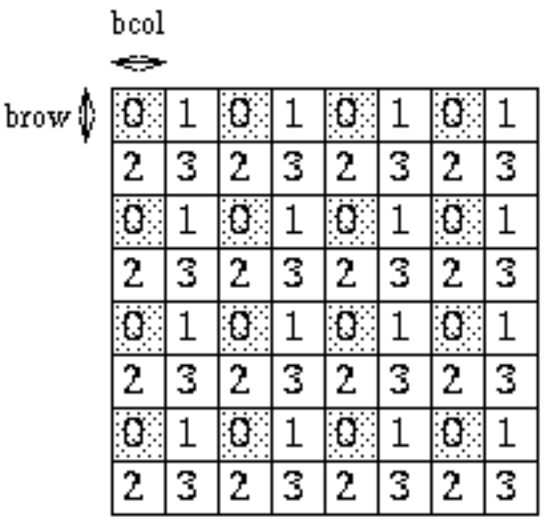
for ib = 1 to n-1 step b … Process matrix b columns at a time
    end = ib + b-1 … Point to end of block of b columns
    apply BLAS2 version of GEPP to get A(ib:n , ib:end) = P' * L' * U'
    … let LL denote the strict lower triangular part of A(ib:end , ib:end) + I

BLAS 3
    A(ib:end , end+1:n) = LL^{-1} * A(ib:end , end+1:n) … update next b rows of U
    A(end+1:n , end+1:n) = A(end+1:n , end+1:n)
        - A(end+1:n , ib:end) * A(ib:end , end+1:n)
            … apply delayed updates with single matrix-multiply
            … with inner dimension b

Gaussian Elimination using BLAS 3



Review: Row and Column Block Cyclic Layout



4) Row and Column Block Cyclic Layout

processors and matrix blocks are distributed in a 2d array

pcol-fold parallelism in any column, and calls to the BLAS2 and BLAS3 on matrices of size brow-by-bcol

serial bottleneck is eased

need not be symmetric in rows and columns

Distributed GE with a 2D Block Cyclic Layout

block size b in the algorithm and the block sizes brow and bcol in the layout satisfy b=brow=bcol.

shaded regions indicate busy processors or communication performed.

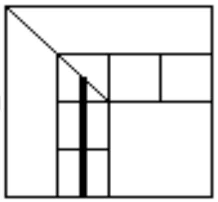
unnecessary to have a barrier between each step of the algorithm, e.g.. step 9, 10, and 11 can be pipelined

Distributed Gaussian Elimination with a 2D Block Cyclic Layout

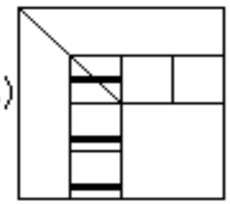
for ib = 1 to n−1 step b

end = min(ib+b−1, n)

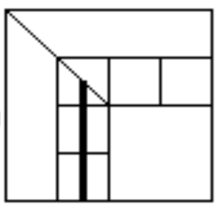
for i = ib to end

(1) 

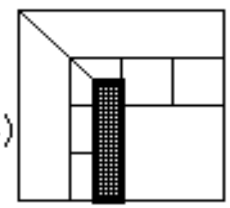
(1) find pivot row k, column broadcast

(2) 

(2) swap rows k and i in block column, broadcast row k

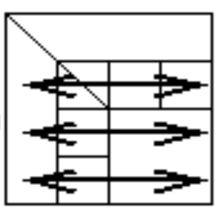
(3) 

(3) $A(i+1:n, i) = A(i+1:n, i) / A(i, i)$

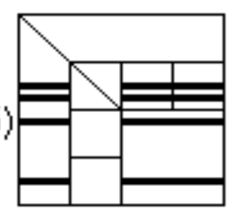
(4) 

(4) $A(i+1:n, i+1:end) -= A(i+1:n, i) * A(i, i+1:end)$

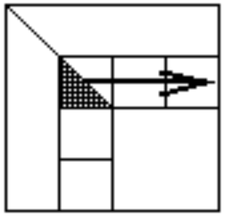
end for

(5) 

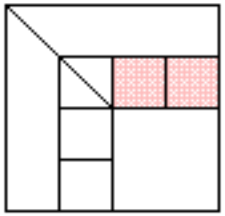
(5) broadcast all swap information right and left

(6) 

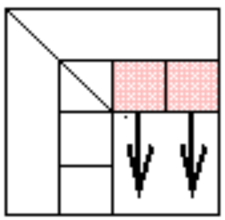
(6) apply all rows swaps to other columns

(7)


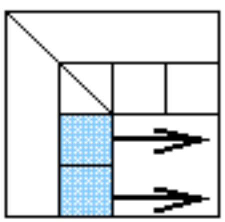
(7) Broadcast LL right

(8)


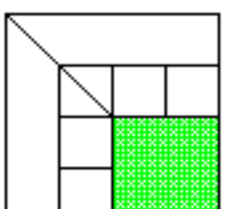
(8) A(ib:end , end+1:n) = LL \ A(ib:end , end+1:n)

(9)


(9) Broadcast A(ib:end , end+1:n) down

(10)


(10) Broadcast A(end+1:n , ib:end) right

(11)


(11) Eliminate A(end+1:n , end+1:n)

Matrix multiply of
green = green - blue * pink

c

ScaLAPACK SOFTWARE HIERARCHY