# Parallel Numerical Algorithms

# Need for standardization

- With the advent of parallel (high performance) computers came the disillusion of bad performance
- The peak rates advertised with the introduction of new machines were mostly not attainable for real life applications
- A need arose to standardize primitives of computations
- This effort also was based on already developed numerical software libraries: LINPACK, EISPACK, FISHPACK, Harwell

# Basic Linear Algebra Subroutines (BLAS)

Three levels

- BLAS 1: vector/vector operations

| | | |
|---|---|---|
| SAXPY | $y \leftarrow y + \alpha . x$ | $x, y =$ vector, $\alpha =$ scalar |
| DOTPR | $\alpha \leftarrow (x, y)$ | |
| SUM | $y \leftarrow y + x$ | |

- BLAS 2: matrix/vector operations

$$y \leftarrow By + \alpha Ax$$
$$y \leftarrow A^T x$$
$$(\alpha = \text{scalar},\ A = \text{matrix},\ x = \text{vector})$$

- BLAS 3: matrix/matrix operations

$$C \leftarrow \beta . B + \alpha . A . B$$
$$C \leftarrow C + A . B.$$

# Input/Output Data Reuse

BLAS 1 Example: Dotproduct ( x, y )

- Input Size: $2n$
- Operation Count: $2n-1$
- Output Size: $1$
- ➔ 1 operation per input element and 2n per output element

BLAS 2 Example: $y = Ax$

- Input Size: $n^2+n$
- Operation Count: $2n^2-n$
- Output Size: $n$
- ➔ 2 operations per input element and 2n per output element

BLAS 3 Example: $C=A.B$

- Input Size: $2n^2$
- Operation Count: $2n^3-n^2$
- Output Size: $n^2$
- ➔ n operations per input element and 2n per output element

# More data reuse leads to

- Better Cache/Register Utilization
- Less Communication Overhead
- More effective input, output, or intermediate data decomposition

# Example Dotproduct (BLAS 1)

```
DO I = 1, N
      C = C + A(I) * B(I)
ENDDO
```

Straightforward parallel execution on P processors:

```
DOALL II = 1,N, N/P
      DO I = II, II+N/P – 1
            C(II) = C(II) + A(I) * B(I)
      ENDDO
      C = C + C(II)
ENDDOALL
```

However, communication costs are involved!!!!!!!

# Why Fortran ????

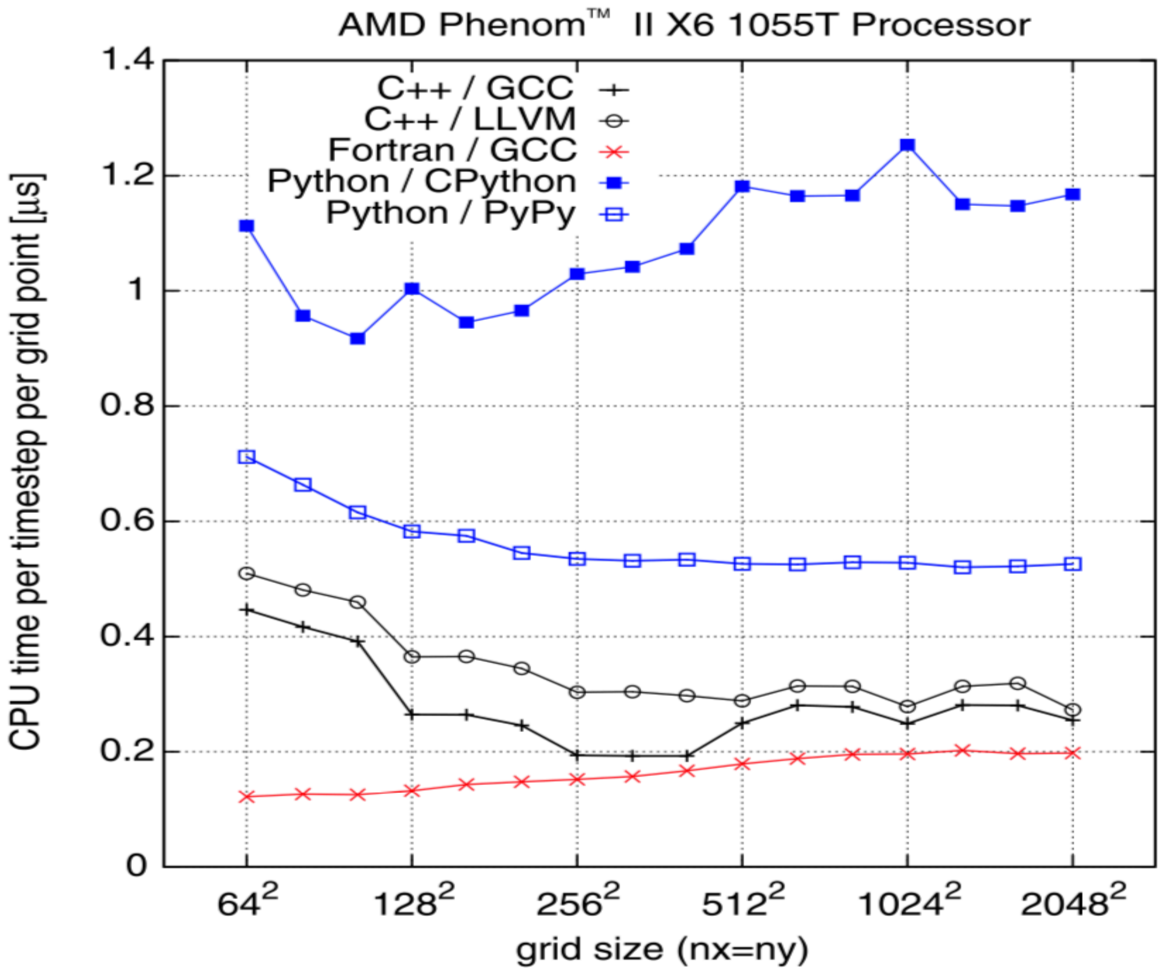



Fig. 4. Same as Fig. 3 for an AMD Phenom™ II 800 MHz processor. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-140379.)

Fortran is still outperforming most other languages in Numerical PDE computations

```
DOALL II = 1, N, N/P        # N/P is the stride, so II = 1, 1+N/P, 1+2*N/P, …
    RECEIVE (A(II:II+N/P-1), B(II:II+N/P-1))
    DO I = II, II+N/P – 1
        C(II) = C(II) + A(I) * B(I)
    ENDDO
    C = C + C(II)     ←synchronization, i.e. SEND C(J) TO MASTER PROCESS
ENDDOALL
```

So, on a total of 2N-1 computations: 2N continuous data transmissions and P separate communications are needed. With $t_s+mt_w$ ($t_s$ startup time, $t_w$ per word transmission time) communication costs for m words, this gives:

$$P.(t_s+(2N/P)t_w)+P.(t_s+t_w) =$$

$$(P+P).t_s+(2N+P)t_w = 2Pt_s + (2N+P)t_w$$

communication costs, which is significant! For instance if $t_s$ is comparable to the cost of a computational step, then the communication overhead is greater than the computational costs (2P+1).

➔ BLAS 1 routines were mainly used for VECTOR computing (pipelining)

vadd, vdotpr, vmultadd, etc.

# Example MatVec (BLAS 2)

```
DO I = 1, N
    DO J = 1, N
        C(I) = C(I) + A(I,J) * B(J)
    ENDDO
ENDDO
```

Parallel execution on P processors:

```
DO I = 1, N
    DOALL JJ = 1, N, N/P
        DO J = JJ, JJ+N/P – 1
            C(JJ) = C(JJ) + A(I,J) * B(J)
        ENDDO
        C(I) = C(I) + C(JJ)
    ENDDOALL
ENDDO
```

This is essentially is a repetition of BLAS 1 (dotproduct) operations!!!!! NOTHING GAINED. HOWEVER…

MatVec can also be computed as:

```
DO J =1, N
    DOALL II = 1, N, N/P
        DO I= II, II+N/P-1
            C(I) = C(I)+A(I,J)*B(J)
        ENDDO
    ENDDOALL
ENDDO
```

In this computation the basic (inner) loop does not execute a dotproduct, but a BLAS 1 SAXPY operation: y = y + a.x

More importantly, the vector C(II:II+N/P-1) can be stored in registers in each processor, and reused N times

Also the fan-in computations for each C(I) are not needed anymore!! So only initial distribution costs are paid for. So, overhead is reduced to

$$Pt_s+(2N)t_w$$

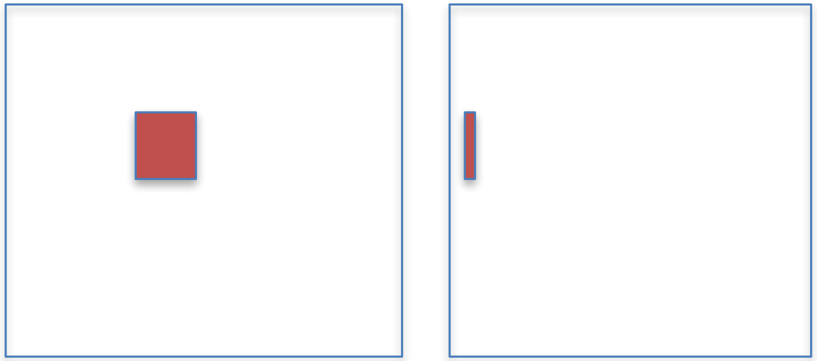
# Example MatMat (BLAS 3)

```
DO I = 1, N
    DO J = 1, N
        DO K = 1, N
            C(I,K) = C(I,K) + A(I,J) * B(J,K)
        ENDO
    ENDDO
ENDDO
```

Then because of the multi dimensionality we have different ways of executing this loop in parallel.

# Middle product form (K-loop outer loop):

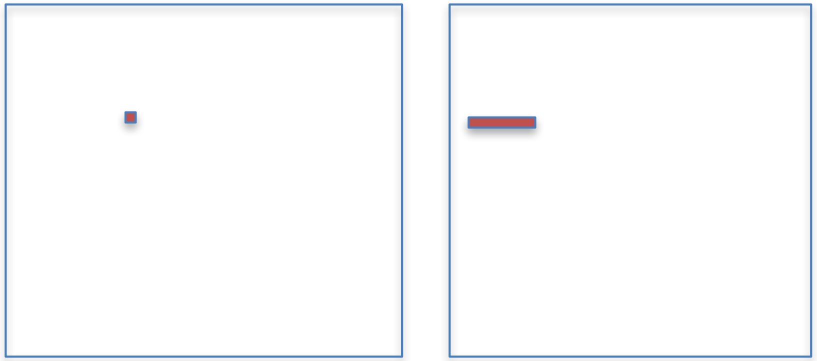
```
DO K = 1, N
    DOALL II = 1,N, N/√P
        DOALL JJ = 1,N, N/√P
            DO I = II, II+N/√P-1
                DO J = JJ, JJ+N/√P-1
                    C(I,K) = C(I,K) + A(I,J) * B(J,K)
                ENDO
            ENDDO
        ENDDOALL
    ENDOALL
ENDDO
```



In this implementation the inner loop is a BLAS 2 MatVec routine.

# Inner product form (I-loop outer loop):

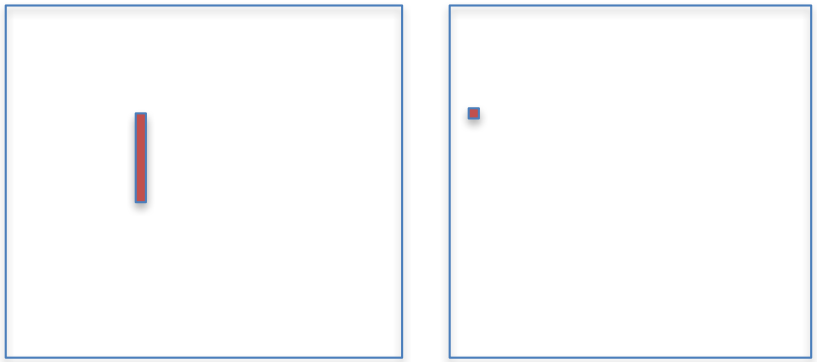
```
DO I = 1, N
    DO J = 1, N
        DOALL KK = 1, N, N/P
            DO K = KK, KK+N/P-1
                C(I,K) = C(I,K) + A(I,J) * B(J,K)
            ENDO
        ENDDOALL
    ENDDO
ENDDO
```



➔ In this implementation the inner loop is a BLAS 1 SAXPY routine.

# Outer product form (J-loop outer loop):

```
DO J = 1, N
    DO K = 1, N
        DOALL II = 1, N, N/P
            DO I = II, II+N/P-1
                C(I,K) = C(I,K) + A(I,J) * B(J,K)
            ENDO
        ENDDOALL
    ENDDO
ENDDO
```



# Another look at MatMat

The original loop can be written as follows:

```
DO II = 1, N, M1
    DO JJ = 1 ,N, M2
        DO KK = 1, N, M3
            DO I = II, II + M1 - 1
                DO J = JJ, JJ + M2 - 1
                    DO K = KK, KK + M3 - 1
                        C(I,K) = C(I,K) + A(I,J) * B(J,K)
                    ENDO
                ENDDO
            ENDDO
        ENDDO
    ENDDO
ENDDO
```

- ➔ Any of these loops can be executed in parallel!!
- ➔ These loops can be permuted in any order as long as II becomes before I, etc.
- ➔ So many different implementations possible
- ➔ M1, M2, and M3 can be used to control the degree of parallelism but also the size of cache usage.

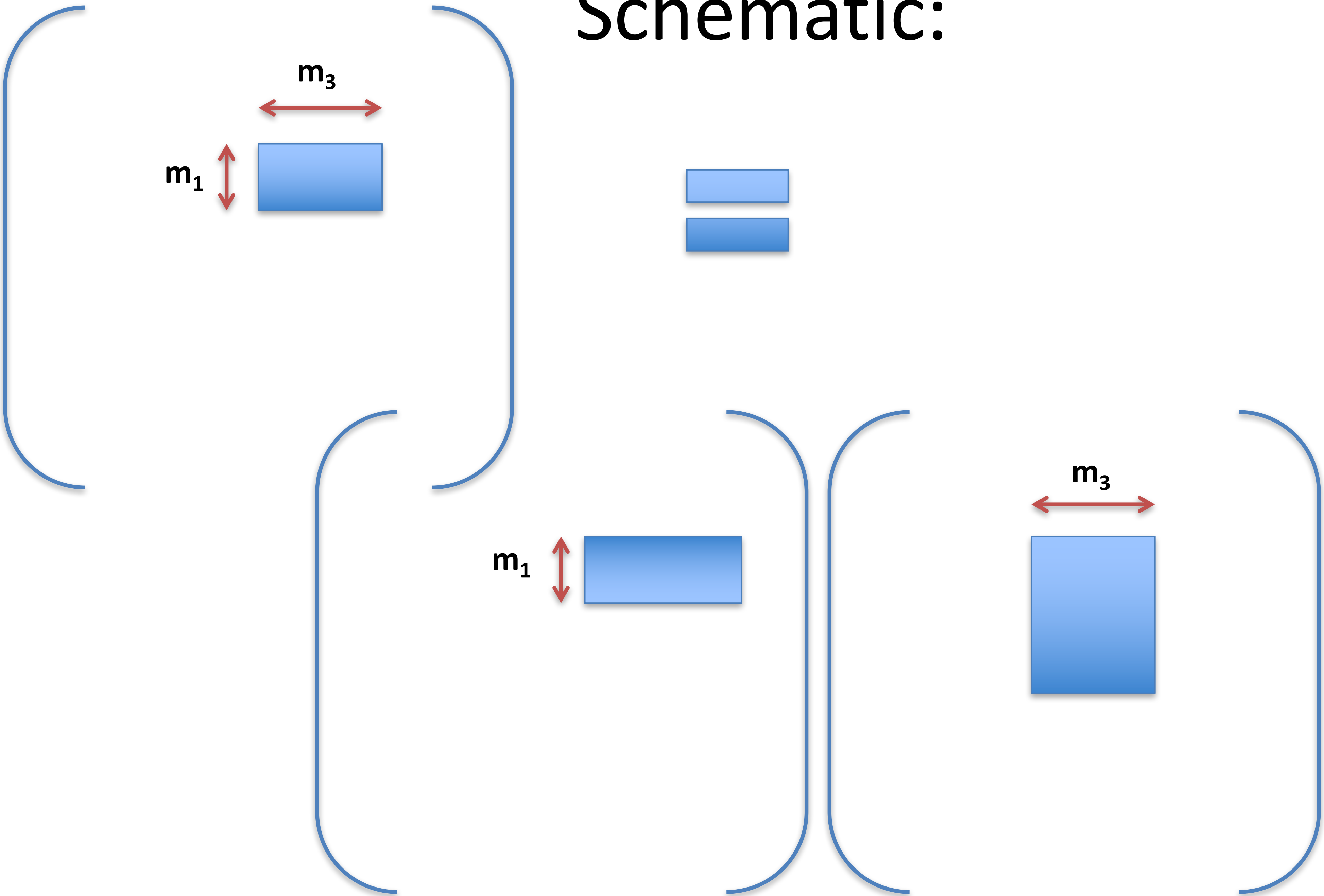
In fact

```
DO I = II, II + M1 - 1
    DO J = JJ, JJ + M2 - 1
        DO K = KK, KK + M3 - 1
            C(I,K) = C(I,K) + A(I,J) * B(J,K)
        ENDO
    ENDDO
ENDDO
```

Corresponds to a sub matrix multiply of size M1xM2 times M2xM3

By choosing M1, M2 and M3 carefully, this triple nested loop can each time run out of cache

# Schematic:



# Embeddings of BLAS routines

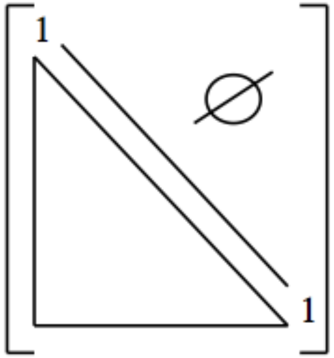
Many scientific computations involve the solution of a system of linear equations

$$
\begin{array}{lllllll}
a_{0,0}x_0 & + & a_{0,1}x_1 & + & \cdots & + & a_{0,n-1}x_{n-1} & = & b_0, \\
a_{1,0}x_0 & + & a_{1,1}x_1 & + & \cdots & + & a_{1,n-1}x_{n-1} & = & b_1, \\
\vdots & & \vdots & & & & \vdots & & \vdots \\
a_{n-1,0}x_0 & + & a_{n-1,1}x_1 & + & \cdots & + & a_{n-1,n-1}x_{n-1} & = & b_{n-1}.
\end{array}
$$

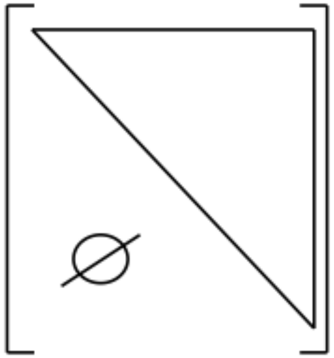
This is written as A$x$ = b where A is an $n$ x $n$ matrix with A[$i$, $j$] = $a_{ij}$, b is an $n$ x $1$ vector [ $b_0$, $b_1$, ... , $b_n$ ]$^{T}$, and $x$ is the solution.

# LU Factorization

Find

L = 


and

U = 


Such that A = L.U

Then solving Ax = b corresponds to solving

$$L\,(U\,x) = b$$

This can be done in 2 steps, triangular solves:

$$L\,c = b \text{ (forward substitution)}$$

$$U\,x = c \text{ (backward substitution)}$$

# Backward substitution U x = y

$$
\begin{array}{ccccccccc}
x_0 & + & u_{0,1}x_1 & + & u_{0,2}x_2 & + & \cdots & + & u_{0,n-1}x_{n-1} & = & y_0, \\
 & & x_1 & + & u_{1,2}x_2 & + & \cdots & + & u_{1,n-1}x_{n-1} & = & y_1, \\
 & & & & & & & & \vdots & & \vdots \\
 & & & & & & & & x_{n-1} & = & y_{n-1}.
\end{array}
$$

The factors L and U can be obtained through Gaussian Elimination

$$\begin{cases} 2x_1 & + & 3x_2 & + & x_3 & = 1 \\ x_1 & + & x_2 & + & 3x_3 & = 2 \\ 3x_1 & + & 2x_2 & + & x_3 & = 3 \end{cases}$$

$$A = \begin{pmatrix} 2 & 3 & 1 \\ 1 & 1 & 3 \\ 3 & 2 & 1 \end{pmatrix}, B = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

```
DO I = 1, N
    PIVOT = A(I, I)
    DO J = I+1, N
        MULT = A(J, I)/PIVOT
        A(J, I) = MULT
        DO K = I+1, N
            A(J, K) = A(J, K) - MULT * A(I, K)
        ENDDO
    ENDDO
ENDDO
```

This yields:

$$\tilde{A} = \begin{pmatrix} 2 & 3 & 1 \\ \frac{1}{2} & -\frac{1}{2} & 2\frac{1}{2} \\ 1\frac{1}{2} & 5 & -13 \end{pmatrix}. \text{ So, } L = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ 1\frac{1}{2} & 5 & 1 \end{bmatrix} \text{ and } U = \begin{pmatrix} 2 & 3 & 1 \\ 0 & -\frac{1}{2} & 2\frac{1}{2} \\ 0 & 0 & -13 \end{pmatrix}.$$

After L and U are computed the system is solved by:

forward substitution:

```
DO I = 1, N
   C(I) = B(I)
   DO J = 1, I-1
      C(I) = C(I) - A(I, J) * C(J)
   ENDDO
ENDDO
```

back substitution:

```
DO I = N, 1
   X(I) = C(I)
   DO J = I+1, N
      X(I) = X(I) - A(I, J) * X(J)
   ENDDO
   X(I) = X(I)/A(I, I)
ENDDO
```

# Block LU decomposition

Write A as follows

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} I & 0 \\ L_{21} & I \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} \\ 0 & B \end{pmatrix}$$

To be stored as:

$$\begin{bmatrix} A_{11}^{-1} & A_{12} \\ L_{21} & B \end{bmatrix}$$

So

$$A = \begin{pmatrix} A_{11} & A_{12} \\ L_{21} A_{11} & L_{21} A_{12} + B \end{pmatrix}$$

Let $k$ be the dimension of $A_{11}$ and $N$-$k$ the dimension of $A_{22}$
Then the algorithm becomes:

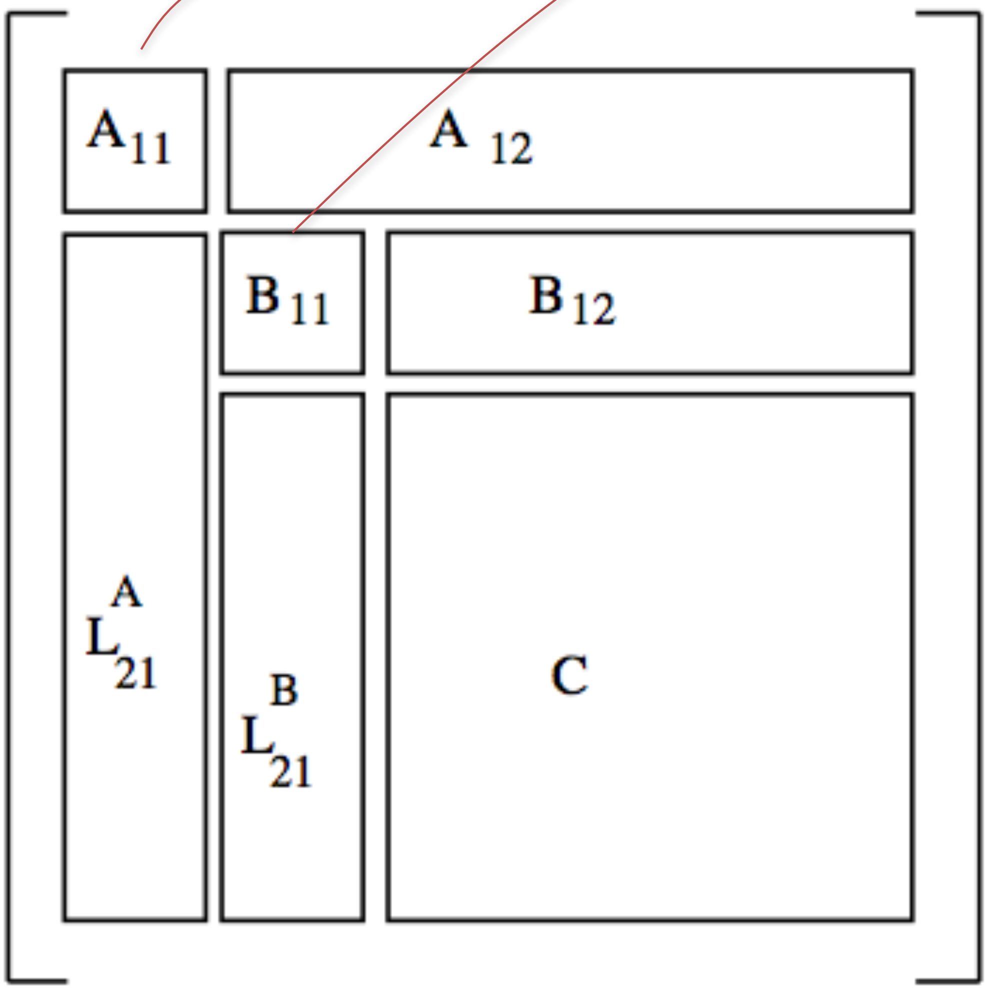
$$\left[\begin{array}{l} A_{11} \leftarrow A_{11}^{-1} \\ A_{21} \leftarrow L_{21} = A_{21} A_{11} \\ A_{22} \leftarrow B = A_{22} - A_{21} A_{12} \end{array}\right.$$

$(A_{21}A_{11})A_{11}^{-1}=A_{21}$

And proceed recursively on $B$ $(A_{22})$

# In a picture

In fact $A_{11}^{-1}$,$B_{11}^{-1}$, etc are stored in these positions




Note that the
I diagonal blocks
do not need to
be kept.

As a results

➔ This algorithm only has only to compute the inverse of $A_{11}$, otherwise **only matrix multiplies** are performed

The only complication is that back substitution is a bit more tedious.

# Backward Substitution

$$\begin{bmatrix} U_1 & \tilde{U}_1 & & \\ & U_2 & \tilde{U}_2 & \\ & & U_3 & \tilde{U}_3 \\ \emptyset & & & U_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix}$$

1. Solve $U_4 x_4 = c_4$
2. $c_3 = c_3 - \tilde{U}_3 \cdot x_4$
3. Solve $U_3 x_3 = c_3$
4. $c_2 = c_2 - \tilde{U}_2 \cdot \begin{bmatrix} x_3 \\ x_4 \end{bmatrix}$
5. Solve $U_2 x_2 = c_2$
6. $c_1 = c_1 - \tilde{U}_1 \cdot \begin{bmatrix} x_2 \\ x_3 \\ x_4 \end{bmatrix}$
7. Solve $U_1 x_1 = c_1$

**Note that $U_4 x_4 = c_4$ can be solved directly by $x_4 = A_{44}^{-1} c_4$ etc**

# Forward Substitution

$$\left[\begin{array}{cccc} I & & & \\ L_2 & I & & \emptyset \\ \multicolumn{2}{c}{L_3} & I & \\ \multicolumn{3}{c}{L_4} & I \end{array}\right] \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

1. $c_1 = b_1$
2. $c_2 = b_2 - L_2 \cdot c_1$
3. $c_3 = b_3 - L_3 \cdot \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$
4. $c_4 = b_4 - L_4 \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}$