

---

# CS 267 Applications of Parallel Processors

## Lecture 10: Large Dense Linear Systems - Distributed Implementations

**2/21/97**

**Horst D. Simon**

**<http://www.cs.berkeley.edu/cs267>**

## Review - Lecture 9

---

- **computational electromagnetics and linear systems**
- **rewritten Gaussian elimination as vector and matrix-vector operation (level 2 BLAS)**
- **discussed the efficiency of level 3 BLAS in terms of reducing number of memory accesses**

# Outline - Lecture 10

---

- **Layout of matrices on distributed memory machines**
- **Distributed Gaussian elimination**
- **Speeding up with advanced algorithms**
- **LINPACK and LAPACK**
- **LINPACK benchmark**
- **Tflops result**

# Review of Gaussian Elimination

---

Now we use Matlab (data parallel) notation to express the algorithm even more compactly:

```
for i = 1 to n-1
```

```
    A(i+1:n, i) = A(i+1:n, i) / A(i, i)
```

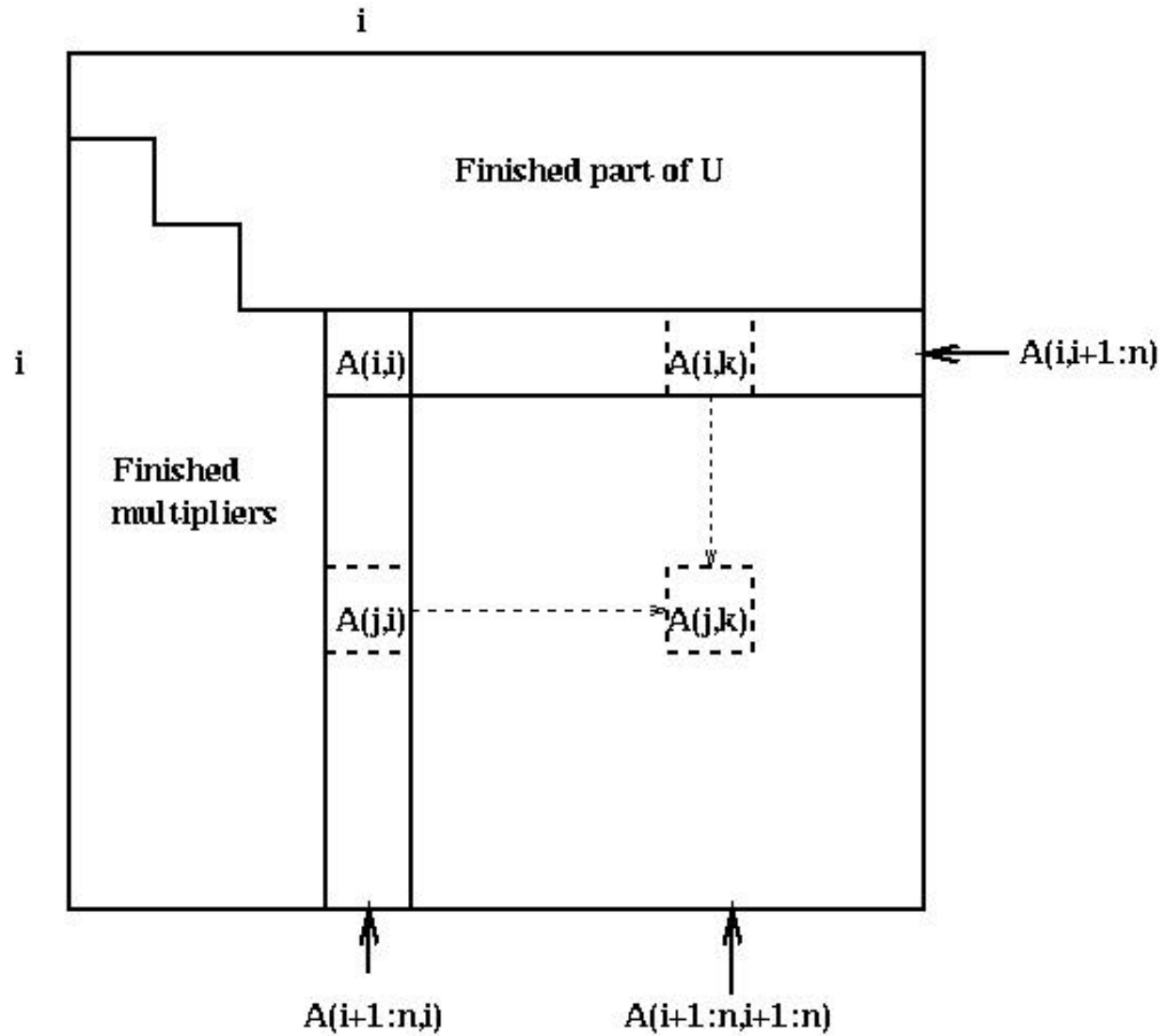
```
    A(i+1:n, i+1:n) = A(i+1:n, i+1:n) -  
                      A(i+1:n, i)*A(i, i+1:n)
```

The inner loop consists of one vector operation, and one matrix-vector operation.

**Note that the loop looks elegant, but no longer intuitive.**

# Review of Gaussian Elimination (cont.)

Work at step 1 of Gaussian Elimination



# Partial Pivoting

---

Reordering the rows of  $A$  so that  $A(i,i)$  is large at each step of the algorithm.

At step  $i$  of the algorithm, row  $i$  is swapped with row  $k > i$  if  $|A(k,i)|$  is the largest entry among  $|A(i:n,i)|$ .

```
for i = 1 to n-1
  find and record k where
     $|A(k,i)| = \max_{i \leq j \leq n} |A(j,i)|$ 
  if  $|A(k,i)| = 0$ , exit with a warning
    that  $A$  is singular, or nearly so
  if  $i \neq k$ , swap rows  $i$  and  $k$  of  $A$ 
   $A(i+1:n, i) = A(i+1:n, i) / A(i,i)$ 
  ... each quotient lies in  $[-1,1]$ 
   $A(i+1:n, i+1:n) = A(i+1:n, i+1:n) -$ 
     $A(i+1:n, i) * A(i, i+1:n)$ 
```

## How to Use Level 3 BLAS ?

---

The current algorithm only uses level 1 and level 2 BLAS.

Want to use level 3 BLAS because of higher performance.

The standard technique is called *blocking* or *delayed updating*.

We want to save up a sequence of level 2 operations and do them all at once.

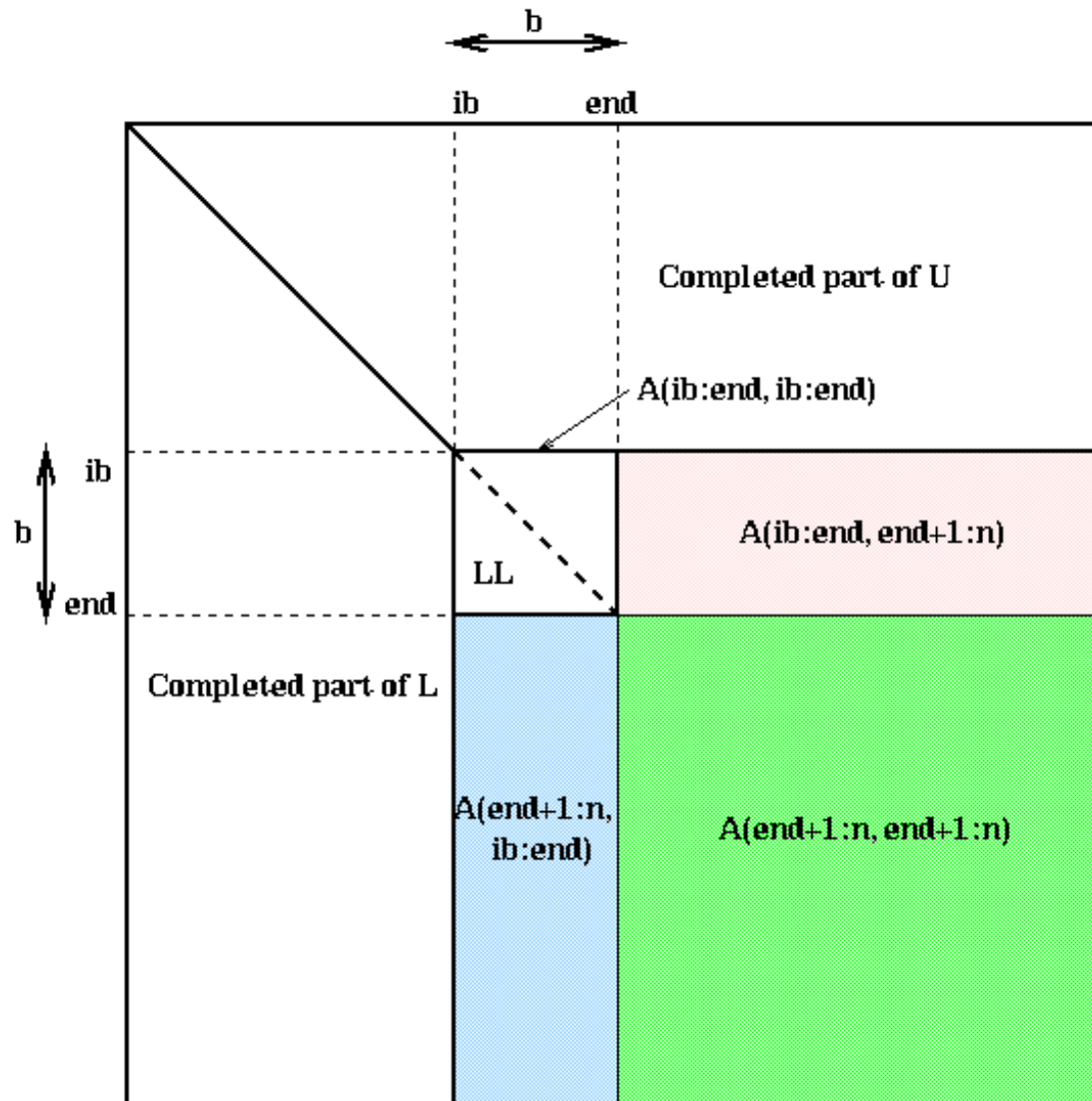
# How to Use Level 3 BLAS in LU Decomposition

---

- process the matrix in blocks of  $b$  columns at a time
- $b$  is called the block size.
- do a complete LU decomposition just of the  $b$  columns in the current block, essentially using the above BLAS2 code.
- then update the remainder of the matrix doing  $b$  rank-one updates all at once, which turns out to be a single matrix-matrix multiplication of size  $b$

# Block GE with Level 3 BLAS

Current Eliminating BLAS3



# Block GE with Level 3 BLAS

---

Gaussian elimination with Partial Pivoting,  
BLAS3 implementation

```
... process matrix b columns at a time
for ib = 1 to n-1 step b
  ... point to end of block of b columns
  end = min(ib+b-1,n)

  ... LU factorize A(ib:n,ib:end) with BLAS2
  for i = ib to end
    find and record k where
       $|A(k,i)| = \max_{\{i \leq j \leq n\}} |A(j,i)|$ 
    if  $|A(k,i)|=0$ , exit with a warning
      that A is singular, or nearly so
    if  $i \neq k$ , swap rows i and k of A
     $A(i+1:n, i) = A(i+1:n, i) / A(i,i)$ 
    ... only update columns i+1 to end
     $A(i+1:n, i+1:end) = A(i+1:n, i+1:end)$ 
       $- A(i+1:n, i)*A(i, i+1:end)$ 
  endfor
```

## Block GE with Level 3 BLAS (cont.)

---

```
... Let LL be the b-by-b lower triangular
... matrix whose subdiagonal entries are
... stored in A(ib:end,ib:end), and with
... 1s on the diagonal. Do delayed update
... of A(ib:end, end+1:n) by solving
... n-end triangular systems
... (A(ib:end, end+1:n) is pink below)
A(ib:end, end+1:n) =
    LL \ A(ib:end, end+1:n)

... do delayed update of rest of matrix
... using matrix-matrix multiplication
... (A(end+1:n, end+1:n) is green below)
... (A(end+1:n, ib:end) is blue below)
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)

endfor
```

## Block GE with Level 3 BLAS (cont.)

---

- LU factorization of  $A(ib:n,ib:end)$  uses the same algorithm as before (level 2 BLAS)
- Solving a system of  $n$ -end equations with triangular coefficient matrix  $LL$  is a single call to a BLAS3 subroutine (`strsm`) designed for that purpose.
- No work or data motion is required to refer to  $LL$ ; done with a pointer.
- When  $n \gg b$ , almost all the work is done in the final line, which multiplies an  $(n-end)$ -by- $b$  matrix times a  $b$ -by- $(n-end)$  matrix in a single BLAS3 call (to `sgemm`).

## How to select $b$ ?

---

**$b$  will be chosen in a machine dependent way to maximize performance. A good value of  $b$  will have the following properties:**

- $b$  is small enough so that the  $b$  columns currently being LU-factorized fit in the fast memory (cache, say) of the machine.**
- $b$  is large enough to make matrix-matrix multiplication fast.**

# LINPACK - LAPACK - ScaLAPACK

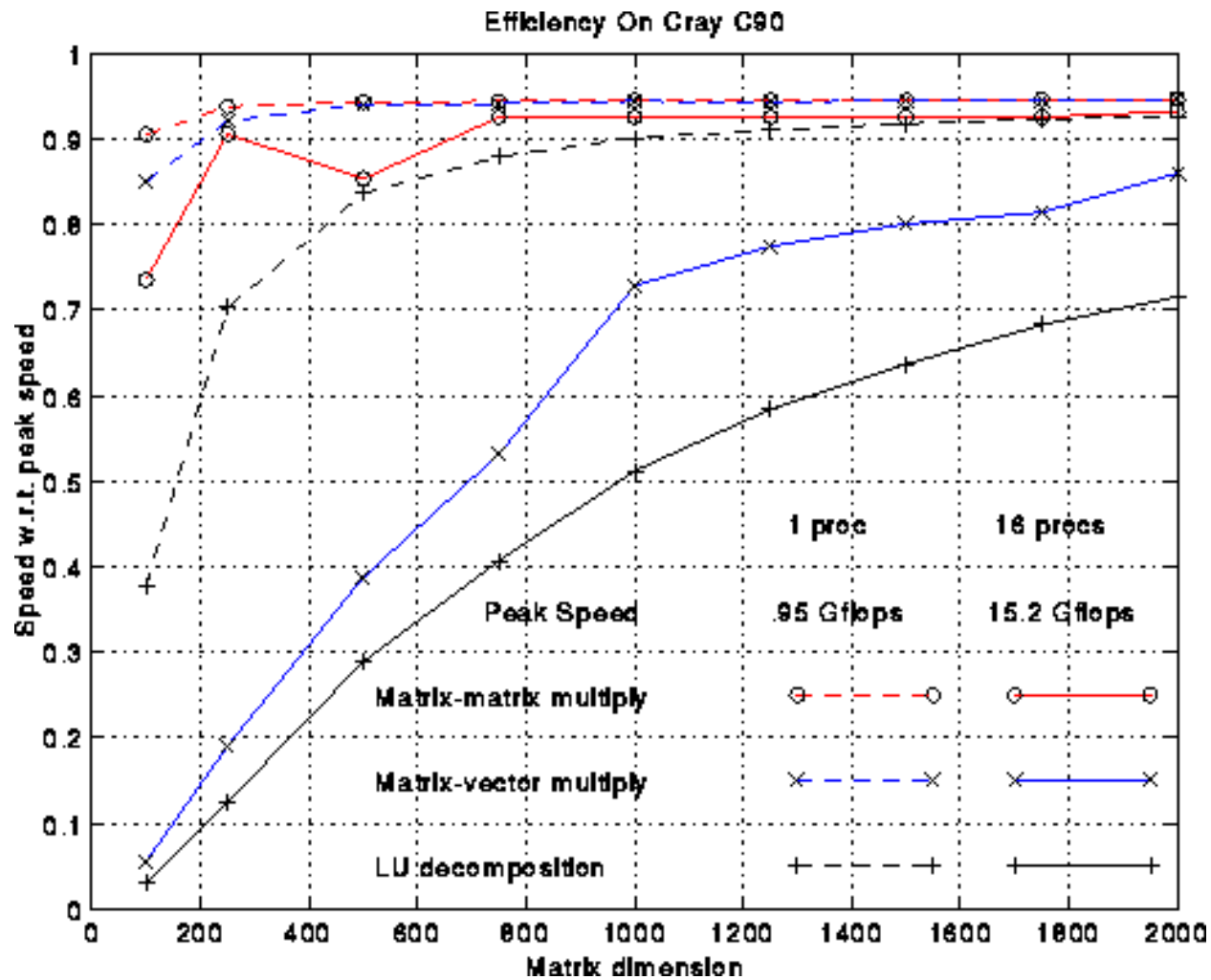
---

**LINPACK** - linear systems, least squares problems  
level 1 BLAS - late 70s

**LAPACK** - redesigned LINPACK to include eigenvalue software, level 3 BLAS for parallel and shared memory parallel machines - late 80s

**ScaLAPACK** - scaleable LAPACK based on BLACS for communication, distributed memory machine - mid 90s

# Efficiency on Cray C90

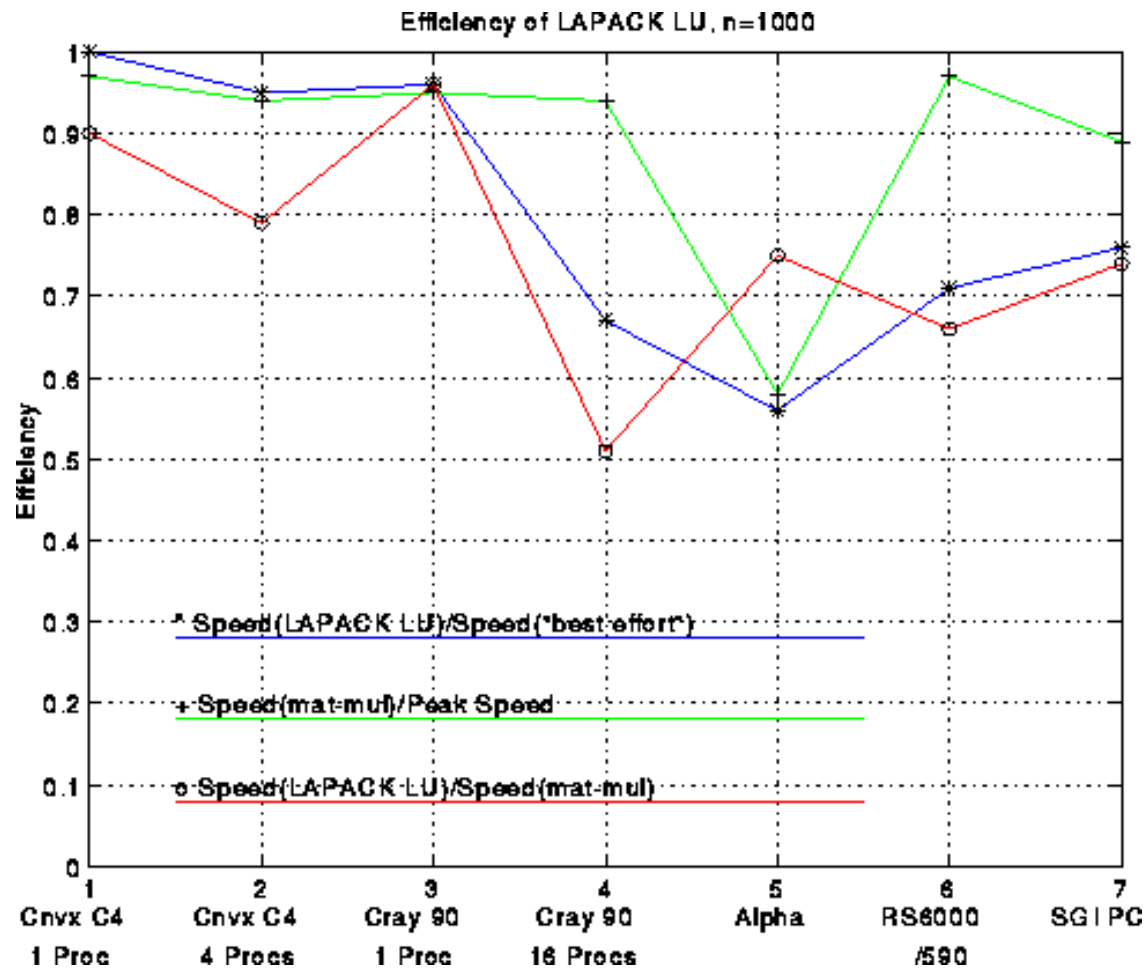


# Comparison of Different Machines

---

<b>Machine</b>	<b>#Procs</b>	<b>Clock Speed (MHz)</b>	<b>Peak Mflops</b>	<b>Block Size b</b>
<b>Convex C4640</b>	<b>1</b>	<b>135</b>	<b>810</b>	<b>64</b>
<b>Convex C4640</b>	<b>4</b>	<b>135</b>	<b>3240</b>	<b>64</b>
<b>Cray C90</b>	<b>1</b>	<b>240</b>	<b>952</b>	<b>128</b>
<b>Cray C90</b>	<b>16</b>	<b>240</b>	<b>15238</b>	<b>128</b>
<b>DEC Alpha 3000-500X</b>	<b>1</b>	<b>200</b>	<b>200</b>	<b>32</b>
<b>IBM RS 6000/590</b>	<b>1</b>	<b>66</b>	<b>264</b>	<b>64</b>
<b>SGI Power Challenge</b>	<b>1</b>	<b>75</b>	<b>300</b>	<b>64</b>

# Efficiency of LAPACK LU, for n=1000



## Efficiency of LAPACK LU, for $n=1000$

---

**LU factorization is almost as efficient as matrix-matrix multiply for most machines, except on C90 (16 processors). (why?)**

**LAPACK - LU is almost as good as best vendor effort. Trade-off between performance and portability.**

**Vendors place a premium on LU performance - why?**

# LINPACK Benchmark

---

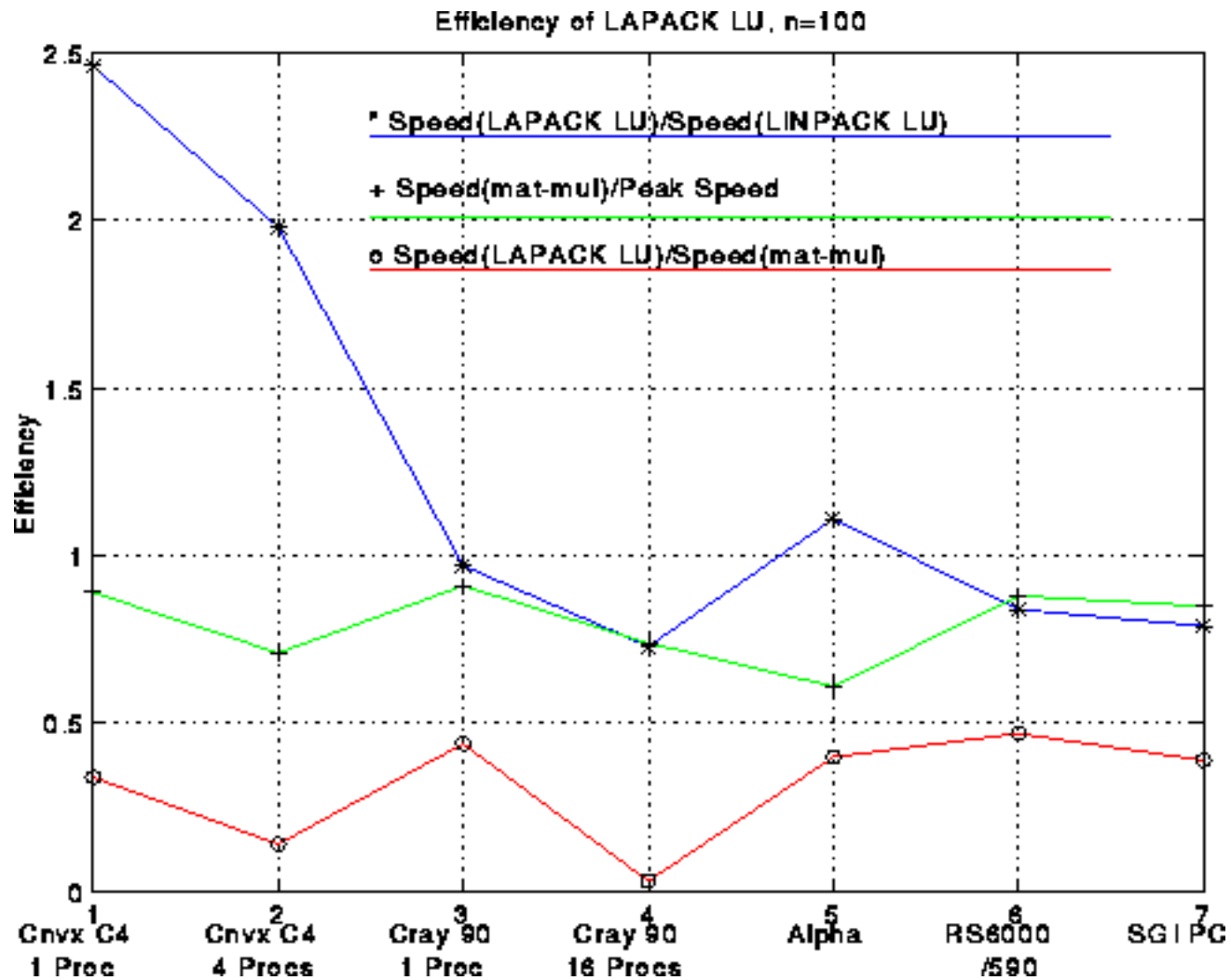
- named after the LINPACK package
- originally consisted of timings for 100-by-100 matrices; no vendor optimization(code changes) permitted
- interesting historical record, with literally every machine for the last 2 decades listed in decreasing order of speed, from the largest supercomputers to a hand-held calculator.
- as machines grew faster 1000-by-1000 matrices were introduced (all code changes allowed).
- a third benchmark was added for large parallel machines, which measured their speed on the largest linear system that would fit in memory, as well as the size of the system required to get half the Mflop rate of the largest matrix.

# LINPACK Benchmark

---

Computer	Num_Procs	Rmax(GFlops)	N
-----	-----	-----	---
Intel ASCI Option Red (200 MHz Pentium Pro)	7264	1068.	
CP-PACS* (150 MHz PA-RISC based CPU)	2048	368.2	
Intel Paragon XP/S MP (50 MHz OS=SUNMOS)	6768	281.1	
Intel Paragon XP/S MP (50 MHz OS=SUNMOS)	6144	256.2	
Numerical Wind Tunnel* (9.5 ns)	167	229.7	
Intel Paragon XP/S MP (50 MHz OS=SUNMOS)	5376	223.6	
HITACHI SR2201/1024(150MHz)	1024	220.4	
Fujitsu VPP500/153(10nsec)	153	200.6	
Numerical Wind Tunnel* (9.5 ns)	140	195.0	
Intel Paragon XP/S MP (50 MHz OS=SUNMOS)	4608	191.5	
Numerical Wind Tunnel* (9.5 ns)	128	179.2	

# Efficiency of LAPACK LU, for n=100



# Data Layouts for Distributed Memory Machines

---

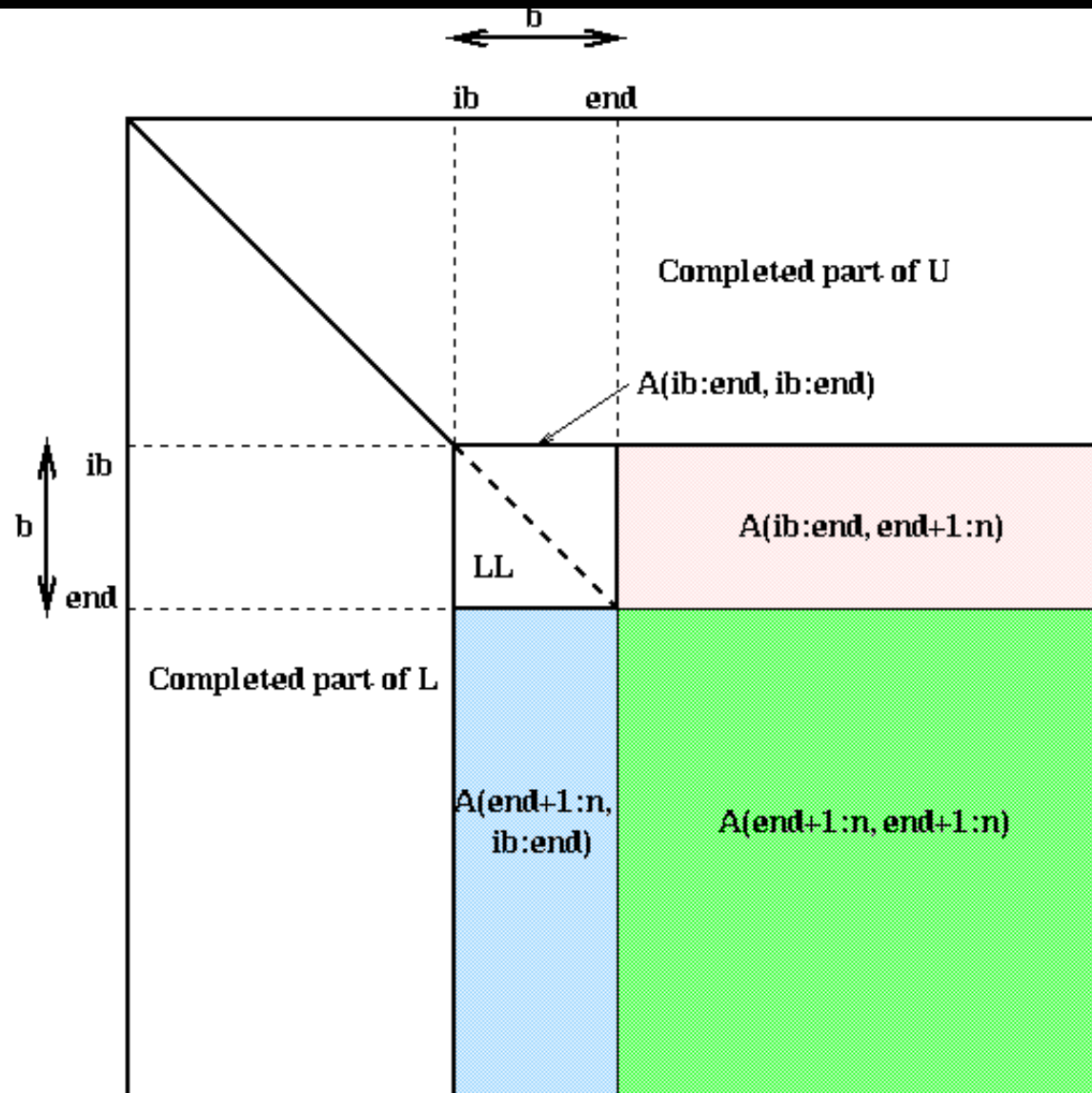
The two main issues in choosing a data layout for Gaussian elimination are

- 1) load balance, or splitting the work reasonably evenly among the processors
- 2) ability to use the BLAS3 during computations on a single processor, to account for the memory hierarchy on each processor.

Several layouts will be discussed here. All these are part of HPF. Solving linear systems served as a prototype for these designs.

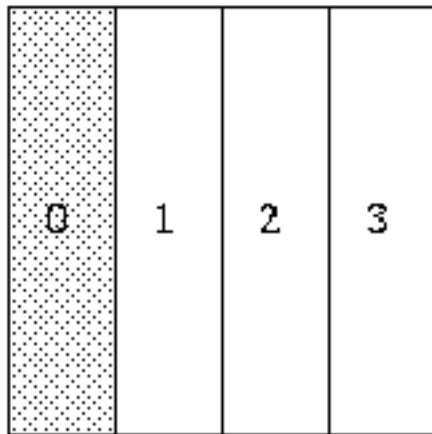
# Gaussian Elimination using BLAS 3

Gaussian Elimination using BLAS 3



# Column Blocked

---



1) Column Blocked Layout

**n=16 and p=4.**

**column  $i$  is stored on processor  $\text{floor}(i/c)$  where  $c=\text{ceiling}(n/p)$  is the maximum number of columns stored per processor.**

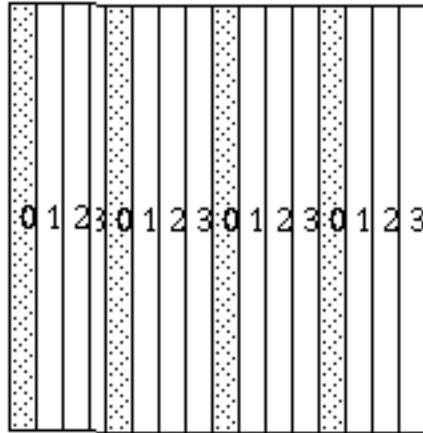
**does not permit good load balancing.**

**after  $c$  columns have been computed processor 0 is idle**

**row blocked has similar problem**

# Column Cyclic

---



2) Column Cyclic Layout

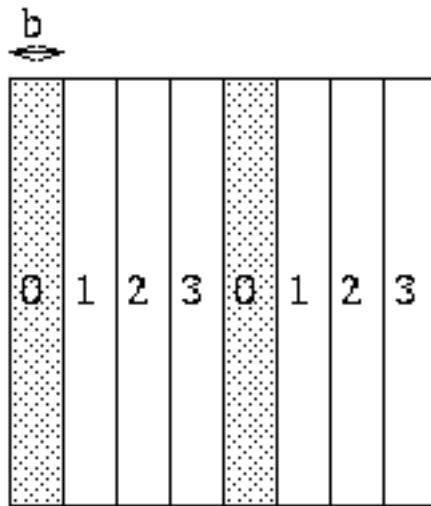
**each processor owns approximately 1/p-th of the square southeast corner of the matrix**

**good load balance**

**single columns are stored rather than blocks means we cannot use the BLAS3 to update**

**transpose of this layout, the Row Cyclic Layout, has a similar problem.**

# Column Block Cyclic



3) Column Block Cyclic Layout

**$n=16$ ,  $p=4$  and  $b=2$   
 $b$  not necessarily  
BLAS3 block size**

choose a block size  $b$ , divide the columns into groups of size  $b$ , distribute these groups cyclically

for  $b > 1$ , slightly worse balance than the Column Cyclic Layout;  
can use the BLAS2 and BLAS3

$b < c$ , better load balance than the Columns Blocked Layout, but can only call the BLAS on smaller subproblems, take less advantage of the local memory hierarchy

disadvantage that the factorization of  $A(ib:n, ib:end)$  will take place on perhaps just on one processor; possible serial bottleneck.

# Row and Column Block Cyclic

---

bcol

brow

0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3

## 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**

# Skewed Block

0	1	2	3
1	2	3	0
2	3	0	1
3	0	1	2

5) Block Skewed Layout

each row and each column is shared among all  $p$  processors

so  $p$ -fold parallelism is available for any row operation or any column operation

in contrast, the 2D block cyclic layout can have at most  $\sqrt{p}$ -fold parallelism in all the rows and all the columns

not useful for Gaussian elimination, but in a variety of other matrix operations

## Distributed GE with a 2D Block Cyclic Layout

---

**block size  $b$  in the algorithm and the block sizes  $b_{row}$  and  $b_{col}$  in the layout satisfy  $b=b_{row}=b_{col}$ .**

**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) find pivot row  $k$ , column broadcast

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

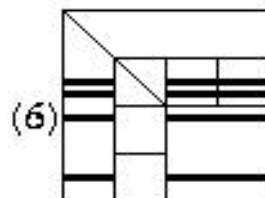
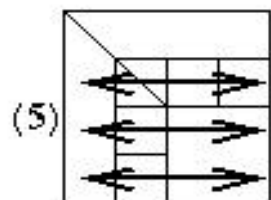
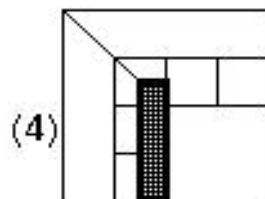
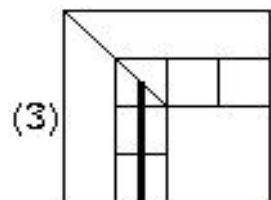
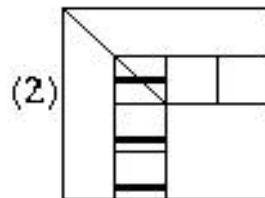
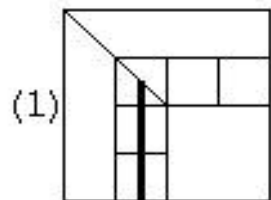
        (3)  $A(i+1:n, i) = A(i+1:n, i) / A(i, i)$

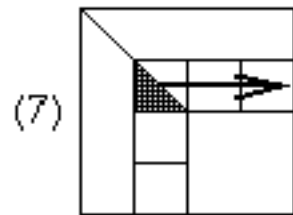
        (4)  $A(i+1:n, i+1:end) -= A(i+1:n, i) * A(i, i+1:end)$

    end for

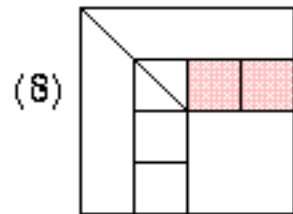
(5) broadcast all swap information right and left

(6) apply all rows swaps to other columns

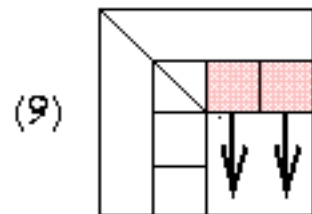




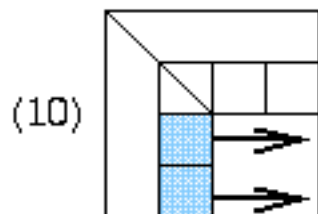
(7) Broadcast LL right



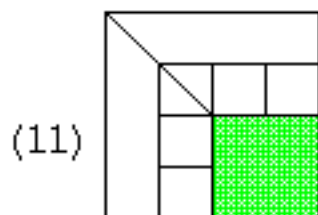
(8)  $A(ib:end, end+1:n) = LL \setminus A(ib:end, end+1:n)$



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



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



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

# ScaLAPACK LU Performance Results

---

	Processor Grid	Block Size	Values of $N$				
			2000	5000	7500	10000	15000
IBM SP2	1 × 4	50	426	606			
	2 × 8	50	767	1574	1925	2165	
	4 × 16	50	1026	3182	4433	5730	7151
Intel MP Paragon	1 × 4	32	215	283			
	2 × 8	32	479	876	1016		
	4 × 16	32	768	2147	2888	3383	3988
NoW SPARC Ultra ATM	2 × 2	64	193	336			
	2 × 4	64	193	491			
	2 × 6	64	207	611		939	

## Teraflop/s Performance Result

---

**“Sorry for the delay in responding. The system had about 7000 200Mhz Pentium Pro Processors. It solved a 64bit real matrix of size 216000. It did not use Strassen. The algorithm was basically the same that Robert van de Geijn used on the Delta years ago. It does a 2D block cyclic map of the matrix and requires a power of 2 number of nodes in the vertical direction. The basic block size was 64x64. A custom dual processor matrix multiply was written for the DGEMM call. It took a little less than 2 hours to run.”**