

# Parallel Numerical Algorithms

## Chapter 6 – LU Factorization

Prof. Michael T. Heath

Department of Computer Science  
University of Illinois at Urbana-Champaign

CSE 512 / CS 554



# Outline

- 1 LU Factorization
  - Motivation
  - Gaussian Elimination
- 2 Parallel Algorithms for LU
  - Fine-Grain Algorithm
  - Agglomeration Schemes
  - Scalability
- 3 Partial Pivoting

# LU Factorization

- System of linear algebraic equations has form

$$Ax = b$$

where  $A$  is given  $n \times n$  matrix,  $b$  is given  $n$ -vector, and  $x$  is unknown solution  $n$ -vector to be computed

- Direct method for solving general linear system is by computing *LU factorization*

$$A = LU$$

where  $L$  is unit lower triangular and  $U$  is upper triangular



# LU Factorization

- System  $Ax = b$  then becomes

$$LUx = b$$

- Solve lower triangular system

$$Ly = b$$

by forward-substitution to obtain vector  $y$

- Finally, solve upper triangular system

$$Ux = y$$

by back-substitution to obtain solution  $x$  to original system

# Gaussian Elimination Algorithm

LU factorization can be computed by Gaussian elimination as follows, where  $U$  overwrites  $A$

```

for  $k = 1$  to  $n - 1$                                 { loop over columns }
    for  $i = k + 1$  to  $n$                                 { compute multipliers
         $\ell_{ik} = a_{ik}/a_{kk}$                                 for current column }
    end
    for  $j = k + 1$  to  $n$ 
        for  $i = k + 1$  to  $n$                                 { apply transformation to
             $a_{ij} = a_{ij} - \ell_{ik}a_{kj}$                                 remaining submatrix }
        end
    end
end
    
```



# Gaussian Elimination Algorithm

- In general, row interchanges (pivoting) may be required to ensure existence of LU factorization and numerical stability of Gaussian elimination algorithm, but for simplicity we temporarily ignore this issue
- Gaussian elimination requires about  $n^3/3$  paired additions and multiplications, so model serial time as

$$T_1 = t_c n^3/3$$

where  $t_c$  is time required for multiply-add operation

- About  $n^2/2$  divisions also required, but we ignore this lower-order term



# Loop Orderings for Gaussian Elimination

- Gaussian elimination has general form of triple-nested loop in which entries of  $L$  and  $U$  overwrite those of  $A$

```
for _____  
  for _____  
    for _____  
       $a_{ij} = a_{ij} - (a_{ik}/a_{kk}) a_{kj}$   
    end  
  end  
end
```

- Indices  $i$ ,  $j$ , and  $k$  of **for** loops can be taken in any order, for total of  $3! = 6$  different ways of arranging loops



# Loop Orderings for Gaussian Elimination

- Different loop orders have different memory access patterns, which may cause their performance to vary widely, depending on architectural features such as cache, paging, vector registers, etc.
- Perhaps most promising for parallel implementation are  $kij$  and  $kji$  forms, which differ only in accessing matrix by rows or columns, respectively

# Gaussian Elimination Algorithm

- *kji* form of Gaussian elimination

```
for  $k = 1$  to  $n - 1$   
  for  $i = k + 1$  to  $n$   
     $l_{ik} = a_{ik}/a_{kk}$   
  end  
  for  $j = k + 1$  to  $n$   
    for  $i = k + 1$  to  $n$   
       $a_{ij} = a_{ij} - l_{ik} a_{kj}$   
    end  
  end  
end
```

- Multipliers  $l_{ik}$  computed outside inner loop for greater efficiency

# Parallel Algorithm

## Partition

- For  $i, j = 1, \dots, n$ , fine-grain task  $(i, j)$  stores  $a_{ij}$  and computes and stores

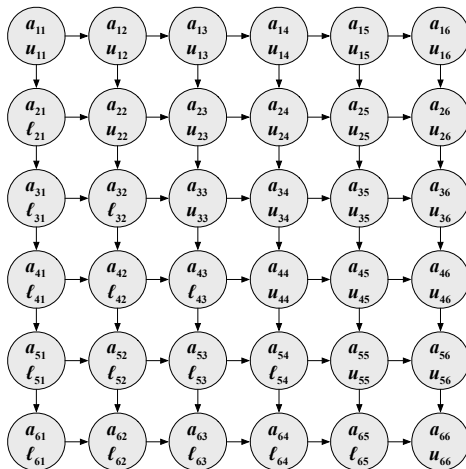
$$\begin{cases} u_{ij}, & \text{if } i \leq j \\ l_{ij}, & \text{if } i > j \end{cases}$$

yielding 2-D array of  $n^2$  fine-grain tasks

## Communicate

- Broadcast entries of  $A$  vertically to tasks below
- Broadcast entries of  $L$  horizontally to tasks to right

# Fine-Grain Tasks and Communication



# Fine-Grain Parallel Algorithm

```
for  $k = 1$  to  $\min(i, j) - 1$   
    recv broadcast of  $a_{kj}$  from task  $(k, j)$            { vert bcast }  
    recv broadcast of  $\ell_{ik}$  from task  $(i, k)$          { horiz bcast }  
     $a_{ij} = a_{ij} - \ell_{ik} a_{kj}$                          { update entry }  
end  
if  $i \leq j$  then  
    broadcast  $a_{ij}$  to tasks  $(k, j)$ ,  $k = i + 1, \dots, n$  { vert bcast }  
else  
    recv broadcast of  $a_{jj}$  from task  $(j, j)$            { vert bcast }  
     $\ell_{ij} = a_{ij}/a_{jj}$                                  { multiplier }  
    broadcast  $\ell_{ij}$  to tasks  $(i, k)$ ,  $k = j + 1, \dots, n$  { horiz bcast }  
end
```



# Agglomeration

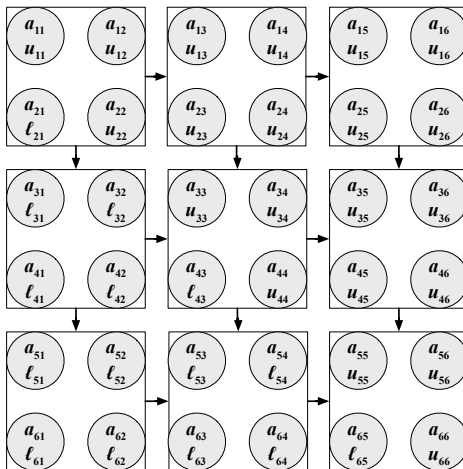
## *Agglomerate*

With  $n \times n$  array of fine-grain tasks, natural strategies are

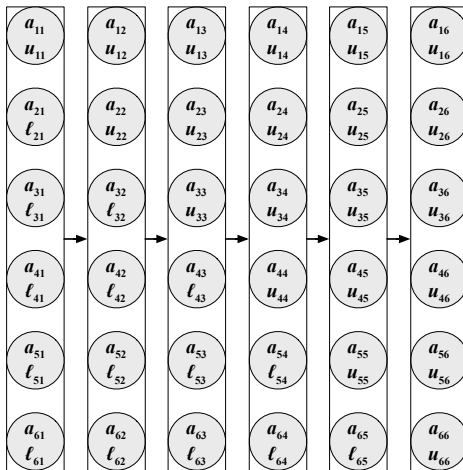
- 2-D: combine  $k \times k$  subarray of fine-grain tasks to form each coarse-grain task, yielding  $(n/k)^2$  coarse-grain tasks
- 1-D column: combine  $n$  fine-grain tasks in each column into coarse-grain task, yielding  $n$  coarse-grain tasks
- 1-D row: combine  $n$  fine-grain tasks in each row into coarse-grain task, yielding  $n$  coarse-grain tasks



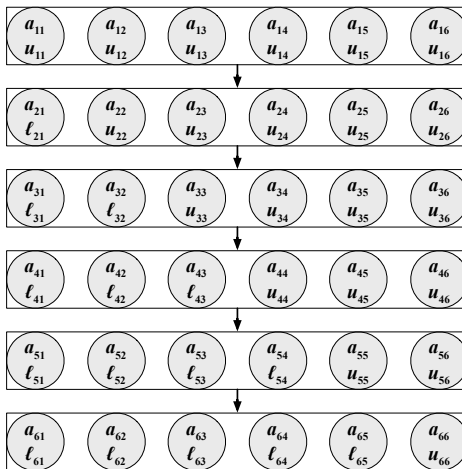
# 2-D Agglomeration



# 1-D Column Agglomeration



# 1-D Row Agglomeration



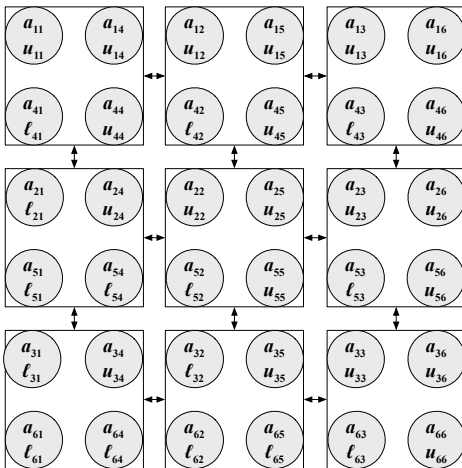
# Mapping

## *Map*

- 2-D: assign  $(n/k)^2/p$  coarse-grain tasks to each of  $p$  processes using any desired mapping in each dimension, treating target network as 2-D mesh
- 1-D: assign  $n/p$  coarse-grain tasks to each of  $p$  processes using any desired mapping, treating target network as 1-D mesh



# 2-D Agglomeration with Cyclic Mapping



# Coarse-Grain 2-D Parallel Algorithm

```
for  $k = 1$  to  $n - 1$   
  broadcast  $\{a_{kj} : j \in \text{mycols}, j \geq k\}$  in process column  
  if  $k \in \text{mycols}$  then  
    for  $i \in \text{myrows}, i > k$   
       $l_{ik} = a_{ik} / a_{kk}$                                 { multipliers }  
    end  
  end  
  broadcast  $\{l_{ik} : i \in \text{myrows}, i > k\}$  in process row  
  for  $j \in \text{mycols}, j > k$   
    for  $i \in \text{myrows}, i > k,$   
       $a_{ij} = a_{ij} - l_{ik} a_{kj}$                                 { update }  
    end  
  end  
end
```



# Performance Enhancements

- Each process becomes idle as soon as its last row and column are completed
- With block mapping, in which each process holds contiguous block of rows and columns, some processes become idle long before overall computation is complete
- Block mapping also yields unbalance load, as computing multipliers and updates requires successively less work with increasing row and column numbers
- Cyclic or reflection mapping improves both concurrency and load balance

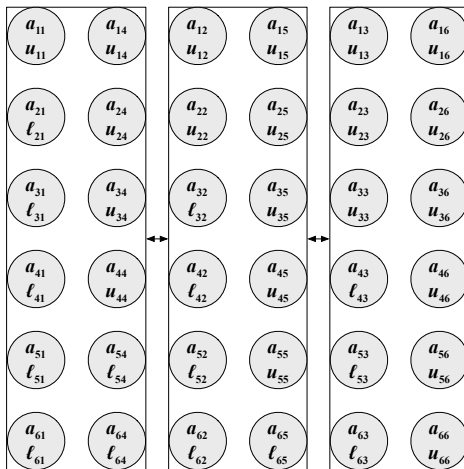


# Performance Enhancements

Performance can also be enhanced by overlapping communication and computation

- At step  $k$ , each process completes updating its portion of remaining unreduced submatrix before moving on to step  $k + 1$
- Broadcast of each segment of row  $k + 1$ , and computation and broadcast of each segment of multipliers for step  $k + 1$ , could be initiated as soon as relevant segments of row  $k + 1$  and column  $k + 1$  have been updated by their owners, before completing remainder of their updating for step  $k$
- This *send ahead* strategy enables other processes to start working on next step earlier than they otherwise could

# 1-D Column Agglomeration with Cyclic Mapping



# 1-D Column Agglomeration

- Matrix rows need not be broadcast vertically, since any given column is contained entirely in only one process
- But there is no parallelism in computing multipliers or updating any given column
- Horizontal broadcasts still required to communicate multipliers for updating

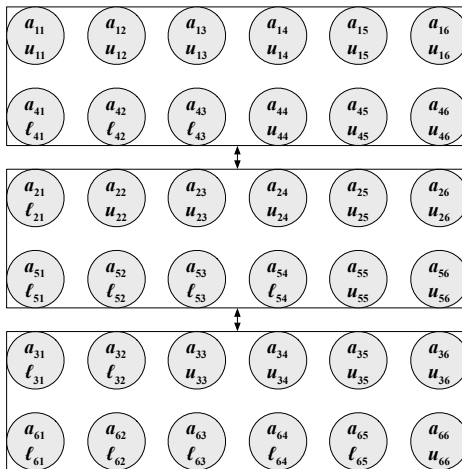


# Coarse-Grain 1-D Column Parallel Algorithm

```
for  $k = 1$  to  $n - 1$   
  if  $k \in \text{mycols}$  then  
    for  $i = k + 1$  to  $n$   
       $\ell_{ik} = a_{ik} / a_{kk}$                                 { multipliers }  
    end  
  end  
  broadcast  $\{\ell_{ik} : k < i \leq n\}$                        { broadcast }  
  for  $j \in \text{mycols}, j > k$   
    for  $i = k + 1$  to  $n$   
       $a_{ij} = a_{ij} - \ell_{ik} a_{kj}$                             { update }  
    end  
  end  
end
```



# 1-D Row Agglomeration with Cyclic Mapping



# 1-D Row Agglomeration

- Multipliers need not be broadcast horizontally, since any given matrix row is contained entirely in only one process
- But there is no parallelism in updating any given row
- Vertical broadcasts still required to communicate each row of matrix to processes below it for updating

# Coarse-Grain 1-D Row Parallel Algorithm

```
for  $k = 1$  to  $n - 1$   
  broadcast  $\{a_{kj} : k \leq j \leq n\}$            { broadcast }  
  for  $i \in \text{myrows}, i > k,$   
     $l_{ik} = a_{ik}/a_{kk}$                        { multipliers }  
  end  
  for  $j = k + 1$  to  $n$   
    for  $i \in \text{myrows}, i > k,$   
       $a_{ij} = a_{ij} - l_{ik} a_{kj}$                { update }  
    end  
  end  
end
```



# Performance Enhancements

- Same performance enhancements as for 2-D agglomeration apply to both 1-D column and 1-D row agglomerations as well, including cyclic mapping and send ahead strategy



# Scalability for 2-D Agglomeration

- Updating by each process at step  $k$  requires about  $(n - k)^2/p$  operations
- Summing over  $n - 1$  steps

$$\begin{aligned} T_{\text{comp}} &\approx t_c \sum_{k=1}^{n-1} (n - k)^2/p \\ &\approx t_c n^3/(3p) \end{aligned}$$



# Scalability for 2-D Agglomeration

- Similarly, amount of data broadcast at step  $k$  along each process row and column is about  $(n - k)/\sqrt{p}$ , so on 2-D mesh

$$\begin{aligned} T_{\text{comm}} &\approx \sum_{k=1}^{n-1} 2(t_s + t_w (n - k)/\sqrt{p}) \\ &\approx 2t_s n + t_w n^2/\sqrt{p} \end{aligned}$$

where we have allowed for overlap of broadcasts for successive steps



# Scalability for 2-D Agglomeration

- Total execution time is

$$T_p \approx t_c n^3 / (3p) + 2 t_s n + t_w n^2 / \sqrt{p}$$

- To determine isoefficiency function, set

$$t_c n^3 / 3 \approx E (t_c n^3 / 3 + 2 t_s n p + t_w n^2 \sqrt{p})$$

which holds for large  $p$  if  $n = \Theta(\sqrt{p})$ , so isoefficiency function is  $\Theta(p^{3/2})$ , since  $T_1 = \Theta(n^3)$



# Scalability for 1-D Agglomeration

- With either 1-D column or 1-D row agglomeration, updating by each process at step  $k$  requires about  $(n - k)^2/p$  operations
- Summing over  $n - 1$  steps

$$\begin{aligned} T_{\text{comp}} &\approx t_c \sum_{k=1}^{n-1} (n - k)^2/p \\ &\approx t_c n^3/(3p) \end{aligned}$$



# Scalability for 1-D Agglomeration

- Amount of data broadcast at step  $k$  is about  $n - k$ , so on 1-D mesh

$$\begin{aligned} T_{\text{comm}} &\approx \sum_{k=1}^{n-1} (t_s + t_w (n - k)) \\ &\approx t_s n + t_w n^2/2 \end{aligned}$$

where we have allowed for overlap of broadcasts for successive steps



# Scalability for 1-D Agglomeration

- Total execution time is

$$T_p \approx t_c n^3 / (3p) + t_s n + t_w n^2 / 2$$

- To determine isoefficiency function, set

$$t_c n^3 / 3 \approx E (t_c n^3 / 3 + t_s n p + t_w n^2 p / 2)$$

which holds for large  $p$  if  $n = \Theta(p)$ , so isoefficiency function is  $\Theta(p^3)$ , since  $T_1 = \Theta(n^3)$



# Partial Pivoting

- Row ordering of  $A$  is irrelevant in system of linear equations
- Partial pivoting takes rows in order of largest entry in magnitude of leading column of remaining unreduced matrix
- This choice ensures that multipliers do not exceed 1 in magnitude, which reduces amplification of rounding errors
- In general, partial pivoting is required to ensure existence and numerical stability of LU factorization



# Partial Pivoting

- Partial pivoting yields factorization of form

$$PA = LU$$

where  $P$  is permutation matrix

- If  $PA = LU$ , then system  $Ax = b$  becomes

$$PAx = LUx = Pb$$

which can be solved by forward-substitution in lower triangular system  $Ly = Pb$ , followed by back-substitution in upper triangular system  $Ux = y$



# Parallel Partial Pivoting

- Partial pivoting complicates parallel implementation of Gaussian elimination and significantly affects potential performance
- With 2-D algorithm, pivot search is parallel but requires communication within process column and inhibits overlapping of successive steps
- With 1-D column algorithm, pivot search requires no communication but is purely serial
- Once pivot is found, index of pivot row must be communicated to other processes, and rows must be explicitly or implicitly interchanged in each process



# Parallel Partial Pivoting

- With 1-D row algorithm, pivot search is parallel but requires communication among processes and inhibits overlapping of successive steps
- If rows are explicitly interchanged, then only two processes are involved
- If rows are implicitly interchanged, then mapping of rows to processes is altered, which may degrade concurrency and load balance
- Tradeoff between column and row algorithms with partial pivoting depends on relative speeds of communication and computation



# Alternatives to Partial Pivoting

- Because of negative effects of partial pivoting on parallel performance, various alternatives have been proposed that limit pivot search
  - pivoting only within block of rows
  - threshold pivoting
  - pairwise pivoting
- Such strategies are not foolproof and trade off some degree of stability and accuracy for speed
- Stability and accuracy may be recovered via iterative refinement, but this has its own cost



# References

- J. W. Demmel, M. T. Heath, and H. A. van der Vorst, Parallel numerical linear algebra, *Acta Numerica* 2:111-197, 1993
- G. A. Geist and C. H. Romine, LU factorization algorithms on distributed-memory multiprocessor architectures, *SIAM J. Sci. Stat. Comput.* 9:639-649, 1988
- A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing*, 2nd. ed., Addison-Wesley, 2003
- B. A. Hendrickson and D. E. Womble, The torus-wrap mapping for dense matrix calculations on massively parallel computers, *SIAM J. Sci. Stat. Comput.* 15:1201-1226, 1994

# References

- J. M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum Press, 1988
- J. M. Ortega and C. H. Romine, The  $ijk$  forms of factorization methods II: parallel systems, *Parallel Computing* 7:149-162, 1988
- Y. Robert, *The Impact of Vector and Parallel Architectures on the Gaussian Elimination Algorithm*, John Wiley & Sons, 1990
- S. A. Vavasis, Gaussian elimination with pivoting is P-complete, *SIAM J. Disc. Math.* 2:413-423, 1989