# Parallel Numerical Algorithms

## Chapter 4 – Sparse Linear Systems
## Section 4.2 – Banded Matrices

Michael T. Heath and Edgar Solomonik

Department of Computer Science
University of Illinois at Urbana-Champaign

CS 554 / CSE 512

# Outline

# Banded Linear Systems

- *Bandwidth* (or *semibandwidth*) of $n \times n$ matrix $\boldsymbol{A}$ is smallest value $w$ such that

$$a_{ij} = 0 \quad \text{for all} \quad |i - j| > w$$

- Matrix is *banded* if $w \ll n$
- If $w \gg p$, then minor modifications of parallel algorithms for dense LU or Cholesky factorization are reasonably efficient for solving banded linear system $\boldsymbol{Ax} = \boldsymbol{b}$
- If $w \lessapprox p$, then standard parallel algorithms for LU or Cholesky factorization utilize few processors and are very inefficient

# Narrow Banded Linear Systems

- More efficient parallel algorithms for narrow banded linear systems are based on *divide-and-conquer* approach in which band is partitioned into multiple pieces that are processed simultaneously
- Reordering matrix by nested dissection is one example of this approach
- Because of fill, such methods generally require more total work than best serial algorithm for system with dense band
- We will illustrate for tridiagonal linear systems, for which $w = 1$, and will assume pivoting is not needed for stability (e.g., matrix is diagonally dominant or symmetric positive definite)

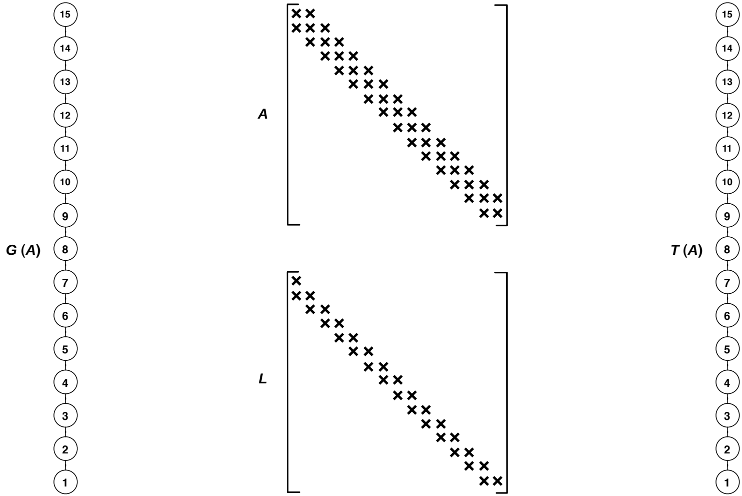
# Tridiagonal Linear System

- *Tridiagonal* linear system has form

$$\begin{bmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}$$

- For tridiagonal system of order $n$, LU or Cholesky factorization incurs no fill, but yields serial thread of length $\Theta(n)$ through task graph, and hence no parallelism
- Neither *cdivs* nor *cmods* can be done simultaneously

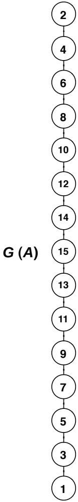
# Tridiagonal System, Natural Order

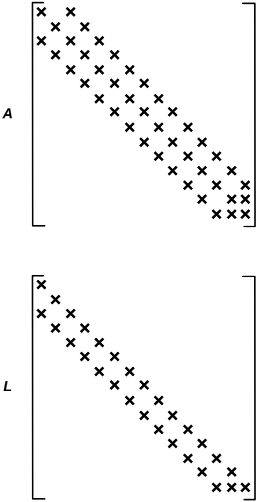


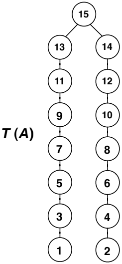
# Two-Way Elimination

- Other orderings may enable some degree of parallelism, however
- For example, elimination from both ends (sometimes called *twisted* factorization) yields two concurrent threads (odd-numbered nodes and even-numbered nodes) through task graph and still incurs no fill

# Tridiagonal System, Two-Way Elimination



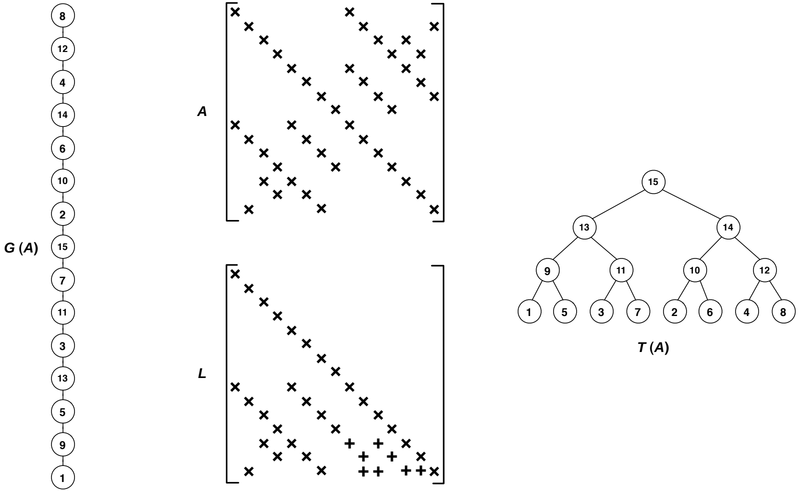



# Odd-Even Ordering

- Repeating this idea recursively gives *odd-even* ordering (variant of nested dissection), which yields even more parallelism, but incurs some fill

# Tridiagonal System, Odd-Even Ordering



# Cyclic Reduction

- Recursive nested dissection for tridiagonal system can be effectively implemented using *cyclic reduction* (or *odd-even reduction*)
- Linear combinations of adjacent equations in tridiagonal system are used to eliminate alternate unknowns
- Adding appropriate multiples of $(i-1)$st and $(i+1)$st equations to $i$th equation eliminates $x_{i-1}$ and $x_{i+1}$, respectively, from $i$th equation
- Resulting new $i$th equation involves $x_{i-2}$, $x_i$, and $x_{i+2}$, but not $x_{i-1}$ or $x_{i+1}$

# Cyclic Reduction

- For tridiagonal system, $i$th equation

$$a_i\, x_{i-1} + b_i\, x_i + c_i\, x_{i+1} = y_i$$

is transformed into

$$\bar{a}_i\, x_{i-2} + \bar{b}_i\, x_i + \bar{c}_i\, x_{i+2} = \bar{y}_i$$

where

$$\begin{aligned} \bar{a}_i &= \alpha_i\, a_{i-1}, & \bar{b}_i &= b_i + \alpha_i\, c_{i-1} + \beta_i\, a_{i+1} \\ \bar{c}_i &= \beta_i\, c_{i+1}, & \bar{y}_i &= y_i + \alpha_i\, y_{i-1} + \beta_i\, y_{i+1} \end{aligned}$$

with $\alpha_i = -a_i/b_{i-1}$ and $\beta_i = -c_i/b_{i+1}$

# Cyclic Reduction

- After transforming each equation in system (handling first two and last two equations as special cases), matrix of resulting new system has form

$$\begin{bmatrix} \bar{b}_1 & 0 & \bar{c}_1 & & & & \\ 0 & \bar{b}_2 & 0 & \bar{c}_2 & & & \\ \bar{a}_3 & 0 & \bar{b}_3 & 0 & \bar{c}_3 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & \bar{a}_{n-2} & 0 & \bar{b}_{n-2} & 0 & \bar{c}_{n-2} \\ & & & \bar{a}_{n-1} & 0 & \bar{b}_{n-1} & 0 \\ & & & & \bar{a}_n & 0 & \bar{b}_n \end{bmatrix}$$

# Cyclic Reduction

- Reordering equations and unknowns to place odd indices before even indices, matrix then has form

$$\begin{bmatrix} \bar{b}_1 & \bar{c}_1 & & & & & & & \\ \bar{a}_3 & \bar{b}_3 & \ddots & & & & & & \\ & \ddots & \ddots & \bar{c}_{n-3} & & & & & \\ & & \bar{a}_{n-1} & \bar{b}_{n-1} & 0 & & & & \\ & & & 0 & \bar{b}_2 & \bar{c}_2 & & & \\ & & & & \bar{a}_4 & \bar{b}_4 & \ddots & & \\ & & & & & \ddots & \ddots & \bar{c}_{n-2} \\ & & & & & & \bar{a}_n & \bar{b}_n \end{bmatrix}$$

# Cyclic Reduction

- System breaks into two independent tridiagonal systems that can be solved simultaneously (i.e., divide-and-conquer)
- Each resulting tridiagonal system can in turn be solved using same technique (i.e., recursively)
- Thus, there are two distinct sources of potential parallelism
  - simultaneous transformation of equations in system
  - simultaneous solution of multiple tridiagonal subsystems

# Cyclic Reduction

- Cyclic reduction requires $\log_2 n$ steps, each of which requires $\Theta(n)$ operations, so total work is $\Theta(n \log n)$
- Serially, cyclic reduction is therefore inferior to LU or Cholesky factorization, which require only $\Theta(n)$ work for tridiagonal system
- But in parallel, cyclic reduction can exploit up to $n$-fold parallelism and requires only $\Theta(\log n)$ time in best case
- Often matrix becomes approximately diagonal in fewer than $\log n$ steps, in which case reduction can be truncated and still attain acceptable accuracy

# Cyclic Reduction

- Cost for solving tridiagonal system by best serial algorithm is about

$$T_1 \approx 8\,\gamma\,n$$

  where $\gamma$ is time for one addition or multiplication

- Cost for solving tridiagonal system serially by cyclic reduction is about

$$T_1 \approx 12\,\gamma\,n\,\log_2 n$$

  which means that efficiency is less than 67%, even with $p = 1$

# Parallel Cyclic Reduction

- *Partition*: task $i$ stores and performs reductions on $i$th equation of tridiagonal system, yielding $n$ fine-grain tasks
- *Communicate*: data from "adjacent" equations is required to perform eliminations at each of $\log n$ stages
- *Agglomerate*: $n/p$ equations assigned to each of $p$ coarse-grain tasks, thereby limiting communication to only $\log p$ stages
- *Map*: Assigning contiguous rows to processors is better than cyclic mapping in this context
- "Local" tridiagonal system within each processor can be solved by serial cyclic reduction or by LU or Cholesky factorization

# Parallel Cyclic Reduction

- Parallel execution time for cyclic reduction is about

$$T_p \approx 12\,\gamma\,(n\,\log_2 n)/p + (\alpha + 4\,\beta)\,\log p$$

- Algorithm efficiency is $E_p = \Omega(1/\log n)$ relative to optimal serial counterpart, but relative to $T_1 = \Theta(n \log n)$, it is strongly and weakly log-scalable
- Can decrease work to $W_p = \Theta(n \log p)$ by doing work-efficient serial algorithm locally
- Can lower communication cost to $\Theta(\alpha \log_k p + \beta k \log_k p)$ by exchanging ghost-zones of size $k$ and doing $\log_2 k$ cyclic reduction steps per exchange

# Parallel Tridiagonal Linear Solve

- We can achieve asymptotic work-efficiency and log-scalability via the following scheme, let

$$\boldsymbol{A} = \underbrace{\begin{bmatrix} 0 & & & \\ x_1 & \ddots & & \\ & \ddots & \ddots & \\ & & x_{n-1} & 0 \end{bmatrix}}_{\mathcal{D}_{-1}(\boldsymbol{x})} + \underbrace{\begin{bmatrix} y_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & y_n \end{bmatrix}}_{\mathcal{D}_0(\boldsymbol{y})} + \underbrace{\begin{bmatrix} 0 & z_1 & & \\ & \ddots & \ddots & \\ & & \ddots & z_{n-1} \\ & & & 0 \end{bmatrix}}_{\mathcal{D}_{+1}(\boldsymbol{z})}$$

- Then define cyclic permutation matrix $\boldsymbol{P}$ so that

$$\boldsymbol{PAP}^T = \begin{bmatrix} \mathcal{D}_0(\boldsymbol{y}_{\text{odds}}) & \mathcal{D}_0(\boldsymbol{z}_{\text{odds}}) + \mathcal{D}_{-1}(\boldsymbol{x}_{\text{evens}}) \\ \mathcal{D}_0(\boldsymbol{x}_{\text{odds}}) + \mathcal{D}_{+1}(\boldsymbol{z}_{\text{evens}}) & \mathcal{D}_0(\boldsymbol{y}_{\text{evens}}) \end{bmatrix}$$

  for which LU factorization reduces to a tridiagonal matrix of dimension $n/2$ using $O(n)$ fully concurrent work

# Block Tridiagonal Systems

- Relatively fine granularity may make cyclic reduction impractical for solving single tridiagonal system on some parallel architectures
- Efficiency may be much better, however, if there are many right-hand sides for single tridiagonal system or many independent tridiagonal systems to solve
- Cyclic reduction is also applicable to *block tridiagonal* systems, which have larger granularity and hence more favorable ratio of communication to computation and potentially better efficiency

# Iterative Methods

- Tridiagonal and other banded systems are often amenable to efficient parallel solution by iterative methods
- For example, successive diagonal blocks of tridiagonal system can be assigned to separate tasks, which can solve "local" tridiagonal system as preconditioner for iterative method for overall system

# References – Banded Systems


- J. Dongarra and S. Johnsson, Solving banded systems on a parallel processor, *Parallel Computing* 5:219-246, 1987
- S. L. Johnsson, Solving narrow banded systems on ensemble architectures, *ACM Trans. Math. Software* 11:271-288, 1985
- E. Polizzi and A. H. Sameh, A parallel hybrid banded system solver: the SPIKE algorithm, *Parallel Computing* 32:177-194, 2006
- Y. Saad and M. Schultz, Parallel direct methods for solving banded linear systems, *Linear Algebra Appl.* 88:623-650, 1987

# References – Tridiagonal Systems


- L.-W. Chang, J. A. Stratton, H.-S. Kim, and W.-M. W. Hwu, A scalable, numerically stable, high-performance tridiagonal solver using GPUs, SC12, Salt Lake City, Utah, November 10-16, 2012
- Ö. Eğecioğlu, C. K. Koc, and A. J. Laub, A recursive doubling algorithm for solution of tridiagonal systems on hypercube multiprocessors, *J. Comput. Appl. Math.* 27:95-108, 1989
- M. Hegland, On the parallel solution of tridiagonal systems by wrap-around partitioning and incomplete LU factorization, *Numer. Math.* 59:453-472, 1991
- S. L. Johnsson, Solving tridiagonal systems on ensemble architectures, *SIAM J. Sci. Stat. Comput.* 8:354-392, 1987

# References – Tridiagonal Systems


- A. Krechel, H.-J. Plum, and K. Stüben, Parallelization and vectorization aspects of the solution of tridiagonal linear systems, *Parallel Computing* 14:31-49, 1990
- V. Mehrmann, Divide and conquer methods for block tridiagonal systems, *Parallel Computing* 19:257-280, 1993
- E. E. Santos, Optimal and efficient parallel tridiagonal solvers using direct methods, *J. Supercomputing* 30:97-115, 2004
- X.-H. Sun and W. Zhang, A parallel two-level hybrid method for tridiagonal systems and its application to fast Poisson solvers, *IEEE Trans. Parallel Distrib. Sys.*, 15:97-106, 2004

# References – Multifrontal Methods


- I. S. Duff, Parallel implementation of multifrontal schemes, *Parallel Computing* 3:193-204, 1986
- A. Gupta, Parallel sparse direct methods: a short tutorial, IBM Research Report RC 25076, November 2010
- J. Liu, The multifrontal method for sparse matrix solution: theory and practice, *SIAM Review* 34:82-109, 1992
- J. A. Scott, Parallel frontal solvers for large sparse linear systems, *ACM Trans. Math. Software* 29:395-417, 2003

# References – Scalability


- A. George, J. Lui, and E. Ng, Communication results for parallel sparse Cholesky factorization on a hypercube, *Parallel Computing* 10:287-298, 1989
- A. Gupta, G. Karypis, and V. Kumar, Highly scalable parallel algorithms for sparse matrix factorization, *IEEE Trans. Parallel Distrib. Systems* 8:502-520, 1997
- T. Rauber, G. Runger, and C. Scholtes, Scalability of sparse Cholesky factorization, *Internat. J. High Speed Computing* 10:19-52, 1999
- R. Schreiber, Scalability of sparse direct solvers, A. George, J. R. Gilbert, and J. Liu, eds., *Graph Theory and Sparse Matrix Computation*, pp. 191-209, Springer-Verlag, 1993

## References – Nonsymmetric Sparse Systems


- I. S. Duff and J. A. Scott, A parallel direct solver for large sparse highly unsymmetric linear systems, *ACM Trans. Math. Software* 30:95-117, 2004
- A. Gupta, A shared- and distributed-memory parallel general sparse direct solver, *Appl. Algebra Engrg. Commun. Comput.*, 18(3):263-277, 2007
- X. S. Li and J. W. Demmel, SuperLU_Dist: A scalable distributed-memory sparse direct solver for unsymmetric linear systems, *ACM Trans. Math. Software* 29:110-140, 2003
- K. Shen, T. Yang, and X. Jiao, S+: Efficient 2D sparse LU factorization on parallel machines, *SIAM J. Matrix Anal. Appl.* 22:282-305, 2000