

CSE 412/CS 454/MATH 486

Parallel Numerical Algorithms

11. Cholesky Factorization

Prof. Michael T. Heath
Department of Computer Science
University of Illinois at Urbana-Champaign

Copyright © 2002, Michael T. Heath – p.155

Cholesky Factorization

- Symmetric positive definite matrix A has *Cholesky factorization*

$$A = LL^T$$

where L is lower triangular matrix with positive diagonal entries

- Linear system

$$Ax = b$$

can then be solved by forward-substitution in lower triangular system $Ly = b$, followed by back-substitution in upper triangular system $L^T x = y$

Copyright © 2002, Michael T. Heath – p.255

Computing Cholesky Factorization

- Algorithm for computing Cholesky factorization can be derived by equating corresponding entries of A and LL^T and generating them in correct order
- For example, in 2×2 case

$$\begin{bmatrix} a_{11} & a_{21} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} \ell_{11} & 0 \\ \ell_{21} & \ell_{22} \end{bmatrix} \begin{bmatrix} \ell_{11} & \ell_{21} \\ 0 & \ell_{22} \end{bmatrix}$$

- So we have

$$\ell_{11} = \sqrt{a_{11}}, \quad \ell_{21} = a_{21}/\ell_{11}, \quad \ell_{22} = \sqrt{a_{22} - \ell_{21}^2}$$

Copyright © 2002, Michael T. Heath – p.355

Computing Cholesky Factorization

- All n square roots are of positive numbers, so algorithm is well defined
- Only lower triangle of A is accessed, so strict upper triangular portion need not be stored
- Factor L is computed in place, overwriting lower triangle of A
- Pivoting is not required for numerical stability
- About $n^3/6$ multiplications and similar number of additions are required (about half as many as for LU)

Copyright © 2002, Michael T. Heath – p.555

Computing Cholesky Factorization

```

for  $k = 1$  to  $n$ 
   $a_{kk} = \sqrt{a_{kk}}$ 
  for  $i = k + 1$  to  $n$ 
     $a_{ik} = a_{ik}/a_{kk}$ 
  end
  for  $j = k + 1$  to  $n$ 
    for  $i = j$  to  $n$ 
       $a_{ij} = a_{ij} - a_{ik} a_{jk}$ 
    end
  end
end

```

Copyright © 2002, Michael T. Heath – p.455

Parallel Algorithm for Cholesky

Partition

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

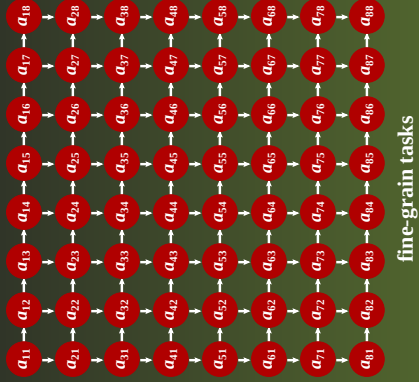
$$\begin{cases} \ell_{ij}, & \text{if } i \geq j \\ \ell_{ji}, & \text{if } i < j \end{cases}$$

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

- Zero entries in upper triangle of L need not be computed or stored, so for convenience in using 2-D mesh network we redundantly compute ℓ_{ij} , for $i > j$, as both task (i, j) and task (j, i)

Copyright © 2002, Michael T. Heath – p.655

Parallel Algorithm for Cholesky



Copyright © 2002, Michael T. Heath – p.7/55

Parallel Algorithm for Cholesky

```

else
  recv  $a_{j,j}$ 
   $a_{i,j} = a_{i,j} / a_{j,j}$ 
  broadcast  $a_{i,j}$  to tasks  $(i, k), k = j + 1, \dots, n$ 
end

```

Copyright © 2002, Michael T. Heath – p.9/55

Implementations of Cholesky

Each choice of index in outer loop yields different algorithm, named for portion of matrix updated by basic operation in inner loops

- *Submatrix-Cholesky*: with k in outer loop, inner loops perform rank-1 update of remaining unreduced *submatrix* using current column
- *Column-Cholesky*: with j in outer loop, inner loops compute current *column* using matrix-vector product that accumulates effects of previous columns
- *Row-Cholesky*: with i in outer loop, inner loops compute current *row* by solving triangular system involving previous rows

Copyright © 2002, Michael T. Heath – p.11/55

Parallel Algorithm for Cholesky

```

Program for task  $(i, j)$ 
for  $k = 1$  to  $\min(i, j) - 1$ 
  recv  $a_{k,j}$ 
  recv  $a_{i,k}$ 
   $a_{i,j} = a_{i,j} - a_{i,k} a_{k,j}$ 
end
if  $i = j$  then
   $a_{ii} = \sqrt{a_{ii}}$ 
  broadcast  $a_{ii}$  to tasks  $(k, i)$  and  $(i, k), k = i + 1, \dots, n$ 
else if  $i < j$  then
  recv  $a_{ii}$ 
   $a_{i,j} = a_{i,j} / a_{ii}$ 
  broadcast  $a_{i,j}$  to tasks  $(k, j), k = i + 1, \dots, n$ 

```

Copyright © 2002, Michael T. Heath – p.8/55

Parallel Algorithm for Cholesky

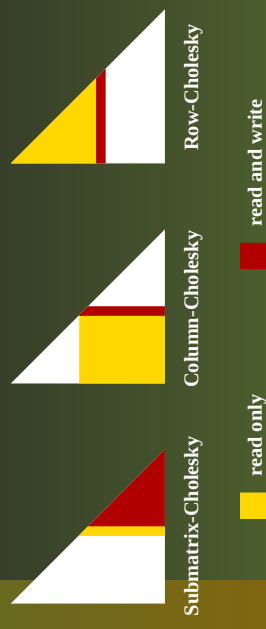
Agglomeration

- Agglomeration of fine-grain tasks produces
 - 1-D row
 - 1-D column
 - 2-D
- parallel algorithms analogous to those for LU factorization, with similar performance and scalability
- Column-oriented algorithms are most often used in practice, especially for sparse matrices, so we focus primarily on that case

Copyright © 2002, Michael T. Heath – p.10/55

Implementations of Cholesky

Memory access patterns of different implementations differ markedly



Copyright © 2002, Michael T. Heath – p.12/55

Implementations of Cholesky

Submatrix-Cholesky

```

for  $k = 1$  to  $n$ 
   $a_{kk} = \sqrt{a_{kk}}$ 
  for  $i = k + 1$  to  $n$ 
     $a_{ik} = a_{ik} / a_{kk}$ 
  end
  for  $j = k + 1$  to  $n$ 
    for  $i = j$  to  $n$ 
       $a_{ij} = a_{ij} - a_{ik} a_{jk}$ 
    end
     $a_{jj} = \sqrt{a_{jj}}$ 
    for  $i = j + 1$  to  $n$ 
       $a_{ij} = a_{ij} / a_{jj}$ 
    end
  end
end
end

```

Copyright © 2002, Michael T. Heath – p.1355

Column-Cholesky

```

for  $j = 1$  to  $n$ 
  for  $k = 1$  to  $j - 1$ 
    for  $i = j$  to  $n$ 
       $a_{ij} = a_{ij} - a_{ik} a_{jk}$ 
    end
  end
   $a_{jj} = \sqrt{a_{jj}}$ 
  for  $i = j + 1$  to  $n$ 
     $a_{ij} = a_{ij} / a_{jj}$ 
  end
end
end

```

Copyright © 2002, Michael T. Heath – p.1355

Column Operations

Column-oriented algorithms can be stated more compactly by introducing column operations

- $cdiv(j)$: column j is divided by square root of its diagonal entry

```

 $a_{jj} = \sqrt{a_{jj}}$ 
for  $i = j + 1$  to  $n$ 
   $a_{ij} = a_{ij} / a_{jj}$ 
end

```

- $cmod(j, k)$: column j is modified by multiple of column k , with $k < j$
- ```

for $i = j$ to n
 $a_{ij} = a_{ij} - a_{ik} a_{jk}$
end

```

Copyright © 2002, Michael T. Heath – p.1455

## Implementations of Cholesky

### Submatrix-Cholesky

```

for $k = 1$ to n
 $cdiv(k)$
 for $j = k + 1$ to n
 $cmod(j, k)$
 end
end
end

```

- right-looking
- immediate-update
- data-driven
- fan-out

### Column-Cholesky

```

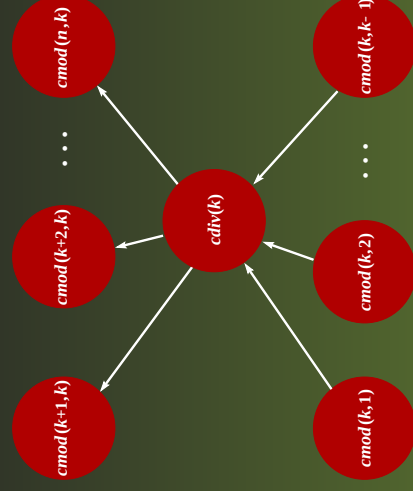
for $j = 1$ to n
 for $k = 1$ to $j - 1$
 $cmod(j, k)$
 end
 $cdiv(j)$
end
end

```

- left-looking
- delayed-update
- demand-driven
- fan-in

Copyright © 2002, Michael T. Heath – p.1355

## Data Dependences



Copyright © 2002, Michael T. Heath – p.1655

## Data Dependences

- $cmod(k, *)$  operations along bottom can be done in any order, but they all have same target column, so updating must be coordinated to preserve data integrity
- $cmod(*, k)$  operations along top can be done in any order, and they all have different target columns, so updating can be done simultaneously
- Performing  $cmods$  concurrently is most important source of parallelism in column-oriented factorization algorithms
- For dense matrix, each  $cdiv(k)$  depends on immediately preceding column, so  $cdivs$  must be done sequentially

Copyright © 2002, Michael T. Heath – p.1755

## Sparse Matrices

- Matrix is *sparse* if most of its entries are zero
- Want to store and operate on only nonzero entries
  - e.g.,  $cmod(j, k)$  need not be done if  $a_{jk} = 0$
- But more complicated data structures required incur extra overhead in storage and arithmetic operations
- Matrix is “usefully” sparse if it contains enough zero entries to be worth taking advantage of them to reduce storage and work required
- In practice, sparsity worth exploiting for family of matrices if there are only  $O(n)$  nonzero entries, i.e., (small) constant number of nonzeros per row or column

Copyright © 2002, Michael T. Heath – p.1855

## Sparsity Structure

- For sparse matrix  $M$ , let  $M_{i*}$  denote its  $i$ th row and  $M_{*j}$  its  $j$ th column
- Define  $Struct(M_{i*}) = \{k < i \mid m_{ik} \neq 0\}$
- Define  $Struct(M_{*j}) = \{k > j \mid m_{kj} \neq 0\}$
- $Struct(M_{i*})$  is nonzero structure of row  $i$  of strict lower triangle of  $M$
- $Struct(M_{*j})$  is nonzero structure of column  $j$  of strict lower triangle of  $M$

Copyright © 2002, Michael T. Heath – p.19/55

## Sparse Cholesky Algorithms

### Submatrix-Cholesky

```

for $k = 1$ to n
 $cdiv(k)$
 for $j \in Struct(L_{*k})$
 $cmod(j, k)$
 end
end

```

- right-looking
- immediate-update
- data-driven
- fan-out

### Column-Cholesky

```

for $j = 1$ to n
 for $k \in Struct(L_{j*})$
 $cmod(j, k)$
 end
 $cdiv(j)$
end

```

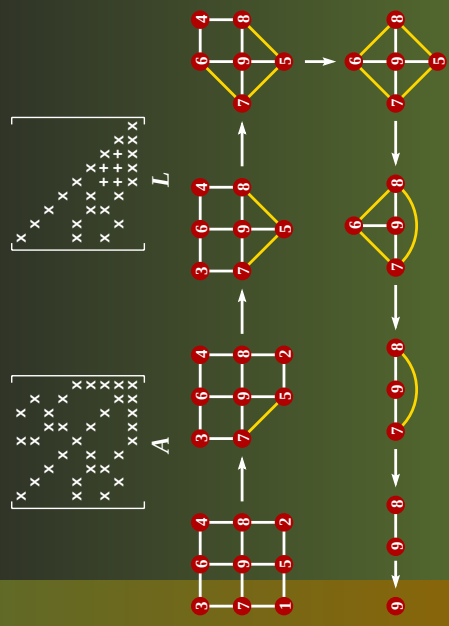
- left-looking
- delayed-update
- demand-driven
- fan-in

Copyright © 2002, Michael T. Heath – p.20/55

## Graph Representation

- Graph  $G(\mathcal{A})$  of symmetric  $n \times n$  matrix  $\mathcal{A}$  is undirected graph having  $n$  vertices, with edge between vertices  $i$  and  $j$  if  $a_{ij} \neq 0$
- At each step of factorization, corresponding vertex is eliminated from graph
- Neighbors of eliminated vertex in previous graph become *clique* (fully connected subgraph) in modified graph
- Entries of  $\mathcal{A}$  that were initially zero may become nonzero; such new nonzeros are called *fill*

Copyright © 2002, Michael T. Heath – p.21/55



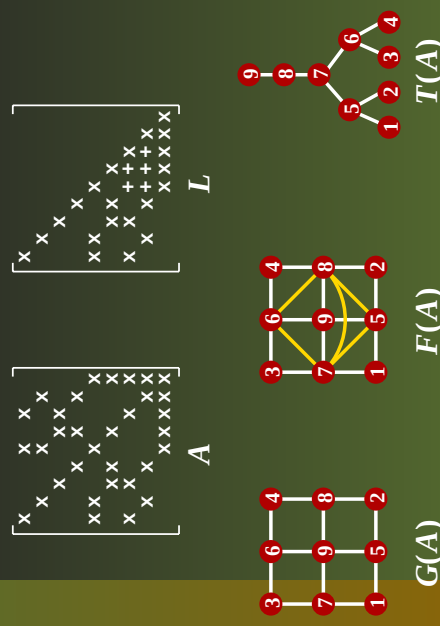
Copyright © 2002, Michael T. Heath – p.22/55

## Elimination Tree

- $parent(j)$  is row index of first offdiagonal nonzero in column  $j$  of  $L$ , if any, and  $j$  otherwise
- Elimination tree  $T(\mathcal{A})$  is graph having  $n$  vertices, with edge between vertices  $i$  and  $j$ , for  $i > j$ , if  $i = parent(j)$
- If matrix is irreducible, then elimination tree is single tree with root at vertex  $n$ ; otherwise, it is more accurately termed *elimination forest*
- $T(\mathcal{A})$  is spanning tree for *filled graph*  $F(\mathcal{A})$ , which is  $G(\mathcal{A})$  with all fill edges added
- Each column of Cholesky factor  $L$  depends only on its descendants in elimination tree

Copyright © 2002, Michael T. Heath – p.23/55

## Example: Elimination Tree



Copyright © 2002, Michael T. Heath – p.24/55

## Ordering

- Amount of fill depends on order in which variables are eliminated
- Example: “arrow” matrix — if first row and column are dense, then factor fills in completely, but if last row and column are dense, then they cause no fill

$$\begin{bmatrix} x & x & x & x & x & x & x \\ x & x & & & & & \\ x & & x & & & & \\ x & & & x & & & \\ x & & & & x & & \\ x & & & & & x & \\ x & & & & & & x \end{bmatrix}$$

$$\begin{bmatrix} x & & & & & & \\ & x & & & & & \\ & & x & & & & \\ & & & x & & & \\ & & & & x & & \\ & & & & & x & \\ & & & & & & x \end{bmatrix}$$

Copyright © 2002, Michael T. Heath – p.25/55

## Symbolic Factorization

- For SPD matrices, ordering can be determined in advance of numeric factorization
- Only locations of nonzeros matter, not their numerical values, since pivoting is not required for numerical stability
- Once ordering is selected, locations of all fill entries in  $L$  can be anticipated and efficient static data structure set up to accommodate them prior to numeric factorization
- Structure of column  $j$  of  $L$  is given by union of structures of lower triangular portion of column  $j$  of  $A$  and prior columns of  $L$  whose first nonzero below diagonal is in row  $j$

Copyright © 2002, Michael T. Heath – p.27/55

## Parallel Sparse Cholesky

- In sparse submatrix- or column-Cholesky, if  $a_{jk} = 0$ , then  $cm\odot(j, k)$  is omitted
- Sparse factorization thus has additional source of parallelism, since “missing”  $cm\odot$ s may permit multiple  $cd\imath$ s to be done simultaneously
- Elimination tree shows data dependences among columns of Cholesky factor  $L$ , and hence identifies potential parallelism
- At any point in factorization process, all factor columns corresponding to *leaf* nodes of elimination tree can be computed simultaneously

Copyright © 2002, Michael T. Heath – p.29/55

## Ordering Heuristics

General problem of finding ordering to minimize fill is NP-complete, but there are cheap heuristics that effectively limit fill

- *Bandwidth or profile reduction*: reduce distance of nonzero diagonals from main diagonal (e.g., RCM)
- *Minimum degree*: eliminate node having fewest neighbors first
- *Nested dissection*: recursively split graph into pieces, numbering nodes in *separators* last

Copyright © 2002, Michael T. Heath – p.26/55

## Solving Sparse SPD Systems

Basic steps in solving sparse SPD systems by Cholesky factorization

1. *Ordering*: Reorder rows and columns of matrix so Cholesky factor suffers relatively little fill
2. *Symbolic factorization*: Determine locations of all fill entries and allocate data structures in advance to accommodate them
3. *Numeric factorization*: Compute numeric values of entries of Cholesky factor
4. *Triangular solution*: Compute solution by forward- and back-substitution

Copyright © 2002, Michael T. Heath – p.28/55

## Parallel Sparse Cholesky

- *Height* of elimination tree determines longest serial path through computation, and hence parallel completion time
- *Width* of tree determines degree of parallelism available
- Short, wide, well-balanced elimination tree desirable for parallel factorization
- Structure of elimination tree depends on ordering of matrix
- So ordering should be chosen *both* to preserve sparsity and to enhance parallelism

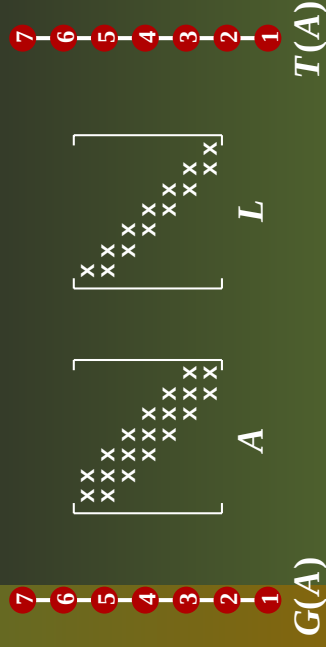
Copyright © 2002, Michael T. Heath – p.30/55

## Levels of Parallelism

- **Fine-grain**
  - Task is one multiply-add pair
  - Available in either dense or sparse case
  - Difficult to exploit effectively in practice
- **Medium-grain**
  - Task is one *cmod* or *cdiv*
  - Available in either dense or sparse case
  - Accounts for most of speedup in dense case
- **Large-grain**
  - Task computes entire set of columns in subtree of elimination tree
  - Available only in sparse case

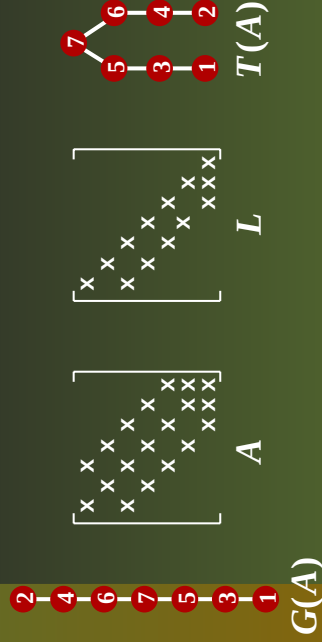
Copyright © 2002, Michael T. Heath – p.33/55

## Example: Band Ordering



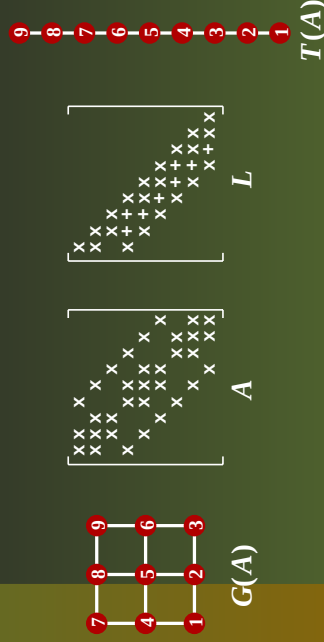
Copyright © 2002, Michael T. Heath – p.32/55

## Example: Minimum Degree



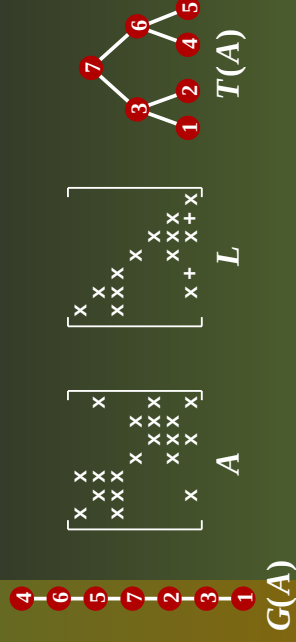
Copyright © 2002, Michael T. Heath – p.33/55

## Example: Band Ordering



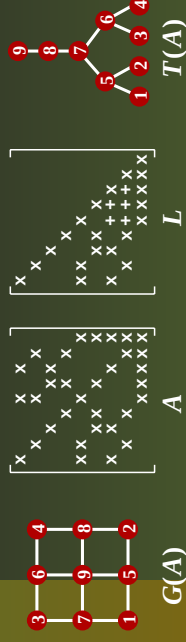
Copyright © 2002, Michael T. Heath – p.35/55

## Example: Nested Dissection



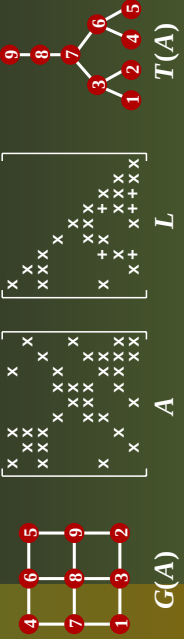
Copyright © 2002, Michael T. Heath – p.34/55

## Example: Minimum Degree



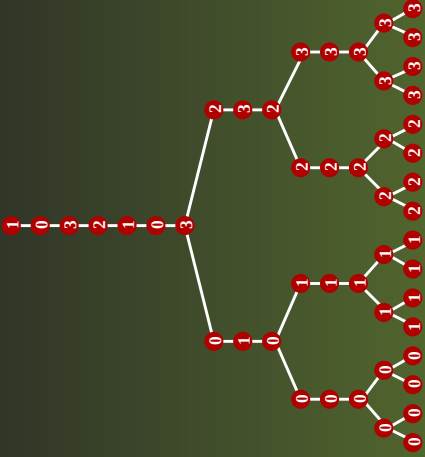
Copyright © 2002, Michael T. Heath – p.36/55

## Example: Nested Dissection



Copyright © 2002, Michael T. Heath – p.38/55

## Subtree Mapping



Copyright © 2002, Michael T. Heath – p.38/55

## Fan-Out Sparse Cholesky

```

while $mycols \neq \emptyset$
 receive any column of L , say $L_{*:k}$
 for $j \in mycols \cap Struct(L_{*:k})$
 $cmod(j, k)$
 if column j requires no more $cmods$ then
 $cdiv(j)$
 send $L_{*:j}$ to tasks in
 $map(Struct(L_{*:j}))$
 $mycols = mycols - \{j\}$
 end
 end
end

```

Copyright © 2002, Michael T. Heath – p.41/55

## Mapping

- Cyclic mapping of columns to processors works well for dense problems, because it balances load and communication is global anyway
- To exploit locality in communication for sparse factorization, better approach is to map columns in subtree of elimination tree onto local subset of processors
- Still use cyclic mapping within dense submatrices (“supernodes”)

Copyright © 2002, Michael T. Heath – p.38/55

## Fan-Out Sparse Cholesky

```

for $j \in mycols$
 if j is leaf node in $T(A)$ then
 $cdiv(j)$
 send $L_{*:j}$ to tasks in
 $map(Struct(L_{*:j}))$
 $mycols = mycols - \{j\}$
 end
end

```

Copyright © 2002, Michael T. Heath – p.40/55

## Fan-In Sparse Cholesky

```

for $j = 1$ to n
 if $j \in mycols$ or $mycols \cap Struct(L_{j*}) \neq \emptyset$ then
 $u = 0$
 for $k \in mycols \cap Struct(L_{j*})$
 $u = u + \ell_{jk} L_{*:k}$
 if $j \in mycols$ then
 incorporate u into factor column j
 while any aggregated update
 column for column j remains,
 receive one and incorporate it
 into factor column j
 end
 end
 $cdiv(j)$
 end
end

```

Copyright © 2002, Michael T. Heath – p.42/55

## Fan-In Sparse Cholesky

```

else
 send u to task $map(j)$
end
end
end
end

```

Copyright © 2002, Michael T. Heath – p.43/55

## Multifrontal Method

- *Extend\_add* operation, denoted by  $\oplus$ , merges matrices by taking union of their subscript sets and summing entries for any common subscripts

Copyright © 2002, Michael T. Heath – p.45/55

## Multifrontal Algorithm

Factor( $j$ )

Let  $\{\hat{i}_1, \dots, \hat{i}_r\} = \text{Struct}(L_{*j})$

$$\text{Let } \mathbf{F}_j = \begin{bmatrix} a_{j,j} & a_{j,\hat{i}_1} & \dots & a_{j,\hat{i}_r} \\ a_{\hat{i}_1,j} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{\hat{i}_r,j} & 0 & \dots & 0 \end{bmatrix}$$

For each child  $i$  of  $j$  in elimination tree

Factor( $i$ )  
 $\mathbf{F}_j = \mathbf{F}_j \oplus \mathbf{U}_i$   
 end

Copyright © 2002, Michael T. Heath – p.47/55

## Multifrontal Method

- Multifrontal algorithm operates recursively, starting from root of elimination tree for  $\mathbf{A}$
- Dense frontal matrix  $\mathbf{F}_j$  is initialized to have nonzero entries from corresponding row and column of  $\mathbf{A}$  as its first row and column, and zeros elsewhere
- $\mathbf{F}_j$  is then updated by *extend\_add* operations with update matrices from its children in elimination tree
- After updating of  $\mathbf{F}_j$  is complete, its partial Cholesky factorization is computed, producing corresponding row and column of  $\mathbf{L}$  as well as update matrix  $\mathbf{U}_j$

Copyright © 2002, Michael T. Heath – p.44/55

## Example: *Extend\_add*

$$\begin{bmatrix} a_{11} & a_{13} & a_{15} & a_{18} \\ a_{31} & a_{33} & a_{35} & a_{38} \\ a_{51} & a_{53} & a_{55} & a_{58} \\ a_{81} & a_{83} & a_{85} & a_{88} \end{bmatrix} \oplus \begin{bmatrix} b_{11} & b_{12} & b_{15} & b_{17} \\ b_{21} & b_{22} & b_{25} & b_{27} \\ b_{51} & b_{52} & b_{55} & b_{57} \\ b_{71} & b_{72} & b_{75} & b_{77} \end{bmatrix} = \begin{bmatrix} a_{11} + b_{11} & b_{12} & a_{13} & a_{15} + b_{15} & b_{17} & a_{18} \\ b_{21} & b_{22} & 0 & b_{25} & b_{27} & 0 \\ a_{31} & 0 & a_{33} & a_{35} & 0 & a_{38} \\ a_{51} + b_{51} & b_{52} & a_{53} & a_{55} + b_{55} & b_{57} & a_{58} \\ b_{71} & b_{72} & 0 & b_{75} & b_{77} & 0 \\ a_{81} & 0 & a_{83} & a_{85} & 0 & a_{88} \end{bmatrix}$$

Copyright © 2002, Michael T. Heath – p.46/55

## Multifrontal Algorithm

Perform one step of dense Cholesky:

$$\mathbf{F}_j = \begin{bmatrix} \ell_{j,j} & & & \\ \ell_{i_1,j} & & & \\ \vdots & & & \\ \ell_{i_r,j} & & & \end{bmatrix} \begin{bmatrix} 1 & & & \\ & o & & \\ & & \ell_{j,j} & \\ & & \ell_{i_1,j} & \dots & \ell_{i_r,j} \end{bmatrix} \begin{bmatrix} o & & & \\ & \mathbf{U}_j & & \\ & & I & \end{bmatrix}$$

Copyright © 2002, Michael T. Heath – p.48/55

## Advantages of Multifrontal Method

- Most arithmetic operations performed on dense matrices, which reduces indexing overhead and indirect addressing
- Can take advantage of loop unrolling, vectorization, and optimized BLAS to run at near peak speed on many types of processors
- Data locality good for memory hierarchies, such as cache, virtual memory with paging, or explicit out-of-core solvers

Copyright © 2002, Michael T. Heath – p.49/55

## Advantages of Multifrontal Method

- Naturally adaptable to parallel implementation by processing multiple independent fronts simultaneously on different processors
- Parallelism can also be exploited in dense matrix computations within each front

Copyright © 2002, Michael T. Heath – p.50/55

## Summary

Principal ingredients in efficient parallel algorithm for sparse Cholesky factorization

- Ordering to obtain relatively short and well balanced elimination tree while also limiting fill
- Multifrontal or supernodal approach to exploit dense subproblems effectively
- Subtree mapping to localize communication
- Cyclic mapping of dense subproblems to achieve good load balance
- 2-D algorithm for dense subproblems to enhance scalability

Copyright © 2002, Michael T. Heath – p.51/55

## Scalability of Sparse Cholesky

- Performance and scalability of sparse Cholesky depend on sparsity structure of particular matrix
- Sparse factorization requires factorization of dense matrix of size  $\mathcal{O}(\sqrt{n})$  for 2-D grid problem with  $n$  grid points, so isoefficiency function is at least  $\mathcal{O}(p^3)$  for 1-D algorithm and  $\mathcal{O}(p^{3/2})$  for 2-D algorithm
- More detailed analysis of sparse case, as well as empirical evidence, suggests that isoefficiency function is at least  $\mathcal{O}(p^2)$  even for 2-D algorithm
- Thus, even best current algorithms for this problem are not very scalable

Copyright © 2002, Michael T. Heath – p.52/55

## References

- J. W. Demmel, M. T. Heath, and H. A. van der Vorst, Parallel numerical linear algebra, *Acta Numerica* 2:111-197, 1993
- I. S. Duff, Parallel implementation of multifrontal schemes, *Parallel Computing* 3:193-204, 1986
- A. George and J. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice Hall, 1981
- M. T. Heath, Parallel direct methods for sparse linear systems, D. E. Keyes, A. Sameh, and V. Venkatarishman, eds., *Parallel Numerical Algorithms*, pp. 55-90, Kluwer, 1997
- M. T. Heath, E. Ng and B. W. Peyton, Parallel algorithms for sparse linear systems, *SIAM Review* 33:420-460, 1991

Copyright © 2002, Michael T. Heath – p.53/55

## References

- J. Liu, Computational models and task scheduling for parallel sparse Cholesky factorization, *Parallel Computing* 3:327-342, 1986
- J. Liu, Reordering sparse matrices for parallel elimination, *Parallel Computing* 11:73-91, 1989
- J. Liu, The role of elimination trees in sparse factorization, *SIAM J. Matrix Anal. Appl.* 11:134-172, 1990
- J. Liu, The multifrontal method for sparse matrix solution: theory and practice, *SIAM Review* 34:82-109, 1992
- D. O'Leary and G. W. Stewart, Data-flow algorithms for parallel matrix computations, *Comm. ACM* 28:840-853, 1985

Copyright © 2002, Michael T. Heath – p.54/55

## References

- D. O'Leary and G. W. Stewart, Assignment and scheduling in parallel matrix factorization, *Linear Algebra Appl.* 77:275-299, 1986
- 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