How to Achieve 10+ Gflops with Direct Linear Solvers

Xiangmin Jiao

April 15, 1999

Overview

1. Background review (2 minutes)
2. Sequential implementation of symmetric solvers (3 minutes)
3. Parallel implementation of symmetric solvers (25 minutes)
4. Parallel sparse LU factorization with pivoting (10 minutes)
5. Available software packages (5 minutes)

Review: Phases of Sparse Direct Solver

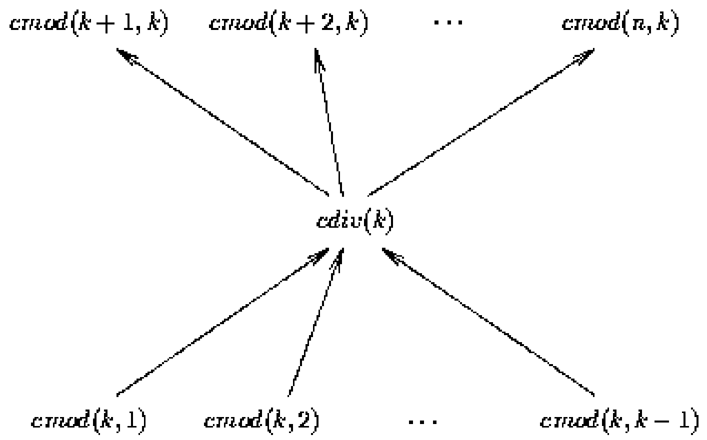
- A system of linear equation $Ax = b$
- Preprocessing steps
  - Ordering
  - Symbolic factorization
  - Insight: elimination tree
- **Numerical factorization**
  - Fan-in, fan-out, fan-both (Ashcraft)
  - Multifrontal, supernodal
- Triangular solution
- Iterative refinement

Boost Sequential Performance

- Reduce workload
  - Ordering: to reduce fills and hence arithmetic operations
- Reduce runtime overhead
  - Symbolic factorization: avoid memory management overhead
- Avoid indirect memory access
- Improve caching performance
- Exploit microparallelism
  - Multifrontal methods
  - Supernodal approaches
  - Optimized dense kernels (BLAS)

Potential Parallelism: Medium-Grain

```
for j = 1, n
    for k = 1, j − 1
        for i = j, n
            a_ij = a_ij − a_ik · a_jk        {cmod(j, k)}
    a_jj = sqrt(a_jj)
    for k = j + 1, n
        a_kj = a_kj / a_jj                   {cdiv(j)}
```

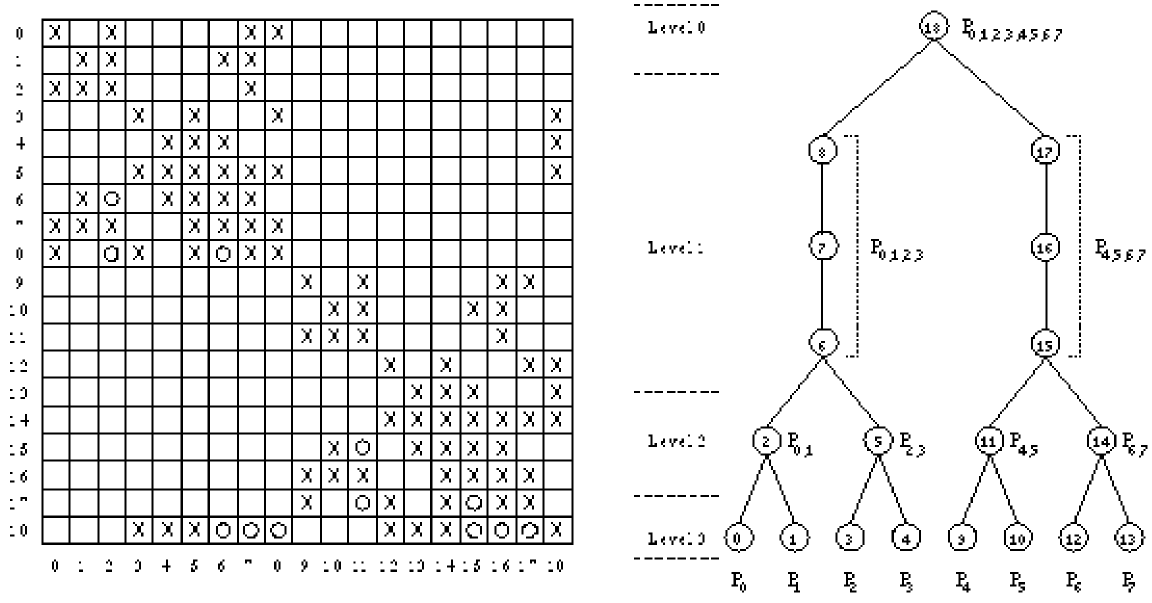


At the bottom: $cmod(k,*)$ can be done in any order.

At the top: $cmod(*,k)$ can be done simultaneously.

Very important for both dense and sparse solvers.

Potential Parallelism: Large-Grain

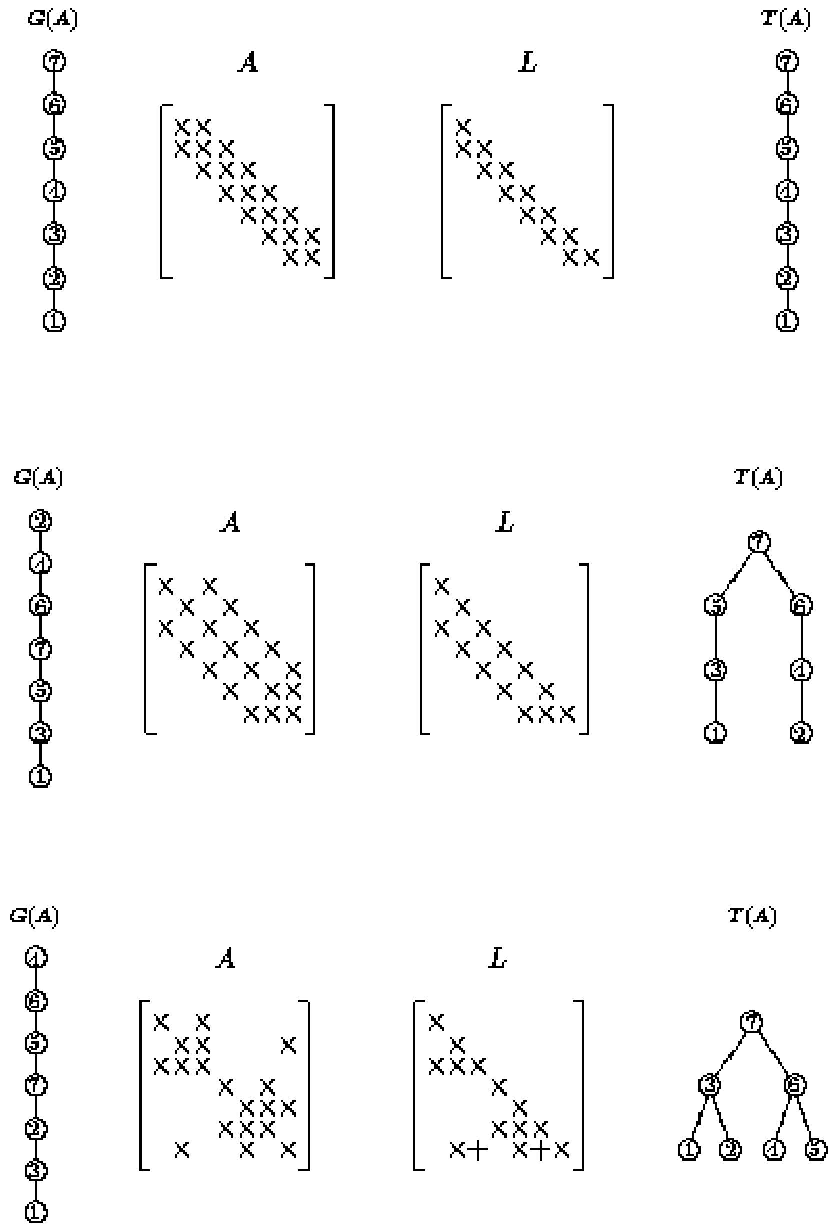


(a) (b)

The computation of a subtree can be done in parallel.

Only available in sparse solvers, especially for larger systems.

Impact of Ordering on Parallelism



Tradeoff between fill and parallelism.

Task Assignment

Shared or distributed memory?

- Shared memory
  - Data shared by processors
  - Task queue and dynamic load balancing
  - Challenge: synchronization

- **Distributed memory**
  - Data distributed across processors
  - Domain decomposition
  - Static task assignment and load balancing
  - Challenge: communication and load balancing

Task Assignment Cont'd

- Block or cyclic mapping, or combinations
  - Often used in dense cases
  - Risk wasting large-grain parallelism
  - Incur unnecessary communication
- Assign subtrees of elimination tree to neighboring processors
  - "Subtree-to-subcube" with hypercube
  - "Subtree-to-submesh" with mesh

Fan-out and Fan-in Algorithms

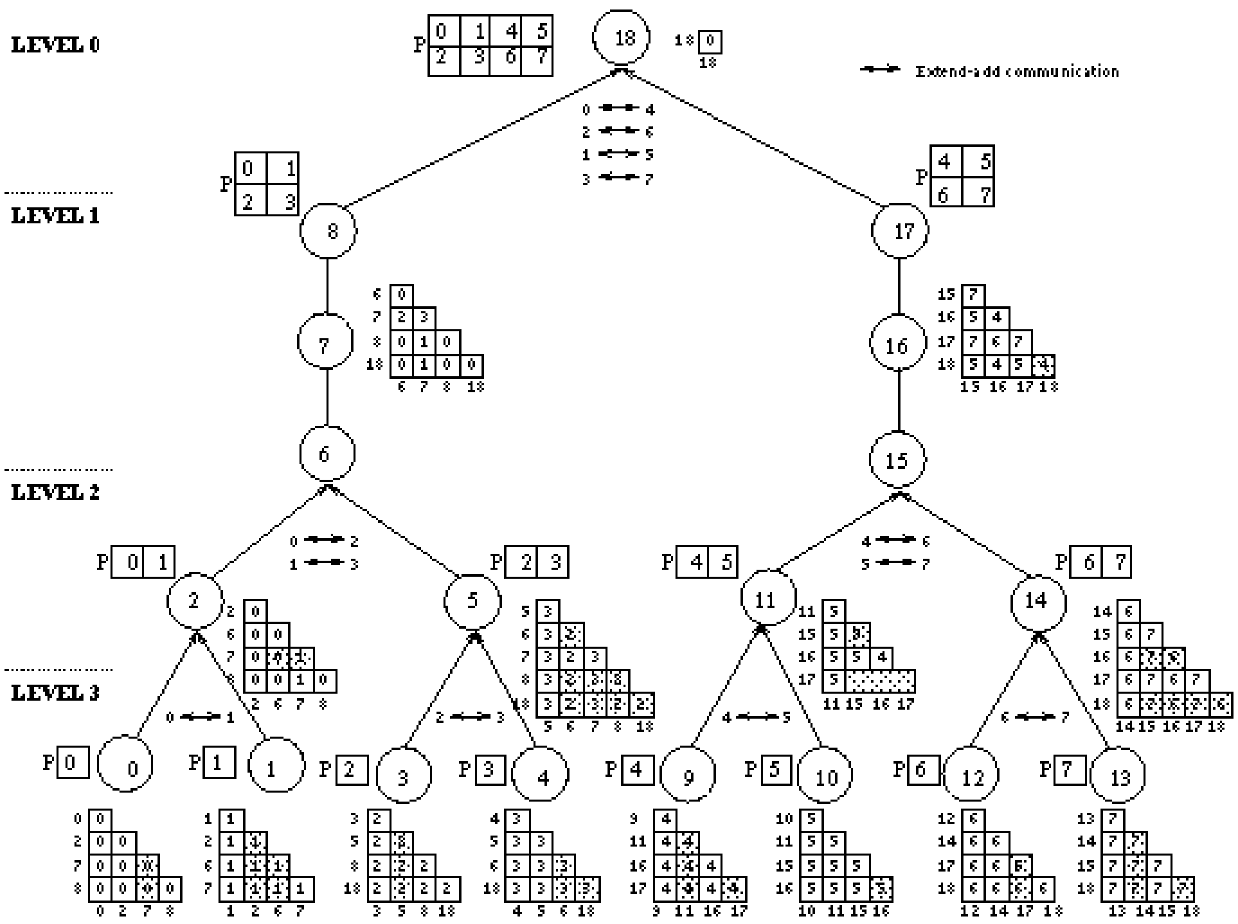
- Fan-out sparse Cholesky factorization
  - ✔ Simplest to implement
  - ✔ Exploit both large- and medium-grain parallelism
  - ✘ High communication frequency and volume
  - ✘ Relatively inefficient computation inner loop
- Fan-in sparse Cholesky factorization
  - ✔ Exploit both large- and medium-grain parallelism
  - ✔ Decrease communication frequency and volume
  - ✔ More efficient computational inner loop
  - ✔ High efficiency for sufficiently large problems
  - ✘ Sparse kernel computation

Supernodal Approaches

- Partition the matrix into supernodes
- Relaxed supernodes for better caching performance
- Put a limit on supernode size for better caching and more parallelism
- Extract a supernodal e-tree from the nodal e-tree

$$A = \begin{pmatrix}
1 & \bullet & \bullet & \bullet & & \bullet & \bullet & & \bullet & \bullet & \bullet \\
\bullet & 2 & & & & & & & & & \\
\bullet & & 3 & & & & & & & & \\
\bullet & & & 4 & & & & & & & \\
 & & & & 5 & \bullet & & & & & \\
\bullet & & & & \bullet & 6 & & & & & \\
\bullet & & & & & & 7 & & & & \\
 & & & & & & & 8 & \bullet & & \\
\bullet & & & & & & & \bullet & 9 & & \\
\bullet & & & & & & & & & 10 & \\
\bullet & & & & & & & & & & 11
\end{pmatrix}
\qquad
L = \begin{pmatrix}
1 & & & & & & & & & & \\
\bullet & 2 & & & & & & & & & \\
\bullet & \bullet & 3 & & & & & & & & \\
\bullet & \bullet & \bullet & 4 & & & & & & & \\
 & & & & 5 & & & & & & \\
\bullet & \bullet & \bullet & \bullet & \bullet & 6 & & & & & \\
\bullet & \bullet & \bullet & \bullet & & \bullet & 7 & & & & \\
 & & & & & & & 8 & & & \\
\bullet & \bullet & \bullet & \bullet & & \bullet & \bullet & \bullet & 9 & & \\
\bullet & \bullet & \bullet & \bullet & & \bullet & \bullet & & \bullet & 10 & \\
\bullet & \bullet & \bullet & \bullet & & \bullet & \bullet & & \bullet & \bullet & 11
\end{pmatrix}$$

Multifrontal Algorithm



Ingredients of a Parallel Sparse Cholesky

- Ordering for short and well-balanced e-tree and limiting fill
- Multifrontal approach to exploit dense subproblems
- Subtree-to-subcube mapping for reduced communication
- Block-cyclic mapping of subproblems for load balancing
- Two dimensional partitioning for scalability

Implement Nonsymmetric Solvers

- Pivoting
  ✔ Life-saver for numerical stability, but
  ✘ Trouble-maker for implementation
- Limit fill
  – Apply symmetric ordering on $A^T A$ (or $A^T + A$).
  – COLAMD. Larimore, Davis, Gilbert, and Ng. 1998.
- Manage fill (memory allocation)
  – Static or dynamic symbolic factorization
    * Tradeoff between management simplicity and resource economy
  – Special pivoting
- Improve single node performance
  – Multifrontal approach
  – Supernodal approach (fan-in, fan-out, fan-both)

Sparse LU Factorization with Partial Pivoting

Static symbolic factorization

✔ Simpler, and hence easier to build a 2D code
✔ Mostly BLAS-3 kernel due to more structured supernodes
✘ Overestimates fills and introduces extra operations
  ✔ typically around 30%, and performs surprisingly well
  ✘ for about 15% matrices, overestimates fills up to 300%

Dynamic symbolic factorization

✔ No (or less) extra fills and operations
✘ More sophisticated and expensive runtime control
✘ Hard to predict resource needs

Hybrid approaches?

Limit Dynamic Change Using Special Pivoting

- Gaussian elimination with *static pivoting*
  - Perform pivoting statically
  - Pre-pivot large elements to the diagonal (before ordering for sparsity)
  - Control diagonal magnitude on the fly
  - Monitor error and use iterative refinement and extra precision when needed
  - X. Li, and J. Demmel 1998.

- Gaussian elimination with *rook pivoting*
  - Favor local maximum over global maximum
  - User provided threshold
  - Adopted by SPOOLES (Ashcraft et al.)

Available Software Packages

	mat	parallel	ord	sym	num	tri	note
SPOOLES	ns	mt/MPI	s,three		sn,fi	p,bf	BPW,1D
SuperLU _MT	ns	mt	s,colmmd	p,dyn	sn,fi	s,bf	UCB,1D
WSSMP	s	mt/MPI	p,nd	s,sta	mf	p,bf	IBM,2D
PSPASES	spd	mt/MPI	ParMetis	p,sta	mf	p,bf	UMN,2D
CAPSS	spd	MPI	p,cnd	p,sta	mf	p,bf	UTK
SuperLU	ns	-	colmmd	dyn	sn,fi	bf	UCB
UMFPACK	ns	-	colamd		mf	bf	UFL

For a comprehensive list:
`http://www.netlib.org/utk/people/JackDongarra/la-sw.html`

Summary

- Factors affect performance
  - Single-node speed
    - Caching, indirect indexing, microparallelism
    - Multifrontal and supernodal
  - Concurrency
    - Medium-grain, and large-grain
  - Computational overhead
    - Task and data management, overestimated fill
  - Communication overhead
    - Fan-out (most), Fan-in(less), Multifrontal(least)
  - Load balance
    - Block-cyclic
    - Subtree-to-subcube
  - Scalability: next section
- They interact with each other
- To achieve 10+ Gflops *easily*
  - SPD: Use existing high-quality packages
  - Nonsymmetric: Good luck!

References

- M. T. Heath, Parallel direct methods for sparse linear systems, in *Parallel Numerical Algorithms*, ed. by D. E. Keyes, A. Sameh, and V. Venkatakrishnan, Kluwer Academic Publishers, Boston, 1997, pp. 55-90.
- A. Gupta, G. Karypis, and V. Kumar, Highly scalable parallel algorithms for sparse matrix factorization.
- A. Gupta, F. Gustavson, M. Joshi, G. Karypis, and V. Kumar, Design and implementation of a scalable parallel direct solver for sparse symmetric positive definite systems: preliminary results.
- J. Demmel, J. Gilbert, and X. Li, An asynchronous parallel supernodal algorithm for sparse Gaussian elimination, *SIAM J. Matrix Anal. Appl.*.
- X. Li and J. Demmel. Making Sparse Gaussian Elimination Scalable by Static Pivoting. *Proceedings of Supercomputing 98*.