A Hybrid Multithreaded Direct Sparse Triangular Solver

Andrew M. Bradley


SIAM CSC16 SAND2016-10150 C
Problem Statement

• Solve R P T Q x = b
  • Upper or lower sparse triangular matrix T
  • Row scaling R
  • Permutations P, Q
  • Solution and RHS x, b
  • (Everything that is needed for LDL, LU, incomplete factorizations, etc.)
  • Efficient to absorb user data

• For many sequential RHS with
  • Same T or
  • Same nonzero pattern pat(T)

• On a multi/many-core node
Solution Approach

- Symbolic phase
  - Find parallelism in $\text{pat}(T)$, the graph of $T$

- Numeric phase
  - Load data structures with numbers

- Solve phase
Motivation: Level Scheduling

![Graph showing the relationship between Level Index and \(\log_{10} \text{#Rows}\)](image-url)
Motivation: Level Scheduling
Motivation: Hybrid

\begin{itemize}
  \item \texttt{log}_{10} Level Index
  \item \texttt{log}_{2} \#Rows
  \item Reorder
\end{itemize}
Motivation: Hybrid

Solve phase on Knights Corner
Elastic cube, bilinear hexes, 86490 unknowns, L from LDL, NodeND

Hybrid solver
Level scheduling only
Recursive blocking only

Reorder

Motivation:
Hybrid
Software: HTS

- Trilinos/packages/shylu/hts
- C++ and OpenMP
  - Templated on row pointer, column index, and scalar types
- CSR, CSC, forward, transpose, conjugate inputs
- Effective nonzero pattern reuse
- Will be an option in Ifpack2::LocalSparseTriangularSolver
  - Interface will support nonzero pattern reuse
Algorithms: Switching Method

- Want robustness to downward and upward spikes in $N_i$.
- Use levels 1 to $k$:

\[ n_{\text{good}} \sim 10, \quad f_{\text{bad}} \sim 1\% \]

$N_i \equiv \text{size of level set } i$

$C_i \equiv \sum_{j \leq i} N_j$

$C_{i}^{\text{bad}} \equiv \sum_{j \leq i \cap N_j < n_{\text{good}}} N_j$

$k \equiv \arg \max_{k} \quad N_k \geq n_{\text{good}} \cap C_{k}^{\text{bad}} \leq f_{\text{bad}} C_k$
Algorithms: Level Scheduling
Algorithms: Pruned Point-to-Point

Algorithms: Recursive Blocking

sparse or dense
parallel or serial

Algorithms:
Recursive
Blocking

parallel mvp
inverse
serial mvp
serial trisolve
serial trisolve
parallel mvp
inverse parallel mvp
Results: UMFPACK LU on IB and KNC

UMFPACK LU, Ivy Bridge, 20 threads
OMP_PROC_BIND=spread OMP_PLACES=cores

- Level scheduling
- Recursive blocking
- Hybrid

Solve phase speedup w.r.t. MKL trisolver

UMFPACK LU, Knights Corner, 240 threads
KMP_AFFINITY=compact

- Level scheduling
- Recursive blocking
- Hybrid

Solve phase speedup w.r.t. MKL trisolver

Straightforward reference serial trisolver speedup w.r.t. MKL trisolver
Results: UMFPACK LU on IB and KNC

UMFPACK LU, Ivy Bridge, OMP_PROC_BIND=spread OMP_PLACES=cores

Solve phase speedup w.r.t. MKL trisolver

(Numeric phase time) / (parallel solve time)

(Symbolic phase time) / (parallel solve time)

UMFPACK LU, Knights Corner, KMP_AFFINITY=compact

Solve phase speedup w.r.t. MKL trisolver

(Numeric phase time) / (serial solve time)

(Symbolic phase time) / (serial solve time)
Solve phase speedup w.r.t. MKL trisolver

UMFPACK LU, Ivy Bridge
20 threads, 822 UF matrices
OMP_PROC_BIND=spread OMP_PLACES=cores

Median for $\geq N$

UMFPACK LU, Knights Corner
240 threads, 824 UF matrices
KMP_AFFINITY=compact

Median for $\geq N$
Future Work

- Point-to-point level scheduling
  - Group rows into tasks to minimize dependencies
  - Size tasks to reflect level of synchronization

- Hybrid
  - Switching method(s)
  - Does not have to be 3 blocks; alternate

- HTS
  - Improve formatting of recursively blocked part to take further advantage of dense sub-blocks

- Direct sparse methods on GPU?