PARALLEL TRIANGULAR SOLVES: A TALE OF TWO ALGORITHMS

{ SHELBY LOCKHART AND SAMAH KARIM }

PROBLEM

Sparse triangular solves are difficult to parallelize due to their irregular storage structure and the sequential nature of backward and forward substitution algorithms, as seen below.

Sequential Forward Substitution

for $j = 0, \ldots n$ **do**
  $x_j = b_j / l_{jj}$
  for $i = j + 1$ to n **do**
    $b_i = b_i - l_{ij} x_j$
  end for
end for

PARALLEL DIRECT SOLVE

Attempts to parallelize triangular solvers come with a great communication cost, due to every process needing access to all components of the solution vector. This translates to a broadcast at every iteration as seen in the Row Fan-Out algorithm.[2]

Parallel 1-D Row Fan-Out Forward Substitution

for $j = 1 \ldots n$ **do**
  if $j \in myrows$ **then**
    $x_j = b_j / l_{jj}$
  end if
  Broadcast x_j
  for $i \in myrows, i > j$ **do**
    $b_i = b_i - l_{ij} x_j$
  end for
end for

With sparse matrices, using a primitive block-row partitioning of matrices will result in less computation, but the communication overhead will be the same as in the dense case.

APPROXIMATE SOLVE

When used in preconditioning, finding an exact solution to the triangular solve becomes less important. Hence an approximate iterative solver can be used to get a "good-enough" solution at every step of the preconditioned Krylov solver. One approach is the Jacobi method.[1]

Starting from an initial guess $x^{(0)}$, compute next iterate as follows:

$$x^{(k+1)} = (I - D^{-1}L)x^{(k)} + D^{-1}b$$

where D is the matrix consisting of the diagonal of L

Sequential Jacobi Method

Compute $z = D^{-1}b$, using row scaling
Compute $M = I - D^{-1}L$, by scaling rows of L and shifting the diagonal entries
for $k = 1, \ldots$ **do**
  SpMV to get $Mx^{(k)}$
  saxpy to get $x^{(k+1)} = Mx^{(k)} + z$
end for

PARALLEL APPROXIMATE SOLVE

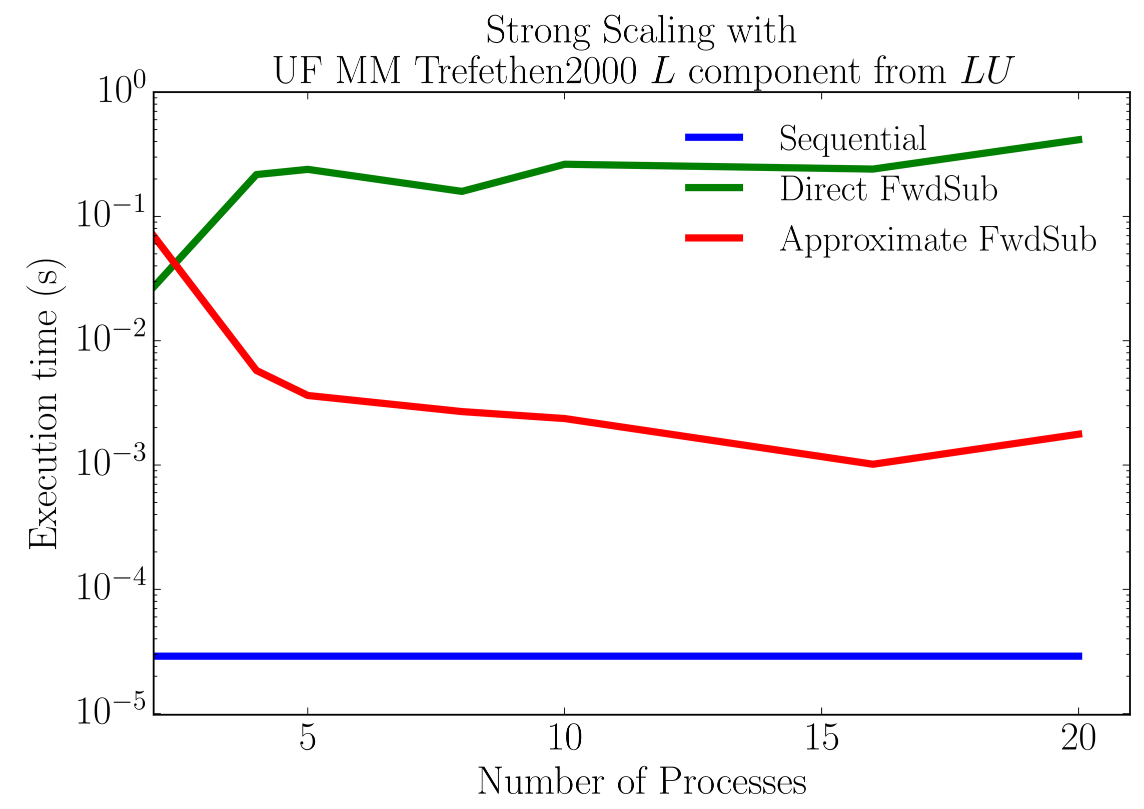
Jacobi method parallelizes well:

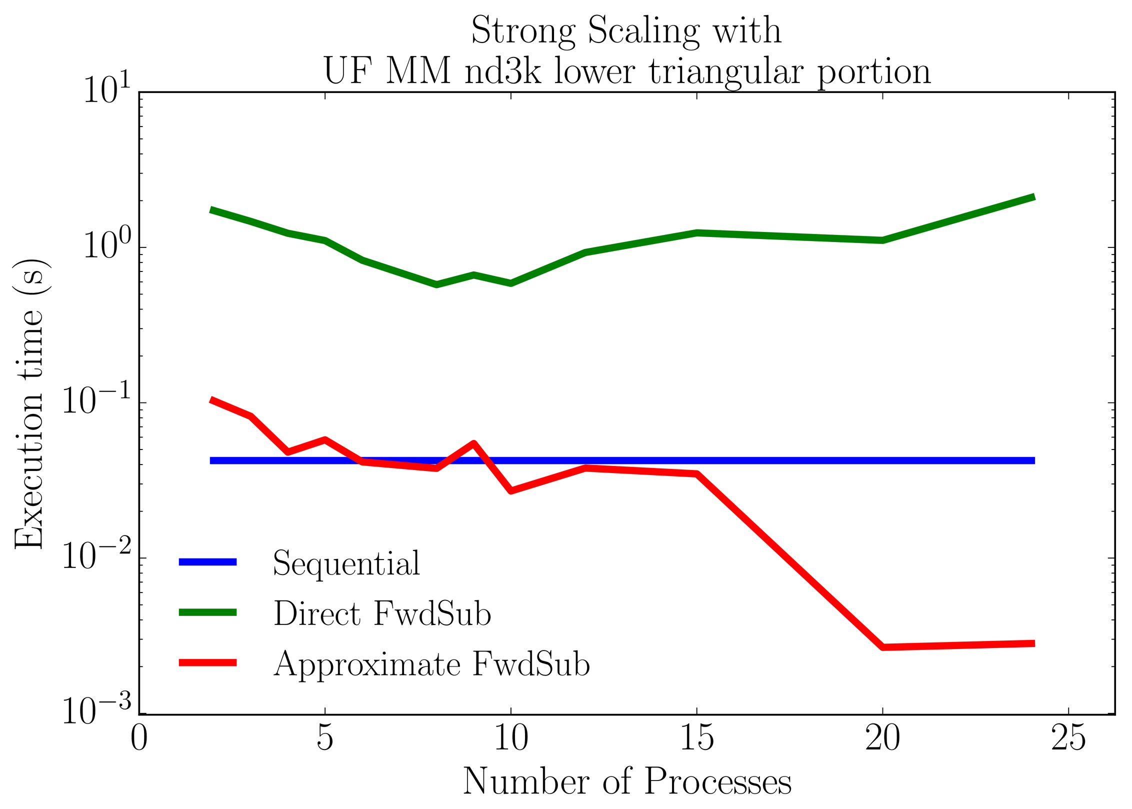
- All components of the current iterate $x^{(k+1)}$ depend only on components of the previous iterate $x^{(k)}$
- They can be updated simultaneously

All basic operations can be done in parallel:

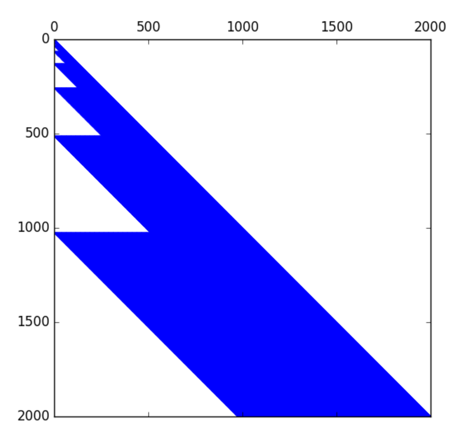
- Row scaling operation with depth $= 1$
- SpMV with depth $= \log(n)$
- saxpy with depth $= 1$

STRONG SCALING EXPERIMENTS



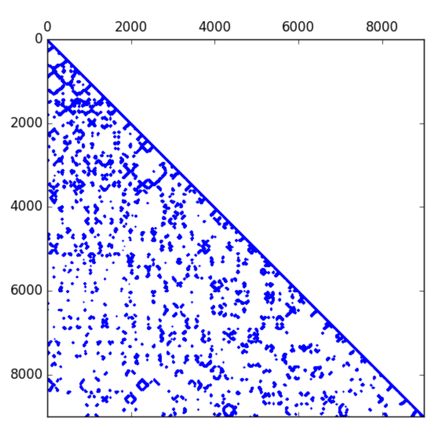


Trefethen2000[3]



1350949 nnz

nd3k[3]



164434 nnz

Experimental Parameters:

- UF Trefethen2000 L component from LU; $n = 2000$
- UF nd3k lower triangular part; $n = 9000$
- Each approximate solve was 5 Jacobi iterations

PARALLEL COST MODELS

- α: latency of sending a message
- β: bandwidth cost of sending 1 byte
- γ: cost of performing one flop
- b_c: cost of a broadcast on a given mesh
- a_c: cost of an all-gather on a given mesh
- nnz: non-zero entries in the matrix
- p: number of processes
- T_p: execution time using p processes

We can develop upper bound cost models for the dense and sparse cases for each parallel algorithm. For the direct solve, we will assume that the computation and communication is not overlapped.[2] The computation cost is given by:

$$n + 2\sum_{j=1}^{n-1} \left\lceil \frac{n-j}{p} \right\rceil$$

whereas the sparse approximate solve has a lower computation cost determined by the number of non-zeros.

Direct Solve:

$$T_p = (\alpha + \beta)(n-1)b_c + \gamma\Big(\frac{n^2 + 2np - 2n}{2p}\Big) \quad (1)$$

Dense Approximate Solve:

$$T_p = \alpha a_c + \beta n + \gamma\Big(\frac{n^2 + 6n}{2p}\Big) \quad (2)$$

Sparse Approximate Solve:

$$T_p = \alpha a_c + \beta n + \gamma\Big(\frac{3n + nnz}{p}\Big) \quad (3)$$

MACHINE SPECS

2x Broadwell-EP 12-core Xeon

256 GiB of 2133 DDR4 RAM

REFERENCES

[1] H. Anzt, E. Chow, and J. Dongarra Iterative Sparse Triangular Solves for Preconditioning Euro-Par 2015: Parallel Processing: 21st International Conference on Parallel and Distributed Computing pp. 650–661 (2015)

[2] M.T. Heath and C.H. Romine. Parallel Solution of Triangular Systems on Distributed-Memory Multiprocessors SIAM Journal of Scientific Computing Vol. 9 pp. 558–588 (1988)

[3] T. Davis and Y. Hu University of Florida Sparse Matrix Collection na-digest 92 (1994)