Performance Portable Supernode-based Sparse Triangular Solver for Manycore Architecture

Ichitaro Yamazaki, Sivasankaran Rajamanickam, and Nathan David Ellingwood

Sandia National Laboratories, Albuquerque, New Mexico, USA

International Conference on Parallel Processing (ICPP20)

Edmonton, Canada, August 20, 2020

Background

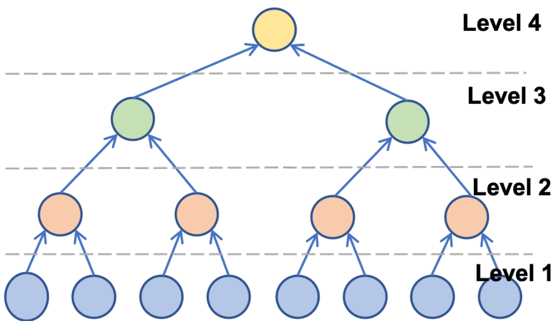
- important kernel in many applications, but challenging to parallelize
  - Sparsity structure may limit the parallel scalability
- focus on particular cases where each process uses sparse direct solve
  - SIERRA-Structural Dynamics (SIERRA-SD): distributed-memory domain-decomposition based linear solver that uses a **local** direct solver and applies SpTRSV $\backsim 10^4$ times for each factorization
  - Low Mach fluid simulation: multigrid preconditioner that uses **local** direct solver on a coarse grid and potentially as a smoother
- study two algorithmic variants
  - Supernode/block based level-set scheduling to exploits hierarchical parallelism
  - Partitioned inverse to transform SpTRSV into a sequence of SpMV

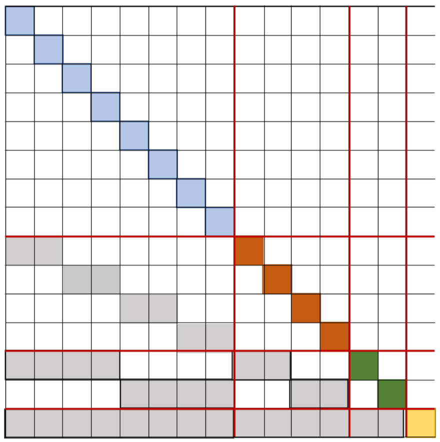
Triangular solve with level-set scheduling [Anderson & Saad'89]

- Dense triangular solve computes each solution element in sequence through backward/forward substitution

- For a sparse triangular matrix, multiple independent elements can be computed at each step

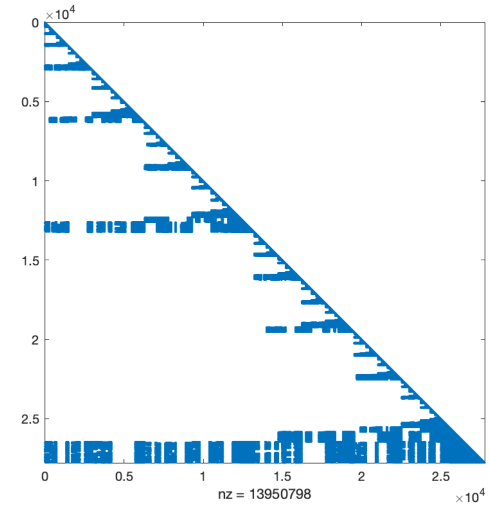
- Level-set scheduling finds a independent elements (e.g., using DAG), and computes these elements in parallel at each level

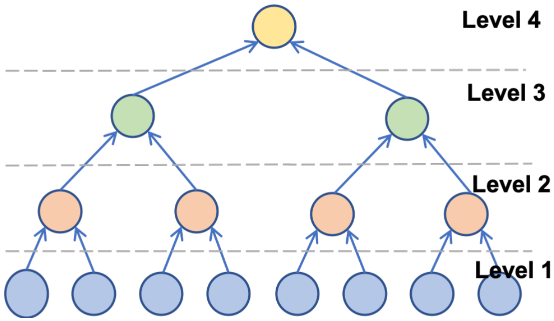




Supernode-based level-set scheduling

- Sparsity often limits the available parallelism
  - lots of levels with small number of tasks at each level (e.g., tri-diagonal matrix)
- We exploit the block structure in the matrix
  - direct factorization leads to triangular matrices with the block structure called supernodes
  - merge columns with a similar sparsity structure into a singe block column
  - these columns in a supernode leads to the chain
- We used supernode-based level-set scheduling
  - reduces the number of levels
  - batched kernels for hierarchical parallelism
    - all the leaf-supernodes in parallel
    - threaded kernels (e.g., BLAS/LAPACK) on each block column





Partitioned inverse with supernode-based level-set

Sandia National Laboratories

- Dense triangular solve with the diagonal block is fundamentally sequential (chain)
- Invert diagonal block to replace TRSM with GEMV for computing the solution blocks, and then use another GEMV to update the RHS
  - Use batched GEMV to update all solutions in parallel with a single kernel launch

```
for each level
  parallel-for each s in this level
    // compute sth solution
    x_s := L_{s,s}^{-1} x_s
    // use sth solution
    // to update child RHS
    for each non-empty block L_{i,s}
     x_i := x_i − L_{i,s} x_s
    end for
  end for
end for
```

update with single gemv with gather/scatter of x

(b) Push (col-major/left-look).

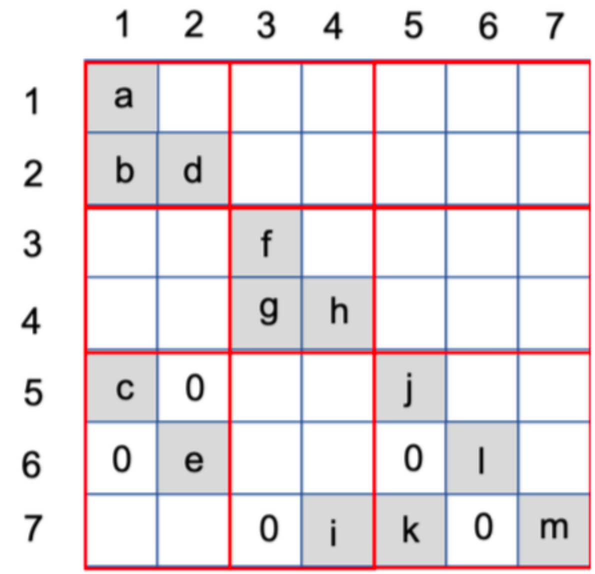
- Apply the inverse of the diagonal blocks to the corresponding off-diagonal blocks to merge these two batched GEMV calls into one
  - Partitioned inverse [Alvardo, Pothen, Schreiber,93] based on level-set partition of supernodes
  - It transforms SpTrsv into a sequence of SpMVs

$$L^{-1} = \prod_{\ell=1}^{n_\ell} L_\ell^{-1},$$

  - Instead of batched GEMVs, we can use a single SpMV call
    - no operation with explicit zeros, but lose block structure

Implementation

- Kokkos & Kokkos-kernels
  - Portable to different manycore architectures
  - Some more details in the paper
- Data structure
  - CSR/CSC, with explicit zeros to form supernodal blocks for dense operations, e.g., TRSM+GEMV
- Interfaced with `SuperLU` & `Cholmod` packages



colptr: 1 5 8 11 13 16 18 19

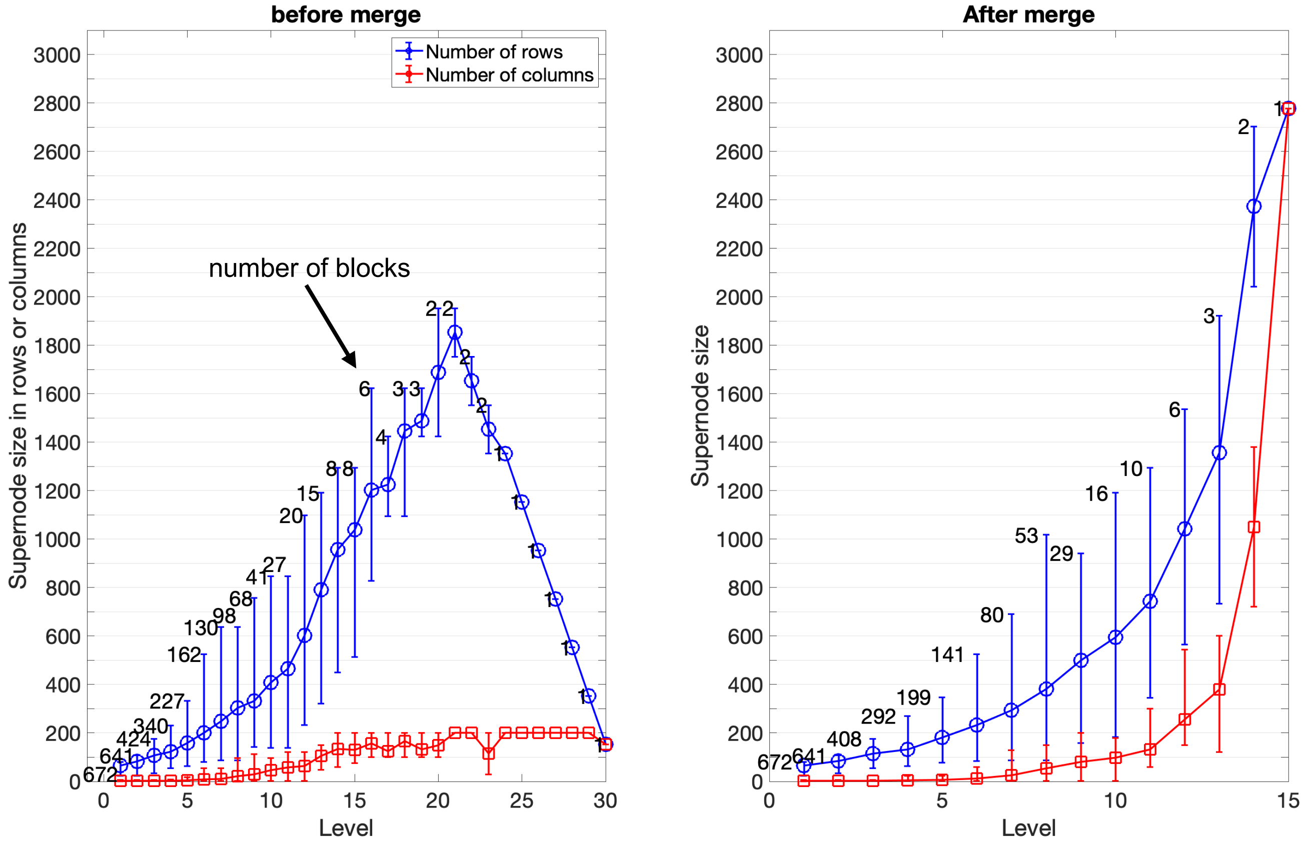
values: a b c 0 d 0 e f g 0 h i j 0 k l 0 m

rowind: 1 2 5 6 2 5 6 3 4 7 4 7 5 6 7 6 7 7

Experiment setups

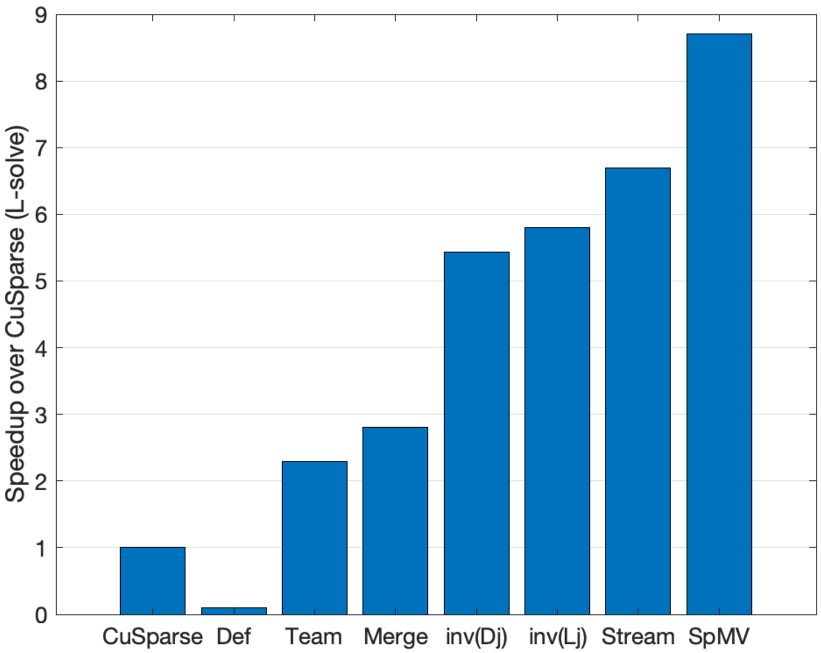
- `SuperLU` to factor the matrix with METIS ordering
- Performance on an NVIDIA V100 and P100 GPU
  - gcc compiler version 6.40 or 5.40 and nvcc 10.1 or 10.0
- Performance comparison with NVIDIA's CuSPARSE, `cusparseDcsrsv2_solve`
  - Use level-set scheduling `cusparseDcsrsv2_analysis` with `CUSPARSE_SOLVE_POLICY_USE_LEVEL`

SIERRA-SD matrix (n=27k)



- Lots of small blocks in the beginning and a fewer larger blocks at the end
- Merging block columns with the same sparsity pattern reduce the number of levels and increase the compute intensity per level

Performance results with SIERRA-SD on V100



	symbolic	compute	L-solve CSC	U-solve CSR	U-solve CSC	fill-ratio
CuSparse	0.0487	0.2587	0.0087	0.0167	0.0185	13.7
Default	0.3000	0.2712	0.0888	0.0918	0.1343	28.9
Team	0.2921	0.3067	0.0038	0.0111	0.0051	28.9
Merge	0.5611	0.6444	0.0031	0.0083	0.0037	35.4
InvertDiag	0.5575	1.2576	0.0016	0.0076	0.0024	35.4
InvertOff	0.7067	7.2798	0.0015	–	0.0023	35.4
Stream(5)	0.7063	7.3001	0.0013	–	0.0020	35.4

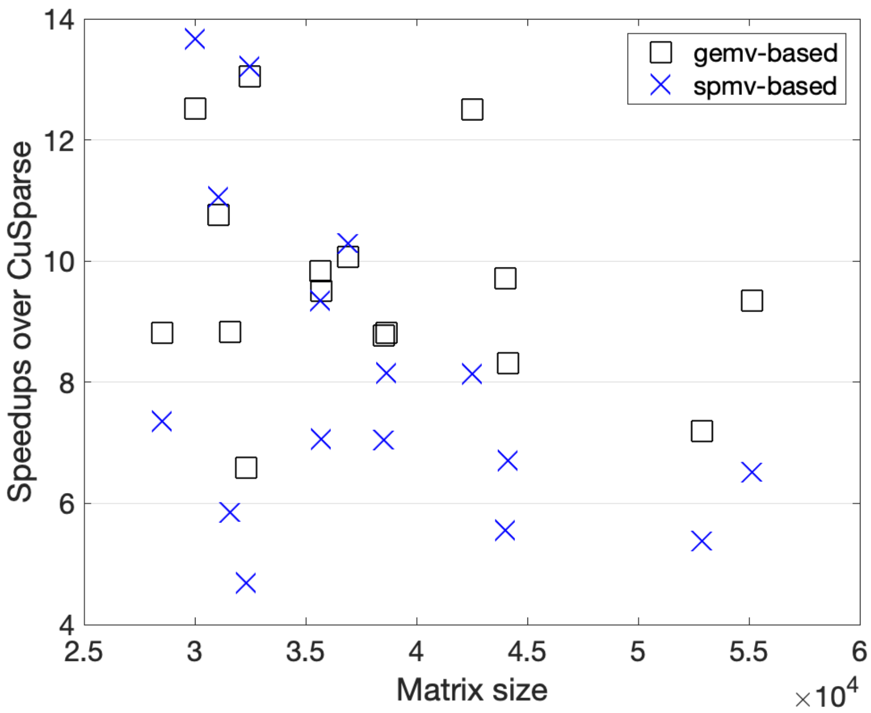
(a) batched `gemv` based implementation.

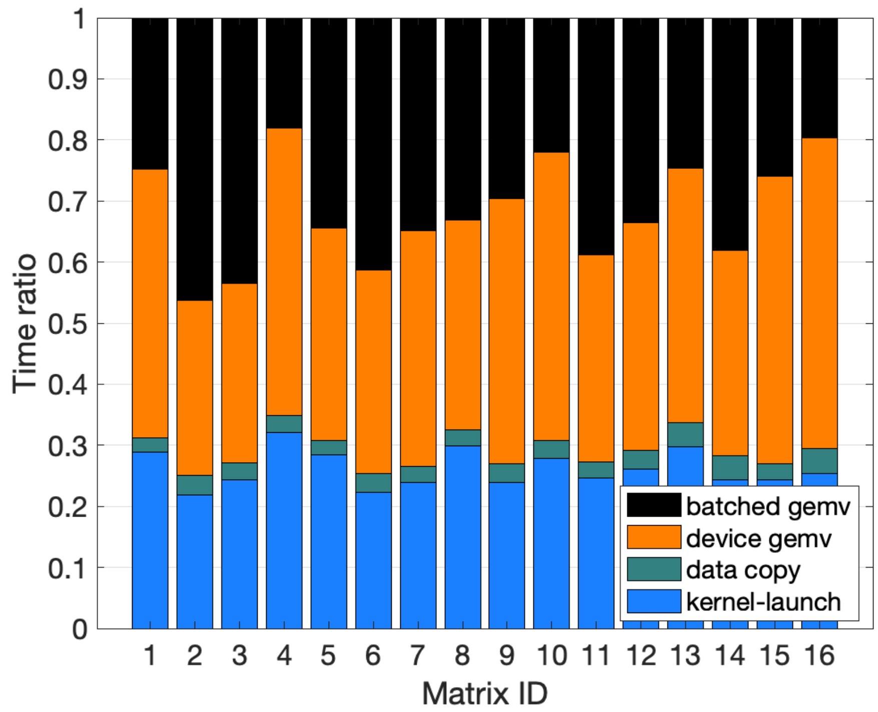
	symbolic	compute	L-solve	U-solve	fill-ratio
InvertDiag	0.6939	1.6820	0.0021	0.0018	14.7
InvertOff	0.6912	7.8646	0.0010	0.0012	15.5

(b) `spmv` based implementation.

- Default uses a standard device-level kernel (e.g., CuBLAS) on each block
- Speedups using team-level or batched kernels
- Further speedup with inversion (up to 8.7x)
- Same solution accuracy using all the approaches

Performance results with SIERRA-SD on P100

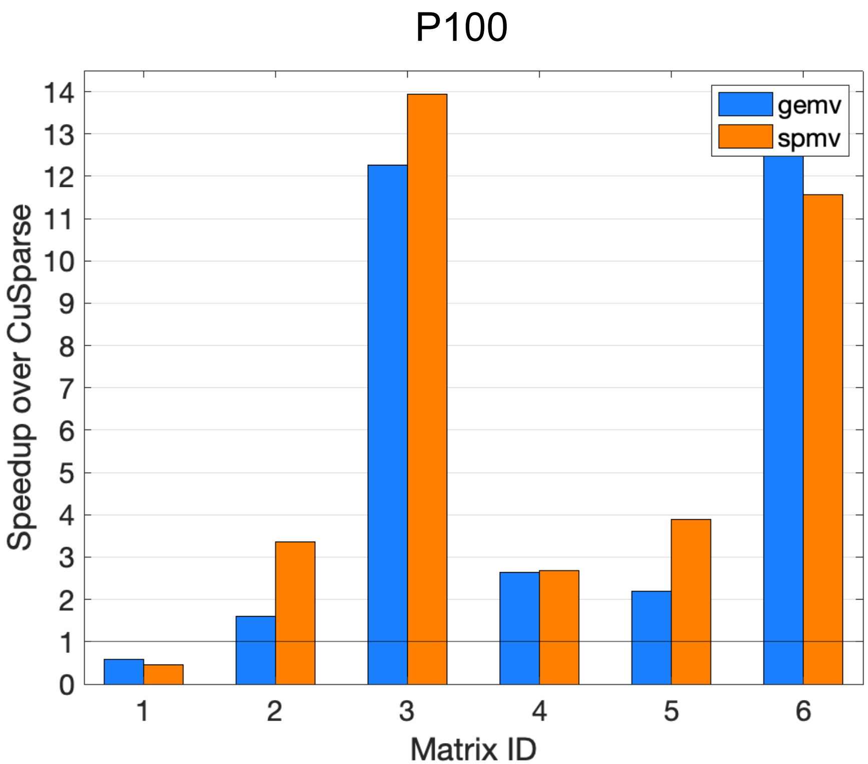


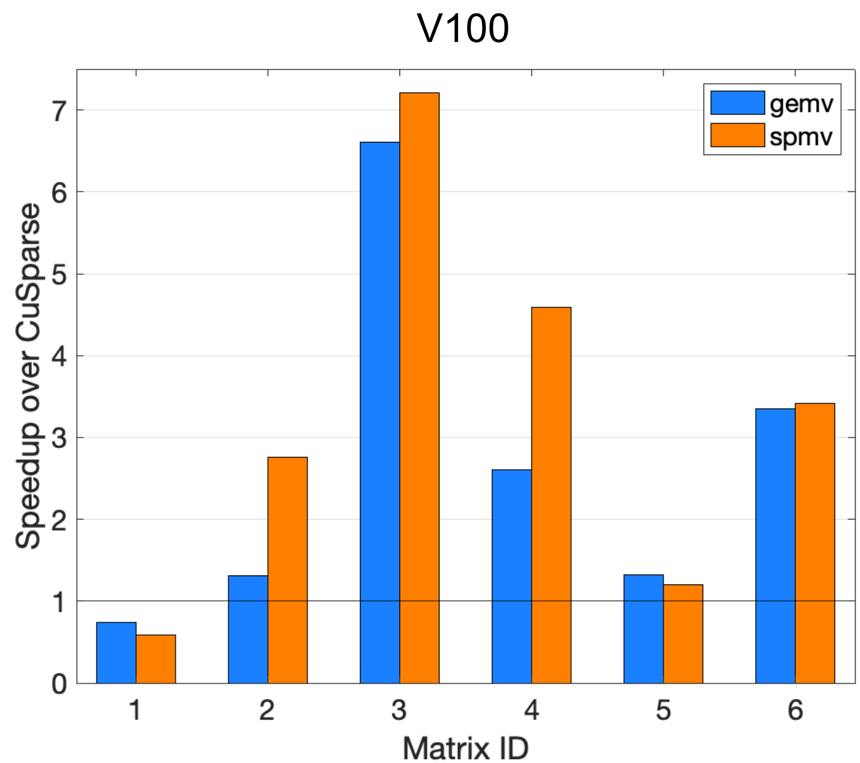


- Varying, but significant, speedups for different sizes of matrices
- Kernel-launch time can become significant

Performance results with SuiteSparse matrices

Sandia National Laboratories





id	name	type	n	$\frac{nnz}{n}$	n_ℓ	error
1	ACTIVSg70K	power system grid	69,999	12.6	83	0.003
2	dawson5	structural problem	51,537	770.4	1277	3.512
3	qa8fk	acoustic problem	66,127	653.3	22	0.006
4	FEM3Dtherm	thermal problem	17,880	324.6	15	0.008
5	thermal1	thermal problem	82,654	58.7	27	0.002
6	apache1	3D finite difference	80,800	240.2	25	0.002
7	apache2	3D finite difference	715,176	53.6	32	0.001
8	helm2d03	2D problem	392,257	14.9	109	0.018

- Performance depends on
  number of levels and sizes of supernodes

Final remarks

- SpTRSV is an important kernel in many applications, but a challenge to parallelize
- We studied two algorithmic variants where sparse direct factorization is used
  - Supernode/block based SpTRSV exploits hierarchical parallelism
  - Partitioned inverse transforms SpTRSV into a sequence of SpMV
- We implemented using Kokkos and Kokkos-kernels
  - Portable to different manycore architectures
    - Some performance results on CPUs in the paper
- We show performance results with SIERRA-SD (C. Dohrmann)
  - Up to 8.3x speedup over CuSPARSE on V100, and 17.5x using partitioned inverse
- Further extensions
  - Performance improvements (reducing setup time, improving kernel performance, reducing kernel launch costs)
  - Interface with other packages including ILU
- It is available from Kokkos-kernels and Trilinos packages
  - https://github.com/kokkos/kokkos-kernels
  - https://github.com/trilinos/Trilinos