Linear Solver Improvements in the ComPASS4 Project

LBNL Solver Team: Pieter Ghysels, Mathias Jacquelin, Esmond Ng
ComPASS4 PI's: James Amundson, Weiming An, Ann Almgren, Warren Mori, Esmond Ng and Stefan Wild

BERKELEY LAB · UCLA · Fermilab · Argonne NATIONAL LABORATORY

Linear Solvers in ComPASS4: Advanced Poisson Solvers for Beam Dynamics

A detailed understanding of the dynamics of lost particles and the subsequent energy deposition in the material constituting the accelerator and its surroundings is crucial to enabling higher-intensity accelerators. This requires accurate models of the fields in the vicinity of the boundaries (apertures). The FFT solvers available in Synergia are fast, but hard to adapt to complex boundaries.
New solvers need to be fast enough to be applied $\mathcal{O}(10^5) - \mathcal{O}(10^8)$ times in a large-scale simulation, while being optimized for the relatively small number of degrees of freedom necessary to simulate smooth beam distributions.

- Direct solvers can amortize setup/factorization over many consecutive solves
- Direct sparse solvers are highly efficient on small-to-medium sized problems, and efficiently use modern hardware through BLAS3/2 calls
- Iterative solvers converge quickly with a good initial guess, from a previous solve
- Krylov type solvers can reuse or recycle basis information to accelerate subsequent solves

SymPACK Triangular Solve Improvements

- Applying Gaussian elimination to a sparse matrix destroys some of the zero entries
- Producing **fill** entries
- **Ordering problem**: finding "good" row and column permutations to reduce fill
- Finding the best ordering is NP-complete $\Rightarrow$ rely on heuristic algorithms:
  - Multiple Minimum Degree, Approximate Minimum Degree
  - Nested-dissection ((PAR)Metis and (PT)Scotch graph partitioners)
- Larger blocks important for performance (manycore, GPUs)
- Sparse matrix computations suffer low computational intensity
- Heuristic to form larger blocks:
  - Heuristic based on TSP [Pichon, Faverge, Ramet, Roman]
  - **Faster heuristic, preserving quality**

Refined ordering in supernodes to improve block structure and data locality

The heuristic

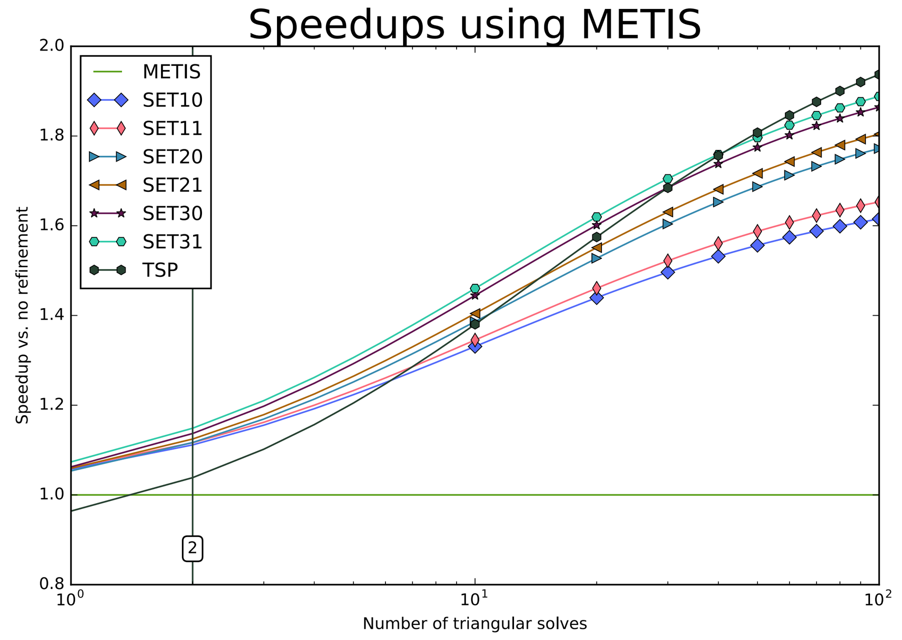
- After symbolic factorization, supernodes are assigned **labels**
- Maintain an ordered partition of the vertex set (initially the supernode set)
- Refine the partition during successive steps:
  - Process the adjacency sets $adj(S)$ in reverse order of labels
  - Partition each set S into two sets:

$$S' := S \cap adj(S) \qquad S'' := S \setminus adj(S).$$

- If S' and S'' are non-empty, replace S by S' and S''
- Six variants of the heuristic:
  - "Natural" order: supernodes are processed by decreasing order of label
  - Maximal Cardinality Search (MCS) order: supernodes are processed by decreasing order of cardinality
  - Largest subtree first order: supernodes are processed by decreasing size of subtrees
  - For each processing order, either place S' before S'', or alternate

Experimental evaluation

Speedups using METIS



- Set of 58 matrices from Florida Sparse Matrix Collection
- Experiments using **symPACK** sparse solver performed on NERSC Cori
- Refinement when doing multiple solves using METIS
- Single 68 KNL core node
- Heuristic beneficial when doing more than **two** solves

Spectral Partitioning and Fill-Reducing Nested Dissection Reordering

Nested dissection is a heuristic to reorder a sparse matrix to reduce fill (memory usage and flops) in sparse factorization based solvers. Traditional implementations of nested dissection do not scale well, and quality degrades with increasing concurrency. Hence, this matrix reordering is becoming a major bottleneck in sparse direct solvers (SuperLU, STRUMPACK, SymPACK, MUMPS, ...) and in preconditioners based on sparse factorization like ILU.
We revisit the idea of using spectral partitioning to find a vertex separator in the nested dissection algorithm. Spectral partitioning can be implemented efficiently on current parallel and heterogeneous architectures, including GPU's.

- The Graph Laplacian for a graph $G(V,E)$

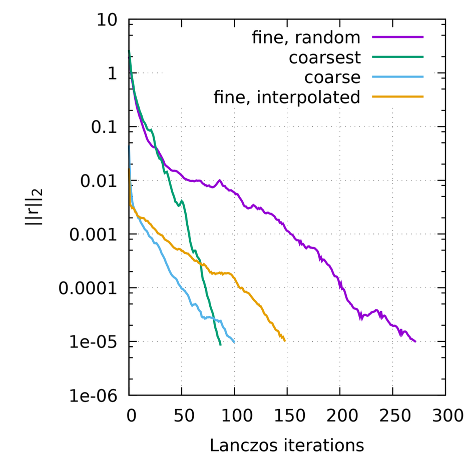
$$L(G) = \text{diag}(\text{degree}(v_i)) - \text{adj}(G)$$

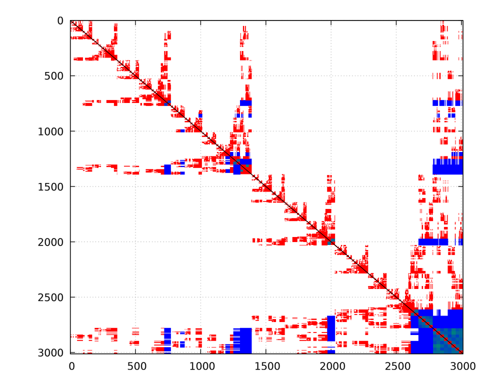
- Fiedler vector: smooth eigenvector of $L(G)$
- Partition along contour of Fiedler vector, by minimizing

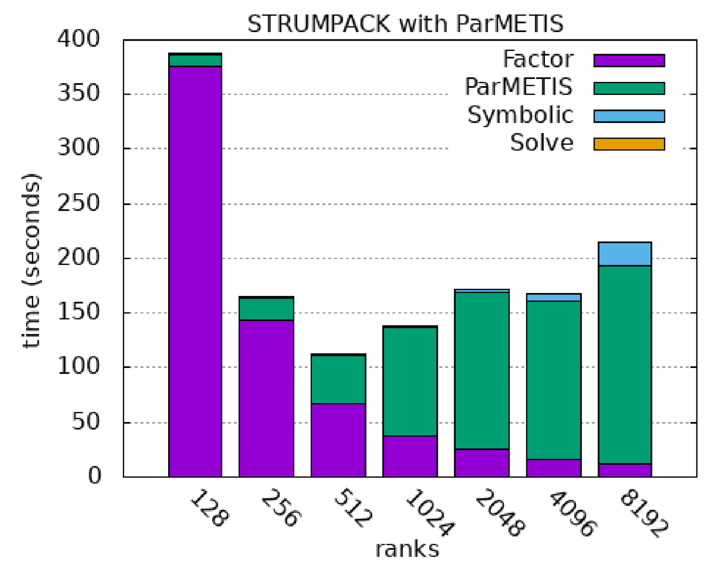
$$Q(V_1, V_2) = \frac{\text{cut}(V_1, V_2)}{\min(\text{Vol}(V_1), \text{Vol}(V_2))}$$

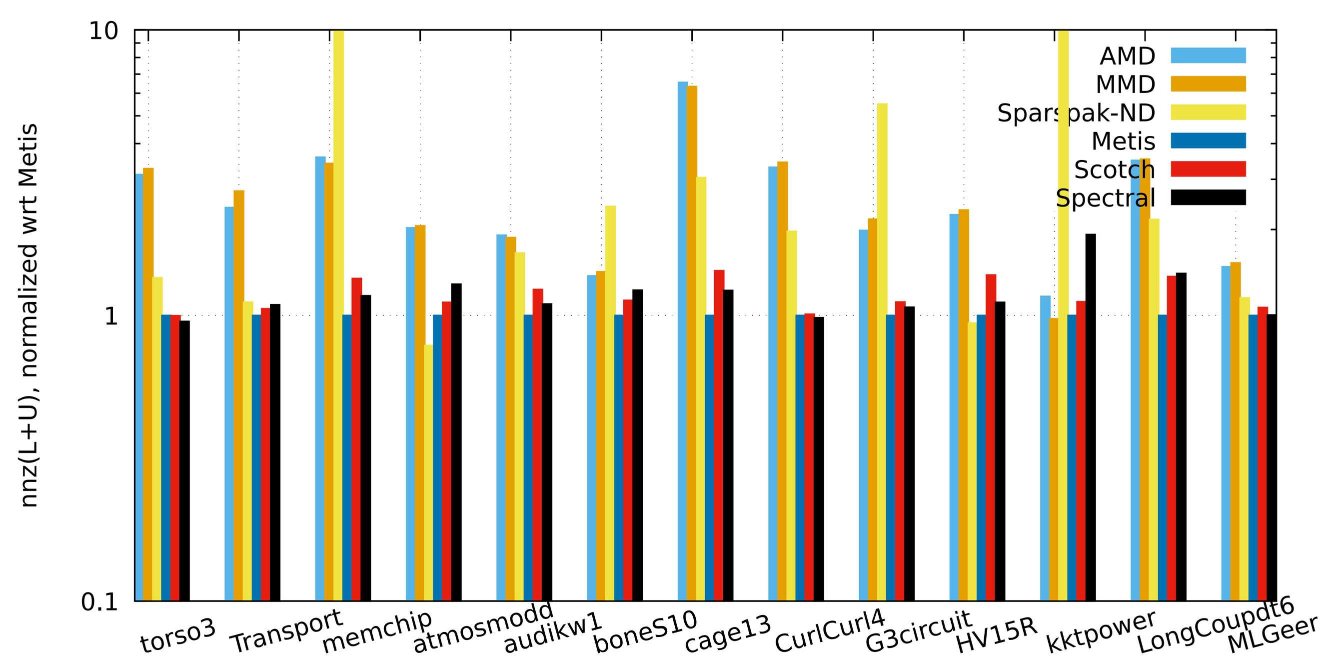
- Edge separator to vertex separator S
- Order V_1, V_2, S, recursion on V_1 and V_2

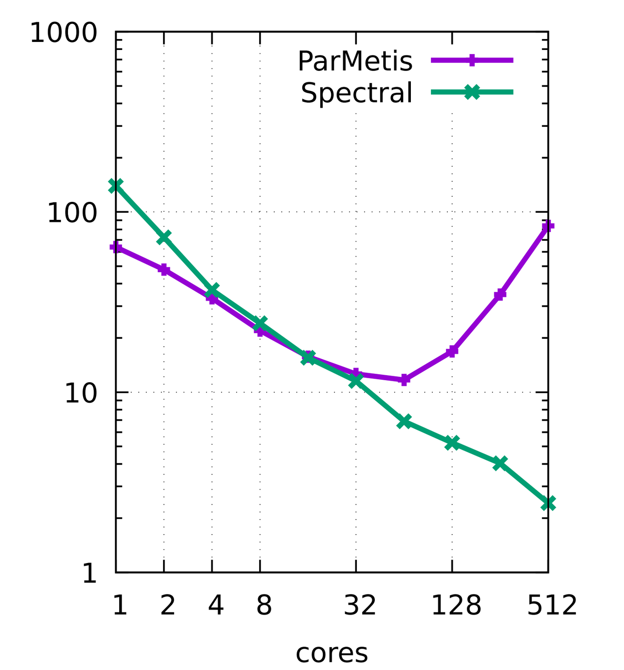
- Efficient parallel multilevel Lanczos solver
- OpenMP+MPI, GPU acceleration is WIP

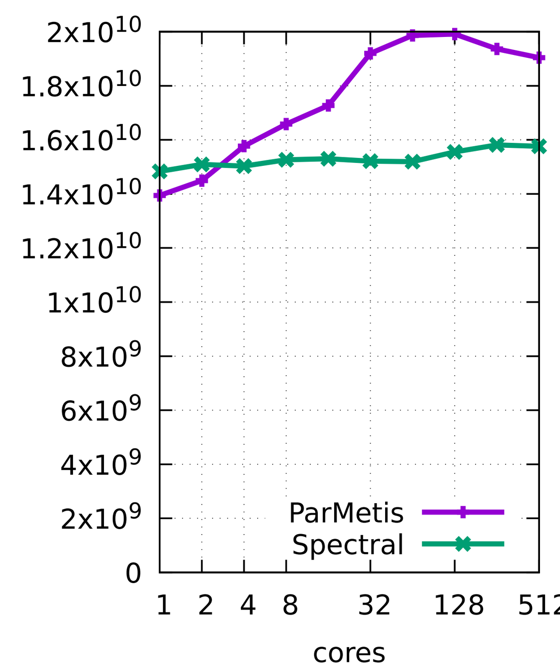








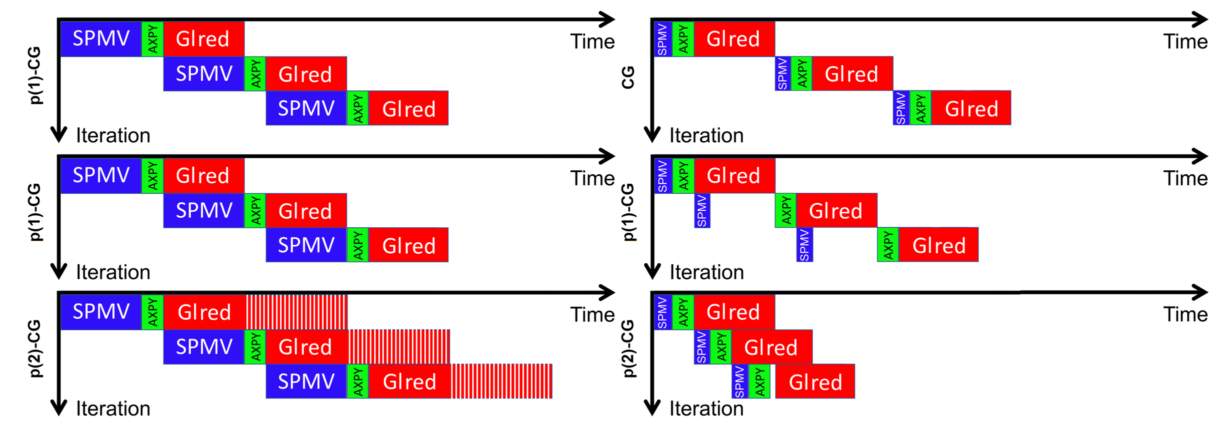


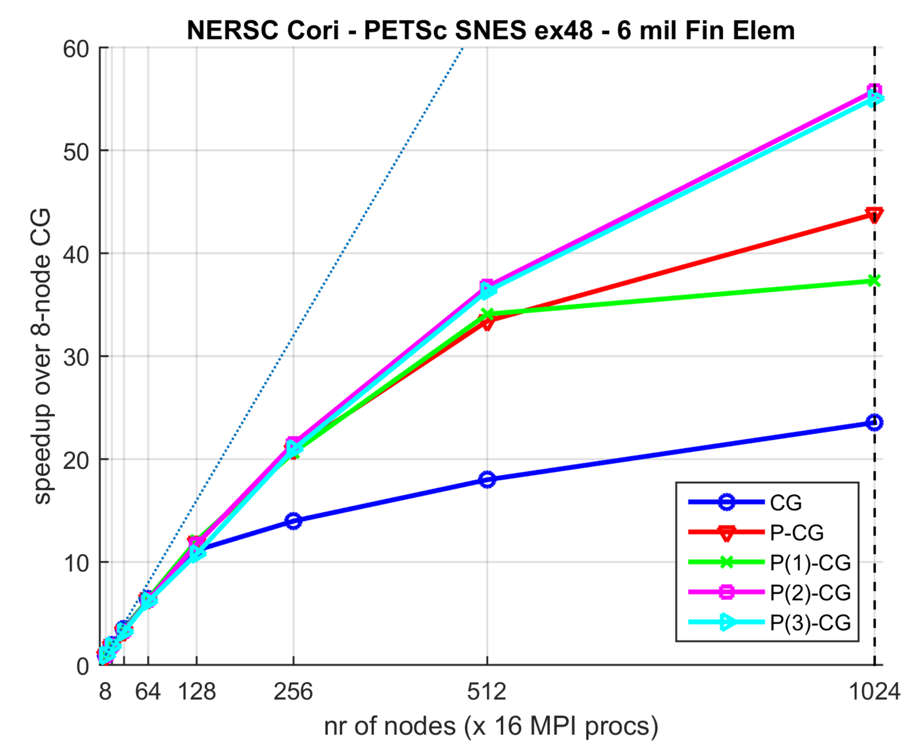


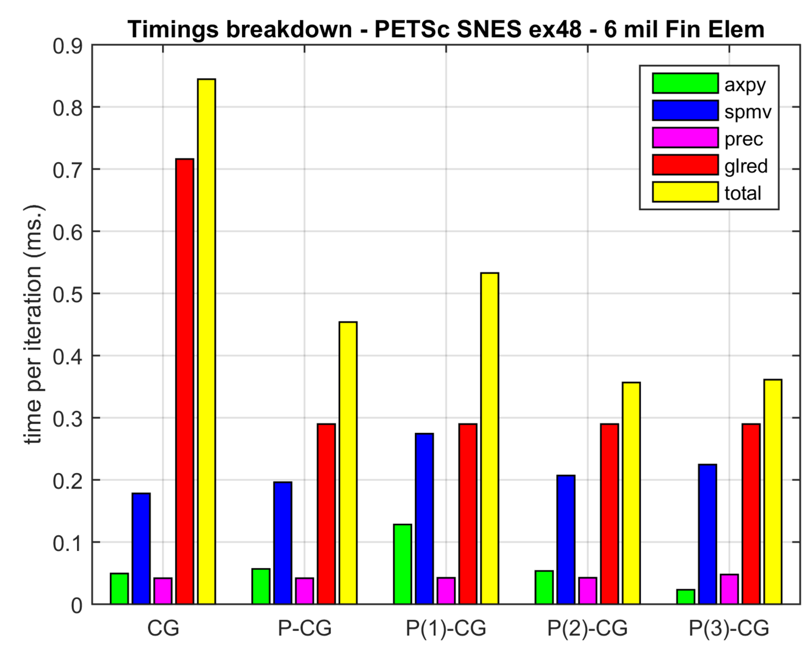
Pipelined(ℓ) Preconditioned Conjugate Gradients (With S. Cools, Universiteit Antwerpen)

Performance results comparing CG and communication hiding or "pipelined" variants, designed for improved strong scaling by overlapping the global reduction phase (MPI_Iallreduce) with computation (SpMV, AXPY, ...), and other communication.

- Pipelining ℓ steps can lead to $\mathcal{O}(\ell)$ speedup
- Numerically stable recurrences, high final accuracy [Cools ea., arxiv.org/abs/1902.03100]
- Preconditioned pipelined CG and GMRES in PETSc [Ghysels ea., SISC, 35(1), C48–C71.]







- 3D Hydrostatic Ice Sheet Flow, PETSc SNES ex48
- The Blatter/Pattyn equations are discretized using $200 \times 200 \times 150$ finite elements
- A Newton-Krylov outer-inner iteration
- Block Jacobi preconditioner (one block per rank; approx inverted using ILU)
- `-ksp_type pipelcg -ksp_pipelcg_pipel <l> -ksp_pipelcg_lmin 0.0 -ksp_pipelcg_lmax 2.0`
- Chebyshev iso monomial basis, for stability