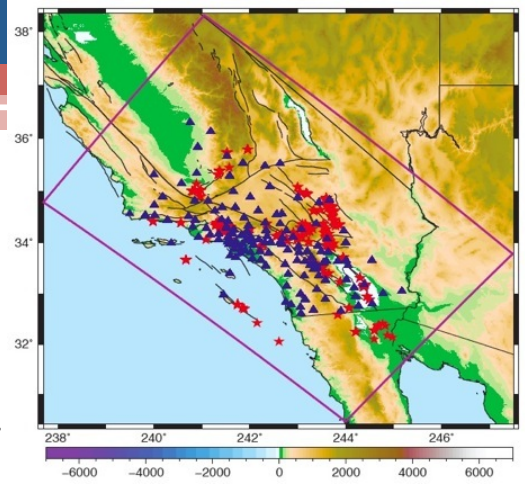
A Scalable Parallel LSQR Algorithm for Solving Large-Scale Linear System for Seismic Tomography

He Huang, Liqiang Wang, Po Chen(University of Wyoming)
John Dennis (NCAR)

UNIVERSITY OF WYOMING

LSQR in Seismic Tomography

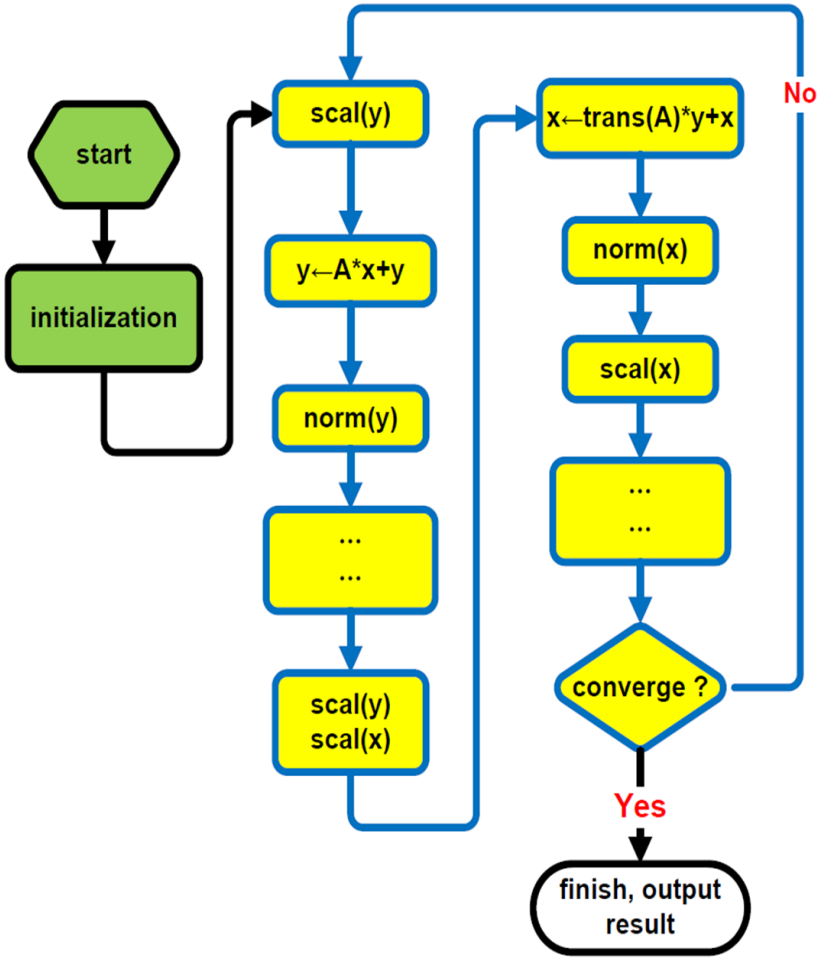


- Seismic Tomography: image sub-surface of [illegible] structures using seismic waves.
- **LSQR** (Least Squares with QR factorization):

  a numerical method for solving sparse linear equations in an iterative way.
  - Widely used in seismic tomography.
  - Features:
    - Highly efficient and capable of solving different types of linear systems for large linear inversion problems.
    - The estimated solution usually converges fast.

LSQR

- Solves linear systems, $\mathbf{A}\times\mathbf{x} = \mathbf{b}$, where **A** is a huge sparse matrix, **x** is the solution variables, and **b** is a right hand side vector (constant).
- Requires loading entire matrix into memory.
- The iterative steps of LSQR include several basic linear algebra operations, e.g. scale and norm and two matrix vector multiplications i.e., $\mathbf{y} \leftarrow \mathbf{y} + \mathbf{A}\times\mathbf{x}$ and $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{A}^{\mathbf{T}}\times\mathbf{y}$, where **y** is initially derived from vector **b**.



Real-World Application Challenges

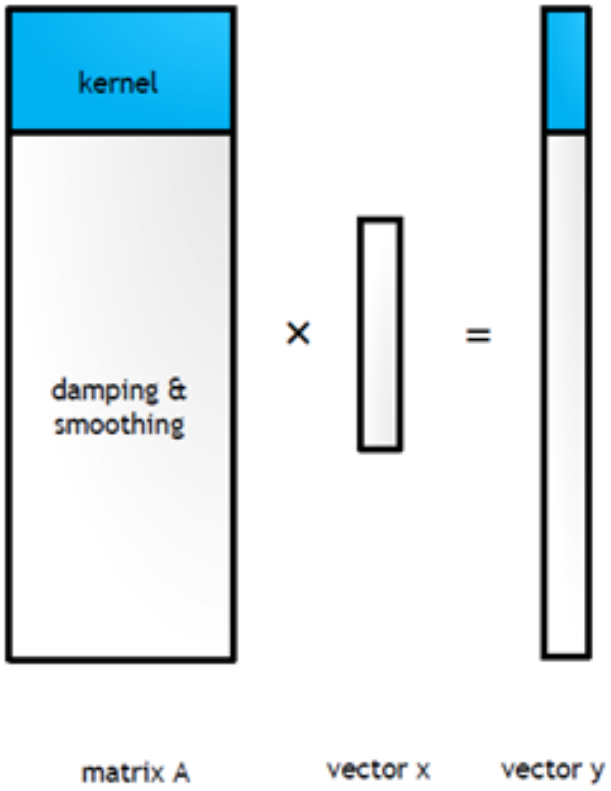
- Memory-intensive
  - LSQR requires load the entire dataset into the memory. But the matrix can be very huge and very sparse. The dataset could be much larger than the above dataset depending on the geological region of interest to calculate.
- Compute-intensive
  - Thousands or even more of iterations. Every iteration is compute-intensive.
- Communication-intensive
  - Massive communication between compute nodes.

General Idea

- Propose a **partitioning strategy** that is based on the special structure of the matrix. Specially, SPLSQR (Scalable Parallel LSQR) contains a novel data decomposition strategy that treats different components of the matrix separately.
- SPLSQR algorithm was optimized with **scalable communication** volume between a fixed and modest number of communication neighbors.

Matrix Layout

- In structure seismology, the coefficient matrix (A) is composed of kernel (top) and damping (bottom) component.
- Kernel: <1% rows; > 90% of nonzeros; sparse but relatively dense compared with damping, unstructured.
- Damping: >99% rows; < 10% of nonzeros; extremely sparse, structured.



Major Steps of SPLSQR Algorithm Based on MPI Programming Model

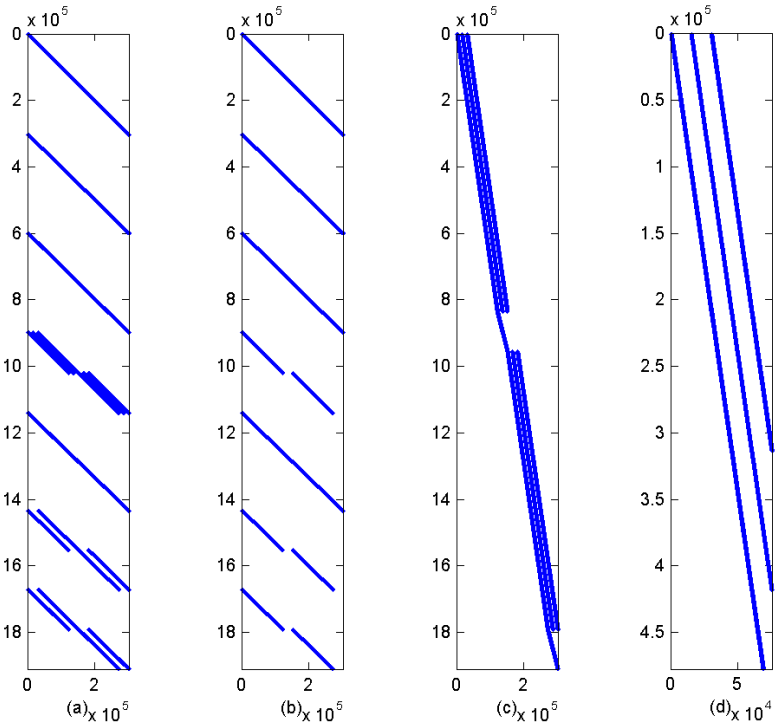
(1) Reorder damping component to minimize bandwidth.
(2) Decomposition
- partition kernel across columns
- Partition damping across rows
- Partition damping transpose across columns

(3) Iterative steps
- Step-1
  - Multiply kernel with vector.
  - Global sum partial result vector.
  - Communicate with neighbors.
  - Multiply damping with vector.
- Step-2
  - Multiple kernel transpose with vector
  - Communicate with neighbor.
  - Multiple damping transpose with vector.
- Step-3
  - Sum result vector.

(4) Test convergence: if false go to (3), else exit and output final result.

Matrix Reordering

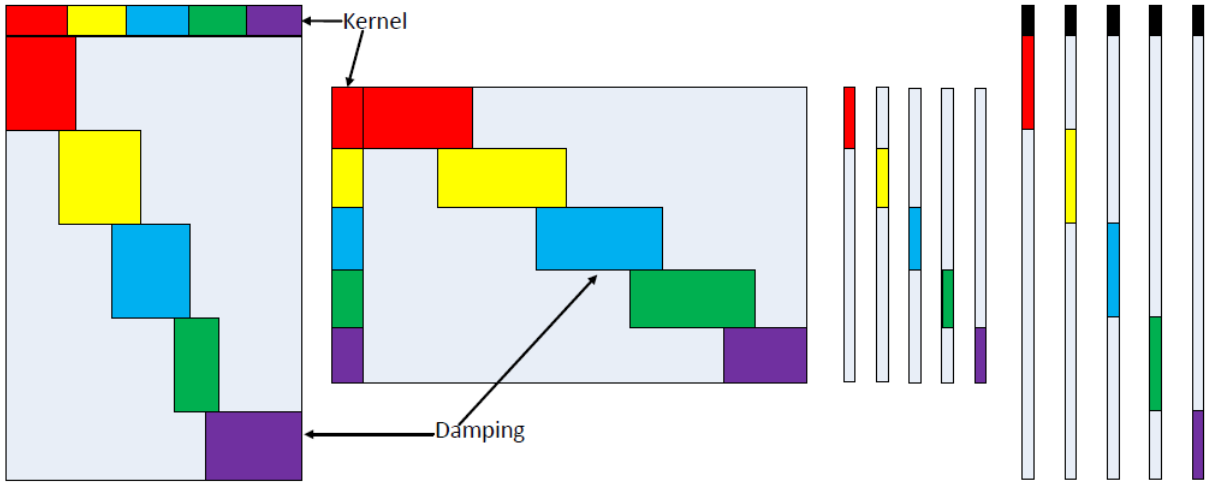
- The original damping submatrix with big bandwidth results in huge communication volume because there are large overlaps between MPI tasks after decomposition.
- Goal: move nonzeros of damping to diagonal area to minimized bandwidth, and thus to reduce overlap in later parallel computation, and reduce communication eventually.
- Perform row permutations such that the position of leading nonzero of each row is in descending order.
- Result: several thin bands in the diagonal area. This structure can reduce communication.



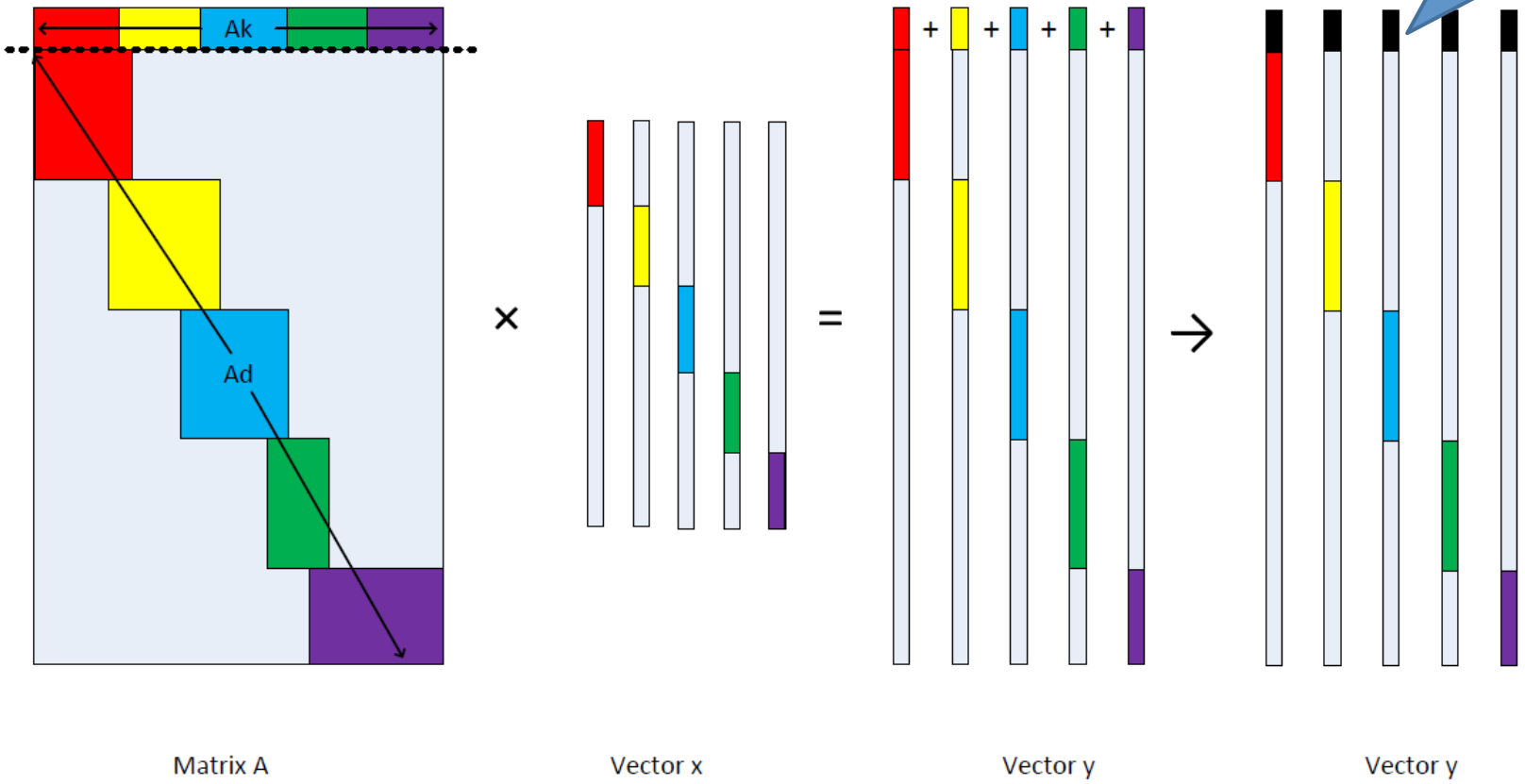
- (a) original damping
- (b) leading nonzero of (a)
- (c) result of applying row permutation for (a)
- (d) enlargement of top of part (c), clearly show several thin bands.

Data Decomposition

- Different colors represent different MPI tasks.
- The matrix has a kernel component (top) and a damping component (bottom).
- In memory, store a single copy of the kernel component in compressed sparse column (CSC), and store two copies of the damping component of the matrix, i.e., one copy of the original in compressed sparse row (CSR) and one copy of the transposed matrix in CSC, .
- Kernel component of the matrix is partitioned by columns, while the damping component of the matrix is partitioned by rows. The transposed damping matrix uses the same partitioning as the kernel component.



Parallel computation: y←y +A×x

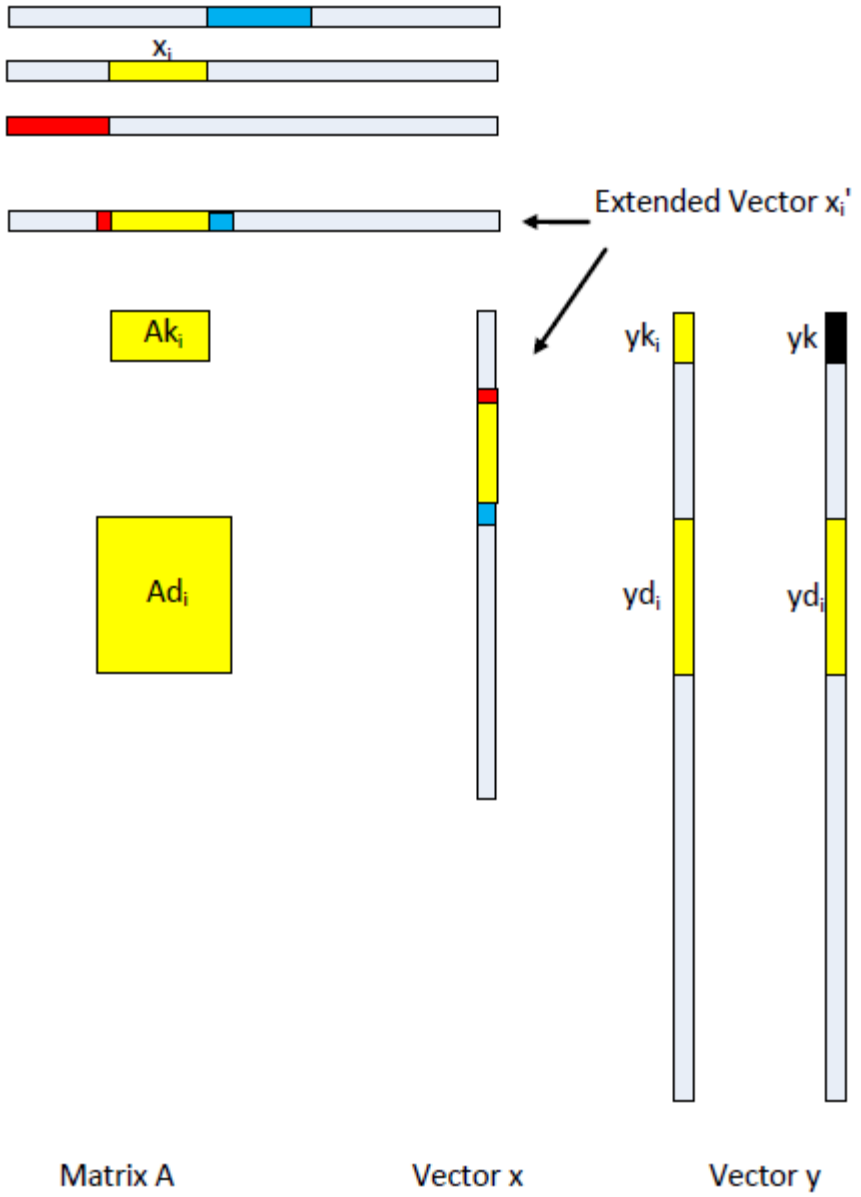


1. Each task multiplies its local piece of kernel Ak_i with local piece of x_i and yields its kernel part of vector, yk_i .
2. A reduction across all tasks is performed on yk_i to combine the partial results (black).

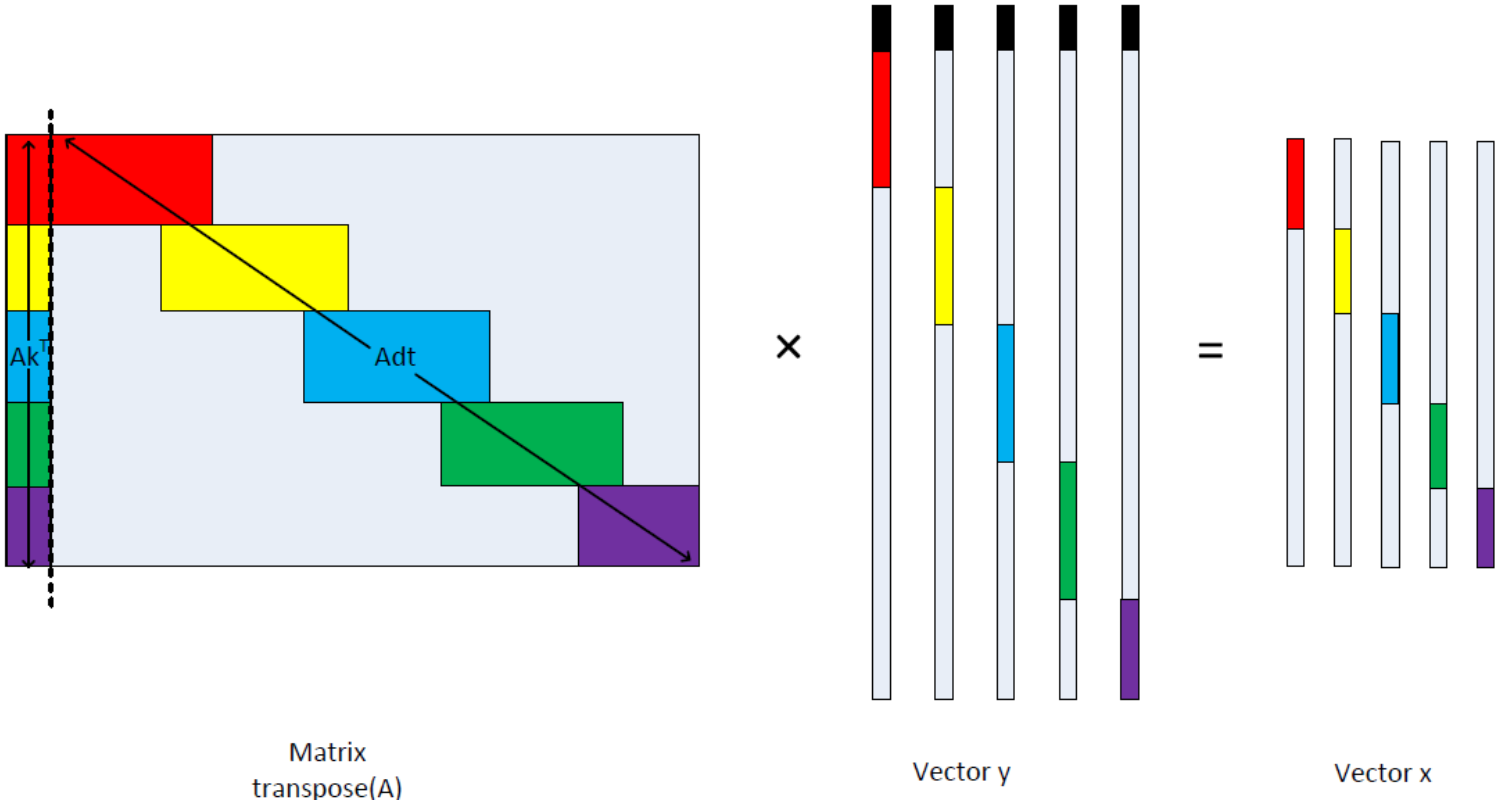
Parallel computation: y←y +A×x

3. Reconstruct extended x'_i from neighbors. E.g., yellow task communicates red and blue tasks.

4. The extended x'_i is multiplied with local damping Ad_i and yields local vector yd_i.

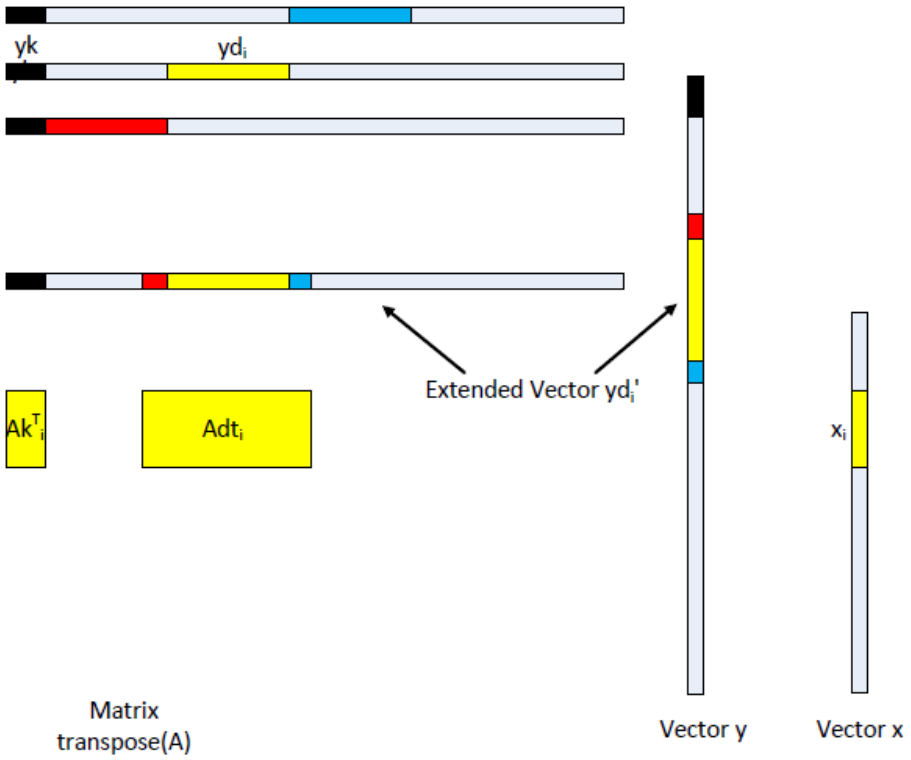


Parallel computation: $x \leftarrow x + A^T \times y$



1. Each task multiplies its local Ak^T_i with yk to construct xk_i.

Parallel computation: $x \leftarrow x + A^T \times y$



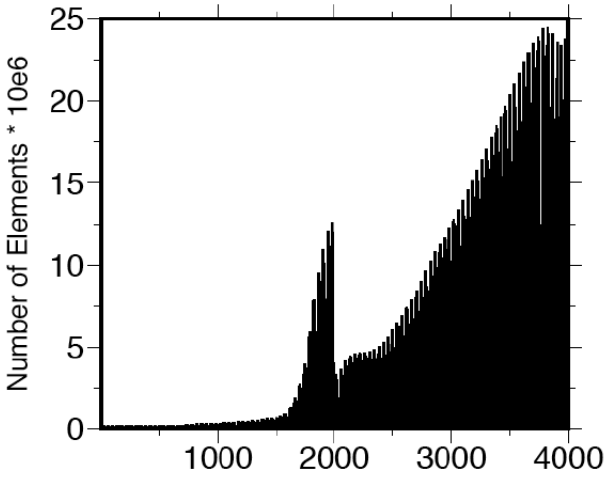
2. Reconstruct extended yd'_i from neighbors. E.g., yellow task communicates red and blue tasks.
3. The extended yd'_i is multiplied with local damping Adt_i and yields local vector xd_i.
4. $x_i = xk_i + xd_i$.

Communication

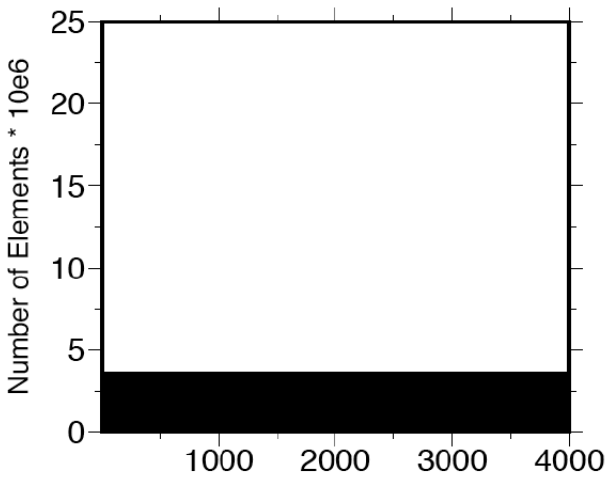
- MPI_Allreduce on kernel part of vector y, vector length = number of row in kernel which is trivial compared with the whole matrix.
- Local task compares local vector with required extended vector and decides which neighbors it needs to communicate.
- Communication is scalable:
  - larger core -> less overlaps with neighbors -> less communication volume.

Optimization: Load Balance

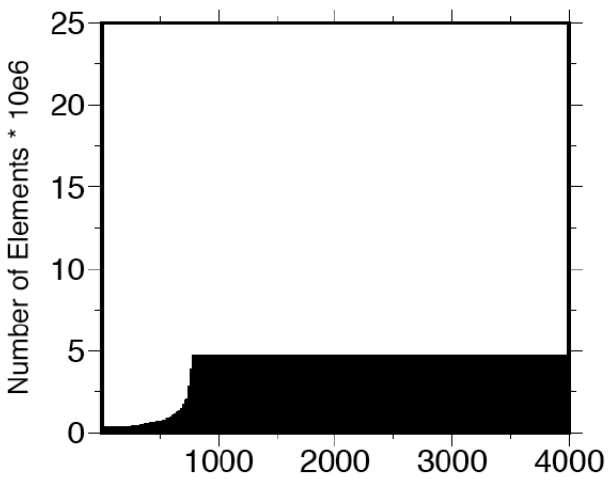
- The figures show number of nonzeros in each MPI task (processor).
- Left: Nonzeros in kernel is not evenly distributed. So evenly partition columns results in load imbalance.
- Middle: Strategy: uneven column partition that makes each MPI task has similar data load (number of nonzeros). However, perfect load balance result in uneven partition of vector, and thus result in communication imbalance.
- Right: Trade-off between computation load and communication load. Set max_nonzero/avg_nonzero=1.30



max_nonzero/avg_nonzero=6.71



max_nonzero/avg_nonzero=1.00

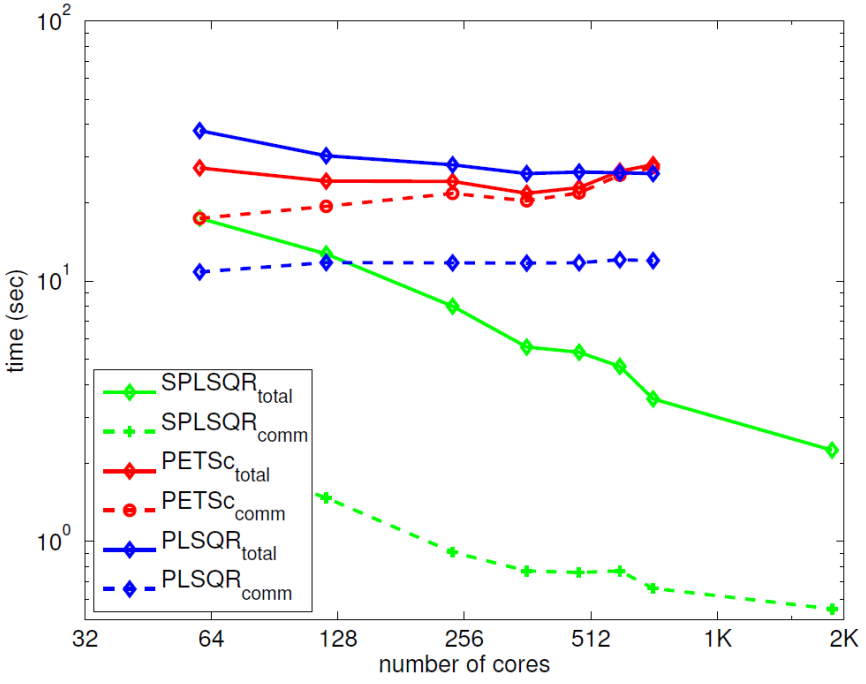


max_nonzero/avg_nonzero=1.30

Experiment: Kraken

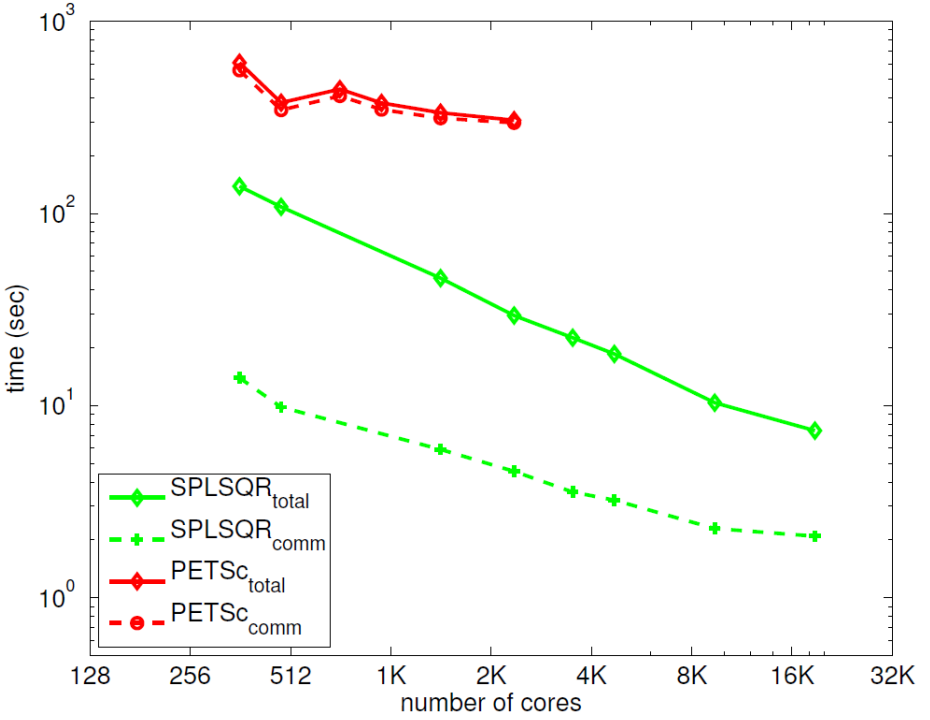
		DEC3	ANGF
nx, ny, nz	physical domain ($nx \times ny \times nz$)	$165 \times 256 \times 16$	$496 \times 768 \times 50$
m	# column	1,351,680	38,092,800
n_k	# rows in kernel	3,543	3,543
n_d	# of rows in damping	8,877,544	261,330,576
nnz_{Ak}	# non-zeros kernel	183,113,885	5,321,630,642
nnz_{Ad}	# non-zeros damping	27,596,000	818,542,016

- The figure shows total and communication time for 100 iterations of the SPLSQR, PLSQR, and PETSc implementations for the DEC3 data set from 60 to 1920 cores.
- Execution time of the SPLSQR algorithm is 1.7x less than PETSc at 60 cores, and is 7.8x less at 720 cores.
- Communication cost for SPLSQR is over 50x less than either PLSQR or PETSc.



Experiment: Kraken

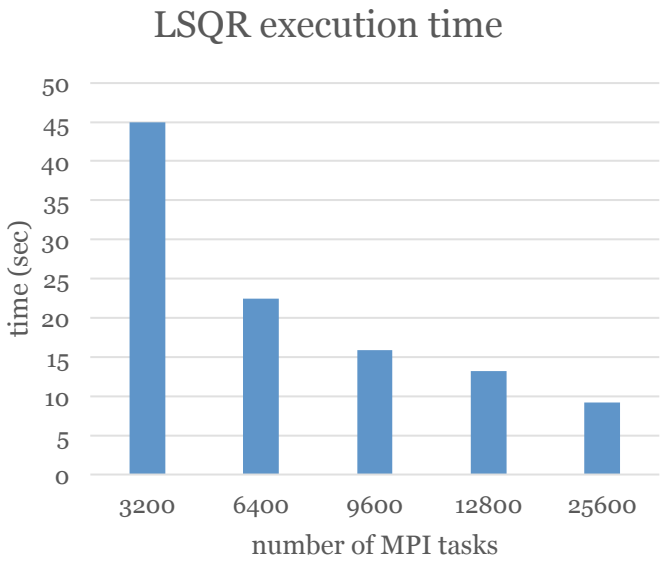
- The figure shows total and communication time for 100 iterations of SPLSQR and PETSc for the ANGF data set from 360 to 19,200 cores.
- The reduction in execution time for SPLSQR versus PETSc varies from a low of 4.3x on 360 cores to a high of 9.9x on 2400 cores.
- SPLSQR algorithm significantly reduces communication cost versus PETSc by greater than a factor of 100x.



Initial Experiment: Blue Waters

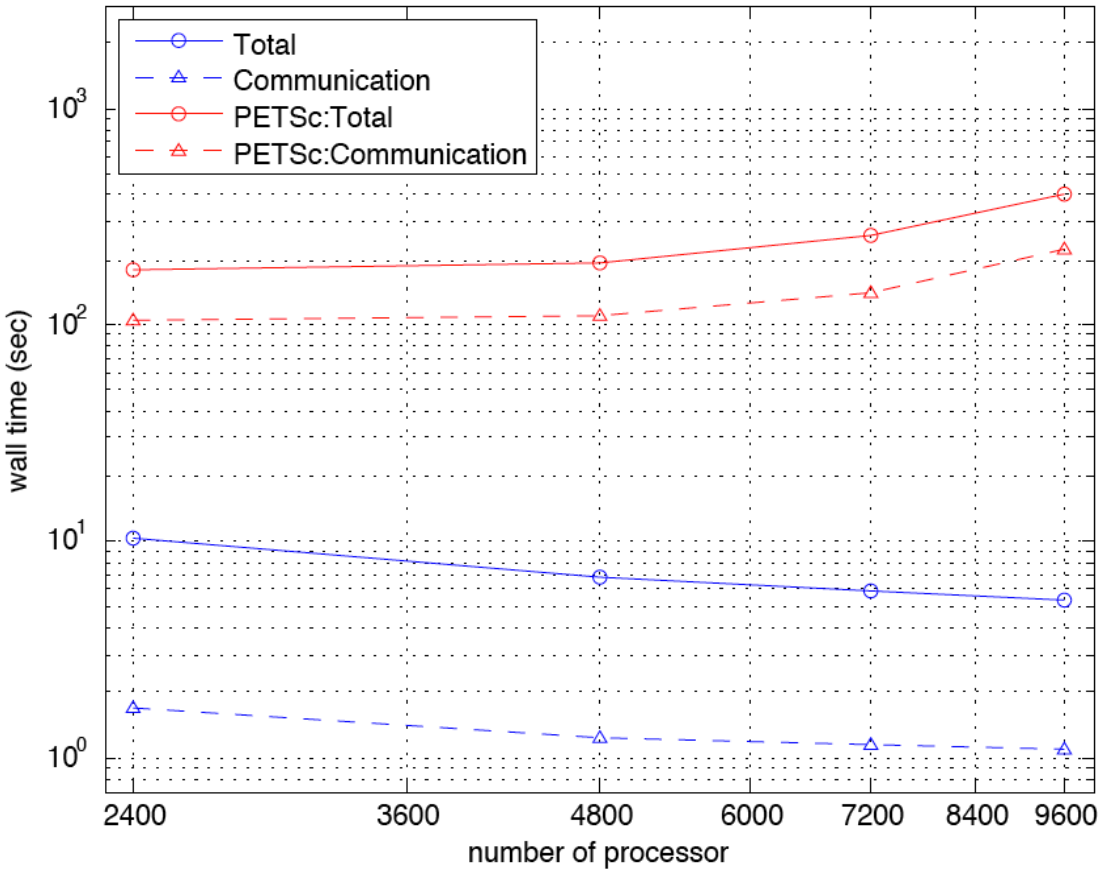
		EQANGF62K	EQANGF125K
nx, ny, nz	physical domain ($nx \times ny \times nz$)	$496 \times 768 \times 50$	$496 \times 768 \times 50$
m	# column	38,093,022	38,093,067
n_k	# rows in kernel	6,2851	125,520
n_d	# of rows in damping	261,330,576	261,330,576
nnz_{Ak}	# non-zeros kernel	68,074,889,274	143,524,414,175
nnz_{Ad}	# non-zeros damping	818,542,016	818,542,016

- Perform much larger experiment than Kraken.
- The figure shows preliminary result on EQANGF62K using BW XE nodes
- Scalable from 3200 cores to 25,600 cores.
- Will do more experiment using XE or XK nodes in the future.



Initial Experiment: Yellowstone

- Performance comparisons between ours and PETSc's implementations of the LSQR algorithm for 100 LSQR iterations of the 12K dataset from 2,400 to 9,600 processors.
- Tested on EQANGF125K

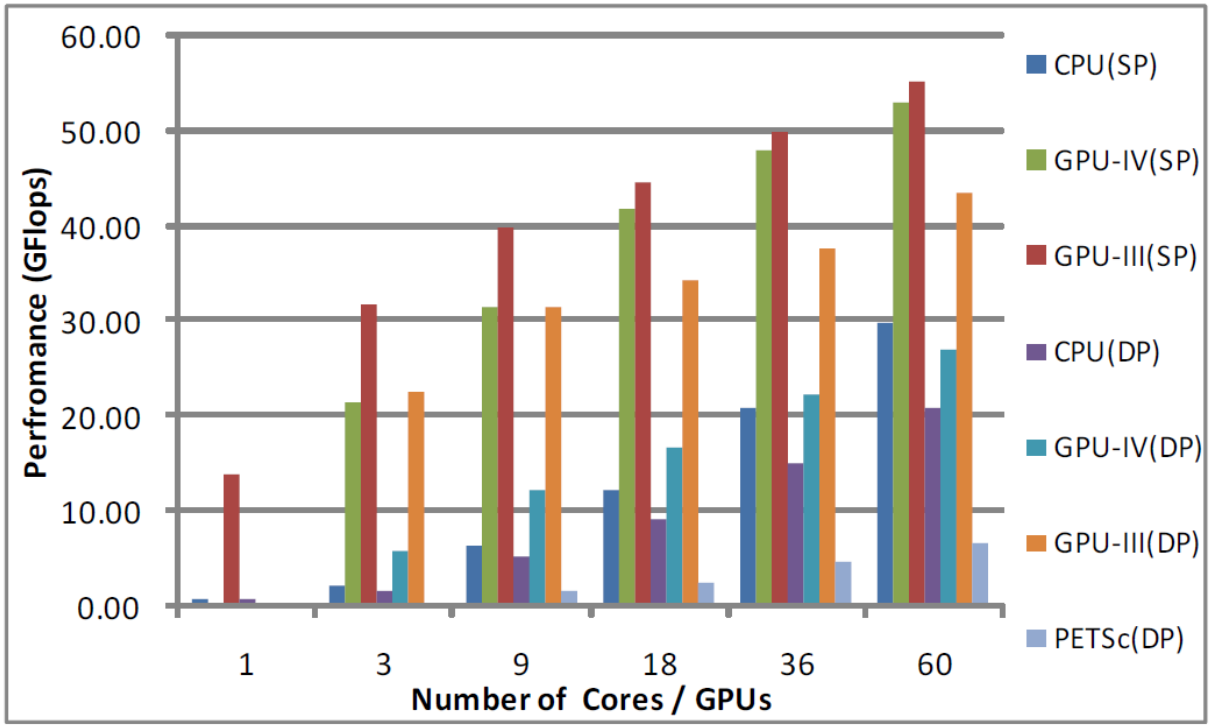


Selecting the best CUDA kernel for SpMV ($y \leftarrow y + A \times x$ and $x \leftarrow x + A^T \times y$)

Implementation	Matrix storage format	$y \leftarrow y + A \times x$ kernel	$x \leftarrow x + A^T \times y$ kernel
CPU	one copy of CSR	our own csrmv (CSR)	our own csrmv trans (CSR)
GPU-I	one copy of CSR	CUSPARSE csrmv (CSR)	CUSPARSE csrmv trans (CSR)
GPU-II	one copy of CSC	CUSPARSE csrmv trans (CSC)	CUSPARSE csrmv (CSC)
GPU-III	one copy of CSR, one copy of CSC	CUSPARSE csrmv (CSR)	CUSPARSE csrmv (CSC)
GPU-IV	one copy of CSR	CUSPARSE csrmv (CSR)	our own csrmv trans (CSR)

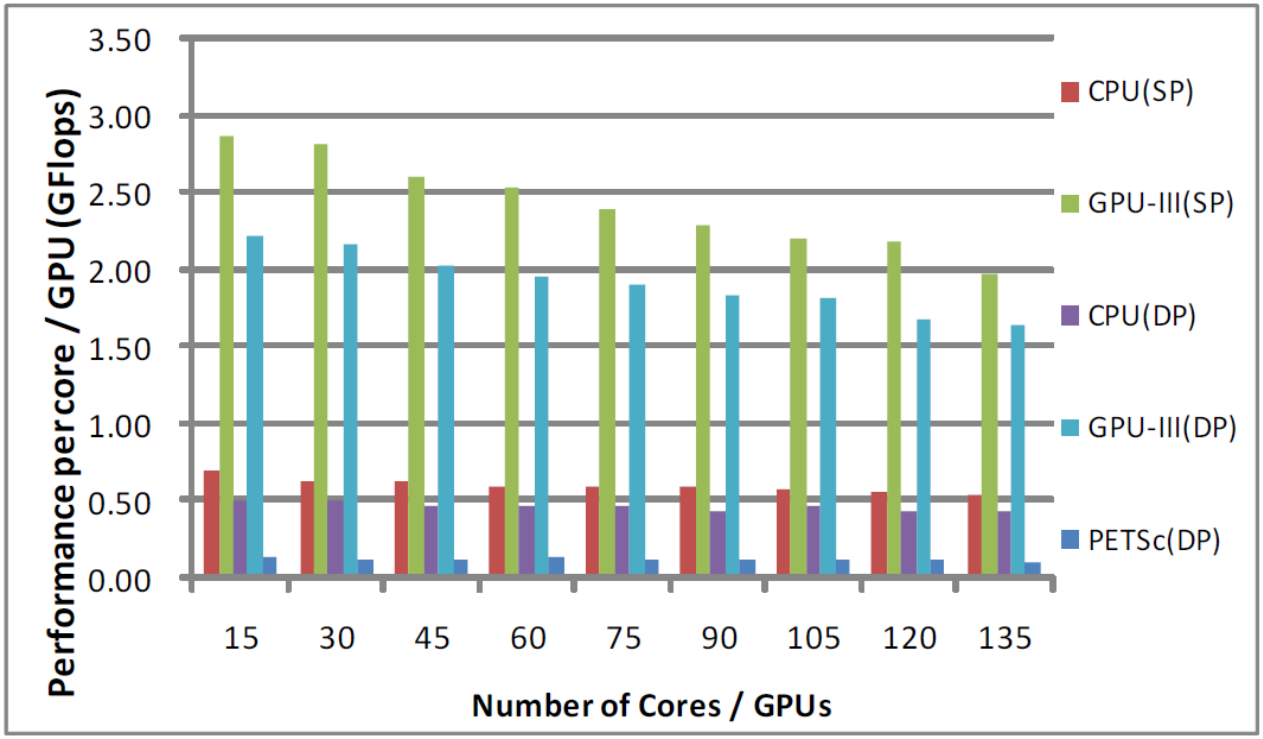
- GPU-I: pass matrix in CSR and vector directly to cusparseXcsrmv in cuSPARSE library to calculate $\mathbf{y \leftarrow y + A \times x}$ and $\mathbf{x \leftarrow x + A^T \times y}$.
- GPU-II: cuSPARSE library supports matrix format conversion between different storage formats;
  - Convert matrix from CSR to CSC. Only do once at the first iteration.
  - Treat CSC matrix as a CSR matrix, invoke cusparseXcsrmv.
- GPU-III: combination of I and II; one copy of CSR, and one copy of CSC.
- GPU-IV: write own kernel to perform $\mathbf{x \leftarrow x + A^T \times y}$, without transform the matrix, save half space.

Strong scalability



- **Strong scalability** defines how the execution time varies with the number of cores for a fixed problem size.
- Our multi GPUs approach is scalable from 1 to 60 CPU cores/ GPUs
- GPU approach is faster than corresponding CPU approach.
- Better performance than PETSc.

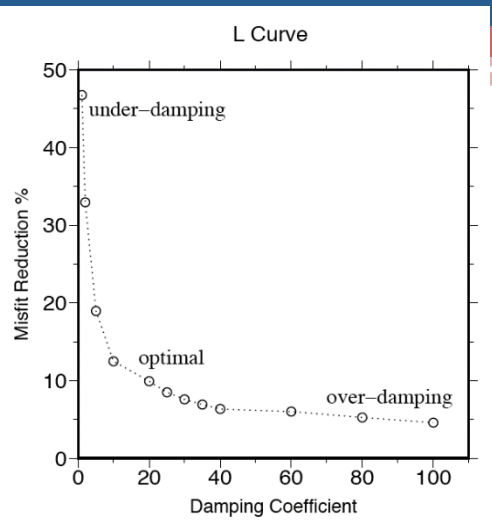
Weak Scalability

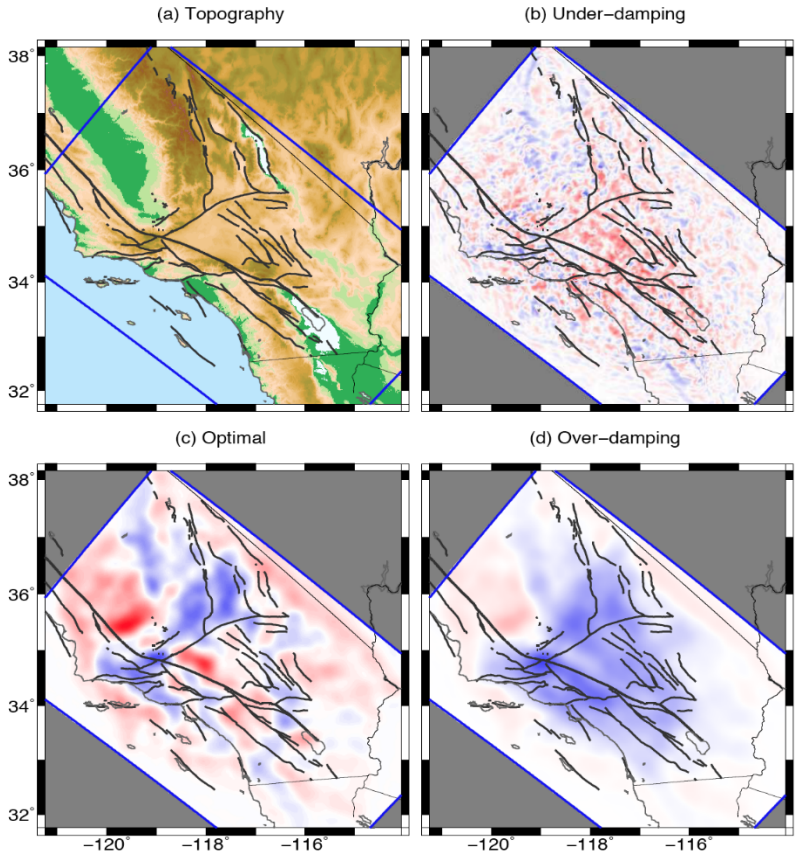


- **Weak scalability** shows how the execution time varies with the number of cores for a fixed problem size per core.
- Y axis is the performance per CPU core or per GPU.
- Performance per CPU core or per GPU slows down a little as the number of cores/GPUs increases.
- GPU approach is faster than corresponding CPU approach.
- Better performance than PETSc.

Science Impact

- Many LSQR runs are required to find the optimal damping coefficients. It is now feasible for large scale seismic tomographic inversion thanks to the improved algorithm. Top figure shows misfit reduction when try different damping coefficients.
- (a) The map shows the study area: the topography and major faults (thick black lines) of southern California.
- (b, c, d) are perturbation maps, the red regions represent velocity reduction areas and the blue regions represent velocity increase areas.
- (b) under-damping: perturbations are oscillated.
- (c) optimal: perturbations show many correlations with geological structures.
- (d) over-damping: perturbations are too smooth.





Conclusion and Future Work

- SPLSQR algorithm utilizes particular characteristics of coefficient matrix that include both pseudo-dense and sparse components.
- Demonstrate that the SPLSQR algorithm has scalable communication volume and significantly reduces communication cost compared with existing approaches.
- Utilize GPU direct technology to speed up communication between GPUs across network.
- Will do more large scale experiment in Blue Waters.