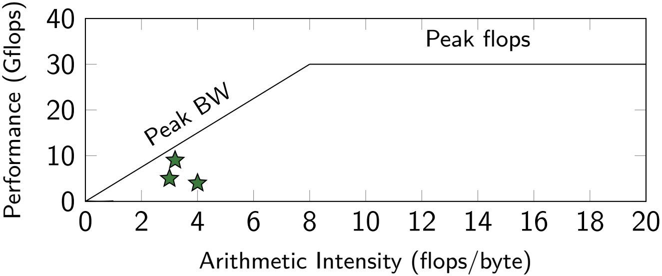
Structure-Aware Methods for Sparse Linear Systems

Joseph McLaughlin

November 2025

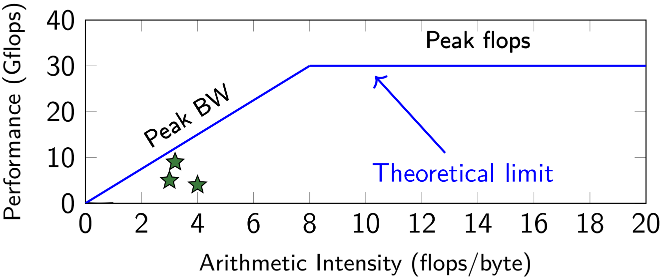
The Gap Between Peak and Achieved Performance

- Sparse linear systems derived from partial differential equations (PDEs) rely on a variety of sparse tasks.
- Such tasks only achieve 1–5% of peak on data-parallel devices.



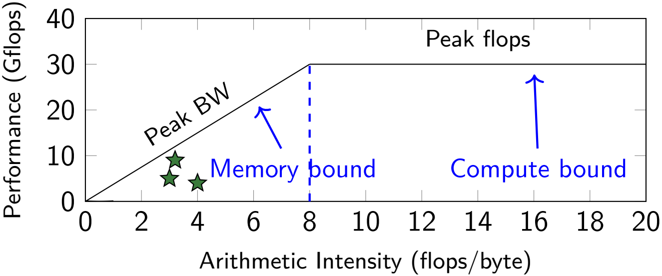
The Gap Between Peak and Achieved Performance

- Sparse linear systems derived from partial differential equations (PDEs) rely on a variety of sparse tasks.
- Such tasks only achieve 1–5% of peak on data-parallel devices.



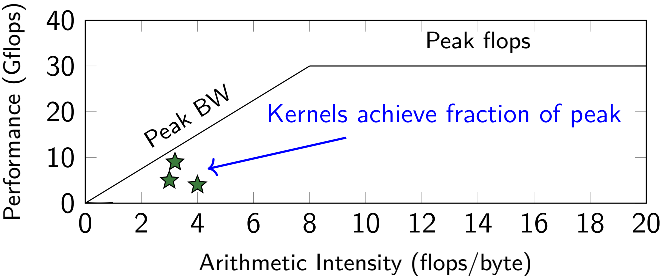
The Gap Between Peak and Achieved Performance

- Sparse linear systems derived from partial differential equations (PDEs) rely on a variety of sparse tasks.
- Such tasks only achieve 1–5% of peak on data-parallel devices.



The Gap Between Peak and Achieved Performance

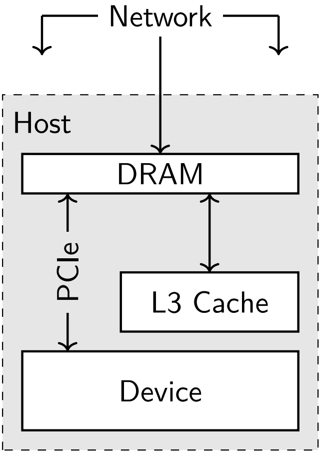
- Sparse linear systems derived from partial differential equations (PDEs) rely on a variety of sparse tasks.
- Such tasks only achieve 1–5% of peak on data-parallel devices.



The Memory Hierarchy

- Memory performance only degrades the further you move away compute units.
- At least $10 - 100\times$ worse throughput at every step up the hierarchy.

	Latency	Bandwidth	Capacity
Network	100 ms	25 GB/s	
DRAM	100 μs	300 GB/s	1 TB
PCIe	10 ms	128 GB/s	
Device	5 μs	3.4 TB/s	128 GB



Algorithmic Levers that Address the Gap

Memory footprint

- Efficient representations remain in cache and DRAM more often.

Communication pattern

- Effective coordination ensures that network calls are performed efficiently when necessary.

Algorithmic Levers that Address the Gap

Memory footprint

- Efficient representations remain in cache and DRAM more often.

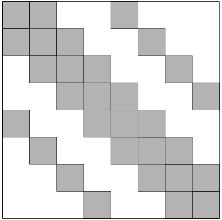
Communication pattern

- Effective coordination ensures that network calls are performed efficiently when necessary.

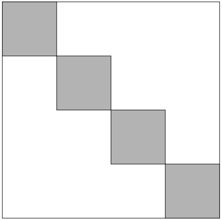
At scale we need to carefully tune both of these levers.

What PDE Discretizations Give Us

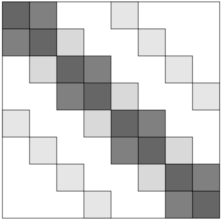
Patterned sparsity



Block structure



Connection strength



- Each property enables multiple algorithmic strategies.

Reducing The Memory Footprint

- **Historic question:** Does the problem fit into memory?
- **Contemporary question:** Where in the hierarchy does it fit?

- **Example:** 2D Poisson problem, $n = 8\,000\,000$
  - Original matrix $\approx$ 512 MB in CSR
  - LU Factors (natural ordering) $\approx$ 180 GB in CSR

Reducing The Memory Footprint

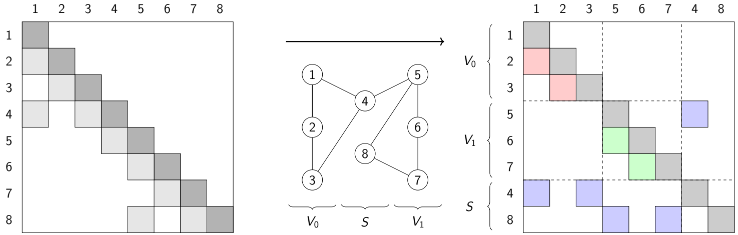
- **Historic question:** Does the problem fit into memory?
- **Contemporary question:** Where in the hierarchy does it fit?

- **Example:** 2D Poisson problem, $n = 8\,000\,000$
  - Original matrix $\approx$ 512 MB in CSR
  - LU Factors (natural ordering) $\approx$ 180 GB in CSR

Too big for GPU memory

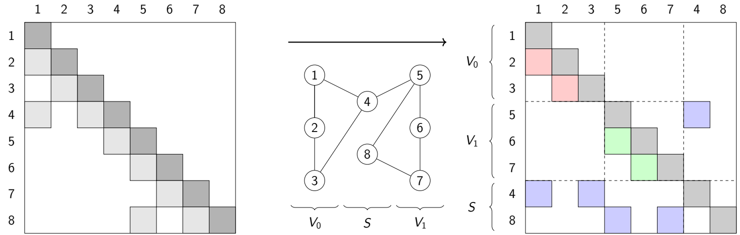
Reordering Methods

- **Example:** Nested Dissection
- Takes advantage of geometric separators
- Forms a graph problem: $G = (V, E)$
  - Vertices consist of rows and columns
  - Edges consist of off-diagonal
  - Find partition $V = (V_0, V_1, S)$ for a small separator S
  - Apply recursively on V_0 and V_1



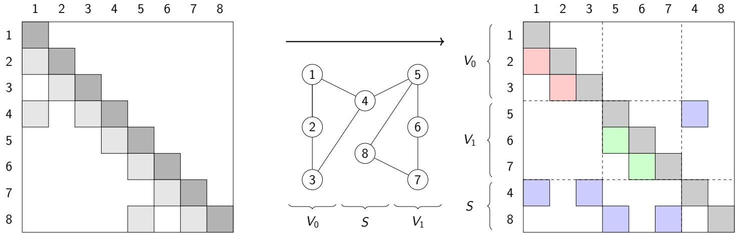
Reordering Methods

- Eliminating a variable creates fill between its neighbors
- Neighbors are already eliminated or in the same subgraph
- No fill between disconnected regions



Reordering Methods

- Dimensionality guarantees small separators
- For low dimensional spaces ($d < 4$) the separator is small
  - $O(\sqrt{n})$ for 2D
  - $O(n^{2/3})$ for 3D
- Natural ordering factors $\approx$ 180 GB
- ND factors $\approx$ 2 GB



Reordering Methods

Foundational work

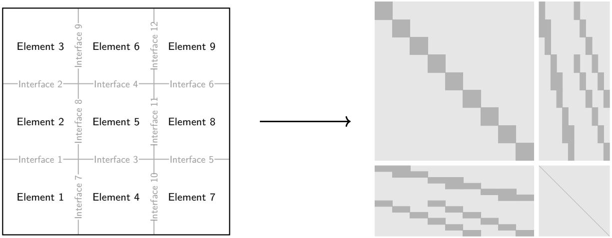
- Minimum Degree (1967)
  - Always eleminate the vertex with the smallest degree
- Nested Dissection (1973)
  - Separator-based
- Hypergraphs (2011)
  - Net-based partitioning for non-symmetric problems

Current directions

- Fast MD (2021)
  - Improves time complexity from $O(n^3)$ to $O(nm)$
- Data Reduction + ND (2021)
  - Reduction on graph before ND
- Temporal Reuse (2025)
  - Partitions inherent from prior timestep

Static Condensation

- **Static Condensation**: the process of reducing the problem
- A variety of discretizations yield block structures (*e.g.*, HDG, hybridized FD, spectral elements)
- Elements do not communicate interiors and are coupled only by neighboring element

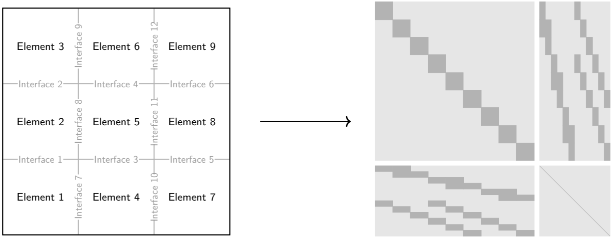


Static Condensation

- Block structure enables static condensation:

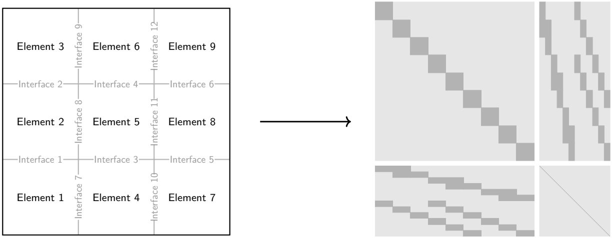
$$\begin{bmatrix} A_{II} & A_{I\Gamma} \\ A_{\Gamma I} & A_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} x_I \\ x_\Gamma \end{bmatrix} = \begin{bmatrix} b_I \\ b_\Gamma \end{bmatrix}$$

- Eliminate interiors: $Sx_\Gamma = b_\Gamma - A_{\Gamma I} A_{II}^{-1} b_I$
- Solve reduced system for x_Γ
- Recover interiors: $x_I = A_{II}^{-1}(b_I - A_{I\Gamma} x_\Gamma)$
- **Solve each block of A_{II} independently and in parallel**



Static Condensation

- Memory footprint depends on block structure.
- For the $n = 8\,000\,000$ 2D Poisson problem:
  - Natural ordering LU factors $\approx$ 180 GB
  - 10×10 DOFs per element yields factors for $A_{II} \approx$ 200 MB, S $\approx$ 25 GB
- Requires tuning to optimally load memory hierarchy.



Static Condensation

Foundational work

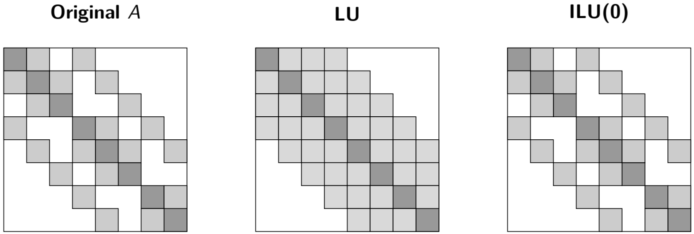
- Original derivation
  - Guyan Reduction (1965)
- High order numerical methods
  - SEM (2005)
  - HDG (2009)
- GPU implementations (2019)
  - Hybridization + SC enable dense local problems

Current directions

- Mixed Precision (2020)
  - Low precision for solving A_{II}
- Nested Condensation (2021)
  - Perform static condensation on S
- Macro-elements (2023)
  - Cuts against conventional small elements

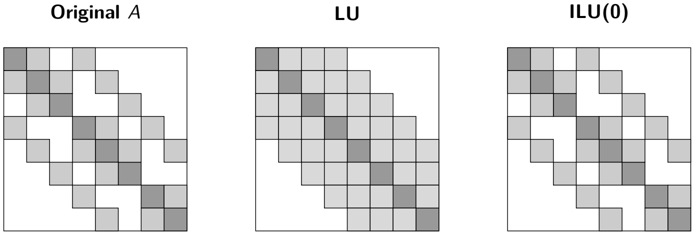
Incomplete Factorization

- When structure is insufficient we turn to incomplete methods.
- Incomplete methods limit fill-in during factorization.
- Act as preconditioners for iterative methods.
- Largely split between conditional (ILU(k)) and threshold (ILUT).



Incomplete Factorization

- Memory tradeoff:
  - Direct: 180 GB; solve once.
  - ILU(0): 512 MB; iterate until convergence.



Incomplete Factorization

Foundational work

- ILU(0) (1977)
- Modified ILU (1978)
- ILUT (1994)

Current directions

- Fine-grained parallel ILU (2015)
- GPU parallel (2020)
- GNN-based ILU (2023)

Managing The Memory Footprint

- Each method significantly reduces the problem's memory footprint through different structural properties

Applications often compose these methods to scale

- Static condensation + reorder elements + ILU on global problem
- Reordering + ILU + solve iteratively

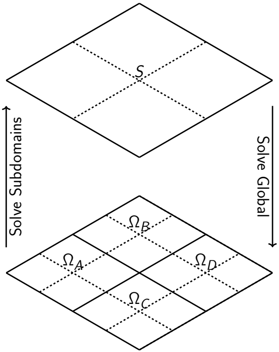
Minimizing Distributed Communication

- Issues with sparse problems at scale
  - Work (local) shrinks as you add processors
  - Bandwidth and latency remains constant
  - Communication eventually dominates
- When we have to utilize slow parts of the memory hierarchy, how do we do so effectively?
- Achieve this by reducing the volume of data or reducing the communication frequency.

Domain Decomposition

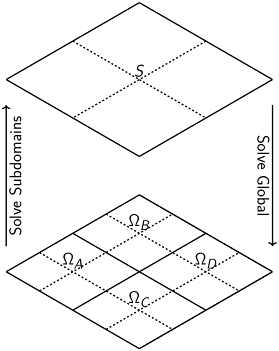
- Similar Schur complement structure, but at different scales
- S is never formed; applied via subdomain solve

- **Communication pattern**
  - Subdomain solves (none)
  - Apply S (halo)
  - Coarse solve (global, small)



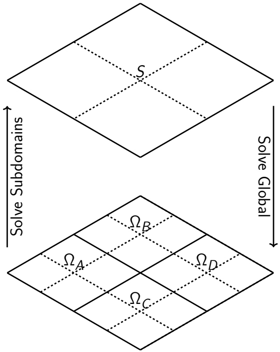
Domain Decomposition

- Solve the global problem implicitly with S
  - Minimize the interface error between subdomains
  - Optionally solve as a single coarse problem
- Coarse problem improves iterations from $O(\ell)$ to $O(1)$ for ℓ subdomains
- Specifics of coarse problem depend on the method (*e.g.*, BDDC, FETI-DP)



Domain Decomposition

- Good communication pattern and performs more work with less communication than multigrid
- The work itself may be less optimal
- Better for complicated domains that are less smooth



Domain Decomposition

Foundational work

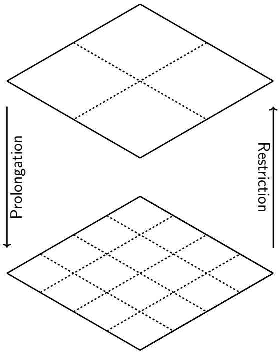
- Non-overlapping methods
  - FETI (2001)
  - BDD (2003)
- Two level methods
  - FETI-DP (1991)
  - BDDC (1993)

Current directions

- Adaptive coarse spaces (2011)
  - Coarse space chosen by slowly converging interfaces
- Three level methods (2022)
  - Recursive coarse space construction

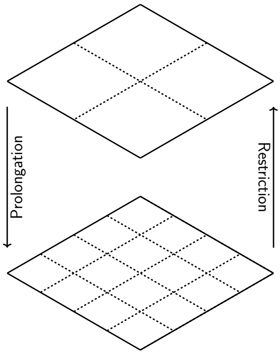
Multigrid

- Hierarchy of increasingly coarse grids
- Never solve fine grid directly; smooth and correct
- **Communication pattern**
  - Smooth (halo)
  - Restrict (halo)
  - Coarse solve (global, small)
  - Prolongate (halo)
  - Smooth (halo)



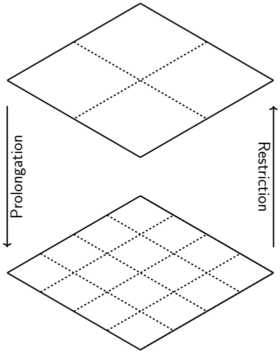
Multigrid

- Restrict error on a series of coarser grids
  - *Smoothing* removes oscillatory error
  - Only smooth error remains
  - Smooth error appears oscillatory at coarser grids
- Solve on coarse grid
- Prolongate solution to fine grid
  - Same, smooth error appears oscillatory on coarse grid
  - Removed on coarse grid



Multigrid

- Communicates more frequently, performs less work
- Work scales linearly ($O(n)$)
- Better for less complicated domains



Multigrid

Foundational work

- GPU implementations
  - Geometric MG (2011)
  - Algebraic MG (2014)
- Exascale MG (2012)
- Matrix-free (2019)

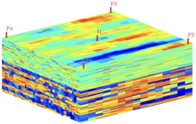
Current directions

- Mixed-precision AMG (2023)
- Tensor Core AMG (2024)
- Improved communication bounds on AMG (2025)

Limits of Structural Assumptions

- Our methods rely on structural assumptions:
  - Smooth error for multigrid
  - Good separators for reordering
  - Diagonal dominance for ILU
  - Block or element structure for condensation
- **Real problems** often push the boundaries one or more of these assumptions
- Methods become less effective
  - Slow convergence or outright divergence
  - Excessive fill or breakdown in factorization
  - Coarse spaces misses important values

Application: Reservoir Simulation (2025)



Context

- 100 million unknowns on CPU clusters
- Darcy flow with coeffs differing by 10^5–10^8
- Thin high-permeability channels and low-permeability barriers

Challenge

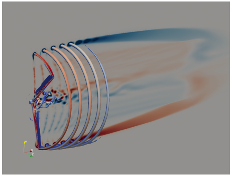
- Large coefficient jumps create localized error that MG and DD do not capture

How it is handled in practice

- Design coarse-space around this high contrast structure
- Keep subdomains inside highly contrasted regions

Takeaway: The entire solver must conform with high contrast information; no single method does this alone.

Application: ExaWind (2024)



Context

- 40 billion grid points
- Frontier (4,000 MI250X nodes)
- Multi-scale CFD
- Unstructured near-blade + structured background

Challenge

- Overset system lacks exploitable sparsity structure

How it is handled in practice

- Decompose into regional subdomains
- Each subdomains uses its own MG variant

Takeaway: Decomposition brings out the local structure that exists but is obscured by global coupling

Summary

- At scale, sparse PDE solvers are limited by **memory** and **communication**, not arithmetic.
- PDE discretizations provide **structure** that closes the gap:
  - patterned sparsity (reordering)
  - block structure (static condensation)
  - approximate structure (ILU)
- Each method targets a specific bottleneck:
  - memory footprint (ND, SC, ILU)
  - communication pattern (DD, MG)
- No single method achieves both; production codes at exascale compose methods to address multiple bottlenecks simultaneously.
- **Performance at scale requires composition:** matching structure exploitation to hardware constraints.

Questions