BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science

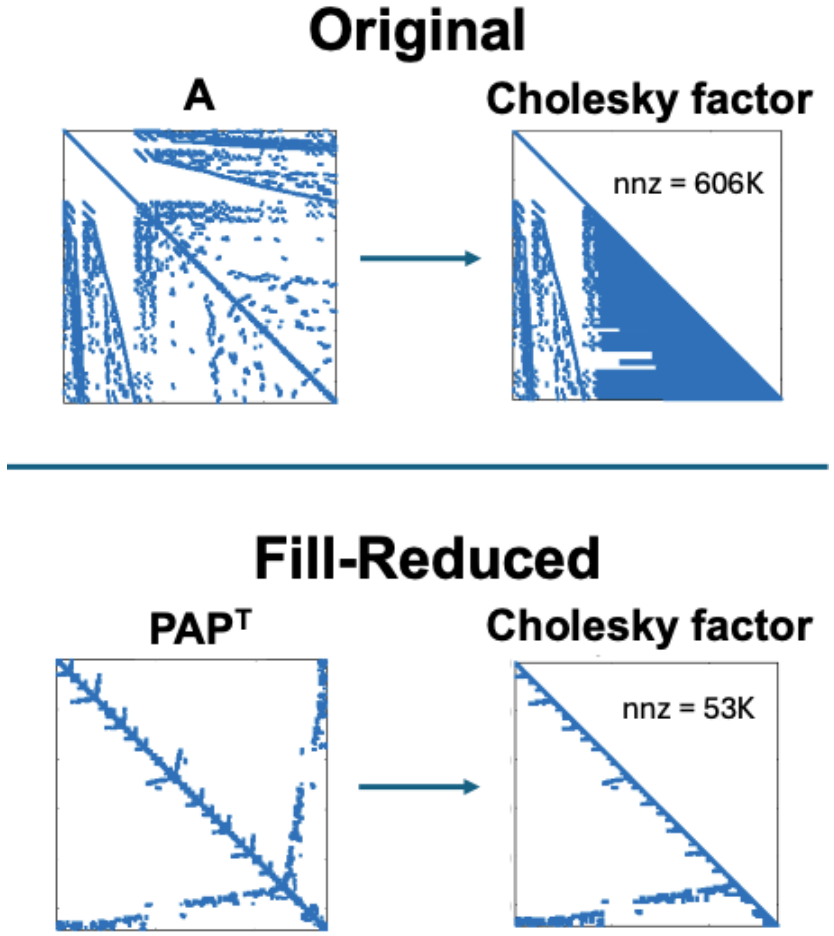
Parallelizing the Approximate Minimum Degree Ordering Algorithm: Strategies and Evaluation

Yen-Hsiang Chang[1], Aydın Buluç[2], James Demmel[1]

[1]Electrical Engineering and Computer Science Department, UC Berkeley
[2]Computational Research Division, Lawrence Berkeley National Lab

Reordering for sparse Cholesky is necessary

- **Reducing fill-in** for sparse Cholesky factorization is a cornerstone for solving sparse symmetric positive definite systems.
- Nested dissection and the **approximate minimum degree (AMD)** algorithm are the two heuristics being widely used nowadays.



Why do we even want to parallelize the AMD algorithm?

We want to make it more future-proof before Amdahl's law becomes prominent.

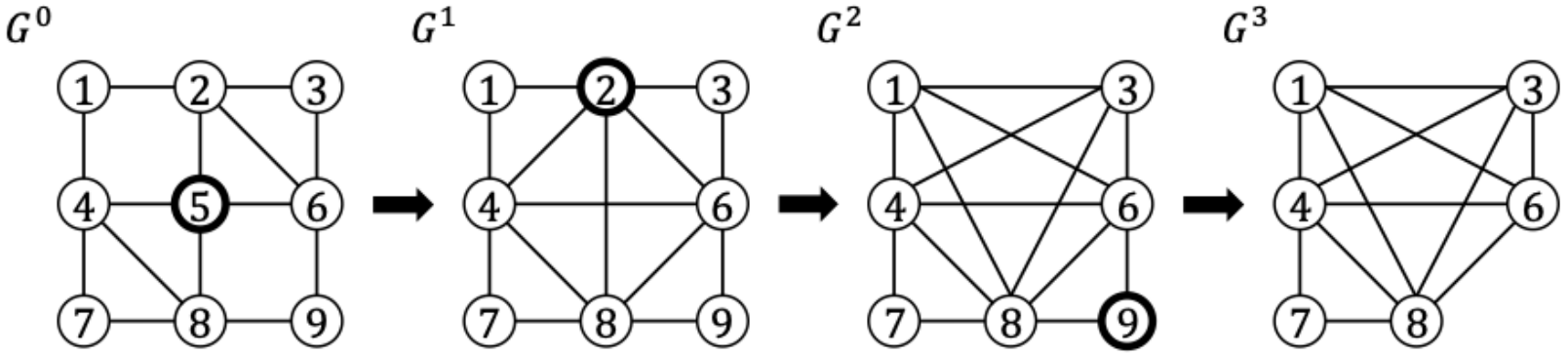
- While significant work has been done to parallelize nested dissection on both shared and distributed memory systems, no similar efforts have been made for the AMD algorithm.
- Meanwhile, GPU Cholesky solvers have steadily improved over time, driven by advances in software, and are expected to continue accelerating with future GPU generations.
  - But the AMD algorithm has no improvement in the past decades.

Table 1: Time (in seconds) for SuiteSparse AMD on an AMD EPYC 7763 CPU, and for Cholesky solver cuSolverSp (v11.6.1) and its successor cuDSS (v0.5.0) on an A100 GPU.

Matrix Name	SuiteSparse AMD	cuSolverSp	cuDSS
nd24k	0.83	117.17	4.11
ldoor	1.28	11.01	3.55
Flan_1565	4.18	out of memory	16.85

The AMD algorithm: elimination graphs

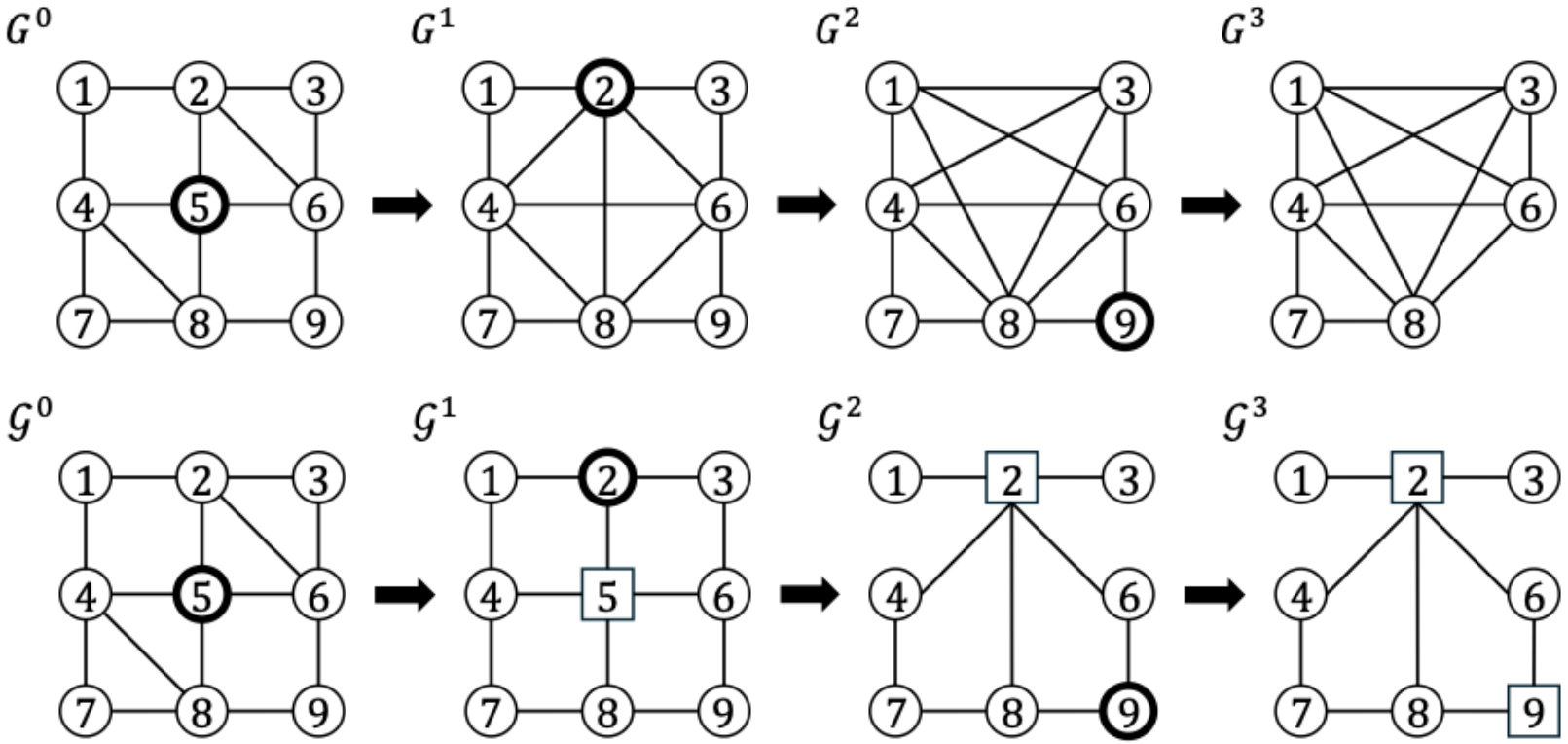
- Sparse Cholesky factorization can be represented using elimination graphs, where a rank-1 update is equivalent to removing a vertex and adding a clique in its neighborhood.
- To reduce fill-ins, the heuristic is to select a minimum degree vertex as the pivot to eliminate at each step.



The AMD algorithm: quotient graphs

Use the invariant $N_v = \left(\mathcal{A}_v \cup \bigcup_{e \in \mathcal{E}_v} \mathcal{L}_e\right) \setminus \{v\}$.

- To represent the sparsity pattern more efficiently, quotient graphs represent cliques more compactly—storing only the set of vertices in each clique.



The AMD algorithm: degree approximation

Maintaining degrees using the invariant $N_v = \left(\mathcal{A}_v \cup \bigcup_{e \in \mathcal{E}_v} \mathcal{L}_e\right) \setminus \{v\}$ is a bottleneck.

- To mitigate the degree update bottleneck, the AMD algorithm estimates an upper bound for the exact degree using three heuristics:
  - the size of the remaining submatrix
  - the worst-case fill-ins
  - union bound without double counting the neighborhood of the pivot

$$d_v^k = \min \left\{ \begin{array}{l} n - k - 1, \\ d_v^{k-1} + |\mathcal{L}_p \setminus \{v\}| - 1, \\ |\mathcal{A}_v \setminus \{v\}| + |\mathcal{L}_p \setminus \{v\}| + \sum_{e \in \mathcal{E}_v} |\mathcal{L}_e \setminus \mathcal{L}_p| \end{array} \right\}$$

Why is it hard to parallelize the AMD algorithm?

Intra-elimination parallelism is not enough!

- A straightforward approach to extract parallelism for the AMD algorithm is to parallelize using atomic operations. But there is limited amount of work with high contention.

$$d_v^k = \min \left\{ \begin{array}{l} n - k - 1, \\ d_v^{k-1} + |\mathcal{L}_p \setminus \{v\}| - 1, \\ |\mathcal{A}_v \setminus \{v\}| + |\mathcal{L}_p \setminus \{v\}| + \sum_{e \in \mathcal{E}_v} |\mathcal{L}_e \setminus \mathcal{L}_p| \end{array} \right\}$$

Algorithm 1 Computation of $|\mathcal{L}_e \setminus \mathcal{L}_p|$ for all $e \in \overline{V}$

```
1: Assume w(e) < 0 for all e ∈ V̄.
2: for each variable v ∈ L_p do
3:     for each element e ∈ E_v do
4:         if w(e) < 0 then
5:             w(e) ← |L_e|
6:         w(e) ← w(e) − 1
```

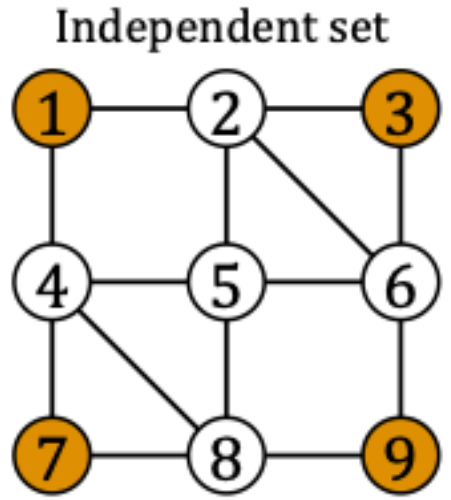
Table 2: Average sizes of the sets across all elimination steps, representing the amount of parallelism, the amount of work, and the number of unique elements accessed, respectively, by applying intra-elimination parallelism.

Matrix Name	$\lvert\mathcal{L}_p\rvert$	$\sum_{v \in \mathcal{L}_p} \lvert\mathcal{E}_v\rvert$	$\lvert\bigcup_{v \in \mathcal{L}_p} \mathcal{E}_v\rvert$
nd24k	329.7	587.5	14.0
Flan_1565	43.8	64.8	10.2
nlpkkt240	80.5	542.8	56.3

Why is it hard to parallelize the AMD algorithm?

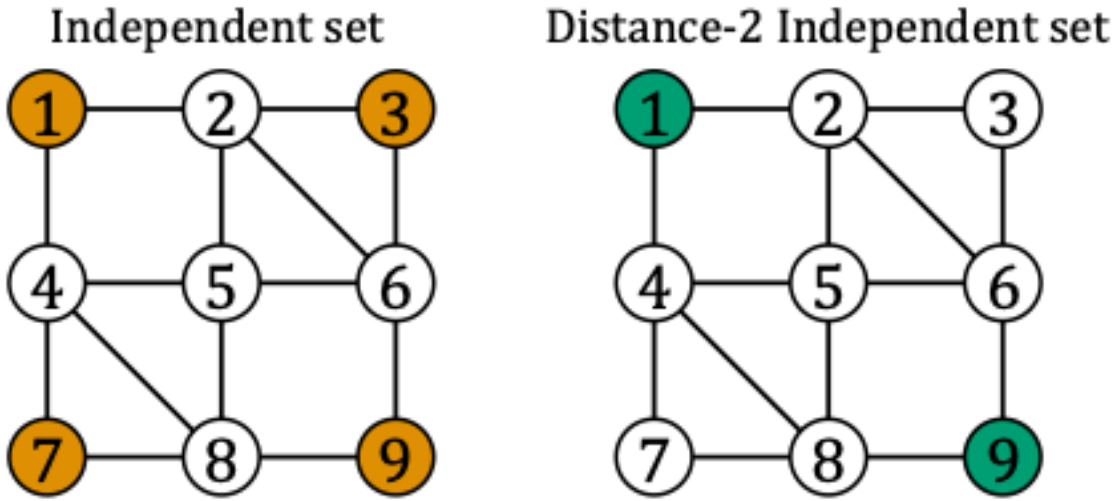
Inter-elimination parallelism is not enough either!

- The multiple minimum degree algorithm eliminates a maximal independent set of pivots with a consolidated degree update afterward.
- In the original design, performance gains from multiple elimination stemmed from the overlap of pivots' neighborhoods.
- However, such overlap is not well-suited to the AMD algorithm due to:
  - high contention
  - complex degree approximation



Our main contribution 1: multiple elimination via distance-2 independent sets

- To avoid high contention and complex degree approximation, we exploit **multiple elimination via distance-2 independent sets**.
- This allows each pivot to work independently in both quotient graph and degree updates.



Our main contribution 2: relaxation of the minimum degree criteria

- To get more parallelism, we relax the minimum degree criteria by a multiplicative factor $mult$.
- This indeed affects the reordering quality, but in a controllable manner as shown in our preprint.

Table 3: Average sizes of maximal distance-2 independent sets across all elimination steps with various values of multiplicative relaxation factor, $mult$, where all variables in the set are eliminated in each step.

Matrix Name	$mult = 1.0$	$mult = 1.1$	$mult = 1.2$
nd24k	2.2	9.0	10.9
Flan_1565	42.0	448.5	678.1
nlpkkt240	57.5	4084.5	6695.8

Other contributions

- To make the full AMD algorithm parallelized, we also implemented the following techniques:
  - The quotient graph data structure is redesigned to allow concurrent updates.
  - The approximate degree lists, which are used for selecting pivots, are duplicated to mitigate contention.
  - Distance-2 independent sets are found in parallel using an analog of the Luby's randomized maximal independent set algorithm.
- All details are included in our preprint.

Evaluation: environment and matrix suite

- We evaluated the performance on an AMD EPYC 7763 CPU with 64 cores.

Table 4: Matrices used in our experiments sorted by #nonzeros. Matrices pre2, HV15R, and stokes are nonsymmetric.

Matrix Name	#rows	#nonzeros	Problem Type
pre2	659K	5.83M	circuit simulation
nd24k	72.0K	28.7M	3D mesh
ldoor	952K	42.5M	structural
dielFilterV3real	1.10M	89.3M	electromagnetics
Flan_1565	1.56M	114M	structural
HV15R	2.02M	283M	fluid dynamics
Queen_4147	4.15M	317M	structural
stokes	11.4M	349M	semiconductor
nlpkkt240	28.0M	761M	optimization

Evaluation: comparison of AMD time

Table 5: Ordering time comparison between SuiteSparse AMD and our parallel AMD implementation with 1 thread (1t) and 64 threads (64t). Matrices were randomly permuted five times with all methods evaluated on the same set of permutations to decouple tie-breaking issues. We report the median ordering time across the five runs. Speedups ratios are computed directly based on the medians.

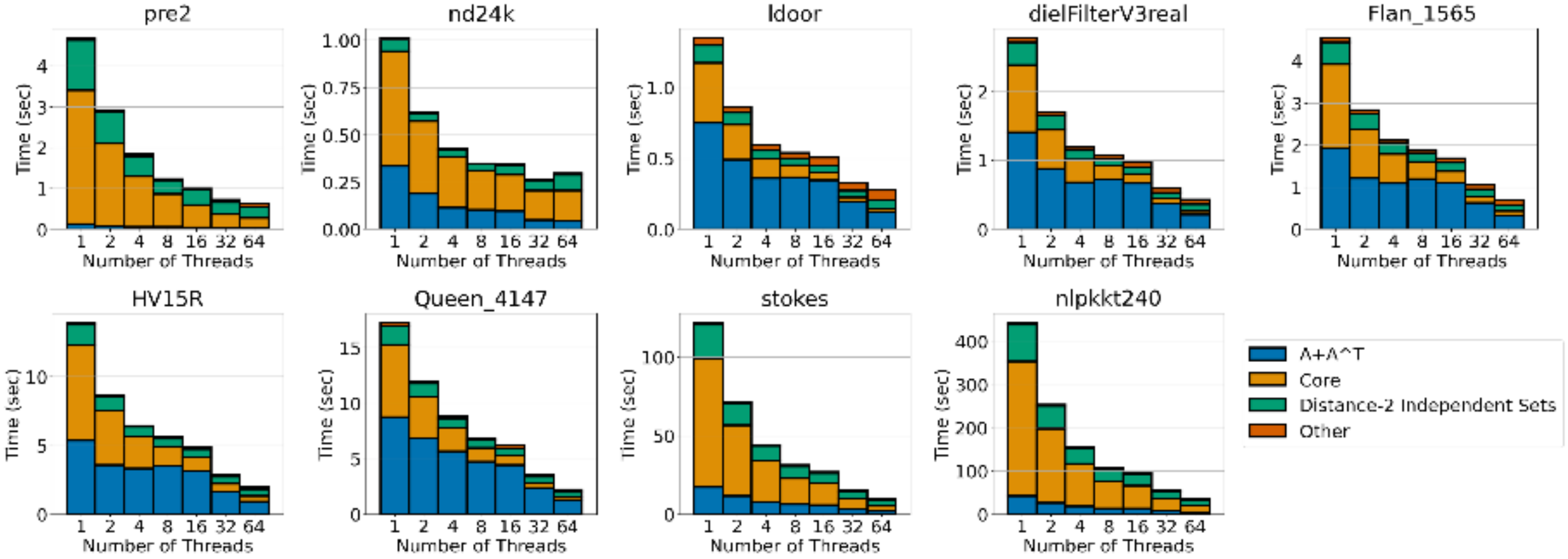
Matrix Name	Ordering Time (sec)			Our Speedup		
	SuiteSparse	Ours		over SuiteSparse		Self
		1t	64t	1t	64t	64t/1t
pre2	2.01	4.70	0.62	0.43×	3.25×	7.56×
nd24k	0.82	1.01	0.26	0.81×	3.15×	3.89×
ldoor	1.37	1.35	0.29	1.01×	4.79×	4.74×
dielFilterV3real	2.63	2.78	0.46	0.95×	5.74×	6.04×
Flan_1565	4.20	4.51	0.80	0.93×	5.28×	5.68×
HV15R	11.75	13.86	2.15	0.85×	5.46×	6.42×
Queen_4147	13.82	17.16	2.46	0.81×	5.61×	6.93×
stokes	79.32	121.06	10.63	0.66×	7.46×	11.30×
nlpkkt240	296.67	433.95	35.75	0.68×	8.30×	12.21×

Evaluation: comparison of AMD fill-ins

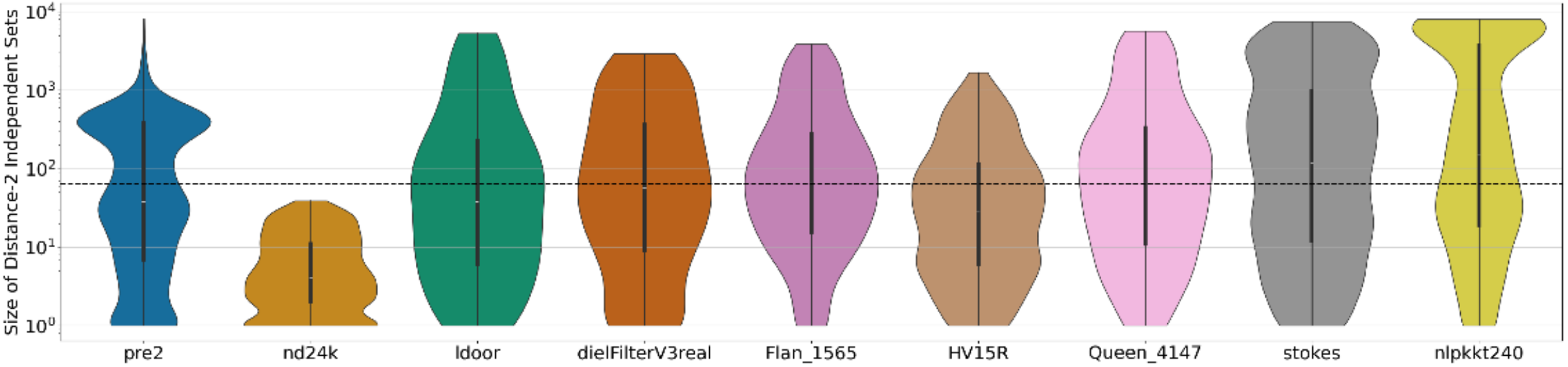
Table 6: Fill-ins comparison between SuiteSparse AMD and our parallel AMD implementation with 1 thread (1t) and 64 threads (64t). Matrices were randomly permuted five times with all methods evaluated on the same set of permutations to decouple tie-breaking issues. We report the median fill-in across the five runs. Fill-in ratios are computed directly based on the medians.

Matrix Name	#Fill-ins			Our Fill-in Ratio over SuiteSparse	
	SuiteSparse	Ours			
		1t	64t	1t	64t
pre2	1.84e+08	1.93e+08	1.91e+08	1.05×	1.04×
nd24k	5.09e+08	5.23e+08	5.07e+08	1.03×	1.00×
ldoor	1.52e+08	1.57e+08	1.58e+08	1.03×	1.04×
dielFilterV3real	9.65e+08	1.06e+09	1.07e+09	1.10×	1.11×
Flan_1565	3.72e+09	3.96e+09	4.01e+09	1.07×	1.08×
HV15R	5.65e+10	5.65e+10	5.80e+10	1.00×	1.03×
Queen_4147	7.72e+10	8.35e+10	8.63e+10	1.08×	1.12×
stokes	7.76e+10	8.23e+10	8.00e+10	1.06×	1.03×
nlpkkt240	1.60e+12	1.83e+12	1.79e+12	1.14×	1.12×

Evaluation: time breakdown



Evaluation: sizes of distance-2 independent sets



Conclusion and future work

- We have presented a scalable AMD implementation, using distance-2 independent sets and designing specialized concurrent data structures tailored to the AMD algorithm.
- To make our approach more scalable, hybrid approaches might be needed, e.g., combining intra- and inter-elimination parallelism or fusing AMD and nested dissection.
- We are also planning to add more evaluation experiments, including comparison with nested dissection, end-to-end effect when incorporating with Cholesky solvers, and augmenting the matrix suite.

Q&A

https://arxiv.org/abs/2504.17097