# Tackling Parallelization Challenges of Randomized Preconditioners with Dependency Tracking

Tianyu Liang[1] Chao Chen[2] Yotam Yaniv[3] Hengrui Luo[4] David Tench[3]
Xiaoye S. Li[3] Aydın Buluç[3] James Demmel[1] Jie Chen[5]

[1]UC Berkeley [2]NC State University [3]Lawrence Berkeley National Laboratory [4]Rice University [5]Amazon Web Services

SC26, Chicago, November 2026

Code: `github.com/Tianyu-Liang/Parallel-Randomized-Cholesky`

# The problem: large sparse graph Laplacian systems

$$L\,x = b, \qquad L \text{ a graph Laplacian (more generally, symmetric diagonally dominant)}$$

**Where they come from**

- spectral graph partitioning and clustering
- graph-based machine learning
- finite-difference / finite-element discretizations of elliptic PDEs

**Scale in this talk:** $10^6$ to $5 \times 10^7$ unknowns, $10^8$ nonzeros

**How they are solved**

- direct: reliable, but fill-in hits a memory wall
- iterative (PCG): cheap per step, lives or dies by the **preconditioner**

## The question

Can we build a preconditioner that is **robust on irregular graphs** and **parallel enough for a GPU**?

# Two standard preconditioners and their failure modes

**Incomplete Cholesky (IC)**

- memory-efficient, easy to apply
- drops fill by level or threshold
- can break down on irregular graphs: negative pivots, or thousands of iterations
- setup is sequential by nature

**Algebraic multigrid (AMG)**

- excellent on structured PDE meshes
- multilevel hierarchy costs setup time and memory
- can fail on heterogeneous graphs; can run out of GPU memory

## Approximate Cholesky (AC): a randomized IC

Kyng & Sachdeva (2016); Gao, Kyng & Spielman (2023). Replace each clique update by **sampled edges**.
**Provably breakdown-free** for any elimination order. Sequential reference implementation: `rchol` (Chen et al., 2021).

# Fill-in is clique formation; AC samples the clique

One step of Cholesky on vertex $k$ forms a Schur complement. On the graph: the neighbors of $k$ become a **clique**.

$$L \leftarrow L - L^{(k)} + C^{(k)}, \qquad L^{(k)} = \sum_{i \in \mathcal{N}_k} (-\ell_{ki})\, \mathbf{b}_{ki} \mathbf{b}_{ki}^{\top}, \quad \mathbf{b}_{ki} = \mathbf{e}_k - \mathbf{e}_i$$

**Exact Cholesky**

- $C^{(k)}_{\text{exact}} = L^{(k)} - \ell_{kk}^{-1} L(:,k) L(k,:)$
- a vertex of degree $d$ creates $O(d^2)$ edges
- cliques become dense: the memory wall

**Approximate Cholesky**

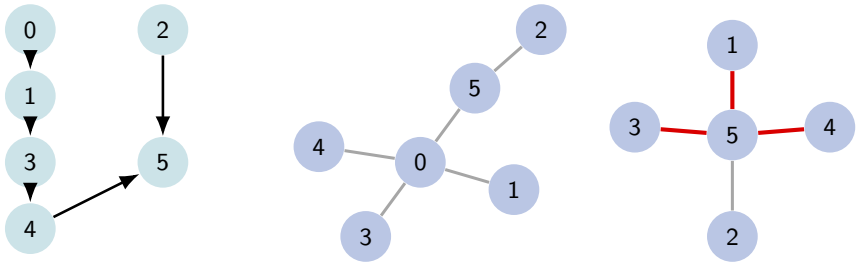
- $C^{(k)}$ is a **random spanning-tree-like graph** on $\mathcal{N}_k$ with $O(|\mathcal{N}_k|)$ edges
- $\mathbb{E}[C^{(k)}] = C^{(k)}_{\text{exact}}$: unbiased
- every realization is a Laplacian, so the factorization **never breaks down**

`SampleClique`$(L, k)$: sort neighbors by $|\ell_{ki}|$; walk the list, pairing each neighbor $i$ with one neighbor $j$ sampled with probability $\propto |\ell_{kj}|$, weight $S\, \ell_{ki}/\ell_{kk}$.

# The setup bottleneck: stochastic fill defeats static scheduling

- Exact Cholesky: the fill pattern is known symbolically, so the **elimination tree** schedules the parallel factorization in advance.
- AC: the sampled edges are **random**, so the dependency structure only exists at run time.
- The exact elimination tree is a valid schedule for AC, but a pessimistic one: it serializes vertices that AC leaves independent.

**(a) Elimination tree** **(b) Input graph** **(c) After eliminating** 0




(a) The elimination tree serializes $1 \to 3 \to 4 \to 5$. (c) After eliminating 0, AC's sampled tree (red) leaves 1, 3, 4 ready at once.

# ParAC: dynamic dependency tracking

**Execution model: online readiness**

- $\texttt{dp}[v]$ = number of unprocessed vertices $v$ still depends on
- a vertex is **ready** when $\texttt{dp}[v] = 0$; ready vertices go into a queue
- eliminating $k$: sample edges, *increment* dp of new dependents, then *decrement* dp of $k$'s neighbors; whoever reaches 0 is enqueued
- increments are recorded before decrements, so no vertex is enqueued early

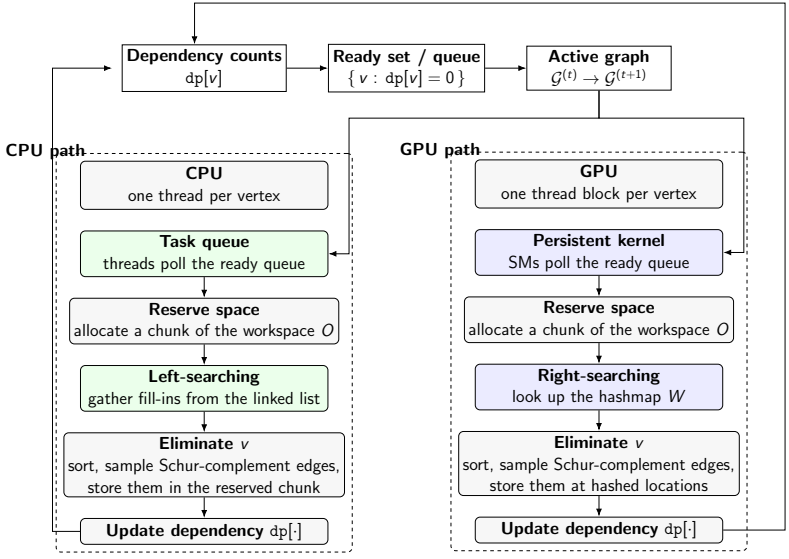
**What it preserves**

- for a prescribed elimination order, ParAC changes **only the scheduling**
- same numerical updates, same output distribution as AC
- therefore inherits the **breakdown-free** guarantee

**What it gains**

- parallelism that cannot be scheduled statically
- no separator hierarchy to build (unlike `rchol`'s nested dissection)

# The ParAC pipeline on CPUs and GPUs



# CPU design: left-searching with linked lists

**One thread per vertex**

- threads poll the ready queue; spin-wait briefly when empty
- each vertex **reserves a chunk** of a pre-allocated workspace for its factor column
- sampled updates are stored where they are produced and **chained by a linked list** per target vertex
- a vertex gathers its fill by walking its list ("left-searching", like left-looking sparse LU)

**Concurrency control**

- atomic exchange keeps the linked list consistent when several pivots update one target
- atomic add / sub on dp; atomic append to the queue
- no locks, no dynamic allocation in the hot loop

**Per-vertex work**: merge duplicates, sort by $|\ell_{ik}|$, suffix sums, sample $|\mathcal{N}_k| - 1$ edges, write the normalized column.

# GPU design: persistent kernel, right-searching with a hash map

**Why the CPU design does not port**

- pointer chasing cannot use intra-block parallelism
- dynamic allocation inside kernels is prohibitive
- per-vertex work varies: warp divergence

**Design**

- **one thread block per vertex**; neighbors mapped across the warp
- **persistent kernel**: one launch for the whole factorization, blocks poll the queue
- **right-searching**: updates are written near `hash`(target); the target probes from its hash code

**Inside a block**

- merge duplicates by sort + prefix sum (warp-level, dynamic sizes handled in-house)
- parallel suffix sums; sampling of $|\mathcal{N}_k| - 1$ edges in parallel
- atomics on `dp` and the queue, as on the CPU

**Hashing matters**: nearby start locations cause linear-probing conflicts among concurrently active pivots; we choose the hash to spread the active targets.

# Experimental setup

**Hardware (NERSC Perlmutter)**

- CPU: AMD EPYC 7763, 32 threads
- GPU: NVIDIA A100, 40 GB

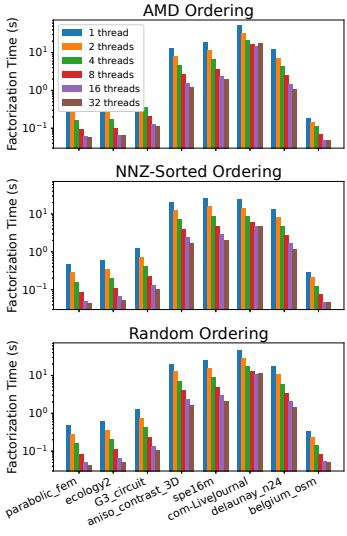
**Baselines**

- CPU: MATLAB `ichol` (threshold, iso-memory), `hypre` BoomerAMG
- GPU: `cuSPARSE` IC(0), `AmgX`

**Solver**: PCG to relative residual $10^{-8}$; totals include setup and symbolic analysis.

| Matrix | Columns | NNZ |
|---|---|---|
| parabolic_fem | 525,825 | 3,674,625 |
| ecology1 / ecology2 | 1,000,000 | 4,996,000 |
| apache2 | 715,176 | 4,817,870 |
| G3_circuit | 1,585,478 | 7,660,826 |
| uniform / aniso / contrast 3D Poisson | 14,348,907 | 100,088,055 |
| spe16m | 16,003,008 | 111,640,032 |
| GAP-road | 23,947,347 | 57,708,624 |
| com-LiveJournal | 3,997,962 | 69,362,378 |
| europe_osm | 50,912,018 | 108,109,320 |
| delaunay_n24 | 16,777,216 | 100,663,202 |
| venturiLevel3 | 4,026,819 | 16,108,474 |
| belgium_osm | 1,441,295 | 3,099,940 |

# Ordering on the CPU: fill-reducing AMD is a strong default




- three orderings: AMD, NNZ-sorted (ascending degree), random
- ordering changes both the fill AC produces and the dependency structure ParAC sees
- on 32 threads, **AMD** gives the lowest factorization time across the suite

# CPU results: total time to solution (s), iterations in parentheses

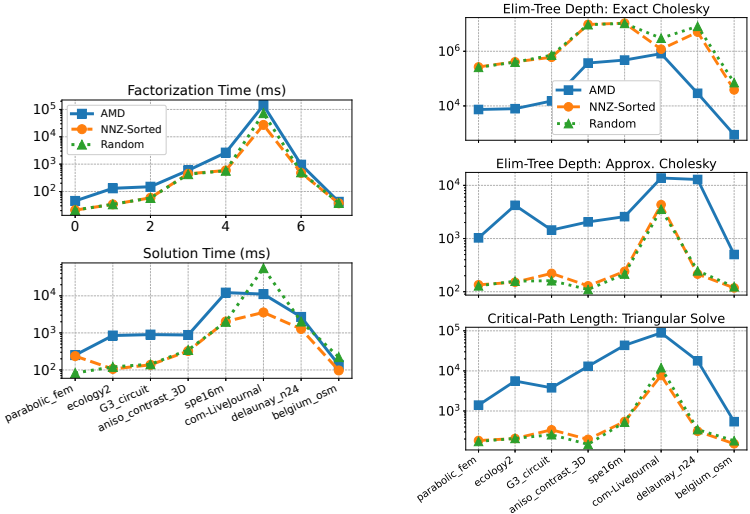
| | **ParAC** | | MATLAB `ichol` | | `hypre` AMG | |
|---|---|---|---|---|---|---|
| Matrix | total | (it.) | total | (it.) | total | (it.) |
| parabolic_fem | 0.72 | (36) | 3.69 | (231) | **0.41** | (7) |
| ecology1 | 1.15 | (42) | 16.25 | (637) | **0.62** | (7) |
| apache2 | 0.85 | (31) | 5.11 | (225) | **0.61** | (9) |
| G3_circuit | 2.30 | (48) | 9.86 | (222) | **1.16** | (8) |
| uniform Poisson | 19.35 | (30) | 70.58 | (102) | **14.84** | (8) |
| aniso Poisson | **6.77** | (11) | 9.70 | (7) | 8.88 | (6) |
| contrast Poisson | 77.24 | (142) | 63.05 | (91) | **13.73** | (8) |
| spe16m | 32.14 | (53) | 63.50 | (85) | **15.68** | (9) |
| GAP-road | 40.48 | (71) | 1728.99 | (2285) | **26.67** | (13) |
| com-LiveJournal | **35.15** | (23) | 207.62 | (15) | 253.62 | (18) |
| europe_osm | 86.89 | (72) | 2755.48 | (2008) | **65.56** | (15) |
| delaunay_n24 | 19.79 | (33) | 1134.15 | (1701) | **17.52** | (10) |
| venturiLevel3 | 5.71 | (51) | 169.78 | (1458) | **3.59** | (9) |
| belgium_osm | 1.42 | (43) | 7.16 | (215) | **1.23** | (11) |

**vs.** `ichol`: faster on 14 of 15; up to **57.3×** (delaunay_n24); fewer iterations on 12 of 15.

**vs.** `hypre`: AMG wins on structured PDE meshes; ParAC wins on com-LiveJournal by **7.2×** and on anisotropic Poisson; close on the road and Delaunay graphs.

**Setup cost**: ParAC's factorization is a small fraction of the total on every matrix.

# Ordering on the GPU: NNZ-sorted wins




- same three orderings, A100
- **NNZ-sorted** consistently beats AMD on the GPU
- AMD reduces fill but produces *deeper* dependency chains in the sampled factors and longer triangular solves
- the GPU's 7.6× higher memory bandwidth rewards orderings that expose concurrency

# GPU results: total time to solution (ms), iterations in parentheses

| | ParAC | | cuSPARSE IC(0) | | AmgX | |
|---|---|---|---|---|---|---|
| Matrix | total | (it.) | total | (it.) | total | (it.) |
| parabolic_fem | 527 | (40) | 467 | (923) | **68** | (10) |
| ecology1 | **162** | (48) | 1146 | (1846) | 246 | (24) |
| ecology2 | 163 | (49) | 1402 | (2181) | **96** | (11) |
| apache2 | 177 | (25) | 723 | (1141) | **147** | (11) |
| G3_circuit | 227 | (37) | 1033 | (1019) | **131** | (11) |
| uniform Poisson | 2937 | (28) | 5174 | (256) | **1268** | (9) |
| aniso Poisson | 941 | (10) | 8551 | (431) | **520** | (11) |
| contrast Poisson | 5625 | (127) | 12544 | (638) | **710** | (12) |
| spe16m | 2865 | (48) | 15437 | (694) | **650** | (13) |
| GAP-road | **3607** | (106) | 213456 | (cap) | 3749 | (32) |
| com-LiveJournal | **35224** | (27) | 3517 | (95) | out of memory | |
| europe_osm | **7546** | (104) | 444754 | (cap) | 11430 | (28) |
| delaunay_n24 | 2052 | (46) | 107135 | (4555) | **839** | (13) |
| venturiLevel3 | 552 | (54) | 14756 | (4391) | **167** | (11) |
| belgium_osm | **144** | (50) | 4201 | (5432) | 860 | (28) |

**vs. IC(0)**: up to **52.2**× (delaunay_n24); IC(0) hits the $10^4$-iteration cap on two road networks.

**vs.** AmgX: AMG wins on structured meshes; ParAC wins on the irregular graphs by up to **6.0**× (belgium_osm), and runs where AmgX exhausts 40 GB (com-LiveJournal).

ParAC totals include the mandatory cuSPARSE symbolic analysis.

# Robustness: 200 heterogeneous SDDM systems

**SDDM2023 chimera benchmark** (Gao, Kyng & Spielman)

- 50 base graphs recursively combining expanders, grids, trees, random graphs
- weighted and unweighted; with and without boundary
- $n = 100{,}000$ vertices each
- uniform acceptance: residual below $10^{-7}$, no crash, no iteration cap

| Solver | Platform | Failures / 200 |
|---|---|---|
| `hypre` | CPU | 15 |
| **ParAC** | CPU | **0** |
| `AmgX` | GPU | 19 |
| **ParAC** | GPU | **0** |

`hypre`: crashes on all four variants of one seed, iteration caps elsewhere. `AmgX`: fails on 19 weighted or unweighted Laplacians.

## The point

Breakdown-free by construction is not a theoretical nicety. On heterogeneous graphs it is the difference between converging and not.

# Where ParAC belongs

**Use AMG when**

- the system is a structured PDE discretization
- a multilevel hierarchy fits in memory and converges in $\sim$10 iterations

**Use ParAC when**

- the graph is irregular (road networks, social graphs, Delaunay meshes, chimeras)
- device memory cannot hold an AMG hierarchy
- classical IC fails to converge or breaks down
- setup cost matters: ParAC's factorization is cheap and parallel

**Headroom not spent here**: AC($k$) invests more in construction to cut iterations; the factorization-cost margin leaves room for it.

# Conclusions and what is next

**ParAC**: a parallel execution framework for randomized incomplete Cholesky

- numerical kernel: the AC clique sampler of Gao, Kyng & Spielman, unchanged
- the advance: **dynamic dependency tracking**, online readiness instead of a precomputed separator hierarchy
- CPU: lock-free linked lists; GPU: persistent kernel with hashed right-searching
- up to **57.3**× (CPU) and **52.2**× (GPU) over deterministic IC; up to **7.2**× / **6.0**× over AMG on irregular graphs; **0/200** failures where AMG fails on 15 and 19

**Open questions**

- a mathematical account of the fill-reduction vs. parallelism trade-off in the ordering
- choosing the elimination order *dynamically* as sampled fill changes (maximal independent sets)
- mixed-sign SDD systems without the Laplacian reduction
- ParAC as a parallel, breakdown-free **smoother inside AMG**: how much sampling can be dropped?

`github.com/Tianyu-Liang/Parallel-Randomized-Cholesky` Thank you.

# Backup: the AC algorithm

**Require:** Laplacian $L \in \mathbb{R}^{N \times N}$
**Ensure:** $G$ lower triangular, $L \approx GG^\top$
1: $G = 0$
2: **for** $k = 1$ **to** $N - 1$ **do**
3: $\quad G(:, k) = L(:, k)/\sqrt{\ell_{kk}}$
4: $\quad L = L - L^{(k)} + \texttt{SampleClique}(L, k)$
5: **end for**

**Require:** $L$, index $k$
**Ensure:** Laplacian $C$ of sampled edges
1: $\mathcal{N}_k = \{i \neq k : \ell_{ki} \neq 0\}$, sorted by $|\ell_{ki}|$ ascending
2: $S = \ell_{kk}$
3: **while** $|\mathcal{N}_k| > 1$ **do**
4: $\quad$ pop the first $i$; $S = S + \ell_{ki}$
5: $\quad$ sample $j \in \mathcal{N}_k$ with probability $|\ell_{kj}|/S$
6: $\quad C = C - \dfrac{S\,\ell_{ki}}{\ell_{kk}}\,\mathbf{b}_{ij}\mathbf{b}_{ij}^\top$
7: **end while**

Provenance: Kyng & Sachdeva 2016 (spectral guarantee); Gao, Kyng & Spielman 2023 (this sampling rule); `rchol`, Chen, Liang, Biros 2021 (breakdown-free, sequential, nested dissection).

# Backup: CPU results, full

| | ParAC | | | | MATLAB ichol | | | | hypre | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Matrix | fact | solve | total | it | fact | solve | total | it | setup | solve | total | it |
| parabolic_fem | 0.06 | 0.66 | 0.72 | 36 | 0.23 | 3.46 | 3.69 | 231 | 0.25 | 0.16 | 0.41 | 7 |
| ecology1 | 0.06 | 1.09 | 1.15 | 42 | 0.13 | 16.12 | 16.25 | 637 | 0.39 | 0.23 | 0.62 | 7 |
| ecology2 | 0.06 | 1.11 | 1.17 | 43 | 0.13 | 16.19 | 16.32 | 844 | 0.41 | 0.23 | 0.64 | 7 |
| apache2 | 0.12 | 0.73 | 0.85 | 31 | 0.40 | 4.71 | 5.11 | 225 | 0.32 | 0.29 | 0.61 | 9 |
| G3_circuit | 0.11 | 2.19 | 2.30 | 48 | 0.43 | 9.43 | 9.86 | 222 | 0.68 | 0.48 | 1.16 | 8 |
| uniform Poisson | 2.61 | 16.74 | 19.35 | 30 | 8.96 | 61.62 | 70.58 | 102 | 9.37 | 5.47 | 14.84 | 8 |
| aniso Poisson | 1.18 | 5.59 | 6.77 | 11 | 5.17 | 4.53 | 9.70 | 7 | 4.22 | 4.66 | 8.88 | 6 |
| contrast Poisson | 1.57 | 75.67 | 77.24 | 142 | 6.70 | 56.35 | 63.05 | 91 | 8.23 | 5.50 | 13.73 | 8 |
| spe16m | 2.00 | 30.14 | 32.14 | 53 | 7.76 | 55.74 | 63.50 | 85 | 8.52 | 7.16 | 15.68 | 9 |
| GAP-road | 0.83 | 39.65 | 40.48 | 71 | 1.68 | 1727.31 | 1728.99 | 2285 | 13.28 | 13.39 | 26.67 | 13 |
| com-LiveJournal | 17.32 | 17.83 | 35.15 | 23 | 193.48 | 14.14 | 207.62 | 15 | 236.16 | 17.46 | 253.62 | 18 |
| europe_osm | 1.67 | 85.22 | 86.89 | 72 | 2.70 | 2752.78 | 2755.48 | 2008 | 31.71 | 33.85 | 65.56 | 15 |
| delaunay_n24 | 1.10 | 18.69 | 19.79 | 33 | 5.81 | 1128.34 | 1134.15 | 1701 | 10.09 | 7.43 | 17.52 | 10 |
| venturiLevel3 | 0.21 | 5.50 | 5.71 | 51 | 0.90 | 168.88 | 169.78 | 1458 | 1.95 | 1.64 | 3.59 | 9 |
| belgium_osm | 0.05 | 1.37 | 1.42 | 43 | 0.08 | 7.08 | 7.16 | 215 | 0.63 | 0.60 | 1.23 | 11 |

# Backup: GPU results, full (ms)

| | ParAC | | | | cuSPARSE IC(0) | | AmgX | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Matrix | fact | solve | total | it | total | it | setup | solve | total | it |
| parabolic_fem | 21 | 237 | 527 | 40 | 467 | 923 | 52 | 16 | 68 | 10 |
| ecology1 | 34 | 107 | 162 | 48 | 1146 | 1846 | 46 | 200 | 246 | 24 |
| ecology2 | 34 | 106 | 163 | 49 | 1402 | 2181 | 74 | 22 | 96 | 11 |
| apache2 | 49 | 94 | 177 | 25 | 723 | 1141 | 118 | 29 | 147 | 11 |
| G3_circuit | 59 | 138 | 227 | 37 | 1033 | 1019 | 98 | 33 | 131 | 11 |
| uniform Poisson | 819 | 1780 | 2937 | 28 | 5174 | 256 | 1106 | 162 | 1268 | 9 |
| aniso Poisson | 442 | 324 | 941 | 10 | 8551 | 431 | 361 | 159 | 520 | 11 |
| contrast Poisson | 545 | 4851 | 5625 | 127 | 12544 | 638 | 516 | 194 | 710 | 12 |
| spe16m | 587 | 2028 | 2865 | 48 | 15437 | 694 | 440 | 210 | 650 | 13 |
| GAP-road | 481 | 2985 | 3607 | 106 | 213456 | 10000 | 438 | 3311 | 3749 | 32 |
| com-LiveJournal | 26354 | 3698 | 35224 | 27 | 3517 | 95 | out of memory | | | |
| europe_osm | 1040 | 6041 | 7546 | 104 | 444754 | 10000 | 873 | 10557 | 11430 | 28 |
| delaunay_n24 | 465 | 1420 | 2052 | 46 | 107135 | 4555 | 336 | 503 | 839 | 13 |
| venturiLevel3 | 131 | 373 | 552 | 54 | 14756 | 4391 | 114 | 53 | 167 | 11 |
| belgium_osm | 39 | 86 | 144 | 50 | 4201 | 5432 | 53 | 807 | 860 | 28 |