INVITED TALKS

# SPIKe: Generating Practical SpMV Kernels for Sparse Iterative Methods on GPUs

Luhan Wang, Haipeng Jia , Kun Li , Zongyuan He, Shengguo Li, Ting Cao, Yunxin Liu, Yunquan Zhang, Yifeng Chen

Peking University · Chinese Academy of Sciences · Tsinghua University

SC26 hpc unites | Chicago

# The contradiction in preprocessing-centric SpMV

## Kernel Performance

- Higher kernel performance often requires sophisticated matrix-specific preprocessing (including reordering, format conversion, and auto-tuning)
- More preprocessing requires more iterations to amortize
- ***Kernel speedups need increasing Iteration counts to recover the preprocessing cost***

Gap

## Practical utility of the kernel

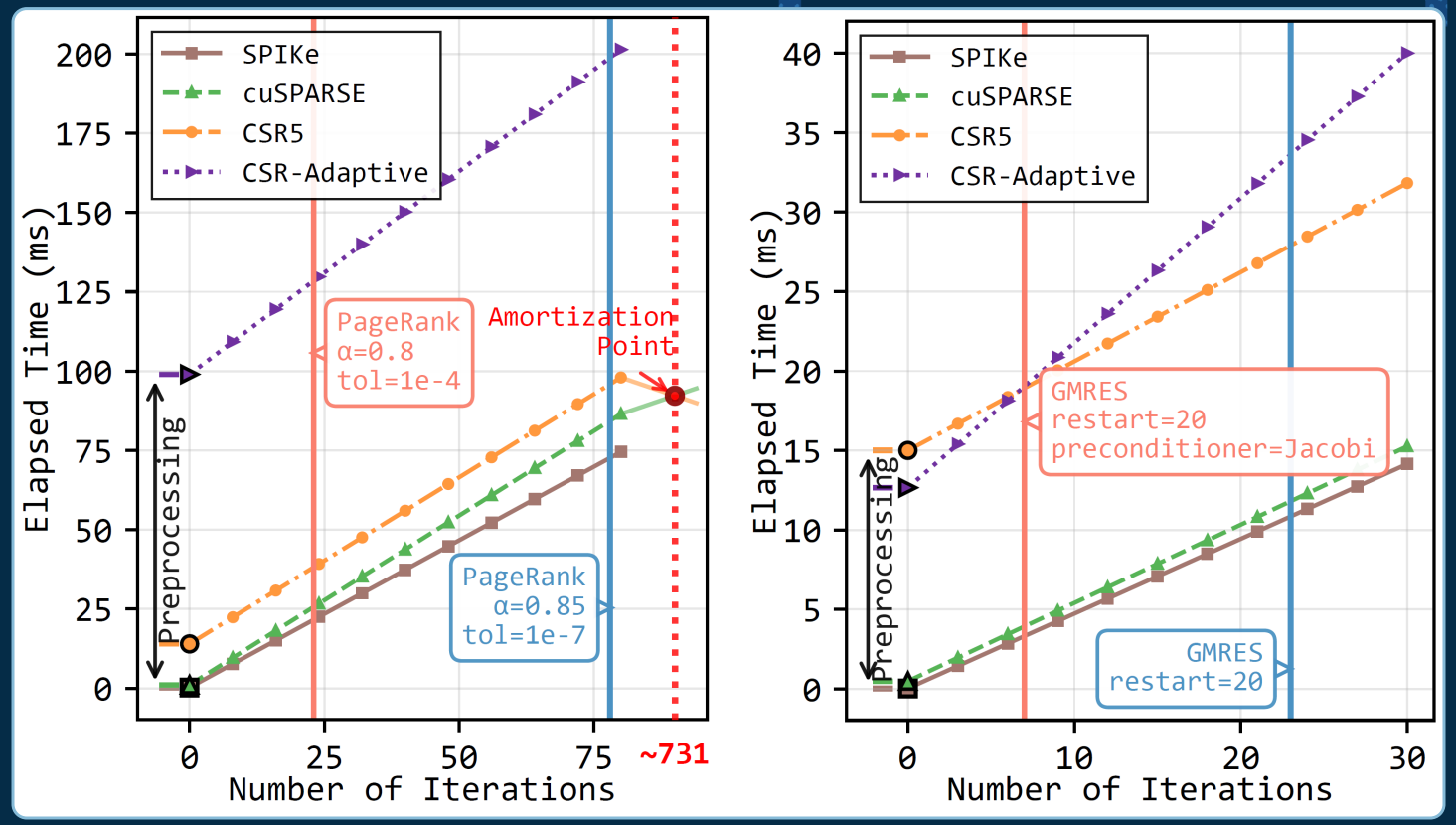
- Algorithmic advances actively reduce required iteration counts (e.g. Advanced preconditioners)
- Well-conditioned linear systems and approximate PageRank may also finish quickly
- ***Iteration counts algorithmically decrease***

**More preprocessing requires more amortization—**
**but fewer iterations leave less time to pay it back.**



SC26

# The preprocessing overhead wall




$$T_{e2e} = T_{pre} + N_{iter} \times T_{spmv} + T_{other}$$

**23 vs. ~731**

PageRank: 23 iterations to converge **vs.** ~731 for CSR5 to beat cuSPARSE

**7 iterations**

Jacobi-preconditioned GMRES convergence

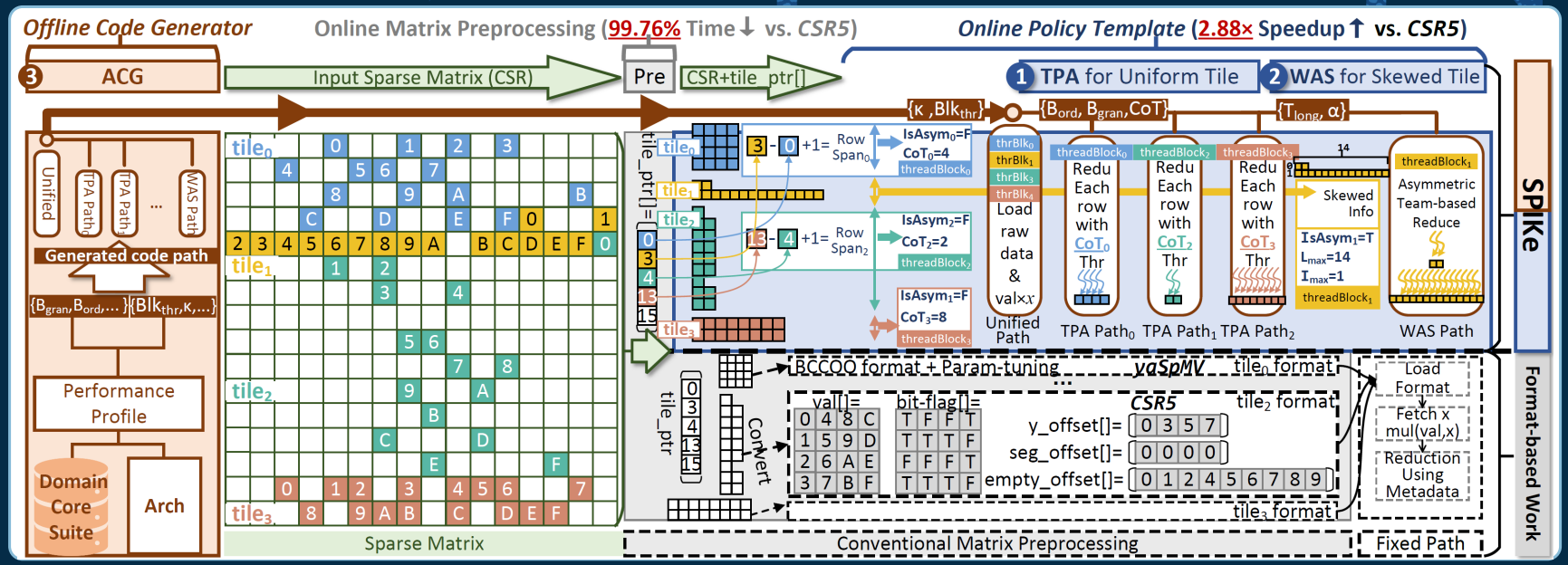
**CSR5 vs. Merge-SpMV: 1.11–1.35× kernel gains require 155–913 iterations to break even**

H100 examples: PageRank on europe_osm; GMRES on cage15. The Fig. 1(a) elapsed-time crossover is schematic.



SC26

# Observation 1: preprocessing is tile-to-code-path adaptation




- Equal-nonzero tiles still contain different sparse structures
- Conventional nnz-split methods (CSR5, YaSpMV) standardize tiles to a format for a fixed code path
- SPIKe dynamically matches raw CSR tiles to specialized code paths

**Tile analysis runs inside SpMV--> Key challenge: lightweight tile-pattern analysis at runtime.**

# Observation 2: matrices from one application domain share tile distributions

**≥ 128 rows**

Road-domain tile spans

**≈ 1 row**

Biology-domain tile spans

- **Tile distributions show intra-domain clustering and inter-domain divergence**
- Matrices in one domain often share tile-structure distributions
- Similar distributions favor similar code configurations on the same GPU
- Distributions diverge across application domains

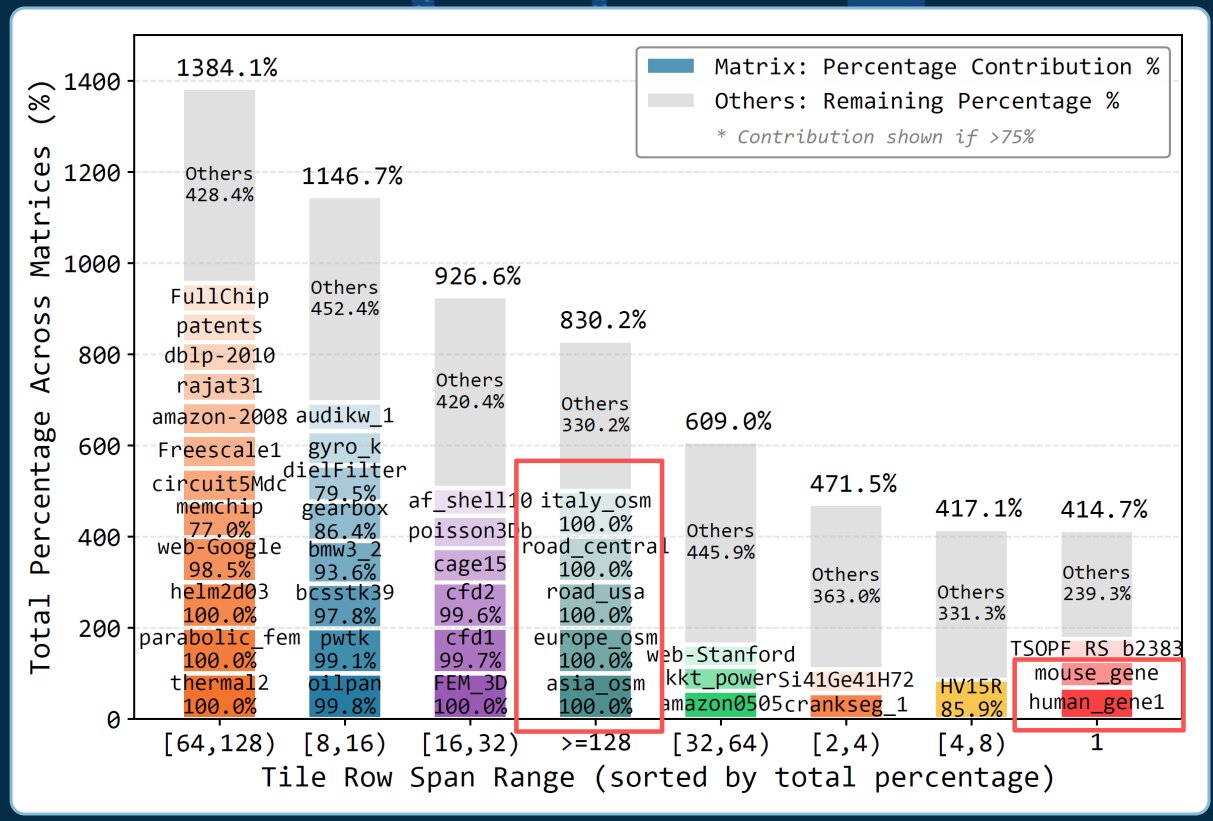



Figure example: 512 nonzeros per tile



SC26

# Two observations lead to SPIKe's two components

**OBSERVATION 1**

Preprocessing performs runtime tile-to-code-path adaptation, but per-tile analysis need to stay lightweight.

**OBSERVATION 2**

Same-domain matrices have similar tile distributions and therefore similar configuration preferences on a fixed GPU.

**ONLINE POLICY TEMPLATE**

TPA: Thread-Level Parallelism Adaptation
WAS: Warp-Level Asymmetric Scheduling

instantiates

**OFFLINE CODE GENERATOR**

ACG searches representative matrices and instantiates one configuration for each architecture-domain pair.

**Before deployment, ACG instantiates/tunes the TPA/WAS branches and parameters.
Inside SpMV, the Online Policy Template selects a path for each raw CSR tile.**

Per-matrix online preprocessing only constructs tile_ptr; ACG remains outside the application's critical path.

# Online Policy Template: TPA routes uniform tiles by row span

Thread-Level Parallelism Adaptation (TPA)

- Row span = rows covered by one fixed-size nonzero tile; It is a lightweight proxy for average row length in CSR
- Small span → more cooperative threads per row
- Large span → more inter-row parallelism
- All threads in a block take the same generated branch

**Algorithm 1:** SPIKe Overview

// **Offline Stage**
// ACG(Arch, Domain) $\Rightarrow \{Blk_{\text{thr}}, \kappa, B_{\text{gran}}, B_{\text{ord}}, \alpha, T_{long}\}$
**Input :** $x[]$, CSR{row_ptr[], val[], col_idx[]}
**Output:** $y[]$
1 __shared__ s_mid$[Tile_{\text{nnz}}]$; // $Tile_{\text{nnz}} \leftarrow Blk_{\text{thr}} \times \kappa$
2 $tile_start_nnz \leftarrow Tile_{\text{nnz}} \times blockIdx.x$;
/* Unified Loading & Multiplication Block
3 **for** $i \leftarrow 0$ **to** $\kappa$ **do**
4 $\quad lid \leftarrow i \times Blk_{\text{thr}} + threadIdx.x$;
5 $\quad gid \leftarrow tile_start_nnz + lid$;
6 $\quad$ s_mid$[lid] \leftarrow$ val$[gid] \times x$[col_idx$[gid]$];
7 __syncthreads();
8 **if** *tile is uniform* **then**
9 $\quad \langle \mathcal{T}_{\text{TPA}}^{B_{\text{gran}}, B_{\text{ord}}} \rangle$(row_ptr, s_mid, $y$) // **TPA Policy** Alg.2
10 **if** *tile is skewed* **then**
11 $\quad \langle \mathcal{T}_{\text{WAS}}^{\alpha, T_{long}} \rangle$(row_ptr, s_mid, $y$) // **WAS Policy** Alg.3

**Compile-time thread counts enable unrolling and fixed shuffle offsets.**

Algorithm 2 example: 512 nonzeros per tile and 128 threads per block. Generated configurations may differ.



SC26

# Online Policy Template: TPA routes uniform tiles by row span

## Thread-Level Parallelism Adaptation (TPA)

- Row span = rows covered by one fixed-size nonzero tile; It is a lightweight proxy for average row length in CSR
- Small span → more cooperative threads per row
- Large span → more inter-row parallelism
- All threads in a block take the same generated branch

**Algorithm 2:** TPA policy template $\mathcal{T}_{\text{TPA}}$ instance

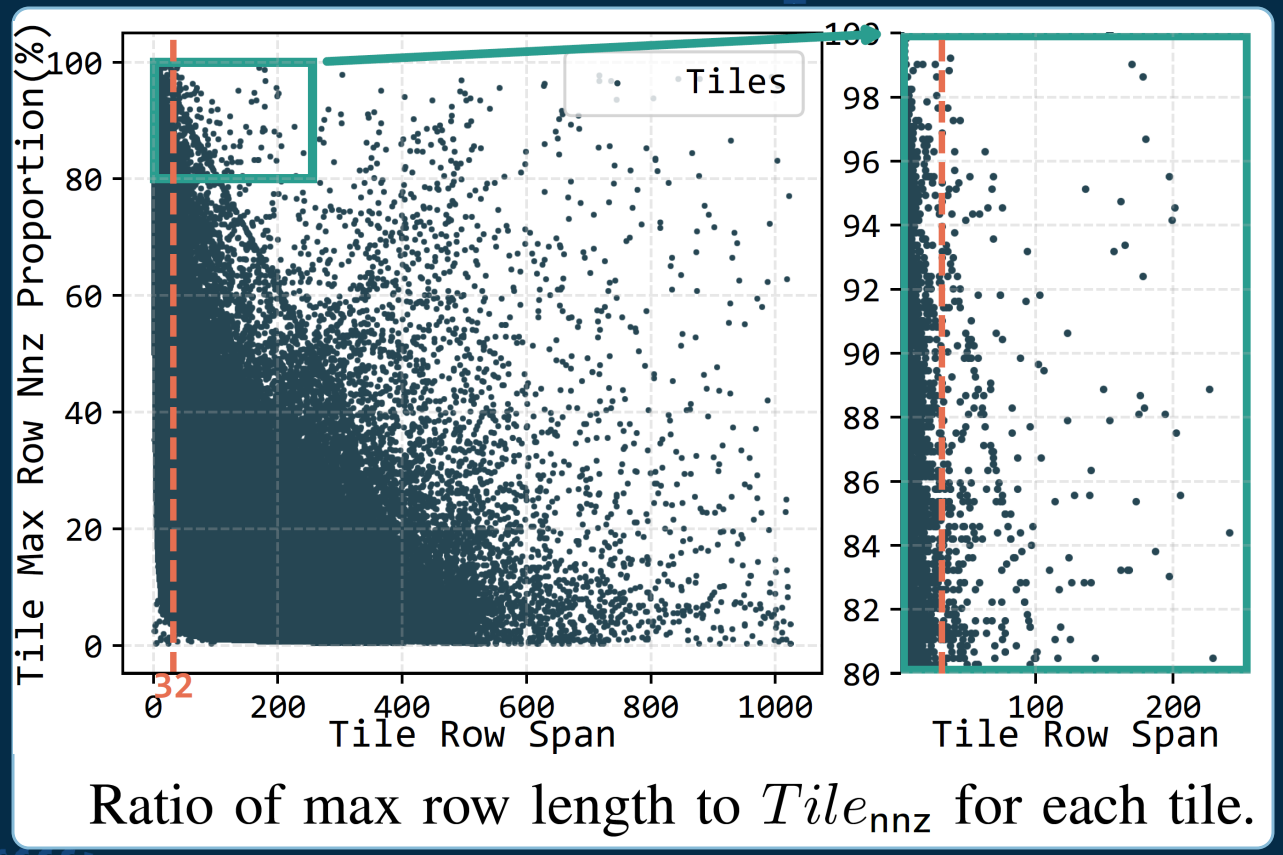
**Input :** row_ptr[], tile_ptr[], s_mid[]
**Output:** $y$[]
// **Offline Config:**
ACG(Arch, Domain) $\Rightarrow Blk_{\text{thr}} = 128, \kappa = 4, B_{\text{gran}}, B_{\text{ord}}$
// **Online Execution:** Launched $Blk_{\text{thr}}$ threads per block
// Alg.1 Unified Loading & Multiplication Block
1 `rs` $\leftarrow$ tile_ptr$[blockIdx.x + 1]$ − tile_ptr$[blockIdx.x]$; // row_span
// Generated `REDUCE_BLOCK`: if-else chain ordered by Fig. 3
2 **if** `rs` $\in [64, 128)$ ***[[likely]]*** **then**
`reduceRowCodePath`$\langle \text{CoT} = 2 \rangle$(row_ptr, s_mid, $y$) ;
3 **else if** `rs` $\in [8, 16)$ **then** `reduceRowCodePath`$\langle \text{CoT} = 16 \rangle(\dots)$ ;
4 **else if** `rs` $\in [16, 32)$ **then** `reduceRowCodePath`$\langle \text{CoT} = 8 \rangle(\dots)$ ;
5 **else if** `rs` $\geq 128$ **then** `reduceRowCodePath`$\langle \text{CoT} = 1 \rangle(\dots)$ ;
// ... other customized branches omitted for brevity ...
6 **else if** `rs` $= 1$ **then** `reduceRowCodePath`$\langle \text{CoT} = 128 \rangle(\dots)$ ;

**Compile-time thread counts enable unrolling and fixed shuffle offsets.**

Algorithm 2 example: 512 nonzeros per tile and 128 threads per block. Generated configurations may differ.

# Online Policy Template: WAS detects skewed tiles with one warp

Warp-Level Asymmetric Scheduling (WAS)




Ratio of max row length to $Tile_{nnz}$ for each tile.

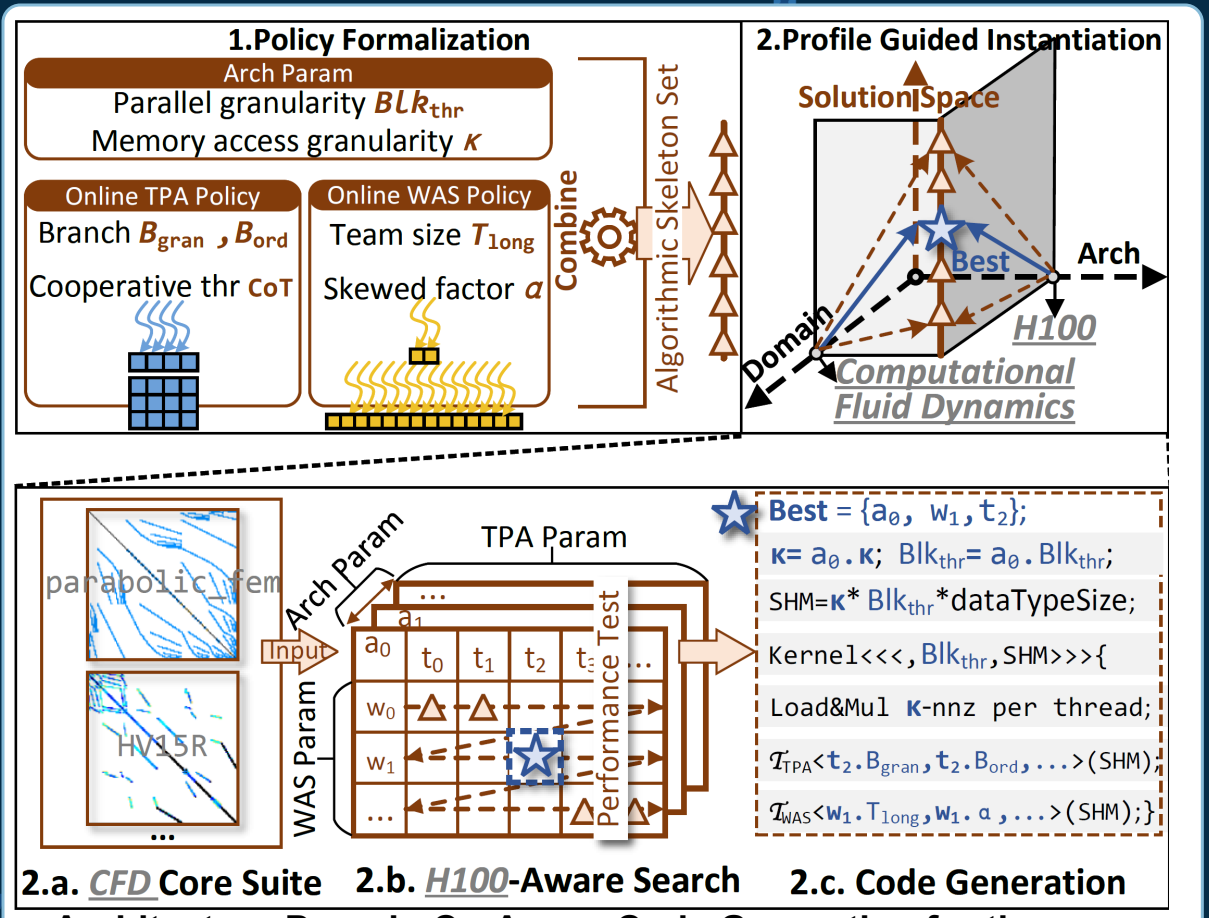
**99.91%**

of observed highly skewed tiles span at most 32 rows

- For row spans ≤ 32, the first warp reads row lengths and finds the maximum
- Warp shuffles aggregate the longest row's length and index
- Most threads reduce the long row; remaining threads process short rows

Observed on Table III matrices with 512 nonzeros/tile and longest-row fraction >80%.

# Offline Code Generator: ACG instantiates the online templates and derives a shared domain configuration




**Architecture-Domain Co-Aware Code Generation for the Computational Fluid Dynamics (CFD) domain on H100.**

1. **Formalize**
   Branch ranges, thread counts, skew thresholds, block sizes
2. **Measure across the domain**
   K-means representatives + simulated annealing on the target GPU
3. **Generate**
   Jinja2 embeds selected parameters as source-code constants

**CFD example: 5 matrices from a 174-matrix domain; ~2 min profiling on H100**

The ~2 min cost is a reported CFD example—not a fixed cost for very domain or architecture.

# Evaluation setup and performance

**2,294**

SuiteSparse matrices
FP64 kernel evaluation

**H100 + A100**

80 GB GPUs
CUDA 12.8

**8 kernel baselines**

CUSP · LightSpMV · Merge-SpMV
CSR-Adaptive · CSR5 · cuSPARSE
DASP · Drawloom

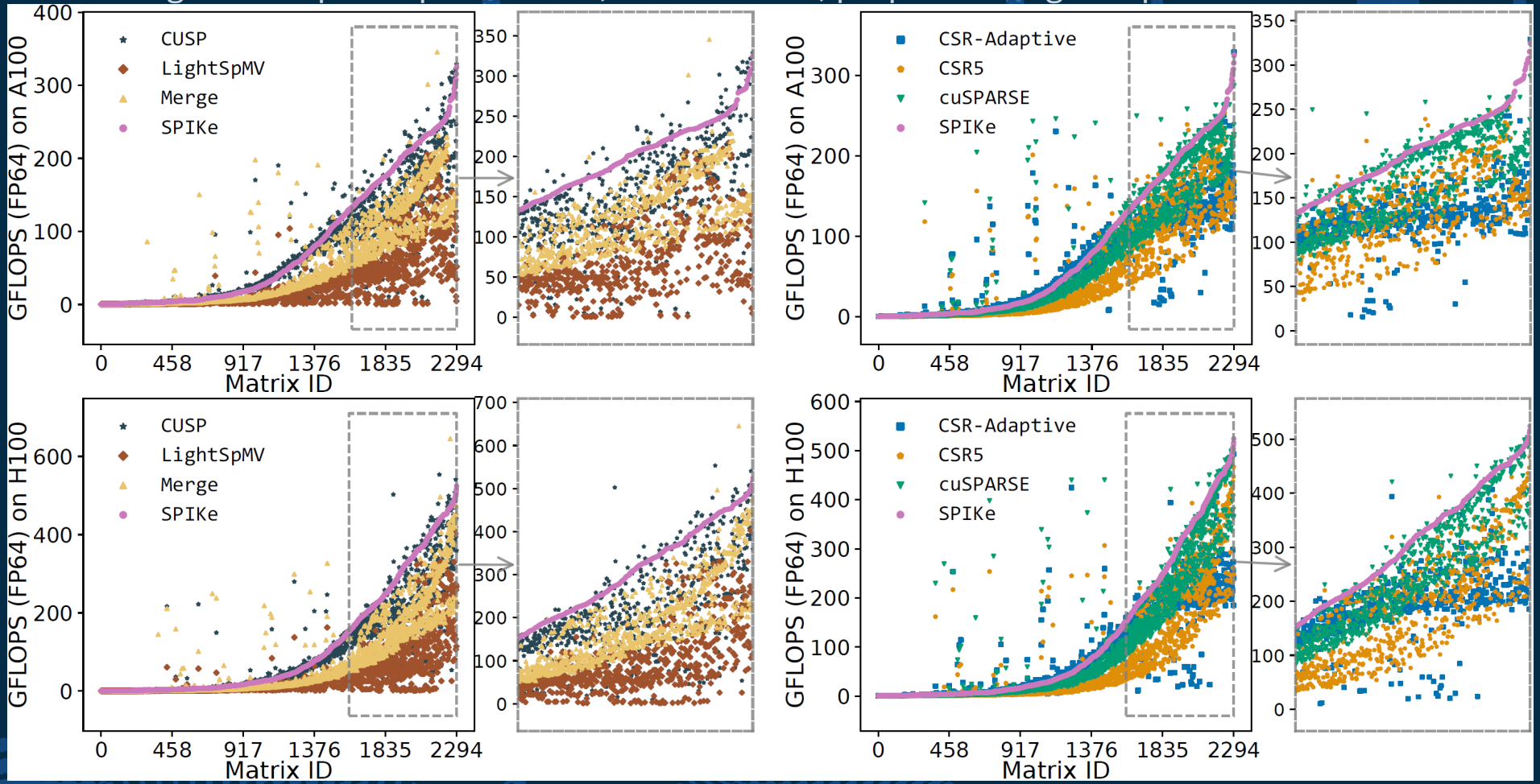
**Applications**

PageRank FP32 + GMRES FP64
on H100

Kernel time includes TPA/WAS. Online Preprocessing only constructs tile_ptr. ACG runs before deployment.

# Average kernel gains over kernel baselines on both GPUs

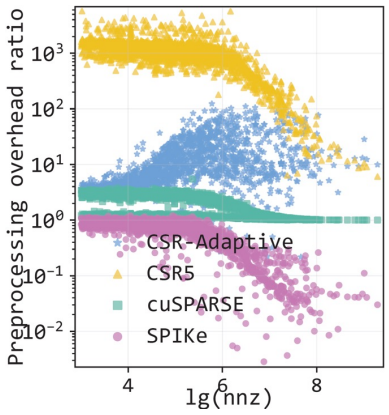
Average FP64 speedups across 2,294 matrices; preprocessing setup time is excluded.



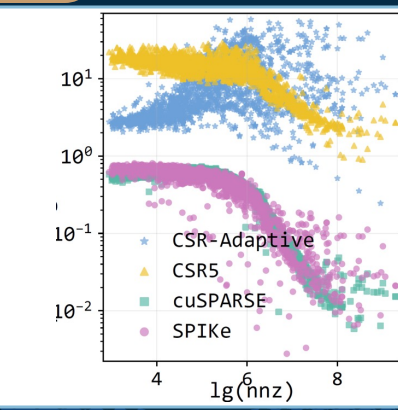

Additional H100 representative-set comparison: 1.32× over DASP and 1.24× over Drawloom (Fig. 11).

# High kernel efficiency with minimal online preprocessing

**H100**




**A100**




Plots show Tpre / Tspmv on a logarithmic scale.

**2.88×**

average H100 kernel speedup over CSR5
2,294 matrices

**99.76%**

lower H100 preprocessing time than CSR5
representative-set comparison

Online setup: lightweight tile_ptr construction only

- No complex tile descriptors
- No data transposition
- No online parameter tuning



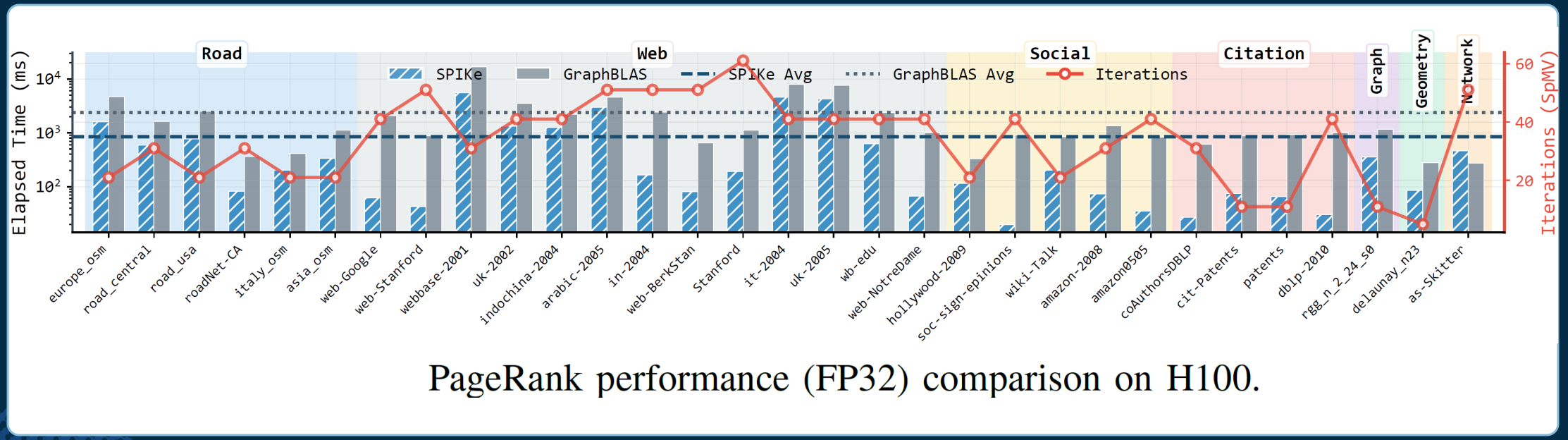
SC26

# PageRank gains persist with only 5 to 61 iterations

**2.78×**
average end-to-end speedup over SuiteSparse GraphBLAS

**5–61**
iterations across tested graph analytics matrices

Damping 0.85
Tolerance 1e-4
PLUS_TIMES semiring



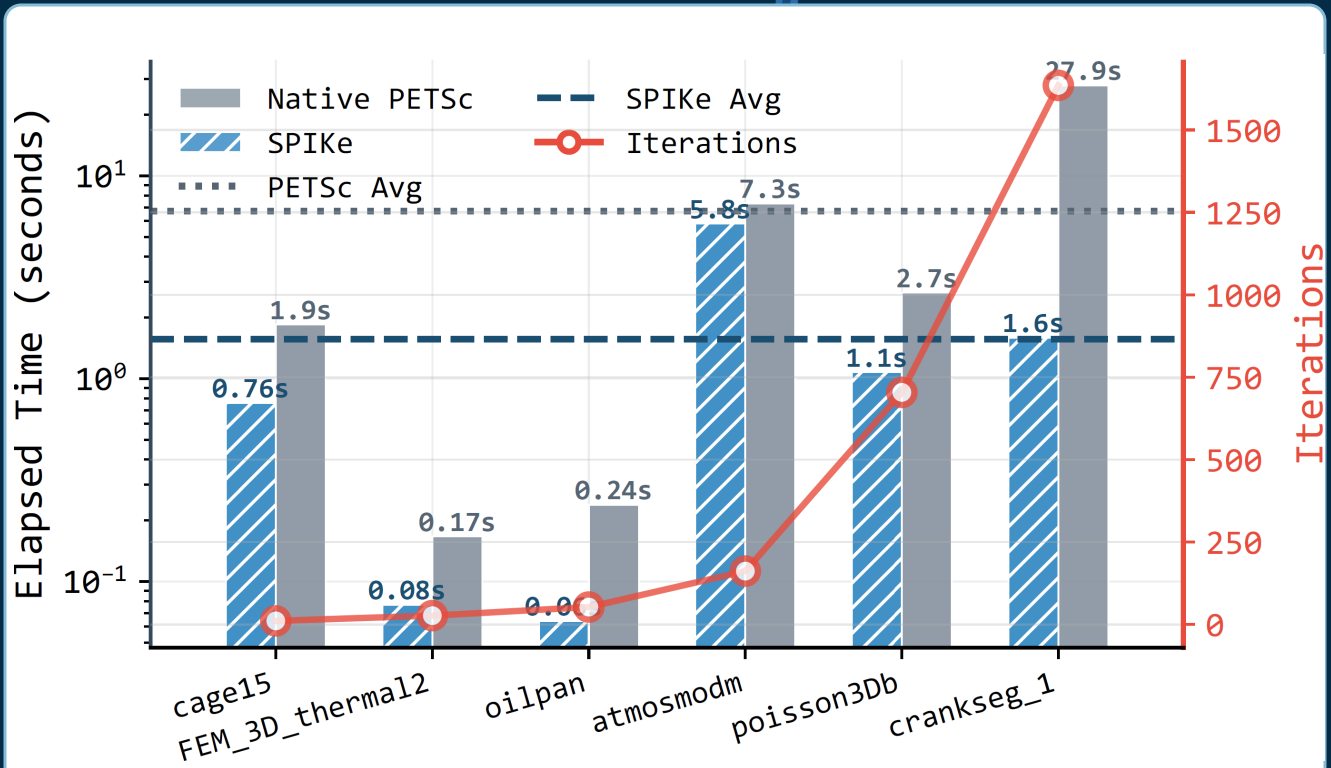

PageRank performance (FP32) comparison on H100.

H100, FP32. Bars show elapsed time; the red line shows iteration counts.



SC26

# GMRES gains span 11 to 1635 iterations




GMRES performance (FP64) comparison on H100.

**4.26×**

average acceleration reported in the paper

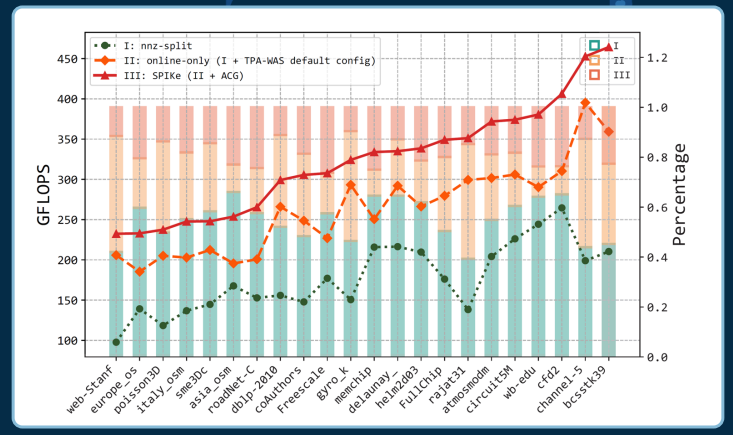
**40.17 s → 9.4 s**

total solve time: native PETSc → SPIKe

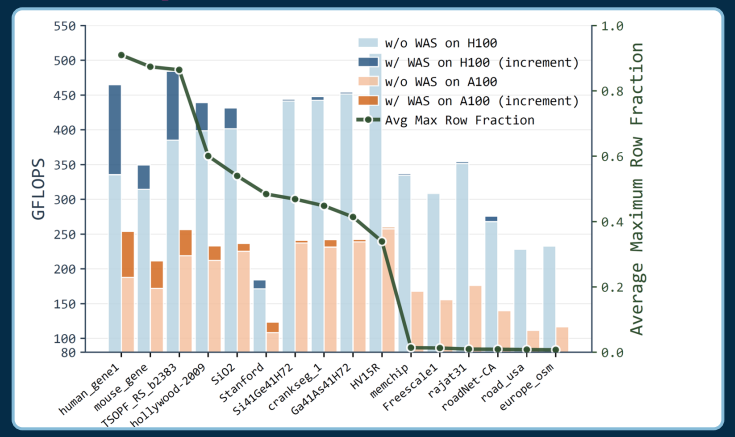
- Six convergent matrices; identical convergence behavior
- Per-matrix speedups: 1.26–17.43×

H100, FP64. PETSc AIJ vs. SPIKe MatShell; Jacobi; restart 30; relative tolerance 1e-6; max 2,000 iterations.

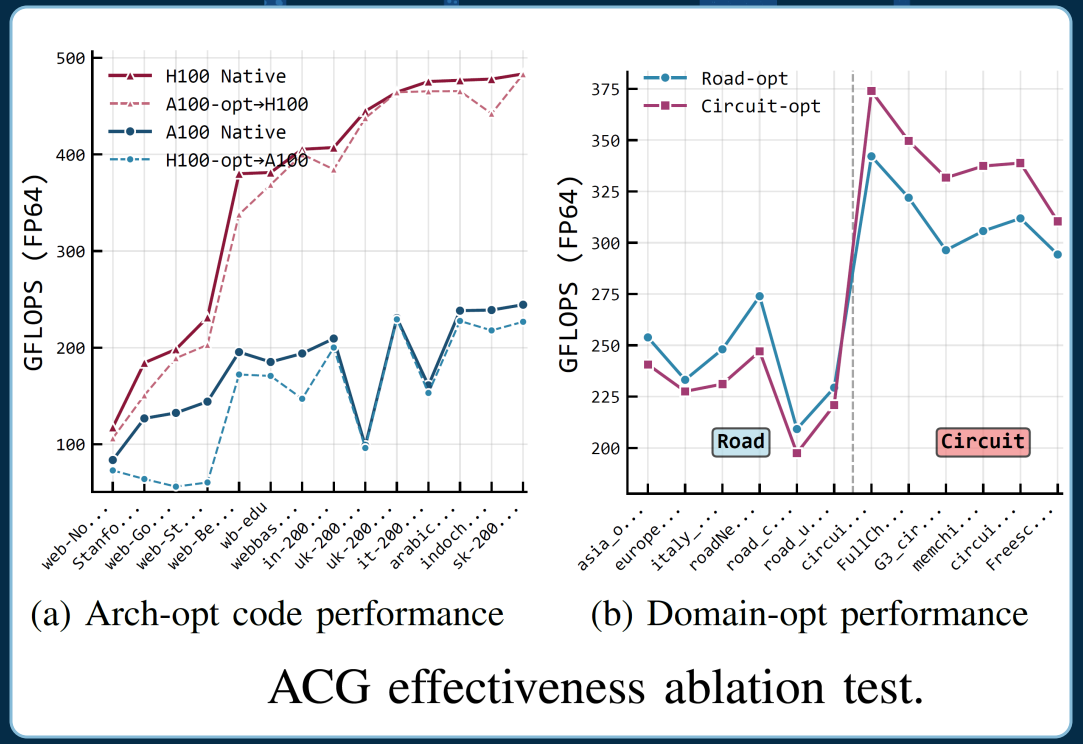
# Ablations isolate online policies and offline generation




**nnz-split → + TPA/WAS → + ACG**




**WAS gains concentrate on matrices with skewed tiles**




**Architecture + domain**

Architecture-optimal parameters differ between A100 and H100
Domain-optimized kernels favor their target sparsity patterns

WAS improves skewed cases such as human_gene1 and gives no improvement on uniform europe_osm (Fig. 15).



SC26

# Removing the high-iteration constraint

**SPIKe liberates preprocessing-intensive sparse matrix computations from the high-iteration constraint.**

### Online Policy Template

TPA + WAS route each raw tile

### Offline Code Generator

ACG instantiates templates and shares one configuration per architecture-domain pair

### Online setup

Construct only tile_ptr

**2.78× PageRank** **4.26× GMRES**

# Thank you.