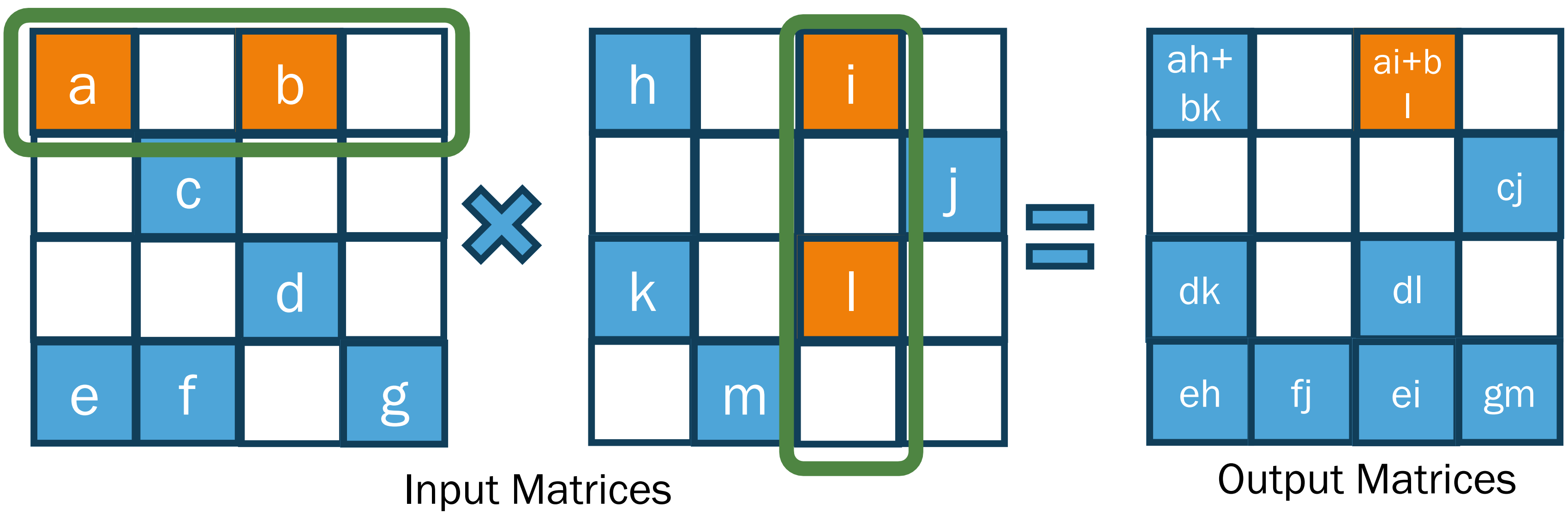
High-Performance Sparse Matrix-Matrix Products on Intel KNL and Multicore Architectures

Yusuke Nagasaka†, Satoshi Matsuoka§†
Ariful Azad‡, Aydın Buluç‡

† Tokyo Institute of Technology
§ Riken Center for Computational Science
‡ Lawrence Berkeley National Laboratory

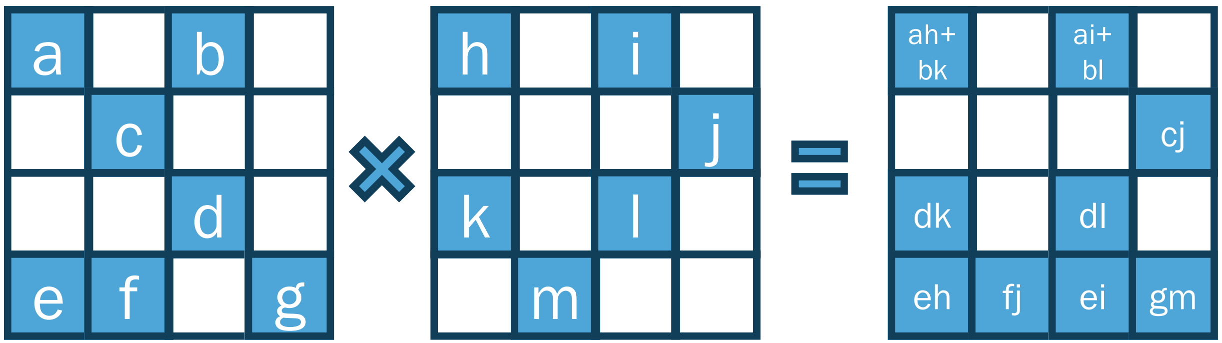
Sparse General Matrix-Matrix Multiplication (SpGEMM)

- Key kernel in graph processing and numerical applications
  - Markov clustering, Betweenness centrality, triangle counting, …
  - Preconditioner for linear solver
    - AMG (Algebraic Multigrid) method
  - **Time-consuming part**



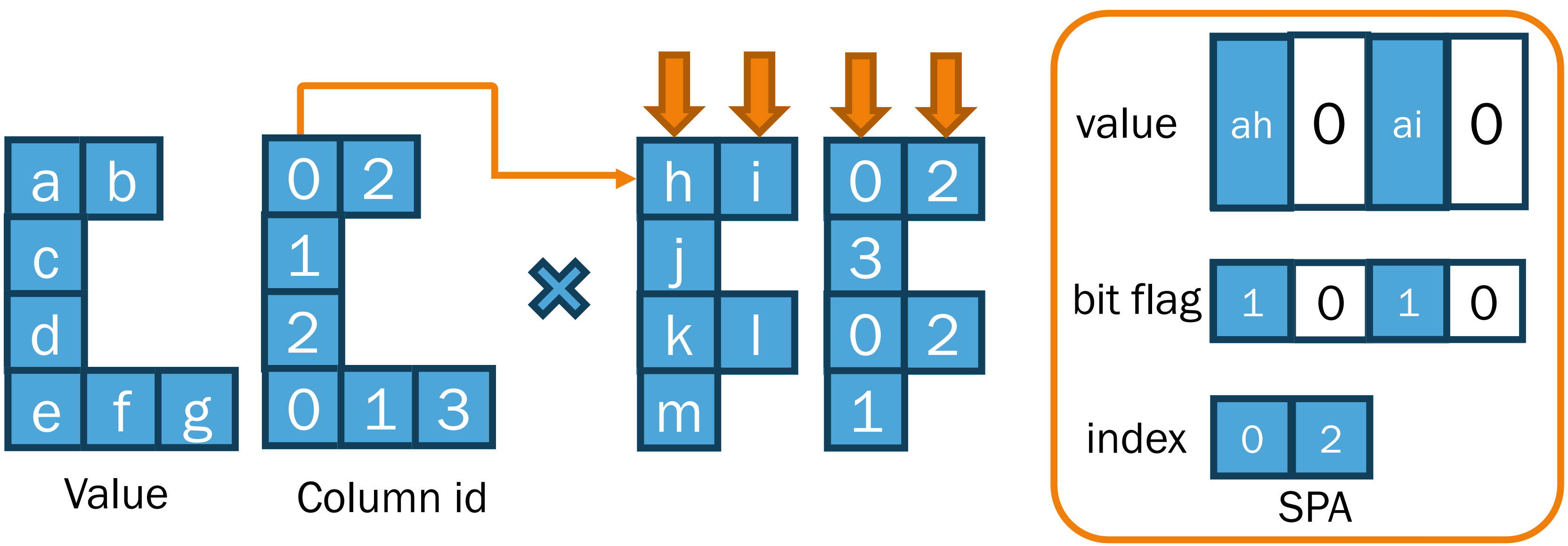
Accumulation of intermediate products

Sparse Accumulator (SPA) [Gilbert, SIAM1992]



Input Matrices

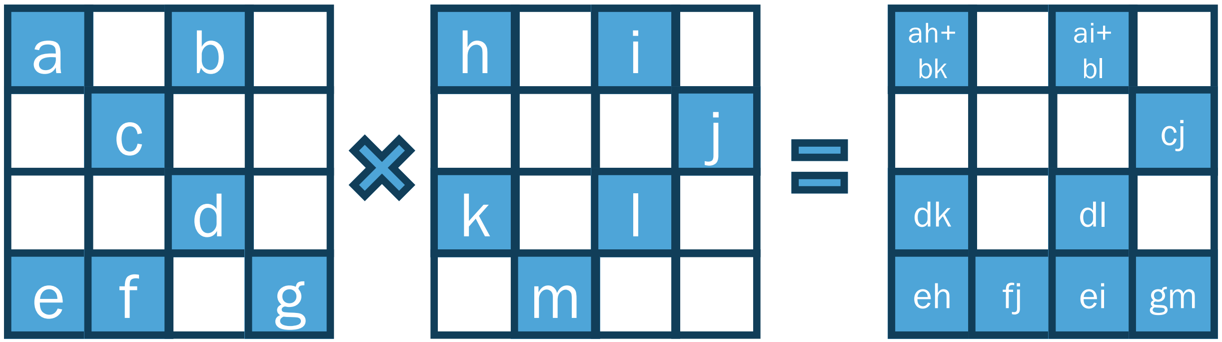
Output Matrices



Input matrices in sparse format

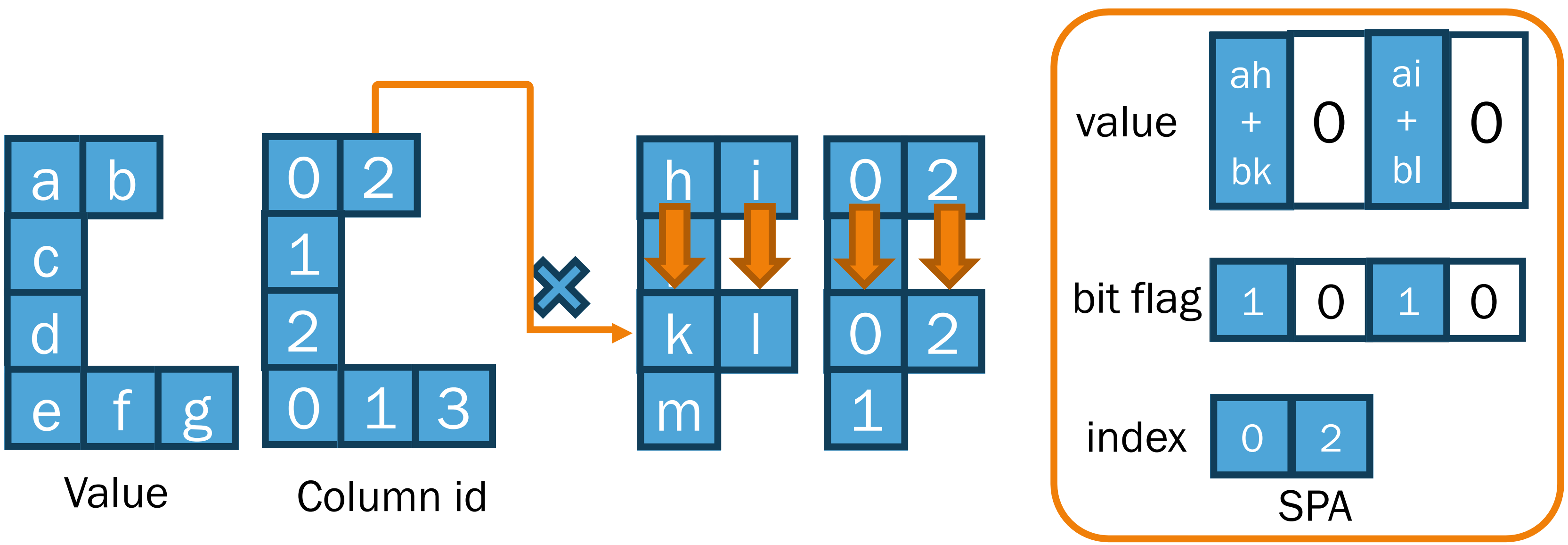
Accumulation of intermediate products

Sparse Accumulator (SPA) [Gilbert, SIAM1992]



Input Matrices

Output Matrices



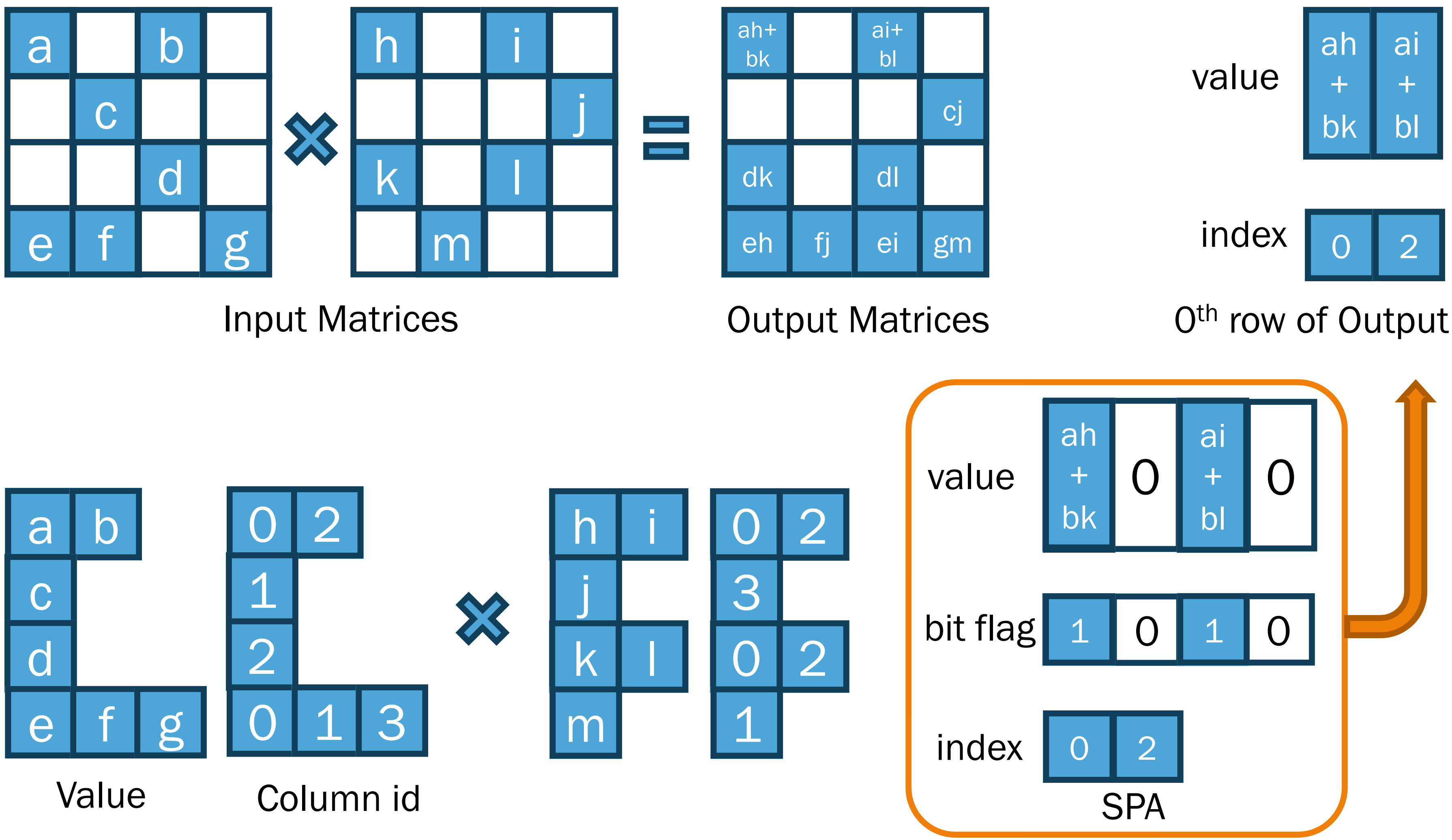
Value

Column id

Input matrices in sparse format

Accumulation of intermediate products

Sparse Accumulator (SPA) [Gilbert, SIAM1992]



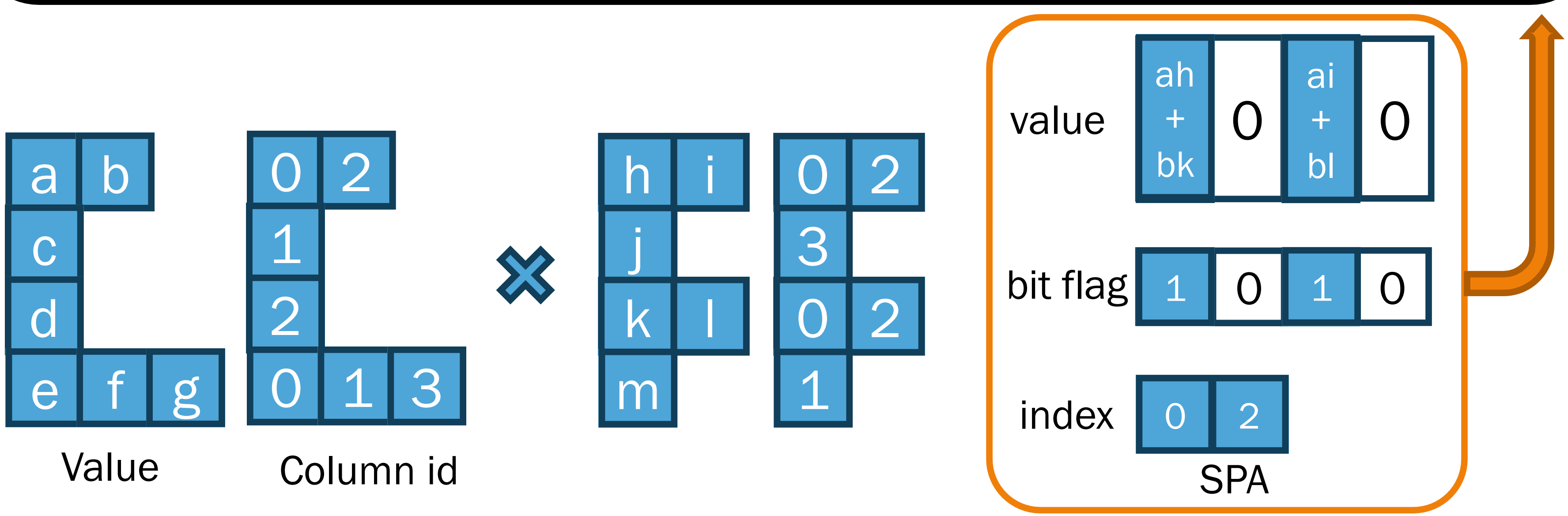
Input matrices in sparse format

Accumulation of intermediate products

Sparse Accumulator (SPA) [Gilbert, SIAM1992]

☺ **Efficient accumulation of intermediate products: Lookup cost is O(1)**

☹ **Requires O(#columns) memory by one thread**



Input matrices in sparse format

Existing approaches for SpGEMM

- Several sequential and parallel SpGEMM algorithms
  - Also packaged in software/libraries

Algorithm (Library)	Accumulator	Sortedness (Input/Output)
MKL	-	Any/Select
MKL-inspector	-	Any/Unsorted
KokkosKernels	HashMap	Any/Unsorted
Heap	Heap	Sorted/Sorte
Hash	Hash Table	Any/Select

Existing approaches for SpGEMM

- Several sequential and parallel SpGEMM algorithms
  - Also packaged in software/libraries

Questions?

(a) What is the best algorithm/implementation **for a problem** at hand?

(b) What is the best algorithm/implementation **for the architecture** to be used in solving the problem?

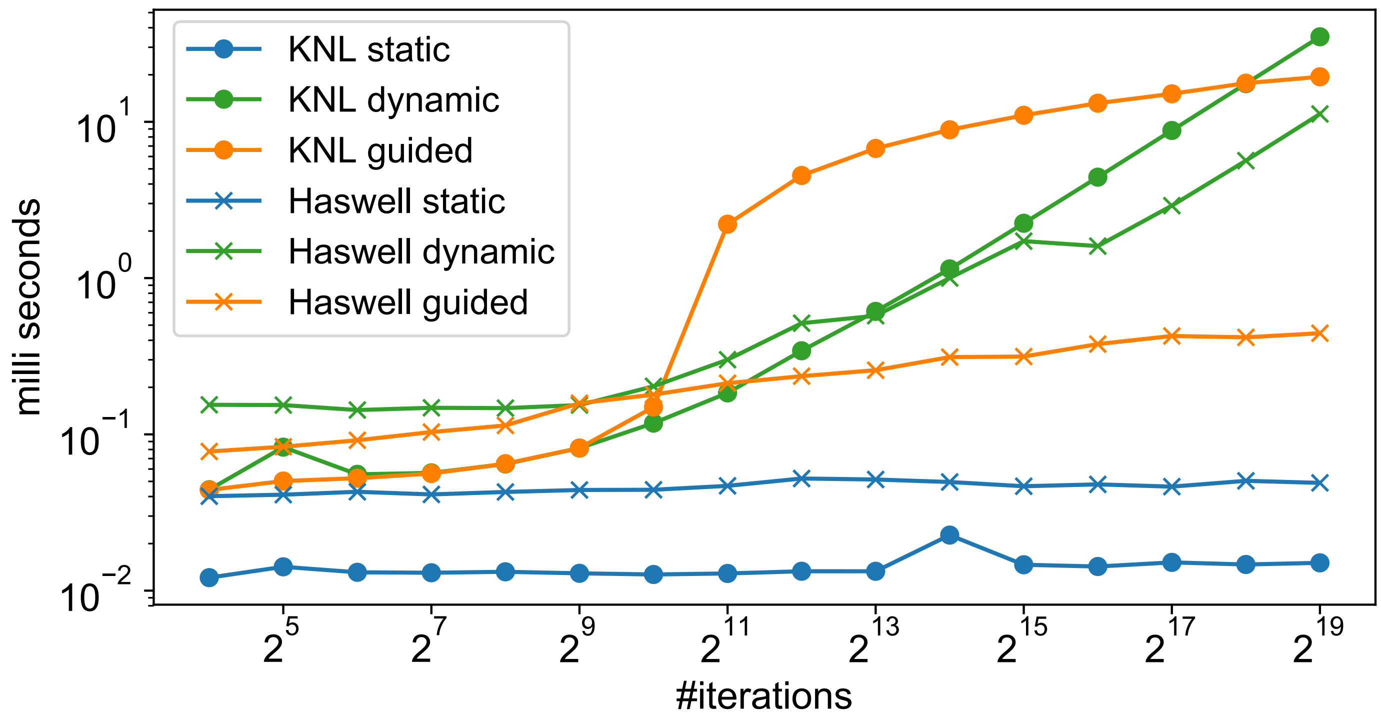
Hash	Hash Table	Any/Select

Contribution

- We characterize, optimize and evaluate existing SpGEMM algorithms for real-world applications on modern Multi-core and Many-core architectures
  - Characterizing the performance of SpGEMM on shared-memory platforms
    - Intel Haswell and Intel KNL architectures
    - **Identify bottlenecks and mitigate them**
  - Evaluation including several use cases
    - A^2, Square x Tall-skinny, L*U for triangle counting
  - Showing the **impact of keeping unsorted output**
  - **A recipe for selecting the best-performing algorithm for a specific application scenario**

Benchmark for SpGEMM
Thread scheduling cost

- Evaluates the scheduling cost on Haswell and KNL architectures
  - OpenMP: static, dynamic and guided
- **Scheduling cost hurts the SpGEMM performance**

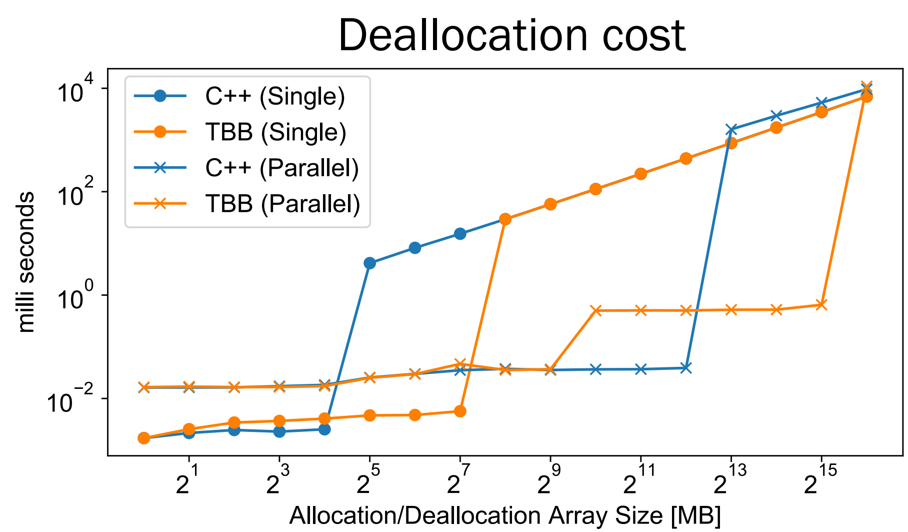


Benchmark for SpGEMM
Memory allocation/deallocation cost

- Identifies that **allocation/deallocation of large memory space is expensive**
- Parallel memory allocation scheme
  - Each thread independently allocates/deallocates memory and accesses only its own memory space
  - **For SpGEMM, we can reduce deallocation cost**

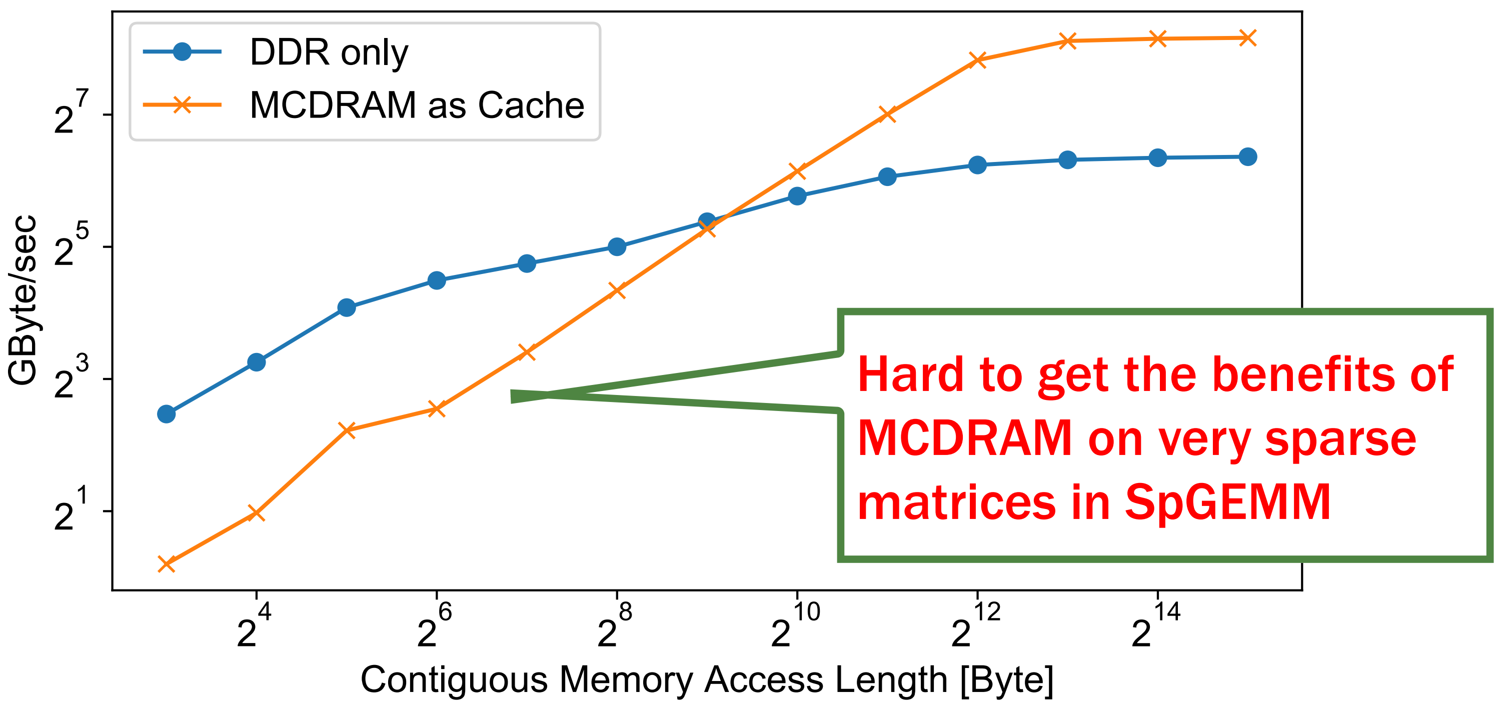
Parallel memory allocation

$eachN \leftarrow N / nthreads$
$\textsc{Allocate}(a, nthreads)$
for $tid \leftarrow$ **to** $nthreads$ **in parallel**
  do $\textsc{Allocate}(a[tid], eachN)$
  do for $i \leftarrow$ **to** $eachN$
    do $a[tid][i] \leftarrow i$
  do $\textsc{Deallocate}(a[tid], eachN)$
$\textsc{Deallocate}(a[tnum])$



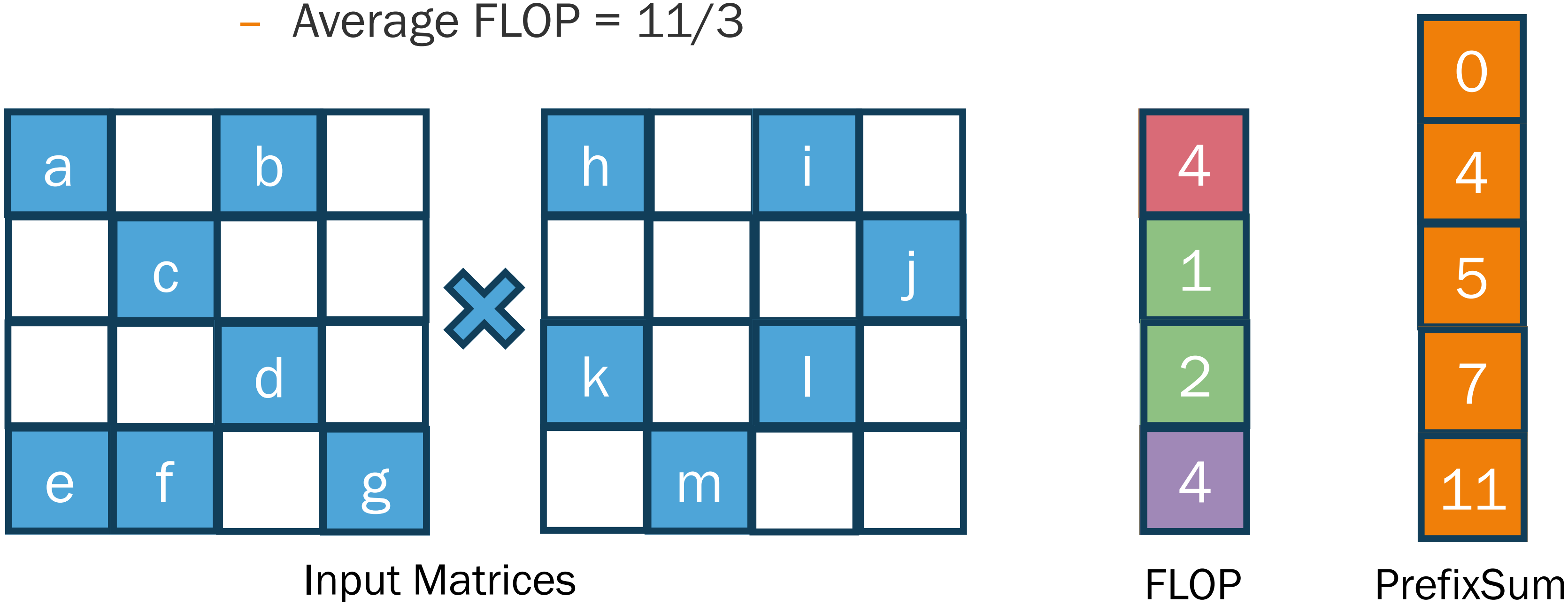
Benchmark for SpGEMM
Impact of MCDRAM

- MCDRAM provides high memory bandwidth
  - Obviously **improves stream benchmark**
  - Performance of stanza-like memory access is **unclear**
    - Small blocks of consecutive elements
    - Access to rows of B in SpGEMM



Architecture Specific Optimization
Thread scheduling

- **Good load-balance with static scheduling**
  - Assigning work to threads by FLOP
  - Work assignment can be efficiently executed in parallel
    - Counting required FLOP of each row
    - PrefixSum to get total FLOP of SpGEMM
    - Assigning rows to thread (Eg. shows the case of 3 threads)
      - Average FLOP = 11/3

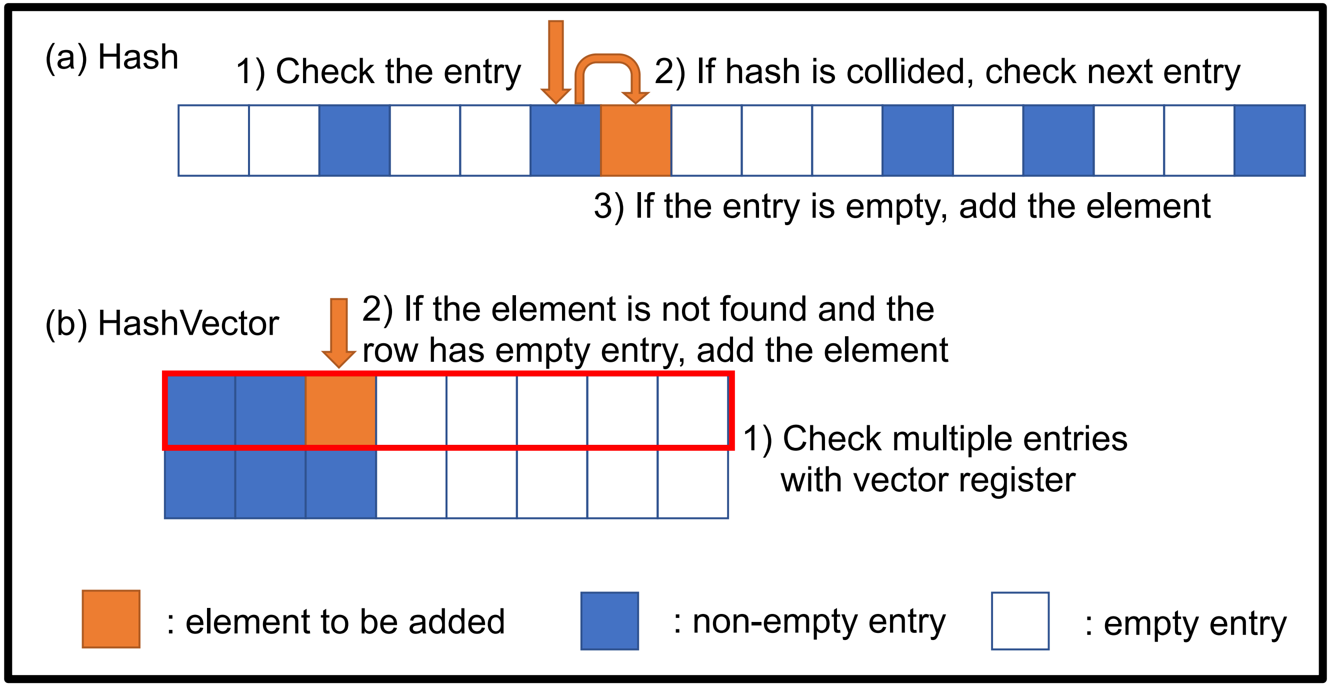


Architecture Specific Optimization
Accumulator for Symbolic and Numeric Phases

- Optimizing algorithms for Intel architectures
- Heap [Azad, 2016]
  - Priority queue indexed by column indices
  - **Requires logarithmic time to extract elements**
  - **Space efficient**: $O(nnz(a_{i*}))$
    - Better cache utilization
- Hash [Nagasaka, 2016]
  - Uses hash table for accumulator, based on GPU work
    - **Low memory usage and high performance**
  - Each thread once allocates the hash table and reuses it
  - **Extended to HashVector to exploit wide vector register**

Architecture Specific Optimization
HashVector

- Utilizing 256 and 512-bit wide vector register of Intel architectures for hash probing
  - **Reduces the number of probing caused by hash collision**
  - Requires a few more instructions for each check
    - **Degrades the performance when the collisions in Hash are rare**



Performance Evaluation

Matrix Data

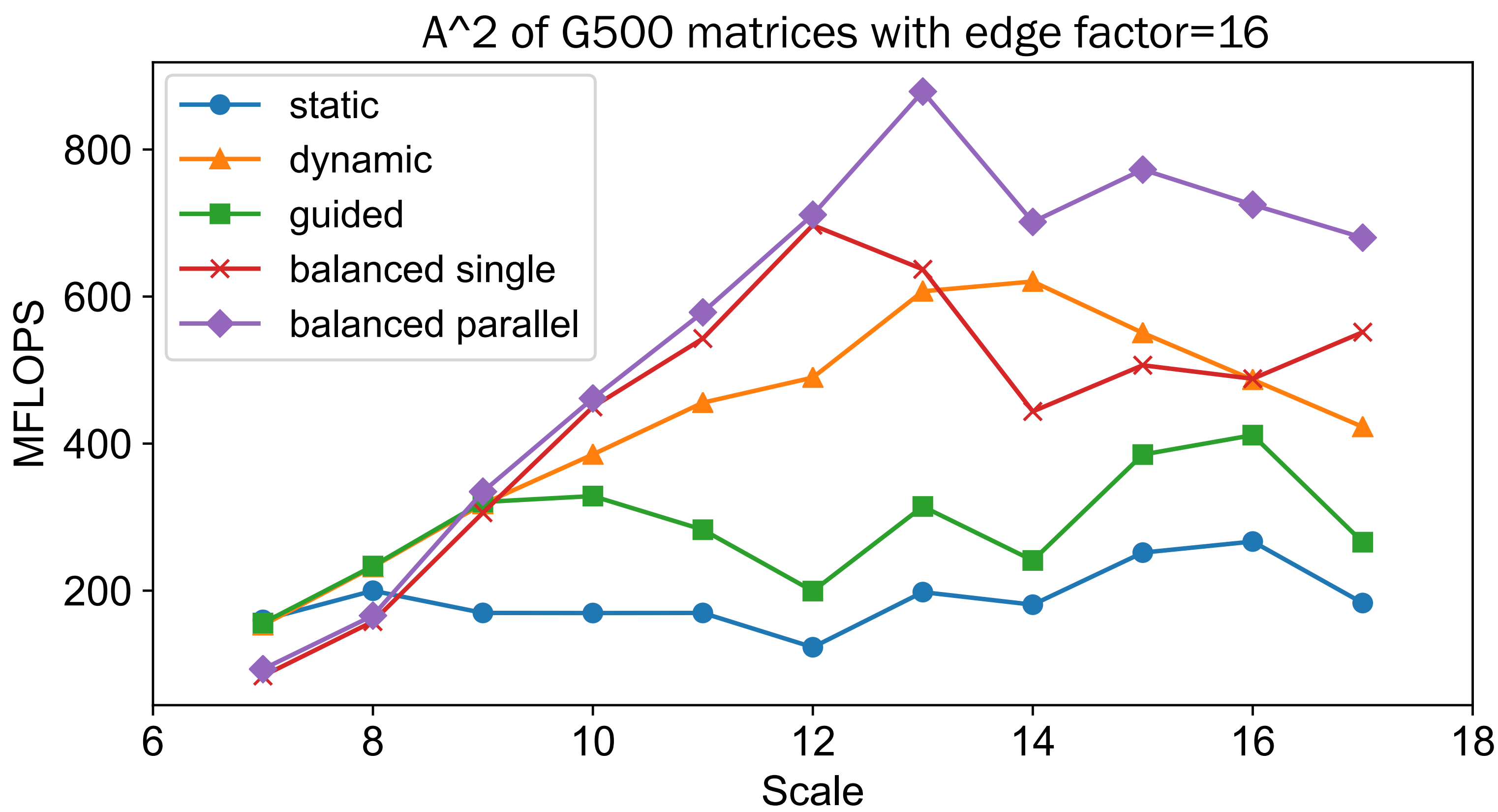
- Synthetic matrix
  - R-MAT, the recursive matrix generator
  - Two different non-zero patterns of synthetic matrices
    - **ER**: Erdős–Rényi random graphs
    - **G500**: Graphs with power-law degree distributions
      - Used for Graph500 benchmark
  - Scale ***n*** matrix: 2^n-by-2^n
  - ***Edge factor***: the average number of non-zero elements per row of the matrix
- SuiteSparse Matrix Collection
  - 26 sparse matrices used in several past work

Evaluation Environment

- **Cori system @NERSC**
  - Haswell Cluster
    - Intel Xeon Processor E5-2698 v3
    - 128GB DDR4 memory
  - KNL Cluster
    - Intel Xeon Phi Processor 7250
      - 68 cores
      - 32KB/core L1 cache, 1MB/tile L2 cache
      - 16GB MCDRAM
      - Quadrant, cache
    - 96GB DDR4 memory
  - OS: SuSE Linux Enterprise Server 12 SP3
  - Intel C++ Compiler (icpc) ver18.0.0
    - -g –O3 -qopenmp

Benefit of Performance Optimization
Scheduling and memory allocation

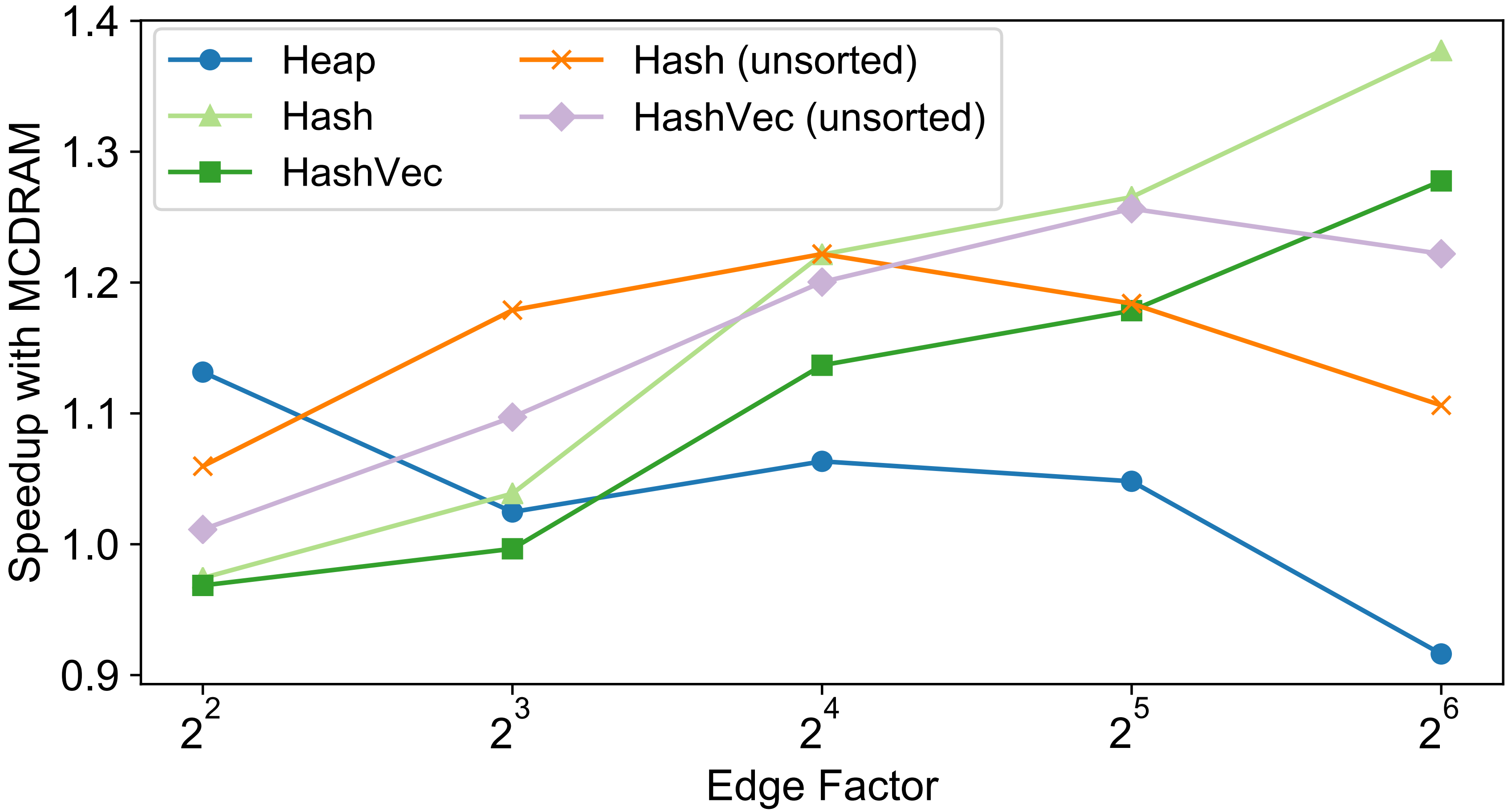
- **Good load balance** with static scheduling
- For larger matrices, parallel memory allocation scheme keeps high performance



Benefit of Performance Optimization

Use of MCDRAM

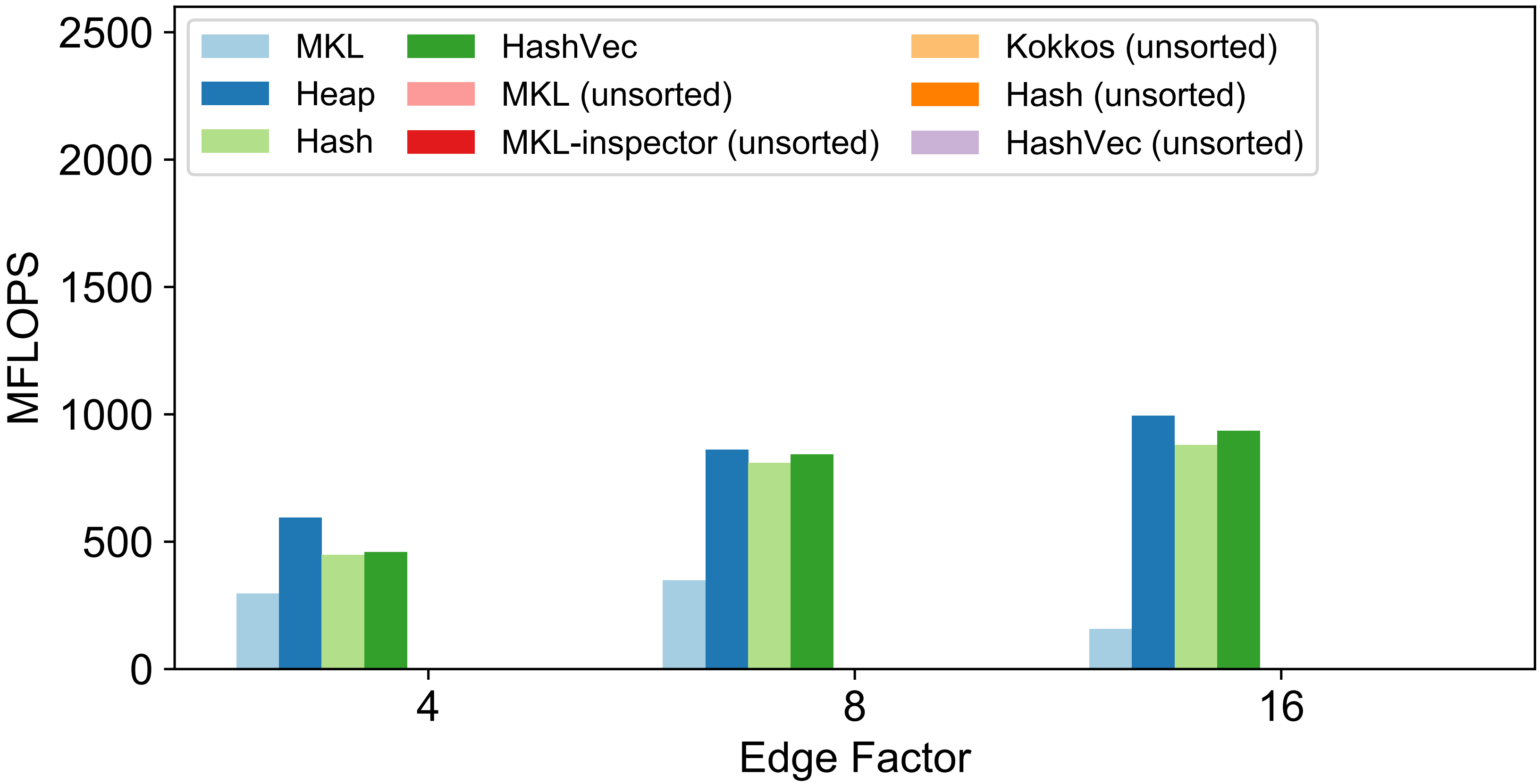
- **Benefit of MCDRAM especially on denser matrices**



Performance Evaluation

A^2: Scaling with density (KNL, ER)

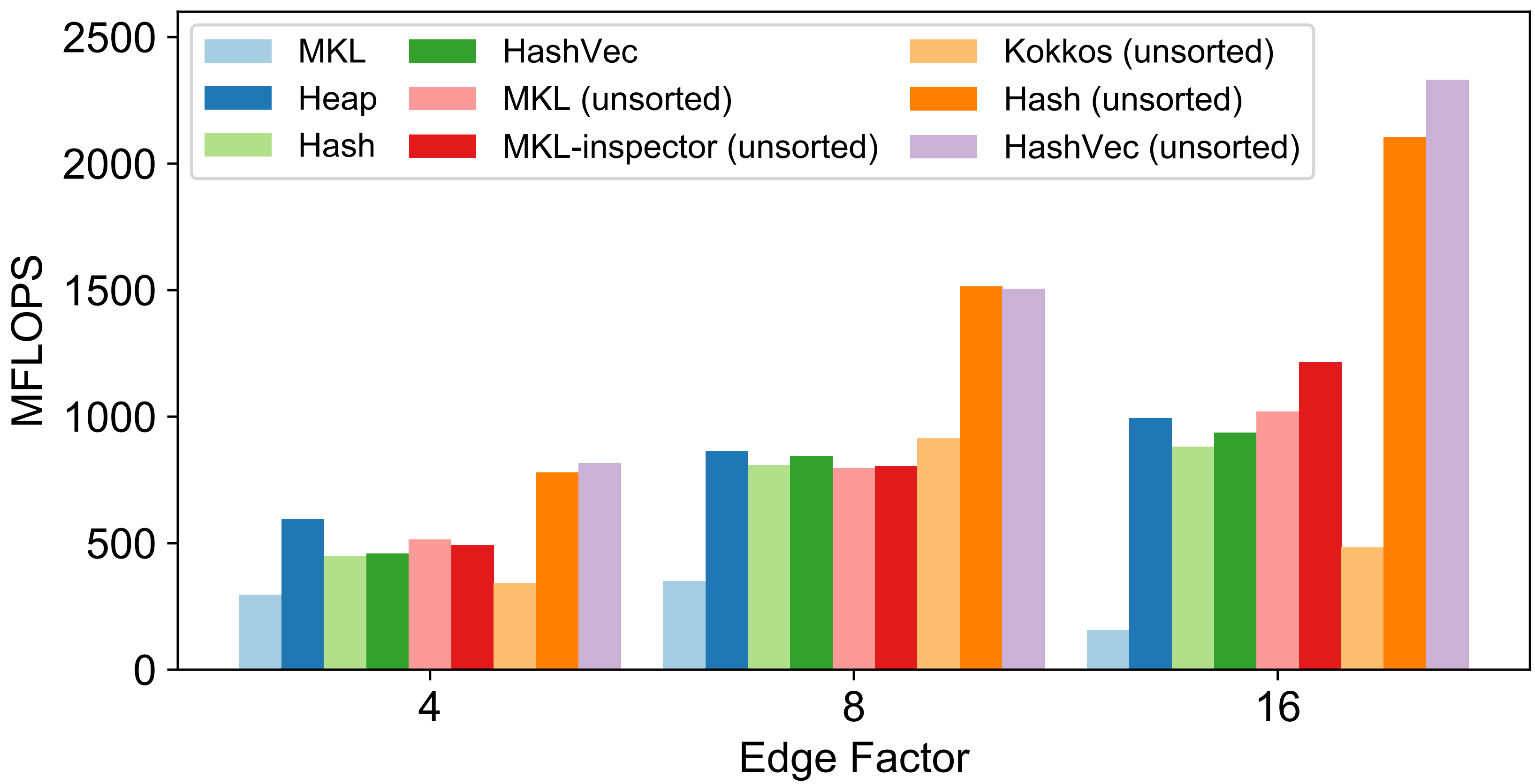
- Scale = 16
- Different performance trends
  - **Performance of MKL degrades with increasing density**



Performance Evaluation

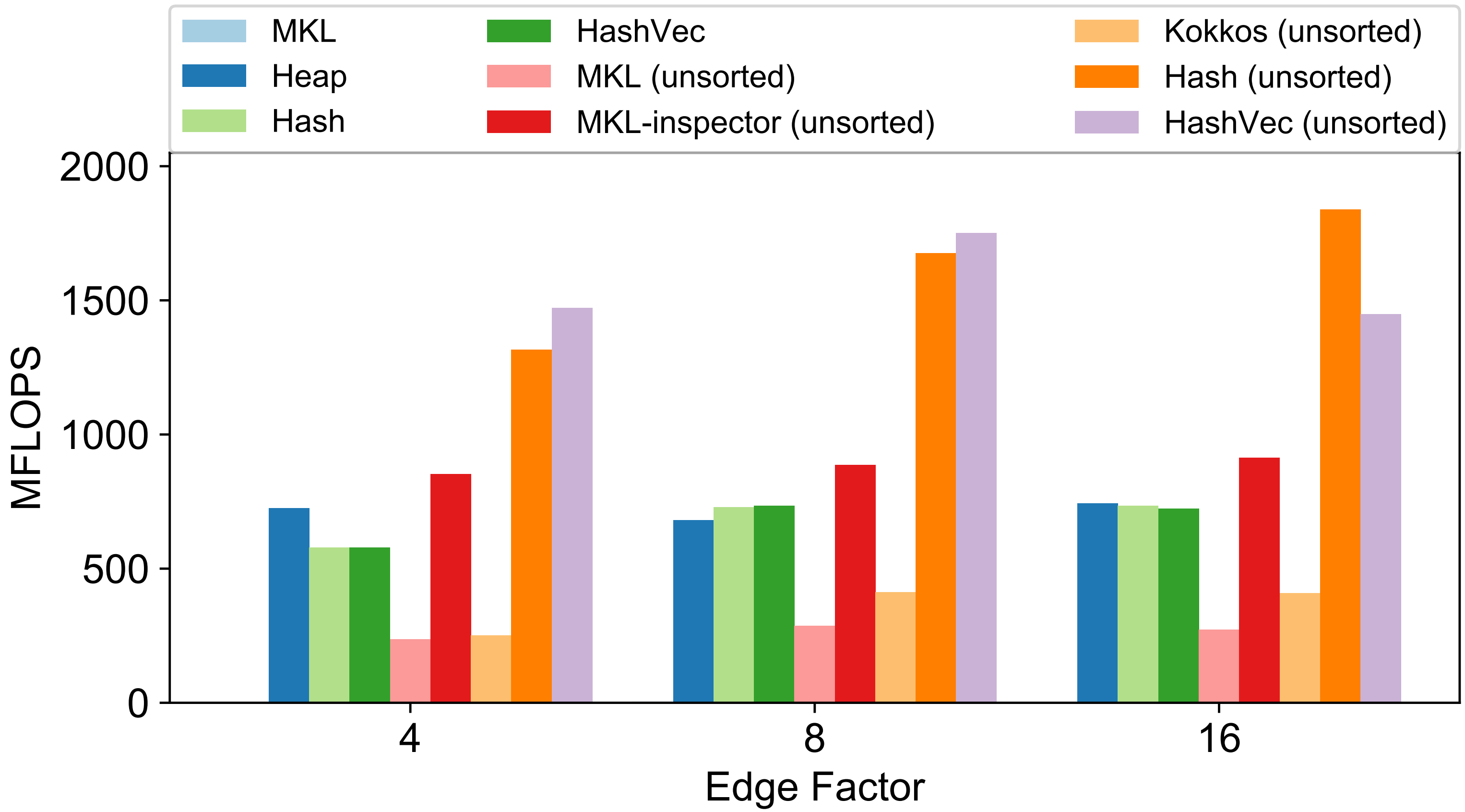
A^2: Scaling with density (KNL, ER)

- **Performance gain with keeping output unsorted**



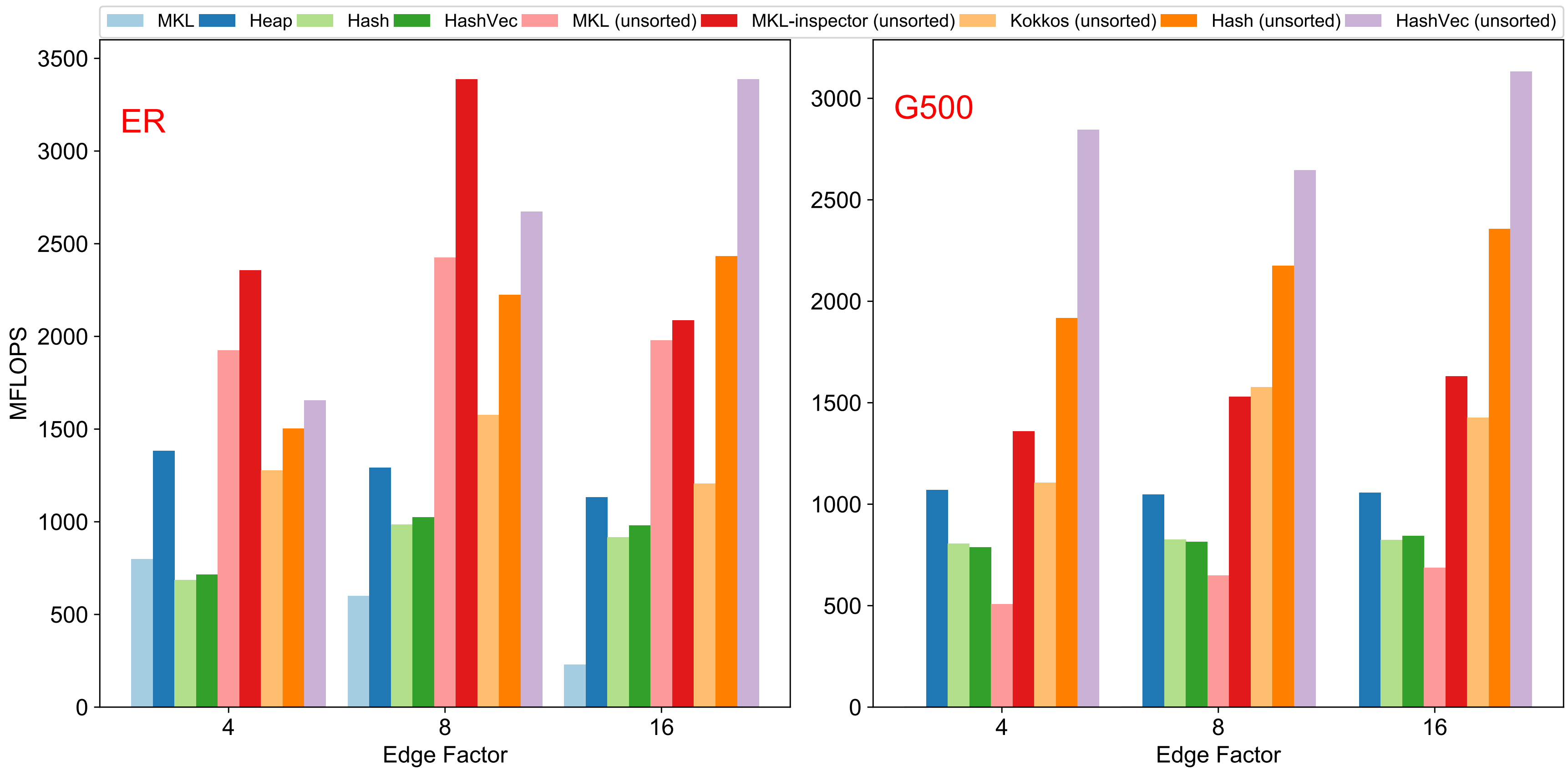
Performance Evaluation
A^2: Scaling with density (KNL, G500)

- Denser inputs do not simply bring performance gain
  - **Different from ER matrices**



Performance Evaluation
A^2: Scaling with density (Haswell)

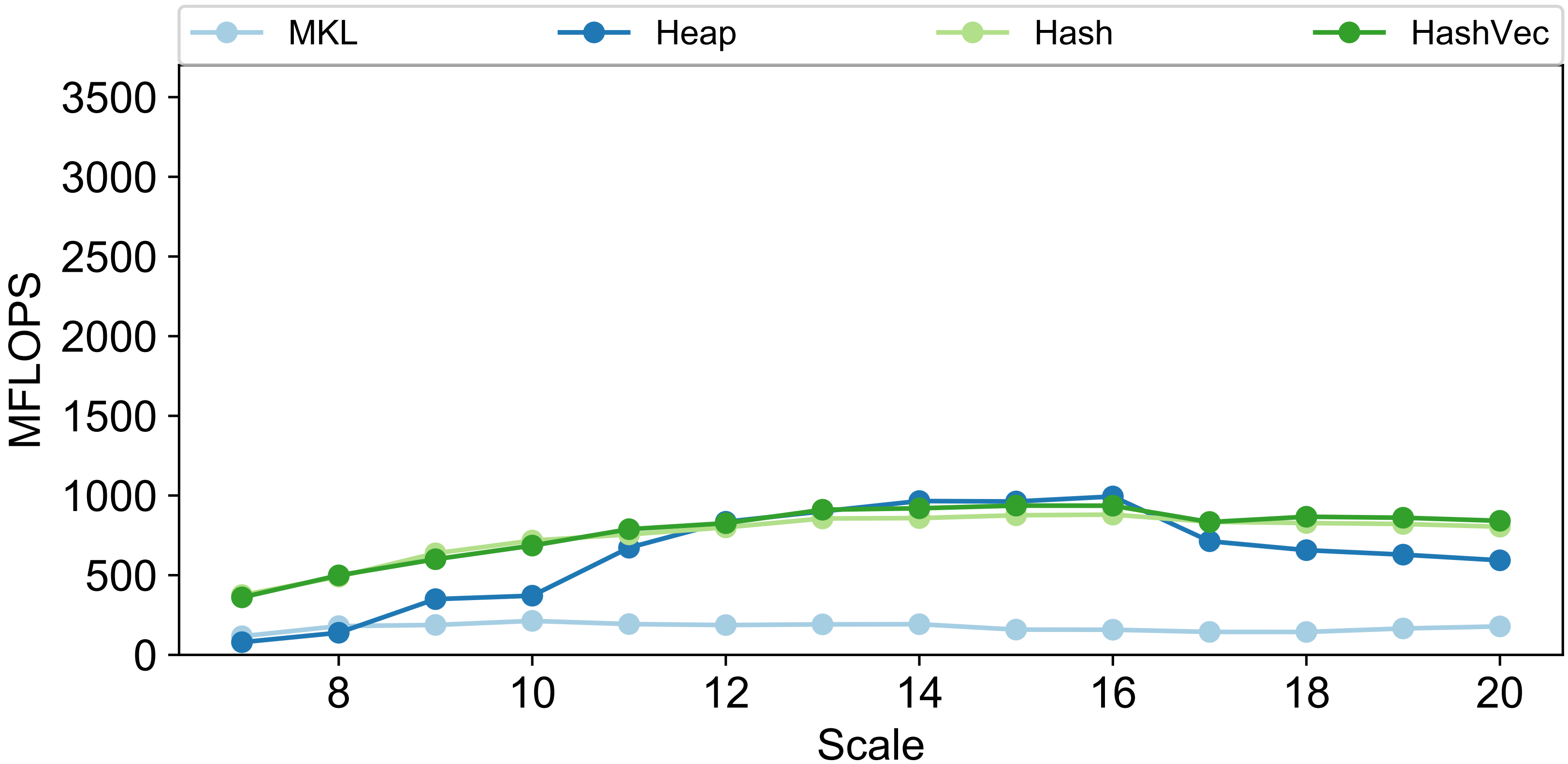
- HashVector achieves much higher performance



Performance Evaluation

A^2: Scaling with input size (KNL, ER)

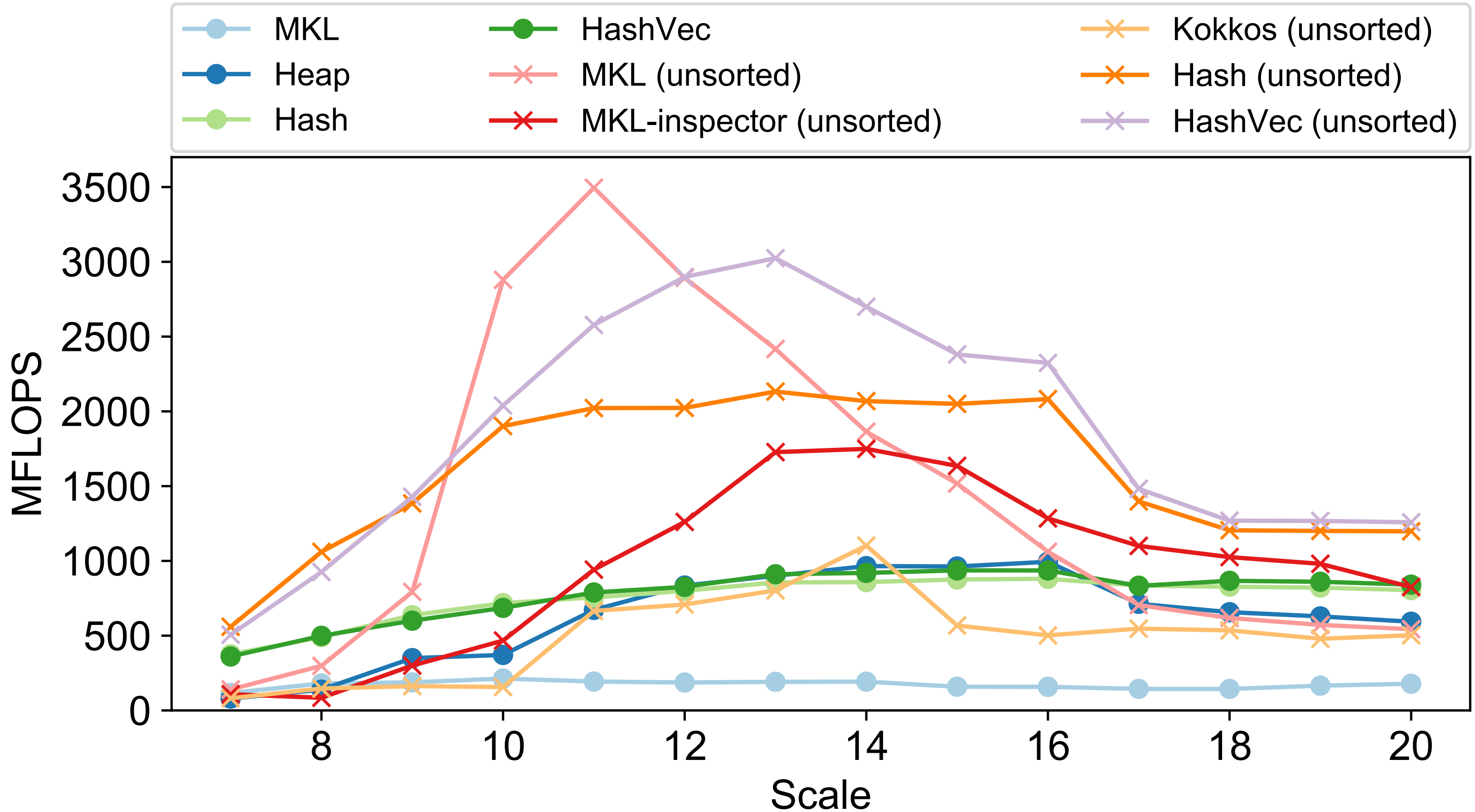
- Edge factor = 16
- Hash and HashVector show good performance in any input size



Performance Evaluation

A^2: Scaling with input size (KNL, ER)

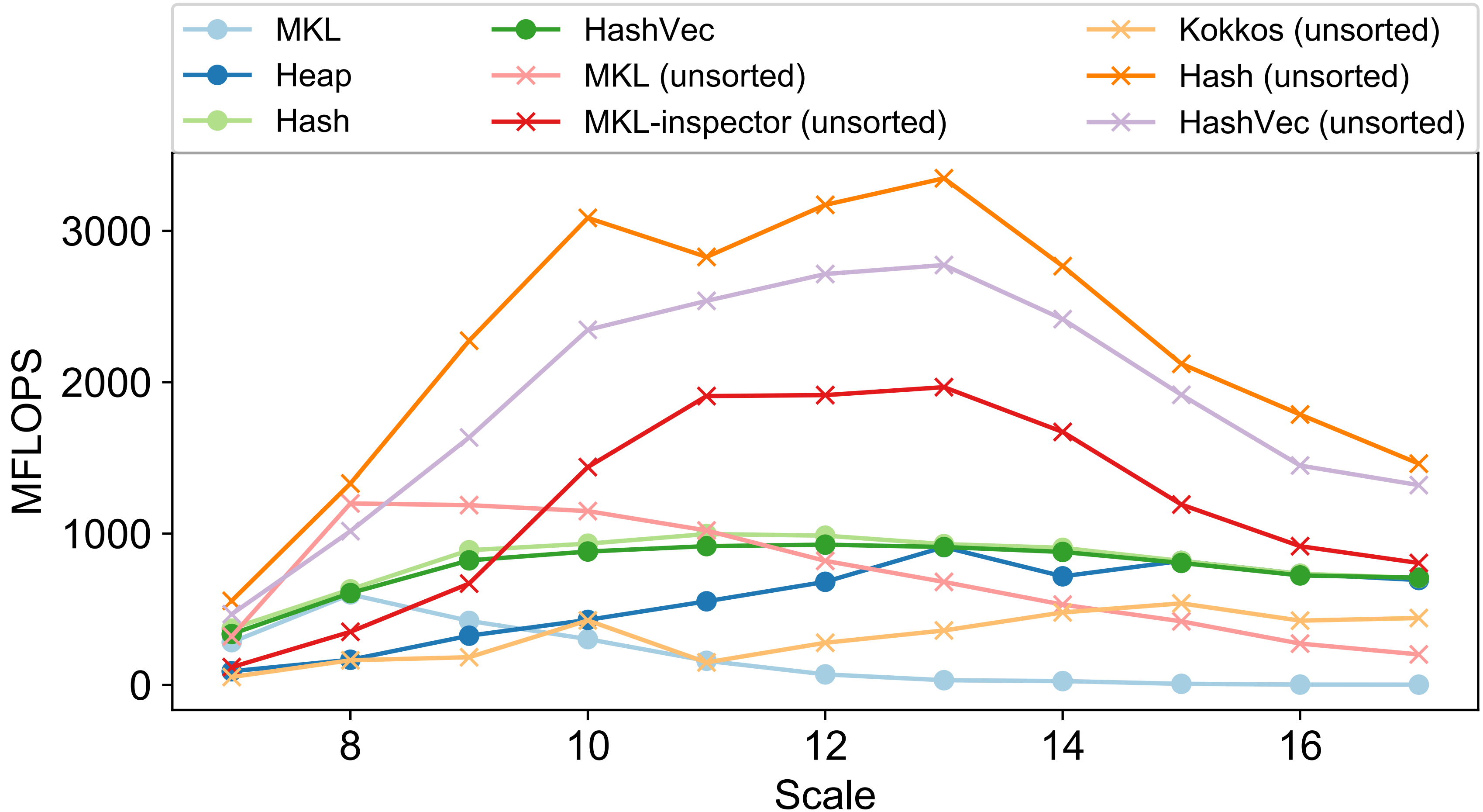
- **Performance gain with keeping output unsorted**
- MKL for small scale ⇔ HashVector for large scale



Performance Evaluation

A^2: Scaling with input size (KNL, G500)

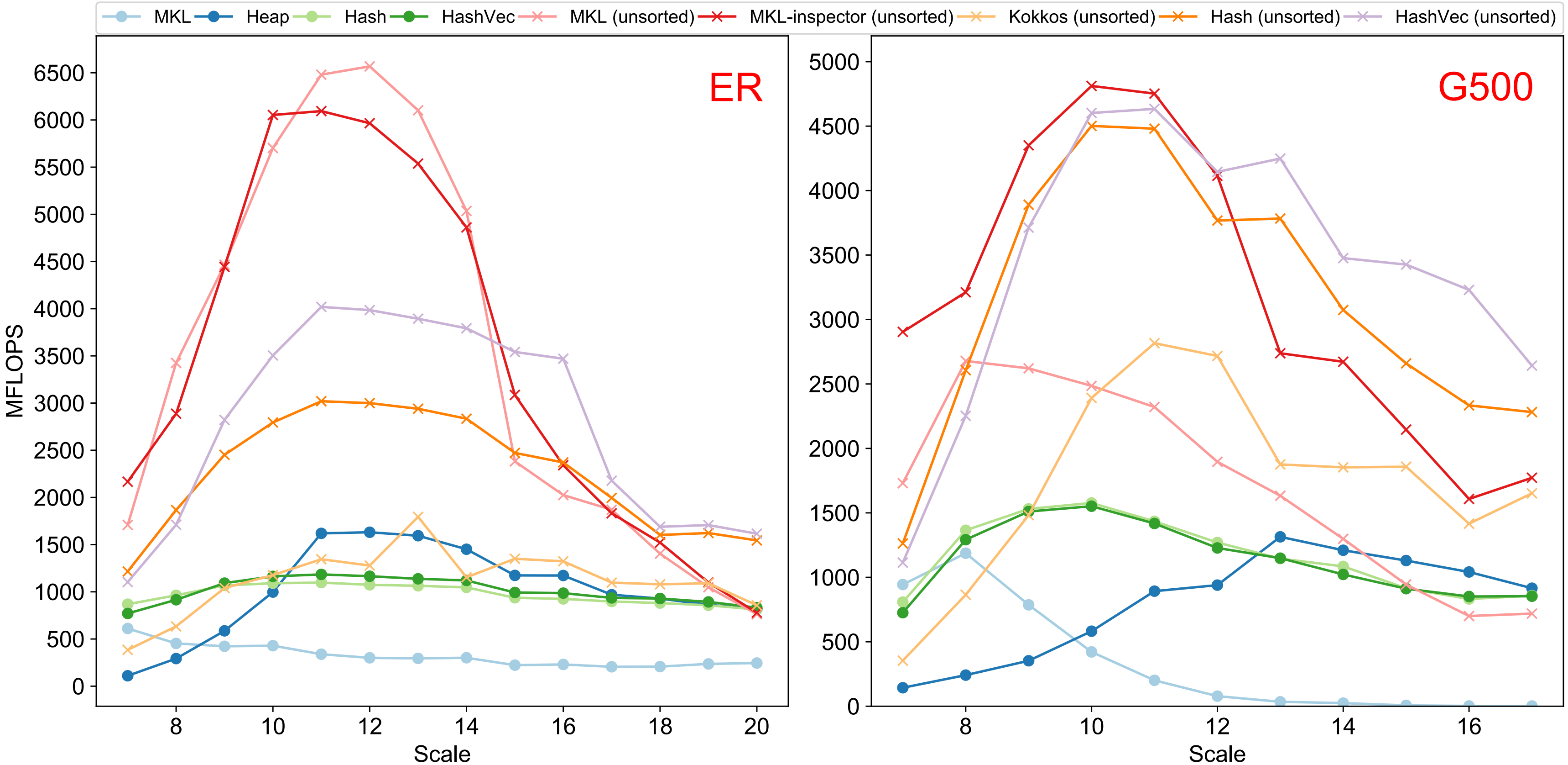
- **Hash is best performer**



Performance Evaluation

A^2: Scaling with input size (Haswell)

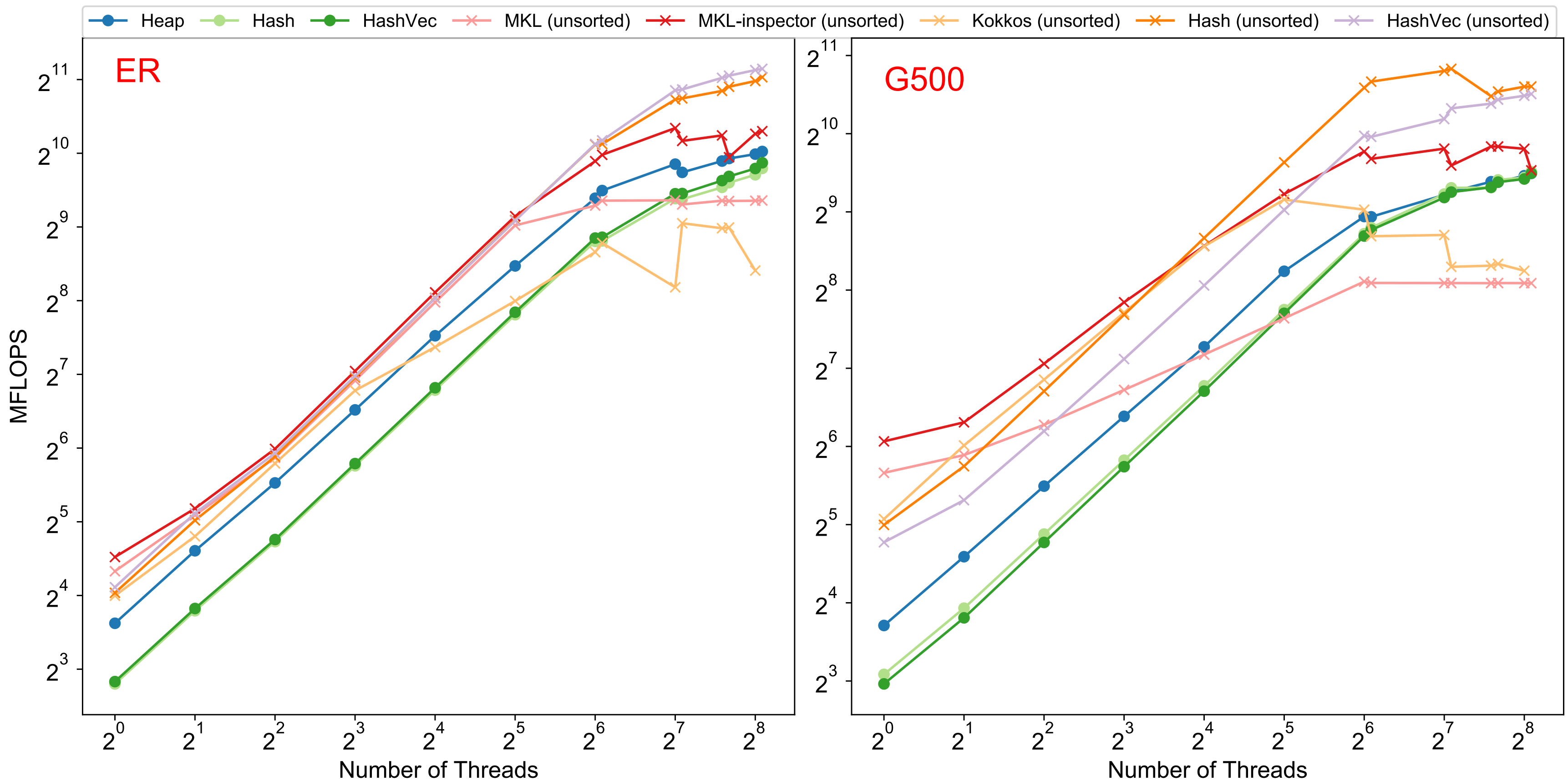
- **More clear performance trend of KNL**
  - MKL for smaller scales
  - Hash and HashVector for larger scales



Performance Evaluation

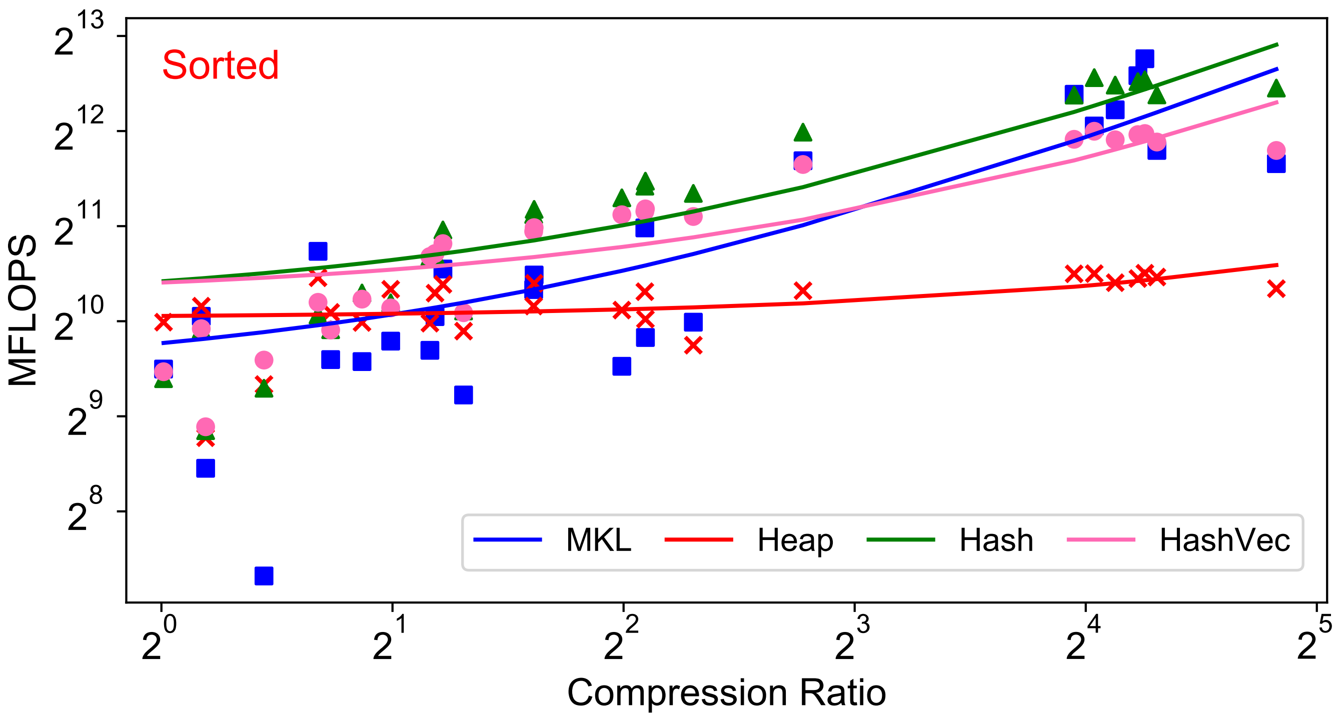
A^2: Scalability (KNL)

- **Good scalability of Hash and HashVec** even after 64 threads



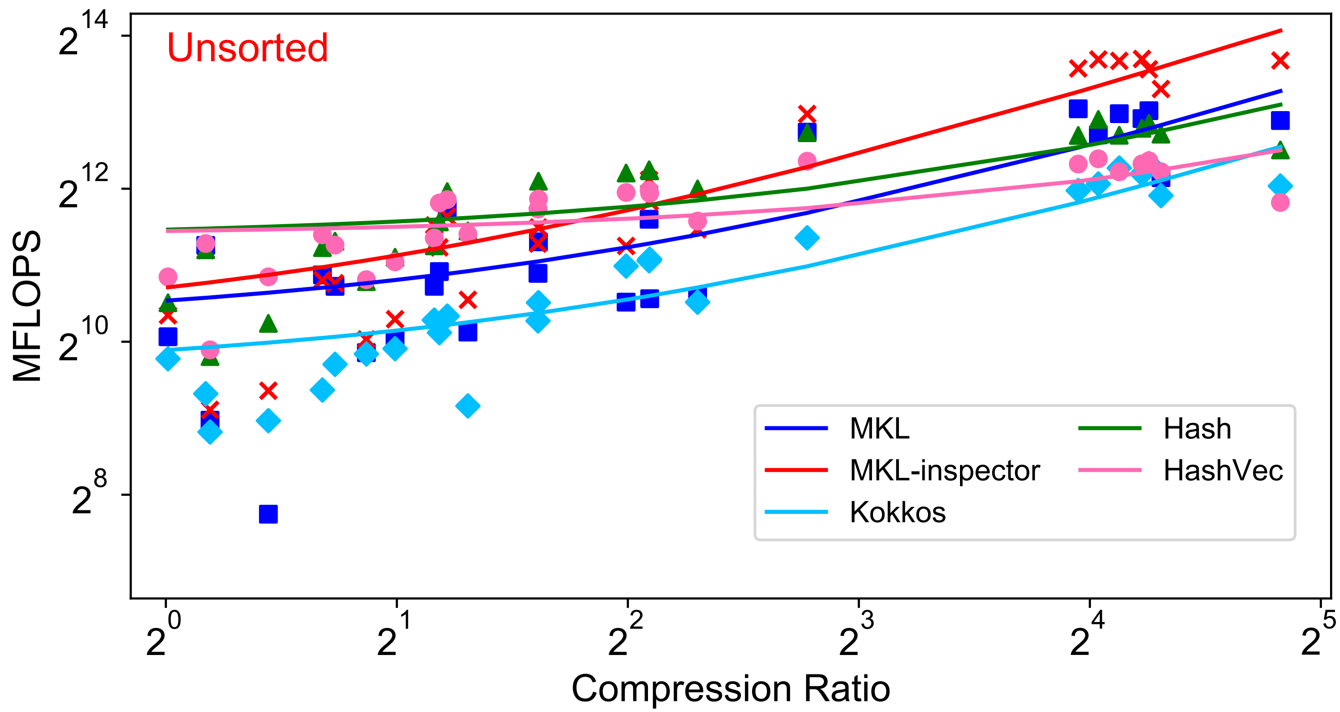
Performance Evaluation
A^2: Sensitivity of compression ratio (KNL)

- Evaluation on SuiteSparse matrices
- Compression ratio (CR): #flop/#non-zero of output
- Heap: stable performance
- **MKL and Hash: Better performance with higher CR**



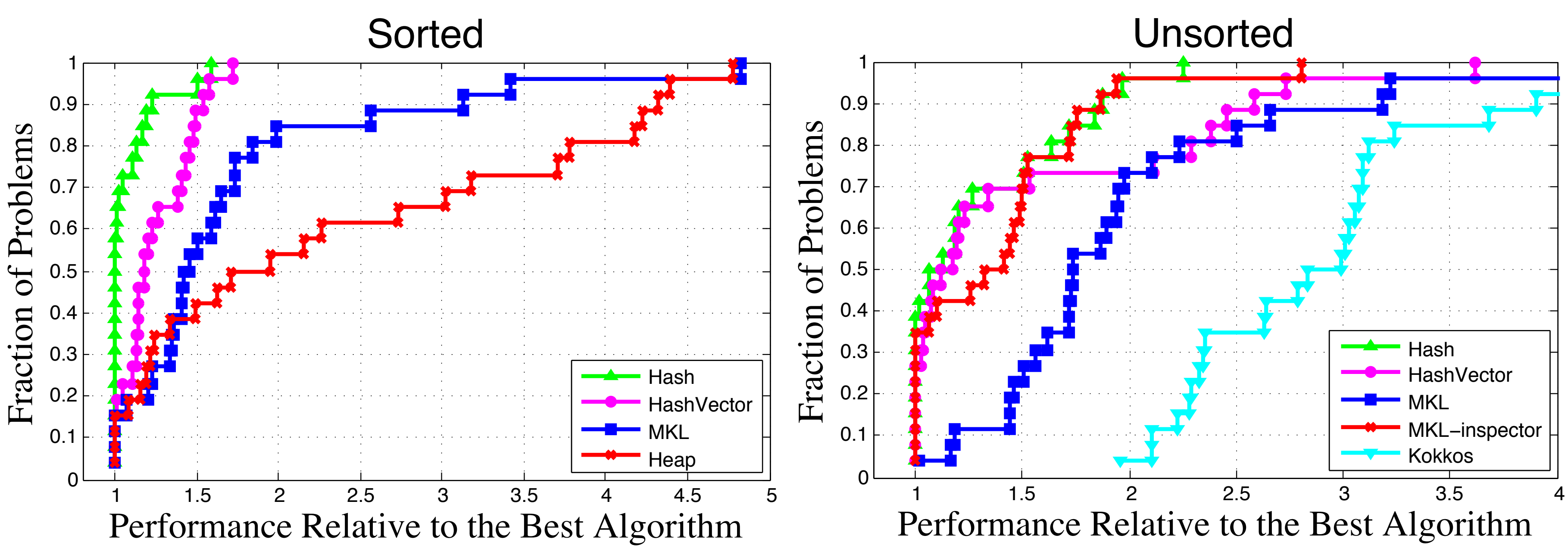
Performance Evaluation
A^2: Sensitivity of compression ratio (KNL)

- Hash for low CR ⇔ MKL family for high CR
- KokkosKernel underperforms other kernels



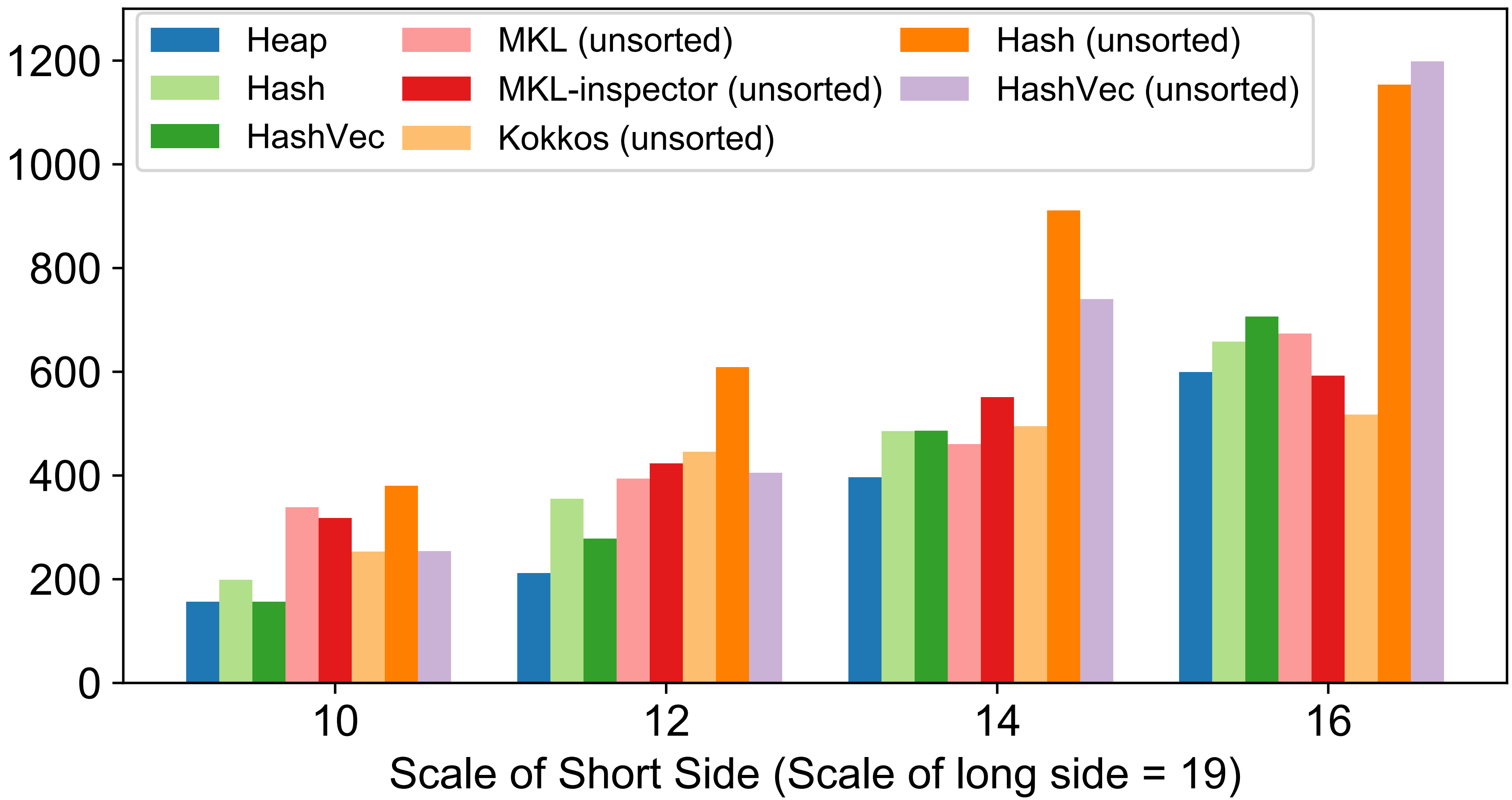
Performance Evaluation
A^2: Profile of Relative Performance

- **<u>Sorted</u>**: Hash is best performer for 70% matrices
  – **Runtime of Hash is always within 1.6x of the best**
- **<u>Unsorted</u>**: Hash, HashVector and MKL-inspector perform equally
  – **Each of them performs the best for about 30%**



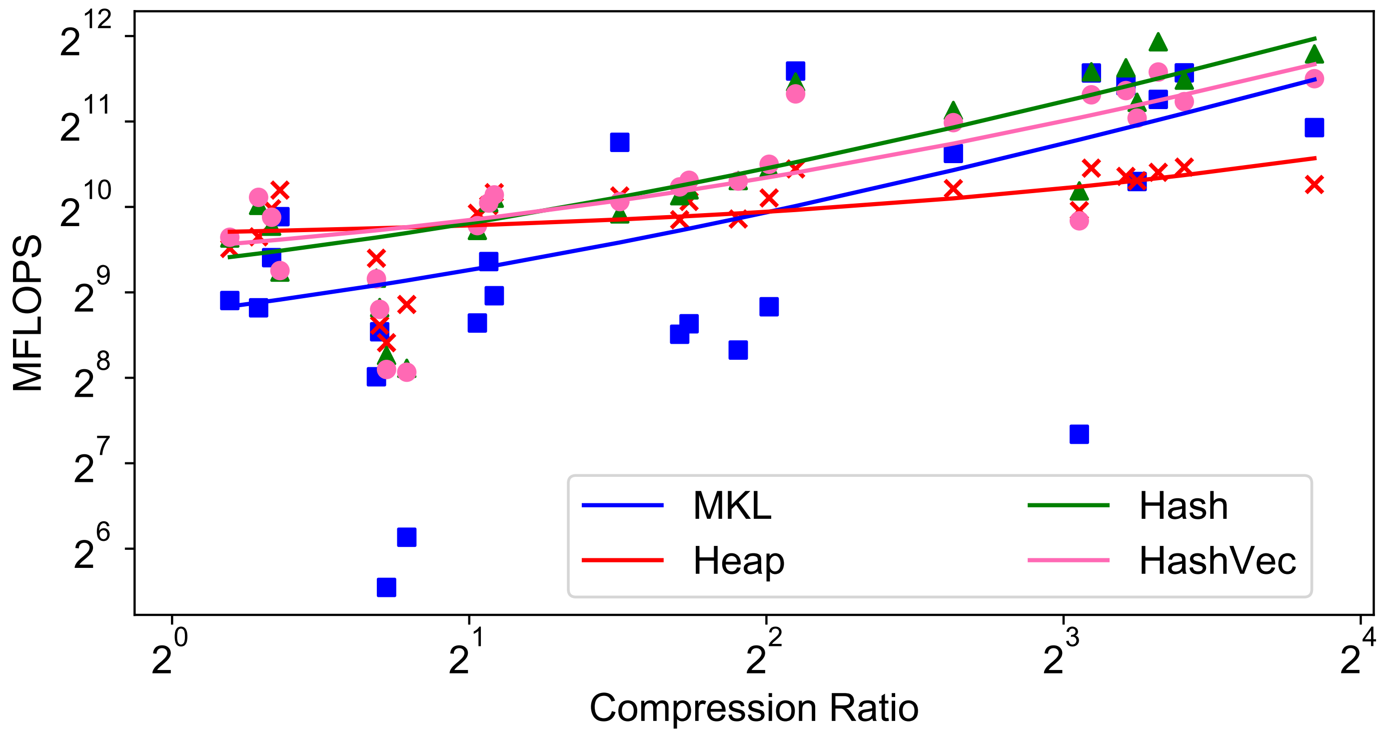
Performance Evaluation
Square x Tall-skinny matrix (KNL)

- Multiple BFS, Betweenness Centrality
- **Hash or HashVec is the best performer**



Performance Evaluation
Triangle Counting on SuiteSparse matrices (KNL)

- Reorders and transforms a matrix to L and U
  - L is lower triangle and U is upper triangle
- Similar performance trend to that of A^2
  - **Hash and HashVector generally overwhelm MKL**



Empirical Recipe for SpGEMM on KNL

(a) Real data specified by compression ratio (CR)

		High CR (>2)	Low CR (<=2)
A x A	Sorted	Hash	Hash
	Unsorted	MKL-inspector	Hash
L x U	Sorted	Hash	Heap

(b) Synthetic data specified by sparsity and non-zero pattern

		Sparse (Edge factor <=8)		Dense (Edge factor > 8)	
		Uniform	Skewed	Uniform	Skewed
A x A	Sorted	Heap	Heap	Heap	Hash
	Unsorted	HashVec	HashVec	HashVec	Hash
Tall-Skinny	Sorted	-	Hash	-	HashVec
	Unsorted	-	Hash	-	Hash

Conclusion

- Performance analysis of SpGEMM on Intel KNL and multicore architectures
  - Optimizing implementation for these architectures
    - **Identify the bottlenecks**
  - Evaluation in various use cases
    - **Clarify which SpGEMM algorithm works well**
  - Highlighting the **benefit of leaving matrices unsorted**
  - **Empirical recipe** for selecting the best-performing algorithm for a specific application scenario

Source code is publicly available at
https://bitbucket.org/YusukeNagasaka/mtspgemmlib