Pattern-Aware Vectorization for Sparse Matrix Computations

The UNIVERSITY of OKLAHOMA

Khaled Abdelaal
School of Computer Science
University of Oklahoma

Richard Veras
Center for Computation & Technology
Louisiana State University

Martin Kong
School of Computer Science
University of Oklahoma

Data-driven applications

Sparse data is dominant

- Big Data Analytics
- Social Networks
- Scientific Computing
- Machine Learning

Efficient storage and computation algorithms needed

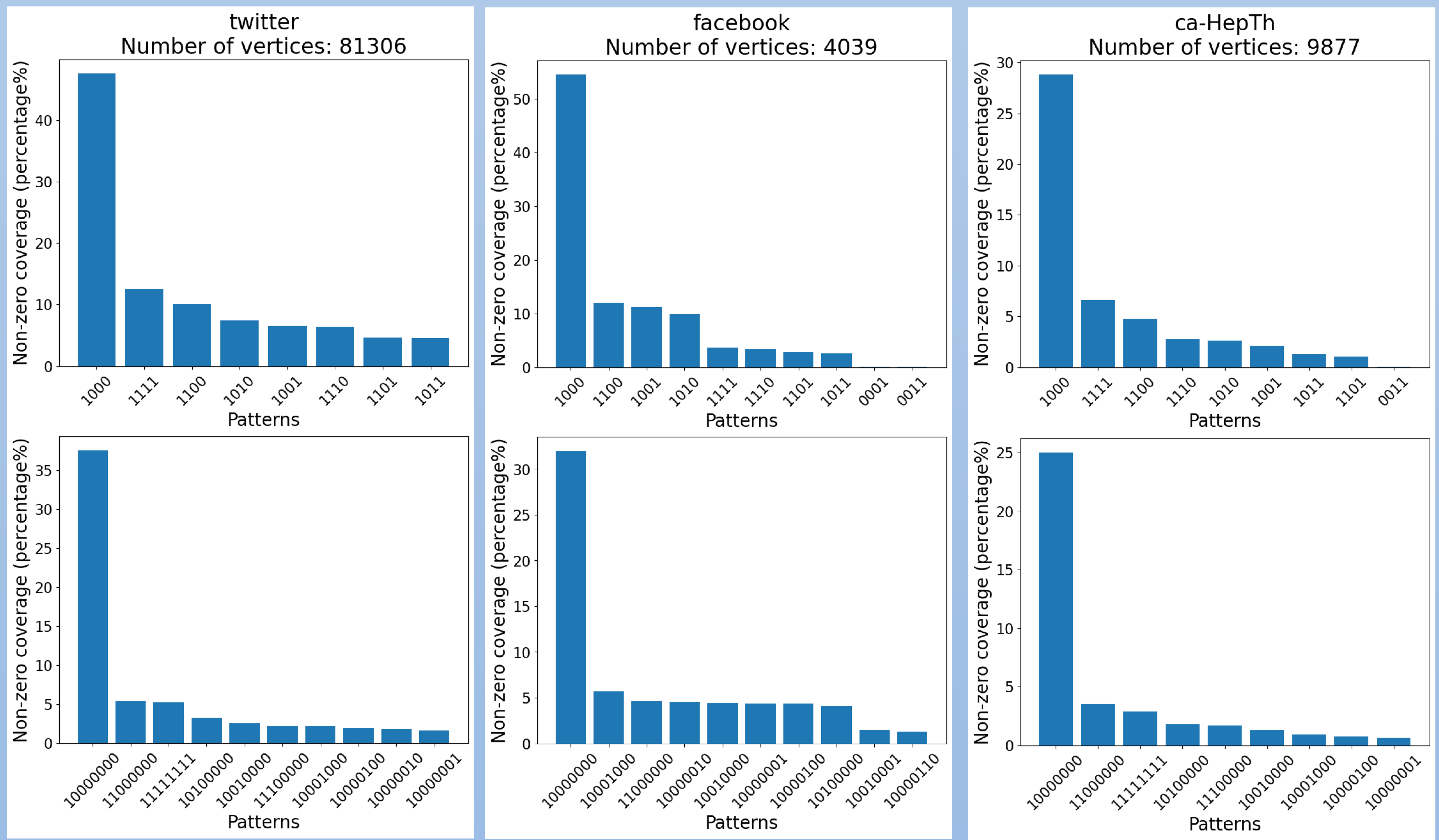
Objective

Fast, low-overhead computations on sparse matrices

Related Work Limitations

- Accelerators (i.e., GPUs) provide high speed-ups but limited memory capacity.
  - Unified memory with automatic host/device paging degrades performance
- Limited compiler support (i.e., GCC) for automatic vectorization on sparse structures
- Existing storage formats for sparse matrices do not automatically extract dense segments, not seizing the potential for aggressive vectorization

Motivation



Patterns non-zero coverage for vector size 4 (top) and 8 (bottom) for twitter, facebook and hepTh graphs from SNAP

- Potential for vectorization
- Patterns with a single non-zero coverage drops by extending vector size
- Focusing on only a few patterns for vectorization covers most of the matrix

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Sample patterns of vector size 4 in a sparse 4 x 8 matrix

Approach

- *Generate a list of dense regions within the sparse matrix*
- *Efficient scanning algorithm (or user input)*

Dense Segments Extraction

- *Use hierarchical representations of sparse matrices to enable fast exploration*

- *Data-level parallelism (SIMD) for sparse matrix computations*
- *Control-flow free vector code*

Vector Code Generation

- *Use Gather (load NNZ from sparse matrix using index vectors) and Scatter (store results to sparse locations)*

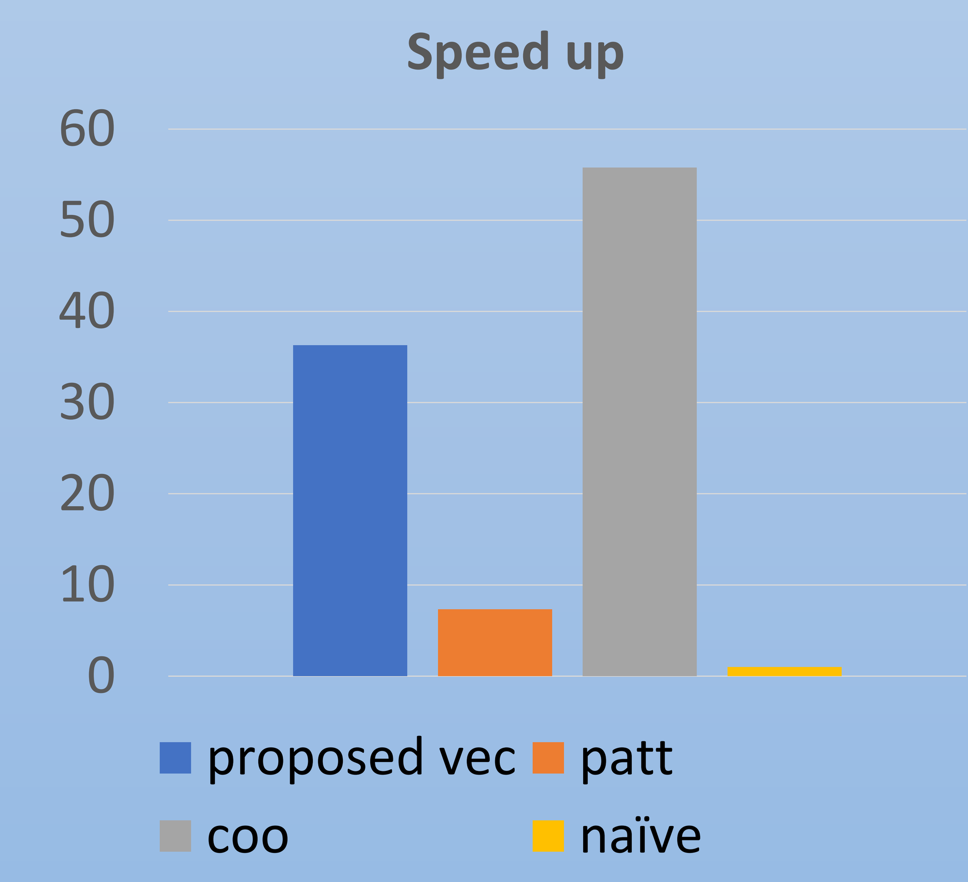
- Generate codelets tuned to each vector pattern
- Implement in back-end compilers (limited existing support in GCC)

Visiting Patterns using Codelets

- *Make use of vector intrinsics*
- *Polyhedral approach to be explored*

Evaluation

- **Kernel:** Sparse matrix-dense vector multiplication
- **Sparse matrix**: facebook adjacency matrix from SNAP (4039 x 4039), 176468 Single FP NNZs
- Memory: main 16 GB, L3 cache: 12 MB, L2 cache 1.5 MB, L1 cache: 192 KB
- Compiler: clang++ v10.0
- CPU: 12 cores (only sequential execution used)
- **Metric:** Speedup over naïve dense multiplication
- ***Proposed vec*** (dense segments start locations metadata + vectorization using **AVX256** intrinsics) achieves around **4.95x** speedup over patt (clang auto-vectorization with metadata). COO achieves 1.5x speedup over proposed vec



Conclusion

- A new approach to enable more efficient sparse data computation using:
  - Extraction and exploitation of metadata such as dense segments locations **ONGOING**
  - Generation of vector code to use data-level parallelism **ONGOING**
  - Efficiently visiting patterns using codelets **ONGOING**
- Preliminary results **DONE**
  - Potential of exploiting additional metadata
  - Potential of generating vector code combined with efficient storage formats