Subgraph Counting: Methods, Frameworks, and Applications

Ali Khan · Advisor: **Sanjukta Bhowmick**

Department of Computer Science and Engineering, University of North Texas

Motivation and Existing Gap

Social networks, protein–protein interaction networks, MPI event graphs and road networks are all modelled as graphs. Subgraph counting — how often a small pattern occurs inside a larger network — is a fundamental primitive for analysing them, underpinning graphlet kernels, network comparison, fraud detection and non-determinism analysis in HPC. It is a variant of subgraph isomorphism, NP-complete in general.

THREE GAPS IN CURRENT METHODS

- Static inputs. Most counting methods assume a small, static network, yet real networks are large and change continuously as edges arrive and disappear.
- Hardware-specific code. A bespoke kernel for every storage format × every backend imposes an engineering cost that grows with the accelerator ecosystem.
- Matching lags counting. Exact matching must locate, not merely tally, pattern instances, and has not benefited proportionally from advances in motif counting.

Research Contributions

This dissertation closes the gaps above with three systems, developed in the columns to the right:

A **ParaDyMS.** Parallel counting of all motifs of size k ≤ 5 on fully dynamic networks with batched insertions and deletions, on CPUs and GPUs through Kokkos.

B **PyGB-LAPIS.** A compilation pipeline lowering Python-GraphBLAS through MLIR and LAPIS into Kokkos-flavoured C++ across five sparse formats and multiple backends.

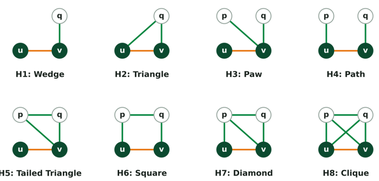
C **Graphlet Matching.** Exact subgraph matching that decomposes query and host into size-4 graphlets and searches a compressed graphlet-overlap representation.

ONE SOURCE, TWO PLATFORMS — KOKKOS

Every system targets the **Kokkos C++ Performance Portability ecosystem**, writing each kernel once and compiling it to both **OpenMP and CUDA** — so the same source scales to large, real-world graphs in reasonable time.

Background: Motifs

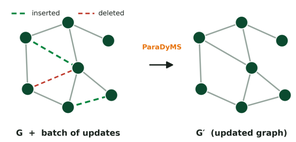
A motif (or graphlet) is a small induced pattern; a motif count is its frequency in a larger network. H1–H8 are the motifs of size 3 and 4 around an updated edge (u, v).



A ParaDyMS

Dynamic motif counting at scale — published, IEEE CCGrid 2025

Objective. Compute global frequencies of all motifs of size k ≤ 5 on fully dynamic networks under batched edge insertions and deletions.



A batch of one million insertions and deletions transforms G into G′.

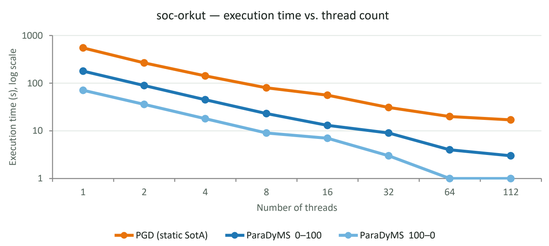
APPROACH — COMPUTE THE DELTA, NOT THE WHOLE

$$\text{inserted } (u,v): \quad f(H, G') = f(H, G) + H^{add}(u,v) - H^{minus}(u,v)$$
$$\text{deleted } (u,v): \quad f(H, G') = f(H, G) - H^{add}(u,v) + H^{minus}(u,v)$$

Motifs spanning several updated edges are normalised by the number of updated edges they contain; motifs mixing insertions and deletions apply the deletion terms first, skipping deleted edges.

ALGORITHM — THREE STEPS PER UPDATED EDGE

- Integer mapping. Each neighbour w of (u,v) is labelled 1–9, encoding its connection type and whether each incident edge lies in E or E′ (labels 1–4 → wedges, 5–8 → triangles).
- Local counts. A distance-2 scan aggregates counts into small fixed-size arrays indexed by connection type; size-4 motifs come from combining label pairs along the two legs.
- Global counts. Local counts around every updated edge are combined, with normalisation, into updated global frequencies.
- Cost: $O(|E'| \cdot d^2_{avg})$ serial, $O((|E'|/P) \cdot d^2_{avg})$ on P threads. Kokkos nested parallelism maps the outer edge loop and the inner distance-2 loop onto CPU threads or GPU teams from one source.



Six networks, 1 M-edge batches, insert/delete ratios 100-0, 50-50 and 0-100.

KEY CONTRIBUTIONS

- First method to count all motifs of size k ≤ 5 on fully dynamic networks; up to 124× faster than the static state of the art.
- Single Kokkos source runs on OpenMP and CUDA; released as an open-source artifact and evaluated on TACC leadership systems.

B PyGB-LAPIS

Python-GraphBLAS to Kokkos-flavoured C++ through MLIR and LAPIS

Objective. Move the combinatorial burden of supporting many storage formats × many hardware backends off the application developer and onto the compiler.

STAGE 1
Python-GraphBLAS
BFS as vxm · triangle counting as mxm · SSSP as weighted vxm, over user-defined semirings

↓

STAGE 2 · MLIR
MLIR: linalg → sparse_tensor
linalg.generic bodies with explicit indexing maps; sparse_tensor.binary / .reduce over the non-zero structure

↓

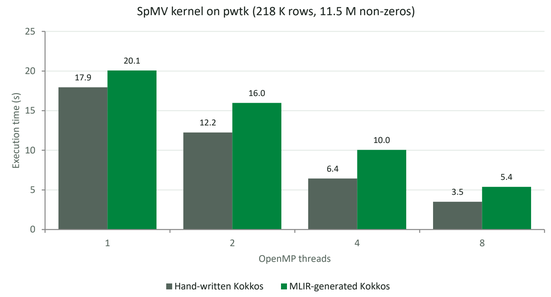
STAGE 2 · LAPIS
LAPIS lowering → SCF · MemRef · Kokkos dialects
kokkos.thread_parallel and kokkos.range_parallel loops over CSR-style memrefs

↓

STAGE 3
Kokkos-flavoured C++
KOKKOS_LAMBDA · TeamPolicy · ThreadVectorRange — team and vector parallelism sized at runtime

ONE FRONTEND, MANY TARGETS

Formats: COO · CSR · CSC · DCSR · DCSC
Backends: multicore CPU · NVIDIA · AMD · Intel GPUs and APUs, via Kokkos



Target algorithm: 54GE simultaneous four-node graphlet enumeration, re-implemented in Kokkos.

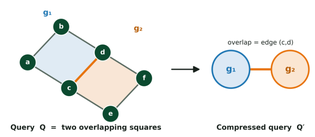
KEY CONTRIBUTIONS

- First pipeline carrying Python-GraphBLAS to both CPU and GPU through MLIR and LAPIS, over five sparse storage formats.
- Built inside production compiler infrastructure (MLIR, LAPIS, Kokkos) with Sandia collaborators; generated kernels match hand-written performance.

C Graphlet Matching

Exact matching over a compressed graphlet-overlap representation

Objective. Locate — not merely tally — every induced instance of an arbitrary-size query Q inside a host graph G, with **many queries answered together in a single pass** rather than one search restarted per query.



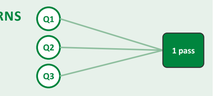
Key insight: break the query into size-4 graphlets joined by their overlaps.

METHOD — THREE STEPS

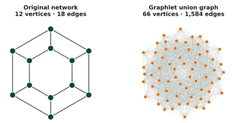
- Compressed query Q′. Build the graphlet union graph of Q, then find a small subgraph whose union reproduces Q.
- Host representation G′. Construct and compress the graphlet union graph of G.
- Search. Reducing the search to two graphlets sharing an edge cuts $O(v^k)$ enumeration to $O(v^3)$ in the simplest case.

MULTI-QUERY SEARCH — ONE PASS, MANY PATTERNS

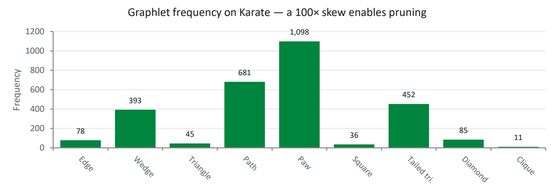
Most systems (GraphPi, CuTS) restart from scratch per query. **Queries with overlapping motifs are merged into a single meta-search tree**, and the search prioritises the sections of greatest overlap, so intermediate work is shared across every query answered in the pass.



CHALLENGE — REPRESENTATION BLOW-UP



On Karate (34 v, 78 e) the union graph reaches 2,363 vertices and 1,569,175 edges.



Mitigation: traverse the union graph on the fly and prune graphlet classes absent from Q.

KEY CONTRIBUTIONS

- Multiple simultaneous queries answered in a single pass via a shared search tree, not one restart per query like prior systems.
- Exact matching for arbitrary-size queries by size-4 graphlet decomposition, with the union graph traversed on the fly and pruned by frequency.

Where This Transfers

- Computational biology — graphlet degree distributions for PPI network comparison and protein structure alignment.
- Fraud and anomaly detection — motif frequency shifts in evolving transaction and social graphs.
- HPC performance analysis — graph kernels that detect non-determinism in MPI event traces.
- Compiler and accelerator R&D — sparse code generation portable across NVIDIA, AMD and Intel targets.

Future Work

- Dynamic counting of larger motifs — extend ParaDyMS's delta-based updates beyond k ≤ 5 as neighbourhood sizes grow.
- Dynamic graphs for the compiler and matching — extend PyGB-LAPIS and graphlet matching to fully dynamic networks, unifying all three thrusts under one framework.
- Compression with larger graphlets — explore graphlets beyond size 4 to tighten the union-graph representation and support bigger queries.

Publications

- A. Khan, N. Tan, J. Marquez, M. Taufer, S. Bhowmick. *ParaDyMS: Parallel Dynamic Motif Counting at Scale.* IEEE CCGrid 2025.
- A. Khan, B. Kelley, K. Liegeois, S. Rajamanickam. *Generating Kokkos Code from Sparse Tensor Algebra in Python Using MLIR.* Sandia CSRI Proceedings 2023 (SAND2023-13916R).
- A. Khan, S. Bhowmick. ***Multi-Query Subgraph Matching with Graphlet Compression.*** In preparation.

This work is supported by the U.S. National Science Foundation under awards 1900765, 1900888 and 2341588, and by Sandia National Laboratories through the Computer Science Research Institute. Experiments used TACC Stampede3 and Lonestar6. ParaDyMS was published at IEEE CCGrid 2025; the MLIR-to-Kokkos work appears in the Sandia CSRI Proceedings 2023 (SAND2023-13916R).