LESS: Low-power Energy-efficient Subgraph Isomorphism on FPGA

Roberto Bosio*, Giovanni Brignone, Filippo Minnella, M. Usman Jamal, Luciano Lavagno
Politecnico di Torino, Turin, Italy
*e-mail: roberto_bosio@polito.it

***Abstract*—Low-power energy-efficient subgraph isomorphism (LESS) is an open-source field-programmable gate array-only low-memory subgraph matching solver designed for energy efficiency. Depending on the input datagraph, the energy consumption of LESS, averaged on different diverse queries, is up to 38× and 93× lower than CPU and GPU solvers respectively.**

***Index Terms*—subgraph isomorphism, FPGA, energy efficiency**

I. Introduction

Subgraph isomorphism is an NP-hard graph matching problem consisting in finding every instance of a specific pattern (query graph) within a larger data graph. As an example, Fig. 1 highlights the matches of the query $\{u_0, u_1, u_2\}$ against the data graph in Fig. 1b. Subgraph isomorphism is applied to various domains such as social network analysis [1], chemical compound search [2], and resource description framework (RDF) query processing [3].

The literature proposed different computing architectures, with an emphasis on performance, particularly for GPU-based solutions. However, energy efficiency is critical for many applications such as databases. LESS[1] is a novel low-power energy-efficient subgraph isomorphism implementation. Unlike previous solutions, LESS does not require a power-hungry CPU host, as the field-programmable gate array (FPGA) handles both the pre-processing and the matching tasks while still requiring a limited amount of resources. The architecture, deployed on a low-power embedded FPGA, demonstrates superior energy efficiency compared to state-of-the-art CPU and GPU solutions in real-world graph experiments, outperforming RapidMatch [4] by up to 3.6×, DAF [5] by up to 38.6×, and GSI [6] by up to 93.7× in terms of average energy consumption.

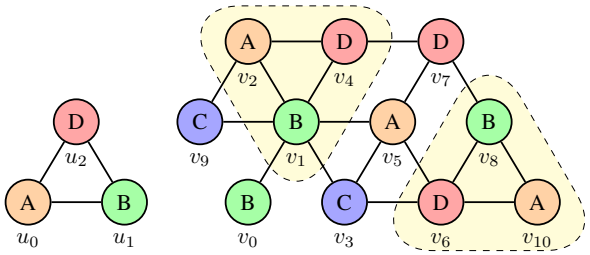
II. Architecture

The architecture of LESS, shown in Fig. 2, consists of two main phases. The *pre-processing* phase reorganizes and loads the graphs to the data structures. The *enumeration* phase iteratively builds the matches via set intersections.

Both phases are implemented on FPGA according to the dataflow paradigm, where tasks concurrently process different partial solutions. Furthermore, the tasks are internally pipelined.

This work was partially supported by the Key Digital Technologies Joint Undertaking under the REBECCA Project with grant agreement number 101097224, receiving support from the European Union, Greece, Germany, Netherlands, Spain, Italy, Sweden, Turkey, Lithuania, and Switzerland.

[1]The code is open-source at https://github.com/robertoBosio/LESS.



(a) Query graph. (b) Data graph.

Fig. 1. Query (a) and data graph (b) example. Matches highlighted in yellow.

A. Pre-processing

The pre-processing step aims at shrinking the search space. Our pre-processing focuses on building hash tables and enhancing spatial locality to improve caching performance.

The *Filter* step groups graph edges based on relations and filters vertices with labels not in the query.

Then, the *Data structures* step builds a hash table for each relation to keep track of the different adjacency lists. Additionally, it generates the Bloom filters. Each Bloom filter is associated with a hash table bucket and represents the vertices within it. The data structure shares similarities with partitioned compressed sparse row representation, but differs for a second layer of hashing, for quickly checking vertex presence, and the overflow handling. The off-chip memory usage is proportional to the number of query edges and the dimension of the hash tables, which are determined at runtime given the dimension of the graphs and the amount of available memory. Large hash tables reduce hash collision at the expense of increased off-chip memory usage. Conversely, small tables save memory but increase the number of slow and energy-hungry off-chip memory transactions in the enumeration phase.

B. Enumeration

The enumeration phase extends a partial match to cover an additional query vertex in a graph. Similarly to previous studies, our approach exploits set intersection. Specifically, given a partial solution and a query vertex to be covered, it identifies potential extensions by selecting query vertices already mapped in the vertex's neighborhood and intersecting their mapping's adjacency list. The algorithm reads, expands, and updates partial matches until no expandable solutions are left. The technique for computing the intersection is based on finding the smallest set among the considered ones and then probing its elements

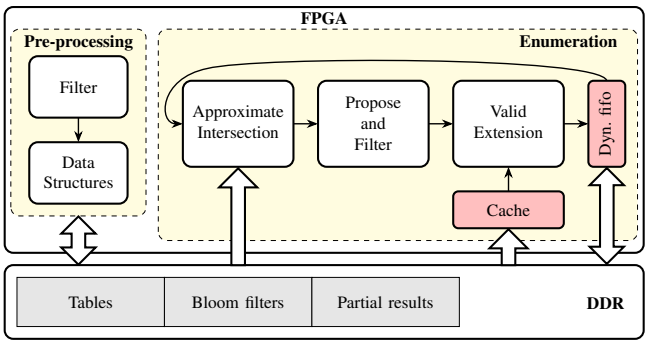


Fig. 2. The architecture of LESS.

against the other set's hash tables, respectively done in the *Propose and Filter* and *Valid Extension* tasks.

To reduce the memory accesses, the *Approximate Intersection* phase intersects small Bloom filters representing adjacency lists. This approximate representation allows filtering wrong matches without checking the existence of the edge in memory.

C. Memory management

Graph processing is often memory-bound due to little to no locality. The issue is even more critical in the FPGA context, because of the limited on-chip memories.

LESS couples high-locality data structures with an open-source FPGA cache [7], to introduce and exploit both spatial and temporal locality. Spatial locality is achieved through re-organized adjacency lists, optimizing memory accesses during extension set validity checks. Temporal locality stems from a breadth-first search approach, prioritizing data reuse between consecutive extensions of partial matches.

Additionally, to handle a potentially large number of partial solutions, LESS employs a dynamic FIFO that efficiently switches between on-chip and off-chip memory storage, effectively managing data overflow while masking memory latency.

III. Results

We compared LESS with the state-of-the-art open-source solutions for vertex-labeled subgraph matching, namely RapidMatch, DAF, and GSI. We described LESS in C++ compatible with Xilinx Vitis HLS 2022.2 and deployed it on an AMD Kria KV260 with 4 GB of off-chip memory and 3 MB of on-chip memory. The kernel runs at 290 MHz. We executed CPU-based solutions on a CentOS 7 workstation equipped with a 16-core Intel i7-6900K CPU (3.2 GHz) and 128 GB of host memory. We executed GPU-based solutions on an NVIDIA GeForce GTX 1070 (1.5 GHz) with 1920 CUDA cores and 8 GB of global memory. The GPU and the FPGA share the same 16 nm technology, and the CPU uses a similar one (i.e., 14 nm).

A. Datasets and queries

Following previous studies, we utilize five real-world data graphs from SNAP [8]. For each data graph, we randomly assigned labels to the vertices, taken from a small set to avoid trivial queries, as a larger number of labels would noticeably reduce the search space. We generated 30 random query templates, with 3 to 8 vertices and 3 to 26 edges.

TABLE I
Throughput comparison.

Algorithm	Throughput (Msolution/s)				
	Enron	Github	Gowalla	Dblp	Wikitalk
LESS	12.15	6.17	6.66	20.39	3.24
RM	**31.89**	**21.92**	**23.70**	**83.15**	**31.37**
DAF	20.14	10.11	6.88	15.72	12.68
GSI	2.81	1.93	1.12	–	–

B. Comparison

We measured the time elapsed and the energy consumed from the start of the algorithm execution until all the matches were found, neglecting the time spent loading the data graph to memory from disk. Figure 3 reports the energy consumption of the algorithms on each dataset, averaged on the different queries. Similarly, Table I reports the average throughput expressed as solutions per time unit. GSI failed to solve most of the queries on Dblp and Wikitalk due to memory overflow. Thus, we omitted this data.

LESS systematically outperforms the energy efficiency of state-of-the-art implementations, achieving up to 3.6× lower energy per query compared to RM (Enron), up to 38.6× compared to DAF (Dblp) and up to 93.7× compared to GSI (Enron).

While maintaining its top-tier energy efficiency, the advantage of LESS over the other solutions is less evident on Wikitalk, due to the skewed distribution of edges across the graph which reduces the filtering capability of the Bloom filters.

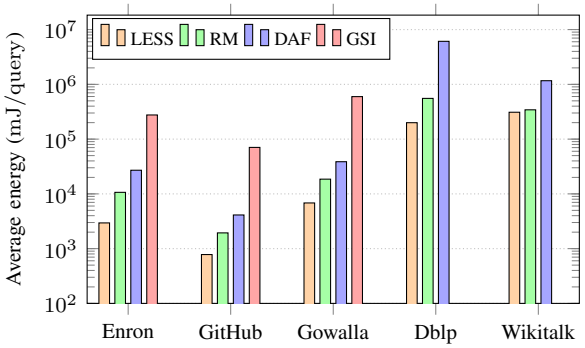


Fig. 3. Average energy consumption per query of each algorithm

References

[1] W. Fan, "Graph Pattern Matching Revised for Social Network Analysis," in *Int. Conf. Database Theory*, 2012, pp. 8–21.

[2] X. Yan *et al.*, "Graph Indexing: A Frequent Structure-Based Approach," in *ACM SIGMOD Int. Conf. Manag. Data*, 2004, pp. 335–346.

[3] L. Zou *et al.*, "Gstore: Answering sparql queries via subgraph matching," *Proc. VLDB Endow.*, vol. 4, no. 8, pp. 482–493, May 2011.

[4] S. Sun *et al.*, "Rapidmatch: a holistic approach to subgraph query processing," *Proc. VLDB Endow.*, vol. 14, no. 2, pp. 176–188, Oct. 2020.

[5] M. Han *et al.*, "Efficient Subgraph Matching: Harmonizing Dynamic Programming, Adaptive Matching Order, and Failing Set Together," in *ACM SIGMOD Int. Conf. Manag. Data*, 2019, pp. 1429–1446.

[6] L. Zeng *et al.*, "GSI: GPU-friendly subgraph Isomorphism," in *IEEE ICDE*, 2020, pp. 1249–1260.

[7] G. Brignone *et al.*, "Array-Specific Dataflow Caches for High-Level Synthesis of Memory-Intensive Algorithms on FPGAs," *IEEE Access*, vol. 10, pp. 118 858–118 877, 2022.

[8] J. Leskovec *et al.*, *SNAP Datasets: Stanford large network dataset collection*, http://snap.stanford.edu/data, Jun. 2014.