FAST: FPGA-based Subgraph Matching on Massive Graphs

Xin Jin†, Zhengyi Yang § , Xuemin Lin § , Shiyu Yang♮, Lu Qin‡, You Peng §
†East China Normal University, § University Of New South Wales,
♮Guangzhou University, ‡University of Technology Sydney

ICDE 2021

Outline

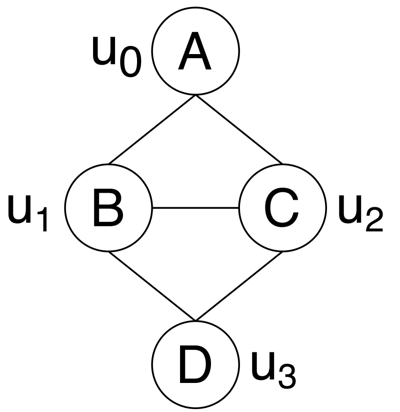
- Introduction
- System Overview
- Software Preprocessing
- Hardware Implementation
- Experiments
- Conclusion

Introduction

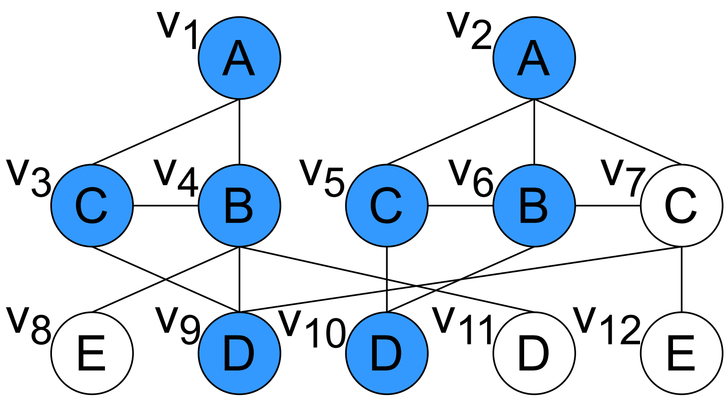
Problem Definition

Subgraph Matching:

Given a query graph $\boldsymbol{q}$ and a data graph $\boldsymbol{G}$, the problem is to extract all subgraph isomorphic embeddings of $\boldsymbol{q}$ in $\boldsymbol{G}$.



query graph $\boldsymbol{q}$



data graph $\boldsymbol{G}$

e.g.

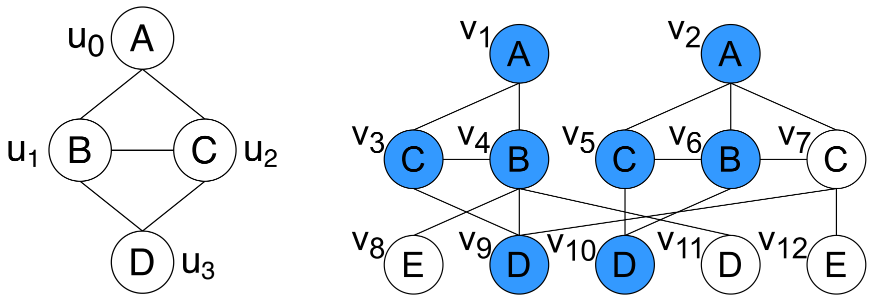
$(u_0, u_1, u_2, u_3) \rightarrow (v_1, v_4, v_3, v_9)$

$(u_0, u_1, u_2, u_3) \rightarrow (v_2, v_6, v_5, v_{10})$

Existing Solutions - CPU

Backtracking Framework:

- Auxiliary data structure to find a candidate set $C(u)$ for each query node u (e.g. $C(u_0) = \{v_1, v_2\}$).
- Apply backtracking based on a linear order of query nodes, called matching order (e.g. (u_0, u_1, u_2, u_3)).



query graph ***q*** data graph ***G***

State-of-the-art algorithms:

- CFL[SIGMOD 2016]、DAF [SIGMOD 2019]、CECI [SIGMOD 2019]

Existing Solutions - GPU

Join-based solutions:

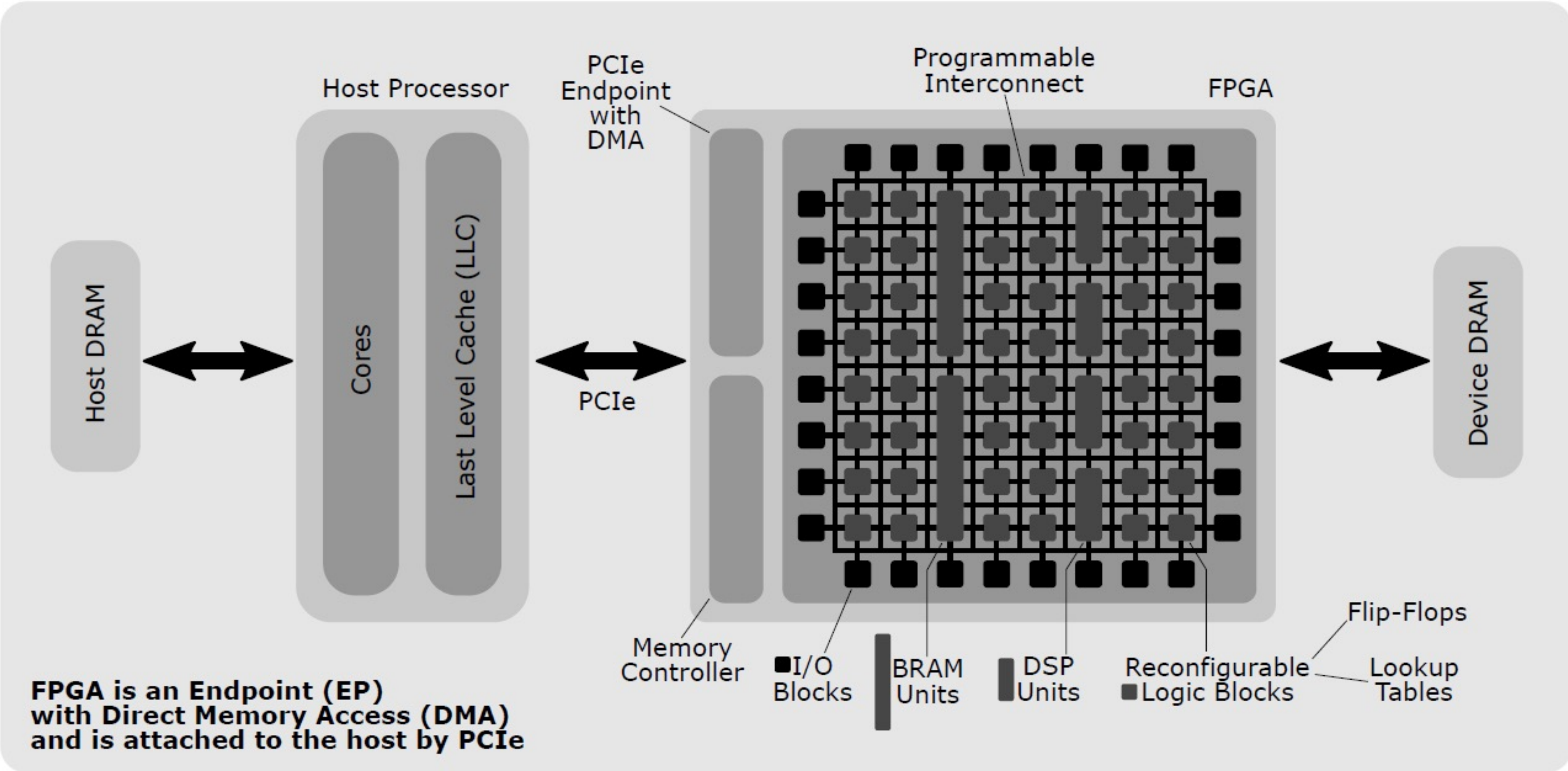
- Collect candidates for each query edge or node and join them in GPUs
- Two-step output schema or Prealloc-Combine to solve writing conflicts

State-of-the-art algorithms:

- GpSM [DASFAA 2015]、GunrockSM [HPDC 2016]、GSI [ICDE 2020]

FPGA

Properties: reprogrammable, massive parallelism, energy-efficient



Challenges

Strictly pipelined design on FPGA:

- No data dependencies among iterations
- Much lower clock frequency than CPUs

→ CPU-FPGA co-design framework & Matching process decomposition

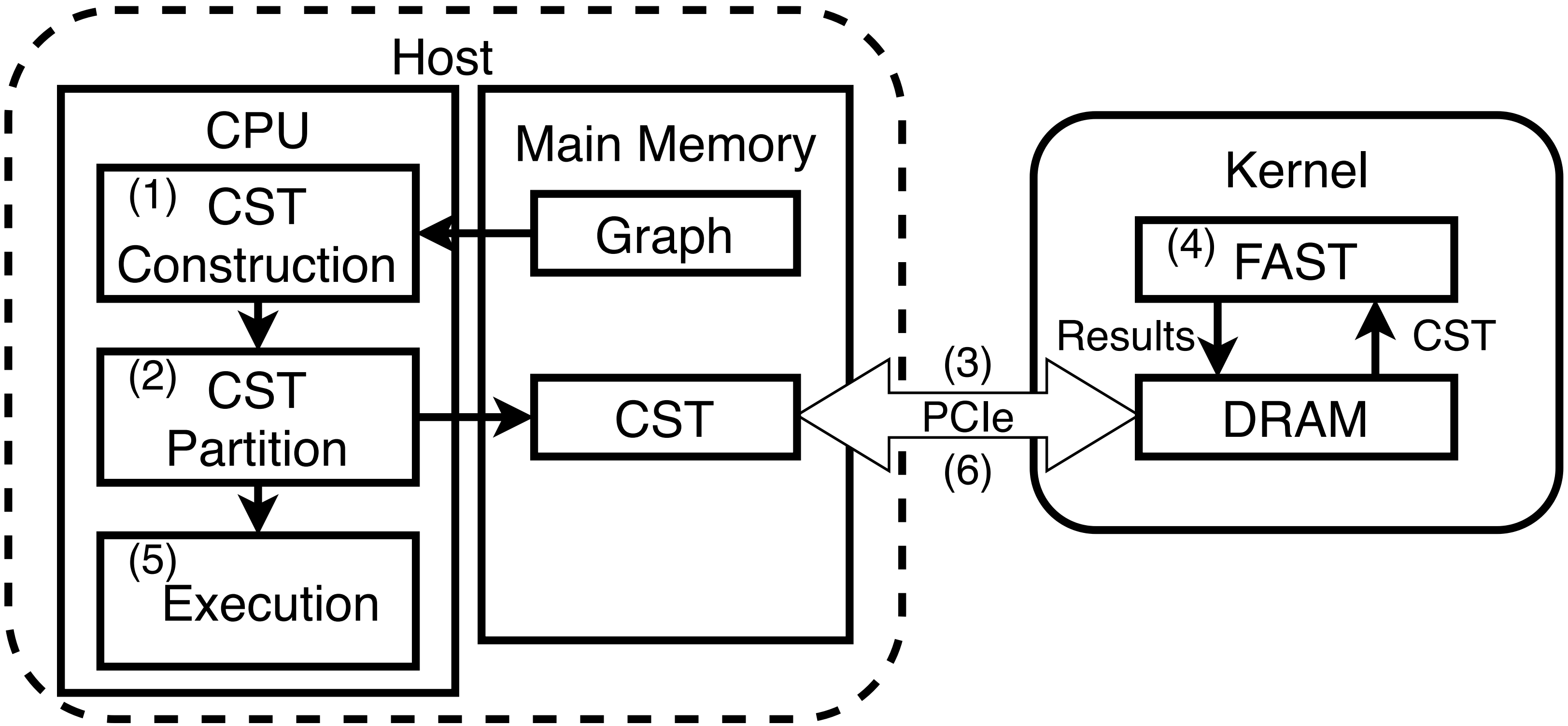
Limited FPGA on-chip memory:

- Small sizes of on-chip memory (BRAM) (tens of megabytes)
- High fetching cost from external memory (DRAM)

→ CST partition & BRAM-only buffer design

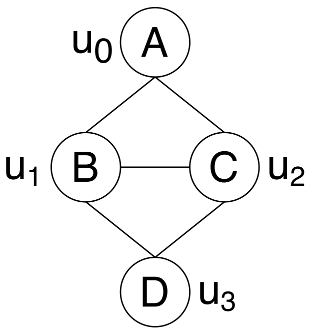
System Overview

System Overview

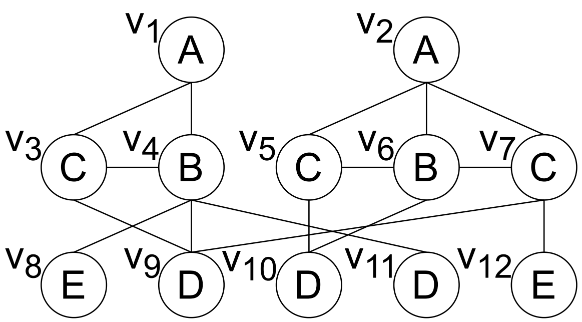


Software Preprocessing

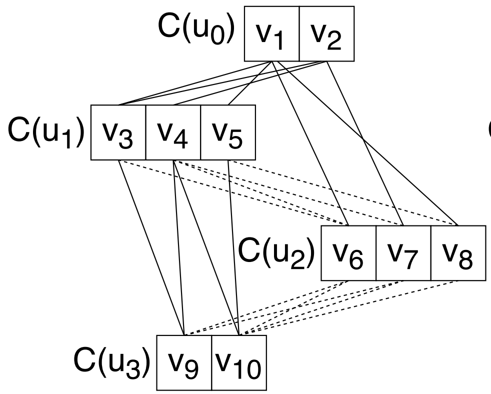
Candidate Search Tree (CST)



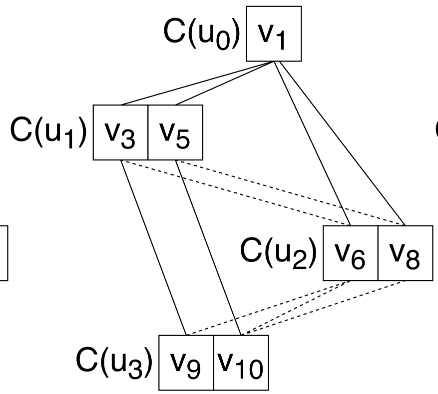
query graph q



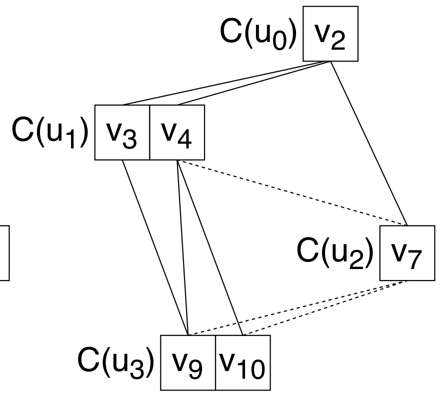
data graph G



CST



1st partition



2nd partition

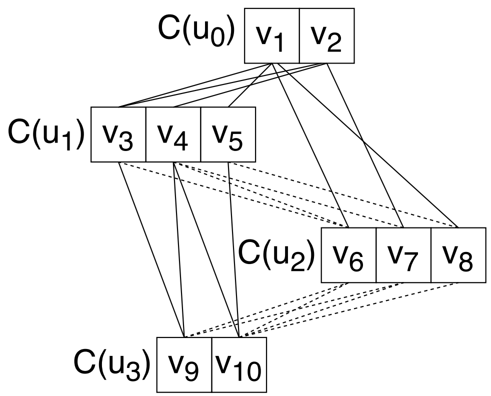
CST

- Complete search space
- Use all edge information in q

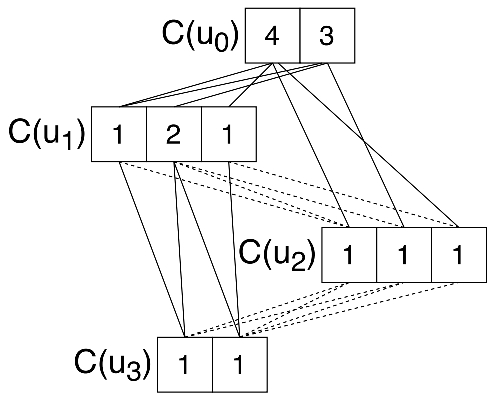
CST Partition:

- Partition if $|CST| > \delta_S$ or $D_{CST} > \delta_D$
- Top-down partition

Workload Estimation



CST



workload

workload estimation:

- $C_u(v) = 1$ if u is a leaf node.
- $C_u(v) = \prod_{u' \in u.child} \sum_{v' \in N_{u'}^{v'}(v)} C_{u'}(v')$ otherwise.
- $W_{CST} = \sum_{v \in C(u_r)} C_{u_r}(v)$.

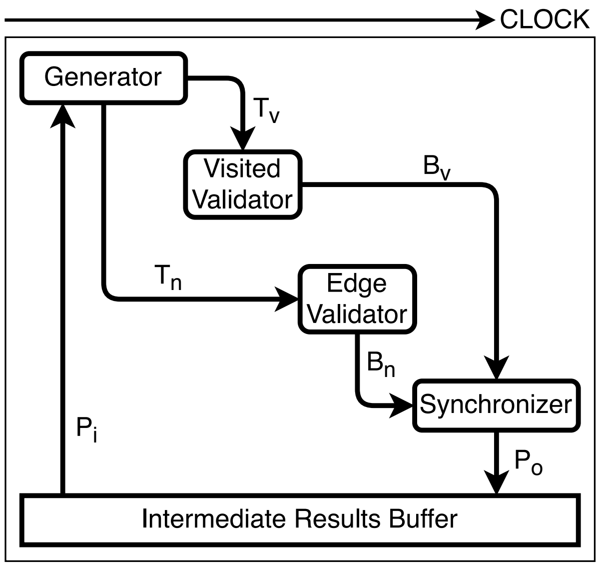
Hardware Implementation

Kernel Design

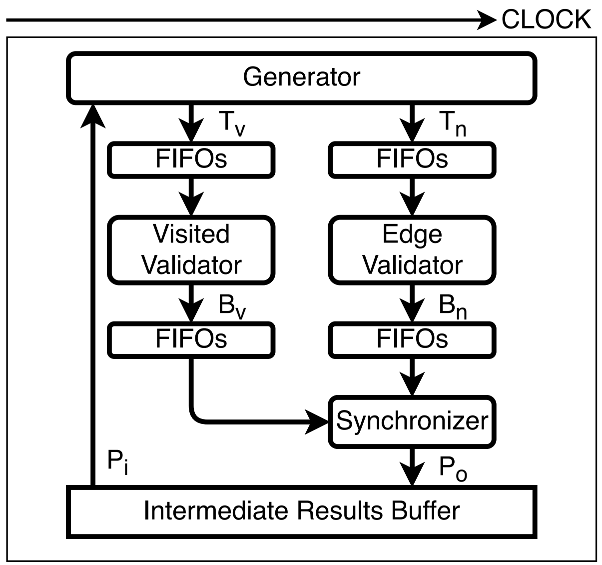
Basic modules

- *Generator*
  - Read N partial results from the buffer
  - Expand each partial result p by mapping next node in the matching order
- Validator:
  - Visited Validator: if p contains repeated nodes
  - Edge Validator: if p has corresponding mapping for the non-tree edge
- Synchronizer:
  - Write back valid results into the buffer or DRAM

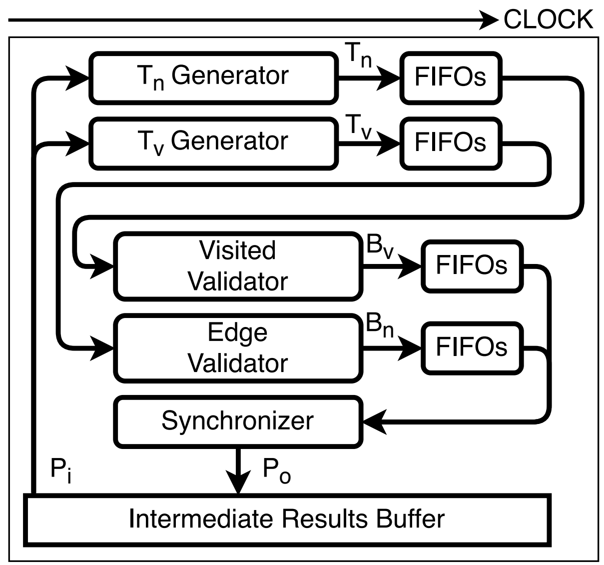
Optimization



Basic version



Task parallelism



Generator Separation

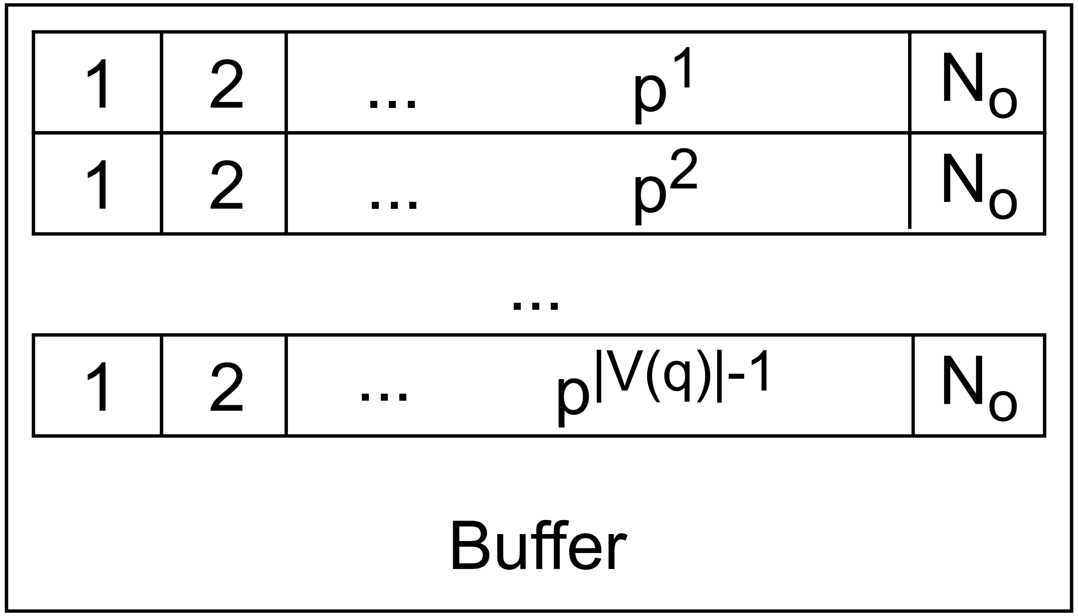
Task parallelism

- Extra buffering (e.g. FIFOs) introduced between the modules

Generator Separation

- Split Generator into t_v Generator and t_n Generator and copy expanded partial results

Buffer Design



Buffer design

- Each round, expand p_n with maximum n (p_n denotes a partial result maps n query nodes)
- For any $n \in [1, |V(q) - 1|]$, the number of p_n does not exceed N_o
- Allocate $(|V(q)| - 1) \times N_o$ space for the buffer

Experiment

Experiment Settings

- ***Host:*** 250GB memory + 10TB disk + 8-core Intel Xeon E5-2620 CPUs
- ***FPGA:*** Xilinx Alveo U200, 64GB DRAM + 35MB BRAM + 300MHz
- ***Datasets:*** *4* data graphs generated by the LDBC social network benchmark (LDBC-SNB), simulating a real social network for 3 years
- ***Queries:*** 8 queries adopted from LDBC-SNB workloads

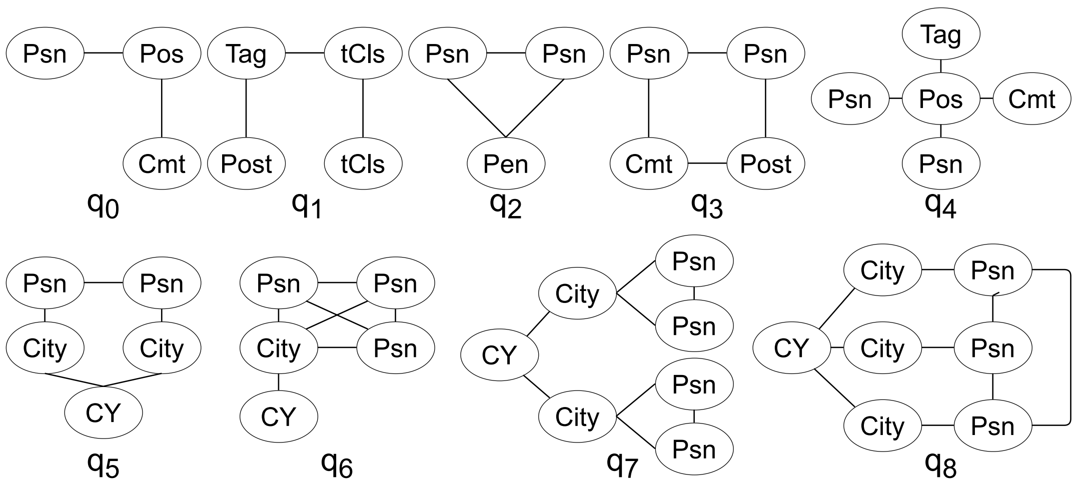
Datasets

Data Graphs

CHARACTERISTICS OF DATASETS.

Name	$\lvert V_G \rvert$	$\lvert E_G \rvert$	$\bar{d}_G$	D_G	# Labels
DG01	3.18M	17.24M	10.84	464,368	11
DG03	9.28M	52.65M	11.34	1,346,287	11
DG10	29.99M	176.48M	11.77	4,282,812	11
DG60	187.11M	1.25B	13.33	26,639,563	11

Query Graphs



Algorithms

CPU-based

- CFL[SIGMOD 2016]、DAF [SIGMOD 2019]、CECI [SIGMOD 2019]

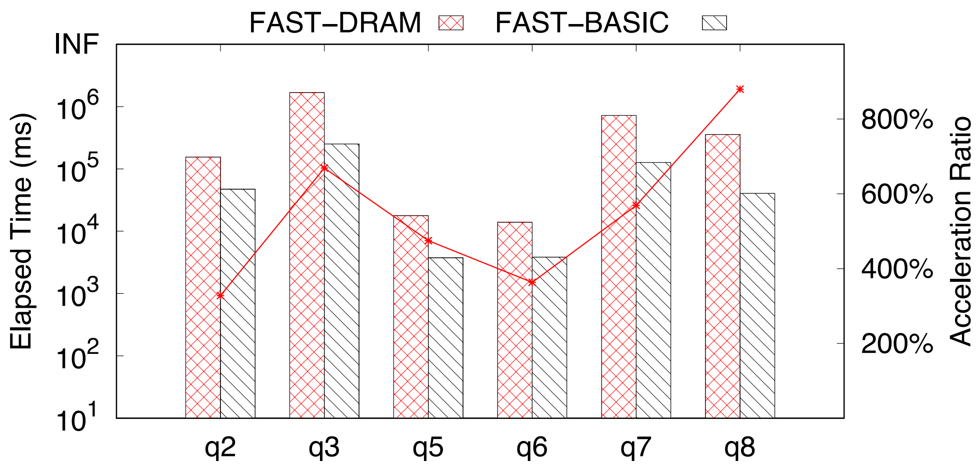
GPU-based

- GpSM [DASFAA 2015]、GSI [ICDE 2020]

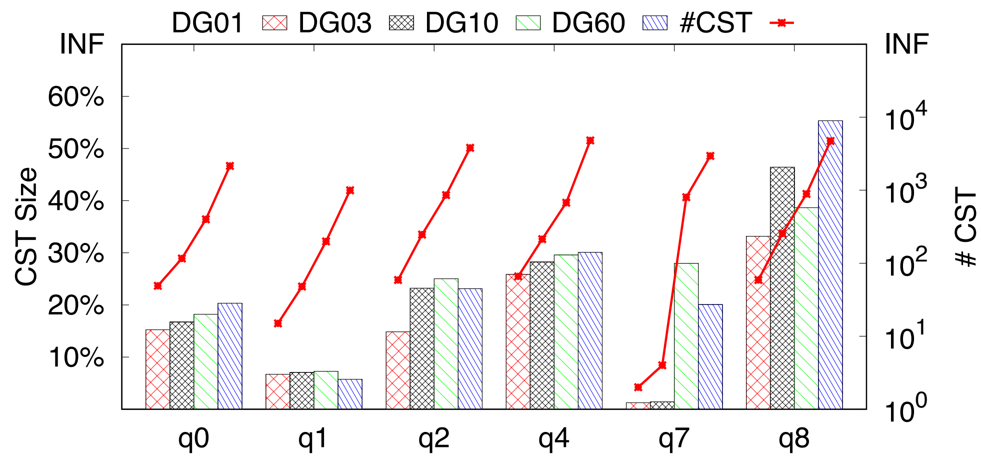
Five versions of FAST

- FAST-DRAM: fetches data from DRAM without any other optimizations.
- FAST-BASIC: fetches data from BRAM without any other optimizations.
- FAST-TASK: FAST-BASIC + task parallelism
- FAST-SEP: FAST-TASK + generator separation
- FAST-SHARE: FAST-SEP + CPU shares tasks

Necessity of CST Partition



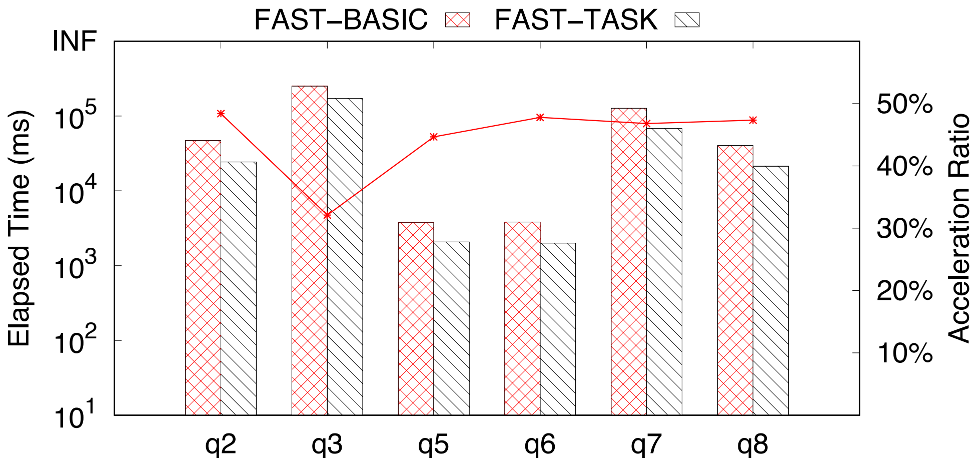
Elapsed time for DG10



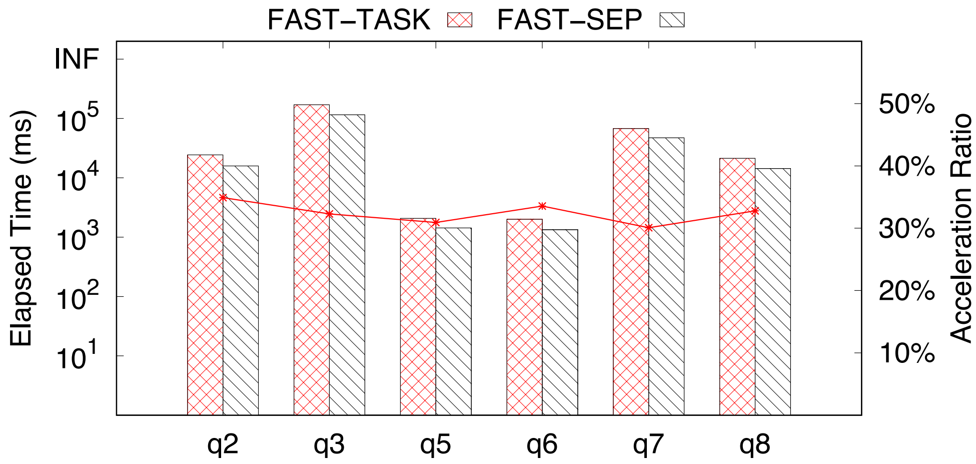
The number and total size of partitioned CST

5.0x speedup & Scalable

Evaluating Optimizations



Effectiveness of Task Parallelism

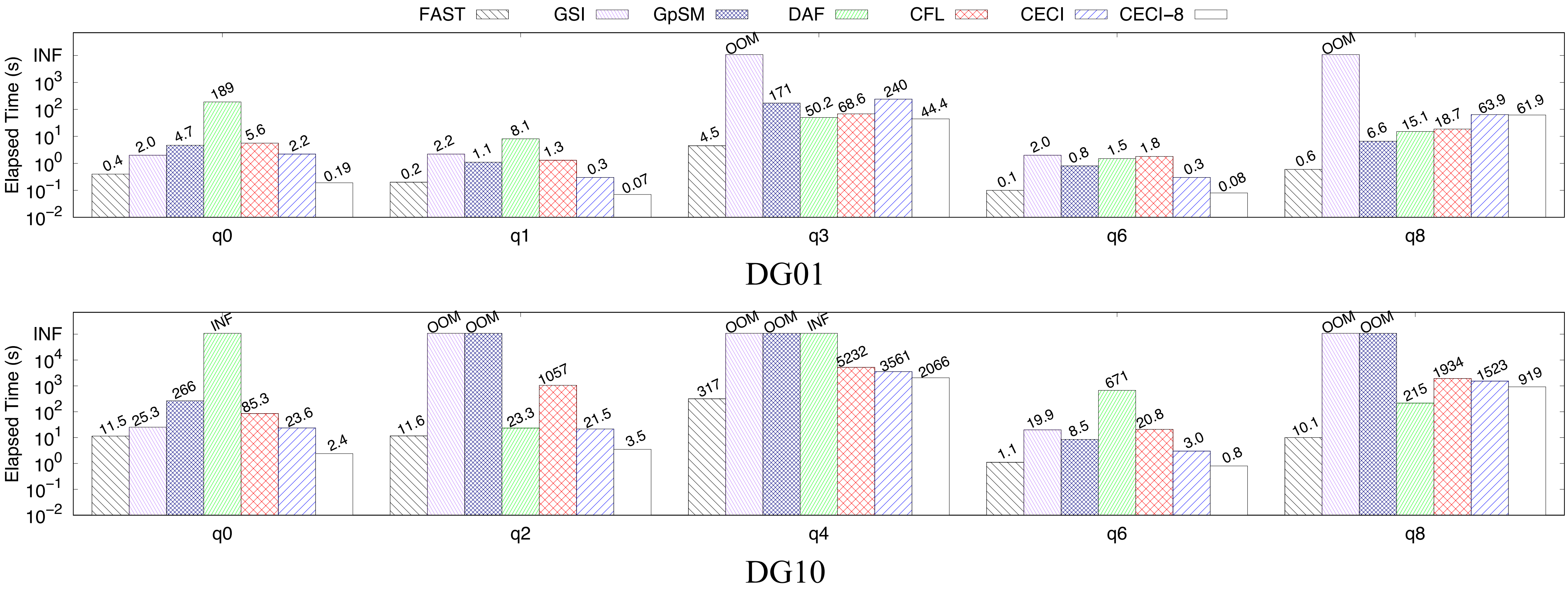


Effectiveness of Generator Separation

Up to 50% improvements

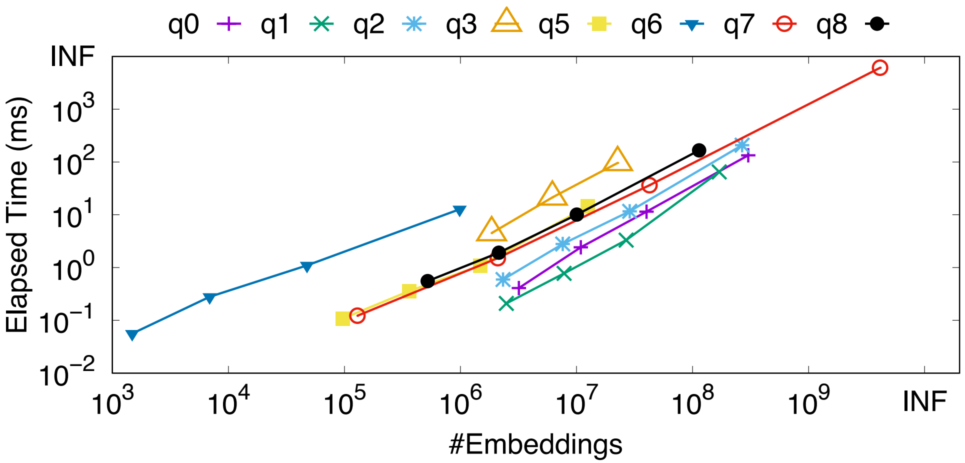
Up to 35% improvements

Comparing with Existing Algorithms

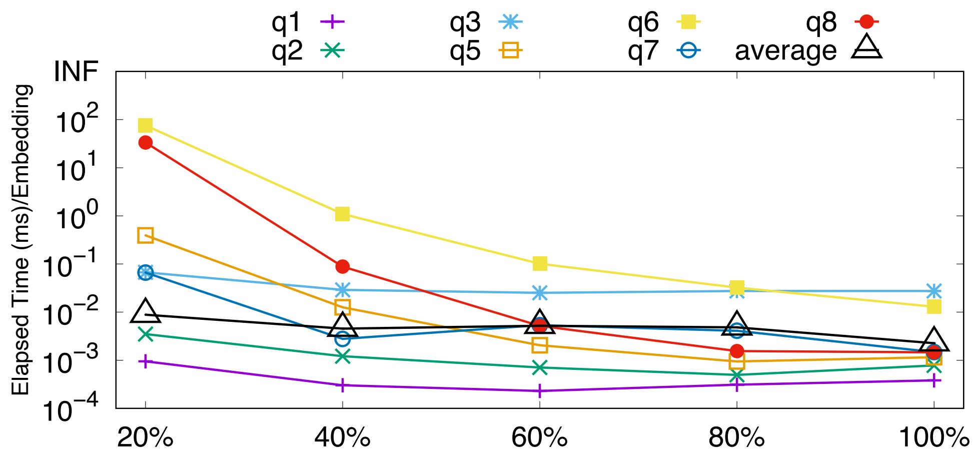


Outperform for all queries & 24.6x average speedup

Scalability



Scalability Testing of FAST (vary x)



Scalability Testing of FAST (vary $|E(G)|$)

Conclusion

Conclusion

The first CPU-FPGA co-designed framework to accelerate subgraph matching.

- a well-designed scheduler + a fully pipelined matching algorithm
- two optimizations with task parallelism and task generator separation.

A BRAM-only matching process to fully utilize FPGA's on-chip memory.

- an auxiliary data structure CST and its efficient partition strategy
- a BRAM-only partial results matching

Extensive experiments using the industrial-standard LDBC benchmark.

- outperforms the state-of-the-art algorithms by orders of magnitude
- the only algorithm that can scale to the billion-scale graph on a single machine

Thank you!