Subgraph Matching

20240102

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*

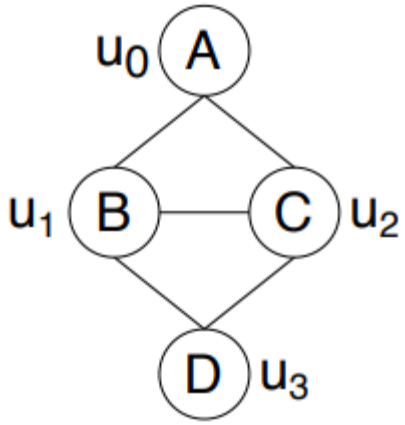
xinjin@stu.ecnu.edu.cn, {zyang,lxue}@cse.unsw.edu.au,
syyang@gzhu.edu.cn, lu.qin@uts.edu.au, you.peng@unsw.edu.au

Challenges

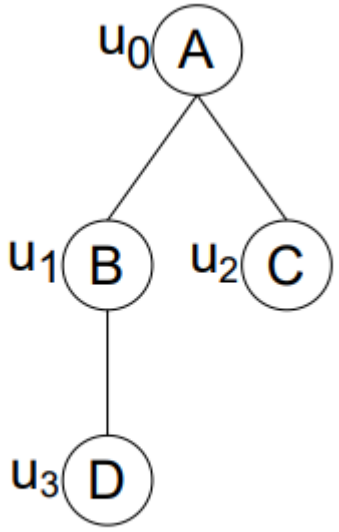
- Strictly pipelined design on FPGA.
- Limited FPGA on-chip memory.
  - BRAM
  - DRAM (higher read latency)

CST Structure

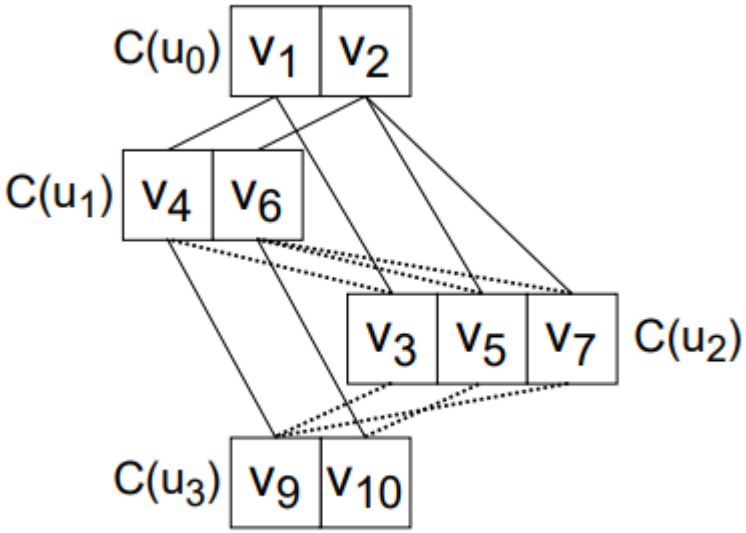
- Candidate Search Tree
- A complete search space



(a) Query graph



(a) BFS Tree t_q

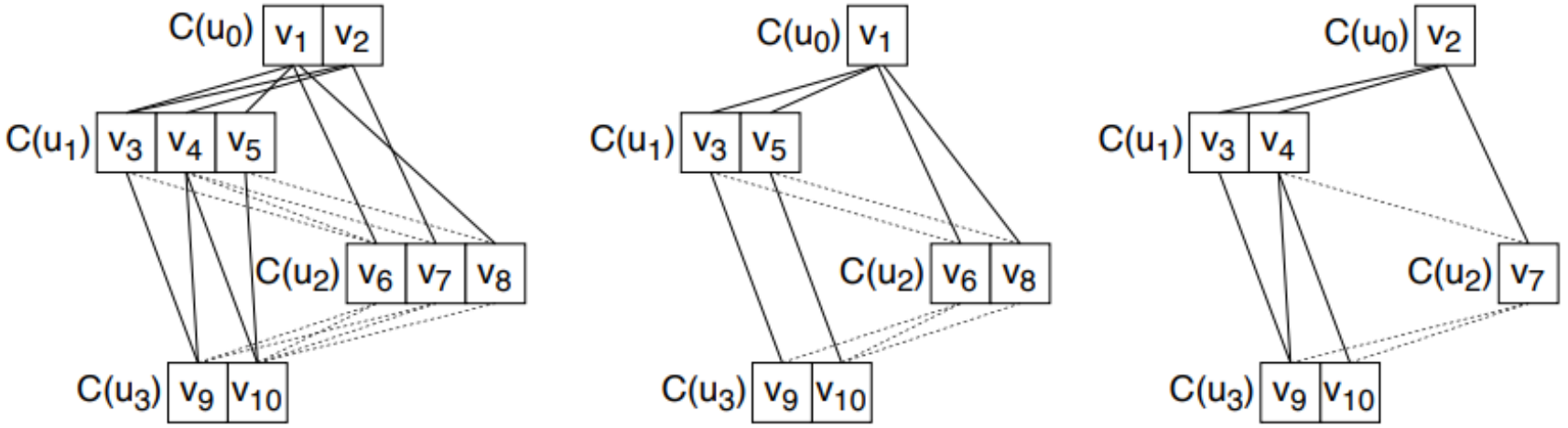


(b) CST

CST Partition

Limited by on-chip resources on FPGAs, CST is often too large to be fully loaded into BRAM.

Partition $C(u_0) = \{v_1, v_2\}$ into k parts evenly. (k=2)



(a) Initial CST with $k = 2$ (b) 1st partition of CST (c) 2nd partition of CST

Fig. 4. Running Example of Scheduling

No overlap of search space between **(b)** and **(c)**.
(在 Enumeration 阶段的 search space).

Overview

(2) Limited by FPGA on-chip resources, CST is often too large to be fully loaded into BRAM. The host side partitions CST to satisfy the size constraint.

(5) CPU shares a small portion of matching tasks to improve the overall throughput. (Workload estimation: 连同 false positive 一起考虑, 使用 dynamic programming)

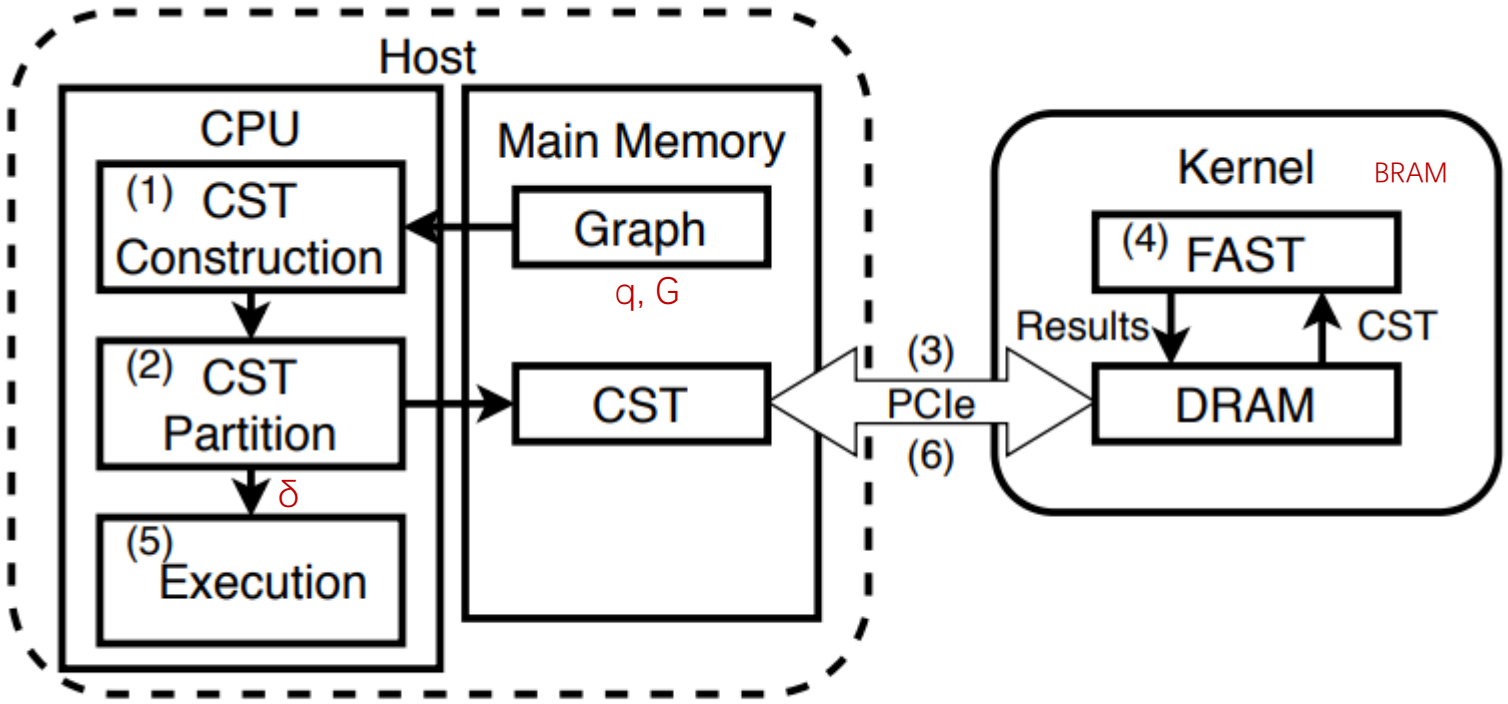


Fig. 2. The overall system architecture

Pipelined Matching Algorithm

- Typical backtracking algorithms
  - Sequential design cannot be pipelined because of data dependencies among iterations.
- Decompose the matching process into 3 steps
  - **Generator** expands partial results by matching the next vertex;
  - **Validator** verifies whether a new partial result is valid;
  - **Synchronizer** collects results.

Algorithm 4: FAST(CST, $\mathcal{O}$)

Input: CST, $\mathcal{O}$
Output: $\mathcal{M}$

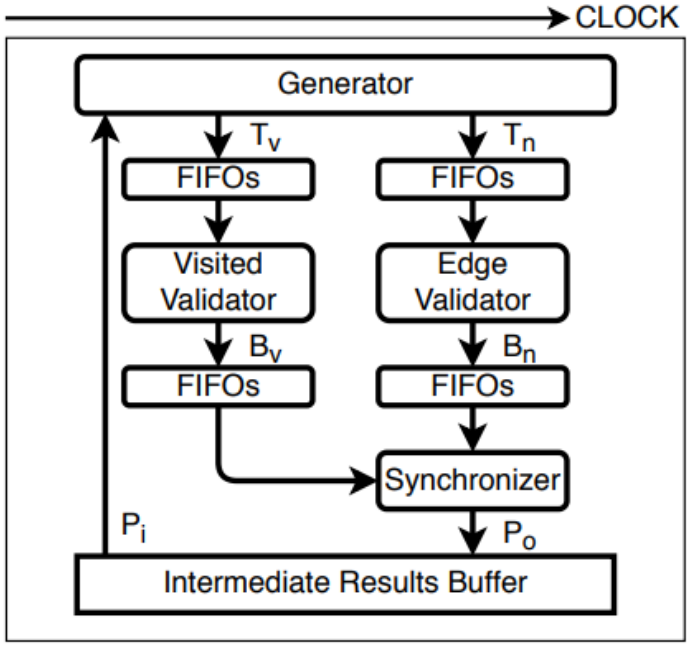
1 $\mathcal{M} \leftarrow \emptyset$; $\mathcal{P} \leftarrow \emptyset$;
2 **foreach** *candidate* v *of root vertex* ***pipeline*** **do**
3 $\quad \mathcal{P}.push(\{v\})$;
4 **while** $\mathcal{P} \neq \emptyset$ **do**
5 $\quad \mathcal{P}_o, \mathcal{T}_v, \mathcal{T}_n \leftarrow Generator(\mathcal{P}, \text{CST}, \mathcal{O})$;
6 $\quad \mathcal{B}_v \leftarrow VisitedValidator(\mathcal{T}_v)$;
7 $\quad \mathcal{B}_n \leftarrow EdgeValidator(\text{CST}, \mathcal{T}_n)$;
8 $\quad Synchronizer(\mathcal{M}, \mathcal{P}, \mathcal{P}_o, \mathcal{B}_v, \mathcal{B}_n)$;
9 **return** $\mathcal{M}$

$\mathcal{P}$ records partial results.
$\mathcal{M}$ records complete results.
Visited validation tasks $\mathcal{T}_v$.
Edge validation tasks $\mathcal{T}_n$.
Validation bits $\mathcal{B}_v$ and $\mathcal{B}_n$.

Task Parallelism

- Buffer: FIFOs on FPGA
- In contrast to loop parallelism, when task parallelism is deployed, different execution modules are allowed to operate simultaneously.

- 枚举时需要内存大，故 CST Partition
- 不是因为图本身放不进 BRAM



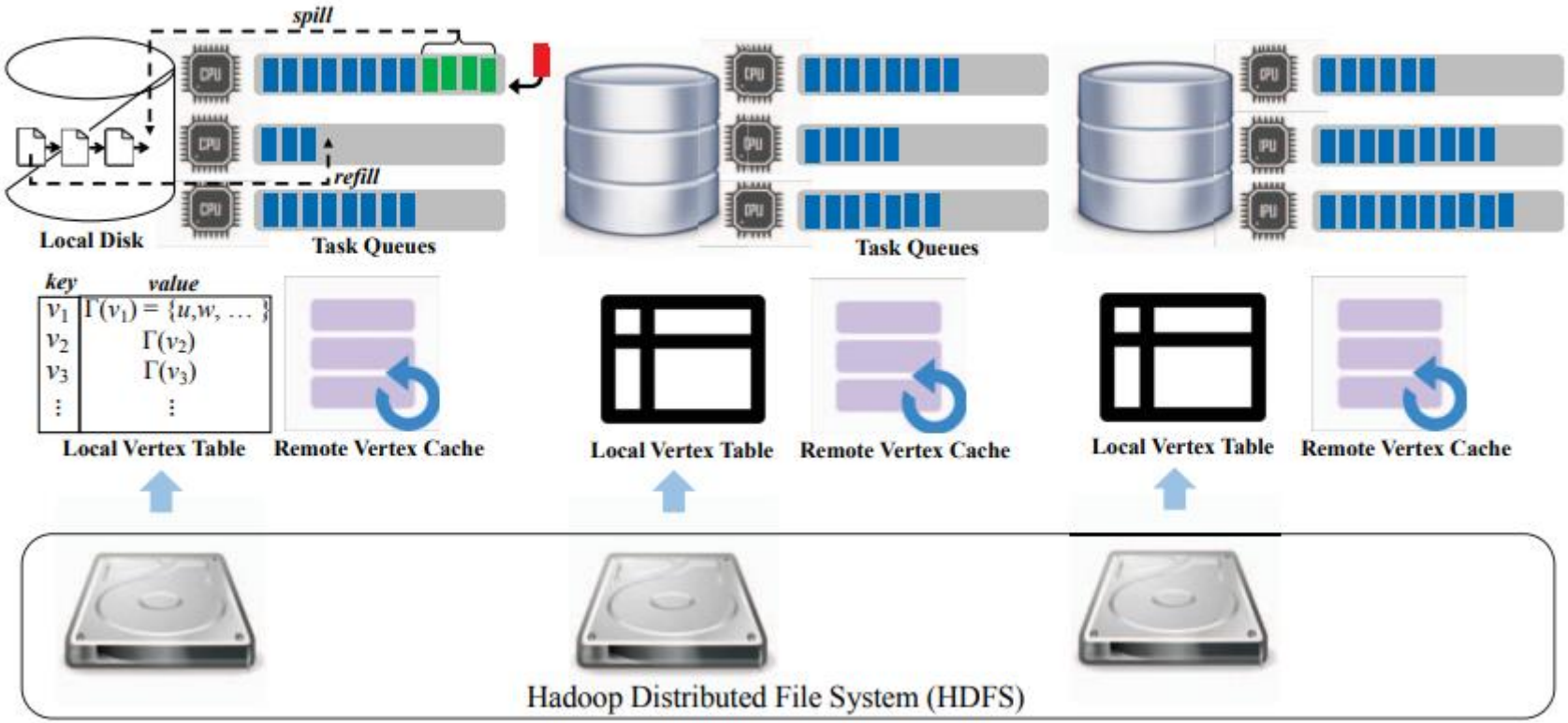
G-thinker: A Distributed Framework for Mining Subgraphs in a Big Graph

Da Yan[*1], Guimu Guo[*2], Md Mashiur Rahman Chowdhury[*3],
M. Tamer Özsu[†4], Wei-Shinn Ku[‡5], John C.S. Lui[+6]

[*]*University of Alabama at Birmingham,* {[1]yanda, [2]guimuguo, [3]mashiur}@uab.edu
[†]*University of Waterloo,* [4]tamer.ozsu@uwaterloo.ca
[‡]*Auburn University,* [5]weishinn@auburn.edu
[+]*The Chinese University of Hong Kong,* [6]cslui@cse.cuhk.edu.hk

- The first truly CPU-bound distributed framework G-thinker
- CPU-bound, not IO-bound

Overview


spill
refill
Local Disk
Task Queues
Task Queues
key
value
v_1 $\Gamma(v_1) = \{u, w, \ldots\}$
v_2 $\Gamma(v_2)$
v_3 $\Gamma(v_3)$
Local Vertex Table
Remote Vertex Cache
Local Vertex Table
Remote Vertex Cache
Local Vertex Table
Remote Vertex Cache
Hadoop Distributed File System (HDFS)

GPU-Accelerated Subgraph Enumeration on Partitioned Graphs

Wentian Guo, Yuchen Li*, Mo Sha, Bingsheng He, Xiaokui Xiao, Kian-Lee Tan
National University of Singapore, *Singapore Management University
{wentian,sham,hebs,tankl}@comp.nus.edu.sg
xkxiao@nus.edu.sg,*yuchenli@smu.edu.sg

Related Works

- CPU-based
  - Single-machine: slow
  - Distributed: costly
- GPU-based
  - Cannot scale to data graph larger than GPU memory

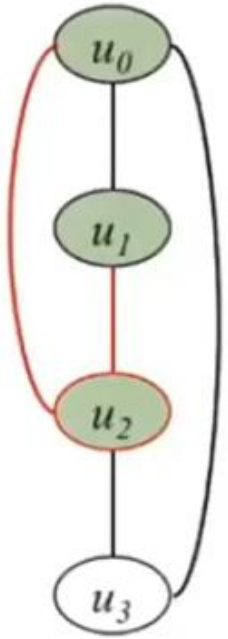
Partition Based Enumeration (PBE)

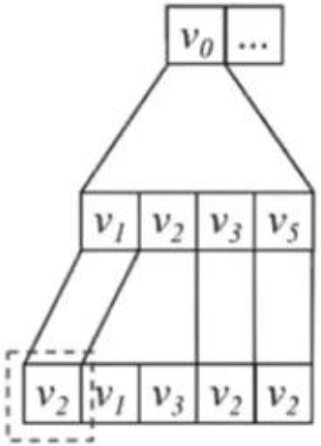
- Divide the graph into multiple partitions, each of which can fit into the device memory.

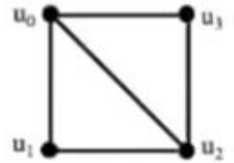
- Intra-partition search: avoid IO for subgraph within the partition
- Inter-partition search: challenging as it can cause PCI-e traffic
  - This paper propose a different approach

Intra-partition Search: BFS

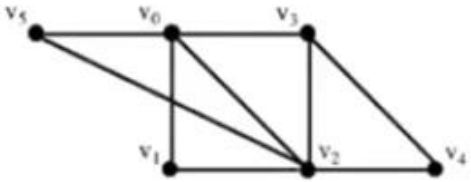
这是 BSP (Bulk synchronous parallel)
此外还有 ASP



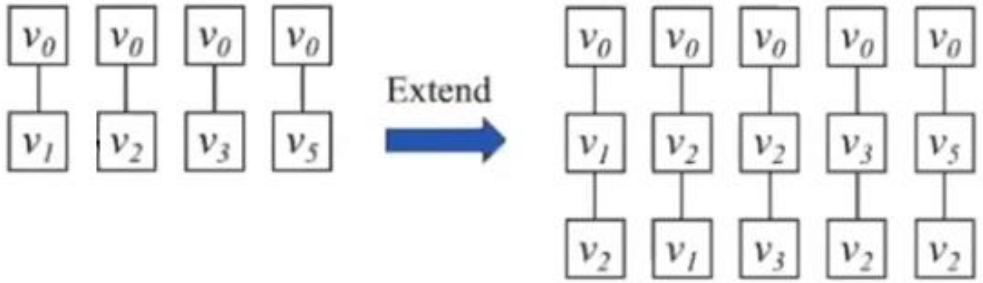




Pattern graph P



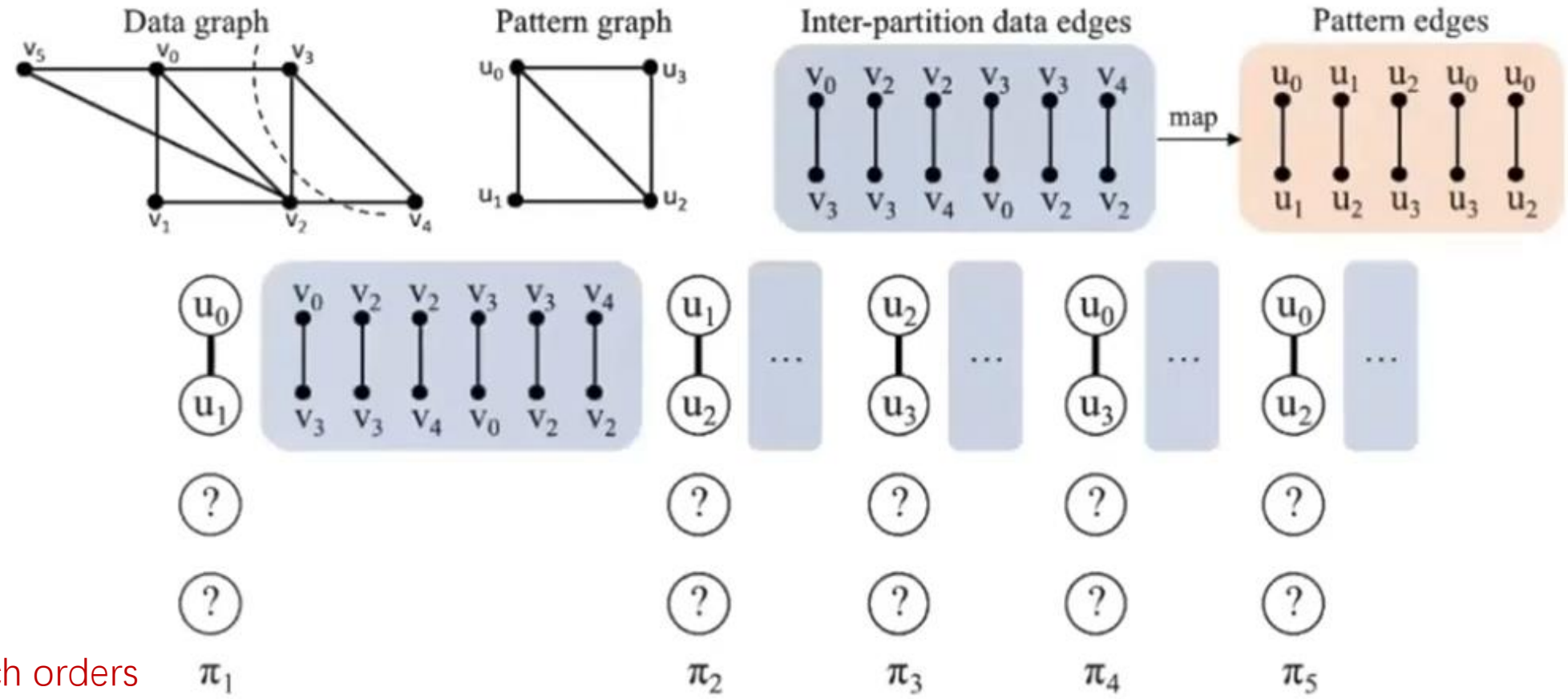
Data graph G



1. Select match order
2. Iteratively extend the partial instances to match one more pattern vertex

Inter-partition Search

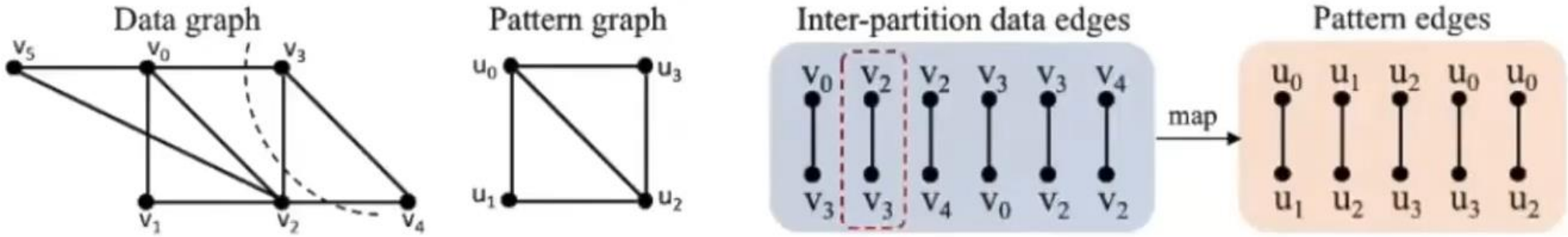
- *Pruning stage: map inter-partition data edges to each pattern edge*

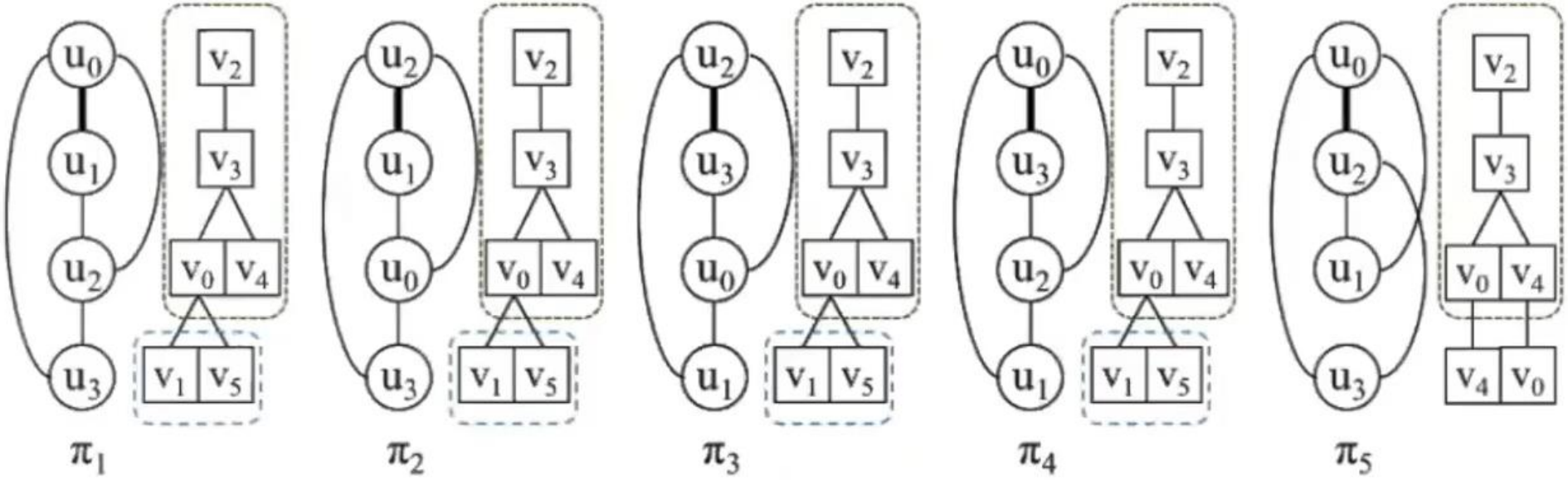


Multiple match orders

Inter-partition Search

- Pruning stage: map inter-partition data edges to each pattern edge
- *Enumeration stage: iteratively extend the partial instances*





Shared Execution

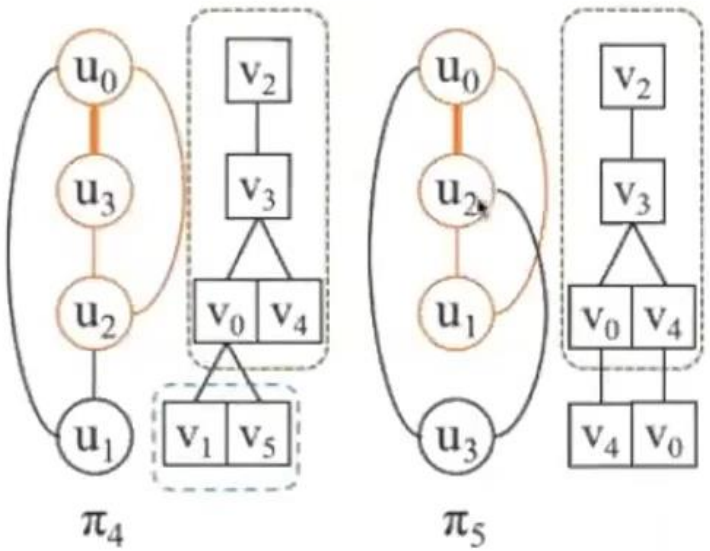
Shared execution can group multiple match orders together and enumerate the partial instances altogether

- The grouping is possible when two match orders π_1 and π_2 are **prefix-equivalent** at level L, i.e., the induced subgraph $P_L^{\pi_1}$ and $P_L^{\pi_2}$ are isomorphic

P_L^{π} is the induced subgraph on the vertex set $\{\pi(1) \cdots \pi(L)\}$

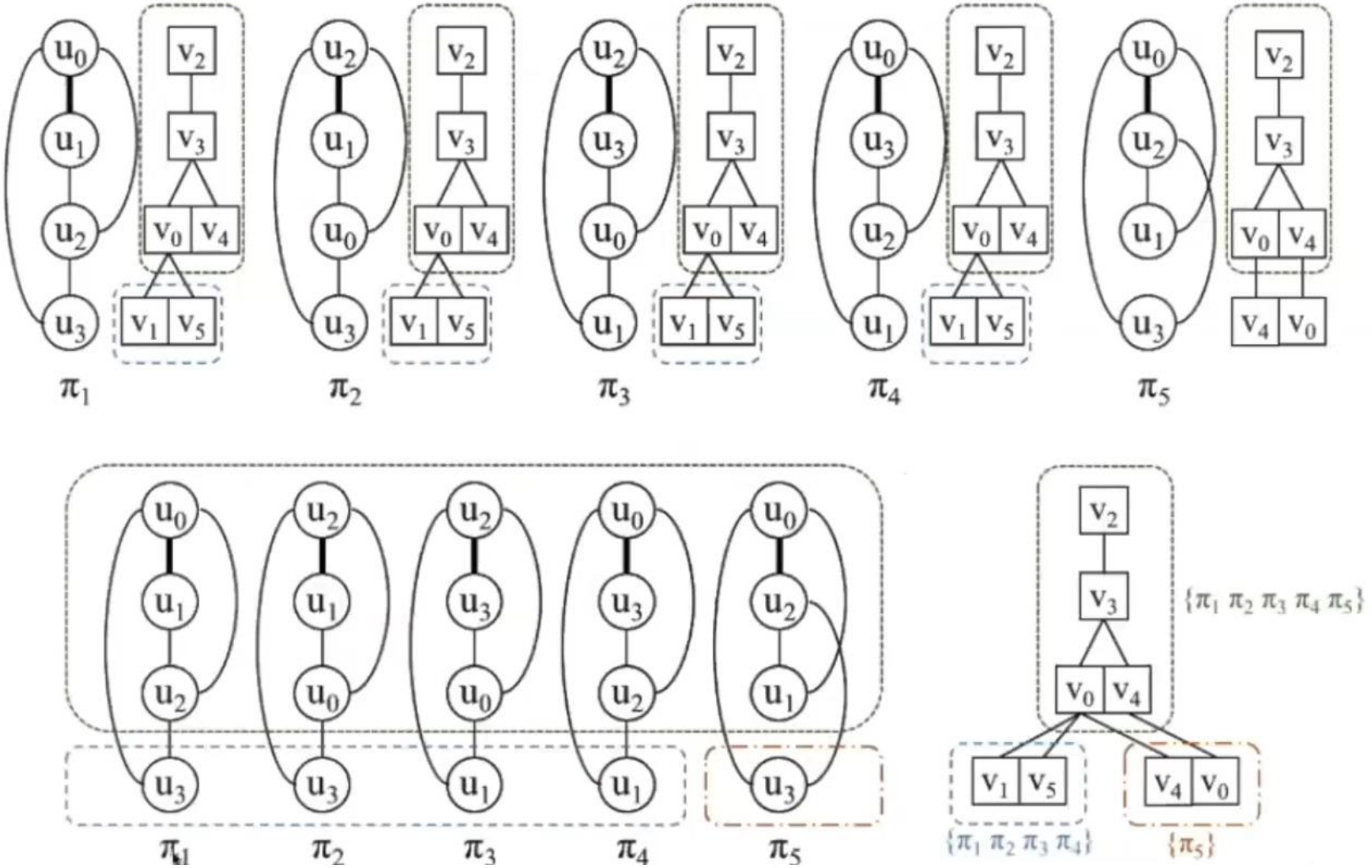
For different match orders, the same cross-partition subgraphs could be repeatedly generated to match the pattern vertices.

Redundant search happens.



Example: π_4 and π_5 are prefix-equivalent at level 3

Shared Execution



Efficient GPU-Accelerated Subgraph Matching

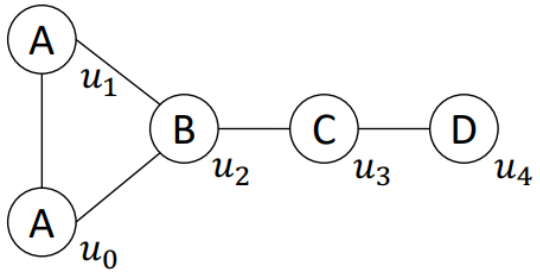
XIBO SUN, The Hong Kong University of Science and Technology, Hong Kong SAR
QIONG LUO, The Hong Kong University of Science and Technology, Hong Kong SAR and The Hong Kong University of Science and Technology (Guangzhou), China

Subgraph Matching on CPU and GPU

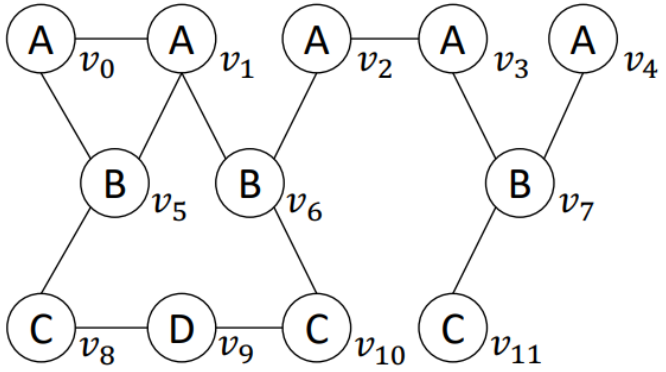
- CPU-based algorithms are highly optimized
  - Matching on some graphs is still time-consuming

- Some researchers start to adopt modern hardware, including GPU
  - 1. Ineffective filtering and ordering
  - 2. Memory-inefficient BFS enumeration
  - This paper addresses above two problems

Trie (CSR)

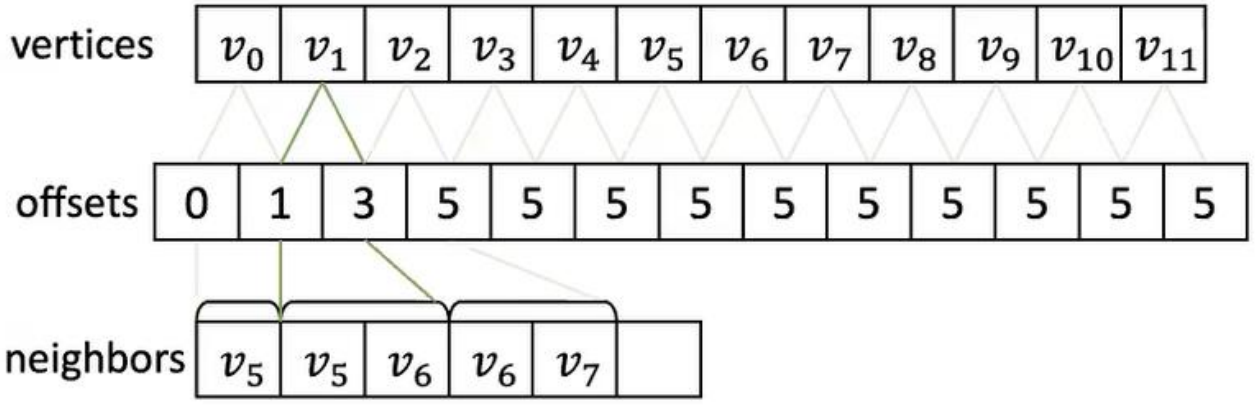


(a) Query graph Q.



(b) Data graph G.

- Trie (CSR) is commonly used to store a relation,
  - Good for retrieving neighbors of a vertex
  - **Many vertices and offsets for big relations**
  - **Expensive to update**



u_0	u_1
v_0	v_1
v_1	v_0
v_2	v_3
v_3	v_2

u_0	u_2
v_0	v_5
v_1	v_5
v_1	v_6
v_2	v_6
v_3	v_7

u_1	u_2
v_0	v_5
v_1	v_5
v_1	v_6
v_2	v_6
v_3	v_7

u_2	u_3
v_5	v_8
v_6	v_{10}

u_3	u_4
v_8	v_9
v_{10}	v_9

Index

Cuckoo Hashing

Cuckoo Hashing

procedure insert(x)
  if lookup(x) **then return**
  loop MaxLoop **times**
    if $T_1[h_1(x)] = \perp$ **then** { $T_1[h_1(x)] \leftarrow x$; **return** }
    $x \leftrightarrow T_1[h_1(x)]$
    if $T_2[h_2(x)] = \perp$ **then** { $T_2[h_2(x)] \leftarrow x$; **return** }
    $x \leftrightarrow T_2[h_2(x)]$
  end loop
  rehash(); insert(x)
end

A group of m independent hash functions, each corresponding to a separate hash table. (m=2)

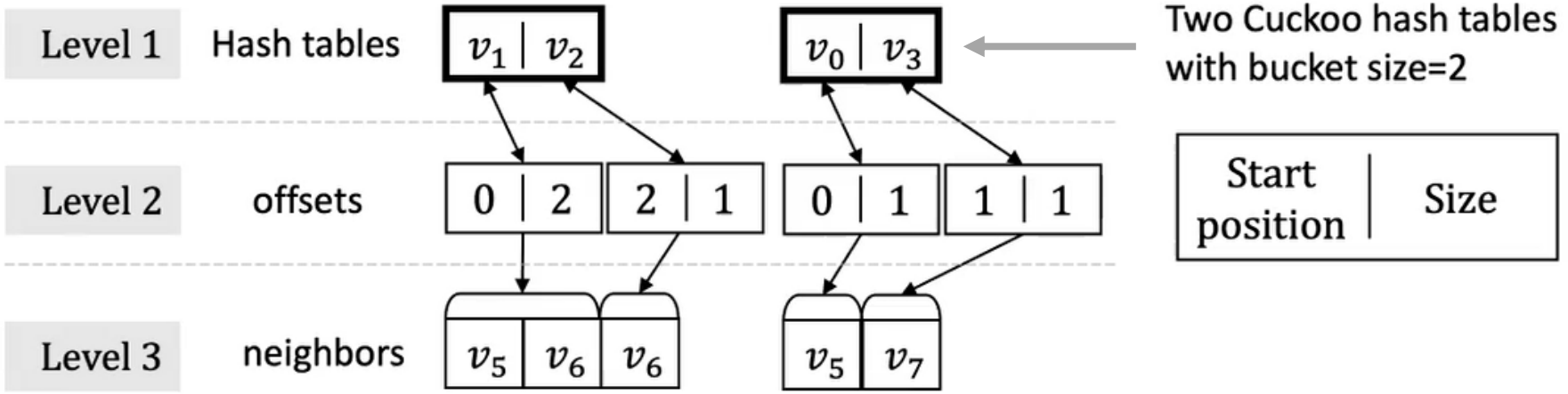
Cuckoo Tries

To support dynamic maintenance of candidates for filtering.
It is difficult to maintain dynamic data structures on the GPU.

- Each relation is implemented as a set of Cuckoo tries
  - Level 1 corresponds to Cuckoo hash tables
    - **Multiple** hash tables, **Worst case $O(1)$** access time, no warp divergence
    - **Bucketize elements** to fully utilize the high memory bandwidth
  - Level 2 indicates the position of each neighbor set
    - On filtering, the modification of one neighbor set does not affect the others
  - Level 3 stores the neighbor sets consecutively
- Efficient batch-insertion, deletion, and search process

Hash 本质是空间的映射

u_0	u_2
v_0	v_5
v_1	v_5
v_1	v_6
v_2	v_6
v_3	v_7

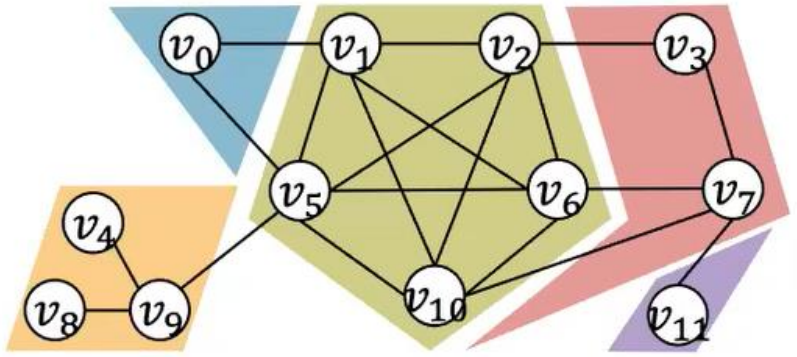


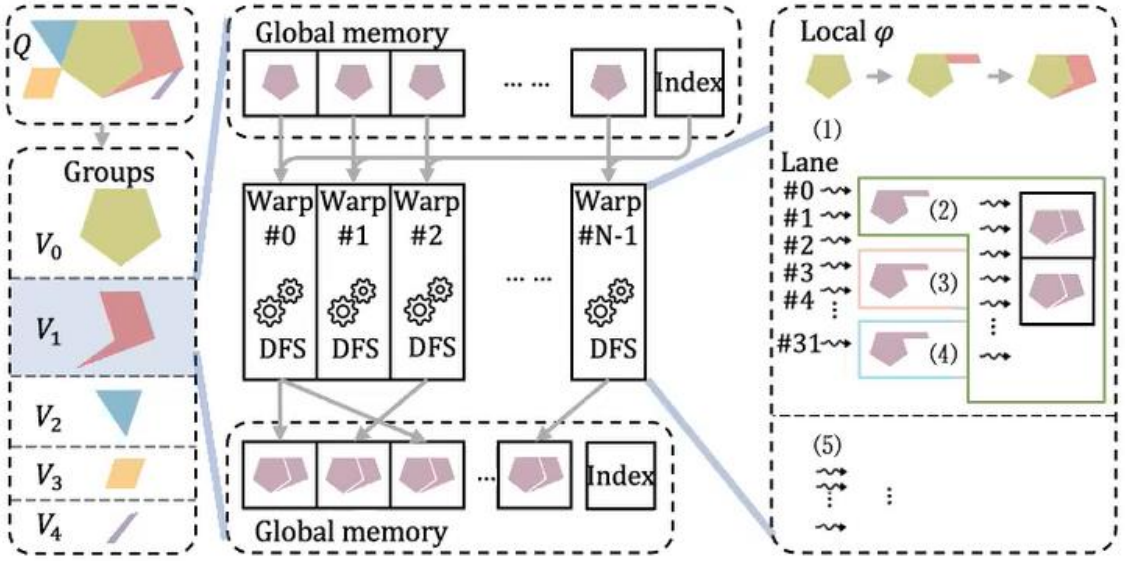
Enumeration: Parallel BFS and DFS

- Parallel-BFS enumeration
  - Utilized by most GPU-based algorithms
  - In each step, all partial results are extended by one vertex
  - A warp extends a partial match of size m to all partial matches of size $m + 1$
  - Large memory consumption and many memory accesses
- Parallel-DFS enumeration
  - A warp obtains an edge and get all complete matches containing the edge
  - Alleviate the memory issues
  - Severe load imbalance due to the search space

Hybrid BFS-DFS Enumeration

- Hybrid parallel BFS-DFS extension method
  - Organize vertices in Q into groups $(V_0, V_1, \ldots, V_n)$ based on the structure of Q
    - Dense vertex, then sparse vertices, and finally tree vertices
  - Extend vertices within the same group in DFS
  - Write all partial results when a group is finished (BFS)





- 结束