Accelerating Regular Path Queries using FPGA

Kento Miura, Toshiyuki Amagasa, Hiroyuki Kitagawa

University of Tsukuba

Kitagawa & Amagasa
Data Engineering

Outline

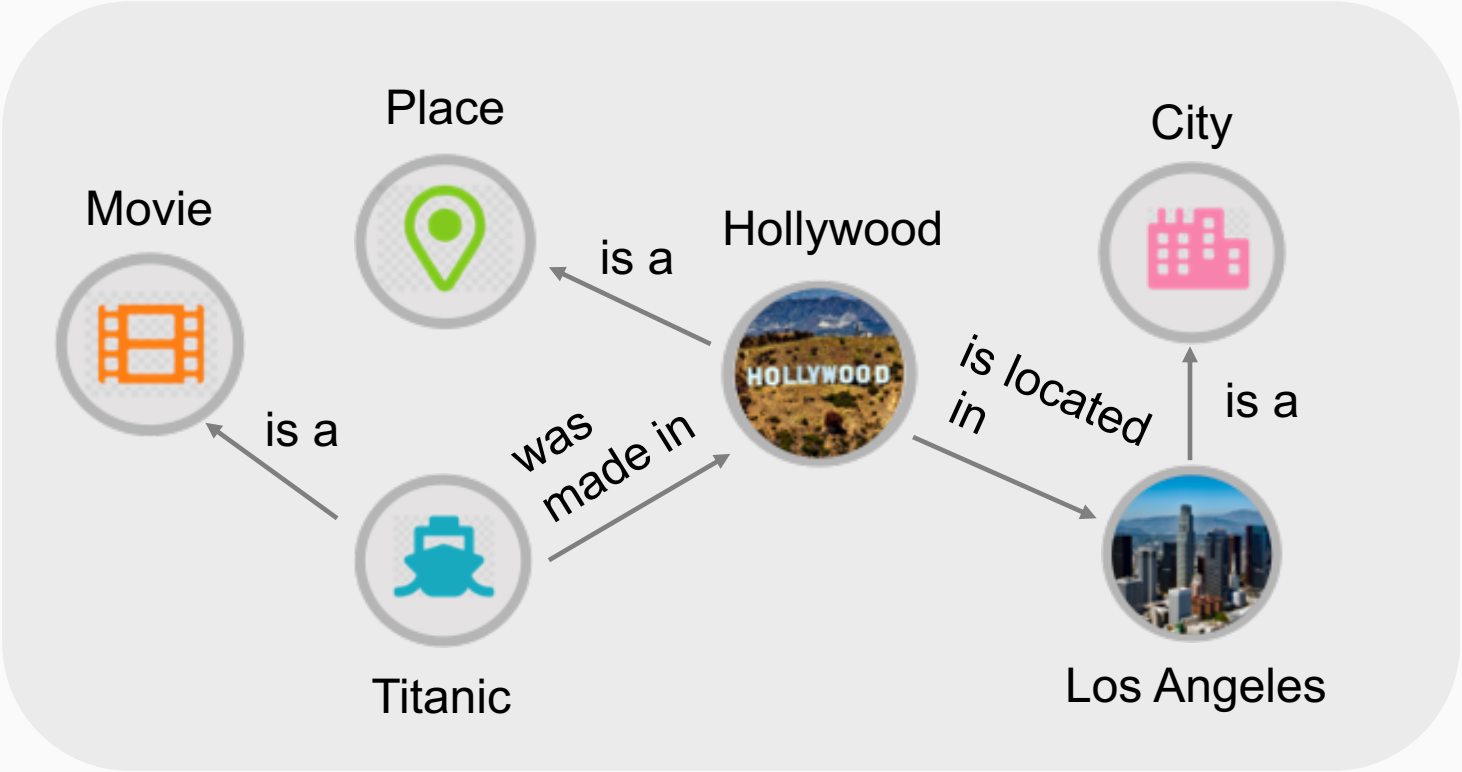
- Background and Objective
- Related Work
- The proposed method
- Experiment and Discussion
- Conclusion

Outline

- **Background and Objective**
- Related Work
- The proposed method
- Experiment and Discussion
- Conclusion

Labeled graph

- Graph with labels on vertices or edges
  - Labeled graphs are used in many fields



Labeled graph example

Regular Path Query (RPQ)

- A query to retrieve pairs of vertices that are reachable via such paths whose label sequences conform to the user specified regular expression

Query example

RPQ : $a \circ b \circ b$

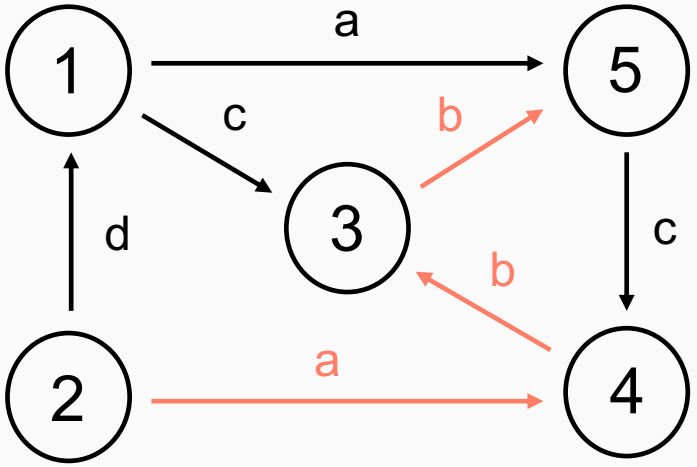
Find paths in order of $a \rightarrow b \rightarrow b$

$= \{(2,5)\}$

RPQ : $(a \circ c^{-1}) \cup (c^{1,2})$

Find paths $a \rightarrow c^{-1}$ (c follows in inverse order) or k $(1 \leq k \leq 2)$ repetitions of c

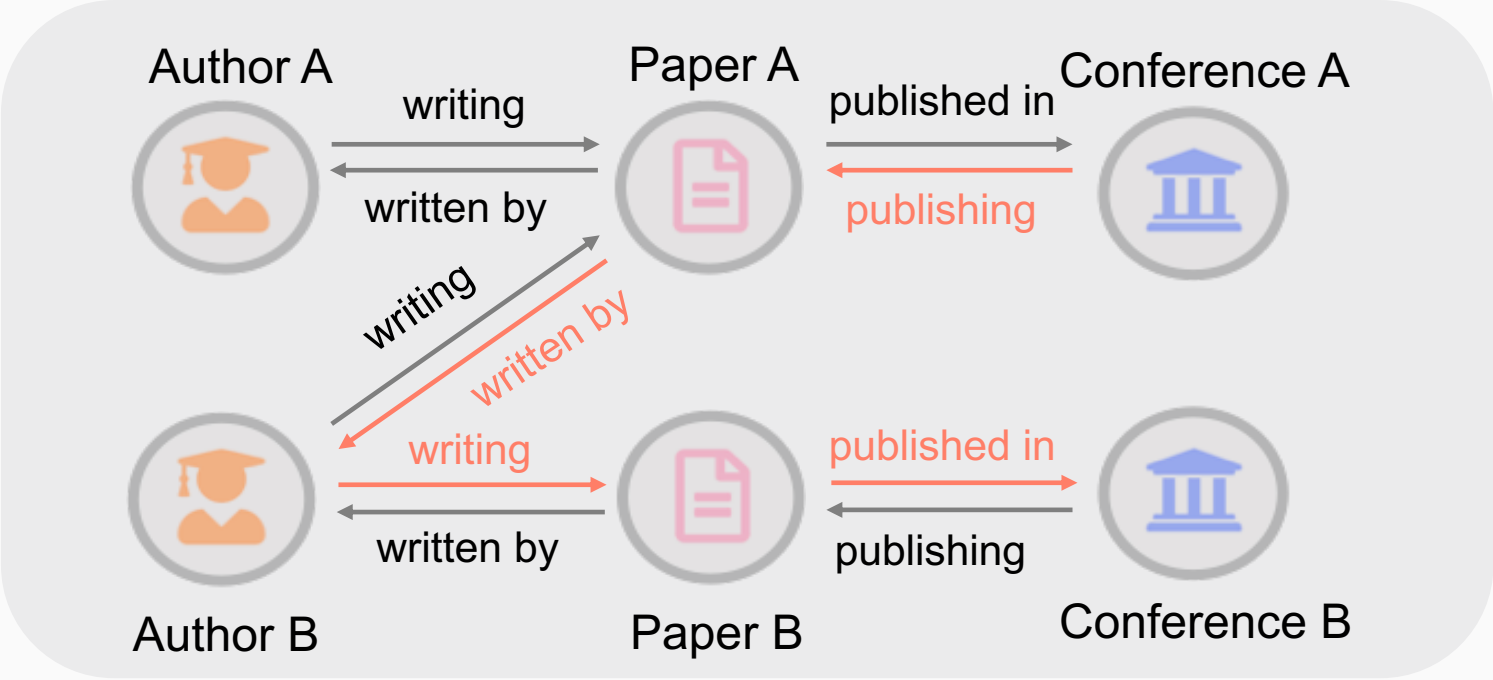
$= \{(1,4), (2,3)\}$



In this research, we focused on RPQs with only composition operation ($\circ$)

Actual RPQ usage example

- Example. What other conferences have authors who submitted to conference A?



Paper submission information graph

(Start vertex = Conference A)

RPQ : publishing $\circ$ *written by* $\circ$ *writing* $\circ$ *published in*

Problem

Running an RPQ on a large graph takes a lot of time

Acceleration of RPQ processing is required

[1] Mendelzon, A.O., Wood, P.T.: Finding regular simple paths in graph databases

FPGA

- FPGA is a device that allows users to implement arbitrary logic circuits by programming dynamically
- The implemented circuit operates in parallel

  → FPGA can process a large amount of data at high speed by **parallel processing** and **pipelining**

- High-level synthesis (HLS) has enabled FPGA programming using C, C++, OpenCL

In order to speedup RPQ

- Main approach
  - RPQ processing can be divided into multiple stages and **can be pipelined**

We considered that RPQ could be processed efficiently by FPGA.

- Objective
  - **Acceleration of RPQ processing using FPGA**

Outline

- Background and Objective
- **Related Work**
- The proposed method
- Experiment and Discussion
- Conclusion

Examples of speedup using FPGA

- Graphicionado [Ham et al. 2016]

  FPGA accelerator for graph analysis processing such as PageRank and collaborative filtering

  About **10~100 times faster** than CPU only

- Sidler et al. 2017

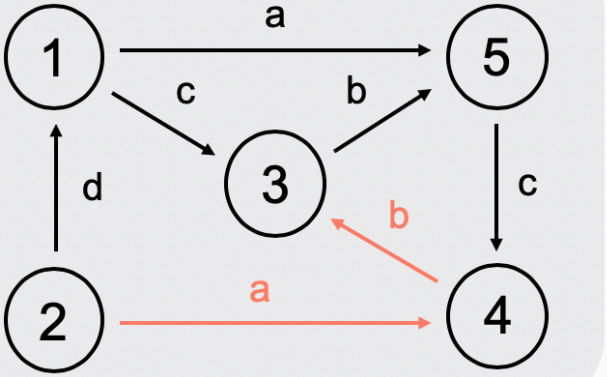
  Accelerate queries that include regular expressions to databases using CPU-FPGA architecture

  1.76 to 6.54 times faster than running in software

Research on RPQ speedup

Path Index

- All occurrences of paths up to length k are extracted and stored in corresponding indexes



Target Graph

k = 1

Path	Source	Destination
a	1	5
a	2	4
b	3	5
b	4	3
c	1	3
c	2	1
c	5	4

k = 2

Path	Source	Destination
$a \circ b$	2	3
$a \circ c$	1	4
$b \circ b$	4	5
$b \circ c$	3	4
$c \circ a$	2	5
$c \circ b$	1	5
$c \circ b$	5	3
$c \circ c$	2	3

Path Index (k = 2)

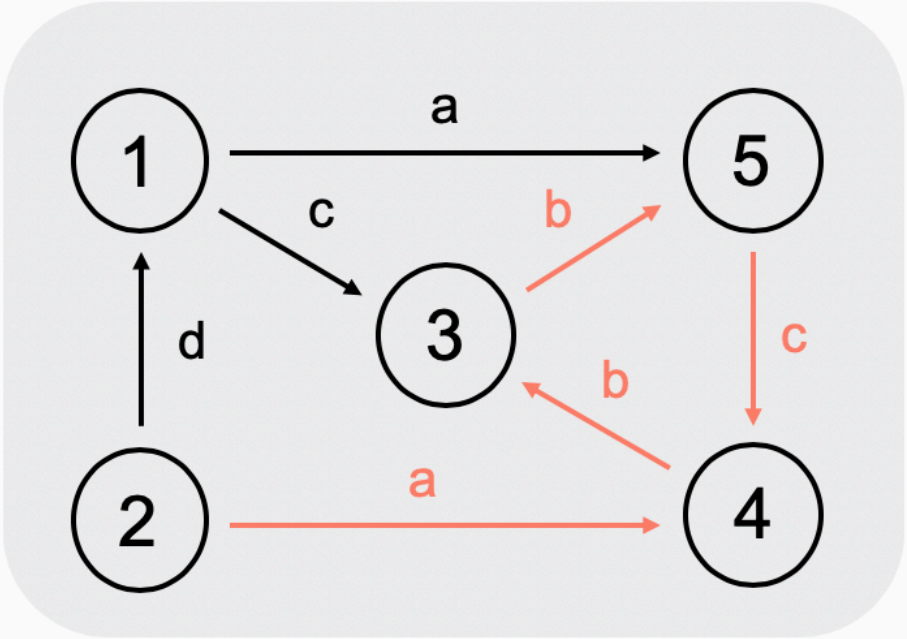
Limitation: Larger graphs require more storage space

Outline

- Background and Objective
- Related Work
- **The proposed method**
- Experiment and Discussion
- Conclusion

RPQ pipeline processing

Example. RPQ : $a \circ b \circ b \circ c$



Stage1

Edges with a ⋈ Edges with b

join

2→4
1→5

3→5
4→3

2→3

Stage2

Results of stage1 ⋈ Edges with b

join

2→3

3→5
4→3

2→5

Stage3

Results of stage2 ⋈ Edges with c

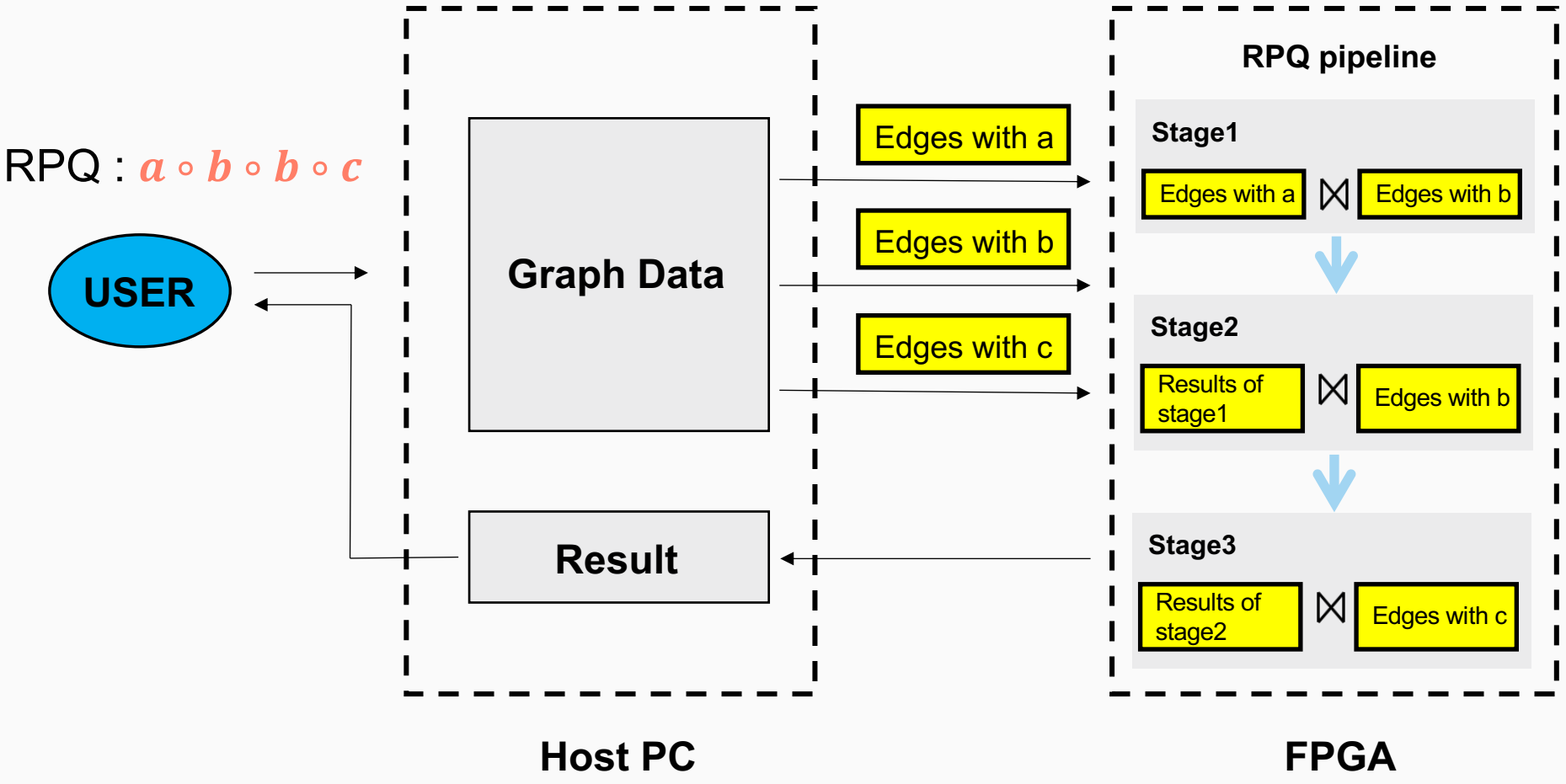
join

2→5

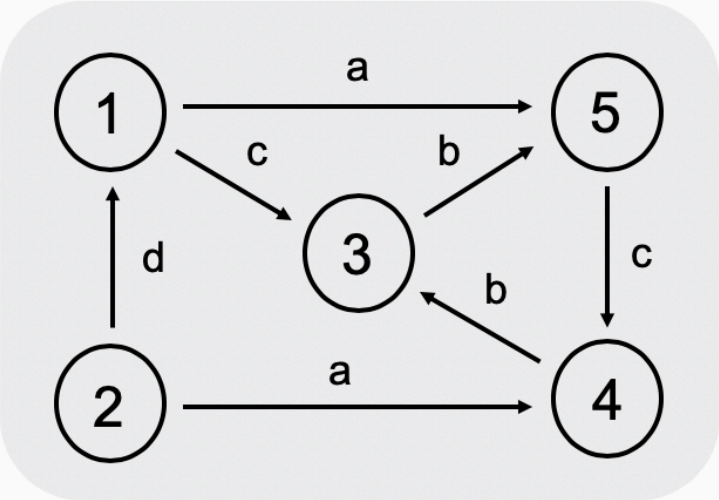
1→3
5→4

Result : 2→4

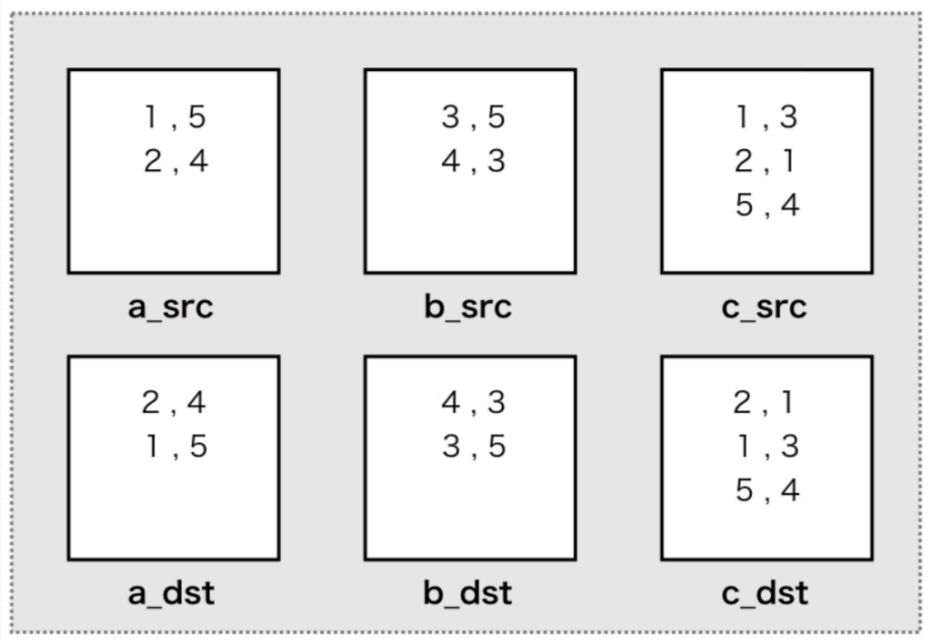
Process overview



Host : Storage format of graph data



Target Graph

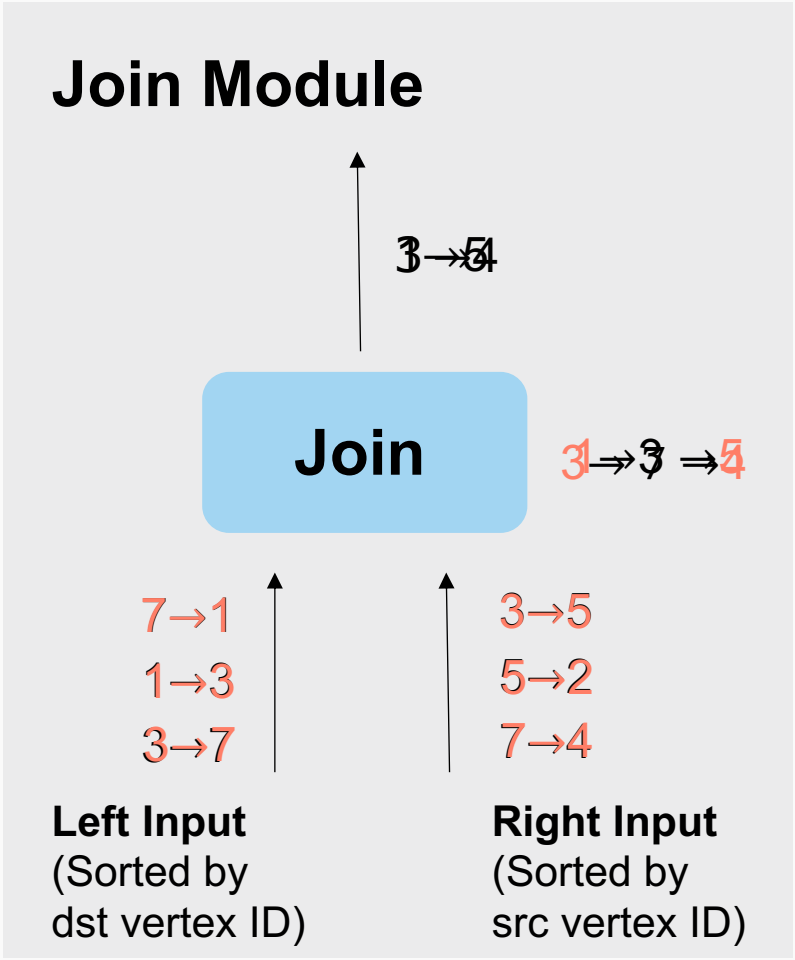


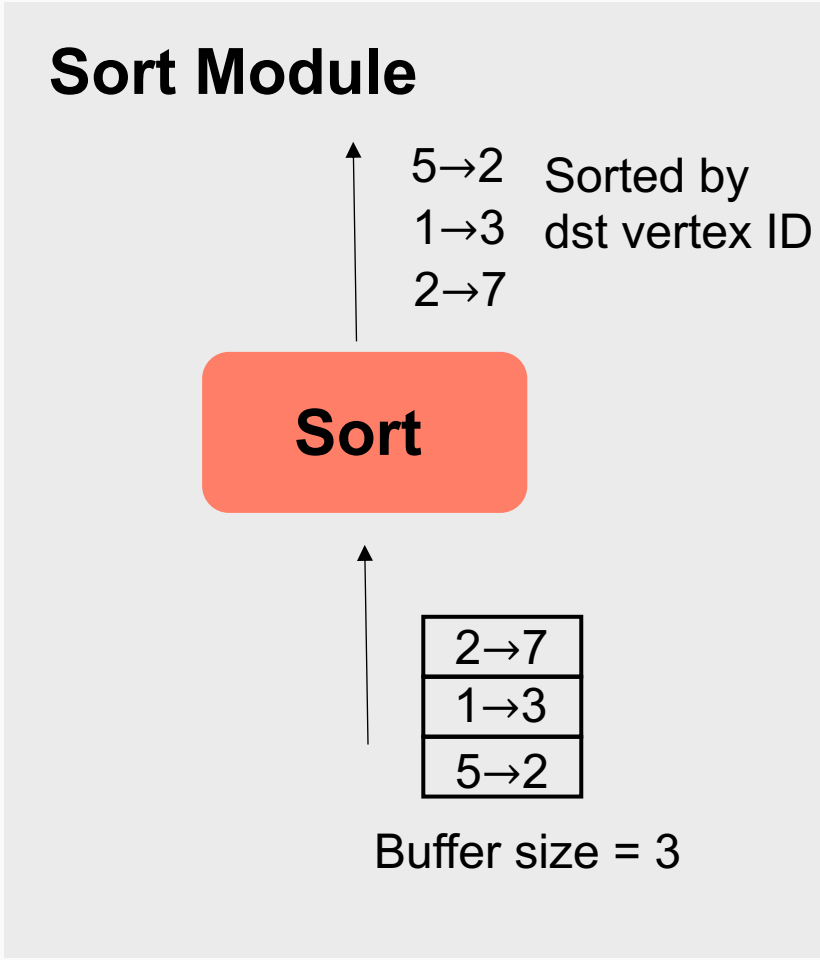
Storage area

a_src : File that holds the edge with label a. Sorted by source ID

a_dst : File that holds the edge with label a. Sorted by destination ID

FPGA : Module implementation





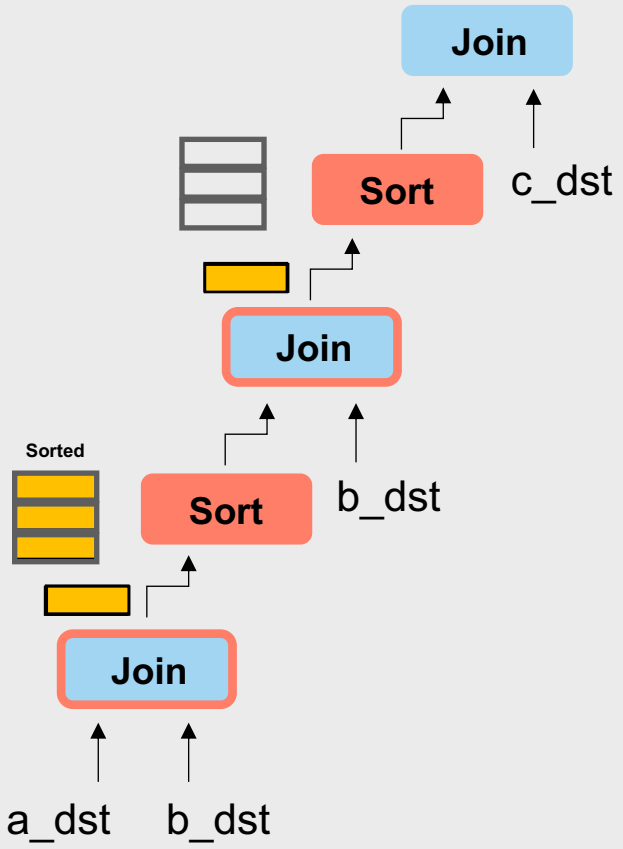
Serial configuration

Join : inactive

Join : active

Activated Join module processes in parallel

RPQ : $a \circ b \circ b \circ c$

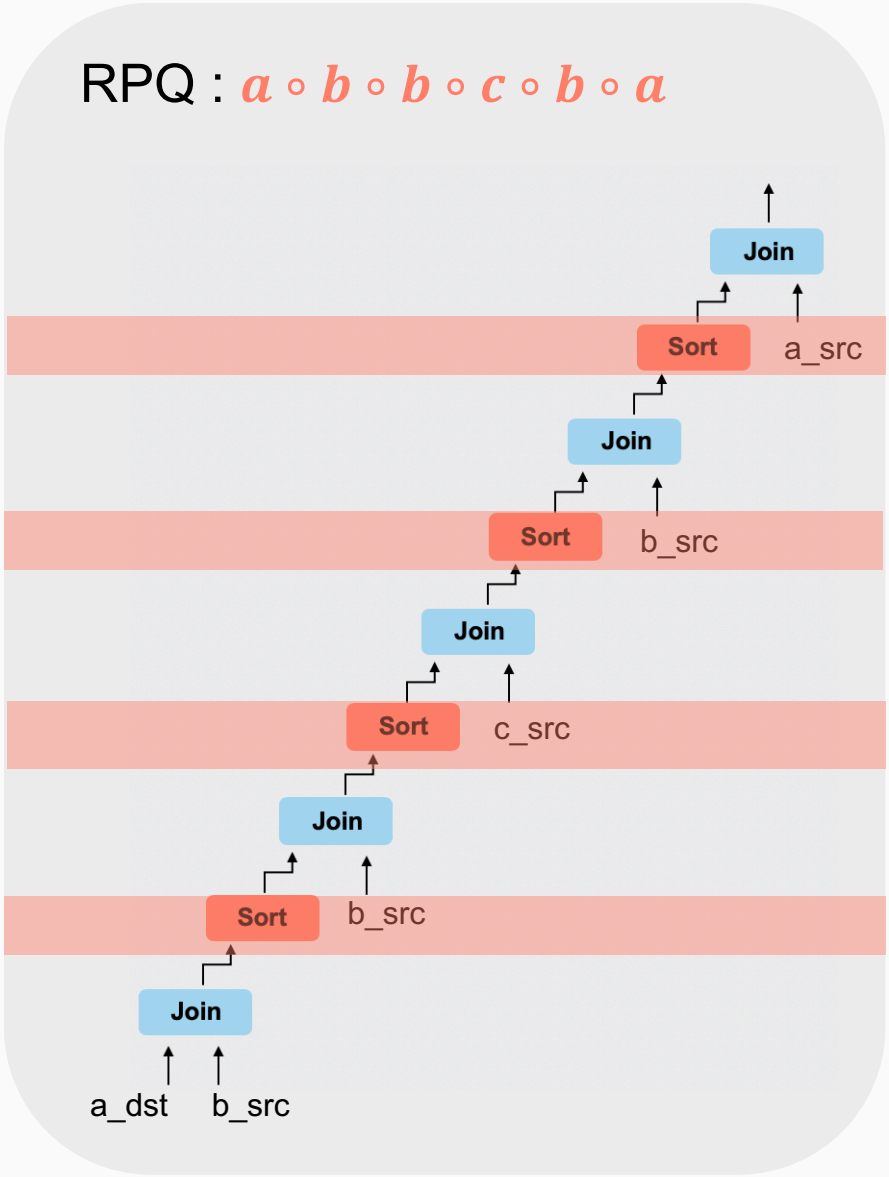


Possible limitation

Each Sort module buffers data

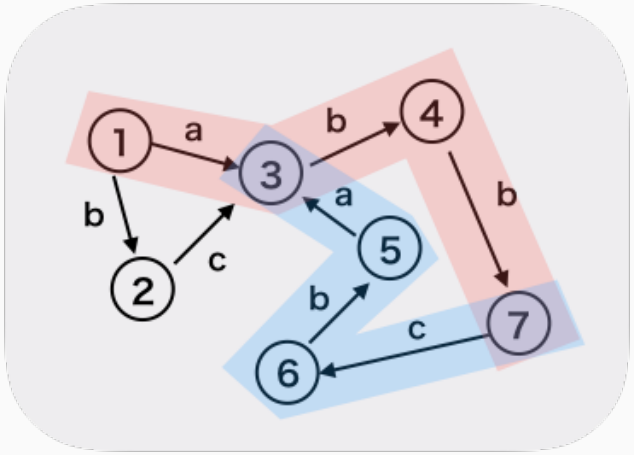
It takes time for the data to reach the module near the top

Buffering



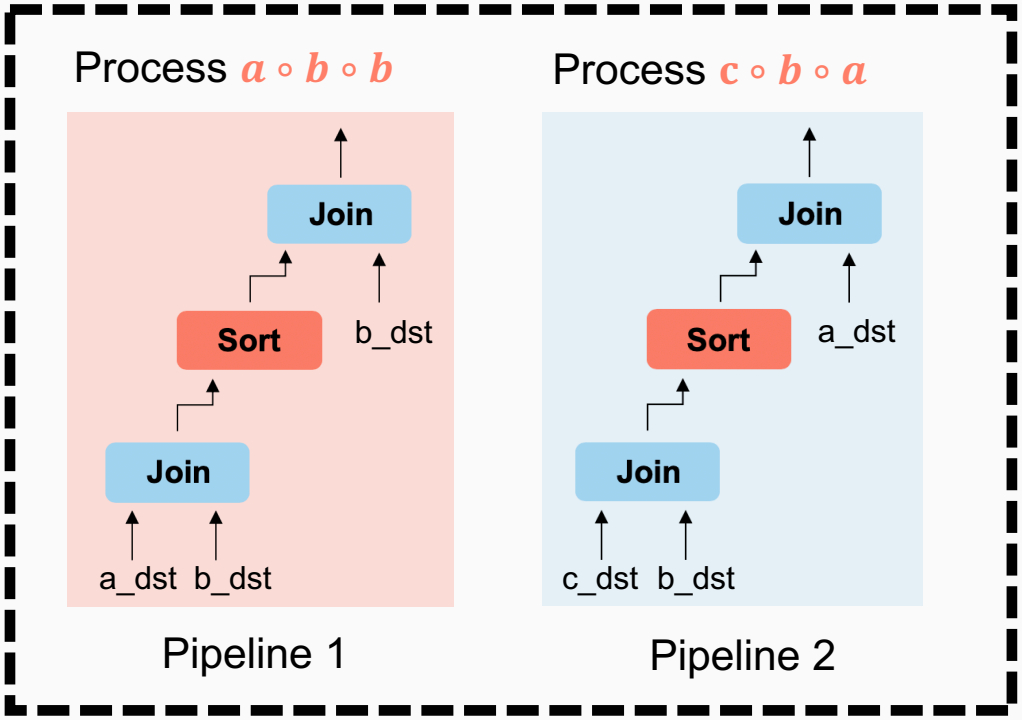
Parallel configuration

RPQ : $a \circ b \circ b \circ c \circ b \circ a$

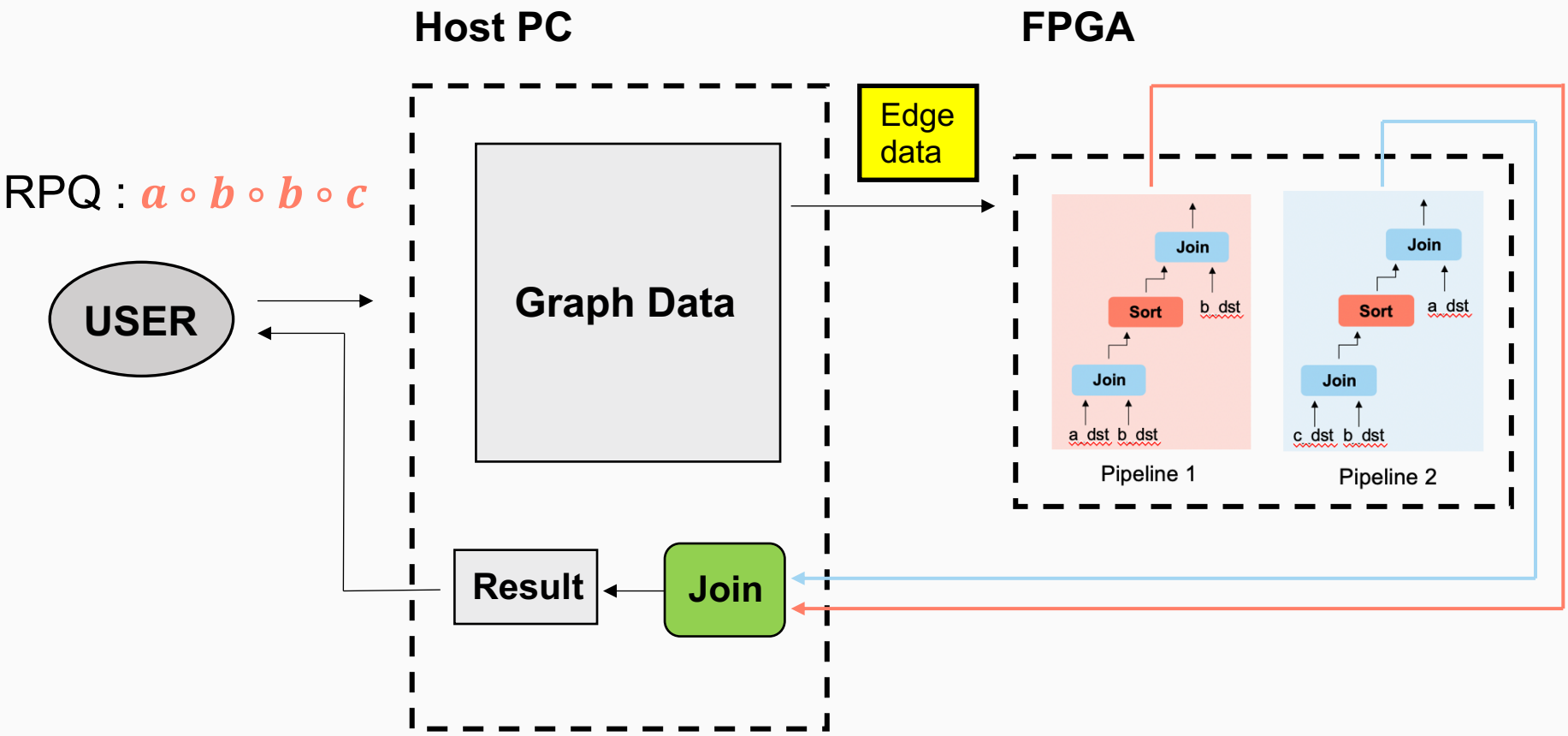


Target Graph

FPGA



Parallel configuration



The final join of the two pipelines is performed on the Host

Outline

- Background and Objective
- Related Work
- The proposed method
- **Experiment and Discussion**
- Conclusion

Experimental setup

- **Host PC**
  - CPU : Intel Core i7-7700K 3.60 GHz × 8
  - OS : Linux version 3.10.0
  - Memory : 31.1GiB
- **FPGA**
  - Xilinx Kintex UltraScale FPGA KCU1500
  - System Logic Cells : 1,451K
  - Block RAM : 75.9 Mb
  - Global Memory : 16 GB DDR 4

Performance for small graph

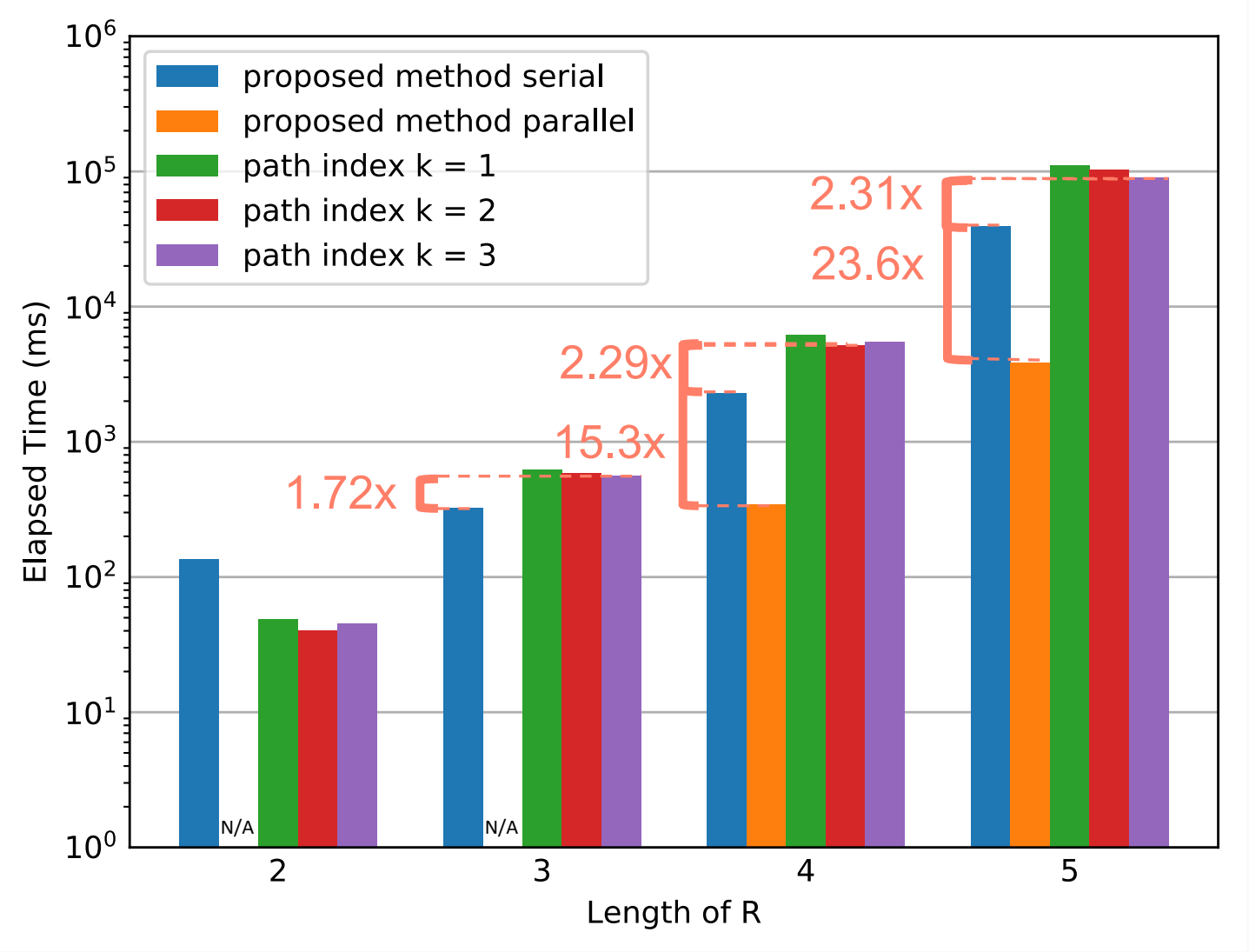
- Dataset : **advogato**

  - # vertexes : 6,541
  - # edges : 51,127
  - # label types : 3

- Investigate performance by changing RPQ path length $|R|$

- Comparison method : Path Index ($1 \leq k \leq 3$)

Performance for small graph



Dataset and queries

- Dataset : **DBLP-Citation-network V10 Dataset**

  - # vertexes : 4,850,632
  - # edges : 38,973,022
  - # label types : 6

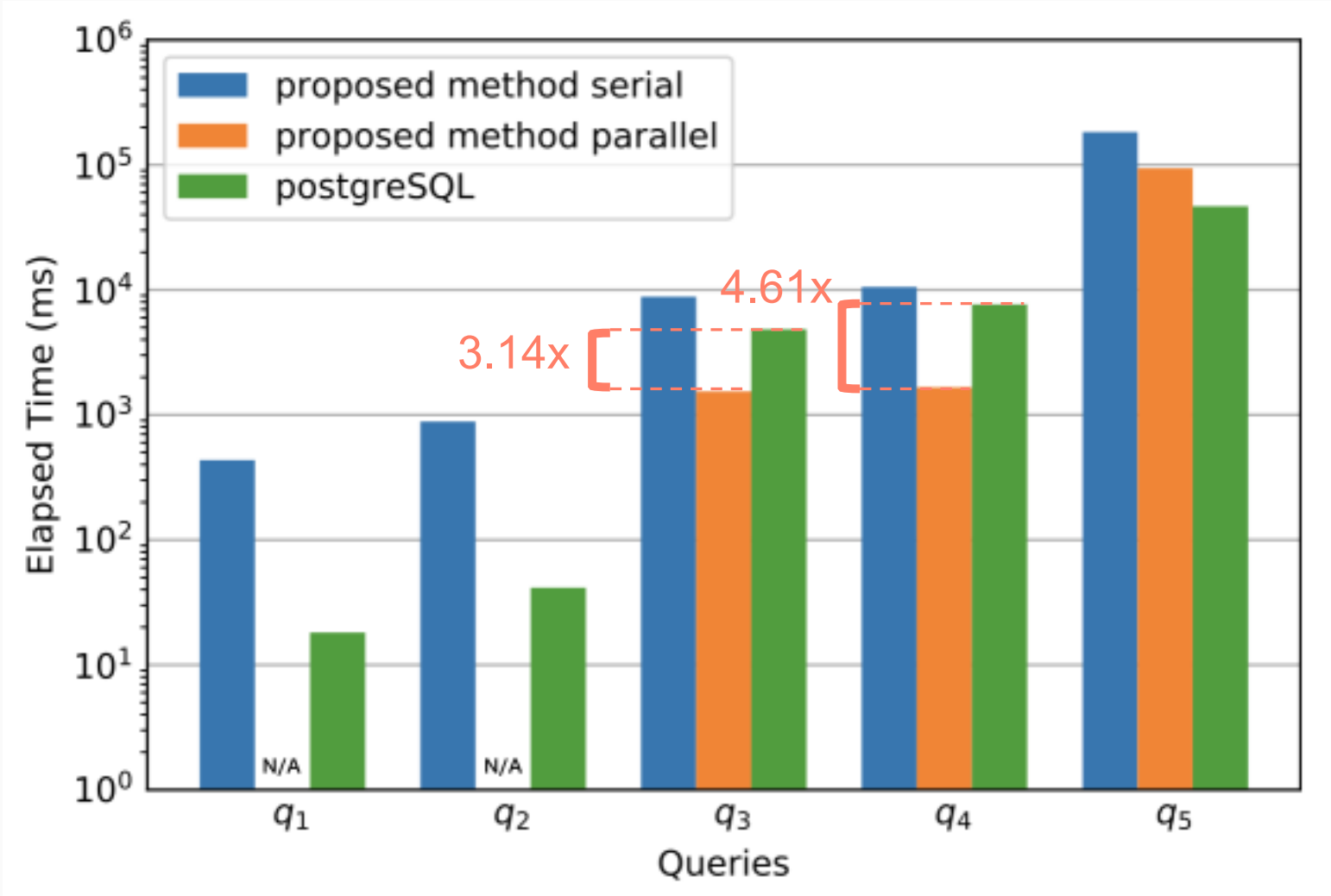
- Queries : Executed five queries as shown

Query	Path length $\|R\|$
q_1	2
q_2	3
q_3	4
q_4	4
q_5	5

Comparison method

- Comparison method : **PostgreSQL**
  - Path Index could not be created because the graph is large (Several tens of GB or more when k = 2)
  - We searched for vertex pairs that satisfy the path specified by RPQ using the PostgreSQL Select clause.

Performance for large graph

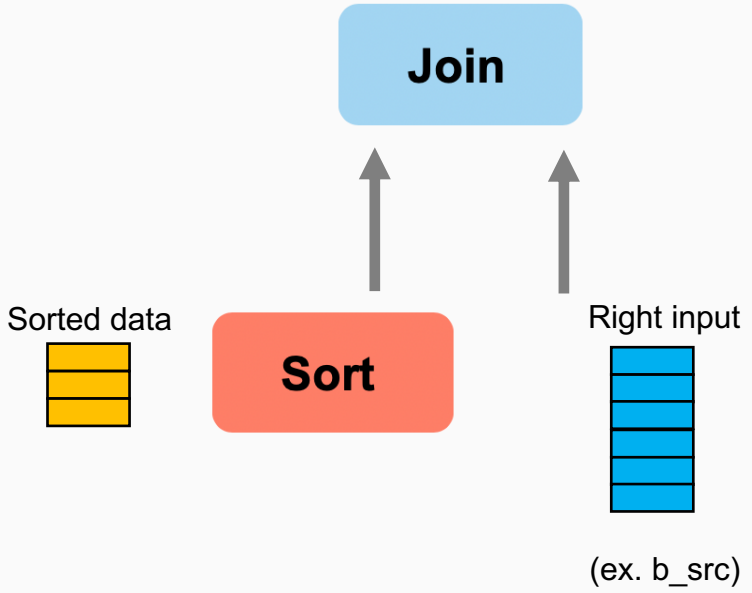


Discussion

- When processing large graphs
  - → significant drop in performance

Cause

Every time data is sent to the Join module, a lookup from the beginning is required



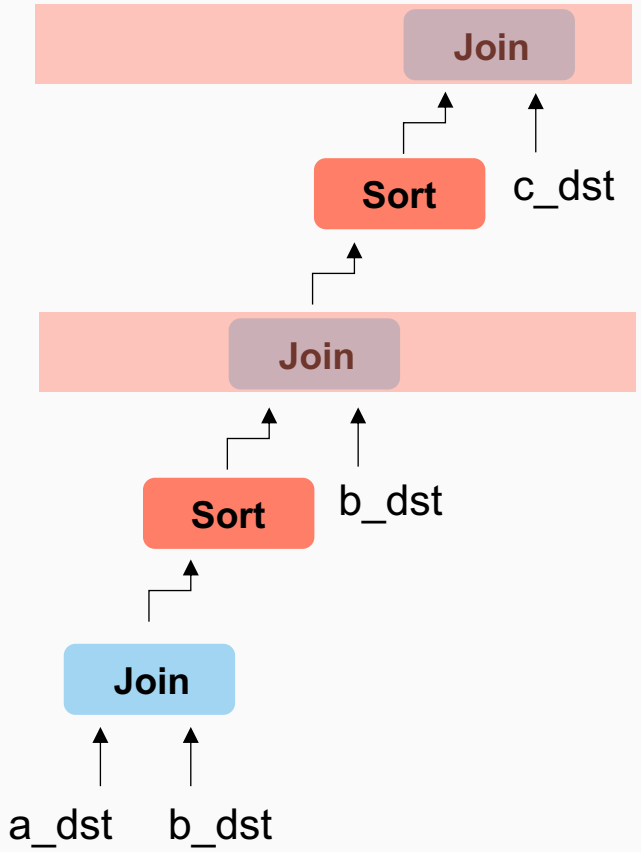
Discussion

- When processing large graphs
  - → significant drop in performance

Shorter pipeline length is preferred

↓

Further increase the number of pipelines in a serial configuration



Outline

- Background and Objective
- Related Work
- The proposed method
- Experiment and Discussion
- **Conclusion**

Conclusion

- Objective
  - Acceleration of RPQ processing using FPGA
- The proposed method
  - Proposed pipeline processing method for RPQ on FPGA using join and sort modules
  - In order to further increase the parallelism of processing, a parallel configuration was proposed.
- Performance
  - About 1.72 to 23.6 times faster for small graphs
  - Performance decreased for large graphs

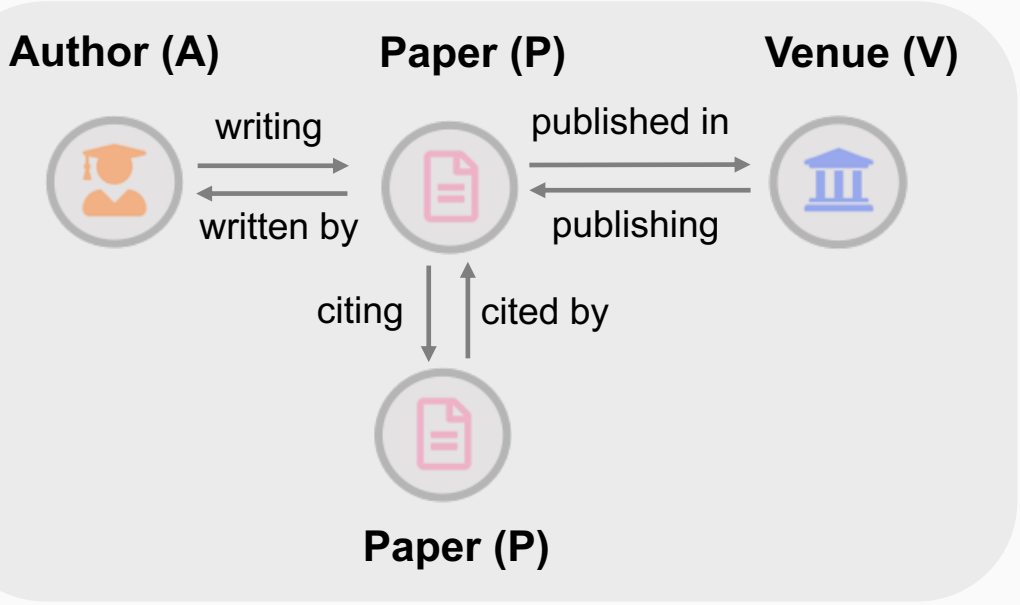
Conclusion

- Future work
  - Searching for ways to improve performance in large graphs
  - Supports RPQ processing other than composition operation

Thank you for your attention

Performance for large graph

- Dataset : **DBLP-Citation-network V10 Dataset**



Vertex types and relationships

- # vertexes : 4,850,632
  - # P : 3,079,007
  - # A : 1,766,547
  - # V : 5,078
- # edges : 38,973,022
- # label types : 6

Experiment2 : Queries

$q_1 : R = writing \circ written\ by$

$q_2 : R = writing \circ citing \circ written\ by$

$q_3 : R = writing \circ published\ in \circ publishing \circ written\ by$

$q_4 : R = publishing \circ written\ by \circ writing \circ written\ by$

$q_5 : R = writing \circ citing \circ published\ in\ \circ citing \circ written\ by$

- Due to the excessive size of query results, it was not feasible to enumerate all occurrences of query results

→ Specify a specific vertex as the starting vertex

q_1, q_2, q_3, q_4 : source vertex = “Hiroyuki Kitagawa”

q_5: source vertex = “very large data bases”

Setting for the PostgreSQL

edges

src	dst	label_id
163954	1766548	0
1766548	163954	1
⋮	⋮	⋮

labels

label_id	label
0	writing
1	written by
2	publishing
3	published in
4	citing
5	cited by

Query example (q_1 : $R = writing \circ written\ by$)

```
SELECT e1.src, e2.dst FROM edges AS e1, edges as e2
WHERE e1.src = 260069 AND e1.label_id = 0 AND
e2.label_id = 1 AND e1.dst = e2.src;
```