Multi-Node Multi-GPU Datalog

Ahmedur Rahman Shovon, Yihao Sun, Thomas Gilray, Kristopher Micinski, Sidharth Kumar

The HARP Lab

WASHINGTON STATE UNIVERSITY

UIC UNIVERSITY OF ILLINOIS CHICAGO

Syracuse University


Introduction and motivation
3
MNMGDatalog Implementation
23
Evaluation
41
Conclusion
47

Introduction and motivation

Declarative Programming Paradigm

Users expresses what to achieve with the data rather than how to accomplish it

User

UserID	UserName	UserEmail	Country
101	Alice	alice@example.com	USA
102	Bob	bob@example.com	USA
103	Eve	eve@example.com	Australia

WHAT

SELECT **UserID** FROM **User** WHERE **Country** = 'USA';

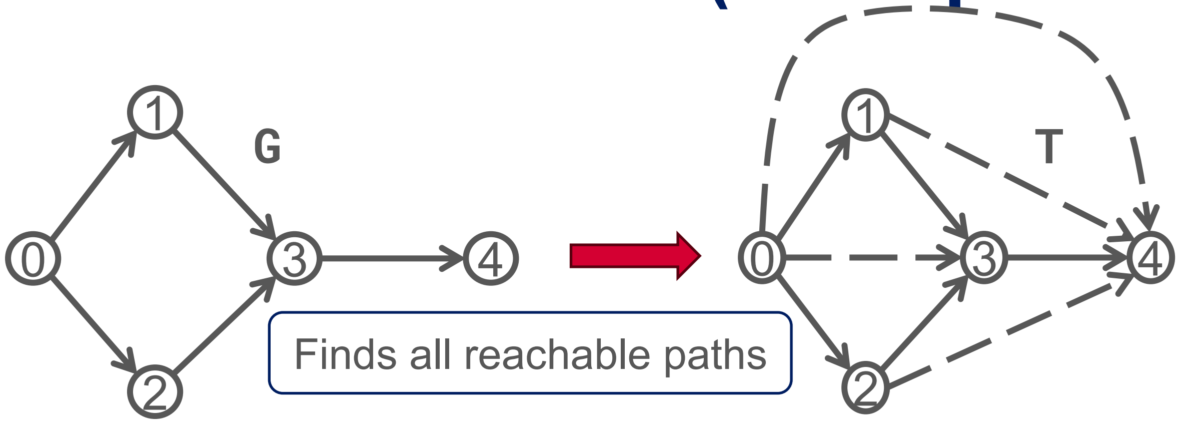
~~**HOW**~~

Another approach: Datalog (suitable for recursive queries)

- Makrynioti, N., & Vassalos, V. (2019). Declarative data analytics: A survey. IEEE Transactions on Knowledge and Data Engineering, 33(6), 2392-2411.
- Salesforce. (2024). Click, Not Code: The Benefits of Declarative Programming vs. Imperative Programming retrieved from https://www.salesforce.com/products/platform/best-practices/declarative-programming-vs-imperative-programming/ on 01/24/2026

UIC

Datalog on Recursive Queries (Example: Transitive Closure)



Finds all reachable paths

Paths can be discovered multiple times

Transitive Closure using Recursive SQL

Transitive Closure using Datalog

Datalog simplifies recursive queries

```sql
-- Recursive CTE for Tra
WITH RECURSIVE Trans
-- Base case: start with
        SELECT source, target
        FROM edges
        UNION
-- Recursive case: find new edges by joining with previous results
        SELECT tc.source, e.target
        FROM TransitiveClosure tc
        JOIN edges e ON tc.target = e.source
)
SELECT DISTINCT source, target FROM TransitiveClosure;
```

```
// Base case: Direct edges
tc(X, Y) :- edges(X, Y).
// Recursive case: Indirect connections
tc(X, Z) :- tc(X, Y), edges(Y, Z).
```


- Shovon, A. R. (2024, April 23). Recursive queries in SQL and Datalog. Ahmedur Rahman Shovon. Retrieved June 22, 2024, from https://arshovon.com/blog/recursive-queries/

Datalog Rules to Iterative Relational Algebra

Datalog Rules

tc(X, Y) :- edges(X, Y).
tc(X, Z) :- tc(X, Y), edges(Y, Z).

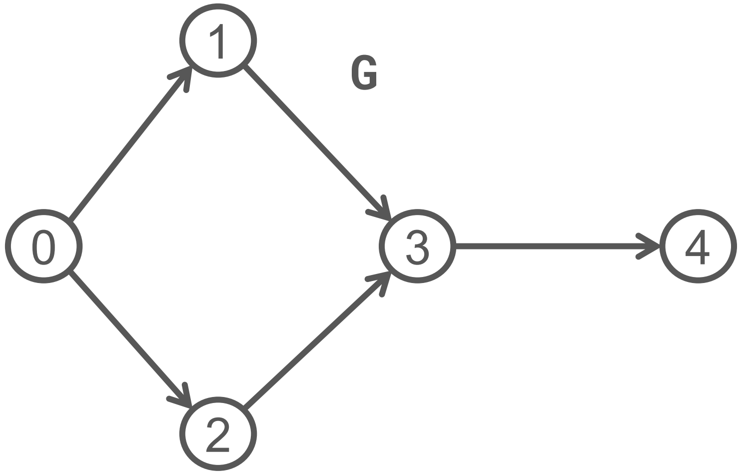
Iterative Relational Algebra

$$F_G(T) \triangleq G \cup \Pi_{1,2}(\rho_{0/1}(T) \bowtie_1 G)$$

Relational algebra: **Union** **Projection** **Join**

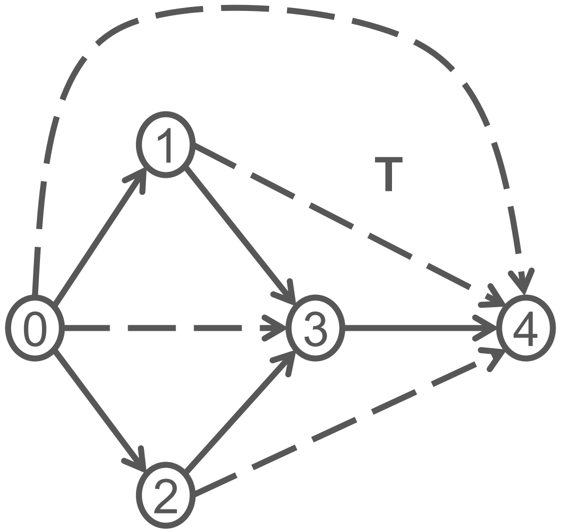
- Gilray, T., & Kumar, S. (2019, December). Distributed relational algebra at scale. In *2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC)* (pp. 12-22). IEEE.

Transitive Closure with Datalog



0	1
0	2
1	3
2	3
3	4

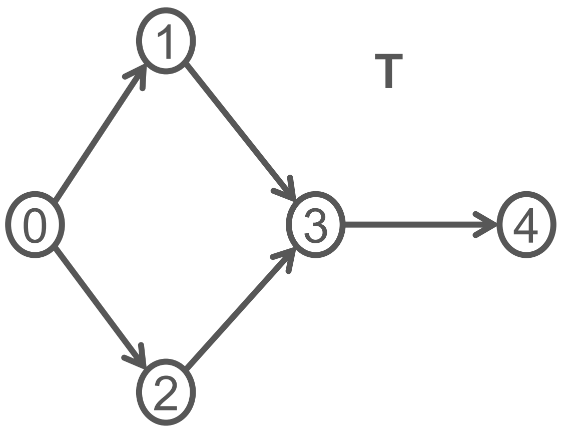
```
tc(X, Y) :- edges(X, Y).
tc(X, Z) :- tc(X, Y), edges(Y, Z).
```



0	1
0	2
1	3
2	3
3	4
0	3
1	4
2	4
0	4

$$F_G(T) \triangleq G \cup \Pi_{1,2}(\rho_{0/1}(T) \bowtie_1 G)$$

Transitive Closure: Iterations 1



T

0	1
0	2
1	3
2	3
3	4

$$F_G(T) \triangleq G \cup \Pi_{1,2}\boxed{(\rho_{0/1}(T) \bowtie_1 G)}$$

Transitive Closure: Iterations 1

$\rho_{0/1}(T)$

1	0
2	0
3	1
3	2
4	3

$\bowtie$

G

0	1
0	2
1	3
2	3
3	4

$$F_G(T) \triangleq G \cup \Pi_{1,2}\boxed{(\rho_{0/1}(T) \bowtie_1 G)}$$

Transitive Closure: Iterations 1

$\rho_{0/1}(T)$

1	0
2	0
3	1
3	2
4	3

$\bowtie$

G

0	1
0	2
1	3
2	3
3	4

$\rho_{0/1}(T) \bowtie G$

1	0	3

$$F_G(T) \triangleq G \cup \Pi_{1,2}\boxed{(\rho_{0/1}(T) \bowtie_1 G)}$$

Transitive Closure: Iterations 1

$\rho_{0/1}(T)$

1	0
2	0
3	1
3	2
4	3

$\bowtie$

G

0	1
0	2
1	3
2	3
3	4

$\rho_{0/1}(T) \bowtie G$

1	0	3
2	0	3

$$F_G(T) \triangleq G \cup \Pi_{1,2}\boxed{(\rho_{0/1}(T) \bowtie_1 G)}$$

Transitive Closure: Iterations 1

$\rho_{0/1}(T)$

1	0
2	0
3	1
3	2
4	3

$\bowtie$

G

0	1
0	2
1	3
2	3
3	4

$\rho_{0/1}(T) \bowtie G$

1	0	3
2	0	3
3	1	4

Transitive Closure: Iterations 1

$F_G(T) \triangleq G \cup \Pi_{1,2}\boxed{(\rho_{0/1}(T) \bowtie_1 G)}$

$\rho_{0/1}(T)$

1	0
2	0
3	1
3	2
4	3

$\bowtie$

G

0	1
0	2
1	3
2	3
3	4

$\rho_{0/1}(T) \bowtie G$

1	0	3
2	0	3
3	1	4
3	2	4

$$F_G(T) \triangleq G \cup \boxed{\Pi_{1,2}(\rho_{0/1}(T) \bowtie_1 G)}$$

Transitive Closure: Iterations 1

$\rho_{0/1}(T)$

1	0
2	0
3	1
3	2
4	3

$\bowtie$

G

0	1
0	2
1	3
2	3
3	4

$\rho_{0/1}(T) \bowtie G$

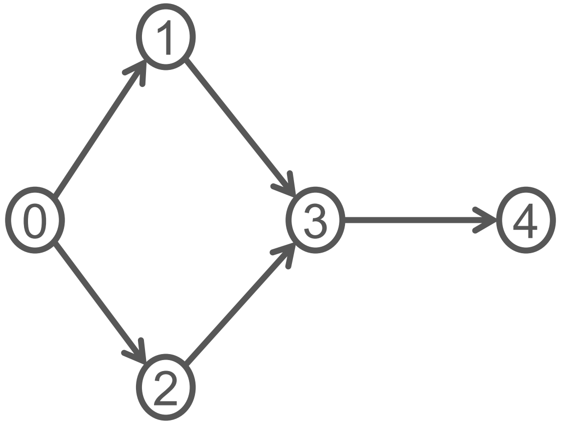
1	0	3
2	0	3
3	1	4
3	2	4

$\Pi_{1,\ 2}(\rho_{0/1}(T) \bowtie G)$

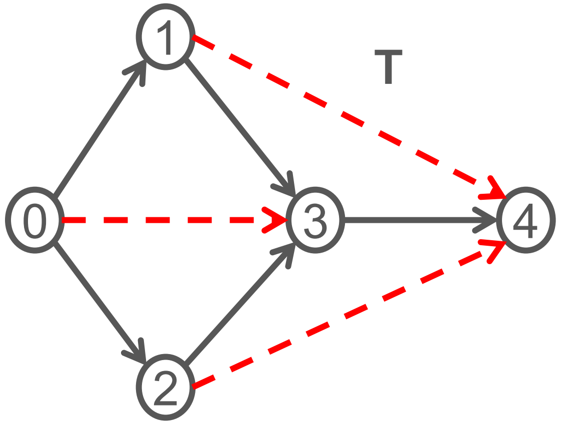
0	3
1	4
2	4

Transitive Closure: Iterations 1

$$F_G(T) \triangleq \boxed{G \cup \Pi_{1,2}(\rho_{0/1}(T) \bowtie_1 G)}$$



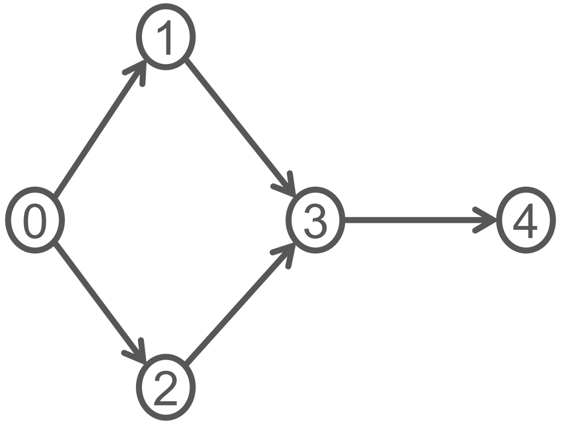
0	1
0	2
1	3
2	3
3	4

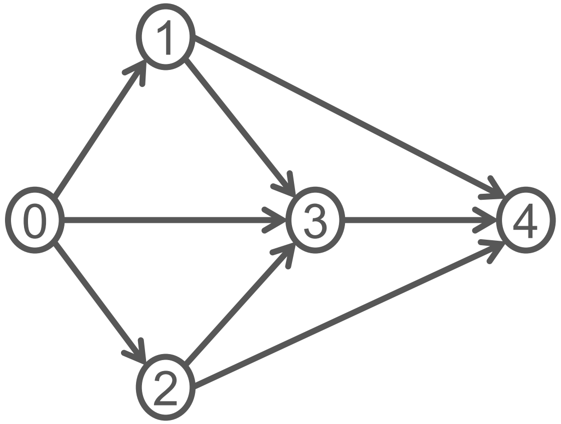


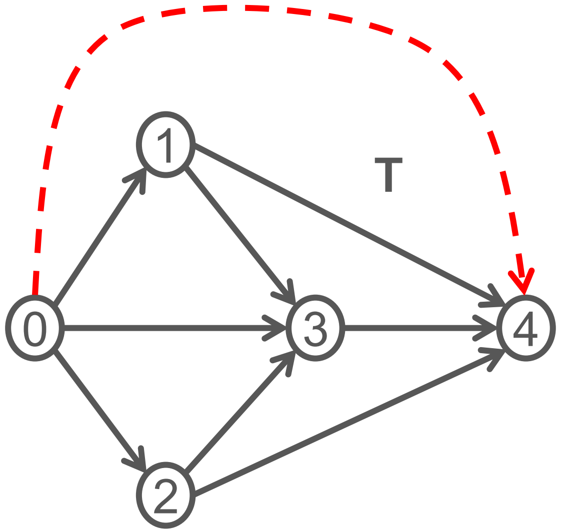
0	1
0	2
1	3
2	3
3	4
0	3
1	4
2	4

$$F_G(T) \triangleq G \cup \Pi_{1,2}(\rho_{0/1}(T) \bowtie_1 G)$$

Transitive Closure: Iterations 2







0	1
0	2
1	3
2	3
3	4

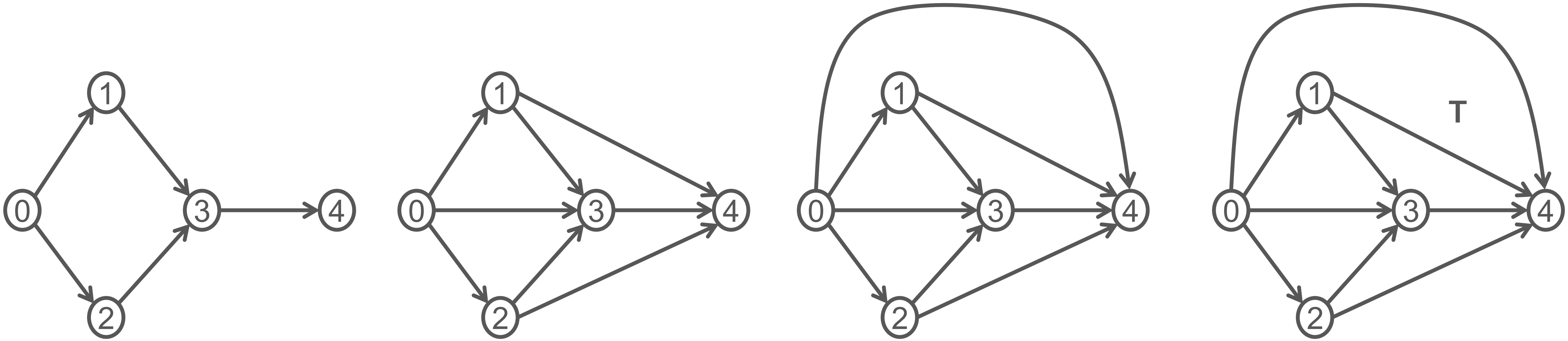
0	1
0	2
1	3
2	3
3	4
0	3
1	4
2	4

0	1
0	2
1	3
2	3
3	4
0	3
1	4
2	4
0	4

$$F_G(T) \triangleq G \cup \Pi_{1,2}(\rho_{0/1}(T) \bowtie_1 G)$$

Transitive Closure: Iterations 3

Fixed-point reached



0	1
0	2
1	3
2	3
3	4

0	1
0	2
1	3
2	3
3	4
0	3
1	4
2	4

0	1
0	2
1	3
2	3
3	4
0	3
1	4
2	4
0	4

0	1
0	2
1	3
2	3
3	4
0	3
1	4
2	4
0	4

Traditional Graph Mining Applications

Applications

Transitive closure

```
tc(X, Y) :- edges(X, Y).
tc(X, Z) :- tc(X, Y), edges(Y, Z).
```

Triangle counting

```
2cl(X, Y) :- edges(X, Y), X < Y.
2cl(X, Y) :- edges(Y, X), X < Y.
triangles(X, Y, Z) :- 2cl(X, Y), 2cl(Y, Z), 2cl(X, Z).
```

Connected components

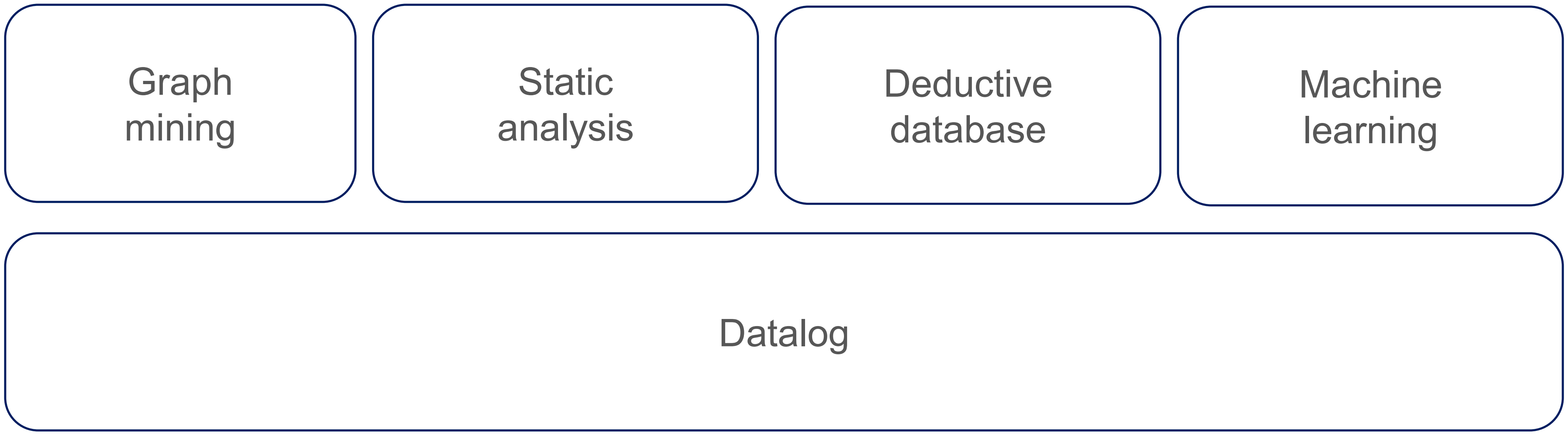
```
cc(X, X) :- edges(X, _).
cc(Y, $MIN(Z)) :- cc(Y, Z), edges(X, Y).
```

Same generation

```
sg(x, y) :- edges(p, x), edges(p, y), x ≠ y.
sg(x, y) :- edges(a, x), sg(a, b), edges(b, y), x ≠ y.
```


- De Moor, O., Gottlob, G., Furche, T., & Sellers, A. (Eds.). (2012). Datalog Reloaded: First International Workshop, Datalog 2010, Oxford, UK, March 16-19, 2010. Revised Selected Papers (Vol. 6702). Springer.
- Huang, S. S., Green, T. J., & Loo, B. T. (2011, June). Datalog and emerging applications: an interactive tutorial. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (pp. 1213-1216).
- Gilray, T., & Kumar, S. (2017). Toward parallel cfa with datalog, mpi, and cuda. In Scheme and Functional Programming Workshop.
- Zomorodian, A. (2012). Topological data analysis. Advances in applied and computational topology, 70, 1-39.

Datalog Applications



Datalog Implementations

Multi-threaded	Distributed (Apache Spark)	Multi-node Multi-threaded	Single-GPU	Multi-node Multi-GPU
Soufflé	RDFox	SLOG	GPUJoin	
LogicBlox	Radlog	PRAM	GPULog	
Nemo	BigDatalog		GPUDatalog	

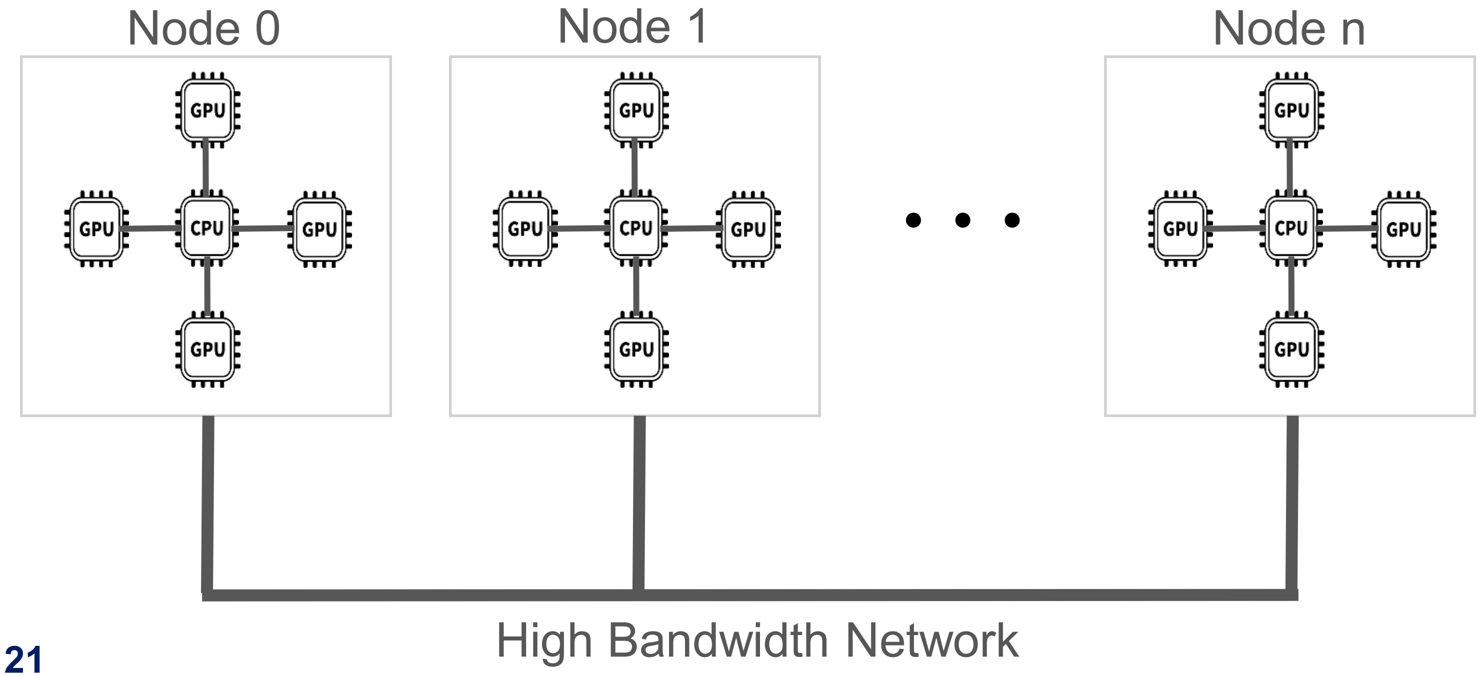
- Herbert Jordan, Bernhard Scholz, and Pavle Subotić. 2016. Soufflé: On synthesis of program analyzers. In Computer Aided Verification: 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II 28, Swarat Chaudhuri and Azadeh Farzan (Eds.). Springer, Springer International Publishing, Cham, 422–430.
- Boris Motik, Yavor Nenov, Robert Piro, Ian Horrocks, and Dan Olteanu. 2014. Parallel materialisation of datalog programs in centralised, main-memory RDF systems. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 28.
- Shkapsky, A., Yang, M., Interlandi, M., Chiu, H., Condie, T., & Zaniolo, C. (2016, June). Big data analytics with datalog queries on spark. In Proceedings of the 2016 International Conference on Management of Data (pp. 1135-1149).
- Yavor Nenov, Robert Piro, Boris Motik, Ian Horrocks, Zhe Wu, and Jay Banerjee. 2015. RDFox: A Highly-Scalable RDF Store. In The Semantic Web - ISWC 2015, Marcelo Arenas, Oscar Corcho, Elen Simperl, Markus Strohmaier, Mathieu d'Aquin, Kavitha Srinivas, Paul Groth, Michel Dumontier, Jeff Heflin, Krishnaprasad Thirunarayan, and Steffen Staab (Eds.). Springer International Publishing, Cham, 3–20.
- Gilray, T., Sahebolamri, A., Sun, Y., Kunapaneni, S., Kumar, S., & Micinski, K. (2024). Datalog with First-Class Facts. arXiv preprint arXiv:2411.14330.

Motivation: Datalog on Multi-Node Multi-GPU Systems

Datalog

Relational algebra $F_G(T) \triangleq G \cup \Pi_{1,2}(\rho_{0/1}(T) \bowtie_1 G)$

Multi-node multi-GPU systems



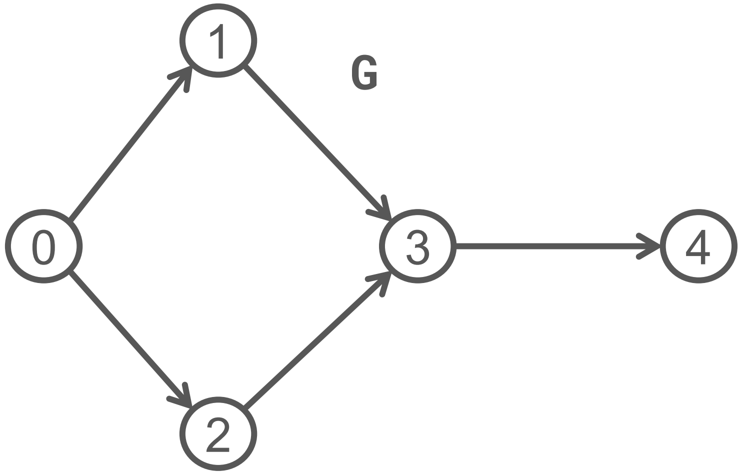
High performance implementation for highly expressive language

Contributions

- Radix-hash-based data partitioning strategy for iterative computation
- CUDA-Aware non-uniform all-to-all communication targeting iterative relational algebra
- Scalable recursive aggregation on GPUs
- Introduced **MNMGDatalog** the first multi-node multi-GPU Datalog engine
    - Single-GPU: Up to 7× speedup over GPULog
    - Multi-threaded: Up to 33× over Soufflé
    - Multi-node multi-threaded: Up to 32× speedup over SLOG

MNMGDatalog Implementation

Requirements for Multi-node Multi-GPU Datalog

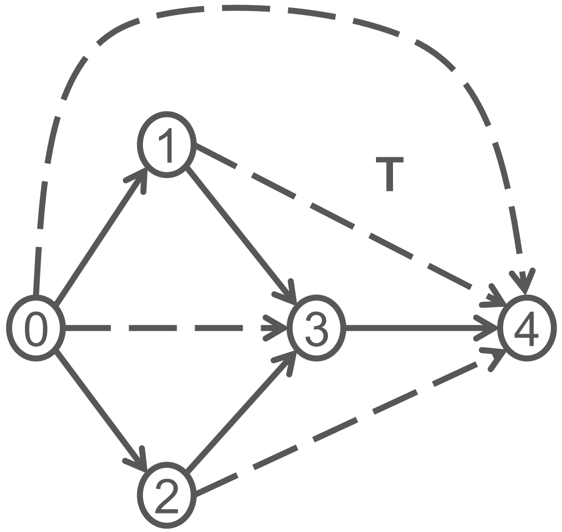


0	1
0	2
1	3
2	3
3	4

1 Workload partitioning

tc(X, Y) :- edges(X, Y).
tc(X, Z) :- tc(X, Y), edges(Y, Z).

2 Data representation



0	1
0	2
1	3
2	3
3	4
0	3
1	4
2	4
0	4

3 Communication

MNMGDatalog Implementation

Hash-based data distribution	GPU-optimized data representation	GPU-Aware all-to-all communication
1 Workload partitioning	2 Data representation	3 Communication

MNMGDatalog: Radix-hash-based partitioning

Hash-based data distribution

GPU-optimized data representation

GPU-Aware all-to-all communication

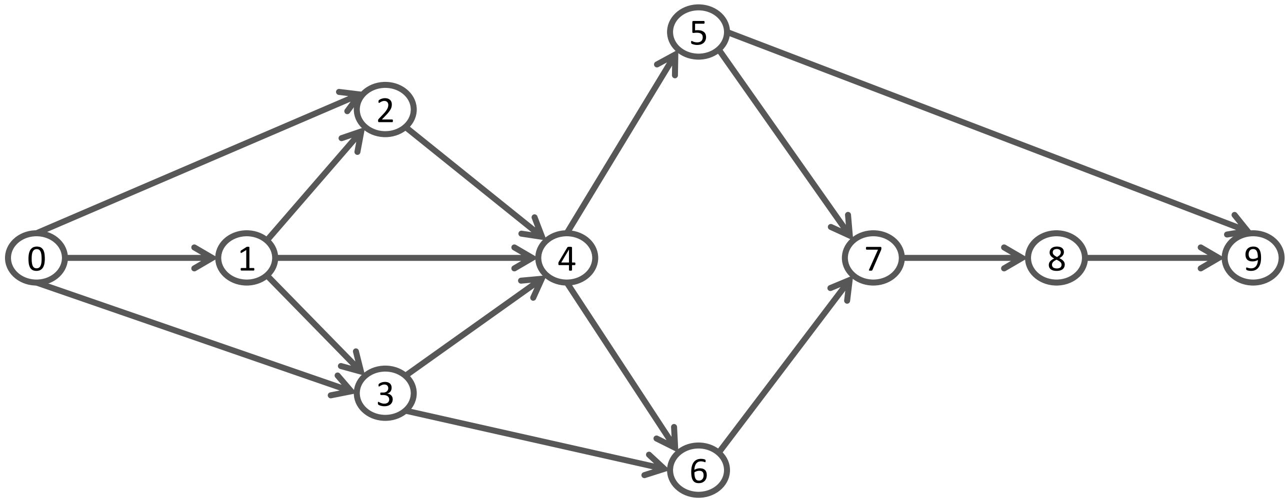
1 Workload partitioning

2 Data representation

3 Communication

The join column decides the GPU (bucket-id)

(k)	
0	1
0	2
0	3
1	2
1	3
1	4
2	4
3	4
3	6
4	5
4	6
5	7
5	9
7	6
7	8
8	9



Hash-partitioning based on the Join Column

Kumar, S., & Gilray, T. (2020). Load-balancing parallel relational algebra. In *High Performance Computing: 35th International Conference, ISC High Performance 2020, Frankfurt/Main, Germany, June 22–25, 2020, Proceedings 35* (pp. 288-308). Springer International Publishing.

The join column decides the GPU (bucket-id)

(k)

0	1
0	2
0	3
1	2
1	3
1	4
2	4
3	4
3	6
4	5
4	6
5	7
5	9
7	6
7	8
8	9

Hash(k) % 3

0	1
0	2
0	3
1	2
1	3
1	4
2	4
3	4
3	6
4	5
4	6
5	7
5	9
7	6
7	8
8	9

0

0	1
0	2
0	3
3	4
3	6

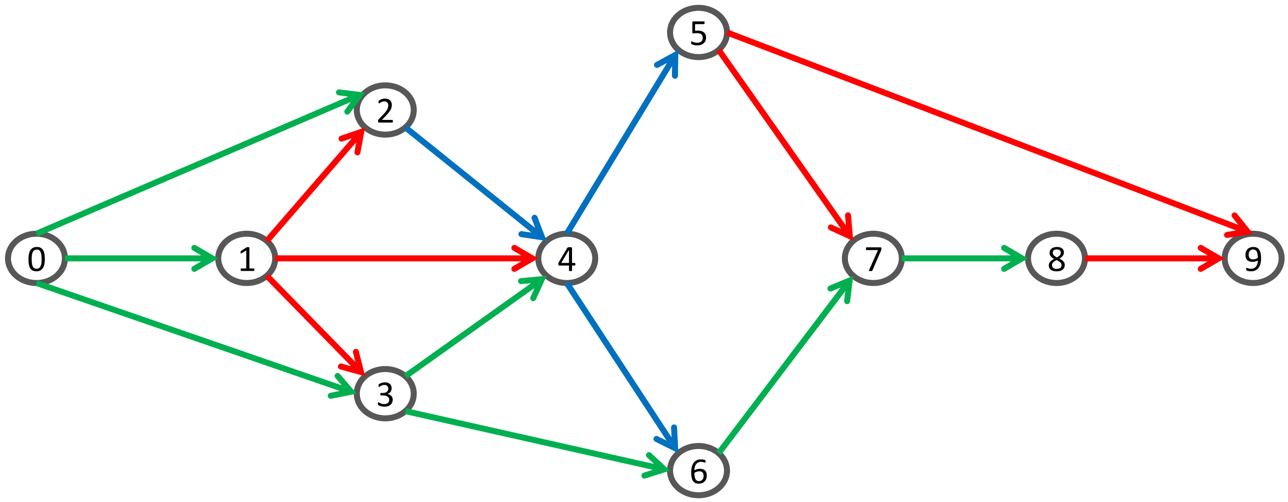
1

7	6
7	8
5	7
5	9
8	9

2

1	2
1	3
1	4
2	4
4	5
4	6

GPU (ID)



Kumar, S., & Gilray, T. (2020). Load-balancing parallel relational algebra. In *High Performance Computing: 35th International Conference, ISC High Performance 2020, Frankfurt/Main, Germany, June 22–25, 2020, Proceedings 35* (pp. 288-308). Springer International Publishing.

MNMGDatalog: Data representation

Hash-based data distribution

GPU-optimized data representation

GPU-Aware all-to-all communication

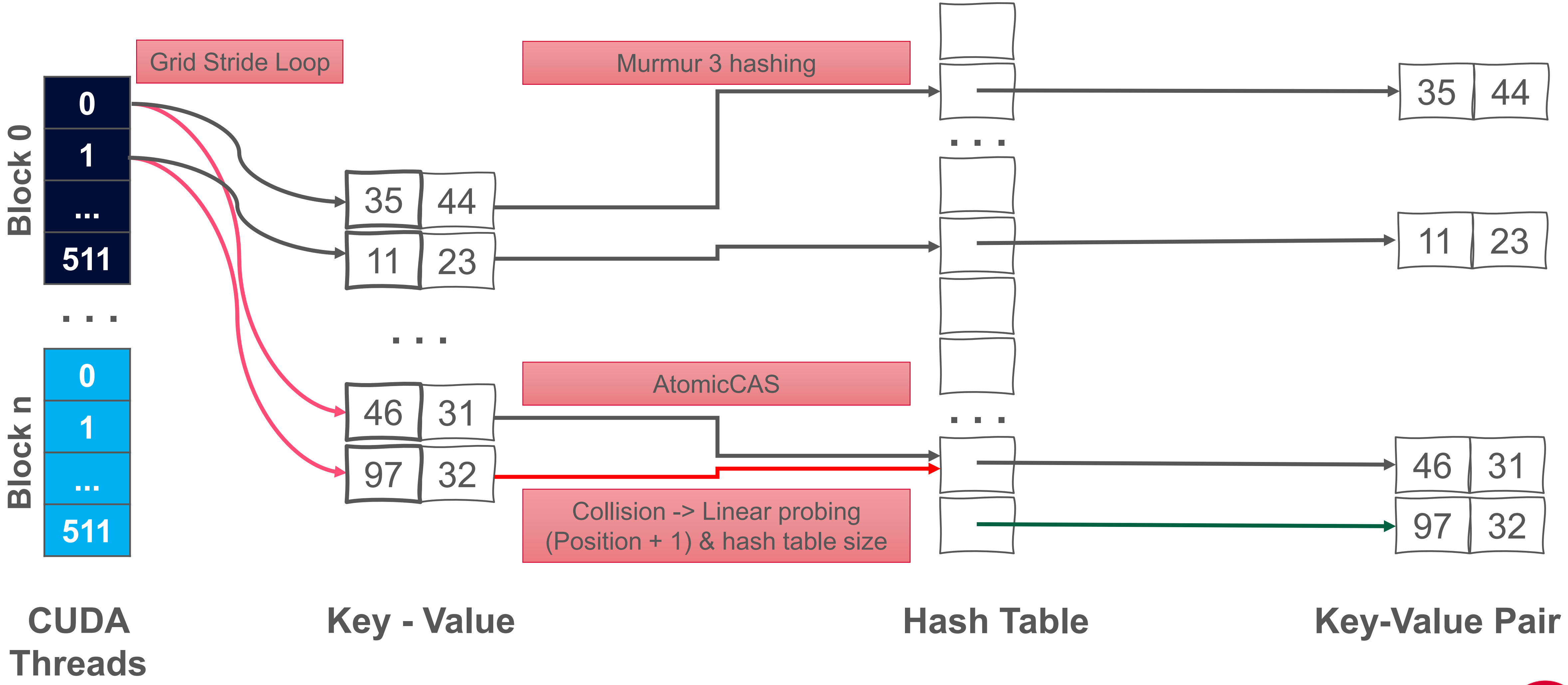
1 Workload partitioning

2 Data representation

3 Communication

$$F_G(T) \triangleq G \cup \Pi_{1,2}(\rho_{0/1}(T) \bowtie_1 G)$$

GPU Hash Table (Open Addressing, Linear Probing)



Shovon, A. R., Gilray, T., Micinski, K., & Kumar, S. (2023). Towards iterative relational algebra on the {GPU}. In 2023 USENIX Annual Technical Conference (USENIX ATC 23) (pp. 1009-1016).

$$F_G(T) \triangleq G \cup \Pi_{1,2}(\rho_{0/1}(T) \bowtie_1 G)$$

Performing Hash Join on GPU

Static Hash Table

Key	Value

$\bowtie$

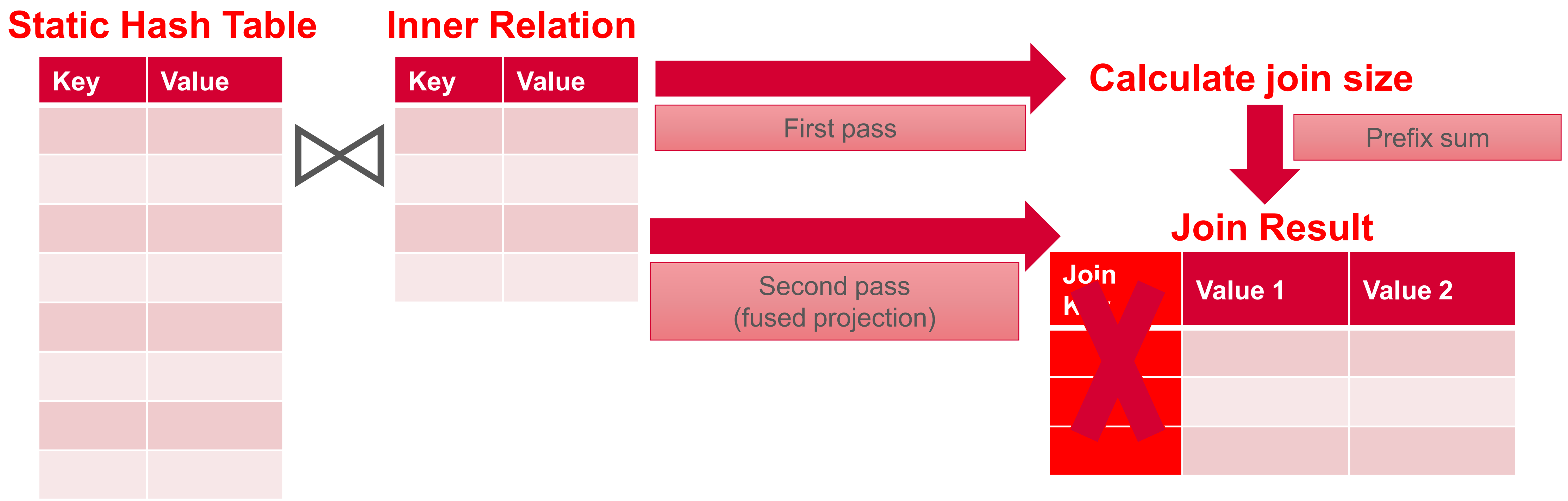
Inner Relation

Key	Value

First pass → **Calculate join size**

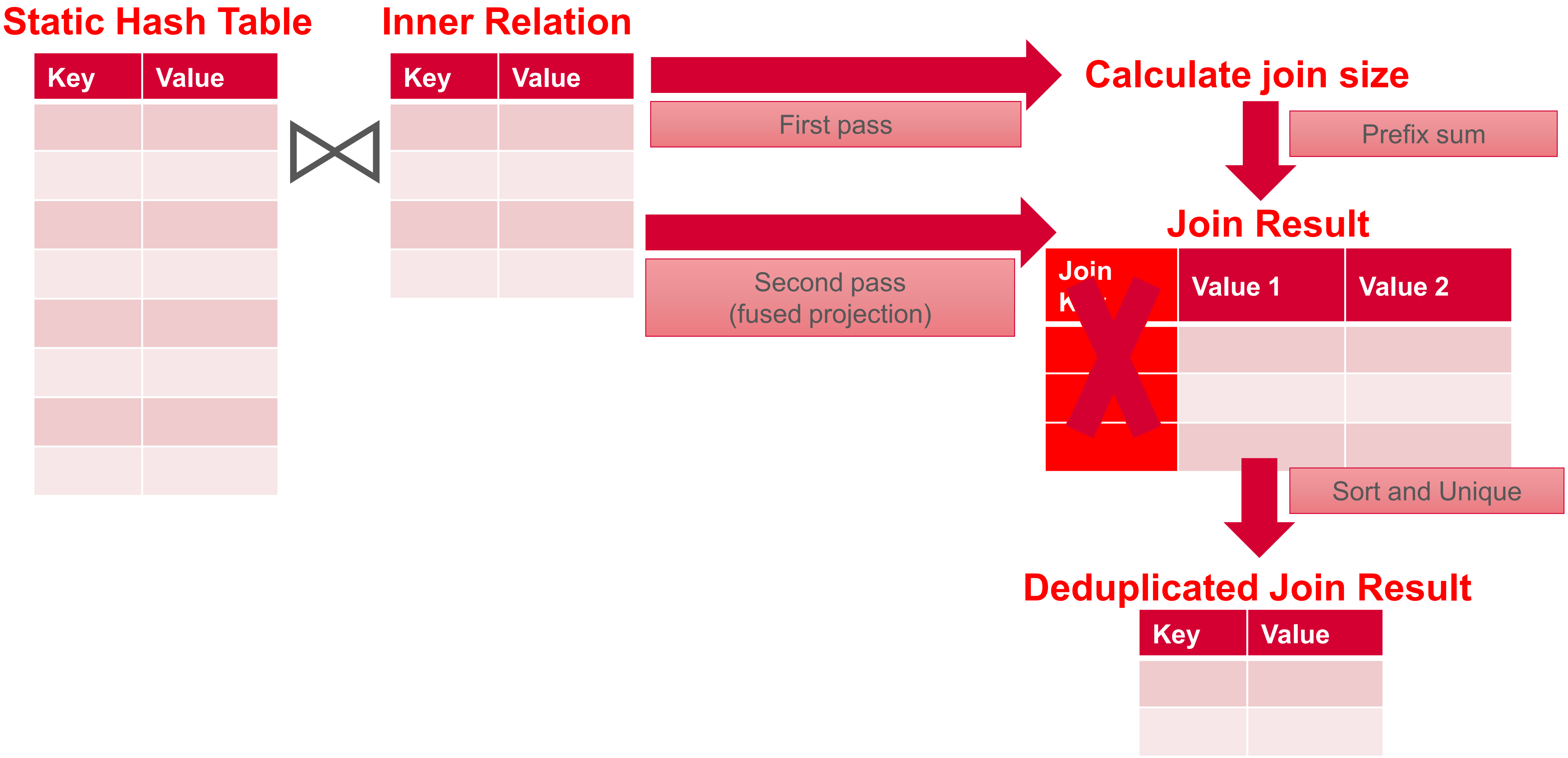
$$F_G(T) \triangleq G \cup \boxed{\Pi_{1,2}(\rho_{0/1}(T) \bowtie_1 G)}$$

Performing Hash Join on GPU



$$F_G(T) \triangleq G \cup \Pi_{1,2}(\rho_{0/1}(T) \bowtie_1 G)$$

Performing Hash Join on GPU



MNMGDatalog: Communication for Iterative RA

Hash-based data distribution

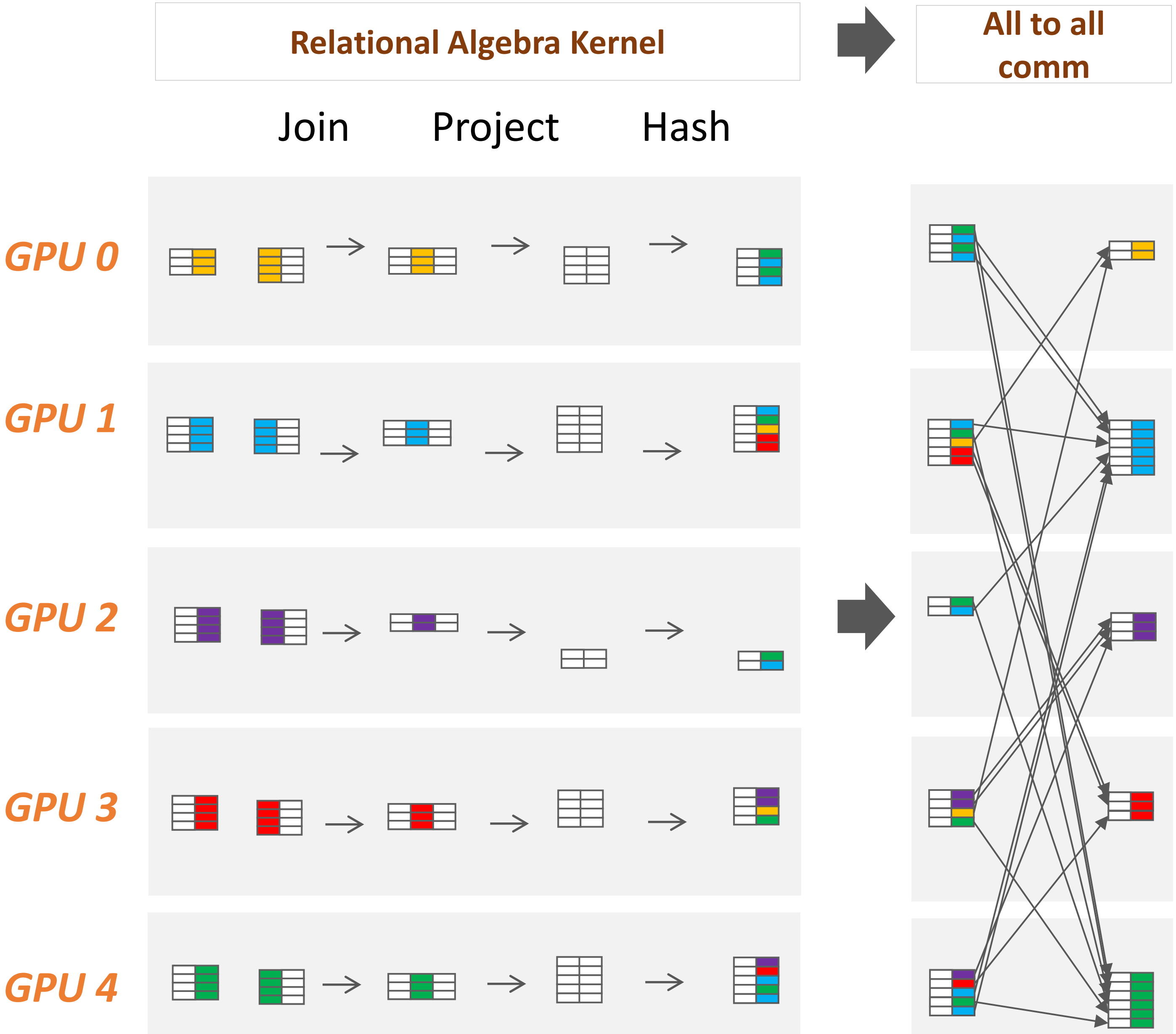
GPU-optimized data representation

GPU-Aware all-to-all communication

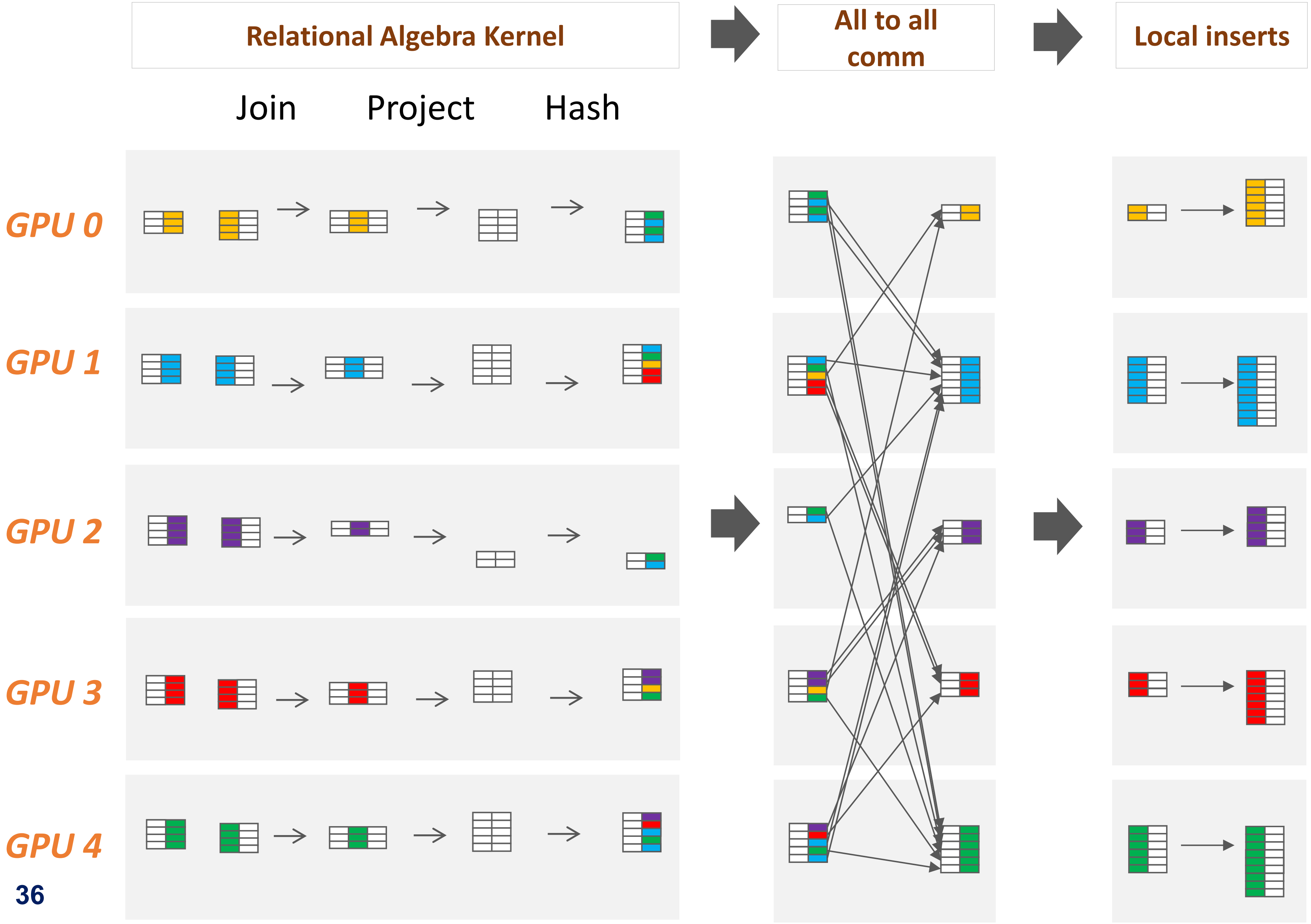
1 Workload partitioning

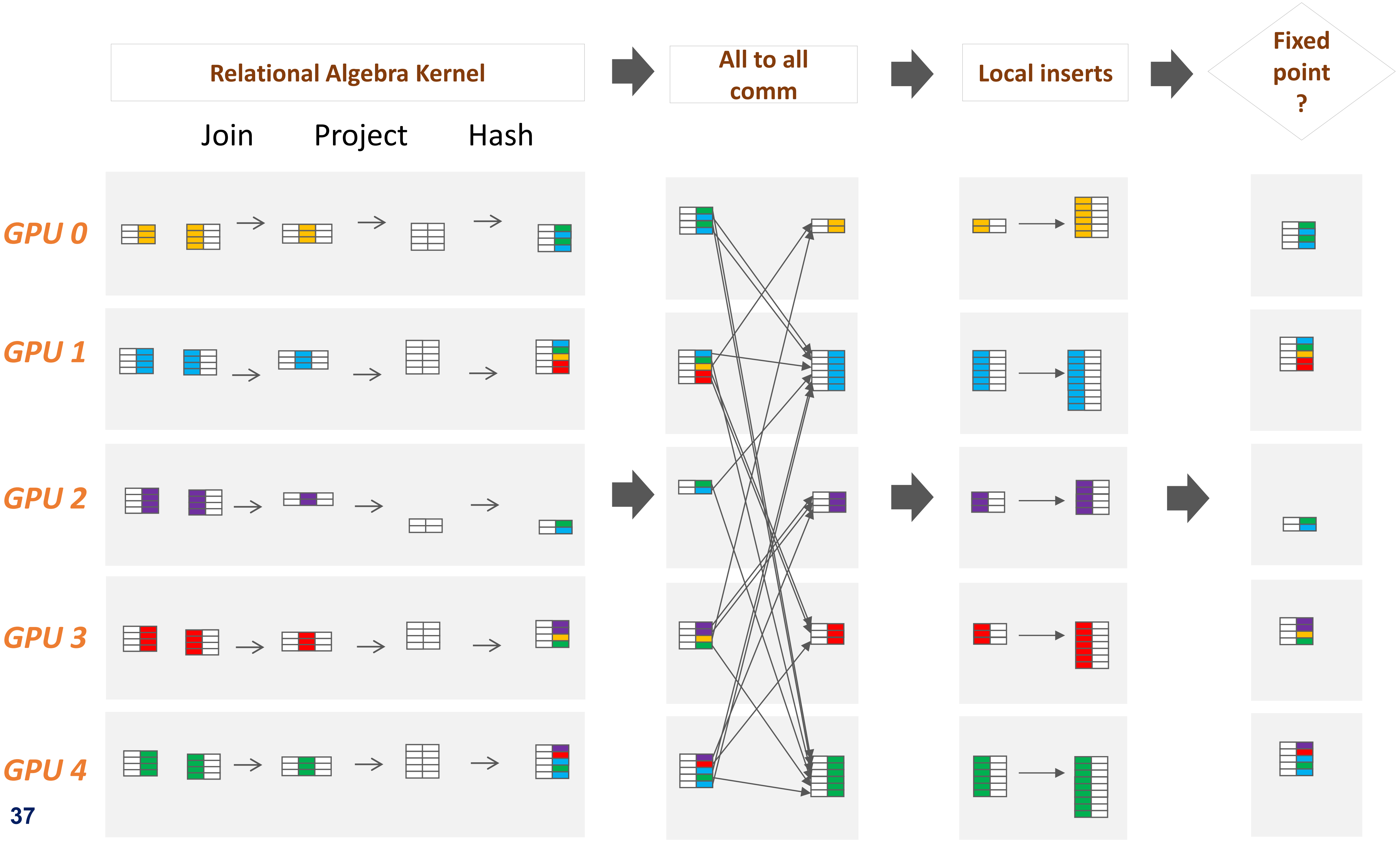
2 Data representation

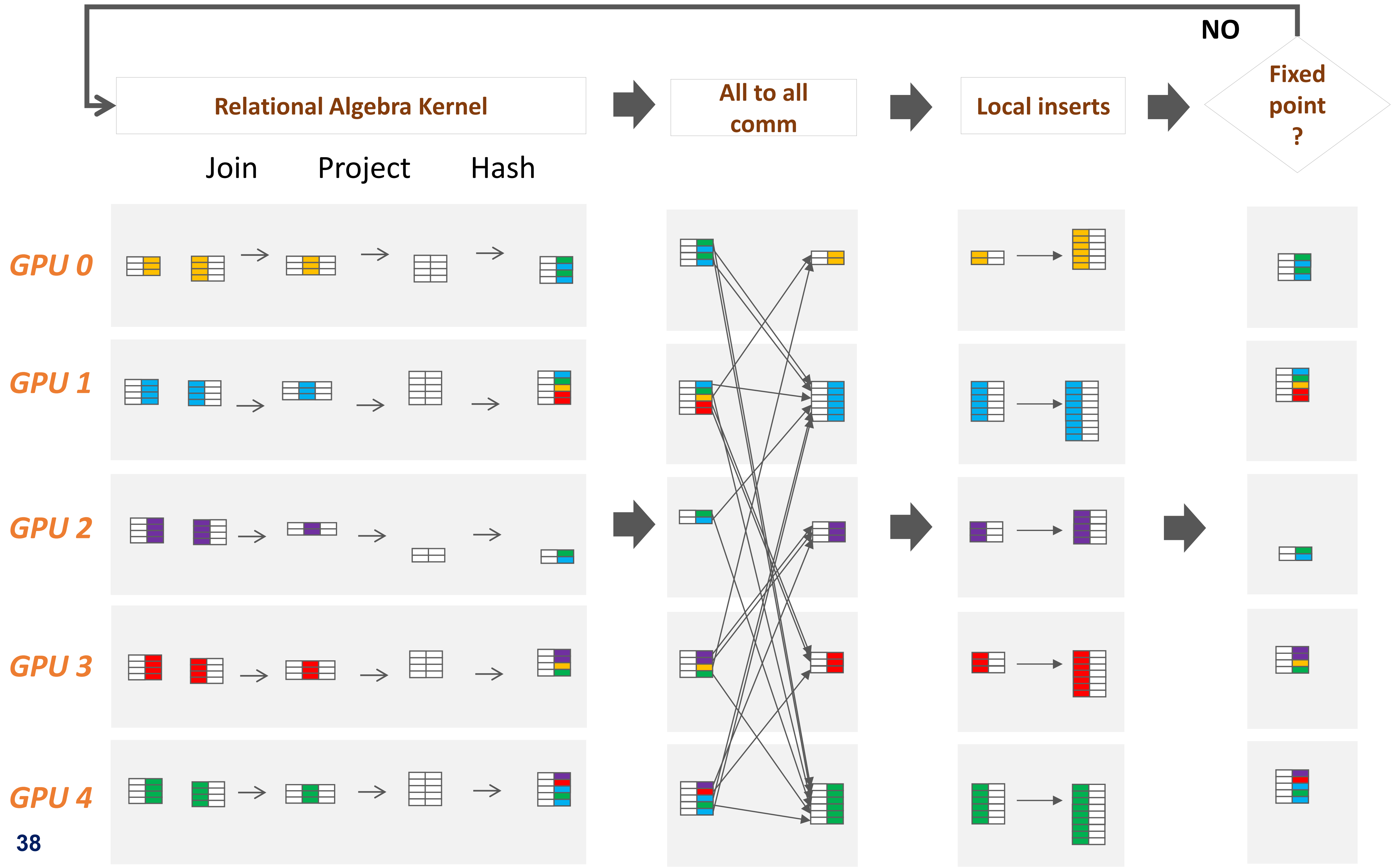
3 Communication

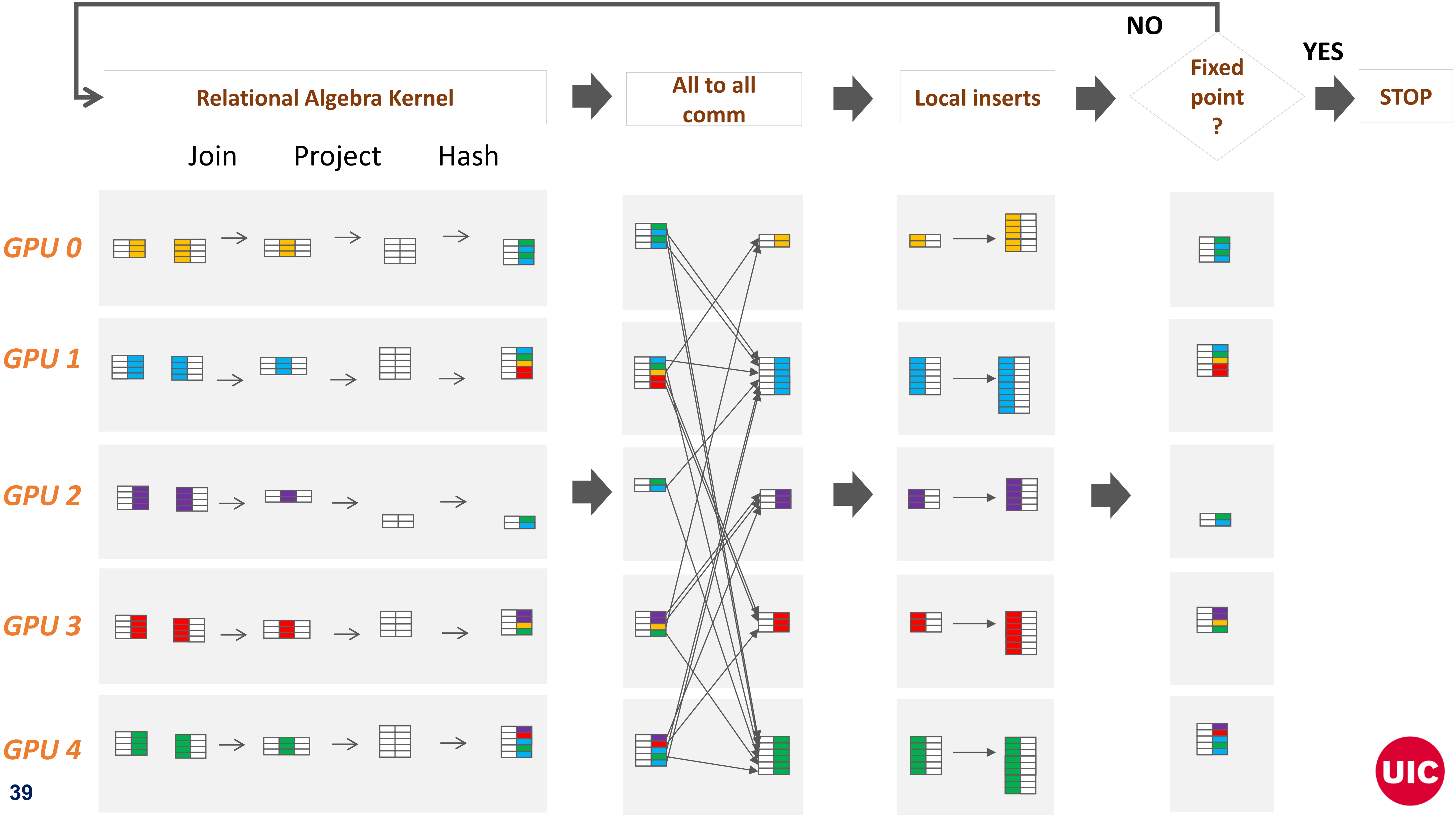


Kumar, S., & Gilray, T. (2020). Load-balancing parallel relational algebra. In *High Performance Computing: 35th International Conference, ISC High Performance 2020, Frankfurt/Main, Germany, June 22–25, 2020, Proceedings 35* (pp. 288-308). Springer International Publishing.

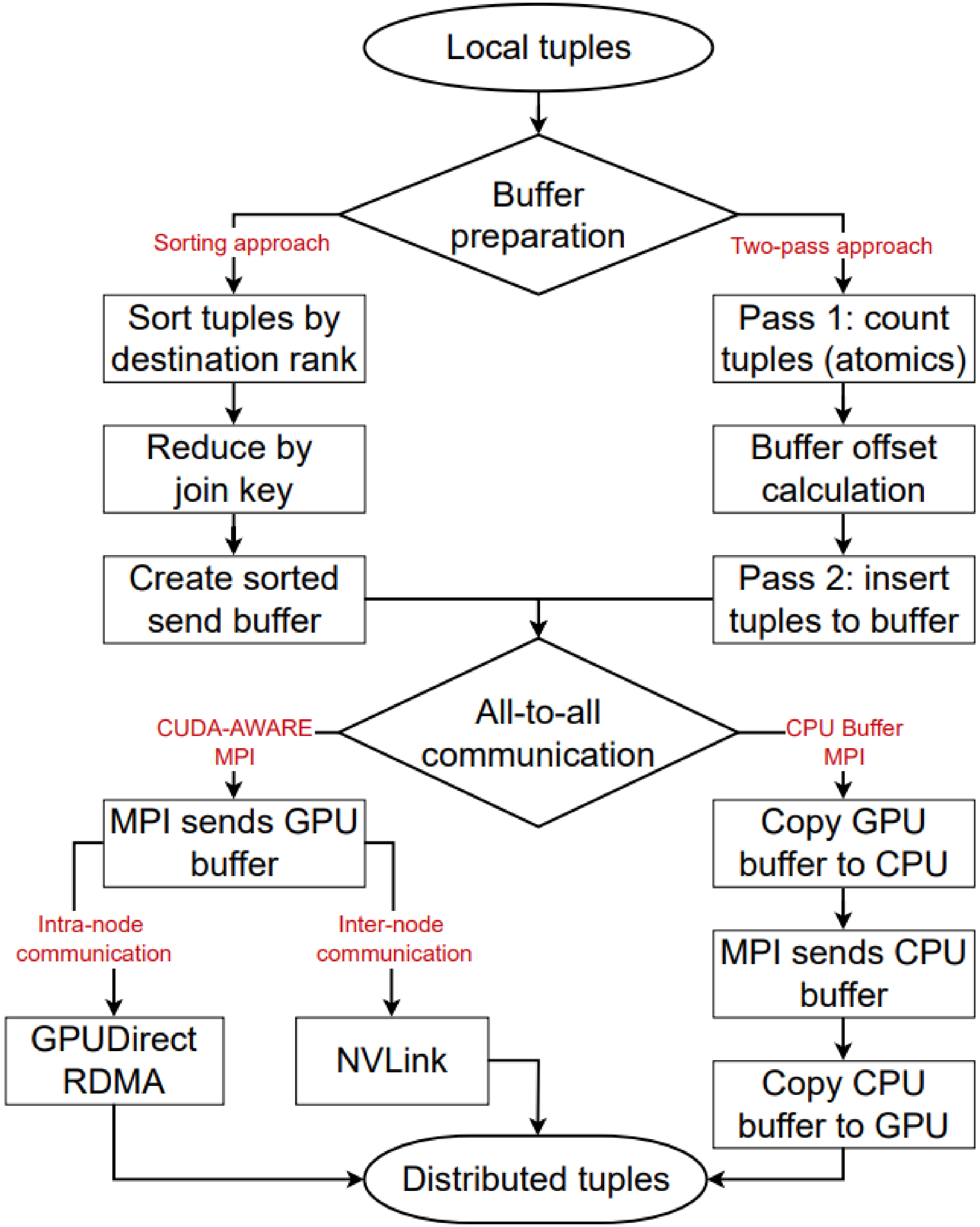

Relational Algebra Kernel
All to all comm
Local inserts
Join
Project
Hash
GPU 0
GPU 1
GPU 2
GPU 3
GPU 4


Relational Algebra Kernel
All to all comm
Local inserts
Fixed point ?
Join
Project
Hash
GPU 0
GPU 1
GPU 2
GPU 3
GPU 4


NO
Relational Algebra Kernel
All to all comm
Local inserts
Fixed point ?
Join
Project
Hash
GPU 0
GPU 1
GPU 2
GPU 3
GPU 4



Relational Algebra Kernel
All to all comm
Local inserts
Fixed point ?
NO
YES
STOP
Join
Project
Hash
GPU 0
GPU 1
GPU 2
GPU 3
GPU 4



All-to-all communication phases in MNMGDatalog

Evaluation

Evaluation environment, applications, datasets

Polaris supercomputer from **Argonne National Lab**

CPU: AMD EPYC 7543P processors with 32 cores

GPU: 4 NVIDIA A100 GPUs per node interconnected by NVLink

Software: CUDA (12), SLOG (32 threads), Soufflé (128 threads)

Apps: Transitive Closure, Same Generation, Weakly Connected Component

Datasets: Stanford large network, SuiteSparse, Road network

Baseline Datalog Engines:

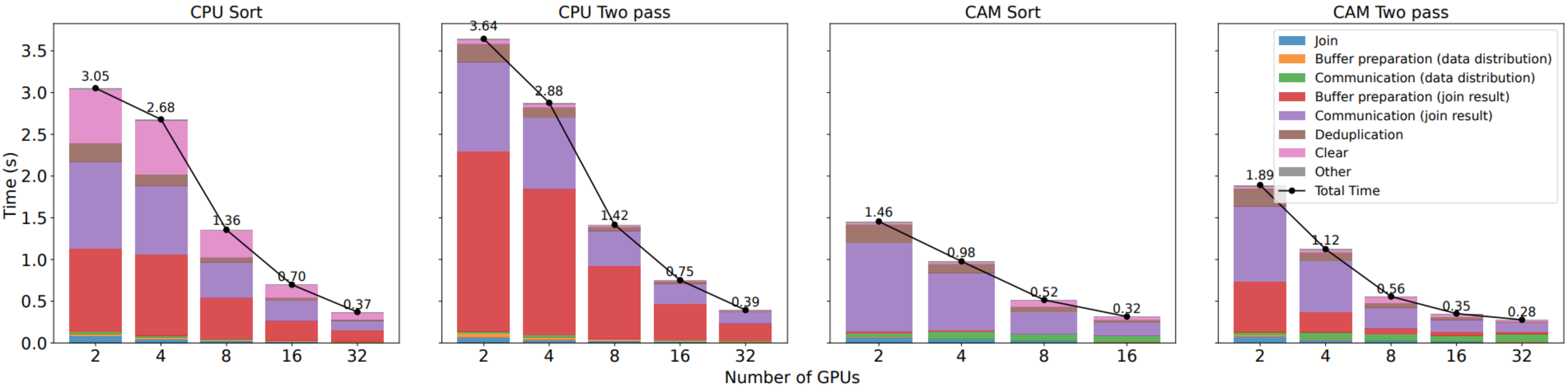
Multi-node multi-threaded: SLOG

Multi-threaded: Soufflé

Single-GPU: GPULog, GPUJoin, cuDF

- Argonne Leadership Computing Facility. 2022. Polaris. https://www.alcf.anl.gov/polaris
- Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data
- Timothy A. Davis and Yifan Hu. 2011. The University of Florida Sparse Matrix Collection. ACM Trans. Math. Softw. 38, 1, Article 1 (dec 2011), 25 pages. doi:10.1145/2049662.2049663

Single iteration of fixed-point benchmark



Single iteration of fixed-point iteration benchmark with 10M tuples

Single GPU benchmark

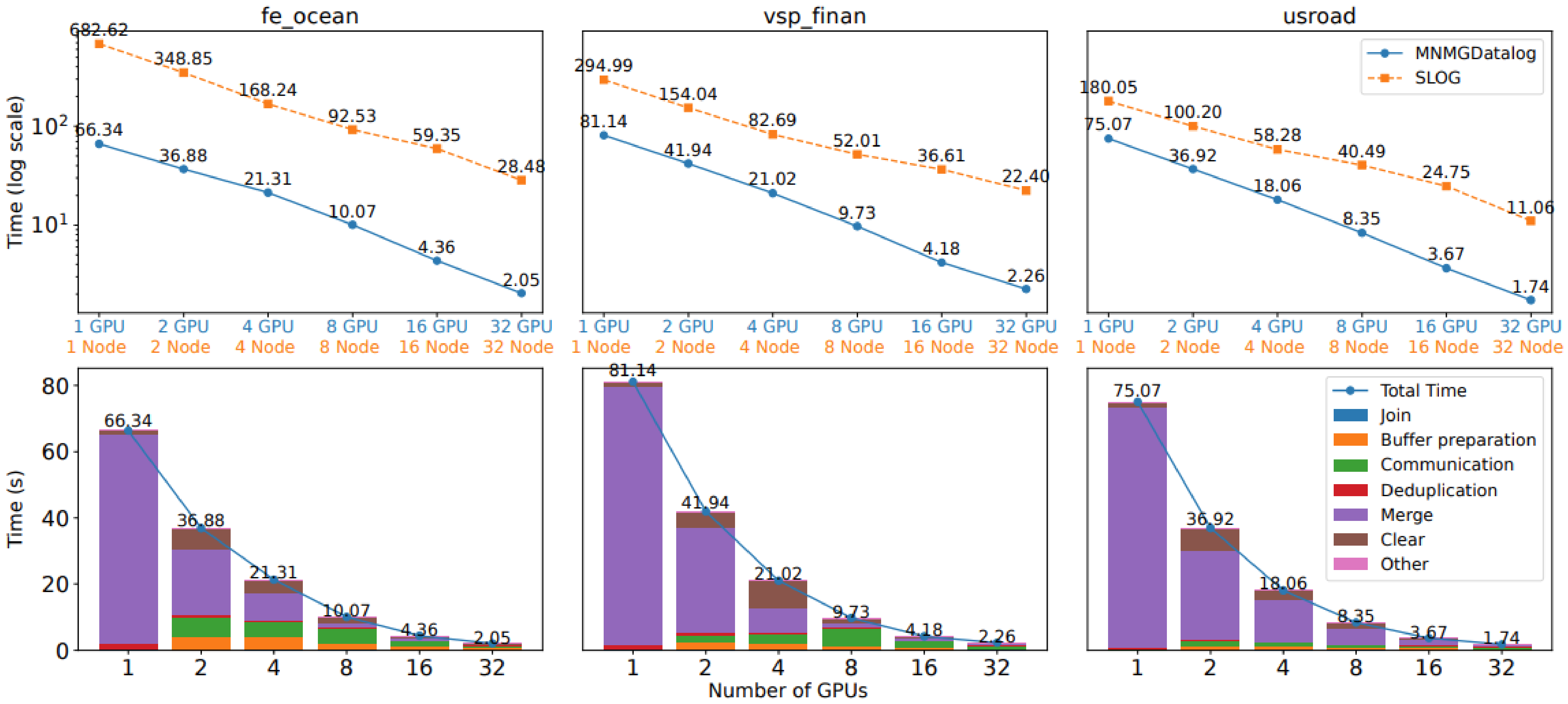
Dataset name	*TC* edges	Time (s) MNMGDATALOG	GPULOG	Soufflé	GPUJoin
com-dblp	1.91B	**13.58**	26.95	232.99	OOM
fe_ocean	1.67B	**66.34**	72.74	292.15	100.30
usroads	871M	**75.07**	78.08	222.76	364.55
vsp_finan	910M	**81.14**	82.75	239.33	125.94

Transitive closure benchmark on single-GPU

Dataset name	*SG* size	Time (s) MNMGDATALOG	GPULOG	Soufflé	cuDF
fe_body	408M	**9.08**	18.41	74.26	OOM
loc-Brightkite	92.3M	**1.66**	11.67	48.18	OOM
fe_sphere	205M	**3.55**	7.88	48.12	OOM
CA-HepTH	74M	**0.60**	4.79	20.12	21.24

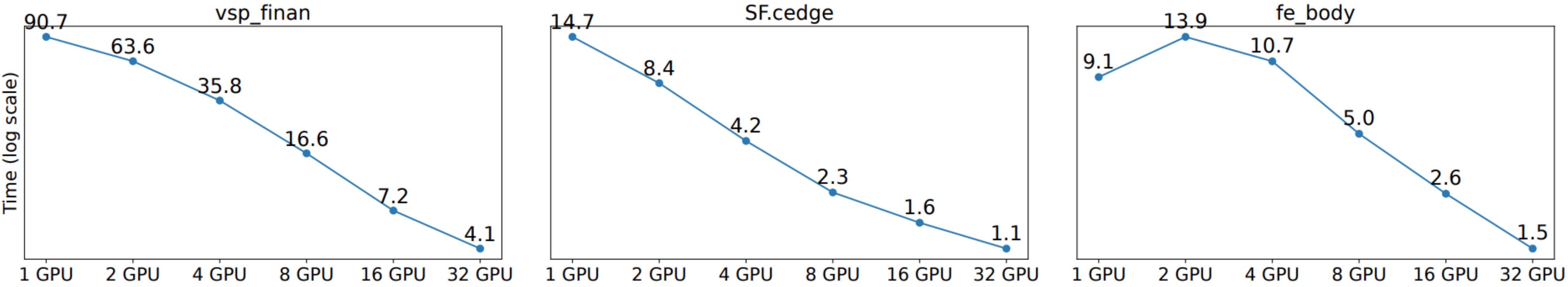
Same generation benchmark on single-GPU

Multi-node multi-GPU benchmark

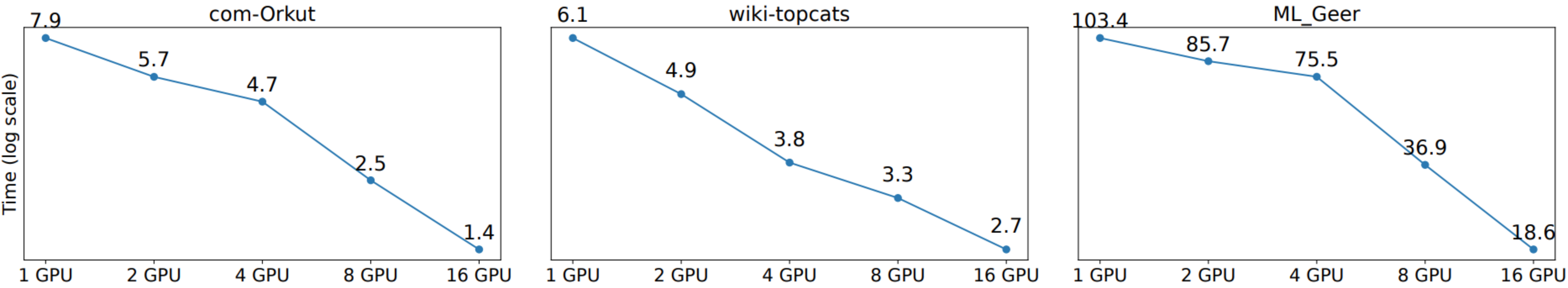


Transitive closure benchmark and breakdown up to 32 GPUs

Multi-node multi-GPU benchmark (Continue)



Same generation benchmark up to 32 GPUs



Weakly connected component benchmark (scalable recursive aggregation) up to 16 GPUs

Conclusion

Conclusion and future work

- Presented **MNMGDatalog** the first multi-node, multi-GPU Datalog engine
- Designed for efficient execution of recursive queries over internet-scale datasets in scale
- The highest-performant Datalog engine outperforming state-of-the-art
  - GPU-based engine (GPULog) by 7x
  - CPU-based engine (Soufflé) by 33x
  - Distributed engine (SLOG) by 32x
- Improve robustness and portability
  - Add per-iteration checkpoint/restart capability
  - More versions targeting different GPUs and HPC systems

https://github.com/harp-lab/MNMGDatalog

Thank You

ashov@uic.edu

UIC UNIVERSITY OF ILLINOIS CHICAGO