Accelerating Iterative Joins Toward a Modern Datalog Backend on GPU

Ahmedur Rahman Shovon (ashov@uic.edu), Sidharth Kumar (sidharth@uic.edu)
University of Illinois Chicago

Background

Declarative programming focuses on "WHAT" to achieve rather than "HOW".

Users

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

WHAT **SELECT** UserID **FROM** Users **WHERE** Country='USA'; HOW

Advanced approach: Logic programming (Datalog)

Datalog rules to compute Transitive Closure (TC) of a relation

```
TC(x, y) :- Edge(x, y).
TC(x, z) :- TC(x, y), Edge(y, z).
```

Operationalized as a **fixed-point iteration** using F_G

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

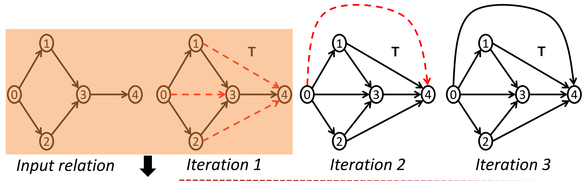
Datalog rules compiled down to relational algebra operators (Union, Projection, Rename, Join)

Accelerating the **iterative joins** is crucial as it is the most expensive operation in Datalog rule evaluation

Challenges

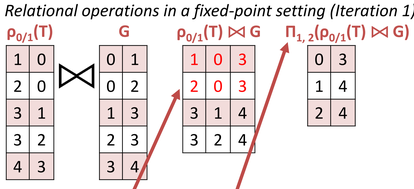
Efficiently mapping **iterative join** operations to **GPUs** poses unique challenges:

Iterations in Transitive Closure (TC) Computation



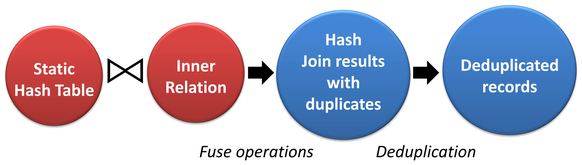
Input relation *Iteration 1* *Iteration 2* *Iteration 3*

Relational operations in a fixed-point setting (Iteration 1)



1. Duplicates on join results
2. Storing intermediate results
3. Fusing multiple operations
4. Redundant computations

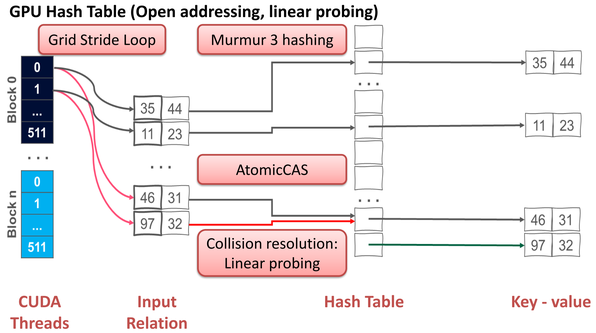
We present a single **GPU-based open addressing hash join (GPUJoin)** designed to accelerate **iterative joins** for Datalog evaluation:



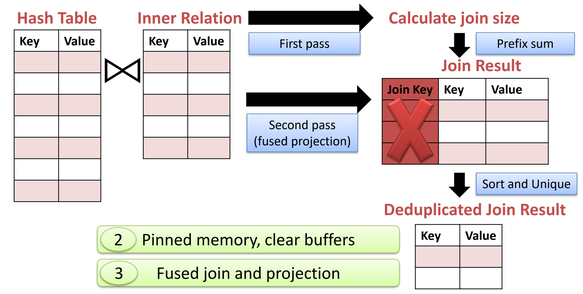
Accelerating Iterative Joins with GPUJoin

We introduce a GPU-optimized open addressing hash table tailored for relational data to perform binary hash joins.

GPU Hash Table (Open addressing, linear probing)



Performing Hash Join on GPU



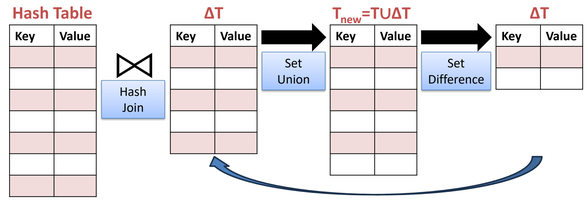
2. Pinned memory, clear buffers
3. Fused join and projection

Transitive closure computation (naïve vs semi-naïve evaluation)

```
T=∅
Loop
  T_new=E(x,y)∪Π_x,y(E(x,z)⋈T(z,y))
  If T_new=T:
    then break
  EndIf
  T=T_new
EndLoop
```

```
T=∅, ΔT=E
Loop
  ΔT(x,y)=Π_x,y(E(x,z)⋈ΔT(z,y))
  If ΔT=∅:
    then break
  EndIf
  T=T∪ΔT
EndLoop
```

Transitive closure computation single iteration (semi-naïve evaluation)

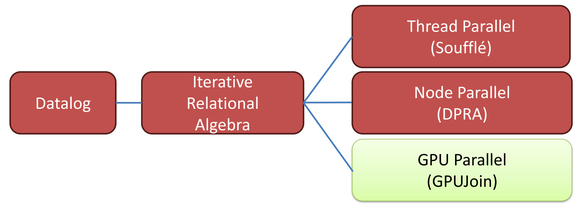


Continue until ΔT≠∅

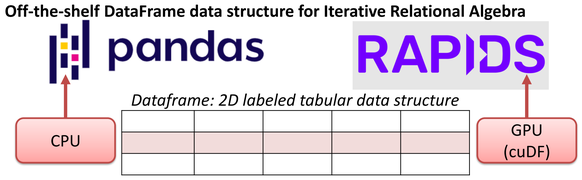
1. Efficient deduplication
4. Avoid redundant computations

Experiments

GPUJoin lays the groundwork for a modern Datalog backend on GPU, specifically tailored to iterative relational algebra.



Off-the-shelf DataFrame data structure for Iterative Relational Algebra



DataFrame based Datalog applications

✓ Advantages	✗ Limitations
✓ Abstract memory management	✗ No fusing
✓ Abstract thread block configuration	✗ Memory + computation overhead
✓ Same API signatures for CPU, GPU	✗ No consecutive operation
✓ Easy-to-code interface	✗ Memory limitation

CUDA advantages over DataFrame

- Fuse operations
- Thread-block configuration
- Memory management
- Optimize data structure

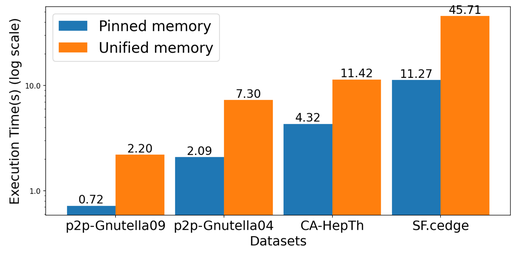
Experiment platform and datasets

ThetaGPU supercomputer from **Argonne National Lab**
CPU: AMD EPYC 7742 processors with 3.31GHz clock speed, 128 cores
GPU: NVIDIA A100 Tensor Core GPU with 40GB GPU memory, 108 SM
Environment: CUDA (11.4, 3456x512), Soufflé (2.3, 128 threads), cuDF (22.06)
Datasets: Stanford large network, SuiteSparse, Road network datasets

Our GPU Hash Table performance

Build rate:
- ✓ Random synthetic graph: **400 million keys/second**
- ✓ String graph: **4 billion keys/second**

Pinned vs unified memory performance comparison for iterative joins (TC)



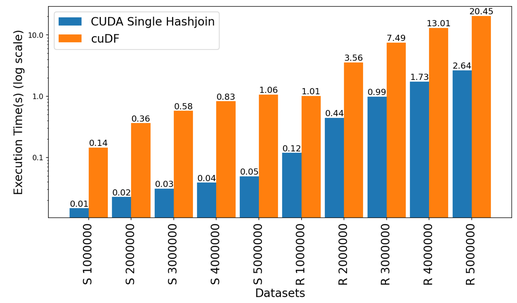
Experiments (Continue)

We evaluated GPUJoin on large datasets with state-of-the-art CPU Datalog solver (Soufflé) and GPU library (cuDF).

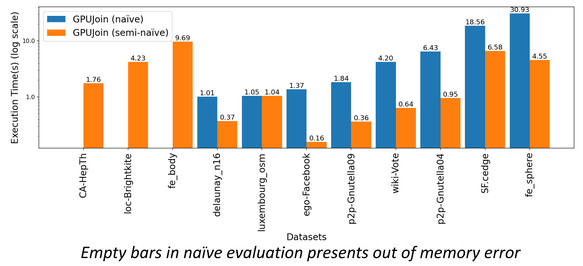
Transitive closure performance comparison (GPUJoin vs Soufflé vs cuDF)

Dataset	Type	Rows	TC size	Iterations	GPUJoin(s)	Soufflé(s)	cuDF(s)
fe_ocean	U	409,593	1,669,750,513	247	138.237	536.233	Out of Memory
p2p-Gnutella31	D	147,892	884,179,859	31	Out of Memory	128.917	Out of memory
usroads	U	165,435	871,365,688	606	364.554	222.761	Out of Memory
fe_body	U	163,734	156,120,489	188	47.758	29.07	Out of Memory
loc-Brightkite	U	214,078	138,269,412	24	15.88	29.184	Out of Memory
SF.cedge	U	223,001	80,498,014	287	11.274	17.073	64.417
fe_sphere	U	49,152	78,557,912	188	13.159	20.008	80.077
CA-HepTh	D	51,971	74,619,885	18	4.318	15.206	26.115
p2p-Gnutella04	D	39,994	47,059,527	26	2.092	7.537	14.005
p2p-Gnutella09	D	26,013	21,402,960	20	0.72	3.094	3.906
wiki-Vote	D	103,689	11,947,132	10	1.137	3.172	6.841
cti	U	48,232	6,859,653	53	0.295	1.496	3.181
delaunay_n16	U	196,575	6,137,959	101	1.137	1.612	5.596
luxembourg_osm	U	119,666	5,022,084	426	1.322	2.548	8.194
ego-Facebook	U	88,234	2,508,102	17	0.544	0.606	3.719
cal.cedge	U	21,693	501,755	195	0.489	0.455	2.756
TG.cedge	U	23,874	481,121	58	0.198	0.219	0.857
wing	U	121,544	329,438	11	0.085	0.193	0.905
OL.cedge	U	7,035	146,120	64	0.148	0.181	0.523

Single join performance comparison (GPUJoin vs cuDF)



Naïve vs semi-naïve evaluation of TC computation in GPUJoin



Empty bars in naïve evaluation presents out of memory error

Conclusion

Our contributions:
- High performance GPU hash table for iterative RA
- Operations optimization (fuse join and projection)
- Overcome deduplication challenge
- Efficient GPU memory management (pinned memory and buffer clearance)
- Semi-naïve evaluation for avoiding redundant computation
- Open-sourced code, data, documentation: https://github.com/harp-lab/usenixATC23 (Published in USENIX ATC 2023, IA^3 2022)

Acknowledgement

This work was funded in part by NSF RII Track-4 award 2132013, NSF collaborative research award 2217036, and NSF collaborative research award 2221811. We are thankful to the ALCF's Director's Discretionary (DD) program for providing us with compute hours to run our experiments on the ThetaGPU supercomputer located at the Argonne National Laboratory.