universität innsbruck

amazon

Brie: A Specialized Trie for Concurrent Datalog

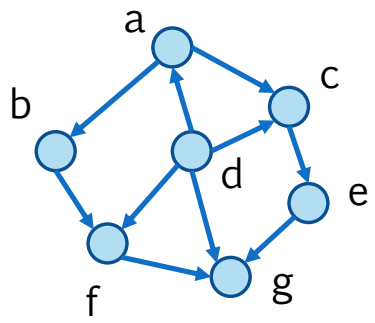
Herbert Jordan[1], Pavle Subotić[3], David Zhao[2], and Bernhard Scholz[2]

PMAM 2019, 17 February 2019, Washington, DC

DPS
Distributed and Parallel Systems

1) University of Innsbruck
2) University of Sydney
3) Amazon

Datalog (by Example)



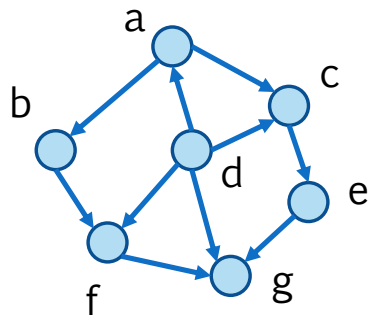
graph

from	to
a	b
a	c
b	f
c	e
d	a
d	c
…	…

edge relation

Are there cycles?

Datalog (by Example)



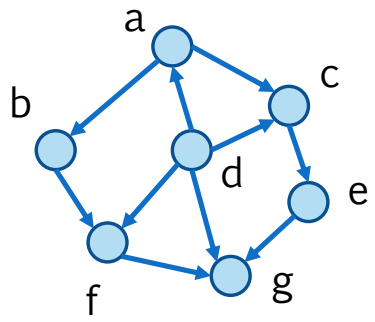
graph

from	to
a	b
a	c
b	f
c	e
d	a
d	c
…	…

edge relation

Is the graph connected?

Datalog (by Example)



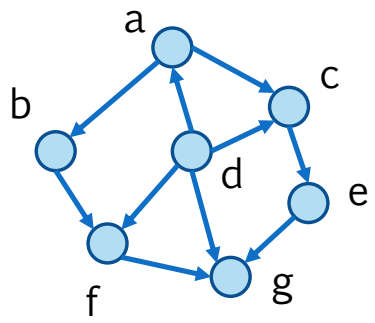
graph

from	to
a	b
a	c
b	f
c	e
d	a
d	c
…	…

edge relation

Which nodes are connected?

Datalog (by Example)



graph

from	to
a	b
a	c
b	f
c	e
d	a
d	c
…	…

edge relation

```
path(X,Y) :- edge(X,Y).

path(X,Z) :- path(X,Y),
             edge(Y,Z).
```

Datalog
query

Datalog

› Benefits:
- a concise formalism for powerful data analysis
- lately major performance improvements and tool support

› Applications:
- data base queries
- program analysis
- security vulnerability analysis
- network analysis

} 100s of relations and rules, billions of tuples, all in-memory

Query Processing

relations → set of integer tuples

rules → sequence of relational algebra operations on sets

Example

```
path(X,Z) :- path(X,Y), edge(Y,Z).
```

$$delta \leftarrow path$$

while ($delta \neq \emptyset$) {

$$new \leftarrow \pi(delta \bowtie edge) \setminus path$$

$$path \leftarrow path \cup new$$

$$delta \leftarrow new$$

}

computational expensive and dominating part

Needed

› efficient data structure for relations
  – maintain set of n-dimensional tuples

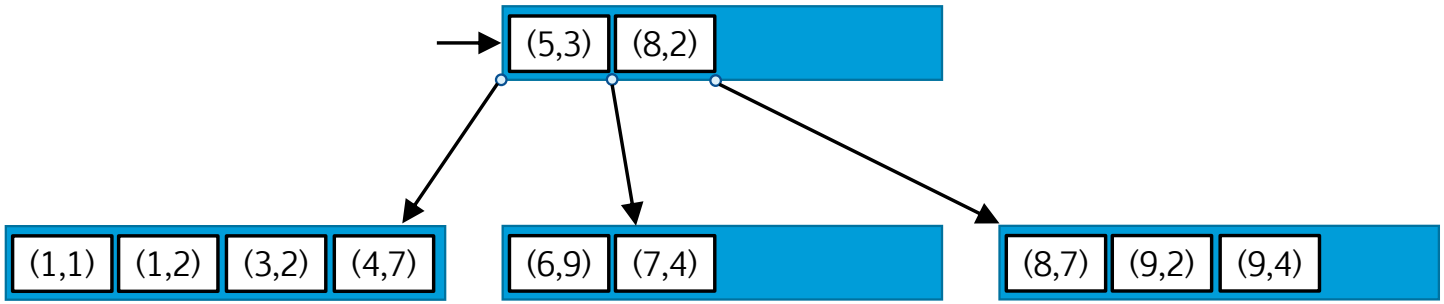
  – efficient support for
    › insertion,
    › scans,
    › range queries,
    › membership tests,
    › emptiness checks

  } well supported by **B-trees**

  – efficient synchronization of concurrent inserts

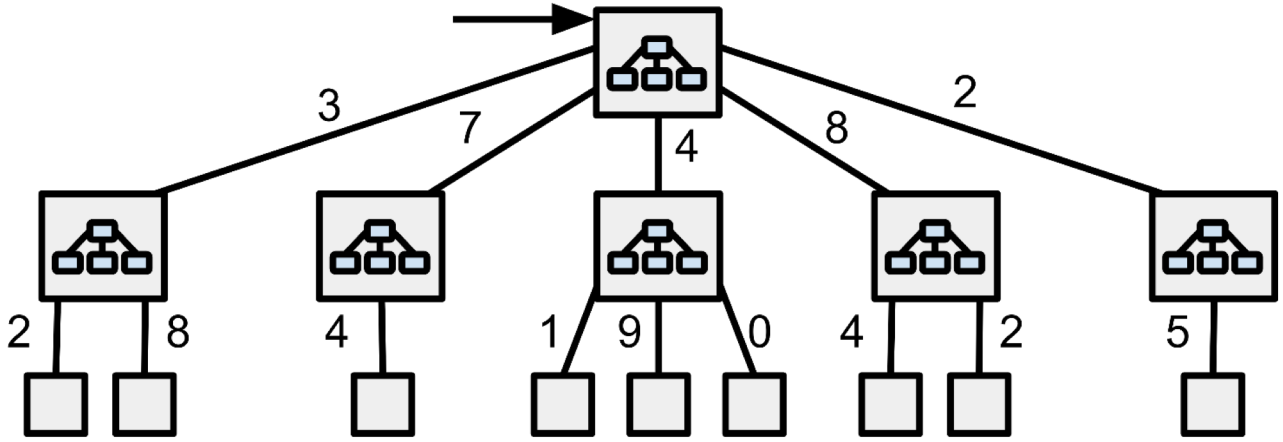
  } challenging

B-tree Issues

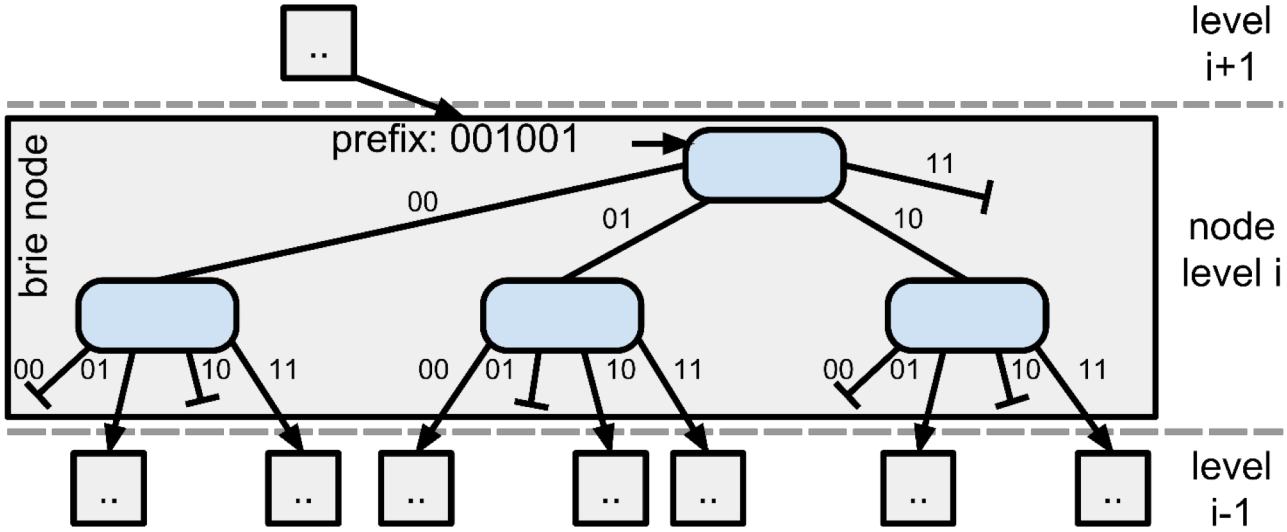


› Concurrent inserts:
  – require sophisticated locking scheme
  – while holding locks, costly operations are performed
    › binary search operations, and inserts in sorted arrays

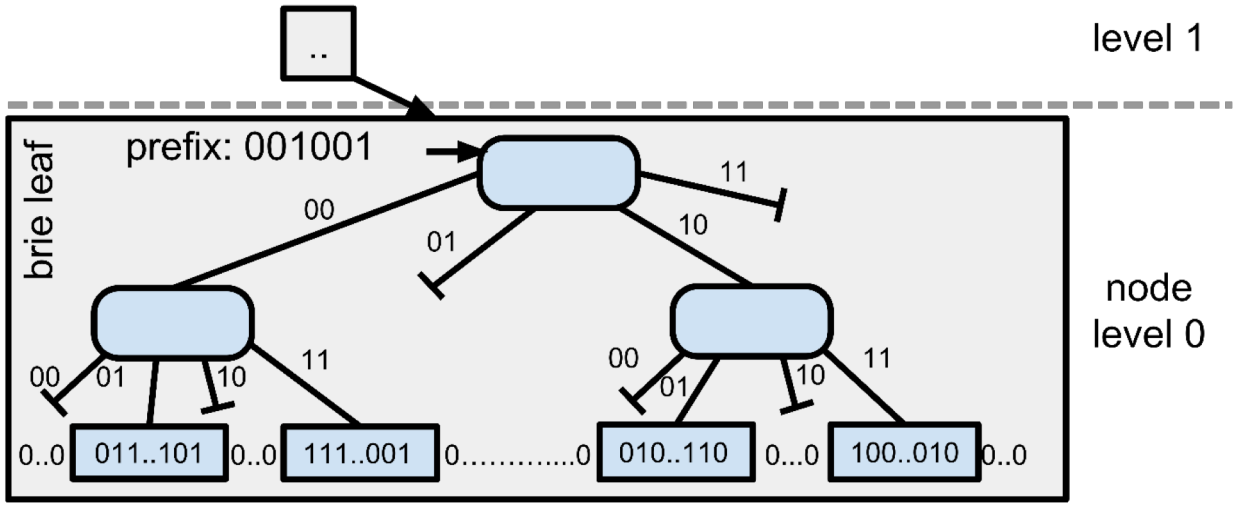
Brie



Brie – Inner Node



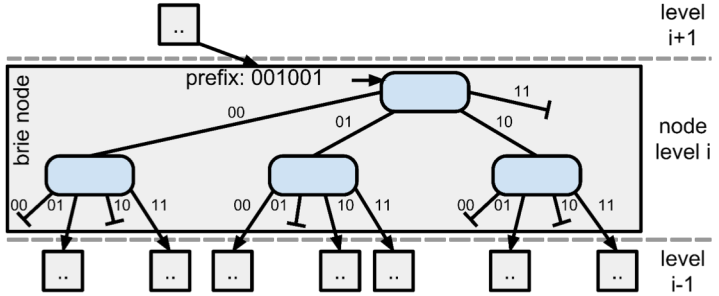
Brie – Leaf Node

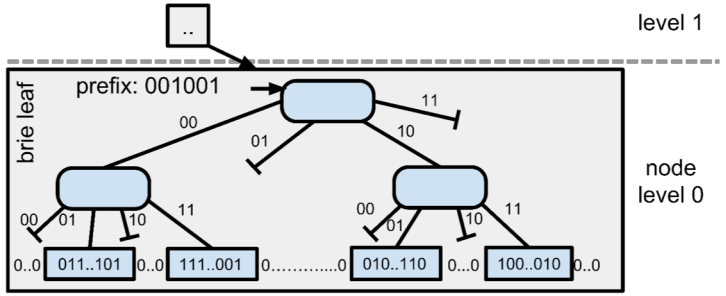


Synchronizing Inserts

› Insertion

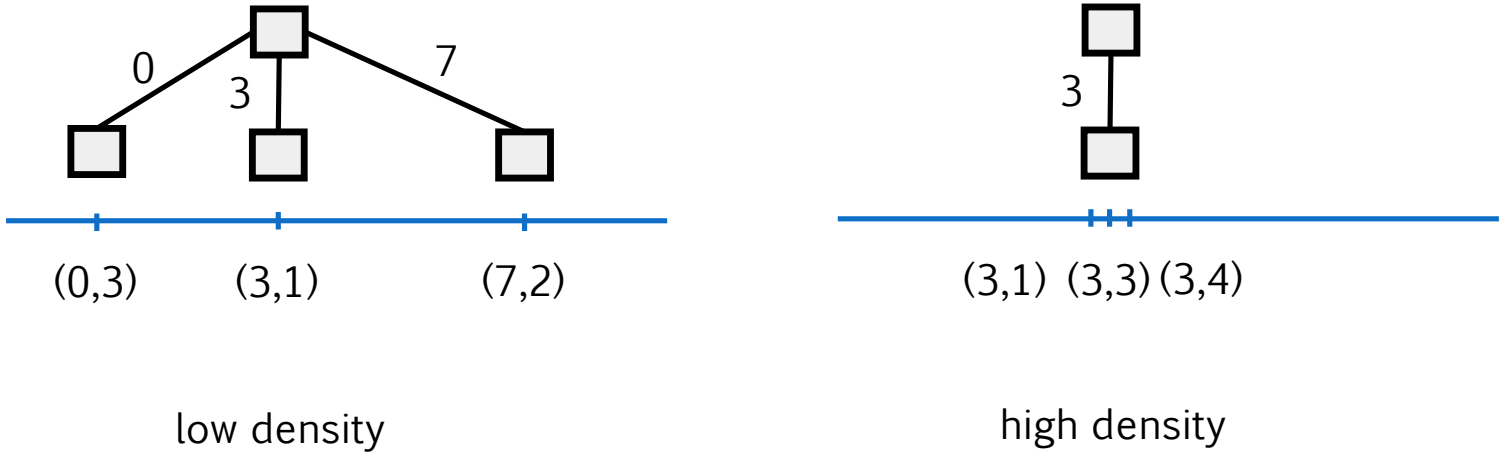
1. navigate down the tree
   - › insert sub-trees on demand using CAS
2. If inner node tree needs to grow
   - › introduce new root node using CAS
3. add 1-bit to leaf level mask
   - › using atomic bitwise or





Data Density

Performance is density dependent:

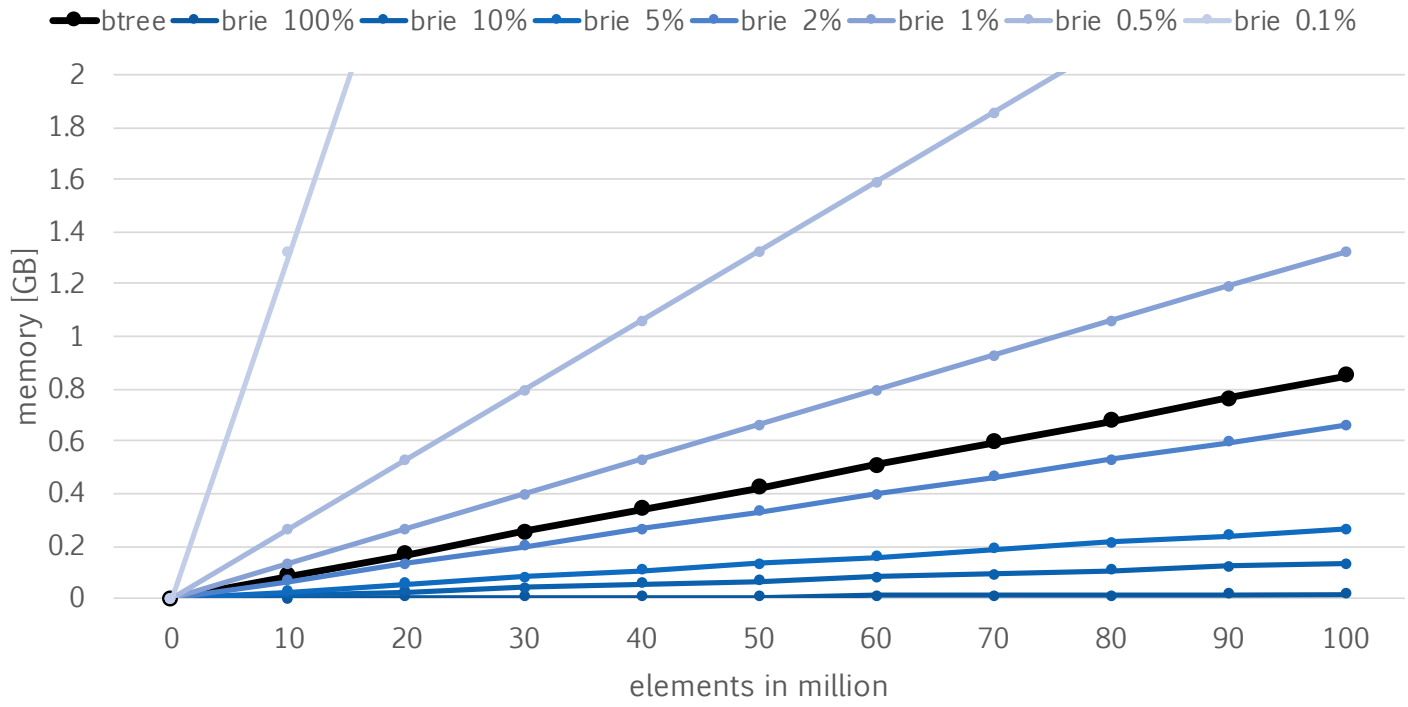


low density

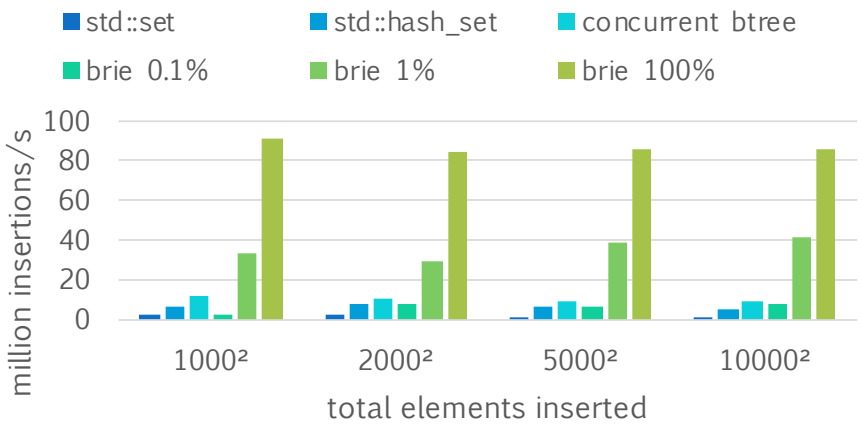
high density

Density: ratio of included points vs. spanned interval

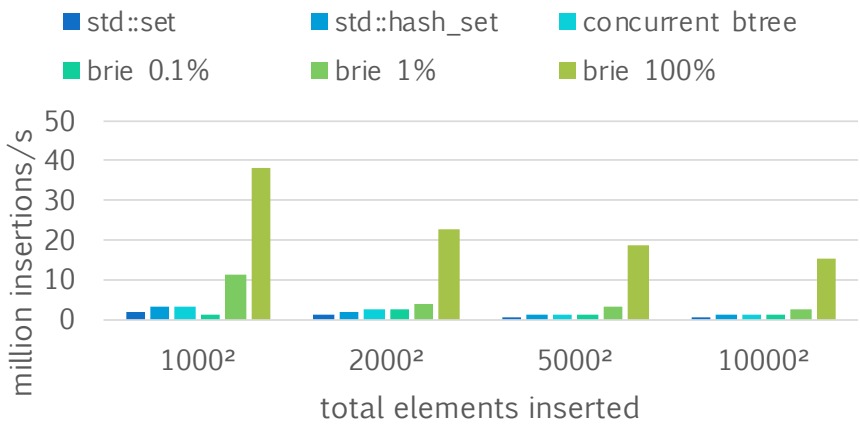
Memory Usage



Sequential Performance

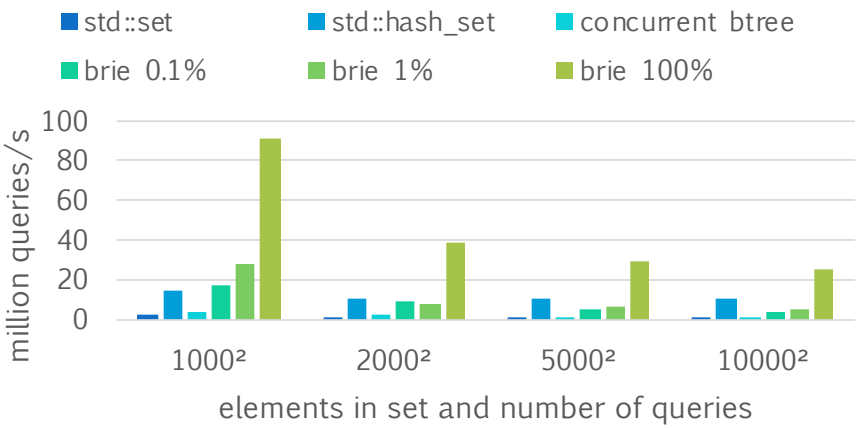


ordered insertion

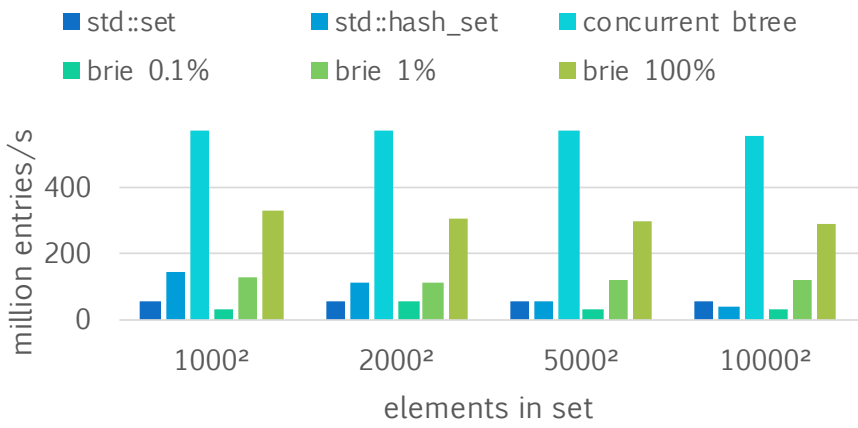


random order insertion

Sequential Performance (2)

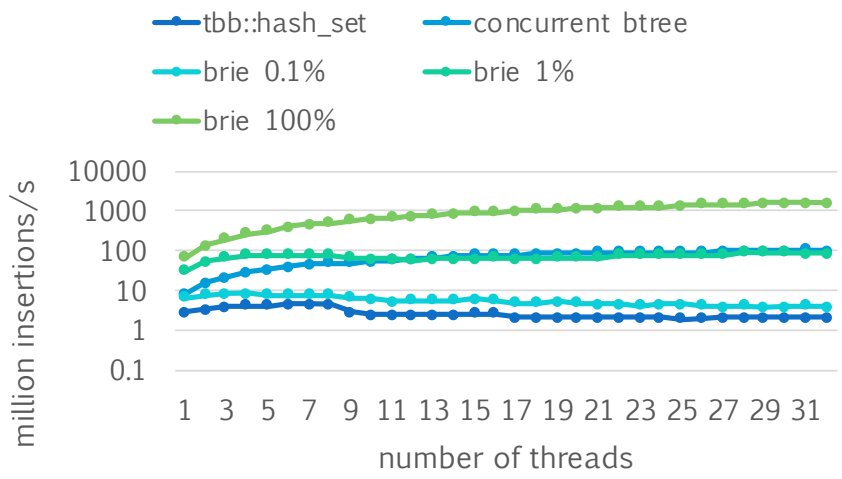


membership test (random order)

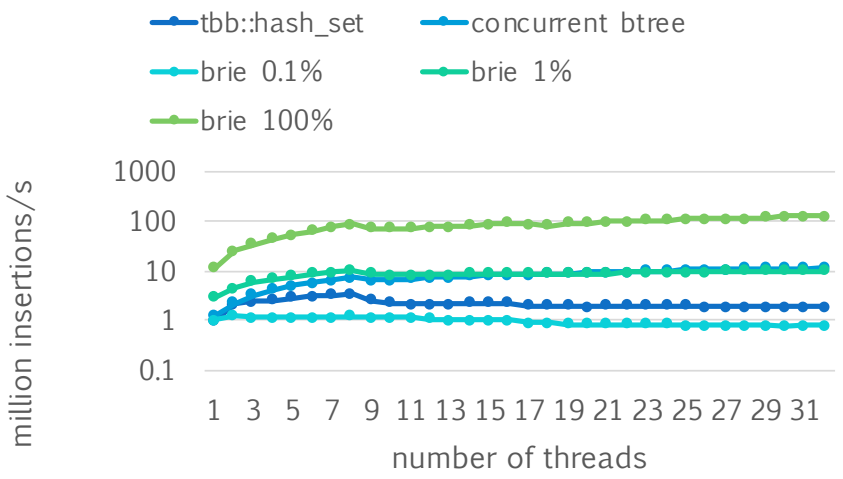


full range scan

Parallel Performance



ordered insertion

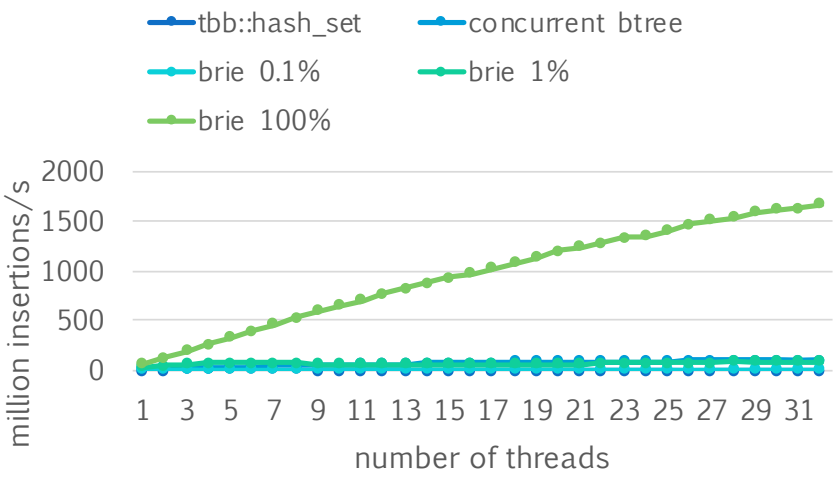


random order insertion

4x8 core Intel Xeon E5-4650

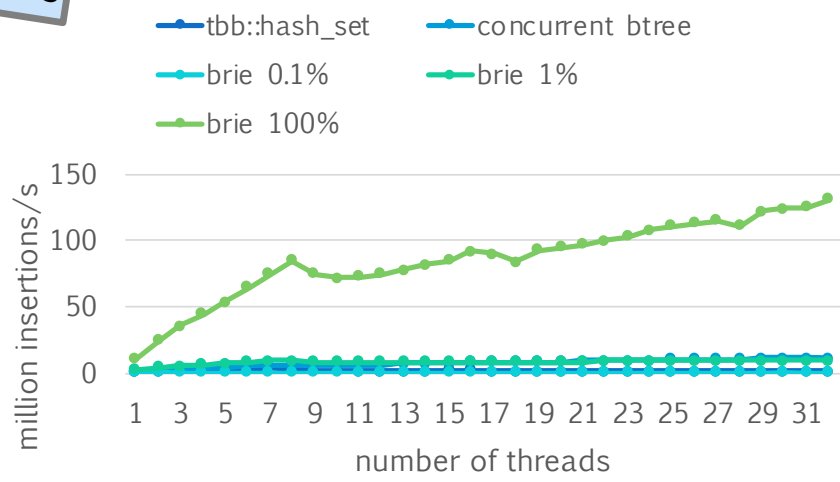
Parallel Performance

up to 15x faster than B-trees



ordered insertion

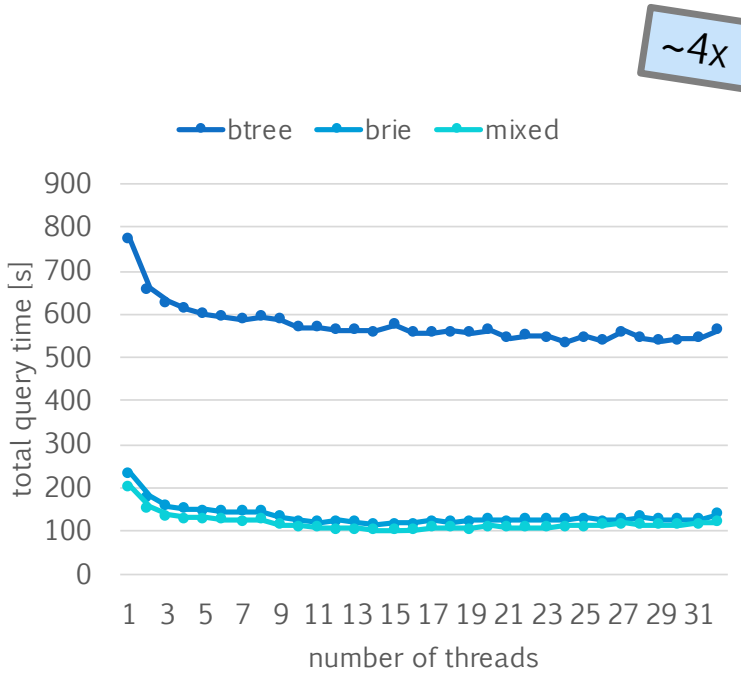
up to 11x faster than B-trees

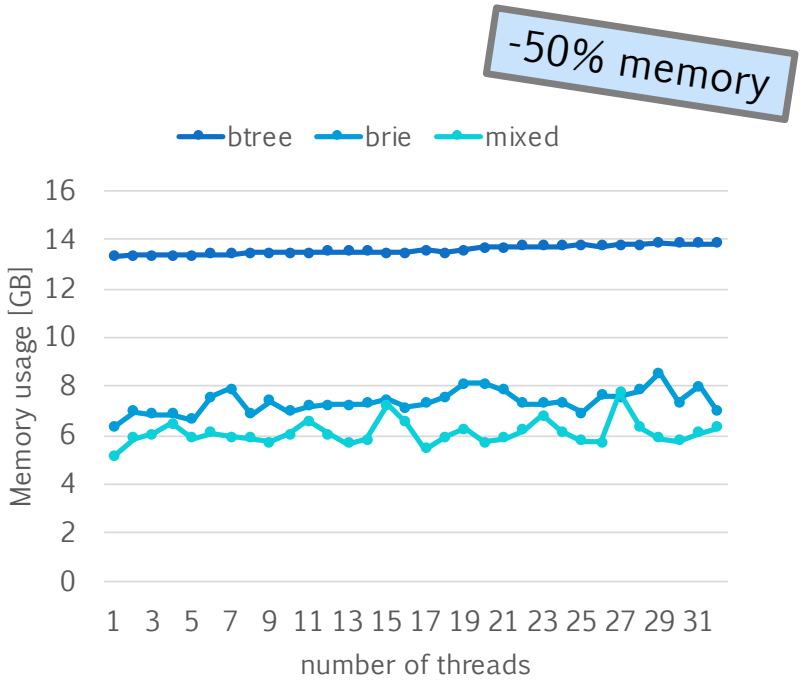


random order insertion

4x8 core Intel Xeon E5-4650

Datalog Query Processing





context sensitive
var-points-to analysis

Conclusion

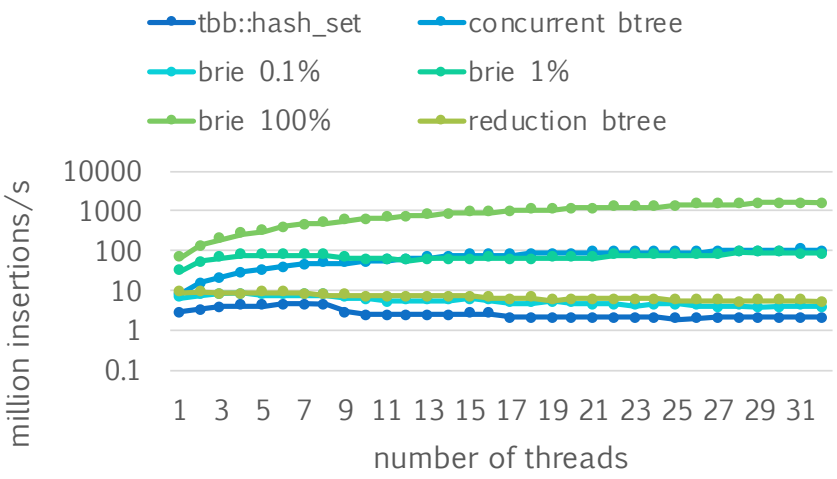
- Developed concurrent set for Datalog relations:
  - Trie derived structure + blocked nodes
    - enables fast relational operations
  - Low overhead synchronization
    - atomic operation based synchronization sufficient
- Results:
  - up to **5-17x** faster for sequential insert and query operations
  - up to **15x** faster for parallel insertion operations
  - up to **4x** faster and **50%** less memory for real-world **query processing**
- Future work:
  - investigate other data structures for specialized use cases

Thank you!

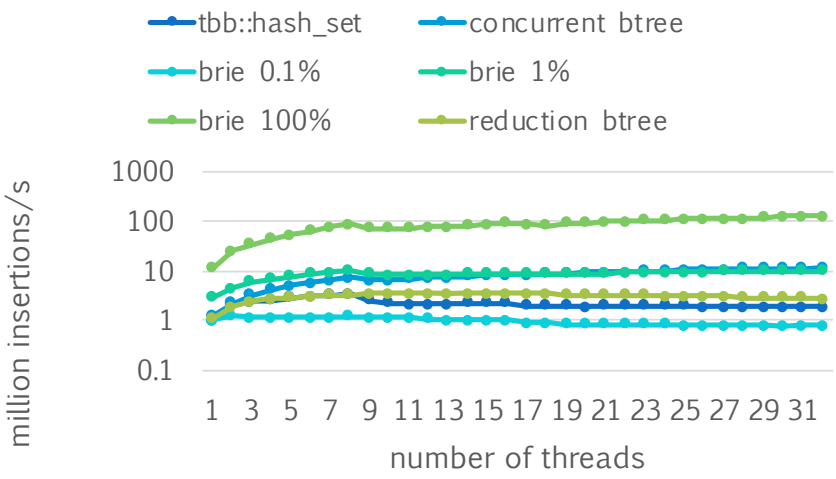
visit us on https://souffle-lang.github.io

sources: https://github.com/souffle-lang/souffle/blob/master/src/Brie.h

Parallel Performance



ordered insertion

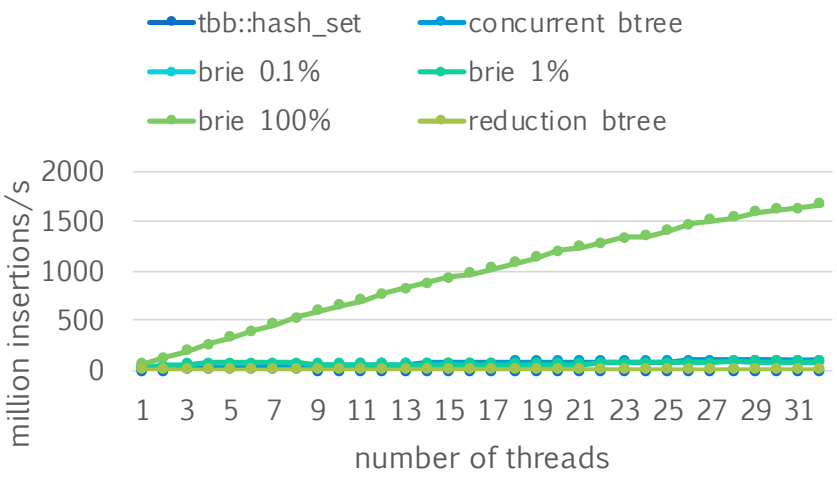


random order insertion

4x8 core Intel Xeon E5-4650

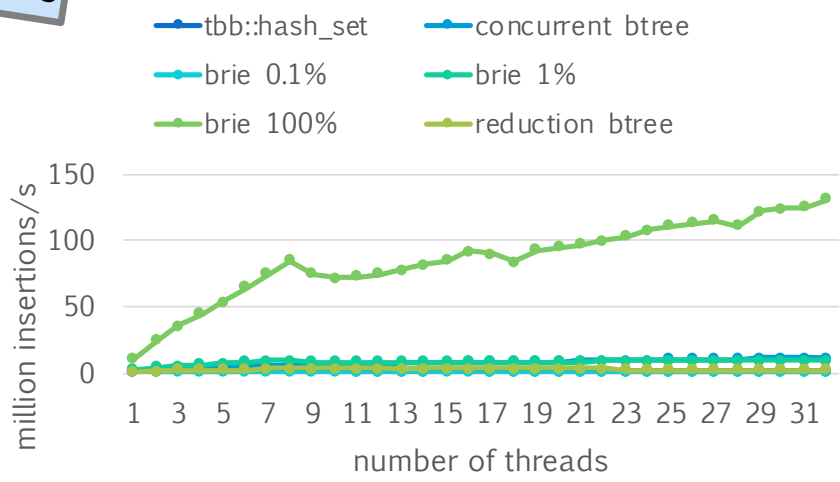
Parallel Performance

up to 15x faster than B-trees



ordered insertion

up to 11x faster than B-trees



random order insertion

4x8 core Intel Xeon E5-4650

Example

```
path(X,Z) :- path(X,Y),
             edge(Y,Z).
```

$$delta \leftarrow path$$

while ($delta \neq \emptyset$) {

$$new \leftarrow \pi(delta \bowtie edge) \setminus path$$

$$path \leftarrow path \cup new$$

$$delta \leftarrow new$$

}