A Macro-Benchmarking Library for the QLever SPARQL Engine

Andre Schlegel

Institute for computer science at the University of Freiburg, Germany

02.08.2024

Problem

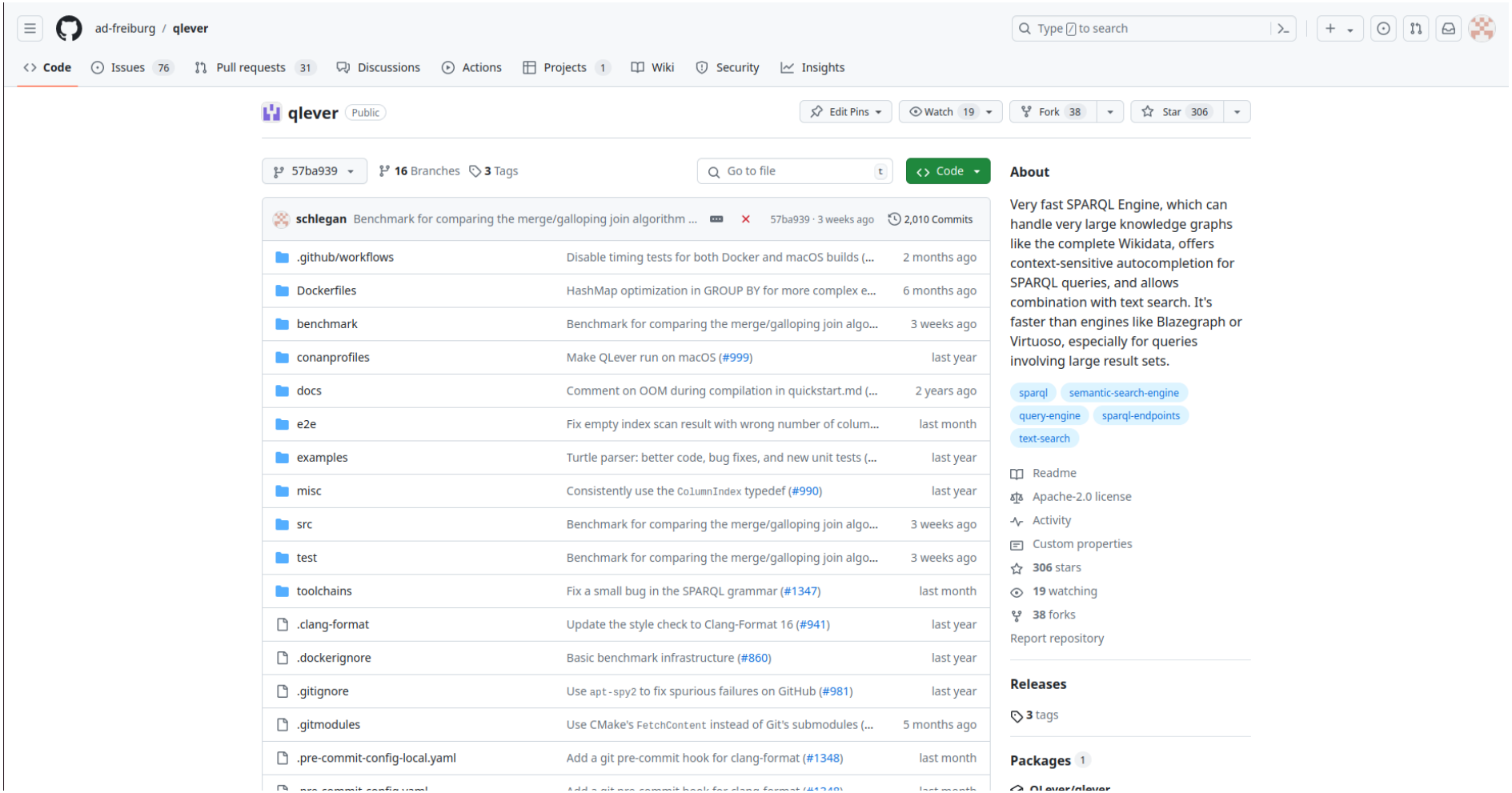


Figure: Project page of the “QLever SPARQL engine” [1]

Join operator

Left input table

6	4	1
6	4	3
2	4	9
2	2	3
9	1	9

Right input table

9	1	9	3
5	3	5	3
8	4	9	3

Join operator

Left input table

6	4	1
6	4	3
2	4	9
2	2	3
9	1	9

Right input table

9	1	9	3
5	3	5	3
8	4	9	3

Join operator

Left input table

6	4	1
6	4	3
2	4	9
2	2	3
9	1	9

Right input table

9	1	9	3
5	3	5	3
8	4	9	3

Output table

6	4	1	9	9	3
6	4	3	5	5	3
2	2	3	5	5	3

Merge and galloping join algorithm

- Used by “QLever SPARQL engine” [1]
- Input tables sorted by join column
- Output table sorted by join column

Hash join algorithm

- Input tables not sorted
- Output table not sorted

Simplified hash join algorithm

- Hash map
  - Key: Join column entries smaller input table
  - Value: Lists of all smaller input table rows with the key as join column entry
- Iterate bigger input table join column

Which has the shorter execution time?

More importantly: **When?**

Which has the shorter execution time?

- Always?
- Only specific situations or inputs?
  - Switch between algorithms

Which has the shorter execution time?

- **Not** theoretical algorithm execution time
- **Implementation** execution time

Implementation execution time

- Implementation details
- Used library functions execution times
- Used third-party library functions execution times

Questions?

My solution

Macro-benchmarking library

Benchmark

- “a computer program that measures the ... speed of computer software ... ” [2]
- Execution time

Benchmark

- Micro benchmark
  - Short execution times
  - Normally shorter than 0.1 seconds
- Macro benchmark
  - Long execution times
  - Normally longer than 1 second

Macro benchmark

- “QLever SPARQL engine” focus short execution times [1]
- In context: multiple seconds to multiple days
- Macro benchmark better fit

Macro benchmark suite

Set of:

Freestanding macro benchmarks

CLI output excerpt from execution of “BenchmarkExamples.cpp” in “QLever SPARQL engine”, adjusted for readability [1]

```
Single measurement 'Exponentiate once'
    time: 1e-06s

Single measurement 'Recursivly exponentiate multiple
                    times'
    metadata: {"amount-of-exponentiations":
                10000000000}
    time: 7.55948s
```

Macro benchmarks organized in tables

CLI output excerpt from execution of “BenchmarkExamples.cpp” in “QLever SPARQL engine”, adjusted for readability [1]

Table 'Exponents with the given basis'

Basis	0	1	2	3	4
2	0.0000	0.0000	0.0000	0.0000	0.0000
3	0.0000	0.0000	0.0000	0.0000	0.0000
Time difference	0.0000	0.0000	0.0000	0.0000	0.0000

Macro benchmarks organized in tables

CLI output excerpt from execution of “BenchmarkExamples.cpp” in “QLever SPARQL engine”, adjusted for readability [1]

```
Table 'Adding exponents'
    metadata: {"manually-set-fields":"Row 2"}
```

	2^10	2^11
2^10	0.0000	0.0000
2^11	0.0000	0.0000
Values written out	1024+1024 and 1024+2048	1024+2048 and 2048+2048

Freestanding macro benchmarks and tables organized into groups

CLI output excerpt from execution of “BenchmarkExamples.cpp” in “QLever SPARQL engine”, adjusted for readability [1]

```
Group 'loopAdd'
    metadata: {"operator":"+"}
    Measurements:
        Single measurement '42+69'
            time: 0s

    Tables:
        Table 'Addition'
```

+	42	24
42	0.0000	0.0000
24	0.0000	0.0000

Runtime configuration options

Excerpt from generated runtime configuration option documentation of “JoinAlgorithmBenchmark.cpp” in “QLever SPARQL engine”, adjusted for readability [1]

```
Option 'max-memory' [string]
Value: "0B"
Description: Max amount of memory that any 'IdTable'
is allowed to take up. '0' for unlimited memory.
When set to anything else than '0', configuration
option 'max-bigger-table-rows' is ignored.
Example: 4kB, 8MB, 24B, etc. ...
```

Runtime configuration options

Excerpt from generated runtime configuration option documentation of “JoinAlgorithmBenchmark.cpp” in “QLever SPARQL engine”, adjusted for readability [1]

```
Option 'max-time-single-measurement' [float]
Value: 0.000000
Description: The maximal amount of time, in seconds,
any function measurement is allowed to take. '0'
for unlimited time. Note: This can only be checked,
after a measurement was taken.
Required invariants:
    - 'max-time-single-measurement' must be bigger
      than, or equal to, 0.
```

Macro-benchmarking library output

- Display
- “JSON” file [3]

Questions?

Evaluation of my macro-benchmarking library

Example macro benchmark comparison:

- Merge and galloping join algorithm implementation
- Hash join algorithm implementation

Macro benchmark suites executions

- Five different random number generator seeds for the random generation of input table entries
- For every seed 20 executions
- Why repeat seed? Explained later

Sample specifications - Computer hardware

Computer “Ural” from the institute for computer science at the University of Freiburg

- AMD Ryzen 7 3700X 8-Core Processor
- RAM 130 GiB

Macro benchmark suites measurements evaluation focus

- Row ratio
- Sorting configuration
- Hash join speedup

Row ratio

$$\frac{\text{Number rows bigger input table}}{\text{Number rows smaller input table}}$$

Sorting configuration

Sorting status of the smaller and bigger input table before merge and galloping join algorithm implementation

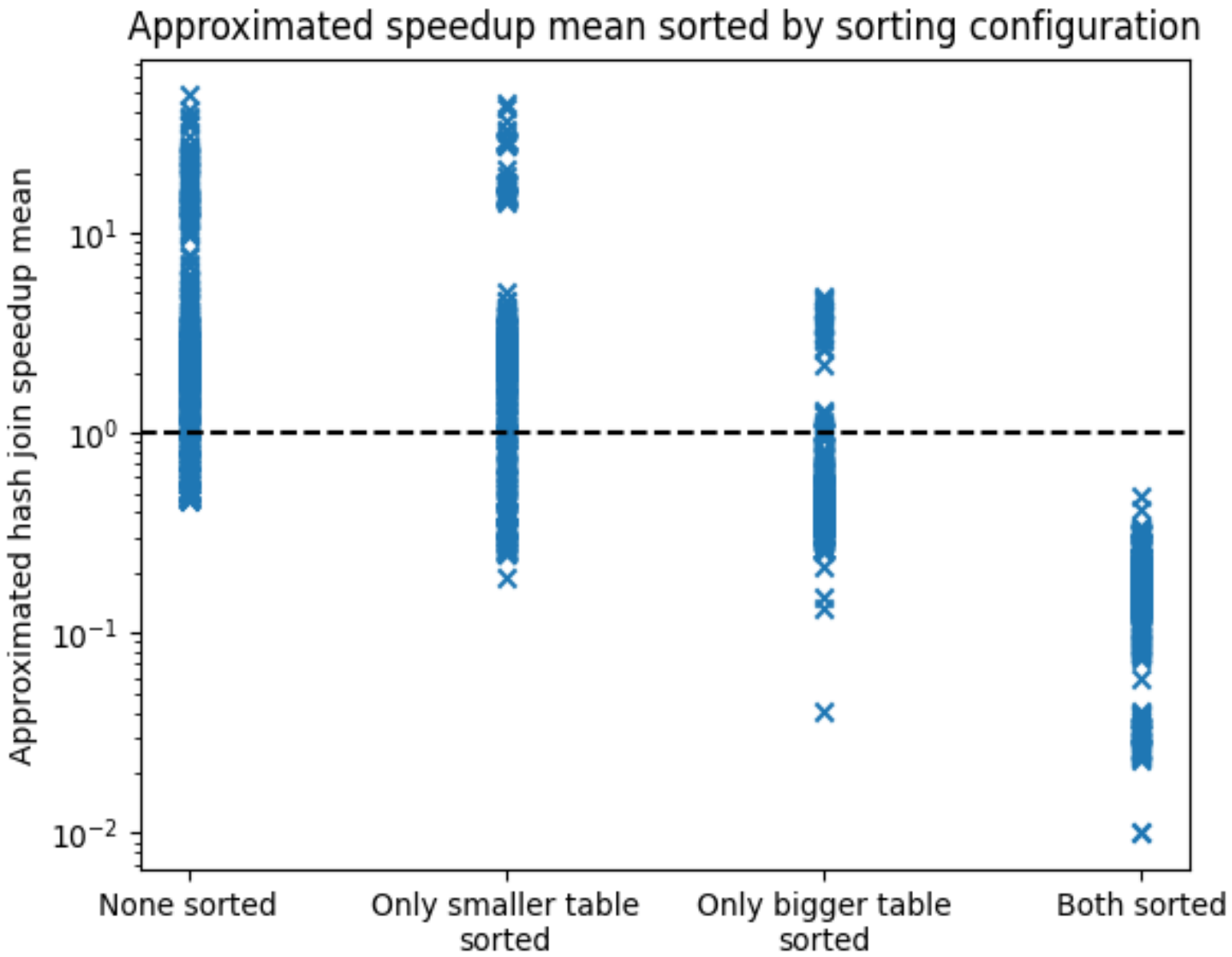
Hash join speedup

$$\frac{\text{Execution time merge and galloping join} + \text{Execution time sorting tables}}{\text{Execution time hash join}}$$

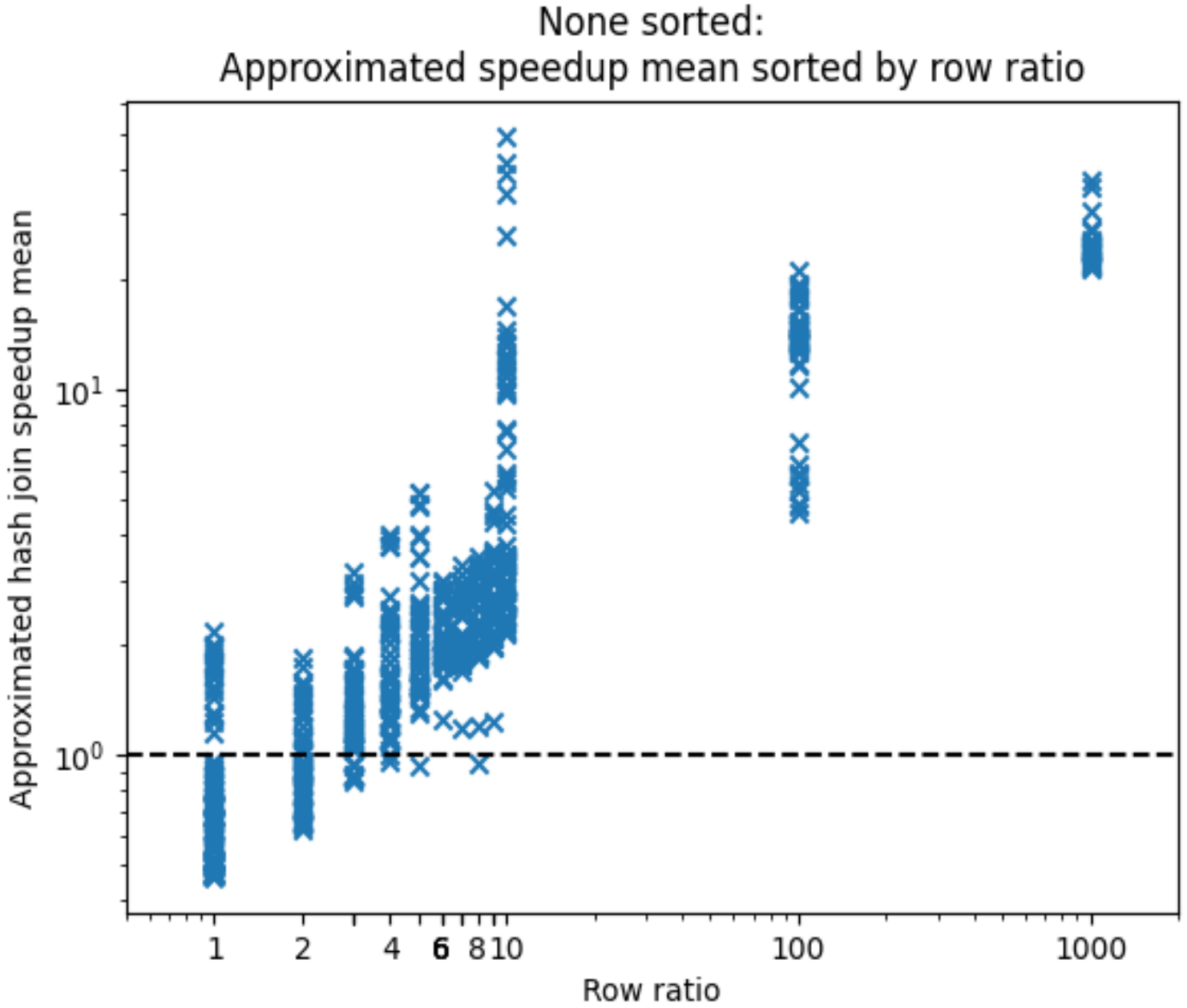
Hash join speedup

- Multiple executions of same macro benchmarks with same random number generator seed
- Multiple results of the same hash join speedup
- Approximate hash join speedup mean

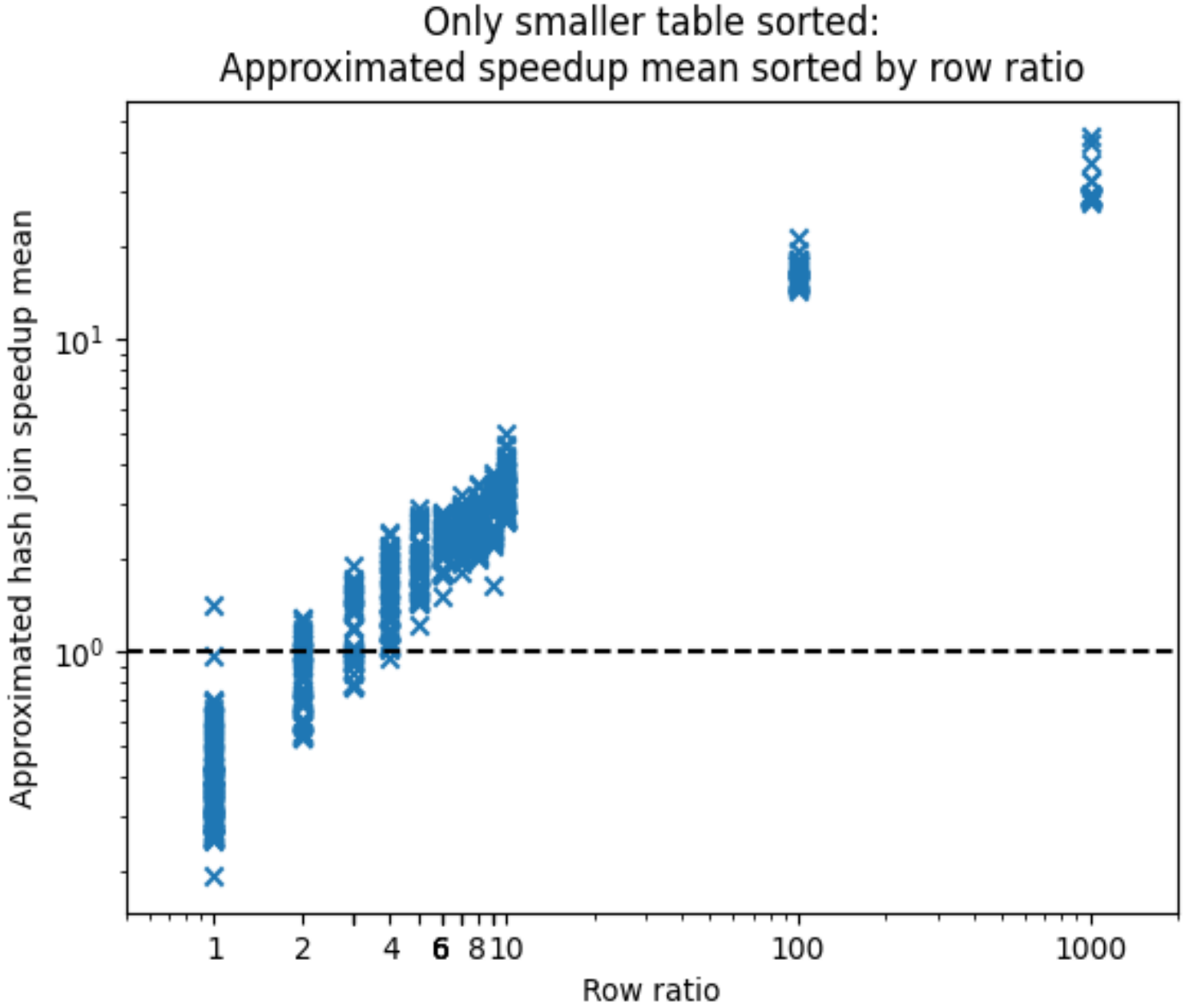
Macro benchmark suites measurements evaluations



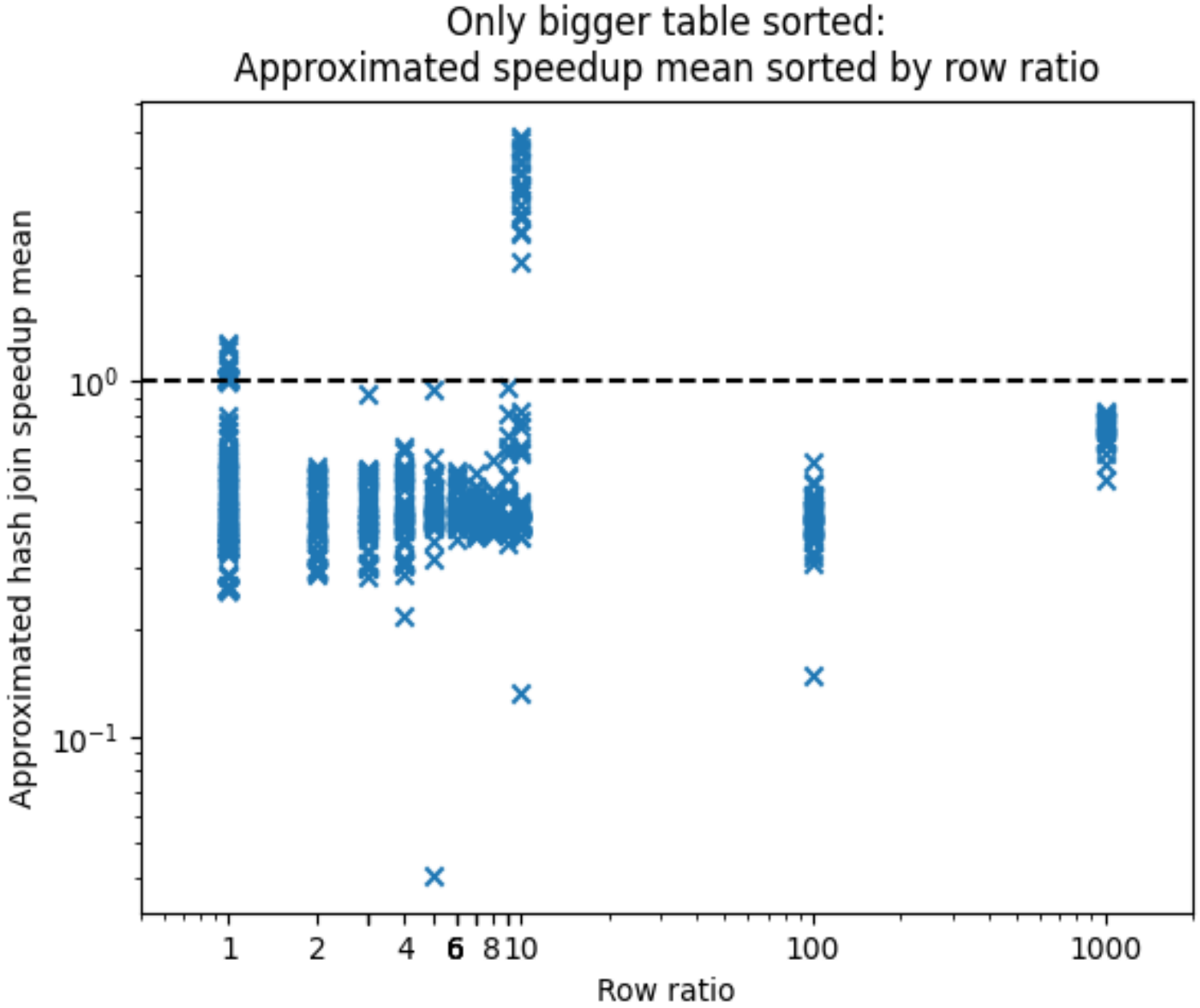
Macro benchmark suites measurements evaluations



Macro benchmark suites measurements evaluations



Macro benchmark suites measurements evaluations



Macro benchmark suites evaluation

Hash join algorithm implementation faster than/equal to merge and galloping join algorithm implementation, when:

- Bigger input table not sorted by join column
- Minimum row ratio 4

How did my macro-benchmarking library help with building those macro benchmark suites and evaluating their measurements?

Macro benchmark suites broad structure

- Macro benchmark suite for every generalized input table situation
  - Example: Smaller input table number rows grows and row ratio constant
- Tables of macro benchmarks for more specific input table situations
  - Example: Smaller input table number rows grows and row ratio always 10

Macro benchmark suites broad structure

- Every table of macro benchmarks repeated with every sorting configuration
- Every table of macro benchmarks same structure

Macro benchmark suites broad structure

CLI output excerpt from execution of "JoinAlgorithmBenchmark.cpp" in "QLever SPARQL engine", adjusted for readability [1]

Rows in smaller table	Time for sorting	Merge and galloping join	Sorting + merge and galloping join	Hash join	Rows in result table	Hash join speedup
10000	0.1493	0.0024	0.1517	0.0142	2962	10.6522
100000	1.6001	0.0240	1.6241	0.1301	29229	12.4864

Macro benchmark suites broad structure

CLI output excerpt from execution of “JoinAlgorithmBenchmark.cpp” in “QLever SPARQL engine”, adjusted for readability [1]

```
metadata: {
  "smaller-table-sorted": false,
  "bigger-table-sorted": false,
  "ratio-rows": 100.0
}
```

Macro benchmark suites broad structure

CLI output excerpt from execution of "JoinAlgorithmBenchmark.cpp" in "QLever SPARQL engine", adjusted for readability [1]

```
General metadata: {
  "value-changing-with-every-row":
    "smaller-table-num-rows",
  "overlap-chance": 50.0,
  "random-seed": 1235993526,
  "smaller-table-num-columns": 20,
  "bigger-table-num-columns": 20,
  "max-time-single-measurement": "infinite",
  "max-memory": 20000000000
}
```

Macro benchmark suites broad structure

Advantages over other libraries and frameworks:

+ Measurements better organized

Macro benchmark suites broad structure

Advantages over other libraries and frameworks:

- \+ Measurements better organized
- \+ Easier identification of behavior over multiple rows

Macro benchmark suites broad structure

Advantages over other libraries and frameworks:

- \+ Measurements better organized
- \+ Easier identification of behavior over multiple rows
- \+ Additional data for easier processing and easier evaluations

Noise

Noise: Time between measured and actual execution time

Noise

- Algorithms for minimizing noise influence on evaluation require repeated macro benchmark execution
- Macro-benchmarking library designed for single execution
- Multiple execution with same random generator seed

Runtime configuration options

Advantages over other libraries and frameworks:

+ Easier adjustment parameter

Runtime configuration options

Advantages over other libraries and frameworks:

+ Easier adjustment parameter
+ Easier parameter experimentation

Runtime configuration options

Advantages over other libraries and frameworks:

- + Easier adjustment parameter
- + Easier parameter experimentation
- + Reminder through generated runtime configuration option documentation

Questions?

Bibliography I

[1] Chair of Algorithms and Data Structures of the institute for computer science at the University of Freiburg, *QLever SPARQL engine*, version commit 57ba939b32ad30f3c4802f2e6050aedc1b13b2d9, Jun. 7, 2024. [Online]. Available: https://github.com/ad-freiburg/qlever (visited on 06/26/2024).

[2] "BENCHMARK | English meaning - Cambridge Dictionary," (), [Online]. Available: https://dictionary.cambridge.org/dictionary/english/benchmark (visited on 06/16/2024).

[3] Wikipedia contributors, "JSON — Wikipedia, The Free Encyclopedia," (Jun. 14, 2024), [Online]. Available: https://en.wikipedia.org/w/index.php?title=JSON&oldid=1229014619 (visited on 06/20/2024).

Bibliography II

[4] S. Chaudhary, "Why 1.5 Is Used in the IQR Rule for Outlier Detection," (Jan. 24, 2024), [Online]. Available: https://builtin.com/articles/1-5-iqr-rule (visited on 06/15/2024).

[5] R. Bevans, "Understanding Confidence Intervals | Easy Examples & Formulas," (Aug. 7, 2020), [Online]. Available: https://www.scribbr.com/statistics/confidence-interval/ (visited on 06/12/2024).

[6] *Picobench*, version 2.07, Mar. 6, 2024. [Online]. Available: https://github.com/iboB/picobench (visited on 05/31/2024).

[7] *Nanobench*, version 4.3.11, Feb. 16, 2023. [Online]. Available: https://nanobench.ankerl.com/ (visited on 05/31/2024).

[8] *Catch2*, version 3.6.0, May 5, 2024. [Online]. Available: https://github.com/catchorg/Catch2 (visited on 05/31/2024).

Bibliography III

[9] *Sltbench*, version r-2.4.0, Aug. 23, 2020. [Online]. Available: https://github.com/ivafanas/sltbench?tab=readme-ov-file (visited on 05/31/2024).

[10] *Folly benchmark component*, version v2024.05.06.00, May 6, 2024. [Online]. Available: https://github.com/facebook/folly (visited on 05/31/2024).

[11] *CppBenchmark*, version 1.0.4.0, Sep. 27, 2023. [Online]. Available: https://github.com/chronoxor/CppBenchmark (visited on 05/31/2024).

[12] *Hyperfine*, version v.1.18.0, Oct. 5, 2023. [Online]. Available: https://github.com/sharkdp/hyperfine (visited on 05/31/2024).

[13] *Celero*, version v2.8.5, Dec. 26, 2022. [Online]. Available: https://github.com/DigitalInBlue/Celero (visited on 05/31/2024).

Bibliography IV

[14] *Google benchmark*, version v1.8.3, Aug. 31, 2023. [Online]. Available: https://github.com/google/benchmark (visited on 05/31/2024).

Appendix

Merge and galloping join algorithm

- Chooses between:
  - Merge join algorithm
  - Galloping join algorithm

Simplified merge join algorithm

- $\frac{\text{Number rows bigger input table}}{\text{Number rows smaller input table}} < 1000$
- Iterate left input table join column
- Linear search right input table join column

Simplified galloping join algorithm

- $\frac{\text{Number rows bigger input table}}{\text{Number rows smaller input table}} \geq 1000$
- Iterate smaller input table join column
- Binary search bigger input table join column

Setting runtime configuration options

Two ways:

- “JSON” file [3]
- Shorthand configuration string

Setting runtime configuration options - “JSON” file [3]

CLI output excerpt from execution of “BenchmarkExamples.cpp” in “QLever SPARQL engine”, adjusted for readability [1]

```
{
  "date": "22.3.2023",
  "num-signs": 10 000,
  "coin-flip-try": [
    false,
    false
  ],
  "accounts": {
    "personal": {
      "steve": -41.900001525878906
    }
  }
}
```

Setting runtime configuration options - Shorthand configuration string

JSON like syntax for use with CLI.

Setting runtime configuration options - Shorthand configuration string

Simplified definition for the shorthand configuration language from “ConfigShorthand.g4” in “QLever SPARQL engine” [1].

```
shortHandString : assignments EOF;
assignments : (listOfAssignments+=assignment ',')*
    listOfAssignments+=assignment;
assignment : NAME ':' content;
object : '{' assignments '}';
list : '['(listElement+=content',')*
    listElement+=content']';
content : LITERAL|list|object;
```

Setting runtime configuration options - Shorthand configuration string

Simplified definition for the shorthand configuration language from “ConfigShorthand.g4” in “QLever SPARQL engine” [1].

```
// The literals.
LITERAL : BOOL | INTEGER | FLOAT | STRING;
BOOL : 'true' | 'false';
INTEGER : '-'?[0-9]+;
FLOAT : INTEGER'.'[0-9]+;
STRING : '"' .*? '"';

NAME : [a-zA-Z0-9\-_]+;
```

Setting runtime configuration options - Shorthand configuration string

Example

```
date : "2.8.2024", accounts : { steve : 20 }
```

Macro benchmark suites structure - Table entries generation

Given parameter:

- Number of rows
- Number of columns
- Which column is join column
- Join column sample size ratio
- Probability for overlap
- Seed for random number generator

Macro benchmark suites structure - Table entries generation

Simplified algorithm overview:

1. Create two randomly filled tables with disjoint join columns elements
2. Create join column overlap

Macro benchmark suites structure - Table entries generation

Single table entries generation:

1. Fill all columns except join column with random numbers.
2. Define random set as a set of numbers with size “Number of rows” · “Sample size ratio”.
3. Fill join column randomly with elements from the random set using uniform distribution.

Macro benchmark suites structure - Table entries generation

Overlap creation:

1. Go through all elements in set of smaller table join column elements.
2. Every element has same given overlap probability for overlap.
3. If no overlap event, nothing happens.

Macro benchmark suites structure - Table entries generation

If overlap event:

1. Choose random elements in set of bigger table join column elements with uniform distribution.
2. Replace all occurrences of smaller table join column element with bigger table join column element.

Macro benchmark suites executions - Random generator seed

Macro benchmark suit execution	Seed
1 to 20	146081905
21 to 40	193901340
41 to 60	288613237
61 to 80	155923003
81 to 100	4648133

Noise

Causes:

- Computer switches between process
- Benchmark overhead
- Hardware
- Etc.

Noise

- Micro benchmarks noise range: Milliseconds (0.001 seconds)

Noise

- Micro benchmarks noise range: Milliseconds (0.001 seconds)
- Theory: Macro benchmark noise ignorable

Noise

- Micro benchmarks noise range: Milliseconds (0.001 seconds)
- Theory: Macro benchmark noise ignorable
- Reality: Macro benchmark noise can be bigger than one second

Noise

Macro benchmark measurement problems:

1. More noise than measurement $\Rightarrow$ Useless
2. Noticeable difference in repeated macro benchmark measurement $\Rightarrow$ Need true mean

Noise

Solution: Algorithms for approximating the true mean and identifying measurement with more noise than measurement.

Filtering measurements

- Problem: Measurements with more noise than measurement
- Self-made algorithm for filtering

Reminder: Macro benchmark suites broad structure

Rows in smaller table	Time for sorting	Merge and gallop-ing join	Sort-ing + merge and gallop-ing join	Hash join	Rows in re-sult table	Hash join speedup
10000	0.1493	0.0024	0.1517	0.0142	2962	10.6522
100000	1.6001	0.0240	1.6241	0.1301	29229	12.4864

Table from execution of “JoinAlgorithmBenchmark.cpp” in “QLever SPARQL engine”, adjusted for readability [1].

Filtering measurements - My algorithm

For every list of all executions of same table row with same seed:

1. Max noise $=$ Longest measurement $-$ Shortest measurement
2. Calculate max noise for every macro benchmark in the row
3. If all measurements of single execution smaller than their max noise, delete execution

“1.5 IQR rule” [4]

- List of numbers
- Defines outliers based on number distribution

Filtering measurements - My algorithm

4. Identify hash join speedup outlier via “1.5 IQR rule” [4]
5. If single execution has hash join speedup outlier, delete execution

Mean problem

Normal mean not fit for macro benchmarks

Mean problem

Example macro benchmark *A*

- Minimum measured execution time *A*: 1 second
- Maximum measured execution time *A*: 2 seconds
- Measured execution time *A* true mean: 1.1 second

Mean problem

Example macro benchmark *A*

Repeated executions:

1. 1.1 second
2. 2 seconds
3. 1.5 seconds

Mean: ~ 1.53

My algorithm for true mean approximation

- Not a general solution
- After filtering of measurements
- Focus on hash join speedup true mean

“99% confidence interval” [5]

- Statistic formula
- Approximates lower/upper bound of true mean of list of numbers
- 99% chance correct

Reminder: Macro benchmark suites broad structure

Rows in smaller table	Time for sorting	Merge and gallop-ing join	Sort-ing + merge and gallop-ing join	Hash join	Rows in re-sult table	Hash join speedup
10000	0.1493	0.0024	0.1517	0.0142	2962	10.6522
100000	1.6001	0.0240	1.6241	0.1301	29229	12.4864

Table from execution of “JoinAlgorithmBenchmark.cpp” in “QLever SPARQL engine”, adjusted for readability [1].

My algorithm for true mean approximation

For every filtered list of all executions of same table row with same seed:

1. If the list has less than two entries, no return value
2. Calculate hash join speedup "99% confidence interval" lower and upper bound [5]
3. If lower bound less than zero and upper bound bigger than one, no return value

My algorithm for true mean approximation

4. If lower bound less than zero, set lower bound to 0.01
5. Return lower and upper bound

Macro benchmark suites executions - Important runtime configuration options values

- Minimum number smaller input table rows: 10000
- Minimum number bigger input table rows: 100000
- Number columns smaller input table: 20

Macro benchmark suites executions - Important runtime configuration options values

- Number columns bigger input table: 20
- Overlap-chance: 50%
- Sample size ratio join column smaller input table: 1

Macro benchmark suites executions - Important runtime configuration options values

- Sample size ratio join column bigger input table: 1
- Max memory for single table: 20 GB
- Benchmarking sample size ratios join columns: $[0.1, 1, 10]$

List of compared micro-benchmarking libraries

- “Picobench” [6]
- “Nanobench” [7]
- “Catch2” [8]
- “Sltbench” [9]
- “Folly benchmark component” [10]
- “CppBenchmark” [11]
- “Hyperfine” [12]
- “Celero” [13]
- “Google benchmark” [14]