Benchmark Results Report

This report summarizes the benchmark analysis of VectorTree by comparing its performance with:

1. std::vector
2. a simple persistent vector

VectorTree is created to obtain a persistent version of std::vector for multithreaded usage. In other words, the target of VectorTree is to achieve the performance of std::vector while keeping the persistency to deal with the concurrency issues. Hence, the 1st container to compare with is std::vector. The 2nd one is a very simple definition for a persistent vector. The flowchart of any operation on the simple persistent vector is as follows:

1. create a copy of the original vector,
2. apply the request on the copy,
3. return the modified copy.

Four fundamental operations are inspected in this benchmark:

1. emplace_back
2. pop_back
3. pop_front
4. traversal

The 3rd operation is actually not a property of the vector data structure such that std::vector interface does not include it. It requires one-step move operation on all elements which is linear, O(N). Its the same, even more problematic, in case of a tree of vectors. VectorTree solves this problem using swap-and-pop idiom while loosing the order of elements. Hence, the pop_front graphs showing a better performance for VectorTree, actually shows the performance support coming from the swap-and-pop idiom. In other words, consider the graphs for the pop_front operation as the comparison between:

- pop_front: move<T> with O(N)
- swap-and-pop: front() = back(); pop_back(): copy<T> with O(2*BufferSize)

Summary of the test parameters:
  1. Inspected operations: emplace_back, pop_back, pop_front and traversal
  2. Inspected objects: Two types are inspected: type_small and type_large
  3. Original container size: BUFFER_SIZE_1 (32), BUFFER_SIZE_2 (1024) and BUFFER_SIZE_3 (32768)

The two object types are defined simply as follows:
  1. type_small: an object with one field of int
  2. type_large: an object with an array of 256 elements of int

The number of the three groups of test parameters are:
  1. # of the inspected operations: 4
  2. # of the inspected objects: 2
  3. # of the inspected original container sizes: 3

Hence, the combination of the test parameters yield:
  Test count = 4 * 2 * 3 = 24

Note that the tests are performed with a VectorTree of buffer size of 32.

The tests follow a simple algorithm:

1. Initialize a container with a predefined size, N
2. Aplly the operation iteratively N times on this original container
3. Measure timing

A templated wrapper class is created to simulate a uniform interface for the inspected four operations which helps to simplify the google benchmark macros. A base template (VectorTree) and two specializations (std::vector and persistent_vector) simulate the three containers to be compared.

DEFINE_BENCHMARK macro defines a shortcut to the google benchmark macros.

The tests are quite simple and the corresponding graphs are self-explanatory.
Hence, i will not go through the results in detail.
In summary:

- For containers of small size, although VectorTree is the worst one,
  it performs efficiently (600ms):
  See graphs with BUFFER_SIZE_1
- For containers of large size, VectorTree performs way better than
  the simple persistent vector:
  See graphs with BUFFER_SIZE_3
- VectorTree approaches to std::vector
  in case of the three fundamental operations:
  See graphs with BUFFER_SIZE_3 and especially with type_large
- The swap-and-pop idiom provides
  an efficient solution to the pop_front operation:
  See graphs with pop_front

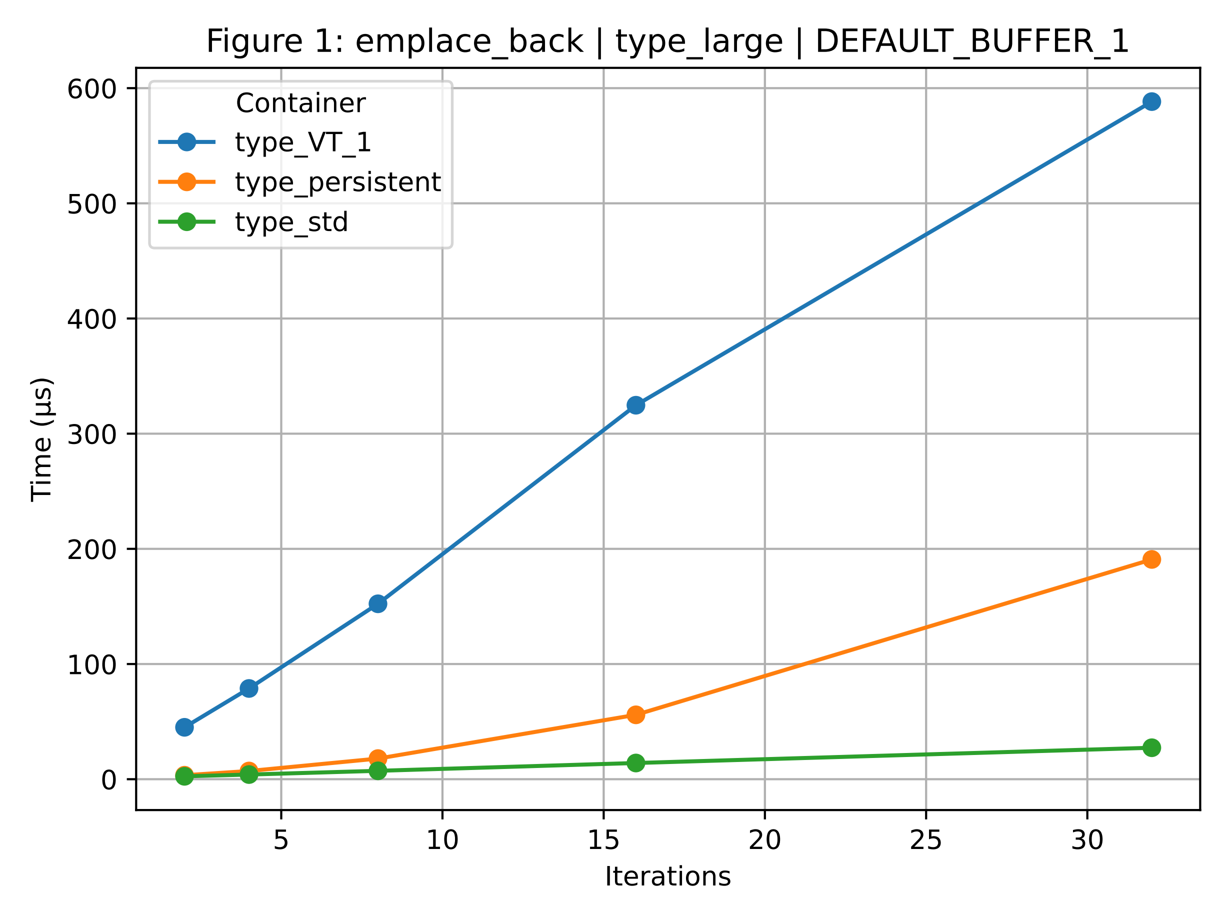


Figure 1: emplace_back | type_large | DEFAULT_BUFFER_1
Container
type_VT_1
type_persistent
type_std
Time (µs)
Iterations
0
100
200
300
400
500
600
5
10
15
20
25
30

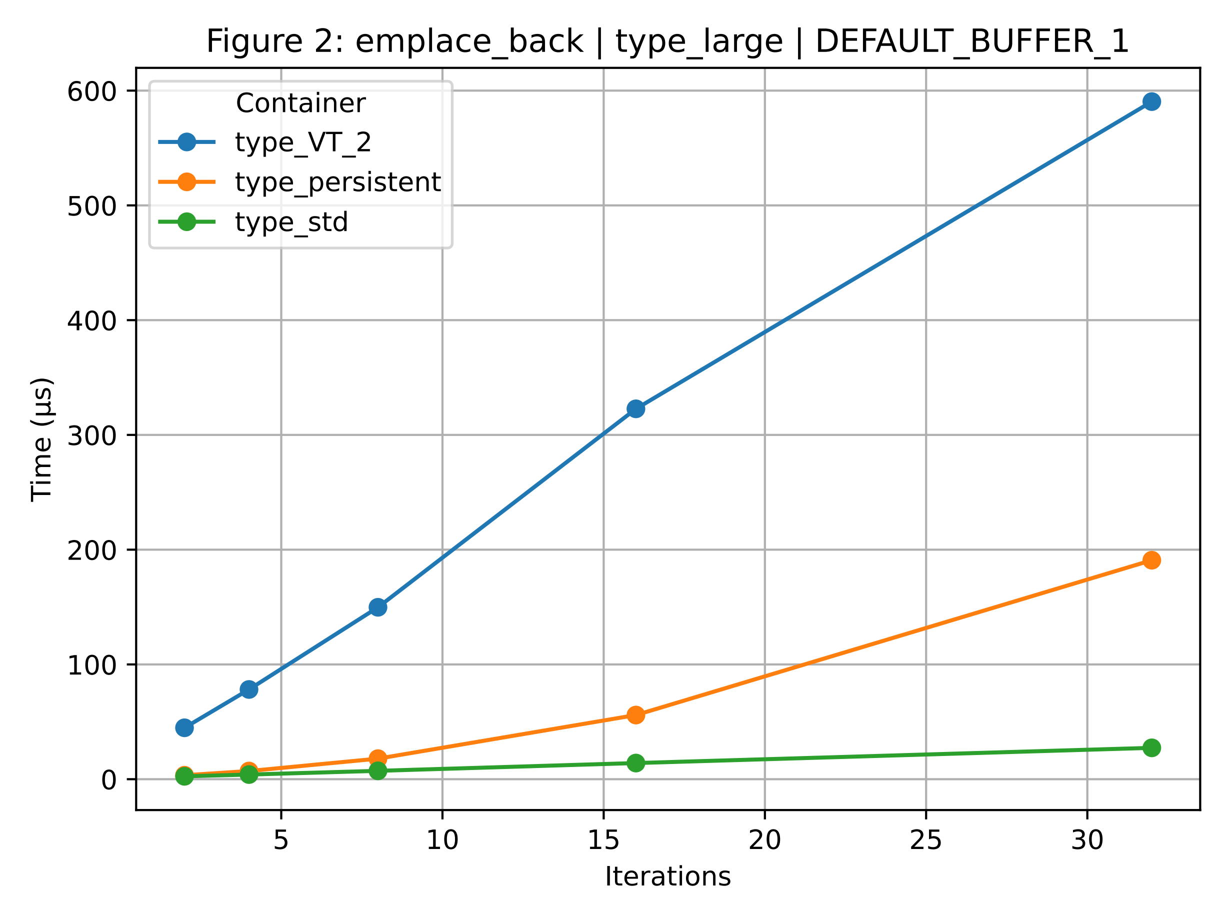


Figure 2: emplace_back | type_large | DEFAULT_BUFFER_1
Container
type_VT_2
type_persistent
type_std
Time (µs)
Iterations
0
100
200
300
400
500
600
5
10
15
20
25
30

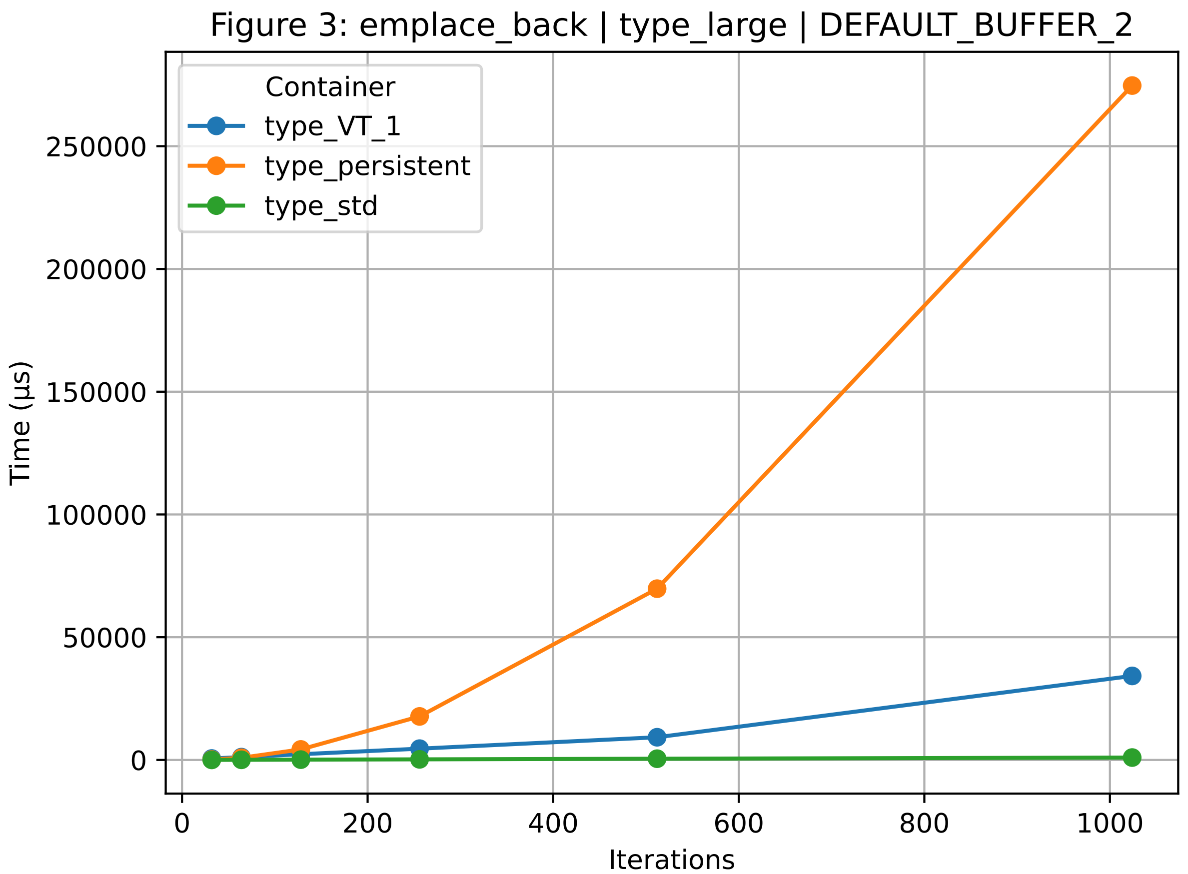


Figure 3: emplace_back | type_large | DEFAULT_BUFFER_2
Container
type_VT_1
type_persistent
type_std
250000
200000
150000
100000
50000
0
0
200
400
600
800
1000
Time (µs)
Iterations

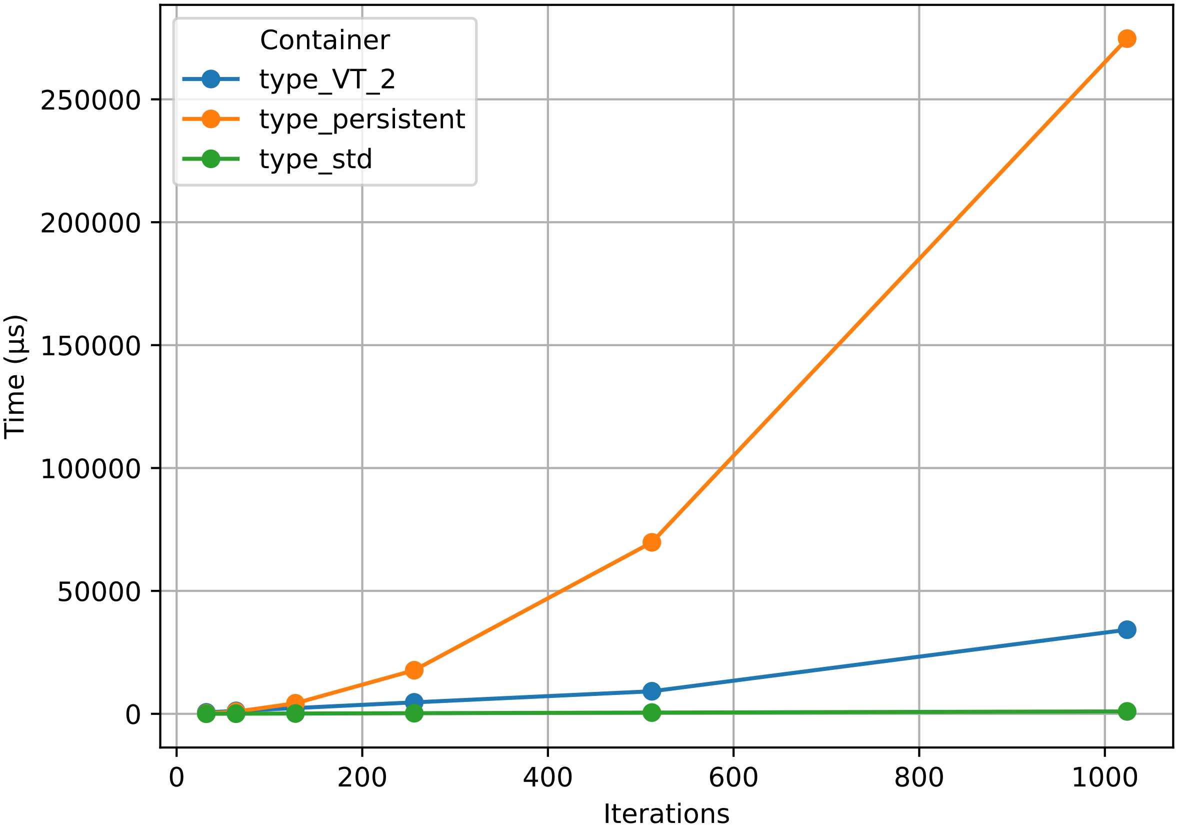


Figure 4: emplace_back | type_large | DEFAULT_BUFFER_2

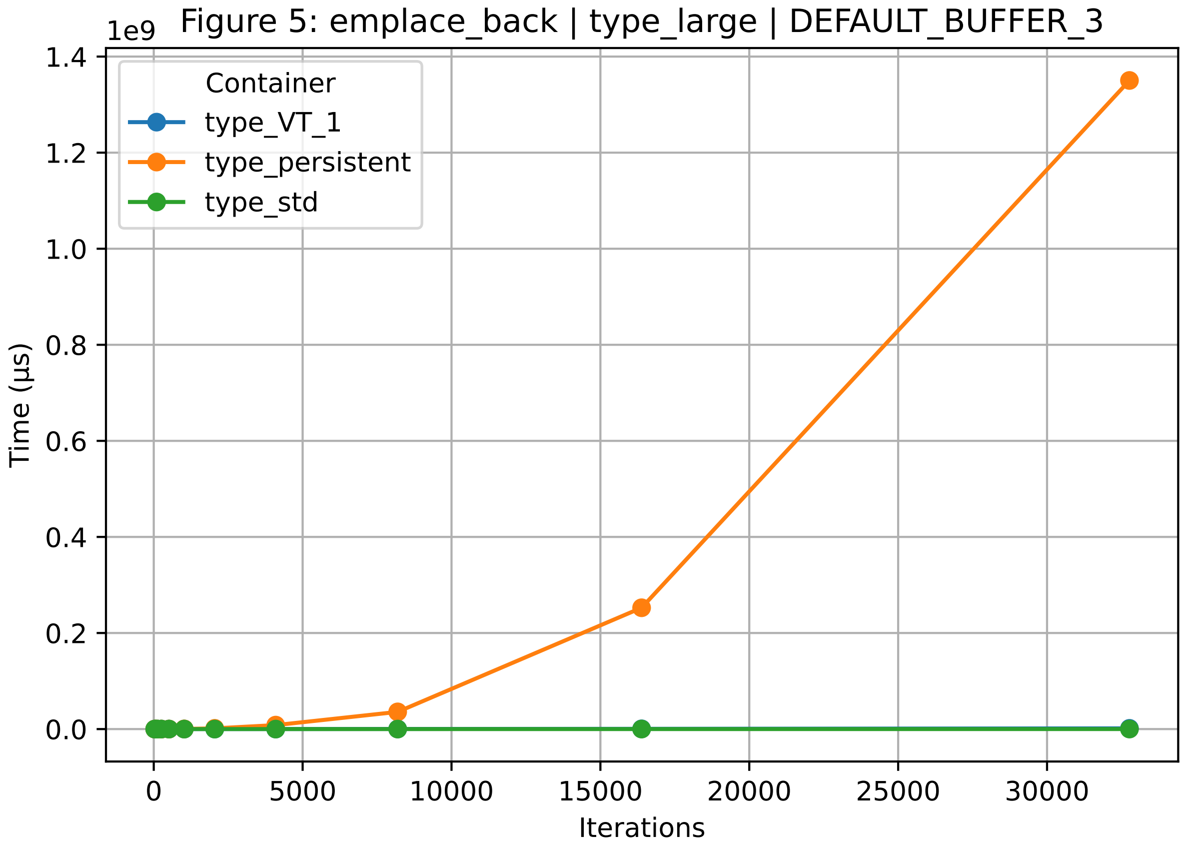


Figure 5: emplace_back | type_large | DEFAULT_BUFFER_3
1e9
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Time (µs)
0
5000
10000
15000
20000
25000
30000
Iterations
Container
type_VT_1
type_persistent
type_std

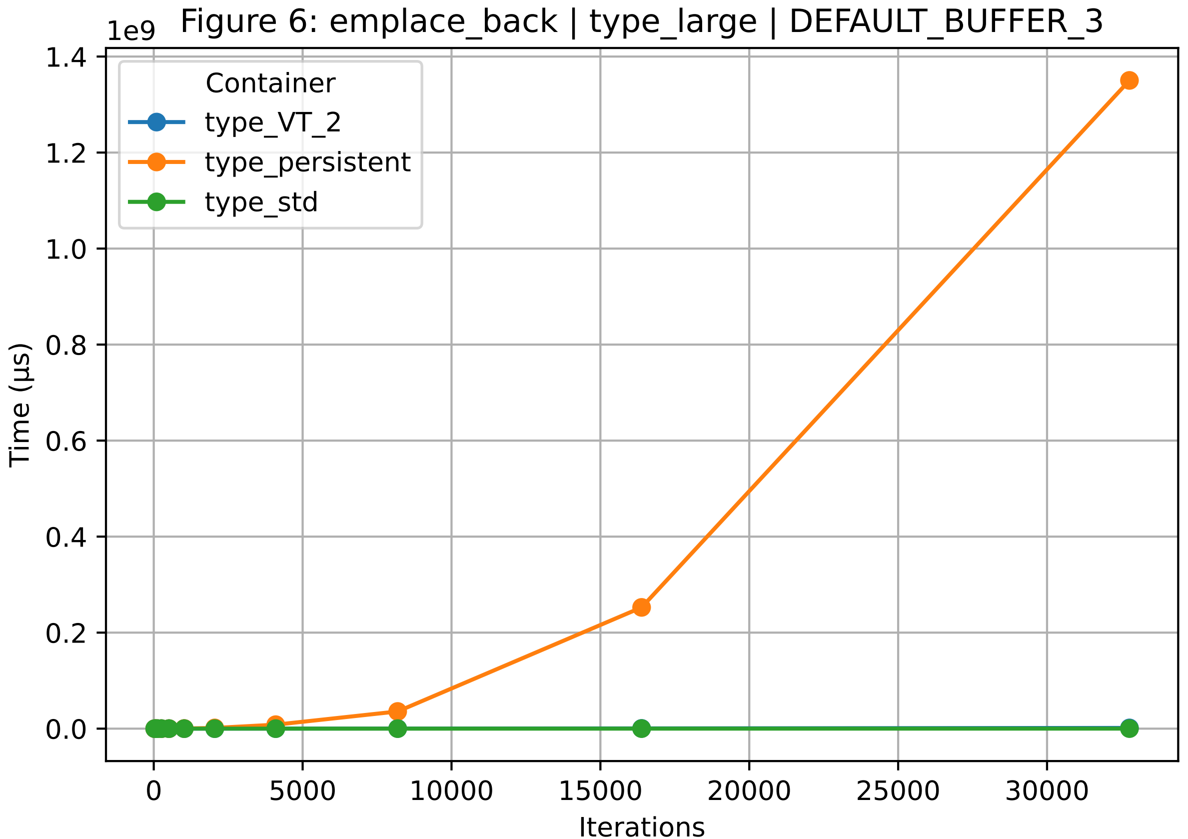


Figure 6: emplace_back | type_large | DEFAULT_BUFFER_3
1e9
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Time (µs)
0
5000
10000
15000
20000
25000
30000
Iterations
Container
type_VT_2
type_persistent
type_std

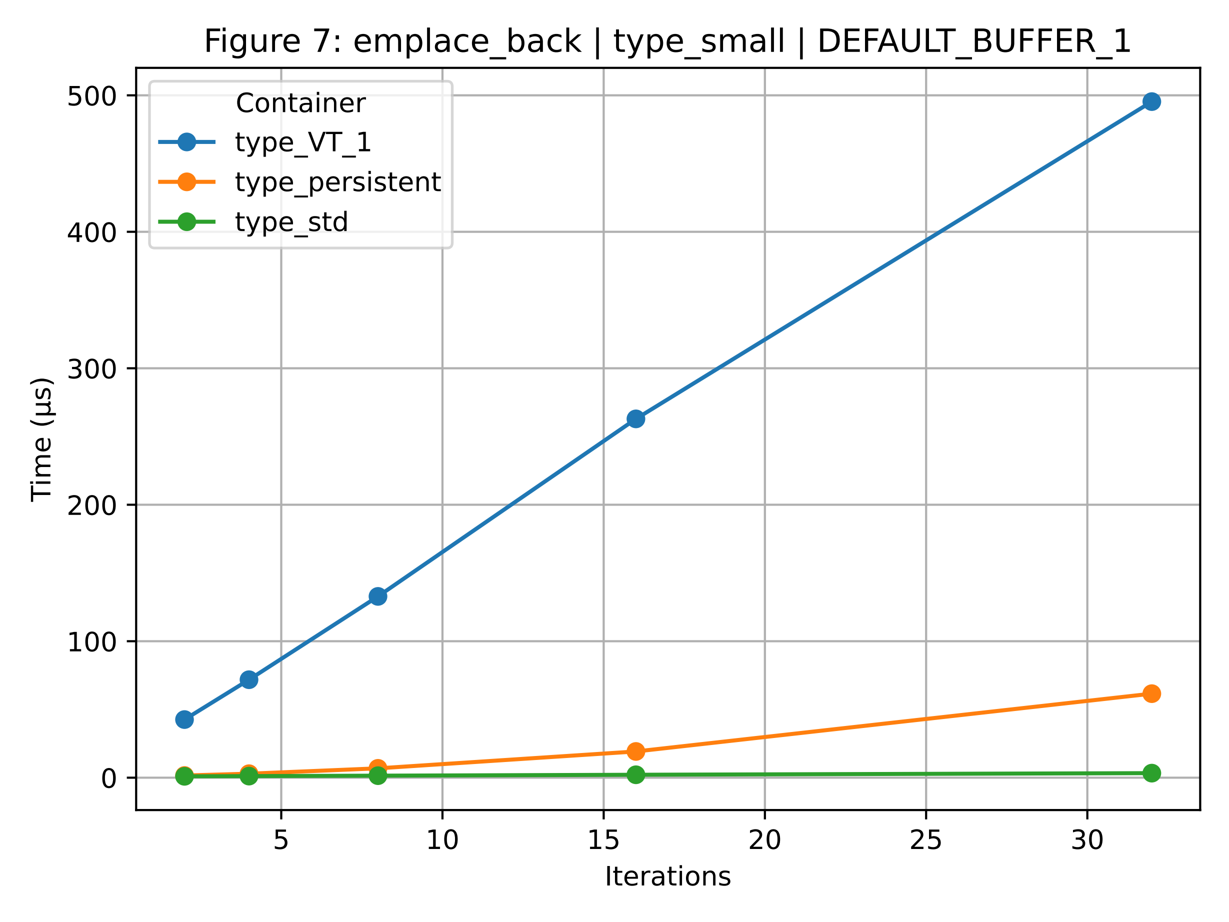


Figure 7: emplace_back | type_small | DEFAULT_BUFFER_1
Container
type_VT_1
type_persistent
type_std
Time (µs)
Iterations
0
100
200
300
400
500
5
10
15
20
25
30

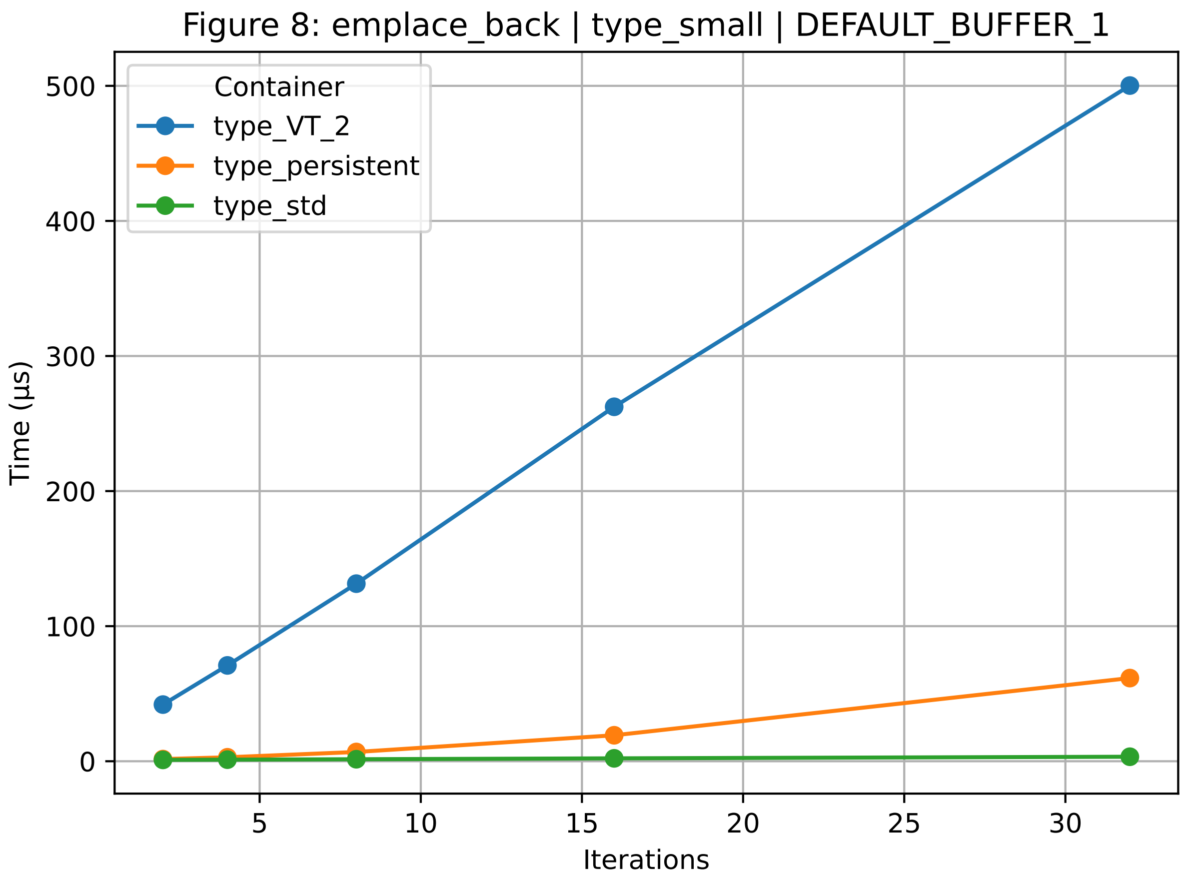


Figure 8: emplace_back | type_small | DEFAULT_BUFFER_1
Container
type_VT_2
type_persistent
type_std
Time (µs)
Iterations
0
100
200
300
400
500
5
10
15
20
25
30

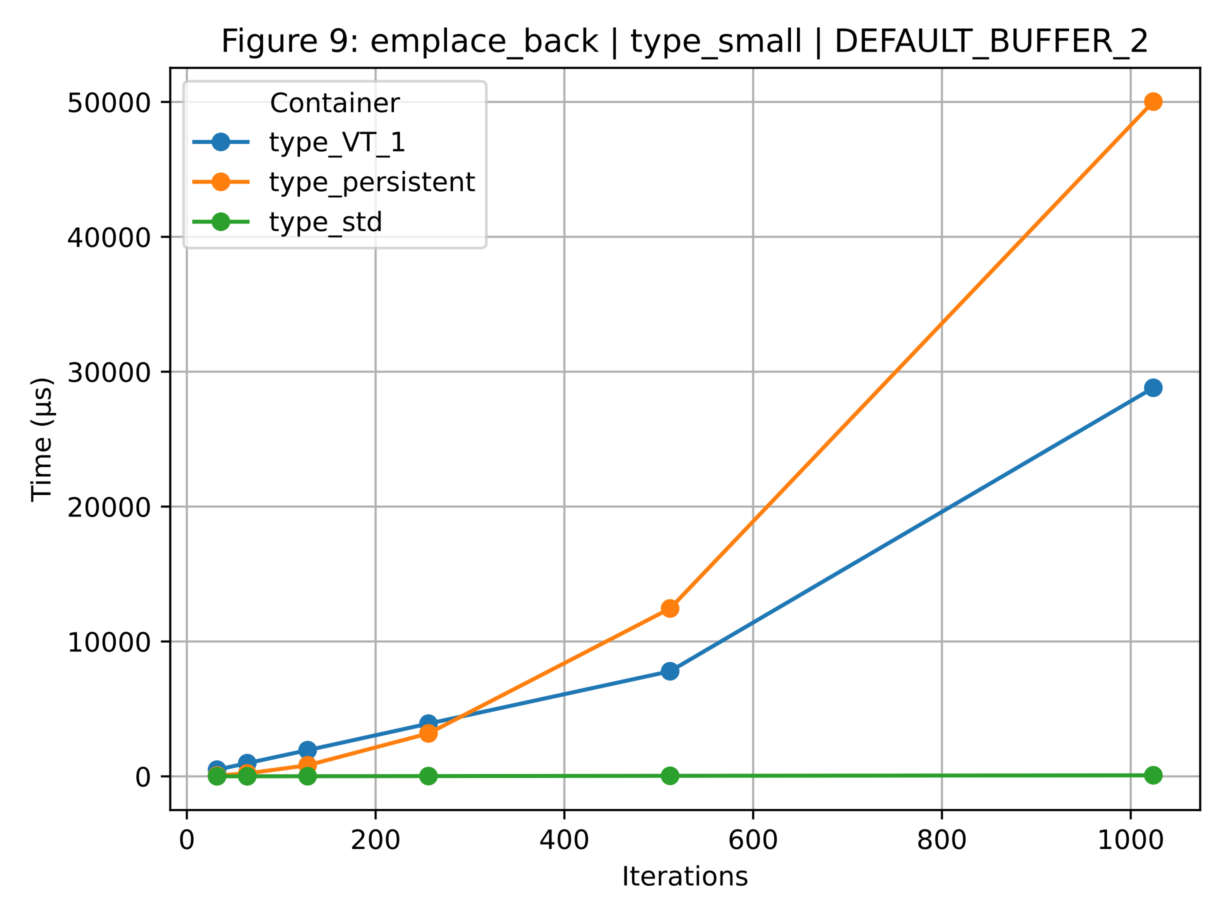
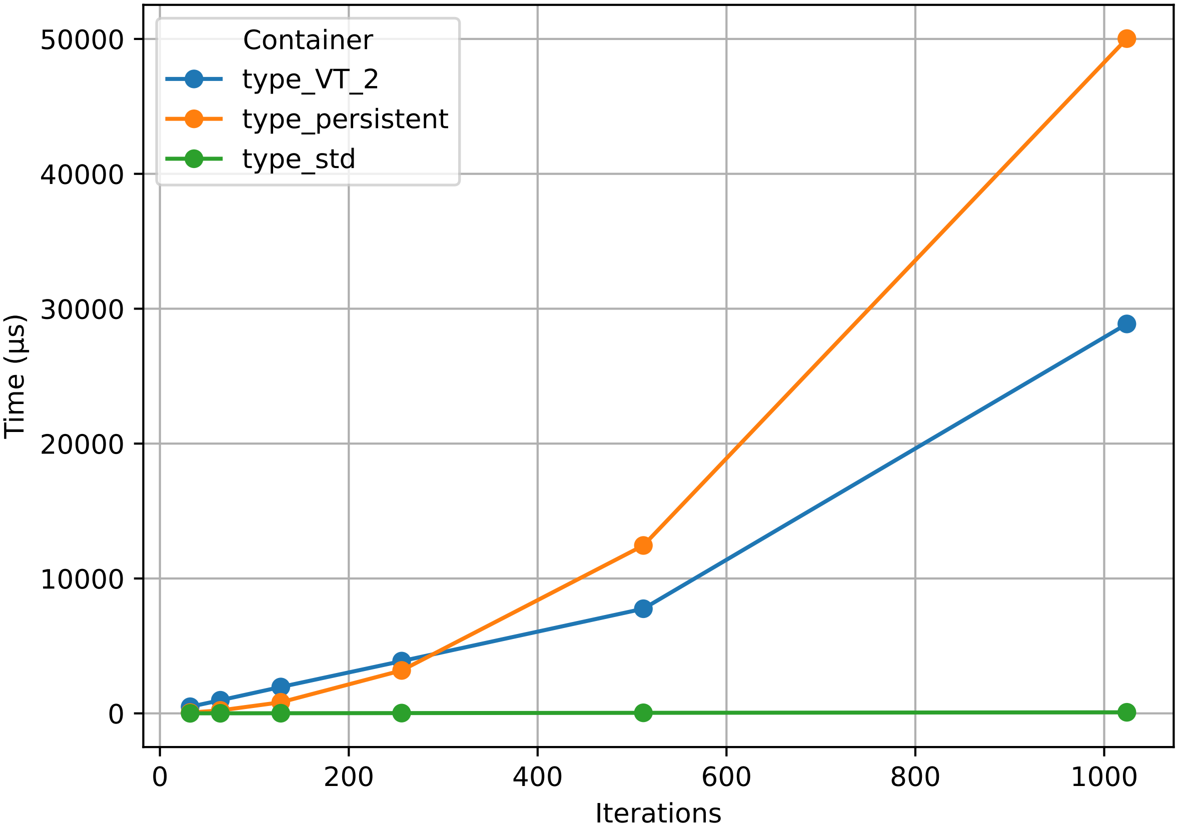


Figure 9: emplace_back | type_small | DEFAULT_BUFFER_2
Container
type_VT_1
type_persistent
type_std
Time (µs)
Iterations
0
10000
20000
30000
40000
50000
0
200
400
600
800
1000

Figure 10: emplace_back | type_small | DEFAULT_BUFFER_2



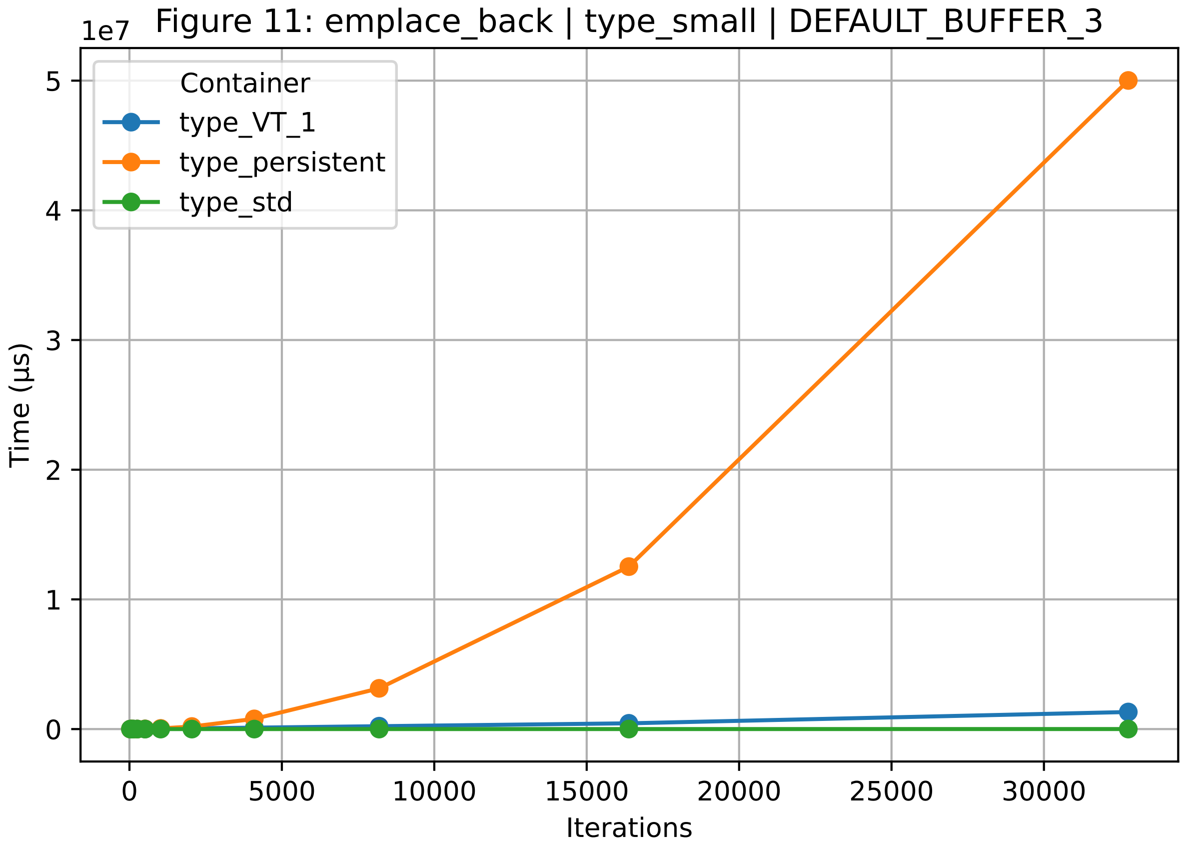


Figure 11: emplace_back | type_small | DEFAULT_BUFFER_3
1e7
Container
type_VT_1
type_persistent
type_std
Time (µs)
Iterations
0
1
2
3
4
5
0
5000
10000
15000
20000
25000
30000

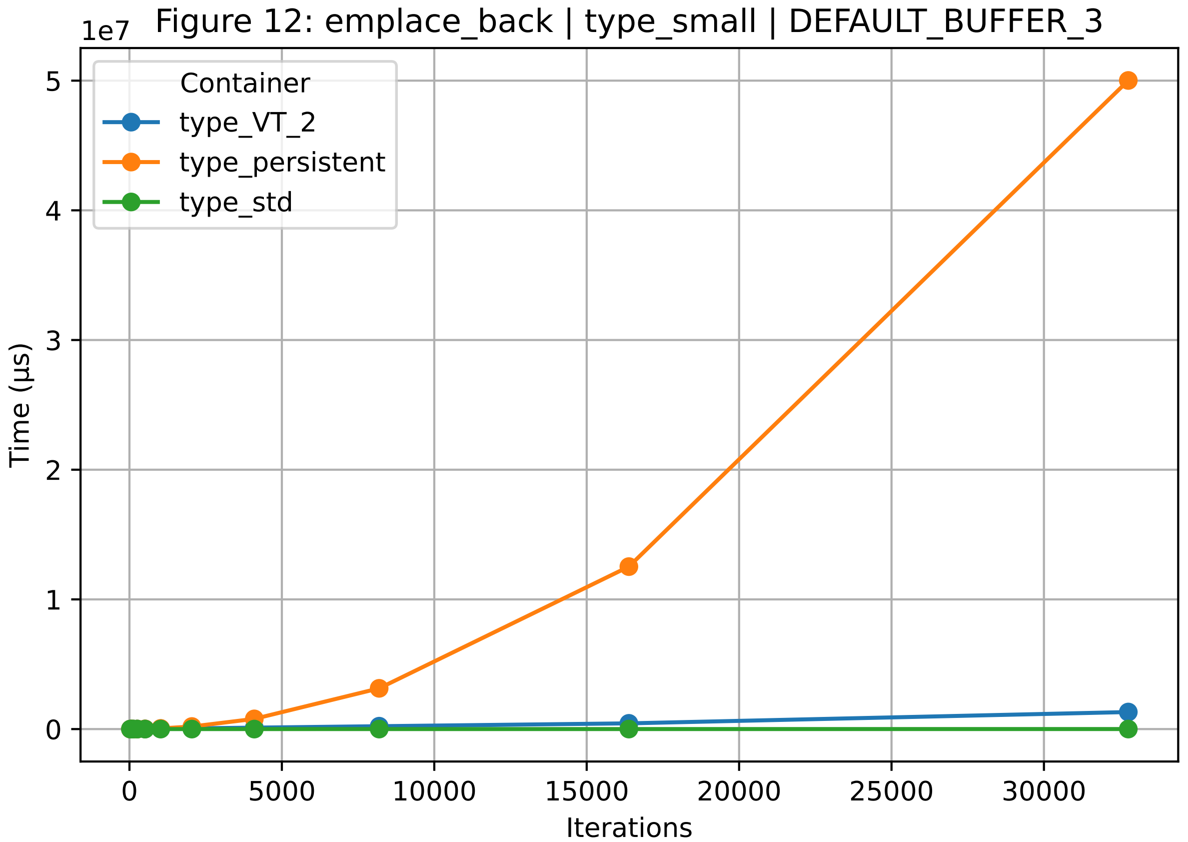


Figure 12: emplace_back | type_small | DEFAULT_BUFFER_3
1e7
Container
type_VT_2
type_persistent
type_std
Time (µs)
Iterations
0
1
2
3
4
5
0
5000
10000
15000
20000
25000
30000

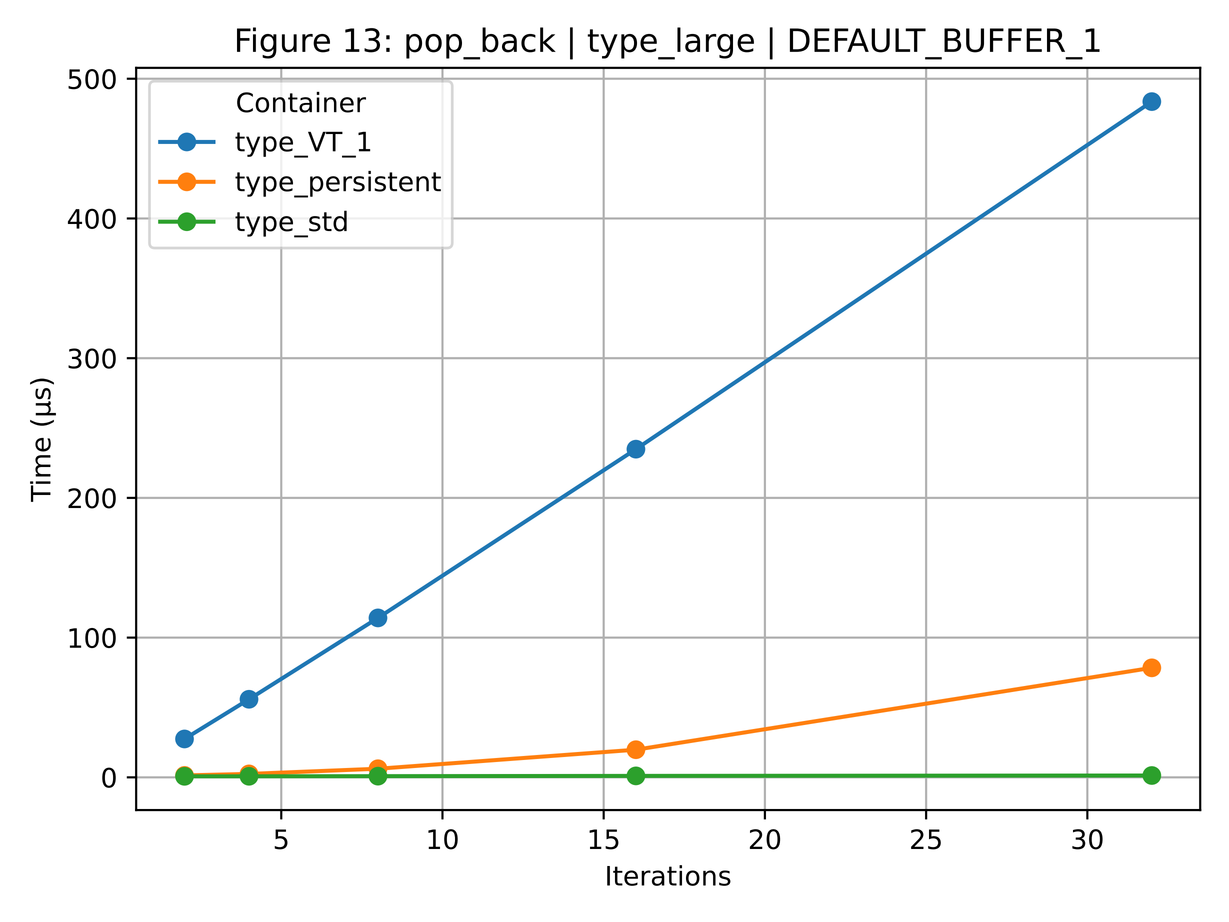


Figure 13: pop_back | type_large | DEFAULT_BUFFER_1
Container
type_VT_1
type_persistent
type_std
Time (µs)
500
400
300
200
100
0
5
10
15
20
25
30
Iterations

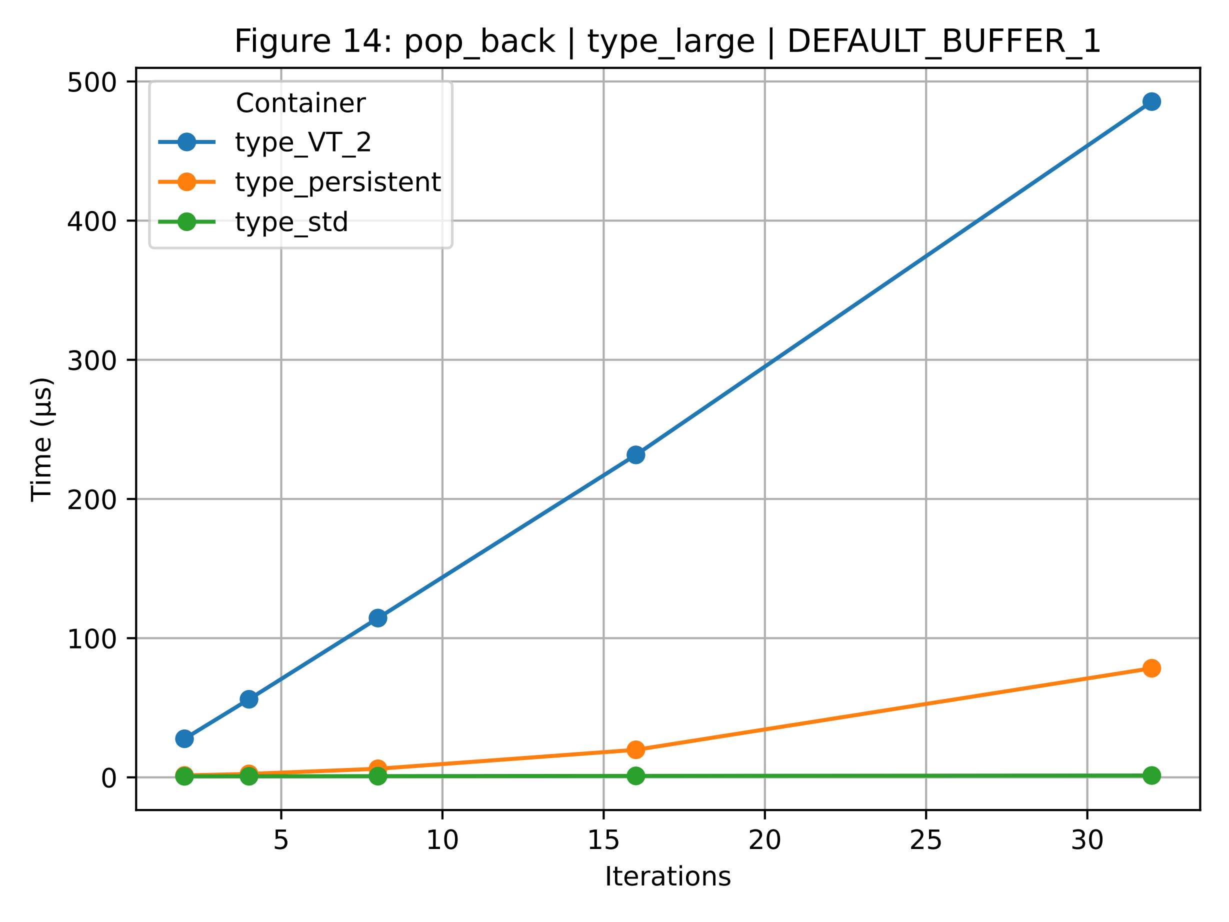


Figure 14: pop_back | type_large | DEFAULT_BUFFER_1
Container
type_VT_2
type_persistent
type_std
Time (µs)
Iterations
0
100
200
300
400
500
5
10
15
20
25
30

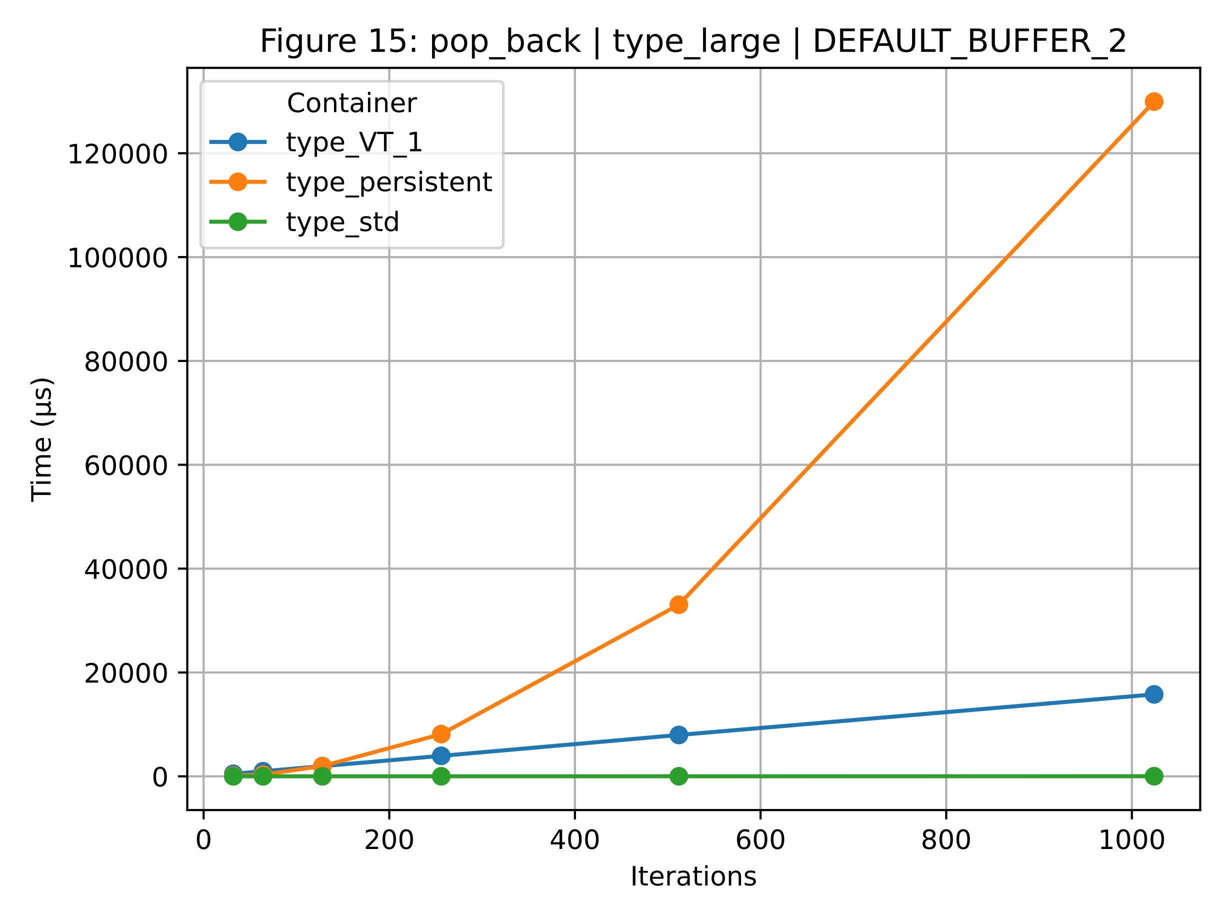


Figure 15: pop_back | type_large | DEFAULT_BUFFER_2
Container
type_VT_1
type_persistent
type_std
Time (µs)
Iterations
0
20000
40000
60000
80000
100000
120000
0
200
400
600
800
1000

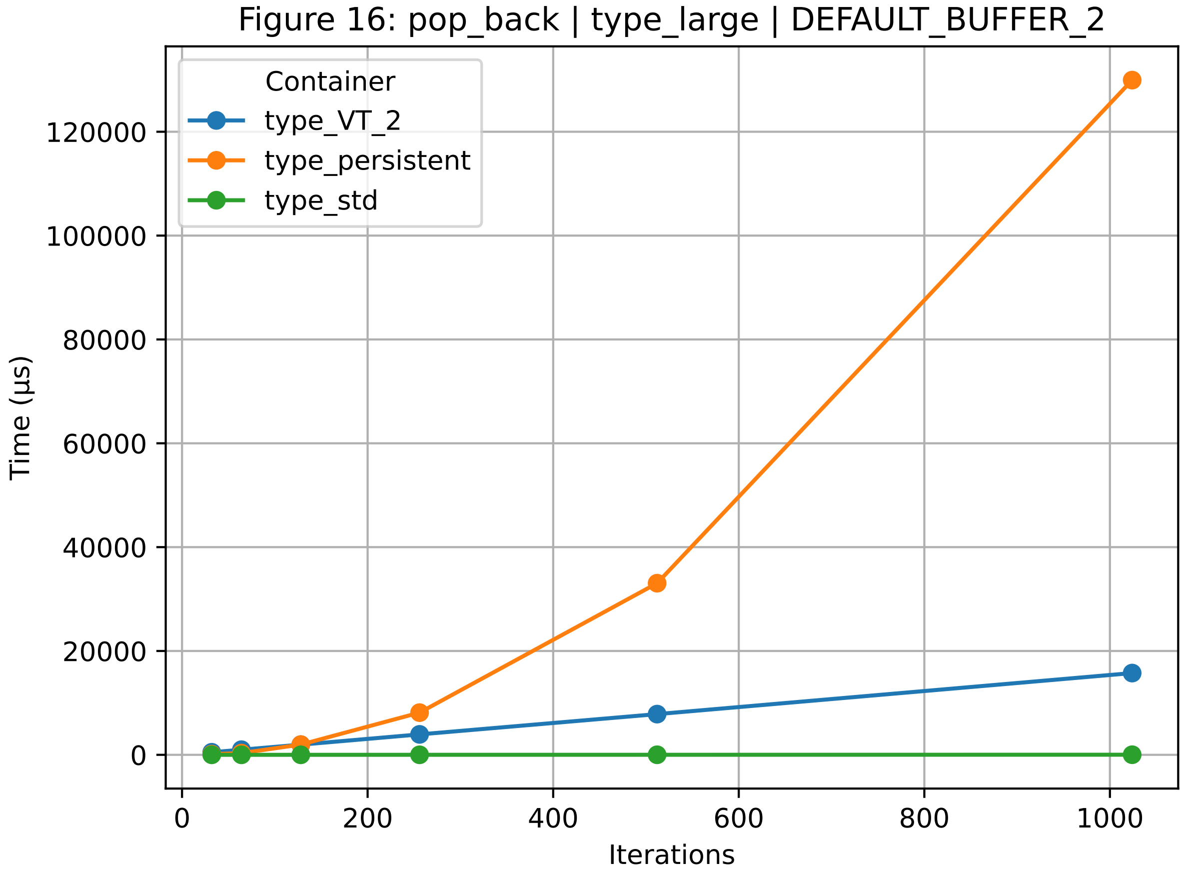


Figure 16: pop_back | type_large | DEFAULT_BUFFER_2
Container
type_VT_2
type_persistent
type_std
Time (µs)
Iterations
0
20000
40000
60000
80000
100000
120000
0
200
400
600
800
1000

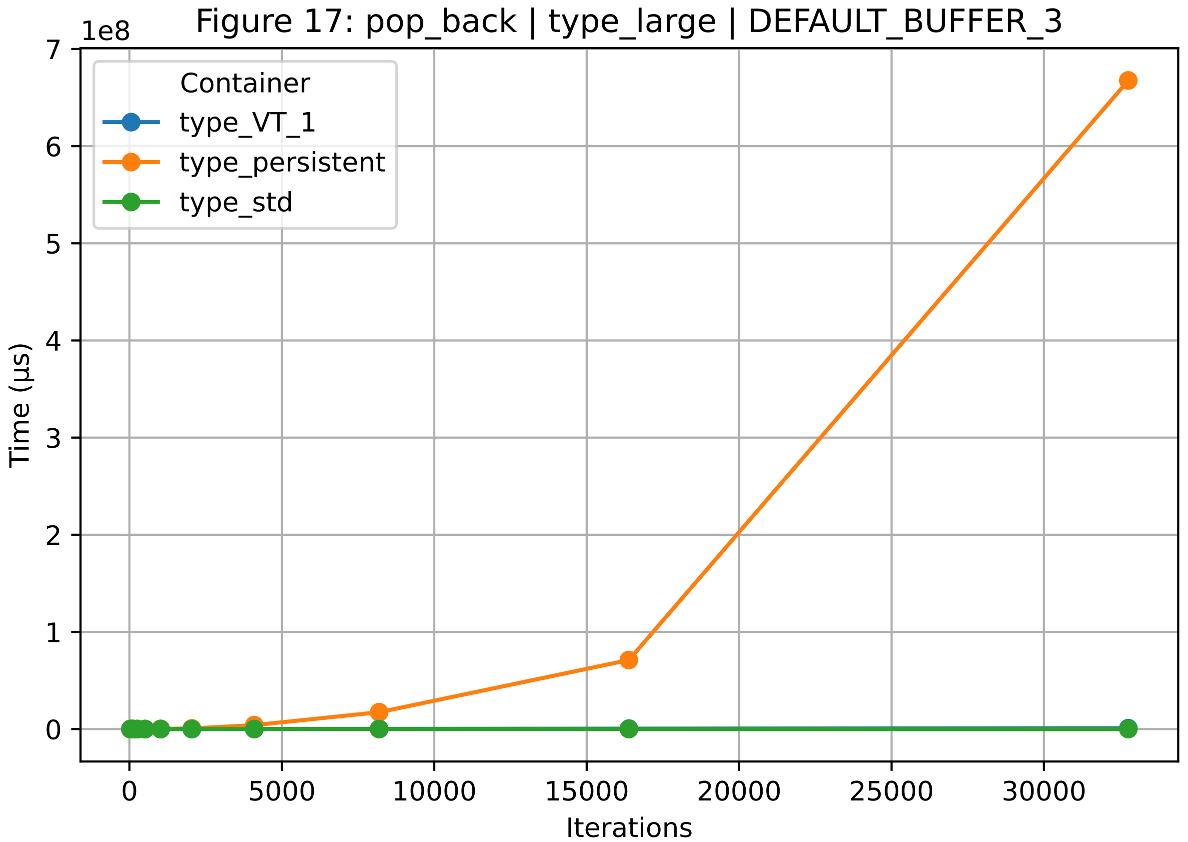


Figure 17: pop_back | type_large | DEFAULT_BUFFER_3
1e8
Container
type_VT_1
type_persistent
type_std
Time (µs)
Iterations
0
1
2
3
4
5
6
7
0
5000
10000
15000
20000
25000
30000

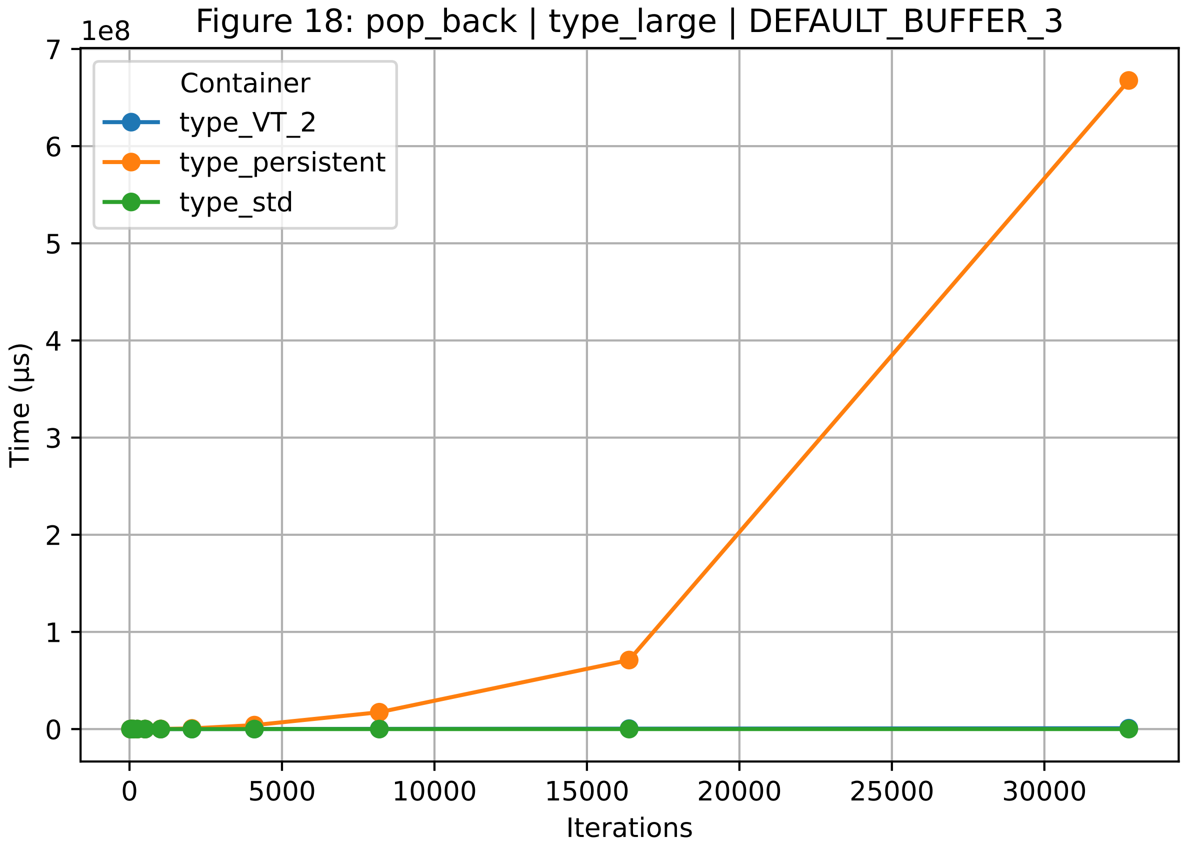
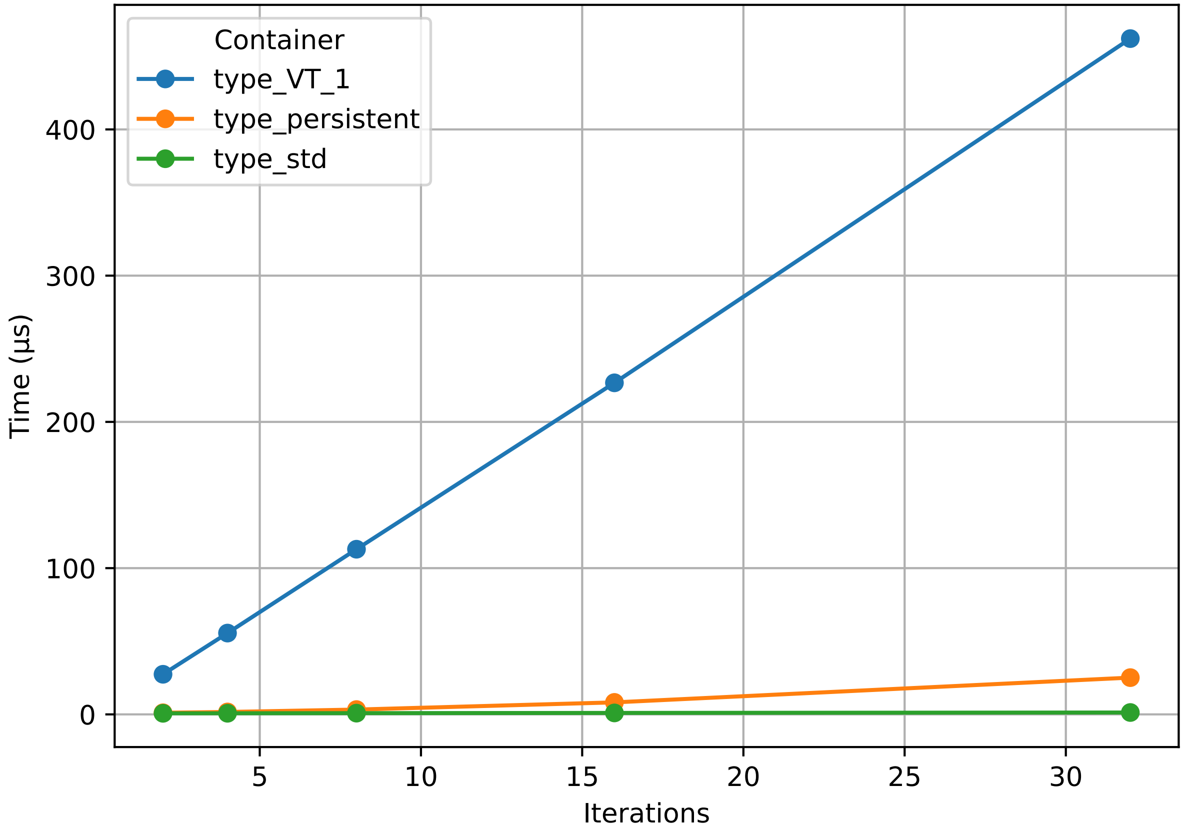


Figure 18: pop_back | type_large | DEFAULT_BUFFER_3
1e8
Container
type_VT_2
type_persistent
type_std
Time (µs)
Iterations
0
1
2
3
4
5
6
7
0
5000
10000
15000
20000
25000
30000

Figure 19: pop_back | type_small | DEFAULT_BUFFER_1



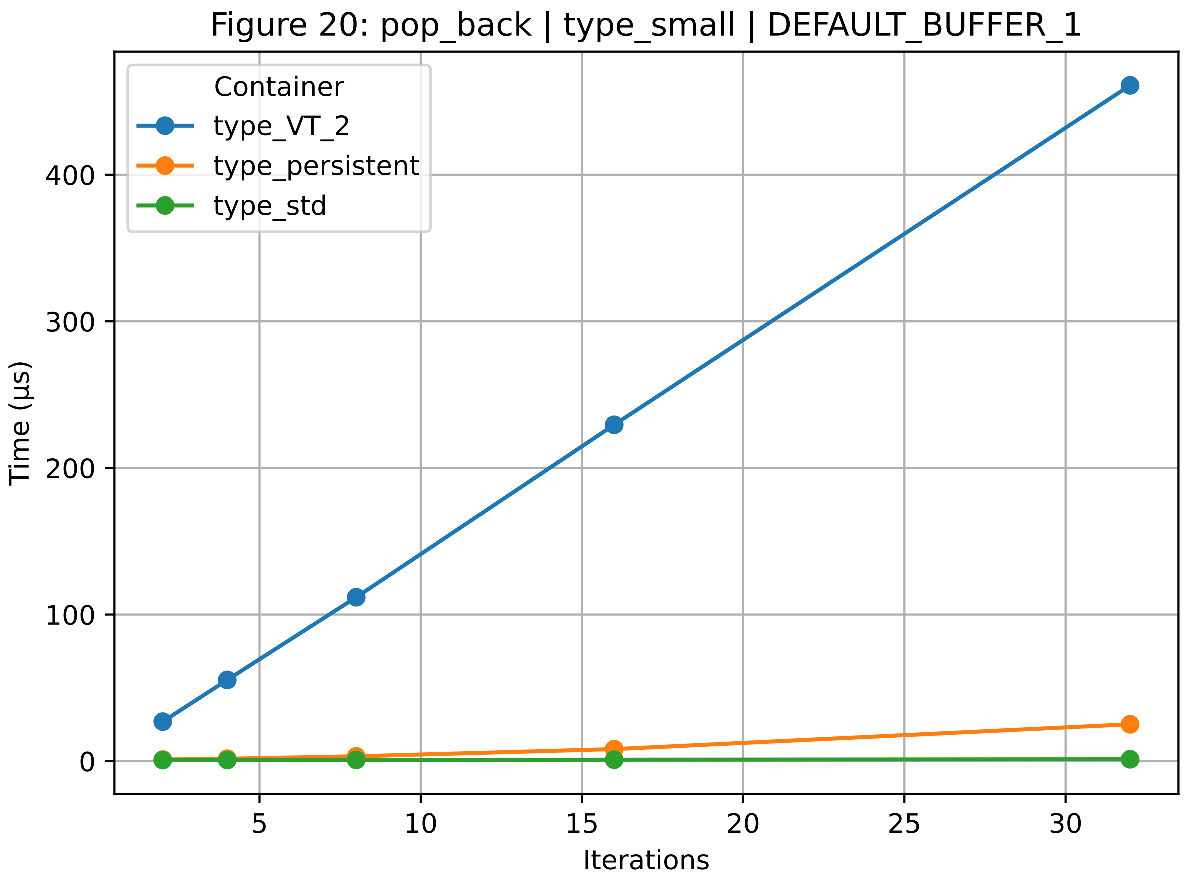


Figure 20: pop_back | type_small | DEFAULT_BUFFER_1
Container
type_VT_2
type_persistent
type_std
Time (µs)
0
100
200
300
400
5
10
15
20
25
30
Iterations

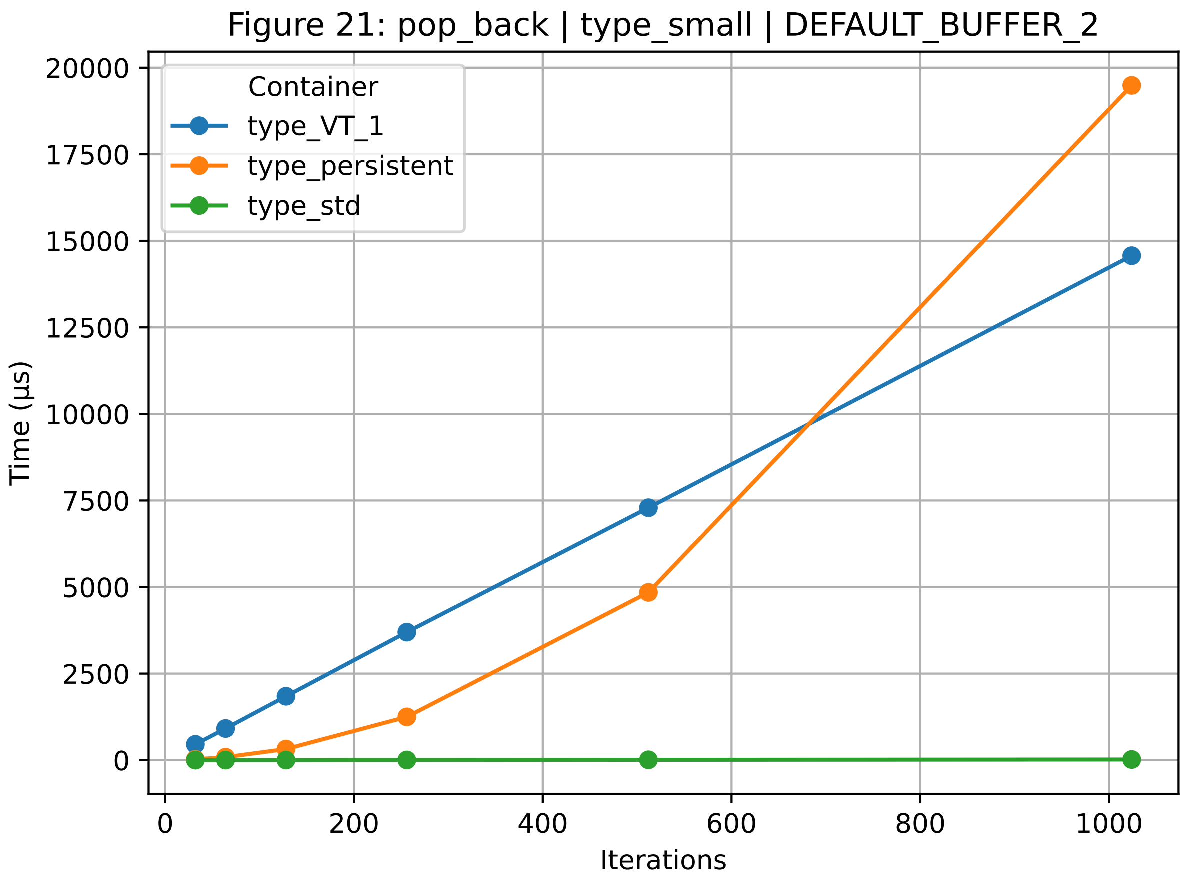


Figure 21: pop_back | type_small | DEFAULT_BUFFER_2
Container
type_VT_1
type_persistent
type_std
Time (µs)
Iterations
0
2500
5000
7500
10000
12500
15000
17500
20000
0
200
400
600
800
1000

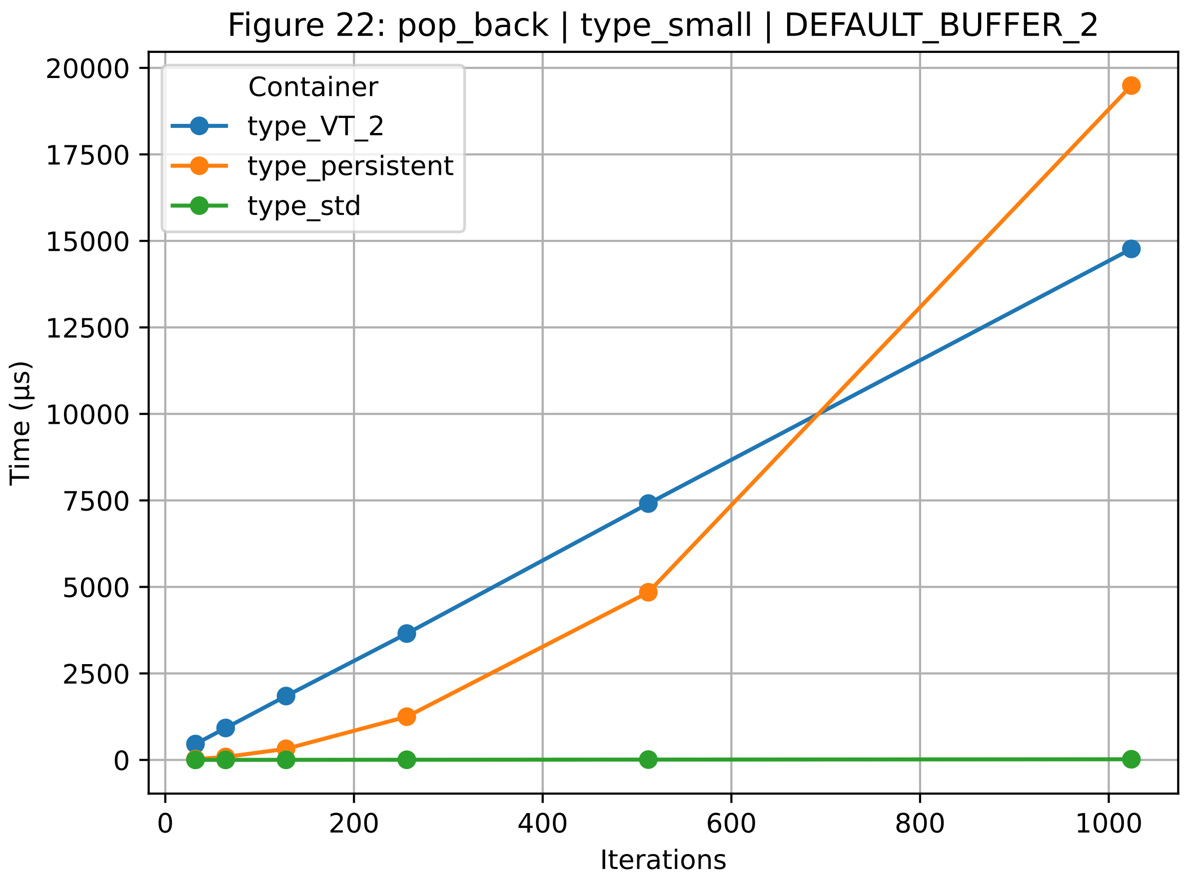


Figure 22: pop_back | type_small | DEFAULT_BUFFER_2
Container
type_VT_2
type_persistent
type_std
Time (µs)
Iterations
0
2500
5000
7500
10000
12500
15000
17500
20000
0
200
400
600
800
1000

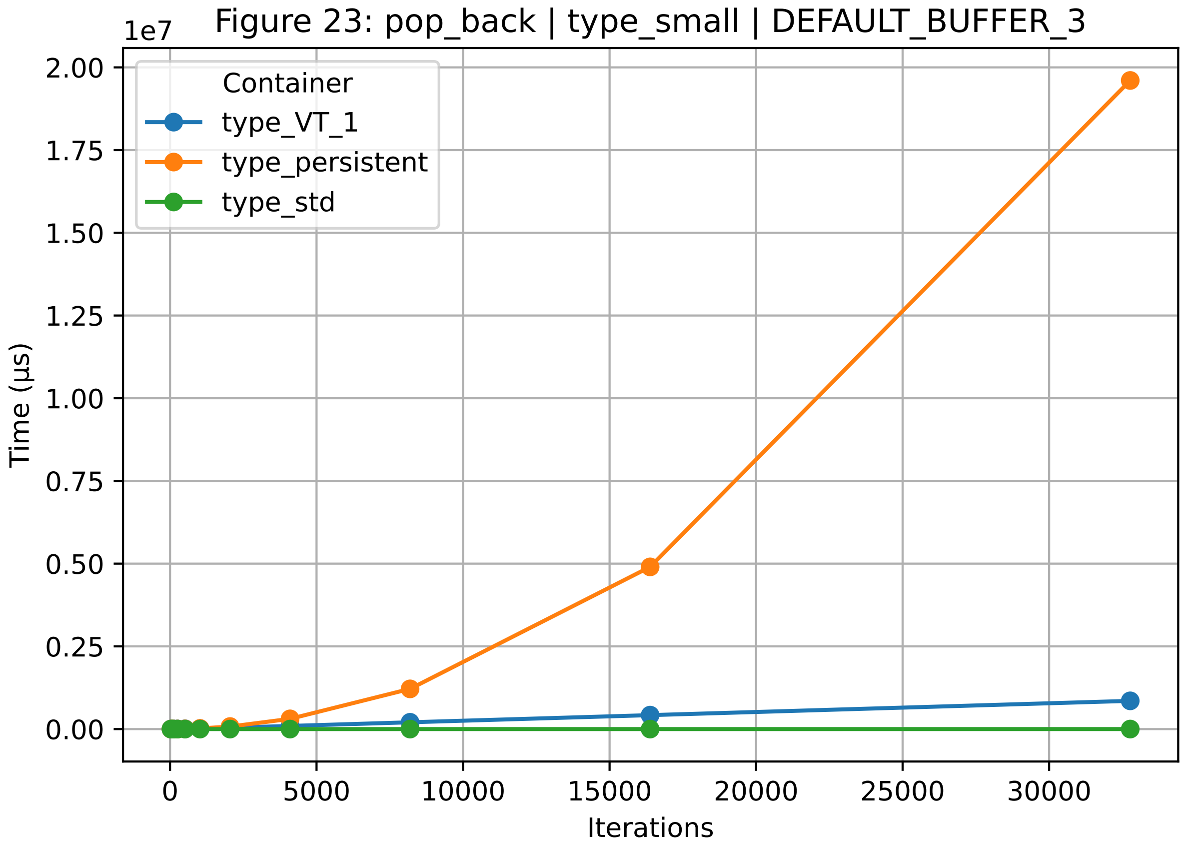


Figure 23: pop_back | type_small | DEFAULT_BUFFER_3
1e7
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Time (µs)
0
5000
10000
15000
20000
25000
30000
Iterations
Container
type_VT_1
type_persistent
type_std

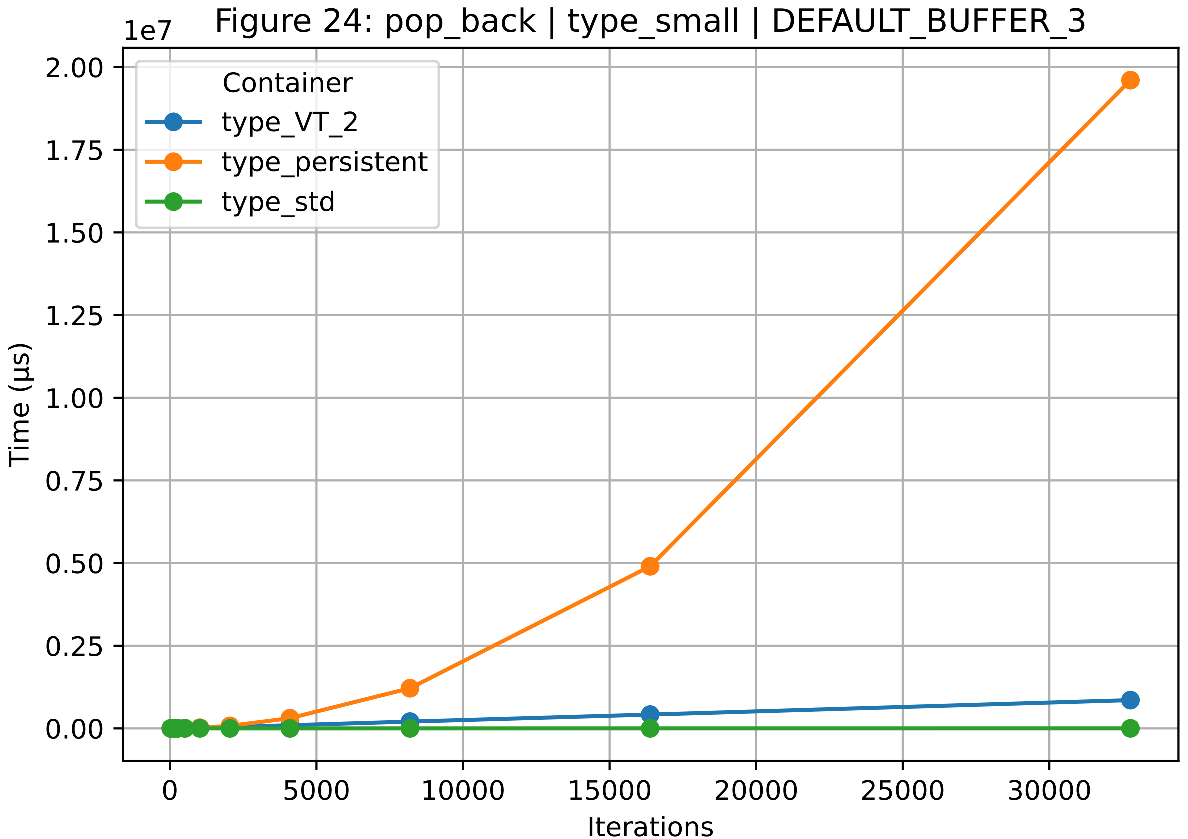
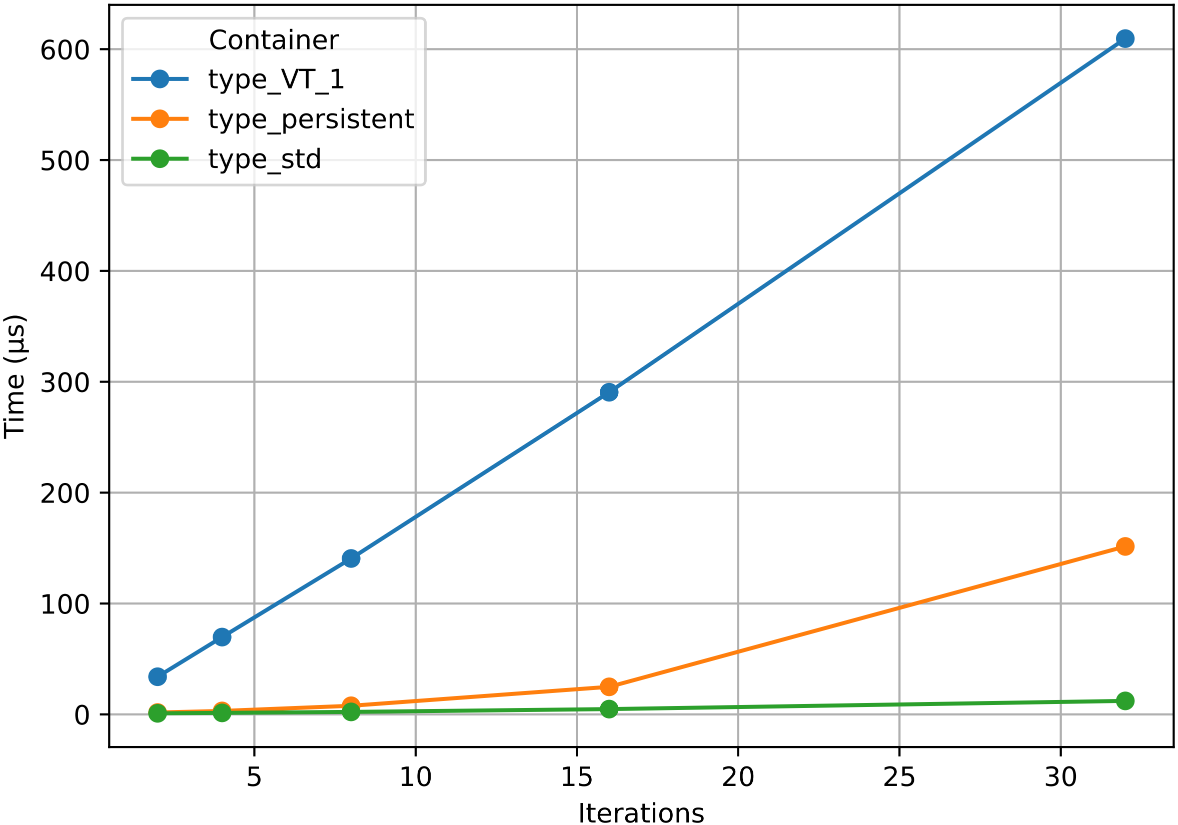


Figure 24: pop_back | type_small | DEFAULT_BUFFER_3
1e7
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Time (µs)
Container
type_VT_2
type_persistent
type_std
0
5000
10000
15000
20000
25000
30000
Iterations

Figure 25: pop_front | type_large | DEFAULT_BUFFER_1



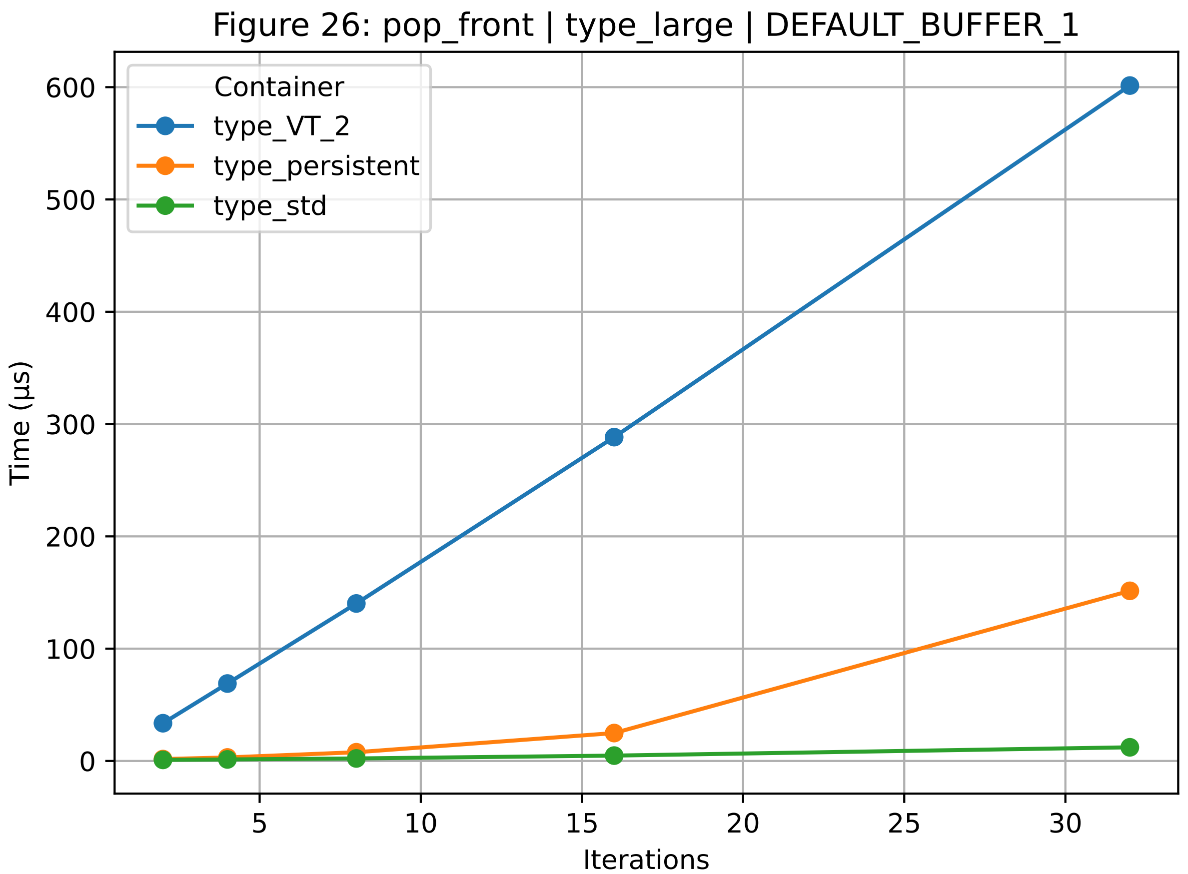


Figure 26: pop_front | type_large | DEFAULT_BUFFER_1
Container
type_VT_2
type_persistent
type_std
Time (µs)
Iterations
0
100
200
300
400
500
600
5
10
15
20
25
30

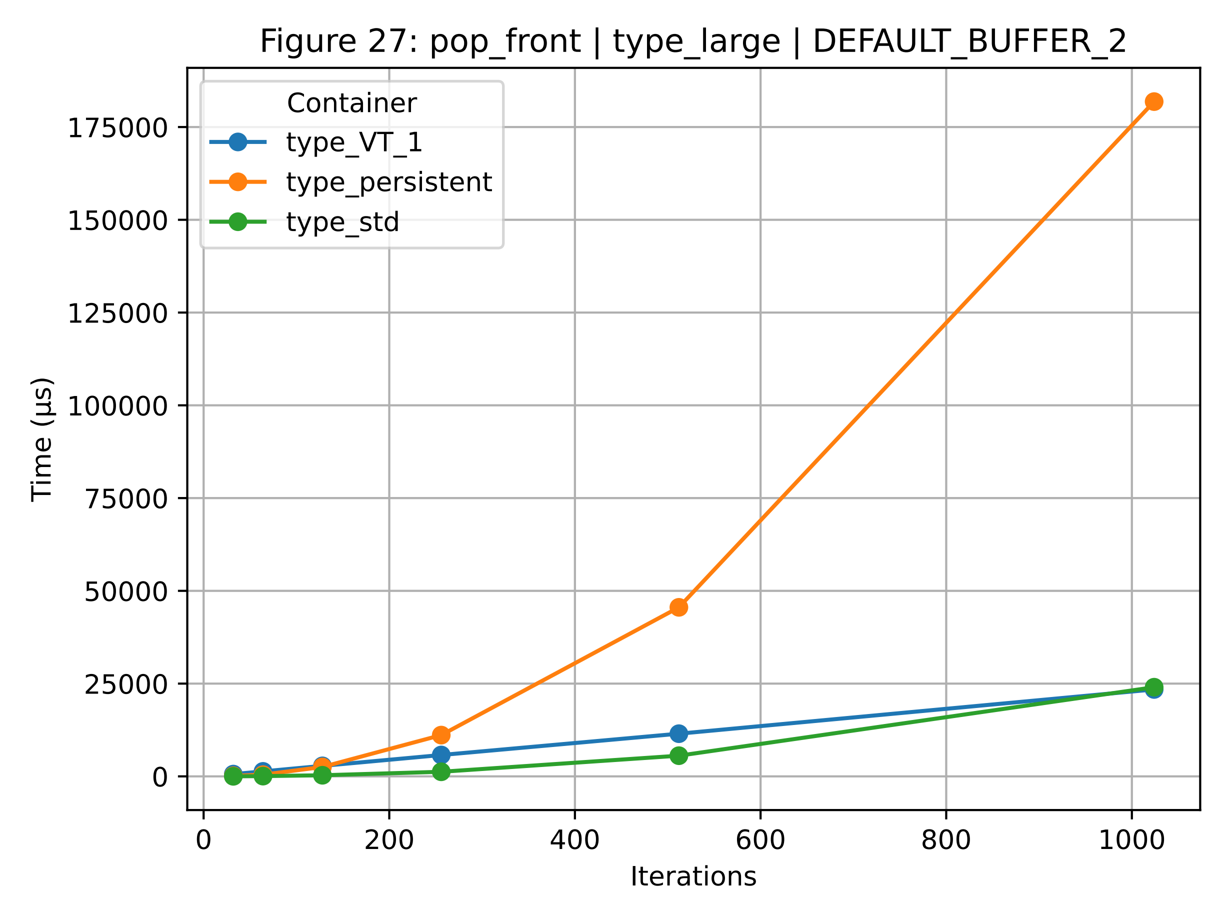


Figure 27: pop_front | type_large | DEFAULT_BUFFER_2
Container
type_VT_1
type_persistent
type_std
Time (µs)
175000
150000
125000
100000
75000
50000
25000
0
0
200
400
600
800
1000
Iterations

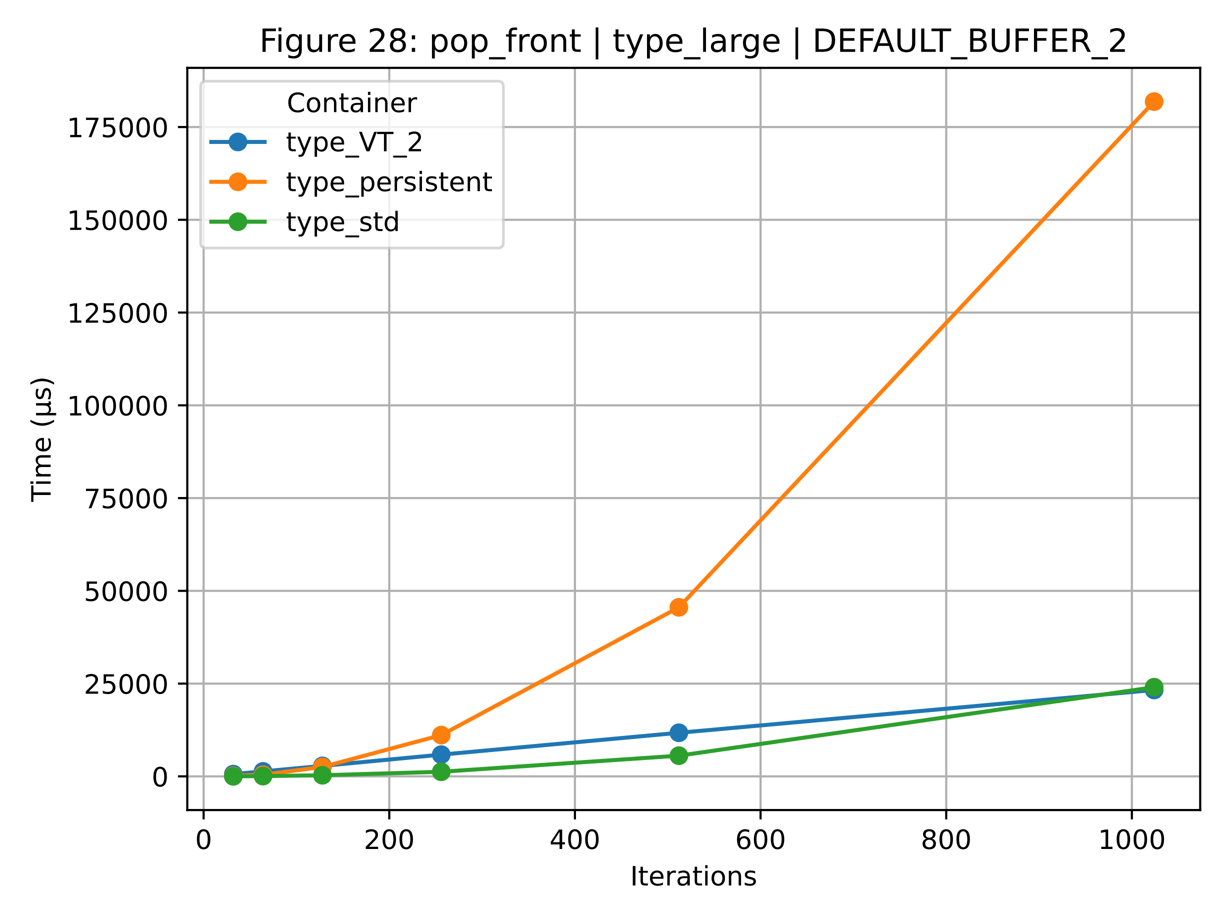


Figure 28: pop_front | type_large | DEFAULT_BUFFER_2
Container
type_VT_2
type_persistent
type_std
Time (µs)
175000
150000
125000
100000
75000
50000
25000
0
0
200
400
600
800
1000
Iterations

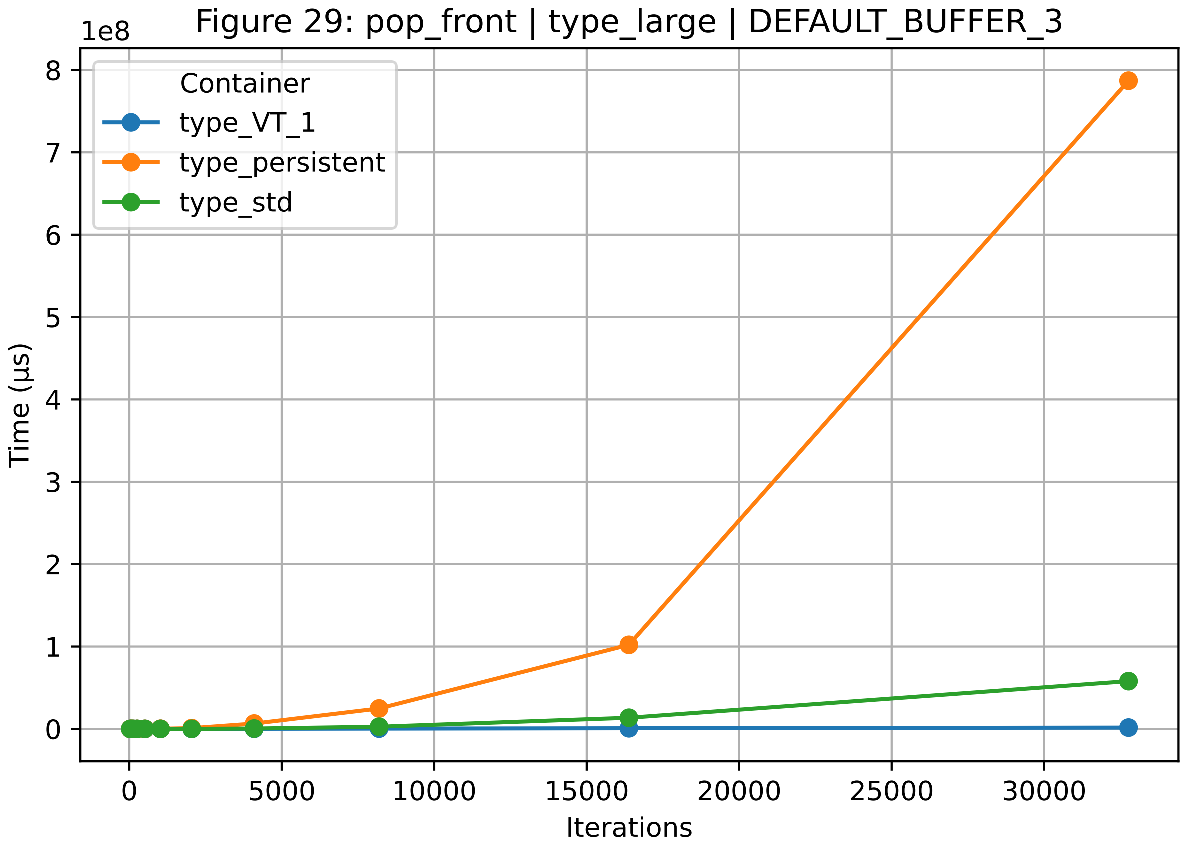


Figure 29: pop_front | type_large | DEFAULT_BUFFER_3
1e8
Container
type_VT_1
type_persistent
type_std
Time (µs)
Iterations
0
1
2
3
4
5
6
7
8
0
5000
10000
15000
20000
25000
30000

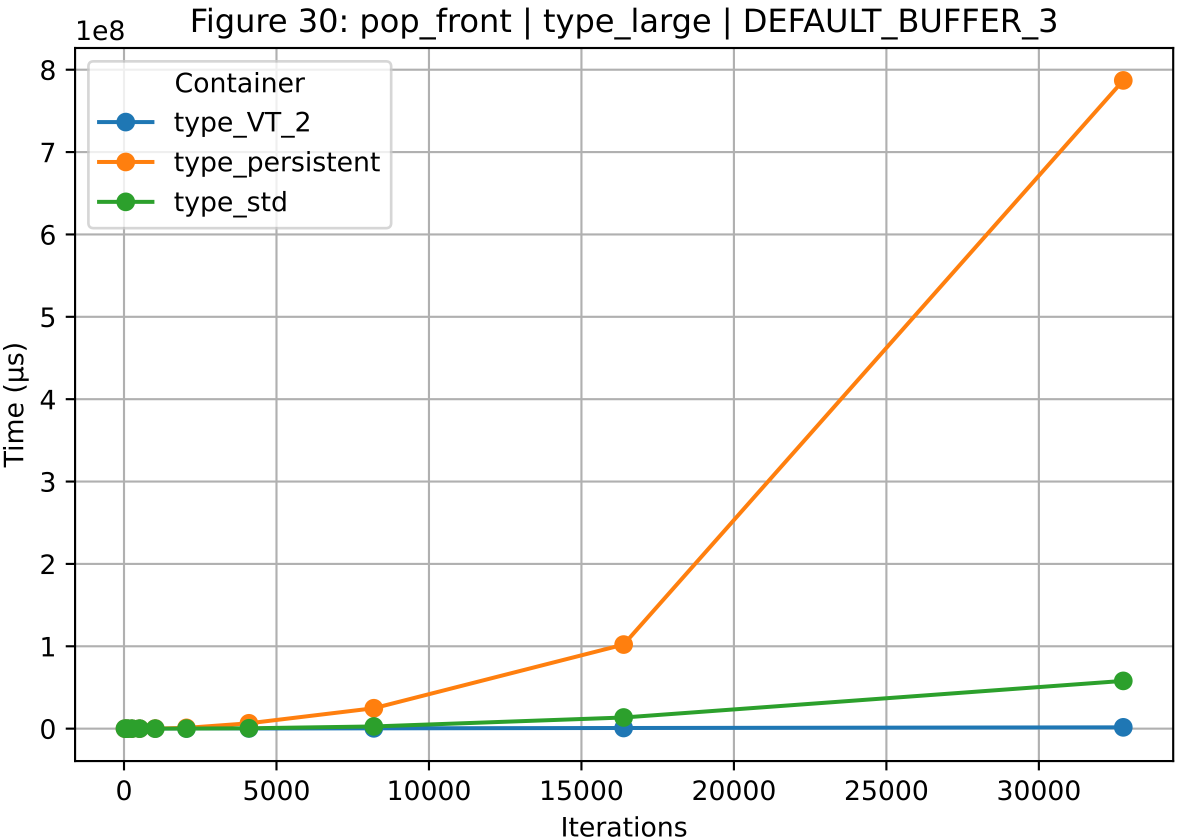


Figure 30: pop_front | type_large | DEFAULT_BUFFER_3
1e8
8
7
6
5
4
3
2
1
0
Time (µs)
Container
type_VT_2
type_persistent
type_std
0
5000
10000
15000
20000
25000
30000
Iterations

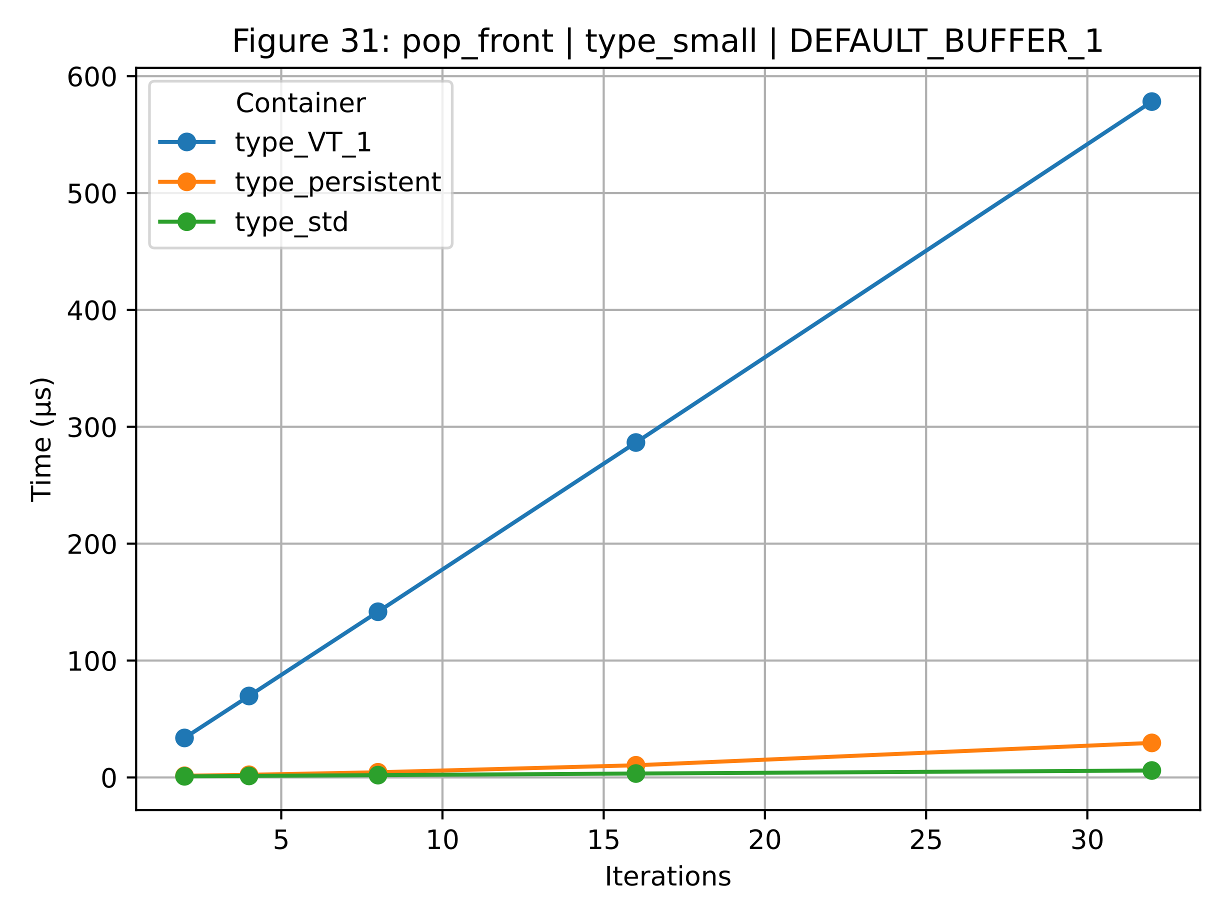


Figure 31: pop_front | type_small | DEFAULT_BUFFER_1
Container
type_VT_1
type_persistent
type_std
Time (µs)
Iterations
0
100
200
300
400
500
600
5
10
15
20
25
30

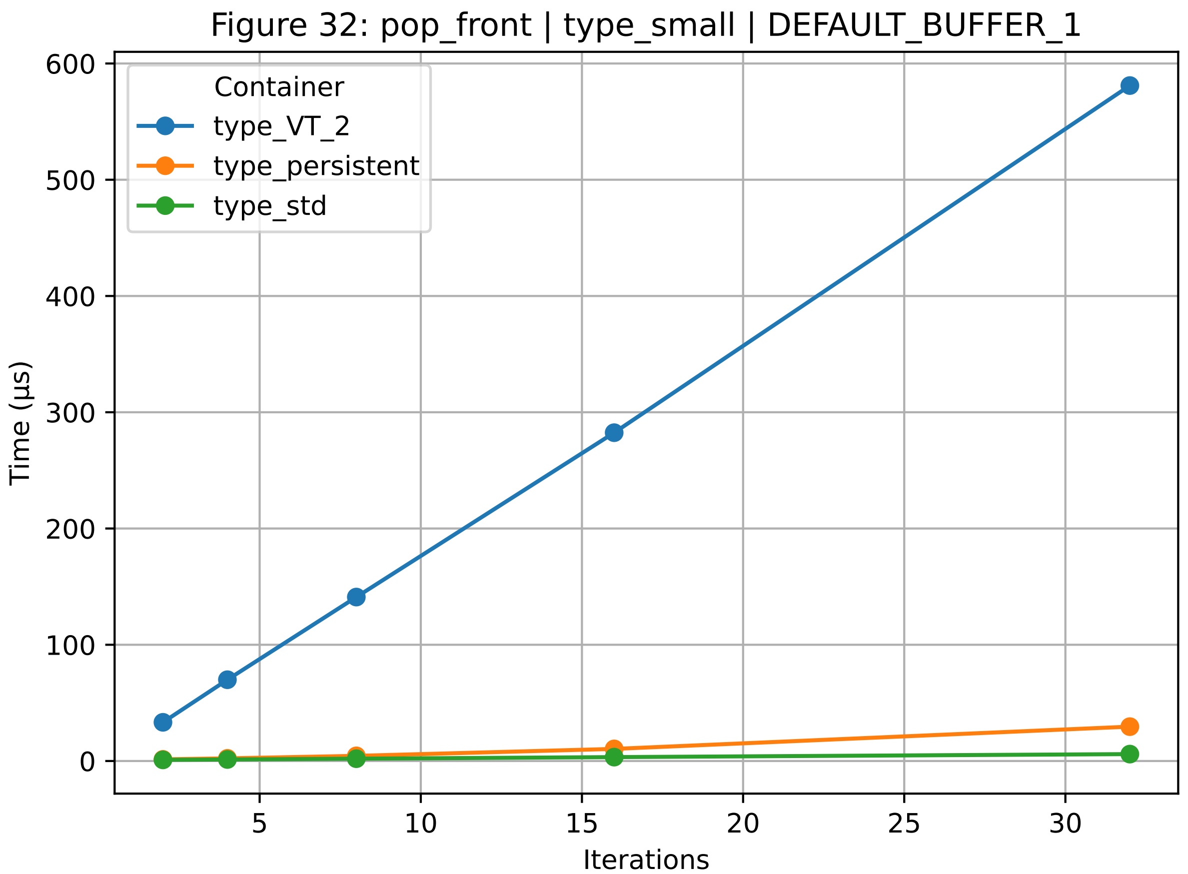


Figure 32: pop_front | type_small | DEFAULT_BUFFER_1
Container
type_VT_2
type_persistent
type_std
Time (µs)
Iterations
0
100
200
300
400
500
600
5
10
15
20
25
30

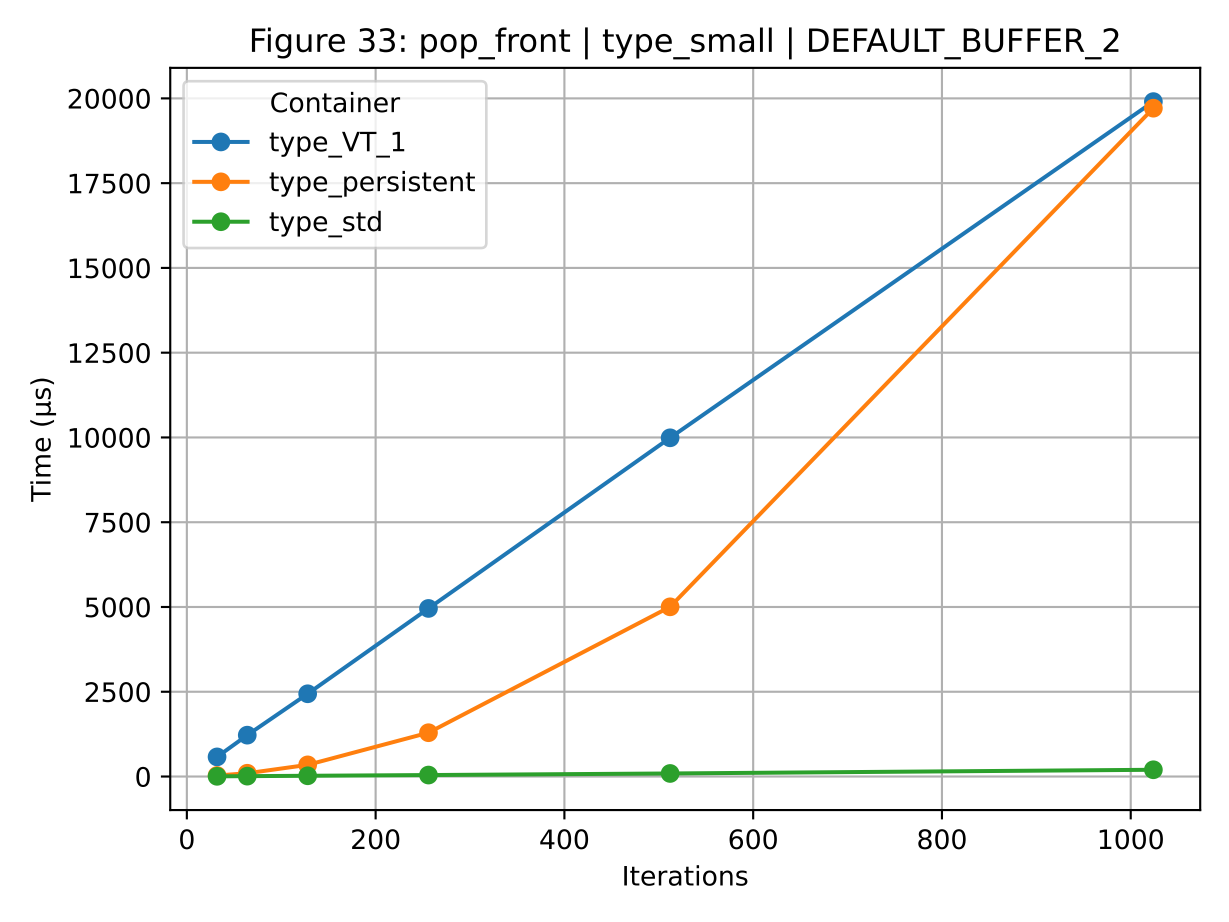


Figure 33: pop_front | type_small | DEFAULT_BUFFER_2
Container
type_VT_1
type_persistent
type_std
Time (µs)
Iterations
0
2500
5000
7500
10000
12500
15000
17500
20000
0
200
400
600
800
1000

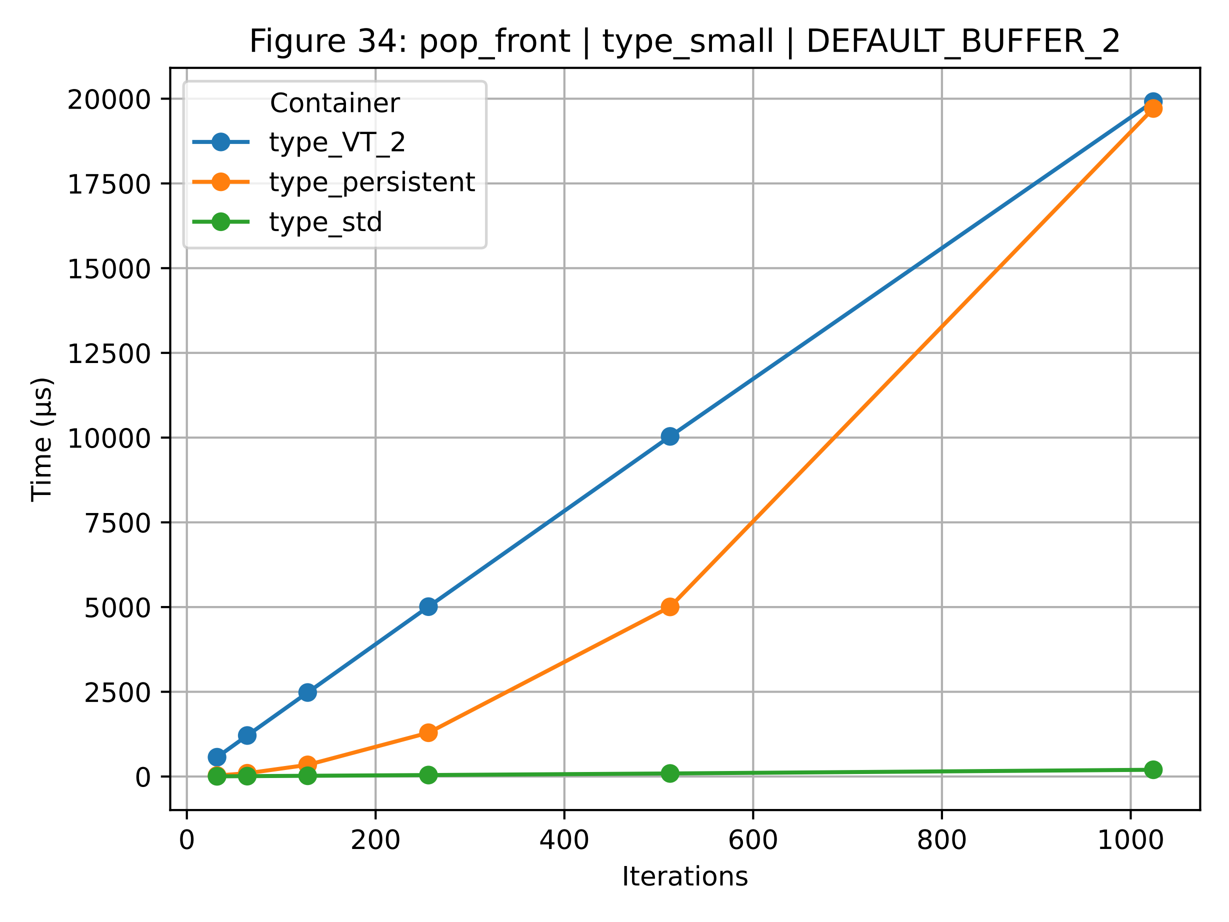


Figure 34: pop_front | type_small | DEFAULT_BUFFER_2
Time (µs)
Iterations
Container
type_VT_2
type_persistent
type_std
0
2500
5000
7500
10000
12500
15000
17500
20000
0
200
400
600
800
1000

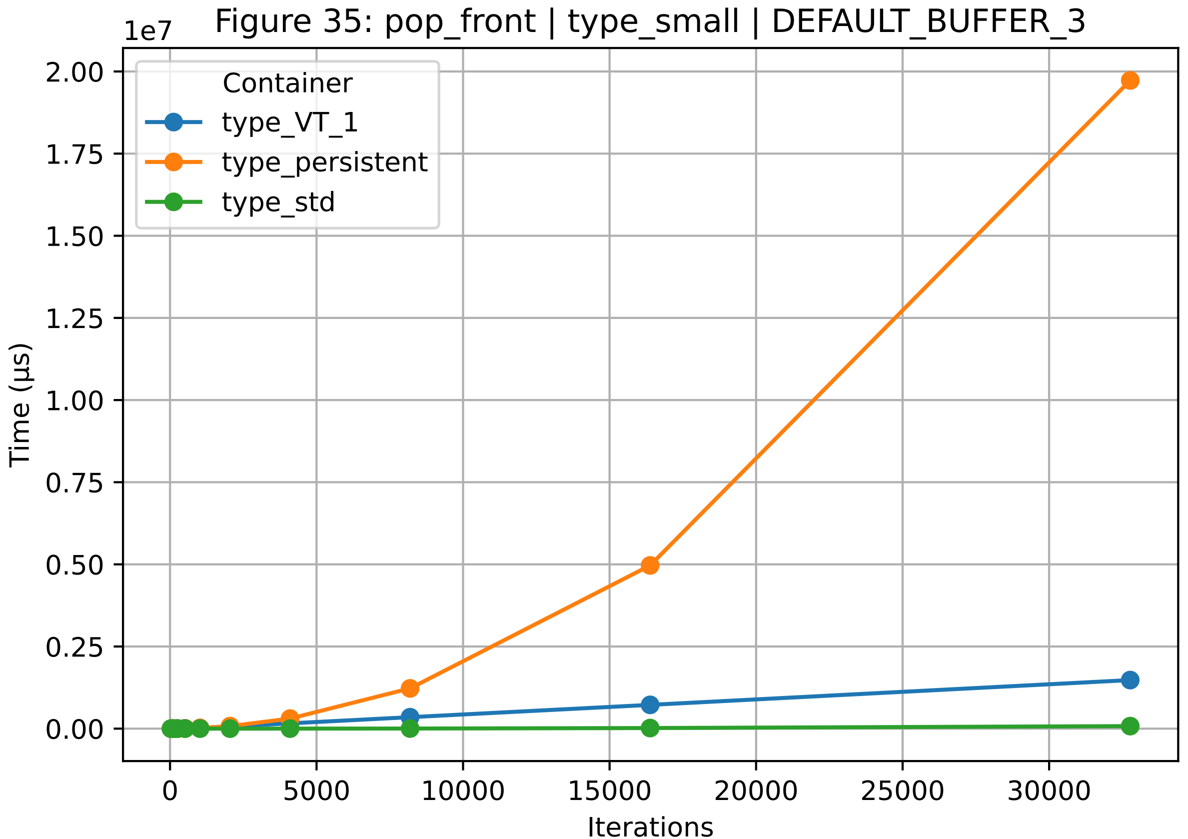


Figure 35: pop_front | type_small | DEFAULT_BUFFER_3
1e7
Container
type_VT_1
type_persistent
type_std
Time (µs)
Iterations
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
5000
10000
15000
20000
25000
30000

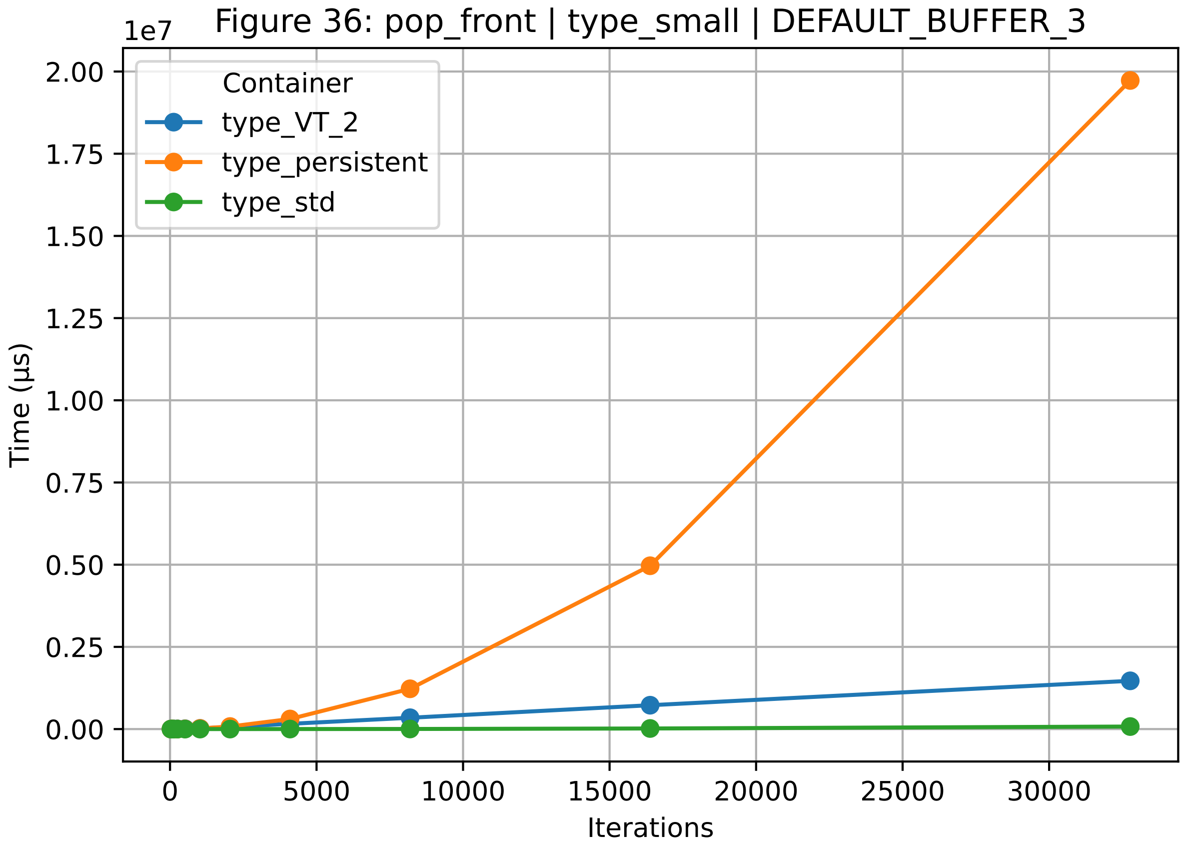


Figure 36: pop_front | type_small | DEFAULT_BUFFER_3
1e7
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Time (μs)
Container
type_VT_2
type_persistent
type_std
0
5000
10000
15000
20000
25000
30000
Iterations

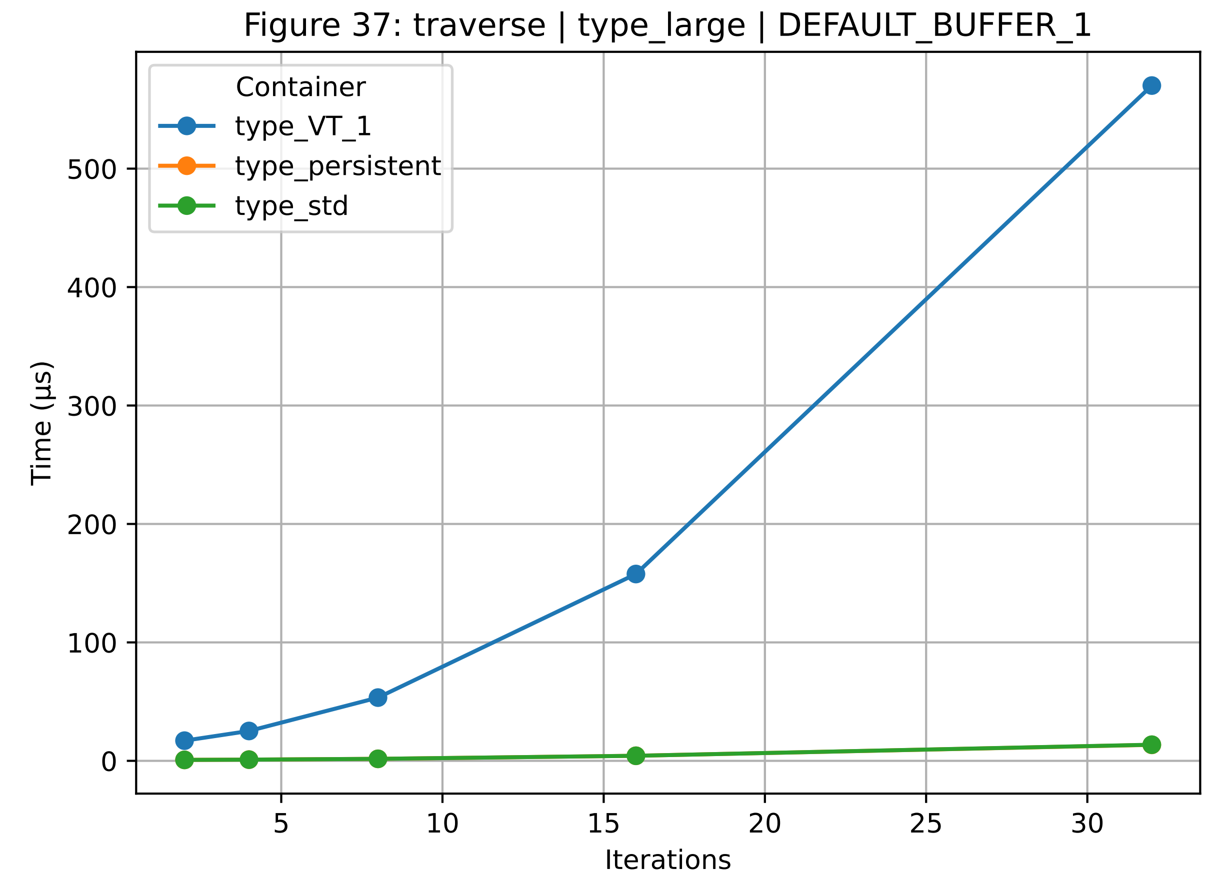


Figure 37: traverse | type_large | DEFAULT_BUFFER_1
Container
type_VT_1
type_persistent
type_std
Time (µs)
0
100
200
300
400
500
5
10
15
20
25
30
Iterations

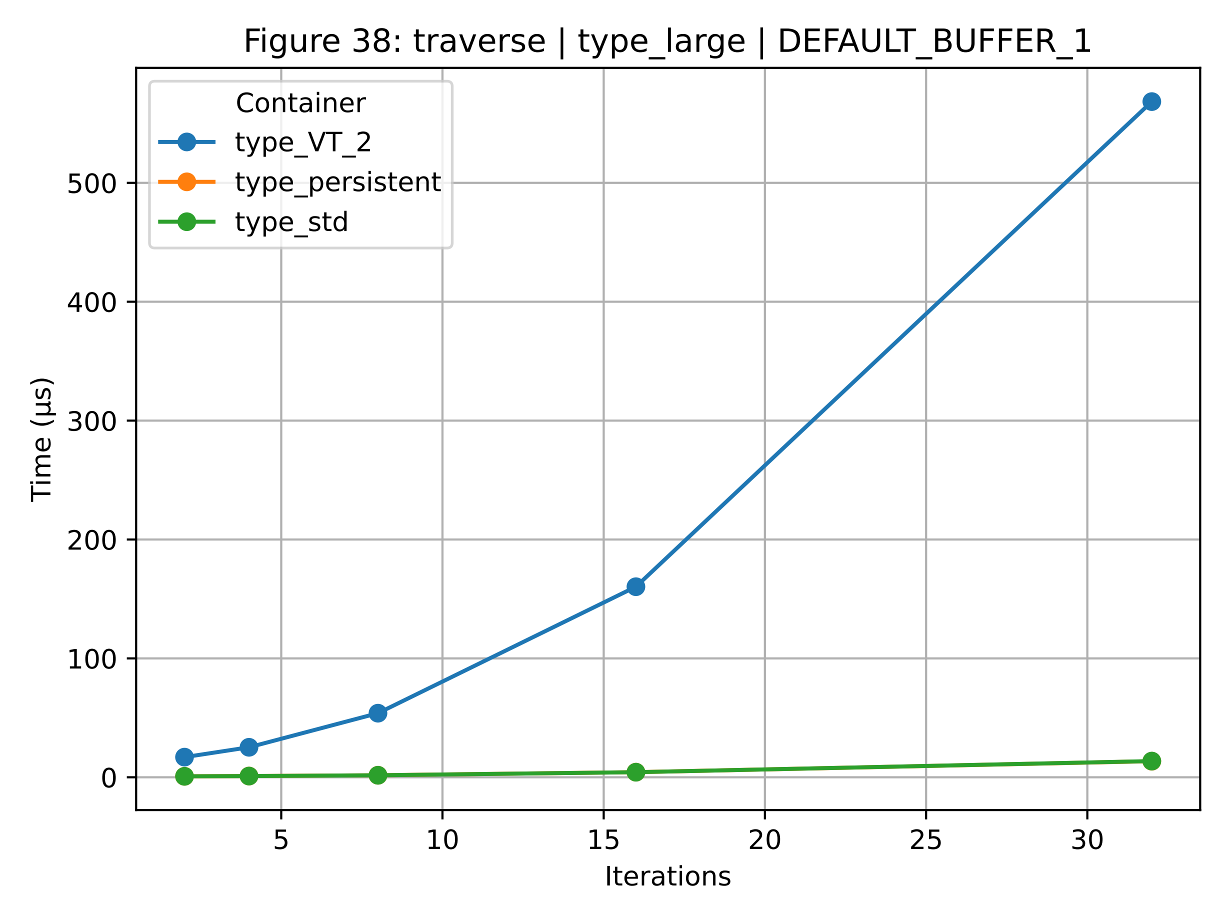


Figure 38: traverse | type_large | DEFAULT_BUFFER_1
Container
type_VT_2
type_persistent
type_std
Time (µs)
0
100
200
300
400
500
5
10
15
20
25
30
Iterations

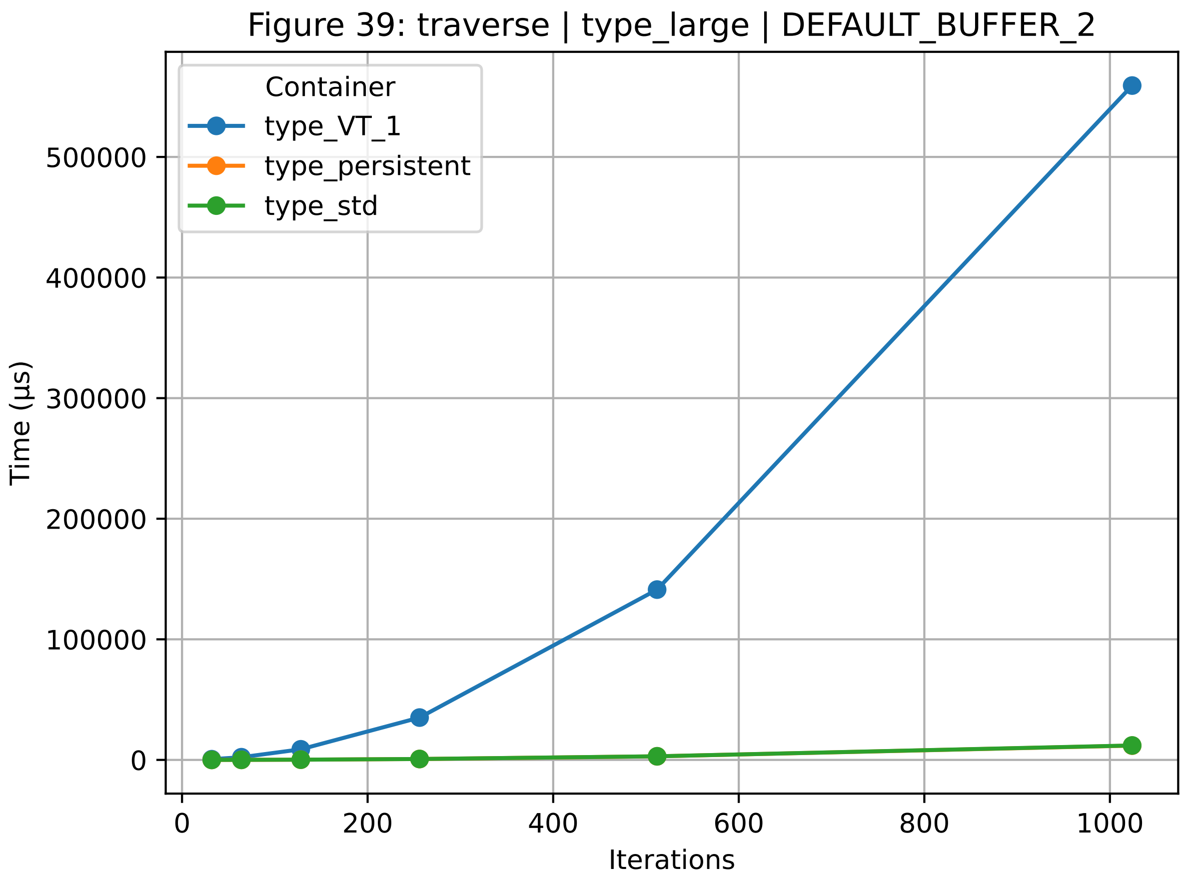


Figure 39: traverse | type_large | DEFAULT_BUFFER_2
Container
type_VT_1
type_persistent
type_std
Time (µs)
Iterations
0
100000
200000
300000
400000
500000
0
200
400
600
800
1000

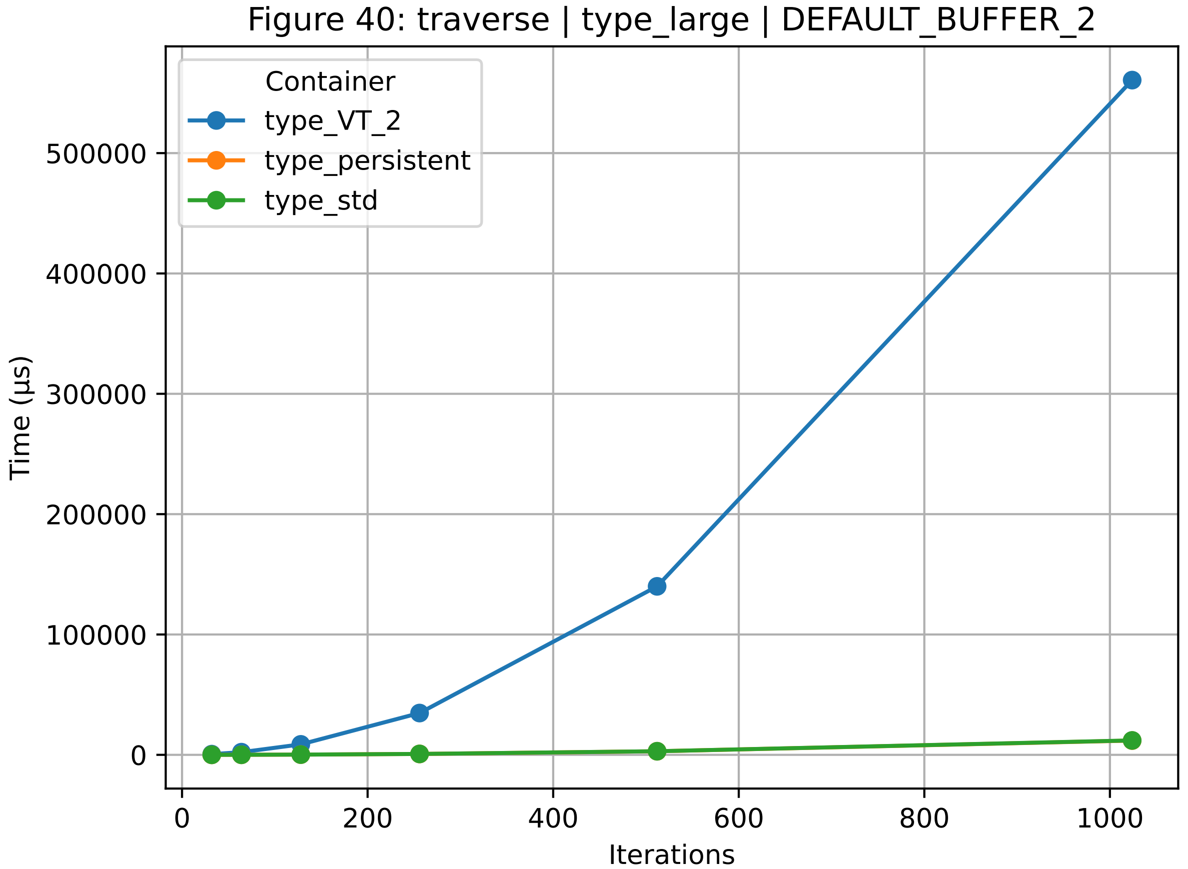
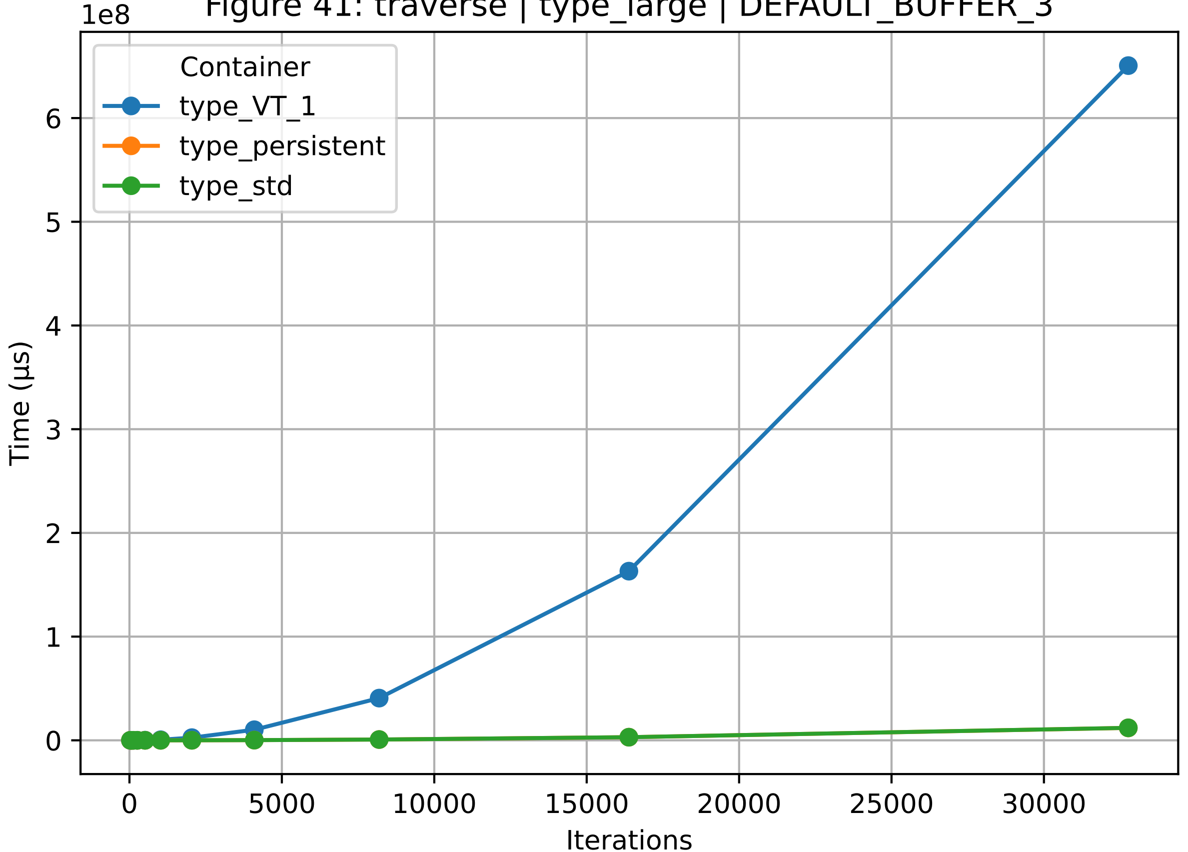


Figure 40: traverse | type_large | DEFAULT_BUFFER_2
Container
type_VT_2
type_persistent
type_std
Time (µs)
500000
400000
300000
200000
100000
0
0
200
400
600
800
1000
Iterations

Figure 41: traverse | type_large | DEFAULT_BUFFER_3



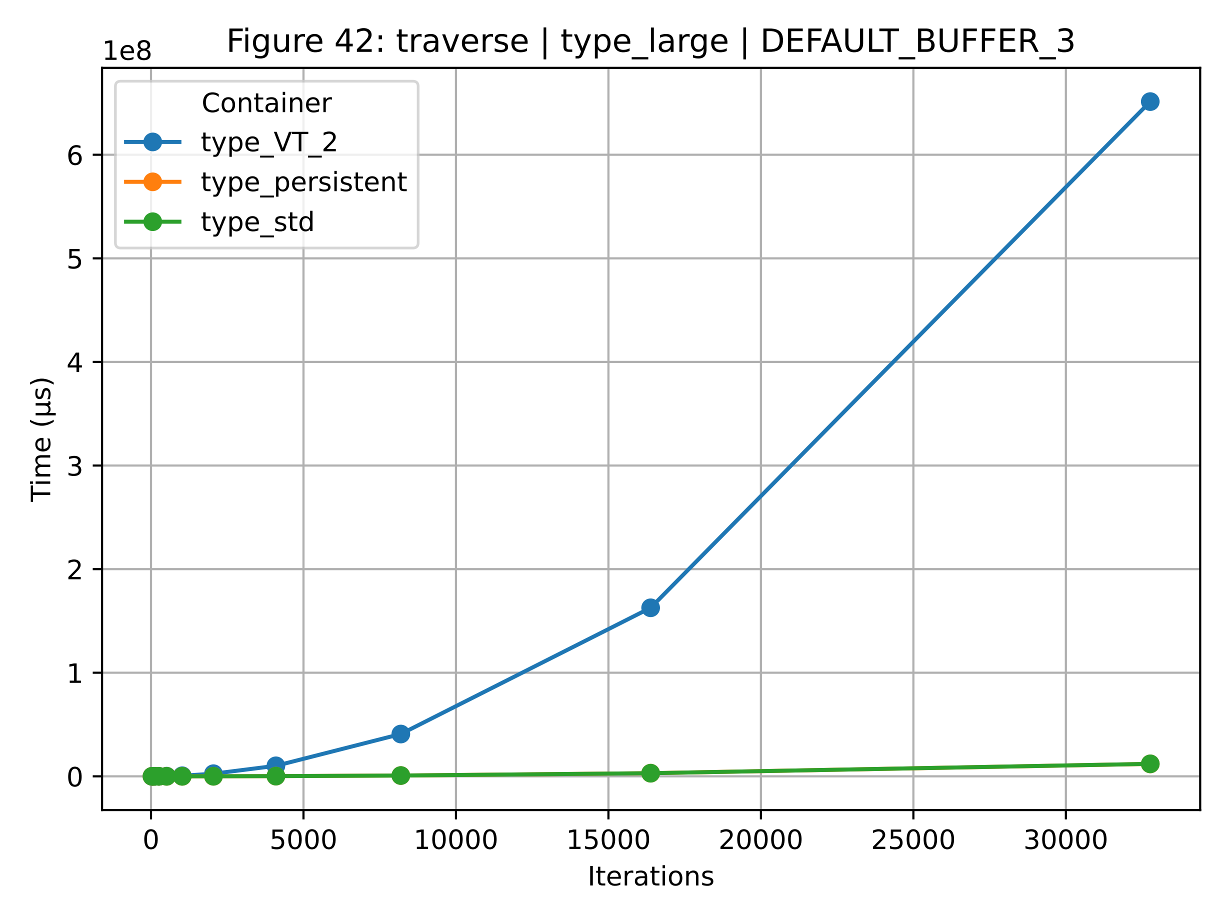


Figure 42: traverse | type_large | DEFAULT_BUFFER_3
1e8
Container
type_VT_2
type_persistent
type_std
Time (µs)
Iterations
0
1
2
3
4
5
6
0
5000
10000
15000
20000
25000
30000

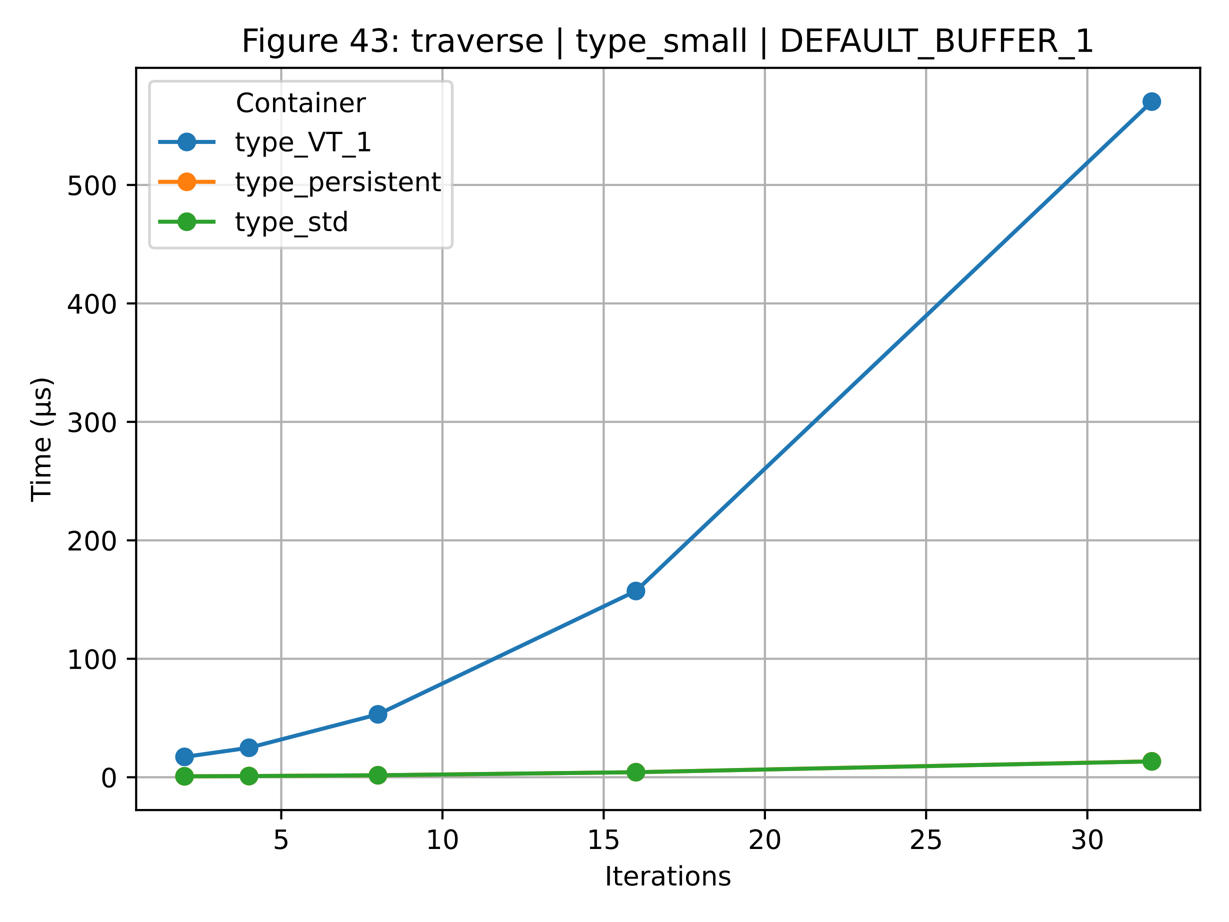


Figure 43: traverse | type_small | DEFAULT_BUFFER_1
Container
type_VT_1
type_persistent
type_std
Time (µs)
0
100
200
300
400
500
5
10
15
20
25
30
Iterations

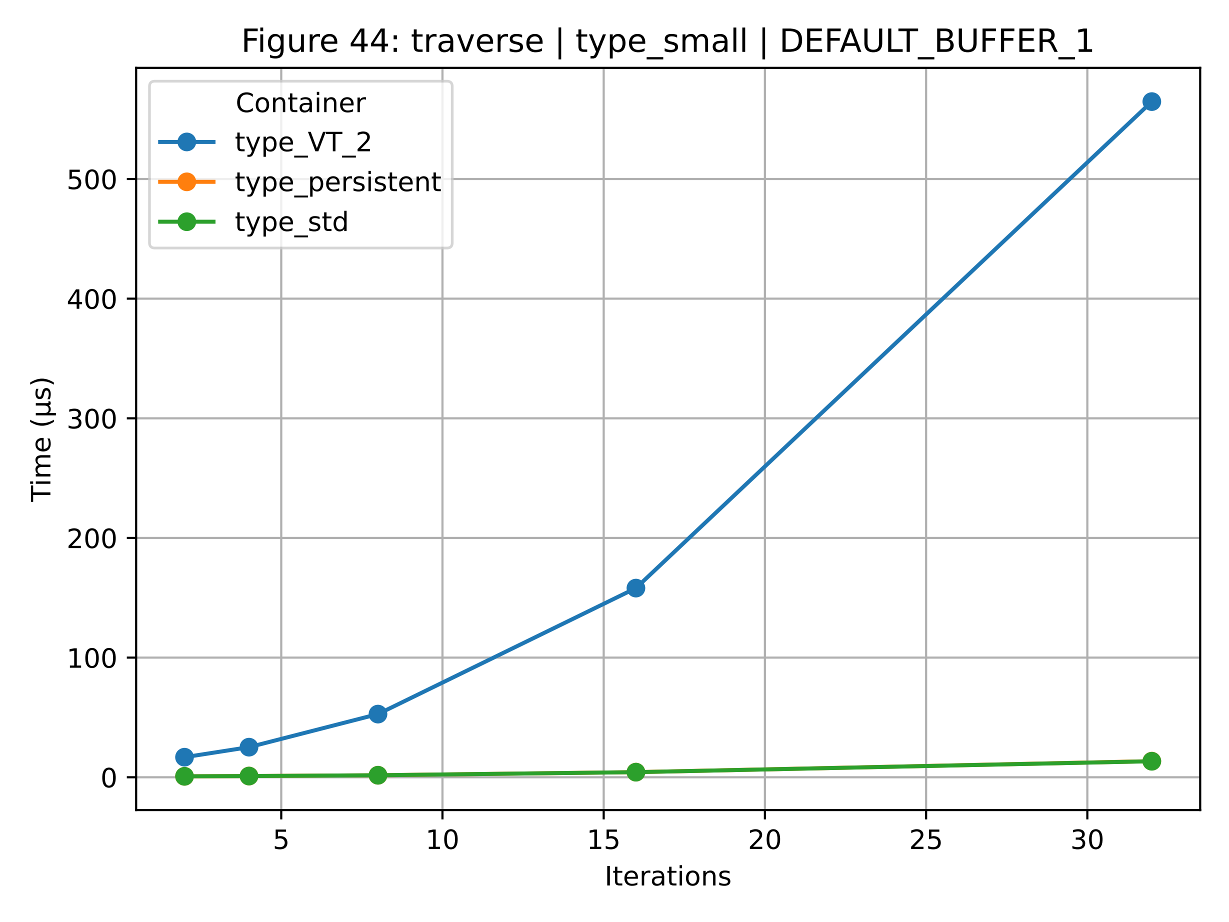


Figure 44: traverse | type_small | DEFAULT_BUFFER_1
Container
type_VT_2
type_persistent
type_std
500
400
300
200
100
0
Time (µs)
5
10
15
20
25
30
Iterations

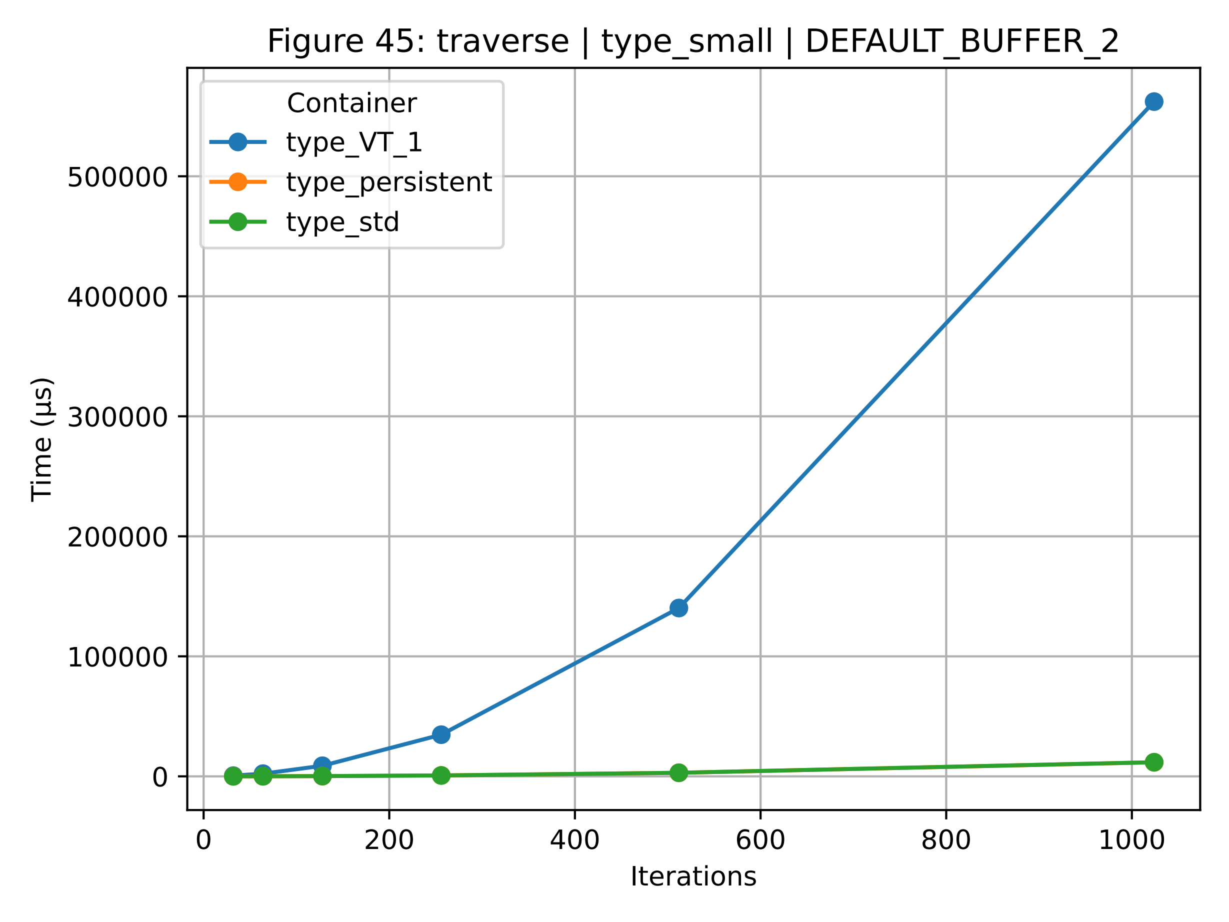


Figure 45: traverse | type_small | DEFAULT_BUFFER_2
Container
type_VT_1
type_persistent
type_std
Time (µs)
0
100000
200000
300000
400000
500000
0
200
400
600
800
1000
Iterations

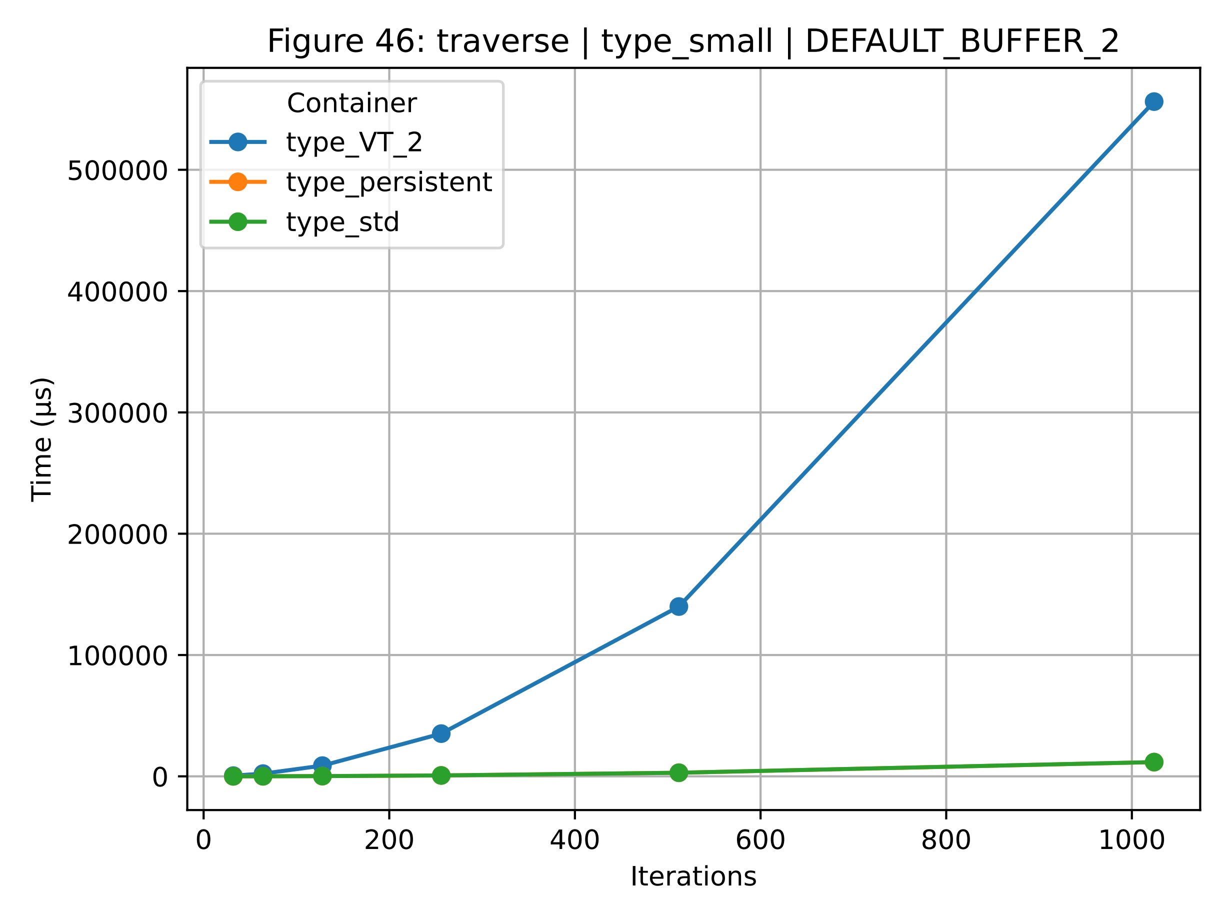


Figure 46: traverse | type_small | DEFAULT_BUFFER_2
Container
type_VT_2
type_persistent
type_std
Time (μs)
0
100000
200000
300000
400000
500000
0
200
400
600
800
1000
Iterations

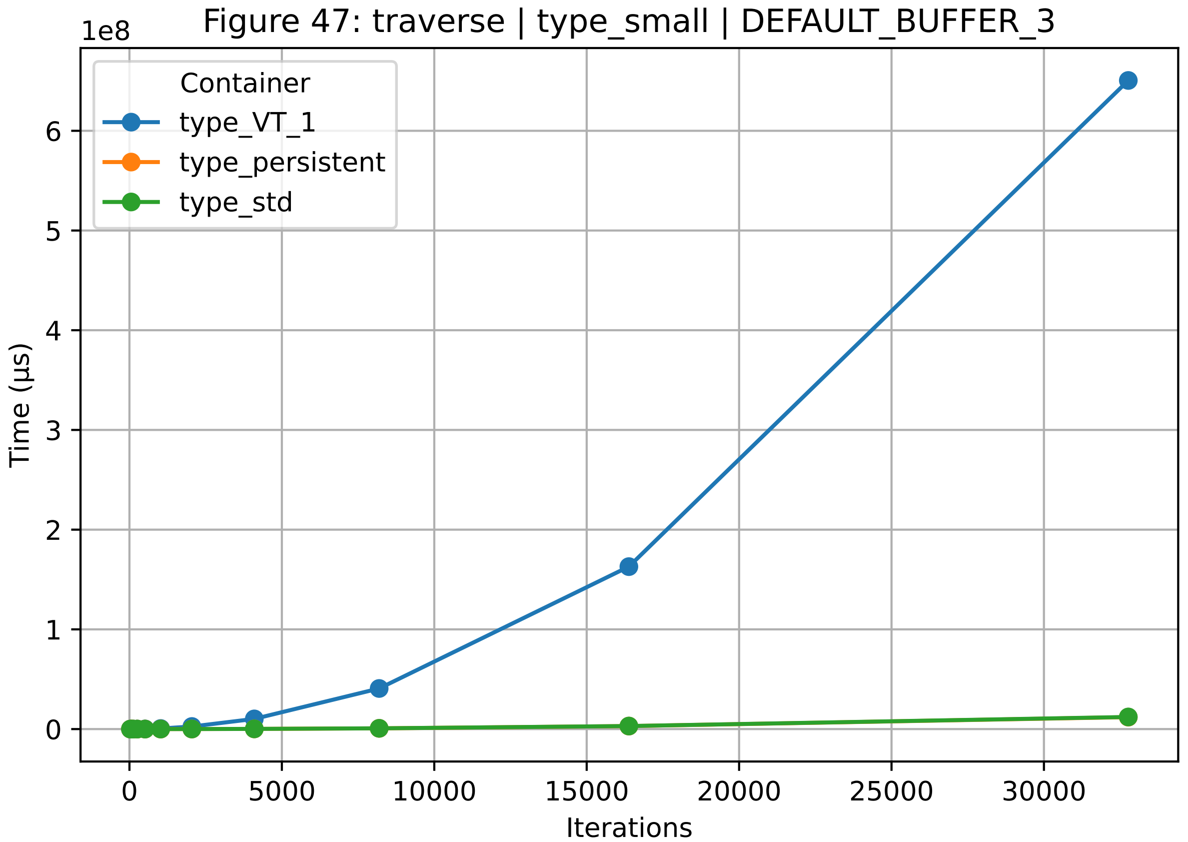


Figure 47: traverse | type_small | DEFAULT_BUFFER_3
1e8
Container
type_VT_1
type_persistent
type_std
Time (µs)
Iterations
0
1
2
3
4
5
6
0
5000
10000
15000
20000
25000
30000

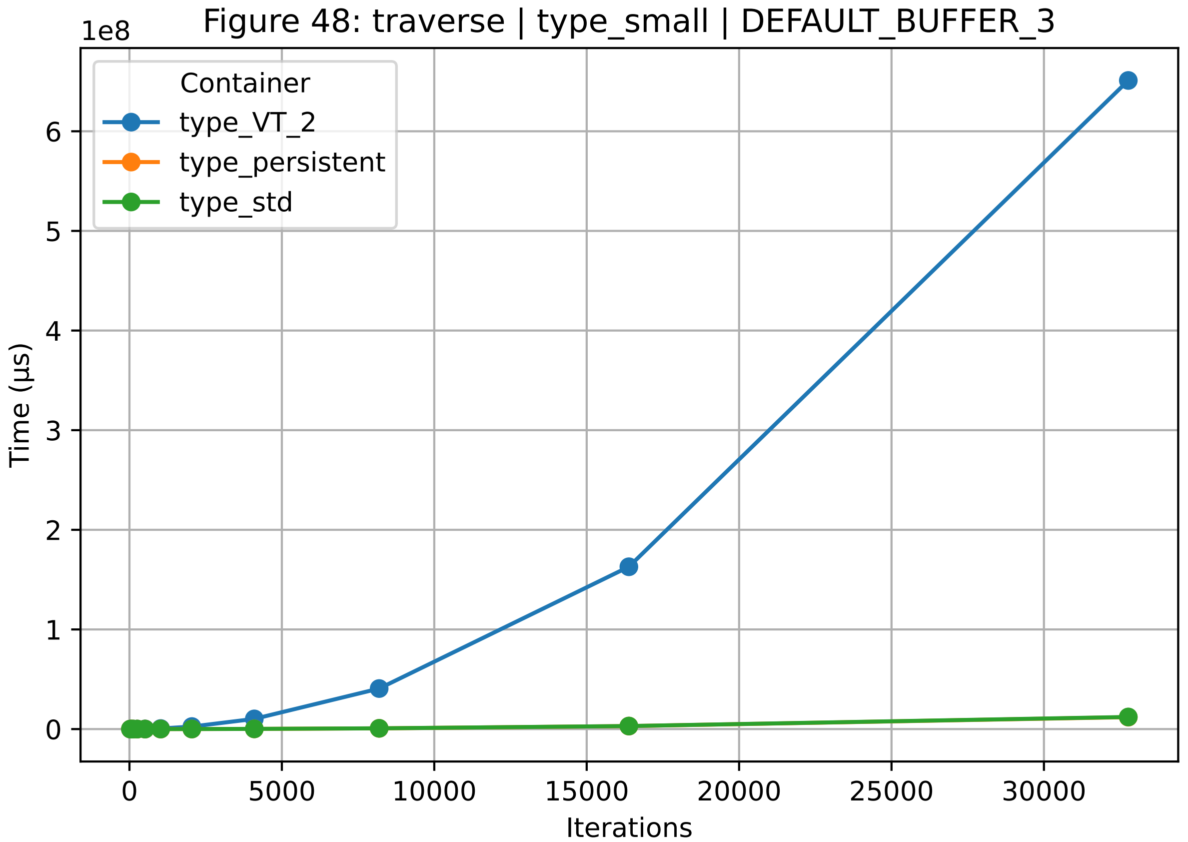


Figure 48: traverse | type_small | DEFAULT_BUFFER_3
1e8
Container
type_VT_2
type_persistent
type_std
0
1
2
3
4
5
6
Time (µs)
0
5000
10000
15000
20000
25000
30000
Iterations