

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

MASTER THESIS

Relaxed Radix Balanced Trees as Immutable Vectors Scala

Author:

Nicolas STUCKI

Supervisor:

Vlad URECHE

*A thesis submitted in fulfilment of the requirements
for the degree of Master in Computer Science*

in the

LAMP
Computer Science

December 2014

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Abstract

School of Computer and Communications
Computer Science

Master in Computer Science

**Relaxed Radix Balanced Trees
as Imutable Vectors Scala**

by Nicolas STUCKI

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Contents

Abstract	i
Contents	ii
List of Figures	iv
List of Tables	vi
Abbreviations	vii
1 Introduction	2
1.1 Main Section 1	2
1.2 Main Section 2	2
2 Vector Structure	4
2.1 Radix Balanced Vectors	4
2.1.1 Tree structure	4
2.1.2 Operations	4
2.1.2.1 Apply	4
2.1.2.2 Updated	4
2.1.2.3 Additions	4
Append	4
Prepend	4
Concatenation and Insert	5
2.1.2.4 Splits	5
2.2 Parallel Vectors	5
2.2.1 Splitter Iterator	5
2.2.2 Combiner Builder	5
2.3 Relaxed Radix Balanced Vectors	5
2.3.1 Relaxed Tree structure	5
2.3.2 Relaxed Operations	6
2.3.2.1 Apply (get element at index)	6
2.3.2.2 Updated	6
2.3.2.3 Additions	6
Append	6
Prepend	6
Concatenation	6

Insert	6
2.3.2.4 Splits	7
3 Implementation and Optimizations	8
3.1 Where is time spent?	8
3.1.1 Arrays	8
3.1.2 Computing indices	8
3.1.3 Abstractions	9
3.2 Displays	9
3.2.1 As cache	9
3.2.2 For transient states	9
3.3 Builder	10
3.4 Iterator	10
3.5 Relaxing the Radix	10
3.5.1 Relaxing Displays	10
3.5.2 Relaxing the Builder	10
3.5.3 Relaxing Iterator	10
4 Performance	11
4.1 In practice: Running on JVM	12
4.1.1 Cost of Abstraction and JIT Inline	12
4.2 Measuring performance	12
4.3 Generators	12
4.4 Benchmarks	12
4.4.1 Apply	12
4.4.2 Concatenation	12
4.4.3 Append	12
4.4.4 Prepend	12
4.4.5 Splits	12
4.4.6 Iterator	12
4.4.7 Builder	12
4.4.8 Parallel split-combine	12
4.4.9 Memory footprint	12
5 Testing	26
5.1 Teststing correctness	26
5.1.1 Invariant Assertions	26
5.1.2 Unit tests	26
5.2 Main Section 2	26
6 Related Work	27
6.1 RRB-Vectors in Clojure	27
7 Conclusions	28

List of Figures

2.1	Radix Balanced Tree Structure	4
2.2	Radix Balanced Tree	5
2.3	Relaxed radix example	5
2.4	Concatenation example with blocks of size 4: Rebalancing level 0	6
2.5	Concatenation example with blocks of size 4: Rebalancing level 1	6
2.6	Concatenation example with blocks of size 4: Rebalancing level 2	7
2.7	Concatenation example with blocks of size 4: Rebalancing level 3	7
3.1	Accessing element at index 526843 in a tree of depth 5. Empty nodes represent collapses subtrees.	8
3.2	Displays	9
3.3	Radix Balanced Tree Transient state	9
3.4	Radix Balanced Tree	10
4.1	Time to execute 10k apply operations on sequential indices.	12
4.2	Time to execute 10k apply operations on random indices.	13
4.3	Time to execute 10k apply operations on sequential indices. Comparing performances for different block sizes and different implementation of the concatenation inner branch rebalancing (Complete/Quick).	13
4.4	Execution time for a concatenation operation on two vectors. In theory (and in practice) Vector concatenation is $O(left+right)$ and the rrbVector concatenation operation is $O(\log_{32}(left + right))$	14
4.5	Time to execute 256 append operations. This shows the amortized cost of the append operation.	15
4.6	Time to execute 256 append operations. This shows the amortized cost of the append operation. Comparing performances for different block sizes and different implementation of the concatenation inner branch rebalancing (Complete/Quick).	16
4.7	Time to execute 256 prepend operations. This shows the amortized cost of the prepend operation.	17
4.8	Time to execute 256 prepend operations. This shows the amortized cost of the append operation. Comparing performances for different block sizes and different implementation of the concatenation inner branch rebalancing (Complete/Quick).	18
4.9	Execution time of take and drop.	19
4.10	Excecution time to iterate through all the elements of the vector.	20
4.11	Excecution time to iterate through all the elements of the vector. Comparing performances for different block sizes and different implementation of the concatenation inner branch rebalancing (Complete/Quick).	21

4.12	Execution time to build a vector of a given size.	22
4.13	Execution time to build a vector of a given size. Comparing performances for different block sizes.	23
4.14	Benchmark on map and parallel map using the function $(x \Rightarrow x)$ to show the difference time used in the framework. This time represents the time spent in the splitters and combiners of the parallel collection (iterator and builder for the sequential version).	23
4.15	Benchmark on map and parallel map using the function $(x \Rightarrow x)$ to show the difference time used in the framework. This time represents the time spent in the splitters and combiners of the parallel collection.	24
4.16	Memory Footprint	25

List of Tables

Abbreviations

JIT	J ust I n T ime
RB	R adix B alanced
RRB	R elaxed R adix B alanced

I

Chapter 1

Introduction

1.1 Main Section 1

1.2 Main Section 2

I

Chapter 2

Vector Structure

2.1 Radix Balanced Vectors

2.1.1 Tree structure

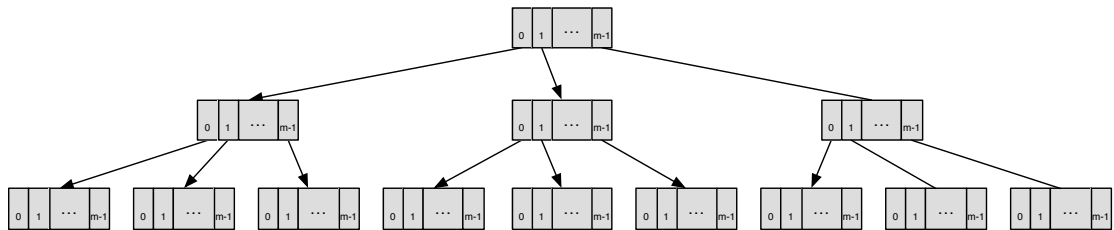


FIGURE 2.1: Radix Balanced Tree Structure

2.1.2 Operations

2.1.2.1 Apply

2.1.2.2 Updated

2.1.2.3 Additions

Append

Prepend

Concatenation and Insert

2.1.2.4 Splits

2.2 Parallel Vectors

2.2.1 Splitter Iterator

2.2.2 Combiner Builder

2.3 Relaxed Radix Balanced Vectors

2.3.1 Relaxed Tree structure

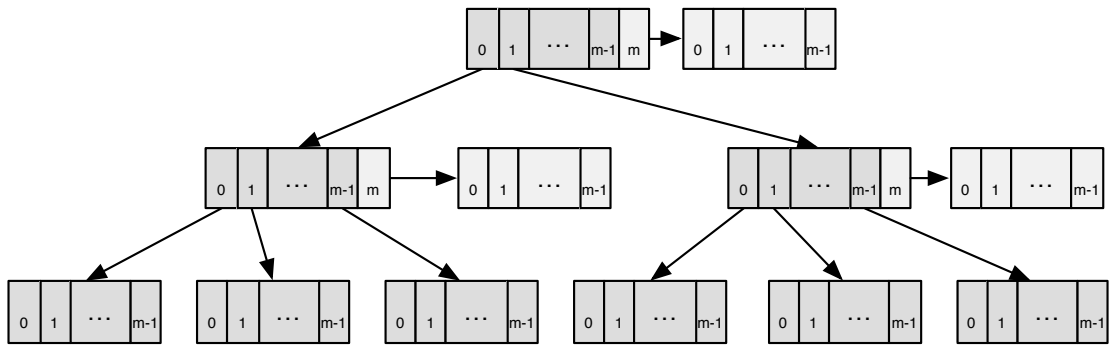


FIGURE 2.2: Radix Balanced Tree

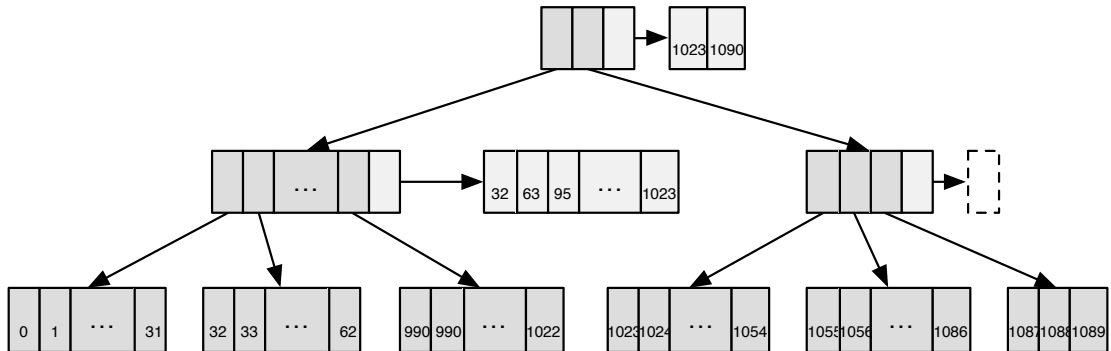


FIGURE 2.3: Relaxed radix example

2.3.2 Relaxed Operations

2.3.2.1 Apply (get element at index)

2.3.2.2 Updated

2.3.2.3 Additions

Append

Prepend

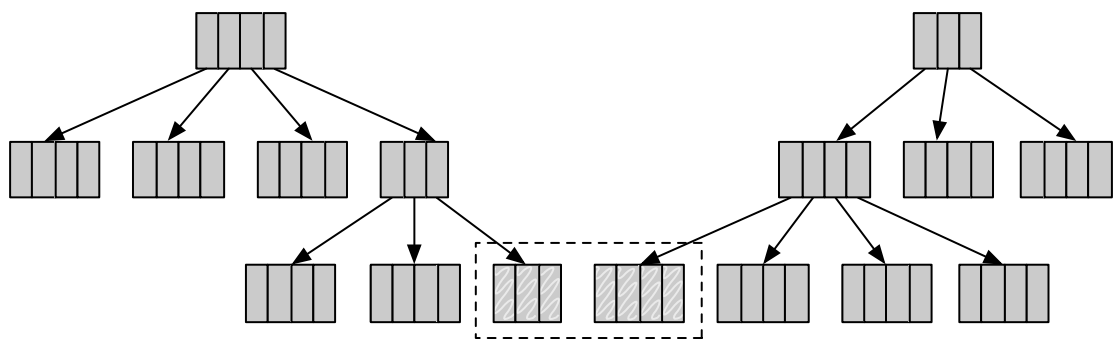


FIGURE 2.4: Concatenation example with blocks of size 4: Rebalancing level 0

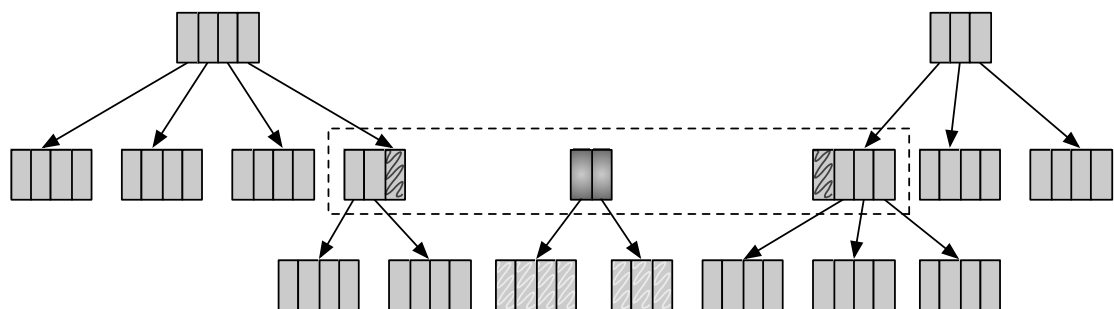


FIGURE 2.5: Concatenation example with blocks of size 4: Rebalancing level 1

Concatenation

Insert

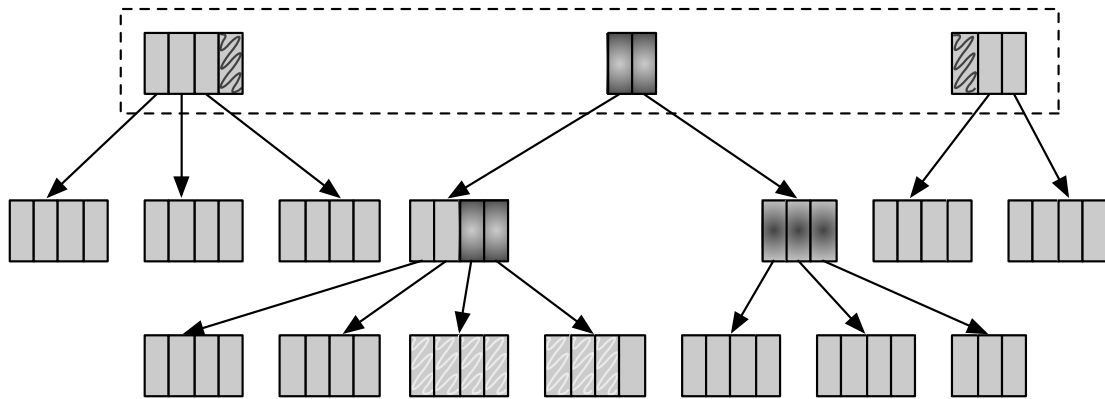


FIGURE 2.6: Concatenation example with blocks of size 4: Rebalancing level 2

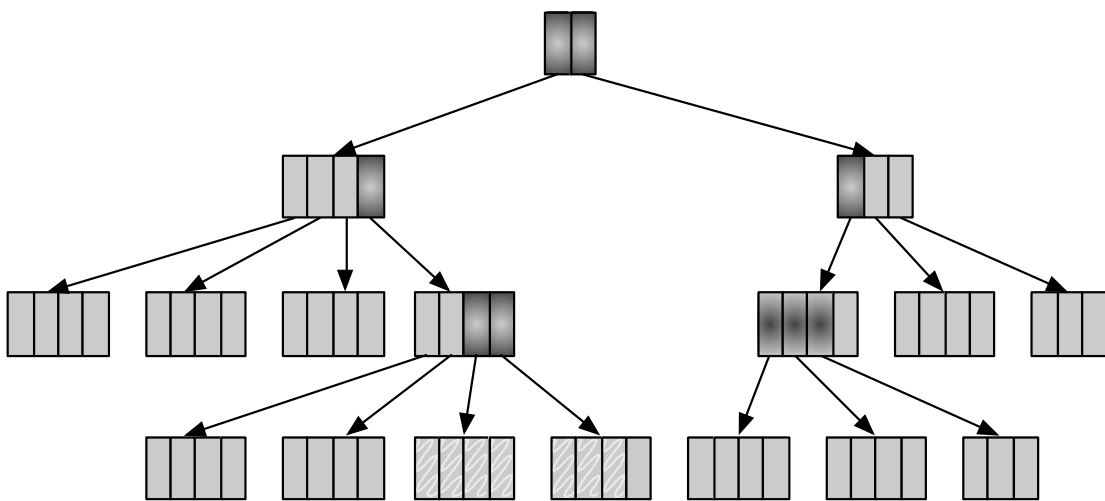


FIGURE 2.7: Concatenation example with blocks of size 4: Rebalancing level 3

2.3.2.4 Splits

I

Chapter 3

Implementation and Optimizations

3.1 Where is time spent?

3.1.1 Arrays

3.1.2 Computing indices

$$526843 = \underbrace{00}_{0} \underbrace{000000}_{0} \underbrace{10000}_{16} \underbrace{00010}_{2} \underbrace{01111}_{15} \underbrace{11011}_{27}$$

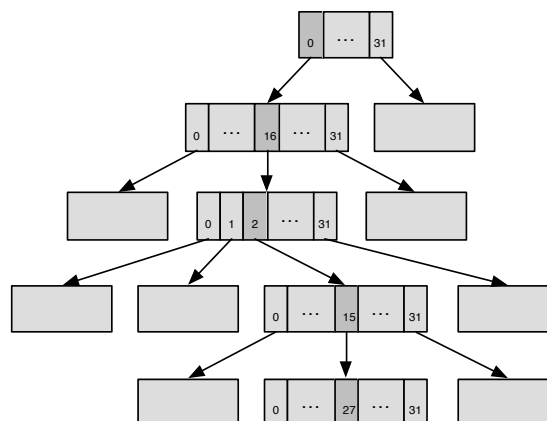


FIGURE 3.1: Accessing element at index 526843 in a tree of depth 5. Empty nodes represent collapses subtrees.

3.1.3 Abstractions

3.2 Displays

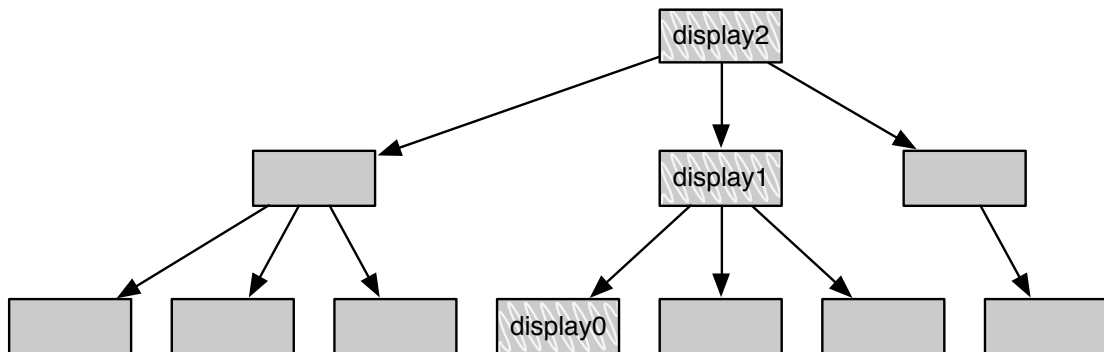


FIGURE 3.2: Displays

3.2.1 As cache

3.2.2 For transient states

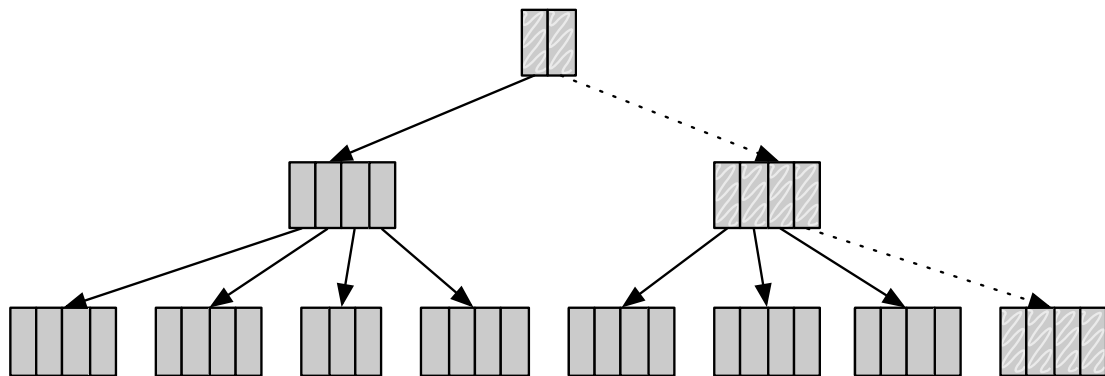


FIGURE 3.3: Radix Balanced Tree Transient state

3.3 Builder

3.4 Iterator

3.5 Relaxing the Radix

3.5.1 Relaxing Displays

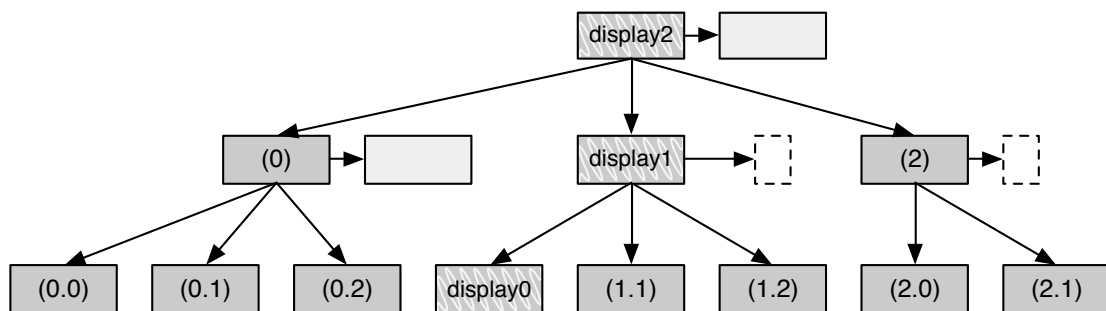


FIGURE 3.4: Radix Balanced Tree

3.5.2 Relaxing the Builder

3.5.3 Relaxing Iterator

I

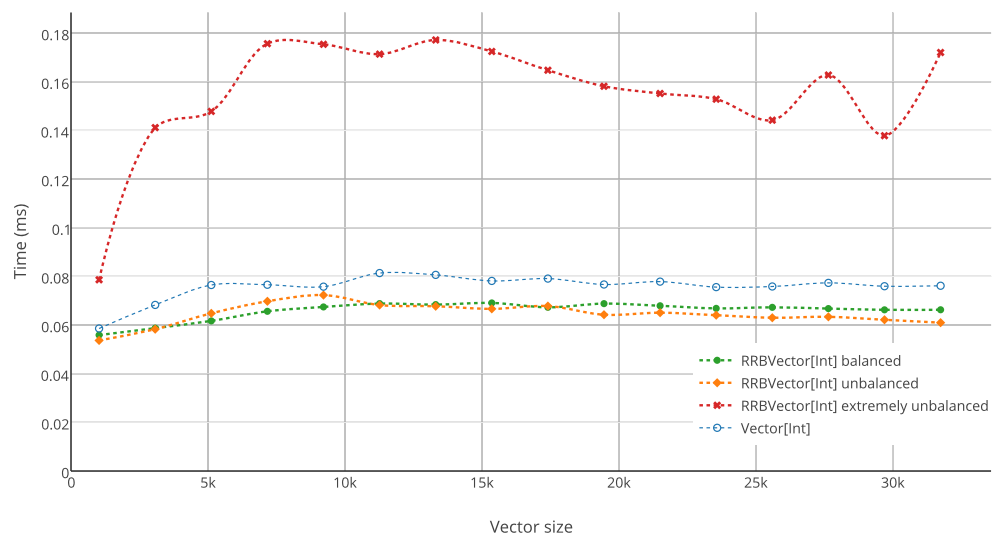
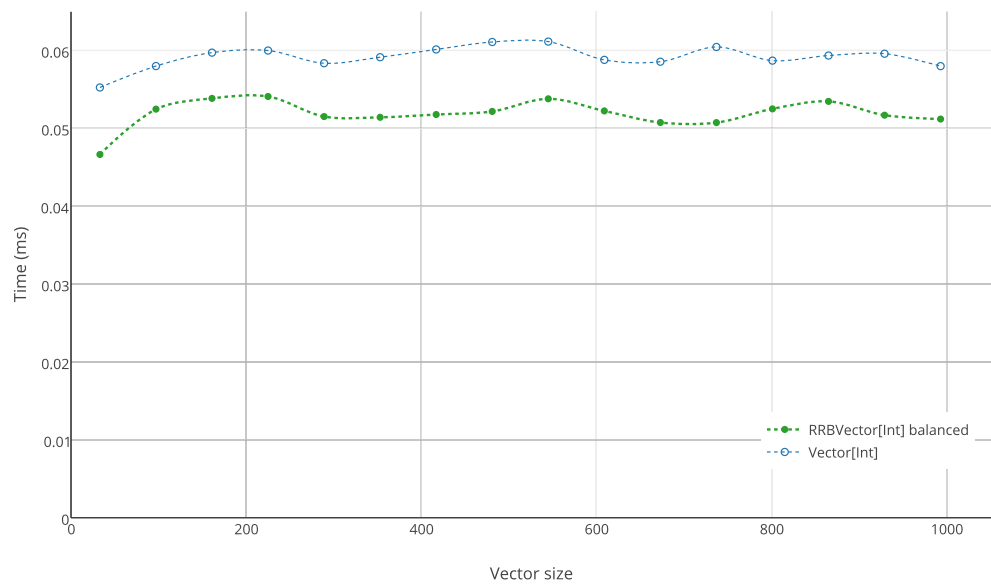


FIGURE 4.1: Time to execute 10k apply operations on sequential indices.

Chapter 4

Performance

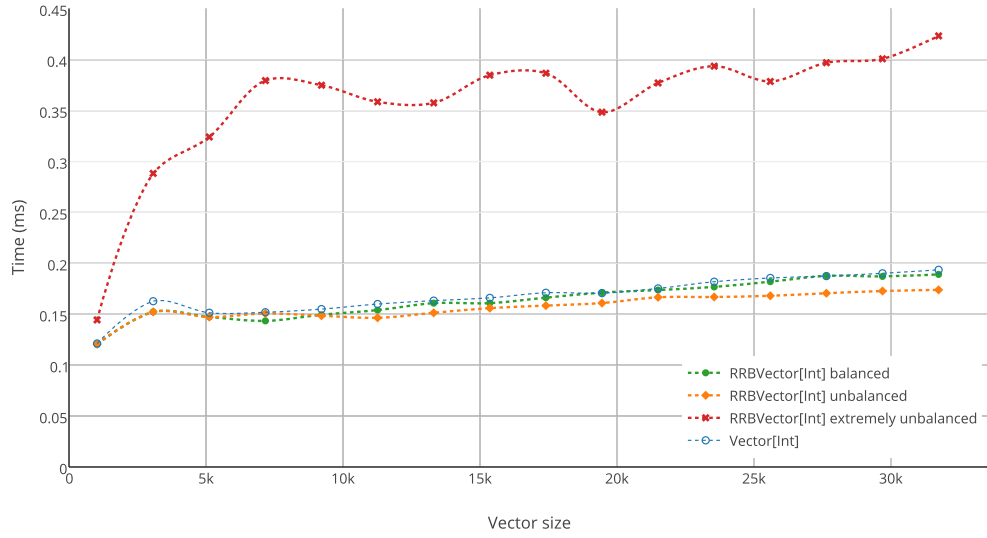


FIGURE 4.2: Time to execute 10k apply operations on random indices.

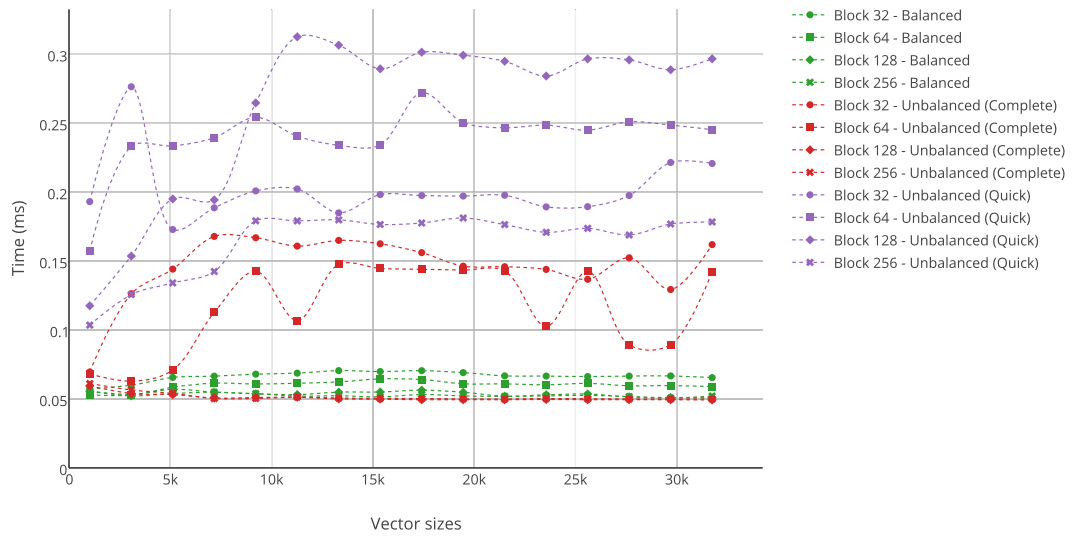


FIGURE 4.3: Time to execute 10k apply operations on sequential indices. Comparing performances for different block sizes and different implementation of the concatenation inner branch rebalancing (Complete/Quick).

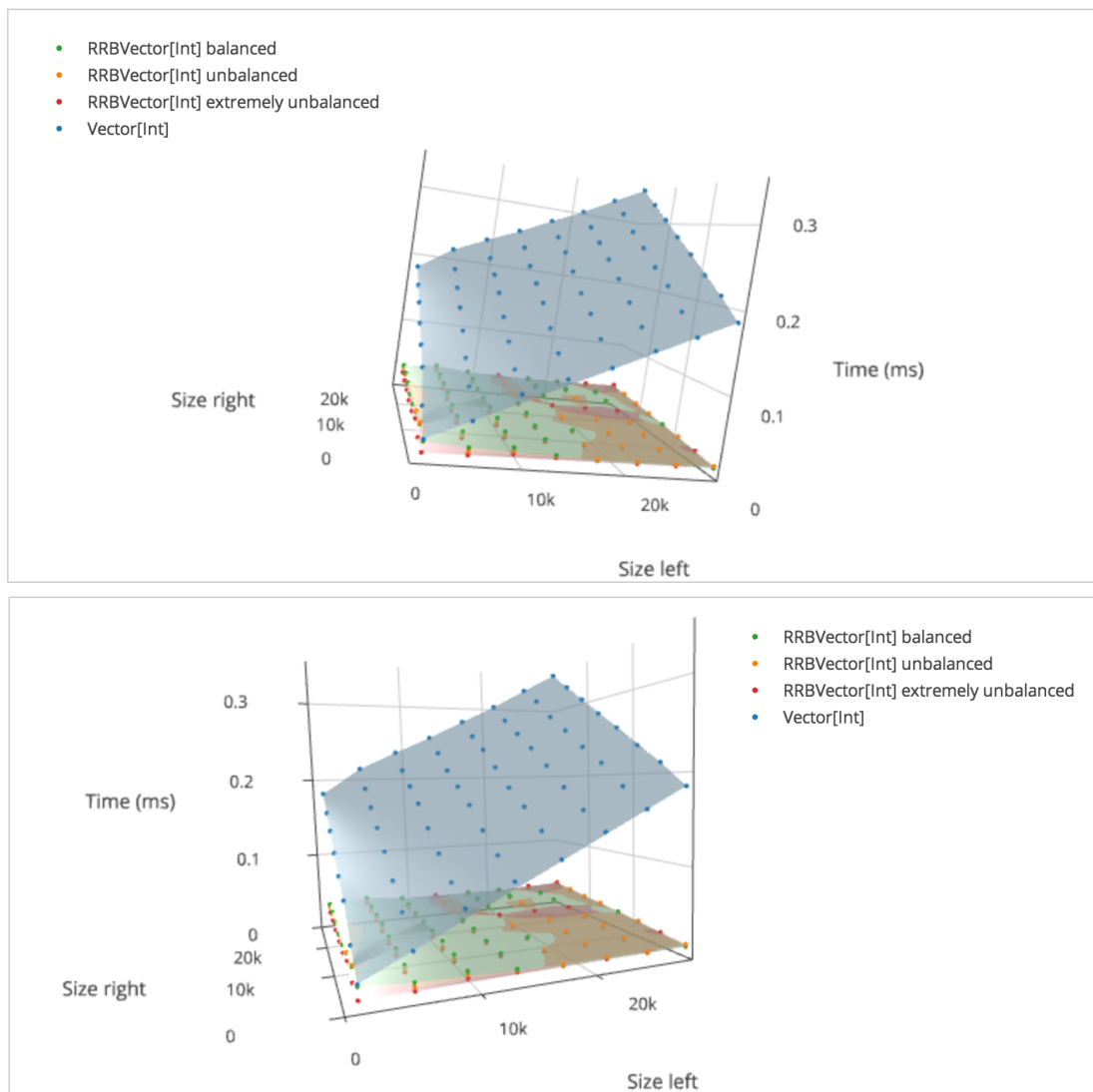


FIGURE 4.4: Execution time for a concatenation operation on two vectors. In theory (and in practice) Vector concatenation is $O(left + right)$ and the rrbVector concatenation operation is $O(\log_{32}(left + right))$.

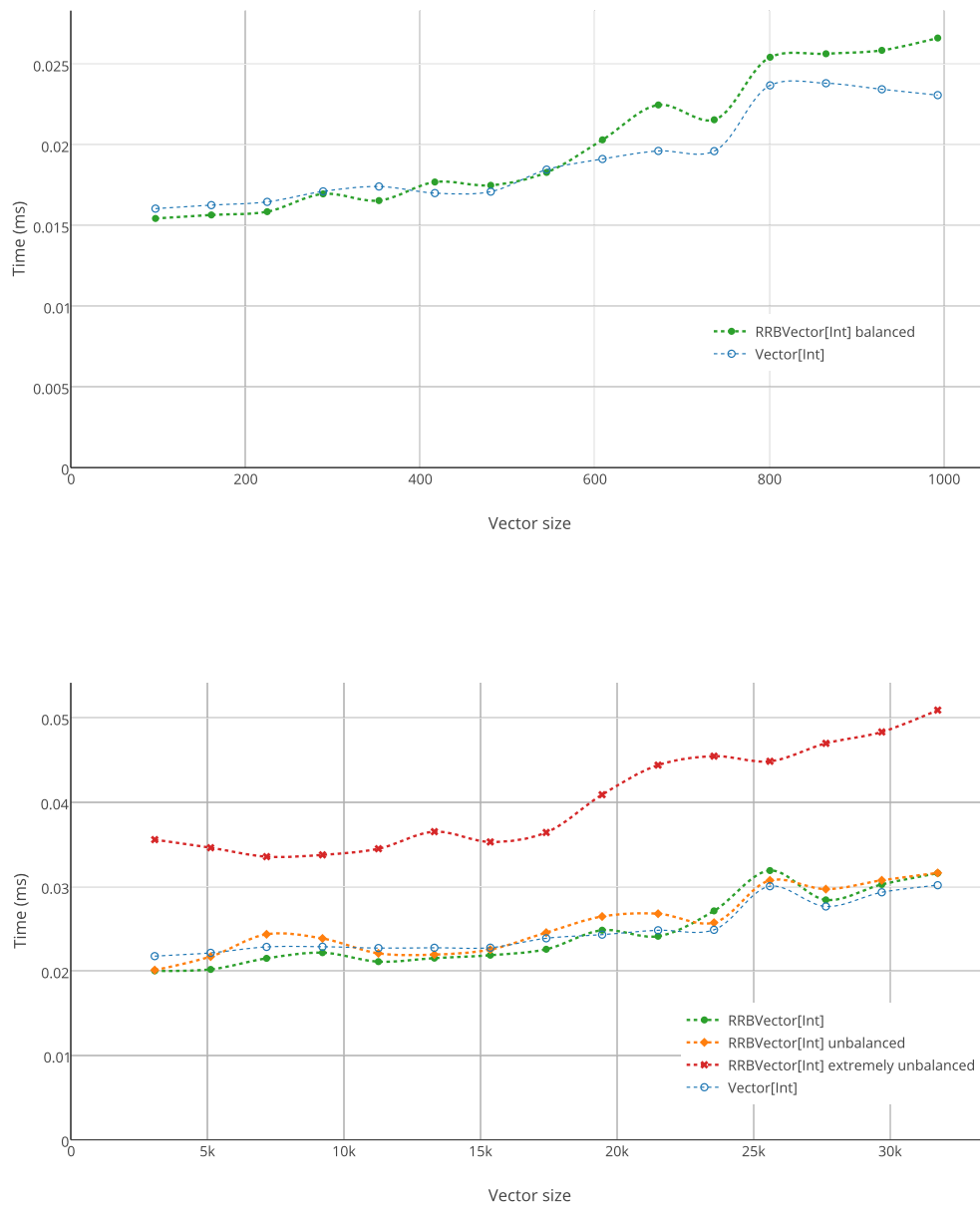


FIGURE 4.5: Time to execute 256 append operations. This shows the amortized cost of the append operation.

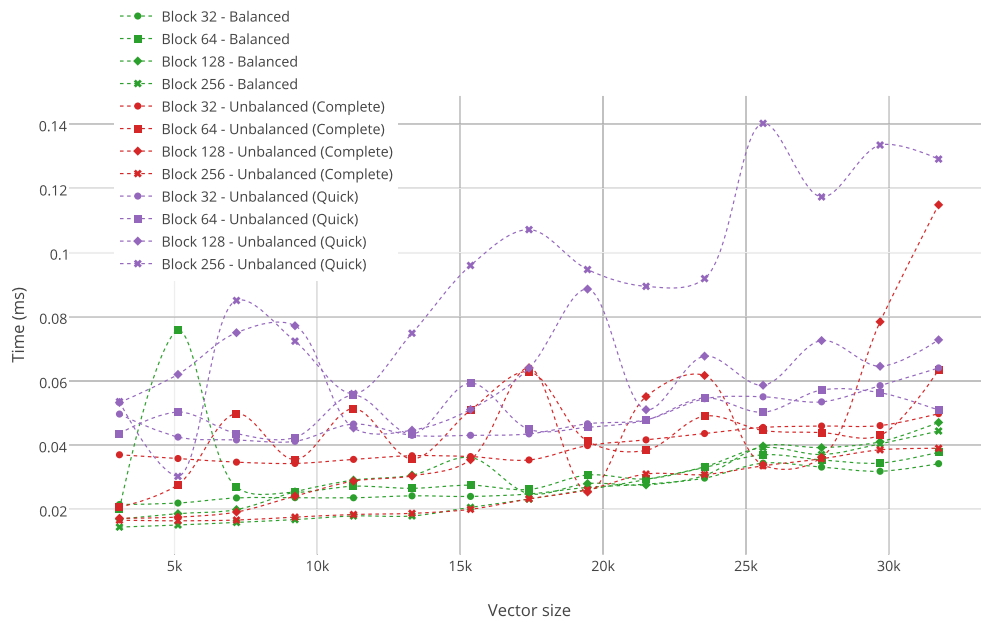


FIGURE 4.6: Time to execute 256 append operations. This shows the amortized cost of the append operation. Comparing performances for different block sizes and different implementation of the concatenation inner branch rebalancing (Complete/Quick).

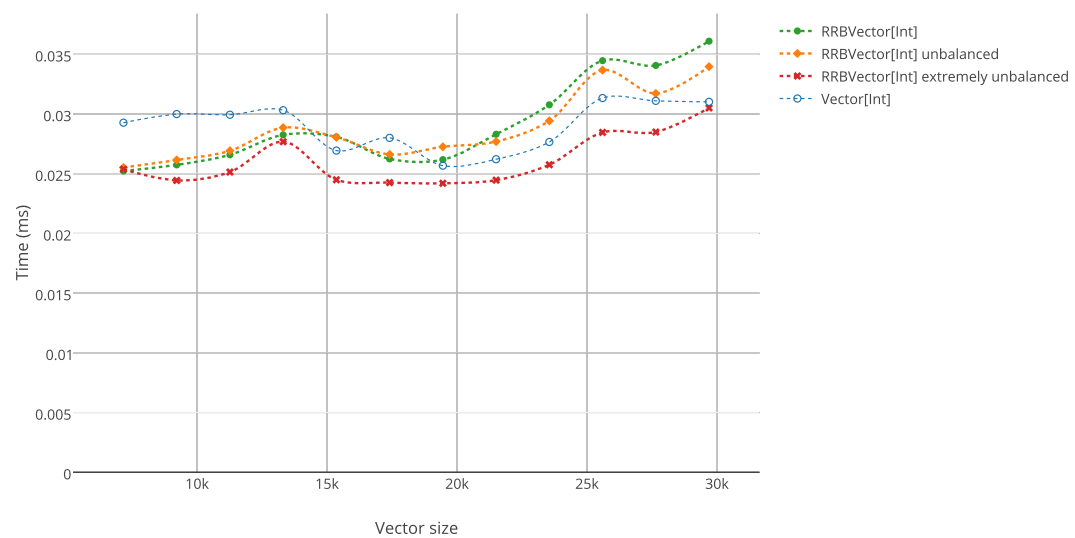
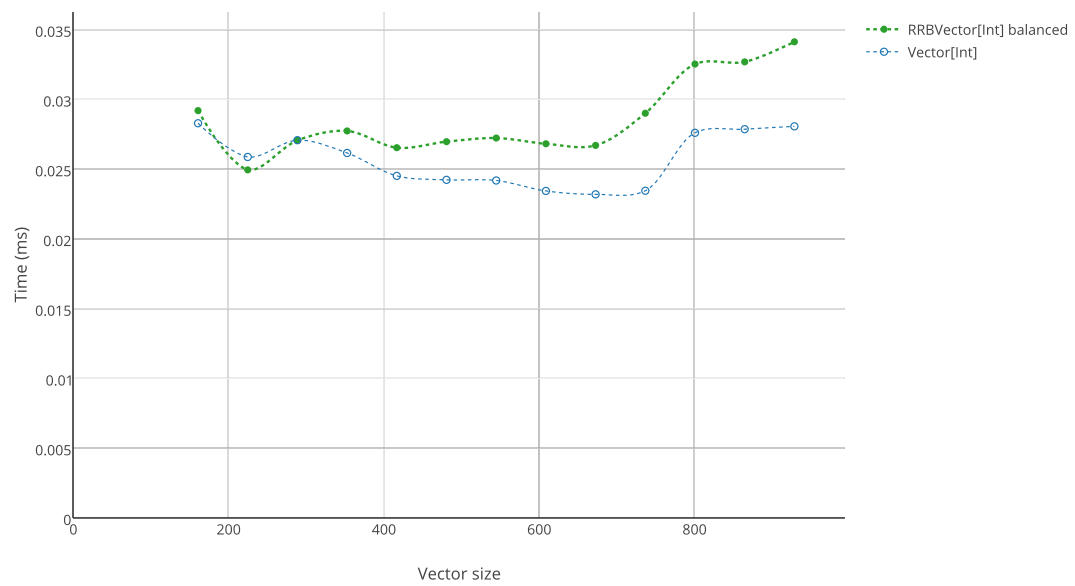


FIGURE 4.7: Time to execute 256 prepend operations. This shows the amortized cost of the prepend operation.

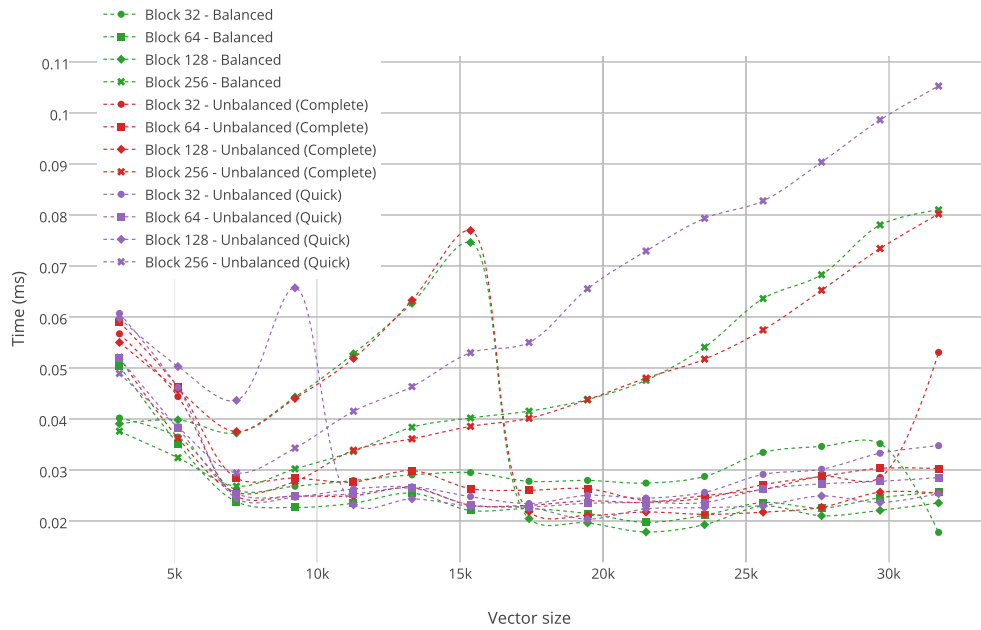


FIGURE 4.8: Time to execute 256 prepend operations. This shows the amortized cost of the append operation. Comparing performances for different block sizes and different implementation of the concatenation inner branch rebalancing (Complete/Quick).

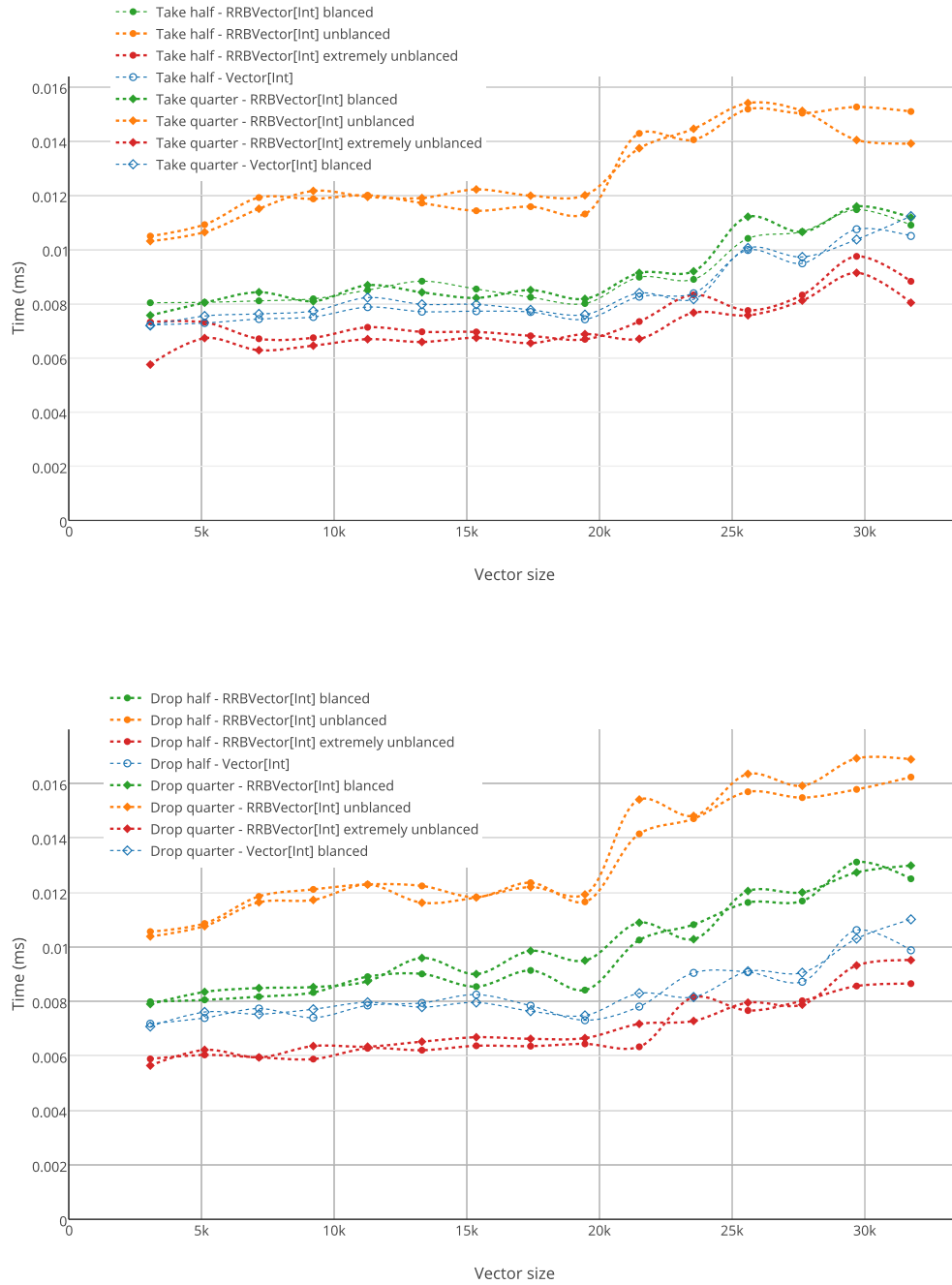


FIGURE 4.9: Execution time of take and drop.

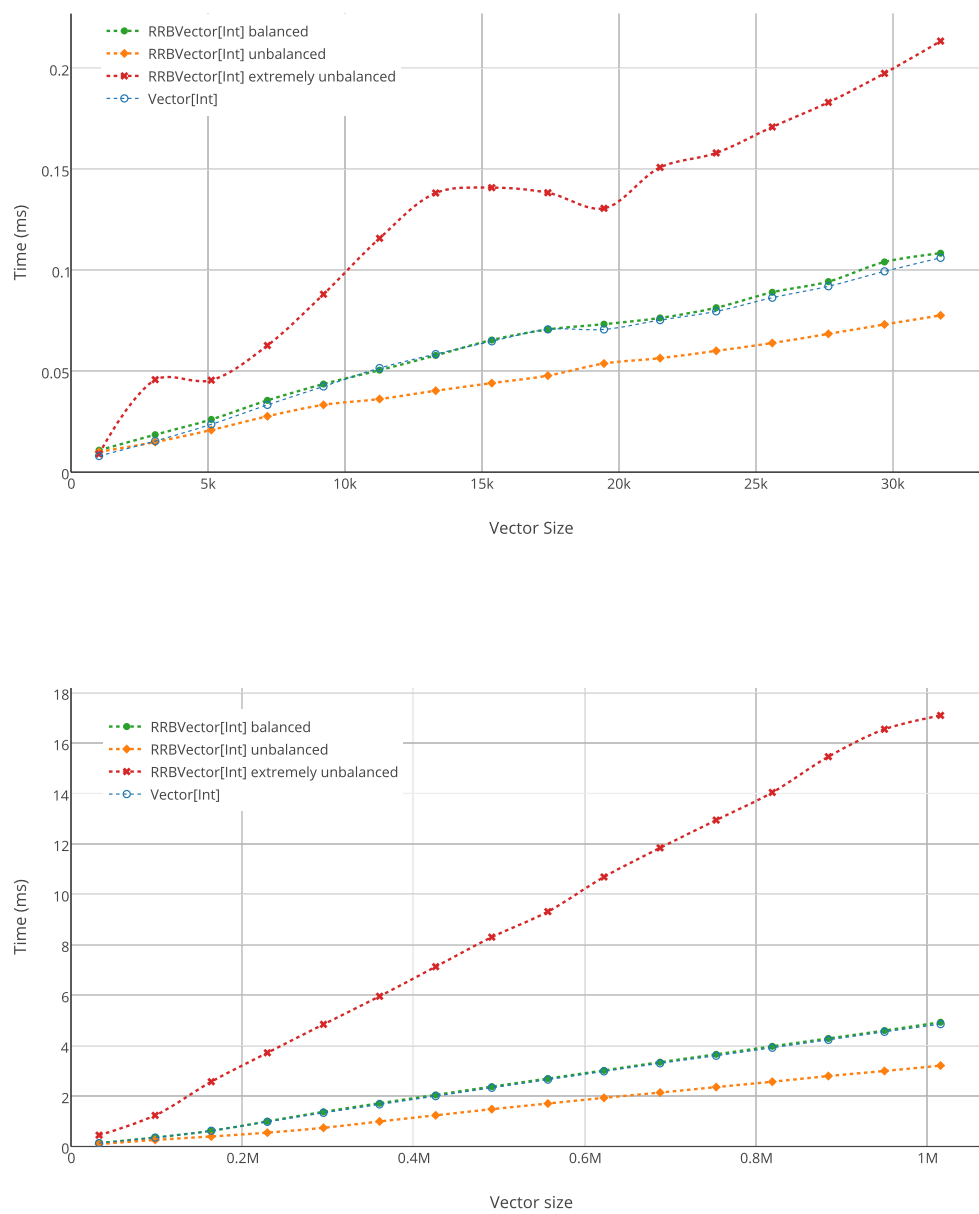


FIGURE 4.10: Execution time to iterate through all the elements of the vector.

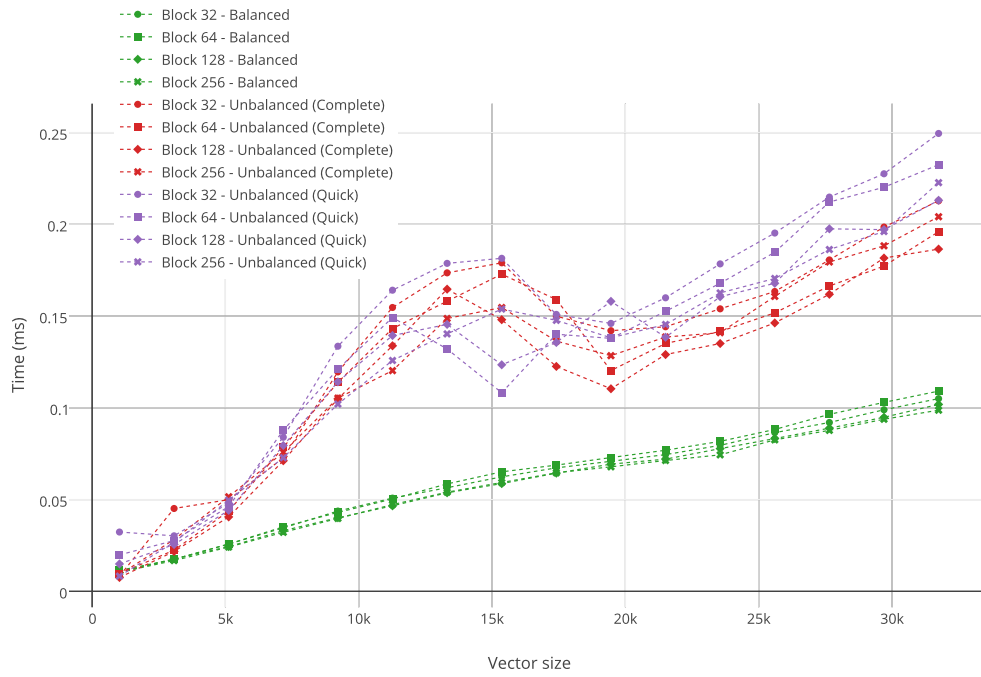


FIGURE 4.11: Execution time to iterate through all the elements of the vector. Comparing performances for different block sizes and different implementation of the concatenation inner branch rebalancing (Complete/Quick).

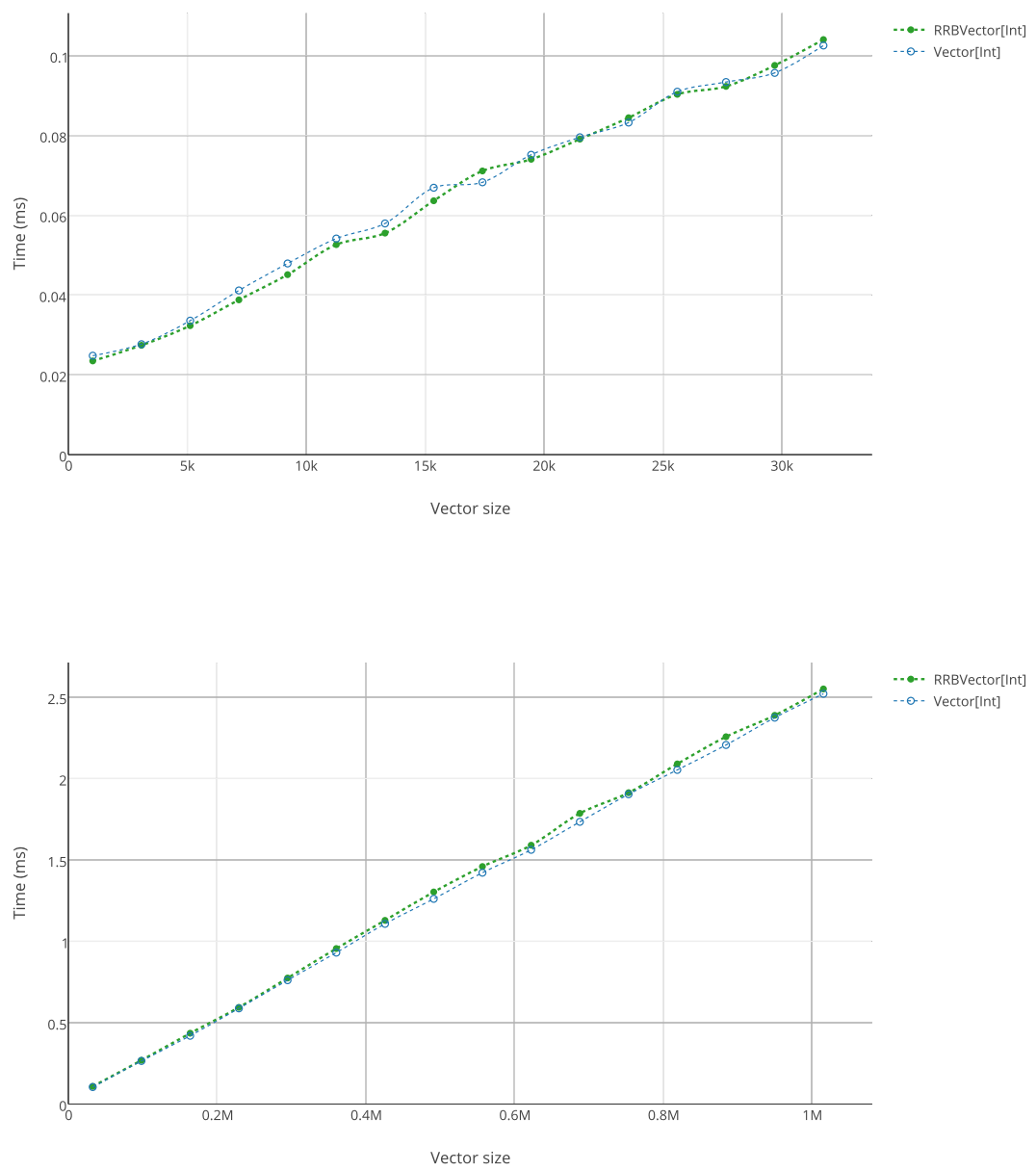


FIGURE 4.12: Execution time to build a vector of a given size.

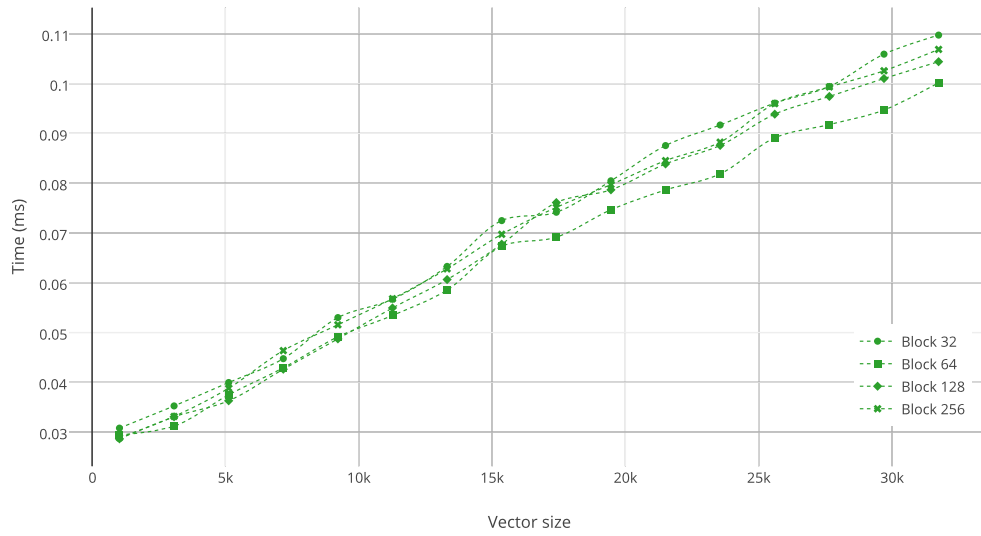


FIGURE 4.13: Execution time to build a vector of a given size. Comparing performances for different block sizes.

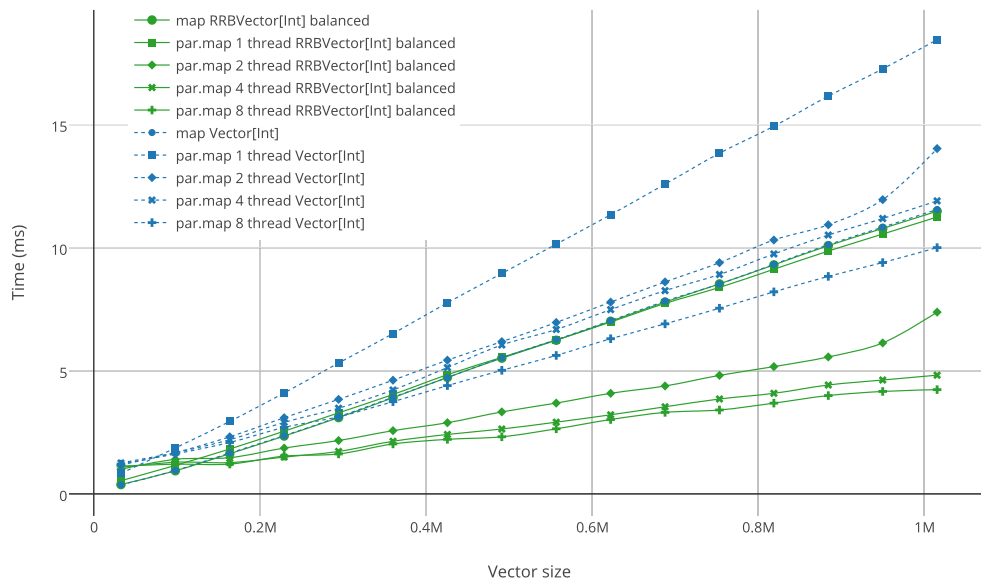


FIGURE 4.14: Benchmark on map and parallel map using the function $(x \Rightarrow x)$ to show the difference time used in the framework. This time represents the time spent in the splitters and combiners of the parallel collection (iterator and builder for the sequential version).

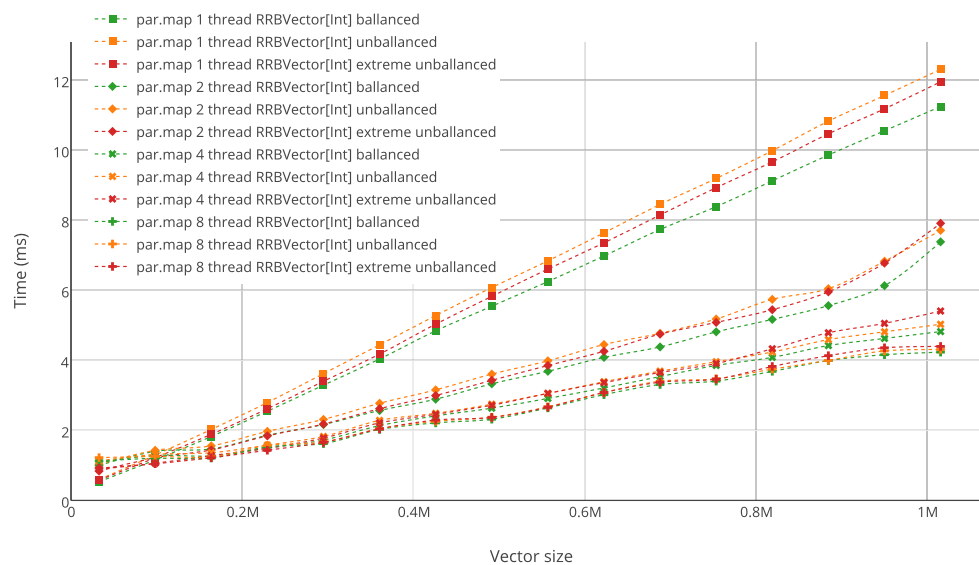


FIGURE 4.15: Benchmark on map and parallel map using the function $(x \Rightarrow x)$ to show the difference time used in the framework. This time represents the time spent in the splitters and combiners of the parallel collection.

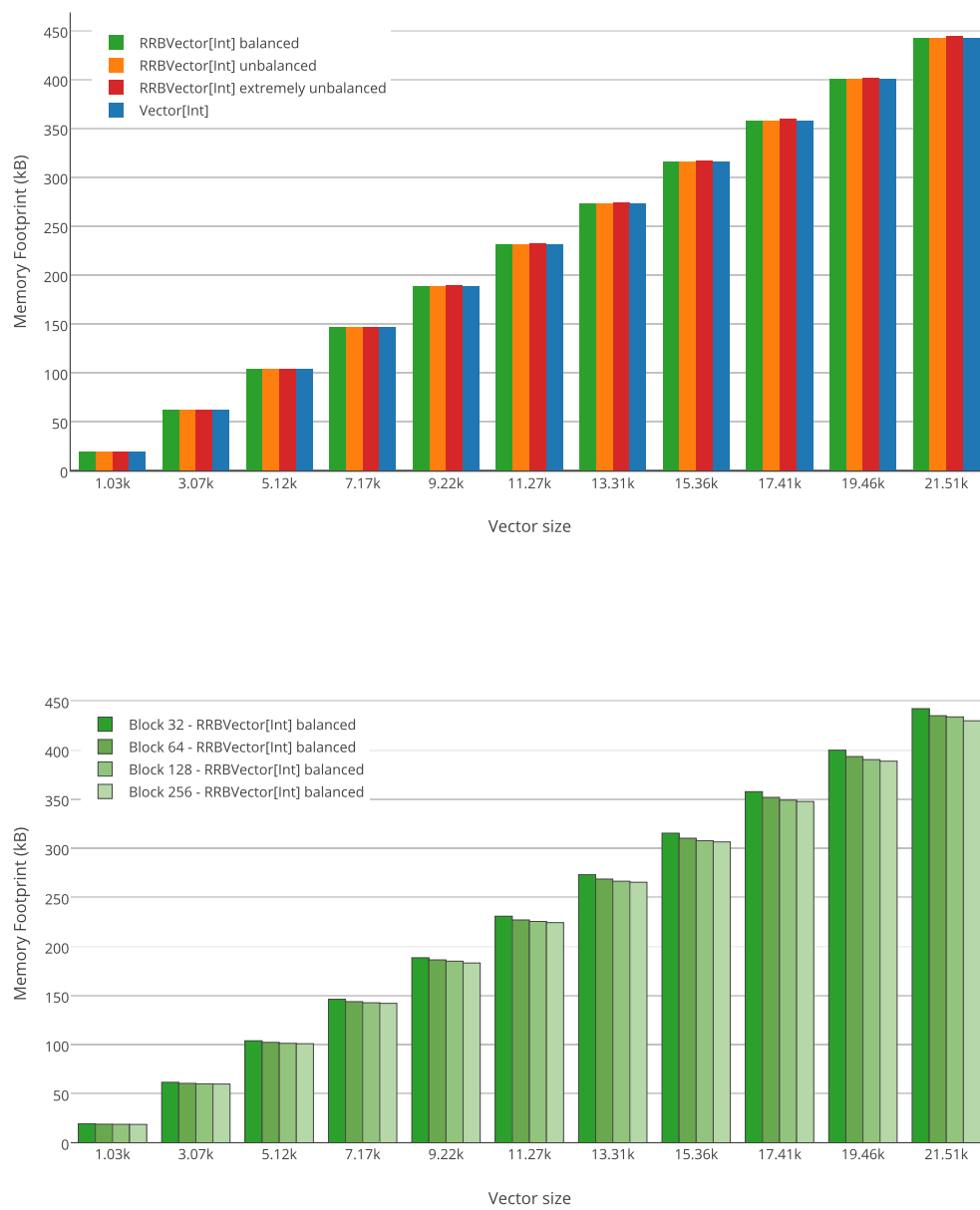


FIGURE 4.16: Memory Footprint

Chapter 5

Testing

5.1 Teststing correctness

5.1.1 Unit tests

5.1.2 Invariant Assertions

I

Chapter 6

Related Work

6.1 RRB-Vectors in Clojure

I

Chapter 7

Conclusions