SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis, . . . in Informatics

Hiding Cache-Misses with Coroutines across different Hardware Architectures

Daniel Gruber

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis, . . . in Informatics

Hiding Cache-Misses with Coroutines across different Hardware Architectures

Verbergen von Cache-Misses mittels Coroutinen auf unterschiedlichen Hardwarearchitekturen

Author:	Daniel Gruber
Examiner:	Alexander Beischl
Supervisor:	Prof. Dr. Thomas Neumann
Submission Date:	22.12.2025

I confirm that this master's thesis, ... is my own work and I have documented all sources and material used.

Munich, 22.12.2025 Daniel Gruber

Acknowledgments

Abstract

Contents

Acknowledgments **iv**

Abstract **v**

1 Introduction **1**

1.1 Section 1

2 Cache Misses, Coroutines and Data Structures Fundamentals **2**

2.1 Coroutine CPP Reference standardisation history 2

2.1.1 Coroutine is a function 2

2.1.2 The co_await, co_return and co_yield operators 2

2.1.3 Example of Coroutines 3

2.1.4 Simple generator (co_yield) example 3

2.1.5 Tricks and Pitfalls 3

2.2 Computer Architecture, Cache Misses 3

2.2.1 Different Computer Architectures, x86-64, amd, apple, mips . . . 3

2.2.2 Introduction to computer achitecture and storage layout 3

2.2.3 What are cache misses? 3

2.2.4 When can they happen? 3

2.2.5 Why are they relevant in data bases and different index structures 3

2.3 Data Structures 3

2.3.1 Hashtable 3

2.3.2 B+ Tree 3

3 Using Coroutines in different DataStructures for hiding cache-misses **4**

3.1 Different coroutine approaches and their trade-offs 4

3.2 Coroutines in chaining Hashtable 4

3.2.1 Implementation 4

3.2.2 Benchmarking and Measurements 4

3.2.3 Results 4

3.3 Coroutines in linear probing hashtable 4

3.3.1 Implementation 4

3.3.2 Measurements 4

3.3.3 Results 4
3.4 Coroutines in B+ Tree 4
3.4.1 Implementation 4
3.4.2 Measurements 4
3.4.3 Results 4

4 Integration of Coroutines in DuckDB 5
4.1 DuckDB Introduction 5
4.2 Understanding the implemented linear Hashtable 5
4.3 Integration of Coroutines in DuckDB 5
4.4 Results of timings 5

5 Evaluation 6
5.1 Coroutine Approaches evaluation 6
5.2 Coroutine in different Data structures evaluation 6
5.3 Coroutine in DuckDB evaluation 6
5.4 Summarized evaluation of coroutines in C++ for hiding cache misses . 6

6 Conclusion 7

Abbreviations 8

List of Figures 9

List of Tables 10

Bibliography 11

1 Introduction

1.1 Section

Citation test [1].

How can Coroutines hide latencies for hashtable/bptree lookups? What is the performance gain generally? What is the performance gain in duckdb?

How does the performance gain differ between various architectures? (Intel x86 vs ARM vs NUMA) What about Threading and Coroutines? Hyperthreading/NUMA

2 Cache Misses, Coroutines and Data Structures Fundamentals

2.1 Coroutine CPP Reference standardisation history

Citation test [1].

2.1.1 Coroutine is a function

A C++ Coroutine is a generalization of a function. Additionally to normal functions, coroutines can suspend execution to be resumed later. Coroutines are defined with the keywords co_await, co_yield and co_return.

Whereas a normal function has a single entry point - the Call operation - and a single exit point - the Return operation -, a coroutine can be suspended and resumed at multiple points in its execution.

When a coroutine is called, it does not execute its body immediately. Instead, it returns a special object known as a coroutine handle or promise object. This object represents the state of the coroutine and allows the caller to control its execution.

and examples from Andreas Fertig highlghitng different concepts with co_yield and co_await

2.1.2 The co_await, co_return and co_yield operators

co_await

awaitable (and awaiter) concept and promiseType

co_return

co_yield

2.1.3 Example of Coroutines

Simple co_await example

2.1.4 Simple generator (co_yield) example

Advanced Example

2.1.5 Tricks and Pitfalls

2.2 Computer Architecture, Cache Misses

2.2.1 Different Computer Architectures, x86-64, amd, apple, mips

2.2.2 Introduction to computer achitecture and storage layout

2.2.3 What are cache misses?

2.2.4 When can they happen?

2.2.5 Why are they relevant in data bases and different index structures

2.3 Data Structures

2.3.1 Hashtable

Chaining vs Open Addressing (Linear Probing)

2.3.2 B+ Tree

3 Using Coroutines in different DataStructures for hiding cache-misses

3.1 Different coroutine approaches and their trade-offs

Citation test [1].

First approach was simple: create a coroutine each time from s cratch for lookup. Bad performance, re-use coroutines so the creatio overhead is minimal and maybe create 5-20 coroutines and reuse them for all the lookups. First with co_yield concept and caller has to resume the coroutine, but for caller it is complex to distinguish if

3.2 Coroutines in chaining Hashtable

3.2.1 Implementation

3.2.2 Benchmarking and Measurements

3.2.3 Results

3.3 Coroutines in linear probing hashtable

3.3.1 Implementation

3.3.2 Measurements

3.3.3 Results

3.4 Coroutines in B+ Tree

3.4.1 Implementation

3.4.2 Measurements

3.4.3 Results

4 Integration of Coroutines in DuckDB

4.1 DuckDB Introduction

4.2 Understanding the implemented linear Hashtable

Citation test [1].

4.3 Integration of Coroutines in DuckDB

DuckDB integration: linear probing hashtabale where if same key it is chained.

Difficulty in understanding the hashtable and implementing coroutines in huge concept of hastable withotu knowing full picture. has to be done in Join phase of the hastable (which is divided in different datachaunks and each datachunk is handled by a thread wher ea thread cna possibly handle more datachunks). In this thread coroutines can easily integrated as they are asynchronoulsy called and ...

4.4 Results of timings

5 Evaluation

5.1 Coroutine Approaches evaluation

5.2 Coroutine in different Data structures evaluation

5.3 Coroutine in DuckDB evaluation

5.4 Summarized evaluation of coroutines in C++ for hiding cache misses

6 Conclusion

Abbreviations

List of Figures

List of Tables

Bibliography

[1] L. Lamport. *LaTeX : A Documentation Preparation System User's Guide and Reference Manual*. Addison-Wesley Professional, 1994.