Benchmarking tree operations

CODAS's ANR Project Deliverable #1

Laure Gonnord, Paul Iannetta, Gabriel Radanne

2021

1 Introduction

In the context of the CODAS's project, we studied the literature on tree algorithms. Inspired by the abundant research on *cache oblivious algorithms* [1], we focused our research programs operating on binary trees (or variant like B-trees) for which crucial operation to optimise are insertion and search. This line of research had already proposed manually-optimised algorithms for cache-locality based on linearised trees (trees stored as arrays), which might be of great inspiration for designing "a polyhedral model on trees". We thus propose two benchmarks to evaluate further optimisations of these operations.

2 OCaml typing phase tree operations

In this benchmark, we propose to evaluate the implementation of dictionaries used as naming environment in a compiler. Indeed, compilers, in particular during name resolution and type-checking, heavily rely on efficiently adding and finding names of functions and variables. To evaluate this use-case, we considered the OCaml type-checker. OCaml [2] is a functional statically-typed language known for its rich type system and efficient compiler. Typing in general and name resolution in particular is a fairly performance-sensitive operation, and the type-checker uses a pointer-based AVL as name environment. An additional challenge is that this environment is used in a *persistent* manner. Since variables respect lexical scoping in OCaml, new variables are registered in a new independent naming environment that is discarded when the scope under consideration is closed.

We instrumented the implementation of the naming environment to log all its operations, so that we can replay them with different tree implementations. Naming environments and names are represented by unique identifiers. We also logged when a scope is closed, to indicate that a particular version of the naming environment is freed. A short excerpt of the log is shown below which demonstrates the type of usage pattern found in this scenario. The `add` and `addl` operations indicate *persistent* insertions of one or multiple values, `find` is a lookup in the tree, and `free` frees the tree. Tree 0 is empty.

```
1223 <- addl(0,kind_478,layout_479) // Tree creation \\
1224 <- add(1223,arr_483) // Tree extension \\
find(1224,arr_483) // Lookup in tree 1224 \\
find(1224,c_init_460) \\
... \\
free(1224) // Tree 1224 is freed \\
find(1223,create_430) // Lookup in tree 1223
```

We created the log of operations done by the naming environment when compiling the OCaml standard library. The result is 125 files obtained in less than a minute, in average each file contains 7300 operations on trees.

3 Databases tree operations

Key-value databases are big dictionaries which are generally implemented using variants of B-trees. To evaluate tree operations in a similar context, we used the scenario described by the Influxdb team [1] to compare several key-value stores such as LevelDB (which uses Log Merge Trees) and LMDB (which uses B+-Trees). There is no real issue in comparing our future implementation performance against their (highly fine-tuned) implementation, but we can still use a typical scenario such that :

1. Insert N random keys in a fresh database ;
2. Delete $N/2$ random keys ;
3. Compress the database ;
4. Read $N/2$ random keys ;
5. Insert $N/2$ random keys.

Typically, we can conduct experiments with $N = 100$K or even $N = 10$M in reasonable timings.

4 Associated repository

The extraction scripts as well as their resulting benchmarks can be found at the following url :

https://gitlab.inria.fr/paiannet/calv

Références

[1] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *40th Annual Symposium on Foundations of Computer Science*, pages 285–297, 1999.

[2] Xavier Leroy, Damien Doligez, Alain Frisch, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon. *The OCaml system release 4.12, Documentation and user's manual*. Projet Gallium, INRIA, April 2021.

1. https://www.influxdata.com/blog/benchmarking-leveldb-vs-rocksdb-vs-hyperleveldb-vs-lmdb-performance-for-influxdb/