**Multi-Core Architecture and Systems (0368-4183)** Lecturer: Adam Morrison

# MLPIndex-Based Range Tree

## 1 Introduction

MLP-Index [2] is an algorithm similar to the Cuckoo Trie [3] algorithm (taught in Lecture 2), but it is specialized to 8-byte keys and doesn't support multi-threaded access. As a result, it's not as general or memory efficient as the Cuckoo Trie, but it's faster on 8-byte keys (see Figure 12 in the Cuckoo Trie paper).

The Linux kernel (and other systems) make use of "range trees" that store non-overlapping ranges. There are various use cases for such trees, for example, to store the virtual memory ranges of a process. (There's some additional discussion here.) Linux has a data structure called the Maple Tree that can be used for this purpose. The Maple Tree supports concurrent readers and writers, but writers must be serialized with a lock.

The goal of this project is to explore if MLP-Index can be used to construct a faster range tree than the Maple Tree.

## 2 Project description

**Two-person groups** Your first goal is to add concurrency support to MLPIndex, using the same concurrency model as the Maple Tree. That is, you need to support multiple optimistic readers (without locking), but can assume that at any time, there is at most one writer. Your additions should minimally impact the speed of MLPIndex operations. (The code of MLPIndex is publicly available, see below.)

Your second goal is to implement a range tree based on the concurrent MLPIndex.

Finally, you need to compare your algorithm to the Maple Tree. You can extract the Maple Tree's code from the Linux kernel and benchmark it in userspace, or you can integrate your MLPIndex code into Linux (however, this is probably much harder).

**Three-person groups** You need to complete the tasks of a two-person project, but in addition, implement the `remove()` operation in MLP Index. (The public code only has `insert()` and `lookup()`.)

## 3 Benchmarking

You can get ideas for how to benchmark range trees from the paper on range locks by Kogan et al. [1].

To run the benchmarks, you will be given access to a multicore server in the TAU lab. You need to run the final experiments there, so that you can test with a large number of threads.

For the final experiments, don't just run the benchmark once. This is vulnerable to measurement noise. Benchmarks should be run multiple times (5–10) and average results collected.

## 4 What to submit

**Report** Submit a report (PDF) describing what you did. The report should include at least the following:

- Background: what you learned about MLPIndex, Maple Tree, range trees, etc.
- Challenges and solutions: what problems you encountered and how you solved them. In the minimum, this will describe your MLPIndex-based range tree design.
- Evaluation: As described above. This should include graphs (performance as a function of the number of threads) and some analysis of the results.

**Code** Submit your code. It must include a README that describes how to re-run your experiments and produce the graphs in your report. (So include any scripts you use for that purpose, for example.)

## 5 Resources

- MLPIndex code: `https://github.com/sillycross/mlpds`.
- Maple Tree code: `https://elixir.bootlin.com/linux/latest/source/lib/maple_tree.c`

## References

[1] Alex Kogan, Dave Dice, and Shady Issa. Scalable range locks for scalable address spaces and beyond. In *EuroSys*, 2020.

[2] Haoran Xu. Efficient Data Structures via Memory Level Parallelism, 2018.

[3] Adar Zeitak and Adam Morrison. Cuckoo trie: Exploiting memory-level parallelism for efficient dram indexing. In *SOSP*, 2021.