An Efficient and Safe B+ Tree for Virtual Memory Management

João Vítor V. Coelho[1], **Samuel Xavier-de-Souza**[1], **Wedson Almeida Filho** [1]

[1]Departamento de Engenharia de Computação e Automação
Universidade Federal do Rio Grande do Norte (UFRN)
Caixa Postal 1524 – 59.078-900 – Natal – RN – Brazil

joao.vitor.venceslau.701@ufrn.edu.br
samuel.xavier@ufrn.br, wedson.almeida@dca.ufrn.br

Abstract. *In high computational demand environments, the efficient management of disjoint intervals poses a challenge for traditional data structures. This research proposes a B+-tree based data structure, inspired by Linux kernel solutions. Preliminary experiments demonstrate better search and insertion performance compared to AVL trees. The solution aims to optimize virtual memory management and has potential for improve HPC workloads, with future plans for evaluation in concurrent environments and high-performance hardware.*

Resumo. *Em ambientes de alta demanda computacional, o gerenciamento eficiente de intervalos disjuntos representa um desafio para estruturas de dados tradicionais. Esta pesquisa propõe uma estrutura de dados baseada em árvore B+, inspirada em soluções do kernel Linux. Experimentos preliminares demonstram melhor desempenho de busca e inserção em comparação com árvores AVL. A solução visa otimizar o gerenciamento de memória virtual e tem potencial para aprimorar cargas de trabalho de HPC, com planos futuros para avaliação em ambientes concorrentes e hardware de alto desempenho.*

1. Introduction

Efficient memory management is a critical challenge in high computational demand environments where dynamic allocation and resource contention require sophisticated mechanisms to ensure performance, isolation, and low latency [Madieu 2017]. Virtual memory is organized into disjoint intervals, each with metadata such as address, size, and permissions [Madieu 2017]. The management of these intervals involves search and modification operations, with searching being critical for memory accesses and page fault handling. In contrast to the demands of modern systems, traditional data structures like AVL trees exhibit limited scalability and cache efficiency due to their depth and the single data element stored per node [Demaine 2002].

2. Related Works

In response to these limitations, the Linux kernel has adopted optimized solutions for memory management, such as the Maple Tree, designed for virtual memory management [Howlett 2021], and the Extent Tree, used in Ext4 file systems [Dara 2023]. However, the tight integration of these structures with the Linux kernel code and their implementation in C limit memory safety guarantees and flexibility for reuse in other contexts.

3. Methodology

This research proposes a B+ Tree variation, which offers better performance and safety for efficient management of disjoint intervals. The approach combine aspects present in both the Maple Tree and the Extent Tree, ultimately offering a safe and extensible solution by being implemented in Rust and allowing its usage both inside and outside kernel space.

4. Experiments

Preliminary experiments compared the performance of the proposed structure with an AVL tree, specialized for disjoint intervals, using sets of random intervals. Figure 1 illustrates the variation in average execution times (in nanoseconds) for search and insertion operations as the amount of data increases. For search operations, the new structure demonstrates speedup of 2.15 at the maximum tested data size. While insertion is initially slower for smaller datasets, our B+ Tree exhibits a speedup of 1.72 for insertion at the maximum tested data size.

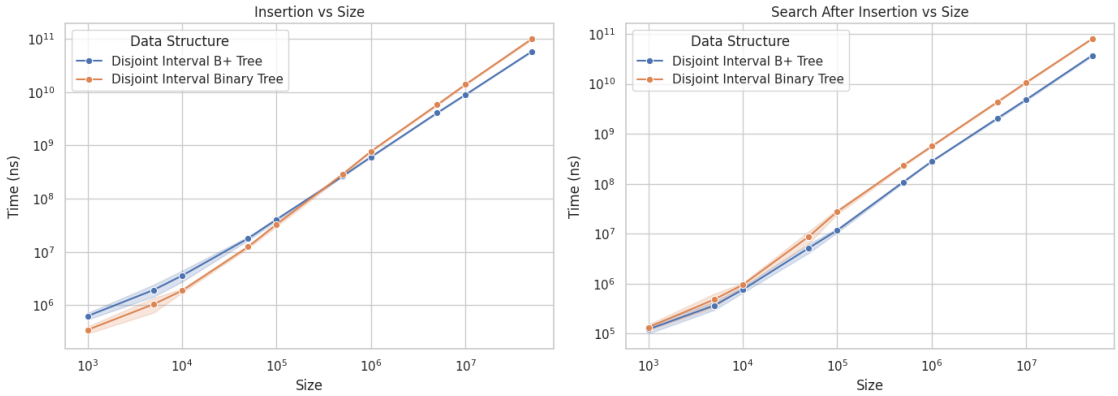

Figure 1. Insertion and Search times (ns) per number of intervals - log scale

5. Conclusion

This research proposed a new B+ Tree variation for the efficient management of disjoint intervals. The structure offers memory safety, flexibility, and better read performance, than binary trees, such as AVL trees, by combining ideas present in trees like the Maple Tree and the Extent Tree. This efficiency has the potential to benefit high-performance computing applications when applied in the context of virtual memory management in operating system kernels. As future work, we intend to explore the implementation of the structure in concurrent environments and evaluate its performance on high-performance hardware with different distributions of intervals for insertions and points for search, as well evaluate the removal operation.

References

Dara, S. (2023). Understanding ext4 disk layout, part 2.

Demaine, E. D. (2002). Cache-oblivious algorithms and data structures. *Lecture Notes from the EEF Summer School on Massive Data Sets*, 8(4):1–249.

Howlett, L. R. (2021). The maple tree, a modern data structure for a complex problem.

Madieu, J. (2017). *Linux Device Drivers Development*. Packt Publishing, Birmingham, England.