Parallel Oblivious Sorting in Intel SGX Enclave

Tianyao Gu, Tian Xie

December 14, 2023

Abstract

Oblivious algorithms have been a popular research topic in cybersecurity due to its resistance to side channel attacks. We implemented a parallel oblivious sorting algorithm in C++ using Intel SGX Enclave, a hardware-based trusted-computing environment. We leveraged OpenMP to achieve multithreading on a 36-core CPU and further improved parallelism through SIMD. We parallelized both the in-enclave computation and the communication with untrusted memory. Our implementation achieves a speedup of 7.5x over the original serial implementation with 8 threads and 16x with 32 threads.

Background of Intel SGX

Intel SGX is a hardware-based trusted-computing environment, which provides a protected region of memory, known as the Enclave Page Cache (EPC). EPC has limited size, and the communication between EPC and untrusted memory is time-consuming. Therefore, it is important to minimize the amount of data swapped in and out of EPC.

Background of Oblivious Sorting Algorithms

Oblivious algorithms ensure that memory access and page swap patterns are independent of secret data, thereby resists side channel attacks.

Background of Flex-way Butterfly Oblivious Sort

To achieve oblivious sorting, Flex-way Butterfly Oblivious Sort applies a random shuffling algorithm followed by a non-oblivious comparison-based sorting algorithm. The key component of the shuffling algorithm is a multi-way butterfly network, emulated with a building block called Merge-split. The algorithm ensures negligible overflow probability and achieves optimal complexity in terms of both computation and I/O.

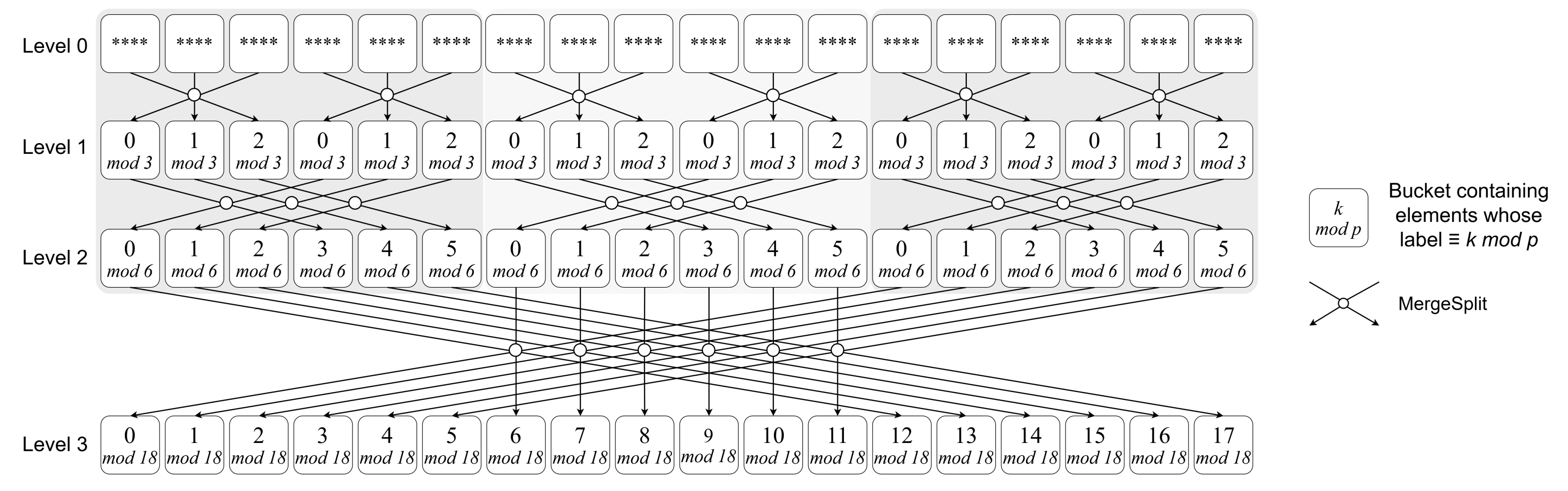

Figure: A Multi-way Butterfly Network

Parallelism Abstraction

- Across merge-split task in each batch
- Within each merge-split task
- Parallel External Merge Sort
- Parallel I/O

Parallelism in I/O

- **Parallelism in I/O:** Identified three I/O patterns and used different strategies for parallelization.
- **Parallelism in Element Movement:** Utilized SIMD for parallelizing data movement, especially for elements with large payloads.
- **Parallelism in Pseudo-Random Numbers Generation:** Parallel generation of secure pseudo-random numbers using multiple generators.

Implementation

- **Overview:** Implemented algorithm based on open-source code [5]. Modifications for parallelism and optimizations for high-end servers and consumer-grade processors.
- **Parallelize Butterfly Network:** Applied OpenMP directives for parallelization, changed routing schedule to iterative, and optimized parameters for maximum parallelism.
- **Parallelize I/O:** Identified three I/O patterns and used different strategies for parallelization.
- **Parallelize Element Movement:** Utilized SIMD for parallelizing element movement, applying C++ intrinsics with AVX512, AVX2, and SSE2 instruction sets.
- **Parallelize Pseudo-Random Number Generation:** Implemented multiple pseudo-random number generators for parallel generation.
- **Optimize Memory Utilization:** Addressed memory fragmentation concerns by allocating a memory pool and implementing a custom allocator.

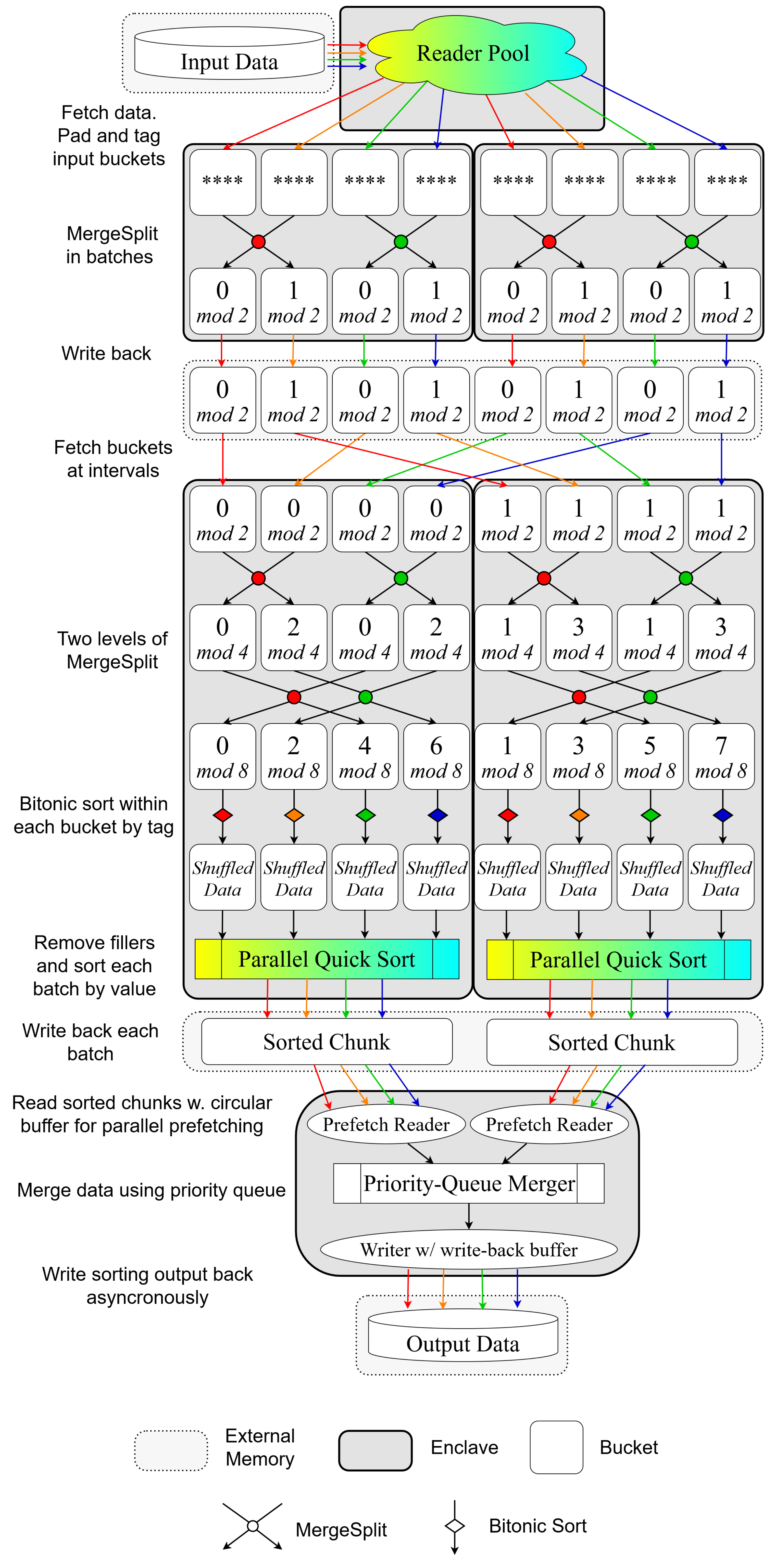

Figure: Diagram of Parallel Oblivious Sort on a toy example. Each color represents a thread. Colorful components are multi-threaded.