Parallel Oblivious Sorting in Intel SGX Enclave

Tianyao Gu, Tian Xie

December 15, 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 3.5x over the original serial implementation with 8 threads and 4.2x with 32 threads. As a byproduct, we also obtained an efficient parallel oblivious random shuffling algorithm, which features and 14x speedup 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.

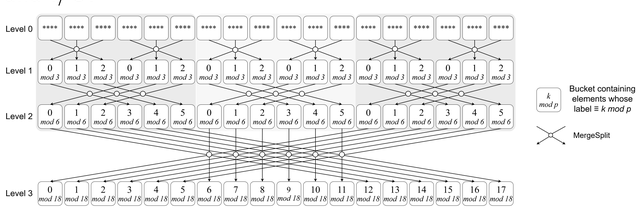

Figure: A Multi-way Butterfly Network

Parallel Oblivious Sorting Implementation Overview

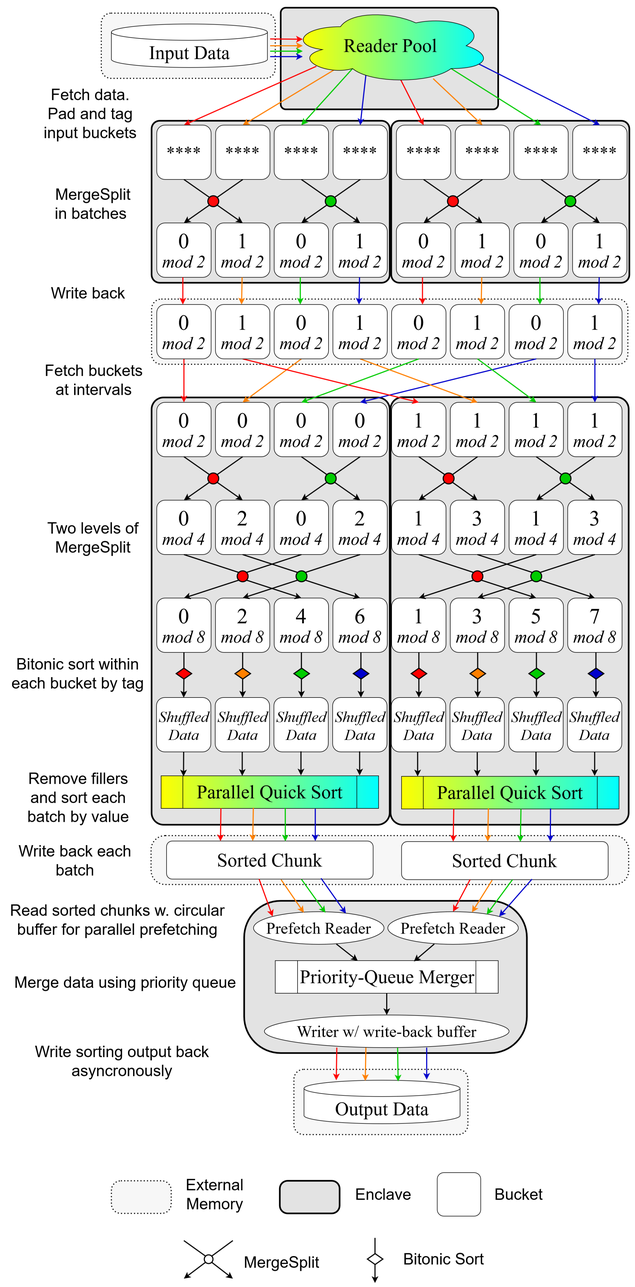

Figure: Diagram of Parallel Oblivious Sort on a toy example. Each color represents a thread. Colorful components are multi-threaded.

Speedup Diagrams for separtate optimizations

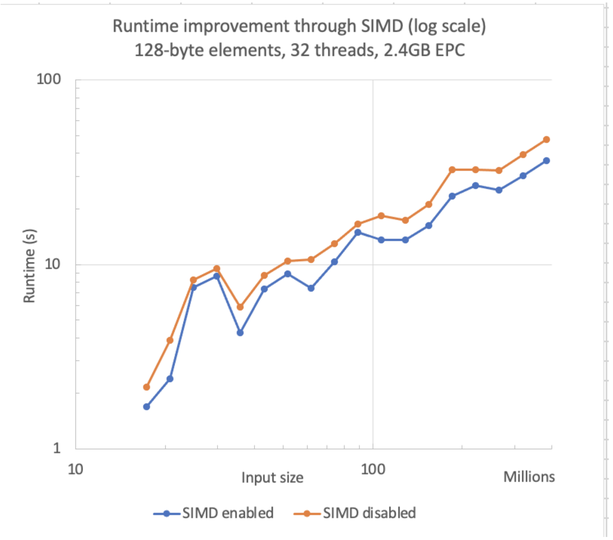

Figure: Runtime Improvement Through SIMD

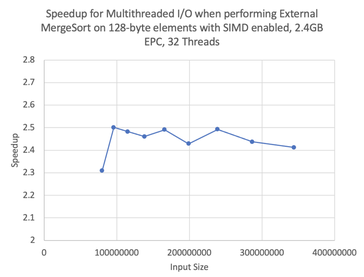

Figure: Speedup for External MergeSort with Multi-threaded I/O

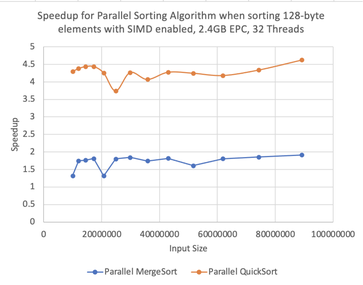

Figure: Speedup using parallel sorting compared to std::sort

Overall Speedup Diagram for Our Parallel K-Way Butterfly Networking Oblivious Sorting/Shuffling

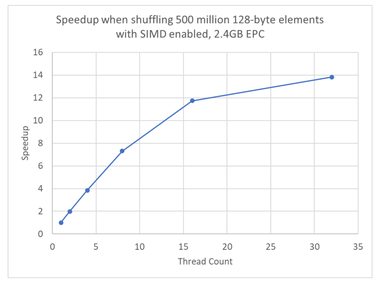

Figure: Speedup for Oblivious Shuffling

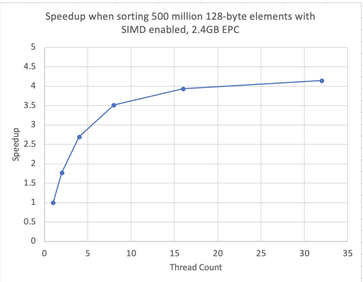

Figure: Speedup for Oblivious Sorting