Data Processing on Modern Hardware
Project – Bitonic and Parallel Sorting

Overview

Sorting is a fundamental operation in database systems, underpinning various tasks such as indexing, query processing (e.g., `ORDER BY`), and searching. As data volumes grow, efficient sorting becomes increasingly critical for achieving low-latency query performance and high-throughput data processing.
This project revisits **Bitonic Sort** in the context of database and analytical workloads. Bitonic Sort is a parallelization-friendly sorting algorithm for both multi-threaded CPU execution and GPU acceleration. It is based on the concept of bitonic sequences—sequences that are first monotonically increasing and then decreasing (or vice versa)—and can be efficiently implemented using fixed, data-independent comparison patterns. These properties make it highly suitable for SIMD and massively parallel architectures like GPUs and an alternative to algorithms like **Radix Sort**, which are harder to parallelize. While Bitonic Sorting is well adopted and optimized for different accelerators, the developments in hardware (e.g., on-board GPUs, Unified Memory Architectures, and richer SIMD instruction sets) might lead to new ways for integration in databases and optimizations.
Students will start with a single-threaded CPU implementation of Bitonic Sort and iteratively optimize the algorithm for:

- Multi-threaded CPU execution
- SIMD vectorization on CPUs
- GPU acceleration using *OpenCL*, *CUDA*, or *Vulkan*

The project includes performance benchmarking on synthetic and database-relevant data distributions and invites students to evaluate trade-offs between hardware utilization, sorting throughput, and implementation complexity.

Background

Unlike Radix Sort, Bitonic Sort is a **comparison-based** sorting algorithm particularly well-suited for parallel hardware. It operates on the concept of *bitonic sequences*, which are sequences that first increase and then decrease, or vice versa. For example, both $3, 4, 5, 6, 10, 9, 8$ and $10, 9, 8, 3, 4, 5, 6$ are bitonic sequences. The key idea is to recursively transform unsorted data into bitonic sequences and then merge them into fully sorted sequences using a predefined pattern of comparisons and swaps; thus, the algorithm is often referred to as *Bitonic Merge Sort*.

As shown in Figure 1, a bitonic merge works by repeatedly comparing and swapping elements at fixed distances, halving the problem size at each step, similar to merge sort, but with operations structured in a way that does not depend on the data values. This predictable and regular structure enables efficient implementation on SIMD units, multi-core CPUs, and GPUs, as it avoids control-flow divergence and can fully exploit hardware parallelism. Although Bitonic

Sort is not asymptotically optimal – with a time complexity of $O(n \log^2(n))$ – its suitability for parallel execution and predictable access patterns make it attractive for sorting moderate-sized datasets in performance-critical database tasks.

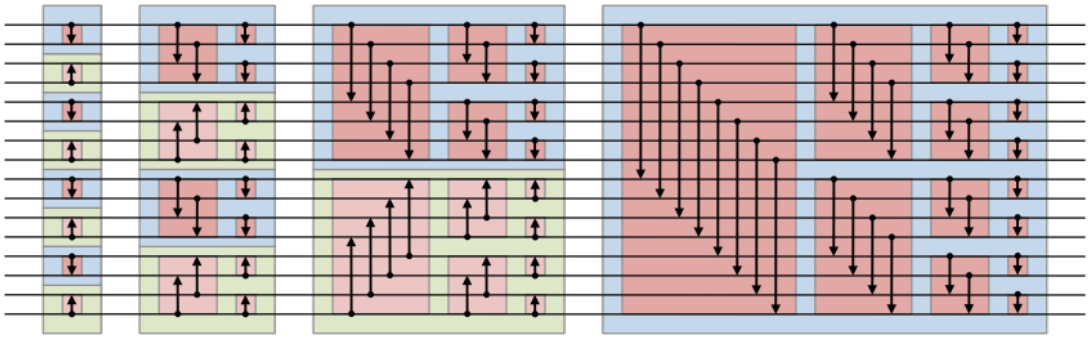

Figure 1: A Bitonic Sort Network for 16 values. The arrows indicate a comparison between two values, so the head of the arrow points to the larger element after the comparison. The boxes illustrate a data group but are not relevant to the algorithm. *Source: Wikipedia*

Objectives

This project aims to explore and optimize Bitonic Sort for high-performance, parallel execution in the context of database systems and analytical engines. A key objective is to implement tiered sorting, where different stages of the algorithm(s) are mapped to the most suitable compute units, such as using CPU cores for early stages and GPUs for large-scale merging while considering data layout and hardware capabilities. In addition, students will also use the fundamental concepts of Bitonic Sorting as a basis to either build a *sorting network* leveraging another sorting algorithm or to parallelize another sorting algorithm.

Furthermore, the project investigates hardware-aware optimizations, including cache-efficient memory access patterns and chiplet-aware scheduling strategies to minimize expensive data movement. Students will analyze how sorting scales with increasing CPU cores or GPU compute units and identify bottlenecks that limit parallelization efficiency. Finally, a comparative evaluation with other parallelizable sorting algorithms like Radix Sort will provide insight into trade-offs in performance, memory usage, and hardware suitability for database workloads.

TL;DR

- Optimize the single-threaded version with SIMD and implement multi-threading
- Accelerate Bitonic Sort using an accelerator (external or internal GPU)
- Use Bitonic Sort to build a sorting network leveraging another sorting algorithm
- Evaluate a tiered-stage Bitonic Sort by comparing it to Radix Sort and other sorting algorithms
- Analyze the potential of Bitonic Sort Networks for modern database and analytical engines (the implementation should support `uint_32` and `float` data types)
- Existing implementations can be used as competitors

Resources

The following resources can help you with the above tasks:

- *`CellSort`: High Performance Sorting on the Cell Processor*: `https://www.vldb.org/conf/2007/papers/industrial/p1286-gedik.pdf`
- *Efficient Implementation of Sorting on Multi-Core SIMD CPU Architecture*: `https://www.vldb.org/pvldb/vol1/1454171.pdf`
- *Optimizing Parallel Bitonic Sort*: `https://liuyehcf.github.io/resources/paper/Optimizing-paral pdf`
- *Fast Sorting Algorithms using AVX-512 on Intel Knights Landing*: `https://hal.inria.fr/hal-01512970v1/document`
- *Bitonic sorting network for n not a power of 2*: `https://hwlang.de/algorithmen/sortieren/bitonic/oddn.htm`
- *The Art of Computer Programming Volume 3: Sorting and Searching*