ULTRA-SORT, EXTREMELY PARALLEL HARDWARE OPTIMIZED SORTING

Saatvik Shah; Dee Dong
Advised by: Anuj Kalia, Prof. Randy Bryant, Prof. Andy Pavlo
Carnegie Mellon University, School of Computer Science

Language Technologies Institute

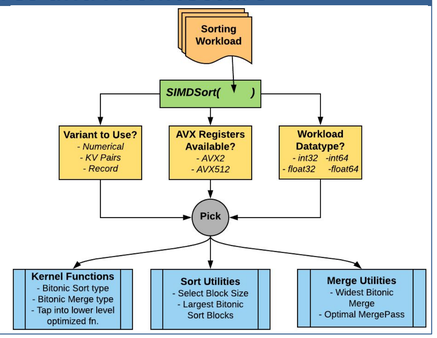
Carnegie Mellon University

Motivation

Sorting is one of the most important phases in many engineering applications. There has been a lot of research to exploit the parallelism one can get out of a simple operation as sorting: SIMD intrinsics, multicore and even NUMA-awareness. The improvements in SIMD width and intrinsics available means an opportunity for better design and speed.
We propose to build a production level DSL-style SIMD sorting system. This should efficiently utilize the hardware it's running on and be optimized for the workload it has to operate on.

System Features and Architecture

- Leveraging SIMD parallelism for the latest hardware - **AVX2 and AVX512**
- Optimized for all numeric data types: *int32, int64, float, double*
- Supports Key-Value or Record Level Sorting(Order-By)
- Parallel and Serial Execution
- GTest Framework integration for exhaustive component testing and Benchmarking
- ***Best case:*** ~8x speedup over std::sort for 1 Million elements and ~14x speedup with multiple cores..



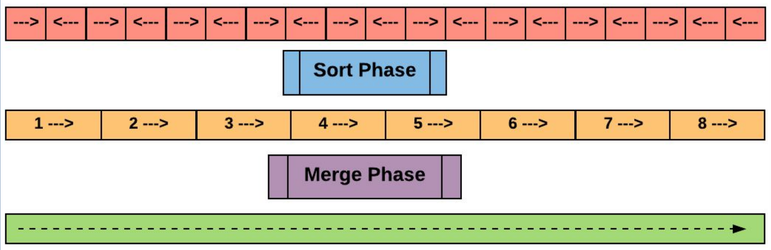
Algorithm Overview

Setup

Consider we want to sort N 32-bit integers. In AVX512, one register can hold 16 such values.

Sort Phase

1. Pack 16 integers into one register, and 16 such registers = 256 integers per block.
2. Apply a 16x16 sorting network to this block to get 16 vertically sorted registers.
3. Apply a 16x16 Transpose.
4. Repeat 1-3 for on the N elements, 256 values at a time.



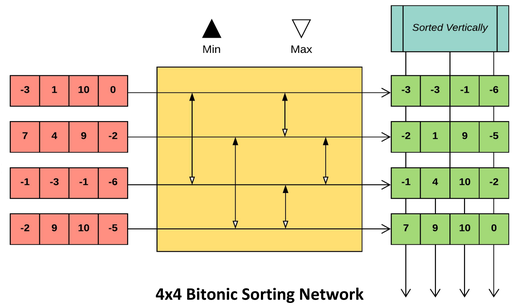
Merge Phase

1. MergePass(X): Merge all consecutive pairs of chunks of size '*X*' into one sorted chunk of size 2**X*.
2. Apply MergePass(16)
3. Apply MergePass(32)
4. Apply MergePass(64)
5. *…… continue till MergePass(Length of Array/2)*

Sort Phase

1. **Bitonic Sorting Network:**
   - SIMD-friendly and branch-free sorting.
   - Used the *Bose Nelson Algorithm* for construction.
   - Custom sizes supported: From 2x2 to 32x32 depends on SIMD register size.

eg. AVX512 can fit 16, 32 bit values - So we can construct a 16x16 register.

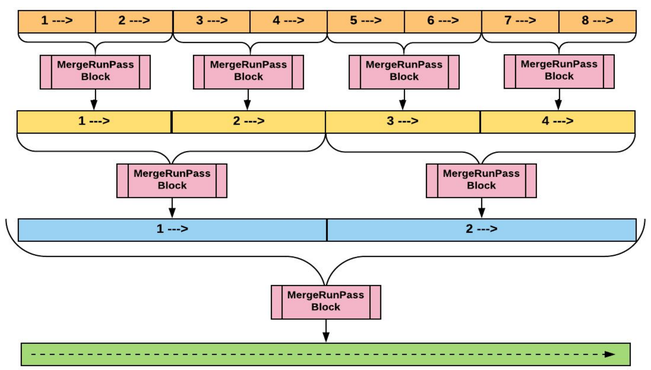


4x4 Bitonic Sorting Network

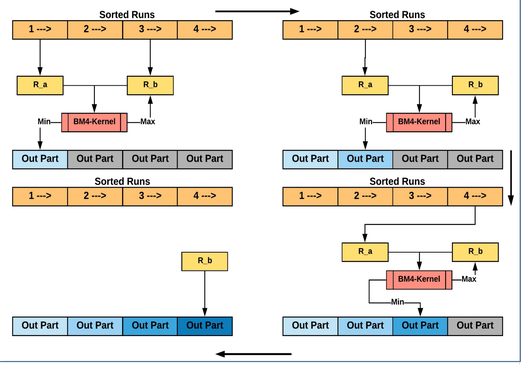
2. **Transpose NxN:** As can be seen in the diagram alongside, a Bitonic sorting network yields a vertically sorted output. In order to sort this horizontally, we apply a SIMD transpose.

Merge Phase

Merging Sorted Runs



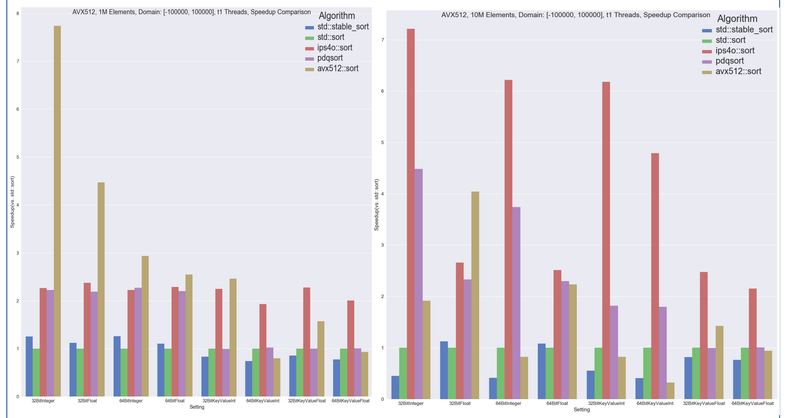
An Example Merge Pass
In the diagram on the left, we've shown how to apply a Bitonic-Merge-4 Kernel to merge 16 elements.
At each step, *Ra* contains the minimum sorted 4, and *Rb* the maximum sorted 4. Store the contents of Ra, and proceed to load next register.



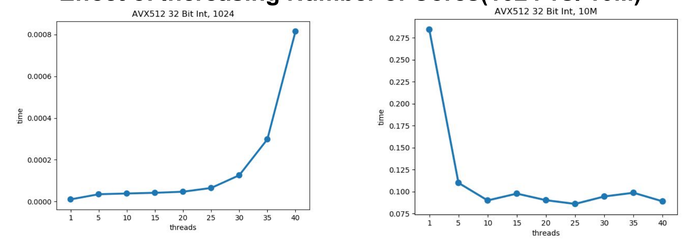
Multicore Parallelism
- SortPhase is B/W Bound
- MergePhase is Compute Bound(but limited parallelism)

Benchmarking and Analysis

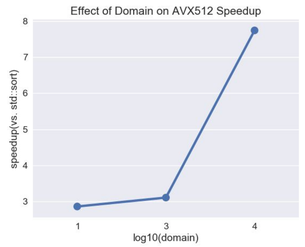
Speedup of AVX-512 with 1M and 10M elements



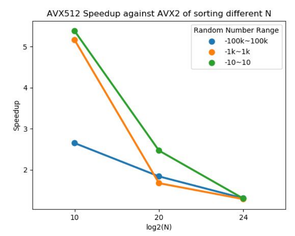
Effect of Increasing Number of Cores(1024 vs. 10M)



Effect of Domain on AVX512 Speedup(1M)



Speedup of AVX512 vs. AVX2



Key Inferences

- AVX512 can achieve **~8x** speedup for numerical sorting and **~4x** speedup for key-value sorting over std::sort. The speedup drops moving from integer to floating point types.
- Going out of cache **hurts**!
- The overhead of thread-management/fork-join parallelism often exceeds speedup.
- The domain of numbers being sorted has an effect on both SIMD and std::sort. The effect is much more on std::sort.

Contact

Dee Dong
Language Technologies Institute
Email: ddong1@andrew.cmu.edu

Saatvik Shah
Language Technologies Institute
Email: saatviks@andrew.cmu.edu

References

1. J. Teubner M. Tamer Ozsu C. Balkesen, G. Alonso. Multi-core, main-memory joins: Sort vs. hashrevisited.PVLDB(1), 7, 2013.
2. J. Chhugani T. Kaldewey A. D. Nguyen A. D. Blas V. W. Lee N. Satish C. Kim, E. Sedlar and P. Dubey. Sort vs. hash revisited: Fast join implementation on modern multi-core cpus. PVLDB2(2) , pp. 1378–1389, 2009.
3. Timothy Furtak, Jose Nelson Amaral, and Robert Niewiadomski. Using simd registers and instructions to enable instruction-level parallelism in sorting algorithms. In Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures , pp. 348–357. ACM, 2007.
4. K. Taura H. Inoue. Simd- and cache-friendly algorithm for sorting an array of structures.PVLDB,8, 2015