High-Performance Parallel Radix Sort on FPGA

Bashar Romanous, Mohammadreza Rezvani, Junjie Huang,
Daniel Wong, Evangelos E. Papalexakis, Vassilis J. Tsotras, Walid Najjar
Department of Computer Science and Engineering, University of California, Riverside
Email: {broma002, mrezv002, jhuan308, danwong}@ucr.edu, {epapalex, tsotras, najjar}@cs.ucr.edu

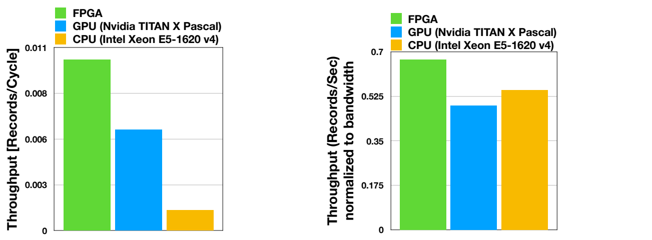
***Abstract*—Sorting is a key part in database operators (like duplicate elimination, sort-merge joins and group-by aggregations). Sorting billions of records in a fast and energy efficient manner has become a key research challenge. In this work, we explore sorting in-memory using a parallel version of Radix Sort to build a high-performance hardware accelerator, called HARS (Hardware Accelerated Radix Sort). Our design enables dividing the unsorted dataset among parallel engines without the need for a merge step. HARS is implemented on Micron's SB-852 FPGA board. The proposed accelerator provides high throughput in-memory sorting at a rate of 44 Million 128-bit records per second. HARS is 1.4x faster than CPU and 1.36x faster than GPU when GPU bandwidth is normalized. Projected performance of a proposed board with a more capable FPGA chip would yield 1.25x higher throughput.**

As technology evolves, new hardware architectures and platforms induce a re-evaluation of sorting algorithms and their implementations on these new architectures. Because the parallel versions of radix sort rely on building and storing a histogram for each Processing Element (PE), it was considered unsuitable for FPGA acceleration as it required too many costly reads and writes to memory. However, new FPGA architectures such as the Xilinx UltraScale+ series [1] and the Intel Stratix 10 DX [2] come with very large on-chip storage that makes it worthwhile to re-evaluate radix sort. Here we describe, implement and evaluate HARS, a parallel implementation of radix sort on an FPGA that takes advantage of this large on-chip storage. Our design enables dividing the unsorted dataset among parallel engines without the need for a merge step.

The main contributions of this paper are:

- A novel parallel in-memory radix sort implementation on FPGA that does not rely on sorting networks and avoids a final, performance limiting, merge step.
- The size of the sorted data is not restricted by the available on-chip memory.
- A constant throughput that is not dependent on the size of data being sorted and scales with on-chip memory size and off-chip memory bandwidth.

Since HARS is designed as an in-memory sorting application, all measurements were done after the data was loaded to global memory. Datasets considered two types of 128-bit records: (1) 80-bit key and 48-bit value (or pointer). (2) 64-bit key and 64-bit value. Even though GPU raw throughput of sorted records per second is 5.4x higher than that of HARS on FPGA, when throughput is normalized by bandwidth, FPGA is 1.2x and 1.36x higher than CPU and GPU respectively.



(a) Unit throughput

(b) Raw throughput normalized by bandwidth

Fig. 1. (a) Throughput normalized by number of PEs, SMs and cores for FPGA, GPU and CPU respectively and (b) per bandwidth, for 268M records dataset

HARS uses both bandwidth and clock cycles more efficiently than both CPUs and GPUs.

The availability of on-chip memory makes it possible to store the histograms locally and reduce the traffic to/from on-board global memory. Using higher radices reduces the number of required iterations in the radix sort algorithm, but increases the sizes of local histograms. Having more PEs increases the parallelism but requires more area on the FPGA(s). Our experimental evaluation has shown that:

- HARS delivers a constant throughput irrespective of the dataset size, unlike the CPU. HARS is therefore more scalable than the CPU implementation.
- The HARS throughput of a single FPGA PE, measured in records/cycle is 1.7x times larger of the GPU's unit of Streaming Multiprocessors (SMs) and 8.2x times larger than that of the CPU's core, as shown in Fig. 1a. Hence demonstrating the computational efficiency of the HARS approach.
- Similarly, when throughput is normalized by bandwidth, HARS is more bandwidth efficient than either the CPU or GPU implementations as shown in Fig. 1b. Thus, HARS's effectiveness in using the available bandwidth is demonstrated.

References

[1] "UltraScale Architecture and Product Data Sheet: Overview," https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf, accessed: 2020-1-12.

[2] "Intel Stratix 10 Dx Product Table," https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/pt/stratix-10-dx-product-table.pdf, accessed: 2020-1-12.