Designing Efficient Sorting Algorithms for Manycore GPUs

Onur Cankur

Designing Efficient Sorting Algorithms for Manycore GPUs

- Nadathur Satish. Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley
- Mark Harris, Michael Garland. NVIDIA Corporation

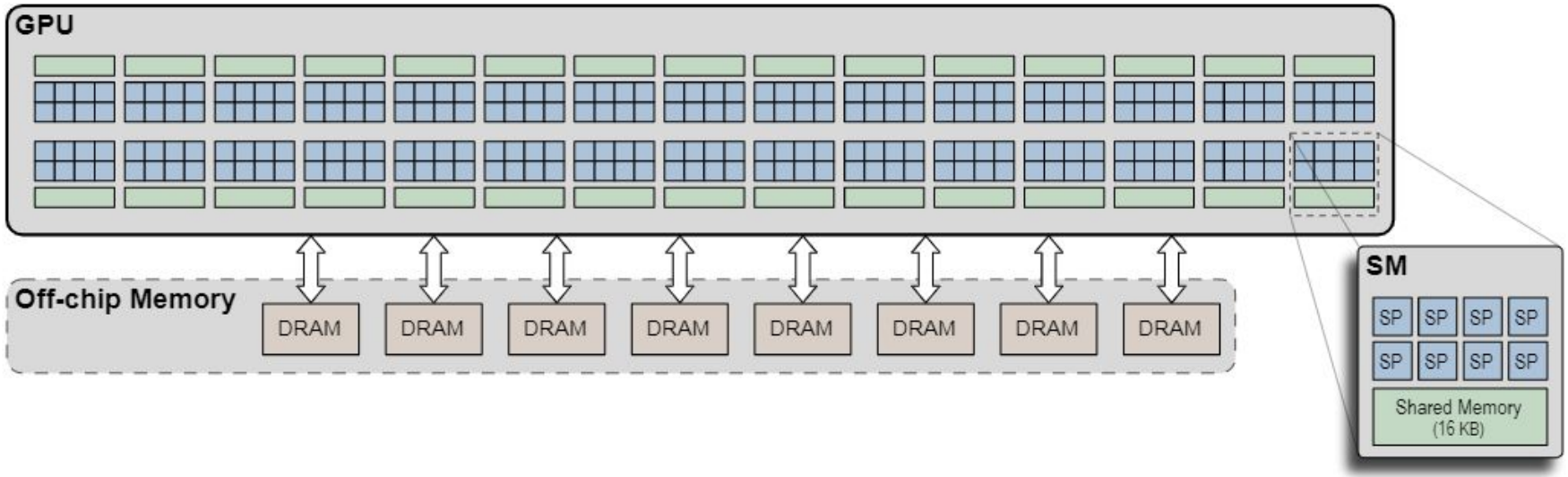
- IEEE International Symposium on Parallel & Distributed Processing
- 2009

Overview

- Designed efficient sorting algorithms using CUDA
- Focused on radix sort and merge sort
- Developed fastest radix sort algorithm compared to other GPU and multicore CPU implementations.

Parallel Computing on the GPU

- Fully programmable manycore chips built around an array of parallel processors.
- GeForce GTX 280 GPU with 240 scalar processor cores (SPs), organized in 30 multiprocessors (SMs).
- 16KB on-chip memory that has very low access latency and high bandwidth, similar to an L1 cache.
- The SM employs a SIMT (Single Instruction, Multiple Thread) architecture.
- Threads are executed in groups of 32 called warps.
- On CUDA, host program executes on the CPU and the parallel kernels execute on the GPU.
- A kernel is a SPMD-style (Single Program, Multiple Data) computation.



Algorithm Design

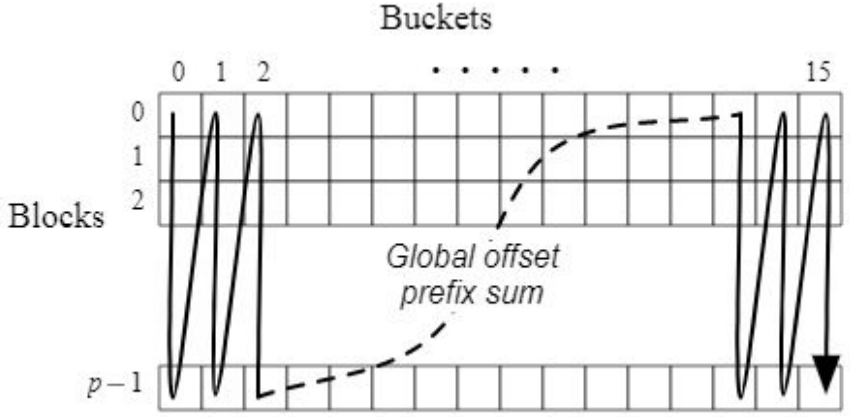
- Divide the work to *p* thread blocks of *t* threads each.
- In this paper, thread block size t = 256.
- Input array size = n.
- Number of thread blocks $p \propto n/t$.

Radix Sort

- Keys are d-digit numbers.
- Sorts on one digit of the keys at a time, from least to most significant.
- Efficient for sorting small keys.
- Complexity of sorting n input records = O(n)
- For a given digit of each key, compute the number of keys whose digits are smaller plus the number of keys with the same digit occurring earlier in the sequence.

Radix Sort

- Divide the sequence into p thread blocks.
- 256 threads in each block.
- Assign 4 elements to each thread which means 1024 elements per block.
- Number of blocks p = n/1024.
- Each digit consists of b bits. Buckets = 2^b.
- **Algorithm**
  - Each block loads and sorts its tile in on-chip memory using b iterations of 1-bit split.
  - Each block writes its 2^b-entry digit histogram and the sorted data tile to global memory.
  - Perform a prefix sum over the p×2^b histogram table, stored in column-major order, to compute global digit offsets.
  - Using prefix sum results, each block copies its elements to their correct output position.



Per-block histograms to be stored in column-major order for prefix sum.

Merge Sort

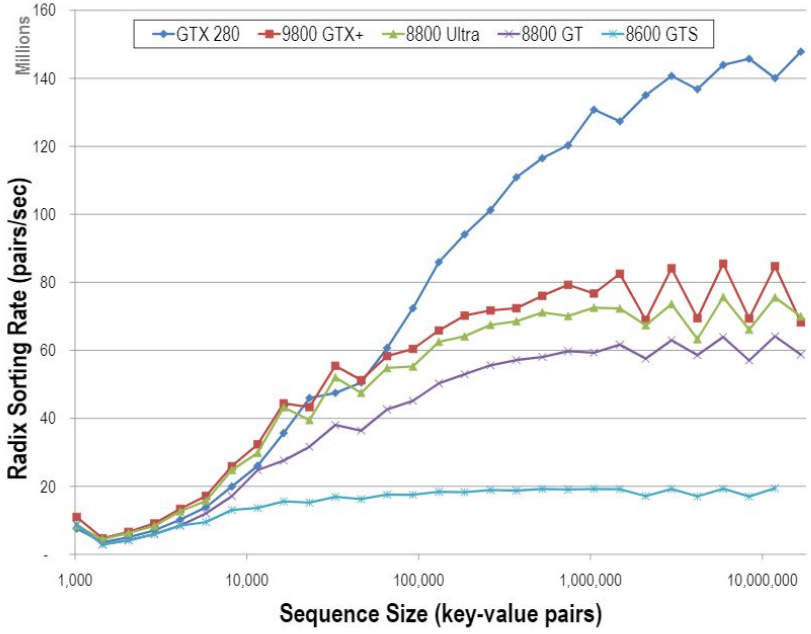
- Divide-and-conquer merge sort
- The merge sort procedure:
  - 1) Divide the input into p equal-sized tiles.
  - 2) Sort all p tiles in parallel with p thread blocks.
  - 3) Merge all p sorted tiles.
- Merging can be done in on-memory if sequences are small.
- Divide larger arrays up into tiles of size at most t.

Performance Analysis

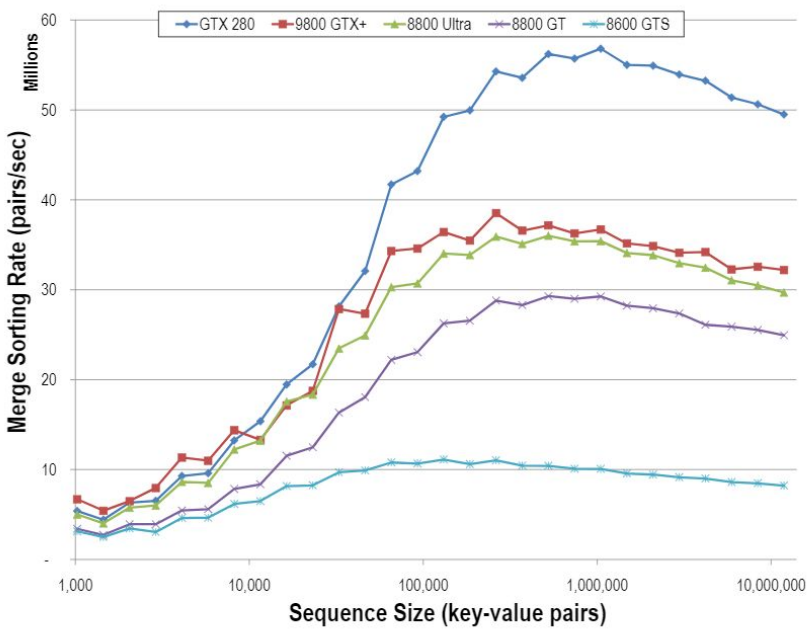
- Tests are based on sorting sequences of key-value pairs.
- Keys and values are 32-bit words.
- Uniform random number generator to produce random keys.
- Report GPU times as execution time not including the data transfer time from CPU to GPU.
- Range of NVIDIA GeForce GPUs:
  - GTX 280 (30 SMs)
  - 9800 GTX+ (16 SMs)
  - 8800 Ultra(16 SMs)
  - 8800 GT (14 SMs)
  - 8600 GTS (4 SMs).

Performance on Different GPUs

- The progressively more parallel devices achieve progressively faster running times.

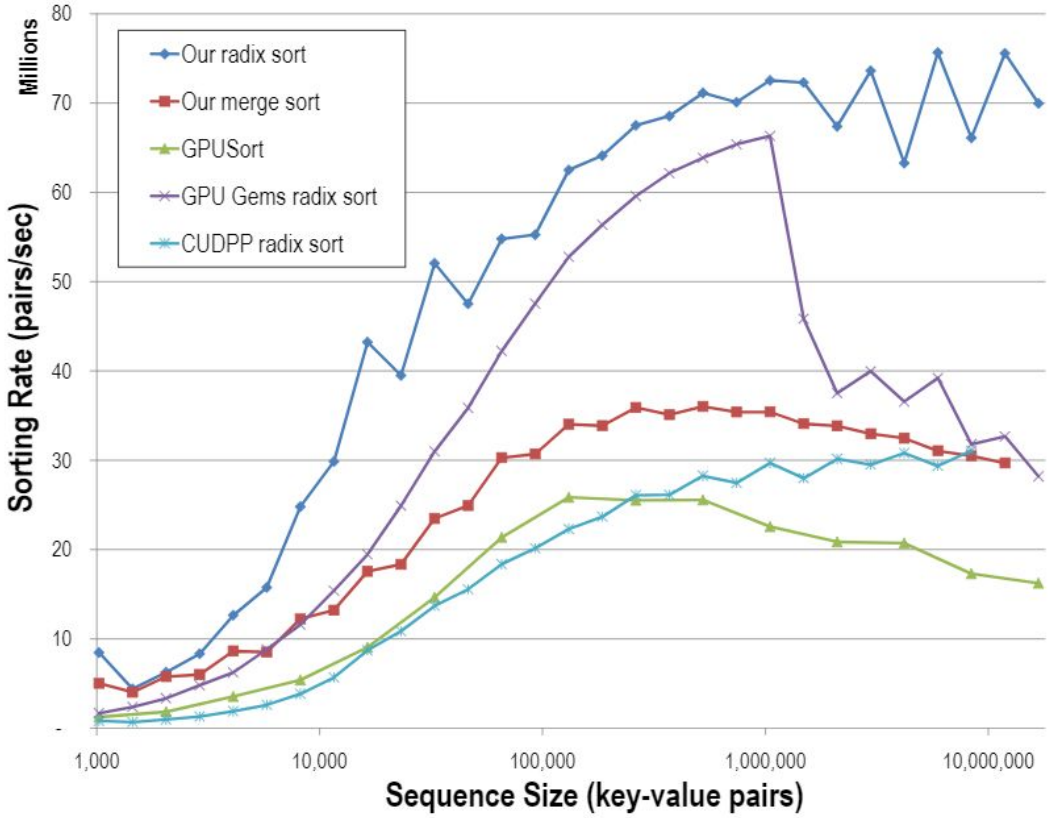


(a) Radix sort

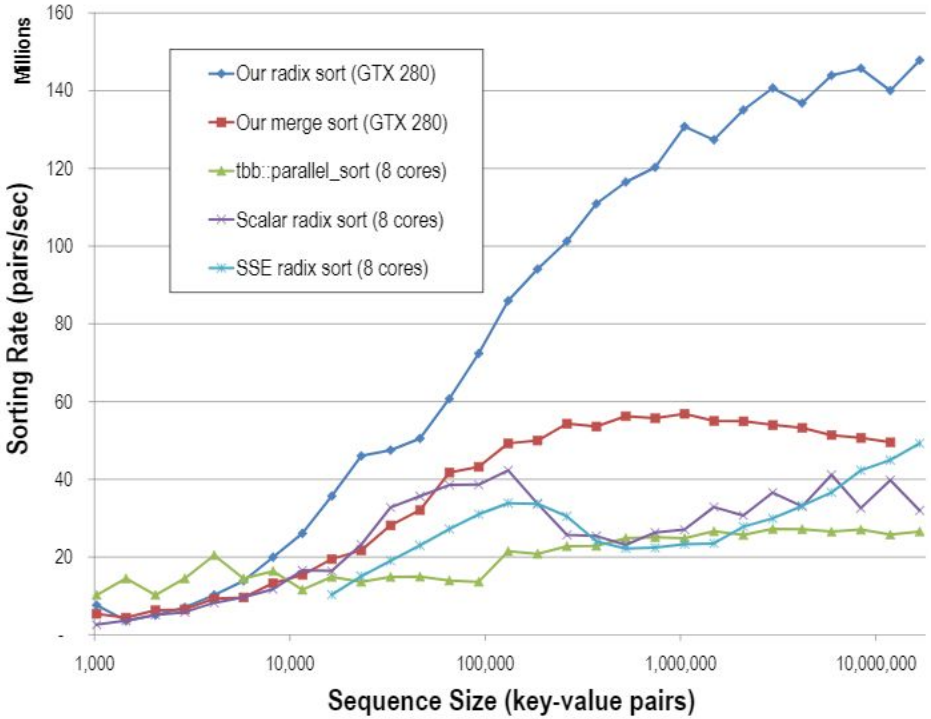


(b) Merge sort

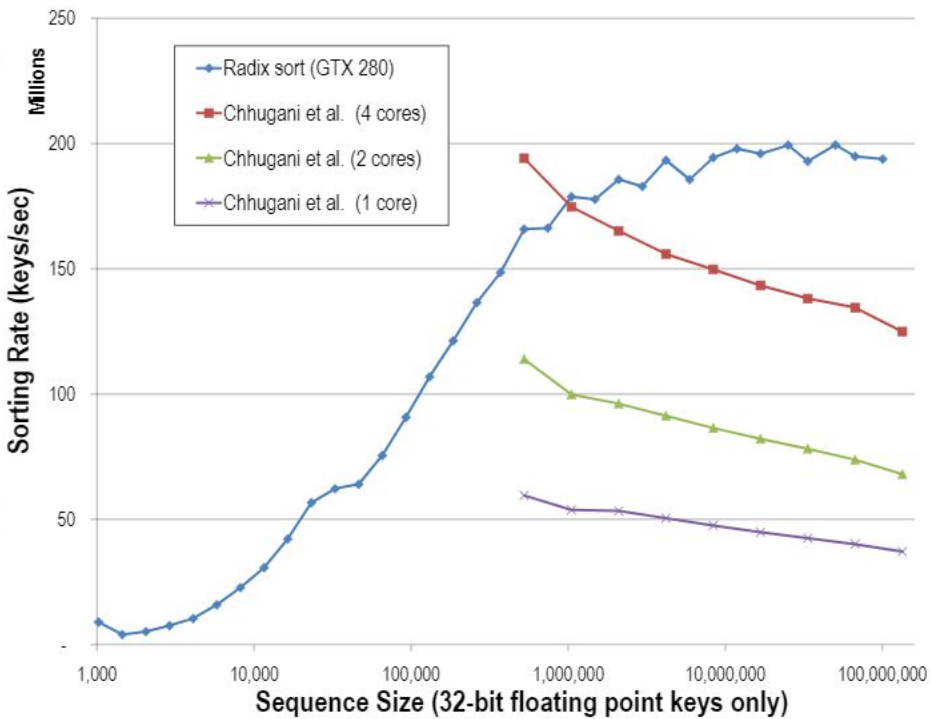
Comparison with GPU-based Methods



Comparison with CPU-based Methods



(a) 8-core Clovertown (key-value pairs)



(b) 4-core Yorkfield (float keys only)