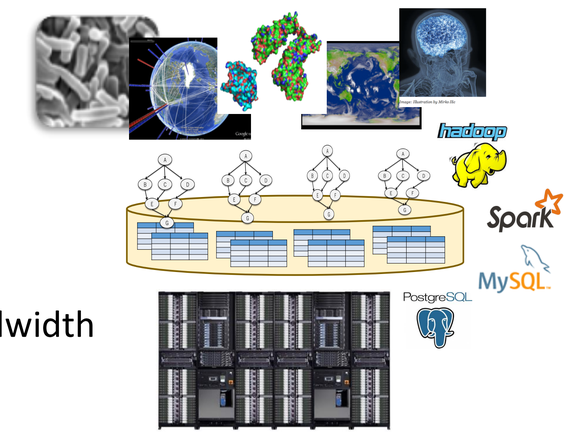
Out-of-core Sorting Acceleration using GPU and Flash NVM

Hitoshi Sato[†‡], Ryo Mizote[†‡], Satoshi Matsuoka[†‡]

[†] Tokyo Institute of Technology, [‡] CREST, JST

Intoroduction

- Sorting is a Key Building Block for Big Data Applications
  - ✓ e.g., Database Management Systems Programming Frameworks, Supercomputing Applications, etc.
  - ✓ Large Memory Capacity Requirement
- Towards Future Computing Architectures
  - ✓ Dropping Available Memory Capacity per Core for Achieving Efficient Bandwidth by Increasing in Parallelism, Heterogeneity, Density of Processors
    - ➢ e.g., Multi-core CPUs, Many-core Accelerators
    - ➢ Post Moore Era
  - ✓ Deeping Memory/Storage Architectures
    - ➢ Device Memory on Many-core Accelerators,
    - ➢ Host Memory on Compute Nodes
    - ➢ Semi-external Memory connected w/ Compute Nodes such as Non-volatile Memory(NVM), Storage Class Memory (SCM)

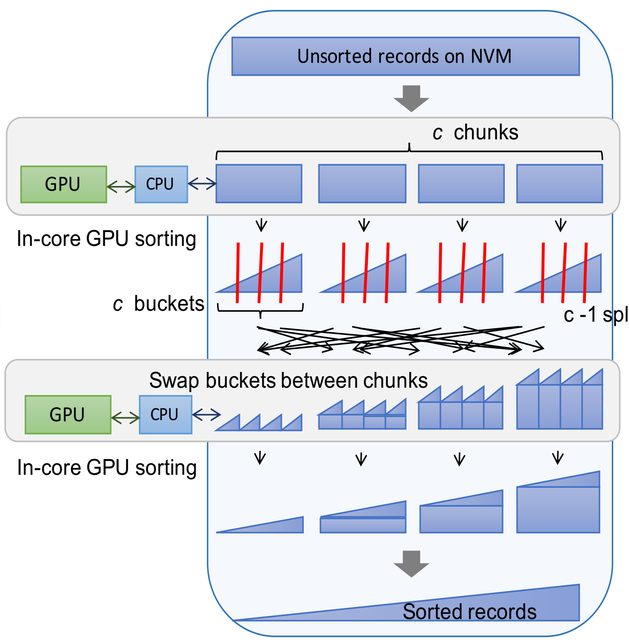


Motivation:
- ✓ How to overcome memory capacity limitation?
- ✓ How to offload bandwidth oblivious operations onto low throughput devices?

Proposal: xtr2sort (Extreme External Sort)

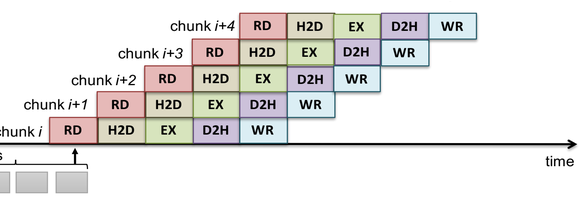
- Sample-Sort-Based Out-of-core Sorting Approach[1][2] for Deep Memory Hierarchy Systems w/ GPU and Flash NVM
- I/O Chunking to fit GPU Memory Capacity in order to exploit Massive Parallelism and Memory Bandwidth of GPU
  - ✓ Employ Asynchronous Data Transfers using CUDA Streams and cudaMemCpyAsync() between CPU and GPU
  - ✓ Page-locked Memory (a.k.a. Pinned Memory) Volumes required
- Pipeline-based Latency Hiding to overlap File I/O between Flash NVM and CPU using Linux Asynchronous I/O System Calls
  - ✓ **Pros:** Fully-overlapped READ/WRITE File I/O
  - ✓ **Cons:** Direct I/O required, e.g., O_DIRECT Flag, Aligned File Offset Memory Buffer, Transfer Size

1. Unsorted records are located on Flash NVM
2. Divide input records into c chunks to fit GPU mem capacity
3. Then, sort the chunks on GPU in the pipeline w/ data transfers
4. Partition each of chinks into c buckets using randomly sampled c-1 splitters
5. Then, swap the buckets between chunks
6. Sort each of the chunks on GPU in the pipeline w/ data transfer
7. Sorted records are placed on Flash NVM



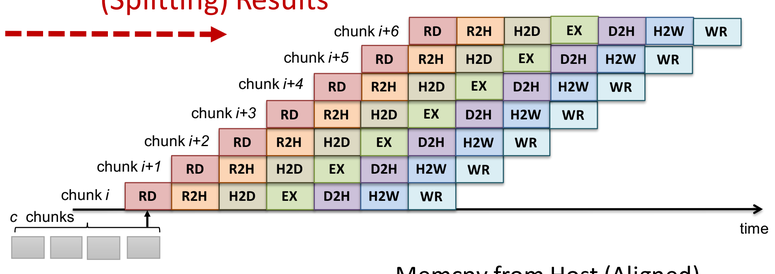
Regular Chunk Size for Aligned File Offset, Memory Buffer, Transfer Size

5 Stage Pipeline Approach
- ✓ 3 CUDA Streams for H2D, EX, D2H
- ✓ Asynchronous I/O for RD, WR
- ✓ 2 READ Pinned Buffers for RD, H2D, and 2 WRITE Pinned Buffers for D2H, WR



Irregular Chunk Size depending on Sampling (Splitting) Results

7 Stage Pipeline Approach



- ✓ 3 CUDA Streams for H2D, EX, D2H
- ✓ Asynchronous I/O for RD, WR
- ✓ 2 POSIX Threads for R2H, D2H
- ✓ 2 READ Aligned Buffers for RD, H2D, 2 WRITE Aligned Buffers for D2H, WR, and 4 Device Pinned Buffers for R2H, H2D, D2H, H2W

RD	READ I/O from NVM	**R2H**	Memcpy from Host (Aligned) to Host (Pinned)	**H2D**	Memcpy from Host (Pinned) to Device	**EX**	Compute on Device
WR	WRITE I/O from NVM	**H2W**	Memcpy from Host (Pinned) to Host (Aligned)	**D2H**	Memcpy from Device to Host (Pinned)		

Experiment

Hardware

CPU	Intel Xeon E5-2699 v3 2.30 GHz (18 cores) x 2 sockets, HT enabled
MEM	DDR4-2133 128 GB
GPU	NVIDIA Tesla K40 w/ 12 GB Mem
NVM	Huawei ES 3000 v1 PCIe SSD 2.4 TB

Software

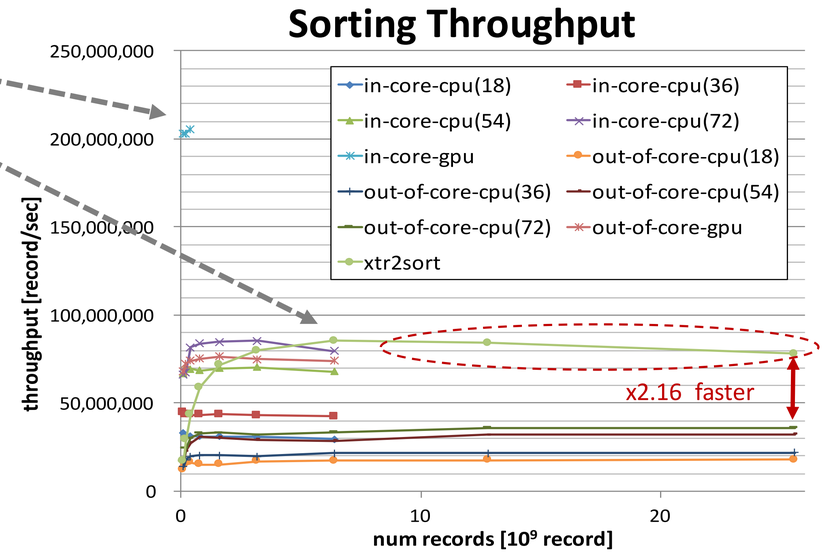
OS	Linux 3.19.8
Compiler	gcc 4.4.7
CUDA	v7.0
Thurst	v1.8.1
File System	xfs

In-core GPU Sorting
~ 0.4 G records
GPU Memory Capacity Limitation

In-core CPU Sorting
~ 6.4 G records
Host (CPU) Memory Capacity Limitation

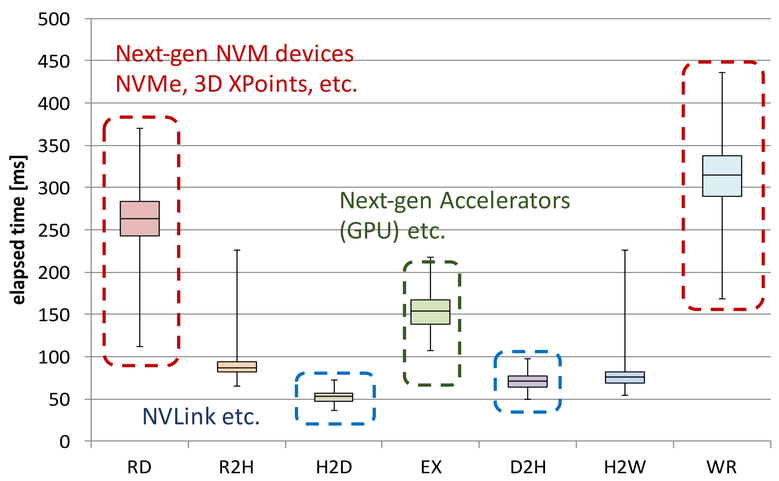
xtr2sort
~ 25.6 G records
x64 larger record size than in-core-gpu
x4 larger record size than in-core-cpu

Sorting Throughput



Comparison using uniformly distributed random records w/ int64_t
- ✓ **in-core-cpu(n)**: In-core CPU sorting w/ libc++ Parallel Mode using n threads
- ✓ **in-core-gpu**: In-core GPU sorting w/ Thrust
- ✓ **out-of-core-cpu(n)**: Same Technique as xtr2sort, but only using CPU (Same Device Mem, n threads)
- ✓ **out-of-core-gpu**: Same Technique as xtr2sort, but only using GPU, no File I/O
- ✓ **xtr2sort**: Proposed Technique

Distribution of Execution Time in Each Pipeline Stage



Summary

- xtr2sort: Sample-sort-based Out-of-core Sorting for Deep Memory Hierarchy Systems w/ GPU and Flash NVM
- Experimental results show that xtr2sort achieves up to
  - ✓ x64 larger record size than in-core GPU sorting
  - ✓ x4 larger record size than in-core CPU sorting
  - ✓ x2.16 faster than out-of-core CPU sorting using 72 threads
- I/O chunking and latency hiding approach works really well for GPU and Flash NVM
- Future work includes performance modeling, power measurement, etc.

[1] Peaters et al. “Parallel external sorting for CUDA-enabled GPUs with load balancing and low transfer overhead”, IPDPSW Phd Forum, pp 1-8, 2010

[2] Ye et al. “GPU Mem Sort: A High Performance Graphics Co- processors Sorting Algorithm for Large Scale In-Memory Data”, GSTF International Journol on Computing, Vol. 1, No.2, pp. 23- 28, 2011