CMPT 984 Project Presentation
Accelerate Data Compression with Utilising GPU

Xinjie (Edward) HU

Simon Fraser University, Database System Lab

Apr. 2025

SIMON FRASER
UNIVERSITY

Outline

Background: CPU or GPU?

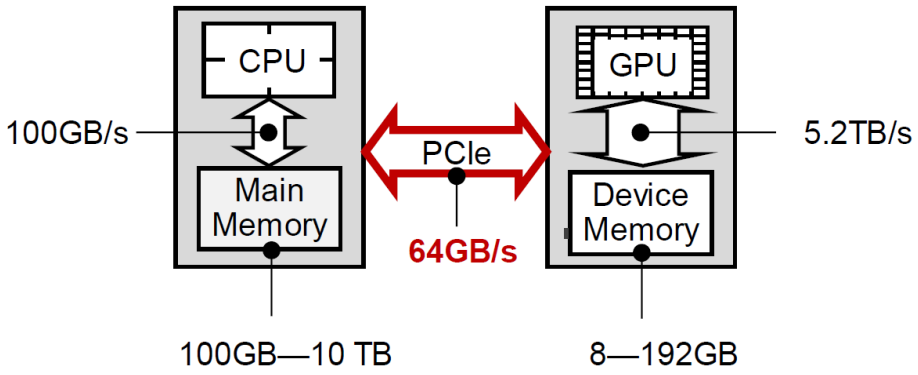

Figure 1.1: Bandwith Comparison

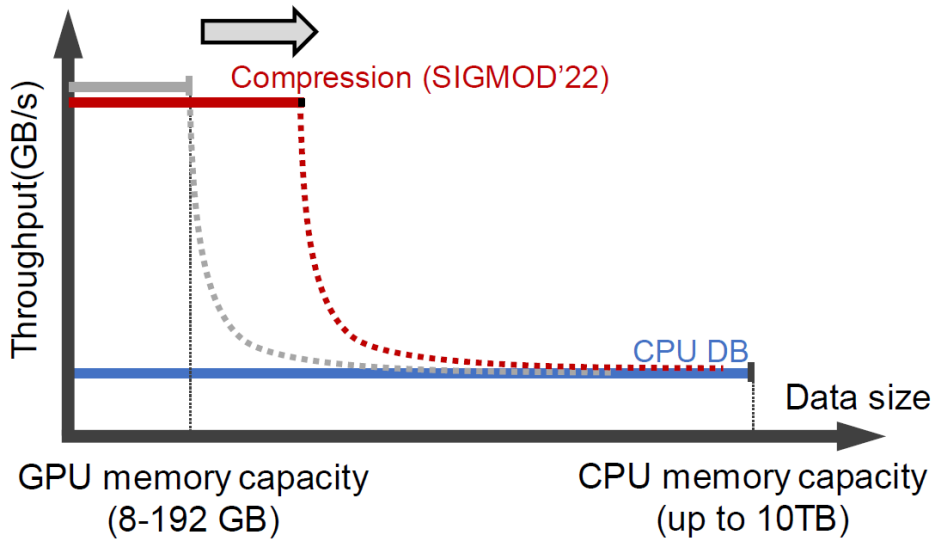

Figure 1.2: Tradeoff between GPU and CPU based DBMS

- Reasons to use GPU: Many many threads, high memory bandwith.
- Reasons to reject GPU: High overhead transmitting data between CPU and GPU, limited memory capacity.
- Both pictures are from Professor Xiangyao YU from UW-Madison, source:`https://uwaterloo.ca/data-systems-group/sites/default/files/uploads/documents/talk-xiangyao-3-12.pdf`

Research Focus

Data Compression:

- A widely used trick in DBMS for efficient storage purpose.
- Use cases:
  - Bulk size documents (legal contracts, logs) and BLOB (image, video).
  - Columnar storage.

Project Vision:

- Find out a better solution to accelerate data compression with using GPU to indirectly improve query processing performance.

Benchmark: Dictionary Encoding

How it Works:

For those symbols (characters) that present frequently in the file, use fewer bits to replace. a.k.a *bit-packing*.

Example

One character takes 8 bits. If 16 frequent characters are selected, only 4 bits are enough to store those characters. With an additional bit used to mark the following bit stream is encoded or not, 5 bits are enough to represent.

Procedures (and opportunities by using GPU):

1. Preselect a collection of frequent symbols, construct mapping relationship between character and encoded value $\rightarrow$ *could load to constant global memory*.
2. Divide the file stream into chunks, each thread encodes a chunk $\rightarrow$ *execute in parallel*
3. As encoded data varies on size, the offset of each encoded data chunk needs to be calculated.
4. Fill encoded chunks into the byte stream according to the offset $\rightarrow$ *execute in parallel*.

Key Hypothesis

Using the following 3 techniques will significantly contribute to GPU compression acceleration.

1. Lookup Dictionary (LD)
2. Memory Coalesced Access (MCA)
3. Parallelized Assembly (PA)

Lookup Dictionary (LD)

- The mapping information from each frequent shown character to bit value is a predetermined dictionary.
- Load to constant memory (a cached, read-only memory shared by threads).
- Time Complexity: $O(1)$.
- Vanilla: a `for` loop, iterate over list to check whether hits or not.

Memory Coalesced Access (MCA)

Consecutive threads access to consecutive data.
Consecutive:

- Threads in GPU are logically organised in *blocks*.
- (blockId, blockDimension, threadId) can uniquely distinguish a thread.
- Primarily refers to consecutive threads within the same block.

Advantage: Reduce number of access to global memory.

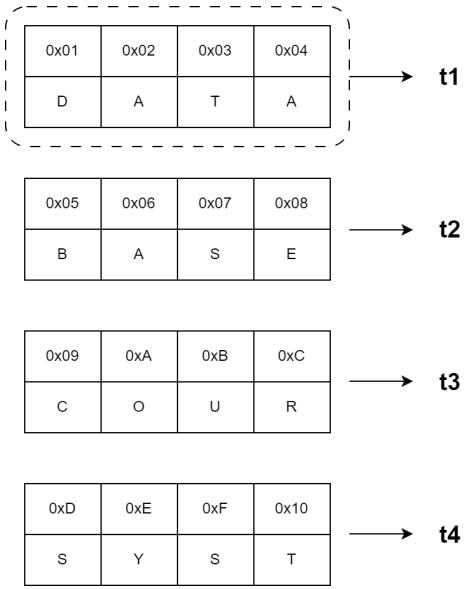

Figure 2.3: Non-Coalesced Access

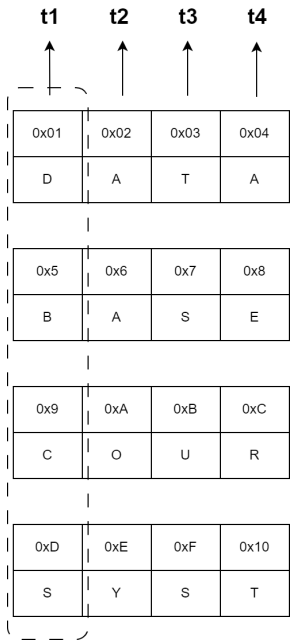

Figure 2.4: Coalesced Access

Parallelized Assembly (PA)

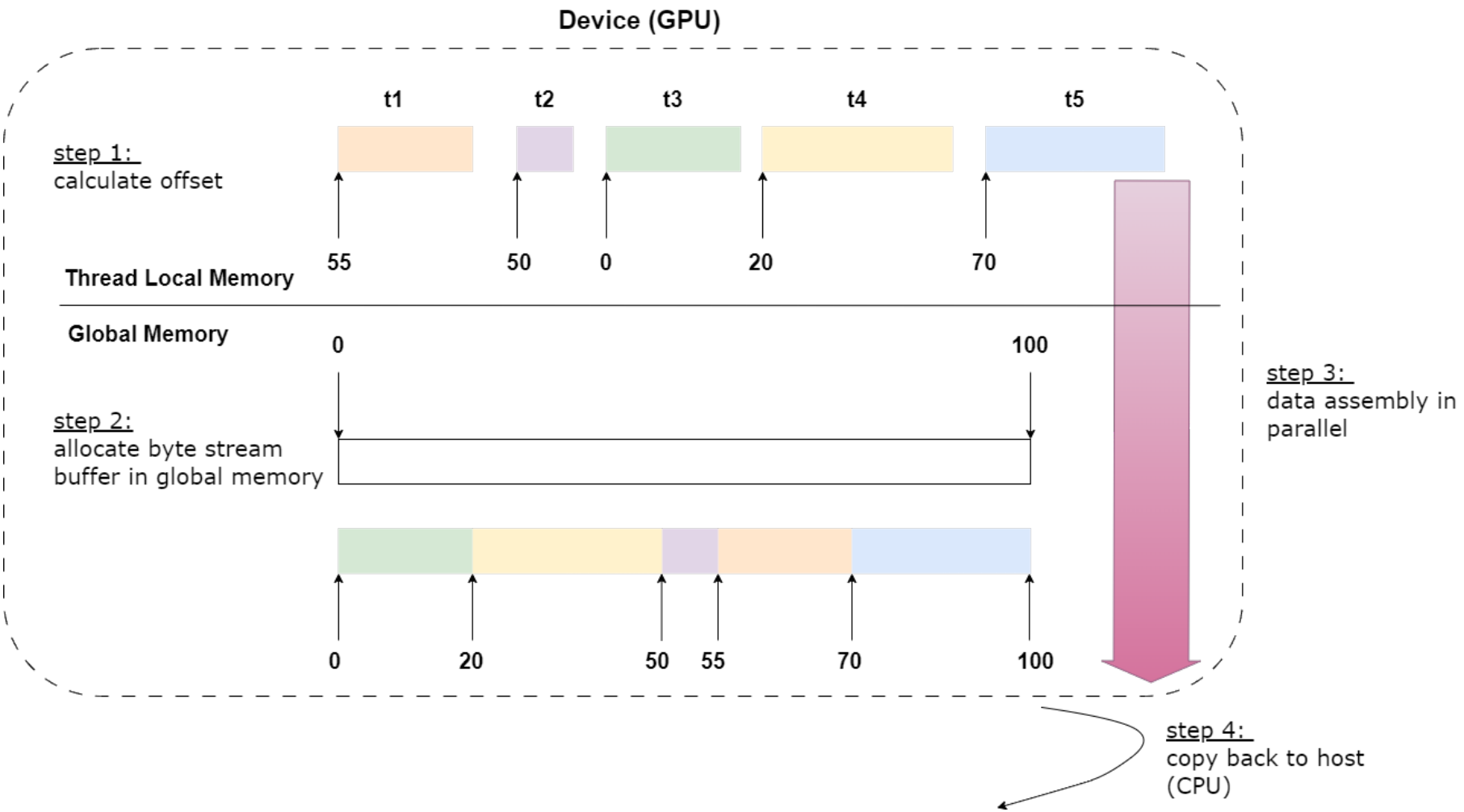

Figure 2.5: Parallelized Assembly

Implementation

- Hardware: GPU in Google Colab (free tier, model unknown).
- Development Tools: CUDA (C++) with NVCC, CMake.
- Profiling Tool: Nsight Compute [1] provided by Nvidia.
- Hyperparametres:
  - 16 fixed characters are preselected according to rule-of-thumb. e.g. ["e", "a", "o", " "] always show up more frequently.
  - Chunk size: 1024 characters.
  - Block Dimension: 256.
- Workloads: Wikipedia Top 1000 Articles in *.txt* format (1000 separate files).
- Pure CPU-based algorithm is also implemented as a comparison.
- All metrics are the mean value after running 10 times.

[1] `https://developer.nvidia.com/blog/using-nsight-compute-to-inspect-your-kernels/`

Evaluation: Space Saving Ratio

- CPU and GPU are comparable.
- Due to the highly skewed distribution of characters, this simple algorithm can achieve fairly decent result.

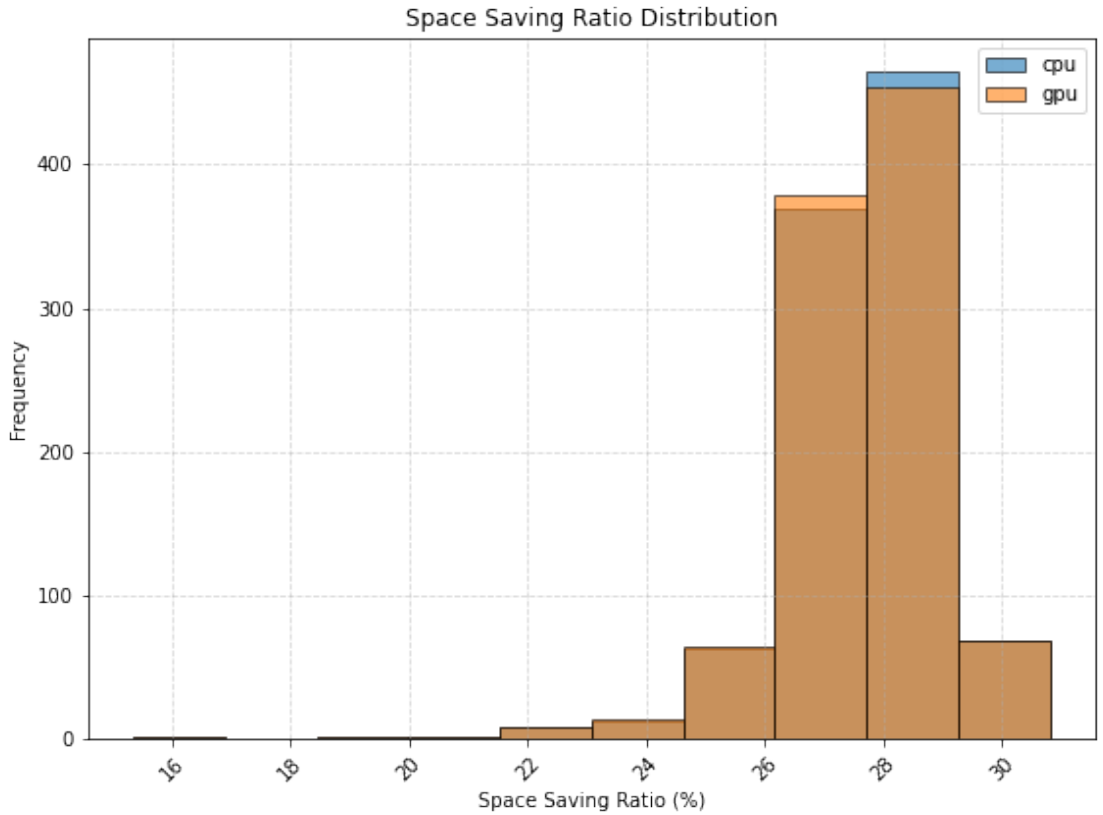

Figure 3.6: CPU and GPU achieves comparable Space Saving Ratio

Evaluation: Latency Breakdown

- Lookup Dictionary in constant memory and Parallelized Assembly are crucial in reducing overall latency.
- Coalesced Memory Access is not satisfactory.

Table 3.1: Compression & Decompression Latency Breakdown

Stages	**All ON**	**OFF: LD**	**OFF: LD&MCA**	**OFF: LD&PA**	**All OFF**	**CPU**
Host to Device	42	36	36	35	35	816
Character Replacement	92	132	132	132	132	
Compute Offset	5	5	5	5	5	
Assemble Data	9	9	9	370	371	
Device to Host	6	6	5	6	6	
Compression Latency	**157**	**191**	**192**	**552**	**553**	**816**
Decompression Latency	**118**	**117**	**117**	**117**	**117**	**586**

- All metrics are measured in `ms`.

Lessons Learned From Profiling

- Nvidia provides handy tools (`ncu` and `nsys`) for programme profiling.
- LD and PA increase memory throughput and reduce elapsed cycles.
- For example, the profiling report of *compress* kernel (utilising LD and MCA).

`compressKernelBitPack_coalesced(const unsigned char`

Section: GPU Speed Of Light Throughput		
Metric Name	Metric Unit	Metric Value
DRAM Frequency	Ghz	10.24
SM Frequency	Ghz	2.23
Elapsed Cycles	cycle	7,606,703
Memory Throughput	%	39.08
DRAM Throughput	%	0.90
Duration	ms	3.40
L1/TEX Cache Throughput	%	34.43
L2 Cache Throughput	%	39.08
SM Active Cycles	cycle	7,550,961.84
Compute (SM) Throughput	%	60.03

`compressKernelBitPack_coalesced(const unsigned char *`

Section: GPU Speed Of Light Throughput		
Metric Name	Metric Unit	Metric Value
DRAM Frequency	Ghz	10.24
SM Frequency	Ghz	2.23
Elapsed Cycles	cycle	5,838,846
Memory Throughput	%	52.51
DRAM Throughput	%	1.18
Duration	ms	2.61
L1/TEX Cache Throughput	%	52.93
L2 Cache Throughput	%	52.32
SM Active Cycles	cycle	5,791,344.15
Compute (SM) Throughput	%	62.42

Figure 3.7: Before (left) and after (right) leveraging LD

- While "This kernel has uncoalesced global accesses resulting in a total of 2205750 excessive sectors" $\rightarrow$ further works need to be done on memory alignment, blockDimension tuning, fully utilising shared memory.

Conclusion

- Dictionary encoding (or bit-packing) is an efficient, economic compression algorithm.
- GPU outperforms CPU (whilst awful GPU design can even perform worse than CPU).
- Features including LD, PA significantly boost performance.
- Limitations: Have not found very suitable query processing related tasks to be applied to. Needs further literature review.
- Further improvements could be done with the help of profiling programme.