Enabling faster processing of big-data using GPU decompression

Andrei Gorgan
a.gorgan@student.utwente.nl
University of Twente
Enschede, The Netherlands

Abstract

Processing big-data has been shown to have a many fold speedup for GPU hardware, however the process of retrieving ready-to-use data from storage devices still requires a process of decompression, currently performed on the CPU. Due to the increasing computational power of GPUs, the decompression step starves the GPU of data, effectively doing nothing until more data is available to process. This research analyses the minimum required speed of decompression on a GPU, such that offloading the decompression step to the GPU, is faster than traditional methods that utilize the CPU. Results show that GPUs cannot outperform CPUs when considering compression ratio, however the improved parallelism of GPUs allows for a 2 times reduction in decompression times.

Keywords: GPGPU, bypassing CPU decompression, big-data compression and decompression algorithms, decompression algorithm recommender system, GPU guidelines

1 Introduction

Ever since the creation of the Electronic Numerical Integrator And Computer (ENIAC), computers have been steadily increasing in complexity and computational power. In recent years, however, big data processing has emered as a difficult task a computer can tackle. Nowadays, big data processing is primarily carried out on specialised pieces of hardware, named *accelerators*, such as Graphical Processing Units (GPUs). The usage of GPUs has been highly aided not only by improvements in the hardware itself, but also by optimizations in the software and firmware ran on GPUs.

For big data, data transfer overhead between different memory spaces and extensive storage footprint, can become significant performance challenges. One solution to reduce this overhead is to *store* data, on persistent memory devices such as HDDs or SSDs, in a compressed form. However, *processing* the data requires it to be decompressed, which in turn adds extra compressing/decompressing steps between storage and processing units (e.g. GPUs or CPUs). Up until now, the path used by the system to load and decompress data into the GPU utilizes the system's CPU, main memory, and a PCIe bus. Specifically, data is loaded from I/O storage to the main memory of the system, decompressed by the CPU, and further stored in the main memory, from where it is transferred to the Video Memory (VRAM) of the GPU.

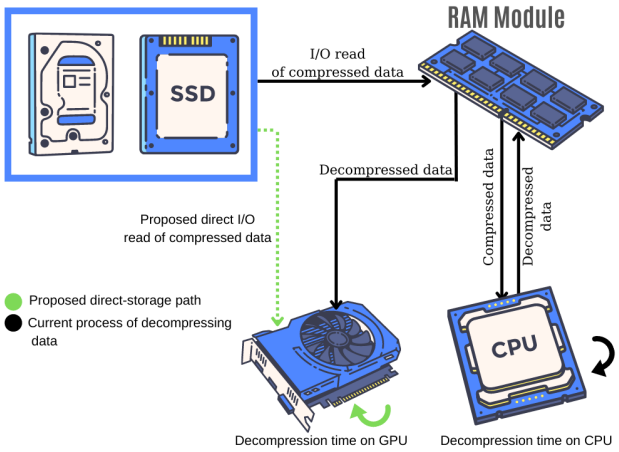

Figure 1. Current and proposed data transfer paths

New developments, like GPUDirect [22], enable direct SSD-to-GPU data paths. Thus, in this work, we investigate the benefits (and pitfalls) of bypassing the CPU and main system memory, and performing decompression in the GPU memory *directly*. Figure 1 shows a comparison between the current path and the one proposed by our research. It illustrates both the current path (i.e., via the CPU), and the proposed path, which streamlines the process. The new, novel, path offloads the decompression process from the CPU to the GPU, assuming (a) the compressed data transfer to GPU memory is shorter, and (b) the decompression on the GPU is not significantly slower than on the CPU. In fact, with the recent developments in GPU computation power, newer and faster decompression algorithms are being developed and implemented on GPUs [7, 20, 23, 29], leading to faster decompression times compared to CPUs.

In this work, we provide a quantitative comparison of these two ways to process big data with accelerators, and

TScIT 37, July 8, 2022, Enschede, The Netherlands

aim to determine what is the more efficient version for a specific case-study from particle physics (provided by CERN [1]). The main question that we aim to answer is:

Is there a performance benefit to use IO-to-GPU instead of IO-to-CPU-to-GPU for compressed data?

The following questions are aimed at building and supporting the main research question:

RQ1: What are accurate analytical models for the two system architectures?
RQ2: Which GPU decompression algorithms provide a speed-up in processing big-data ?
RQ3: What are guidelines to select the most effective (de)compression algorithm for the IO-to-GPU architecture?

2 Related Work

GPUs are a category of hardware accelerators designed to perform massively parallel tasks. One example of such a parallel task is matrix multiplication, a task suitable for the gaming industry which enables GPUs to be a cheaper option than similar hardware accelerators [6, 10]. GPU Decompression has only become viable in recent years, but considering the rate at which GPUs increase in performance [26], it is set on a path to be a very relevant optimization subject into the future. Moreover, as GPU compute power increases, the need for utilizing the full potential of GPUs requires and enables multitasking techniques [4, 33].

A 2010 paper on trends in both hardware and software development for GPUs[19], exemplifies the interest boom that GPUs had around a decade ago, as well as the developments that have been made in both GPU hardware and software. Moreover, the paper concludes with a call for direct connection with non-volatile storage devices, and claims that such a change may bring a new development revolution.

Furthermore, research into the benefits of direct access to non-volatile memory, such as [3], showcase benchmark results and implementations that yield a 35% decrease in data transfer times, as well as a 20% increase in end-to-end performance when using the new proposed path (IO-to-GPU - see Figure 1), bypassing the CPU, compared to the old path (IO-to-CPU-to-GPU). Benefits of direct storage access are further emphasized by [17] and [27], which present the discrepancy between slow storage devices and fast GPU processing times.

Many past works relate that the benefits of direct access to the non-volatile memory are relevant for larger I/O file sizes than for smaller I/O file sizes [3, 7, 21], thus proving the point that the technology is relevant for big-data environments where file size is constantly increasing.

Last but not least, there needs to be a mention to the recent advancements made by NVIDIA, a leader in the development of GPUs. The development and support of CUDA (Compute Unified Device Architecture) enables parallel processing on the GPU, reducing processing bottlenecks [24]. In addition to that, NVIDIA has been creating an easy to access API, named GPUDirect, that enables developers and researchers to easily create a direct link between the non-volatile memory and the GPU [22]. Further developments by NVIDIA include NVComp, a fast in-house built compression and decompression algorithm, specifically made to run on NVIDIA GPUs [23].

[1]CERN is the European Organization for Nuclear Research.

3 Methodology

To answer the main research question, we propose a three-stage research plan, driven by the following research milestones:

3.1 RQ1: Defining analytical models

We conduct a content-analysis of the typical systems and applications that use GPUs for big data processing. As a result of content-analysis, we provide a high level description of the system by means of an UML diagram. The diagram showcases what the components used for the interoperability of non-volatile memory and GPU are. Moreover, further analysis of the interaction between components will enable us to derive functional analytical models of the system(s) under consideration, capturing both the IO-to-CPU-to-GPU and IO-to-GPU configurations.

3.2 RQ2: Analysing GPU-decompression algorithms

To understand the feasibility of GPU decompression - in terms of functionality and speed - we conduct a literature study. The findings of this literature review will lead to a basic understanding of the current situation and limit the scope of the analysis. Based on the analytical models proposed (see *RQ1*), combined with the decompression speeds published in literature, we construct an initial list of candidate GPU decompression algorithms that could enable faster processing times in big-data applications. Furthermore, we conduct detailed experiments to benchmark (some of) these candidate GPU decompression algorithms, aiming to confirm their relevance.

3.3 RQ3: Development of guidelines for selecting a GPU decompression algorithm

Using parameters defined within the analytical model and the list of tested GPU decompression algorithms, we propose a set of guidelines for selecting the most appropriate decompression algorithm for a given application. The guidelines are based on parameters such as compression ratio and decompression speed, and indicate the best GPU decompression algorithm to be used such that the requirements of a stakeholder are best met. For our specific case-study, these guidelines enable CERN to make an informed choice when selecting the decompression algorithm suitable for their specific particle physics applications.

4 Analytical model

Any computer system is comprised of multiple inter - connected components, each acting at different parameters and speeds. Loading compressed data and transforming it into ready-to-use, decompressed data requires precise coordination between the components of a computer system.

To compare the proposed, *IO-to-GPU* system and the current, *IO-to-CPU-to-GPU* one, we focus on execution time. We further assume that runtime of the actual application is *not* affected by the data transfer, and, therefore, is the same for the two systems. Thus, the comparison will effectively focus on the time taken by each system to load the first data into the GPU (video) memory.

To compare the data transfer of the two systems, we develop an analytical model that can be used to determine the time taken by each system for this operation. Utilizing Figure 2 and Figure 3 we define an argument for the importance of each component and the time required by it.

Furthermore, data is transferred using the PCIe connection. The PCIe connection is a serial, low-latency bus used to interconnect many components within a compute system. The connection uses series of 2 channels, one for sending and one for receiving. Therefore, a naming scheme exists to express the number of simultaneous connections that can exist between two components.

For example, GPUs require an x16 connection to be able to operate at their full potential. Moreover, the current generation of PCIe , PCIe 4.0, provides a theoretical bandwidth of up to 64GB/s. Each generation of PCIe aims at doubling the effective transfer speed, version 5.0 aiming to provide a theoretical max speed of 128GB/s [11]. Moreover, PCIe can be used together with other technologies such as NVIDIA's proprietary High-Speed GPU Interconnect, *NVLink*[2] [31]

4.1 The traditional system: IO-to-CPU-to-GPU

As observed in Figure 2, the current system utilizes the RAM memory to store data until it is required by the CPU for decompression. During this transfer, we denote data as being in compressed form, thus expecting it to require less storage space, aiding the time of transfer ($T_{RAM-CPU}$).

With data in memory, the CPU performs the decompression, which varies in time depending on the specifications of the CPU and on the decompression algorithm itself. This time is denoted by $T_{Decompress}$. After the decompression process, the CPU temporarily stores data into system memory $T_{CPU-RAM}$ until it can be transferred to the video memory of the GPU. $T_{CPU-RAM}$ is expected to be greater than $T_{RAM-CPU}$ as the data is now in decompressed form, requiring more space and a longer transfer time.

Last, but not least, a data transfer exists between the main system memory and the GPU video memory ($T_{RAM-GPU}$). This transfer time depends on 3 parameters: (1) read speed

[2]https://www.nvidia.com/en-us/data-center/nvlink/

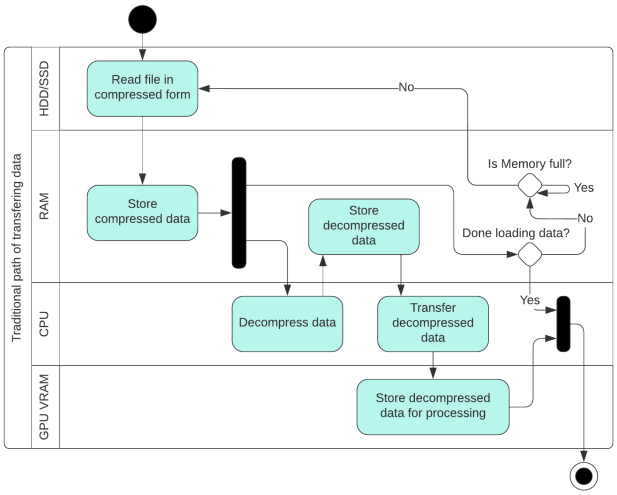

Figure 2. Traditional flow of transferring data

of the CPU RAM; (2) transfer speed of the PCIe bus; (3) write speed of the GPU VRAM. We note that accurately benchmarking (1) and (3) is cumbersome for modern systems, and, therefore, our PCIe benchmark measures all three components into a single $T_{RAM-GPU}$ value.

The total time of transferring data while utilising the traditional path is given by summing all the required times:

$$T_{I-C-G} = T_{RAM-CPU} + T_{Decompress} + T_{CPU-RAM} + T_{RAM-GPU} \quad (1)$$

4.2 The proposed system: IO-to-GPU

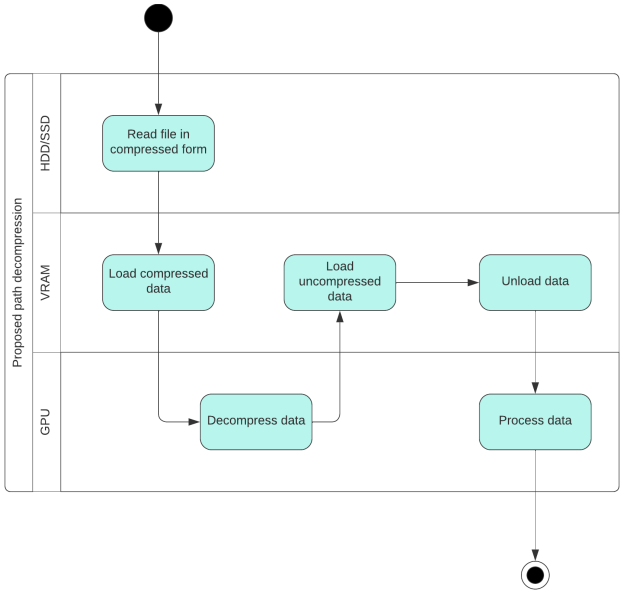

Figure 3. New proposed system workflow: IO-to-GPU

By analysing the proposed system following Figure 3, only three important times can be identified: (1) the transfer time required to load data from non-volatile storage to the

GPU (video) memory, (2) the decompression time required by the GPU ($T_{GPU-Decompress}$), and (3) the time required to store the decompressed data into the video memory again ($T_{GPU-VRAM}$).

We acknowledge that the non-volatile storage devices and their speeds are irrelevant for the analytical model, as the expected performance, while not throttled by other components, is assumed to be the same between the two systems, and, therefore, provides no added value to the comparison. Thus, we will disregard (1) from the final equation. Thus, the total time required for the proposed path consists of summing the times required for each transfer individually:

$$T_{I-G} = T_{GPU-Decompress} + T_{GPU-VRAM} \quad (2)$$

4.3 Complete equation

To ensure the new system (IO-to-GPU) provides better performance than the traditional version, the following must hold:

$$T_{I-G} < T_{I-C-G} \quad (3)$$

To further continue the analysis, we split the times into two categories: Decompression and Transfer. Due to a double transfer of uncompressed data supplemented by a transfer of compressed data compared to two transfers of compressed and uncompressed data respectively, the traditional system yields a lower transfer time compared to the proposed system.

Recent advances in GPU decompression algorithms [17, 20, 28, 29] indicate GPU decompression can outperform CPU decompression, even by as much as 5 times. In the following chapters, we will quantify both the transfer and decompression times for the two different systems, to determine when Equation 3 holds.

5 Evaluation

CERN[3] has provided a real use-case for the technologies presented in this paper. Specifically, this case-study focuses on ROOT, a framework for data processing, born at CERN, at the heart of high-energy physics research. Every day, thousands of physicists use the ROOT data format and analysis tools to analyse their data and/or to perform simulations. As ROOT continuously improves the performance of its tools, the developers want to determine whether the IO-to-GPU system is beneficial for their ROOT data-sets and tools by allowing for faster processing, storing, or accessing data. To this extent, they have provided multiple data-sets that were used to test the algorithms, and a brief description of scenarios of interest. In this section, we discuss in more details how we set-up and benchmark a representative case-study.

[3]CERN is the European Organization for Nuclear Research - https://home.cern/.

5.1 Compression and Decompression algorithms

Compression and decompression on GPUs has been extensively investigated in the past years [2, 7, 8, 17, 20, 23, 28, 29, 32], motivated by rapid GPU development. For our analysis, we investigate the capabilities of NVComp, an NVIDIA developed library aimed at simplifying the transition process of developers to GPU compression and decompression.

NVComp provides two C++ interface levels: (1) a high-level API, intended for new and inexperienced developers, and (2) a low-level API, allowing developers to tweak a larger set of parameters, potentially improving the compression / decompression further. The latest version of NVComp is *2.3* at the time of writing and includes support for 7 decompression algorithms, presented in Table 1.

Cascaded	High-throughput compressor ideal for analytical / tabular data
LZ4	A General-purpose byte-level compressor, suited for a wide range of use cases
Snappy	Similar to LZ4, however is a more popular format used for tabular data
GDeflate	An in-house built compressor based on entropy encoding and LZ77 providing high compression ratios
Deflate	This is a Huffman and LZ77 combination. It is provided as a compatibility compressor for existing deflate-compressed data-sets (such as GZip)
Bitcomp	A compressor built in-house that is designed to be used in scientific computing applications
ANS	An in-house entropy based compressor.

Table 1. Algorithms tested from NVComp version *2.3*

To evaluate the performance of *IO-to-CPU-to-GPU*, we also require performance data for CPU decompression algorithms (see Figure 2 and Equation 3). We have selected the following two CPU algorithms: *ZLIB*[4] and *FastLZMA2*[5]. We simply selected these two algorithms because they are two of the most used, and perform well on CPUs [12].

For the purpose of this research, we have analyzed the performance of each decompression algorithm on uncompressed data provided by CERN, with the end goal of selecting the most suitable algorithm(s) for the specific case (and data) from CERN.

5.2 Experimental Setup

In order to test the algorithms mentioned in the previous section, we have used a local system, named `showcees`, deployed at the University of Amsterdam, and multiple system configurations offered on the Distributed ASCI Supercomputer 6 (`DAS-6`) [1]. The hardware configurations of both

[4]https://zlib.net/

[5]https://github.com/conor42/fast-lzma2

systems are presented in Table 2. The showcees system has been used to develop and test benchmarks, whilst the DAS-6 has been used for performing the final benchmarks with GPUs such as *A4000, A6000* and *A100*.

	UvA Showcees	DAS-6
CPU	Intel Xeon Gold 6148	AMD EPYC-2 7402P AMD EPYC-2 7282
GPU*	NVIDIA GTX 1080Ti	NVIDIA RTX A4000 NVIDIA RTX A5000 NVIDIA RTX A6000 NVIDIA A100
RAM**	400GB	128 GB
OS	Cent OS 7	Rocky Linux 8

Table 2. Specification of hardware used
* - GPU configuration depends on the node tested
** - Defined per node.

Both systems run NVIDIA's CUDA Toolkit version 11.2 and NVComp version 2.2. Furthermore, the systems were also provided with NVIDIA GDS 1.3, enabling a direct path between the storage device and the GPU.

All tests were carried out in an automated way by means of a Python script, which allows setting multiple benchmark parameters such as iteration count (1), chunk size (2), or GPU warm-up (3). These parameters are explained in Table 3. Furthermore, the code used for testing can be found in the GitHub repository [9].

(1)	Defines the number of times the compression is run and returns the average of all runs
(2)	Defines the size of the chunks. In theory, higher chunk sizes allow for higher compression ratios at the expense of less parallelization.
(3)	The number of times the compression methods are accessed before running the benchmark. In practice warming up the kernel code can provide significantly better results [25].

Table 3. Parameters used for testing

Both systems have been tested for their PCIe latency using a benchmark provided by in NVIDIA's CUDA Toolkit. Table 4 illustrates the PCIe version, and the measured transmission speed from CPU to GPU and vice-versa.

System	PCI-E version	HtoD[6] speed	DtoH[7] speed
Showcees	3.0	12000 MB/s	12800 MB/s
DAS-6	5.0	25000 MB/s	16200 MB/s

Table 4. PCIe performance on both systems.

[6]Host-To-Device
[7]Device-To-Host

5.3 The ROOT data-sets

The case-study from CERN focuses on data using a novel, ROOT-specific data format. The data-sets are represented by columnar files called pages. Multiple pages can be concatenated directly, in order, to create an RNTuple. Each page has a limit of 65KB which is an inherited trait of ROOT, allowing for very fast write speeds when creating the page file.

The pages store data of different types: boolean, integer, float, and string. The data can be used for processing in different combinations. For our evaluation, we have assumed the following scenarios: (1) single-file and (2) multi-file. Specifically, in the case of single-file, we assume the processing requires a single page, while in the case of multi-file, we assume the processing requires multiple pages *at the same time*, potentially of different types.

6 Results

In this section we present a subset of our results from benchmarking NVComp for different application scenarios, using both systems and different algorithms. All data gathered through benchmarks has been collected in the project's GitHub repository[8] [9] .

6.1 Single-file applications

The first tests are performed on 5 randomly chosen files from the same data-set, containing integer values. Figure 4 shows the compression ratio across all algorithms for integer-like data types. We observe that compressors designed to deal with analytical or tabular data such as Cascaded, perform much better than general purpose algorithms such as Snappy or LZ4. This can also be seen in Table 5, where the throughput of each compressor is listed. The values in red are algorithms that resulted in a compression ratio lower than 1, deeming the compression useless. The green values are the next best values that provide a compression ratio above 1. Bold cells represent the highest (i.e., best) value in a certain column.

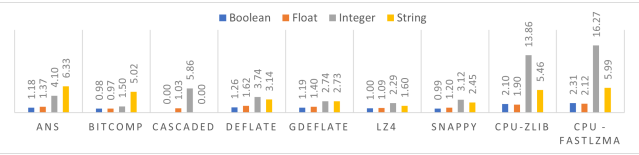

Figure 4. Average compression ratio across all algorithms

Moreover, Table 3 introduces one inter-changeable parameter within the tests, the chunk size, which defines smaller chunks that the GPU divides the data into such that it can perform parallel compression and decompression. Larger chunk sizes typically lead to higher compression ratios, at the expense of less parallelism exposed to the GPU. NVComp defaults to a chunk size of 64KB, which usually proves to be a good starting point for compressing data. However, 64KB

[8]https://github.com/AndreiArdei/NVCOMP_CompressionBenchmarks

Bool		Float		Integer		String		
Compress	Decompress	Compress	Decompress	Compress	Decompress	Compress	Decompress	
0.05	0.07	0.73	1.02	1.23	**2.13**	0.55	**0.86**	ANS
0.42	0.00	**7.68**	0.06	**7.85**	0.04	**3.19**	0.04	Bitcomp
0.00	0.00	3.33	**4.10**	3.94	3.05	0.00	0.00	Cascaded
0.02	0.02	0.22	0.30	0.18	0.40	0.05	0.13	Deflate
0.02	0.05	0.24	0.85	0.16	1.10	0.05	0.25	GDeflate
0.05	0.18	0.55	2.35	0.21	0.47	0.06	0.12	LZ4
0.08	**0.22**	**0.79**	**1.91**	0.28	0.98	0.06	0.26	Snappy
0.01	**0.18**	0.02	0.24	0.00	0.86	0.01	0.43	ZLIB
0.02	0.03	0.03	0.03	0.01	0.27	0.01	0.10	FastLZMA

Table 5. Compression and decompression throughput of each algorithm in GB/s. *In bold: the best results in a certain column; in Red: results with compression ratio less than 1; in Green: the best results per data type with compression ratio larger than 1.*

is too large a size compared to the 65KB file size of each individual page. We measured the impact of the chuck size by benchmarking, and the results are presented in Figure 5. We observe that, whilst a chunk size of 4KB does provide greater compression and decompression speeds, it shows an average drop of 15% in compression ratio. In contrast, higher chunk sizes do provide slightly higher compression ratios, but are slower. We further observe a clear drop-off in both the compression ratio and the (de)compression speeds at the 32KB mark. Intuitively, any chunk size larger than that of the page file performs the same, effectively using the entire file at once, and not performing any paralellization.

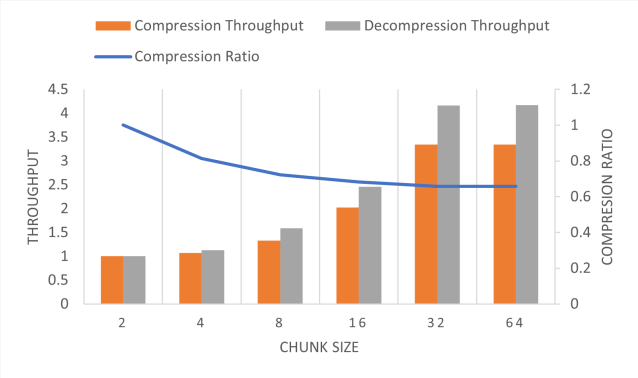

Figure 5. Average change of performance identified at different chunk sizes for a 65KB file.

6.2 Multi-file applications

We further considered the use of CUDA streams [16, 18, 30] as means of parallelizing compression and decompression tasks for multi-file applications. By default, all GPU tasks are submitted to a single queue (called *default stream*) and performed in FIFO[9] order, where each task has exclusive use of the full GPU (i.e., all available CUDA cores). In most cases, this is not an efficient way of performing (small) tasks, which have insufficient work to "fill" the GPU, and therefore under-utilize the system. To counter such underutilization,

[9]First In, First Out

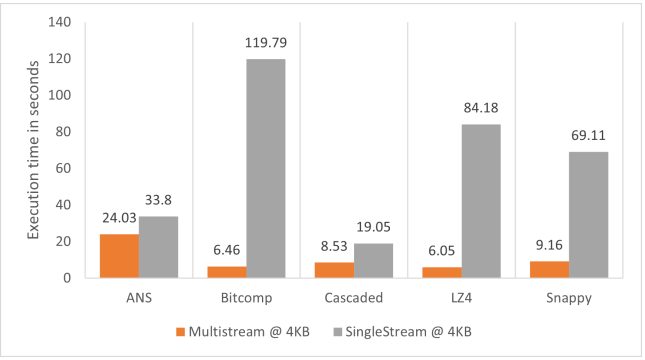

Figure 6. Compression and decompression results utilizing multiple streams performed at 4KB chunks

GPUs can make use of *multiple CUDA streams*. This allows tasks to be distributed over multiple queues, which are executed in parallel, thus providing more work to the GPU and enabling better utilization of the full GPU potential. For example, assuming a 3-task application,such an approach would prove beneficial - We define T_{seq} (Equation 4a) as the time required for the tasks running in sequential order, and T_{par} (Equation 4b) as the time required to when running the tasks in parallel over multiple parallel streams.

$$T_{seq} = T_1 + T_2 + T_3 \tag{4a}$$

$$T_{par} = \max(T_1, T_2, T_3) \tag{4b}$$

Previous benchmark runs have been performed using only one stream for compression and decompression. We have created a multi-stream benchmark that allows the GPU to compress and decompress multiple files in parallel, utilizing the performance of the CUDA cores more efficiently. The performance results gathered with this benchmark are presented in Figure 6. We note that our parallel implementation of *Deflate* and *GDeflate* does not perform correctly, therefore their performance has been omitted.

Furthermore, Figure 7 presents the performance deficit (that is, the additional time required for a sequential task compared to a parallel one) experienced in sequential runs compared to parallel runs as the chunk size increases.

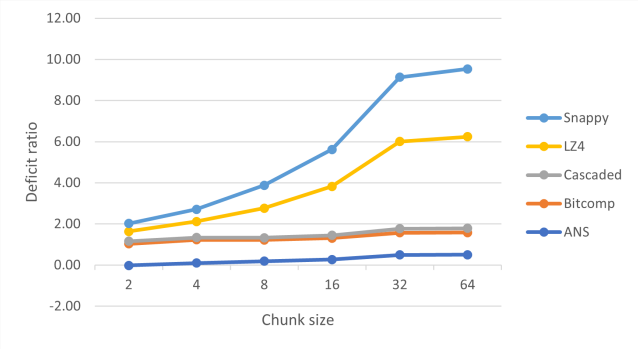

Figure 7. Performance deficit of sequential tasks compared to parallel tasks as chunk size increases.

6.3 Limitations

Throughout our empirical evaluation, we have identified a couple of limitations of our approach. To begin with, for the data provided by CERN, compression and decompression algorithms have proven to be highly dependent on the data type (e.g., double vs. string). A worst-case example of such a dependency, visible in Table 5, are IEEE 754 double-precision floating point numbers [5, 13, 14], which are prevalent in the provided data-sets. This 64 bit representation of floating point numbers proves difficult for compression algorithms to detect a relevant level of entropy and, therefore, the compression ratio is up to 5× worse than for other data-sets. Moreover, in some cases, the overhead created by the compression algorithm results in a compressed file larger than the initial uncompressed file - as seen in the *Float* column of Table 5. This issue can be improved upon by introducing split encoding in ROOTs file system, and thus allowing for better bit packing solutions or adopting more specialized algorithms such as [15].

7 Algorithm selection guidelines

	CPU Decompression	GPU Decompression
Bool	**6.97**	54.14
Float	140.82	**15.42**
Integer	5.42	**3.62**
String	13.51	**5.87**

Table 6. Times (in μs) as predicted by the analytical model.

Following the results from the previous section, as well as the application of section 4 in Table 6, we observe that no single algorithm fits all scenarios. Instead, the NVcomp algorithms perform significantly different for different data-sets. To effectively compare the use of decompression in our two systems, we use the models from Equation 1 and Equation 2 to calculate the overall performance for IO-to-CPU-to-GPU and IO-to-GPU, respectively. Table 7 shows the top three results[10]. We note that the PU-to-memory transfer times (i.e., $T_{RAM-CPU}$, $T_{CPU-RAM}$, $T_{GPU-VRAM}$) are all included in the decompression times during benchmarking (for CPU and GPU, respectively), while the $T_{RAM-GPU}$ is negligible for small file sizes such as the ones we used.

[10]The complete results are presented in [9]

	Compression	Decompression	Ratio
Bool	ANS	ZLIB	FastLZMA
Float	Cascaded	Cascaded	FastLZMA
Integer	Bitcomp	Cascaded	FastLZMA
Strign	Bitcomp	ANS	ANS

Table 7. Best result for each data type

Table 7 displays the best algorithms for compression, decompression and compression ratio in each of the data types. As can be seen, most algorithms appear in each column; however, a few patterns are worth mentioning. Algorithms performed on the CPU show a clear win in all data types when considering the compression ratio, providing an increase factor of at least 2 times. Furthermore, GPU algorithms can utilize their parallelism capabilities to provide much faster compression and decompression throughput performance.

We reinforce that GPU decompression is highly influenced by the chunk size, with smaller chunk sizes resulting in significantly higher throughput for both compression and decompression. However, the increase in throughput comes at the cost of compression ratio. Additionally, Figure 6 shows that we can further exploit parallelization on the GPU by utilizing multiple streams, transforming the sequential processing time in a parallel one, as illustrate in the example in Equation 4a and Equation 4b.

When considering which algorithm to use, the needs of the application play an important role. For example, if compression is used prior to storing data long term, a CPU-based algorithm may offer better performance, due to its increased compression ratio, that results in lower file size. If the goal of compression is to stream data faster, GPU-based algorithms may be considered as the increased parallelism allows for higher throughput of data, however compressing this data with the required algorithm is proven to be slow in general.

8 Conclusion and Future Work

Developments in GPU technology have enabled new ways to transfer, process and utilize data. In this context, we analysed the potential of improving GPU-based big-data analysis by using direct I/O to GPU data transfers. Our analysis stems from a CERN case-study, where data is stored in compressed format, and needs to be processed on a GPU. Thus, we evaluated the performance difference of data transfers with and without the CPU in the loop. The two data paths differ in transferred data volume (compressed or decompressed) and decompression algorithm (CPU- or GPU-based).

8.1 Summary and main findings

To determine whether GPU-based decompression can improve big-data processing when data is stored in a compressed format, we formulated three research sub-questions (see section 1). We answer them as follows.

RQ1: What are accurate analytical models for the two system architectures?

We created analytical models for both systems incorporating GPU hardware. We acknowledge two important limitations of our models. First, they do not include data type as parameter. Our work proved that data types play an important role in (de)compression performance. Second, some of the parameters (i.e., PU to RAM transfers) in these models cannot be measured in isolation; however, our measurements do include them correctly.

We further used the models to estimate the perfomrance of the two GPU systems for single file applications. Our results demonstrate the models can capture the differences between the two architectures, and select the best performing one when calibrated with measured data.

RQ2. Which GPU decompression algorithms provide a speed-up in processing big-data?

For GPU decompression, we focused on algorithms provided by NVComp; for CPU-decompression, we used two popular algorithms. Out of the 6 GPU algorithms, we were unable to determine a best-fit algorithm. We have instead found that data types have a significant impact on the performance of each algorithm. Moreover, when compared to the performance of CPU algorithms we notice a significant increase in throughput, mainly due to the GPU parallelism. We conclude that GPU decompression can provide a speed-up over CPU decompression, and can be used to accelerate big-data processing. Taken together with the faster transfer of compressed data, these results indicate that the proposed data path in this work (i.e., IO-to-GPU) is likely to outperform the traditional one (i.e., IO-to-CPU-to-GPU).

RQ3. What are the guidelines to select the most effective (de) compression algorithm for the GPU ?

Based on our empirical data, we found no winning algorithm. However, using the analytical model, we have developed a guideline system that allows the user to choose the best-fit algorithm for their application. Results are refined using *data type, compression ratio, decompression throughput* as parameters. The guidelines identify that for the intent of storing files in compressed form, CPU algorithms still outperform GPU ones, however once the intent is that of 'on-the-fly' decompression, GPUs are capable of providing a many fold increase in throughput.

Finally, we are able to answer our main research question: ***Is there a performance benefit to use IO-to-GPU instead of IO-to-CPU-to-GPU for compressed data?***. The results in Table 5 and Table 6 show that utilizing an IO-to-GPU architecture enables much greater decompression speeds compared to utilizing an IO-to-CPU-to-GPU architecture. This improvement can be seen especially when the uncompressed data is needed on the GPU, therefore eliminating transfer times and requiring a single transfer of compressed, thus smaller in size, data.

8.2 Limitations and threats to validity

We discuss three potential limitations of this work. First, this paper analyses the performance of GPU decompression algorithms through the NVComp framework and does not consider other, possibly more effective, algorithms. Second, our analysis is based on data from a specific case-study, with small files (around 65KB). This means that the GPU is required to perform more tasks of loading and unloading data from IO, potentially limiting its performance. However, we expect that larger data files will further boost the performance of the GPU, further reinforcing our conclusion that IO-to-GPU is the better architecture. Third, and final, we have analysed the performance of the data path only, without taking into account any specific applications. One example when this is a limitation is the single-file scenario. When transferring a single, small file we cannot fully utilize the throughput of the PCIe.

8.3 Future Work

We identify three directions of future work.

First, the analytical model can still be improved in different ways. (1) Our results (see section 6 indicate data types affect the performance of each algorithm, but are not yet taken into account in the model. (2) Furthermore, we demonstrated successful multi-stream decompression, but this was also not included in the model. (3) Last, but not least, the model can could also the time taken by the processing kernel, thus providing a more accurate comparison between the two systems, which would include both the data flow and the data processing. Such an extended model, for a GPU kernel processing n files, $f_i, i = 1..n$ of data types d_i, would be:

$$T^{new}_{I-G} = \max_{i=1..n}(T_{GPU-VRAM}(f_i) + T_{GPU-Decompress}(f_i, d_i)) + T_{process}(f_1, f_2, ..., f_n)$$

Further analysis and validation are, however, required before adopting this new model.

Additional research is also needed to validate the full model, including the I/O to PU performance. Such validation was unfeasible for the current remote machines due to technical reasons.

Finally, in the current research, we were required to run separate benchmarks at multiple chunk levels in order to find peak performance. Therefore, a different, area of research is a formal definition of chunk size. Analysing if it can be statistically defined as a percentage of the original data size, thus enabling developers to fine-tune their algorithms for improved performance.

References

[1] Bal, H., Epema, D., de Laat, C., van Nieuwpoort, R., Romein, J., Seinstra, F., Snoek, C., and Wijshoff, H. A medium-scale distributed system for computer science research: Infrastructure for the long term. *Computer 49*, 5 (2016), 54–63.

[2] Balsa Rodríguez, M., Gobbetti, E., Iglesias Guitián, J., Makhinya, M., Marton, F., Pajarola, R., and Suter, S. State-of-the-art in compressed gpu-based direct volume rendering. *Computer Graphics Forum 33*, 6 (2014), 77–100.

[3] Bayati, M., Leeser, M., and Mi, N. Exploiting gpu direct access to non-volatile memory to accelerate big data processing. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)* (2020), pp. 1–6.

[4] Bridges, R. A., Imam, N., and Mintz, T. M. Understanding gpu power: A survey of profiling, modeling, and simulation methods. *ACM Comput. Surv. 49*, 3 (sep 2016).

[5] Brisebarre, N., Mezzarobba, M., Muller, J.-M., and Lauter, C. Comparison between binary64 and decimal64 floating-point numbers. In *2013 IEEE 21st Symposium on Computer Arithmetic* (2013), pp. 145–152.

[6] Campeanu, G., Carlson, J., and Sentilles, S. Developing cpu-gpu embedded systems using platform-agnostic components. In *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (2017), pp. 176–180.

[7] Funasaka, S., Nakano, K., and Ito, Y. A parallel algorithm for lzw decompression, with gpu implementation. In *Parallel Processing and Applied Mathematics* (2016), Springer International Publishing, pp. 228–237.

[8] Funasaka, S., Nakano, K., and Ito, Y. A parallel algorithm for lzw decompression, with gpu implementation. In *Parallel Processing and Applied Mathematics* (Cham, 2016), R. Wyrzykowski, E. Deelman, J. Dongarra, K. Karczewski, J. Kitowski, and K. Wiatr, Eds., Springer International Publishing, pp. 228–237.

[9] Gorgan, A. Cern study case - data and charts, Jun 2022.

[10] HajiRassouliha, A., Taberner, A. J., Nash, M. P., and Nielsen, P. M. Suitability of recent hardware accelerators (dsps, fpgas, and gpus) for computer vision and image processing algorithms. *Signal Processing: Image Communication 68* (2018), 101–119.

[11] Harding, S. What is pcie? a basic definition, Feb 2021.

[12] Inikep. Inikep/lzbench: Lzbench is an in-memory benchmark of open-source lz77/lzss/lzma compressors.

[13] Jaiswal, M. K., and Chandrachoodan, N. Efficient implementation of ieee double precision floating-point multiplier on fpga. In *2008 IEEE Region 10 and the Third international Conference on Industrial and Information Systems* (2008), pp. 1–4.

[14] Kahan, W. Ieee standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE 754*, 94720-1776 (1996), 11.

[15] Knorr, F., Thoman, P., and Fahringer, T. Ndzip-gpu: efficient lossless compression of scientific floating-point data on gpus. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2021), pp. 1–14.

[16] Li, H., Yu, D., Kumar, A., and Tu, Y.-C. Performance modeling in cuda streams—a means for high-throughput data processing. In *2014 IEEE international conference on big data (big data)* (2014), IEEE, pp. 301–310.

[17] Li, J., Tseng, H.-W., Lin, C., Papakonstantinou, Y., and Swanson, S. Hippogriffdb: Balancing i/o and gpu bandwidth in big data analytics. *IEEE 9*, 14 (oct 2016), 1647–1658.

[18] Micikevicius, P. Multi-gpu programming. *GPU Computing Webinars, NVIDIA* (2011).

[19] Neelima, and Raghavendra, P. S. Recent trends in software and hardware for gpgpu computing: A comprehensive survey. In *2010 5th International Conference on Industrial and Information Systems* (2010), IEEE, pp. 319–324.

[20] Noordsij, L. Parallelization of variable rate decompression for gpu acceleration, Jun 2019.

[21] NVIDIA. Gtc silicon valley-2019: Efficient distributed storage i/o using nvme and gpudirect in a pcie network, Apr 2019.

[22] NVIDIA. Magnum io gpudirect storage, Feb 2022.

[23] NVIDIA. Nvcomp, Mar 2022.

[24] NVIDIA. What is cuda, Jan 2022.

[25] Ottoni, G., and Liu, B. Hhvm jump-start: Boosting both warmup and steady-state performance at scale. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)* (2021), pp. 340–350.

[26] Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. E., and Purcell, T. J. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum 26*, 1 (2007), 80–113.

[27] Pandey, S., Kamath, A. K., and Basu, A. *GPM: Leveraging Persistent Memory from a GPU*. Association for Computing Machinery, 2022, p. 142–156.

[28] Plauth, M., and Polze, A. Gpu-based decompression for the 842 algorithm. In *2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW)* (2019), pp. 97–102.

[29] Sitaridi, E., Mueller, R., Kaldewey, T., Lohman, G., and Ross, K. A. Massively-parallel lossless data decompression. In *2016 45th International Conference on Parallel Processing (ICPP)* (2016), pp. 242–247.

[30] Sourouri, M., Gillberg, T., Baden, S. B., and Cai, X. Effective multi-gpu communication using multiple cuda streams and threads. In *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)* (2014), IEEE, pp. 981–986.

[31] Temuçin, Y. H., Sojoodi, A., Alizadeh, P., and Afsahi, A. Efficient multi-path nvlink/pcie-aware ucx based collective communication for deep learning. In *2021 IEEE Symposium on High-Performance Interconnects (HOTI)* (2021), pp. 25–34.

[32] Yong, K. K., Chua, M. W., and Ho, W. K. Cuda lossless data compression algorithms: A comparative study. In *2016 IEEE Conference on Open Systems (ICOS)* (2016), pp. 7–12.

[33] Zhao, C., Gao, W., Nie, F., and Zhou, H. A survey of gpu multitasking methods supported by hardware architecture. *IEEE Transactions on Parallel and Distributed Systems 33*, 6 (2022), 1451–1463.