Two High-Performance Alternatives to ZLIB Scientific-Data Compression

Samuel Almeida[1], Vitor Oliveira[2], António Pina[2], and Manuel Melle-Franco[3]

[1] Departamento de Informática, Universidade do Minho, Campus de Gualtar, Braga, Portugal
[2] Laboratório de Instrumentação e Física Experimental de Partículas, Campus de Gualtar, Braga, Portugal
[3] Departamento de Informática, Centro de Ciências e Tecnologias da Computação, Universidade do Minho, Campus de Gualtar, Braga, Portugal
{sam.alm321,manuelmelle}@gmail.com, {vspo,amp}@di.uminho.pt

Abstract. ZLIB is used in diverse frameworks by the scientific community, both to reduce disk storage and to alleviate pressure on I/O. As it becomes a bottleneck on multi-core systems, higher throughput alternatives must be considered, exploring parallelism and/or more effective compression schemes. This work provides a comparative study of the ZLIB, LZ4 and FPC compressors (serial and parallel implementations), focusing on CR, bandwidth and speedup. LZ4 provides very high throughput (decompressing over 1GB/s versus 120MB/s for ZLIB) but its CR suffers a degradation of 5-10%. FPC also provides higher throughputs than ZLIB, but the CR varies a lot with the data. ZLIB and LZ4 can achieve almost linear speedups for some datasets, while current implementation of parallel FPC provides little if any performance gain. For the ROOT dataset, LZ4 was found to provide higher CR, scalability and lower memory consumption than FPC, thus emerging as a better alternative to ZLIB.

Keywords: Data Compression, Scientific Data, ROOT, Parallel Compression, ZLIB, LZ4, FPC.

1 Introduction

Technological developments have lead to increasingly more powerful data processing systems, higher-resolution sensors and higher bandwidth communication networks. Yet systems are getting increasingly farther from Amdahl's balanced computing system [1], as the abilities to create, process, distribute and store data have not grown equally. In particular, the availability of multi-core systems has greatly amplified the ratio between the available computational power and the available input/output bandwidth [2]. As the potential of "big data" repositories in our society is explored [3,4], this balance between the system's ability to analyse, to transmit and to store data becomes even more important.

B. Murgante et al. (Eds.): ICCSA 2014, Part IV, LNCS 8582, pp. 623–638, 2014.
© Springer International Publishing Switzerland 2014

The same applies to the techniques that can affect it, such as caching, pre-fetching and compressing the required data.

Data compression, in particular, has been used for many years to trade computing power for storage capacity as well as for network and storage bandwidth whenever the first more abundant than the later ones. It has been used to improve disk space utilization with compressed packages or file-systems, to speedup WAN traffic over slow links and to increase write bandwidth in some solid-state disks. It also presents a large potential for specialized libraries such as NetCDF, HDF5 and ROOT to store scientific data in more compact forms. Taking into account the 2:1 file size reduction allowed by the LZ77 compression, very significant savings in terms of network bandwidth and disk space are achieved in datasets such those from the LHC's ATLAS (A Toroidal LHC Apparatus) experiment [5], which start in the multi-petabyte range, and that after several pre-processing stages, the data reaches the analysis applications still in the tens of terabytes range.

In [6] it was identified that in the analysed application the processing time associated with compression was indeed significant, where the ZLIB library took 18% of the total CPU-time to expand data prior to analysis. Some of that overhead can be offset by the reduced time associated with writing and reading less data to disk, as long as compression/decompression bandwidth exceeds storage bandwidth. But modern storage systems with well behaved usage patterns can provide much more bandwidth than ZLIB decompression can handle, both because a single modern disk is already capable of sequentially reading faster than the 120 MB/s decompression bandwidth achieved by ZLIB in a single processor core (Intel Xeon E5620) and because ZLIB does not support multiple threads.

As ZLIB compression and decompression becomes a bottleneck, techniques must be devised to improve performance while maintaining the advantages of compression. This work focuses both alternative compression methods, that could provide higher bandwidth, and implementations with a level of parallelism, that could properly explore multi-core systems. We present a comparative performance study of six compressors, in terms of bandwidth, compression ratio (CR) and scalability, using a group of different numeric scientific datasets. This approach permits assessing the compressor behaviour in the context of ROOT files and a few other application areas where compression can be beneficial.

This work focuses on the study of the serial compressors gzip, LZ4 and FPC and their respective multi-threaded counterparts pigz, lz4mt and pFPC.

2 Background Information

Scientific data compression has been used extensively, and *in situ* techniques are also on the rise [7,8,9]. For instance, in the ROOT toolkit [10], compression is used to deal with large volumes of information such as that generated by the Large Hadron Collider's (LHC) experiments. The approach there is to compress objects with the general purpose ZLIB or LZMA compressors prior to writing the data nodes to disk.

ZLIB and LZMA compression is asymmetric, the decompression time can be very small compared to the compression time, which makes it fitting for data that has to be read multiple times. Although they achieve very good CR on general purpose data, it is at the cost of both being time-consuming compressors. Several high-bandwidth compression algorithms have been developed in recent years, sacrificing some CR for faster execution times. LZ4 is a good representative of the modern high-performance compressors, and a potential alternative to ZLIB for ROOT, as it is about an order of magnitude faster than ZLIB when compressing, and around five times faster at decompression [6].

Recent research has focused on the development of compression techniques that explore domain knowledge to achieve higher CR or increased throughput, including techniques with large potential such as compressive sensing, but different application domains may require different techniques and yield very different results, so, that potential is hard to explore. The FPC lossless compressor is one example that uses context information to effectively compress and decompress 64-bit double precision floating-point data. Another one is the compression scheme based on entropy coding that is used in ALICE (A Large Ion Collider Experiment) hosted at CERN, where the differences of the times in two consecutive bunches (group of adjacent samples coming from the sensor pad), produced by ALICE's Time Projection Chamber detector, are coded [11], reducing the entropy of the source by exploiting the time correlation present in the data.

2.1 Compressor Algorithms

The gzip compressor implements the DEFLATE algorithm, which is an evolution of original LZ77 [12]. It is a well-known general-purpose compression system, providing high CR at the cost of performance, due to the use of a form of entropy encoding (Huffman coding). The algorithm is very asymmetric, with compression taking between 2.5 and 10 times the decompression speed, depending on the compression level selected and the input data.

LZ4 by Collet [13] is also a lossless compressor based on LZ77 algorithm, but on current multi-core systems it can reach throughputs of more than 400 MB/s per core when compressing, while during decompression it can achieve more than 1.8 GB/s[1], bound only by RAM bandwidth. The algorithm works by finding matching sequences and then saving them in a LZ4 sequence using a token, that stores the literals length (uncompressed bytes) and the match length, followed by the literals themselves and the offset to the position of the match to be copied from (i.e. a repetition). There are optional fields for literals and match length if necessary, and the offset can refer up to 64 KB. With the offset and the length of the match the decoder is able to proceed, and copy the repetitive data from the already decoded bytes. The simplicity of the algorithm together with the fact that entropy coding is not used makes LZ4 decompression very fast.

FPC [14,15] is a lossless compression algorithm for linear streams of 64-bit floating-point data. This compressor is based on a fast compression algorithm,

[1] `http://code.google.com/p/lz4/` Accessed March 6th.

tailored for scientific floating-point data compression on high-performance environments, where low latencies and high throughput are essential. The FPC algorithm can be implemented entirely with fast integer operations, resulting in a compression and decompression time one to two orders of magnitude faster than other more generic algorithms. It starts by predicting each value in the sequence and performing an exclusive-or operation (`xor`) with the actual value. A good prediction results in a substantial number of leading-zeroes in the calculated difference, which are then encoded by simply using a fixed-width count. After each prediction the predictor tables are updated with the actual double value to ensure that the sequence of predictions are the same during both compression and decompression. The remaining uncompressed bits are output in the end, after the count of leading-zeroes.

2.2 Block-Oriented Parallel Compression

When a stream can be divided in blocks, and in order to increase throughput, the processing can be parallelized as each block is delivered to a different thread and handled independently. But this approach carries some restrictions, as resulting compressed blocks have to be joined into the final output and that may or may not be a simple operation, depending on how the stream is concatenated. While LZ4's output stream is byte aligned, in ZLIB the intermediate output streams are bit aligned, so concatenating them implies a time-consuming data shift. To bypass the shift, intermediate blocks have to be terminated explicitly, which consumes more space (5-6 bytes per block) on the final output stream.

The compressor can also take longer to reach an effective dictionary, lowering the CR, as the contextual information is less rich. Due to this, it should be noted that block parallelism is effective only when processed in blocks of a significant size, and, produces less compressed outputs than a single block implementation.

The use of large data blocks may also not be straightforward. ROOT files, for example, have a tree-like internal structure in which only data nodes are compressed, with many being less than 10 K bytes. Multiple threads can still be used in this case, each writing a different data node, but it requires that multiple threads access ROOT's internal structures concurrently, which is only possible after ROOT version 6.

2.3 Multi-pass Compression

When data streams must be compressed immediately as they reach the compressor (such as low-memory or low-latency communication systems), or when forced to use small data blocks, improving performance by block parallelization becomes more difficult. Multi-pass compression is an alternative approach that, in very particular circumstances that depend on the type of input data, may yield significant compression rates.

Multi-pass compression consists of performing multiple passes of the compressor, with a high-bandwidth compressor on a first stage and a high-compression compressor in a second stage. The *rationale* is that a high-performance first

stage compressor can reduce the stream significantly, leaving a more thorough analysis to the second compressor that has less data to deal with. To use this technique the higher level redundancy must remain visible to the second compressor, which happens with LZ4 but not so much with the bit-oriented output streams generated by ZLIB. It also requires that not all redundancy is seen by the first level compressor, otherwise the second level will see no compression improvement. And while a more capable first stage compressor could be aware of the higher-level redundancies and eliminate them, a real advantage of this approach is that it may achieve similar results in less time.

In very redundant log data the results are surprising enough to be worth mentioning[2], but we found no evidence of this being the case in our scientific-data streams. In particular, we found that a 3.5 GB binary log file from a profiling application could be compressed with ZLIB to 54 MB on a first pass and to 17 MB in a second. With LZ4 on both stages, the first pass left the file at 56 MB and the second at just over 9 MB. With high-compression LZ4 (LZ4HC) on both stages, the first pass got a 44 MB file, the second pass a 2 MB, and after a few more rounds the file was left with a 750 KB size. In terms of throughput, a single step from that application using ZLIB took 265ms, while the same step with LZ4 and two additional passes of LZ4HC took only 16ms. It also proved effective with some text files, where one pass with LZ4 and one with LZ4HC ended up producing a similar size file as a single LZ4HC, in one fifth of the time.

3 Evaluating Compressor Performance

In this section the main considerations regarding testing the compression performance are presented, including characterization of the datasets and the testing procedures.

3.1 Evaluation Datasets

The compressors were tested with several numerical datasets from different backgrounds and sources (see Table 1). The six different disciplines covered by the 33 datasets, mostly from simulation programs, go from molecular and electronic structure modelling, through message, numeric and observational data to particle collision simulation data. After an initial evaluation of the 33 datafiles, a selection of five was made consisting of only one datafile per datagroup, based on its properties and characteristics being representative of each group (Table 2). The entire dataset can be consulted on Table3.

3.2 Dataset Preparation

The datafiles used came in many different source formats (some in binary and others in text format). FPC compresses only double-precision floating-point

[2] A public discussion on this topic is available at
https://groups.google.com/forum/#!msg/lz4c/DcN5SgFywwk/AVMOPri0O3gJ.

Table 1. Characteristics of the datagroups

Datagroup	#Files	Research Area	Software	Data Type
waterglobe	6	water nanodroplet	TINKER	text
engraph	3	graphene flake	TINKER	text
gauss09	4	graphene nanoribbon	Gaussian 09	text
sci-files	13	message, numeric, observational	*diverse sources*	doubles
NTUPs	7	particle collision simulation	LIP code	ROOT files

data, so in order to compare the compressors properly the input files had to undergo some transformation in order to be stripped of additional information and converted to binary, 64bit floating-point numbers.

3.3 Dataset Information Content

The information contents of the datasets are presented in Table 2. Equation (1) describes the percentage of uniques in a dataset, where V is the original vector consisting of all values, and V_{Unique} is the vector with duplicates removed.

$$\text{Uniqueness} = \frac{|V_{Unique}|}{|V|} \times 100\% \,. \tag{1}$$

$$H(V) = -\sum_{i=1}^{N} p(x_i) \times log_2 p(x_i) \,. \tag{2}$$

$$\text{Randomness} = \frac{H(V)}{H(\text{Random}_{unique}(|V|))} \times 100\% \,. \tag{3}$$

Equation (2) represents the Shannon entropy $H(V)$, where N is the number of distinct elements x_i, and $p(x_i)$ the probability of those elements, i.e., the number of x_i occurrences divided by the total number of elements in the file. An element of a dataset depends on the datatype that composes it (8bits ASCII, 32bits single, 64bits double). The randomness is closely related with the entropy as described in (3). Its value reflects how close the Shannon entropy of the datafile is to that of a true 100% unique random datafile with the same number of elements.

These datasets have high degrees of random entropy, in average 81.43%, which indicate that entropy coding will not be very effective and low compression ratios should be expected. The molecular mechanics data correspond to Cartesian coordinates and velocities which by definition have very low content in zeros. Interestingly, the waterglobe.vel has a lower value of randomness than the Cartesian coordinates as the atomic velocities on a molecular dynamics simulation of an equilibrated system are by definition periodical. The electronic density matrix (gauss09_density) was chosen as it has a very low zero content and, expected, high randomness. It is relevant to point out that engraph and the gauss09_density are calculations on graphene systems which are highly regular.

The uniqueness varies more and reaches an average of 44.66%, whilst some files barely contain unique values, others are almost entirely composed of them.

Table 2. Information details from the selected five datasets. R-ness is the Randomness.

Datafiles	Size(MB)	#elements	Unique%	Zeros%	Entropy	R-ness%
waterglobe.vel	1318	172800000	3.30%	0.00%	21.466	78.44%
engraph1_100	650	85190400	72.88%	0.00%	25.664	97.42%
gauss09_density	128	16753366	39.78%	0.05%	22.535	93.90%
msg_sp	277	36263232	98.95%	0.00%	25.032	99.68%
NTUP2_floats	1433	375644746	28.70%	38.77%	15.116	53.07%
`AVG (all)`	860	163954134	44.66%	10.47%	NA	81.43%

What is interesting is that even the files with low uniqueness are highly random (high randomness%). With this early but quite insightful statistical characterization we can already predict that NTUP datagroup (only one shown) should have the best CR with entropy coders (gzip/pigz). No prediction can be made for the FPC compressor, as that would require knowledge about the smoothness, or data continuity of the datasets, which was not analysed.

The metrics calculated here agree with the values from the sci-files datagroup as described in [15,7]. The overall dataset seems well balanced, with datafiles that cover many possible combinations.

3.4 Measuring Parallel Scalability

In order to assess the scalability potential of the algorithms two metrics were considered: speedup ratio and parallel efficiency. Speedup compares the performance of the parallel version to that of the serial version. It is the ratio between the execution time of a compression cycle using the serial compressor and the execution time of the same compression using the multi-threaded compressor, as shown in (4):

$$\text{Speedup} = \frac{\text{exec time}_{serial}}{\text{exec time}_{parallel}} \implies Sp_t = \frac{T_s}{T_t} \,. \tag{4}$$

where T_s is the execution time of the serial version and T_t is the execution time of the parallel version, executed with t threads.

Efficiency is the ratio of the achieved speedup with the expected maximum gain, as defined in (5):

$$\text{Efficiency} = \frac{\text{Speedup}_{parallel}}{\text{\#threads used}} \implies Ef_t = \frac{Sp_t}{t} \,. \tag{5}$$

where Sp_t is the measured speedup for that number t of threads used.

3.5 Parallel Decompression

In pFPC the decompression is symmetric to compression, so in both cases data is chunked and assigned to threads as needed. Due to its internal mechanics,

pFPC performs slower on decompression, but with an approximately linear behaviour where higher compression levels come with increasing compression and decompression times.

In ZLIB and LZ4 the decompression is simpler and much faster than compression. In fact, a single LZ4 decompression stream reaches a large portion of the available memory bandwidth. The decompression does not depend on the selected compression level (asymmetric), and there is very limited opportunity to explore parallelism. In fact, in lz4mt and pigz, decompression in parallel is not really implemented. Although pigz uses a single main thread for decompression, it creates three other threads for reading, writing, and checksum calculations, which can speed up decompression under some circumstances.

3.6 Test-Bench Characterization

The results presented in the following section were performed in cluster nodes containing two six-core Intel Xeon X5650 @ 2.66 GHz CPUs and at least 12 GB of RAM. Hyper-Threading technology was enabled, given that integer performance benefits from it, so a maximum of 24 threads per node were tested.

All timing measurements are represented by the walltime reported by the routine omp_get_wtime() from the OpenMP API. To obtain consistent results, for each file there are *nRuns* executions of compression and decompression per compression level (dependent on the compressor), written to the local hard drive disk and to /dev/null (i.e. data is discarded), which are then transferred compressed (but not measured), through the local network, as evaluated in [16].

The parallel scalability tests ran the same sequential tests, and the number of concurrent threads used (*nthreads*) is passed as a parameter. In each compression/decompression loop, the call for the *compress(file)* or *decompress(file)* receives *nthreads*. The amount of *nthreads* used correspond to the sequence (1,2,4,6..24), i.e. one, two, four, six... until twenty four, hence thirteen different tests in total.

4 Results and Discussion

The following sections present the CR, serial and parallel throughput, speedup ratio and memory requirements of the aforementioned compressors, as described in the previous section.

4.1 Compression Ratio and Serial Throughput

Figure 1 depicts the throughput of the three serial compressors, for both compression and decompression, and the CR yield for the datafile NTUP2. Interval bars are used for the three metrics, representing the minimum and maximum values measured from the five selected datafiles. Three compression levels are used for each compressor, with the exception of LZ4 that only has two modes. The levels are the minimum compression allowed by the algorithm, a selected

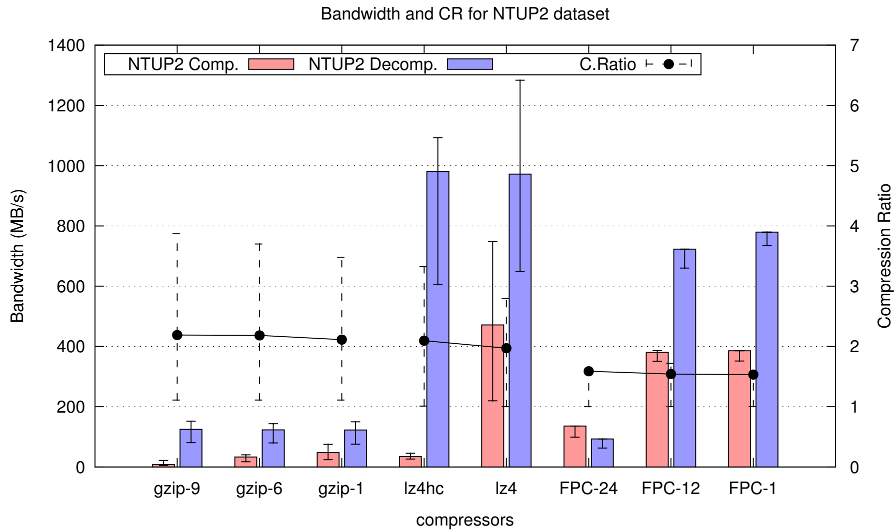

Fig. 1. Bandwidth of compression and decompression from the three different algorithms using NTUP2. The attained compression ratio for each compressor is read on the right y axis.

intermediate level and the maximum (FPC compression has no theoretical maximum, but is bound by the amount of RAM).

The bars in Fig.1 demonstrate immediately the performance advantage of both LZ4 and FPC compared to gzip's lower compression and decompression bandwidth. The compression bandwidth is largely dependent of the compression level selected. In case of gzip (ZLIB), while the highest level of compression is barely readable (8 MB/s), the lowest level is able to reach 48 MB/s. In the case of LZ4 the difference is dramatic, as the highest level of compression reaches 34 MB/s while the lowest reaches 472 MB/s. Regarding FPC, it ranged from 136 MB/s to almost 400 MB/s.

Comparing to the 120 MB/s of ZLIB, LZ4 showed a decompression bandwidth of 1.3 GB/s for one of the five datafiles. For NTUP2, LZ4 reached almost 1 GB/s, while FPC decompression bandwidth ranged from almost 800 MB/s and down to 63 MB/s (depending on the compression level).

Nevertheless, ZLIB provides the highest CR of 2.19 (with a maximum of 3.87), followed by lz4hc with 2.10 (with a maximum of 3.33), lz4 with 1.97 (with a maximum of 2.80), and finally FPC with 1.59. For some other datasets FPC reached the highest CR by a huge difference, e.g. datafile num_plasma attains a CR of 15 (see Table 3), but for the selected five the maximum CR of FPC is only 1.72 (reached when working with the gauss09_alpha dataset) and compression level 12. The CR varies a lot because the advantages of domain knowledge and the heuristics used depend strongly on the data.

4.2 pFPC Irregular Compression Ratio

When it comes to the variability of CR there are some unexpected events with pFPC, as higher compression levels may lead to lower compression. pFPC assigns each thread with chunks, representing a quantity number of double-precision floats to compress (8192 was the elected chunk size), and as more threads are used they will only compress certain parts of the data for the input datafile. Depending on the dimensionality (e.g. number of variables) of the data, the threads receive a chunk and can end up getting the values from the same dimension (variable) as they process the file. Therefore, this will affect the predictions and CR for the best if the same dimension ends up with same thread, or for worse if the threads get chunks from different dimensions.

Figure 2 presents the CR obtained with pFPC for the five datasets varying the compression level. The pattern that appears with gauss09_alpha repeats itself, much more subtly, with engraph1_100. Both datasets show a decrease in CR which then starts to recover with higher levels. With waterglobe.vel.bin the CR line starts to decline after compression level 17 (not visible on current scale), while with NTUP2 it starts to increase after level 14. These events depend on the file itself and the compression level of FPC/pFPC, as these algorithms are based on predictors. The predicted values change with each compression level, thus giving a chance to expose these behaviours. Summing up, higher compression levels in FPC/pFPC do not always yields higher compression ratios.

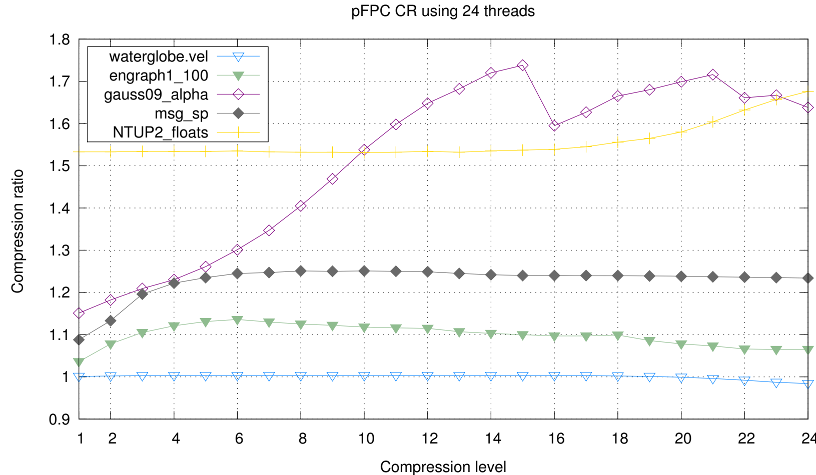

Fig. 2. Compression ratio versus compression level in pFPC

4.3 Parallel Speedup Ratio

Figure 3 presents comparative plots for the speedups of the parallel compressors versus the serial compressors. To compare the speedup ratio from different compression levels, three levels were used, low, medium, and high compression, with

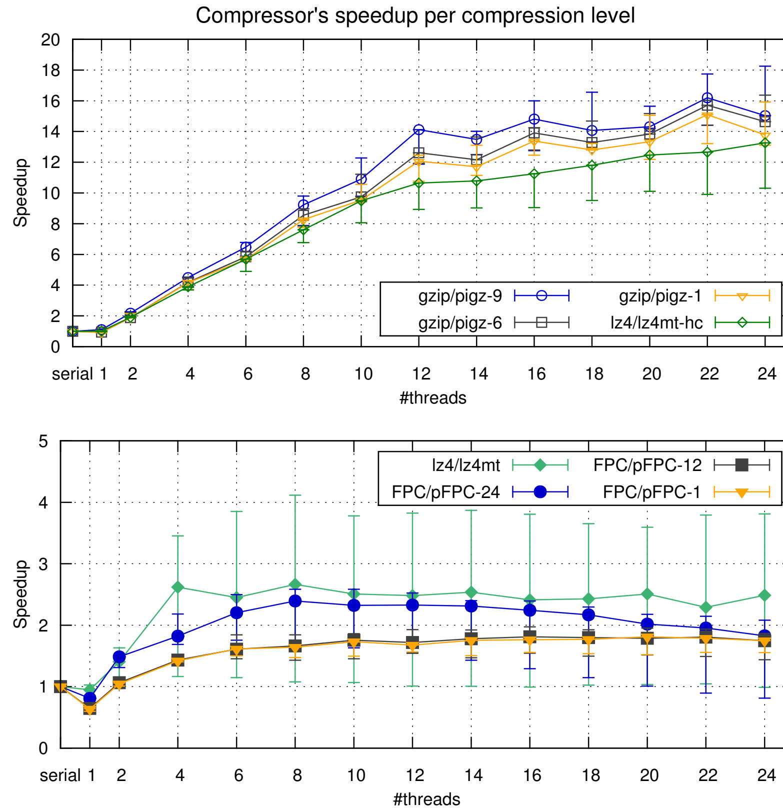

Fig. 3. Speedup versus nthreads used for the NTUP2 datafile. pigz and lz4mthc (*top plot*) with higher speedup, and lz4mt and pFPC (*bottom plot*) with lower speedup.

the exception of lz4/lz4mt that only have low and high compression levels. The NTUP2 dataset is used as the baseline, the remaining four datasets are represented as interval bars (minimum and maximum). The value 'one' corresponding to x=0 represents the baseline for the speedup, and the values on x=1 represent one-threaded version of each compressor/parameter.

Figure 3 shows that speedup varies significantly, either with the datafiles, the compression level used or the compressors themselves. It is split in high and low speedups in order to improve readability. The best performing compressor is pigz when used with maximum compression. After 12 threads, when the maximum number of cores run is reached, the speedup increases more modestly, which demonstrates the benefits of hyper-threading in this kind of workload.

Speedups are larger with more demanding compression levels, as higher compression levels usually mean more computations, thus, longer execution times and a better chance to explore parallelism. This is indeed observable in almost every case for the initial *nthreads*, most notably on pigz with compression level

9, that yields the highest speedups of this study. On the opposite, lz4mt and all pFPC levels show very poor speedups. Using LZ4 in the fast mode is so fast that using multiple threads can actually reduce performance (when the datasets are small the execution times are really low). However, when datasets are bigger and/or the compression level is increased it leads to longer execution times, which in turn yields higher compression speedup (lz4mthc achieves a speedup close to 11 using 12 threads).

Super-linear speedups, or ratios above one, appear mostly with higher compression levels on pigz, and less frequently lz4mt, depending on the datafile. This arises from the fact that the parallel implementation is at times faster than the simpler serial algorithm, which was unexpected. This is particularly noticeable when 12 threads are used.

pFPC shows the worst scaling, presenting the lowest speedup values since the first *nthreads*. The overhead of the multi-threaded version is specially negative for pFPC.

Depicted in Fig.4 is the speedup versus the compression level, using 12 and 24 threads, in order to assess the scalability of the algorithm when the level of compression is increased. The top left plot (pigz 12 threads) shows once again that pigz was faster with one thread than gzip, as the speedups consistently surpass the theoretic limit of 12. The same does not happen with pFPC, that shows a peak followed by a drop with higher compression levels. The behaviour is the same with 12 or 24 threads, with the nuance that the speedups of two files

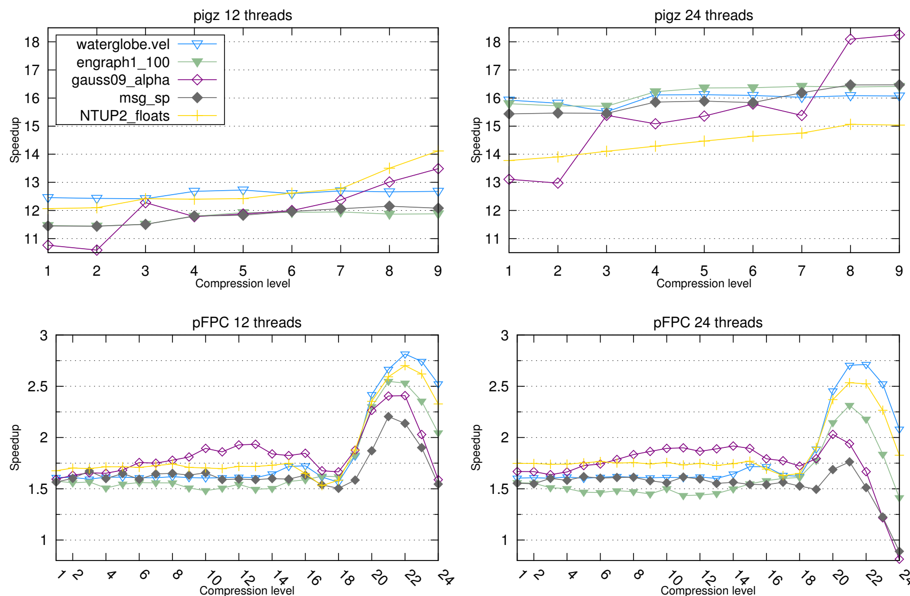

Fig. 4. Speedup versus compression level for pigz (*top plots*) and pFPC (*bottom plots*)

(gauss09_alpha and msg_sp) drop below one with 24 threads when maximum compression settings are used.

4.4 Memory Requirements

The memory required for ZLIB and LZ4 serial compression and decompression is insignificant in a modern system. Nevertheless, the respective parallel compressors, pigz and lz4mt, do use more memory than their serial counterparts. The values observed for pigz were around 10 MB with 12 threads and 18.5 MB with 24 threads, while lz4mt reserves about 100 MB and 196 MB with 12 and 24 threads respectively. For decompression the memory requirements are very similar, and only 100-400 KB lower for both the gzip/pigz and LZ4/lz4mt compressors.

When it comes to FPC memory usage, and in particular to the parallel pFPC, the requirements can be much higher. Serial FPC allocates a table with 2^{n+4} bytes of memory, while pFPC allocates 2^{n+4} bytes for each thread, with n being the compression level selected. This means that the amount of memory used

Table 3. Highest CR and speedup values for each datafile. The third and sixth columns contain, enclosed in square brackets, the algorithm, number of threads and compression level that originated these values.

Datafiles	CR	Largest CR Conditions	MB/s	Sp_t	Largest Sp_t Conditions	MB/s	Ef_t
waterglobe.arc.txt	2.14	[gzip 1 8]	7.3	17.94	[pigz 24 9]	131.6	0.75
waterglobe.1col.arc.txt	2.20	[gzip 1 8]	6.2	18.17	[pigz 24 8]	111.9	0.76
waterglobe.vel.txt	2.19	[gzip 1 8]	6.7	18.10	[pigz 24 8]	122.1	0.75
waterglobe.1col.vel.txt	2.27	[gzip 1 8]	5.3	18.41	[pigz 24 9]	97.1	0.77
waterglobe.arc.bin	1.20	[gzip 1 3]	20.0	16.35	[pigz 24 8]	286.0	0.68
waterglobe.vel.bin	1.48	[gzip 1 5]	19.1	16.12	[pigz 24 5]	307.6	0.67
engraph1_100.txt	2.33	[gzip 1 9]	5.7	18.60	[pigz 24 8]	106.5	0.78
engraph1_100.1col.txt	2.44	[gzip 1 9]	5.0	19.18	[pigz 24 9]	95.5	0.80
engraph1_100.bin	1.22	[gzip 1 3]	20.3	16.42	[pigz 24 7]	276.4	0.68
gauss09_alpha.txt	4.37	[gzip 1 9]	1.9	20.85	[pigz 24 9]	39.4	0.87
gauss09_density.txt	2.36	[gzip 1 9]	4.1	19.28	[pigz 24 9]	78.6	0.80
gauss09_alpha.bin	3.87	[gzip 1 9]	4.6	18.25	[pigz 24 9]	84.7	0.76
gauss09_density.bin	1.09	[FPC 1 24]	82.8	14.83	[pigz 24 9]	294.5	0.62
msg_bt	1.29	[FPC 1 24]	82.4	16.06	[pigz 24 9]	263.5	0.67
msg_lu	1.17	[FPC 1 20]	173.7	15.92	[pigz 24 7]	279.4	0.66
msg_sp	1.26	[FPC 1 24]	116.7	16.47	[pigz 24 9]	217.9	0.69
msg_sppm	7.43	[gzip 1 9]	314.8	15.84	[pigz 24 8]	360.0	0.66
msg_sweep3d	3.09	[FPC 1 24]	166.3	15.40	[pigz 24 7]	272.4	0.64
num_brain	1.16	[FPC 1 24]	96.0	15.68	[pigz 24 5]	263.2	0.65
num_comet	1.16	[gzip 1 9]	88.7	15.76	[pigz 24 9]	236.3	0.66
num_control	1.16	[gzip 1 9]	18.0	15.53	[pigz 24 7]	280.0	0.65
num_plasma	15.00	[FPC 1 24]	127.3	13.05	[pigz 24 5]	322.3	0.54
obs_error	3.54	[FPC 1 24]	91.4	15.94	[pigz 24 8]	193.2	0.66
obs_info	2.27	[FPC 1 24]	65.7	12.30	[pigz 20 7]	232.8	0.62
obs_spitzer	1.23	[gzip 1 3]	18.0	16.45	[pigz 24 9]	203.7	0.69
obs_temp	1.04	[gzip 1 4]	18.3	13.56	[pigz 24 6]	247.7	0.56
NTUP1_floats.bin	2.19	[pigz 1to24 9]	107.8	16.19	[pigz 22 9]	116.2	0.74
NTUP2_floats.bin	2.19	[pigz 1to24 9]	107.9	16.20	[pigz 22 9]	116.3	0.74
NTUP3_floats.bin	2.19	[pigz 1to24 9]	107.6	14.47	[pigz 22 8]	257.2	0.66
NTUP4_floats.bin	2.19	[pigz 1to24 9]	107.8	14.44	[pigz 22 8]	257.1	0.66
NTUP5_floats.bin	2.19	[pigz 1to24 9]	107.7	14.49	[pigz 22 8]	257.6	0.66
NTUP1to5_doubles.bin	4.27	[pigz 1to24 9]	94.9	14.76	[pigz 22 3]	884.7	0.67
NTUP1to5_floats.bin	2.19	[pigz 1to24 9]	114.4	14.84	[pigz 24 8]	263.8	0.62

grows exponentially with the compression level selected. For decompression both FPC and pFPC require about the same memory, because it is needed to refill prediction tables upon the decompression process.

In [15] FPC was tested with n=25, so it took $2^{25+4} = 512M$ bytes of memory, but in order to use the same compression level with 24 threads the tables would occupy 12 GB of memory, which was unavailable in the testing nodes. To stay within the limit n was set to $n = 24$, which allows for the use of all available threads $24 \times 2^{24+4} = 6$GB of memory.

4.5 Compression Ratio and Speedup over the Full Dataset

The highest CR and highest speedup measured are presented on Table 3, with CR summarized on the left three columns and speedup on the rightmost four columns. The speedup values have an extra fourth column that presents the associated parallel compression efficiency of the best attained speedup. As one can verify, the serial algorithms have the best CR, with the exception of NTUPs that are best compressed with pigz, which is also the compressor that achieve highest speedups. The best compression ratios come mostly from higher compression levels, which is expected. pigz has the best speedup for every dataset, with mostly 24 threads, and an overall efficiency above 66%.

5 Conclusions

Selecting an effective compressor for large volumes of scientific data is not an easy task. Firstly, scientific data can be generated and processed with many different usage patterns, which means that the balance between compression and decompression effort, and storage and communication requirements, are not easy to find. Secondly, data can be highly heterogeneous, and the quality of compression depend strongly on the very data. This is relevant, as for large data volumes a poor compressor choice will needlessly increase the CPU time.

We studied high-performance alternatives to ZLIB for high energy physics data and a combination of representative scientific datasets, the general purpose ZLIB/gzip against LZ4 (another general LZ dictionary with increased performance) and the floating-point data compressor FPC. Five datasets with an average random entropy (randomness) of 84.5% were selected, out of 33 datasets, for a more thorough analysis.

In terms of performance, LZ4 is the fastest decompressor, and the faster compressor but only on the lowest compression level. For parallel speedup, pigz yields the best values, but is the slowest compressor when it comes to absolute serial execution times. The most efficient parallel compressor is pigz, but closely followed by lz4mt, presenting efficiency around one when 12 threads are used on a 12 core machine.

As expected, FPC achieved in some cases CR that were unattainable by general purpose compressors [15], arising from the domain-knowledge in the algorithm. We tested if FPC could be a potential alternative to the LZ4 compression

system being implemented in the ROOT toolkit [17]. For 9 datasets FPC had the highest CR of all compressors, while the gzip/pigz pair achieved the highest CR of the remaining 24 datasets. In the five datasets analysed in more detail, FPC shows the highest CR in two cases, but the maximum CR in those cases was 1.29 on data that was difficult to compress by other means. In terms of performance, FPC topped 20% below the performance of LZ4 low compression, which is still much higher than ZLIB. However, parallel scalability was poor, as the multi-threaded implementation seems not to be fully mature yet.

In the context of the high energy physics datasets studied, nevertheless, the behaviour of FPC does not seem competitive. On the ROOT dataset the FPC CR, less than 1.8, was much lower that both LZ4's CR of 2 and ZLIB's CR of 2.1. Also, on practical systems the memory requirements for the pFPC compressor will become critical. In fact, when high compression levels are used together with many threads in pFPC several gigabytes of RAM memory are necessary. On the other hand LZ4 performance is very good, CR degradation is limited and even the parallel implementations use memory sparingly, not becoming an obstacle for other application's activities.

Acknowledgments. This work is funded by National Funds through the FCT - Fundao para a Cincia e a Tecnologia (Portuguese Foundation for Science and Technology) within project PEst-OE/EEI/UI0752/2014, UT Austin — Portugal FCT grant SFRH/BD/47840/2008, and the resources from the project SeARCH funded under contract CONC-REEQ/443/2005. We would also like to thank Nuno Castro and Rafael Silva for their contributions.

References

1. Bell, G., Gray, J., Szalay, A.: Petascale computational systems. Computer 39(1), 110–112 (2006)
2. Hilbert, M., López, P.: The worlds technological capacity to store, communicate, and compute information. Science 332(6025), 60–65 (2011)
3. Staff, S.: Challenges and opportunities. Science 331(6018), 692–693 (2011)
4. Lohr, S.: The age of big data. The New York Times (February 11, 2012)
5. Search for pair production of heavy top-like quarks decaying to a high-p_T W boson and a b quark in the lepton plus jets final state at $\sqrt{s}$=7 TeV with the ATLAS detector
6. Oliveira, V., Pina, A., N.C.F.V.A.O.: Even bigger data: Preparing for the LHC/atlas upgrade. Ibergrid 2012 submission (November 2012)
7. Schendel, E., Jin, Y., Shah, N., Chen, J., Chang, C., Ku, S.H., Ethier, S., Klasky, S., Latham, R., Ross, R., Samatova, N.: ISObar preconditioner for effective and high-throughput lossless data compression. In: 2012 IEEE 28th International Conference on Data Engineering (ICDE), pp. 138–149 (April 2012)
8. Schendel, E.R., Pendse, S.V., Jenkins, J., Boyuka II, D.A., Gong, Z., Lakshminarasimhan, S., Liu, Q., Kolla, H., Chen, J., Klasky, S., Ross, R., Samatova, N.F.: ISObar hybrid compression-I/O interleaving for large-scale parallel I/O optimization. In: Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing, HPDC 2012, pp. 61–72. ACM, New York (2012)

9. Lakshminarasimhan, S., Shah, N., Ethier, S., Ku, S.H., Chang, C.S., Klasky, S., Latham, R., Ross, R., Samatova, N.F.: Isabela for effective in situ compression of scientific data. Concurrency and Computation: Practice and Experience 25(4), 524–540 (2013)
10. Brun, R., Rademakers, F.: Root - an object oriented data analysis framework. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 389(1-2), 81–86 (1997), New Computing Techniques in Physics Research V
11. Nicolaucig, A., Mattavelli, M., Carrato, S.: Compression of tpc data in the alice experiment. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 487(3), 542–556 (2002)
12. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory 23(3), 337–343 (1977)
13. Collet, Y.: Development blog on compression algorithms, `http://fastcompression.blogspot.in/2011/05/lz4-explained.html`
14. Burtscher, M., Ratanaworabhan, P.: High throughput compression of double-precision floating-point data. In: Data Compression Conference, DCC 2007, pp. 293–302 (March 2007)
15. Burtscher, M., Ratanaworabhan, P.: FPC: A high-speed compressor for double-precision floating-point data. IEEE Transactions on Computers 58(1), 18–31 (2009)
16. Welton, B., Kimpe, D., Cope, J., Patrick, C., Iskra, K., Ross, R.: Improving I/O forwarding throughput with data compression. In: 2011 IEEE International Conference on Cluster Computing (CLUSTER), pp. 438–445 (September 2011)
17. Peters, A.J.: Lz4hc compression for root and io baseline evaluation. In: ROOT IO Workshop (December 2013)