Energy Efficient Matrix Computations through Homomorphic Compression

Matthieu Martel
LAMPS Laboratory
Université de Perpignan, France
ORCID: 0000-0002-6238-9651

Célia Picard
Fédération ENAC ISAE-SUPAERO ONERA
Université de Toulouse, France
ORCID: 0000-0002-8715-4365

Abdelmouhaimen Sarhane
Fédération ENAC ISAE-SUPAERO ONERA
Université de Toulouse, France

***Abstract*—Scientific computing is one of the most energy-intensive areas of computer science. Petabytes of data are processed and stored for a single experiment while supercomputers consume tens of megawatts. These last years, new lossy compression techniques dedicated to scientific data have been introduced, such as zfp and sz. They allow one to drastically reduce the size of the data but they require additional computations for compression and decompression. Recently, a new homomorphic compressor, named `blaz`, has been developed which makes it possible to perform matrix operations directly among the compression data. Important gains are obtained since decompression and re-compression operations are avoided and since less operations are needed to compute among the compressed data. In this article, we show that using `blaz`for linear algebra operations among matrices may reduce the energy consumption by a factor 10 compared to standard operations (without any compression) and by a factor 100 compared to the zfp. Beside measures on sequences of matrix operations, a case study in data analysis in presented.**

***Index Terms*—green computing, frugal computing, numerical computations, compression, scientific data, linear algebra**

I. Introduction

It is well known that today, over 10% of the world's electricity production is consumed by information technology (IT) systems [9], [16]. These include personal computers, servers storing data in the cloud and the supercomputers used for climate simulations or for training artificial intelligence. These systems consume a lot of energy. According to ADEME[1] [8], it is estimated that 13% of the world's electricity will be consumed by data centers in 2030, and 51% for the entire IT sector, i.e. the equivalent of 1130 and 4400 nuclear reactors respectively.

In this article, we aim to reduce the resources required in the specific case of large-scale numerical simulations in fields as varied as climatology and astrophysics. This focus is motivated by the fact that these numerical simulations use a significant proportion of global computing resources. As a result, scientific data represent a large part of the data stored worldwide. For example, the Square Kilometer Telescope in Baltimore produces 50 terabytes of data per day, and the ITER thermonuclear reactor in Cadarache, France, plans to produce 2 petabytes of data per day by 2035. In addition to storage, processing these data generates substantial costs. The power consumption of the supercomputers used for numerical simulations is colossal: for example, the two most powerful supercomputers in the world in 2023, the Frontier (Oak-Rigde, USA) and the Fugaku (Riken Center, Japan) have respective outputs of 21.1 and 29.9 megawatts, equivalent to the energy consumption of 26,000 and 37,000 households respectively [2].

It is also important to point out that the overall energy consumption of IT systems is shared into three almost equal parts: storage (30%), processing (30%) and telecommunications (40%) [3].

As a consequence, in order to develop frugal solutions that reduce the energy consumption of IT systems as a whole and avoid rebound effect, it is essential to address these three aspects in their entirety. Indeed, techniques that make it possible to reduce the resources consumed in one of these areas must not significantly increase consumption in the other areas (for example, it is not desirable to reduce storage at the expense of computing by recalculating non-stored data on demand).

Last but not least, reducing the size of the calculations required for large-scale numerical simulations means that supercomputers can be used for longer periods of time. *"Every ten years, the computing power [of supercomputers] is multiplied by* 1000*. As for the lifespan of a supercomputer, it is only four to five years"* following the head of the CEA Bruyères-le-Châtel computing center[4].

In recent work, we have developed a new lossy compressor for scientific data, named `blaz`, enabling calculations to be performed directly on compressed data, without decompression (which is impossible with other existing compressors). Such a compressor is called homomorphic [1], [23]. This imposes a compromise between loss of information on the one hand, and gains in memory and computation on the other. The proposed method is not as efficient as the best compressors for scientific data such as `sz` [6] or `zfp` [20] in terms of compression ratio and accuracy, but it enables calculations to be performed directly on the compressed data,

[1]The french ecological transition agency, `https://www.ademe.fr/`

[2]`https://www.top500.org/`

[3]`https://lejournal.cnrs.fr/articles/numerique-le-grand-gachis-energetique`

[4]`https://www.latribune.fr/technos-medias/informatique/pourquoi-la-france-doit-rester-dans-la-course-des-supercalculateurs-461372.html`

which considerably reduces computations, since i) there is no need to decompress the data and and re-compressing the results of (matrix) operations and ii) since the compressed data is smaller than the original, fewer operations are necessary to calculate with them. This precision is sufficient for many applications, especially when the results are coarsely rounded for graphical display on a screen [4], [10].

The objective of this article is to use the `PowerJoular` tool [24], [25] for energy monitoring, in order to compare the energy consumption of our tool `blaz` to another well-know compressor, `zfp` [20] and to the case of no compression at all. The interest of measuring energy consumption is twofold. First, energy consumption optimization is a goal in its own, for ecological reasons. Second, this measure encompasses other measures such as memory and CPU usages. It avoids rebound effects and provides a global overview of the resources consumed by some application.

The comparison is done for elementary matrix computations. By assessing the energy consumption of these methods, we demonstrate that the energy efficiency gains achieved by `blaz` are higher than a factor 10 compared to the same computations without any compression and higher to a factor 100 compared to the same computations using `zfp` to store the data. In addition, we apply the compressors to a practical use-case in data-science, more precisely sports field scenario in the case of soccer match analysis. Analyzing player movement heat-maps is a crucial tool used by coaches, analysts, and sports scientists to understand player positions, patterns, and tactical behavior during matches. We compute the average player movement heat-map using the `blaz` and `zfp` compressors and evaluate their energy consumption.

The rest of this article is organized as follows. Background material concerning scientific data and the `zfp` and `blaz` compressors is introduced in Section II. Experimental results are presented in Section III. More precisely, we start by introducing in Section III-A `PowerJoular`, the tool used for measuring the energy consumption. Next, our experimental protocol is described in Section III-B and the results are given in Section III-C. The case study is presented in Section III-D. Finally, Section IV concludes.

II. Background: Compression of Scientific Data

In this section we introduce some background material concerning the compression of scientific data. Generalities on scientific data are introduced in Section II-A. Sections II-B and II-C introduce the `zfp` and `blaz` compressors, respectively. Finally, Section II-D briefly compares both compressors and related work is discussed in Section II-E.

A. Scientific Data

Scientific data are commonly referred to as single or multi-dimensional arrays of floating-point numbers [2] (vectors, matrices, tensors, etc.) containing values derived from measurements or resulting from calculations. A particularity of this category of data is that there are many correlations between adjacent elements. For example, the temperature in one area of space evolves smoothly as a function of position, with few or no abrupt changes. These correlations can be used to compress data efficiently, just as they are to compress sound or images (MP3 or JPEG formats for example [11], [13]) with acceptable losses. Lossy compression is much more effective than lossless compression when it comes to reducing data size. It should be noted, however, that compression techniques developed for integers are not suitable for floating-point numbers. It should also be noted that without loss, the compression rate is far too low. For example, the ZIP compression scheme compresses scientific data by an average factor of 2, which is insufficient. A factor greater than 10 is reached by `blaz` and `zfp` can reach factors close to 64.

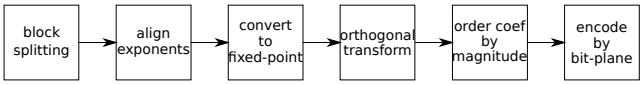

Fig. 1. Overview of `zfp` compression scheme.

Several lossy compressors have been developed in recent years for scientific data, including `zfp` [20] and `sz` [6], [19], [30]. While these compressors are remarkable in terms of data size reduction, they have the major disadvantage that data must be decompressed before computation (e.g., matrix operation) and the result must in turn be compressed. In this sense, they do not fulfill the objective stated in Section I to reduce storage, processing and telecommunications as a whole.

The first approach, chosen by `zfp`(and formerly `fpzip` [22]), consists in setting a compression rate and then obtaining an error on the data compression/decompression depending on the chosen compression ratio. The second approach, chosen by `sz`, is symmetrical and consists in setting a certain error threshold on the compression/decompression process. The compression rate results from this setting. Note that recent versions of `zfp` and `sz` enable the user to tune very finely the compression, e.g. by controlling the error in `zfp` [7].

Our compressor, `blaz` is fixed-rate, like `zfp`and this motivates our choice to compare both tools since it is then possible to compare the compression times and accuracy for a same ratio. In addition, fixed-rate compressors allow random access to the elements of a compressed array which is note the case with variable rate compressors such as `sz`. Random accesses are needed for homomorphic computations.

B. The `zfp` Compressor

In this section, we introduce `zfp`, a fixed-rate compressor for floating-point data that aims at providing lossy compression with random access to the compressed data. `zfp` maps small blocks of 4^d values in d dimensions to a fixed, user-specified number of bits per block. This enables read and write random access to compressed floating-point data at the granularity of blocks. The compression process in `zfp` involves i) aligning the values in a block to a common exponent and ii) converting them to a fixed-point representation. Then iii) an orthogonal block transform is then applied to decorrelate the values, and iv) the resulting coefficients are ordered by

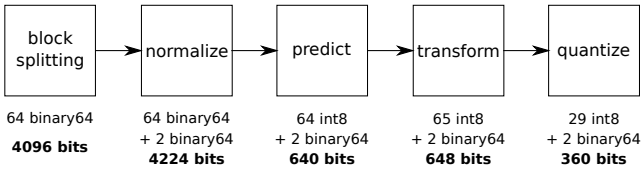

Fig. 2. Overview of `blaz` compression scheme.

expected magnitude. Finally, v), the coefficients are encoded one "bit plane" at a time [3]. The bit stream may be cut at any point, enabling a flexible compression rate at bit-level. An overview of the compression scheme is given in Figure 1.

`zfp` uses a software write-back cache of uncompressed blocks to enhance random access capabilities. This cache reduces the frequency of compression and decompression operations, and by consequence reduces the need for compression or decompression. However, for the purpose of comparing compressors, we will disable this feature.

The performances and applicability of `zfp` have been demonstrated in various domains, such as visualization, quantitative data analysis and numerical simulation [21].

C. The `blaz` Compressor

We introduce below `blaz`, a recently proposed lossy compressor for scientific data [23] enabling calculations to be performed directly on compressed data, without decompression (which is impossible with other compressors such as `zfp`). Such a compressor is called homomorphic [12]. This imposes a compromise between loss of information on the one hand, and gains in memory and computation on the other. `blaz` is fixed-rate, as random access to the compressed data is mandatory to compute among the compressed data. `blaz` is also lossy and operates on IEEE754 `binary32` or `binary64` floating-point number 8×8 blocks. `blaz` uses steps of normalization, prediction, transformation and finally quantization as summarized in Figure 2.

Normalization is used to reduce data variations within the block. As in other compressors [8], [9], normalization is based on the assumption that the elements are correlated. Normalization then consists of replacing the elements of the block with the differences between adjacent values. Next, block element prediction allows floating-point values to be replaced by integers (coded on 8 bits). The range of values in the block is divided by 255 (we call this quantity a step) and prediction consists of storing how many steps are needed to best predict the exact element in the block. As in many other compressors (e.g. JPEG- 2000 [7]), we use a discrete cosine transform (DCT, [22]) of the block resulting from the prediction step. DCTs are used to aggregate large coefficients in the first rows and columns of a matrix and columns of a matrix, with small values appearing in the other elements after transformation. After the DCT, the large coefficients of the block are in the first lines and columns, which contain the bulk of the information. Quantification then consists in considering the small coefficients present in the other rows and columns are negligible and set them to 0 to avoid storing them in the compressed matrix. In practice, `blaz` keeps the first two lines and columns of the block.

`blaz` has been designed in such a way that it is possible to define algorithms for various matrix operations on compressed arrays (addition, multiplication by a constant, scalar product, etc.). These operations are designed for elementary blocks and, readily, for larger larger matrices or 3D arrays, they are repeated for each elementary block. Compressed blocks are stored in a suitable data structure containing, the first exact value of the block, the average slope of the prediction and the remaining coefficients after block transformation and quantization.

For example, for the addition $B = B_1 + B_2$ of two compressed blocks, the first element f of B is the addition of the first elements f_1 and f_2 of the two compressed blocks B_1 and B_2. Similarly, the average slope of B is defined by $s = s1 + s_2$ where s_1 and s_2 are the average slopes of B_1 and B_2. Let $\mathbf{D}_1$ and $\mathbf{D}_2$ denote the blocks obtained after the DCT by passing $\mathbf{B}_1$ and $\mathbf{B}_2$. Intuitively, if no re-scaling were done in our scheme, we could simply add $\mathbf{D}_1$ and $\mathbf{D}_2$ to obtain the block $\mathbf{D}$ corresponding to $\mathbf{B}$. But two re-scalings are done in our scheme, during the normalization stage and among the values resulting from the DCT. It is then possible to re-scale and add the coefficients of the DCT contained in D_1 and D_2. Note that adding the coefficients is possible since the DCT defines a linear map among our blocks. Formally, for a 8×8 block $\mathbf{B}$, the two-dimensional DCT of $\mathbf{B}$, denoted $\mathrm{DCT}(\mathbf{B}) = \mathbf{D}$ is defined, for $0 \leq i, j < 8$, by

$$\mathbf{D}_{ij} = \alpha_i \alpha_j \sum_{u=0}^{7} \sum_{v=0}^{7} \mathbf{B}_{ij} \cos\left[\frac{(2u+1)i\pi}{16}\right] \cos\left[\frac{(2v+1)j\pi}{16}\right]$$

and it follows that for two blocks $\mathbf{B} = \mathbf{B}_1 + \mathbf{B}_2$,

Other operations are handled in a similar way (subtraction, multiplication by a constant, Hadamard product). Particular attention must be paid to the scalar product which currently necessitates to compute the inverse of the DCT (we do believe that this can be improved in the future).

As shown in Figure 2, a compressed block is stored using 29 8-bits integers and 2 `binary64` floating-point numbers. The 29 8-bits integers correspond to the 28 values that we keep after quantization plus one 8-bits integer corresponding to the re-scaling factor φ. A block is then stored into 360 bits instead of the original 4096 bits needed to store 64 `binary64` numbers, yielding a compression rate of 11.37.

D. Brief Comparison of `blaz` and `zfp`

In this section, we briefly compare `blaz` and `zfp` in terms of speed and accuracy. Let us mention that more comprehensive results concerning the speed and accuracy of `blaz` and `zfp` have been introduced in [1], [23].

Table I displays execution times and relative errors for operations among two 1040×1040 matrices A and B corresponding to the discretization of the functions $f(x, y) = x^2 - y$ and $g(x, y) = x^2 \times y^2$ between -2 and 2. The relative errors

represent the mean of the relative errors on the elements of the matrices.

We can observe that blaz is approximately 100 times faster than zfp for addition and multiplication by a constant and approximately 10 times faster for matrix multiplication. Conversely, zfp is more accurate by a factor up to 100, which represents two decimal digits on the result. The additional error introduced by blaz to save CPU and, as we will see in the next section, energy, remains interesting for many application domains.

E. Related Work

In this section, we discuss related work about compression techniques for scientific data. A first related topic is about the use of GPUs to accelerate the compression and decompression of scientific data [15], [18], [27]–[29].

Recently, special attention has been paid to the compression of the parameters of deep neural networks [14], [26]. However, this case is slightly different since the parameters of neural networks generally do not exhibit the same correlations than adjacent values in numerical simulations. In this domain, a related technique, called pruning, consists of removing the less significant weights in order to lighten the network [5], [17].

Finally, many work has focused on applications and benchmarks in various fields of science [3], [4], [19], [31].

III. Power Consumption Evaluation

In this section, we introduce our comparison concerning the energy consumed by blaz with respect to other solutions for matrix computations. Hereafter, energy measurements are carried out using the PowerJoular tool introduced in Section III-A. The methodology is described in Section III-B and the results are presented in Section III-C. A case study concerning data analysis is reported in Section III-D.

A. Energy Consumption Measurement with PowerJoular

In this section, we briefly introduce PowerJoular [24], [25], a power monitoring tool designed to help software developers, system administrators, etc. to understand and analyze the power consumption of their programs and devices. Let us mention that PowerJoular is written in Ada, a language well-known for its energy efficiency, which ensures that PowerJoular has a low fingerprint and contributes to energy efficiency efforts [5]. PowerJoular is designed to automatically detect the hardware configuration and supported modules of a system, allowing it to provide accurate power data. It utilizes the Intel RAPL power data through the Linux Powercap interface to monitor the CPU power consumption. By aggregating power readings from various components such as CPU cores, integrated graphics, memory controller, and last level caches, PowerJoular can compute the power consumption of the system. The tool can also monitor GPU power consumption for NVIDIA's devices. Finally, PowerJoular can also monitor the power consumption of individual processes. This feature enables one to track power data for specific applications or software components.

Matrix	Accu. blaz	Accu. zfp	Time blaz	Time zfp
A	0.53%	0.02%	2.0E^{-5}	–
B	0.44%	0.002%	1.8E^{-5}	–
$A+B$	2.27%	0.27%	2.2E^{-3}	2.8E^{-1}
$k \cdot A$	0.53%	0.03%	2.0E^{-3}	2.1E^{-1}
$A \times B$	0.0009%	0.0002%	1.1E^{0}	1.2E^{1}

TABLE I
Comparison of the performances of blaz and zfp for 1040×1040 matrices. The accuracy is given in relative error. Time is given in seconds.

B. Energy Measurements

We introduce hereafter our experimental protocol. We have developed several C codes to implement various elementary matrix computations, including dot products, additions of matrices, multiplications by a matrix and multiplications of a matrix by a constant. These computations are performed on matrices compressed with blaz and zfp as well as without any compression, for the sake of comparison. In addition to unitary matrix operations, we have also compared the energy consumption when combining sequences of operations before re-compressing the data. This reflects real-world scenarii where matrices undergo sequential computations. We have focus on combining multiple additions or combining addition and multiplication by a matrix, as these operations are frequently encountered in matrix computations and can significantly affect energy consumption.

The specific methods employed to measure energy consumption as well as a detailed explanation of what is being measured are outlined in Section III-C. blaz compress data with a fixed rate of 11.37, so for comparison, we have adjusted zfp to match the same rate. We have also considered matrices whose size ranges from 16×16 to 8192×8192 to capture diverse computational scenarii.

The experiments have been repeated multiple times to account for variations and ensure reliable results. The PowerJoular tool has been used to measure the energy consumption of each computation scenario. This tool provides energy consumption measurements, enabling evaluation of the energy consumed during the execution of matrix computations. We have performed our experiments on a PC equipped with an Intel Xeon CPU E5-2603 v3 and 16GB of RAM. The PC was running Ubuntu 22.04.2 LTS desktop with kernel version 5.19.0-46 and gcc version 11.3. In order to focus solely on measuring the energy consumption of the CPU, we disabled the GPU to prevent parallel execution during the experiment. By doing so, we ensured that the process in running in the CPU only and the energy measurements were specifically attributed to the CPU's power usage.

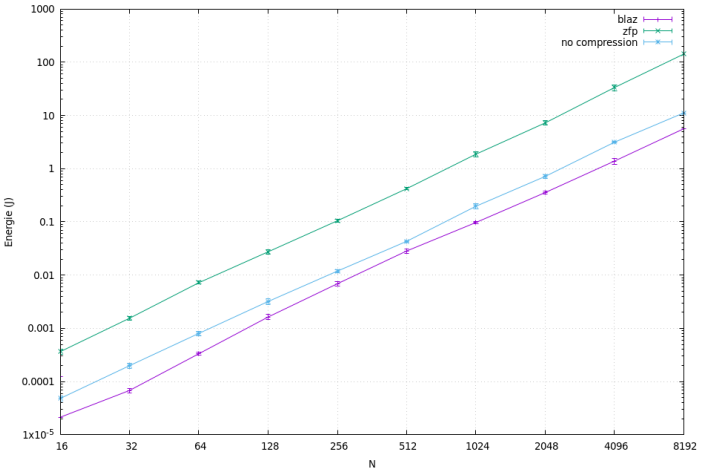

Fig 3.a. Addition

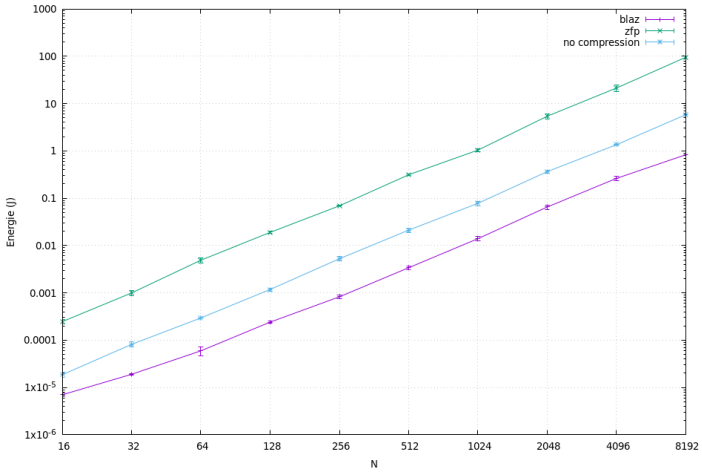

Fig 3.b. Multiplication by a constant

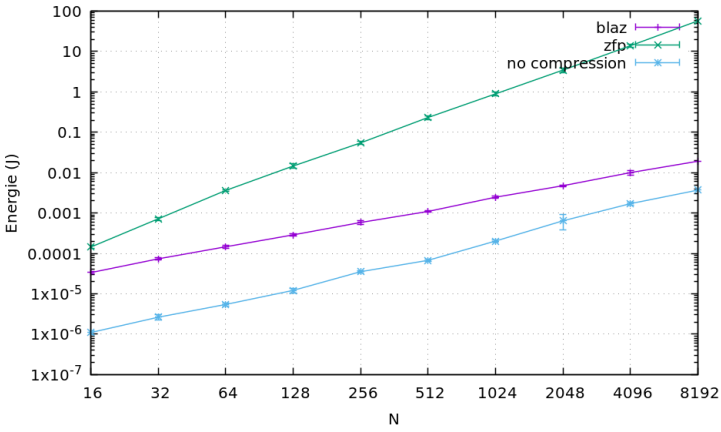

Fig 3.c. Dot product

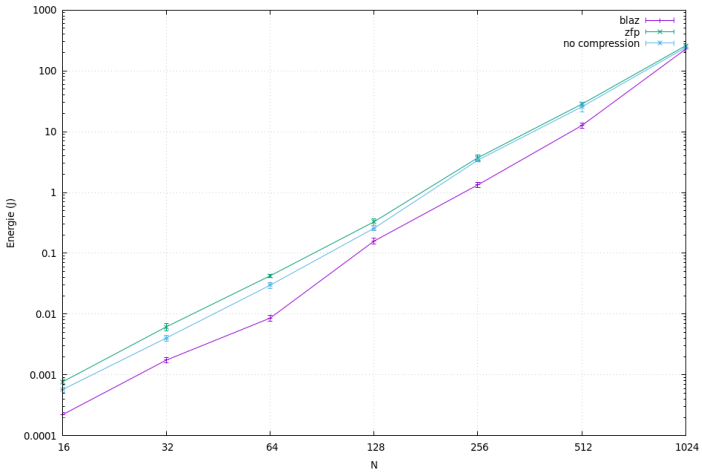

Fig 3.d. Matrix multiplication

Fig. 3. Energy consumption of operations in function of the size N (x-axis) of the matrices (energy given in $10-$logarithmic scale and N in $2-$logarithmic scale).

C. Experimental Results

The obtained energy consumption measurements of each operation for a $N \times N$ matrix of double numbers are shown in Figure 3 (N ranging from 16 to 8192). The results of the experiments demonstrate that `blaz` obtains energy savings in matrix computations compared to both `zfp` and the program which does not use compression, with the exception of the dot product operation. In addition and multiplication by a constant, `blaz` shows a more significant reduction in energy consumption. However, in the case of matrix multiplication, especially for large matrices, the advantage of `blaz` compression vanishes slightly, particularly for very large matrices. This is due to the fact that the current implementation of `blaz` dot product is not optimized and performs a partial decompression of the data (the inverse DCT).

Nevertheless, as shown in Figure 3, `blaz` does not outperform the uncompressed program in terms of energy consumption for the dot product. This can be attributed to the additional operations involved in the dot product algorithm when using `blaz` compression.

Figure 4 presents the results of the energy measurements for a sequence of operations. The experiment has been conducted using matrix sizes of 512×512 for the additions-only sequence and a matrix size of 128×128 for the combined addition and multiplication sequence.

This new experiment shows that `blaz` consistently demonstrates the lowest energy consumption across the various operations, indicating its superior energy efficiency compared to `zfp` and the uncompressed approach. This suggests that `blaz` ability to directly compute on compressed data without the need for decompression and re-compression significantly reduces the energy requirements for matrix computations.

On the other hand, `zfp` shows higher energy consumption compared to `blaz`, due to the additional computations involved in compressing and decompressing the data. The compression and decompression processes in `zfp` introduce overhead, resulting in increased energy consumption. These results support the potential of `blaz` as an energy-efficient

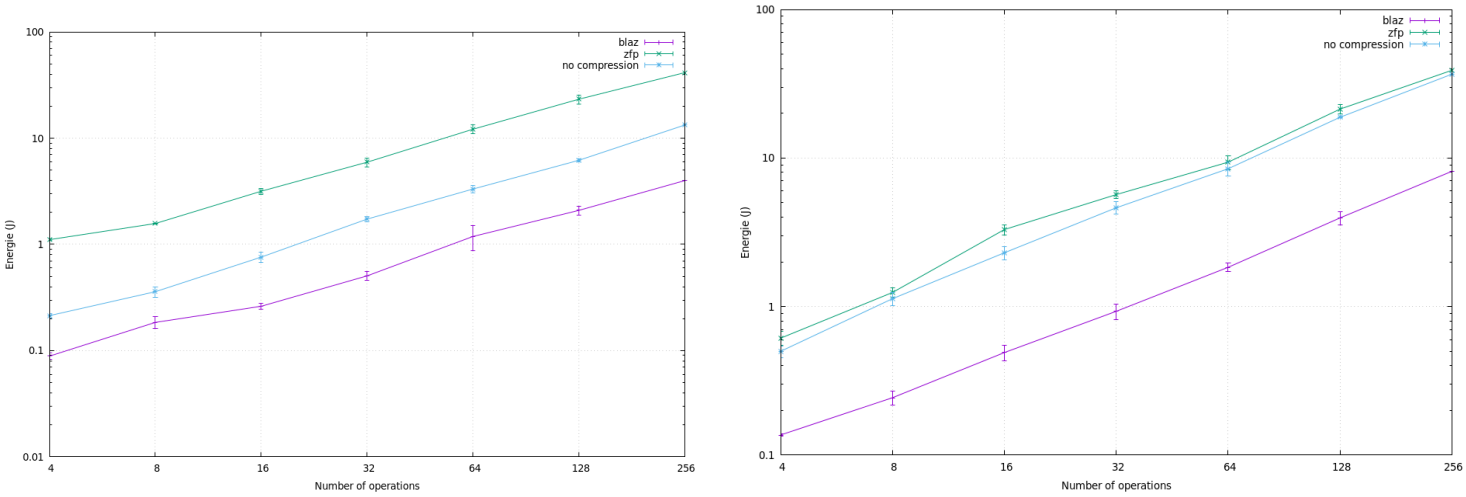

Fig. 4. Energy consumption of a sequence of operations (energy given in $10-$logarithmic scale and number of operations in $2-$logarithmic scale). Left: Addition operations only. Right: Sequence of additions followed by multiplications.

compression technique for matrix computations.

D. Use Case in Data Analysis

In the context of sports analysis, especially in soccer, gaining insights into player movements and positioning is crucial for enhancing team performance, devising effective strategies, and making data-driven decisions. Analyzing player movement heat-maps is a valuable tool used by coaches, analysts, and sports scientists to understand player positions, patterns, and tactical behavior during matches.

A typical player movement heat-map consists of a $2D$ array of floating-point numbers, where each value represents the player's presence or activity level in a specific region of the soccer pitch. For instance, we can create a heat-map where each value corresponds to the average number of times a player has been present within a 10 cm² region of the pitch across the last 10 matches. Considering a standard soccer pitch size of 105×68 meters, the resulting heat-map array would approximately have dimensions of 1048×680.

To evaluate the energy efficiency of the `blaz` and `zfp` compressors in computing the average player movement heat-map, we will generate random heat-maps simulating player positions for comparison. These random heat-maps will serve as a benchmark to assess the energy consumption of `blaz` and `zfp` when processing player movement data from the last 10 games. The energy consumption measurements obtained from computing the average of ten floating-point arrays of size 1050×680 are shown in Table II.

Since `blaz` enables direct computations on the compressed data, it reduces the computational overhead and storage requirements, thereby leading to potential energy savings.

Assume that there are approximately $100,000$ soccer matches played worldwide every year, each involving at least 10 players whose movement needs to be analyzed using player movement heat-maps. For simplicity, let us assume that the average energy consumption difference between `blaz` and `zfp` is 5 joules, as shown in Table II. This means that by using `blaz` in soccer match analysis for all matches worldwide, the soccer community can potentially save approximately 5 megajoules of energy annually. To put this into perspective, 5 MJ is about 0.16% of the average US household energy consumption in a month. These numbers demonstrate the impact that adopting energy-efficient techniques like `blaz` can have, not only in soccer match analysis but in various more important data-intensive fields.

	`blaz`	`zfp`	No Compression
Energy (mJ)	528 ± 53	5700 ± 530	2360 ± 162
Time (ms)	37 ± 0.08	420 ± 4	175 ± 1

TABLE II
ENERGY CONSUMPTION MEASUREMENTS
TO COMPUTE THE AVERAGE OF TEN HEAT-MAPS.

IV. Conclusion and Perspectives

In this article, we have evaluated the energy consumption of matrix computations using compressed data, comparing `blaz`, `zfp` and no compression scenarii. The results consistently showed that `blaz` is significantly more energy efficient, reducing energy consumption for addition, multiplication by a constant, dot product and matrix multiplication compared to `zfp` and the programs without any compression. This demonstrates the potential of `blaz` for sustainable scientific computing. these gains come at the price of a more important relative error introduced by `blaz` with respect to `zfp`. However this error (two decimals in general) may remain acceptable in many situations (especially when the computed data are sent to visualization tools.) An improvement of this work would be to to evaluate the energy consumption of operations in RAM modules.

The application of `blaz` and `zfp` to the data analysis of soccer games also highlights the energy-saving advantage of `blaz`. By computing average player movement heat-maps, `blaz` outperforms `zfp` in terms of energy efficiency when processing data from the last ten games.

These findings are significant for promoting green computing and reducing the environmental impact of scientific computing. To generalize these results to other, larger, applications, `blaz` needs to be extended to more homomorphic operations. Indeed, in future work, we aim at improving `blaz` in several ways. Firstly, the compressor is currently limited to two-dimensional arrays. We plan to treat at least three-dimensional arrays or general tensors. Secondly, we aim at adding new matrix operations to `blaz` such as stencil operations (e.g. `B[i,j]=A[i-1,j]+B[i,j-1]`), reductions (e.g. summing the elements of line or column), etc. Matrix multiplication also requires partial data decompression currently and we want to improve this point. Finally, `blaz` compression ratio is fix (11.37). We plan to improve it (we are aiming for a rate of at least 20) and make it parameterizable.

In summary, this study emphasizes the importance of energy-efficient approaches for scientific computing. We strongly believe that the approach proposed by `blaz` can be improved and generalized to contribute to the development of more sustainable numerical computations which represent an important part of the whole energy consumption of the IT sector.

REFERENCES

[1] Tripti Agarwal, Harvey Dam, Ponnuswamy Sadayappan, Ganesh Gopalakrishnan, Dorra Ben Khalifa, and Matthieu Martel. What operations can be performed directly on compressed arrays, and with what error? In *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, SC-W*, pages 252–262. ACM, 2023.

[2] ANSI/IEEE. *IEEE Standard for Binary Floating-point Arithmetic*, 2008.

[3] A. H. Baker, D. M. Hammerling, S. A. Mickelson, H. Xu, M. B. Stolpe, P. Naveau, B. Sanderson, I. Ebert-Uphoff, S. Samarasinghe, F. De Simone, F. Carbone, C. N. Gencarelli, J. M. Dennis, J. E. Kay, and P. Lindstrom. Evaluating lossy data compression on climate simulation data within a large ensemble. *Geoscientific Model Development*, 9(12):4381–4403, 2016.

[4] Franck Cappello, Sheng Di, Sihuan Li, Xin Liang, Ali Murat Gok, Dingwen Tao, Chun Hong Yoon, Xin-Chuan Wu, Yuri Alexeev, and Frederic T. Chong. Use cases of lossy compression for floating-point data in scientific data sets. *J. High Perform. Comput. Appl.*, 33(6), 2019.

[5] Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. A survey on deep neural network pruning-taxonomy, comparison, analysis, and recommendations. *CoRR*, abs/2308.06767, 2023.

[6] Sheng Di and Franck Cappello. Fast error-bounded lossy HPC data compression with SZ. In *IEEE Int. Parallel and Distributed Processing Symposium, IPDPS*, pages 730–739. IEEE Computer Society, 2016.

[7] James Diffenderfer, Alyson Fox, Jeffrey A. F. Hittinger, Geoffrey Sanders, and Peter G. Lindstrom. Error analysis of ZFP compression for floating-point data. *SIAM J. Sci. Comput.*, 41(3):A1867–A1898, 2019.

[8] Cécile Diguet and Fanny Lopez. Impact spatial et énergétique des data centers sur les territoires, projet enernum, rapport et synthèse, 2019.

[9] Linda C. Harwell, Deborah N. Vivian, Michelle D. McLaughlin, and Stephen F. Hafner. Scientific data management in the age of big data: An approach supporting a resilience index development effort. *Frontiers in Environmental Science*, 7:72, 2019.

[10] Duong Hoang, Pavol Klacansky, Harsh Bhatia, Peer-Timo Bremer, Peter Lindstrom, and Valerio Pascucci. A study of the trade-off between reducing precision and reducing resolution for data analysis and visualization. *IEEE Trans. Vis. Comput. Graph.*, 25(1):1193–1203, 2019.

[11] Abir Jaafar Hussain, Ali Al-Fayadh, and Naeem Radi. Image compression techniques: A survey in lossless and lossy algorithms. *Neurocomputing*, 300:44–69, 2018.

[12] Michela Iezzi. Practical privacy-preserving data science with homomorphic encryption: An overview. In *2020 IEEE International Conference on Big Data (IEEE BigData 2020)*, pages 3979–3988. IEEE, 2020.

[13] Uthayakumar Jayasankar, Vengattaraman Thirumal, and Dhavachelvan Ponnurangam. A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications. *Journal of King Saud University - Computer and Information Sciences*, 33(2):119–140, 2021.

[14] Sian Jin, Sheng Di, Xin Liang, Jiannan Tian, Dingwen Tao, and Franck Cappello. Deepsz: A novel framework to compress deep neural networks by using error-bounded lossy compression. In *High-Performance Parallel and Distributed Computing, HPDC*, pages 159–170. ACM, 2019.

[15] Sian Jin, Pascal Grosset, Christopher M. Biwer, Jesus Pulido, Jiannan Tian, Dingwen Tao, and James P. Ahrens. Understanding gpu-based lossy compression for extreme-scale cosmological simulations. In *2020 IEEE Int. Parallel and Distributed Processing Symposium (IPDPS)*, pages 105–115. IEEE, 2020.

[16] Nicola Jones. How to stop data centres from gobbling up the world's electricity. *Nature*, 561:163–166, 09 2018.

[17] Vinu Joseph, Ganesh Gopalakrishnan, Saurav Muralidharan, Michael Garland, and Animesh Garg. A programmable approach to neural network compression. *IEEE Micro*, 40(5):17–25, 2020.

[18] Fabian Knorr, Peter Thoman, and Thomas Fahringer. ndzip: A high-throughput parallel lossless compressor for scientific data. In *31st Data Compression Conference, DCC 2021*, pages 103–112. IEEE, 2021.

[19] Xin Liang, Sheng Di, Dingwen Tao, Sihuan Li, Shaomeng Li, Hanqi Guo, Zizhong Chen, and Franck Cappello. Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In *IEEE Int. Conference on Big Data*, pages 438–447. IEEE, 2018.

[20] Peter Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2674–2683, 2014.

[21] Peter Lindstrom, Po Chen, and En-Jui Lee. Reducing disk storage of full-3d seismic waveform tomography (F3DT) through lossy online compression. *Comput. Geosci.*, 93:45–54, 2016.

[22] Peter Lindstrom and Martin Isenburg. Fast and efficient compression of floating-point data. *IEEE Trans. Vis. Comput. Graph.*, 12(5):1245–1250, 2006.

[23] Matthieu Martel. Compressed matrix computations. In *IEEE/ACM International Conference on Big Data Computing, Applications and Technologies, BDCAT 2022, Vancouver, WA, USA, December 6-9, 2022*, pages 68–76. IEEE, 2022.

[24] Adel Noureddine. Powerjoular and joularjx: Multi-platform software power monitoring tools. In *18th International Conference on Intelligent Environments*, pages 1–4. IEEE, 2022.

[25] Adel Noureddine and Olivier Le Goaër. Software energy efficiency: New tools for developers. *ERCIM News*, 2022(131), 2022.

[26] James O'Neill. An overview of neural network compression. *CoRR*, abs/2006.03669, 2020.

[27] Jiannan Tian, Sheng Di, Xiaodong Yu, Cody Rivera, Kai Zhao, Sian Jin, Yunhe Feng, Xin Liang, Dingwen Tao, and Franck Cappello. Optimizing error-bounded lossy compression for scientific data on gpus. In *Cluster Computing*, pages 283–293. IEEE, 2021.

[28] Jiannan Tian, Sheng Di, Kai Zhao, Cody Rivera, Megan Hickman Fulp, Robert Underwood, Sian Jin, Xin Liang, Jon Calhoun, Dingwen Tao, and Franck Cappello. Cusz: An efficient gpu-based error-bounded lossy compression framework for scientific data. In *Proceedings of the ACM Int. Conference on Parallel Architectures and Compilation Techniques*, PACT '20, page 3–15. Association for Computing Machinery, 2020.

[29] Boyuan Zhang, Jiannan Tian, Sheng Di, Xiaodong Yu, Yunhe Feng, Xin Liang, Dingwen Tao, and Franck Cappello. FZ-GPU: A fast and high-ratio lossy compressor for scientific computing applications on gpus. In *International Symposium on High-Performance Parallel and Distributed Computing, HPDC*, pages 129–142. ACM, 2023.

[30] Kai Zhao, Sheng Di, Maxim Dmitriev, Thierry-Laurent D. Tonellot, Zizhong Chen, and Franck Cappello. Optimizing error-bounded lossy compression for scientific data by dynamic spline interpolation. In *Int. Conf. on Data Engineering, ICDE 2021*, pages 1643–1654. IEEE, 2021.

[31] Kai Zhao, Sheng Di, Xin Liang, Sihuan Li, Dingwen Tao, Julie Bessac, Zizhong Chen, and Franck Cappello. SDRBench: Scientific data reduction benchmark for lossy compressors. In *IEEE Int. Conference on Big Data*, pages 2716–2724. IEEE, 2020.