Matrix Computations: In Seek of Frugality

Matthieu Martel

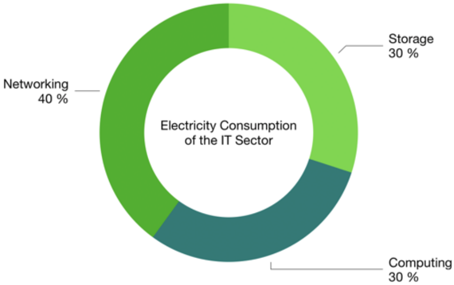
matthieu.martel@univ-perp.fr

GreenDays 2024, Toulouse

Urgent Need of Green Computing!

More than **10%** of the world electric production is used by IT systems

Distribution of power consumption of computer systems in the world:



Frugality mandatory to preserve the Planet:

- For **ecologic** but also for economic reasons
- Savings must address all aspects of IT: **Storage**, **computing** & **networking**

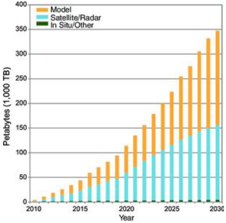
The Special Case of Scientific Data

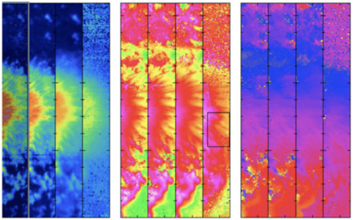
Scientific data:

- Arrays of **correlated** numbers in IEEE754 floating-point formats
- Rather smooth surfaces corresponding to the result of simulations or observations

Orders of magnitude[1]**:**

- Panoramic Survey Telescope (Baltimore): > 1.6 petabytes
- By around 2035, ITER will produce 2 petabytes of data on a daily basis
- Climate data volumes and projections into 2030 for climate models (histogram)





[1] wikipedia

Urgent Need of Compression Techniques for Scientific Data!

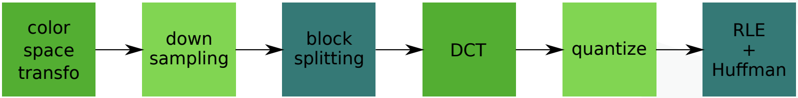
Usual compression techniques not efficient for scientific data

* `zip` (**loseless**) : compression rate $\approx$ 2:1 for scientific data
* Needs for **lossy** compression techniques to go beyong the 2:1 ratio
* Expected ratios: $>$ 10:1

High precision not always needed (e.g. visualization)

Compressors are pipelines of transformations

Example: The JPEG Format[2]



[2] A. Hussain, A. Al-Fayadh, N. Radi, Image compression techniques: A survey in lossless and lossy algorithms, Computer Science Neurocomputing, 300:44–69 (2018)

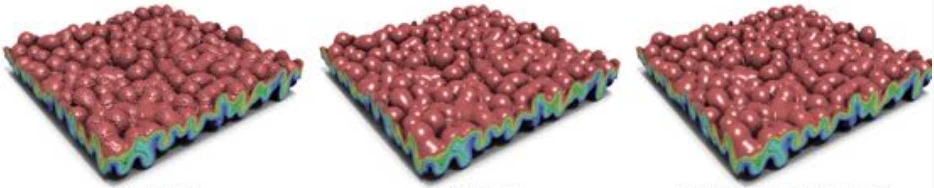
Existing Tools for Lossy Scientific Data Compression

Two symetric approaches for floating-point number compression:

- *error* = *fct*(*compression rate*) `zfp`
- *compression rate* = *fct*(*error*) `sz`

An example with `zfp`:[3]

- Interval volume renderings of compressed double-precision floating-point data on a $384 \times 384 \times 256$ grid
- At **4 bits/double** (16x compression) the image is visually indistinguishable from full 64-bit precision



(a) 1 bit/double (b) 4 bits/double (c) 64 bits/double (no compression)

[3] P. Lindstrom, Fixed-Rate Compressed Floating-Point Arrays, IEEE Trans. Vis. Comput. Graph. 20(12): 2674-2683 (2014)

Homomorphic Matrix Computations

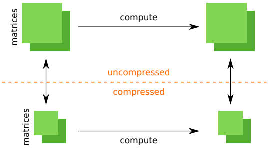
Scientific data compression techniques such as `zfp` and `sz`:

✓ Save storage ✓ Save networking ✗ **Increase computing**

Principle: To compute directly with compressed matrices, without decompression

Similar to **homomorphic encryption**[4]

✓ Saves storage ✓ Saves networking ✓ **Decreases computing**



Advantages:

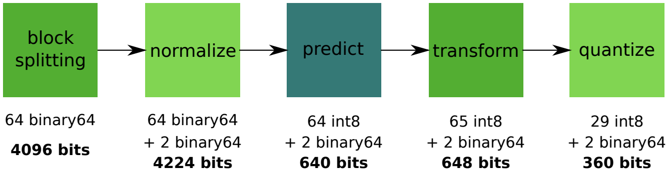
- Avoids to compress/uncompress matrices before using them in computations
- Reduces elementary operations needed to perform the matrix operations

[4] J. Hee Cheon et al, *Protecting Privacy through Homomorphic Encryption* , Springer, 2021

The `blaz` Compressor[5]

- A matrix compressor: lossy, fixed rate (**11.37:1**), block based
- Allows basic linear algebra among compressed matrices

`blaz` **general workflow:**

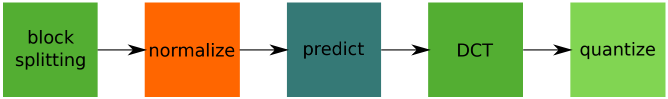


Currently supported operations:

- Without uncompression: addition, subtraction, multiplication by constant
- With partial uncompression: dot product, multiplication

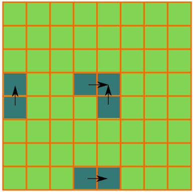
[5] M. Martel, Compressed Matrix Computations, IEEE/ACM Int. Conf. on Big Data Computing, Applications and Technologies, BDCAT, 2022.

Normalization



Normalize:

1) Values replaced by differences between consecutive elements
2) Resulting values divided by mean slope between consecutive values



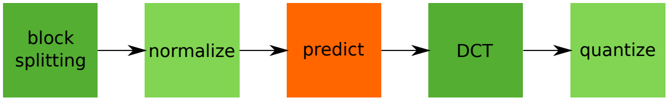
$$\Delta_{\mathbf{0},j+\mathbf{1}} = M_{\mathbf{0},j+\mathbf{1}} - M0, j$$

$$\Delta_{i+\mathbf{1},\mathbf{0}} = M_{i+\mathbf{1},\mathbf{0}} - Mi, 0$$

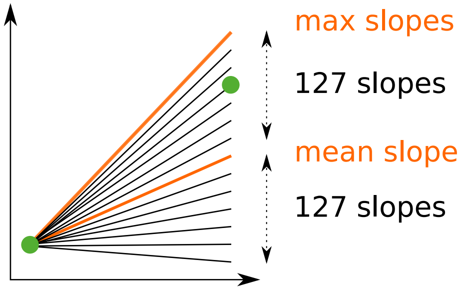
$$\Delta_{i+\mathbf{1},j+\mathbf{1}} = \frac{1}{2}\begin{pmatrix} \left(M_{i+\mathbf{1},j+\mathbf{1}} - Mi, j+1\right) \\ + \\ \left(M_{i+\mathbf{1},j+\mathbf{1}} - Mi+1, j\right) \end{pmatrix}$$

$$0 \leq i, j \leq 7$$

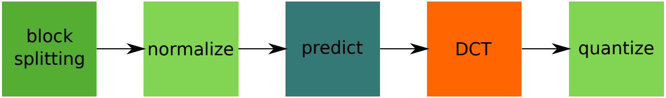
Prediction



Predict: Slope between consecutive values given by a ratio (8 bits signed integer)

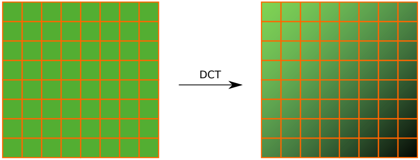


Discrete Cosine Transform



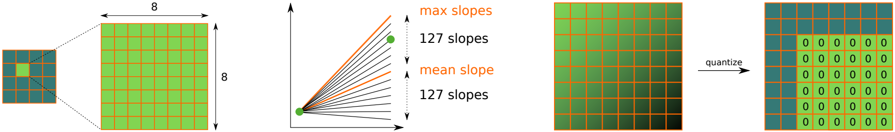
Transform: 2D DCT stores highest coefficients into first lines & columns

$$D_{i,j} = \frac{1}{\sqrt{2N}} C_i C_j \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} M_{x,y} \cos\left[\frac{(2x+1)i\pi}{2N}\right] \cos\left[\frac{(2y+1)j\pi}{2N}\right]$$



$$C_u = \begin{cases} \frac{1}{\sqrt{2}} \text{ if } u = 0 \\ 1 \text{ if } u > 0 \end{cases}$$

`blaz` Matrices: Data Structure



8×8 block B of a compressed matrix encoded by:

- f the **value** of B_{00} (`binary64`)
- s the **mean slope** s of B (`binary64`)
- φ the **scale factor** to normalize the result of the DCT (8-bits integer)
- C array of 28 8-bits values containing the **coefficients quantized** after the DCT

Addition of Two Blocks

$$B = B_1 + B_2$$

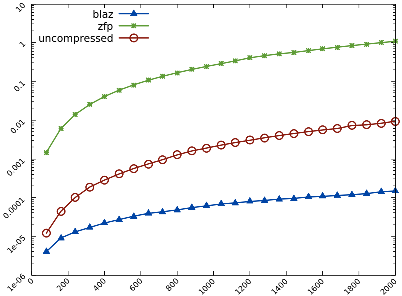
- **First elements:** $f = f_1 + f_2$
- **Mean slope:** $s = s_1 + s_2$
- **Scale factor:** We want $\varphi = \frac{127}{m}$ with $m = m_1 + m_2$ (m max slope). Then
$$\varphi = \frac{127}{m_1 + m_2} = \frac{127}{\frac{127}{\varphi_1} + \frac{127}{\varphi_2}} = \frac{1}{\frac{1}{\varphi_1} + \frac{1}{\varphi_2}} = \frac{\varphi_1 \varphi_2}{\varphi_1 + \varphi_2}$$
- **Discrete Cosine Transform**, for two matrices M_1 and M_2:
$$DCT(M_1 + M_2) = DCT(M_1) + DCT(M_2)$$

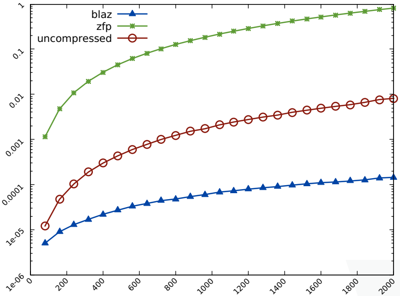
The blaz Library: https://github.com/mmartel66/blaz

```c
typedef struct {
  int width, height;
  double *block_first_elts, *block_mean_slope;
  s_8 *compressed_values;
} Blaz_Compressed_Matrix;


Blaz_Compressed_Matrix *blaz_compress(Blaz_Matrix*);

Blaz_Matrix* blaz_uncompress(Blaz_Compressed_Matrix*);


double blaz_get_matrix_elt(Blaz_Matrix*, int, int);

void blaz_set_matrix_elt(Blaz_Matrix*, double, int, int);


double blaz_get_compressed_matrix_elt(Blaz_Compressed_Matrix*, int, int);

void blaz_set_compressed_matrix_elt(Blaz_Compressed_Matrix*, double, int, int);


Blaz_Matrix *blaz_add(Blaz_Matrix*, Blaz_Matrix*);

Blaz_Compressed_Matrix *blaz_add_compressed(Blaz_Compressed_Matrix*, Blaz_Compressed_Matrix*);


Blaz_Matrix *blaz_sub(Blaz_Matrix*, Blaz_Matrix*);

Blaz_Compressed_Matrix *blaz_sub_compressed(Blaz_Compressed_Matrix*, Blaz_Compressed_Matrix*);


Blaz_Matrix *blaz_mul_cst(Blaz_Matrix*, double);

Blaz_Compressed_Matrix *blaz_mul_cst_compressed(Blaz_Compressed_Matrix*, double);


double blaz_dot_product(Blaz_Matrix*, Blaz_Matrix*, int, int);

double blaz_dot_product_compressed(Blaz_Compressed_Matrix*, Blaz_Compressed_Matrix*, int, int);
```

Research Objects Reviewed

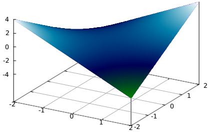
Time Measurements: `Blaz` vs `zfp`



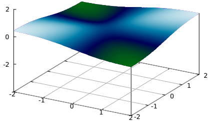


- Time measurement of operations in function of the size of the matrices
- Time given in **logarithmic scale**
- **Left:** Addition. **Right:** Multiplication by a constant

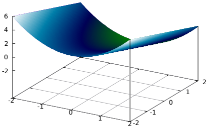
Accuracy Measurements: Test Functions



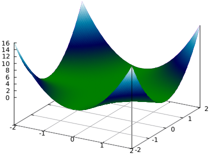
$f_1(x, y) = x \times y$



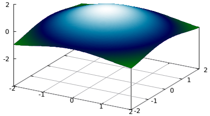
$f_2(x, y) = \frac{xy}{1+x^2+y^2}$



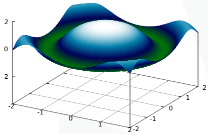
$f_3(x, y) = x^2 - y$



$f_4(x, y) = x^2 \times y^2$



$f_5(x, y) = \cos(\sqrt{x^2 + y^2})$

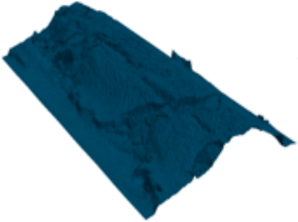


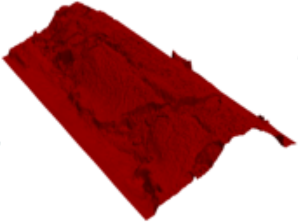
$f_6(x, y) = \cos(x^2 + y^2) \cdot e^{-0.1 \cdot (x^2+y^2)}$

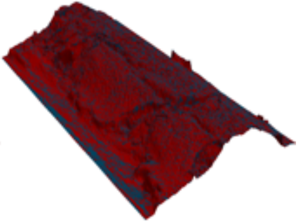
Accuracy Measurements (mean relative errors)

	M_1	M_2	M_3	M_4	M_5	M_6
			Compression/Decompression			
`blaz`	0.43%	0.39%	0.53%	0.44%	1.95%	1.17%
`zfp`	0.0006%	0.0009%	0.02%	0.002%	0.13%	0.15%
			Additions (`blaz` & `zfp`)			
M_1	–	0.98%	0.67%	0.91%	0.72%	0.78%
M_2	0.001%	–	0.62%	1.07%	1.76%	1.61%
M_3	0.82%	0.12%	–	2.27%	0.71%	0.65%
M_4	0.03%	0.42%	0.27%	–	1.68%	1.70%
M_5	0.68%	1.62%	1.25%	0.89%	–	0.94%
M_6	0.31%	1.27%	0.25%	2.32%	0.16%	–

Case Study: Climate Simulation Data







- Scientific Data Reduction Benchmarks[6]
- SDRBench: `https://sdrbench.github.io`
- 3600×1800 matrices, (CESM-ATM dataset 1)

[6] K. Zhao, S. Di, X. Liang, S. Li, D. Tao, J. Bessac, Z. Chen, and F. Cappello, "SDRBench: Scientific Data Reduction Benchmark for Lossy Compressors", International Workshop on Big Data Reduction (IWBDR2020), in conjunction with IEEE Bigdata20.

Conclusion & Perspectives

Conclusion

- It is possible to compute among compressed matrices
- Related work: Precision tuning (programs: `POP`, neural networks)

Perspectives

- Extend `blaz` to more linear algebra operators (stencils, reductions, ...)
- Experiment on real-world applications
- Introduce various compression rates (add interpolation, change quantization)
- Develop a parallel (MPI/GPU) version of `blaz` (block splitting $\Rightarrow$ // scalability)

References

- M. Martel, *Compressed Matrix Computations*, IEEE/ACM International Conference on Big Data Computing, Applications and Technologies, BDCAT 2022
- T. Agarwal, H. Dam, P. Sadayappan, G. Gopalakrishnan, D. Ben Khalifa, M. Martel, *What Operations can be Performed Directly on Compressed Arrays, and with What Error?* 9th International Workshop on Data Analysis and Reduction for Big Scientific Data, ACM Press, 2023, (**Best Paper Award**)
- D. Ben Khalifa and M. Martel, *Compile-Time Optimization of the Energy Consumption of Numerical Computations*, Compiler Frontiers Workshop, ACM Press, 2024