GCPU_OpticalFlow: a GPU accelerated Python software for strain measurement

Ahmed Chabib, Jean-François Witz, Pierre Gosselet, Vincent Magnier

GCPU_OpticalFlow: a GPU accelerated Python software for strain measurement

Ahmed Chabib, Jean-François Witz,
Pierre Gosselet and Vincent Magnier
Univ. Lille, CNRS, Centrale Lille, UMR 9013
LaMcube, F-59000 Lille, France

March 14, 2023

Abstract

This paper introduces an open-source pixel-wise Digital Image Correlation tool written in Python and targeting graphics processing units (GPUs) with the help of Cupy and Rapids-cuCim libraries. It is capable of computing the kinematic fields that transform an image into another in an efficient and quick way and it allows to treat large images in the GPU. Even if GCPU_OpticalFlow can be easily used by communities concerned by the estimation of displacement, it is particularly tuned to estimate consistent strain (gradient) field. The detection of a crack in a material is presented in this work as a demonstration.

Keywords: DIC; Optical Flow; Python; GPU; Mechanics; Strain Measurement; CUDA

1 Motivation and significance

1.1 Introduction

Digital Image Correlation **(DIC)** is one of the most popular optical methods that provide full-field displacement between consecutive images. After the notable advancement of this technique, DIC has become more and more used in several scientific disciplines, notably in materials science for strain measurement in order to deduce the properties of materials.

Most of DIC software are based on a local [24, 25, 26, 27, 28] or a global approach [24, 29, 34, 35]. As a result, the displacement field is calculated over subsets of pixels. We propose through this paper a new open source code named GCPU_OpticalFlow, based on D. Sun [4] methods, able to describe the motion at each single pixel.

The calculation of the kinematic fields from a sequence of images captured with modern high-resolution cameras become increasingly expensive in terms of memory and computation, due to the copious volumes of data (tomography etc.). Hence the need to develop an optimized fast code that takes up less memory. We propose to use a matrix-free implementation of Krylov iterative solvers [14] suited to Graphics Processing Units(GPU) so that no voluminous matrix needs to be stored. Some of the existing DIC programs are written with a low level programming language (C++ for the example) like DICe [24], Ncorr [25] which make their understanding, execution and modification complicated. Our code is implemented in the popular python language and relies on classical libraries to optimize its performance and offload computations on the available hardware

accelerators. Presently, the DIC programs are adapted to different architectures, like CPU [24, 25, 26, 27, 28], Raspberry Pi [23] or even in GPU [20]. In contrast to this software where no assumption on the nature of the displacements is made, the previous work of our research team GPUcorrel [20] is based on a global approach and uses an integrated correlation step. In other terms, the program expects a list of displacement fields to extract from the image sequence and the calculated result is a linear combination of the given basis. Moreover, GPUcorrel uses PyCUDA kernels which have the form of CUDA-C code to target the GPU. Our software uses Cupy [15] and Cucim [16] libraries that contain a GPU-accelerated version of Numpy, Scipy and Skimage functions and called easily using the same syntax. The engine is able to use the alternatives of the GPU libraries on CPU when the GPU or one of its related libraries is missing. For instance, if CuPy is not found, Numpy will be used instead, all in the same single straightforward Python code comprehended by non-experimented developers.

This article and software represents a bridge between the optical flow community that does not deal with mechanical imaging and the DIC community, and enables the users, particularly from mechanical engineering, to measure the pixel-wise displacements and efficiently deduce strain.

1.2 Principles of DIC and Optical Flow

Let f and g be a sequence of two images, where f is the reference image and g is the deformed one. The objective of DIC is to find a transformation of the image that keeps the gray levels invariant between two images.

$$f(x) = g(x + u(x)),\ u \text{ is the displacement field.} \tag{1}$$

It is impossible to estimate the displacement that transforms f to g at each pixel directly from the conservation law (1) as the number of unknowns exceeds the number of equations. To solve this problem, DIC algorithms transform the ill-posed optical flow problem into a well-posed one by using a grid of pixels, aiming at reducing the number of the degrees of freedom [2]. This grid can be local, FEM (Finite Element Method) [36] or even X-FEM (Extended Finite Element Method) [3]. Therefore, the displacement, written u_h, is interpolated on a basis of continuous shape functions with local support. In order to quantify the difference between the images, the integral of squared differences is chosen as a metric. Then the purpose of DIC is the minimization of the objective function (2) with respect to u_h.

$$E(u_h) = \int_\Omega \Big(f(x) - g\big(x + u_h(x)\big)\Big)^2 dx \tag{2}$$

The use of a reduced dimension search space acts in itself as a regularization. Nevertheless, when willing to decrease the size of the grid, a Tikhonov-type regularization is commonly added, leading to a new objective function:

$$E(u_h) = \int_\Omega \Big(f(x) - g\big(x + u_h(x)\big)\Big)^2 + \lambda(\nabla u_h)^2 dx,\ \text{for } \lambda > 0. \tag{3}$$

Optical flow methods share the purpose of DIC but they make the hypothesis that a well mastered regularization makes it possible to get rid of the interpolation grid, leading to computations at the pixel scale, see [1, 7]:

$$E(u) = \int_\Omega \Big(f(x) - g\big(x + u(x)\big)\Big)^2 + \lambda(\nabla u)^2 dx \tag{4}$$

Notice that in [4], other energies were proposed where Li and Osher's median filter [13] was interspersed. This resulted in the removal of the outliers at the cost of a slight energy increase.

2 Software description

2.1 Software functionalities

GCpu_ OpticalFlow is coded in Python language and requires some external modules. It is implemented as simply as possible in order to be understandable and editable by all the researchers regardless of their programming skills. Therefore, the software uses Cupy and cuCim to run on the GPU. Cupy is a high-level GPU-accelerated computing library which contains a GPU version of NumPy and SciPy libraries able to run on NVIDIA CUDA or on AMD ROCm platforms, whereas cuCim is a library that provides a CUDA-accelerated implementation of a wide range of image processing operations existing in scikit-image. This software only works with NVIDIA GPUs with at least the 3.0 version of Compute Capability.

At every warping step, the algorithm requires the resolution of the following linear system:

$$\begin{bmatrix} I_x^2 + \lambda\Delta & I_x I_y \\ I_x I_y & I_y^2 + \lambda\Delta \end{bmatrix} \begin{bmatrix} dU^{k+1} \\ dV^{k+1} \end{bmatrix} = \begin{bmatrix} I_x I_t + \lambda\Delta U^k \\ I_y I_t + \lambda\Delta V^k \end{bmatrix}, \tag{5}$$

where k is the current iteration, dU^{k+1} and dV^{k+1} are respectively the horizontal and the vertical flow increment, U^k and V^k are respectively the horizontal and the vertical current estimated flow field, λ represents the regularization parameter, I_x and I_y are the spatial derivatives of the image, I_t is the temporal derivative and Δ is the discrete Laplace operator. When handling high resolution images, the matrix can not fit in the memory and therefore the resolution is computationally costly.

The preconditioned minimum residual method MINRES [17] is used in this software. The choice of this solver is motivated not only by its ability to take advantage of the symmetry of the system matrix but also by the possibility of a matrix-free implementation. While evaluating the matrix-vector product in the resolution step, the Laplace operator is computed using the *laplace()* function of scipy or cupyx.scipy for GPU, to avoid any additional matrix storage.

In numerical analysis, the preconditioner is a practical tool to increase the rate of convergence, in our case it has been observed that using the following preconditioner P can decrease the number of iterations that MINRES takes to converge.

$$P = \begin{bmatrix} (I_x^2 + 8\lambda)^{-1} & 0 \\ 0 & (I_y^2 + 8\lambda)^{-1} \end{bmatrix} \tag{6}$$

As usually done in DIC methods, the gradient of the second warped image $\nabla g(x+u)$ is replaced by the gradient of the reference image of the sequence $\nabla f(x)$ so that costly computation can be avoided. This can be explained by the fact that at the convergence the images of the sequence converge to the same solution [18].

2.2 Software architecture

To estimate flow fields with large motion, the method uses an incremental multi-resolution technique. A pyramid of images is built by down-sampling the sequence. The flow field estimated in a level is thereafter up-sampled and used to initialize the next level as illustrated in Fig. 1. The main function of the software is *compute_flow_ base()*. It returns u_l and v_l, the horizontal and the vertical displacements for a specified level l after solving the system (5) 10 times and getting the best flow increments, and it takes as one of its arguments the up-sampled displacements computed in the previous level as an initial state. It should be noted that in the top level, the initialization is given by *optical_flow_ tvl1()* function of skimage or cucim.skimage. *compute_flow_ base()* is

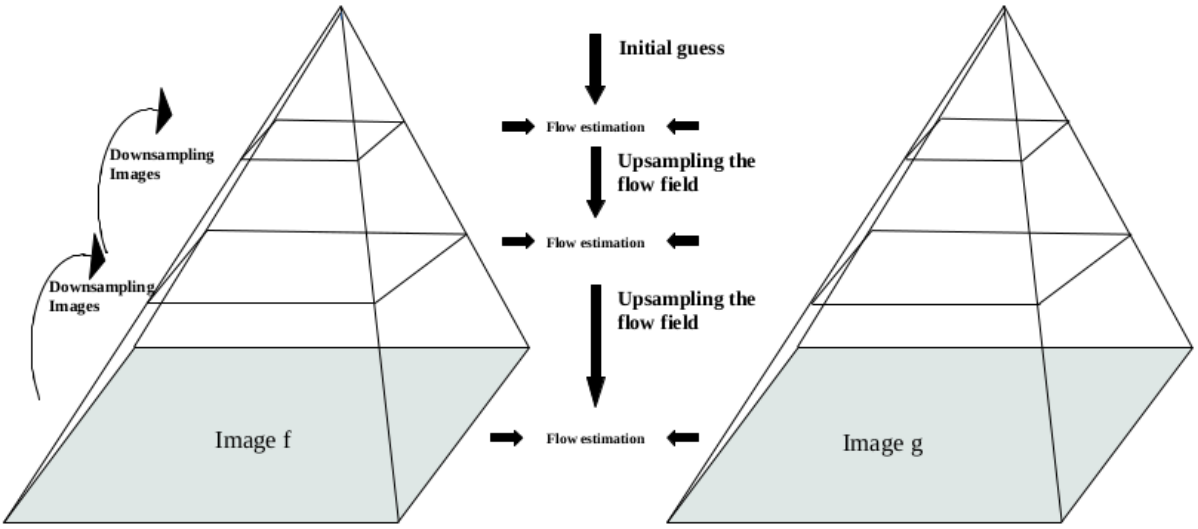


Figure 1: Representation of the pyramidal approach

called at each level by the *compute_flow()* function which allows the management of the pyramid and returns the final estimated displacements. The flowchart in Fig. 2 shows the main functions of the software modules.

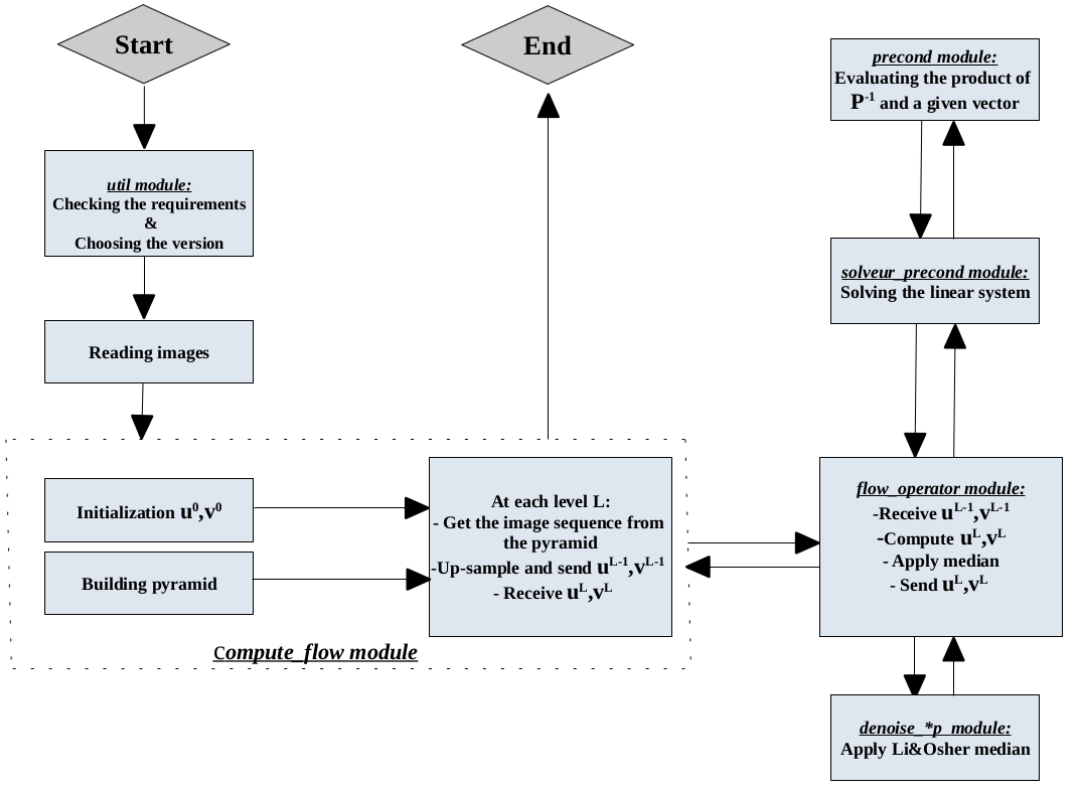


Figure 2: Flow-chart of flow field estimation with *GCPU_OpticalFlow*

3 Illustration

We created, in the folder *Test*, a script named *mainscript.py* which allows the user to quickly test the program. This script is able to estimate the motion from the sequence

of images in the *Images* folder. It generates two *.npy* files, *u_ cucim* and *v_ cucim* corresponding to the horizontal and vertical displacements.

To demonstrate the usefulness of the method, we used two images of a holed ±45 carbon/epoxy composite material specimen Fig. 4 on which a uniaxial tensile force is applied. The setup is schematized in Fig. 3, it generates a sequence of 1120×7830 images where each pixel is equivalent to 0.022 mm. This type of material is chosen because of its particularity of creating cracks, which provides an excellent case to test and to analyze the method on discontinuities.

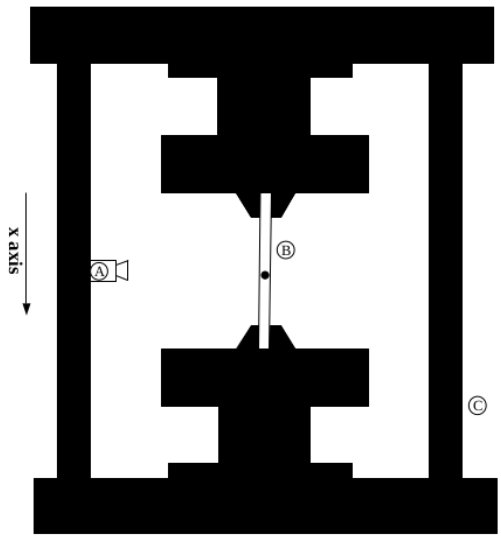


Figure 3: Experimental setup with (A) the camera, (B) the sample under testing and (C) the uniaxial tensile testing machine.

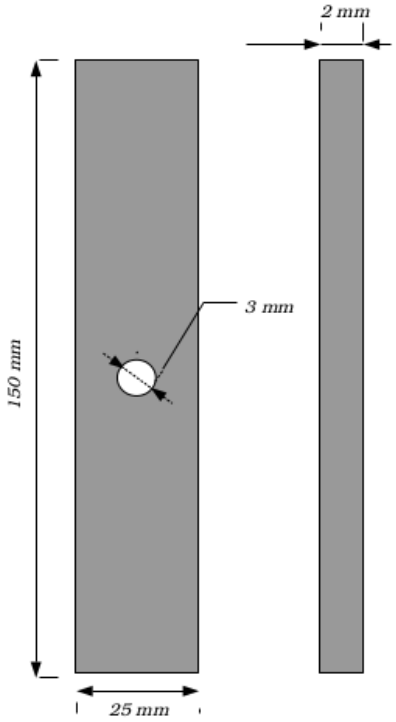
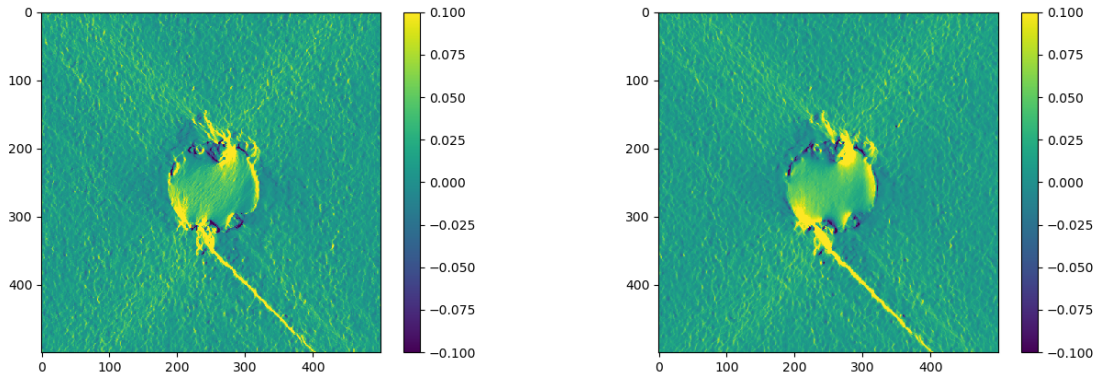


Figure 4: Dimensions of the used ±45 Carbon Epoxy specimen.

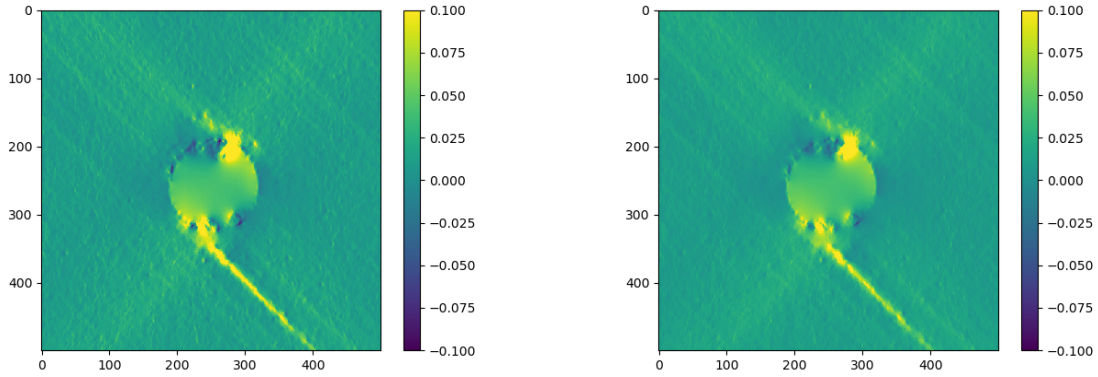
The results presented in Fig. 5 and Fig. 6 show that the method can detect the only crack in the sample. We observe that the choice of the amplitude of the regularization parameter as well as the size of the window of the median filter could play an important role. Increasing the value of λ leads to decreasing the noise on the image but it can lead to strain diffusion as the result of using a quadratic norm. We also remark that increasing the size of the median filter can provide less noisy results.

4 Impact

In order to compare the performance of our method with a global DIC software, we use YaDICS [22], a program which demonstrated its competence to effectively generate the kinematic fields [21] in various disciplines of experimental mechanics. The window size in the DIC approach is decreased to the maximum in order to compare the performance, since GCPU_OpticalFlow generates the fields at every pixel. The result is shown in Fig. 7 and Table 1. We observe, that the crack is clearly detected. Note that the interpolation error given by the norm of the difference between the warped image and the reference one, is 5.42×10^{-2} and 5.64×10^{-2} for YaDICs and GCPU_OpticalFlow respectively, meaning that the quality of the solution is unchanged. The main difference is that the



(a) $\lambda = 3 \times 10^3$ (b) $\lambda = 5 \times 10^3$



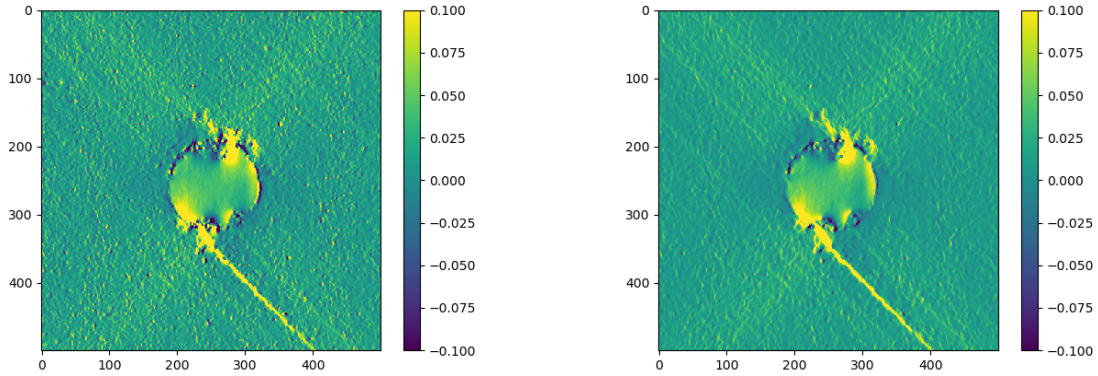
(c) $\lambda = 5 \times 10^4$ (d) $\lambda = 10^5$

Figure 5: Uniaxial strain ε_{xx} in function of λ with fixed 3×3 median filter

DIC method takes more than 11 minutes for YaDICs using an AMD Ryzen Threadripper 1950X 16-Core Processor. On the other side, due to the performance improvement efforts, the GCPU takes only 15 seconds using a the same processor with an NVIDIA GeForce RTX 3070 GPU, meaning that our software in this case is 44 times faster.

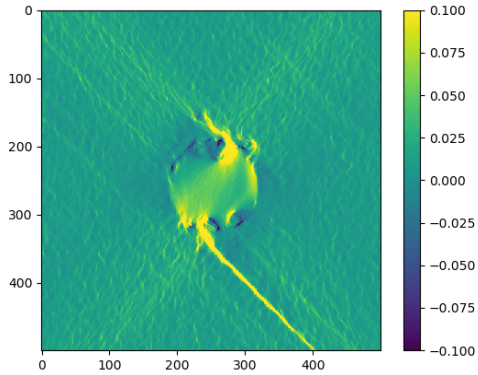
For the purpose of getting a clear idea about the difference between the residuals of both software, a graph is presented in Fig. 8. We mean by the image energy, the energy calculated by first term of the function E (4) related to the gray value constancy, while the gradient displacement energy is the value given by the gradient of the transformation that figures in the second term of the same function E. Different values of λ varying from 3×10^3 to 10^6 are used for GCPU, and various element size between 2 and 50 are used in YaDICs case, since these two parameters are meant to smooth the calculated fields by each program. It can be clearly observed that for an equivalent smoothness level, GCPU provides better quality results as its image energy is lower than YaDICs'. To quantify the difference, the areas under the curves are measured and we can notice that the value given by YaDICs is 6.42 greater than the one calculated by GCPU_OpticalFlow.

The methods used by the program were coded in [6] using Matlab. Unlike Python which became a vastly accepted open-source programming language, Matlab isn't free nor open-source. The Matlab implementation was expensive in terms of memory and calculations. To estimate the motion for an $(N \times N)$ pixels sequence, the storage of $O(N^4)$ coefficients of Laplace matrix and of the main matrix of the linear problem (5) was needed. Thanks to GCPU_OpticalFlow, we are able to solve the same problem using only $O(N^2)$ coefficients. Using this approach, we can treat larger volume of data on the

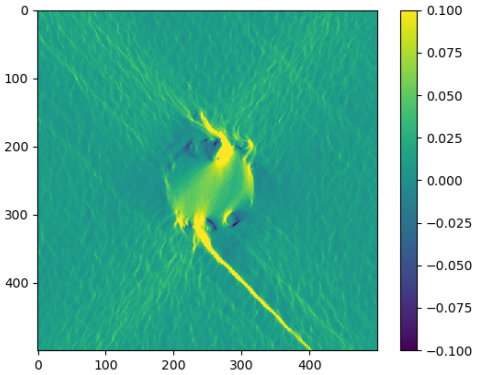


(a) Median 3×3

(b) Median 5×5



(c) Median 9×9



(d) Median 11×11

Figure 6: Uniaxial strain ε_{xx} in function of the median size with fixed $\lambda = 3 \times 10^3$

GPU.

In the future this work will be integrated to Crappy [19], which is a locally developed Python module capable of commanding advanced experimental mechanical tests and able to acquire data in real time.

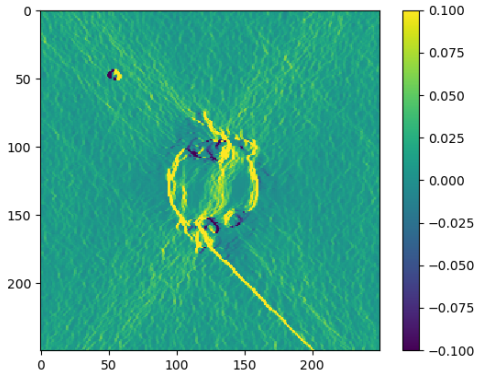
5 Conclusion

The presented software is an open-source Python module for full field displacement and strain computing, able to run in both GPU and CPU and based on D. Sun models. In our implementation, we used a matrix-free Krylov solver to reduce the computational cost and the memory storage. It has been shown by the example that the code computes efficiently the strain at each pixel of the image. We also presented the benefits and limitations of the size of the median filter and the parameter of regularization on the discontinuities.

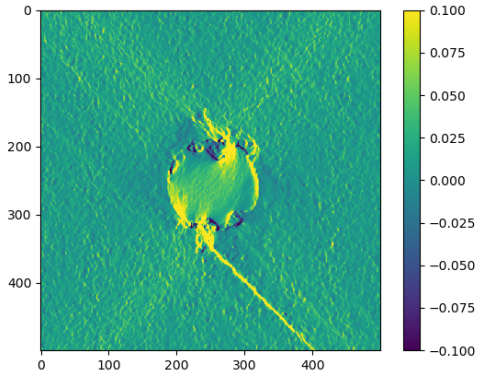
Finally, in the long term this method will be extended to work on three-dimensional tomographic images.

6 Acknowledgements

The authors wish to express their very special appreciation to Mr. Victor Couty and Mr. Adrien Berger for providing them with test images.



(a) YaDICs strain with 2×2 window, 3×3 median



(b) GCPU strain $\lambda = 3 \times 10^3$, 5×5 median

Figure 7: Strain field computed by YaDICs and GCPU_OpticalFlow

Field	Method	Mean	Std	Median	NMAD
u	$YaDICs$	-5.307e+1	2.881e+1	-5.375e+1	2.436e+1
	$GCPU$	-5.315e+1	2.868e+1	-5.388e+1	2.428e+1
Δu	$***$	8.00e-2	1.3e-1	1.3e-1	8.00e-2
v	$YaDICs$	-1.367e-1	3.391e+0	-7.050e-2	2.755e+0
	$GCPU$	-2.210e-1	3.312e+0	-2.500e-1	2.606e+0
Δv	$***$	8.43e-2	7.90e-2	1.79e-1	1.49e-1
ε_{xx}	$YaDICs$	1.35e-1	4.500e-2	1.070e-2	6.832e-3
	$GCPU$	1.32e-1	2.640e-2	1.054e-2	5.678e-3
$\Delta\varepsilon_{xx}$	$***$	3.00e-3	1.86e-2	1.60e-4	1.15e-3
ε_{yy}	$YaDICs$	-7.000e-3	1.840e-2	-8.000e-3	5.37e-3
	$GCPU$	-9.063e-3	1.338e-2	-8.000e-3	3.728e-3
$\Delta\varepsilon_{yy}$	$***$	2.06e-3	5.02e-3	0.00e-3	1.64e−3
ε_{xy}	$YaDICs$	-1.700e-4	2.431e-1	-8.38e-5	2.736e-3
	$GCPU$	-1.731e-4	1.353e-2	0.000e-3	3.564e-3
$\Delta\varepsilon_{xy}$	$***$	3.10e-6	2.29e-1	-8.38e-5	-8.28e-4

Table 1: Statistical indicators of GCPU_OpticalFlow and the interpolated YaDICs fields and their differences

7 Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper

References

[1] B.K.P. Horn and G. Schunck. Determining optical flow. Artificial Intelligence, 17:185–203, 1981.

[2] B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In Proceedings of Imaging Understanding Workshop, pages 121–130, 1981.

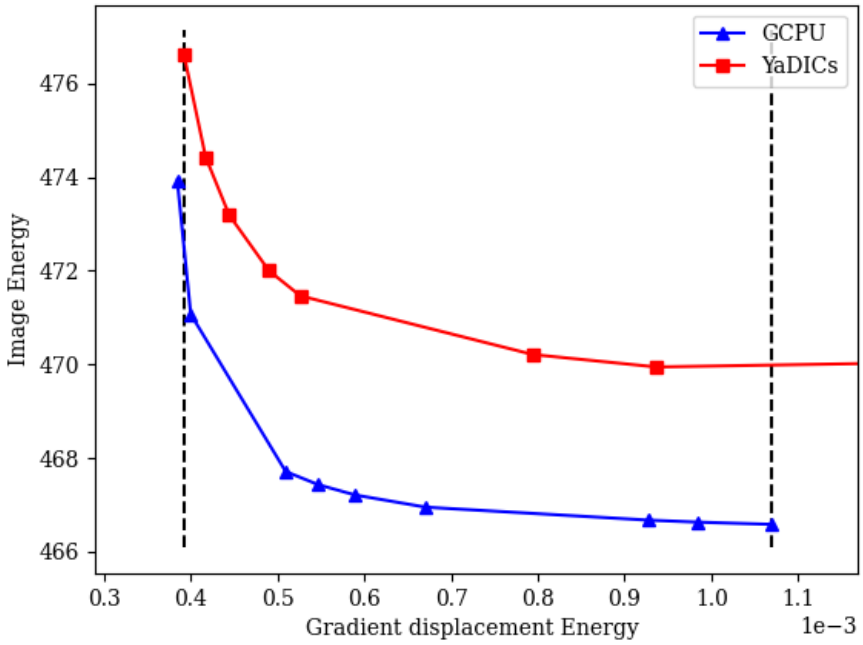


Figure 8: Energy image in terms of gradient displacement energy for both software. The dotted vertical lines indicate the limits of the zone where the two software are compared

[3] J. Réthoré, F. Hild, S. Roux, Shear-band capturing using a multiscale extended digital image correlation technique, Computer Methods in Applied Mechanics and Engineering,Vol. 196, Issues 49–52, 2007.

[4] D. Sun, S. Roth and M. J. Black. Secrets of optical flow estimation and their principles. Computer society conference on computer vision and pattern recognition. IEEE, 2010. p. 2432-2439.

[5] S. Baker, D. Scharstein, J. Lewis, S. Roth, MJ. Black and R. Szeliski. A database and evaluation methodology for optical flow. In ICCV, 2007

[6] Link to Matlab implementation by D. Sun `http://www.cs.brown.edu/people/dqsun/`

[7] M. J. Black and P. Anandan. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. CVIU, 63:75–104, 1996.

[8] A. Wedel, T. Pock, C. Zach, D. Cremers, H. Bischof. An improved algorithm for TV-L1 optical flow. Dagstuhl Motion Workshop, 2008.

[9] LI. Rudin, S. Osher, E. Fatemi. Nonlinear total variation based noise removal algorithms. Phys. D, 60(1-4):259–268, 1992

[10] D. Sun, S. Roth, J. Lewis, and M. J. Black. Learning optical flow. ECCV, vol. 3, p. 83–97 (2008)

[11] A. Blake and A. Zisserman. Visual Reconstruction. The MIT Press, Cambridge, Massachusetts, 1987.

[12] A. Chabib, J-F. Witz, P. Gosselet, V. Magnier. Pixel-wise full-field strain measurements for analysis of strain heterogeneity. PhotoMechanics – iDICs 2021, Nov 2021, Nantes, France. ffhal-03419422

[13] Y. Li and S. Osher. A new median formula with applications to PDE based denoising. Commun. Math. Sci., 7(3):741–753, 2009

[14] Y. Saad. Numerical methods for large eigenvalue problems, SIAM, (2011)

[15] *Nvidia Cupy documentation.* `https://docs.cupy.dev/en/stable`

[16] *Nvidia cuCim documentation.* `https://docs.rapids.ai/api/cucim/stable`

[17] Paige, C. C., M. A. Saunders. Solution of Sparse Indefinite Systems of Linear Equations. SIAM J. Numer. Anal., Vol.12, 1975, pp. 617-629.

[18] J. Neggers, B. Blaysat, JPM. Hoefnagels, MGD. Geers. On image gradientsin digital image correlation: On image gradients in digital image correlation 105 (4) 243–260.

[19] V. Couty, J-F. Witz, C. Martel, F.Bari, A. Weisrock. CRAPPY: Command and Real-Time Acquisition in Parallelized Python, a Python module for experimental setups. SoftwareX, Vol. 16, 2021, pp.100848.

[20] V. Couty, J-F. Witz, P. Lecomte-Grosbras ,J. Berthe, E. Deletombe, M. Brieu. GPU-Correl: A GPU accelerated Digital Image Correlation software written in Python. SoftwareX, 2021, Vol. 16, pp.100815.

[21] N. Dahdah, N. Limodin, A. Bartali, JF. Witz, R. Seghir, E. Charkaluk, JY. Buffiere. Damage Investigation in A319 Aluminium Alloy by X- ray Tomography and Digital Volume Correlation during In Situ High-Temperature Fatigue Tests. In Strain, Wiley-Blackwell, 2016, 52 (4), pp.324-335.

[22] R. Seghir, J-F. Witz, S. Coudert,2014. Yadics-digital image correlation 2/3d software. `http://yadics.univ-lille1.fr/wordpress/`

[23] P. P. Das, M. R. P. Elenchezhian, V. Vadlamudi, K. Reifsnider, R. Raihan. RealPi2dDIC: A Low-cost and open-source approach to in situ 2D Digital Image Correlation (DIC) applications. SoftwareX, 2021, Vol. 13, pp.100645.

[24] D. Turner , P. Crozier, P. Reu. Digital image correlation engine (DICe). 2015, Sandia National Laboratory: Albuquerque, NM, USA.

[25] J. Blaber , B. Adair, A. Antoniou. Ncorr: open-source 2D digital image correlation matlab software. Exp Mech 2015;55(6):1105-22.

[26] V. Belloni, R. Ravanelli, A. Nascetti,M. Di Rita, D. Mattei, M. Crespi. Py2DIC: A new free and open source software for displacement and strain measurements in the field of experimental mechanics. Sensors 2019;19(18). `http://dx.doi.org/10.3390/rs12182906`.

[27] V. Belloni V, R. Ravanelli, A. Nascetti, M. Di Rita, D. Mattei, M. Crespi. Digital image correlation from commercial to fos software: A mature technique for full-field displacement measurements. Int Arch Photogr Remote Sens Spatial Inf Sci 2018;42(2).

[28] R. Ravanelli, A. Nascetti, M. Di Rita, V. Belloni, D. Mattei, N. Nistico, M. Crespi, 2017. A new digital image correlation software for displacements field measurement in structural applications. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XLII-4/W2,pp. 139-145.

[29] SN. Olufsen, ME. Andersen, E. Fagerholt μDIC: An open-source toolkit for digital image correlation. SoftwareX 2020;11:100391. http://dx.doi.org/10.1016/j.softx.2019.100391.

[30] PP. Das, MRP. Elenchezhian, V. Vadlamudi, K. Reifsnider, R. Raihan. RealPi2dDIC: A low-cost and open-source approach to in situ 2D digitalimage correlation (DIC) applications. SoftwareX 2021;13:100645. http://dx.doi.org/10.1016/j.softx.2020.100645.

[31] D. Atkinson, T. Becker A 117 line 2D digital image correlation code written in MATLAB. Remote Sens 2020;12(18). http://dx.doi.org/10.3390/rs12182906.

[32] J. Yang, K. Bhattacharya. Combining image compression with digital image correlation. Experimental Mechanics 2019;59(5):629-42. http://dx.doi.org/10.1007/s11340-018-00459-y.

[33] A. Plyer, G. Le Besnerais, F. Champagnat. Massively parallel lucas kanade optical flow for real-time video processing applications 11 1–18. http://dx.doi.org/10.1007/s11554-014-0423-0.

[34] Passieux J-C. jcpassieux/pyxel, original-date: 2018-10-06T17:55:07Z. https://github.com/jcpassieux/pyxel.

[35] Réthoré J. UFreckles, language: eng. http://dx.doi.org/10.5281/zenodo. 1433776. https://zenodo.org/record/1433776.

[36] G. Besnard, F. Hild, S. Roux. "Finite-element" displacement fields analysis from digital images: application to Portevin-Le Châtelier bands. Experimental Mechanics, Society for Experimental Mechanics, 2006, 46, pp.789-804. ff10.1007/s11340-006-9824-8ff. ffhal-00124513

Metadata

Nr.	Code metadata description	
C1	Current code version	v1.0
C2	Permanent link to code, repository used for this code version	`https://github.com/chabibchabib/GCpu_OpticalFlow`
C3	Permanent link to Reproducible Capsule	
C4	Legal Code License	GPLv2+
C5	Code versioning system used	git
C6	Software code languages, tools, services used	Python, Cupy, OpenCv, Rapids Cucim and Numba
C7	Compilation requirements, operating environments & dependencies	Numpy 1.20.3, Scikit-image 0.16.2, Scipy 1.6.3, OpenCV 4.2.0 or newer versions. Numba, NVIDIA CUDA GPU with the Compute Capability $\geqslant$ 3.0, CUDA Toolkit $\geqslant$ 10.2, Cupy and Cucim
C8	Link to developer documentation, manual	`https://gcpu-opticalflow.readthedocs.io/`
C9	Support email for questions	ahmed.chabib@univ-lille.fr

Table 2: Code metadata