Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

DPIVSoft-OpenCL: A multicore CPU–GPU accelerated open-source code for 2D Particle Image Velocimetry

Jorge Aguilar-Cabello, Luis Parras [*], Carlos del Pino

Andalucía Tech., Universidad de Málaga, Escuela de Ingenierías Industriales, Ampliación Campus Teatinos, 29071, Málaga, Spain

A R T I C L E   I N F O

Article history:
Received 28 March 2022
Received in revised form 6 September 2022
Accepted 3 November 2022

Keywords:
PIV
Fluid mechanics
Velocimetry
Open-CL
GPU
Python

A B S T R A C T

We present a translation of the original Matlab DPIVSoft code to a complete open source code implemented in Python, to perform Particle Image Velocimetry (PIV) in two-dimensions, in parallel, and with interrogation window shifting along with the double-pass window deformation approach using multiple iterations for each pass. The added value of the code is the use of the Open Computing Language (OpenCL) library to parallelize the original code on multiple Intel Central Processing Units (CPUs) and/or Graphics Processing Units (GPUs), so it can be run on all commercially available GPUs. Examples of flow application are included in the text using synthetic images generated from DNS data from John Hopkins Turbulence Database (JHTD) (Perlman, 2007), showing about 90x speedup over the previous Matlab implementation for a given test case.

Code metadata

Current code version	0.2.2
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-22-00074
Permanent link to Reproducible Capsule	
Legal Code License	GNU General Public License v3 (GPLv3)
Code versioning system used	git
Software code languages, tools, and services used	python, OpenCL.
Compilation requirements, operating environments & dependencies	Automatic install with pip https://pypi.org/project/dpivsoft/
If available Link to developer documentation/manual	https://gitlab.com/jacabello/dpivsoft_python/-/tree/master/dpivsoft/Examples/
Support email for questions	jacabello92@gmail.com

1. Introduction

There has been a rapid expansion of descriptive work on Digital Particle Image Velocimetry (DPIV) systems and their uncertainty quantification, along with a very noticeable increase in the large amount of data coming from digital cameras that have a higher spatial resolution [1,2] even using a mobile application [3]. However, to the best of our knowledge, no DPIV open source code has been developed to run on *any* computing platform including both Central Processing Units (CPUs) and Graphics Processing Units (GPUs) making use of Open Computing Language (OpenCL) [4].

The use of OpenCL has been applied to Computational Fluid Dynamics [5], but as far as we are aware, there are not many OpenCL developments for running DPIV [6,7]. Although there is groundbreaking research work regarding an OpenCL code for Tomographic PIV, there is no mention made of how to access the code [6]. Other development of PIV using OpenCL is QuickLab PIV-ng. This software consists of a Graphical User Interface (GUI), over a PIV code based on OpenCL [7], but unfortunately this is a project in progress.

There are other open source alternatives to perform DPIV such as PIVlab [8], JPIV [9] or OpenPIV [10,11]. The OpenPIV source code, also written in Python [10], has pioneered the use of GPUs in its cross-correlation algorithm and is in active development over the last few years. OpenPIV allows pre-processing of images, velocity calculations and post-processing of PIV data. This software architecture is based both on the rapid development of Python itself, and on fast and efficient execution. The GPU-accelerated PIV algorithm has been added as a module for OpenPIV, and can be used with any NVIDIA GPU system. One can use the PyCUDA API to interface the OpenPIV Python source code with a GPU [12]. The same OpenPIV code has been successfully

[*] Corresponding author.
E-mail address: lparras@uma.es (Luis Parras).

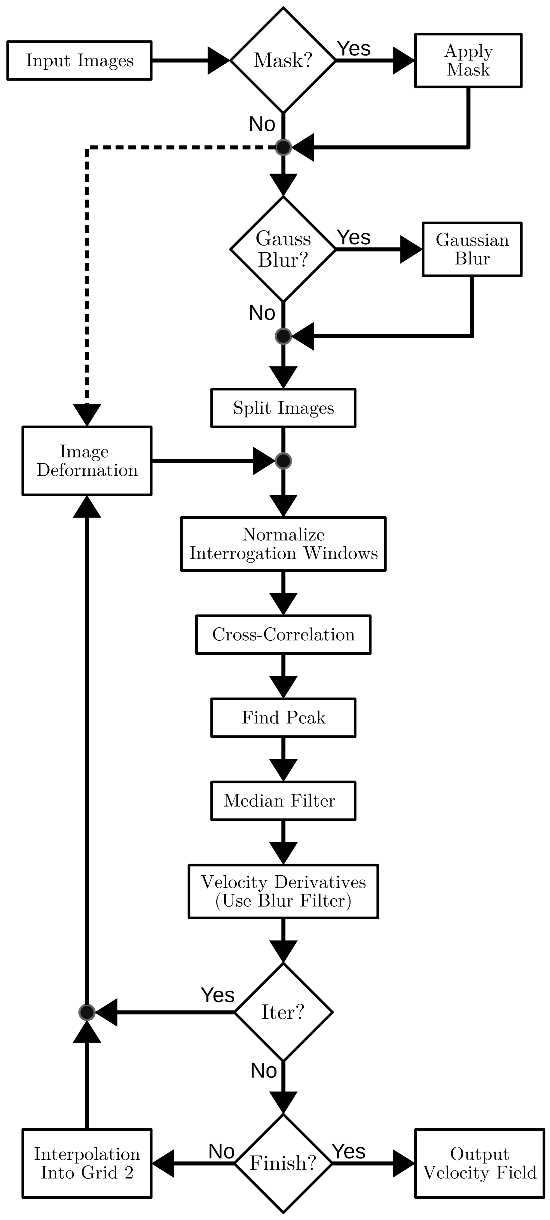


Fig. 1. DPIVSoft algorithm scheme.

adapted to the use of multiple NVIDIA GPUs [13]. Recently, other massively parallel optical flow methods have been developed that allow PIV processing which produce errors comparable to the aforementioned programs [14]. As an example, the FOLKI-PIV developed on GPUs allows for a speedup of 50 times with respect to the CPU version, and it provides very similar results to DPIVSoft [15,16].

The software DPIVSoft has been developed by Patrice Meunier & Thomas Leweke [17]. This program performs a double-pass algorithm with window shifting using image deformation, multiple iterations and sub-pixel accuracy that has proven to accurately reproduce a wide variety of vortex flows with high velocity gradients [18,19]. Hence, the velocity field obtained from the code should be of interest to detect vortices inside a fluid flow [20]. However, we found two drawbacks in the original code DPIVSoft that could be improved. Firstly, it is written in Matlab, which is a proprietary software with a high price per license. Secondly, despite the high calculation speed provided by this software, it is not optimized for use both in parallel execution and on GPUs. However, and given the nature of PIV algorithms and GPUs accessible today, the performance increase can be overwhelming. For both of the above reasons, we describe a (GPU)-accelerated DPIV algorithm using DPIVSoft as the basis for this development (DPIVSoft-OpenCL). Though the software could have been tested with any image generator [21], we rigorously have validated the algorithm using synthetically generated images of theoretical data and velocity fields from Direct Numerical Simulations (DNS) [22].

2. Software description

2.1. Particle image velocimetry with windows deformation

The algorithm consists of a multi-pass approach with window shifting using image deformation, so that windows are deformed according to the velocity gradients in an iterative process. We perform cross-correlation on each pair of windows and the cross-correlation peak is located with sub-pixel precision using a three-points Gaussian fitting estimator [23]. Each of these interrogation windows is normalized so that a noise-to-peak validation criteria can be used on the correlation peaks to avoid spurious vectors. Once the velocity field is obtained, we remove odd values by means of a median filter. The image deformation is performed using the gradients of the filtered velocity field making use of a blur filter to smooth the results. The deformed interrogation windows are used as new input for the cross-correlation. We repeat this process to improve the final velocity field as shown in the flowchart of Fig. 1.

The image deformation is done in a symmetric way, i.e. images at instants t and $t + dt$ are deformed respectively by displacements $-u/2$ and $+u/2$, where u is the displacement found at the previous iteration. This choice is equivalent to a second-order discretization in time.

We use bi-linear interpolation to obtain the velocity field in a finer grid, where the processing is repeated again to obtain the final velocity field.

We have also applied a Gaussian blur filter to the images only in the first iteration to increase the correlation peak of images containing small particles [17]. We have also implemented a mask tool in which the masked area of each correlation window is filled with the mean intensity of the unmasked area of the same correlation window to improve peak detection [24].

2.2. Multicore acceleration: OpenCL

This code has been developed using OpenCL, which is an open language and allows parallelization on AMD and NVIDIA GPUs [4], but the same code can even be run in parallel on CPUs without any modification, as shown below. The hosting platform used on the CPU is Python, which is an increasingly popular language with extensive library support and, unlike the original code platform written in Matlab, is open source. The first step is to read the pair of images, using the library OpenCV in Python. This step is the main bottleneck when loading large images for computation on the GPU. This process can take up to a 40% of the total processing time on the GPU. To increase performance in the DPIVSoft-OpenCL code, the following pair of images are loaded asynchronously during the processing time of the images, except for the first pair. No further communication with the CPU is required until the algorithm has finished.

The parallelism of the calculations implies that it is necessary to hold all the interrogation windows and intermediate results in memory at the same time. Each image pair is divided into k_1 interrogation windows of $N_{x1} \times N_{y1}$ pixels with the gray scale light intensity value. This division is stored, in turn, in two matrices of dimension $N_{x1} \times N_{y1} \times k_1$. Each pixel is written to its GPU memory location by a different thread to increase the computing speed. For the second pass, two new matrices of $N_{x2} \times N_{y2} \times k_2$ are used to store the already deformed correlation windows. To optimize the memory requirements of the code, these matrices, as well as all intermediate variables needed for the second step, point to the same real memory allocation as used in the first grid.

All the matrices used in the GPU are single precision so as not to compromise the performance. Furthermore, the effect on the accuracy of the results is almost negligible, when comparing those obtained in single precision on the GPU with double precision settings, using both the Matlab and the Python versions of DPIVSoft.

We have parallelized most of the operations presented in Fig. 1 at the correlation window level with three exceptions:

- Image pre-processing (masking, gaussian filtering and division the original image into correlation windows) is parallelized at the pixel level.
- In the process of window deformation, the code is parallelized at the pixel level (each core deforms a pixel).
- The Fourier transform necessary for cross-correlation is performed using Reikna library, which has different parallel algorithms depending on the size of the input.

3. Validation

3.1. Synthetic images from both theory and DNS and PIV errors

We compare the exact velocity field averaged at each interrogation window with the results obtained from the PIV code applied to synthetic images. The images are generated by an in-house code that creates 8-bit images using a random distribution of points. The particles were created using a Gaussian light intensity distribution following Eq. (1) as recommended in [2],

$$I(x, y) = I_o \exp\left[-\frac{(x - x_o)^2 + (y - y_o)^2}{d_\tau^2/8}\right] , \tag{1}$$

where d_τ is the diameter of the particle located at (x_o, y_o) with a peak intensity of I_o. The intensity I_0 and particle diameter d_τ are random variables of average and standard deviation of (200, 80) and (2, 0.5), respectively. The tracers are placed randomly along the image with a density of 0.05 particles/pixel.

The error ϵ_{PIV} of one single 2D PIV result at a given point n, $\mathbf{v} = (u_n, v_n)$, compared to that exact (theoretical or DNS) vector provided at the same point $\mathbf{v}_e = (u_{e,n}, v_{e,n})$, is defined in pixels as

$$\epsilon_{PIV} = \sqrt{(u_n - u_{e,n})^2 + (v_n - v_{e,n})^2} . \tag{2}$$

To calculate the exact velocity field from both theoretical or DNS velocity results, the velocity field of all points inside the interrogation windows is averaged to obtain a mean of the velocities in the x and y axes directly in the center of the box. This process is equivalent to performing a box-averaged velocity field which would be the expected result of a perfect PIV processing.

3.2. Fundamental flows

The first test case will be the (theoretical) spatial wavelength response of the method as originally introduced by Scarano and Riethmuller [25] and later used by other authors as a benchmark [13]. This test allows to evaluate the frequency response of the DPIVSoft-OpenCL code. Synthetic images are generated using a sinusoidal displacement field

$$\mathbf{u} = A \sin\left(2\pi \frac{y}{\lambda}\right) \mathbf{e}_x, \tag{3}$$

where y is the vertical coordinate, A is the amplitude of the displacement, λ is the spatial wavelength, and the displacement only affects the x direction ($\mathbf{e}_x$). The amplitude A is fixed to 2 pixels and the wavelengths are chosen to be $\lambda \in [32, 1024]$.

The synthetic images are created with the analytic velocity field and then processed with DPIVSoft-OpenCL using double-pass window deformation approach, with window sizes of 64×64 pixels^2 and 32×32 pixels^2 for the first and second pass, respectively, each having a 50% of overlap. The results are shown in Fig. 2: the velocity profile calculated with DPIVSoft-OpenCL code compared to that given by the real velocity (a) and the frequency response of DPIVSoft-OpenCL code (b).

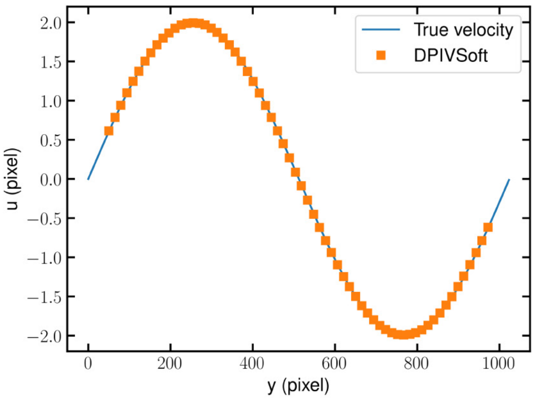
It can be observed in Fig. 2(a) that the maximum displacement and the maximum shear is well calculated by the DPIVSoft-OpenCL code for a wavelength of $\lambda = 1024$. Fig. 2(b) shows the transfer function of the algorithm that is a characterization of foremost importance related to spatial filtering. In this last figure we present the ratio of the average maximum displacement obtained by DPIVSoft with respect to the real maximum imposed in the synthetic images ($A = 2$ pixels) for different ratios of the interrogation window size with respect to the wavelength. The results in squares are compared with the theoretical correlation operator that can be approximated by a simple moving average whose value can be calculated as the cardinal sine curve [25].

3.3. Homogeneous buoyancy driven turbulence

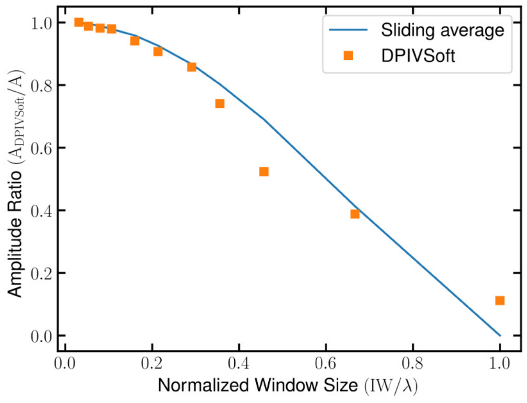
The results shown in this section come from the data from DNS of homogeneous buoyancy driven turbulence on a cubic domain of size $2\pi \times 2\pi \times 2\pi$ using 1024^3 points in a 3D periodic grid. The equations solved are the miscible two-fluid incompressible Navier–Stokes equations, which are obtained from the fully compressible Navier–Stokes equations with two species with different molar masses in the limit of infinite speed of sound, so that the individual densities of the two fluids remain constant. The code used to solve the DNS problem is described in [26] and it is initialized with random blobs. The Reynolds number of the simulation is 12500 and the typical evolution of the flow is starting from rest, then there is a generation of turbulence due to the differential movement created by buoyancy and after that, there is a decay of energy due to turbulence dissipation. For the example taken here, an intermediate solution has been chosen so the flow has multiple low scale vortices, which are very complicated to characterize by PIV measurements due to the resolution. For this case, we use synthetic images of 2048×2048 pixels^2. The simulation results for the velocity in the 1024^2 grid are mapped by interpolation onto a 2048^2 pixels of the generated image. Then, we created a normal distribution of particles with an optimal density, each one with the velocity corresponding to the pixel in which is plotted, obtained from interpolation from the CFD data. For the next picture, each particle has been displaced $\mathbf{v} \cdot \Delta t$. There are no changes in illumination or movements out of the plane so the only source of error in this case is the PIV algorithm itself. The vorticity is shown in Fig. 3(a) to illustrate the complexity of the flow patterns. The error in pixels, ϵ_{PIV}, is depicted in (b), while we present the results from DNS (c) and DPIVSoft-OpenCL using synthetic images (d).

Table 1 shows the error in pixels (mean, max and standard deviation) for different parameters regarding box size and iterations.

Although we carry out the calculation of the overall velocity field in the whole spatial domain accurately, compare Figs. 3 (c) and (d), it is observed that there are very small areas that have large errors in pixels as can be seen in Fig. 3(b) or maximum errors in pixels in Table 1. These large values of ϵ_{PIV} depend strongly on the spatial resolution and also on the synthetic images, but



(a)



(b)

Fig. 2. (a) Example of u_x velocity profile calculated with DPIVSoft-OpenCL compared to that given by the real velocity. The frequency of this velocity field is 1024 pixels. (b) Frequency response of the DPIVSoft-OpenCL code.

Table 1
PIV errors in pixels, ϵ_{PIV}, for buoyancy induced turbulence and different box sizes and passes.

Boxes size (pixels^2)	*Iter*	Error in pixels		
		Mean	Max	Std (σ)
128–64	1	0.16	1.9	0.20
	2	0.15	1.8	0.20
64–32	1	0.11	2.31	0.12
	2	0.10	2.30	0.11
32–16	1	0.11	1.45	0.09
	2	0.10	1.82	0.08

they appear in small areas. We have found that ϵ_{PIV} is correlated with a measure of the maximum velocity difference within the interrogation box. The coefficient of determination is $R^2 = 0.33$ for this strongly mixing-dominated flow. It would be interesting to determine the sources of error in PIV algorithms with iterative window deformation in future studies as it is beyond the scope of this work.

4. Performance

We have tested the performance increase that is achieved using parallel DPIVSoft-OpenCL acceleration. As stated below, this code was initially developed in Matlab, but it has been ported to Python and OpenCL. Hence, we can make use of multiple cores for a given CPU or several GPUs simultaneously. We used two GPU Workstations for the test. The first one (WP1) is an Intel i7-4770 (4 cores and 8 threads) and a NVIDIA GeForce GTX 780 with 3 GB of RAM and 2304 CUDA cores. The second workstation (WP2) is an AMD-Ryzen 9 3950X (16 cores and 32 threads) and a NVIDIA Geforce GTX 3070 with 8 GB of RAM and 5888 cores). The computations have been executed in OpenCL 1.2 which is the most modern version that supports both GPUs.

Once we have described the main characteristics of both GPUs and CPUs, we will characterize the improvement of computing time obtained using DPIVSoft-OpenCL. To that end, we measured the averaged computation time of each image pair for a different total number of correlation boxes in the second pass using all the available techniques allowed by the code: Python, Matlab, Python+OpenCL (CPU) and Python-OpenCL (GPU), for the two different workstations, being the Python implementation the slowest one. The average time is obtained from the computation of 50 images for each case. In all cases, the sizes of the correlation boxes are 32x32 pixels^2 and 16x16 pixels^2 for the first and second pass, respectively, and we increase the number of windows by increasing the size of the image. In Fig. 4 we show the speedup of the different implementations of DPIVSoft-OpenCL on the two different workstations. In WS1 (solid lines), we have compared the average time of calculation for Intel Python, Intel OpenCL and NVIDIA GPU with OpenCL with the average time of executing the original code in Matlab. It can be seen as for small images, the speedup is almost negligible, reaching a plateau for the case of Intel CPU with Python indicating that Matlab is around 6 times quicker than the Python code. We have performed the same calculation using the total number of cores of the Intel i7 CPU, and the behavior is very similar to the results using Python, reaching a plateau for the biggest image, so the speedup is roughly 4.8 times the Matlab processing time for the same image. On the other hand, the speedup by using GPUs keeps increasing as we increase the size of the image until we reach a speedup about 45 for images greater than 5 Mpx with respect to the original implementation in Matlab. Finally, and due to the reduced size of the RAM of the NVIDIA card, the last point could not be calculated. We repeat the process with the second workstation WS2 (dashed lines in Fig. 4). The results AMD CPU with Python reach a plateau for images greater than 2 Mpx of around 4.5 times slower than the Matlab version of the code. We have not been able to measure the DPIVSoft-OpenCL speedup in this 16 threads AMD CPU because AMD is not supporting the OpenCL libraries for their processors. In this last workstation, the NVIDIA card with OpenCL is able to produce a speedup around 90. To complete the comparison with other open-source codes, we have included the results of the processing time in the first workstation, WS1, for PIVLab, which, using parallelization in Matlab, achieves between 2 and 5 times speed-up with respect to DPIVSoft-Matlab, depending on the size of the image. On the other hand, the Python version of OpenPIV-Python produces a speedup of 1.65 with respect to the DPIVSoft-Python version.

5. Impact

The main focus of this study is the adaptation of the original DPIVSoft Matlab code [17] to a full open-source software, using Python and OpenCL in such a way that it can be run in parallel on Intel CPUs or GPUs of any brand. This original software has been successfully used to characterize trailing vortices [19,27] in two-dimensions, among other applications using rotating flows [28]. The effectiveness of our open source code is demonstrated by the following milestones: it can be run using GPUs of different brands (NVIDIA, AMD or even the future Intel Xe product) or multiple Intel CPUs. Furthermore, the relevance of the code is demonstrated by a drastic decrease of the computational time

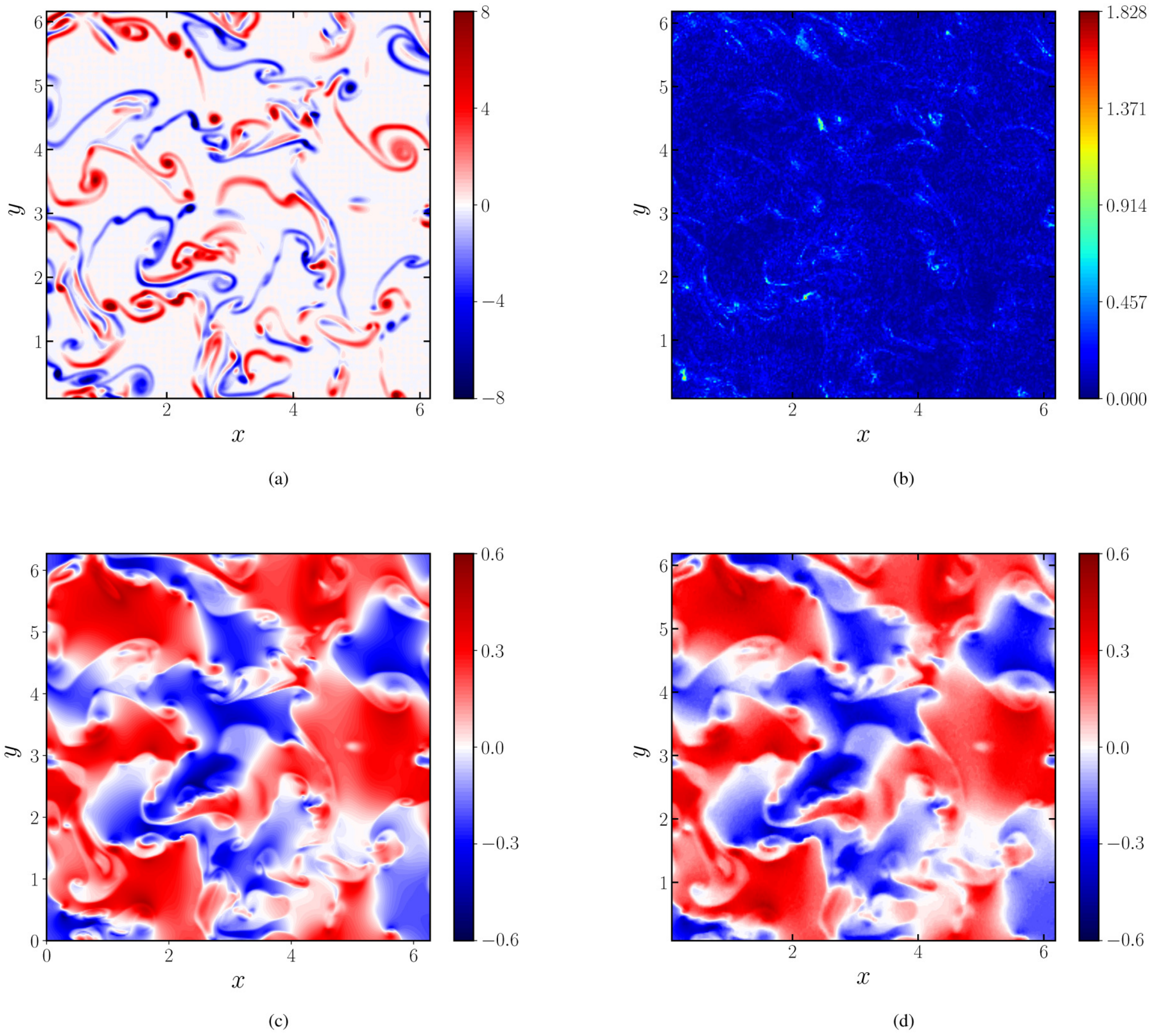


Fig. 3. Vorticity from the DNS data (a), and error ϵ_{PIV} (in pixels) in the estimation of the 2D velocity field (b), u_x velocity field for the DNS data of the homogeneous buoyancy driven turbulence (from JHTD) (c), and results obtained from synthetic images with DPIVSoft-OpenCL (d).

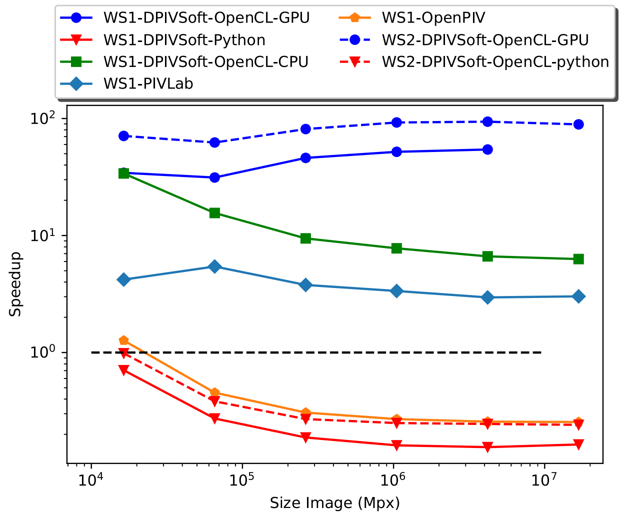


Fig. 4. Comparison on performance of the different implementations of DPIVSoft-OpenCL.

to achieve the same results using two complex problems. This performance improvement, together with the aforementioned ability to be run on different platforms, makes the application of this program an excellent tool for measuring the velocity field in any experimental setup in Fluid Mechanics or for real-time velocity measurement applications, as computation times of 0.12 s are achieved for 2048 px^2 images. Unfortunately, the real-time capability could not be tested as the program is implemented for offline processing, i.e. the images are captured by an external program and then processed by the DPIVSoft-OpenCL software. It is worth mentioning that other recent optical flows software, i.e. FOLKI-PIV, produce similar results with a time of 0.04 s in a different Workstation and with a different setup. This fact implies that an implementation of the Optical Flow correlation technique could further improve the efficiency of the current version of the code DPIVSoft-OpenCL. As a final result, we present in Table 2 a performance summary of the code along with other open-source PIV processing programs for the same settings: image size of 2048 $\times$ 2048 px^2, two passes, the first with 32 $\times$ 32 and the second with 16 $\times$ 16 px^2 interrogation windows, and with 50% overlapping.

6. Conclusions

We have ported the widely used Matlab code DPIVSoft to a full open-source alternative in Python that makes use of the library OpenCL to accelerate the processing in CPUs and GPUs.

Table 2
Summary table of the performance of the code presented together with other open-source PIV processing programs for a 2048 × 2048 px^2 image, two passes, the first with 32 and the second with 16 interrogation windows, with 50% overlapping.

Code	Language	CPU	GPU	RMS error (px)	Runtime (s)	Reference
DPIVSoft-Matlab	Matlab	X		0.08	9.91	[17]
DPIVSoft-python	Python	X		0.06	63.77	Current
DPIVSoft-OpenCL	Python-OpenCL	X		0.10	1.49	Current
DPIVSoft-OpenCL	Python-OpenCL		X	0.10	0.18	Current
PIVLab	Matlab	X		0.06	3.35	[8]
OpenPIV	Python	X		0.13	38.61	[10]

This implementation has multiple advantages: it can be installed in any workstation or cluster, the code can be accelerated by using Intel CPUs or any available GPUs (AMD, NVIDIA or Intel cards). We tested the algorithm using synthetic images in two flows: (i) one dimensional sinusoidal displacement to check the frequency response of the PIV code, and (ii) bidimensional DNS code from JHTD; a mixing buoyant flow, with a very complicated vorticity field. We have checked that the GPU implementation demonstrates the same accuracy than the original code. On the other hand, a huge increase in performance is achieved when using GPUs, which typically increases with the size of the images. The drawback of this implementation is the large amount of RAM needed to run all calculations on GPU without communicating with the host. In any case, with the present implementation, the code is able to accelerate up to 48 times the original Matlab code in some old NVIDIA card (Geforce GTX 780), and up to 94 times in a more recent NVIDIA Geforce GTX 3070 card. This result is very useful because in the last years there has been an increase of the size of the images for PIV as well as an increase of computing power with the new GPU cards. The DPIVSoft-OpenCL implementation works much better for larger size of images, so we are expecting much better results in future GPU cards. This first version of DPIVSoft-OpenCL could be updated including both a filter of the predictor displacement field [29] and other types of image masking as well as the expansion to a third dimension for stereoscopic PIV applications.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Jorge Aguilar-Cabello reports financial support was provided by the Ministerio de Economía y Competitividad of the Government of Spain through the project DPI2016-76151-C2-1-R. Luis Parras reports financial support was provided by the University of Malaga through the project B4-2019-11, 0837002010.

Data availability

Data will be made available on request.

Acknowledgments

This research has been supported by one grant from the Ministerio de Economía y Competitividad of Spain (Grant No. DPI2016-76151-C2-1-R) and partially by the project B4-2019-11, 0837002010 from the Universidad de Málaga and the project PID2021-124692OA-I00 from the Ministerio de Ciencia e Innovación. The authors would like to thank Patrice Meunier for his fruitful discussions on the original DPIVSoft algorithm and his comments on the final version of the manuscript.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.softx.2022.101256.

References

[1] Adrian RJ. Twenty years of particle image velocimetry. Exp Fluids 2005;39:159–69. http://dx.doi.org/10.1007/s00348-005-0991-7.
[2] Raffel M, Willert CE, Scarano F, Kähler CJ, Wereley ST, Kompenhans J. Particle image velocimetry: a practical guide. Springer; 2018, http://dx.doi.org/10.1007/978-3-319-68852-7.
[3] Minichiello A, Armijo D, Mukherjee S, Caldwell L, Kulyukin V, Truscott T, et al. Developing a mobile application-based particle image velocimetry tool for enhanced teaching and learning in fluid mechanics: A design-based research approach. Comput Appl Eng Educ 2020;1–21. http://dx.doi.org/10.1002/cae.22290.
[4] Stone JE, Gohara D, Shi G. OpenCL: A parallel programming standard for heterogeneous computing systems. Comput Sci Eng 2010;12(3):66–73. http://dx.doi.org/10.1109/MCSE.2010.69.
[5] Mossaiby F, Rossi R, Dadvand P, Idelsohn S. OpenCL-based implementation of an unstructured edge-based finite element convection-diffusion solver on graphics hardware. Internat J Numer Methods Engrg 2012;89:1635–51. http://dx.doi.org/10.1002/nme.3302.
[6] Lozhkin VA, Lozhkin YA, Tokarev MP. Application of high performance computing platforms to tomographic particle image velocimetry. Numer Methods Program 2012;13(1):20–7.
[7] Mendes L, Ricardo A, Ferreira RML. A customizable open-source software platform. In: Hydrosensoft, international symposium and exhibition on hydro-environment sensors and software. 2019, p. 1–8.
[8] Thielicke W, Stamhuis E. PIVlab – towards user-friendly, affordable and accurate digital particle image velocimetry in MATLAB. J Open Res Softw 2014;2(1):pe30. http://dx.doi.org/10.5334/jors.bl.
[9] Okamoto K, Nishio S, Saga T, Kobayashi T. Standard images for particle-image velocimetry. Meas Sci Technol 2000;11(6):685–91. http://dx.doi.org/10.1088/0957-0233/11/6/311.
[10] Taylor ZJ, Gurka R, Kopp GA, Liberzon A. Long-duration time-resolved PIV to study unsteady aerodynamics. IEEE Trans Instrum Meas 2010;59(12):3262–9. http://dx.doi.org/10.1109/TIM.2010.2047149.
[11] Ben-Gida H, Gurka R, Liberzon A. OpenPIV-MATLAB—An open-source software for particle image velocimetry; test case: Birds' aerodynamics. Softw X 2020;12:100585. http://dx.doi.org/10.1016/j.softx.2020.100585.
[12] Klöckner A, Pinto N, Lee Y, Catanzaro B, Ivanov P, Fasih A. PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation. Parallel Comput 2012;38(3):157–74. http://dx.doi.org/10.1016/j.parco.2011.09.001.
[13] Dallas C, Wu M, Chou V, Liberzon A, Sullivan PE. Graphical processing unit-accelerated open-source particle image velocimetry software for high performance computing systems. ASME J Fluids Eng 2019;141(11):111401. http://dx.doi.org/10.1115/1.4043422.
[14] Plyer A, Le Besnerais G, Champagnat F. Massively parallel Lucas Kanade optical flow for real-time video processing applications. J Real Time Image Process 2016;11:713–30. http://dx.doi.org/10.1007/s11554-014-0423-0.
[15] Champagnat F, Plyer A, Besnerais GL, Leclaire B, Davoust S, Le Sant Y. Fast and accurate PIV computation using highly parallel iterative correlation maximization. J Real Time Image Process 2011;50(1169). http://dx.doi.org/10.1007/s00348-011-1054-x.
[16] Giannopoulos A, Passaggia P, Mazellier N, Aider J. On the optimal window size in optical flow and cross-correlation in particle image velocimetry: application to turbulent flows. Exp Fluids 2022;63(57). http://dx.doi.org/10.1007/s00348-022-03410-z.
[17] Meunier P, Leweke T. Analysis and treatment of errors due to high velocity gradients in particle image velocimetry. Exp Fluids 2003;35:408–21. http://dx.doi.org/10.1007/s00348-003-0673-2.
[18] Albrecht T, Blackburn HM, Lopez JM, Manasseh R, Meunier P. Triadic resonances in precessing rapidly rotating cylinder flows. J Fluid Mech 2015;778:R1–12. http://dx.doi.org/10.1017/jfm.2015.377.
[19] García-Ortiz JH, Domínguez-Vázquez A, Serrano-Aguilera JJ, Parras L, del Pino C. A complementary numerical and experimental study of the influence of Reynolds number on theoretical models for wingtip vortices. Comput & Fluids 2019;180:176–89. http://dx.doi.org/10.1016/j.compfluid.2018.12.009.

[20] Lindner G, Devaux Y, Miskovic S. VortexFitting: A post-processing fluid mechanics tool for vortex identification. Softw X 2020;12:100604. http://dx.doi.org/10.1016/j.softx.2020.100604.

[21] Mendes L, Bernardino A, Ferreira R. Piv-image-generator: An image generating software package for planar PIV and optical flow benchmarking. Softw X 2020;12:100537. http://dx.doi.org/10.1016/j.softx.2020.100537.

[22] Perlman E, Burns R, Li Y, Meneveau C. Data exploration of turbulence simulations using a database cluster. In: SC '07: proceedings of the 2007 ACM/IEEE conference on supercomputing. 2007, p. 1–11. http://dx.doi.org/10.1145/1362622.1362654.

[23] Westerweel J. Digital particle image velocimetry: theory and application. Delft University Press; 1995.

[24] Theunissen R, Scarano F, Riethmuller M. On improvement of PIV image interrogation near stationary interfaces. Exp Fluids 2008;45(4):557–72. http://dx.doi.org/10.1007/s00348-008-0481-9.

[25] Scarano F, Riethmuller ML. Advances in iterative multigrid PIV image processing. Exp Fluids 2000;29:S051–60. http://dx.doi.org/10.1007/s003480070007.

[26] Livescu D, Mohd-Yusof J, Petersen MR, Grove JW. CFDNS: a computer code for direct numerical simulation of turbulent flows. Technical report, Los Alamos National Laboratory; 2009, LA-CC-09-100.

[27] García-Ortiz JH, Blanco-Rodríguez FJ, Parras L, del Pino C. Experimental observations of the effects of spanwise blowing on the wingtip vortex evolution at low Reynolds numbers. Eur J Mech B/Fluids 2020;80:133–45. http://dx.doi.org/10.1016/j.euromechflu.2019.12.007.

[28] Serrano-Aguilera JJ, Parras L, del Pino C, Rubio-Hernandez FJ. Rheo-PIV of Aerosil® R816/Polypropylene Glycol suspensions. J Non-Newton Fluid Mech 2016;232:22–32. http://dx.doi.org/10.1016/j.jnnfm.2016.03.015.

[29] Schrijer F, Scarano F. Effect of predictor–corrector filtering on the stability and spatial resolution of iterative PIV interrogation. Exp Fluids 2008;45:927–41. http://dx.doi.org/10.1007/s00348-008-0511-7.