CLIJ: Enabling GPU-accelerated image processing in Fiji

Robert Haase[1,2*], Loic A. Royer[3*], Peter Steinbach[1,2], Deborah Schmidt[1,2], Alexandr Dibrov[1,2], Uwe Schmidt[1,2], Martin Weigert[1,2], Nicola Maghelli[1,2], Pavel Tomancak[1,2], Florian Jug[1,2], Eugene W. Myers[1,2]

Abstract Graphics processing units (GPU) allow image processing at unprecedented speed. We present CLIJ, a Fiji plugin enabling end-users with entry level experience in programming to benefit from GPU-accelerated image processing. Freely programmable workflows can sped up image processing in Fiji by factor 10 and more using high-end GPU hardware and on affordable mobile computers with built-in GPUs.

Modern microscopy generates staggering amounts of multi-dimensional image data that place increasing demands on processing flexibility and efficiency. One way to speed up image processing is to exploit the parallel processing capabilities of graphics processing units (GPU).

Recently, GPU-acceleration was used in specific image processing tasks such as reconstruction[1,2], image quality determination[3], image restoration[4], segmentation[5] and visualisation[6]. However, in these tools, GPU code is fulfilling one specific purpose and is not intended to be reused in other contexts. By contrast, most common image processing tasks are solved by building flexible workflows consisting of simple operations in widely used tools such as ImageJ[7] and Fiji[8]. Most of these operations were however programmed at a time when GPUs were not commonly used for general purpose processing. Therefore, typical workflows consisting of core ImageJ operations do not take advantage of GPUs. To address this issue we developed a flexible and reusable platform for GPU-acceleration in Fiji.

Our platform, named CLIJ, complements core *ImageJ* operations with reprogrammed counterparts that take advantage of the Open*CL*[9] framework to execute on GPUs. Within CLIJ, we implemented a wide range of fundamental image processing functions for morphological filters, spatial transforms, image warping, local and global thresholding, minima/maxima detection, logical operations on binary images, 3D-to-2D projections, and methods of descriptive statistics for quantitative measurements (Suppl. Listing 1).

We then asked how much faster GPU-accelerated versions of individual operations run compared to their counterparts on the central processing unit (CPU). GPUs can do certain operations faster because they have many more processing cores then regular CPUs (Figure 1a). In addition, memory access can be multiple times faster on GPUs depending on the GPU hardware. On the other hand, in order to be processed on GPUs, the data and the compiled program have to be first pushed to GPU memory, and later data have to be pulled back to CPU memory. While this introduces an unavoidable overhead to any GPU operation, once the data are on the GPU, functions we implemented typically run faster on GPU compared to CPU. The speed-up depends on the image size and for functions that have parameters, the achievable speed may also depend on the values of the parameters (Figure 1b). Furthermore, after the first execution of a CLIJ operation, performance increases because of reuse of the compiled GPU code. We measured execution time and speedup on two test systems: a consumer laptop (Intel i7-8650U CPU and an Intel UHD 620 GPU), and a professional workstation (two Intel Xeon Silver 4110 CPUs and an Nvidia Quadro P6000 GPU). We observed that CLIJ operations were up to about 100 times faster compared to their counter parts in ImageJ running on the CPU (Figure 1c).

To demonstrate the utility of CLIJ in practical biological image processing, we chose a multi-step example workflow (Suppl. Figure 1) operating on 3D light sheet microscopy data consisting of 300 time points of a *Drosophila* embryo in stages 4-8 expressing histone-RFP to mark the nuclei. The workflow performs Difference-of-Gaussian filtering to reduce background signal and noise, projects the data from 3D to 2D and detects spots to count the nuclei. The processed image stack of each time point consists of $400\times1024\times121$ voxels occupying 189 MB in memory. Since CLIJ operations are new implementations of existing ImageJ functions based on a different computing architecture, we determined how much the output of the GPU-based workflow and the corresponding CPU-based workflow were different. We observed minor absolute differences in the spot count result of 0.9 ± 0.6 percent. Furthermore, we observed differences of 0.05 ± 0.04 percent in spot count depending on which GPU hardware CLIJ was executed (Suppl. Figure 2). While these differences may in practice be negligible (Suppl. Figure 3, Suppl. Video 1), we think users should be aware they may exist. The whole time lapse was processed on the laptop within 2 hours and 44 minutes using ImageJ and

Correspondence: rhaase@mpi-cbg.de & loic.royer@czbiohub.org
* both authors contributed equally
[1] *Max Planck Institute for Molecular Cell Biology and Genetics, Pfotenhauerstr. 108, 01307 Dresden, Germany*
[2] *Center for Systems Biology Dresden, Pfotenhauerstr. 108, 01307 Dresden, Germany*
[3] *Chan Zuckerberg Biohub, 499 Illinois St, San Francisco, CA 94158, USA*

11 minutes using CLIJ. On the workstation processing took 41 minutes using ImageJ and 5 minutes using CLIJ. Thus, these results show that using a consumer laptop, CLIJ enables speedups by a factor of 15. Compared to the laptop CPU, execution on the workstation GPU was 33 times faster. Furthermore, excluding compilation time and file input/output time from the time measurement suggests that real-time image analysis becomes feasible: In a smart microscopy software application, where image data arrives in memory continuously directly from the acquiring camera and GPU code recompilation is not necessary, an estimation of cell count can be made from an image stack in less than 0.5 seconds using the presented CLIJ-based workflow.

Key feature of CLIJ is that it does not require any GPU programming skills, or specialized hardware to be executed. As it is based on the established OpenCL framework, it is not limited to CUDA-compatible GPU devices. The user can assemble CLIJ operations to GPU-accelerated image processing workflows in all programming languages available in Fiji (ImageJ Macro, ImageJ-Ops, BeanShell, JavaScript, Jython, Groovy, and Java). Users can start using CLIJ by simply modifying example code (Suppl. Code 1). Moreover, CLIJ operations can be recorded using ImageJ's macro recorder and further modified in Fiji's script editor where we added CLIJ-specific auto-completion functions and online help (Suppl. Figure 4). CLIJ can be used in the cloud or on computing clusters via ImageJ Jupyter notebooks or command-line interface. We tested CLIJ successfully on GPU-hardware from major vendors (Intel, AMD, Nvidia) and operating systems (Windows, MacOS, Linux). Finally, we facilitate providing additional GPU-accelerated operations to be used within the ImageJ ecosystem and extending CLIJ. Specifically, developers can deploy custom OpenCL code using the modern ImageJ2 plugin mechanism[10] in order to add functionality to CLIJ. For potential CLIJ developers we provide a plugin template together with the full open source code of CLIJ at: https://clij.github.io/

In summary, CLIJ makes it possible to speed up image processing workflows in Fiji to reduce processing time from hours to minutes. Furthermore, CLIJ allows general purpose real-time image processing, e.g. for smart microscopy applications. In order to facilitate adoption of this enabling technology, we have put special emphasis on documentation, code examples, interoperability, accessibility and user convenience. Therefore, CLIJ enables a wide range of imaging scientists with beginner-level programming experience to benefit from GPU-acceleration.

References

1. Preibisch, S. *et al.* Efficient bayesian-based multiview deconvolution. *Nature Methods* **11** (2014).
2. Laine, R. F. *et al.* NanoJ: a high-performance open-source super-resolution microscopy toolbox. *Journal of Physics D: Applied Physics* **52,** 163001 (2019).
3. Culley, S. *et al.* Quantitative mapping and minimization of super-resolution optical imaging artifacts. *Nature Methods* **15,** 263 EP (2018).
4. Weigert, M. *et al.* Content-aware image restoration: pushing the limits of fluorescence microscopy. *Nature Methods* **15,** 1090–1097 (2018).
5. Falk, T. *et al.* U-Net: deep learning for cell counting, detection, and morphometry. *Nature Methods* **16,** 67–70 (2019).
6. Schmid, B. *et al.* 3Dscript: animating 3D/4D microscopy data using a natural-language-based syntax. *Nature Methods* **16,** 278–280 (2019).
7. Schneider, C. A., Rasband, W. S. & Eliceiri, K. W. NIH Image to ImageJ: 25 years of image analysis. *Nature Methods* **9,** 671 (2012).
8. Schindelin, J. *et al.* Fiji: an open-source platform for biological-image analysis. *Nature Methods* **9** (2012).
9. *The Khronos Group. The open standard for parallel programming of heterogeneous systems. https://www.khronos.org/opencl/. Accessed 2018-12-09.*
10. Rueden, C. T. *et al.* ImageJ2: ImageJ for the next generation of scientific image data. *BMC Bioinformatics* **18,** 529 (2017).

Acknowledgements We would like to thank everybody who helped developing and testing CLIJ. In particular thanks goes to Bruno C. Vellutini (MPI-CBG), Curtis Rueden (UW-Madison LOCI), Damir Krunic (DKFZ), Daniel J. White (GE), Gaby G. Martins (IGC), Siân Culley (LMCB MRC), Giovanni Cardone (MPI Biochem), Jan Brocher (Biovoxxel), Johannes Girstmair (MPI-CBG), Juergen Gluch (Fraunhofer IKTS), Kota Miura, Laurent Thomas (Acquifer), Nico Stuurman (UCSF), Pradeep Rajasekhar (Monash University), Tobias Pietzsch (MPI-CBG), Wilsom Adams (VU Biophotonics). This work was supported by the German Federal Ministry of Research and Education (BMBF) under the code 031L0044 (Sysbio II) and the German Research Foundation (DFG) under the code JU3110/1-1.

Competing Interests The authors declare that they have no competing financial interests.

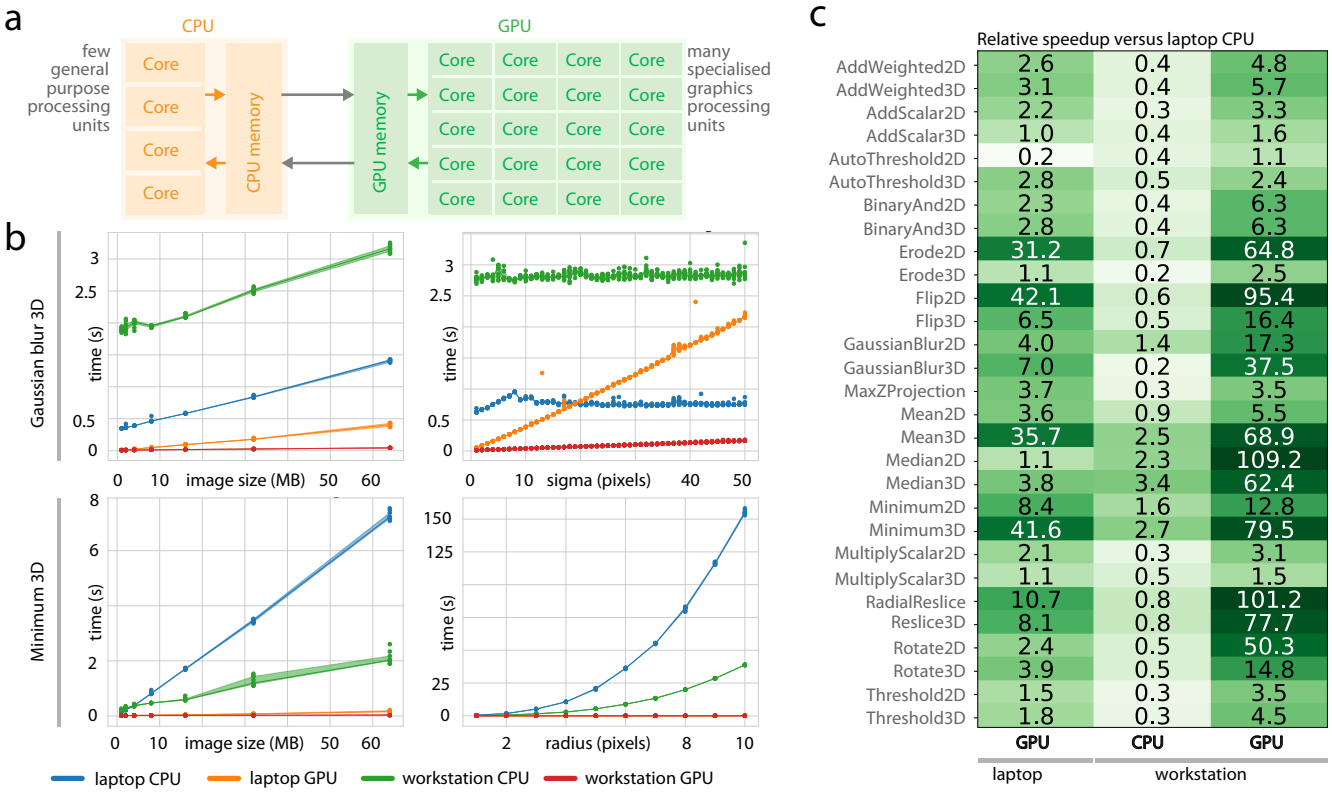

Figure 1: **(a)** GPU-acceleration schematically shows how many GPU cores potentially outperform a CPU with less cores. **(b)** Execution time of the Gaussian blur and minimum filter for different image sizes and parameters. **(c)** Overview of speedup measurements when applied to 16 MB large 8-bit images with respect to computation times on a laptop CPU.