

# GPU-accelerated image processing in ImageJ macro

Robert Haase<sup>12</sup>, Loic A. Royer<sup>3</sup>, Deborah Schmidt<sup>12</sup>, Peter Steinbach<sup>12</sup>, Alexandr Dibrov<sup>12</sup>, Uwe Schmidt<sup>12</sup>, Martin Weigert<sup>12</sup>,

Florian Jug<sup>12</sup>, Eugene W. Myers<sup>12</sup>



1 Max Planck Institute for Molecular Cell Biology and Genetics Dresden, 2 Center for Systems Biology Dresden, 3 Chan-Zuckerberg Biohub, San Francisco

### Introduction

Modern graphics processing units (GPU) enable image processing at unprecedented speed. We present CLIJ, an ImageJ[1]/Fiji[2] plugin that brings GPU-accelerated image processing to the ImageJ macro language. It is based on the Open Computing Language (OpenCL)[3], enabled through a multibackend Java facade named ClearCL[4].

# Benchmarking

To demonstrate the benefits, we benchmarked a common workflow on selected CPUs and GPUs to calculate the speedup compared to the consumer notebook CPU Intel Core i7-8650U. Comparison of speedup and price shows that huge speedup is possible without buying expensive GPUs.



CLIJ currently supports common filtering tasks that can be used to replace ImageJs built-in functionality. Furthermore, we provide macro functions that seamlessly transfer images to and from the GPU for processing. With this approach, users can exploit GPU acceleration without having to learn OpenCLbased programming.



### **Overview**

CLIJ currently offers 102 methods ranging from basic filtering, convolution, deconvolution, spatial transforms, projections, thresholding, local maximum detection, binary image processing and basic measurements. In order to exploit GPU-acceleration optimally it is recommended to implement whole

whole workflows in CLIJ and thereby minimize image transfer between ImageJ and GPU. Furthermore, iterative procedures are suited well because the first execution of CLIJ operations is always slower compared to all following executions.

# ImageJ macro

// add a constant to an image run("Add...", "value=" + value + " stack");

// apply Gaussian blur to image run("Gaussian Blur 3D...", "x=" + sigma + " y=" + sigma + " z=" + sigma); // crop a region

makeRectangle(32, 32, 128, 128); run("Duplicate...", "duplicate range=1-64");

### // initialize GPU gpuName = "Intel UHD"; run("CLIJ Macro Extensions", "cl device=" + gpuName); // push the current image to the GPU input = getTitle(); Ext.CLIJ\_push(input); // define a name for the output image output = "outputImage";

CLIJ-enriched macro

Save

results

// add a constant to an image Ext.CLIJ addImageAndScalar(input, output, value);

// apply Gaussian blur to image Ext.CLIJ\_blur3DFast(input, output, sigma, sigma, sigma);

// crop a region Ext.CLIJ crop3D(input, output, 32, 32, 0, 128, 128, 64);

# Java

### // initialize the GPU String gpuName = "Vega";

CLIJ clij = CLIJ.getInstance(gpuName); // push an image to the GPU and give me handle ClearCLBuffer input = clij.push(imagePlus);

// create an image as large as input ClearCLBuffer output = clij.create(input);

// add a constant to an image clij.op().addImageAndScalar(input, output, value);

// apply Gaussian blur to image clij.op().blurFast(input, output, sigma, sigma, sigma);

### // crop a region

output = clij.create(new long[]{128,128, 64}, input.getNativeType()); clij.op().crop(input, output, 32, 32, 0);

// apply affine transform using ImageScience plugin run("TransformJ Affine", "matrix=transform.mat" + "interpolation=Linear background=0.0 adjust");

// automatic thresholding setOption("BlackBackground", true); setAutoThreshold("Default dark stack"); run("Convert to Mask", "method=Default" + "background=Dark black");

// apply a fixed a threshold setThreshold(threshold, 65535); setOption("BlackBackground", true); run("Convert to Mask", "method=Default" + "background=Light black");

CLU\_affineTransform

🗊 CLIJ_affineTransform 🛛 🕹	B CIII reference I clii-docs X + − □ X	- □ ×
CL Device Intel(R) UHD Graphics 620	$\leftarrow \rightarrow C  \text{a https://clij.github.io/clij-docs/reference} \qquad \qquad \bigstar  \textcircled{0} :$	<u>File Edit Language Templates Run Tools Tabs</u>
source blobs-1.gif 💌	CLIJ_automaticThreshold	<pre>1 // Load sample image 2 run("Blobs (25K)"); 3 inputImage = getTitle();</pre>
Applies an affine transform to an image. Individual transforms must be separated by spaces. Supported transforms: center: translate the coordinate origin to the center of the image -center: translate the coordinate origin back to the initial origin rotate=[angle]: rotate in X/Y plane (around Z-axis) by the given angle in degrees rotateX=[angle]: rotate in Y/Z plane (around X-axis) by the given angle in degrees rotateY=[angle]: rotate in X/Z plane (around Y-axis) by the given angle in degrees rotateY=[angle]: rotate in X/Z plane (around Y-axis) by the given angle in degrees rotateZ=[angle]: rotate in X/Y plane (around Z-axis) by the given angle in degrees scale=[factor]: isotropic scaling according to given zoom factor scaleX=[factor]: scaling along X-axis according to given zoom factor scaleZ=[factor]: scaling along Y-axis according to given zoom factor scaleZ=[factor]: scaling along Z-axis by distance given in pixels translateX=[distance]: translate along X-axis by distance given in pixels translateZ=[distance]: translate along X-axis by distance given in pixels	The automatic thresholder utilizes the threshold methods from ImageJ on a histogram determined on the GPU to create binary images as similar as possible to ImageJ 'Apply Threshold' method. Enter one of these methods in the method text field: [Default, Huang, Intermodes, IsoData, IJ_IsoData, Li, MaxEntropy, Mean, MinError, Minimum, Moments, Otsu, Percentile, RenyiEntropy, Shanbhag, Triangle, Yen] Parameters: Image input, Image destination, String method Available for: 2D, 3D Macro example:	<pre>4 outputImage = getTitle() + "_thresholded"; 5 6 // initialize built-in Intel HD GPU 7 run("CLIJ Macro Extensions", "cl_device=[Intel(R) UHD Graphics 620]"); 8 Ext. CLIJ_push(inputImage); 9 Ext. CLIJ_automaticThreshold(inputImage, outputImage, "Otsu"); 10 Ext. CLIJ_al 11 Ext. CLIJ_a 11 Ext. CLIJ_addImageAndScalar(Image source, Image destination); 12 Ext. CLIJ_addImages(Image summand1, Image summand2, Ext. CLIJ_addImagesWeighted(Image source, Image destination Ext. CLIJ_argMaximumZProjection(Image source, Image destination Ext. CLIJ_automaticThreshold(Image input, Image destination Available for: 2D, 3D</pre>
Example transform: ransform = "center scale=2 rotate=45 -center"; Parameters: Image source, Image destination, String transform Available for: 2D, 3D	<pre>run("CLIJ Macro Extensions", "cl_device="); Ext.CLIJ_push(input); Ext.CLIJ_automaticThreshold(input, destination, method); Ext.CLIJ_pull(destination);</pre>	Run Batch   Started Newiji   Started Newiji   Started Newiji   Started Newiji

// apply affine transform transform = "rotate=45"; // degrees transform = transform + " scaleX=2"; // zoom factor Ext.CLIJ affineTransform(input, output, transform);

// automatic thresholding Ext.CLIJ\_automaticThreshold(input, output, "Default");

// apply a fixed a threshold Ext.CLIJ\_thresholdIJ(input, output, threshold);

// get output image back and show it Ext.CLIJ\_pull(output); // empty GPU memory Ext.CLIJ\_clear();

#### // apply affine transform

AffineTransform3D transform = new AffineTransform3D(); transform.rotate(2, 45); transform.scale(2.0, 1.0, 1.0); clij.op().affineTransform(input, output, transform);

// automatic thresholding clij.op().automaticThreshold(input, output, "Default");

// apply a fixed a threshold clij.op().threshold(input, output, threshold);

// get output image back and show it clij.pull(output).show(); // empty GPU memory input.close(); output.close();

Dilation	Maximum filter	



# **Project goals for widespread availability**

In order to make CLIJ available for everyone, goals of the CLIJ project were

- **Usability:** The ImageJ user is able to GPU-accelerate his workflows directly from known programming interfaces such as ImageJ macro.
- **Documentation:** CLIJs reference guide documents all CLIJ operations and is available in CLIJ plugin dialogs, the web and auto-completion help.
- **Interoperability:** CLIJ functionality can be used from ImageJ macro, from ImageJs Jython and Groovy interfaces as well was from Java.
- Extensibility: OpenCL developers can extend CLIJ with basic knowledge about Java and the SciJava/ImageJ2 eco system.

# **Differences to ImageJ**

When applying filters such as `Dilate' or `Maximum' to 2D and 3D images in ImageJ/Fiji, differences in interpretation of neighborhoods become obvious. CLIJ allows the user to explicitly choose the desired neighborhood.

# **Conclusions & Outlook**

Using CLIJ it becomes feasible to speed up simple workflows by a factor of 10 on any computer which has a built-in Intel HD GPU. Higher performance can be achieved with sophisticated workflows on high-end GPU hardware. More detailed benchmarks comparing various hardware and operations follow soon.

[1] Schneider, C. A.; Rasband, W. S. & Eliceiri, K. W. (2012), NIH Image to ImageJ: 25 years of image analysis, Nature methods 9(7): 671-675

[2] Schindelin, J.; Arganda-Carreras, I. & Frise, E. et al. (2012), Fiji: an open-source platform for biological-image analysis, Nature methods 9(7): 676-682

[3] The Khronos Group, The open standard for parallel programming of heterogeneous systems, https://www.khronos.org/opencl/ accessed 2018-12-09

[4] Royer, L.A., Weigert, M., ClearCL - Multi-backend Java Object Oriented Facade API for OpenCL https://github.com/ClearControl/clearcl accessed 2018-12-09

