GPU-accelerated image processing in ImageJ macro

Robert Haase[12], Loic A. Royer[3], Deborah Schmidt[12], Peter Steinbach[12], Alexandr Dibrov[12], Uwe Schmidt[12], Martin Weigert[12], Florian Jug[12], Eugene W. Myers[12]

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

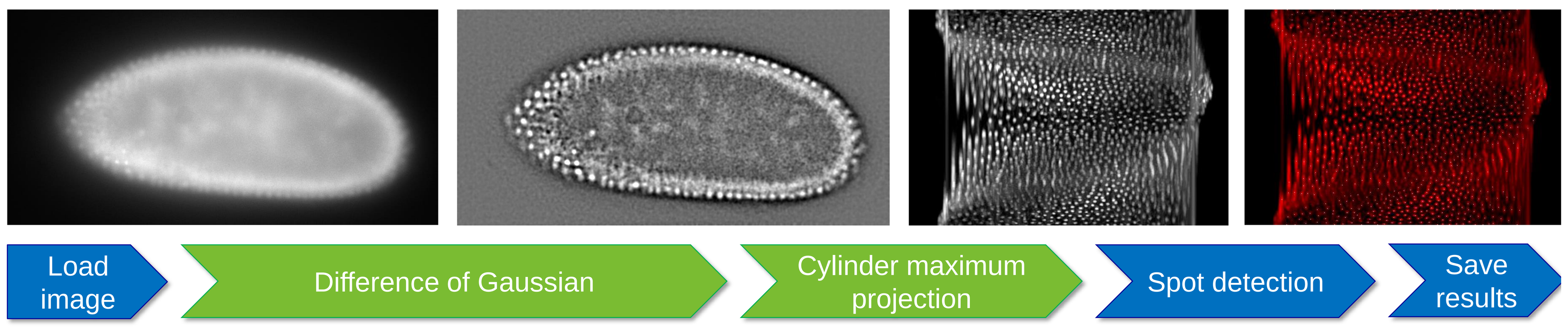
Introduction

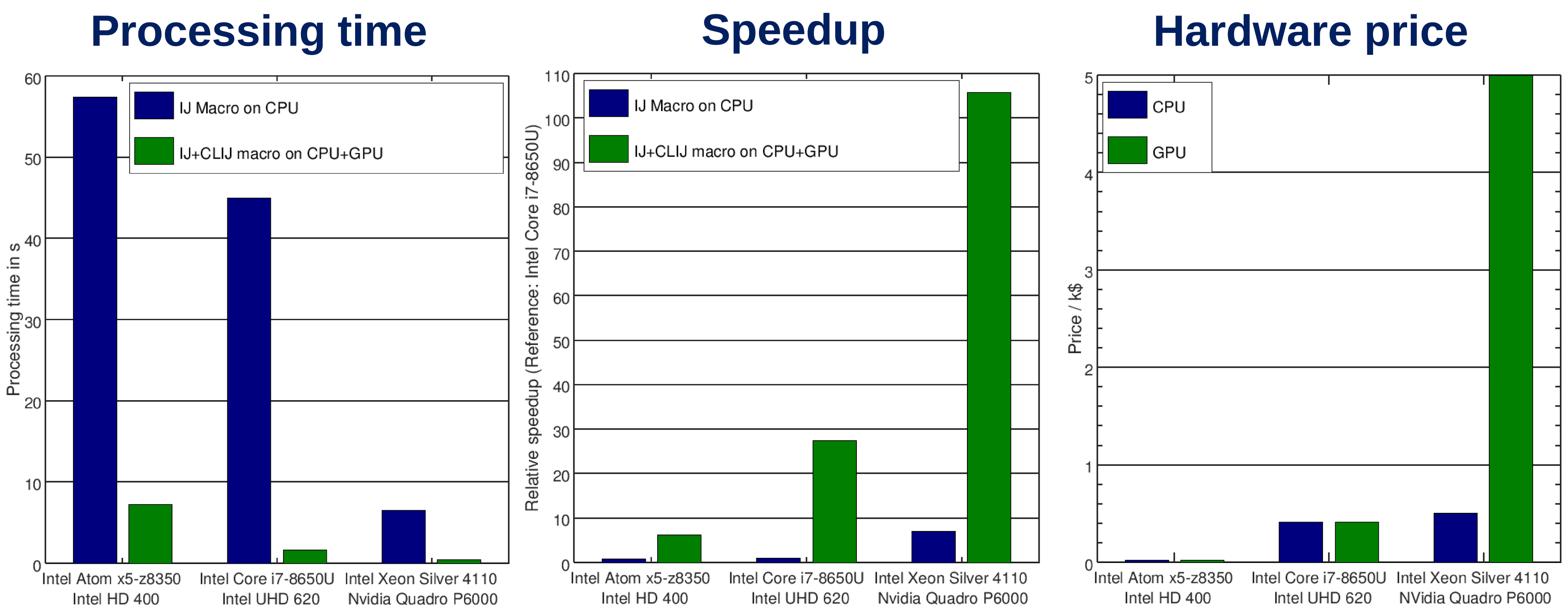
Modern graphics processing units (GPU) enable image processing at unprecedented speed. We present CLIJ, an ImageJ[1]/Fiji[2] plugin that brings GPU-accelerated image processing to the ImageJ macro language. It is based on the Open Computing Language (OpenCL)[3], enabled through a multi-backend Java facade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

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 OpenCL-based programming.

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.





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

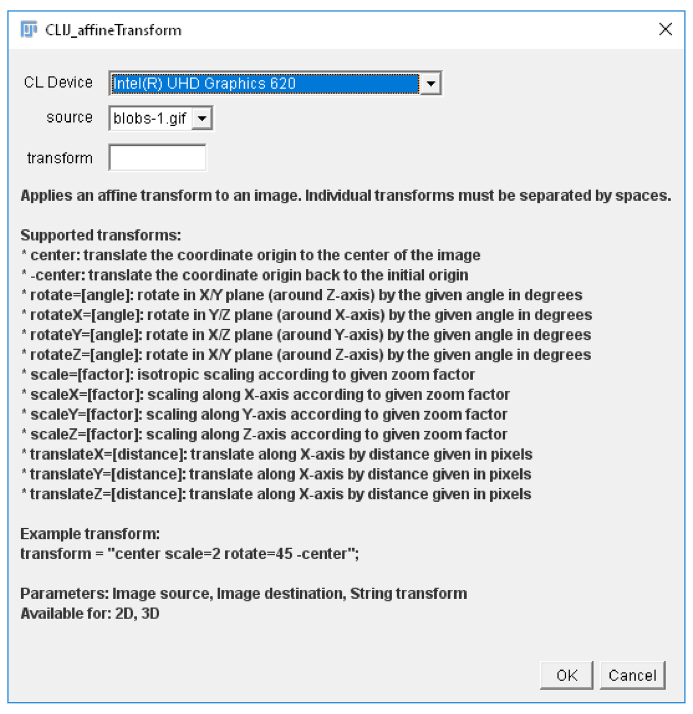
```
// add a constant to an image
run("Add...", "value=" + value + " stack");

// 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");

// 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");
```

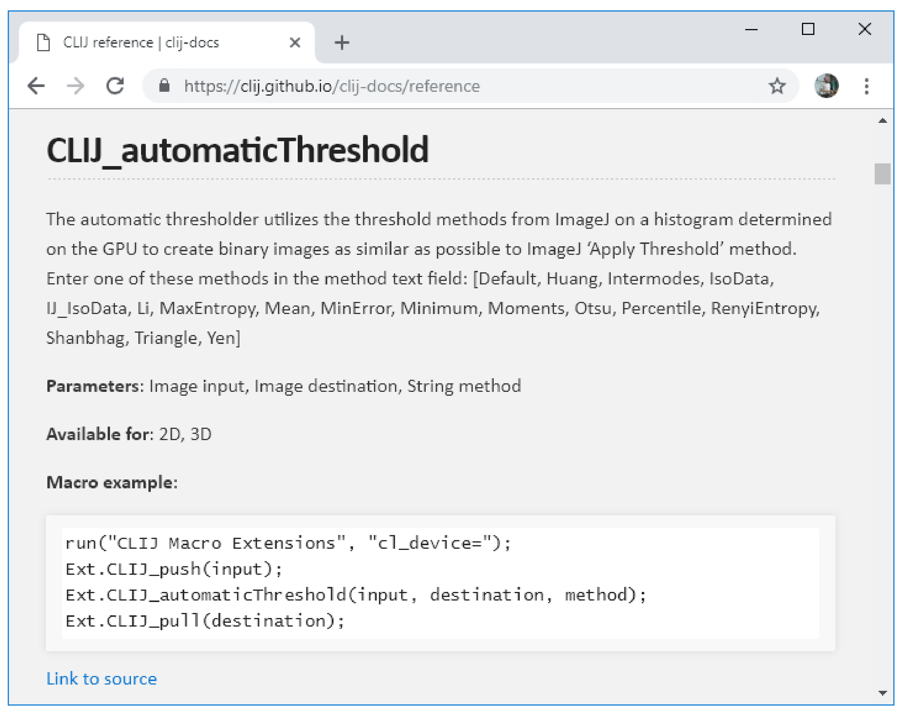
CLIJ-enriched macro

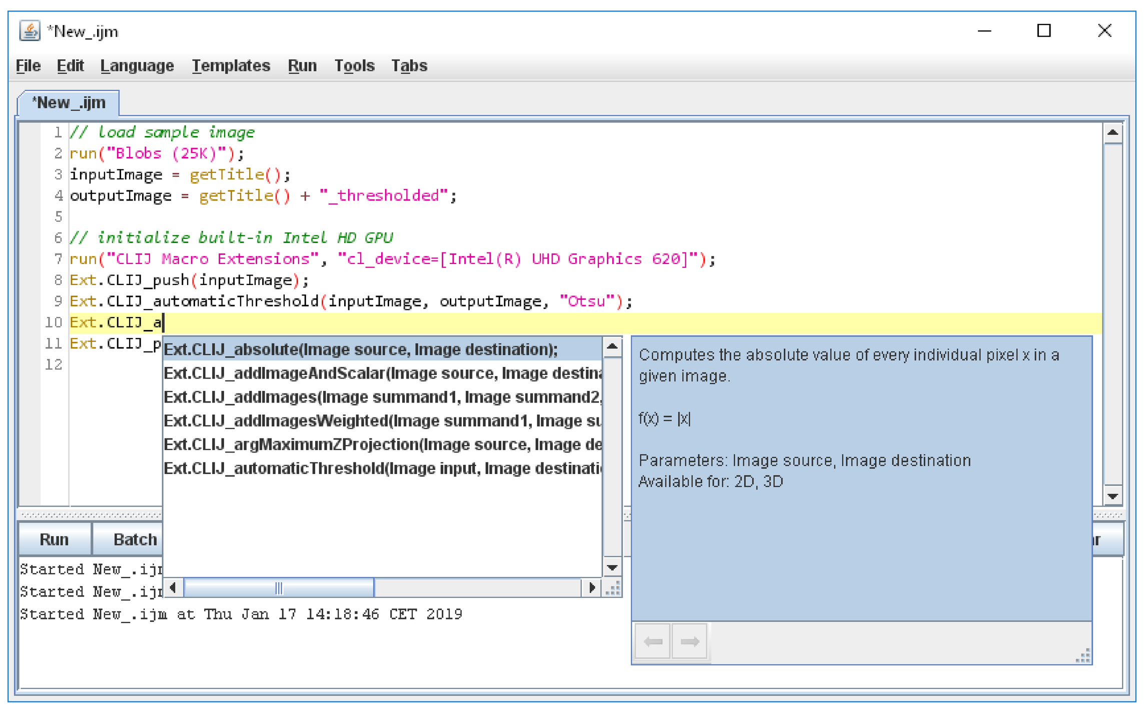
```
// initialize GPU
gpuName = "Intel UHD";
run("CLIJ Macro Extensions", "cl_device=" + gpuName);
// push the current image to the GPU
input = getTitle();
Ext.CLIJ_push(input);
// define a name for the output image
output = "outputImage";

// 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);

// 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();
```

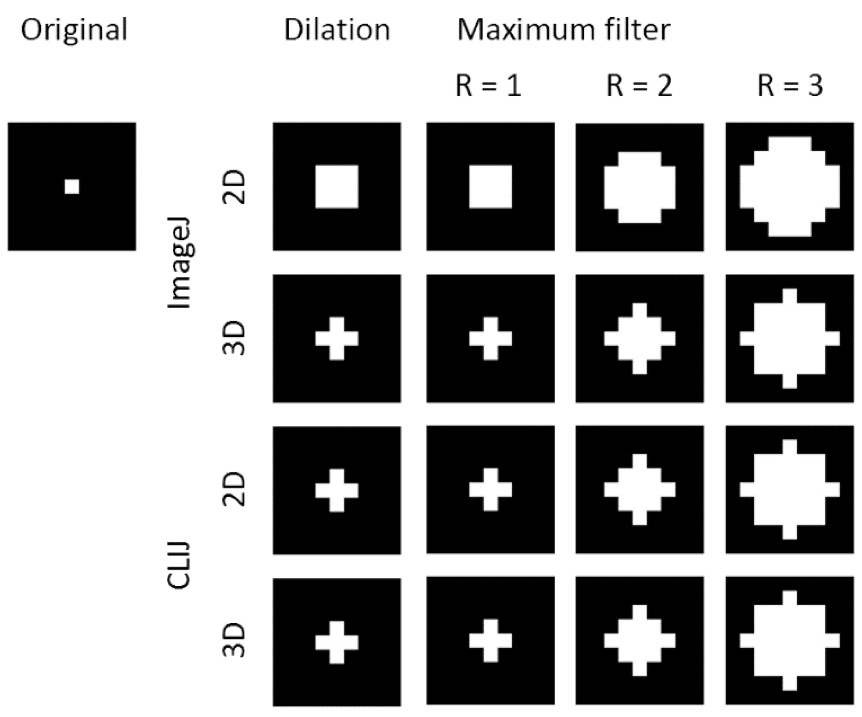
Java

```
// 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
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();
```









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

https://clij.github.io/clij-docs/

Bundesministerium für Bildung und Forschung