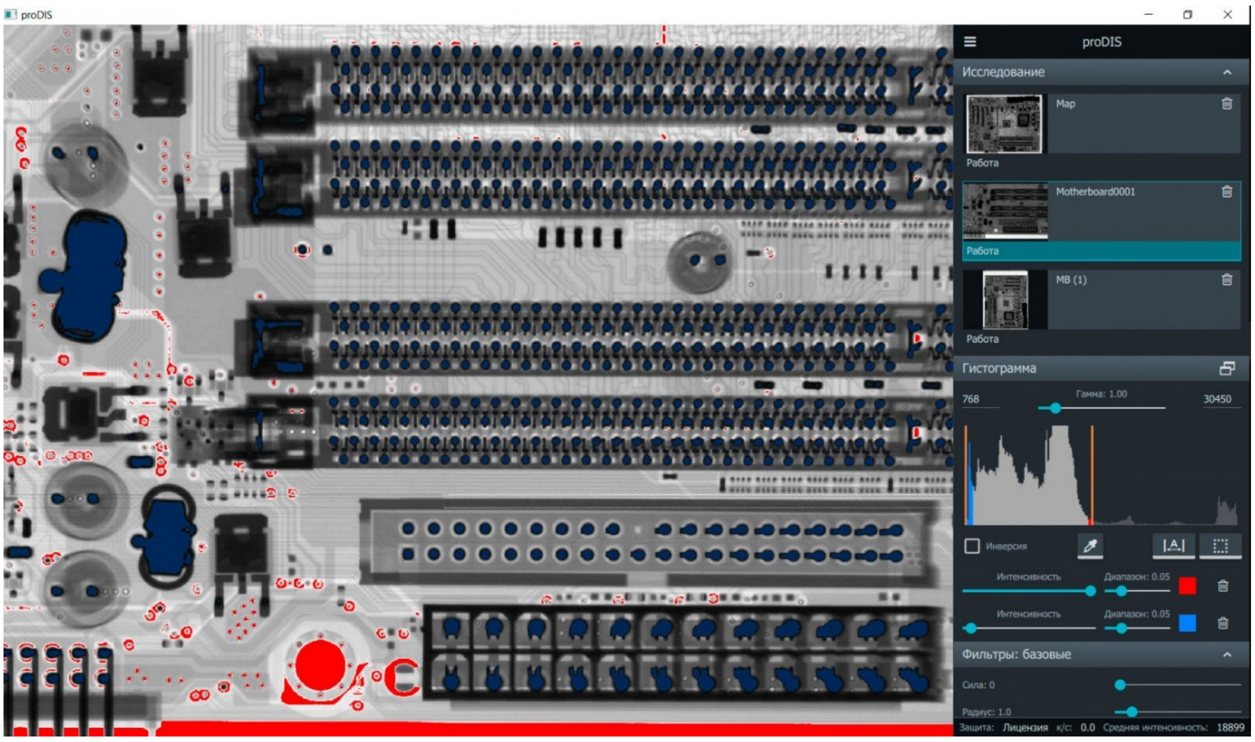
Digital X-ray: walking along Elbrus

Hello, Habr! Last time I wrote about the project on X-ray inspection of PCB. We've made great progress so far, now we have a working prototype of software, moreover, we've taken a closer look at Elbrus processor platform. This experience I would like to tell you about.



Intro

At the start of the project we managed to find additional funding a fundamental term of which was universal cross platform software, it also had to support Russian processors. At that point Elbrus 8C was the most powerful microprocessor for desktop machines (and remains so far, 8CB doesn't seem to be available yet). We bought 2 stations «Elbrus 801-PC» directly from MCST. Their current price is indicated on the site, a year ago they were a bit more expensive.

When purchasing we faced some amusing things, namely, supply request sheets. I suggest that you conclude an NDA at once, it will give you access to up-to-date developer tools (it'll take you about a month to complete the paperwork). The machines arrived quickly, as for the list of equipment – MCST has some problems with customization. It's easier for you to buy the necessary video cards or hardware peripherals and install yourself. The list of proven equipment/video cards you can only find in MCST bug tracker though the list should be published on wiki resources concerning Elbrus.

Operating systems

At the beginning of 2020 you could choose a pre-installed operating system while ordering: the original Elbrus one or the partner one (Alt, Astra). We use Astra Linux in our project. Despite it was officially available, in fact we didn't manage to make an order fast enough. We installed it from live ISO image ourselves by a separate license.

Choosing this or that Operating System – it's just a matter of taste, I like Fly's graphical desktop environment in Astra, subjectively it is faster. On the other hand, Alt software repository is larger, they have closer cooperation with MCST and more interesting set of distributions.

For example, there was no USB live image of Astra for Elbrus – but there were several ones for Alt. Astra backup section worked only under Alt live image.

Elbrus operating system itself was initially developed for the customers' demo. We started from the version 4.0, now 5.0rc2 is the working one – it's better but it's still rather "raw" for the end user. We need a system for adjusting or getting the most out of VLIW architecture. The performance at image processing was maximum on this OS exactly.

UPD: a new version of the OS, Elbrus 6.0, has just been released,. A new Linux kernel and C++20 have been claimed but as we work on Astra, we haven't got update yet.

Architecture

Only C++14 is available at the moment, CUDA and Vulkan are better not to be thought about while OpenGL on AMD video cards performs well. An important point – the version of OpenGL should not be earlier than 3.1, QT 5.11.

As for the rest of programming languages – perhaps, you'll share your experience in comments. I know that some «closed» companies are working in this direction, mainly, on video stream and image processing. Russian processors are being operationalized, step by step.

Now let's give the floor to the project architect **Maxim Titov** (titovmaxim, Unicore Solutions). Our aim is to make good software for work with X-ray systems. So we've started with the most important thing – connection of hardware and stack of image processing. We were very anxious. TLDR: everything went smoothly.

The principal part of the system is an X-ray detector. It is connected by Ethernet and loads 1Gbit bandwidth to the eyeballs, GigE Vision protocol. There are no commercial libraries for Elbrus, libraries with open source code (Aravis, for instance) are not suitable in speed and quality so we've written our own implementation.

We faced the situation when the built-in network card of Elbrus motherboard heavily loads the processor and it's not that easy to get 1Gbit, frames get lost. We had to write an extra fast intermediate buffer to unload the image streaming. Then we processed everything in separate threads. Everything started working, no free bandwidth, no lost frames. You'll have much less problems installing a discrete card onto Elbrus.

Real-time image pre-processing has been implemented on OpenGL as it's the only cross-platform application programming interface we have. It includes calibration measurement and correction of the detector's dead pixels, histogram processing, gamma correction, sharpening and noise reduction, halftone coloring, geometric transformations. It was an interesting task to write rapid image histogram evaluation on OpenGL, not on CUDA, not that typical.

It takes Elbrus ~33 ms in total to process one 3000x3000 16 bit frame on the weakest video card (AMD R5). Very well indeed upon condition of independence from the operating system and full CPU offload for tasks of complex post-processing. More powerful video cards are already available so we have nothing to worry over this issue. For example, on x86 with GeForce RTX 2070 Max-Q we get standard ~2ms and are expecting the same index from Elbrus.

Connection of other components – X-ray generators and mechatronics via RS232, Web cameras via UVC went like clockwork.

In our project we use Qt 5.11 and QML for the interface. All of a sudden everything was seamless here. It works, starts up "OOB", Astra has most of the packages. We are waiting for update to be available as 5.11 has some «discrepancies» and bugs.

A couple of words concerning our fear of C++ 14. Active joint use of C++ and QML often causes troubles so we decided to use our Flow library in the project.

Flow library

The library's advantages are declarativeness, boilerplate code and error reduction. It helps to use functional approach in C++, in particular, there's flexible grouping and composition of functions, lazy initialization, caching, thread safety and background implementation in other threads. By the way, the last point is true of OpenGL that doesn't support multithreading well. All of this is in dynamics with change notifications (no update) on thread detaching/attaching, actions (Functional Approach effects) are integrated with Qt contexts. Good news – automatic control of objects' lifetime and subscriptions with no need to subscribe/unsubscribe manually and concerns for lifetime of this or that, all by itself:) It resembles ReactiveX a bit but it's a matter of state rather than data stream.

It also has its own metasystem (no reflection is C++), we use it instead of QMetaObject. Less writing, better integration with QML (almost like WPF with C#), you can move about the tree freely, particularly, work from QML with QVector in the middle of the tree like with a model with smart diff (without writing QAbstractItemModel), automatic serialization/deserialization of any object in a single command etc.

The library was initially designed for C++ 17 at least. Naturally, we lost type interference when switching to C++ 14 and now we have to write most template parameters manually. However, MCST progress is impressive, we are waiting for new compilers. Elbrus compiler had a couple of peculiarities not previously seen in GCC and MSVC. Auto parameters in lambdas are not always understandable. It can't recapture this into nested lambdas. But it can be easily fixed, in other respects everything is compiled and works flawlessly. Yes, compiler errors in Russian are a bit unusual;)

Due to the peculiarities of Elbrus architecture using exceptions is not recommended. So we immediately rejected them in loaded parts. But we use them widely on the upper level where operations are single. Everything works as expected, our fears were unfounded. There is no construction «fnon-call-exceptions» for easy system-level debugging and processing yet.

It's great that in most cases we can write code "at home", on x86 Linux and then just build it on a remote Elbrus machine. With some skill, there are almost no problems with code compatibility.

OpenCV Perfomance

The most critical things in video stream processing are done on the video card, for less time-consuming post-processing of images we use OpenCV 3.2. This package has been ported to Elbrus, but there is one problem – the performance depends heavily on the version of the package for a particular OS. See the comparison table of the performance of OpenCV packages on Elbrus 8C (1300 MHz) and Intel core i7 (2600 MHz) under different OS / builds of OpenCV:

OpenCV comparison table: Elbrus vs Intel i7

Operation execution time, ms		OS Elbrus 5.0rc2/ Elbrus-8C/ OpenCV 3.2	OS Astra Leningrad 8.1/ Elbrus-8C/ OpenCV 3.2	OS Astra Smolensk 1.6/ Intel Core i7 7700/ OpenCV 3.2	OS Windows 10/ Intel Core i7 9750H/ OpenCV 3.2	OS Windows 10/ Intel Core i7 9750H/ OpenCV 4.4
Convolution	kernel 5x5, 3000x3000, 16S	35	334	99,7	94	105,9
Convolution	kernel 5x5, 3000x3000, 16U	244	280	-	98	106,5
Convolution	kernel 5x5, 3000x3000, 32F	32	271	23,9	24	11,4
Gaussian blur	kernel 5x5, 3000x3000, 16S	15,3	257	36,3	35	5,7
Gaussian blur	kernel 5x5, 3000x3000, 16U	184	251	-	12,5	40
Gaussian blur	kernel 5x5, 3000x3000, 32F	14,5	222	8,1	7,7	6,2

The performance of OpenCV on Elbrus directly depends on low-level EML libraries (see the MCST programming manual, they are optimized for the VLIW architecture), and EML packages depend on the OS distribution. In the Astra we use a fresh build is not available yet, perhaps Alt Linux has it. In case you had such an experience - write in the comments.

If we talk about a routine operation - image convolution, then the performance can be twice as good (16S) and can be twice as bad (32F) compared to i7. For processing of image convolution in OS build with an unoptimized OpenCV library the performance loss is up to 20 times. Yes, 16U convolution at Elbrus is still bad.

Summary

We have learned to live with Elbrus. Gradually MCST and partners (Alt, Astra) are improving the software, improving the service and reducing prices. This makes me happy. On the other hand, their resources are limited, you can wait for six months for something you need.

Perhaps the government will be more active in promoting Russian processors and stimulating their commercial exploitation, we'll see. In any case we are ready for this.