Programming Framework for a Multi-Accelerator Multi-Core High-Performance Platform

Laurent Segers, Bart Spiers, An Braeken, Bruno Da Silva, Erik H. D'Hollander, Jan Lemeire, Abdellah Touhafi, Jan G. Cornelis

Erasmus University College, Brussels,
Vrije Universiteit Brussel, Ghent University

Introduction

- "GUDI" is a combined GPU/FPGA desktop for accelerating image processing applications. A GUI is developed to demonstrate the functionality and performance of this system. Image processing algorithms are selectively executed on the CPU or one of the accelerators.
- A modular design approach is proposed resulting in several plugins which can be used in Qt as well as in GIMP.

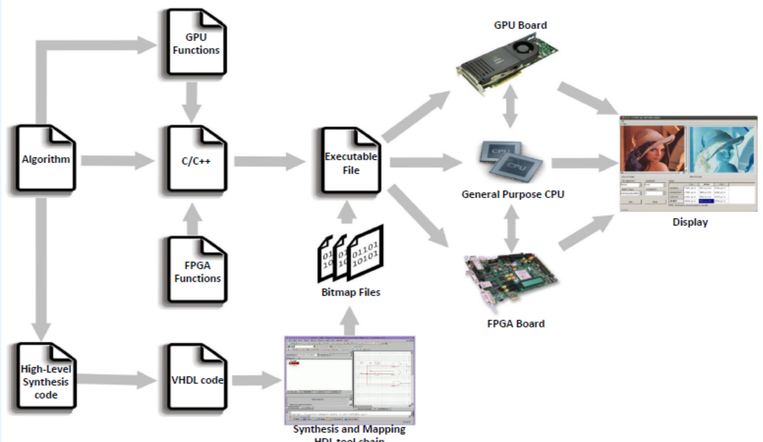

***Figure 1**. GUDI Tool chain. An algorithm is split in several parts which are executed on different technologies in order to speed up its final performance.*

Objectives

- Transparent remote procedure call to accelerators in a heterogeneous computer system.
- Developing a common front-end to a multi-core, multi-accelerator, multi-technology platform.
- Separate platform-independent GUI design from platform-dependent code execution.

Demonstrator

- The GUI is developped with Qt-creator* showing the images before and after processing in a userfriendly way.
- The application processes images using different algorithms, such as erosion, dilation, edge detection,...
- User can select the appropriate algorithm and platform on which to perform the image processing.
- Performance analysis of the image processing algorithms on the different platforms is shown.

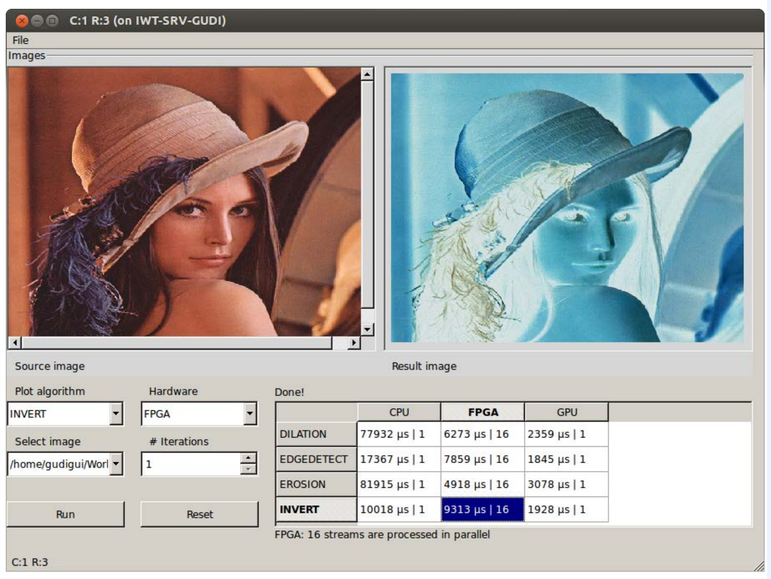

***Figure 2.** GUI of the application: the left image represents the original image, while the output is shown in the right image. One can choose to process an image with a given algorithm on a selected platform by clicking on the corresponding cell in the grid.*

**Qt: Integrated Development Editor, http://qt.digia.com*

Heterogeneous platform

The image processing algorithms run on different hardware platforms:

- **CPU:** The baseline algorithms are executed on the host CPU.
- **GPU:** The algorithms are adapted and recompiled to run on the GPU platform using the OpenCL API. The GPU is then programmed and the data is sent to the GPU GDDR5 memory. The GPU stores the data back into the GDDR5 memory, from which the data can be collected by the host.

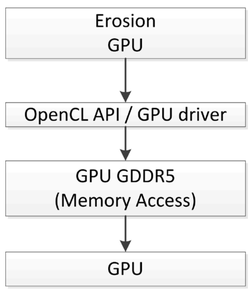

***Figure 3**. Dataflow on the GPU.*

- **FPGA:** The algorithms are rewritten using High-Level Synthesis tools for the FPGA platform. The communication with the FPGA is managed through the Pico-API. The data can be sent and received using streams or using the DDR3 memory.

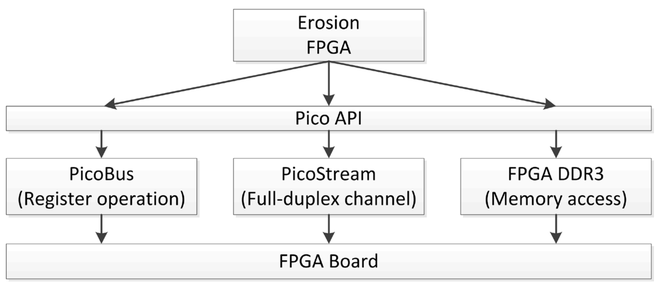

***Figure 4**.Dataflow on the FPGA.*

Supporting multiple platforms increases the compiling complexity and reduces software reusability. Modular programming techniques help overcoming these problems.

Modular concept

Each image processing algorithm on a particular platform (CPU,GPU,FPGA) is considered as one module. Consequently, for one algorithm, there are 3 different modules (plugins).

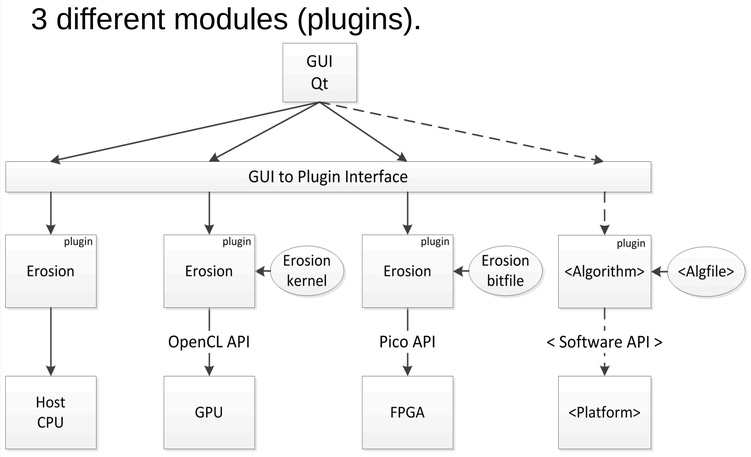

***Figure 5.** Depending on the platform on which the processing algorithm is run, the plugin interfaces the platform using the appropriate API libraries. The plugin also loads the accelerator code "Algfile" (Algorithm-file: Erosion kernel for the GPU or Erosion bitmap file for the FPGA) onto the corresponding platform.*

Acknowledgements

This research has been made possible thanks to a Tetra grant 100132 "A combined GP-GPU/FPGA desktop system for accelerating image processing applications (GUDI)" of the Flanders agency for Innovation by Science and Technology.

Plugin

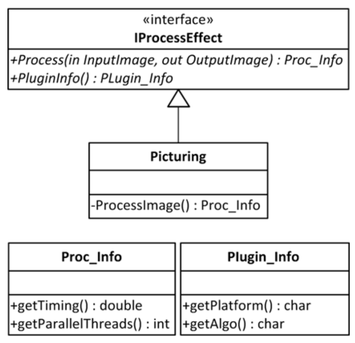

***Figure 6.** Shared library class diagram. Each library inherits the IProcessEffect interface with "Picturing". The interface is visible for both the library and the GUI-application. The Picturing implementation however is only visible within the library.*

- Plugin_Info is called when the shared library is loaded at startup
- Proc_Info is returned when the shared libary plugin finishes processing.

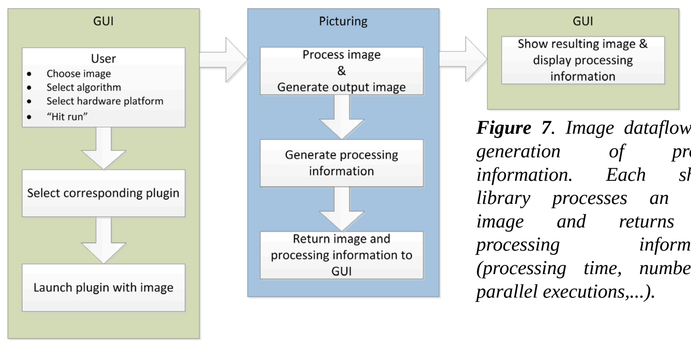

***Figure 7**. Image dataflow and generation of process information. Each shared library processes an input image and returns the processing information (processing time, number of parallel executions,...).*

- All shared libraries have the same way of interacting with the main (GUI) application, therefore they are considered as plugins

Accessing a plugin

Since the main application contains several plugins, these plugins are saved into a pool.The application loads all the plugins from a directory and stores them into the plugin pool.

```
void Run()
{
    //-- ui represents the GUI-object
    QModelIndex idx = ui->ResultTable->currentIndex();
    //-- idx represents the last selected cell in the grid
    std::string HW = HardwareList->at(idx.column());
    //-- take the name of the hardware platform
     std::string ALGO = AlgoList->at(idx.row());
    //-- take the name of the algorithm
    std::vector<IProcessEffect*>::const_iterator processItr
    = std::find_if(pluginPool.begin(), pluginPool.end(),findCorrectAlgo(HW, ALGO));
    /*- findCorrectAlgo looks if Algo and HW correspond to current plugin-in,
      if yes, return 1 so a valid iterator is returned -*/

    if (processItr != pluginPool.end())
    {
        IProcessEffect *process = *processItr;
        Proc_Info* info;
        IplImage *out; //-- openCV image object
        IplImage* selectedImage=Images[ui->ImageSelector->currentIndex()];
        //-- get the correct image with the selected index from the imagescombobox
        info=process->process(selectedImage,&out);
        Display(out,info); //-- Displays the results and the output image in the GUI
    }
}
```

***Code snippet :** When the user selects a cell in the grid (GUI), the Run() function is invoked. This functions looks for the selected cell indices (HW, ALGO) and starts a process to execute the algorithm ALGO on the image using the processing element HW.*

Conclusions and future work

- The modular approach enables the combination of several algorithms running on different hardware platforms into one application.
- A similar approach is used by GIMP*. By adapting the "GUI to Plugin Interface", the plugins can be used into the well-known application.

**GIMP: GNU Image Manipulator Program - http://www.gimp.org/*

Contact details

Laurent Segers, laurent.segers@ehb.be

Bruno Da Silva, brunotiago.da.silva.gomes@ehb.be

Presented at HiPAEC '13 in Berlin