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.

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 developed with Qt-creator* showing the images before and after processing in a user-friendly 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 different platforms is shown.

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.
- **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.

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).

Plugin

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

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.

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.

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.

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

Laurent Segers, laurent.segers@ehb.be
Bruno Da Silva, brunotago.da.silva.gomes@ehb.be


Contact details

Laurent Segers, laurent.segers@ehb.be
Bruno Da Silva, brunotago.da.silva.gomes@ehb.be

Presented at HiP2C’13 in Berlin


Figure 1. 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.

Figure 2. Algorithm on a selected platform by clicking on the corresponding cell in the grid.

Figure 3. Dataflow on the CPU.
Figure 4. Dataflow on the GPU.
Figure 5. Dataflow on the FPGA.

Figure 6. Shared library class diagram. Each library shows the Protocollfier 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.

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