Automatic GPU Code Generation for Android

Oliver Reiche, Richard Membarth, Frank Hannig, and Jürgen Teich

We present the *Heterogeneous Image Processing Acceleration* (HIPA^{cc}) framework. It allows programmers to develop image preprocessing applications while providing high productivity, flexibility, and portability as well as competitive performance. The same algorithm description serves as basis for targeting different GPU accelerators and low-level languages. Hereby, imaging algorithms can be expressed in a compact and productive way by using a domain-specific language (DSL) that is embedded into C++ code. Using the HIPA^{cc} source-to-source compiler, DSL code is compiled to CUDA, OpenCL, C/C++, or even Renderscript code, which targets heterogeneous architectures on recent MPSoCs running Android. Programming those MPSoCs can be challenging, in particular when targeting different architectures (CPU/GPU/DSP). HIPA^{cc} lifts this burden from programmers by automatically applying source code transformations based on domain knowledge and a built-in architecture model.

This demonstration accompanies the conference talk 4.6.3 "Code Generation for Embedded Heterogeneous Architectures on Android". It demonstrates the seamless integration of HIPA^{cc} into the *Android Developer Tools*. Hereby, the eclipse-based build software calls the HIPA^{cc} compiler to process image filters written in DSL code. The compiler call results in *Renderscript* and *Filterscript* source files, which are then automatically further compiled into an Android app binary. Right after the build process, the resulting app is presented on a Nexus 5 smartphone, a Nexus 10 tablet, and an *Arndale* prototyping board. The latter two are based on a Samsung Exynos 5250 MPSoC. All devices are connected to the build system and contain an embedded general purpose GPU (Qualcomm Adreno 320 and ARM Mali T-604). To emphasize the performance of the generated code, the purpose of the app is to measure its execution times on different processor types (CPU/GPU) and give a comparision to functional identical handwritten naïve versions of the same image filters. Besides performance, this demonstrator easily highlights further advantages of automatic code generation from a common DSL description for embedded GPUs, which can be interactively presented as well:

- **Productivity:** The same DSL code is used to generate code for both target languages *Renderscript* and *Filterscript*, simply by switching a compiler flag.
- **Portability:** The same DSL code can be used to generate efficient CUDA, OpenCL, and C/C++ code on desktop machines for discrete GPUs.
- **Filter Variation:** Changes in filter code like modifying mask sizes for local operators, switching boundary handling (clamp, mirror, constant, …), changing image access interpolation (nearest neighbor, linear filtering, …), or adding another stage to an image pyramid can be done by just changing a single line of DSL code.
- **Optimization Techniques:** Enabling code generation optimizations like loop unrolling, constant propagation, iteration space unrolling (compute multiple output pixels per kernel execution) can be done by setting compiler flags. Built-in exploration features can be used to easily evaluate the best optimizations for certain architectures.