Automatic Generation of OpenCL Code for ARM Architectures

Sergio Afonso*, Alejandro Acosta† and Francisco Almeida‡
Universidad de La Laguna
Email: *safonsof@ull.es, †aacostad@ull.es, ‡falmeida@ull.es

I. Introduction

Modern hand-held devices are currently adopting technologies previously only available for desktop computers. These devices now integrate a multi-core CPU, a GPU and DSPs on a chip. From the developer's point of view, the architecture is a traditional heterogeneous system where memory is shared between the CPU and accelerators. The exploitation of all of these heterogeneous processors is required in order to benefit from the computing capabilities of these devices.

Even though frameworks have been created to support software development for hand-held devices, they focus on fast application development. Tools for easily taking full advantage of the underlying hardware are required. Our goal is to provide one such tool without negatively impacting code's maintainability and portability.

In Android we have access to three programming models that we can make use of. *Java* provides a comprehensive API and it's the easiest model to program. Most Android applications are only written in Java, so all developers are familiar with it. *Renderscript* is designed for CPU-intensive tasks. It requires the developer to learn a new language. *C* allows us to access native libraries.

Paralldroid is a development framework that allows the automatic generation of C, Renderscript and OpenCL code in the Android platform. OpenCL is not officially supported on Android, but main hardware vendors support it, and through native code we can accelerate code by using it.

II. Methodology

Using Paralldroid, the developer annotates a Java class with sequential code and certain Java annotations. Paralldroid uses annotations to generate a new program that incorporates the code sections to run on the CPU or GPU, using a user-specified target language. This is achieved through Abstract Syntax Tree transformations.

A data environment is created for each @Target class, which handles communications between Java and the target environment. @Map and @Declare annotations are used in order to specify memory movement semantics. Paralldroid lets the user define data-parallel methods through the @Parallel annotation. All these annotations are applied to the main components of a class (fields, methods, parameters, ...) and not directly to the algorithm's code. This makes it easier to use and more suited to object-oriented programming than other annotation-based approaches like OpenMP or OpenACC.

The OpenCL extension to Paralldroid contains various translator classes that generate, from the same annotated Java code, various languages that interact with each other. There is a Java translator that generates a modified Java code to forward method definitions to the native context, a native translator that generates the OpenCL host code that acts as a bridge between Java and hardware accelerators, and an OpenCL translator that generates the kernel code defined in @Parallel methods. The API of the translated classes is not modified, so that the calling code can run unmodified.

III. Computational results

Benchmarks show that our generated OpenCL code runs faster that Renderscript when running on GPUs. Further optimizations can be implemented, in order to improve its performance.