ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Integrierte Systeme
Integrated Systems Laboratory

Department of Information Technology and Electrical Engineering

Spring Semester 2020

LaTeX Report Template

Semester Project / Master Project

Pierre-Hugues BLELLY
pblelly@student.ethz.ch

May 2020

Supervisors:	Matheus Cavalcante, matheusd@iis.ee.ethz.ch
	Samuel Riedel, sriedel@student.ethz.ch
Professor:	Prof. Luca Benini, lbenini@ethz.ch

Acknowledgements

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Declaration of Originality

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor. For a detailed version of the declaration of originality, please refer to Appendix **??**

Pierre-Hugues BLELLY,
Zurich, May 2020

Contents

List of Figures

List of Tables

List of Acronyms

EEES	Energy Efficient Embedded Systems
FPGA	Field Programmable Gate Array
IIS	Integrated Systems Laboratory
ISA	Instruction Set Architecture
PCMA	Programmable Many Core Accelerator
SoC	System on Chip

Chapter 1

Introduction

1.1 Heterogeneous systems

Thanks to system integration and the better energy efficiency of modern CPUs, embedded systems are becoming quite powerful. They can now manage multiple sensors in real-time. But the power consumption is still an issue. These chips often have a limited power budget as their goal is to be embedded in small systems where the battery capacity doesn't exceed the hundreds of milli Ampere hour.

To improve the energy efficiency and the computing power of those chips, researcher have been trying to find new architecture which would suit those applications. Heterogeneous computing may be one of the solutions to this problem. Heterogeneous systems are often composed of one general-purpose CPU and multiple coprocessors which will be used to handle specific tasks [?]. This strategy has been used in the SoC industry by ARM since 2011 by ARM in it's big.LITTLE architecture [1]. This architecture based on two clusters of ARM Cortex A7 and A15, was designed to increase the computing power in embedded systems such as smartphones while increasing the battery life of the device. Even though this architecture relied on a single ISA, the idea was to use more powerful cores when needed and turn them off when the device was in sleep mode or when the lower power core could handle the task.

These architectures are interesting to increase the power efficiency but can also increase the processing power at the same time. Researchers have been trying to take advantage of multiple ISA and leverage their specificities to increase the overall performances with great success. In this article [2] from 2014, researchers compared homogeneous and heterogeneous systems based on three ISA, under strict design constraint (such as area or thermal dissipation) they observed that the heterogeneous designs performed better than their homogeneous counterpart. Heterogeneous systems are also heavily used in

Data Centers, as the processing needs keep rising, they now use GPU as they are heavily optimized for parallel tasks and are now powerful enough to be used in scientific applications.

Hero [3] is a heterogeneous system developed by the Integrated Systems Laboratory (IIS) and the Energy Efficient Embedded Systems (EEES). This platform is composed of a hard multicore ARM 64 Juno SoC (composed of two Cortex A57 and four Cortex A553) and one PULP cluster (composed of eight RI5CY cores), running on an FPGA.

1.2 Design Issue with heterogeneous systems

Due to their heterogeneous nature, those systems are difficult to program. The compilation process is complex, requiring multiple passes on different toolchains to distribute the work to the host and the accelerator, and to allocate the memory. And this complexity is also forced on the end-user, as he needs to know the target perfectly to make use of all the available resources.

1.3 Currently Available Workflow for Halide

Currently Hero supports OpenMp, which is an API which "defines a portable, scalable model with a simple and flexible interface for developing parallel applications on platforms from the desktop to the supercomputer"[4]. This API has been implemented on hero, and is currently used to develop some test applications. But exploring the design space using OpenMp's directive, isn't perfect, and require the developer to adapt its code to run with a specific schedule. This approach leads to an important development time but also an extensive texting process whenever the schedule is changed to ensure that the resulting code works as intended.

But exploring the design space using OpenMp's directive, isn't perfect, and require the developer to adapt its code to run with a specific schedule. This approach leads to an important development time but also an extensive texting process whenever the schedule is changed to ensure that the resulting code works as intended.

Halide is a programming language that was designed to allow the developer to explore multiple design choices quickly by separating the algorithm from the execution schedule. This language was designed to be used in image or array processing applications. Every processing pipeline designed with Halide has two parts. The first part consist of the functional description of the processing kernel, this is the algorithm that will be executed on the array. And the second part is the schedule of the pipeline. This schedule describes how the algorithm will be executed on the system. This programming model is interesting because the developer can implement the algorithm without having to take into account the boundaries of the functions or the border effects. Then he can quickly bound the different variables of the pipeline and design it's schedule afterward. All the constraints will be asserted during the compilation without any intervention from the developer.

Chapter 2

Preliminaries / Background

2.1 Hero

2.2 Halide Language

2.2.1 Programing model

Halide is a functionnal program embedded into C++ designed to write high performance image and array processing code [5]. This language uses a functionnal paradigm to describe the functionnalities of the pipeline. The scheduling of the pipeline is described separately, which allow the developper to explore a wide range of schedule without having to rewrite most of the code. Every pipeline is a function (`Halide::Func` composed of other functions and expressions (`Halide:expr`). These two objects use special variables (`Halide:Vars`) to describe the operation executed on the array. The code snippet 2.1 describe a basic pipeline which compute the distance of each coordinate of the array from on position specified by the vector `(center_x, center_y)`.

```
Halide::Var x, y;
Halide::Param center_x, center_y;
Halide::Expr offset = Halide::pow(x - center_x, 2)
                    + Halide::pow(y - center_y, 2);

  gradient(x, y) = offset;
```

Listing 2.1: Simple Pipeline Example

After designing the pipeline, we can define it's schedule via the different directive included in Halide. Halide implements all the basic scheduling option like parallelizing, unrolling, splitting ... These options will be described in the section Basic Scheduling Options .

The snippet 2.2 shows a simple schedule applied on our gradient. This schedule consists of parallelizing the execution over the `x` axis, and unrolling along the `y` axis.

```
gradient.parallel(x);
gradient.unroll(y, 10);
```

Listing 2.2: Simple Pipeline Example

To execute our pipeline, Halide provides a large range of options, we can execute it directly using the `.realize(x_max, y_max)` directive, this will execute the pipeline on a rectangle starting from it's top left corner in `(0,0)` to it's bottom right corner in `(x_max, y_max)`.

But Halide also gives the programmer a lot off options to execute the pipeline, it's able to convert the code to C code, llvm assembly file, or already compiled object file specific to a given target. More over, Halide support a wire variety of CPU architecture (X86, ARM, MIPS, PowerPc, Risc-V), operating systems (Linux, Windows, Android, Mac) and also Gpu Api's (Cuda, OpenCL, OpenGl, DirectX ...). Halide support for new architecture is getting better and better, and is by design targeted for cross compilation and Heterogeneous systems.

2.2.2 Debugging Options

2.2.3 Basic Scheduling Options

Halide implement different scheduling instruction, and most of them aren't architecture specific.

Non Architecture Specific Instrutions

- Reorder : Tells halide how to traverse the domain of the pipeline stage (ffor example in a column major or row major way)
- Split : Split a dimension along inner and outer subdimensions.
- Tile : Cut the domain in tiles.
- Fuse : Join two dimensions in a single fused dimension.
- Unroll : Unroll along one dimension.

Some of the primitives such as vectorize or parallel. needs to be implemented on the target platform as they take advantage of the specificities of the system. To do so Halide uses some functions which are defined in a header file we can get when compiling the pipeline.

2.2.4 Porting Halide to new Platforms

From the header file, we can find the functions vital for halide to work and implement them in the pulp runtime. From then, we have to compile the pipeline to a risc-V object file, and then compile the main application using this object file and the provided header. When we will compile the final application using the hero toolchain, the linker will link the halide function calls to the implemented functions in the pulp runtime. Currently, only the memory allocation, print, and fork primitive are implemented but they are sufficient to try some basic parallel schdeules.

2.3 Compilation Workflow

Hero currently have different platforms, and also different workflow. I started by working on the simulation platform which simulate an eight-core pulp cluster. Then I tried to port it to the hardware platform (One hard ARM core and a PULP cluster implemented on the FPGA). The compilation is slightly different for both platform, so I started by working on the simulation platform which is not heterogeneous.

2.4 Simulation

The compilation process for the simulation platform is quite easy, we start by compiling the C++ code which will generate the final pipeline, we then run this application, to generate the pipeline and the matching header file. Then we compile the true application which will call the pipeline. During the process, we add the object file of the pipeline in the source file, g++ will then link the pipeline with the main application and also the halide calls to the pulp runtime functions.

2.5 The full hero platform

The hardware platform has a more complex compiling process, currently, the applications use OpenMp to distribute the code to the pulp cluster. With instructions such as `#pragma omp target device(BIGPULP_MEMCPY)` to explicitely tell the compiler to distribute the following part of the code to the pulp cluster. The application is compiled using Clang and a the clang-offlad bundler to create an heterogeneous application that will run on both platform.

Chapter 3

Design Implementation

To test halide on hero, I used two benchmark. The first one was a basic gradient example, and the second one a matrix multiplication pipeline that I took in the provided examples and then adapted to be used in a hero application. The matrix axample is more interesting, because it represent what a typical signal processing application may do. It is also quite easy to benchmark with different sizes to see the impact of the memory access on the execution time.

```
ImageParam A(type_of<int>(), 2);
ImageParam B(type_of<int>(), 2);
Var x, y;
Func matrix_mul("matrix_mul");
Func out;

RDom k( 0,A.width() );

matrix_mul(x, y) += A(x, k) * B(k, y);

out(x, y) = matrix_mul(x, y);
```

Listing 3.1: Matrix Multiplication Pipeline

3.1 Schedule Implementation

Most of the schedules implemented on halide doesn't require any platform specific implementation as they are only unrolling, splitting or swapping loops. During my project I used two platform specific schedules: vectorization and parallelization. The `.vectorize(x)` instrucion unrolls one loop in assembly, and the vectorization is done by g++ using the SIMD instructions of the chip. For hero, the simd extention wasn't supported by g++

but, we could still use this instructions as it reduced the number of jumps and thus the total execution time.

The `.parallel(x)` instructions uses two functions: `halide_do_par_for` and `halide_do_par_for_fork`. `halide_do_par_for` adds the tasks to the task queue of the pulp cluster, every task will execute `halide_do_par_for_fork` on the corresponding core (if the core id is equal to the task number modulo the number of available cores). Every task consist of a part of the processing pipeline.

Chapter 4

Results

4.1 Test Setup

I benchmarked two applications on two platforms. I benchmarked the halide port on the hardware simulation for the PULP cluste, and one openMp matrix multiplication application on the developpement platform on a Xilinx ZCU102. For both application, I generated random matrices of diffërent sizes, and for each sizes multiplication I counted the number of cycles needed to complete the operation. With this setup, we can easily compare the performances of halide and OpenMp in a real world scenario for at least two basic schedules: Single threaded and Multi Threaded. To give the results a more meaning I also calculated the number of operations per cycles where one operation can either be an addition a multiplication or a memory access (which take 2 cycles each), so for a matrix of size n, the number of operations to finish the multiplication is : $6 * n * * 3 + n * * 2$ (each coefficient needs $2n$ memory accesses, $2n$ additions and multiplications and one memory store).

4.2 Comparaison between OpenMp and Halide on the different platforms

Chapter 5

Conclusion and Future Work

Draw your conclusions from the results you achieved and summarize your contributions. Comparisons (e.g., of hardware figures) with related work are also appropriate here. Point out things that could or need to be investigated further.

5.1 First Section

5.2 Second Section

Bibliography

[1] A. Shan, "big.little processing - arm," 2012, [Online; accessed 17-May-2020]. [Online]. Available: https://web.archive.org/web/20121022055646/http://www.arm.com/products/processors/technologies/bigLITTLEprocessing.php

[2] A. V. D. M. Tullsen, "Harnessing isa diversity: Design of a heterogeneous-isa chip multiprocessor," *ICSA*, pp. 121–132, 2014.

[3] Hero, "Hero: The open heterogeneous research platform," 2020, [Online; accessed 17-May-2020]. [Online]. Available: https://pulp-platform.org/hero.html

[4] OpenMp, "Home - openmp," 2020, [Online; accessed 17-May-2020]. [Online]. Available: https://www.openmp.org/

[5] Halide, "Halide," 2020, [Online; accessed 17-May-2020]. [Online]. Available: https://www.halide-lang.org/

[6] PULP, "Pulp platform," 2020, [Online; accessed 17-May-2020]. [Online]. Available: https://pulp-platform.org

[7] Wikipedia, "Heterogeneous computing - wikipedia," 2020, [Online; accessed 17-May-2020]. [Online]. Available: https://en.wikipedia.org/wiki/Heterogeneous_computing

[8] A. Shan, "Heterogeneous processing: a strategy for augmenting moore's law | linux journal," 2006, [Online; accessed 17-May-2020]. [Online]. Available: https://www.linuxjournal.com/article/8368