Programming models for optimised compilation

Christophe Alias – Inria/LIP, Ludovic Henrio – CNRS/LIP and Matthieu Moy – Université Lyon1/LIP

2018

1 Programming models for optimized compilation

Advisor : Ludovic Henrio
Co-advisors : Matthieu Moy, Christophe Alias

1.1 Context

The advent of parallelism in supercomputers and in more classical end-user computers increases the need for high-level code optimization and improved compilers. Radical changes are needed to further improve power efficiency, which is the limiting factor for large-scale computing. Improving the performance under a limited energy budget must be done by rethinking computing systems at all levels : hardware, software, compilers, and runtimes.

On the hardware side, new architectures such as multi-core processors, Graphics Processing Units (GPUs), many-core and FPGA accelerators are introduced, resulting into complex heterogeneous platforms. In particular, FPGAs are now a credible solution for energy-efficient HPC. An FPGA chip can deliver the same computing power as a GPU for an energy budget 10 times smaller.

A consequence of this diversity and heterogeneity is that a given computation can be implemented in many different ways, with different performance characteristics. An obvious example is changing the degree of parallelism : this allows trading execution time for number of cores used. However, many choices are less obvious : for example, augmenting the degree of parallelism of a memory-bounded application will not improve performance. Most architectures involve a complex memory hierarchy, hence memory access patterns have a considerable impact on performance too. The design-space to be explored to find the best performance is much wider than it used to be with older architectures, and new tools are needed to help the programmer explore it. The problem is even stronger for FPGA accelerators, where programmers are expected to design a circuit for their application ! High-level languages and hardware compilers (HLS, High-Level Synthesis, that takes as input a C or C-like language and produces a circuit description) are required.

To design better analysis and program transformations, we need to design richer intermediate representations (IR) for compilers, that incorporate primitives needed for efficient parallelism. Program transformation can then generate this IR from a source program, or generate an implementation based on this IR. Constructs of this IR could either be inferred by analyzing the source program or added to the programming language offered to the programmer.

The goal of this internship is to explore, through examples, optimization and parallelization opportunities.

1.2 Objectives

The expected outcomes of the internship are :

— Identify primitives needed for efficient parallelism (e.g. efficient FIFO communication), and implement them as a library.

— Identify patterns in the application that map to parallel constructs, and prototype the transformation algorithms to these constructs by applying them manually.

— Generalization of the concepts above as a programming language.

1.3 Suggested steps of the internship

The internship could follow the following steps (the internship will most likely not treat them all, but rather focus on a few of them) :

1. Understand a small HPC application (e.g. a few tens of lines kernel).
2. Apply optimizations by hand. Optimizations of interest here are mainly the ones using data-flow between parts of the program. For instance, a program containing two consecutive loops may in some case be transformed into one process per loop, with a limited-size buffer between processes.
3. Design high-level programming constructs, for example inspired by actor-based models that would allow splitting the program into multiple sub-programs and help optimization. This would make explicit the composition and interactions of sub-programs.
4. Rework the optimization phase (2) using these constructs.
5. Specify formally the semantics of the newly proposed constructs.
6. Specify formally the optimizations and prove their correction with respect to the semantics.
7. Generalization : use the new model on other programs.

This proposal is very flexible, and may evolve as :

— a more theoretical study, that would focus on points 5 and 6 and on the design of the programming model (but would still start with an example program and its optimization).

— or a more applied study, applied to several programs and focused on the realizability of the optimizations (implementation of a simple optimizing compiler).