Ph.D proposal
Programming model to assemble compute kernels safely and efficiently: Future-based synchronization for arrays and matrices

Ludovic Henrio – CNRS/LIP,
Matthieu Moy – Université Lyon1/LIP
2019

1 Context

Until 2006, the typical power-consumption of a chip remained constant for a given silicon area as the transistor size decreased (this evolution is referred to as Dennard scaling [6]). In other words, energy efficiency was following an exponential law similar to Moore’s law. This is no longer true, hence 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.

In particular, parallelism is the obvious key to efficiency. Parallelism has historically been added to existing programming languages as libraries, which had to be handled manually by the programmer. We believe, however, that parallelism should be more tightly integrated in the language and compiler. The compiler should offer safe constructs to the programmer, and be able to generate efficient code.

Parallelism makes the task of finding the most efficient implementation harder: not only the sequential computations should execute fast, but the communications and synchronization must be handled efficiently too. Communications imply some latency, which should occur at the same time as some computation to achieve efficient execution. An analysis of the program can help the compiler to make it parallel and optimize it. In some case, typically small pieces of code with static control loops, the compiler can infer automatically a parallel and efficient implementation. However, compute kernels must also be assembled efficiently, and while the compiler can help, the general case can hardly be dealt with in a fully automatic way. We therefore need constructs to allow the programmer to assemble compute kernels safely and efficiently.

Typical HPC application manipulate large pieces of data, for which dataflow mechanisms are well suited: data can be streamed from one kernel to another, without waiting for the first one to terminate. Synchronization is based on the availability of data.

Several principles have already been proposed, such as dataflow actor languages, data-aware process networks used in the CASH team, the CAL Actor Language. A widely used synchronization mechanism is the notion of Future, a placeholder for a piece of data being computed. Asynchronous function calls can return a Future, which will be resolved when the computation terminate. A Future exposes to the programmer a get operation that waits for the Future to be resolved, and then returns the result of computation.

2 Objective of the Ph.D

The objective of this PhD thesis is to design new coordination patterns between processes that allow the efficient execution of applications made of the assembly of several kernels with well defined data dependencies. The principle of the approach is to design new interaction patterns adapted to the transmission of big quantities of data typically stored in matrices. We believe that the dataflow interaction paradigm is valuable to efficiently couple kernels of computations, but also that polyhedral computation model has proven its merits when it comes to efficiently execute a loop or a few interacting loops. We want to design new data containers that will at the same time allow for optimizations inspired by the polyhedral model and enable dataflow synchronization between processes.
A construct that at the same time can store data when it is available, act as a synchronization entity, and allow for a wide variety of accesses would clearly be inspired by the notion of Futures.

Obvious, but inefficient solutions to the problem include “Future of matrix”, i.e. a Future object whose value is a matrix, and “matrix of Future”, i.e. a Matrix whose elements are Future, themselves containing numbers.

An efficient construct for Future matrix would be more efficient than a “matrix of Futures” that would store one Future for each element of the matrix, which would be inefficient in terms of space and access time, it would also be less monolithic than a “Future of matrix” and only accessible when the full matrix is filled. Instead we want to use optimization techniques to derive the optimal granularity of the synchronization, but also dynamic information allowing the dataflow synchronization to be adapted at runtime depending on the data available. Somehow, such a dataflow addressable matrix will allow both runtime optimization techniques automatically splitting the data to be transmitted into blocks of data of an adequate size, but also dynamic optimization allowing the program to check which quantity of data is available before triggering the next computation step. These features should be provided in a way that is, as much as possible transparent to the programmer, in the same spirit as optimizing compilers.

The resulting framework would then allow automatically to couple high-performance computation kernels in an optimal way.

### 3 Program

The PhD may follow the following steps (some of the steps could be briefly addressed in order to focus on the most promising aspects):

- Study of interaction patterns in concurrent and dataflow programming models, including Futures;
- Study of parallelizing compilers and optimizing compilation;
- Design of an interaction paradigm for concurrent processes communicating by dataflow-synchronized matrices;
- Proposal for a static optimizing compiler using dataflow synchronization on automatically sized block-matrices, where the size of the transmitted data is either given or provided by preliminary benchmarks and heuristics;
- Proposal for a more flexible synchronization pattern where the amount of data that triggers the synchronization can be adjusted dynamically, without changing the behavior of the program;
- Evaluation of the proposed solution;
- Proof of correctness of the optimization and proof of safety of the construct.

### 4 Sources of Inspirations

Among the sources of inspiration to design a new paradigm:

- The notion of actor language, such as CAL [5] or Encore [4].
- The notion of Dataflow Explicit Future [7], that allows a chain of asynchronous method calls to be handled as a single synchronization (i.e. a Future containing a Future containing X is equivalent to a Future containing X).
- The polyhedral model allows very aggressive optimizations of compute kernels, in particular the computation can be scheduled globally and parallelized automatically [3]. However, the polyhedral model does not scale to large programs and cannot deal with non-static control (e.g. while loop whose bound cannot be computed statically). The idea of this thesis is to compose kernels, typically optimized using polyhedral methods, in a more generic and possibly dynamic way. The schedule of the kernels assembled can be used to optimize communication and synchronization, and conversely, knowing the communication patterns between kernels can change the optimal schedule within a kernel (e.g. one kernel can be hinted to produce data in the order they will be consumed by another).
- The Data-aware Process Network (DPN) [1] formalism, which uses heavily dataflow communication, mostly based on FIFO communications. DPN was designed to be applied to hardware circuit generation (aka High-Level Synthesis), and uses fine-grained communication patterns, exchanging individual values one at a time. DPN can be manipulated formally and optimized using the polyhedral model, and are well-suited for intermediate representations within compilers, but we still need user-oriented constructs to be used directly by the programmer.
— Streaming Futures [2], which extend the notion of Futures to streams of data. They are close to the constructs we are looking for, but are limited to in-order communication, and are not directly suited to stream matrices from one part of the program to another.

— Efficient communication schemes such as lock-free FIFO [8]. These schemes can be used as low-level runtime systems, but are not sufficient as a programming model.

Location and Supervising

The Ph.D will take place in LIP, Lyon, France. It will be supervised by Ludovic Henrio, CNRS researcher, and Matthieu Moy, assistant professor at Lyon 1. Ludovic Henrio is a specialist in actor languages and Futures. Matthieu Moy is the leader of the CASH team, his research interests include parallel languages both for HPC and for embedded systems.

Références


