Analyses, Hardware/Software Compilation, Code Optimization for Complex Applications

Christophe Alias & Laure Gonnord

March 9, 2017

The advent of parallelism in supercomputers, in embedded systems (smartphones, plane controllers), and in more classical end-user computers increases the need for high-level code optimization and improved compilers. Being able to deal with the complexity of the upcoming software and hardware while keeping energy consumption at a reasonable level is one of the main challenges cited in the Hipeac Road-map which among others cites the two major issues :

- Enhance the efficiency of the design of embedded systems, and especially the design of optimized specialized hardware.
- Invent techniques to "expose data movement in applications and optimize them at runtime and compile time and to investigate communication-optimized algorithms".

In particular, the rise of embedded systems and high performance computers in the last decade has generated new problems in code optimization, with strong consequences on the research area. The main challenge is to take advantage of the characteristics of the specific hardware (generic hardware, or hardware accelerators). The long-term objective is to provide solutions for the end-user developers to use at their best the huge opportunities of these emerging platforms.

1 Dataflow models for HPC applications

In the last decades, several frameworks has emerged to design efficient compiler algorithms. The efficiency of all the optimizations performed in compilers strongly relies on performant *static analyses* and *intermediate representations*.

The transverse theme of this proposal is the study of the dataflow model for programs: the dataflow formalism expresses a computation on an infinite number of values, that can be viewed as successive values of a variable during time. A dataflow program is structured as a set of *communicating processes* that communicate values through *communicating buffers*.

Examples of dataflow languages include the synchronous languages Lustre and Signal, as well as SigmaC; the DPN representation [4] (data-aware process network) is an example of a dataflow intermediate representation for a parallelizing compiler.

The dataflow model, which expresses at the same time data parallelism and task parallelism, is in our opinion one of the best model for our analyses, verification and code production tools. This model will be our favorite representation for our programs.

2 Compiler algorithms and tools for irregular applications

The dataflow model only is not capable of expressing low-grain parallelism such that instruction parallelism. For this, we advocate for the use of another intermediate representation for the analyses of of the

LIP – UMR CNRS / ENS Lyon / UCB Lyon 1 / INRIA – 69007 Lyon
E-mail : Christophe Alias@ens-lyon.fr, Laure.Gonnord@ens-lyon.fr

dataflow blocks code as well as communicating buffers.

The polyhedral model focus on regular programs, whose execution trace is predictable statically. The program and the data accessed are represented with a single mathematical object endowed with powerful algorithmic techniques for reasoning about it. Unfortunately, most of the algorithms used in scientific computing do not fit totally in this category.

We plan to explore the extensions of these techniques to handle irregular programs with while loops and complex data structures (such as trees, and lists). This raises many issues. We cannot represent finitely all the possible executions traces. Which approximation/representation to choose? Then, how to adapt existing techniques on approximated traces while preserving the correctness?

To address these issues, we plan to incorporate new ideas coming from the abstract interpretation community: control flow, approximations, and also shape analysis; and from the termination community: rewriting is one of the major techniques that are able to handle complex data structures and also recursive programs. Let us point out that although our work is completely *static* in essence, we strongly believe that a definitive solution to performance lies on the static-dynamic combination at compile time, and on clever runtimes.

Targeted applications

- Dataflow programs (SigmaC [5]) for with the low grain parallelism is not taken into account.
- Recursive programs operating on arrays, lists, trees.
- Worklist algorithms: lists are not handled with the polyhedral domain.

Expected impact The long-term expected impact of these work is the significantly widened applicability of various tools/compilers related to parallelization. Analysis and representation of schedules are fundamental parts of compiler analysis, especially important when targeting highly parallel architectures. The extension of static analysis capabilities to complex data structures allows programmers to benefit from sophisticated tools when writing programs dealing with such data structures. The current uses of polyhedral techniques are not limited to automatic parallelization, but are also used for detection of parallel bugs, proving program termination, detecting transient errors, and so on. The proposed work is expected to constitute a major milestone in the area of parallel computing, and especially automatic scheduling, from the theoretical foundations to practical issues.

3 Hardware Compilation for Reconfigurable Architectures

Since the end of Dennard scaling, energy consumption bounds the performance of supercomputers. Computing systems (hardware, software, compilers, runtimes) must be rethinked to deliver better performances with a limited energy budget. This is the purpose of the exaflop computing challenge, which proposes nothing less than bounding the power consumption of an exaflop computer to 20 megawatts. In the last decade, many specialized hardware accelerators (Xeon Phi, GPGPU) has emerged to improve the energy efficiency of mainstream processors by trading the genericity for power consumption. However, the best supercomputers can only reach 8 gigaflops per watt [12], which is far less than the 50 gigaflops per watt required for exaflop computing. An extreme solution would be to trade all the genericity by using specialized circuits. However such circuits (application specific integrated circuits, ASIC) are usually too expensive for the HPC market and lacks of flexibility. Once printed, an ASIC cannot be

modified, any algorithm update/bug fix is made impossible, which is clearly not acceptable. Recently, reconfigurable circuits (Field Programmable Gate Arrays, FPGA) have appeared as a credible alternative for exaflop computing. Major companies (including Intel, Google, Facebook and Microsoft) show a growing interest to FPGA and promising results have already been obtained. For instance, Microsoft has reached 40 gigaflop per watts on a big data deep learning algorithm mapped on Intel Arria 10 FPGA. *We believe that FPGA will become the new building block for supercomputers and data centers.*

With FPGA, we have to reconsider most of the questions addressed in high-performance computing (HPC). For example, how scheduling can leverage dynamic reconfiguration of computing resources? Which trade-offs are induced by dynamic reconfiguration? How to map application kernels to FPGA basic bricks? How to verify the translation? With the growing interest of computer industry to FPGA, there is a is a strong, urging, need for new programming languages, static analysis and compiler technologies. In this proposal, we plan to address the mapping of computation-intensive kernels to FPGA, usually referred as high-level synthesis (HLS), by building on parallelization techniques developed by the HPC community in the last decades. Beyond the usual trade-offs related to parallelism and data reuse, we must address the specificities of FPGA architecture. Which parallel architecture is best suited for a target FPGA? Which compiler analysis to derive such an architecture?

As we already developed in section 1, we plan to leverage the power of *dataflow representations* [4], which capture all the available parallelism. We will develop compiler transformations to derive step by step a relevant parallel architecture. To do so, we may have to *extend/cross-fertilize algorithms developed in both HPC and high-level synthesis*, such as parallelism extraction [6], pipeline scheduling [3], buffer sizing [1] or data transfer optimization [2].

Targeted applications We will target regular kernels used extensively in both HPC and big data applications:

- Compute-intensive kernels used in HPC (linear solvers, matrix factorizations, etc). Many kernels can be found in the Polybench/C benchmark suite [10].
- Data-intensive kernels used in data center applications such as image processing or deep learning.

Expected Impact The short-term impact is to show how high-level synthesis for FPGA can leverage high-performance compiler analysis (parallelization, data reuse, load balance, etc), thereby launching a bridge between HPC and HLS communities. From an industrial point of vue, we plan to transfer the results of this research to the XtremLogic start-up company [13], co-founded by Christophe Alias and Alexandru Plesco.

As for the long-term impact, we believe that high-level synthesis can leverage the concepts involved in the irregular analysis described in section 2, thus extending profitably the scope of our compiler analysis.

4 Low cost analyses for efficient optimization

The design and implementation of efficient compilers becomes more difficult each day, as they need to bridge the gap between *complex languages* and *complex architectures*. On one hand, high-level programming languages tend to become more distant from the hardware which they are meant to command. Application developers use languages that bring them close to the problem that they need to

solve. These languages use constructs such as closures, parametric polymorphism and high order functions, which are not directly supported in hardware. The broad availability of parallel architectures has also led to the revival of languages following non-sequential paradigms, such as CUDA and OpenCL. The contemporary computer, on the other hand, is constantly evolving. New architectures such as multi-core processors, Graphics Processing Units (GPUs) or many-core coprocessors are introduced, resulting into complex heterogeneous platforms. Whereas the old-day compiler could perform an almost direct translation from a sequential program to machine instructions, the present day translator needs to link two very different worlds.

To address these issues, we plan to cross fertilize ideas coming from the abstract interpretation community as well as language design and dataflow semantics. We already have experience in designing low-cost semi relational abstract domains for pointers [9, 8], as well as tailoring static analyses for specialized applications [7, 11].

Targeted applications

- Generalist programs with complex behaviors: detecting non licit memory accessed, memory consumption, hotspots, ...
- Functional properties for large programs.
- GPGPU programs where we want to optimize copies from the global memory to the block kernels, to perform less data accesses and change data layout to improve locality.
- Dataflow programs with arrays and iterators operating on arrays.
- (more long term) Cryptographic algorithms.

Expected impact The short-term expected impact is leverage the applicability of *abstract domains* so that they could be used as base-passes for client analyses/optimizations in state-of-the art compiler infrastructures such as LLVM. In a more long-term, we expect to be able to design application/platform-tailored abstract domains for domain specific languages compiled and run on multicore systems as well as specific machines, without reinventing complete toolchains.

References

[1] Christophe Alias, Fabrice Baray, and Alain Darte. Bee+Cl@k: An implementation of lattice-based array contraction in the source-to-source translator Rose. In *ACM Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES'07)*, 2007.

[2] Christophe Alias, Alain Darte, and Alexandru Plesco. Optimizing remote accesses for offloaded kernels: Application to high-level synthesis for FPGA. In *ACM SIGDA Intl. Conference on Design, Automation and Test in Europe (DATE'13)*, Grenoble, France, 2013.

[3] Christophe Alias, Bogdan Pasca, and Alexandru Plesco. FPGA-specific synthesis of loop-nests with pipeline computational cores. *Microprocessors and Microsystems*, 36(8):606–619, November 2012.

[4] Christophe Alias and Alexandru Plesco. Data-aware Process Networks. Research Report RR-8735, Inria - Research Centre Grenoble – Rhône-Alpes, June 2015.

[5] Pascal Aubry, Pierre-Edouard Beaucamps, Frédéric Blanc, Bruno Bodin, Sergiu Carpov, Loïc Cudennec, Vincent David, Philippe Doré, Paul Dubrulle, Benoît Dupont De Dinechin, François Galea, Thierry Goubier, Michel Harrand, Samuel Jones, Jean-Denis Lesage, Stéphane Louise, Nicolas Morey Chaisemartin, Thanh Hai Nguyen, Xavier Raynaud, and Renaud Sirdey. Extended Cyclostatic Dataflow Program Compilation and Execution for an Integrated Manycore Processor. In *Alchemy 2013 - Architecture, Languages, Compilation and Hardware support for Emerging ManYcore systems*, volume 18 of *Proceedings of the International Conference on Computational Science, ICCS 2013*, pages 1624–1633, Barcelona, Spain, June 2013.

[6] Uday Bondhugula, Albert Hartono, J. Ramanujam, and P. Sadayappan. A practical automatic polyhedral parallelizer and locality optimizer. In *Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation, Tucson, AZ, USA, June 7-13, 2008*, pages 101–113, 2008.

[7] Paul Feautrier, Abdoulaye Gamatié, and Laure Gonnord. Enhancing the Compilation of Synchronous Dataflow Programs with a Combined Numerical-Boolean Abstraction. *CSI Journal of Computing*, 1(4):8:86–8:99, 2012. RR version = http://hal.inria.fr/hal-00780521/en.

[8] Maroua Maalej, Vitor Paisante, Pedro Ramos, Laure Gonnord, and Fernando Pereira. Pointer Disambiguation via Strict Inequalities. In *Code Generation and Optimisation*, Austin, United States, February 2017.

[9] Vitor Paisante, Maroua Maalej, Leonardo Barbosa, Laure Gonnord, and Fernando Magno Quintao Pereira. Symbolic Range Analysis of Pointers. In *International Symposium of Code Generation and Optmization*, pages 791–809, Barcelon, Spain, March 2016.

[10] Louis-Noël Pouchet. Polybench: The polyhedral benchmark suite. *URL: http://www. cs. ucla. edu/˜ pouchet/software/polybench/[cited July,]*, 2012.

[11] Henrique Nazaré Willer Santos, Izabella Maffra, Leonardo Oliveira, Fernando Pereira, and Laure Gonnord. Validation of Memory Accesses Through Symbolic Analyses. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages And Applications (OOPSLA'14)*, Portland, Oregon, United States, October 2014.

[12] The green500 list - november, http://www.green500.org/lists/green201511, 2015.

[13] Xtremlogic sas, http://www.xtremlogic.com, 2017.