Efficient static analyses for GPUs

Laure Gonnord, Matthieu Moy – Université Lyon1/LIP
Caroline Collange – Inria Rennes/IRISA

2020

Keywords

Abstract interpretation, compilation, GPU, scratchpad memory, numerical abstract domains.

Location and supervision

The internchip will take place in LIP, Lyon, in partnership with IRISA, Rennes. It will be cosupervised by Matthieu Moy (60%, main supervisor), Laure Gonnord (20%) at LIP, and Caroline Collange (20%) at IRISA.

General Context

The gap between memory performance and compute performance of processor architectures keeps widening. On modern architectures, external memory accesses cost orders of magnitude more than computations. For instance, the Nvidia Geforce 1080 Ti GPU can perform 100 floating-point operations for each word read from external memory.

As a consequence, reusing data through on-chip memories such as caches or scratchpads is of paramount importance to optimize performance and minimize energy consumption. On-chip memory is fast and low-power but its capacity is severely restricted.

GPUs, FPGAs typically rely on software-managed caches (scratchpad memories) in addition to or as an alternative to hardware-managed caches. The compiler is responsible for data placement in such software-managed on-chip memory. Proper data placement demands accurate static analyses to identify which data should be placed in scratchpad memories. Indeed, caches should only retain data that is effectively reused.

In practice, GPU-targeted programming models like OpenCL and CUDA let software express data reuse by partitioning the parallel domain. A single program, or *kernel*, is executed by many *work-items*, grouped in *work-groups*. Threads within the same work-group may synchronize together and exchange data through user-allocated *local* memory, which physically resides on-chip. Local memory is typically used to stage data coming from the off-chip *device* memory.

The global objective of this thesis is to design efficient analysis to allow code transformation to minimize accesses to the external memory.

Such code transformation need to know the addresses accessed by the program, to be able to pre-fetch the corresponding values.

Internship content

The global objective is to design static analysis of programs to infer properties that allow code transformation to minimize memory accesses. We focus on programs manipulating 2-dimensional arrays, and programs running on GPU, where memory optimization is particularly crucial.

A straightforward range analysis is not sufficient for several reasons :

1. Addresses are rarely constant, but usually relative, e.g. to the start address of an array
2. For programs manipulating multi-dimensional arrays, a convex shape within the array is not convex in terms of address. For example a rectangle area within a 2D-array is a sequence of intervals. Using the usual convex hull operation on such shapes would loose important information.

LIP – UMR CNRS / ENS Lyon / UCB Lyon 1 / Inria – 69007 Lyon
IRISA – UMR CNRS / Centrale Supelec / ENS Rennes / IMT Atlantique / Inria / INSA / Université Bretagne Sud / Université Rennes 1
E-mail : Caroline.Collange@inria.fr et Laure.Gonnord@ens-lyon.fr et Matthieu.Moy@univ-lyon1.fr

As a consequence, we need symbolic analysis to address point 1), and intervals modulo a value to address point 2). The analysis will likely rely on classical abstract interpretation, with a new numerical domain adapted to 2D arrays. The numerical domain should be designed to be scalable, although the scalability constraint is not as strong for GPU compute kernels (which are usually relatively small) than for program-wide analysis.

The intership will consist of some of the following steps :

— Study a set of programs abstracted away from “real world GPU programs”, with the help of Caroline Collange.

— Design a static analysis that would allow program optimisation.

The work can rely on previous work on symbolic range analysis for pointers in the Ph.D of Maroua Maalej [3]. Compared to state of the art array analysis [4, 1], the precision can be relaxed since we do not need any information about the array content, but only information about addresses, hence array indices. Related work include the polyhedral model's tiling transformations [5, 2], able to perform similar transformations. The goal here is to use abstract interpretation to get more widely applicable results.

Références

[1] Patrick Cousot, Radhia Cousot, and Francesco Logozzo. A parametric segmentation functor for fully automatic and scalable array content analysis. In *Proceedings of the 38th ACM Symposium on Principles of Programming Languages*, pages 105–118. ACM, 2011.

[2] Guillaume Iooss. *Detection of linear algebra operations in polyhedral programs*. Theses, Université de Lyon ; Colorado state university, July 2016.

[3] Maroua Maalej Kammoun. *Low-cost memory analyses for efficient compilers*. Theses, Université de Lyon, September 2017.

[4] David Monniaux and Laure Gonnord. Cell morphing : From array programs to array-free horn clauses. In Xavier Rival, editor, *Static Analysis - 23rd International Symposium, SAS 2016, Edinburgh, UK, September 8-10, 2016, Proceedings*, volume 9837 of *Lecture Notes in Computer Science*, pages 361–382. Springer, 2016.

[5] Jingling Xue. *Loop Tiling for Parallelism*. Kluwer, 2000.