A high-level synthesis approach optimizing accumulations in floating-point programs using custom formats and operators

Yohann Uguen, Florent de Dinechin, Steven Derrien

A high-level synthesis approach optimizing accumulations in floating-point programs using custom formats and operators

Yohann Uguen
Univ Lyon, INSA Lyon, Inria, CITI
F-69621 Villeurbanne, France
Yohann.Uguen@insa-lyon.fr

Florent de Dinechin
Univ Lyon, INSA Lyon, Inria, CITI
F-69621 Villeurbanne, France
Florent.de-Dinechin@insa-lyon.fr

Steven Derrien
University Rennes 1, IRISA
Rennes, France
Steven.Derrien@univ-rennes1.fr

Many case studies have demonstrated the potential of Field-Programmable Gate Arrays (FPGAs) as accelerators for a wide range of applications. FPGAs offer massive parallelism and programmability at the bit level. This enables programmers to exploit a range of techniques that avoid many bottlenecks of classical von Neumann computing. However, development costs for FPGAs are orders of magnitude higher than classical programming. A solution would be the use of High-Level Synthesis (HLS) tools, which use `C` as a hardware description language. However, the C language was designed to be executed on general purpose processors, not to generate hardware. Its datatypes and operators are limited to a small number (more or less matching the hardware operators present in mainstream processors), and HLS tools inherit these limitations. To better exploit the freedom offered by hardware and FPGAs, HLS vendors have enriched the C language with integer and fixed-point types of arbitrary size. Still, the operations on these types remain limited to the basic arithmetic and logic ones.

In floating point, the current situation is even worse. The operator set is limited, and the sizes are restricted to 32 and 64 bits. Besides, most recent compilers, including the HLS ones, attempt to follow established standards, in particular C11 and IEEE-754. This ensures bit-exact compatibility with software, but greatly reduces the freedom of optimization by the compiler. For instance, a floating point addition is not associative even though its real equivalent is.

In the present work we attempt to give the compiler more freedom. For this, we sacrifice the strict respect of the IEEE-754 and C11 standards, but we replace it with the strict respect of a high-level accuracy specification expressed by the programmer through a `pragma`.

The case study in this work is a program transformation that applies to floating-point additions on a loop's critical path. It decomposes them into elementary steps, resizes the corresponding subcomponents to guarantee some user-specified accuracy, and merges and reorders these components to improve performance. The result of this complex sequence of optimizations could not be obtained from an operator generator, since it involves global loop information.

For this purpose, we used a compilation flow involving one or several source-to-source transformations operating on the code given to HLS tools (Figure 1).

The proposed transformation already works very well on 3 of the 10 FPMarks where it improves both latency and accuracy by an order of magnitude for comparable area. For 2 more benchmarks, the latency is not improved (but not degraded either) due to current limitations of HLS tools. This defines short-term future work.

The main result of this work is that HLS tools also have the potential to generate efficient designs for handling floating-point computations in a completely non-standard way.

In the longer term, we believe that HLS flows can not only import application-specific operators from the FPGA literature, they can also improve them using high-level, program-level information.

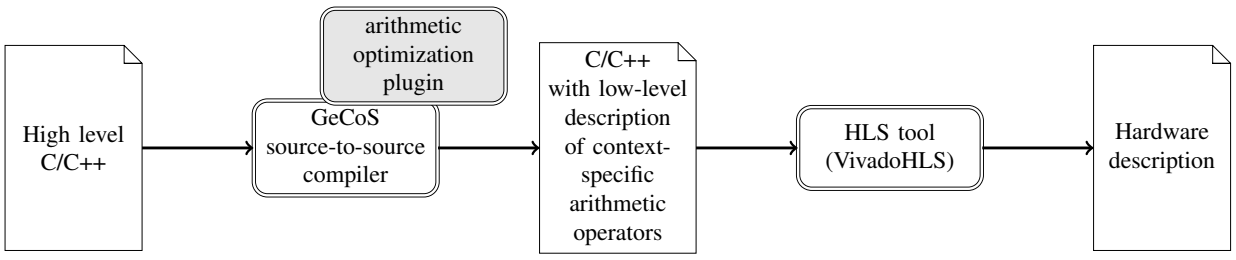

Figure 1: The proposed compilation flow