Ergonomic C/C++ source-to-source analysis and transformations for HLS using Clava

Tiago Santos
INESC-TEC, Faculdade de Engenharia
Universidade do Porto
Porto, Portugal
tiagolascasas@fe.up.pt

João Bispo
INESC-TEC, Faculdade de Engenharia
Universidade do Porto
Porto, Portugal
jbispo@fe.up.pt

João M. P. Cardoso
INESC-TEC, Faculdade de Engenharia
Universidade do Porto
Porto, Portugal
jmpc@fe.up.pt

I. Introduction

The use of High-level Synthesis (HLS) tools as a means of accelerating CPU applications by offloading critical regions to an FPGA is increasingly relevant. However, it still presents significant challenges for software developers, e.g., when exploring parallelism and pipelining and when deciding which application regions should be offloaded to optimize a specific metric. It is commonly accepted that automation support needs to be enhanced to make this flow enticing and easy to use and to alleviate the need for expert knowledge.

Bearing in mind this, we propose using Clava [1], a C/C++ source-to-source compiler. Clava can automatize many design decisions through AST-based analyses and transformations. The current version of Clava runs on Node.js, allowing extensions to be quickly developed in either JavaScript or TypeScript. These extensions can then be easily distributed and combined using the Node Package Manager (NPM).

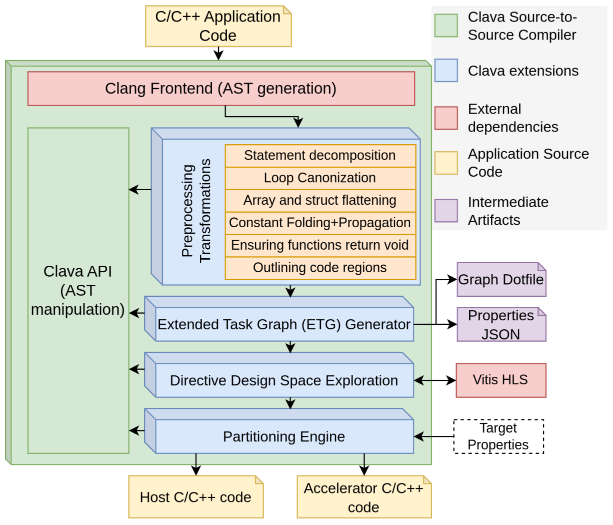

Fig. 1. Software-centric HLS workflow using Clava

II. Clava Use Cases

In this demo, we present Clava, a software-centric HLS workflow based on Clava (see Fig. 1), and show its use through the following use cases:

- An example-driven overview of Clava, showing how a developer can easily write and distribute an extension using TypeScript and NPM. In just a few lines of code, one can modify the AST to, e.g., apply instrumentation to every loop in an application, while visualizing the changes in the AST and in the output source code at compile runtime;
- A demonstration of code transformations done with Clava, such as function inlining/outlining, array and struct flattening, and reduction to a C subset suitable for HLS;
- Generation of an Extended Task Graph (ETG) [2] for a given application, allowing us to have a traditional task-based model of any application. We show how we can use this graph to statically analyze the application, focusing on task-level parallelism, communication, and lifetimes of data items across tasks. Furthermore, we show how to merge tasks in the ETG into a single task, or to split tasks into smaller ones, to decrease/increase the graph's granularity;
- Automatically synthesizing a task from the ETG by interacting with Vitis HLS, and performing Design Space Exploration (DSE) over an HLS directive (e.g., the unrolling factor of a loop);
- Selecting a cluster of tasks for FPGA offloading, extracting them to their compilation unit, and creating the necessary XRT communication boilerplate between host and kernel code.

We show how these use cases are combined into an HLS flow, and how they can be used independently, making them suitable for users interested in specific features or functionalities.

References

[1] J. Bispo and J. M. Cardoso, "Clava: C/c++ source-to-source compilation using lara," *SoftwareX*, vol. 12, p. 100565, 2020. [Online]. Available: https://doi.org/10.1016/j.softx.2020.100565

[2] T. Santos, J. Bispo, and J. M. Cardoso, "A flexible-granularity task graph representation and its generation from c applications (wip)," in *Proceedings 25th Int. Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES'2024)*. New York, NY, USA: ACM, 2024, pp. 178–182. [Online]. Available: https://doi.org/10.1145/3652032.3657580