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, and improving developer productivity by leveraging JIT compilation. By leveraging these languages and runtimes, Clava also utilizes the Node Package Manager (NPM), enabling extensions to be published and shared online. This enhances the reusability and composability of existing transformations for a wide range of new use cases, leading to the incremental development of increasingly complex compilation flows.

II. Clava Use Cases

In this demo, we present a software-centric HLS workflow [2] based on Clava, consisting of several individual extensions that interact with each other. We depict this flow in Fig. 1, and document the extensions in Tab. I. We organize our demo along these scenarios:

- 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. Furthermore, we show how we can use breakpoint-like semantics to render, on the browser, a representation of the AST at selected points in the program;
- A demonstration of some possible code transformations that are available for Clava:
  - Function outlining - by delineating a code region through the use of pragmas or programmatically, the compiler can extract that code region into its own function, creating the call and all the required parameters;
  - Extended function inlining - recursively inlines a call to a function, and all the calls inside that function. Clava is agnostic towards the underlying C/C++ translation units, as it has a single unified AST, which makes this transformation possible for functions with implementations across multiple files;
  - Function voidification - ensures a given function has no return value. It does this by creating a variable for the result, and passing a reference to it as an additional function argument;
  - Struct and array flattening - removes structs by converting their fields into individual variables, and converts all multidimensional arrays into a single dimension. This is useful in improving the HLS synthesizability of C/C++, as it eliminates some indirect pointers;
  - Source code amalgamation - converts a multi-file C/C++ application into a single file, considering only functions in the program's call graph and ensuring all type definitions and global variables are accounted for. It is a useful transformation for simplifying compilation flows, and maximizing tool compatibility.
- Generation of an Extended Task Graph (ETG) [17] 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

TABLE I
Clava NPM packages highlighted in the demonstration

Package	GitHub	NPM
clava	[3]	[4]
clava-visualization	[5]	[6]
clava-code-transforms	[7]	[8]
extended-task-graph	[9]	[10]
clava-vitis-integration	[11]	[12]
clava-scalehls-integration	[13]	[14]
hoopa	[15]	[16]

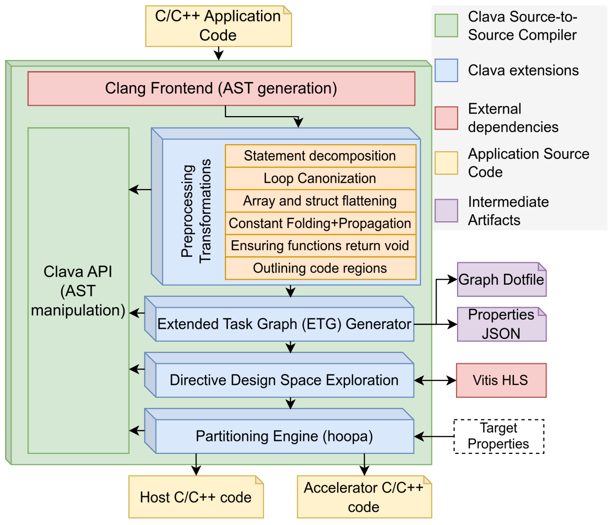
C/C++ Application Code
Clang Frontend (AST generation)
Clava API (AST manipulation)
Preprocessing Transformations
Statement decomposition
Loop Canonization
Array and struct flattening
Constant Folding+Propagation
Ensuring functions return void
Outlining code regions
Extended Task Graph (ETG) Generator
Directive Design Space Exploration
Partitioning Engine (hoopa)
Graph Dotfile
Properties JSON
Vitis HLS
Target Properties
Host C/C++ code
Accelerator C/C++ code
Clava Source-to-Source Compiler
Clava extensions
External dependencies
Application Source Code
Intermediate Artifacts

Fig. 1. Software-centric HLS workflow using Clava

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);
- Integrating Clava with ScaleHLS [18], a state-of-the-art source-to-source tool that transforms and optimizes a software kernel for HLS. We show how to generate MLIR [19] code for any AST function using Polygeist [20], which is then fed into ScaleHLS. The C/C++ code output by ScaleHLS is then parsed back into Clava and integrated into the AST, allowing for the application of additional transformations on top of it;
- Selecting a cluster of tasks for FPGA offloading, extracting them to their compilation unit, and creating the necessary communication boilerplate between host and kernel code. We highlight two partitioning policies: the selection of the task with the highest latency, as provided by Vitis HLS estimates produced on demand; and a cluster of tasks defined manually, which requires the tool to automatically validate their inter- and intra-task dependencies and encapsulate them into a single function. Furthermore, we show that we can generate the CPU-FPGA communication using different APIs, such as AMD's Xilinx Runtime (XRT) and OmpSs@FPGA [21].

Not only can these scenarios be combined into the aforementioned HLS flow, but they are also relevant to people interested only in specific packages or Clava features. We believe this demonstration can promote Clava and its extensions as an enticing component of state-of-the-art compilation flows for heterogeneous systems, as its source-to-source nature preserves both the retargetability and readability of the output source code.

III. Teaser Video

We provide the following teaser video for our demonstration: https://youtu.be/JJVgWboF8OU

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. P. Cardoso, "Ph.d. project: Holistic partitioning and optimization of cpu-fpga applications through source-to-source compilation," in *2025 IEEE 33rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2025, pp. 308–309.

[3] SPeCS, "Clava," https://github.com/specs-feup/clava, 2025, accessed: 2025-05-31.

[4] ——, "Clava," https://www.npmjs.com/package/@specs-feup/clava, 2025, accessed: 2025-05-31.

[5] ——, "Clava visualization," https://github.com/specs-feup/clava-visualization, 2025, accessed: 2025-05-31.

[6] ——, "Clava visualization," https://www.npmjs.com/package/@specs-feup/clava-visualization, 2025, accessed: 2025-05-31.

[7] ——, "Clava code transforms," https://github.com/specs-feup/clava-code-transforms, 2025, accessed: 2025-05-31.

[8] ——, "Clava code transforms," https://www.npmjs.com/package/@specs-feup/clava-code-transforms, 2025, accessed: 2025-05-31.

[9] ——, "Extended task graph," https://github.com/specs-feup/extended-task-graph, 2025, accessed: 2025-05-31.

[10] ——, "Extended task graph," https://www.npmjs.com/package/@specs-feup/extended-task-graph, 2025, accessed: 2025-05-31.

[11] ——, "Clava vitis integration," https://github.com/specs-feup/clava-vitis-integration, 2025, accessed: 2025-05-31.

[12] ——, "Clava vitis integration," https://www.npmjs.com/package/@specs-feup/clava-vitis-integration, 2025, accessed: 2025-05-31.

[13] ——, "Clava scalehls integration," https://github.com/specs-feup/clava-scalehls-integration, 2025, accessed: 2025-05-31.

[14] ——, "Clava scalehls integration," https://www.npmjs.com/package/@specs-feup/clava-scalehls-integration, 2025, accessed: 2025-05-31.

[15] ——, "Hoopa," https://github.com/specs-feup/hoopa, 2025, accessed: 2025-05-31.

[16] ——, "Hoopa," https://www.npmjs.com/package/@specs-feup/hoopa, 2025, accessed: 2025-05-31.

[17] 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

[18] H. Ye, H. Jun, H. Jeong, S. Neuendorffer, and D. Chen, "Scalehls: a scalable high-level synthesis framework with multi-level transformations and optimizations," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022.

[19] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "MLIR: Scaling compiler infrastructure for domain specific computation," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021, pp. 2–14.

[20] W. S. Moses, L. Chelini, R. Zhao, and O. Zinenko, "Polygeist: Raising c to polyhedral mlir," in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '21. New York, NY, USA: Association for Computing Machinery, 2021.

[21] J. Bosch, X. Tan, A. Filgueras, M. Vidal, M. Mateu, D. Jiménez-González, C. Álvarez, X. Martorell, E. Ayguade, and J. Labarta, "Application acceleration on fpgas with ompss@fpga," in *2018 International Conference on Field-Programmable Technology (FPT)*, 2018, pp. 70–77.