Ph.D. Project: Holistic Partitioning and Optimization of CPU-FPGA Applications Through Source-to-Source Compilation

33rd IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM 2025)

May 4-7, Fayetteville, Arkansas, USA

Tiago Santos, João Bispo, João M. P. Cardoso

Faculty of Engineering of the University of Porto and INESC-TEC, Porto, Portugal

{tiagolascasas, jbispo, jmpc}@fe.up.pt

Context & Motivation

1. HW/SW Partitioning

How do we determine the regions for offloading? From offloading hotspots to offloading regions, augmenting the potential for optimizations that increase the overall performance!

2. Code Optimizations for HLS

How to select regions with the overall view of impactful code transformations and optimizations?

```
void foo(int A[100], int B[100]){
    //...
}

void bar(int A[100], int B[100]) {
    for (int i = 0; i < 100; i++)
        A[i] = A[i] + B[i];
}
```

To HW

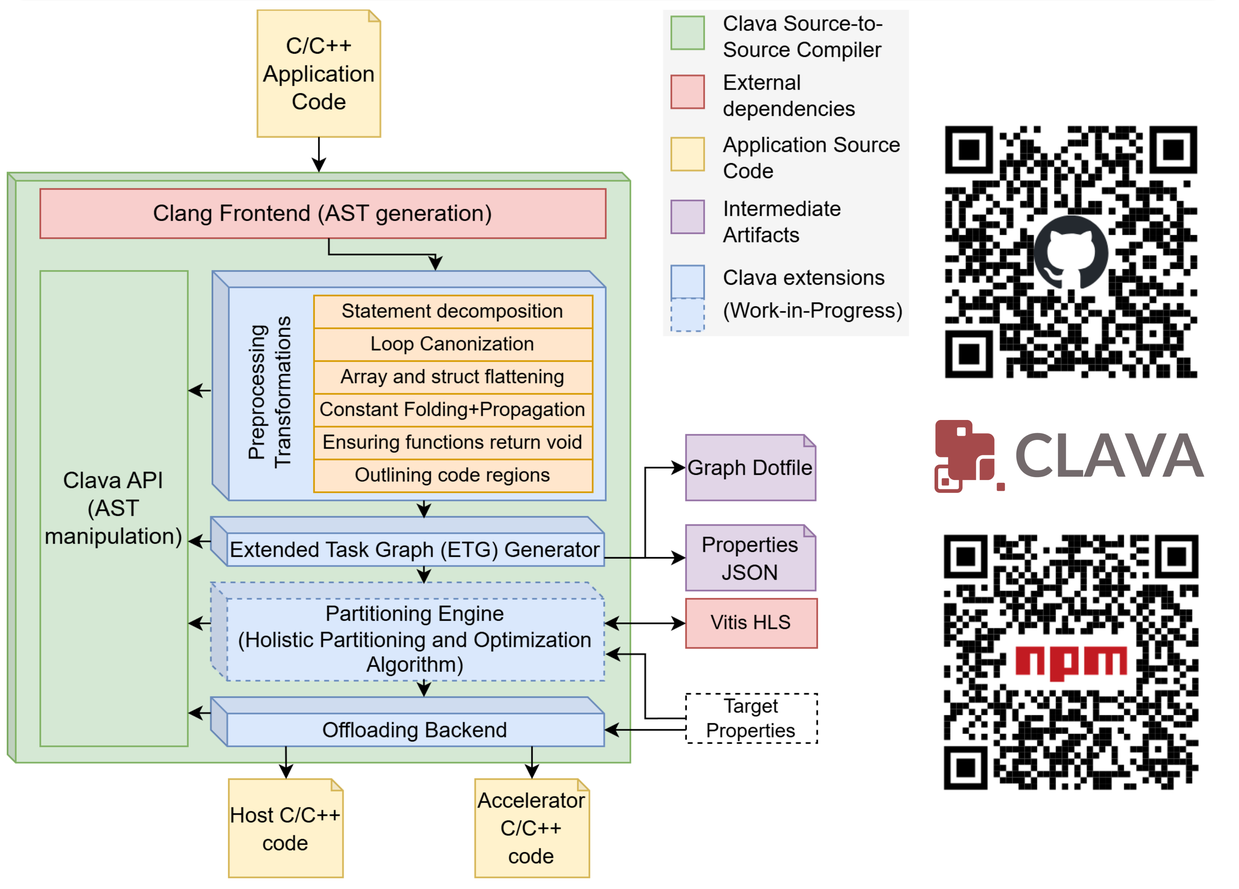
```
void bar(int A[100], int B[100]) {
#pragma HLS array_partition variable=A cyclic
    for (int i = 0; i < 100; i++) {
#pragma HLS unroll factor=20
#pragma HLS pipeline
        A[i] = A[i] + B[i];
    }
}
```

There is already extensive research about each process, when looked at in isolation. But what if they could be a single process, enriched by a holistic view of the application? We address the question:

Given a heterogeneous CPU-FPGA system, does a combined partitioning and optimization scheme for an application achieve higher speedups than those achieved by applying both processes independently?

Can the whole be more than the sum of its parts?

Our Toolchain



CLAVA

Extended Task Graph (ETG)

1. Preprocessing Transformations

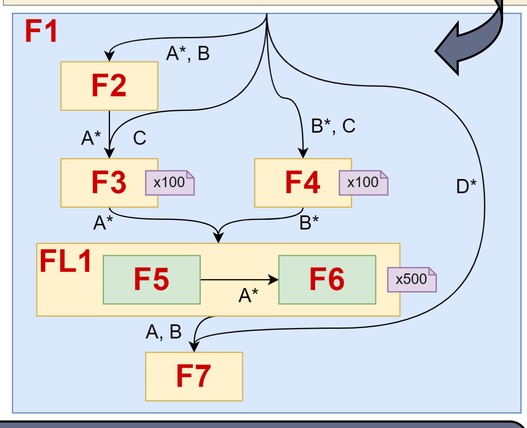
- Reduction to a C/C++ subset to simplify AST structures and improve synthesizability
- Ensuring all branching evaluations are performed over variables, and not expressions
- Outlining of every computation into individual functions, so that functions either only have computations, or only have calls to other functions

The ETG has a 1:1 mapping between a task and a function in the AST, allowing for:

- **Decreasing local granularity by fusing two tasks (function inlining at the AST-level)**
- **Increasing local granularity by splitting a task in two (function outlining at the AST-level)**

2. ETG Generation

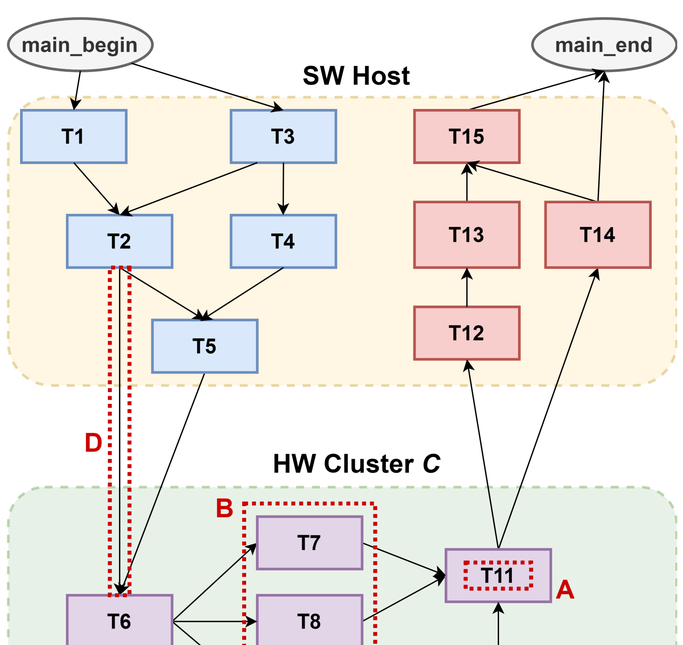
```
void F1(int *A, int *B, int *C, int *D) {
    F2(A, B);
    for (int i = 0; i < 100; i++) {
        F3(A, C[i]);
        F4(B, C[i]);
    }
    for (int i = 0; i < 500; i++) {
        F5(A, B);
        F6(A);
    }
    F7(A, B, D);
}
```



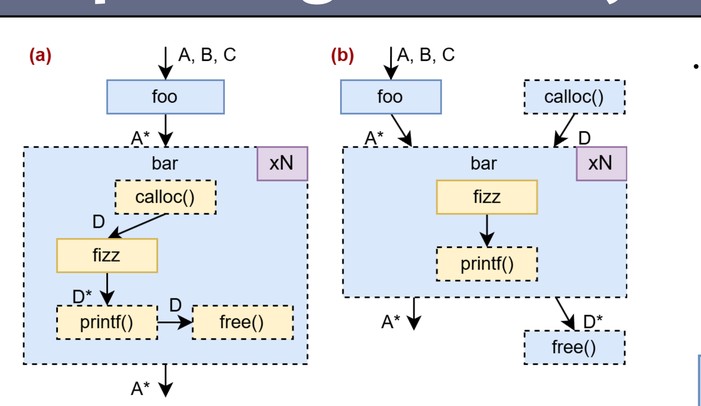
Holistic Partitioning and Optimization

- Initial clusters based on hotspot tasks, measured through CPU profiling
- Each cluster is then greedily expanded with promising tasks from outside the cluster, **in a single pass**
- **Each cluster then offers several design decisions, with optimizations enabled at multiple levels:**

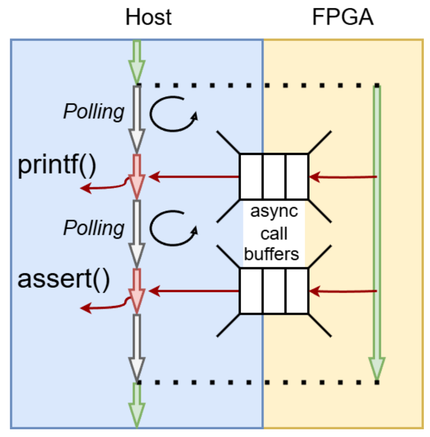
A: intra-task optim. (e.g., loop unrolling)
B: inter-task optim. (e.g., task fusion)
C: intra-cluster optim. (e.g., dataflow patterns)
D: inter-cluster optim. (e.g., FIFO communication from the CPU to the FPGA)



Improving C/C++ Synthesizability



- `malloc()` and `calloc()` hoisting removes dynamic memory allocations out of a region, making it synthesizable. Furthermore, some allocations can be converted to static arrays (subject to resource usage)



- Host functions with unused/nonexistent return values (e.g., `printf()`, `putchar()`) can be executed asynchronously by the kernel, using a "fire-and-forget" mechanism
- See FCCM poster for more details: "On improving the HLS compatibility of large C/C++ code regions"

Ongoing Work

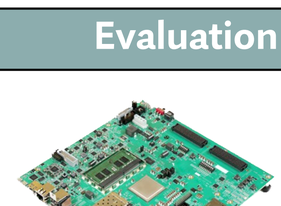
Preliminary Results

- [PACT 2023] Using an edge detection application as a motivating example, we explore holistic optimizations assisted by an ILP-based partitioning algorithm. Our approach reached a speedup of up to 28.7x when compared to an everything-to-FPGA approach
- [LCTES 2024] We propose our ETG representation for applications, and generate and characterize ETGs for the MachSuite and Rosetta suites with several metrics (e.g., number of tasks, degree of parallelism, dataflow opportunities)
- [FCCM 2025] We present two use cases of applications with synthesizability limitations, *disparity* and *llama2*. By leveraging hoisting, we can increase the synthesizable cluster from 42% to 92% for *disparity*, and from 73% to 100% for *llama2*

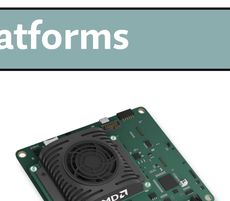
Current Progress

Code Preprocessing Transformations	Done
Test Environment Setup	Done
Task Graph Generation	Done
Characterizing Task Graphs	Done
Enabling Large Cluster Creation	Done
Refining the Holistic Algorithm	WIP
Final Evaluation Against SOTA	TBD

Evaluation Platforms



Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit



Kria KV260 Vision AI Starter Kit

Tiago Santos is partially supported by Fundação para a Ciência e Tecnologia (FCT) PhD scholarship 10.54499/2021.07324.BD