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

35th ACM SIGDA/IEEE CEDA University Demonstration at Design Automation Conference (UDDAC 2025)

June 22-25, San Francisco, California, 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

Why source-to-source for HLS?

```
int dotproduct(int *A, int *B) {
    int prod = 0;
    for (int i = 0; i < N; i++)
        prod += A[i] * B[i];
}
```

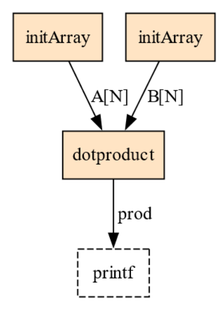
Insert instrumentation

```
clock_t start = clock();
dotproduct(A, B);
clock_t end = clock();
double time = (end-start)/
              CLOCKS_PER_SEC;
printf("%f\n", time);
```

Generate host-kernel communication

```
...
auto A_buf = xrt::bo(device, ...);
auto B_buf = xrt::bo(device, ...);

A_buf.write(A.data());
B_buf.write(B.data());

A_buf.sync(XCL_BO_SYNC_BO_TO_DEVICE);
B_buf.sync(XCL_BO_SYNC_BO_TO_DEVICE);

auto run = kernel(A_buf, B_buf);
run.wait();
...
```

Generate task graphs for HW/SW partitioning



Optimize code for HLS

```
int dotproduct(int A[N], int B[N]) {
#pragma hls array_partition variable=A
cyclic factor=32
#pragma hls array_partition variable=B
cyclic factor=32
    int prod = 0;
    for (int i = 0; i < N; i++) {
#pragma hls unroll factor=32
        prod += A[i] * B[i];
    }
}
```

Why Clava?

CLAVA

High productivity

High composability

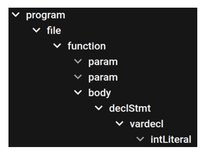
A single, flexible AST

JS TS

Write transformations using modern JIT-compiled languages (JavaScript or TypeScript)

npm

Easily reuse, distribute and combine analysis passes and transformations through NPM



AST-based transformations view the whole input program as a single cohesive entity, abstracting the notion of file/translation unit

Demo: code transformations

```
void foo(int *A){
    int x = 0;
    #pragma clava begin_outline
    int y = 5;
    for (int i = 0; i < 100; i++){
        A[i] = x + y * i;
    }
    #pragma clava end_outline
    A[0] = i;
}
```

```
void outlined(int *A, int x, int *i){
  int y = 5;
  for ((*i) = 0; (*i) < 100; (*i)++){
    A[i] = x + y * (*i);
  }
}

void foo(int *A){
  int x = 0;
  int i;
  outlined(A, x, &i);
  A[0] = i;
}
```

Function outlining

Array flattening

Struct flattening

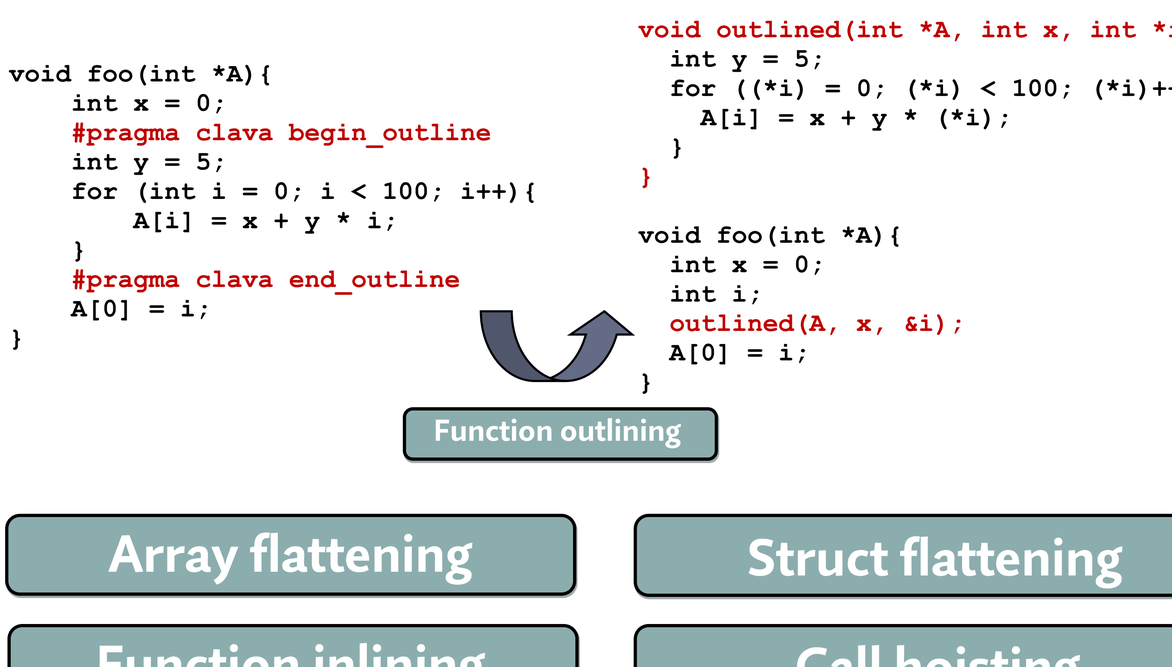
Function inlining

Call hoisting

Reduction to a C subset

Single file amalgamation

Demo: integration with HLS tools

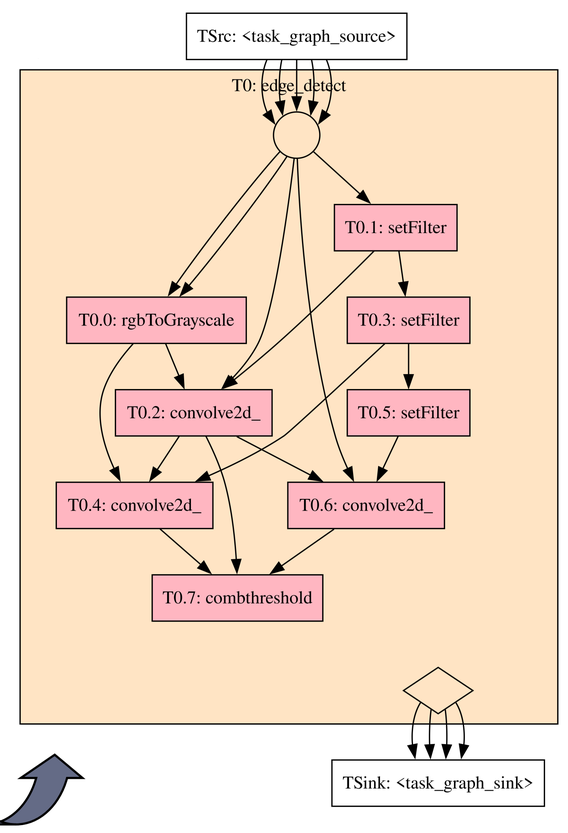


Design-space-exploration using Vitis HLS

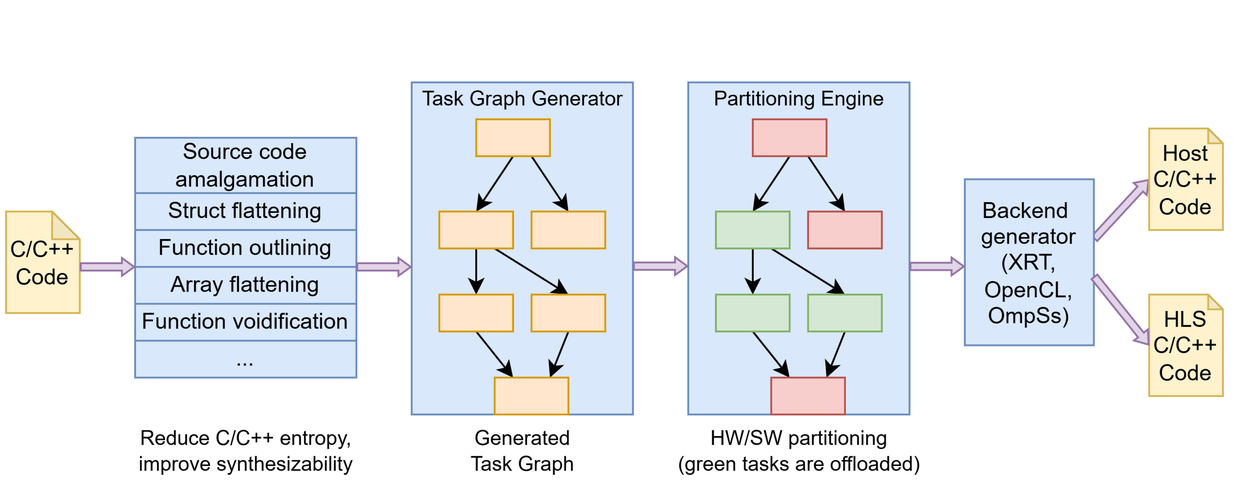
Seamless HLS optimization using ScaleHLS

Demo: generating task graphs

```
void edge_detect(int image_rgb[H][W * 3],
                 int image_gray[H][W],
                 int temp_buf[H][W],
                 int filter[K][K],
                 int output[H][W])
{
  rgbToGrayscale(image_rgb, image_gray);

  filter[0][0] = 1;
  //...
  filter[2][2] = 1;

  convolve2d(image_gray, filter, output);

  filter[0][0] = 1;
  //...
  filter[2][2] = -1;

  convolve2d(output, filter, image_gray);

  filter[0][0] = 1;
  //...
  filter[2][2] = -1;

  convolve2d(output, filter, temp_buf);

  combthreshold(image_gray, temp_buf, output);
}
```



Demo: full HW/SW partitioning flow



By combining the code optimizations, task graph generation, and HLS tools packages with a partitioning algorithm and backend generator, we can compose a full CPU-FPGA HW/SW partitioning and optimization flow for a C/C++ application

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