

# FLOWER: A Comprehensive Dataflow Compiler for High-Level Synthesis

---

Puya Amiri   Arsène Pérard-Gayot   Richard Membarth   Philipp Slusallek

Roland Leiða   Sebastian Hack

December 8, 2021

DFKI, <https://dfki.de>

Computer Graphics Lab, <https://graphics.cg.uni-saarland.de>

University of Mannheim, <https://www.uni-mannheim.de>

Compiler Design Lab, <https://compilers.cs.uni-saarland.de>

Technische Hochschule Ingolstadt, <https://thi.de>



UNIVERSITÄT  
DES  
SAARLANDES



Technische Hochschule  
Ingolstadt

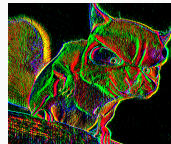
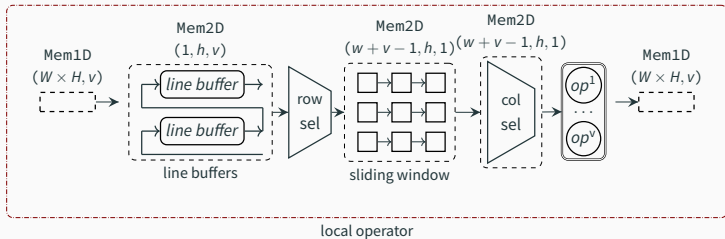
## Issues in HLS design space

- Different microarchitectures from the same untimed description
- Optimized dataflow designs are not easily generated.

*Smarter compilers?*

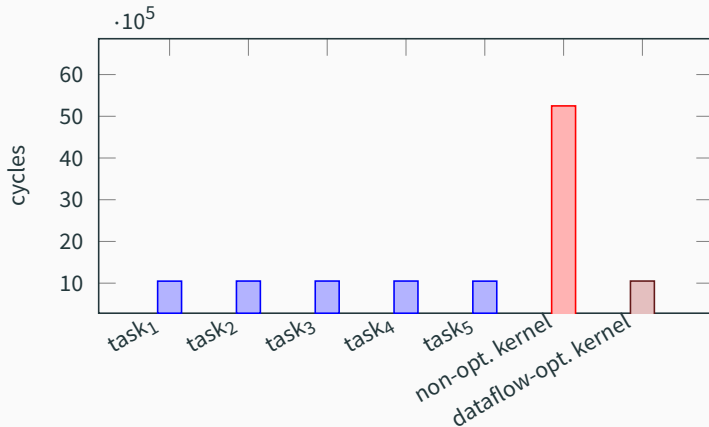
*Domain-specific languages and libraries?*

# Designing Efficient Image Processing Kernels



Individual Components are Optimized

## But what about Inter-Kernel Communications?

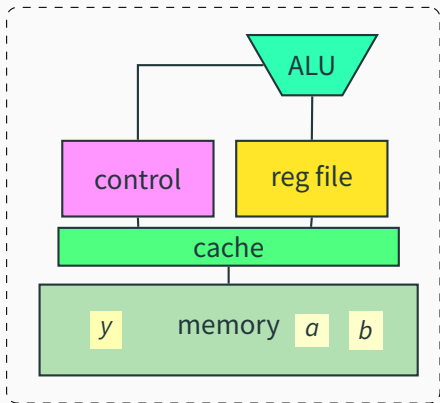


chained convolution filters

kernel execution times in cycles for a frequency of 200 MHz

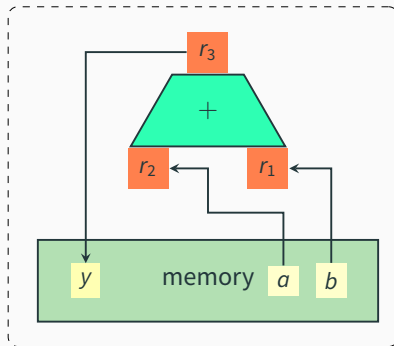
# ISA-based vs Dataflow Architectures

$$y = a + b$$



ISA-based architecture

$$y = a + b$$



dataflow architecture

## Task parallelism

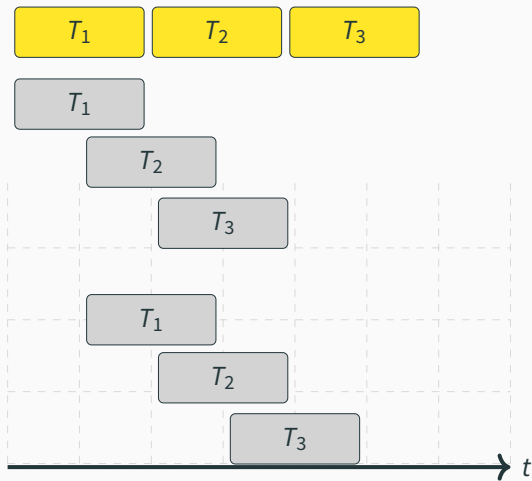
Parallelization of a program across multiple processing elements (tasks).

- represents a **dataflow model**

## Coarse-grained pipelining

- dataflow pipelining
  - realized via dataflow optimization
  - applied among loops or functions
  - offers a system of concurrent tasks

## Coarse-grained Pipelining (Dataflow Optimization)



# Need for an Effective Dataflow Compiler

## Issues

- optimizing *data transfers* is pivotal in *dataflow architectures*, but hard to achieve
- developing multi-platform designs that benefit from different levels of *parallelization and pipelining* is not an easy task
- optimizing for *host and device code* at the same time is not straightforward
- algorithm development is *tightly coupled* to implementation details



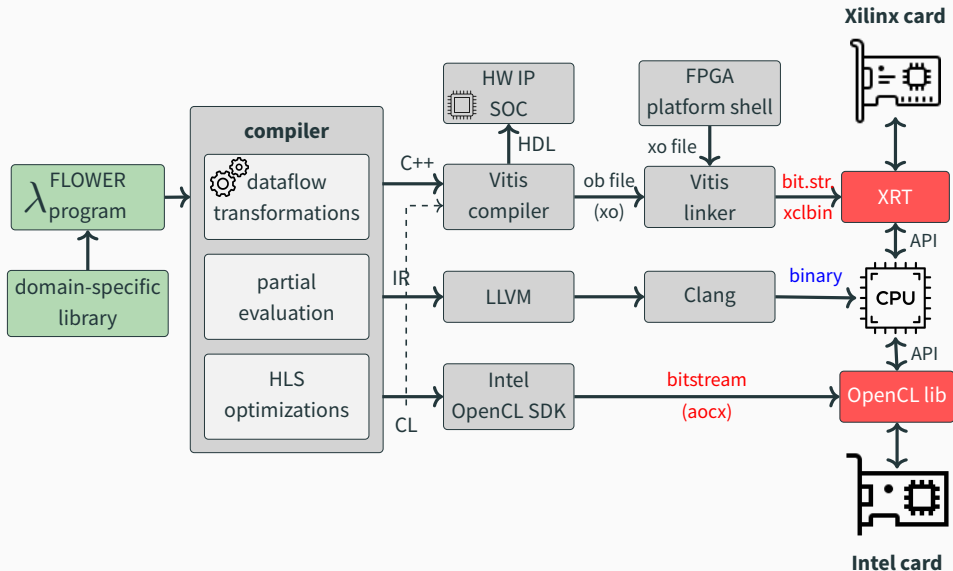
## Related works

- languages for accelerator design
  - Dahlia [Nigam et al., 2020]
- domain-specific languages
  - Hipacc [Reiche et al., 2017]
  - HeteroHalide [Li et al., 2020]
- hardware construction languages
  - LIFT [Kristien et al., 2019]

*Smart compiler + domain-specific libraries*

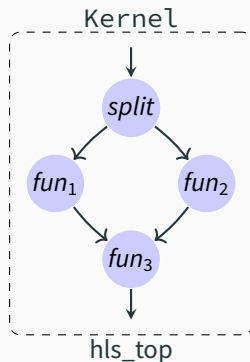
*We introduce **FLOWER**, a comprehensive dataflow compiler*

# FLOWER: Compilation and Workflow



## Describing The Flow of Stream as Tasks

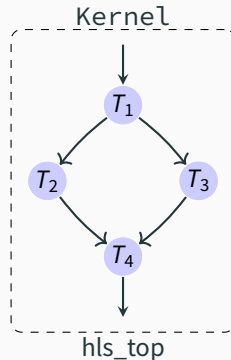
- Task1: splitting
- Task2:  $fun_1$
- Task3:  $fun_2$
- Task 4: merging by  $fun_3$



These tasks define the kernel ***hls\_top***

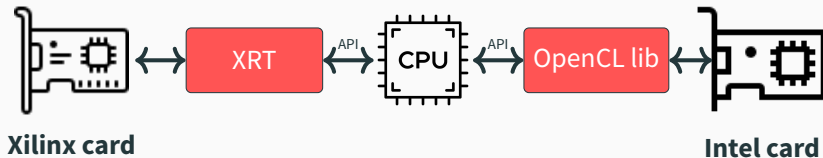
## Generated HLS Code for the Kernel hls\_top

```
void hls_top(int input_data[16], int output_data[16]) {  
    /* ... interface code ... */  
#pragma HLS top name = hls_top  
#pragma HLS DATAFLOW  
    int4_chan chan1_slot, chan2_slot, chan3_slot, chan4_slot;  
    int4_chan* chan1 = &chan1_slot;  
    int4_chan* chan2 = &chan2_slot;  
    int4_chan* chan3 = &chan3_slot;  
    int4_chan* chan4 = &chan4_slot;  
#pragma HLS STREAM variable = chan1 depth = 2  
#pragma HLS STREAM variable = chan2 depth = 2  
#pragma HLS STREAM variable = chan3 depth = 2  
#pragma HLS STREAM variable = chan4 depth = 2  
    task1(input_data, chan1, chan2);  
    task2(chan1, chan3);  
    task3(chan2, chan4);  
    task4(output_data, chan3, chan4);  
}
```

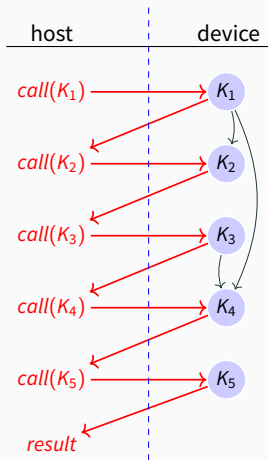


## Host-side Code Generation

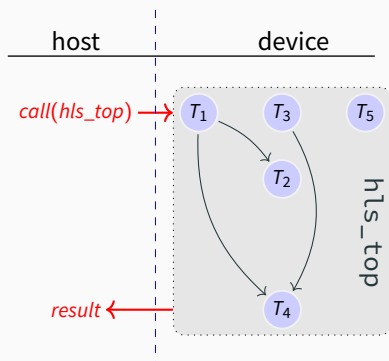
To drive the kernel a corresponding host-side code is automatically generated that calls Xilinx Runtime (XRT) APIs:



## Control Flow and Scheduling (A Higher Level View)



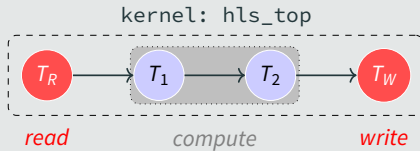
before top-level generation



after scheduling kernels as tasks

## Constructive Agglomeration of Optimizations

1. task pipelining
2. enabling burst transfers



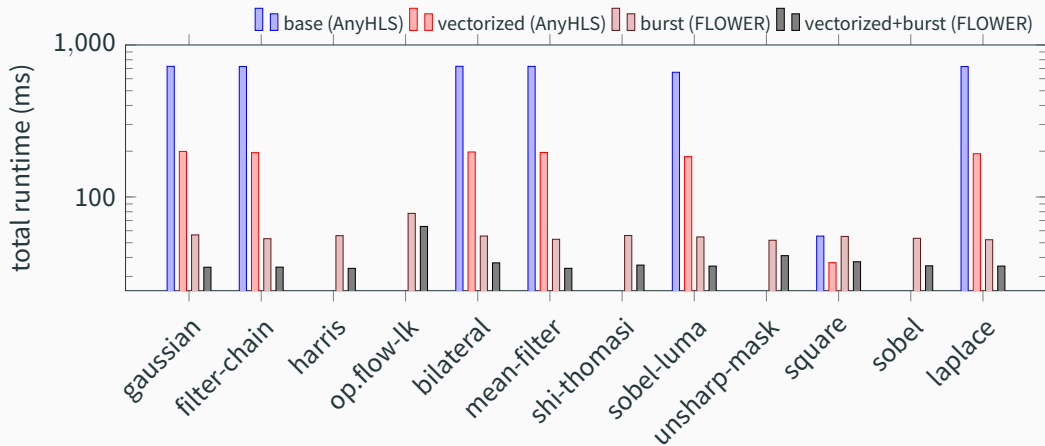
3. vectorization by replicating core operations
4. memory port width resizing
5. aligned memory port and datapath sizes → proper pipelined burst transfers

*Optimized dataflow design*

## Evaluation: FLOWER vs. AnyHLS

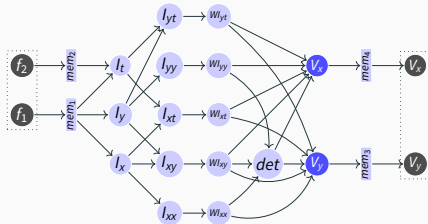
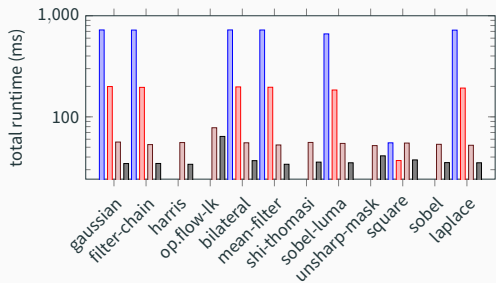
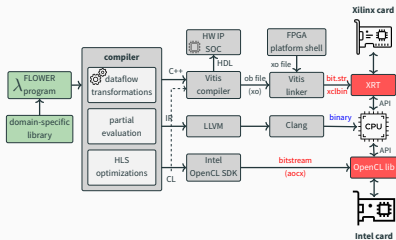
Applications are generated via FLOWER's HLS backend.

Evaluated on Xilinx Alveo U280.





# Conclusions



Thank you!

<https://anydsl.github.io>



Kristien, M., Bodin, B., Steuwer, M., and Dubach, C. (2019).

**High-level synthesis of functional patterns with lift.**


*In Proceedings of the 6th ACM SIGPLAN International Workshop on Libraries, Languages and Compilers for Array Programming*, pages 35–45. ACM.




Li, J., Chi, Y., and Cong, J. (2020).

**HeteroHalide: From image processing DSL to efficient FPGA acceleration.**

*In Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 51–57. ACM.

-  Nigam, R., Atapattu, S., Thomas, S., Li, Z., Bauer, T., Ye, Y., Koti, A., Sampson, A., and Zhang, Z. (2020).  
**Predictable accelerator design with time-sensitive affine types.**  
*In Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 393–407. ACM.
-  Reiche, O., Özkan, M. A., Membarth, R., Teich, J., and Hannig, F. (2017).  
**Generating FPGA-based Image Processing Accelerators with Hipacc.**  
*In Proceedings of the International Conference On Computer Aided Design (ICCAD)*, pages 1026–1033, Irvine, CA, USA. IEEE.  
Invited Paper.

 Özkan, M. A., Pérard-Gayot, A., Membarth, R., Slusallek, P., Leiða, R., Hack, S., Teich, J., and Hannig, F. (2020).

**AnyHLS: High-level synthesis with partial evaluation.**

*IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) (Proceedings of CODES+ISSS 2020)*, 39(11):3202–3214.