TAPA: Fast, High-Frequency, Expressive HLS Dataflow Framework

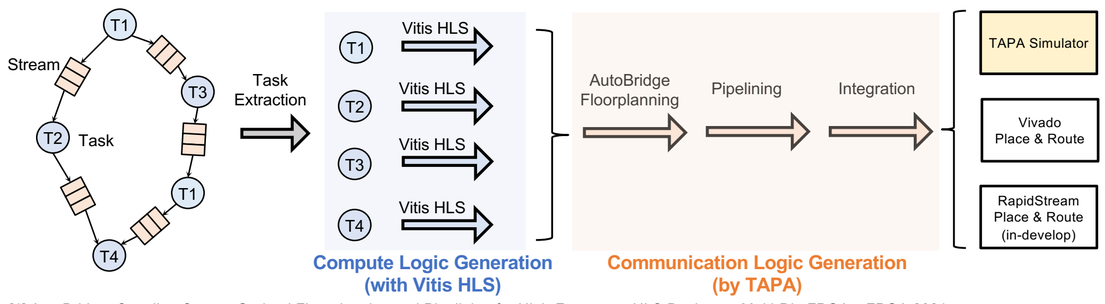
Licheng Guo, Yuze Chi, Jason Lau, Jianyi Cheng, Linghao Song, Weikang Qiao, Zhiru Zhang, Jason Cong

UCLA, Imperial College London, Cornell University

Website: https://github.com/UCLA-VAST/tapa

Overview

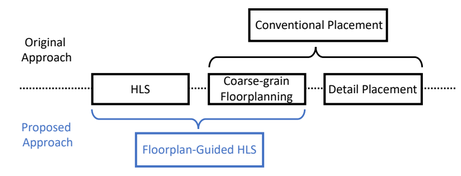
- TAPA compiles C++ dataflow programs into high-quality Verilog RTL
- TAPA is an extension of Vitis HLS and is fully compatible with Xilinx tools
- Originally published in FCCM'21, FPGA'21 (Best Paper), FPGA'22 (Best Paper)
- One-click installation, minimal migration efforts from Vitis HLS, full tech support



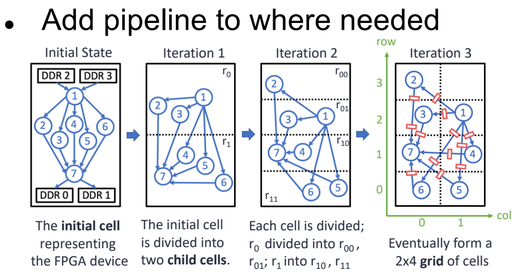
[1] AutoBridge: Coupling Coarse-Grained Floorplanning and Pipelining for High-Frequency HLS Design on Multi-Die FPGAs, FPGA 2021
[2] Extending High-Level Synthesis for Task-Parallel Programs, FCCM 2021

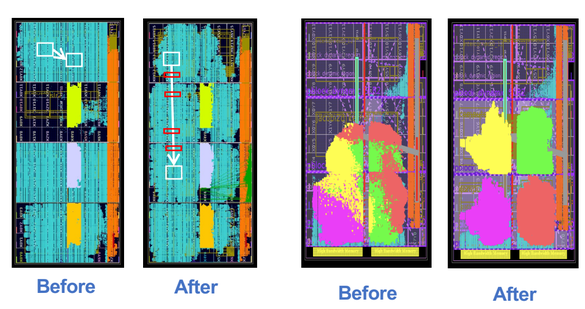
Frequency Advantages

- Co-optimizing HLS pipelining and floorplanning (FPGA'21 Best Paper)
- Avg. 2X Fmax boost (from 147 to 297 MHz) compared to Vitis HLS.



- Add pipeline to where needed

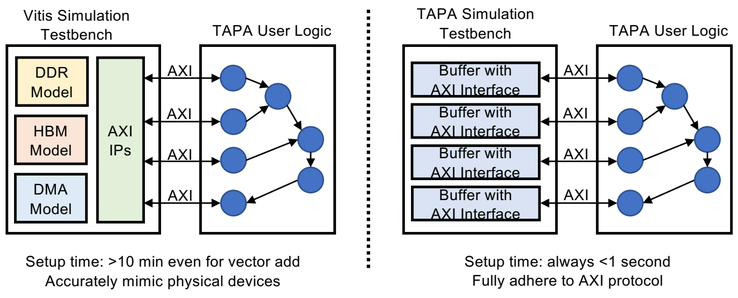




- Floorplanning => reduce local congestion
- Pipelining => remove global critical paths
- Latency Balancing => no throughput drop

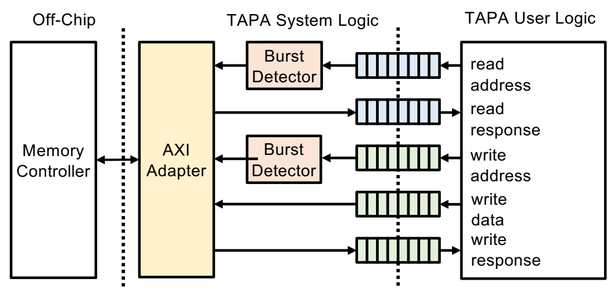
Speed Advantage

- **C-Synthesis**: TAPA invokes Vitis HLS to compile each parallel task separately
- **C-Simulation**: Optimized dataflow simulation 7X faster than Vitis HLS
- **RTL-Simulation**: Customized lightweight simulation model, 8X faster than Vitis HLS
- **Bitstream**: in progress to add RapidStream (FPGA'22 BPA), 5-7X faster than Vivado



Expressiveness Advantage

- Dedicated APIs to expose all AXI channels
  - Abstract external memory as 5 streams
  - Vitis HLS only supports array abstraction
- Runtime burst detection
  - Automatically merging individual accesses



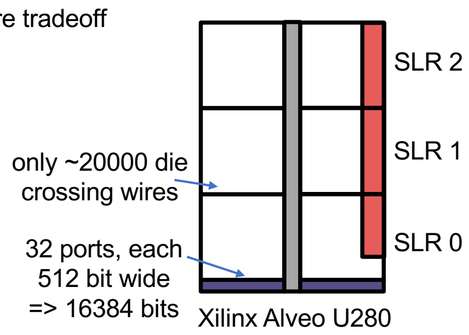
- Hierarchical task definition
- Rich APIs to interact with streams
- Templated and parameterized APIs to help quickly scale up the design, 22% less kernel code, 51% less host code on average.
- Automatic visualization. TAPA generates a dot graph showing you the topology of your design

Since TAPA generates all communication logic RTL, it is easy to define your own APIs and integrate them into the framework!

HBM-Specific Optimization

- Automatic HBM Channel Binding
  - HBM boards have 32 HBM channels and Vitis HLS users must manually select from them.
- Floorplanning Design Space Exploration
  - HBM boards have a severe wire pressure
  - Generate all pareto-optimal floorplans of the area-wire tradeoff
- Reduce area overhead of HBM IO modules
  - With Vitis HLS, using 32 HBM channels will cost 480 BRAM_36K, which is 71% of all BRAMs in SLR 0

Memory Interface	Clock/MHz	LUT	FF	BRAM
`#pragma HLS interface m_axi`	300	1189	3740	15
`async_mmap`	300	1466	162	0

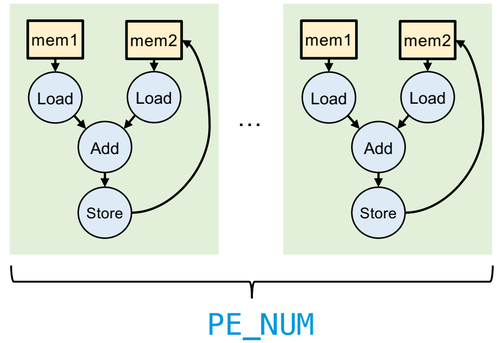


Example TAPA Code 1

- Instantiate the `Add, Load, Store` tasks for `PE_NUM` times
- Each group connects to one external memory port
- Adjust the size and parallelism of the design by changing `PE_NUM`

```
void VecAdd(mmaps<float, PE_NUM> mem1,
            mmaps<float, PE_NUM> mem2,
            uint64_t n) {
  streams<float, PE_NUM, 8> a("a");
  streams<float, PE_NUM, 8> b("b");
  streams<float, PE_NUM, 8> c("c");

  task().invoke(Load, mem1, n, a)
        .invoke(Load, mem2, n, b)
        .invoke(Add, a, b, c)
        .invoke(Store, c, mem2, n);
}
```



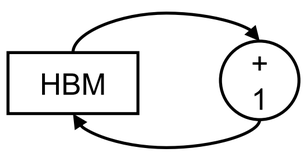
Example TAPA Code 2

```
for(int i_rd_req=0, i_rd_resp=0, i_wr_req=0; i_wr_req<n; ) {
#pragma HLS pipeline II=1

  bool can_read = !hbm.read_data.empty();
  bool can_write = !hbm.write_addr.full() && !hbm.write_data.full();

  // issue read requests to read_addr channel
  if (i_rd_req < n && hbm.read_addr.try_write(i_rd_req))
    ++i_rd_req;

  // receive read response and write out
  if (can_read && can_write) {
    Elem elem = hbm.read_data.read();
    hbm.write_addr.write(i_wr_req);
    hbm.write_data.write(elem+1);
    ++i_wr_req; ++i_rd_resp;
  }
}
```

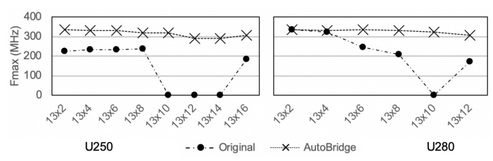


- Read from HBM, add 1, write back
- Hard to express this in Vitis HLS

Successful Cases and Future Plan

Selected Successful Applications:

- Serpens, DAC'22, 270 MHz on U280 with 24 HBM channels. The Vitis HLS baseline failed in routing.
- Sextans, FPGA'22, 260 MHz on U250 with 4 DDR channels. Vivado baseline achieves only 189 MHz.
- SPLAG, FPGA'22, up to a 4.9× speedup over the prior art, up to a 2.6× speedup over 32-thread CPU running at 4.4 GHz, and up to a 0.9× speedup over an A100 GPU (that has 4.1× power and 3.4× HBM bandwidth).
- KNN, FPT'20, achieves 270 MHz on the U280 board. Vivado baseline achieves only 165 MHz.
- AutoSA Systolic-Array Compiler, FPGA'21:



Future Plan

- Parallelize the backend placement and routing. In collaboration with Xilinx Research Lab to integrate the RapidStream project [FPGA'22]
- Add backend support for the Xilinx Versal platforms
- More front-end APIs. With increasing number of users, we are keep collecting requests and adding features. We welcome any contribution from the community!

TAPA User Group

TAPA Documentation