



POLITECNICO
MILANO 1863

DEPARTMENT OF ELECTRONICS
INFORMATION AND BIOENGINEERING

September 1st, 2025

Fast Machine Learning for Science Conference 2025

Configuration, Verification, and Optimization of Generated Accelerators

09.1.2025 | Fabrizio Ferrandi, **Serena Curzel**, Michele Fiorito, Giovanni Gozzi

Pre-Tutorial Survey

Any new people after the break? Take the survey and help us improve the presentation



Outline

- 1. Bambu HLS Quick Start Guide**
- 2. Accelerator Interface**
- 3. Hardware /Software Co-Simulation**
- 4. Kernel Optimizations**

Bambu HLS Quick Start Guide

01

Bambu HLS Quick Start Guide - Python Notebook



<https://github.com/ferrandi/PandA-bambu/tree/dev/panda/documentation/bambu101>

PandA Bambu HLS Tutorials License GPL v3




A group of little tutorials to introduce each aspect of the PandA Bambu High-Level Synthesis tool individually. Learn only what you need and take the best from the tool.

High-Level Synthesis 101

- Introduction to High-Level Synthesis  [Open in Colab](#)
- Kernel Offloading Quick Start  [Open in Colab](#)

High-Level Synthesis 102

- Integrate external IPs with HLS generated designs  [Open in Colab](#)



Bambu HLS Quick Start Guide – Bambu inputs

❖ High-Level Kernel Description

➤ Supported formats are:

```
kernel.c    kernel.cpp    kernel.f    kernel.ll
```

❖ Kernel Testbench Description

➤ Supported formats are:

```
--generate-tb=my_testbench.cpp    --generate-tb=elf:my_binary
```

❖ Target Design Requirements

```
--device-name=nx2h540tsc    --clock-period=15
```

❖ Command Line Options

```
--top-fname=my_kernel    --generate-interface=INFER    -O2
```

Bambu HLS Quick Start Guide – Bambu outputs

❖ HDL description

➤ Supported formats are:

```
my_kernel.v    my_kernel.vhdl
```

❖ Memory description

```
array_198321.mem 0_array_198321.mem
```

❖ Simulation and synthesis script

```
simulate_my_kernel.sh    synthesize_my_kernel.sh
```

Bambu HLS Quick Start Guide - Command Line Options

❖ Many option classes for:

- Compiler Optimizations
- Finite State Machine / Datapath / Memory Architecture
- Accelerator Interface
- Integer/Floating-point Math Optimizations
- Target Device Constraints
- Simulation Environment
- Info/Debug Objects Generation

Bambu HLS Quick Start Guide – Experimental Setups

Bambu can use different predefined optimizations' set

```
--experimental-setup=<setup>
```

BAMBU-AREA: optimized to reduce area utilization

BAMBU-PERFORMANCE: optimized to reduce accelerator latency

BAMBU-BALANCED: optimized for trade-off area/latency

BAMBU-AREA-MP, **BAMBU-PERFORMANCE-MP**, **BAMBU-BALANCED-MP**:
enable support to true dual port memories

Default: **BAMBU-BALANCED-MP**

Accelerator Interface

02

Accelerator Interface - Control Interface

❖ Block-level I/O protocol:

- *input* **start** : starts the computation
- *output* **done** : signal computation completed
- *input* **reset** : resets the accelerator logic

❖ Reset level:

- **low (default)**
- high

❖ Register reset type:

- **no reset (default)**
- async
- sync

Accelerator Interface - Interface Protocols

```
#pragma HLS interface port=<name> mode=<if_type>
```

□ Wires

▶ mode=none <registered>

□ Handshake

▶ mode=<valid|acknowledge|handshake>

□ FIFO/AXI-Stream

▶ mode=<fifo|axis> depth=[0-9]+

□ Array/BRAM

▶ mode=array

□ Memory Bus

▶ mode=<bus|m_axi> bundle=<bundle_name>

Bambu Documentation

<https://github.com/ferrandi/PandA-bambu/wiki/Accelerator-Interfaces>

Accelerator Interface - Memory bus Axi4

```
#pragma HLS interface port=<name> mode=m_axi  
                        bundle=<bundle_name> offset=direct
```

❖ Bambu supports Axi4 ports:

- port=name
- mode=m_axi
- bundle=<bundle_name>
- offset=direct

❖ Bambu supports multiple ports on the same bundle and can have multiple bundle at the same time.

Accelerator Interface - Memory Bus Caches

```
#pragma HLS cache bundle=<name> line_count=2**[0-9]+
```

❑ Cache Line Size (in words)

▶ `line_size=[0-9]+`

❑ Cache Ways

▶ `ways=[0-9]+`

❑ Memory Bus Size (in bits)

▶ `bus_size=[32|64|128|256|512|1024]`

❑ Cache Write Policy

▶ `write_policy=<wt|wb>`

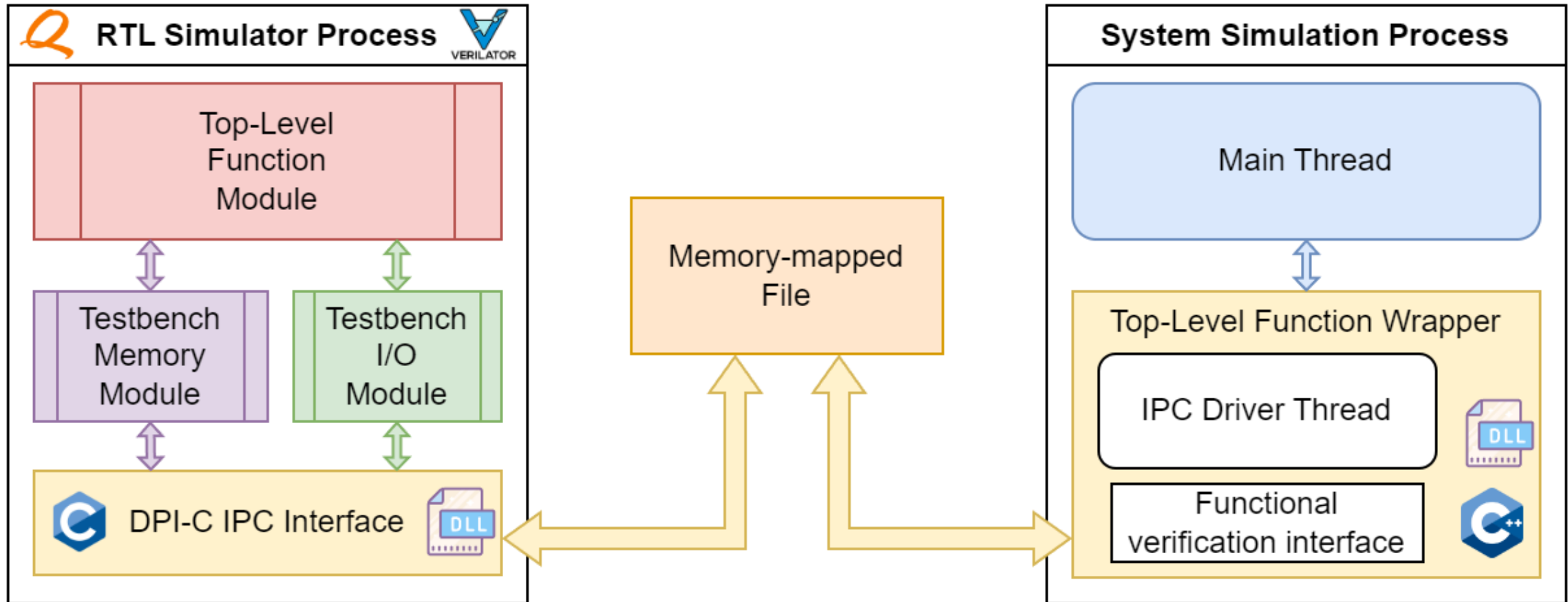
❑ Write Buffer Size

▶ `num_write_outstanding=[0-9]+`

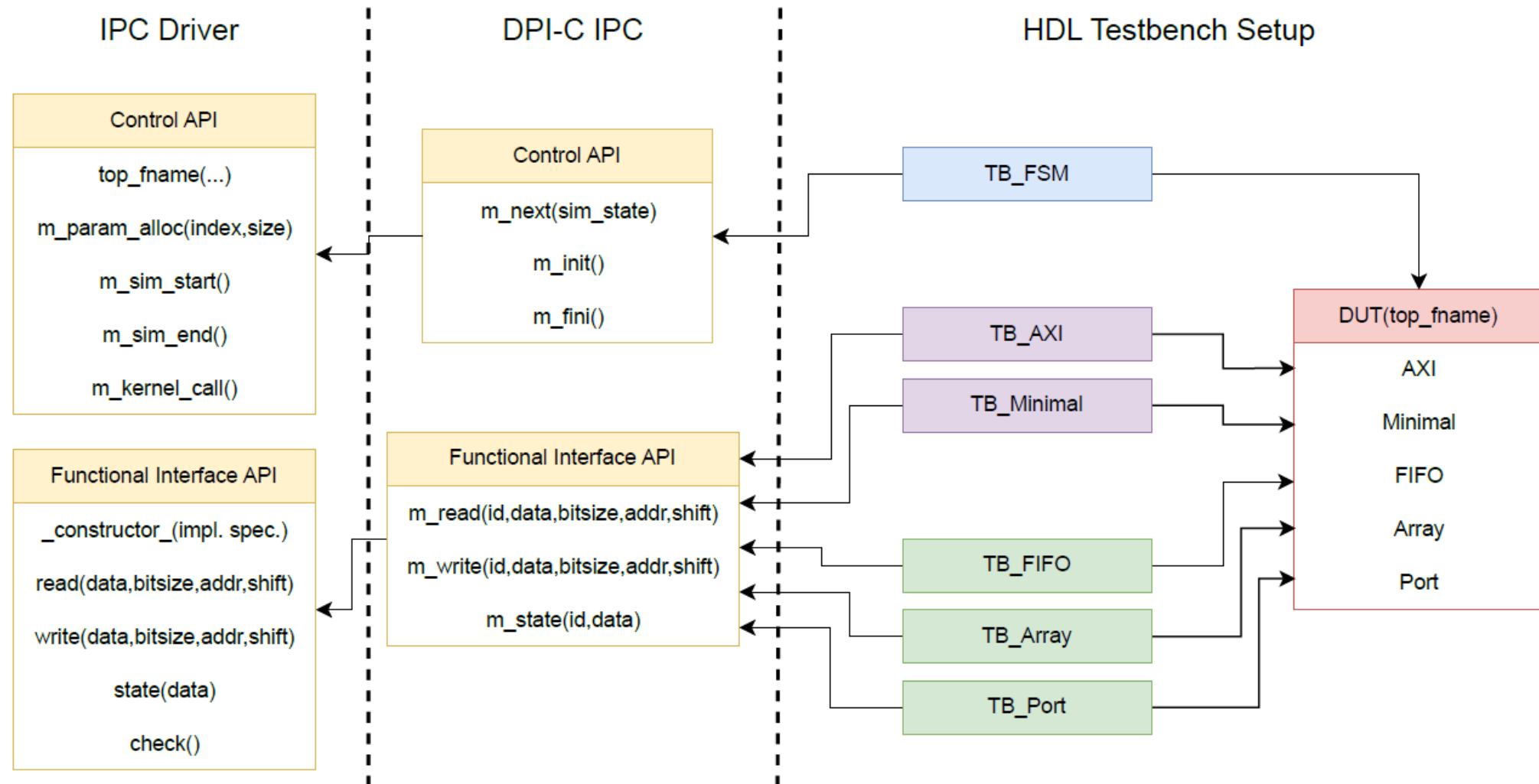
Hardware/Software Co-Simulation

03

HW/SW Co-Simulation - Software Architecture Overview



HW/SW Co-Simulation – Inter-Process Communication API



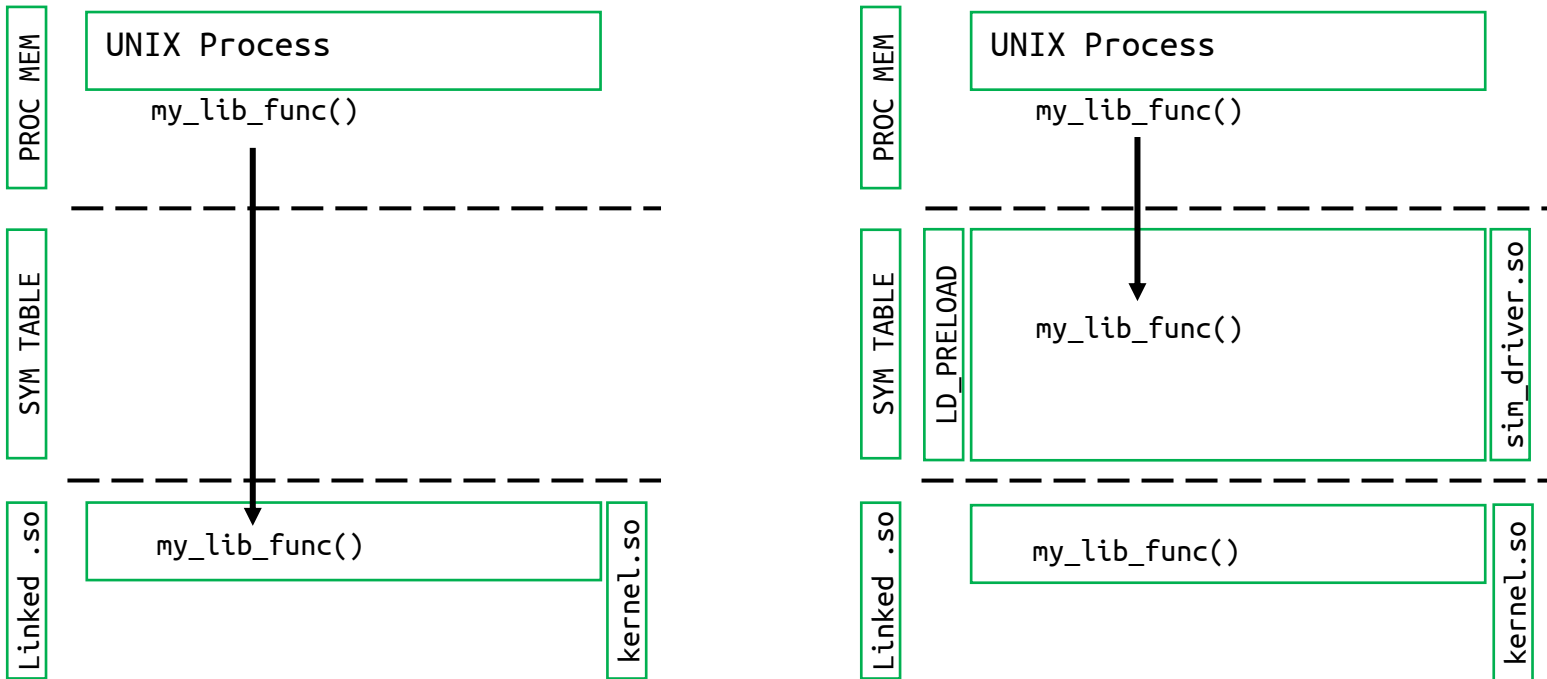
System-Level HW/SW Co-Simulation Integration

- ❑ Pre-built application may interact directly with the simulated hardware environment
 - Only requirement is the application must load the top-level function symbol dynamically (or link to the generated library at compile-time)
 - System application may be written in **ANY** language (even Python) as long as the top-level symbol is linked in C



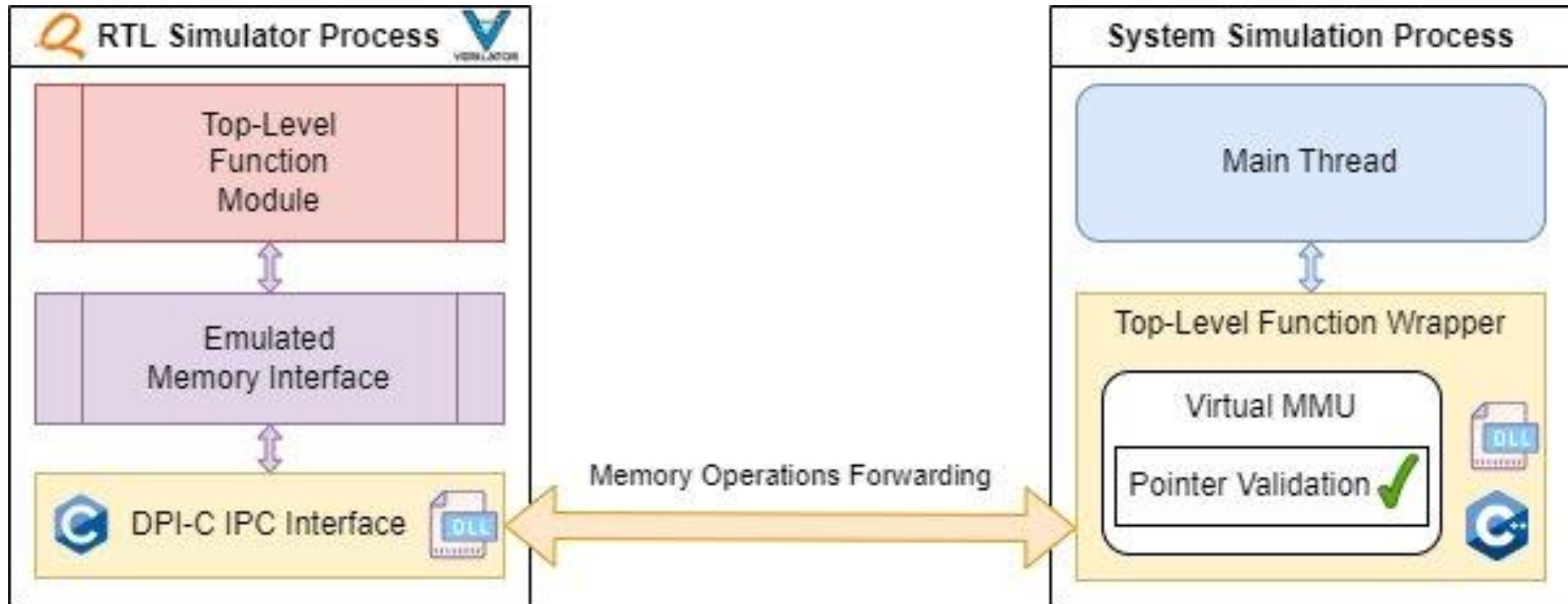
Dynamic Symbol Loading at Runtime

- ❑ Co-Simulation symbol injection is also possible in dynamically linked application
 - Dynamically linked symbols are loaded at runtime by the OS (left)
 - When a required symbol is not found in the symbol table it is loaded
 - The symbol table can be pre-loaded setting the LD_PRELOAD variable (right)



HW/SW Co-Simulation - Memory operations validation

Automated consistency checks on pointer values (*Valgrind-like*)



Experimental Results

□ Comparison with commercial HLS tool simulation environment on MachSuite**

➤ 6.1x faster simulation

➤ 29.5x cycles/s

➤ 14.8x AC/s*

Benchmark	Simulation Time (s)			Cycle/s			AC/s		
	Tool 1	Bambu	Speedup	Tool 1	Bambu	Speedup	Tool 1	Bambu	Speedup
aes	32.82	12.28	2.67x	41	247	6.05x	0.82	1.04	1.26x
backprop	23727.90	2812.11	8.44x	189	8081	42.86x	56.89	866.08	15.22x
bfs_bulk	31.00	10.97	2.83x	386	1838	4.76x	0.72	3.59	4.99x
bfs_queue	30.64	11.91	2.57x	447	1533	3.43x	0.51	2.24	4.39x
fft_strided	153.89	16.73	9.20x	632	4593	7.26x	6.54	33.99	5.20x
fft_transpose	120.34	36.47	3.30x	103	2960	28.87x	14.42	892.10	61.87x
gemm_blocked	600.37	46.54	12.90x	437	35295	80.83x	3.03	142.59	47.05x
gemm_ncubed	712.70	39.12	18.22x	184	14033	76.13x	4.65	48.97	10.54x
kmp	32.62	13.14	2.48x	5989	12483	2.08x	8.32	14.36	1.72x
md_grid	808.32	59.58	13.57x	490	11078	22.59x	15.44	427.29	27.67x
md_knn	261.14	26.43	9.88x	9	2073	219.85x	1.30	75.92	58.31x
merge	31.15	11.56	2.69x	2039	7087	3.48x	4.69	21.61	4.61x
nw	30.27	13.23	2.29x	1121	5065	4.52x	3.40	13.83	4.07x
spmv_crs	69.15	12.59	5.49x	189	540	2.85x	0.75	2.06	2.76x
spmv_ellpack	81.34	13.32	6.11x	316	1002	3.17x	1.26	3.85	3.05x
stencil2d	32.10	11.75	2.73x	3164	11314	3.58x	1.30	4.30	3.31x
stencil3d	41.64	11.87	3.51x	546	4651	8.52x	58.39	6.93	0.12x
viterbi	363.63	144.78	2.51x	810	8303	10.25x	75.57	760.55	10.06x

* AC/s = $\frac{\text{Design Slices} * \text{Simulated Cycles}}{100.000}$ per second

**Results of synthesis of Virtex 7 (xc7vx690tffg1930-3) at 100MHz simulated with AMD Xilinx XSIM

Experimental Results

□ Comparison with commercial HLS tool simulation environment on PolyBenchC (*small*)**

Benchmark	Simulation Time (s)			Cycle/s			AC/s		
	Tool 1	Bambu	Speedup	Tool 1	Bambu	Speedup	Tool 1	Bambu	Speedup
2mm	1104	18.94	58.30x	1165	9303	7.98x	2.48	36.28	14.62x
3mm	1519	21.15	71.83x	1156	14351	12.42x	3.24	79.94	24.70x
atax	119	14.12	8.43x	626	2091	3.34x	0.61	8.13	13.26x
bicg	138	14.77	9.34x	429	1020	2.38x	0.41	2.54	6.24x
covariance	843	25.47	33.10x	1695	18067	10.66x	24.98	223.85	8.96x
doitgen	1126	18.24	61.73x	1746	15381	8.81x	1.96	39.99	20.44x
durbin	92	15.32	6.01x	536	1128	2.11x	1.20	6.57	5.45x
floyd-warshall	175	185.81	0.94x	67394	78820	1.17x	68.74	125.32	1.82x
gemm	575	21.41	26.86x	751	16803	22.37x	1.10	47.72	43.21x
gemver	246	16.13	15.25x	528	2337	4.43x	1.43	15.40	10.73x
gesummv	112	15.11	7.41x	305	286	0.94x	0.25	0.52	2.03x
jacobi-1d	80	15.61	5.12x	252	2778	11.02x	0.92	23.44	25.42x
lu	2209	37.89	58.30x	1871	48064	25.69x	5.50	269.64	49.02x
ludcmp	1756	29.30	59.93x	1507	24406	16.20x	5.94	272.37	45.87x
mvt	134	13.72	9.76x	860	1154	1.34x	1.41	4.34	3.08x
nussinov	40	44.24	0.90x	51812	48669	0.94x	173.57	188.84	1.09x
seidel-2d	9594	314.69	30.49x	2736	67825	24.79x	32.20	346.58	10.76x
symm	908	19.58	46.36x	804	12508	15.56x	1.27	34.02	26.79x
syr2k	1010	22.55	44.78x	310	26662	85.88x	0.46	72.52	156.78x
syrk	401	18.69	21.45x	722	17021	23.57x	0.98	51.40	52.34x
trisolv	135	14.52	9.30x	656	1335	2.03x	0.96	5.19	5.38x
trmm	452	17.10	26.43x	3260	21895	6.72x	5.31	52.33	9.85x

➤ 27.8x faster simulation

➤ 13.2x cycles/s

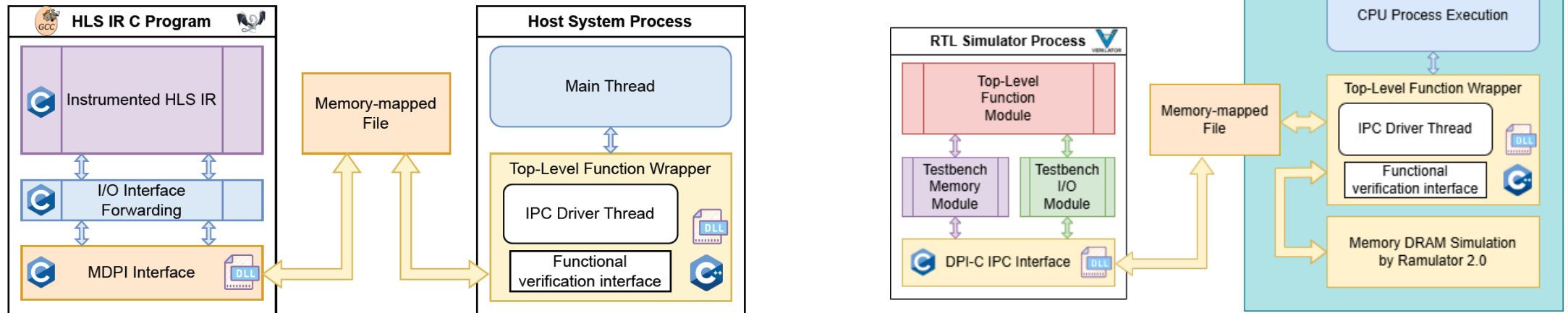
➤ 24.4x AC/s*

* AC/s = $\frac{\text{Design Slices} * \text{Simulated Cycles}}{100.000}$ per second

**Results of synthesis of Virtex 7 (xc7vx690tffg1930-3) at 100MHz simulated with AMD Xilinx XSIM

Current & Future Development

- ❑ The proposed co-simulation environment enabled further research on:
 - Faster C-based simulation environment (accepted at ICCAD '25)
 - Timing-accurate RAM simulation (under development)
 - Gem5 simulation integration (want to contribute?)



Kernel Optimizations

04

Kernel Optimizations

- ❑ Performance and area of the generated accelerators can be improved by tuning the design flow
 - ▶ GCC/CLANG target-independent optimizations
 - ▶ Bambu IR hardware-oriented optimizations
 - ▶ HLS algorithms (allocation, scheduling, binding)

- ❑ Best design flow for all accelerators does not exist
 - ▶ Trade off between area and performance
 - ▶ Effects of the single optimizations can be different on different input applications

- ❑ Default optimization flow:
 - ▶ **Balanced** area/performance trade-off

Kernel Optimizations

❖ Front-End Compiler Optimizations

- Optimization presets
- Function inlining
- Loop unrolling

❖ Hardware-Oriented Optimizations

- Bambu HLS Transformations Engine
- Integer/Floating-point arithmetic optimization

❖ Bambu HLS Algorithms

- Functional/Loop pipelining
- System of Difference Constraints

Kernel Optimizations - Front-end compiler

- ❑ Target-independent optimizations only
- ❑ Users can tune this part of the flow:

- ▶ Selecting optimization level:

```
-O0 or -O1 or -O2 or -O3 or -Os
```

- ▶ Enabling/disabling single optimization:

```
-f<optimization> -fno-<optimization>
```

- ▶ Tuning parameters: --param

```
--param <name>=<value>
```

Kernel Optimizations - Impact of optimization level presets

- ❖ Bambu defaults are used changing front-end compiler optimization level only

Optimization level	Clock cycles	LUTs
O0	15764	11675
O1	7892	11052
O2	4679	10276
O3	3854	15679
O3 all inline	1327	13550

- ❖ -O3 is not necessarily the best choice
 - Can improve **performances**
 - Can increment **area**

Kernel Optimizations – Intermediate Representation

- Hardware-oriented optimizations

- Many optimization techniques:
 - ▶ Single instruction optimizations
 - ▶ Multiple instruction optimizations
 - ▶ Rewriting IR
 - ▶ Restructuring of Control Flow Graph

- Same optimization can be applied many times
 - ▶ Fixed point iteration optimization flow

Kernel Optimizations - IR Analysis

- ❑ Collects information over IR to be used by other optimizations and HLS back-end
- ❑ **Data flow** analysis
 - ▶ Scalar: based on SSA
 - ▶ Aggregates (i.e. Front-end+Bambu alias analysis)
- ❑ **Graphs** Computation
 - ▶ Call Graph, CFG, DFG, ...
- ❑ **Loops** identification
- ❑ **Bit Value** Analysis
 - ▶ Compute for each SSA variable which bit are used, which are fixed, which are useless
- ❑ **Range** Analysis

Kernel Optimizations - Single Instruction Optimization

□ IR lowering

make single instructions more suitable to be implemented on FPGA

- ▶ Expansion of multiplication by constant
- ▶ Expansion of division by constant
- ▶ Etc.

□ Bit Value Optimization

exploit information from previous IR analyses to make bitwise optimizations

- ▶ Shrink operations to the only significant bits

Kernel Optimizations – Multiple Instructions Optimization

- ❑ **Common Subexpression Elimination**
- ❑ **Dead Code Elimination**
- ❑ **Extract pattern** (e.g., three input sum)
- ❑ **LUT transformations**
 - ▶ Merging multiple Boolean operations into a single LUT-based operation
- ❑ **Conditional Expression** Restructuring
- ❑ **Commutative Expression** Restructuring

Kernel Optimizations – Rewriting IR

□ Struct assignment

- ▶ Replaced with memcpy call

□ Floating point operations

- ▶ Replaced with function calls

□ Integer divisions

- ▶ Replaced with function calls

Kernel Optimizations – Restructuring of Control Flow Graph

- ❑ **Merging** of conditional branch
 - ▶ Creation of multiple target branch
- ❑ **Basic Block Manipulation**
 - ▶ Remove (empty, dead, ...)
 - ▶ Split
 - ▶ Merge
- ❑ **Code motion**
- ❑ **Speculation**

Kernel Optimizations – Integer Division Algorithms

- User can control integer division implementation:

```
--hls-div=<implementation>
```

- Available implementations:

- ▶ `none`: HDL-based pipelined restoring division
- ▶ `nr1`: C-based non-restoring division with unrolling factor equal to 1
- ▶ `nr2`: C-based non-restoring division with unrolling factor equal to 2
- ▶ **NR** (default): C-based Newton-Raphson division
- ▶ `as`: C-based align divisor shift dividend method

Kernel Optimizations – Floating-point support

□ Possible ways of implementing floating point ops:

- ▶ **Softfloat (default)**: customized faithfully rounded (nearest even) implementation

```
--soft-float
```

- ▶ Subnormals: subnormal numbers support can be enabled through

```
--fp-subnormal
```

- ▶ Softfloat GCC: GCC soft-based implementation

```
--soft-fp
```

- ▶ FloPoCo generated VHDL modules

```
--flopoco
```

Kernel Optimizations – Libm versions

- HLS flow exploited to generate hardware implementation of soft-defined libm functions

- Two different versions of libm are available
 1. **Faithfully rounded libm** (default)
 2. Classical libm built integrating existing libm source code from glibc, newlib, uclibc and musl libraries.
 - Worse performances and area

Kernel Optimizations – System of Difference Constraints

- ❑ Global scheduling based on ILP formulation
- ❑ Results are exploited to perform
 - ▶ Speculation
 - ▶ Code Motion

+ Improve performances of accelerators

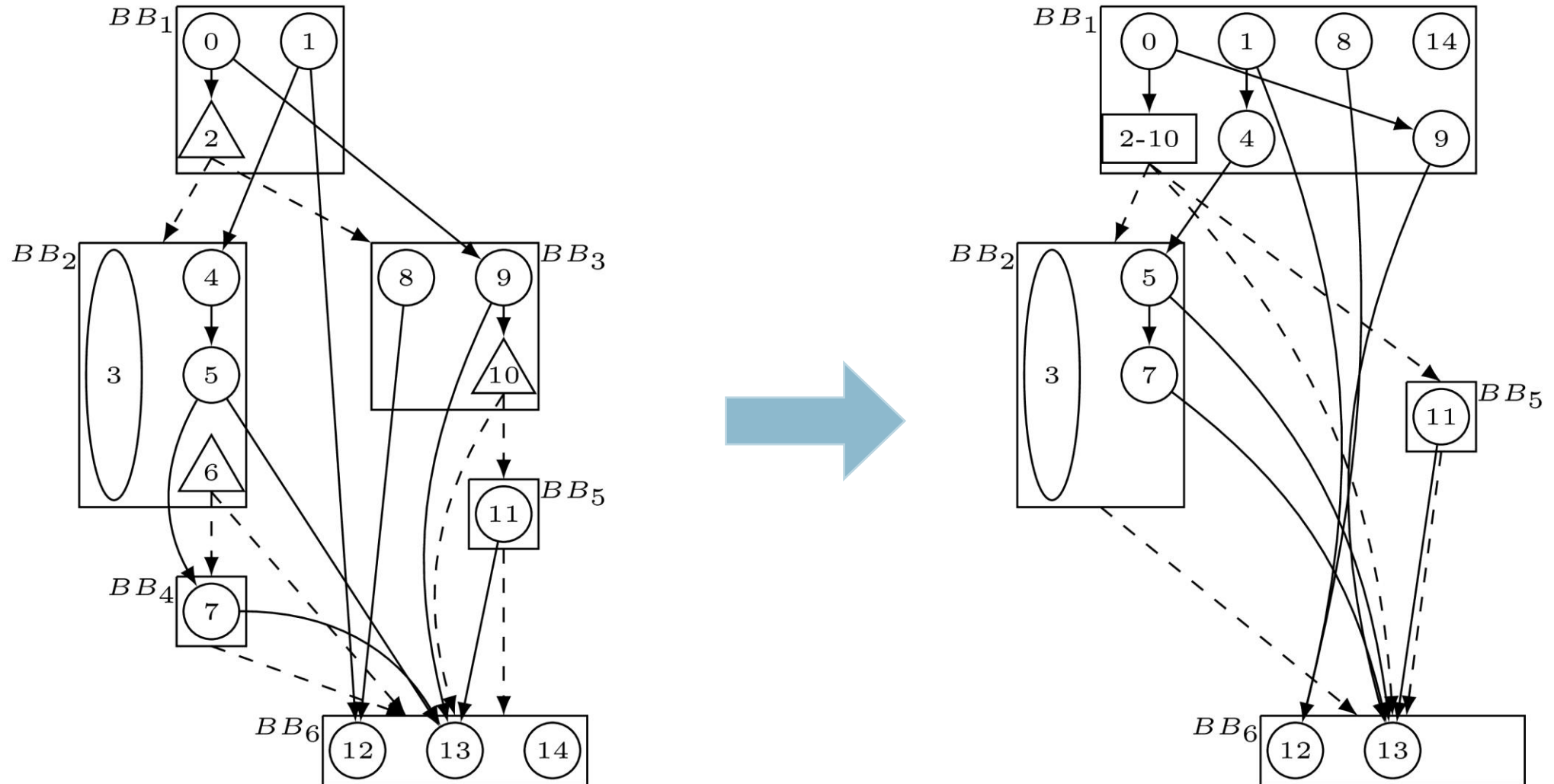
- Potentially increment area of accelerators

- Increase High Level Synthesis time

```
--speculative-sdc-scheduling
```

Kernel Optimizations - Example of scheduling

op'



Questions

Bambu HLS Open-Source Project

<https://github.com/ferrandi/PandA-bambu>

Bambu HLS Documentation

<https://docs.bambuhls.eu>

Thank you!

Bambu HLS Releases

<https://release.bambuhls.eu>



This work has been partially supported by the Spoke 1 "FutureHPC & BigData" of the Italian Research Center on High-Performance Computing, Big Data and Quantum Computing (ICSC) funded by MUR Missione 4 - Next Generation EU (NGEU).