POLITECNICO MILANO 1863
DEPARTMENT OF ELECTRONICS INFORMATION AND BIOENGINEERING

Bambu: Open-Source HLS for Automated FPGA/ASIC Acceleration

CGO 2026 – SODA tutorial

31.01.2026 | Serena Curzel

Outline

1. **Bambu HLS**
   Synthesis flow, inputs, outputs
   Quality of results
   Hands-on exercises
2. **Verification**
   Hands-on exercises
3. **Custom data formats: TrueFloat**
   Hands-on exercises
4. **Interfaces and Caches**
   Hands-on exercises
5. **Compiler-driven innovation: Bambu + MLIR**
6. **Multi-threaded accelerators: SPARTA**

Tutorial Colab Notebook

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

Bambu HLS

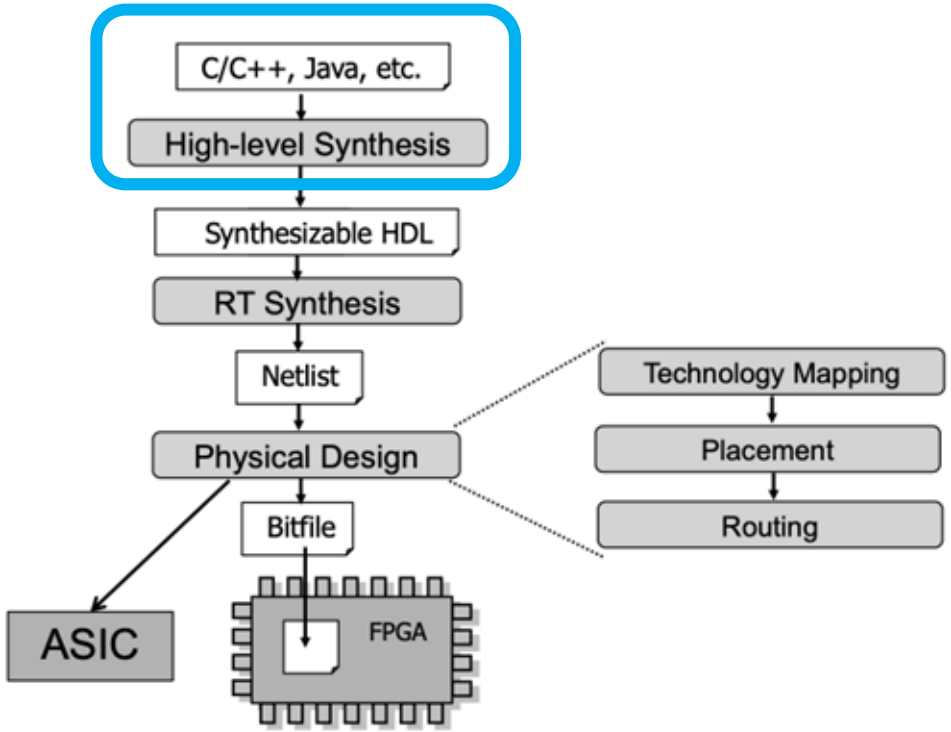
01

Bambu: a modern open-source HLS tool

High-Level Synthesis:

- High-level behavioral design through sw programming languages
- Automated translation to Register-Transfer Level
- Like compilers, but for hardware

→ Increased performance of FPGA/ASIC made available to software developers without hardware design expertise



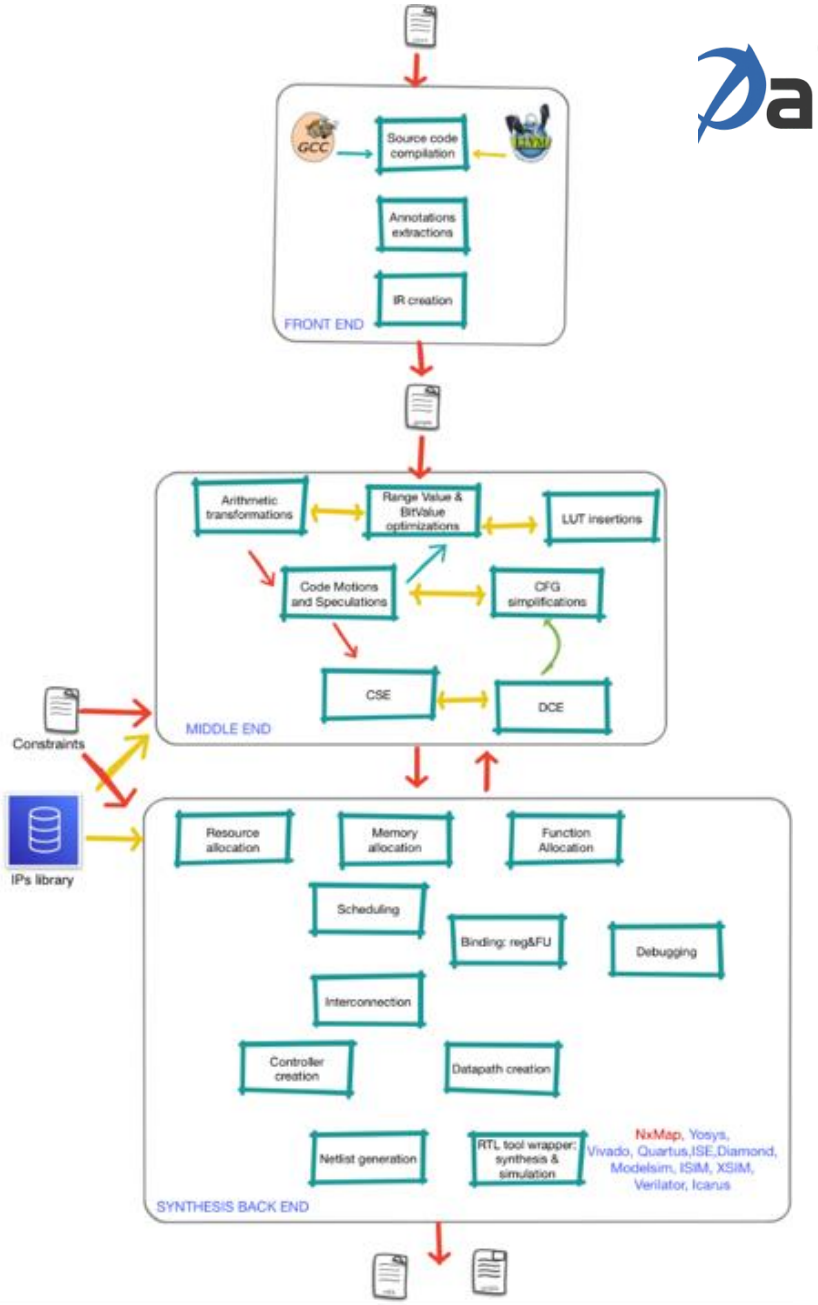
Bambu: a modern HLS tool

Bambu HLS

- Developed and maintained by the PandA group at Politecnico di Milano since 15+ years
- **Open-source**

Synthesis flow:

- Compilation and optimization of the intermediate representations (IR)
- Allocation of resources
- Scheduling of operations
- Binding operations to resources
- Generation of synthesizable RTL



F. Ferrandi et al., Invited: Bambu: an Open-Source Research Framework for the High-Level Synthesis of Complex Applications, 58th ACM/IEEE Design Automation Conference (DAC), 2021.

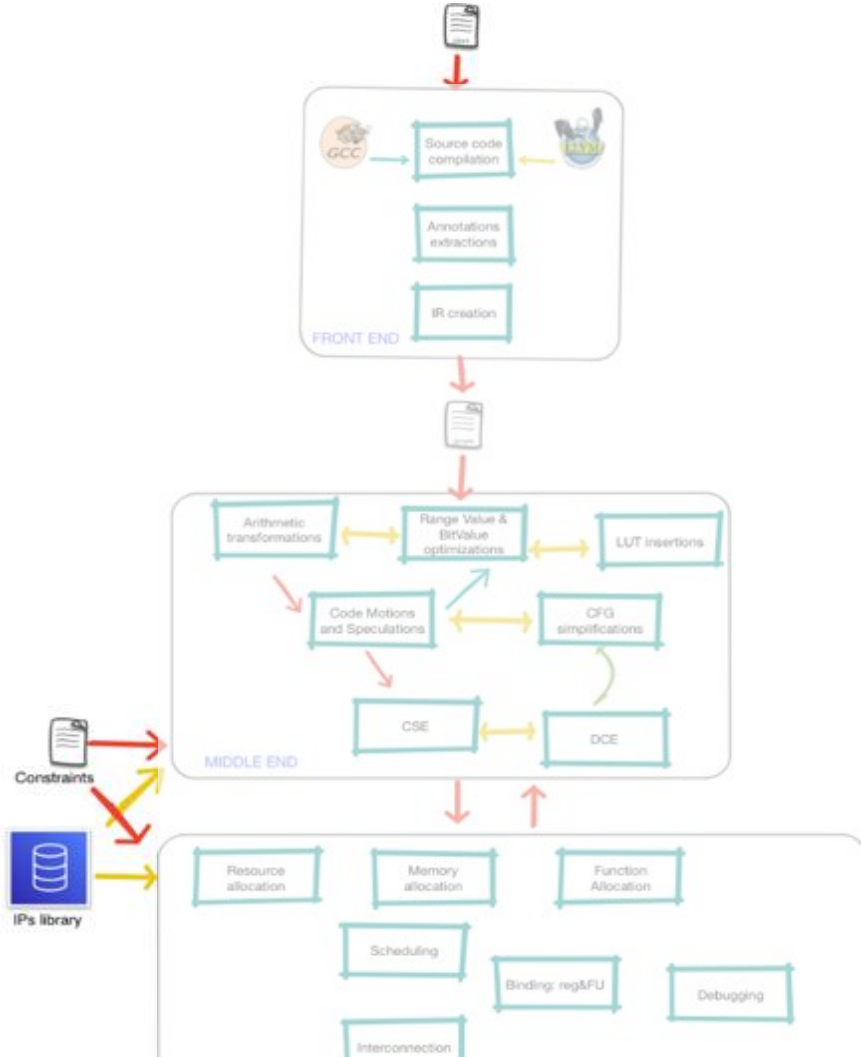
Bambu: a modern HLS tool

Inputs:

- C/C++/LLVM IR
  - Frontend compilation through GCC or Clang
  - Complete support for ANSI C (except for recursion)
    - Pointers, user-defined data types, etc..
  - Support for STL-like C++ structures
  - Support for fixed-point HLS types

- Command-line optimization directives and constraints, pragmas

- Library of functional units *characterized* for each target
  - Performance estimation essential for scheduling and optimization



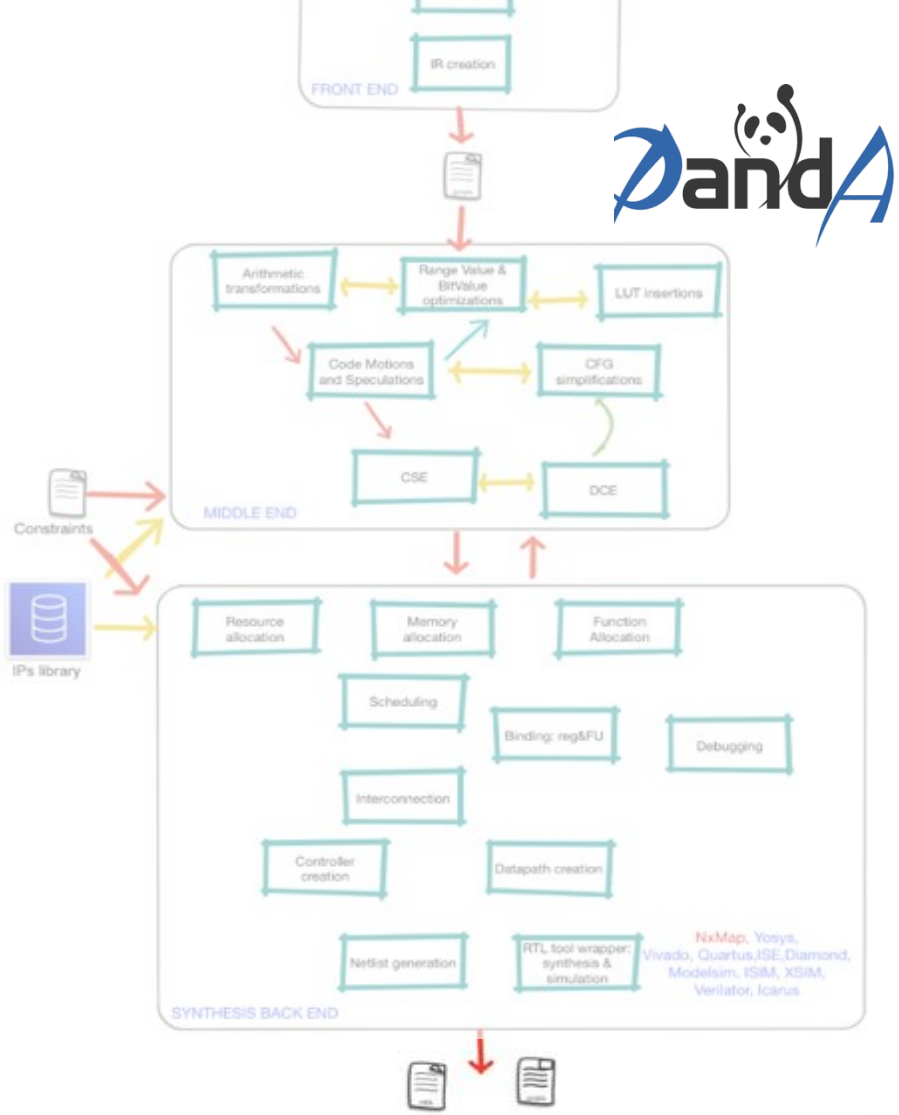
Bambu: a modern HLS tool

Outputs:

- Synthesizable Verilog or VHDL
  - Target-specific
  - FPGA targets from AMD/Xilinx, Intel/Altera, Lattice, NanoXplore
  - ASIC targets through OpenROAD (Nangate 45, ASAP7)

- **Automatically generated testbench**
  - RTL simulation with Verilator/Modelsim/XSIM

- Scripts for logic synthesis/implementation
  - Vivado/Quartus/Diamond...

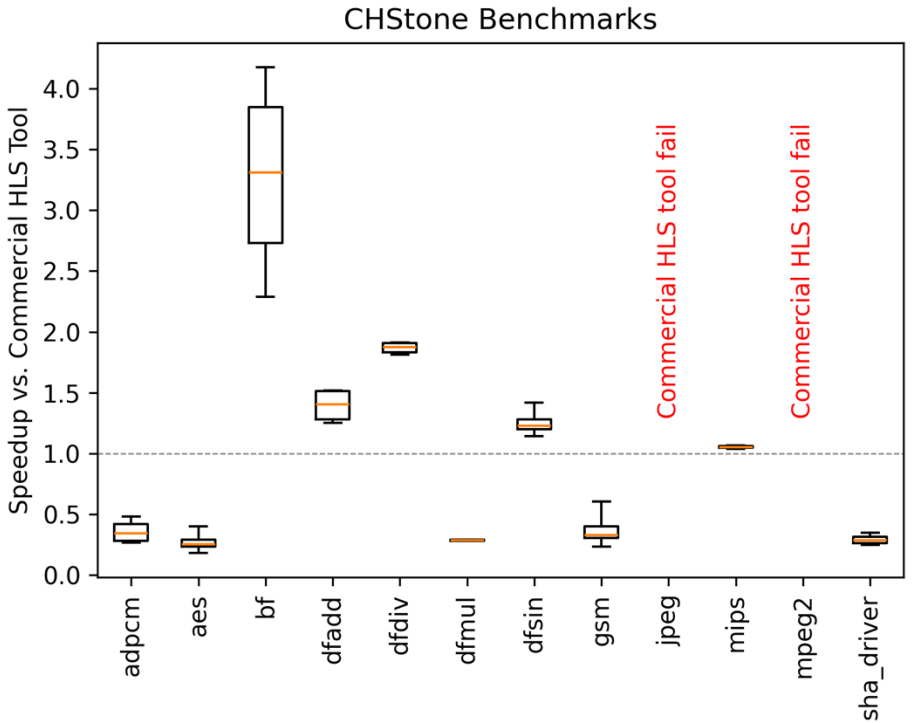


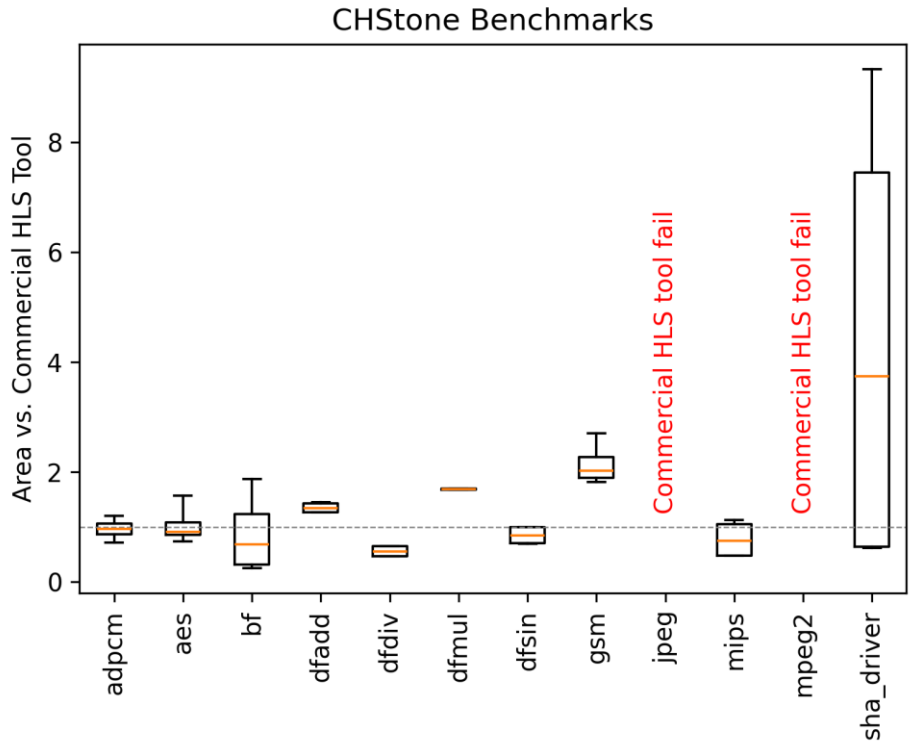
Bambu quality of results: CHStone

Speedup and area consumption over commercial HLS tool across different Bambu configurations.

*Latency is measured in ns (clock cycles * achieved period post-implementation). > 1 is better.*

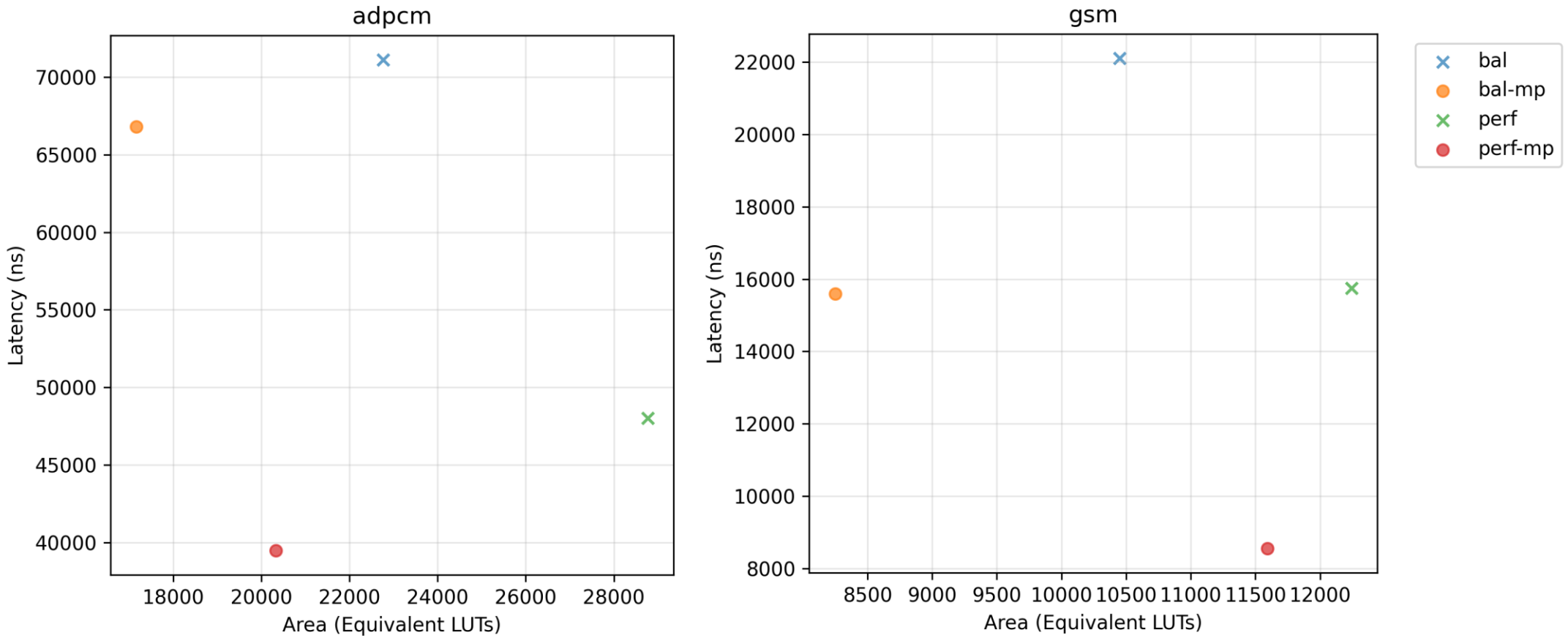
*Area is measured in Equivalent LUTs (BRAMs * 40 + DRAMs * 40 + DSPs * 40 + Registers * 0.5 + LUTs). < 1 is better.*





Bambu quality of results: CHStone

Pareto plots (Latency vs. Area) for selected benchmarks. *Points marked with x are dominated.*



Bambu quality of results

Other benchmarks + raw data at: https://github.com/ferrandi/PandA-bambu/wiki/Quality-of-Results

🐼 Bambu HLS Quality of Results

This page reports results obtained with Bambu HLS on widely used benchmark suites. For each benchmark suite, you will find:

- **Summary plots**: overall performance against other tools (when available).
- **Trade-off analysis**: Pareto plots for latency vs. area.
- **Detailed results**: tables with full post-implementation metrics for each accelerator benchmark.

Notes:

- Results obtained with an internal development version of Bambu (July 2025).

Benchmark Suites

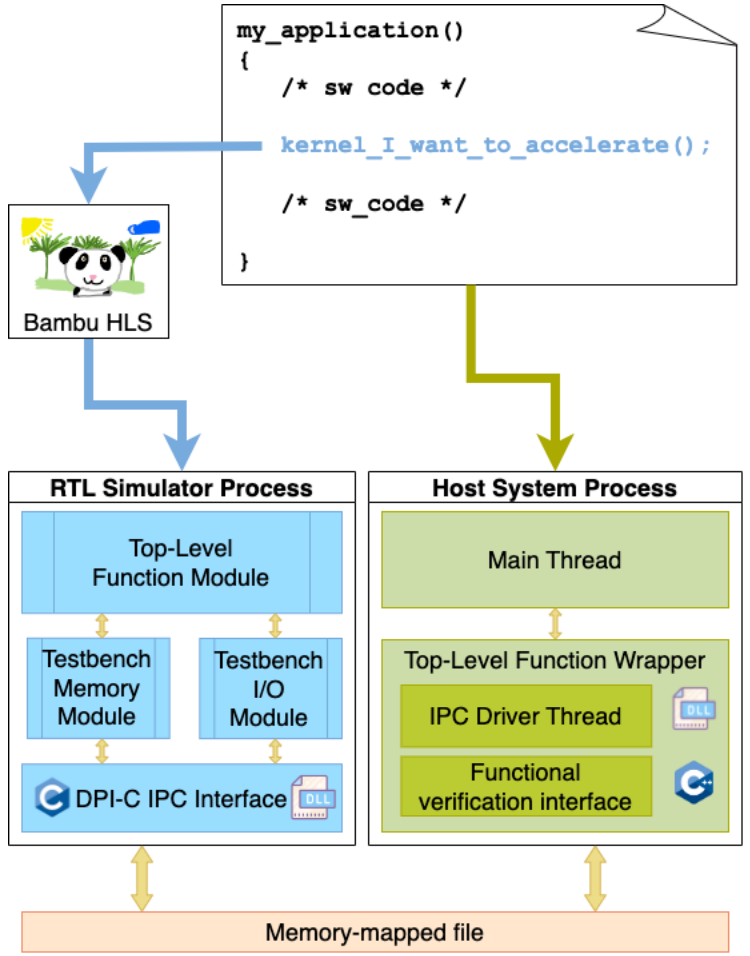
- CHStone
- MachSuite
- PolyBench

Verification

02

Accelerator – host co-simulation

- HW/SW co-simulation framework
  - RTL testbench and infrastructure automatically generated by Bambu
  - Reports number of clock cycles with the given input data
  - Compares HW results with SW, or defers verification to the testbench code

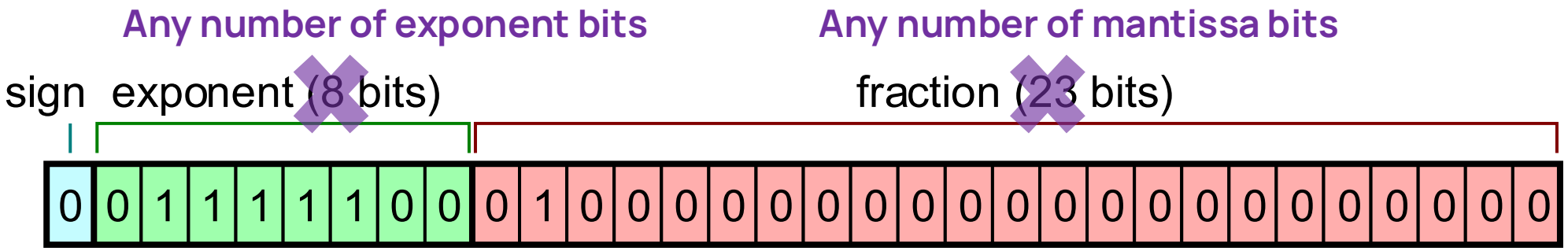


M. Fiorito et al., Augmented Co-Simulation for Fast Functional and System-Level Verification of HLS Accelerators, 44th IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2025.

Custom data formats: TrueFloat

03

Custom floating-point formats



But also:

- Custom exponent bias
- Rounding to nearest even or truncation
- IEEE exceptions, saturation, or overflow
- Hidden one, or not
- Subnormals, or not
- Dynamic or static sign

Custom floating-point formats

Opportunities

- Fewer bits
- Tailored format

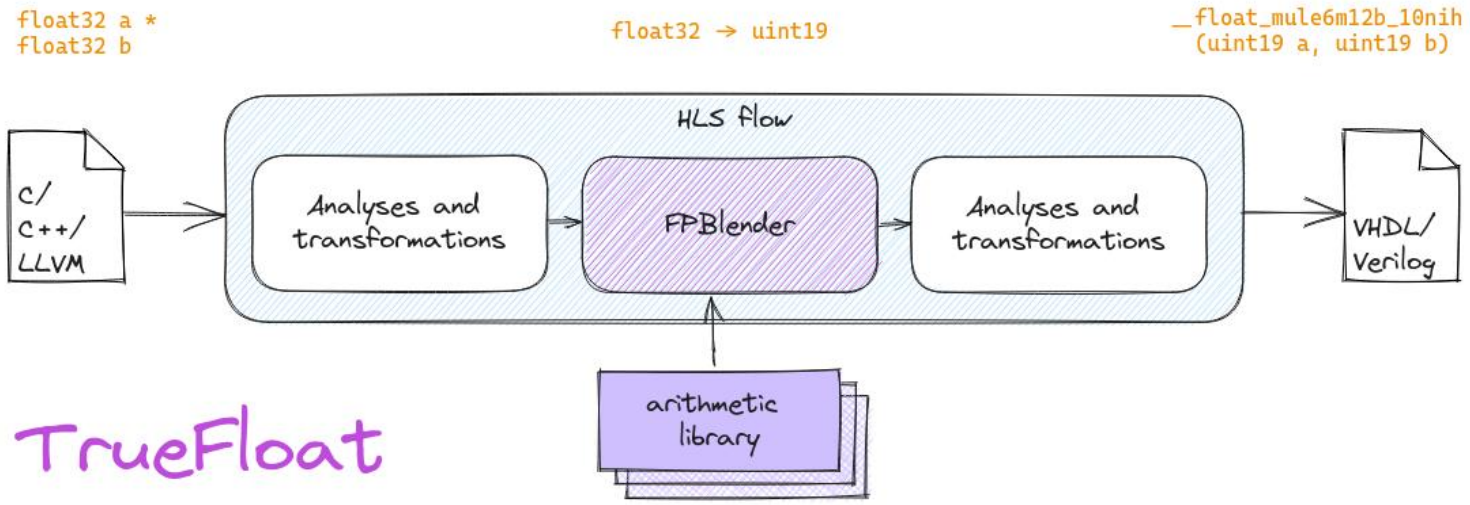
Improved Quality of Results (QoR)

- Faster functional units
- Smaller logic
- More accelerators in parallel

Challenges

- Assessing impact on accuracy
- Design methodology (libraries? HLS?)
- Wide design space to explore

TrueFloat



- Users specify the required format, Bambu generates optimized arithmetic units
- Integrated within the HLS flow -> QoR higher than inserting «black box» IPs
- Effortless translation between encodings through command-line options

M. Fiorito et al., TrueFloat: A Templatized Arithmetic Library for HLS Floating-Point Operators. In Embedded Computer Systems: Architectures, Modeling, and Simulation. SAMOS 2023.

TrueFloat

```
float myAdd(float a, float b)
{
  return a + b;
}
```

↓

–fp-format=myAdd*e6m12b-63noh

↓

```
float myAdd(float a, float b)
{
  unsigned long long _a = __float_to_e6m12b_63noh(*((
      unsigned int*)&a), spec);
  unsigned long long _b = __float_to_e6m12b_63noh(*((
      unsigned int*)&b), spec);
  unsigned long long _res = __adde6m12b_63noh(_a, _b, spec);
  unsigned int _out = __e8m23b_127nih_to_float(_res, spec);
  return *((float*)&_out);
}
```

- Input code contains standard floating-point operations and types
- Per-function specialization string
- Bambu handles type replacement, conversions, arithmetic units generation

M. Fiorito et al., TrueFloat: A Templatized Arithmetic Library for HLS Floating-Point Operators. In Embedded Computer Systems: Architectures, Modeling, and Simulation. SAMOS 2023.

TrueFloat

FPBlender:

- Compiler pass handling FP operations in Bambu
- Uses results from previous analysis passes
- Informs subsequent transformations

TrueFloat library:

- Templatized soft-float implementations in C using basic integer operations
- Input operands and specialization arguments
- Inter-procedural analysis and constant propagation optimize the code

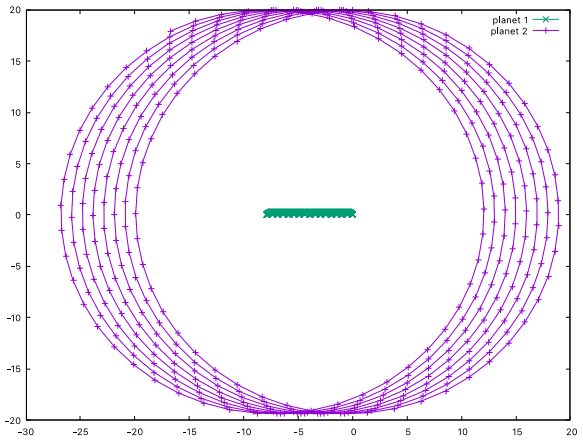
```
unsigned long long __float_adde6m12b_63noh(
  unsigned long long a, unsigned long long b,
  bits8 exp_bits, bits8 frac_bits,
  int32 exp_bias, bits8 rnd, bits8 exc,
  flag one, flag subnorm, bits8 sign)
{
 ...
 if(rnd == RND_NEVEN)
 {
   LSB_bit    = (RSig0 >> 3) & 1;
   Guard_bit  = (RSig0 >> 2) & 1;
   Round_bit  = (RSig0 >> 1) & 1;
   Sticky_bit = (RSig0 & 1) | sb;
   round  = Guard_bit & (LSB_bit | Round_bit | Sticky_bit);
 }
...
 if(rnd)
   Rrounded = RExp0RSig1 + round;
 else
   Rrounded = RExp0RSig1;
 ...
```

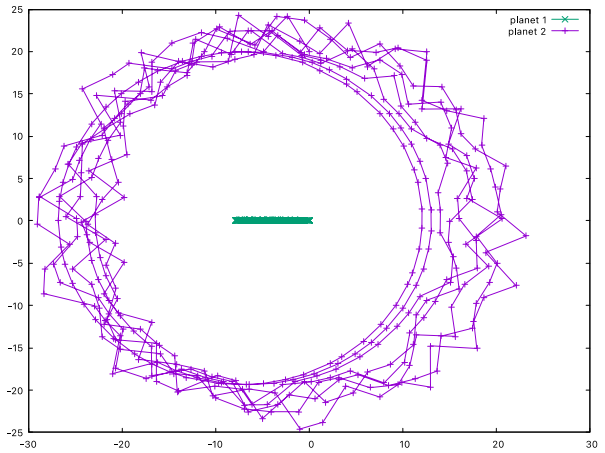
M. Fiorito et al., TrueFloat: A Templatized Arithmetic Library for HLS Floating-Point Operators. In Embedded Computer Systems: Architectures, Modeling, and Simulation. SAMOS 2023.

TrueFloat

Testing TrueFloat configurations is particularly easy through the Bambu cosimulation framework

- Example: main program which runs the simulation and plots the outputs
  - Use the same main program, plot outputs from the simulated accelerator
  - Assess errors visually





S. Curzel and M. Gribaudo. Custom Floating-Point Computations for the Optimization of ODE Solvers on FPGA. In: Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures and Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM). 2025

Interfaces and Caches

04

Interface generation

Supported protocols:

- Minimal Memory Interface (simple Bambu bus)
- FIFO, AxiStream
- Handshake
- Array
- AXI4-Master

(Mostly) compatible with Vitis HLS interfaces

Requested by the user through `--generate-interface=INFER` and either

- An `architecture.xml` file
- Interface pragmas in the code

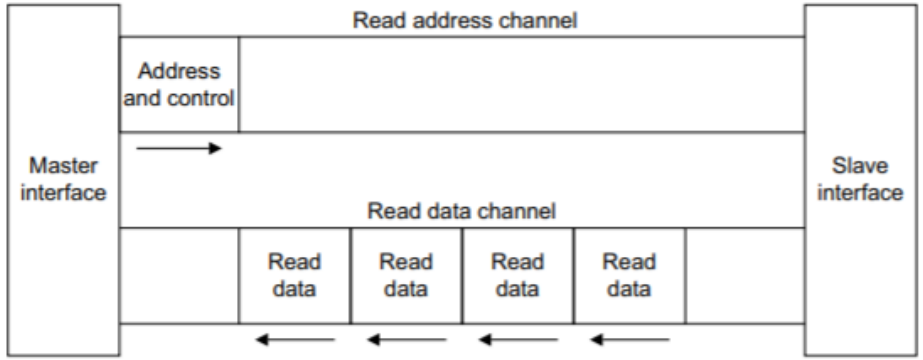
```
#pragma HLS bus mode = m_axi
#pragma HLS interface port = edges mode = bus
#pragma HLS interface port = level mode = bus
void bfs(edge_t* edges, level_t* level, ...)
```

```
<?xml version="1.0"?>
<module>
  <function name=»bfs" symbol=»bfs">
    <bundles>
      <bundle alignment="1" bank_number="0" mode="m_axi" name="bus"></bundle>
      <bundle mode="bus" name="edges"></bundle>
      <bundle mode="bus" name="level"></bundle>
...
```

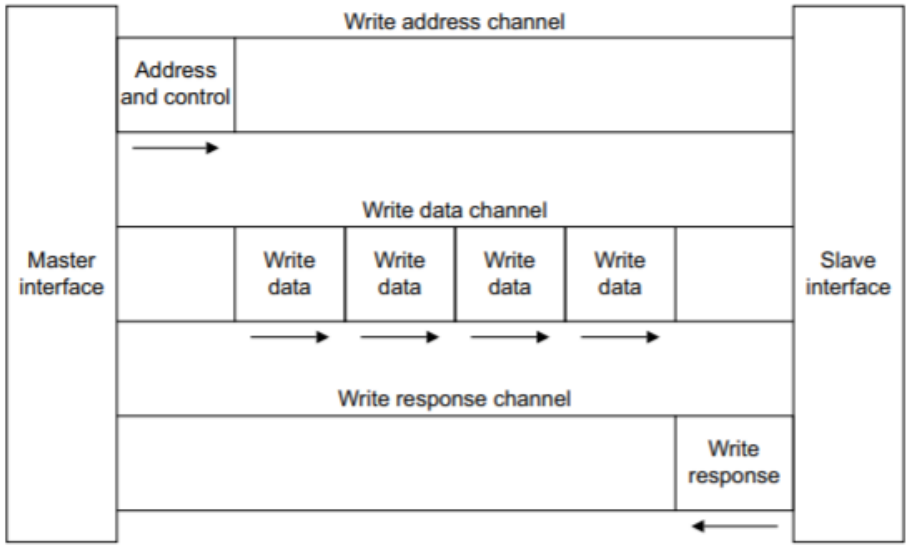
Interface generation – Axi4/AxiS

Support to Axi4 master and AxiS interfaces

- Easier integration with ARM processor (e.g., on the NG-ULTRA board)
- Automatic generation of AXI testbench supported
- Memory latency can be configured



Channel architecture for read transaction with Axi4
(AMBA® AXI™ and ACE™ Protocol Specification)

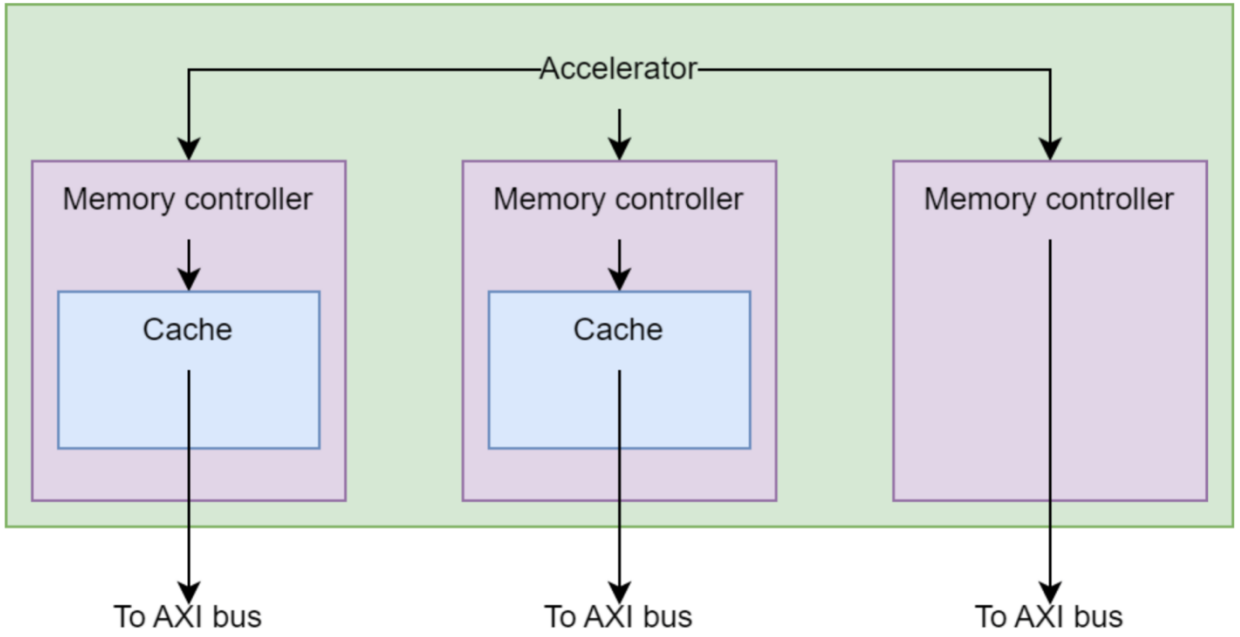


Channel architecture for write transaction with Axi4
(AMBA® AXI™ and ACE™ Protocol Specification)

Interface generation - Caches

Support for caches on AXI4 interfaces:

- Customizable cache parameters: line size, cache size, replacement/write policy...
- May improve performance when data accesses are contiguous and predictable



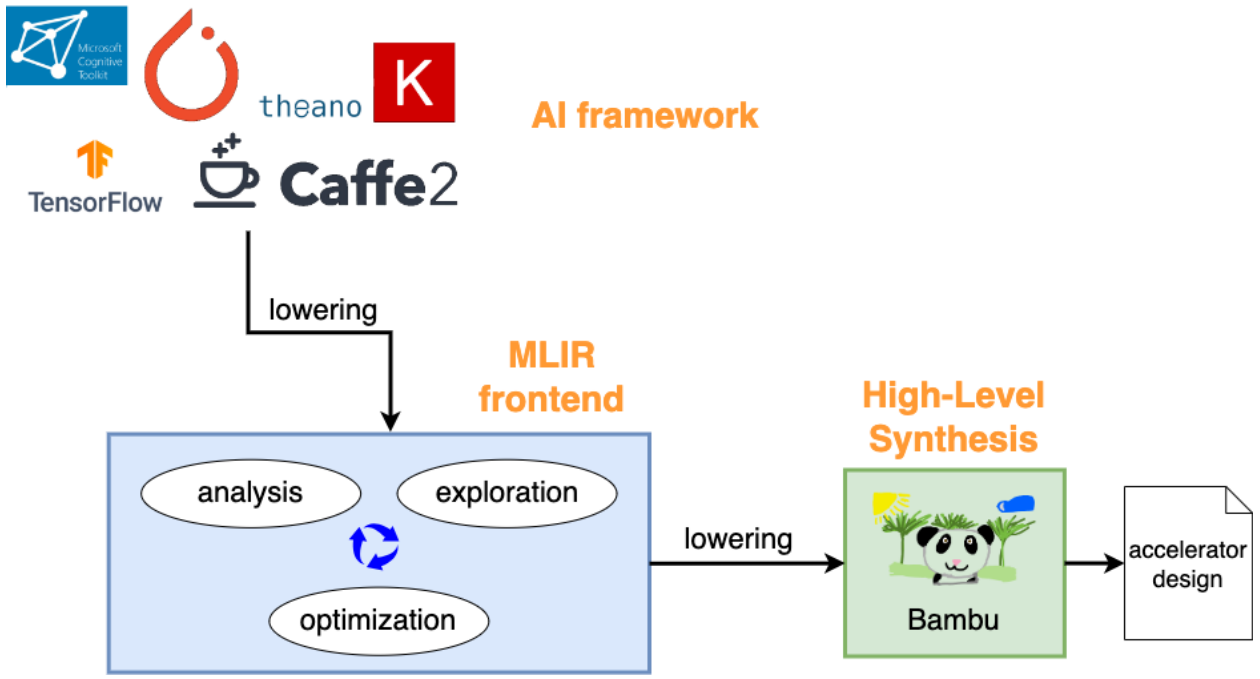
Schematic of an accelerator with three memory channels (two with caches, one cacheless)

Compiler-driven innovation: Bambu + MLIR

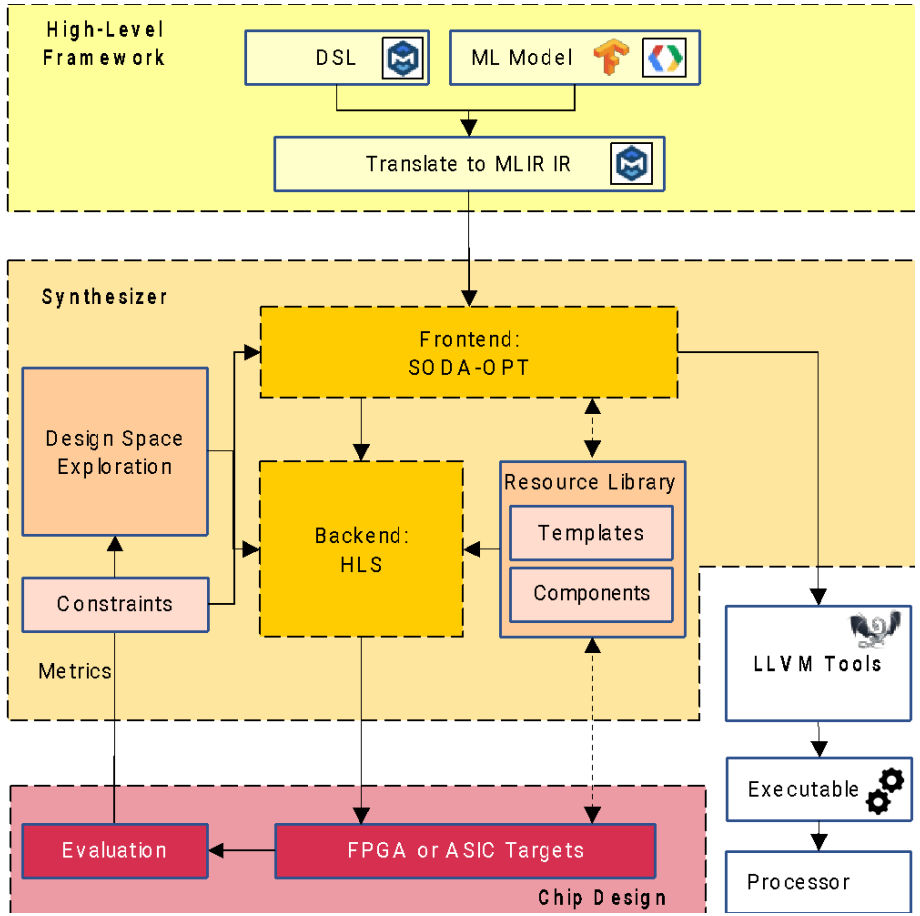
05

Compiler-driven innovation: MLIR

- Multi-Level Intermediate Representation (MLIR), infrastructure for domain-specific compilers
  - Used predominantly for AI
- MLIR -> LLVM dialect -> LLVM IR -> Bambu
  - Any design written in MLIR can be synthesized!



The SODA Synthesizer



- Modular, multi-level, **open-source hardware compiler** from high-level programming frameworks to silicon
- Compiler-based frontend based on MLIR (SODA-OPT)
- Compiler-based HLS backend (Bambu)
- Generates **synthesizable Verilog** for a variety of FPGA and ASIC targets
- Optimizations at all levels are performed as compiler passes

S. Curzel et al., End-to-End Synthesis of Dynamically Controlled Machine Learning Accelerators, in IEEE Transactions on Computers, vol. 71, no. 12, pp. 3074-3087, 1 Dec. 2022, doi: 10.1109/TC.2022.3211430
N. B. Agostini et al., Bridging Python to Silicon: The SODA Toolchain, in IEEE Micro, vol. 42, no. 5, pp. 78-88, 1 Sept.-Oct. 2022, doi: 10.1109/MM.2022.3178580

The SODA Synthesizer

SODA Bambu script:

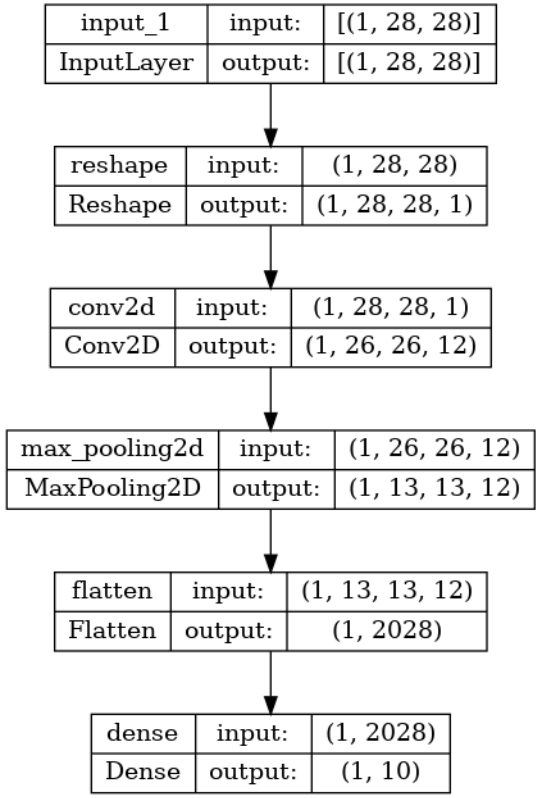
```
bambu -v3 --print-dot \
  -lm --soft-float \
  --compiler=I386_CLANG16  \
  --device=nangate45 \
  --clock-period=5 \
  --experimental-setup=BAMBU-BALANCED-MP \
  --channels-number=2  \
  --memory-allocation-policy=ALL_BRAM \
  --disable-function-proxy \
  --generate-tb=main_kernel_testbench.c \
  --simulate --simulator=VERILATOR \
  --verilator-parallel \
  --top-fname=main_kernel \
  input.ll 2>&1 | tee bambu-log
```

ASIC OpenRoad target

Shortcut for many other optimizations

Memory-level parallelism

CNN acceleration on FPGA



CNN trained on MNIST

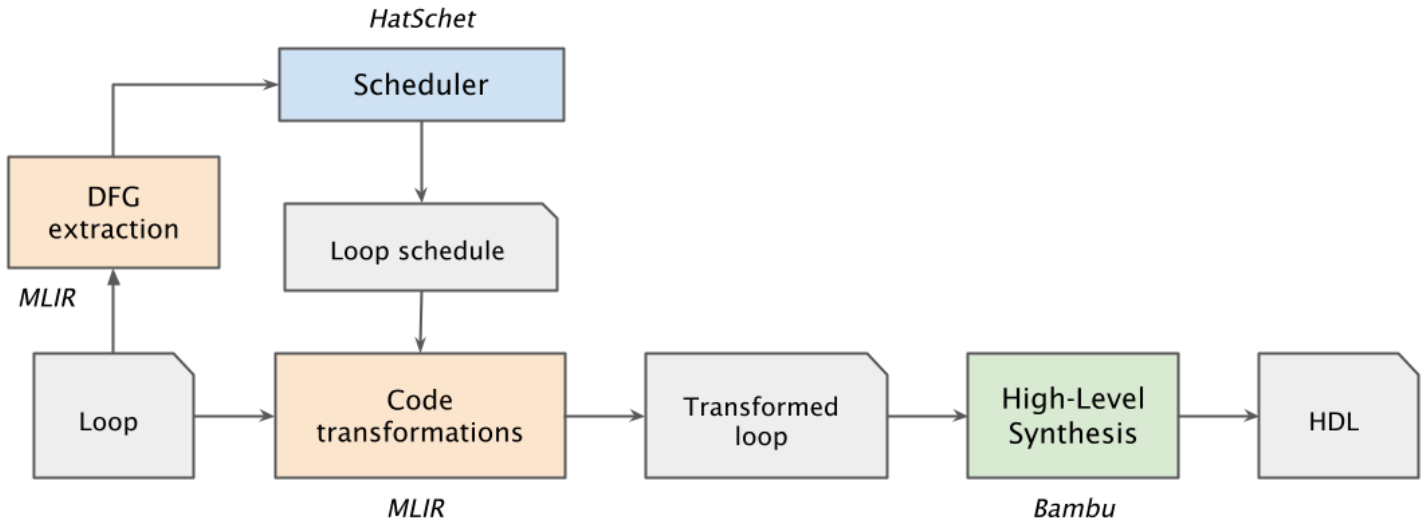
	NG-Ultra
LUTS	4627
Registers	5714
Frequency	45.7 MHz
DSP	54
MEM	34
cycles	169,649
latency	3.7ms

Synthesis result targeting an NG-Ultra FPGA

- Neural network trained and quantized to 8-bit to reduce area and improve hardware performance
- TFLite to MLIR export
- Bambu synthesis

HERMES

Bambu + MLIR: high-level loop pipelining



- Goal: provide a pre-optimized IR to the HLS tool (with pipelined loops)
- Bambu HLS
  - Can synthesize LLVM IR (natural target for MLIR)
  - Does not support loop pipelining (*)

(*) now available upon request

S. Curzel et al., Pre-Scheduling of Affine Loops for HLS Pipelining. In: Euro-Par 2024: Parallel Processing Workshops. 2025, pp. 431–442

Bambu + MLIR: high-level loop pipelining

```
1  func @example(%arg0: memref<1000xi32>) {
2    affine.for %arg1 = 0 to 1000 {
3      %0 = affine.load %arg0[%arg1]
4      %1 = arith.muli %0, %0
5      affine.store %1, %arg0[%arg1]
6    }
7    return
8  }
```

```
# II 1
# vertex;          cycle;   functional_unit
affine.load_1;       0;          load0
arith.muli_2;        1;          mul0
affine.store_3;      2;          store0
```

```
1  #map = affine_map<(d0) -> (d0 - 2)>
2  func @example(%arg0: memref<1000xi32>,
                 %arg1: memref<1000xi32>) {
3    %c0 = arith.constant 0 : index
4    %0 = affine.load %arg0[%c0] : memref<1000xi32>
5    %c1 = arith.constant 1 : index
6    %1 = affine.load %arg0[%c1] : memref<1000xi32>
7    %2 = arith.muli %0, %0 : i32
8    %3:2 = affine.for %arg2 = 2 to 1000
           iter_args(%arg3 = %1, %arg4 = %2) -> (i32, i32) {
9      %5 = affine.load %arg0[%arg2] : memref<1000xi32>
10     %6 = arith.muli %arg3, %arg3 : i32
11     %7 = affine.apply #map(%arg2)
12     affine.store %arg4, %arg1[%7] : memref<1000xi32>
13     affine.yield %5, %6 : i32, i32
14   }
15   %4 = arith.muli %3#0, %3#0 : i32
16   %c998 = arith.constant 998 : index
17   affine.store %3#1, %arg1[%c998] : memref<1000xi32>
18   %c999 = arith.constant 999 : index
19   affine.store %4, %arg1[%c999] : memref<1000xi32>
20   return
21 }
```

S. Curzel et al., Pre-Scheduling of Affine Loops for HLS Pipelining. In: Euro-Par 2024: Parallel Processing Workshops. 2025, pp. 431–442

Bambu + MLIR: high-level loop pipelining

```
  # Forward results if needed
  soda-opt input.mlir -pass-pipeline="func.func(forward-results)" -o=forwarded.mlir"

  # Apply if conversion where possible
  soda-opt forwarded.mlir -pass-pipeline="func.func(if-conversion)" -o=converted.mlir"

  # Extract data flow graph from the mlir code.
  soda-opt converted.mlir -pass-pipeline="func.func(data-flow-graph)"

  # Generate default resources.xml file
  python3 $ROOT/output/generate_hatschet_resources.py

  # Obtain a schedule for the given data flow graph, scheduler, and resource file.
  hatschet dfg.graphml --scheduler=MOOVAC --resource=resources.xml --writeschedule=schedule.csv

  # Rebuild mlir following the scheduled received from HatSchet.
  soda-opt converted.mlir -pass-pipeline="func.func(loop-scheduling {schedule=schedule.csv})"
-o=scheduled.mlir

 # Lower, translate to LLVM IR, synthesize with Bambu.
 soda-opt scheduled.mlir -lower-affine ...
```

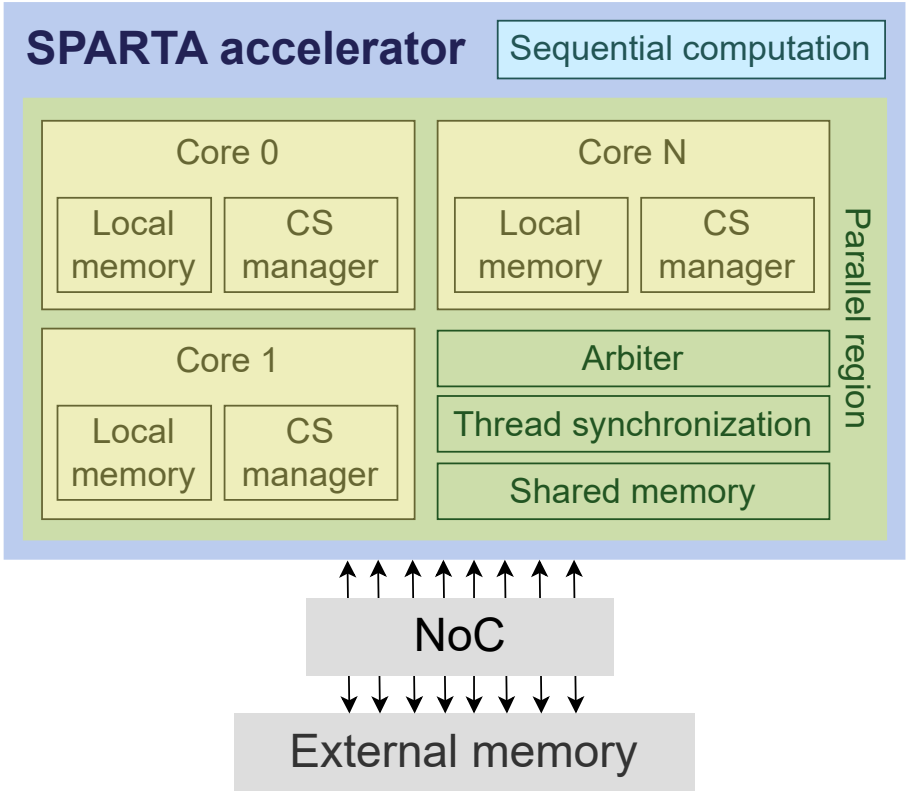
S. Curzel et al., Pre-Scheduling of Affine Loops for HLS Pipelining. In: Euro-Par 2024: Parallel Processing Workshops. 2025, pp. 431–442

Multi-threaded accelerators: SPARTA

06

Synthesis of PARallel multi-Threaded Accelerators (SPARTA)

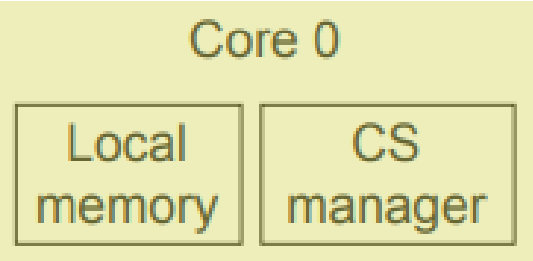
- Starts from high-level languages (C/C++) annotated with OpenMP pragmas
  - Parallel, for, sections
  - Reduction, shared, private, firstprivate
  - Critical, barrier
- Produces an accelerator with multiple cores that can execute concurrently



G. Gozzi et al., SPARTA: High-Level Synthesis of Parallel Multi-Threaded Accelerators. In: ACM Trans. Reconfigurable Technol. Syst. 18.1 (Dec. 2024)

SPARTA synthesis flow

```
void dot_product(int A[M], int B[M], int res[M]){
   int sum = 0;
#pragma omp parallel num_threads(M) for
   for(i = 0; i < M; i++){
      prod(A[i], B[i], res[i]);
   }
}
```



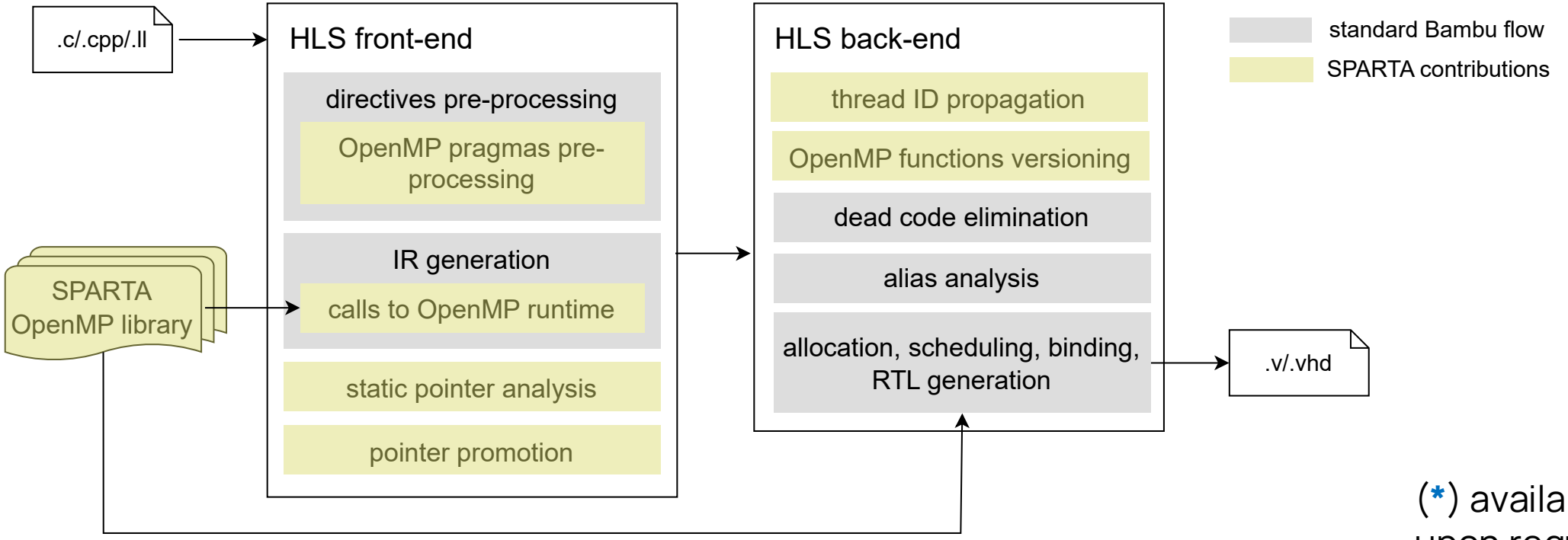

```
define i32 @dot_product (...) {
   ...
   call void __kmpc_push_num_threads(M)
   call void __kmpc_fork_call(.omp_outlined., ...)
   ...
}
define void @ .omp_outlined.(...) {
   %6 = call i32 __kmp_bambu_tid_from_gtid(i32 %0)
   ...
   call void prod(...)
   call void __kmp_bambu_barrier_reached(i32 %6)
   call void __kmp_bambu_wait_all_threads()
   ...
}
```

- Each outlined function is mapped to a core, which can be shared through *context switching*
  - Hide external memory access latency

G. Gozzi et al., SPARTA: High-Level Synthesis of Parallel Multi-Threaded Accelerators. In: ACM Trans. Reconfigurable Technol. Syst. 18.1 (Dec. 2024)

SPARTA synthesis flow

- Integration to the standard synthesis flow of Bambu (*) exploiting the LLVM OpenMP runtime

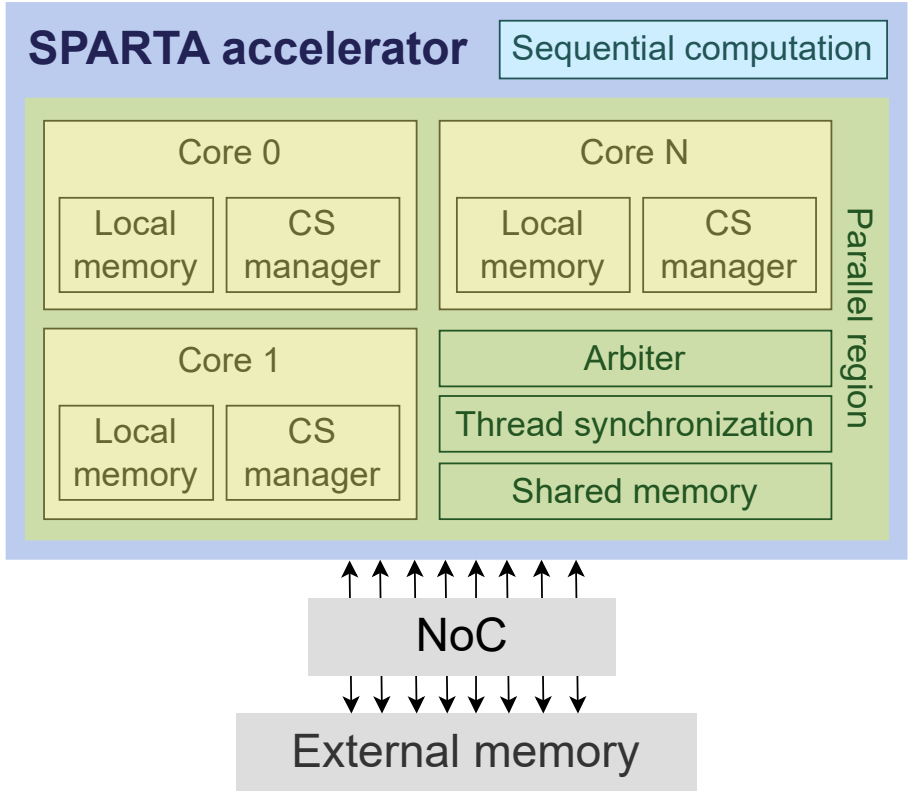


(*) available upon request

G. Gozzi et al., SPARTA: High-Level Synthesis of Parallel Multi-Threaded Accelerators. In: ACM Trans. Reconfigurable Technol. Syst. 18.1 (Dec. 2024)

SPARTA architecture

- Multiple parallel cores
- Hardware for arbitration and synchronization
- Efficient implementation of reductions
- Configurable number of channels towards external memory and custom NoC
- Works very well for irregular applications



G. Gozzi et al., SPARTA: High-Level Synthesis of Parallel Multi-Threaded Accelerators. In: ACM Trans. Reconfigurable Technol. Syst. 18.1 (Dec. 2024)

Thank you for the attention!

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

serena.curzel@polimi.it
https://panda.deib.polimi.it/

More details

Bambu features

Frontend passes:

- Source code optimizations
  - Alias analysis, dead-code elimination, hoisting, loop optimizations, etc...

Interface generation - FIFO

C++ FIFO interface support:

- ac_ channel<T>
- hls::stream<T>

```
void sum3numbers(ac_channel<ap_uint<64>>& a,
                 ac_channel<ap_uint<64>>& b,
                 ac_channel<ap_uint<64>>& c,
                 ac_channel<ap_uint<64>>& d)
{
   int i;
   for(i = 0; i < 8; ++i)
      d.write(a.read() + b.read() + c.read());
}
```

Example of c++ code with ac_ channels

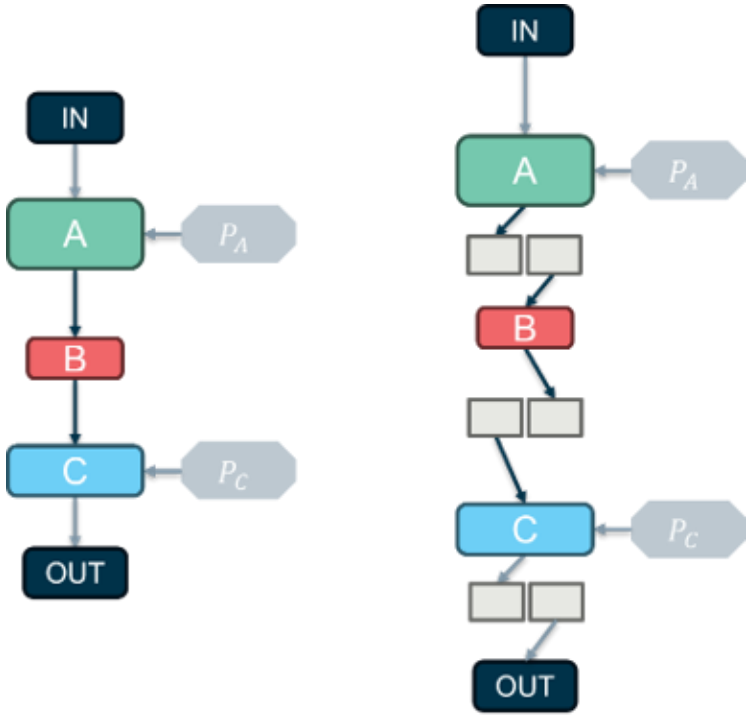
Dataflow

```
class SimpleSystem {
  AddBlock<5> A;
  MulBlock B;
  SubSystem C;
  ac_ channel<int> x, y;

 public:
  void top(ac_ channel<int>& in1, ac_ channel<int>& in2,
        ac_ channel<int>& in3,
        ac_ channel<int>& out) {
    #pragma HLS dataflow
    A.compute(in1, x);
    B.compute(x, in2, y);
    C.compute(y, in3, out);
  }
};

void dataflow_ top(ac_ channel<int>& in1, ac_ channel<int>& in2,
           ac_ channel<int>& in3,
             ac_ channel<int>& out) {
  static SimpleSystem sys;
  sys.top(in1, in2, in3, out);
}
```

Source code in C++ with pragma dataflow



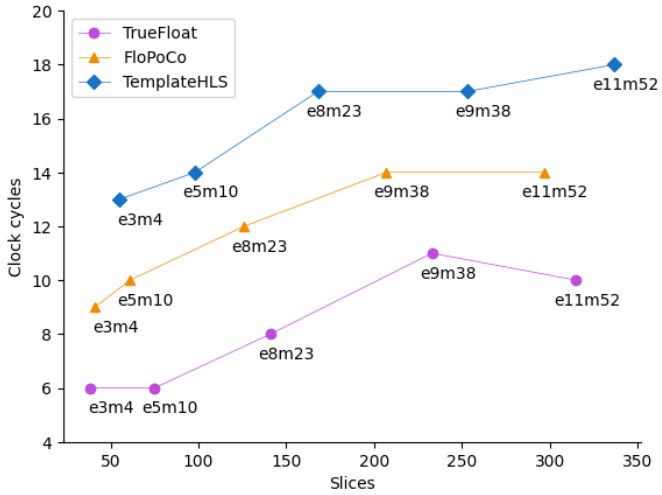
Comparison of the resulting task graph with and without dataflow

Dataflow

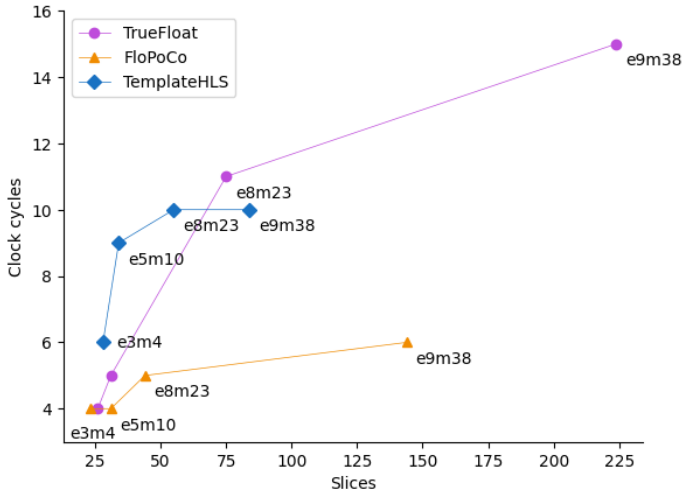
Dataflow support

- ac_ channel & hls_ stream
- FIFO depth could be controlled by pragmas
- Struct passed as template parameter supported
- Data field member support could be improved
- ac_ int and ac_ fixed supported
  - Synthesis efficiency improved when PandA-Bambu is used with Clang16

TrueFloat vs state of the art



Adder



Multiplier

PolyBench 2mm

HLS tool	Cycles	Slices	DSPs	LUTs	Registers
Bambu	65250	422	2	1022	891
Vitis HLS	76708	413	14	908	1220

TrueFloat: A Templatized Arithmetic Library for HLS Floating-Point Operators

Michele Fiorito, Serena Curzel(✉), and Fabrizio Ferrandi
Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milan, Italy
{michele.fiorito,serena.curzel,fabrizio.ferrandi}@polimi.it

https://doi.org/10.1007/978-3-031-46077-7_35

Impact of TrueFloat specialization strings

	FPAdd				FPMult				
Spec	Cycles	Slices	LUTs	Registers	Cycles	Slices	LUTs	DSPs	Registers
nih	11	280	750	961	12	216	447	10	927
nihs	12	383	979	1219	13	210	448	10	936
noh	11	259	723	869	12	206	400	10	964
tih	9	218	632	763	10	167	370	10	607
toh	9	221	610	676	10	150	316	10	674

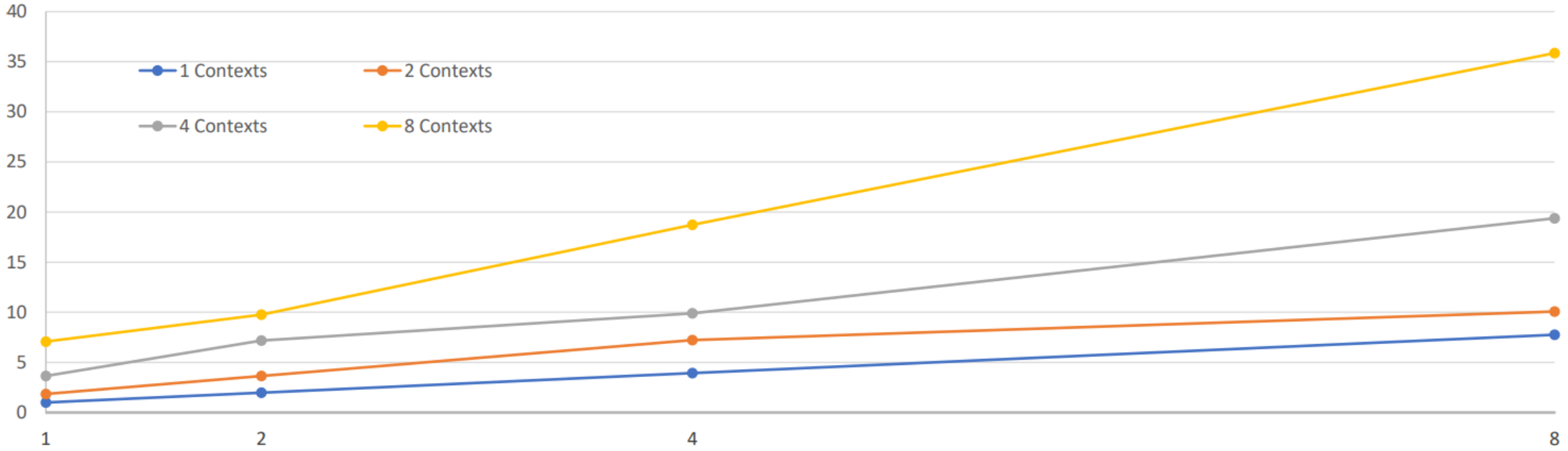
IEEE FP64

TrueFloat: A Templatized Arithmetic Library for HLS Floating-Point Operators

Michele Fiorito, Serena Curzel(✉), and Fabrizio Ferrandi
Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milan, Italy
{michele.fiorito,serena.curzel,fabrizio.ferrandi}@polimi.it

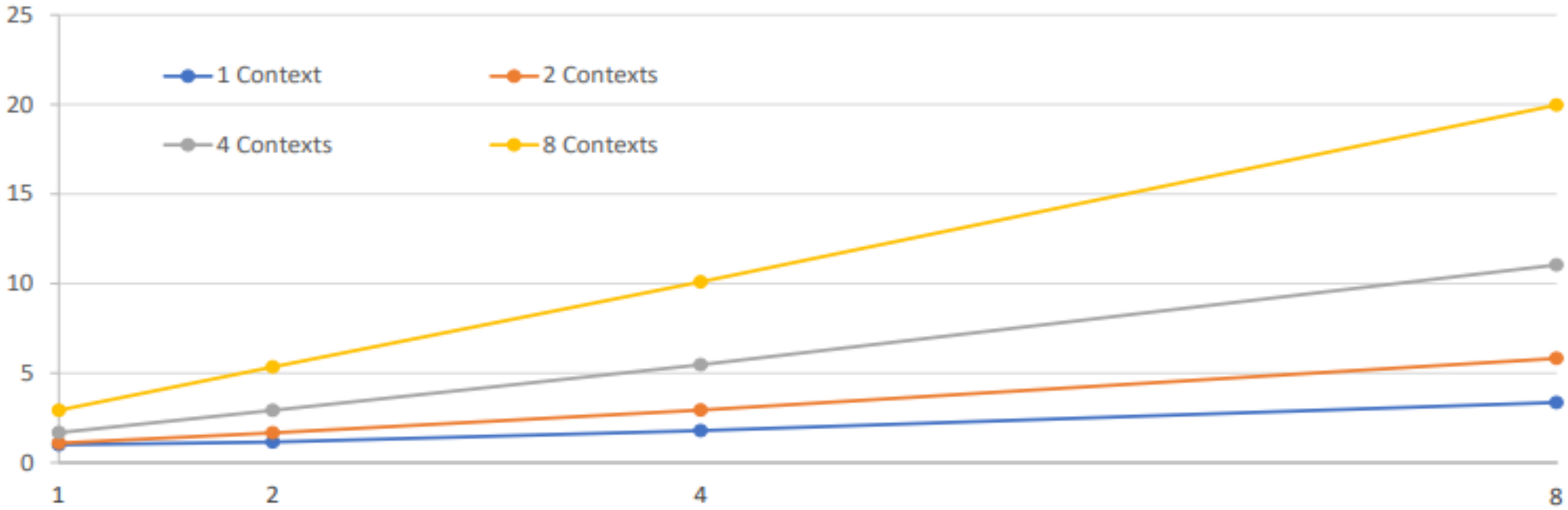
https://doi.org/10.1007/978-3-031-46077-7_35

SPARTA results



Speed up of SPARTA accelerators over the sequential baseline for the Connected Components benchmark with different cores and contexts

SPARTA results



Speed up of SPARTA accelerators over the sequential baseline for the Triangle Count benchmark with different cores and contexts