OpenMP Device Offloading to FPGAs using the Nymble Infrastructure

Jens Huthmann, Lukas Sommer, Artur Podobas, Andreas Koch, Kentaro Sano

Jens Huthmann

Lukas Sommer

Artur Podobas

Andreas Koch

Kentaro Sano

Motivation

- FPGAs as interesting acceleration platform for HPC
  - Greater architectural flexibility than GPUs, …
  - Less initial design cost than dedicated ASIC accelerator

- Programming with Hardware Description Languages requires expert knowledge and significant amounts of time

- High-level programming interface desirable
  - Can we use OpenMP device offloading and parallel constructs for this?

Nymble HLS Compiler

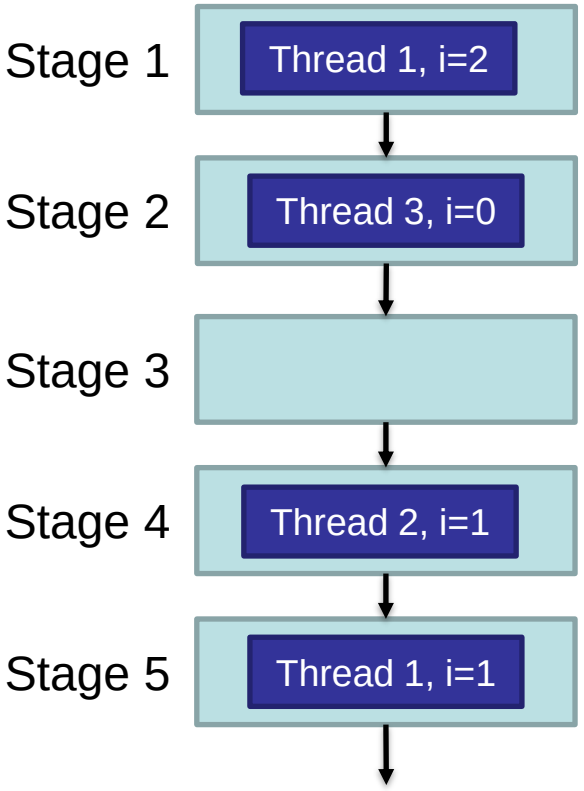
- Academic HLS compiler framework

- LLVM-based frontend, originally used custom pragmas for HLS
  - LLVM-IR allows to optimize across the OpenMP frontend & HLS backend border
  - Prior approaches often limited by source-level (C/C++ + HLS-pragmas) interfaces of commercial HLS compilers

- Generates complete accelerator design as Verilog code
  - Can be synthesized for FPGA using vendor tools

Nymble-MT Execution Model

- **Simultaneous** multithreading inside FPGA accelerator
- Execution in accelerator organized in so-called stages
  - Stage contains all operations scheduled in a time-step
- Multiple loop iterations and multiple threads simultaneously active
  - Thread re-ordering possible after stall

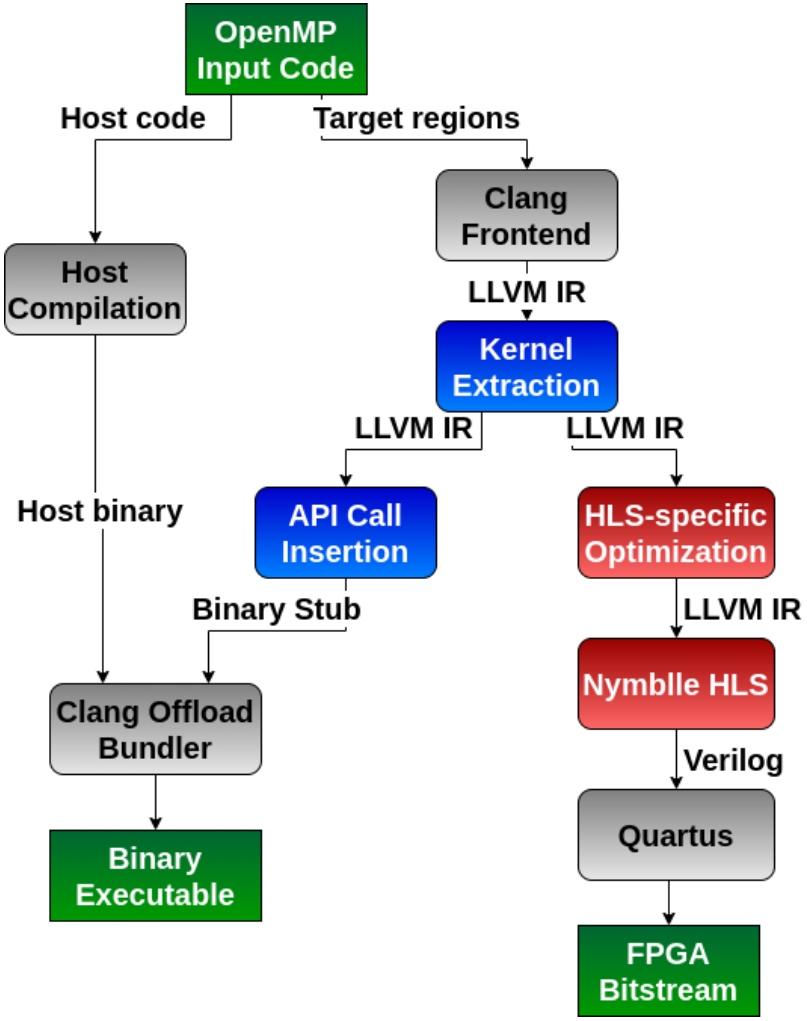


Goal

- Develop OpenMP frontend for Nymble compiler
- Map OpenMP parallelism to simultaneously executing HW threads in Nymble-MT execution model
  - `teams distribute`
  - `parallel (for)`
- Add HW support for OpenMP synchronization constructs
  - Critical regions
  - Barriers
- Optimize across border between OpenMP frontend and Nymble HLS backend based on LLVM IR

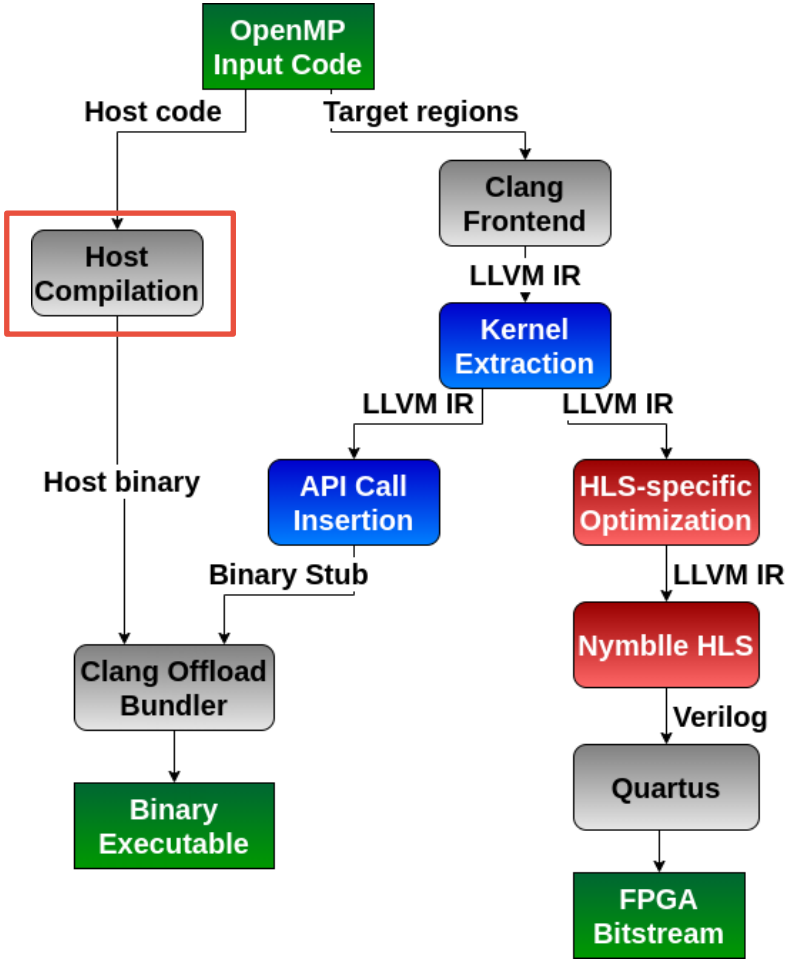
COMPILATION FLOW

Compilation Flow



Compilation Flow – Input Code

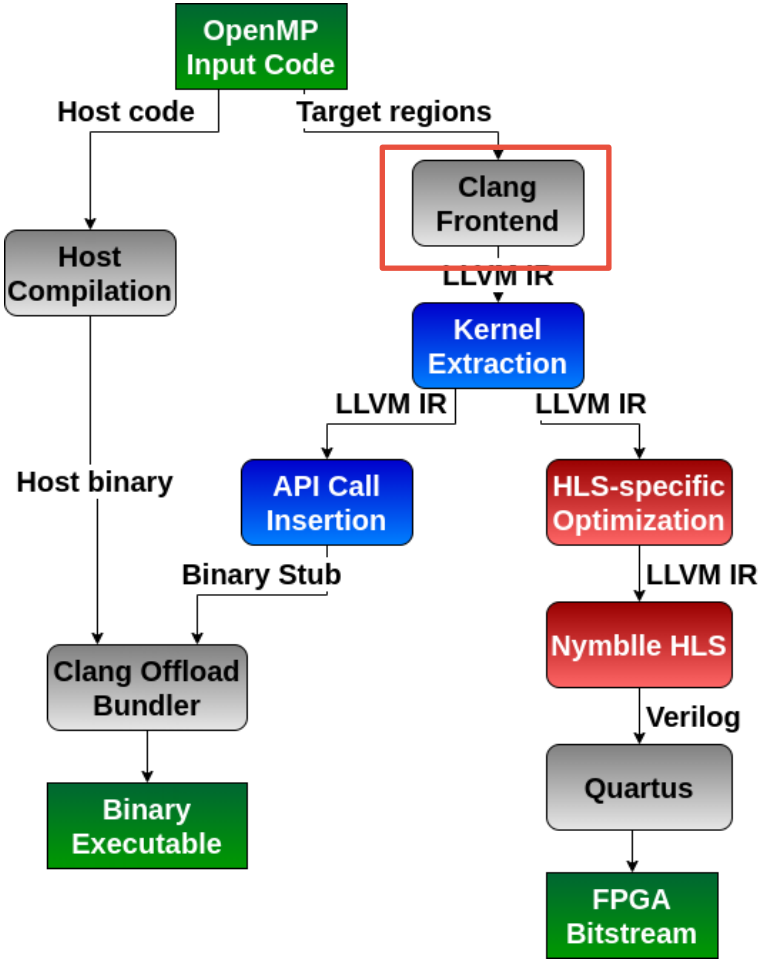
- Separate compile passes for host and each device executed by Clang driver
- Host compilation completely unchanged
  - Any OpenMP host construct supported by Clang can be used



Compilation Flow – Device Frontend

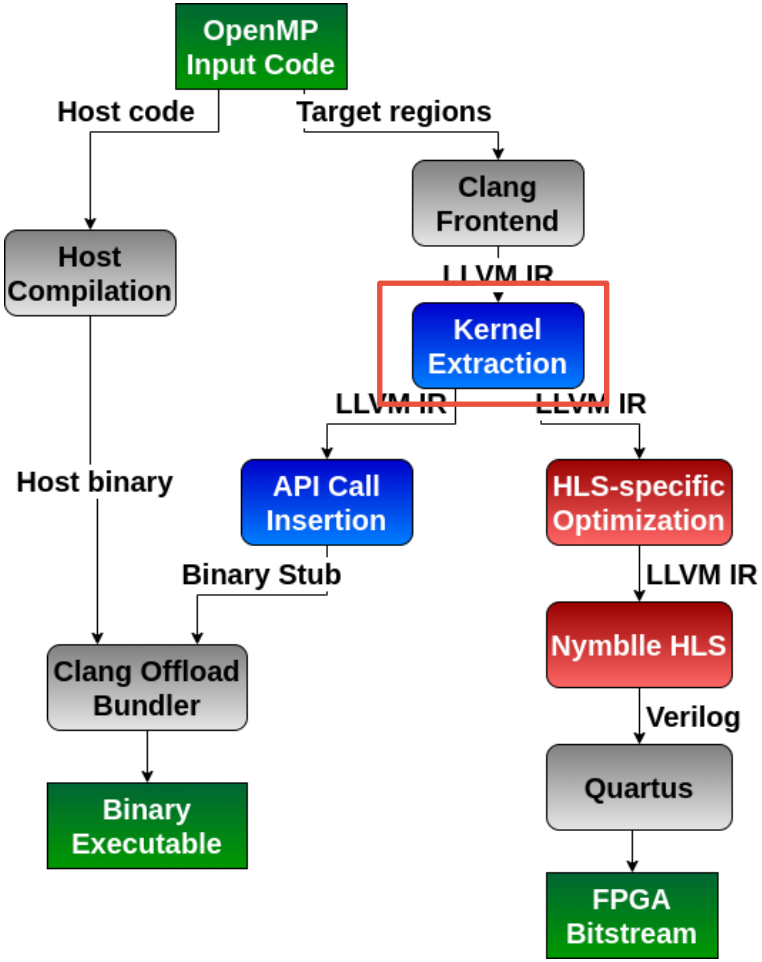
Currently supported constructs in target regions:

- **`teams distribute`** or
- **`parallel`** (**`for`**), static schedule
- Full support for data management
  - **`map`** clause, including map type
- Synchronization constructs
  - **`critical`**
  - **`barrier`**



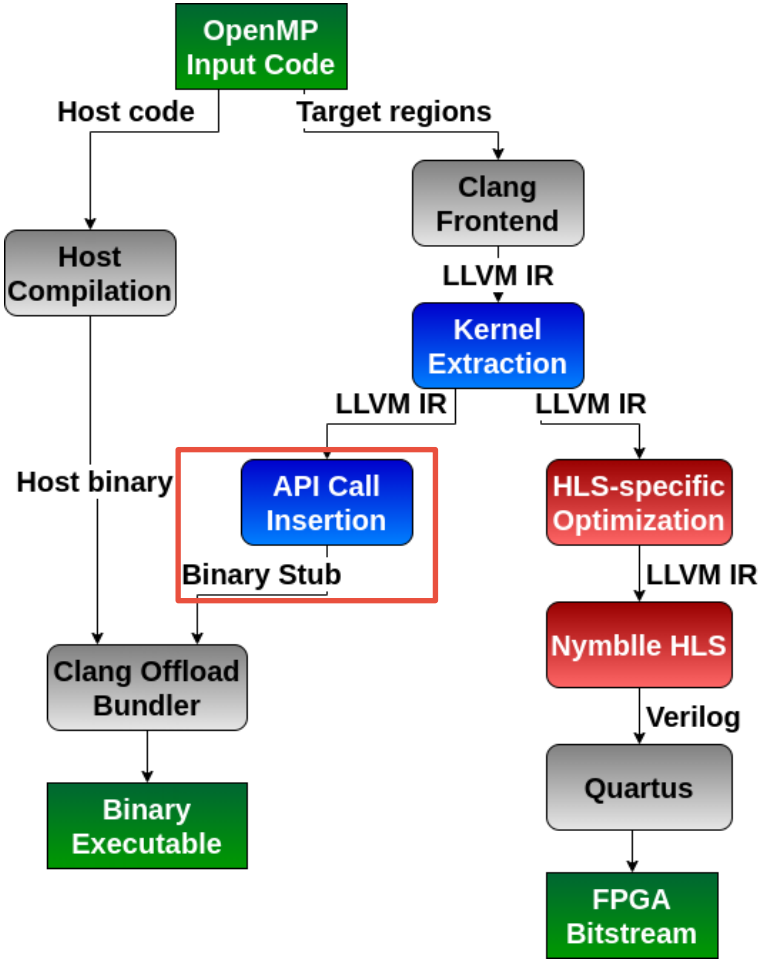
Compilation Flow – Kernel Extraction

- Split target region
  - Small stub remains in software
  - Hardware region extracted into function
    - Will be handled by Nymble HLS
- Currently limited to a single target region due to limitations of hardware shell

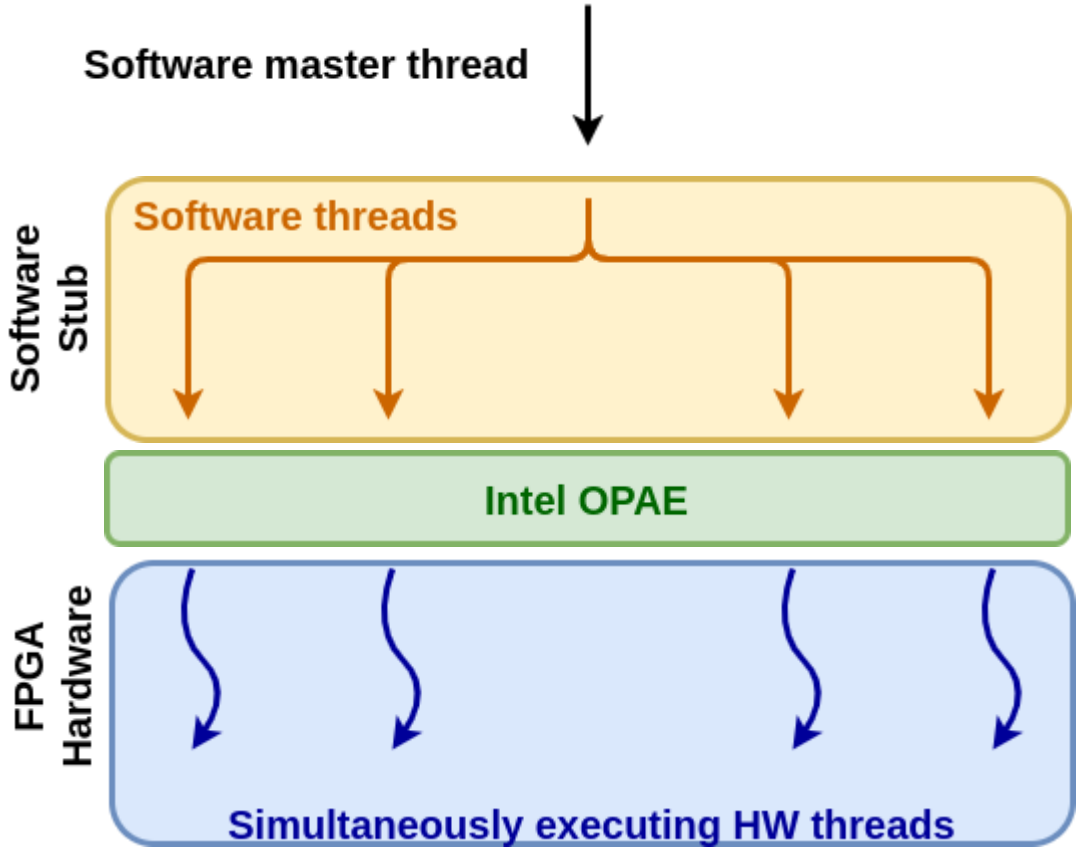


Compilation Flow – Software Stub

- Small stub remains in software
  - Initializes parallel execution through **`libomp`**, e.g., calculates schedule
  - Uses Intel Open Programmable Acceleration Engine (OPAE) API to pass parameters to accelerator and launch HW
- Software stub is compiled to software binary (e.g., x86-64)



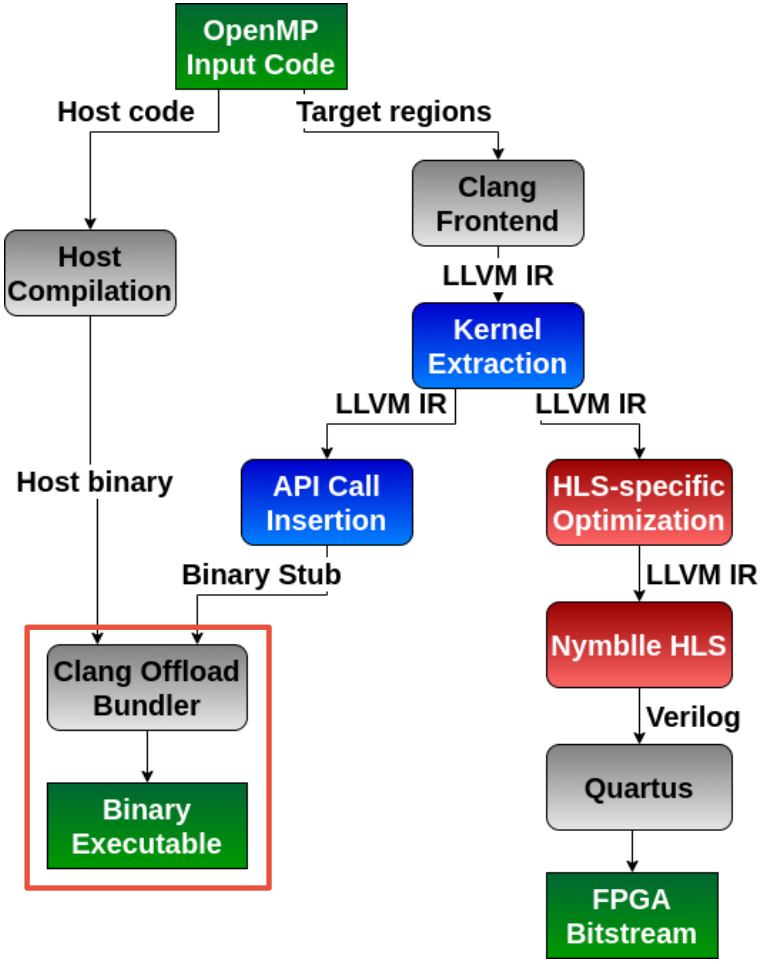
Binary Software Stub



- Software thread spawning initiated by **`libomp`**
- Calculates static worksharing schedule
- Passes parameters to hardware
- 1-to-1 mapping from software threads to FPGA hardware threads

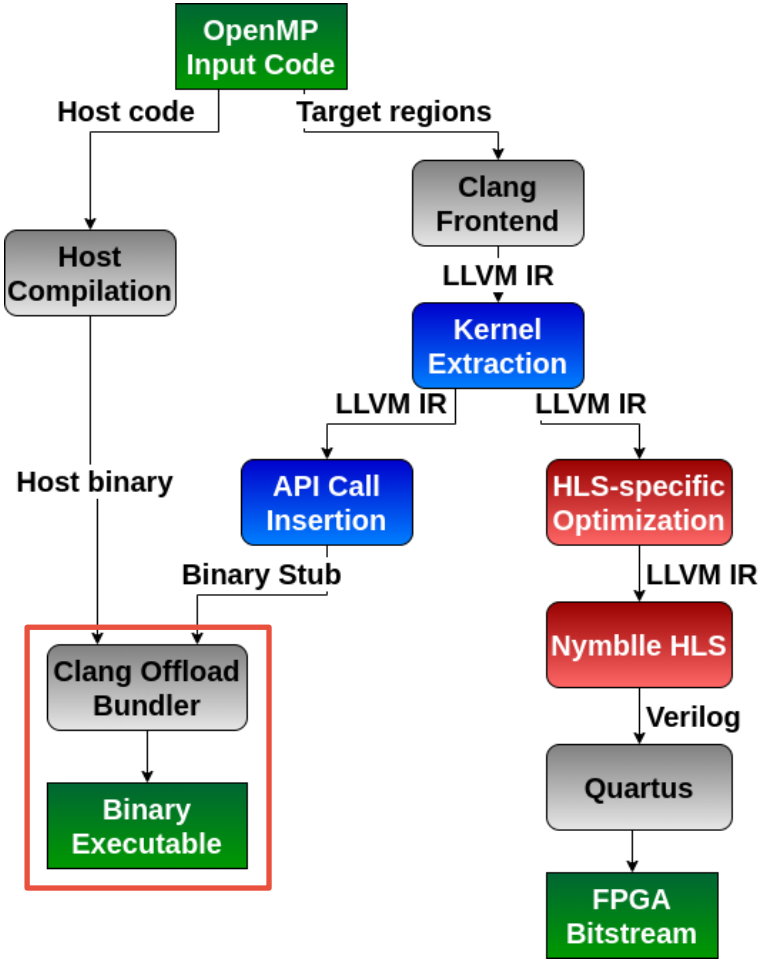
Compilation Flow – Bundling

- Compiled software stub is included in „fat“ binary
- Will be loaded by LLVM‘s **`libomptarget`** during runtime
- **`libomptarget`** plugin handles hardware launch and data management



Compilation Flow – Bundling

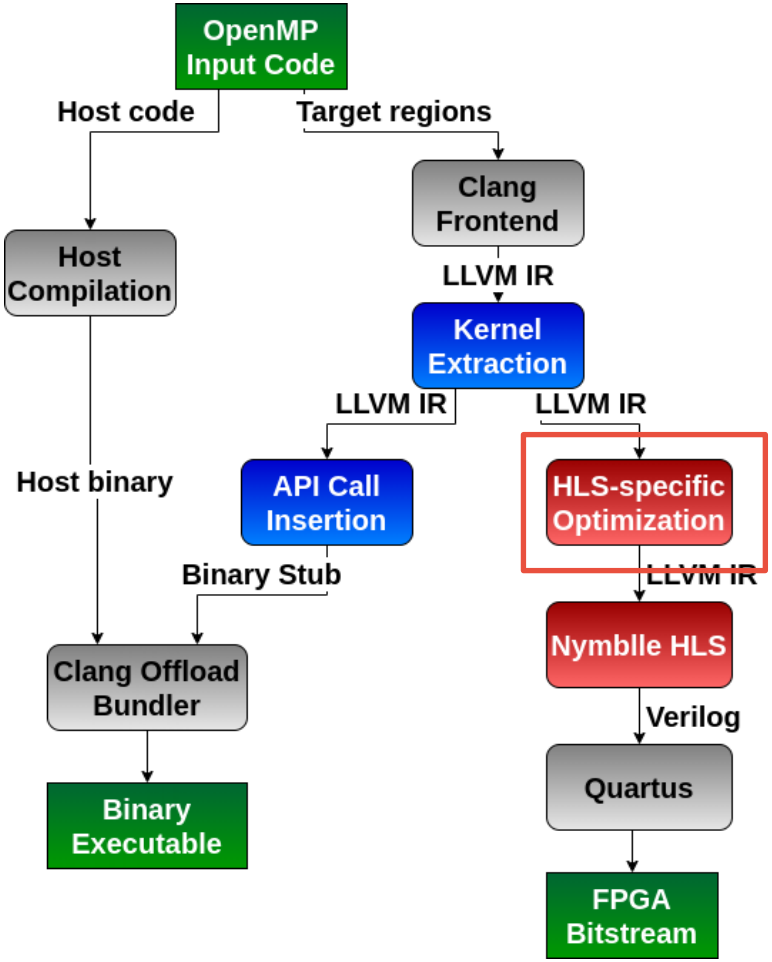
- Compiled software stub is included in „fat“ binary
- Will be loaded by LLVM‘s **`libomptarget`** during runtime
- **`libomptarget`** plugin handles hardware launch and data management



Compilation Flow – LLVM IR Transformations

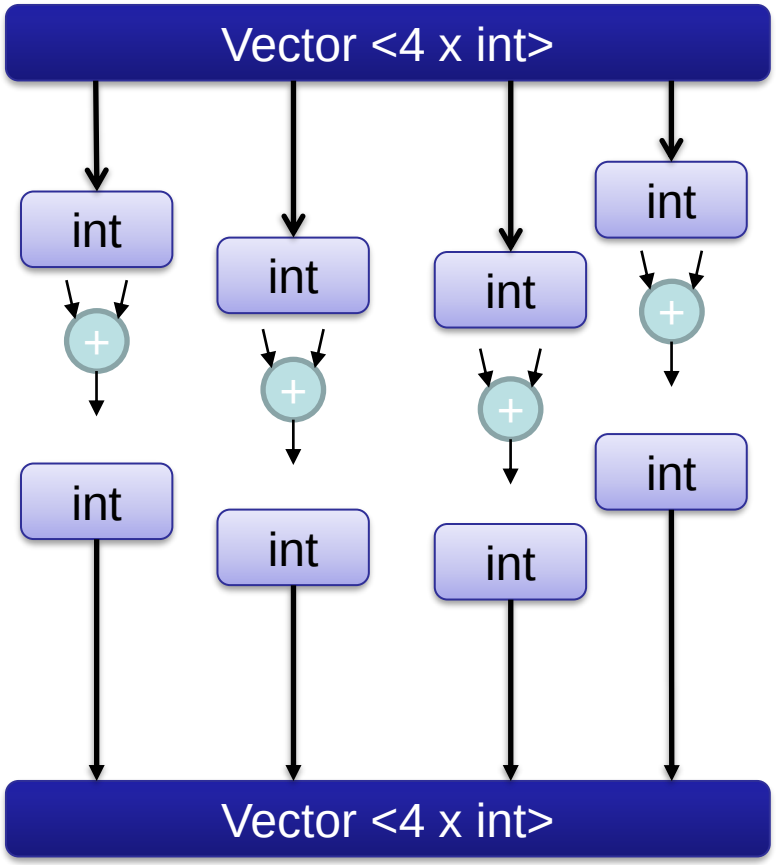
- Combination of generic and HLS-specific optimizations and transformations based on LLVM IR

- Translate OpenMP constructs inside target region
  - Critical regions & barrier: Hardware semaphore access
  - API functions (e.g., **`omp_get_num_threads`**): Register read



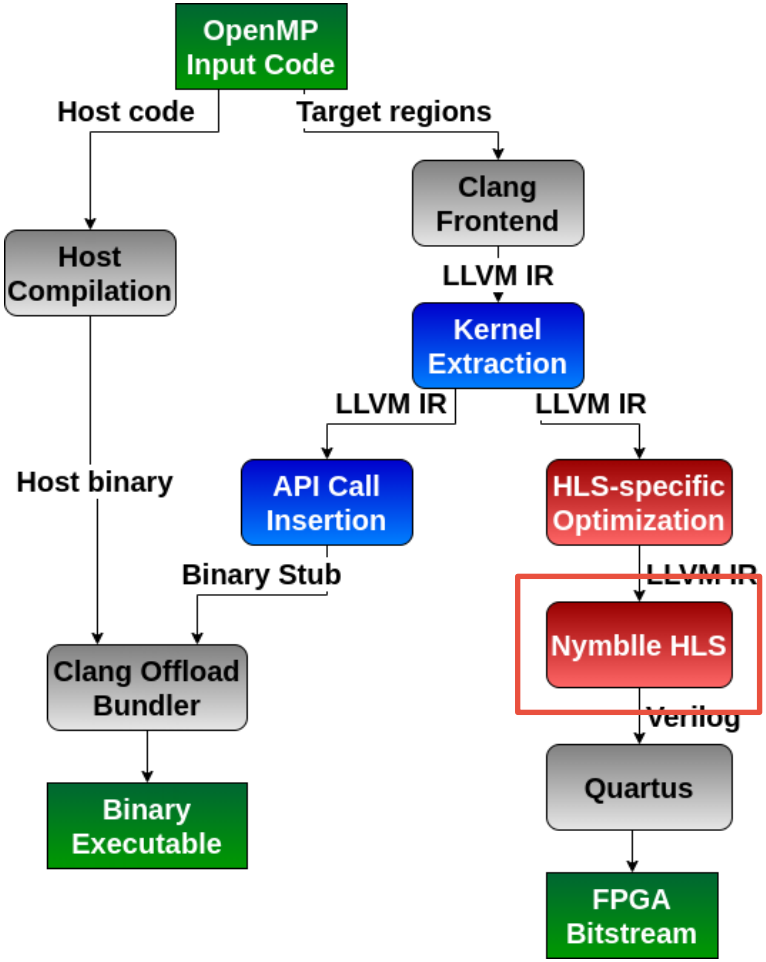
Optimization Example

- Vectorization of load & stores allows to better exploit memory interface bandwidth
- Vectorized arithmetic operators inside FPGA hardware not beneficial
  - Complicates HLS scheduling
- Split vectorized operations, extract/insert becomes no-op (wire selection/assign) in FPGA hardware



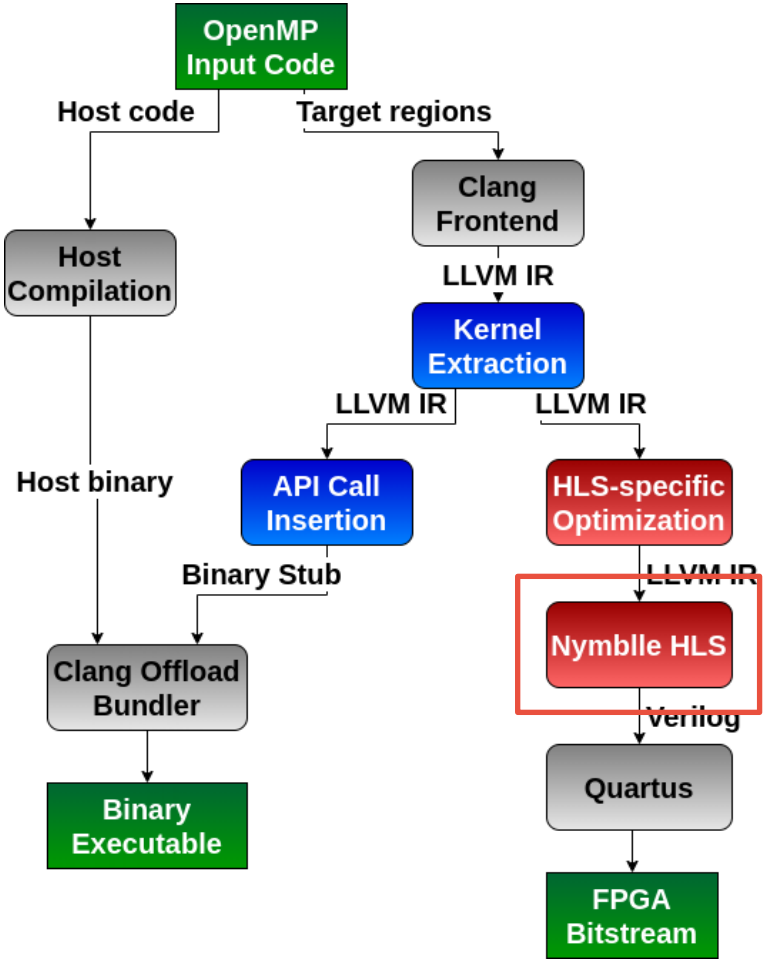
Compilation Flow – Nymble HLS Backend

- Translates LLVM IR to CDFG intermediate representation
  - Combination of control-flow and dataflow-graph
  - Exposes instruction-level parallelism
- Performs HLS steps:
  - Allocation
  - Scheduling
  - Binding



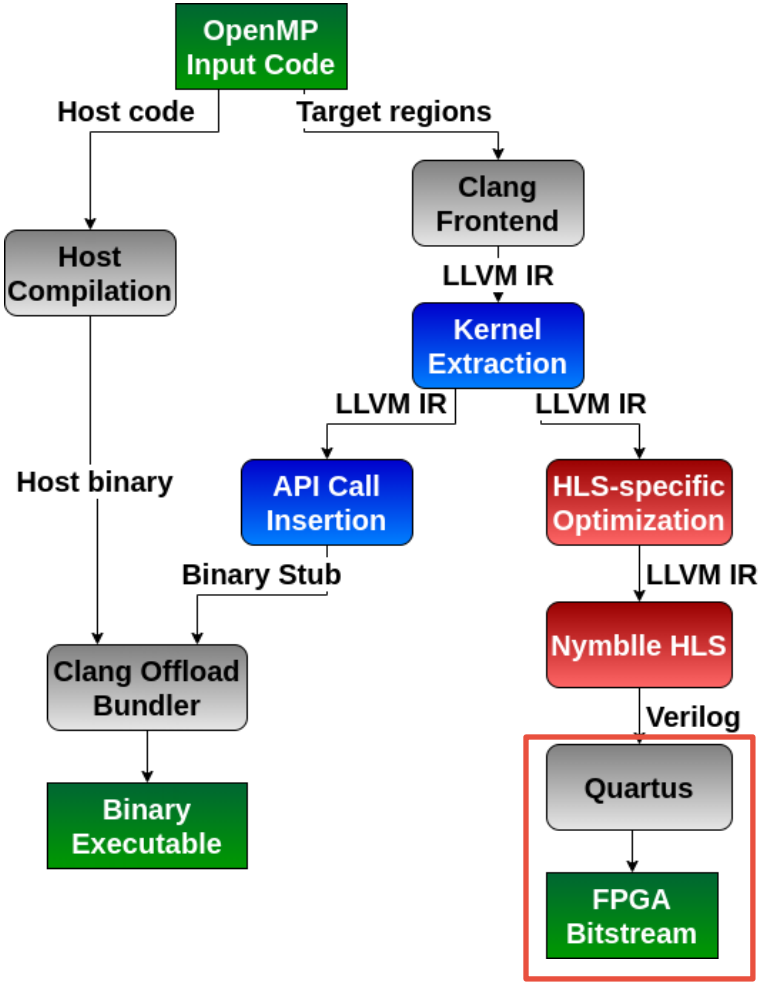
Compilation Flow – Nymble HLS Backend

- Performs additional optimizations
  - Construction of static regions to reduce controller logic
  - Placement of thread reordering stages
- Generates Verilog design containing datapath and complete controller logic (token-based)
- Accelerator allows simultaneous execution of multiple HW threads



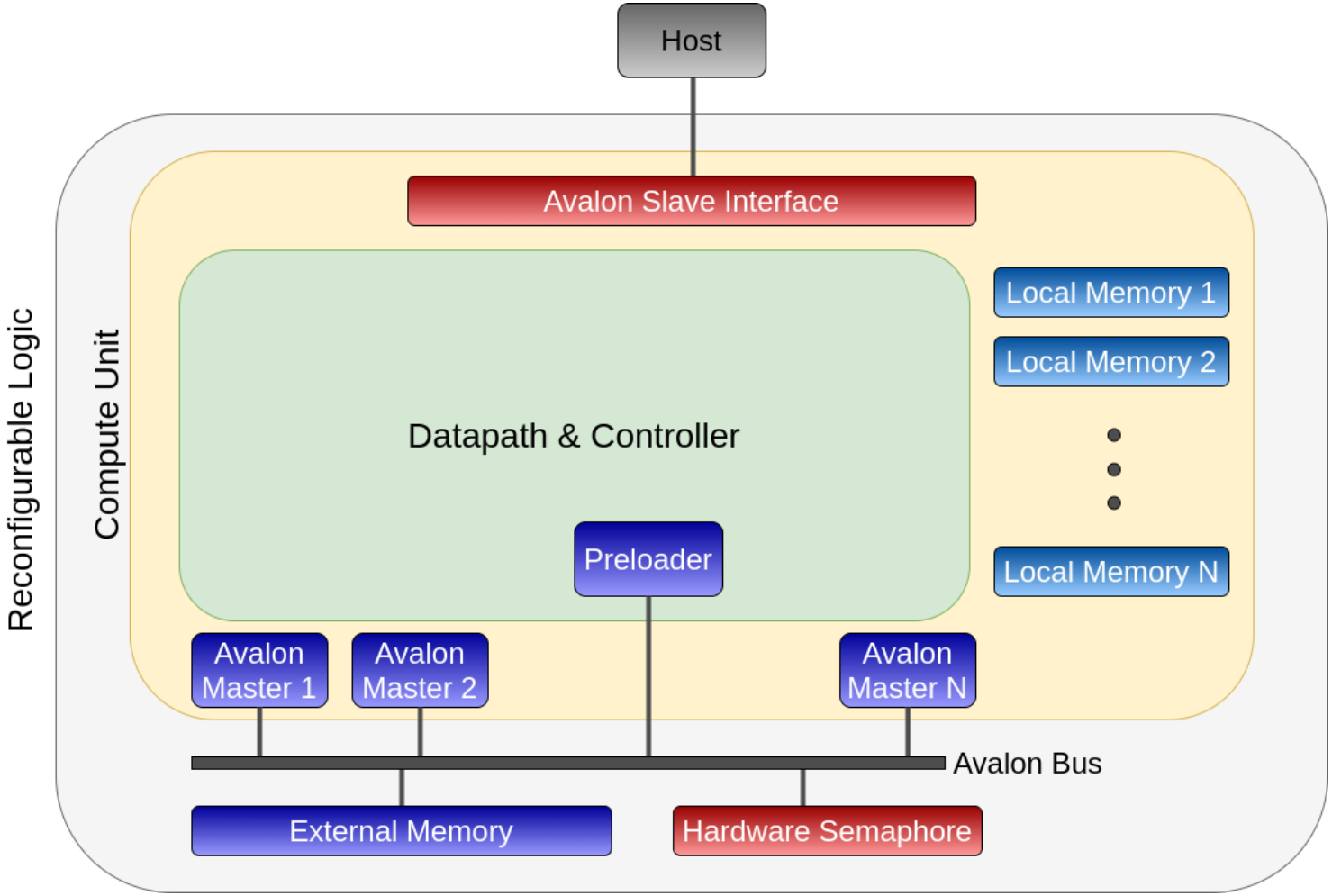
Compilation Flow – Bitstream Synthesis

- Embeds accelerator into OPAE shell design
- Performs FPGA bitstream synthesis using vendor tools
  - Translates Verilog generated by Nymble HLS into bitstream
- Bitstream pre-loaded onto FPGA before application execution



HARDWARE ARCHITECTURE

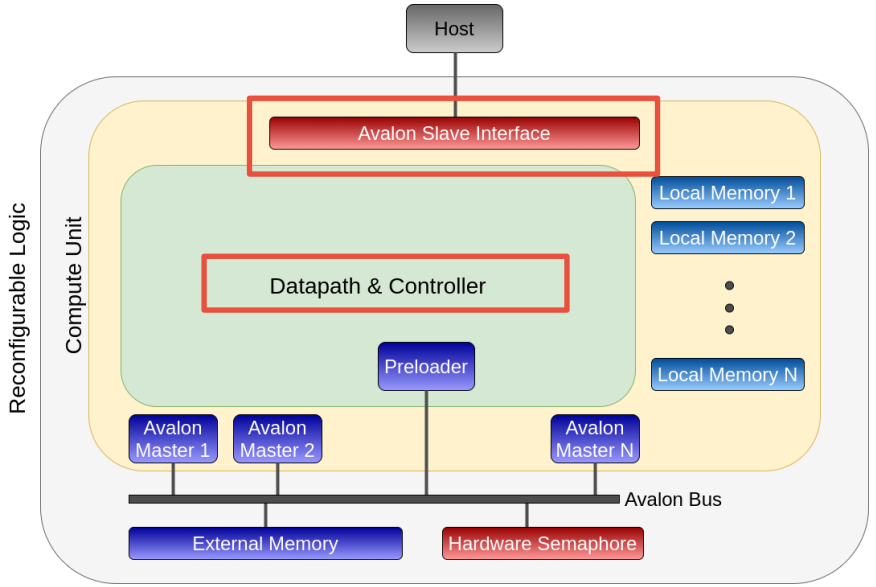
Hardware Architecture



Hardware Architecture - Datapath

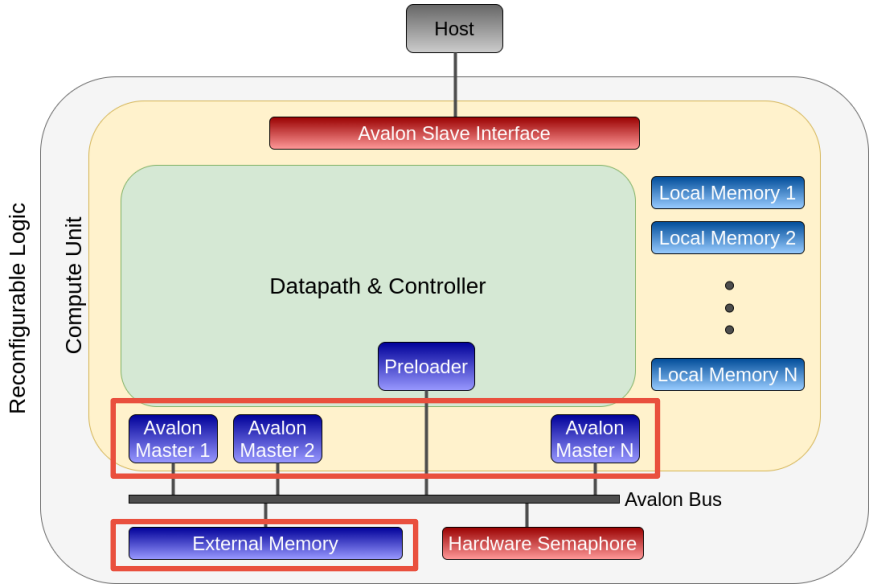
- Datapath & Controller completely generated by Nymble HLS backend as Verilog code

- Avalon Slave Interface
  - I/O-mapped registers
  - Used by Intel OPAE to pass parameters to hardware accelerator



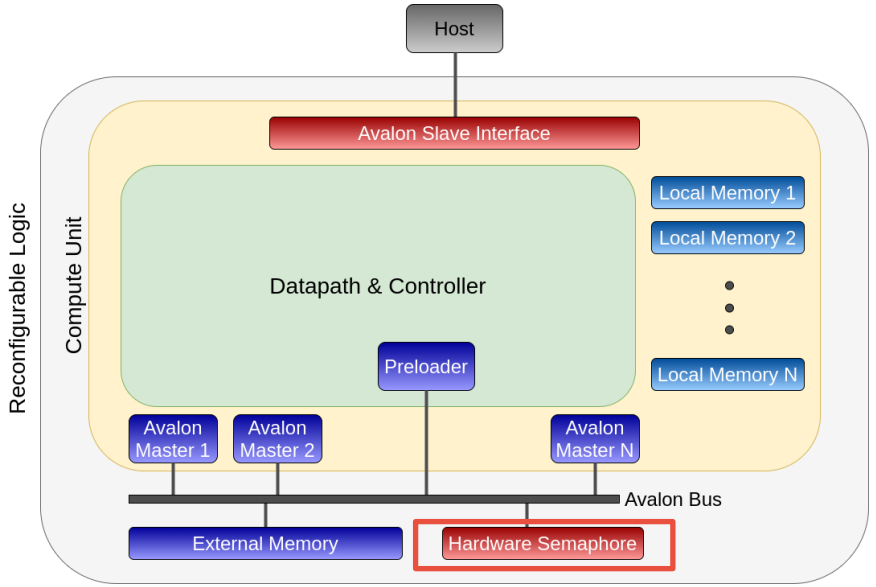
Hardware Architecture – External Memory

- External DDR memory on FPGA board
  - Can also be used to exchange data with host via OpenMP data management clauses
- Avalon Masters
  - One master interface per hardware thread
  - Allows to issue multiple transfers in parallel, re-ordering can occur



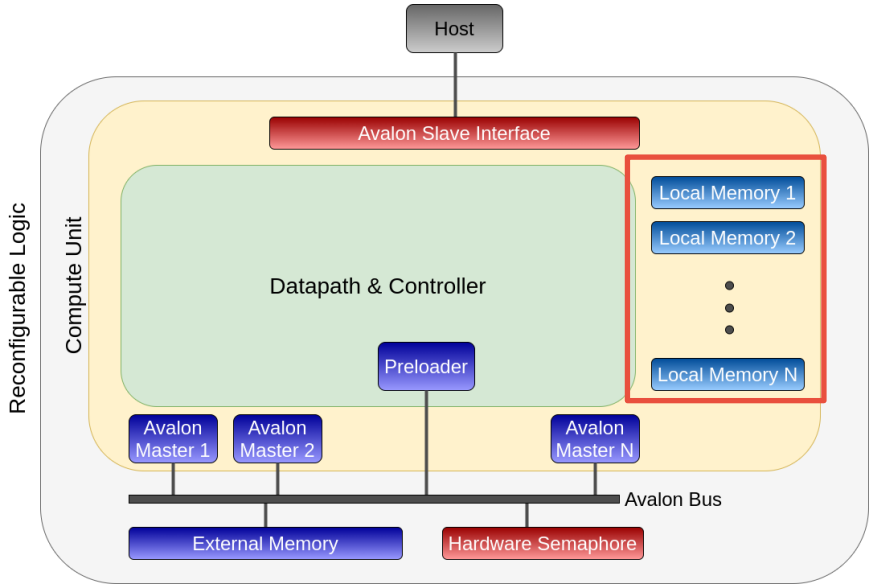
Hardware Architecture – Hardware Semaphore

- Soft IP from Intel IP library
- Connected via bus, accessed with load/store via Avalon master interface
- Used to realize synchronization constructs
  - Critical regions
  - Barriers



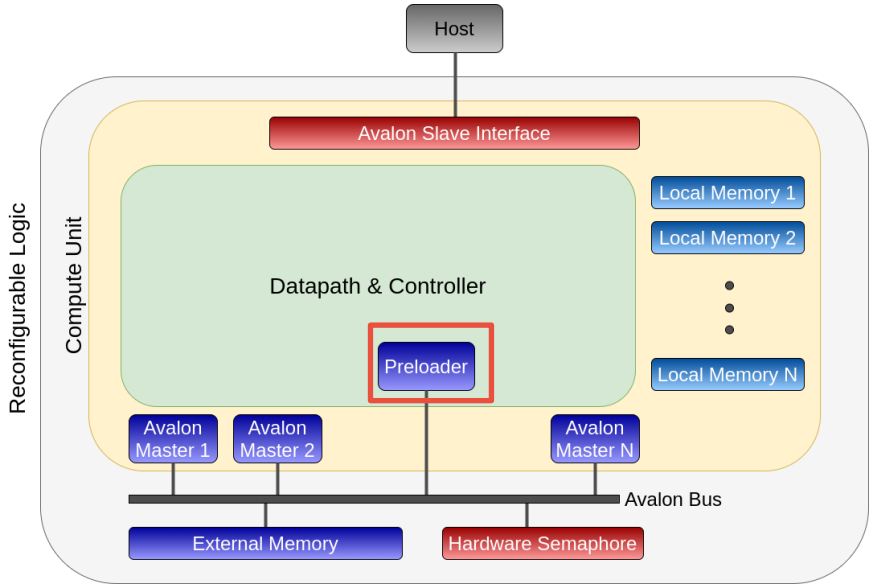
Hardware Architecture – Local Memories

- Small on-chip memories
- One memory per thread
- Very fast access (1-2 clock cycles)
- Not directly accessible from host, cannot be used to exchange data with host
- Ideally suited for intermediate results or pre-loading of data from external memory



Hardware Architecture – Preloader

- Specialized unit for efficient loading of data from external memory to local memory
- More efficient than in-code loop
- Automatically instantiated for template function **`omp_target_preload`**



Preload Function

```
template<typename T, int N>
void omp_target_preload(size_t offset, size_t stride,
  size_t num_transfers, void* globalSrc, void* localDst)
```

- Execute **`num_transfers`** transfers with **`N`** elements of type **`T`** from external memory at **`globalSrc`** to local memory at **`localDst`**
- First transfer uses **`offset`**
- Skip **`stride`** elements between every transfer
- Use of template function allows to optimize preloading unit for efficient usage of memory bandwidth

EVALUATION

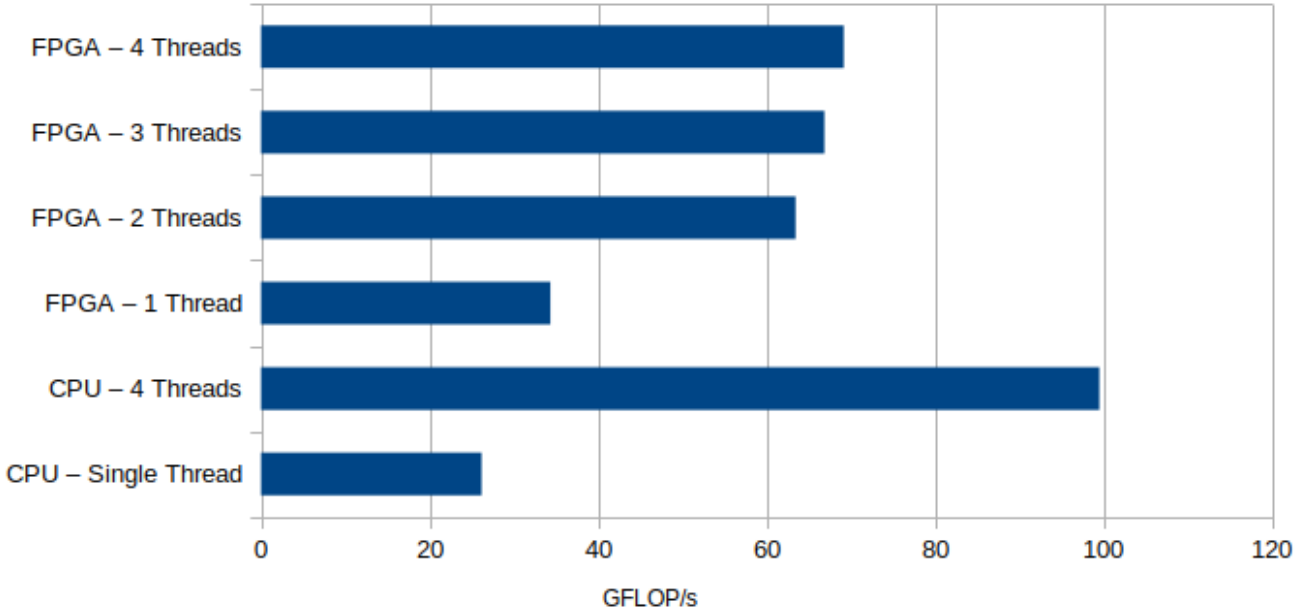
Evaluation Setup

- OpenMP frontend implementation based on LLVM 9
- Intel Quartus Prime 18.1.2 used for FPGA bitstream synthesis
- Evaluation system
  - Intel HPC-scale FPGA card (PAC D5005, Stratix 10)
  - Xeon Gold 5122 CPU (4 cores)
  - Connected via PCIe 3.0, 16x

GEMM Benchmark

- Well understood kernel, common in many applications
- Implementation details
  - Blocked version of GEMM
  - Double buffering in local memory
    - Uses hardware pre-loader
    - Computation and pre-loading of next block happen in parallel
  - Partial unrolling & pipelining of computation loop to exploit spatial parallelism
  - Up to four hardware threads active simultaneously
- Performance comparison with ATLAS BLAS library (four threads)
- Matrices size: 8192 x 8192 elements single-precision float (256 MiB)

GEMM Evaluation



- Accelerator hardware footprint: 14% of logic resources
- Operating frequency 183 MHz
- CPU vs. FPGA
  - CPU: 4 threads on **4 cores**, 19-119 GiB/s memory bandwidth
  - FPGA: 4 threads on **single core**, ~11.7 GiB/s memory/bus bandwidth

Conclusion

- Prototype of OpenMP device offloading for FPGAs
- Interaction with HLS based on (LLVM) IR instead of source-level
  - Allows for optimizations and transformations crossing border between frontend and HLS backend
- OpenMP parallelism mapped to simultaneous multi-threading in FPGA accelerator
- LLVM **`libomptarget`** plugin based on Intel OPAE
  - Supports full set of data management constructs and clauses, including map-type specifier
- Hardware architecture supports local scratchpad memories and thread synchronization


TECHNISCHE
UNIVERSITÄT
DARMSTADT

THANK YOU!

Embedded Systems & Applications