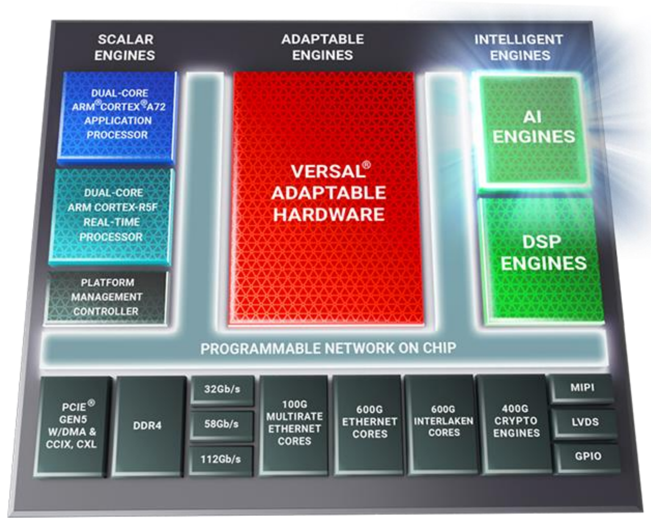
DAC Young Fellows

PolyAIE: A Dataflow Compiler for Heterogeneous Compute Platforms

Hanchen Ye (hanchenye.com), *Advisor:* Prof. Deming Chen, *Affiliation:* UIUC ECE

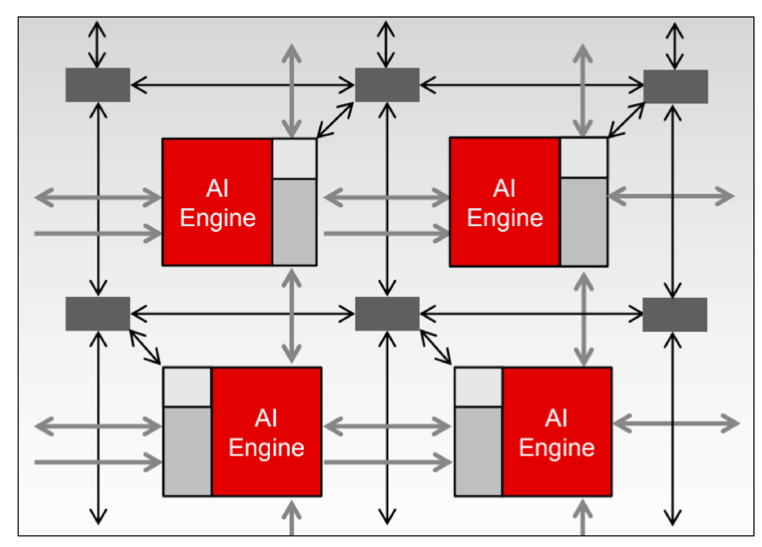
AMD-Xilinx Versal ACAP Architecture



Scalar Engine: Dual-Core ARM CPU
Adaptable Engine: FPGA
Intelligent Engine: AI-Engine Array

Three compute engines and hard IPs, including PCIe and DDR controllers, can communicate with each other through a **Network-on-Chip (NoC)**.

AI-Engine (AIE) Array Architecture [1]



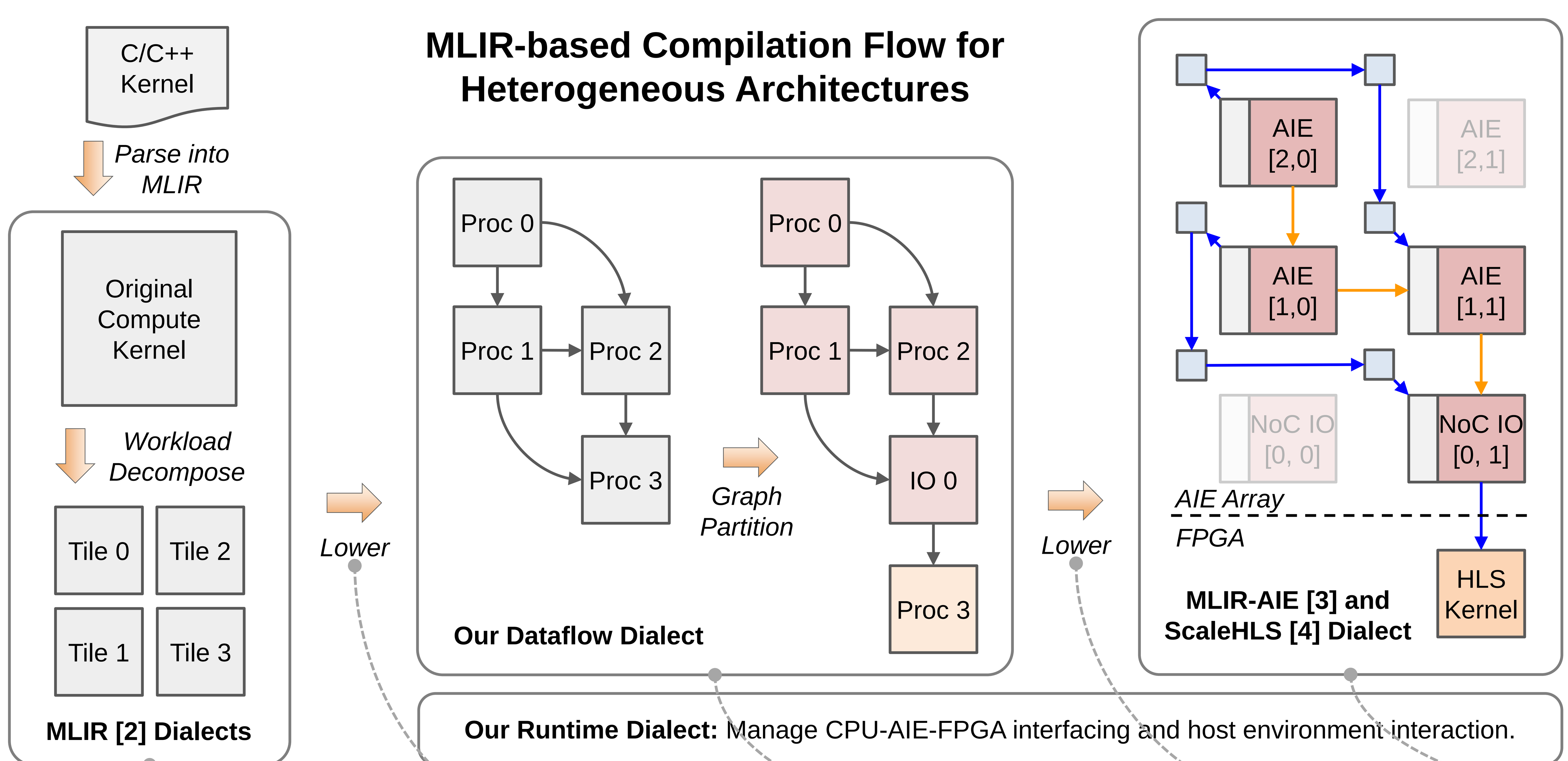
Each AIE has:

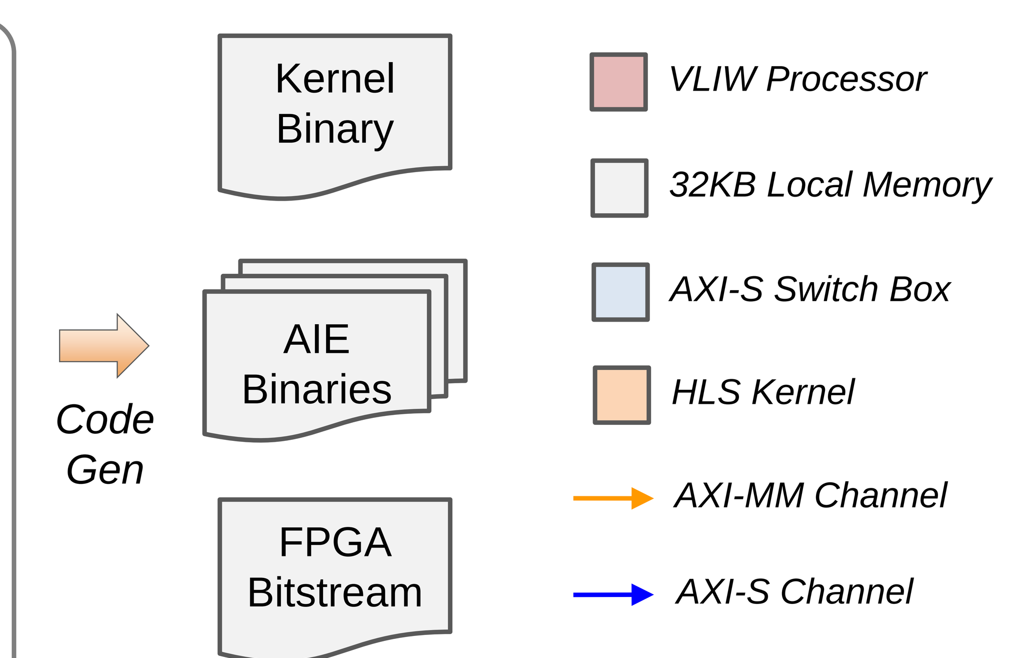
- A 32KB local buffer
- A VLIW core that can process 128 INT8 MACs per clock cycle at 1GHz

Communication between AIEs:
Adjacent AIEs: Local buffer sharing
Non-adjacent AIEs: An AXI-S network with configurable AXI-S switches

How to program it?

- There's a huge gap between the heterogeneous hardware and applications, e.g., NN models.
- How to model the hardware at a high level and distribute the workloads efficiently?
- How to manage the computing, memory, and I/O resources of different hardware components?





[1] Samuel Bayliss. "Compiling for Xilinx AI Engine using MLIR." C4ML Workshop, 2020.
[2] Chris Lattner, et al. "Mlir: Scaling compiler infrastructure for domain specific computation." CGO, 2021. https://mlir.llvm.org/
[3] Stephen Neuendorffer, et al. "The Evolution of Domain-Specific Computing for Deep Learning." MCAS, 2021. https://xilinx.github.io/mlir-aie/
[4] Hanchen Ye, et al. "ScaleHLS: A New Scalable High-Level Synthesis Framework on Multi-Level Intermediate Representation." HPCA, 2022. https://github.com/hanchenye/scalehls

Phase 1
- Workload decompose
- Polyhedral-based loop and memory access optimization

Phase 2
- Dependency extraction
- Dataflow graph generation
- Runtime generation

Phase 3
- Dataflow graph partition
- Interface node insertion
- Placement

Phase 4
- Dataflow to AIE conversion
- Dataflow to HLS conversion
- Intra-node optimizations

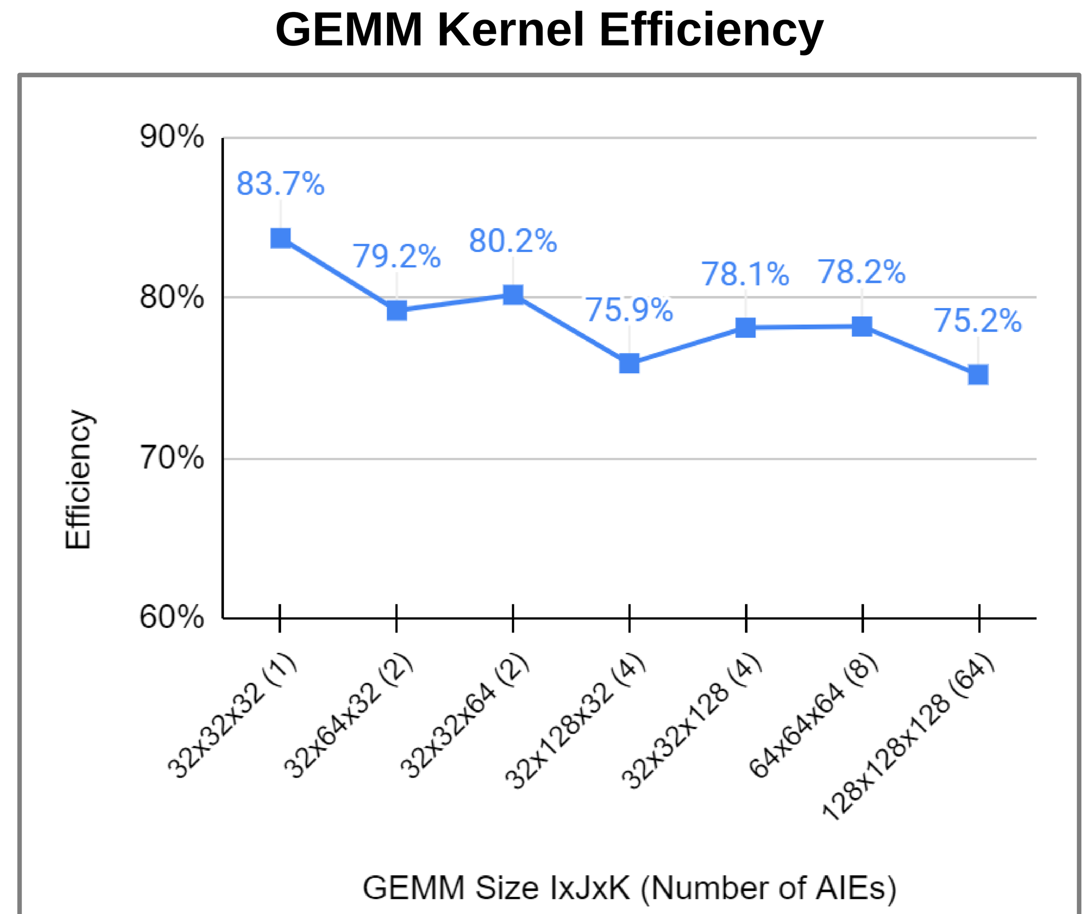
Phase 5
- Buffer and lock allocation
- DMA descriptor generation
- *Routing (using MLIR-AIE)*

Initial Experimental Results

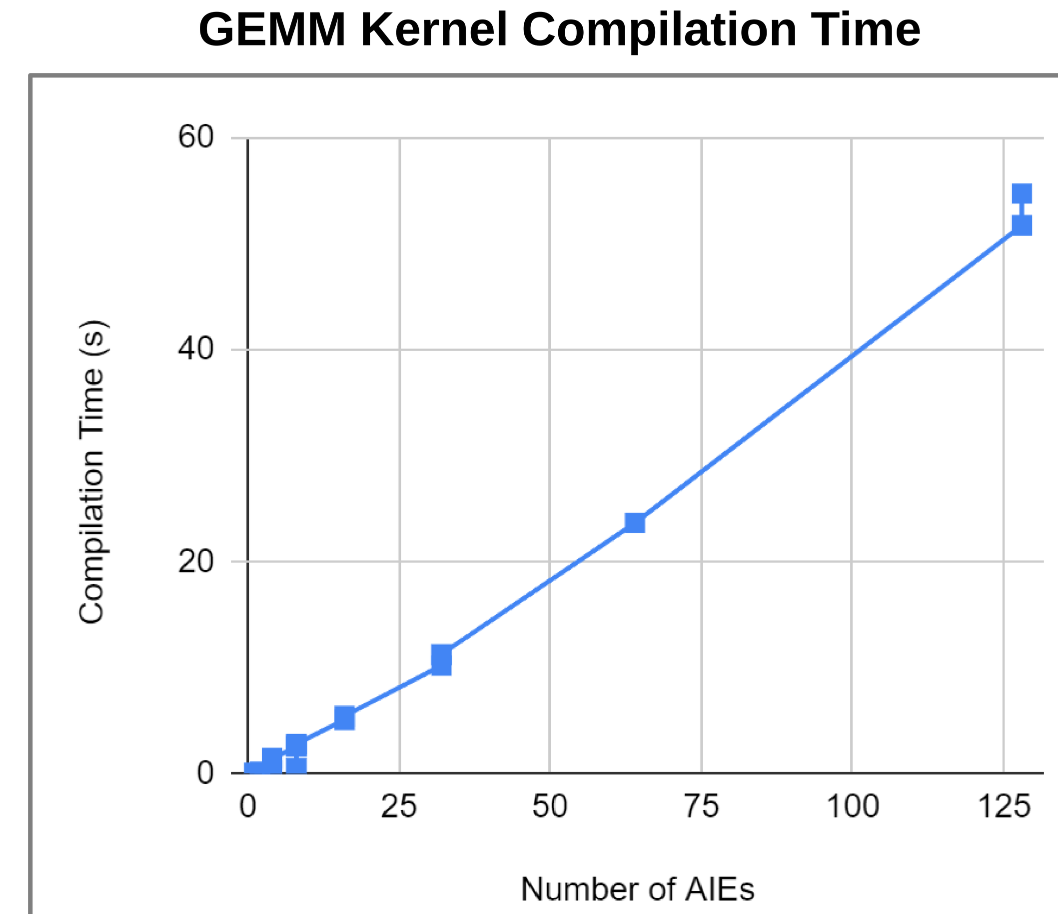
The binaries of GEMM kernels generated by PolyAIE are offloaded to an AMD-Xilinx Versal **VCK190** board for evaluation.

- **Single AIE efficiency:** A 32x32x32 GEMM kernel is mapped to one AIE and achieves 83.7% efficiency compared with the maximum possible performance of one AIE under 1GHz.
- **Multiple AIE efficiency:** A 128x128x128 GEMM achieves an efficiency of 75.2% on 64 AIEs. The efficiency drop mainly comes from the cost of data movement and synchronization between AIEs.
- **Compilation time:** The proposed dataflow layer models the hardware at a proper granularity, which supports efficient transforms, such as placement, while avoiding the complexity brought by redundant hardware details. Meanwhile, the intra-node optimizations can be applied in parallel. Overall, the compilation time increases linearly as the number of AIEs increases.

GEMM Kernel Efficiency



GEMM Kernel Compilation Time



Special thanks to Stephen Neuendorffer, Kristof Denolf, Jack Lo, and Samuel Bayliss from AMD-Xilinx and Prof. Peipei Zhou and Jinming Zhuang form University of Pittsburgh!