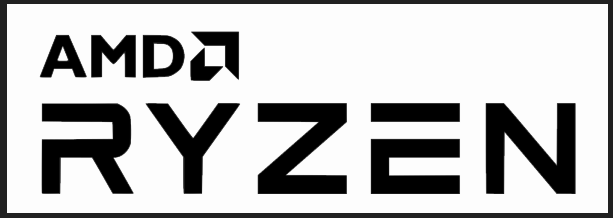
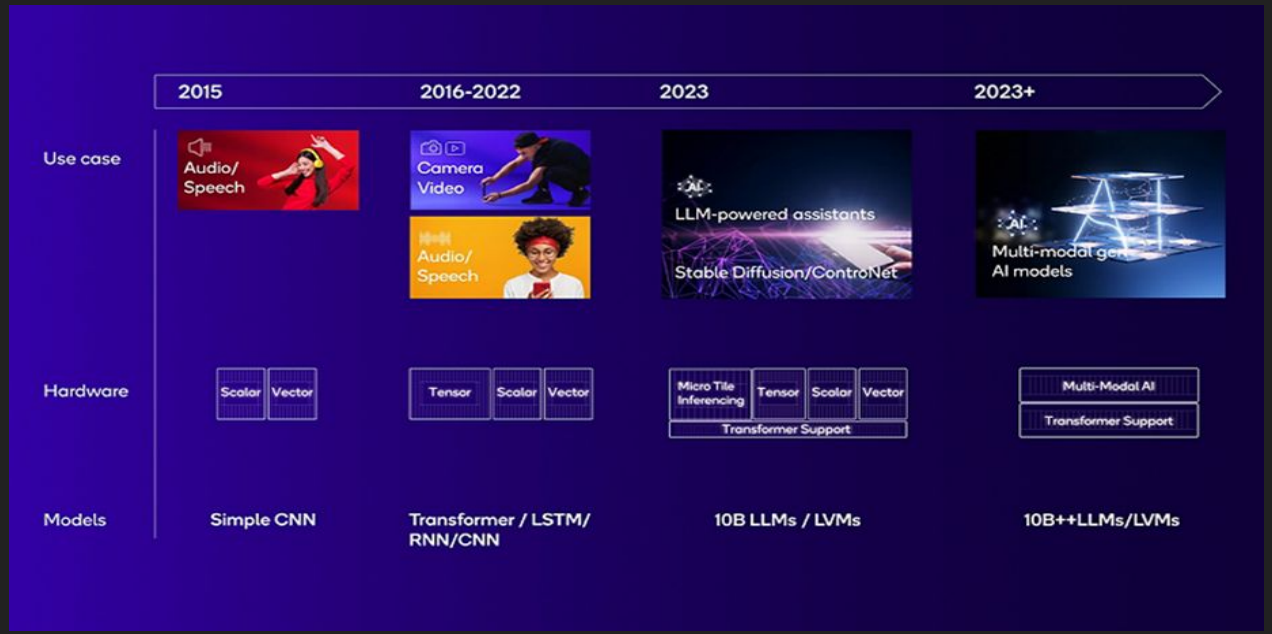


Exploiting Structured Sparsity for Efficient Inference on Client/Edge NPUs

Yaswanth Raparti
Design Engineering Manager
RyzenAI
AMD

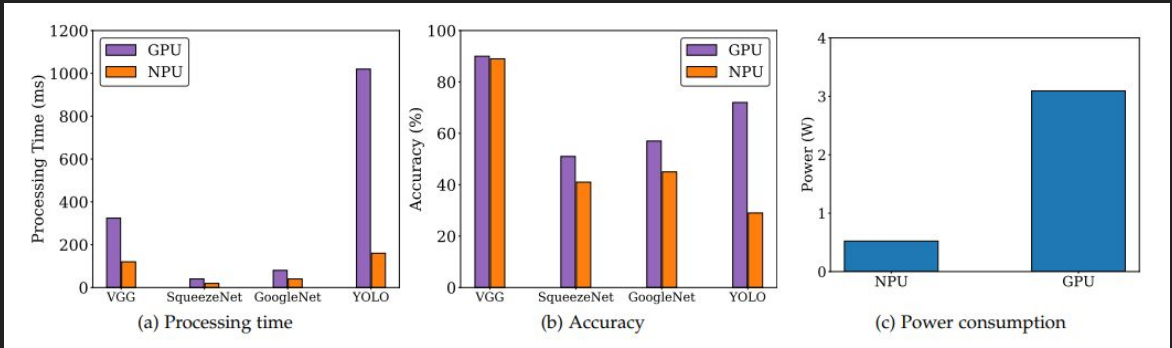
AI Acceleration on Client/Edge Devices

- Wide range of AI/ML applications in edge and client devices
- Object detection, image generation, translation, recommendation etc.
- Generative AI models with billions of parameters
- Latency critical and power constrained



Efficiency of Inference Accelerators

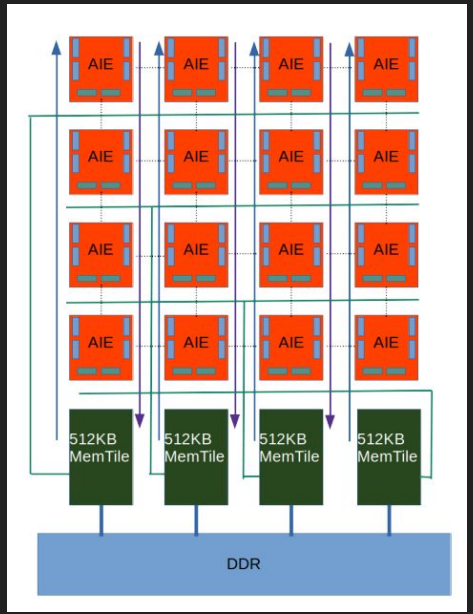
- GPUs are designed to hide latency with several thread blocks scheduled on streaming multi-processors
  - Poor in energy efficiency
  - Generates temperature hotspots
- Inference on client devices require high energy efficiency
  - Measured in TOPS/W
- NPUs are application specific instruction processors (ASIP)
  - Tailor made for low control overhead, and high instruction throughput



T. Tan et al. “Thermal-Aware Scheduling for Deep Learning on Mobile Devices with NPU”, IEEE TMC 2020

NPU Architecture

- Array of AI Engine cores (arranged in a grid)
  - Each core has VLIW pipeline with SIMD instructions
  - 8x8x8 MatMul instructions
  - Computes in int4/Int8/fp16/bfp16 data type
  - 64KB data memory
  - 16KB program memory
- A row of Memory tiles
  - 512KB each with 64KB banks
  - Each memory tile supports a column of AIE cores
  - Connected to cores using interconnect streams (4 NB, 4 SB)
  - DMA engine supports 4D tiled Rd Wr access patterns
- A row of Shim tiles
  - Connects memory tiles with DRAM and external controller
  - DMA engine supports 3D tiled Rd Wr access patterns

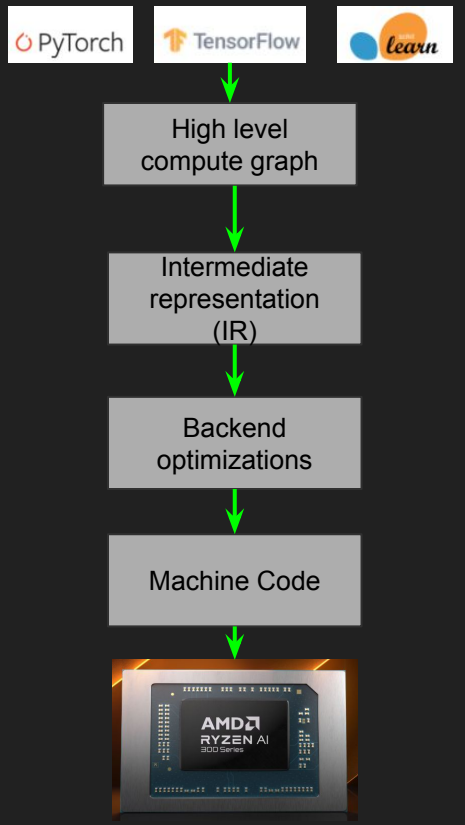


AI Engine Array

Inference on NPU

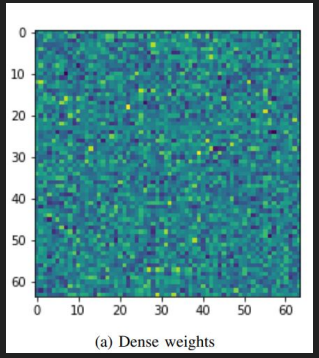
- NPUs are designed for energy efficiency
  - Lack of hardware caching schemes
  - Low-bit precision math
    - Mostly support 4/8/16-bit data-types
  - Lack of non-linear hardware functions
  - Restrictions on data ordering (tiling)
  - Shared DRAM bandwidth
  - Programming models not standardized

- Need for powerful ML compilers to map applications on NPU
  - Graph level optimizations to save DRAM bandwidth
  - Explore the possible tilings for each operation
    - Aim for high data-reuse
  - Dataflow scheduling for efficient compute and dataflow
    - Reduce the control overhead and maximize utilization
    - Resolve deadlocks and starvation

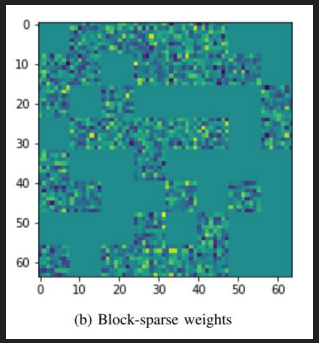


Exploiting Sparsity in NPUs

- Most neural network layers are compute/memory intensive
  - Model compression techniques are used to reduce the BW load
- Sparsity is supported by different training and inference platforms
  - Random sparsity N:M (Dense sparsity)
  - Structured sparsity (coarse grained $m_x n$ blocks)
- Random sparsity
  - Training is well explored (Hubara et al. NeurIPS 2021)
  - Achieves high compression
  - Requires specialized hardware for computation
  - High overhead in decoding
- Structured sparsity
  - Training is challenging
  - Low overhead
  - Does not required specialized hardware



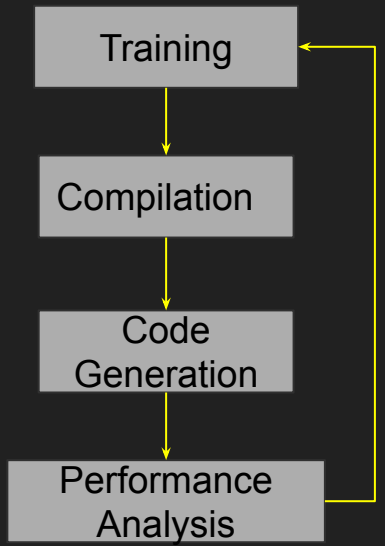
(a) Dense weights



(b) Block-sparse weights

We use structured sparsity to avoid the need for specialized hardware

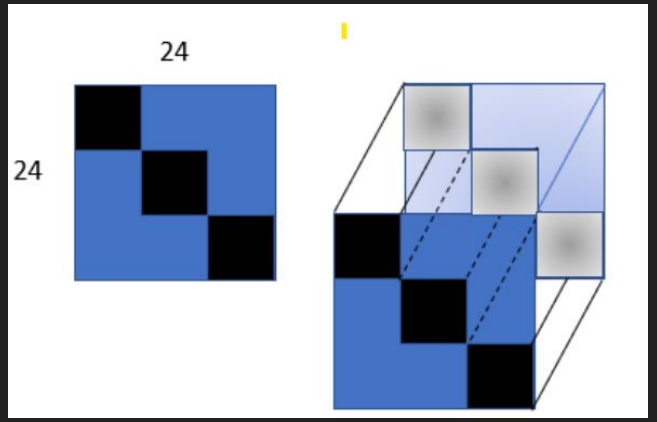
Exploiting Structured Sparsity on AIE



- Training on pre-trained network
  - choose block size
  - Sparsity ratio
- Compilation for each layer
  - Tile the activation and weights per column
  - Create subvolumes
  - Compress data
  - Maximize reuse
- Code generation
  - Runtime DMA configuration
  - Minimize control overhead
- Performance Analysis
  - Measure achieved speedup
  - Feedback to training module to pick different block size

Training for Block Sparsity

- Block sizes are multiples of 8x8
  - Corresponds to the HW MatMul granularity
  - Keep inner dimension contiguous
  - Helps with spatial locality
    - E.g. weights [Ic, H, W, Oc] -> weights' [Ic', H, W, Oc']

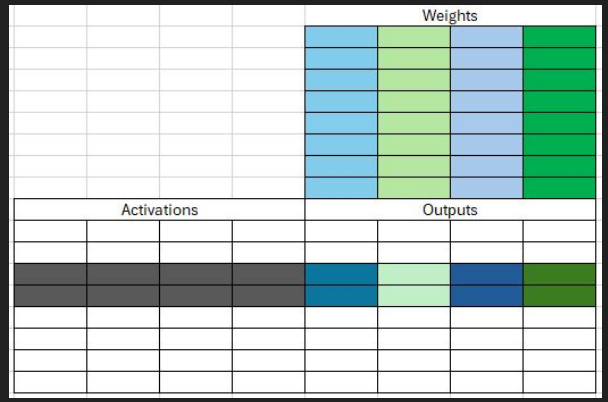


- Searching for optimal sparse ratio
  - Sparse ratio as incremental
    - Start with minimum sparse ratio, increment no. of sparse blocks till max sparsity is reached
  - Sparse ratio as a learnable parameter
  - Predetermined sparsity ratio and full training

Compilation Framework

- Process each layer to find
  - Suitable memory tile allocation
  - Subvolume determination
  - Dataflow schedule
- Memory tile allocation
  - Activations, weights and outputs share mem-tile
  - Break down the op into subvolumes that fit in mem-tile and AIE core
- Schedule subvolumes to maximize reuse
  - Compress the subvolumes
    - Each subvolume has dense data and a mask
- At the end of each layer
  - Check if output can fit in mem-tile for reuse in next layer
  - Or evict the output to DDR
- Dataflow Schedule
  - Generate a graph of instructions that represent dataflow
  - Represented using IR

Activation Reuse

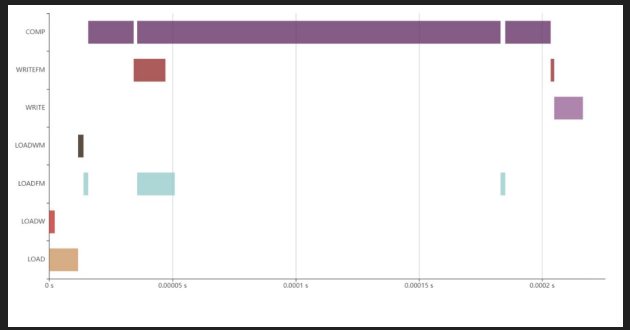


Code Generation

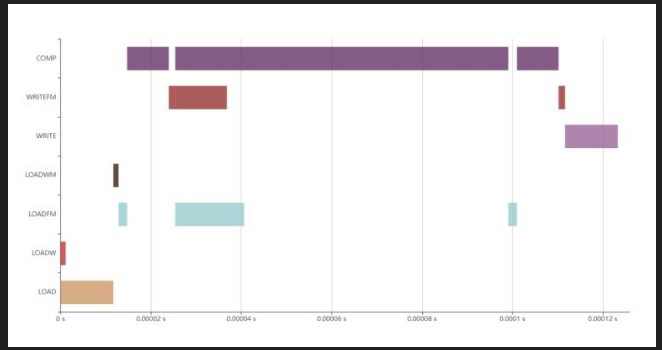
- Takes dataflow IR as input
- Generate runtime control code
  - Programs DMA controllers to read/write data
  - Kernels are programmed at initialization
- Code generation goals
  - Minimize runtime control overhead
  - Dynamic management of DMA channels
  - Maximize resource utilization
  - Avoid deadlocks
- DRC checks
  - Run the DRC tool to catch potential failures
  - Register overflows
  - Buffer overwrites
  - Deadlocks

Experimental Results

- Block size is set to 8x8
- Evaluated using CNNs
  - Conv layers are more complex than GeMMs
  - GeMM can be a special case of 1x1 Conv
- Observed ~50% improvement in latency 0.5 sparse ratio



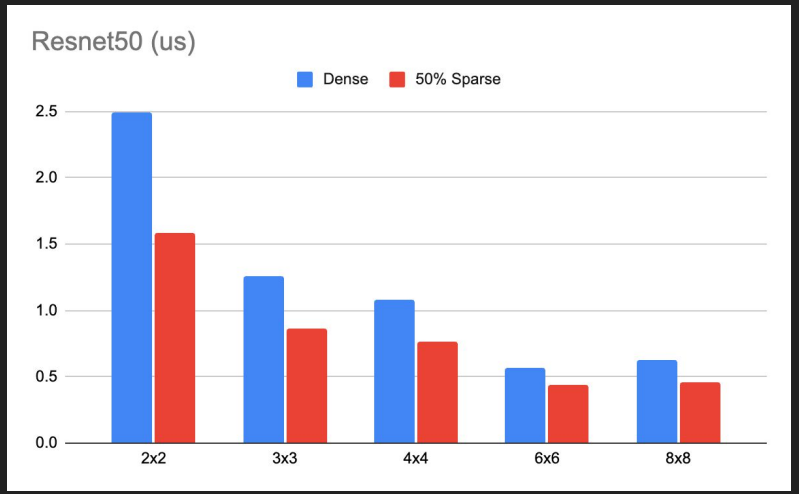
Dense Conv

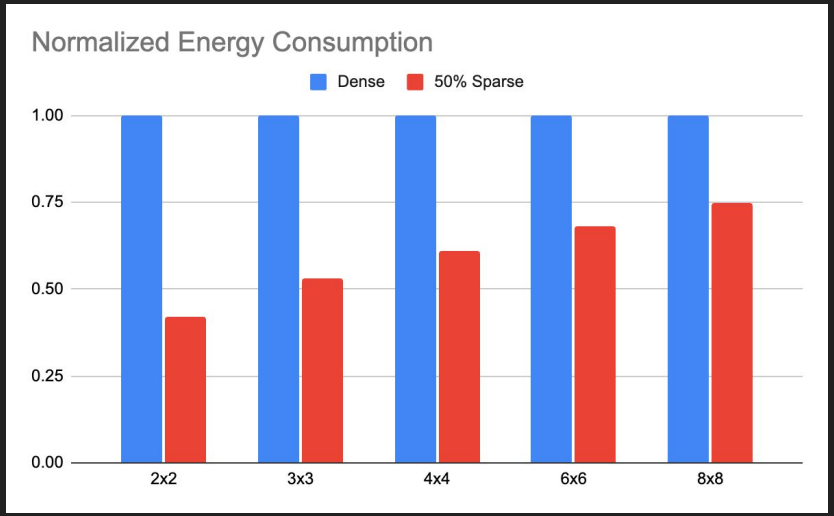


50% sparse conv

E2E Latency & Energy

- ~40% improvement in latency
- ~60% improvement in energy consumption
- Usage of smaller grids help exploit sparsity better due to pipelining of compute and dataflow





Thank you