VLIW Scheduling: Timing Is All You Need

Tamino Steinert
Friedrich Schiller University Jena
Jena, Germany
tamino.steinert@uni-jena.de

Alexander Breuer
Friedrich Schiller University Jena
Jena, Germany
alex.breuer@uni-jena.de

***Abstract*—AMD's XDNA2 neural processing unit (NPU) holds 32 compute tiles in an 8×4 array, each a six-issue VLIW core where the schedule must resolve nearly all pipeline hazards. Each compute tile achieves 1843 GFLOPS by always issuing a `VMAC.F` operation, an $8 \times 8 \times 8$ block-floating-point (BFP16) matrix multiply-accumulate. This corresponds to an array-wide performance of 59 TFLOPS.**

We present a hand-scheduled BFP16 tensor-contraction kernel that saturates a core with `VMAC.F` operations in its steady state. To issue `VMAC.F` operations without gaps, input-tile reuse is required, forcing a 2×2 output block. Standard double buffering of the four output tiles needs eight accumulator registers, whereas a compute tile's core has only five. The five-register constraint is solved by time-multiplexing the block's four output tiles onto two accumulator registers and using the remaining three for buffering.

The presented kernel achieves 1777 of the 1843 GFLOPS compute peak on a single compute tile out of hot L1 scratchpad memory, $5\times$ the mlir-aie `mm_bfp.cc` kernel's performance. With the operands streamed from DRAM, a pre-tiled M3840×N3840×K8192 matrix multiplication reaches 1518 GFLOPS on one compute tile and 21.4 TFLOPS on the whole NPU — $4.4\times$ and $2\times$ the mlir-aie kernel's performance.

***Index Terms*—Neural processing unit, AMD XDNA2, Ryzen AI Max, VLIW, Instruction scheduling, Register allocation, Block floating point, Tensor contraction**

I. The XDNA2 NPU

XDNA2 is the second-generation neural processing unit (NPU) architecture of AMD's Ryzen AI processors [1]. An XDNA2 NPU is a 2D array of tiles: an 8×4 grid of compute tiles above a row of eight memory tiles and a DRAM-connected shim tile row. Each compute tile is a six-issue VLIW core with 64 KiB of L1 data scratchpad memory. A core issues one instruction of up to six fixed-latency operations each cycle, requiring software to resolve nearly all pipeline hazards through scheduling. The NPU can achieve 1024 FLOP/cycle on each compute tile by issuing `VMAC.F` operations, each an $8 \times 8 \times 8$ block-floating-point (BFP16) matrix multiply-accumulate. This corresponds to 59 TFLOPS array-wide at 1.8 GHz. We present a tensor-contraction kernel issuing a `VMAC.F` in every instruction in its steady state.

II. Tiled Data Layout

The workload of the kernel is described by the einsum $m_2 k_1 m_1 m_0 k_0,\ n_2 k_1 n_1 n_0 k_0 \rightarrow m_1 m_2 n_2 n_1 m_0 n_0$. The tensor notation is generalized row-major: dimensions are written in descending order of their strides. The sizes $|m_0|$, $|n_0|$, and $|k_0|$ are fixed at 8, the shape of one `VMAC.F`; $|m_1|$ and $|n_1|$ are 2, forming the output block (see Section III); $|m_2|$, $|n_2|$, and $|k_1|$ are runtime parameters. For this work we set $|m_2|$, $|n_2|$, and $|k_1|$ to 6, 5, and 8, which is equivalent to a pre-tiled M96×N80×K64 matrix multiplication.

III. Tensor-contraction kernel schedule

A `VMAC.F` consumes two 576-bit 8×8 BFP16 input tiles, and the hardware possesses two 512-bit load units. Therefore, in-register reuse of input tiles is required to issue one `VMAC.F` per cycle. The 2×2 output blocking in Fig. 1 uses each loaded input tile twice, reaching a `VMAC.F`-to-load ratio of 2:1 per unit.

An 8×8 FP32 output tile fills exactly one 2048-bit accumulator register. The 2×2 block therefore needs four accumulator registers, and standard double buffering would need eight while a core has only five [2].

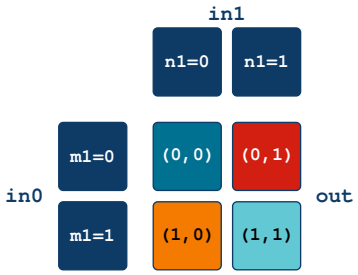


Fig. 1. The 2×2 output blocking used by our XDNA2 tensor-contraction kernel. Two 8×8 BFP16 tiles of each input (`in0` and `in1`) are loaded to perform four `VMAC.F` operations which accumulate onto the 2×2-blocked FP32 output tiles of size 8×8. This achieves an input reuse of two, which is greater than the required 1.125.

We solve the five-register constraint by time-multiplexing four output tiles onto two registers. Each tile of the output block is computed by an accumulation chain of dependent `VMAC.F` operations stepping through k_1 every four cycles. `VMAC.F` has a six-cycle latency, but reads its accumulator in the first half of cycle 4 and writes it in the second half of cycle 6. Between the read and write, the new partial sum is computed in the pipeline, allowing the register to hold a different value in between. Two chains issued two cycles apart can share one register (Fig. 2): each read lands in the half-cycle before the write that overwrites the register. Two registers then hold all four partial sums.

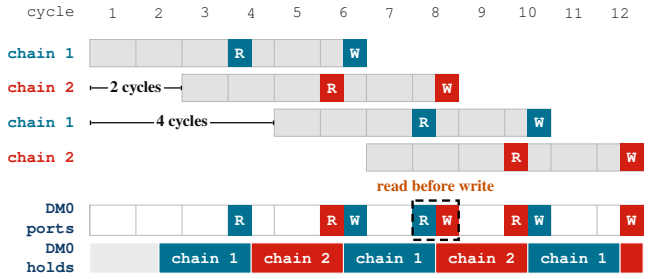


Fig. 2. Two chains sharing accumulator register `DM0`. Each chain issues every four cycles, and a `VMAC.F` reads its accumulator in the first half of its fourth cycle and writes it in the second half of its sixth. Offsetting the second chain by two cycles therefore puts every read in the half-cycle before the write that displaces it, so the register alternates between the two partial sums and each chain reads back its own.

IV. Evaluation

We benchmark our kernel and compare it against mlir-aie's `mm_bfp.cc` kernel [3] on a Ryzen AI Max PRO 390 SoC at its 1.8 GHz turbo clock. The `mm_bfp.cc` kernel was adapted to match our FP32 output and compiled with the same total shape (M96×N80×K64) using the *xchesscc* compiler of Ryzen AI Software 1.3.1 [4].

The single-compute-tile results are shown in Table I. On a single compute tile and out of hot L1 scratchpad memory, our kernel obtains 1777 GFLOPS and runs 5× faster than `mm_bfp.cc`. With the operands streamed from DRAM for a pre-tiled M3840×N3840×K8192 matrix multiplication, it falls short of that hot-memory rate. Measured without compute, the DMA engines sustain 25.4 GB/s for this access pattern. With an on-device arithmetic intensity of 77.6 FLOP/B, the memory ceiling is 1971 GFLOPS. This indicates that the lower performance is caused by interference with the DMA engines' asynchronous data movement.

The 32-compute-tile benchmark results are shown in Table II. Our kernel is memory-bound, and the end-to-end gap to `mm_bfp.cc` narrows from 4.4× on a single compute tile to 2× on the whole NPU Increasing the output tiling size would raise the arithmetic intensity, but the FP32 output buffer would no longer fit into two of the four 16-KiB banks, which could cause bank conflicts. This tiling with double-buffered inputs uses 85.5% of the L1 scratchpad memory.

V. Conclusion and Future Work

Five accumulators sustain one multiply per cycle once a schedule trades registers for instruction-level timing. The kernel, benchmark harness, and a full derivation are publicly documented [5].

The array is bound by data movement, so further gains must come from moving fewer bytes or raising the arithmetic intensity on the L1 scratchpad memory. Therefore, alternative variants may use a smaller output data type. Future work could also integrate this kernel, with other hand-written variants, into a tensor framework that compiles a tensor contraction specified by an einsum to the NPU.

The gained insight into XDNA2 VLIW scheduling could also be used to improve kernel performance on AIE-ML v2 [6], the AI Engine architecture of AMD's Versal adaptive SoCs, as both architectures share the same predecessor XDNA1/AIE-ML [7].

TABLE I

One compute tile of a Ryzen AI Max PRO 390 at 1.8 GHz, BFP16 in, FP32 out. Kernel shapes match a pre-tiled M96×N80×K64 matrix multiplication. End-to-end benchmarks perform the compute of a pre-tiled M3840×N3840×K8192 matrix multiplication.

One compute tile	**GFLOPS**	**% peak**
Operands L1-resident		
Compute peak: 1024 FLOP/cycle	1843	100.0%
This work	**1777**	**96.4%**
`mlir-aie/mm_bfp.cc`	353	19.1%
Streamed from DRAM		
This work, end to end	**1518**	**82.4%**
`mlir-aie/mm_bfp.cc`, end to end	344	18.7%

TABLE II

All 32 compute tiles, pre-tiled M3840×N3840×K8192 matrix multiplication. Shares are of the tiling's memory ceiling as the array is DRAM-bound at the tiling's on-device arithmetic intensity.

Whole NPU (8 × 4 compute tiles)	**GFLOPS**	**% ceiling**
Memory ceiling: 386 FLOP/B × 55.3 GB/s	21,357	100.0%
This work, end to end	**21,352**	**100.0%**
`mlir-aie/mm_bfp.cc`, end to end	10,549	49.4%

Acknowledgments

This work was supported by the Carl Zeiss Foundation (project "Interactive Inference").

Claude [8] was used to assist with writing and editing this summary. The authors reviewed and verified all content and take full responsibility for it.

References

[1] A. Rico *et al.*, "AMD XDNA NPU in Ryzen AI Processors," *IEEE Micro*, vol. 44, no. 6, pp. 73–82, 2024, doi: 10.1109/MM.2024.3423692.

[2] AMD/Xilinx, "llvm-aie (Peano): LLVM back end and scheduling model for AI Engines." [Online]. Available: https://github.com/Xilinx/llvm-aie

[3] AMD, "mlir-aie: BFP16 matmul kernel aie_kernels/aie2p/mm_bfp.cc (commit #d71539a)." [Online]. Available: https://github.com/Xilinx/mlir-aie

[4] AMD, "Ryzen AI Software (version 1.3)," [Online]. Available: https://ryzenai.docs.amd.com/en/1.3/

[5] T. Steinert and A. Breuer, "Hello XDNA!: Microarchitecture Notes, Inferred ISA and Assembly Kernels," 2026. [Online]. Available: https://tnzr.org/xdna

[6] AMD, "Versal Adaptive SoC AIE-ML v2 Architecture Manual (AM027)," 2025. [Online]. Available: https://docs.amd.com/r/en-US/am027-versal-aie-ml-v2

[7] AMD, "Versal Adaptive SoC AIE-ML Architecture Manual (AM020)," 2024. [Online]. Available: https://docs.amd.com/r/en-US/am020-versal-aie-ml

[8] Anthropic, "Claude," 2026. [Online]. Available: https://claude.ai