# AMD XDNA NPU in Ryzen AI Processors

Alejandro Rico, *AMD, Austin, TX, 78735, USA*
Satyaprakash Pareek, *AMD, San Jose, CA, 95124, USA*
Javier Cabezas, *AMD, Austin, TX, 78735, USA*
David Clarke and Baris Ozgul, *AMD, D24 T683, Dublin, Ireland*
Francisco Barat, *AMD, 3000, Leuven, Belgium*
Yao Fu, *AMD, San Jose, CA, 95124, USA*
Stephan Münz and Dylan Stuart, *AMD, D24 T683, Dublin, Ireland*
Patrick Schlangen, *AMD, 50933, Cologne, Germany*
Pedro Duarte, *AMD, D24 T683, Dublin, Ireland*
Sneha Date, *AMD, San Jose, CA, 95124, USA*
Indrani Paul, *AMD, Austin, TX, 78735, USA*
Jian Weng, *AMD, Beijing, 100029, China*
Sonal Santan, Vinod Kathail, Ashish Sirasao, and Juanjo Noguera, *AMD, San Jose, CA, 95124, USA*

*The AMD Ryzen 7040 series is the first x86 processor with an integrated neural processing unit (NPU). The AMD XDNA technology in the NPU of Ryzen artificial intelligence (AI) processors provides optimized compute and memory resources for the needs for AI workloads and a specialized data movement architecture that allows to minimize bandwidth requirements leading to higher performance and efficiency. Across a selection of neural network benchmarks, XDNA provides between 4.3× and 33× better performance per watt, leading to extended battery life. The AI-optimized capabilities of the Ryzen 7040 NPU enable new AI experiences that are not possible without XDNA, making it a fundamental component in today's Ryzen-AI-powered devices and setting the foundation for an exciting roadmap toward future AI capabilities in mobile PCs.*

Artificial intelligence (AI) is a rapidly advancing field. Models become more compute and memory intensive, and use cases more ubiquitous. The current trends and projected requirements in the space motivate the use of specialized hardware for AI from the cloud to the edge. Many of the most exciting developments are happening in the client space with many applications taking advantage of AI for image processing,[1] video call enhancements,[2] and the upcoming holistic generative AI capabilities in Windows Copilot.[3]

0272-1732 © 2024 IEEE
Digital Object Identifier 10.1109/MM.2024.3423692
Date of publication 10 July 2024; date of current version 11 December 2024.

To satisfy the compute and capability needs of these use cases, AMD launched the AMD Ryzen 7040 processors, formerly codenamed "Phoenix." Featuring "Zen 4" CPUs, an integrated Radeon 700M GPU, and a first-of-its-kind neural processing unit (NPU), it is the first ever x86 processor with a dedicated AI acceleration engine.

The Ryzen 7040 NPU is the first instance of AMD XDNA technology. It includes innovative features and increased throughput compute engines for higher power efficiency to maximize battery life in mobile devices when running machine learning (ML) applications. XDNA not only enables these emerging AI workloads at the edge but also does it with improved privacy and security by avoiding sending sensitive data to the cloud. This design also provides higher connectivity resilience with cloud-to-edge transition of AI workloads when losing Internet connection,[4] and AI experience personalization by learning from its usage on the device.

The AMD XDNA compute, memory, and specialized data movement architecture are tailored to the characteristics of AI workloads. In this architecture, the parts of control that can be determined statically are moved to compile time. Data movement, by being explicit, allows to control and minimize memory bandwidth requirements. These features, among others, simplify the hardware resulting in higher performance and energy efficiency compared to more general-purpose dynamically scheduled designs, such as CPUs and GPUs.

In a set of neural network benchmarks, XDNA demonstrates performance per watt improvements between 4.3× and 33×, which leads to enabling enhanced AI experiences and better battery life.

## RYZEN AI PROCESSORS

The AMD Ryzen 7040 series is the first x86 processor with a dedicated AI accelerator engine. It is manufactured in TSMC N4 techonology with 15 metal layers and 25.4 billion transistors in 178 mm$^2$.[5] It integrates up to 8 "Zen 4" CPU cores with 64-kB L1 and 1 MB L2 caches per core, 16 MB of shared L3 cache, and a Radeon 700 M GPU with up to 12 graphic cores. The XDNA NPU makes the Ryzen 7040 series a first of its kind. CPU, GPU, and NPU share main memory through four LPDDR5x/DDR5 memory controllers totaling a 128-bit wide interface. With the maximum supported LPDDR5X-7500 memory, it provides a peak bandwidth of 120 GB/s. CPU, GPU, and NPU share main memory and allow zero-copy off-load of inference requests from the CPU to the NPU with the NPU having direct access to models and inference data (e.g., input/output activations) stored in main memory.

The XDNA NPU in Ryzen 7040 processors provides a performance of 10 teraoperations per second (TOPS). This performance, added on top of the AI TOPS provided by the CPU and GPU, result in a total of 33 TOPS at the system-on-chip (SoC) level.

## XDNA NPU ARCHITECTURE

XDNA is an evolution of the AMD AI Engine that has been successfully deployed across multiple markets as part of the AMD Versal Adaptive SoCs[6,7] and is also part of AMD Alveo V70 accelerator cards.[8] XDNA implements a scalable dataflow architecture with the main goal of efficient compute while minimizing memory bandwidth requirements both internally and externally. It is designed to exploit the characteristics of ML workloads, moving parts of the control that can be determined statically to compile time. The XDNA NPU design is highly scalable by implementing a tiling XDNA array architecture, enabling the creation of different sized 2-D arrays to fit the compute needs of different devices.

Figure 1 shows a die photo of the Ryzen 7040 series processor highlighting the XDNA NPU and a block

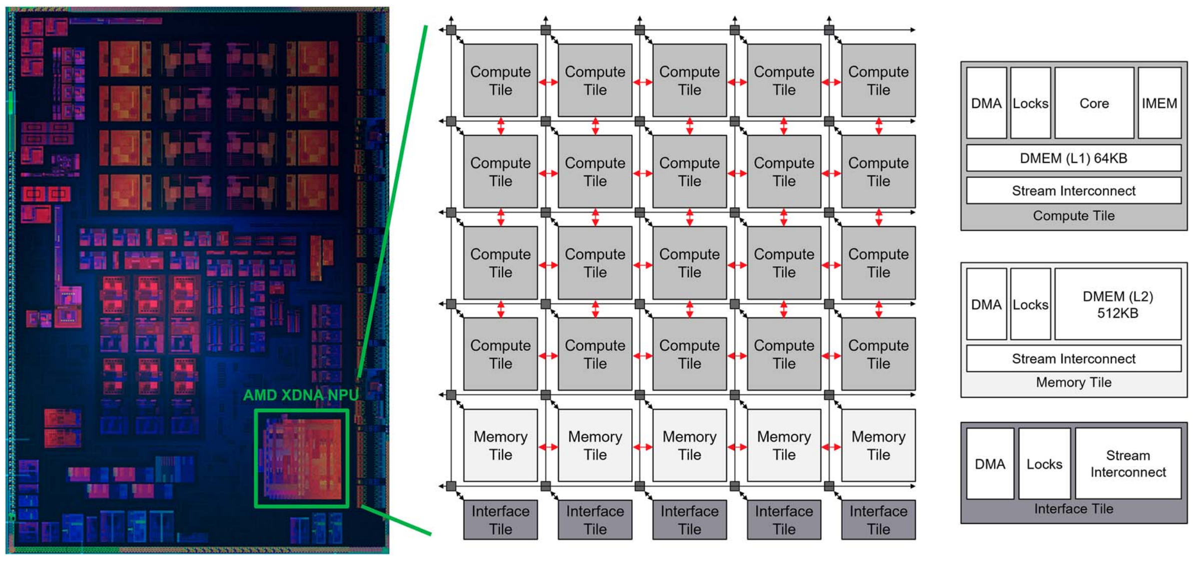

**FIGURE 1.** Ryzen 7040 die photo and NPU array top-level block diagram.

diagram of XDNA array architecture. It is composed of three variants of tiles, namely compute tiles, memory tiles, and interface tiles. Compute tiles include the XDNA cores with high-performance and efficient number-crunching capabilities and the first level of memory (L1). This first level of memory in each compute tile has a dedicated instruction memory (IMEM), and access to the four nearest compute tile data memories (DMEM). Memory tiles include the second level of memory (L2) while interface tiles provide communication with external blocks and access to main memory. To move data between these three levels of memory hierarchy there is a dedicated efficient data movement architecture with direct memory access (DMA) engines, configurable streaming interconnect, and hardware locks for synchronization.

## Generational Improvements

The XDNA evolution over the first-generation AI Engine includes design decisions targeting AI workloads. The major improvements are as follows:

- *Increased compute throughput per tile*: This targets higher efficiency on compute intensive neural networks.
- *Increased data memory size from 32 kB to 64 kB per tile*: This is more capacity to accommodate operands for higher compute throughput.
- *New data types int4 and bf16*: These are important data types in ML workloads.
- *New memory tiles*: These provide on-chip storage for weights and activations to reduce main memory bandwidth requirements.
- *Compression and sparsity support*: This reduces compute and memory bandwidth requirements in the presence of sparsity in neural networks.
- *Zero-value padding*: This reduces main memory bandwidth requirements when using halo regions as in convolutional neural networks (CNN).

## Compute Cores

The XDNA core is a highly optimized very-long instruction word (VLIW) processor featuring single-instruction multiple-data operations that supports both fixed-point and floating-point data types.

To reduce the code density problems of traditional VLIW architectures, the variable-length VLIW instruction formats range from 16 to 128 bits and support instructions where up to six operations can be issued in parallel. The code is fetched from the local IMEM in the tile.

One VLIW slot is dedicated to the matrix unit. This unit can sustain a throughput of one matrix operation per clock cycle. The datapath can be reconfigured on the fly for different operand precisions and formats, as shown in Table 1. *Operand formats* indicate the data format (i.e., number of bits per element) for the two matrix operands. These are all integer numbers except for the special case of *bfloat16* multiplication. The number of accumulator lanes represent the total number of dot-product operations performed in the matrix multiply operation. The accumulator is added to the matrix multiply as in traditional multiply-accumulate (MAC) operations. The data format of the accumulation is indicated as *accumulator precision*. For the integer cases, accumulation can be on 32 or 64 bits. For the *bfloat16* case, accumulation is in single precision floating point (*float*). The final column summarizes the achievable number of MAC operations per clock cycle.

The unit has special modes for fast Fourier transform processing, with operations directly using complex numbers with 16-b and 32-b precision data types (*cint16* and *cint32*).

The XDNA core contains support for 50% structured weight sparsity for ML inference applications. This effectively doubles the peak performance of the matrix unit by eliminating unnecessary multiplications by zero. Furthermore, the matrix unit can also compute element-wise MAC operations.

Parallel to the matrix unit, there is a vector add unit and custom units for vector shuffling and shifting. Besides addition, the vector add unit can perform subtractions, comparisons, and min/max operations. The vector shuffling unit is used to adapt memory formats to those needed by the matrix unit.

**TABLE 1.** Matrix multiply formats.

| Operand Formats | Accumulator Lanes | Accumulator Format | Number of MACs |
|---|---|---|---|
| int8 × int4 | 32 | int32 | 512 |
| int8 × int8 | 32 | int32 | 256 |
| int16 × int8 | 32 | int32 | 128 |
| int16 × int8 | 16 | int64 | 128 |
| int16 × int16 | 32 | int32 | 64 |
| int16 × int16 | 16 | int64 | 64 |
| int32 × int16 | 16 | int64 | 32 |
| cint16 × cint16 | 8 | int64 | 16 |
| cint32 × cint16 | 8 | int64 | 8 |
| bfloat16 × bfloat16 | 16 | float | 128 |

Multiple data conversion operations are available in both standalone formats and combined with load/store operations. Narrowing data conversions take accumulator results and produce data with reduced precision for the next stage. Widening data conversions are used in several algorithms.

Two VLIW slots are dedicated to load operations, and one slot to store operations that access the scratchpad DMEM in the tile. These three slots can concurrently transfer up to 256 bits per slot and contain dedicated address generation units (AGUs). The AGUs support specialized 2-D and 3-D addressing modes for efficient addressing of tensors. The available memory bandwidth enables efficient usage of the matrix unit. Moreover, one of the load slots supports on-the-fly decompression during data loading by inserting zero weights before performing convolutions. This feature ensures that only nonzero weights need to be stored in the local data memory.

Besides the DMA transfers between tiles, the XDNA core can transfer data directly from other cores in the array via 32-bit streams or from neighboring cores via 512-bit cascade streams.

Finally, a slot is dedicated to scalar operations and control flow instructions.

## Shared Memory

To reduce demand on main memory bandwidth and save energy, the XDNA array has a pool of shared on-chip memory (L2). This is implemented as a row of memory tiles.

Each tile contains 512 KB of high-density, high-bandwidth memory, giving a total of 2.5-MB shared memory in the Ryzen 7040 XDNA NPU. When combined with the compute tile DMEM, this gives a total of 3.75-MB on-chip scratchpad memory for application data.

This shared pool of scratchpad memory is used for short to long-term storage of application input, output, and intermediate data. For example, a portion of ML weight buffer may be prefetched from main memory once and then multicast to multiple compute tile DMEM with very high bandwidth. Intermediate activation tensors produced within the array can be stored in L2 and later transferred back to the compute tiles (L1) without consuming any main memory bandwidth.

## Data Movement Architecture

In order to achieve best application performance, the data movement architecture is responsible of transferring inputs and outputs between the core, L1, L2, and main memory with sufficient speed as to keep the core busy.

The key transfers are between main memory and L2, and between L2 and L1. Furthermore, the flexible interconnect supports more combinations such as main memory direct to L1 and direct L1 to L1 transfers.

In the XDNA architecture, a data transfer consists of a source DMA channel (MM2S), a destination DMA channel (S2MM), and a configured stream interconnect between the two, see Figure 2. The MM2S channel reads from memory and puts data on the stream. The S2MM channel gets data from the stream and writes to memory. Each of the compute tile, memory tile, and interface tiles have a stream interconnect switch, and a number of MM2S and S2MM DMA channels. The DMA channels in the interface tile read and write to main memory via the NPU network-on-chip and SoC-level fabric.

Data movement is scheduled in an eager fashion, leveraging hardware locks and back-pressure mechanisms to achieve synchronization. The interconnect ports needs to be configured before data movement is initiated, and then control of the source and destination DMA channels can occur asynchronously with the streaming interconnect protocol maintaining data integrity.

Each tile has a hardware lock unit that enables efficient synchronization on data buffers in NPU memory. The compute core and each DMA channel has a request interface to this unit allowing each to issue

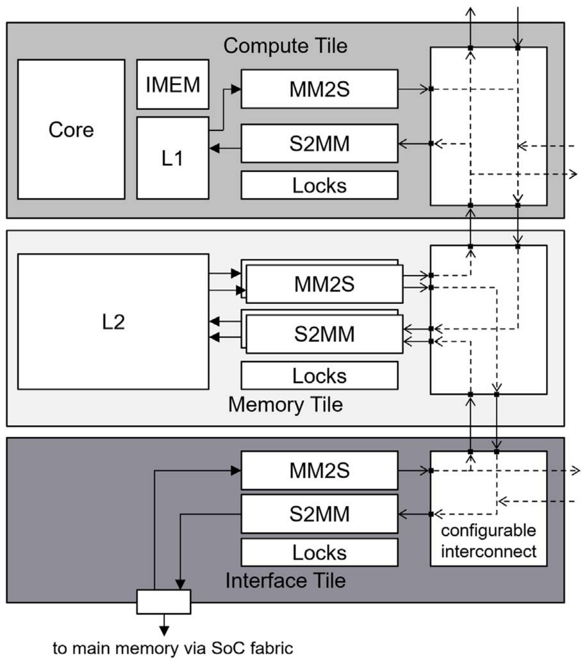

FIGURE 2. XDNA data movement architecture.

lock-acquire and lock-release requests. Tiles have 16 to 64 independent locks, and each lock can be in one of 64 states. A lock-acquire request will only be granted if the lock is in the necessary state. The software kernel running on the core or the DMA channel will not access the data until after the lock-acquire has been granted. The core or DMA channel will issue a lock release after it has completed accessing the data.

Since parallelism is key to performance, the XDNA DMA channels and compiler tools support double buffering of tensors. For example, double buffering of input and output buffers in L1 memory allows overlapping data transfers with computation, which is a key fundamental feature to hide data transfer latency in the XDNA architecture.

The broadcast and selective multicast features in the configurable interconnect enables reading and transmitting data once from the source and delivering it to multiple destinations, while only requiring a single tile-to-tile stream route.

## Dedicated DMAs

Each DMA channel is an independent engine with its own memory interface, stream port, and lock request interface.

To efficiently transform and distribute the buffers that are typically found in ML workloads, the DMA channels support a number of features. This includes up to 5-D tensor addressing, and insertion of padding into data pushed to the stream. As an example, these features enable a MM2S channel in the memory tile to extract a subtensor from a larger tensor in L2 memory, add zero-value padding words to the halo region and send to the compute cores for NxN convolution kernels.

The tensors in ML networks (weights and activations) may sometimes contain a large number of zeros. For example the output of the *ReLU*(*x*) operator is zero for negative values of *x*. The XDNA architecture takes advantage of this via a compression feature implemented in the DMA channels. The compression is lossless, where zero-valued bytes are optimized away. At the point of compression, a bit-mask is inserted ahead of a block of data, such that each bit in the mask represents a byte in the data block. If a data byte is zero, the bit is set to 0, and the data byte is not transmitted. At the point of decompression, the mask is used to reinsert the zero-valued bytes. This compression algorithm operates independently of application data type. This gives the XDNA architecture a theoretical maximum of 8$\times$ less data to be transmitted, and 4$\times$ acceleration of stream bandwidth. This compression feature also helps with accelerating main memory bandwidth, for both fetching of weight tensors and spilling and restoring of intermediate activations. Static weights may be compressed at compile time, and stored and transmitted in compressed format, and then decompressed within the core datapath.

## Isolation Capabilities

The XDNA architecture allows the execution of multiple ML workloads both concurrently (time sharing) and in parallel (spatial), which is supported by the hardware isolation capabilities. The NPU array can be divided into disjoint sets of columns named *partitions*. Each partition can run an independent application. XDNA supports up to four partitions, so up to four applications can run in parallel, but a larger number can use the array concurrently via time sharing.

The NPU architecture provides mechanisms to secure partition boundaries so that applications cannot interfere with each other. Within the array, isolation registers can be configured to block accesses from neighboring columns: core load/store requests, lock requests, and data transfers over the streaming interconnect. Additionally, all DMA transfers performed by the interface tile DMAs to and from main memory are tagged with the process address space identifier (PASID) to enforce security at address translation time.

## Control Subsystem

Task scheduling and data movement are orchestrated by a dedicated command processor in the NPU. This frees the design of XDNA cores to focus on execution of matrix and vector operations. This processor also acts as the entry point to use the NPU, receiving commands from the CPU drivers and applications and notifying completion. There are two main types of commands executed by this processor.

Application-level control commands:

- *Load application images*: These are compute kernels, and interconnect configuration required to implement the communication patterns defined by the application.
- *Dataflow orchestration*: This configures DMA transfers and keep track of the execution of the network operators.

System-level management commands:

- *Spatial configuration*: This handles partition management requests from the host drivers by configuring isolation registers and PASID.
- *Time sharing*: This orchestrates context switches to support many concurrent application contexts

per partition. This includes security tasks such as clearing the contents of the memories in the array to prevent data leaks.

- *Error management*: This handles error interrupts from the NPU array and forward them to host drivers.

## SOFTWARE ARCHITECTURE

### Programming Model

#### *XDNA*

The XDNA NPU provides a dataflow programming model. Kernels implementing network operators are mapped on compute cores and data movement is orchestrated by programming DMA engines available in all tiles. This is achieved using a graph-based programming interface that 1) defines the kernels to be used for each operator, 2) declares the tensors used by each operator in all levels of the memory hierarchy, 3) specifies the data dependencies between network layers, and 4) describes how tensors are spatially distributed across different kernels running on compute tiles. This includes broadcast and multicast patterns that minimize streaming bandwidth requirements. Advanced grouping abstractions are available 1) to map kernels in neighboring tiles to create reduction chains (leveraging the cascade interface) or sharing data with other kernels.

The software view of the NPU has three levels in its memory hierarchy. The lifetime of data in this memory hierarchy spans the execution of the whole neural network. This allows applications to define optimal data movement. This is in contrast to other programming models used by AI accelerators, such as CUDA, OpenCL, and SyCL, that force applications to write intermediate results back to the last levels of the memory hierarchy with much finer granularity (e.g., at kernel boundary), increasing memory bandwidth requirements. The three-level memory hierarchy is exploited in the programming models as follows:

- *Kernel scratchpad (L1)*: This distributed space is composed of the memories in compute tiles that holds the data required to execute tasks. In neural networks, tensors in a layer are divided into tiles that can be processed independently.
- *Shared scratchpad (L2)*: This distributed space is composed by the memory tiles that contains data required to execute the current layer, such as weights and intermediate feature maps. These data are tiled and transferred to the L1 space to be consumed by the kernels running on the compute tiles. In many networks, intermediate feature maps produced by the previous layer can be directly consumed by the next one without having to be transferred to main memory. If an intermediate feature map does not fit in L2, it is divided into tiles (for L2), which are successively fetched from main memory. Furthermore, if space is sufficient, weights for the next layers can be prefetched to this space to avoid cross-layer execution bubbles.
- *External main memory*: This contains input/output data, weights, and spill buffers for intermediate feature maps (in case they do not fit in the internal memory levels).

Using the information in the graph, the toolchain compiles the kernels, generates the configuration for the streaming interconnect, and generates the DMA configurations, locking, and synchronization schemes for all the network layers. This includes chaining of data transfers across the different memory hierarchy levels and automatic double-buffering to hide the latency of the data transfers. Computing this information at compile time allows to dedicate the compute resources to the execution of the kernels, maximizing the utilization of the matrix unit in the cores.

#### *System*

The XDNA NPU is exposed as a PCIe accelerator device for discovery and enumeration purposes and directly connects to the cache-coherent interconnect for main memory accesses.

The NPU host runtime provides facilities to offload compute-intensive parts and allocate memory that is both accessible by the CPU and the DMAs in XDNA interface tiles. It also creates work queues for applications to submit work to the NPU and wait for completion. The granularity of each computation is defined at compile time, and it can span from a single kernel function to more complex compute sequences such as a full neural network inference. Commands contain pointers to the input and output buffers used during computation.

The NPU allows multiple applications to run concurrently and securely on the XDNA array. The kernel mode driver implements spatial and temporal sharing and QoS among applications.

### Software Flow and Framework

The Ryzen AI software flow[9] builds on top of ONNX Runtime inference flows[10] and provides users access to the NPU through the Vitis AI Execution Provider (Vitis AI EP). VitisAI EP uses the ONNX runtime (ONNX-RT) extensible execution providers (EP) framework, which provides a standard interface to execute the ONNX models optimally on a given hardware platform. This interface enables flexibility and optimizes the execution by taking advantage of the computing capabilities of the target platform. For a model to be executed on Ryzen AI, the model needs to be exported from PyTorch and TensorFlow frameworks into the ONNX format.

The ONNX-RT flow has three well defined stages.

1) *Model preparation*: Model export is the responsibility of the end-user. Once model is exported, model quantization can be done using Vitis AI quantizer[11] or third-party tools for the supported data types. The Vitis AI quantizer supports post-training quantization and quantization-aware training flows.
2) *Model compilation*: Vitis AI EP provides a unified execution flow for various strategies for model execution on XDNA which includes graph mode, eager mode, and mixed-mode execution.
3) *Model execution*: ONNX-RT is the default runtime that divides the graph into subgraphs to send to all the relevant EPs, and EPs then execute the assigned tasks. ONNX-RT has well-defined application programming interface (APIs) to interact with an EP, including APIs for querying an EP for its capabilities, compiling a model, and running a model.

## Spatial and Time-Shared Modes

One of the novel software architectural features in Ryzen AI software is its ability to share the compute resources spatially, temporally, or a hybrid of the two. The XDNA NPU has spatially distributed compute, memory, and interface tiles that can be grouped column-wise to create partitions. Each partition can be of variable size from one to five columns in Ryzen 7040 processors.

For the workloads requiring real-time quality of service, such as real-time audio and video processing in video conference calls, part of the compute resources can be dedicated to a specific workload. As an example, one partition comprising two columns (40% of compute) can be dedicated for computing a real-time workload while the rest of the compute resources can be time-shared among other best effort use cases such as offline content creation and document summarization. The scheduling of multiple applications on the same partition is implemented by the command processor with fine granularity.

Figure 3 depicts examples of time sharing, spatial sharing, and hybrid mode. In pure time sharing mode [Figure 3(a)] the entire NPU is time shared by various applications. Purely spatial allocation [Figure 3(b)] can have up to four concurrent applications running with their own dedicated resources, and thus, these applications can run without any resource contention among them.

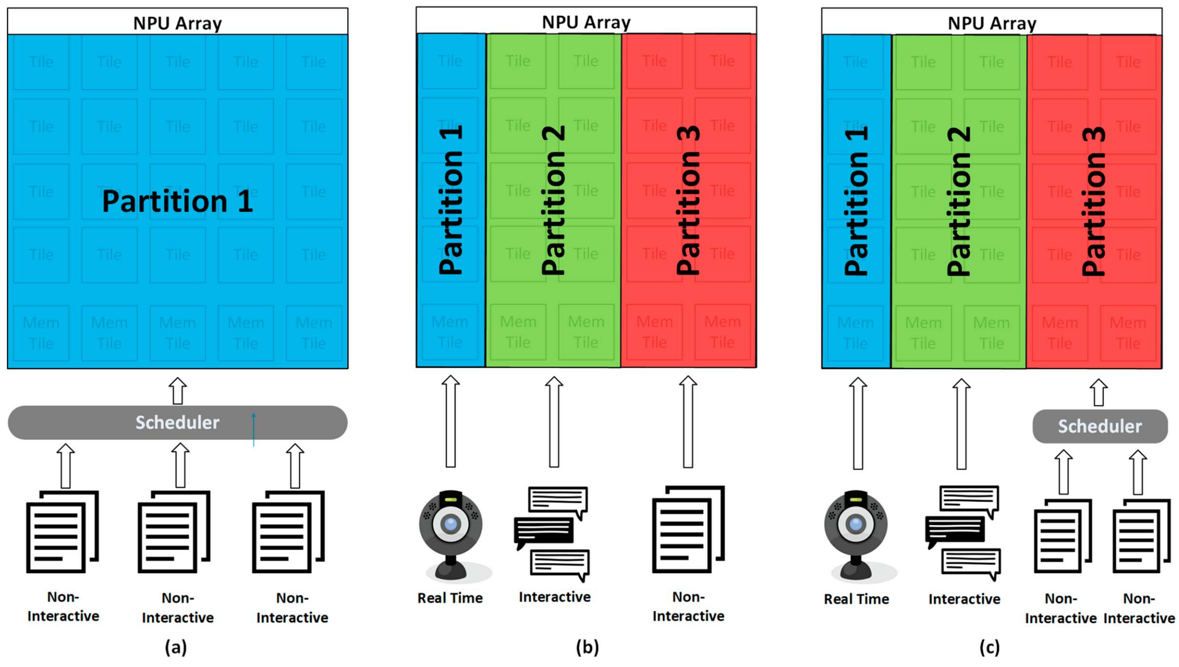

**FIGURE 3.** XDNA array sharing modes across multiple applications. (a) Time sharing. (b) Spatial sharing. (c) Hybrid mode.

Based on the requirements of the applications running on the system, a hybrid of spatial sharing and time-sharing can be chosen [Figure 3(c)]. For example, real-time video conferencing may use a single-column partition (Partition 1) exclusively by means of spatial allocation, an interactive application may exclusively use a two-column partition (Partition 2) due to its higher compute requirements, and the rest of the compute and memory resources composing a third partition (Partition 3) can be utilized for all other workloads. This unique capability allows the XDNA NPU to guarantee quality of service and deterministic latency and throughput in a loaded system for real-time applications.

## XDNA ADVANTAGE

The XDNA NPU in Ryzen AI processors provides a performance advantage and, also, power and energy efficiency benefits for being optimized for ML workloads. We evaluate the performance and efficiency improvement of running on XDNA with a set of nine ML benchmarks including CNNs, Gaussian adversarial networks, and super-resolution networks (SR) compared to running on the CPU on an ASUS ROG Zephyrus G14 equipped with a top-of-the-line Ryzen 7940HS processor with 16 GB of DDR5-4800 memory. The software environment is the Ryzen AI software version 1.1 that can be downloaded and installed following the instructions in the documentation website.[12] The CPU version uses optimized advanced vector extensions code using all eight CPU cores.

Figure 4 shows the performance and performance per watt advantage of running on the NPU compared to the CPU. The XDNA NPU consistently provides speedup across these benchmarks with performance improvements ranging from 1.3× to 9.7×. Given that the NPU runs these models more efficiently, it also does it with significantly lower power which results in between 4.3× and 33× better performance per watt.

### Video and Audio Effects Enablement

Millions of laptops equipped with Ryzen 7040 series processors seamlessly use the XDNA NPU to accelerate the execution of Microsoft Windows Studio Effects (WSE).[2] These video and audio enhancing features, including background blur, eye contact, auto framing and voice focus, are compute intensive. Battery life is paramount in laptop devices, and therefore these features need to be enabled without significantly affecting battery life.

The power efficiency advantage of the XDNA NPU stands out when running the models used for WSE. Windows provides two power modes: *balanced* and *best efficiency*. Running WSE on the NPU in balanced mode results in 5.3× better performance per watt compared to the CPU. Using best-efficiency mode improves the efficiency of running WSE on the CPU by 50%, but the reduced performance does not meet the required frames per second (FPS). In best efficiency mode, the NPU achieves the required FPS and further increases normalized efficiency to 5.4× compared to the CPU in balanced mode.

Running a Teams call benefits from the Ryzen 7040 series processors incorporating XDNA technology. The energy efficiency of the XDNA NPU enables WSE on Teams with little extra power compared to running Teams without effects. This makes the availability of these features hardware dependent: Windows allows

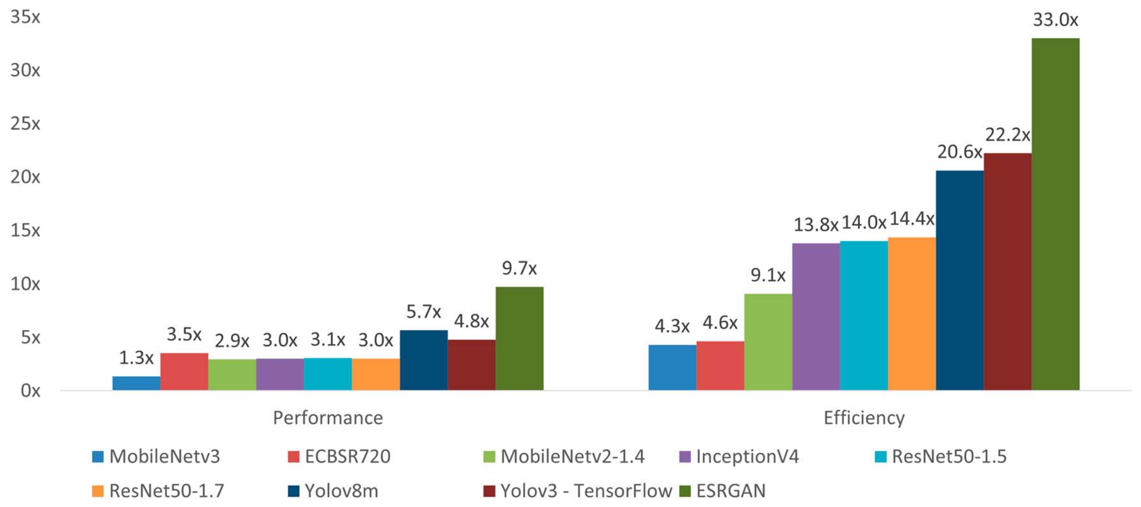

**FIGURE 4.** Neural network performance and performance per watt (efficiency) of XDNA (normalized to CPU).

them to be enabled only if the system is equipped with AI hardware acceleration such as the XDNA NPU.

## CONCLUSION

The AMD Ryzen 7040 series with XDNA NPU technology is the first instance of Ryzen AI processors. The AI-optimized hardware and software architecture of XDNA achieves higher performance and efficiency compared to more general-purpose architectures such as CPUs and GPUs for the execution of AI workloads. Integrating XDNA enables a full set of new AI experiences, making it a fundamental component of today's Ryzen-AI-powered mobile PCs. The road ahead is a new era of AI PCs with increasing AI use cases in image processing, video enhancements, and generative AI. These new experiences raise the demands for compute and memory bandwidth of AI tasks, requiring further innovation and specialization to extend performance and efficiency within the strict power budget of mobiles PCs.

## REFERENCES

1. L. Su. "AMD advancing AI." AMD. Accessed: Jul. 31, 2024. [Online]. Available: https://www.amd.com/en/corporate/events/advancing-ai.html
2. "Windows studio effects overview." Learn Microsoft. Accessed: Jul. 31, 2024. [Online]. Available: https://learn.microsoft.com/en-us/windows/ai/studio-effects
3. "Copilot in windows and other AI-powered features." Microsoft. Accessed: Jul. 31, 2024. [Online]. Available: https://www.microsoft.com/en-us/windows/copilot-ai-features
4. "AMD Ryzen AI cloud to client demo." AMD. Accessed: Jul. 31, 2024. [Online]. Available: https://community.amd.com/t5/ai-discussions/amd-ryzen-ai-cloud-to-client-demo/td-p/622159
5. M. Subramon, D. Kramer, and I. Paul, "AMD Ryzen$^{TM}$ 7040 series: Technology overview," in *Proc. IEEE Hot Chips 35 Symp.*, 2023, pp. 1–27.
6. S. Ahmad et al., "Xilinx first 7nm device: Versal AI Core (VC1902)," in *Proc. IEEE Hot Chips 31 Symp.*, 2019, pp. 1–28.
7. J. Noguera, "Xilinx edge processors," in *Proc. IEEE Hot Chips 33 Symp. (HCS)*, Palo Alto, CA, USA, 2021, pp. 1–21, doi: 10.1109/HCS52781.2021.9567521.
8. L. Su. "AMD highlights future of high-performance and adaptive computing during opening keynote of CES 2023." AMD. Accessed: Jul. 31, 2024. [Online]. Available: https://www.amd.com/en/newsroom/press-releases/2023-1-4-amd-highlights-future-of-high-performance-and-adap.html
9. AMD. "RyzenAI Software gitHub page." GitHub. Accessed: Jul. 31, 2024. [Online]. Available: https://github.com/amd/RyzenAI-SW
10. Microsoft. "ONNX Runtime execution providers." ONNX Runtime. Accessed: Jul. 31, 2024. [Online]. Available: https://onnxruntime.ai/docs/execution-providers
11. "Vitis AI quantizer for ONNX." AMD. Accessed: Jul. 31, 2024. [Online]. Available: https://ryzenai.docs.amd.com/en/latest/vai_quant/vai_q_onnx.html
12. "Ryzen AI Software 1.1 installation instructions." AMD. Accessed: Jul. 31, 2024. [Online]. Available: https://ryzenai.docs.amd.com/en/latest/inst.html

**ALEJANDRO RICO** is a principal member of the technical staff at AMD, Austin, TX, 78735, USA. His research interests include domain-specific accelerators, parallel computer architectures, and artificial intelligence. Rico received his Ph.D. degree in computer architecture from UPC-Barcelona. Contact him at alex.rico@amd.com.

**SATYAPRAKASH PAREEK** is a principal member of the technical staff at AMD, San Jose, CA, 95124, USA. His research interests include domain-specific accelerators, hardware–software co-design, and edge artificial intelligence. Pareek received his master's degree in communication and signal processing from IIT Bombay. Contact him at satyaprakash.pareek@amd.com.

**JAVIER CABEZAS** is a principal member of the technical staff at AMD, Austin, TX, 78735, USA. His research interests include software–hardware co-design, performance portability, and high-performance computing. Cabezas received his Ph.D. degree in computer architecture from UPC-Barcelona. Contact him at javier.cabezas@amd.com.

**DAVID CLARKE** is a principal member of the technical staff at AMD, D24 T683, Dublin, Ireland. His research interests include parallel data movement and synchronization, parallel computer architectures, and artificial intelligence. Clarke received his Ph.D. degree in computer science from the University College Dublin. Contact him at david.clarke@amd.com.

**BARIS OZGUL** is a director at AMD, D24 T683, Dublin, Ireland. His research interests include heterogeneous parallel computer architectures, domain-specific accelerators, and machine learning. Ozgul received his Ph.D. degree in electrical and electronics engineering from Bogazici University. Contact him at baris.ozgul@amd.com.

**FRANCISCO BARAT** is a principal member of the technical staff at AMD, 3000, Leuven, Belgium. His research interests include low-power very-long instruction word architectures, compilation techniques, and artificial intelligence. Barat received his Ph.D. degree in computer architecture from KU Leuven. Contact him at francisco.barat@amd.com.

**YAO FU** is a principal member of the technical staff at AMD, San Jose, CA, 95124, USA. His research interests include system-level performance optimization, hardware–software co-design, and artificial intelligence. Fu received his Ph.D. degree in electrical and computer engineering from Iowa State University. Contact him at yao.fu@amd.com.

**STEPHAN MÜNZ** is a member of the technical staff at AMD, D24 T683, Dublin, Ireland. His research interests include acceleration artificial intelligence, performance optimized programming, and power-efficient architectures. Münz received his M.Sc. degree in electronics and information technology from KIT-Karlsruhe. Contact him at stephan.munz@amd.com.

**DYLAN STUART** is a member of the technical staff at AMD, D24 T683, Dublin, Ireland. His research interests include hardware–software co-design, artificial intelligence, and computer architecture. Stuart received his M.A.Sc. degree in computer architecture from the University of Toronto. Contact him at dylan.stuart@amd.com.

**PATRICK SCHLANGEN** is a member of the technical staff at AMD, 50933, Cologne, Germany. His research interests include accelerator programming, high-performance computing, and system-level optimization. Schlangen received his M.Sc. degree in electrical engineering from KIT-Karlsruhe. Contact him at patrick.schlangen@amd.com.

**PEDRO DUARTE** is a member of the technical staff at AMD, D24 T683, Dublin, Ireland. His research interests include vector-matrix processors, very-long instruction word compilers, and high-throughput/low-power microarchitecture. Duarte received his M.Sc. degree in electrical and computer engineering from the University of Coimbra. Contact him at pedro.duarte@amd.com.

**SNEHA DATE** is a senior member of technical staff at AMD, San Jose, CA, 95124, USA. Her research interests include domain-specific accelerators, instruction set architectures, and artificial intelligence inference on edge. Sneha received her M.S. degree in electrical engineering from the University of Houston. Contact her at sneha.date@amd.com.

**INDRANI PAUL** is a senior fellow at AMD, Austin, TX, 78735, USA. Her research interests include power-management architecture, performance/performance-per-watt solutions, and hardware–firmware–software coordinated power-efficiency optimizations. Paul received her Ph.D. degree in computer engineering from the Georgia Institute of Technology. Contact her at indrani.paul@amd.com.

**JIAN WENG** is a senior manager at AMD, Beijing, 10029, China. His research interests include artificial intelligence (AI) accelerator design, AI compiler design, and loop optimization. Weng received his master's degree in electronics and integrated circuits from Tsinghua University. Contact him at jian.weng@amd.com.

**SONAL SANTAN** is a senior fellow at AMD, San Jose, CA, 95124, USA. His research interests include low-overhead runtime stack design, acceleration offload engines, and efficient hardware–software interfaces. Santan received his B.Tech. degree in electrical engineering from IIT-Kanpur. Contact him at sonal.santan@amd.com.

**VINOD KATHAIL** is a senior fellow at AMD, San Jose, CA, 95124, USA. His research interests include machine learning, heterogenous programming environments, and high-level synthesis. Kathail received his Sc.D. degree in electrical engineering and computer science from the Massachusetts Institute of Technology. Contact him at vinod.kathail@amd.com.

**ASHISH SIRASAO** is a corporate vice president at AMD, San Jose, CA, 95124, USA. His research interests include hardware–software co-design, programming paradigms for domain-specific accelerators, and deep-learning algorithms. Sirasao received his M.Tech degree in electrical engineering from IIT Mumbai. Contact him at ashish.sirasao@amd.com

**JUANJO NOGUERA** is a senior director at AMD, San Jose, CA, 95124, USA. His research interests include domain-specific accelerators, hardware–software co-design, and artificial intelligence. Noguera received his Ph.D. degree in computer architecture from UPC-Barcelona. Contact him at juanjo.noguera@amd.com.