The AI Compute Extensions (ACE) for x86

x86 Ecosystem Advisory Group

Whitepaper

Date:	04/15/2026
Version:	1.0
Prepared by:	Stuart Biles, Brian Thompto, Michael Estlick, Eric Schwarz, Thomas Fox, Gabriel Loh, Marius Evers, Michael Clark (AMD); Alexander Heinecke, Pradeep Dubey, Ido Ouziel (Intel)

The AI Compute Extensions (ACE) for x86

Stuart Biles[1], Brian Thompto[1], Michael Estlick[1], Eric Schwarz[1]
Thomas Fox[1], Gabriel Loh[1], Marius Evers[1], Michael Clark[1]
Alexander Heinecke[2], Pradeep Dubey[2], Ido Ouziel[2]

[1]AMD [2]Intel

Abstract — In this paper we introduce the AI Compute Extensions for the x86 ISA. ACE offers a significant increase in matrix multiply performance, scalability and energy efficiency. ACE integrates seamlessly with AVX10, providing a low-friction and ubiquitous matrix acceleration capability for the x86 ecosystem.

Background

Matrix multiplication forms the backbone of neural networks and large language models, serving as the main computational mechanism for operations like forward and backward propagation, weight updates, and layer computations.

While SIMD (Single Instruction, Multiple Data) extensions such as AVX10 can be used for matrix multiplication, the scalability and compute density available can be limited. The introduction of accelerated matrix multiplication can increase performance; however efficient interaction between matrix multiplication and other vectorized kernels is essential to support today's machine learning models.

To address these challenges, we have developed ACE (AI Compute Extensions) to improve matrix multiplication performance, scalability and energy efficiency for x86. ACE integrates seamlessly with AVX10, offering a versatile and powerful compute capability in both matrix and vector domains.

ACE accelerates matrix multiplication whilst providing great flexibility to x86 implementation teams; it allows for performance tuning and efficient reuse of existing AVX10 investments, creating a scalable matrix acceleration framework suitable for a wide range of implementations.

ACE offers ubiquitous matrix acceleration from laptop to data center servers. This cross-platform capability democratizes AI and reduces developer friction when compared to offloading compute tasks to specialized hardware. AMD, in partnership with Intel and the x86 EAG (Ecosystem Advisory Group) [EAG24], is readying ACE as the standard matrix acceleration architecture for x86, further enhancing the already vibrant x86 ecosystem.

ACE Overview

ACE introduces matrix acceleration based on outer product operations designed to work integrally with AVX10. An ACE outer product operation provides software with a 16× compute density win over an equivalent AVX10 multiply-accumulate operation whilst consuming the same number of input vectors.

ACE supports native matrix multiplication of popular AI data formats, as summarized in Table 1. In a first for commercial processor architectures, ACE supports data types from the OCP MX standard, including inline block scaling [OCP23].

Category	Formats
Integer	INT8
Floating Point	OCP FP8, OCP MXFP8, OCP MXINT8, BF16

Table 1: ACE v1 native datatypes

An ACE outer product operation for 8-bit input element data is depicted in Figure 1. Two AVX10 512-bit ZMM vector registers (blue and yellow) are consumed as input. Each input vector holds a 16×4 input matrix of 8-bit data which are presented to the outer product operation. At each row and column intersection, the product of the input terms is accumulated with a 32-bit accumulator sub-tile residing in the destination tile register (green).

The outer product algorithm was first proposed in 1995 [SUM95]. Outer product acceleration featured in several academic processor case studies including OuterSPACE [OUT18]. Outer product matrix acceleration was subsequently adopted in commercial processors with IBM Power10 [HOT20] and followed later by others.

The two-dimensional nature of the tile register provides N^2 compute density scaling when compared to a SIMD dot product operation, as shown in Table 2.

Operation	Multiplies per operation	AVX input vectors consumed
AVX INT8	64	2
ACE INT8	1024	2
AVX BF16	32	2
ACE BF16	512	2

Table 2: Relative compute density of AVX10 and ACE matrix multiply primitives

The semantics of the ACE outer products instructions are constant for the programmer; the same result is always generated. Implementations can tailor outer product performance to suit the area budget of the implementation.

ACE introduces processor state for 8 tile registers and the state needed to support OCP block scaling, as shown in Table 3.

State Component	Dimensions	Number
Tile Register	512b × 16 rows	8
Block Scale Register	1024b	1

Table 3: Processor state introduced with ACE

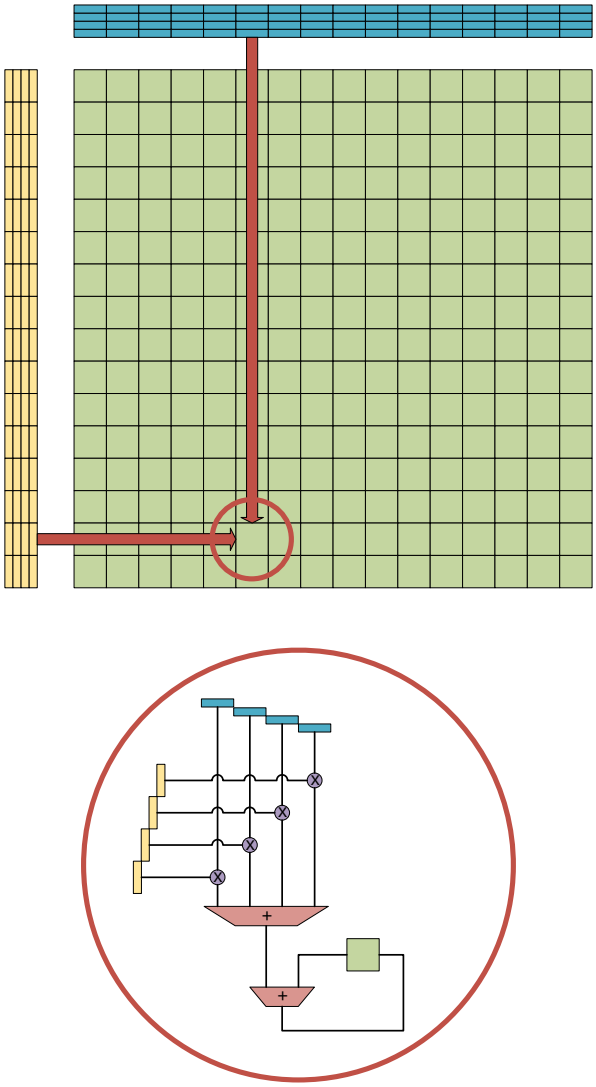

Figure 1: ACE outer product operation for 8-bit data; the insert depicts the operation that occurs at each intersection of row and column input data

ACE is revealed to software as a new palette under the AMX accelerator framework, allowing reuse of much of the system programmer model and operating system support for AMX.

ACE and AVX10: an Organic Partnership

ACE's outer product operation architecture is a great fit with the advanced vector features of modern x86 implementations. Modern out-of-order processors allocate significant silicon budget to vector register files; it is not uncommon for modern designs to contain hundreds of physical vector registers. This investment provides the AVX10 unit with the resources needed to identify ILP through independent operations and decoupling from the latency of the cache hierarchy. ACE reuses this solid foundation to provide input matrix data in a cost-efficient manner.

The use of AVX10 vector registers for input matrix data provides an excellent opportunity for software to pre-process and format data using the extensive AVX10 capabilities just in time for consumption by outer product operations. Inline format conversion is well supported by ACE and AVX10, as discussed in the section Data Format Agility.

In addition to consuming vectors as input, ACE provides capabilities to transfer data efficiently between ACE tile and AVX10 vector register files. This agile communication lends itself to pre- and post-matmul kernels that are to be executed using the extensive computation capabilities of AVX10.

The fit for ACE with AVX10 is further solidified when considering the scheduling footprint presented by the out-of-order operations. A typical AVX10 multiply-accumulate operation consumes two input vectors and accumulates on to a third destination. ACE outer products present a nearly identical template to the processor's scheduling logic, accumulating into a tile register instead of an AVX vector register, see Listing 1. With an appropriate level of hardware investment in the outer product datapath, the latency of ACE operations is not dissimilar to their AVX10 cousins. Overall, this presents an organic and natural fit with the scheduling logic orchestrating instruction execution where ACE can be viewed as an additional functional unit in the AVX10 engine.

AVX10 VNNI INT8

```
__m512i _mm512_dpbssd_epi32(__m512i src, __m512i a, __m512i b);
```

ACE INT8 Outer Product

```
void __tile_top4bssd(__tile1024i *dst, __m512i a, __m512i b);
```

Listing 1: Comparison between AVX10 VNNI INT8 multiply-accumulate and ACE INT8 outer product intrinsics

Simple LPGEMM Kernel

A simple outline of a single tile register LPGEMM kernel is shown in Listing 2. The kernel steps the single tile register as a window over the output matrix `C`. At each new window location, the tile register is zeroed, and outer products run through all K for the input matrices `A` and `B`. At the end of each run through K, the tile register contents are transferred to `C` in memory via `_tile_movrow`.

```
for (int n = 0; n < N; n += ACE_TILE_N) {
 for (int m = 0; m < M; m += ACE_TILE_M) {
  __tile1024i acc;
  __tile_zero(&acc);
  for (int k = 0; k < K; k += 4) {
   __m512i a = _mm512_load_si512(&A[k/4][m]);
   __m512i b = _mm512_load_si512(&B[k/4][n]);
   __tile_top4bssd(&acc, a, b);
  }
  for (int mc = 0; mc < ACE_TILE_M; mc++) {
   __m512i row = __tile_movrow(acc, mc);
   _mm512_store_si512(&C[m+mc][n], row);
  }
 }
}
```

Listing 2: Simple single tile ACE LPGEMM kernel

Blocked Register Kernels

ACE's eight tile registers facilitate blocked register kernels in software, where a larger window in the output matrix is established using multiple tile registers. Blocked register kernels bring significant benefits in terms of vector input reuse and reduced demand on the cache hierarchy.

The arrangement of tile registers and shared vector inputs for a 4×2 blocked kernel is illustrated in Figure 2. This arrangement facilitates an output window dimension of 64×32 over the output matrix.

A comparison of relative input data bandwidth between a single and 4×2 tile kernel is shown in Table 4. As can be seen from the table, the blocked kernel significantly reduces the effective rate of 512-bit vector loads needed for each outer product operation.

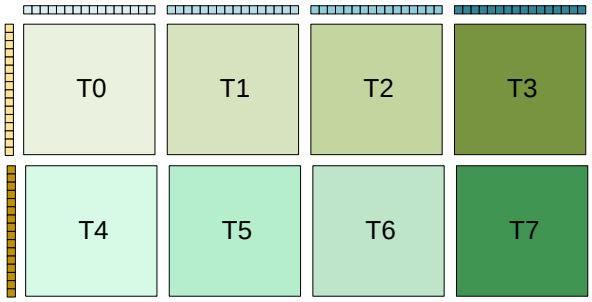

Figure 2: Arrangement of tile registers and vector inputs for 4×2 blocked register kernel

With the AVX10 architectural guarantee of 512-bit vectors and the provision of eight tile registers available for accumulation, ACE offers a competitive architectural resource compared with other solutions.

Config	OPs / iter.	AVX10 vecs / iter.	Loads / OP
1×1	1	2	2
4×2	8	6	0.75

Table 4: Comparison of single and 4×2 blocked register kernel vector bandwidth demands

Block Scale Support

ACE supports inline OCP MX block scaling for relevant outer product operations; ACE introduces register state for 8 groups of 16 8-bit block scale values, sufficient to support OCP MX format data in blocked register kernels using all 8 ACE tile registers.

The scale group for each input to the outer product operation is encoded in the instruction and separably addressable for algorithmic flexibility. An illustration of an outer product combining 8-bit element values and OCP MX weights is shown in Figure 3.

Data Format Agility

Innovation in machine learning algorithms has driven interest in low precision data formats to enable faster processing and reduce demands on memory bandwidth. ACE includes dedicated format conversion operations for the popular OCP MX data types (FP4, FP6 and FP8). These hardware operations allow optimized conversion to native compute types supported by the ACE outer product operations and quantization from higher precision formats.

There is continued research into future data formats, including storage formats that minimize the memory footprint of ever increasing LLM models. Having flexibility to adapt to future data formats in software is very attractive.

A key observation is that reduced precision data formats lend themselves to brute force LUT (lookup table) conversion rather than requiring a more programmatic treatment that is typically used when using software algorithms to convert between higher precision formats such as FP32 and FP64.

AVX10 boasts a powerful family of LUT operations in the form of VPERM. AVX10's architectural guarantee of 512-bit vector support further amplifies their utility. For the byte variants of VPERM, VPERMB and VPERMI2B, the 512-bit vector forms support a 64-entry and 128-entry LUT, respectively. This enables arbitrary data conversion of formats up to 7-bits in size. These operations process all 64 lanes simultaneously,

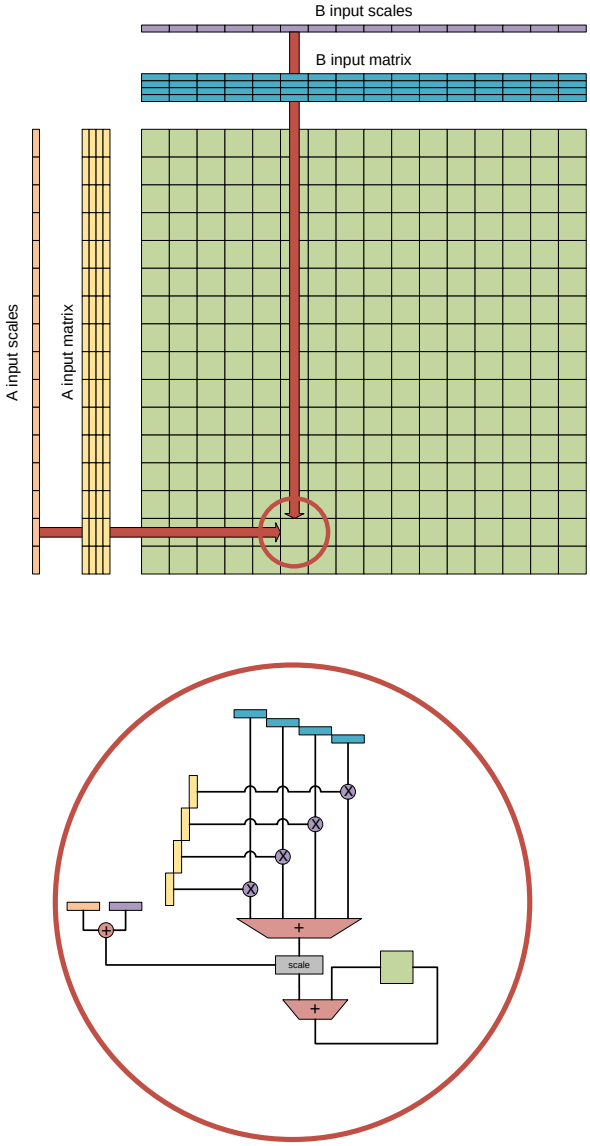

Figure 3: Outer product with inline block scale; the insert depicts the operation that occurs at each intersection of row and column input and scale data

allowing for highly efficient software defined format conversion.

ACE further improves software defined format conversion, with the introduction of the VUNPACKB instruction. VUNPACKB allows efficient marshalling of packed data into 8-bit lanes; arguments to the instruction allow selection of input data widths from 2 to 7 bits, with options to specify an offset into the input vector and optional sign extension for use with integer data types.

It is anticipated that VUNPACKB has applications outside of the AI domain, as narrow element storage types are found in other applications where storage efficiency matters.

The VPERM instruction class provides significant future proofing to x86 and machine learning workloads using AVX10 and ACE. The flexibility of VPERM allows emerging storage encoding techniques such as codebooks [QUI24] to be supported. For example, by appending a 3-bit dictionary select with a 4-bit element format, up to 8 codebooks may be

selected from the same pair of LUT registers configured for use with VPERMI2B.

Software Enablement

ACE is a natural extension to the AVX10 instruction set. Initial software enablement is underway, including integration with compilers, debuggers, and profilers. Upcoming efforts will focus on delivering optimized kernels, library and ML runtime integrations:

- Deep Learning and HPC libraries (e.g., lower-precision GEMMs, LLM primitives)
- Popular Python-based libraries such as NumPy and SciPy
- Machine learning frameworks including PyTorch and TensorFlow

This work will coordinate with the x86 EAG to ensure broad and consistent adoption across the x86 platform, benefiting developers and users throughout the ecosystem.

Standardization with the x86 EAG

The x86 Ecosystem Advisory Group (EAG) was launched in October 2024 to strengthen the future of x86 computing. This advisory group brings together AMD, Intel, and key ecosystem partners, all committed to advancing the x86 platform through collaborative decision-making, standardized features, and developer-friendly innovations.

With input from the EAG, AMD and Intel have worked together to align and refine the ACE ISA, delivering standardized matrix acceleration features across the x86 ecosystem. The alignment has produced several positive outcomes: the resulting architecture proposal incorporates ideas and contributions from both vendors, as well as insights from the broad market reach of the EAG community.

AMD and Intel continue to cooperate on the future roadmap for ACE and AVX10, aiming to embrace new opportunities in AI and other workload domains. The widespread adoption and high performance of x86 make it an ideal choice for developers; the addition of ACE to the ISA further strengthens the future of the x86 ecosystem.

References

SUM95 R. A. van de Geijn, J. Watts, *SUMMA: Scalable Universal Matrix Multiplication Algorithm*, https://www.cs.utexas.edu/ftp/techreports/tr95-13.pdf

OUT18 Subhankar Pal et al., *OuterSPACE: An Outer Product based Sparse Matrix Multiplication Accelerator*, https://blaauw.engin.umich.edu/wp-content/uploads/sites/342/2019/12/OuterSPACE-An-Outer-Product-based-Sparse-Matrix-Multiplication-Accelerator.pdf

HOT20 W. Starke and B. Thompto, *IBM's POWER10 Processor*, https://hc32.hotchips.org/assets/program/conference/day1/HotChips2020_Server_Processors_IBM_Starke_POWER10_v33.pdf

OCP23 B. Rouhani et al., *OCP Microscaling Formats (MX) Specification Version 1.0*, https://www.opencompute.org/documents/ocp-microscaling-formats-mx-v1-0-spec-final-pdf

QUI24 A. Tseng, J. Chee, Q. Sun, V. Kuleshov, and C. De Sa., *QuIP#: Even better LLM quantization with Hadamard incoherence and lattice codebooks*, https://openreview.net/forum?id=9BrydUVcoe

EAG24 *Intel and AMD Form x86 Ecosystem Advisory Group to Accelerate x86 Innovation*, https://www.amd.com/en/newsroom/press-releases/2024-10-15-intel-and-amd-form-x86-ecosystem-advisory-group-to-accelerate.html