Accelerating oneDNN on AArch64

Jonathan Deakin, Milos Puzovic and team

5 June 2024

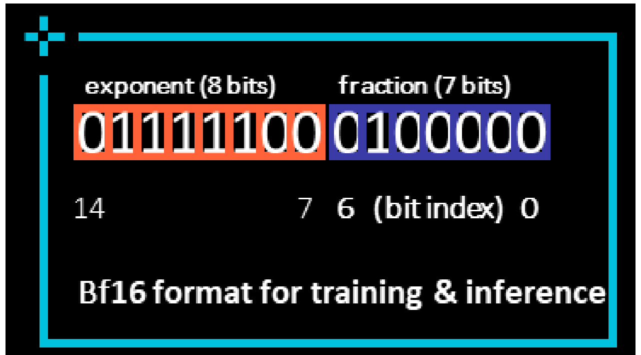
Introduction

Acceleration on CPU: basics

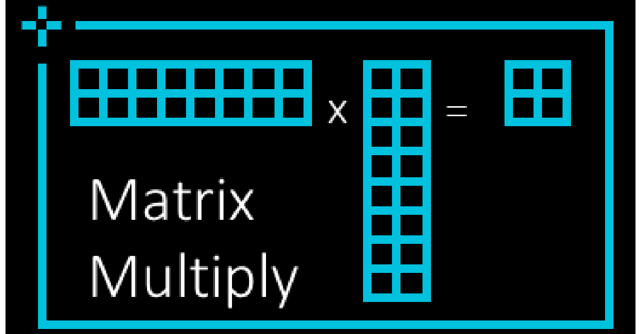
- Scalar -> NEON/SVE SIMD
  - NEON is 128bit, ~4x throughput for f32 (per instruction), ~16x for i8
  - SVE has variable vector length, enabling vector length agnostic programming
- Multi-threading
  - Theoretically ~number of core x speedup
  - Practically always a bit less, sometimes a lot less
- Memory access
  - Data reuse
  - Contiguous access, or packing for specific instructions
- Quantization: convert f32 -> i8 => ~4x speedup
- Fast math: convert f32 -> bf16 => ~2x speedup
- Function approximation, particularly for exp-based activations

BFloat16 Extension

- Armv8.6-A extension, mandatory in Armv9-A
- Four new instructions
- Available in SVE, Neon (AArch64 and AArch32)
- Accepts BF16 inputs, does not generate BF16 results
- Accumulating an FP32 intermediate result for better accuracy



Armv8.6-A Bfloat16



Armv8.6-A BF16 Matmul

ML Specific features in Arm Neoverse cores

Features enhancing ML performance on Arm Neoverse CPUs

Arch	• Matrix multiplication (INT8)
Arch	• Matrix-multiply-and-accumulate (*MMLA*) (v8.6)
Micro Arch	• SVE vector length
Arch	• BFloat16 support (v8.6)

	Neoverse-N1	Neoverse-V1	Neoverse-N2	Neoverse-V2
Vector units	**NEON 128-bit x 2**	**NEON 128-bit x 4** **SVE 256-bit x 2**	**NEON 128-bit x 2** **SVE 128-bit x 2**	**NEON 128-bit x 4** **SVE 128-bit x 4**
FP32	✓	✓	✓	✓
FP16	✓	✓	✓	✓
BF16	✗	✓	✓	✓
Dot Product	✓	✓	✓	✓
Integer Matrix Multiplier (I8MM)	✗	✓	✓	✓

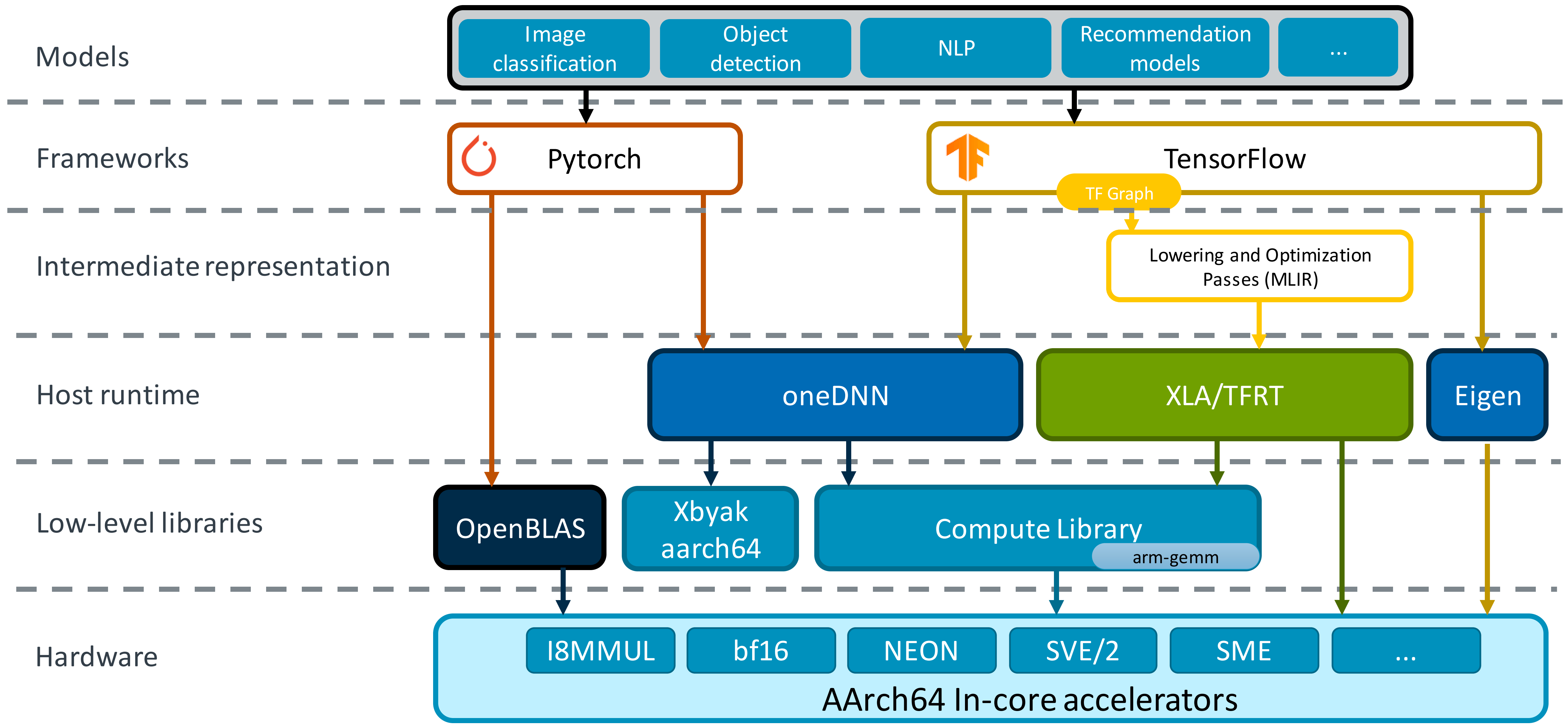
MACs/cycle in Arm Neoverse Cores

	Neoverse-N1	Neoverse-V1	Neoverse-N2	Neoverse-V2
FP32 MACs/cycle	8	16	8	16
FP16 MACs/cycle	16	32	16	32
BF16 MACs/cycle	N/A	32 (BFDOT) 64 (BFMMLA)	16 (BFDOT) 32 (BFMMLA)	32 (BFDOT) 64 (BFMMLA)
INT8 MACs/cycle	32 (DOT)	64 (DOT) 128 (I8MM)	32 (DOT) 64 (I8MM)	64 (DOT) 128 (I8MM)

Note: FLOP/cycle (or OP/cycle for int8) = 2 x MACs/cycle x clock speed

arm

Software stack



Previous work

The Arm Compute Library

Optimized low-level kernels for Arm CPUs

Key Features:

- Over 100 machine learning functions
- Targeting Arm CPUs and GPUs
- Multiple convolution algorithms (GEMM, Winograd, FFT and Direct)
- Multiple data types: f32, f16, s8, u8, bf16
- Micro-architecture optimization for key ML primitives
- Highly configurable build options enabling lightweight binaries
- Open-source, via permissive MIT license

https://github.com/ARM-software/ComputeLibrary

oneDNN + ACL

- Accelerated primitives in oneDNN using ACL
  - Matmul + inner product
  - Convolutions (GEMM, indirect, Winograd, depthwise)
  - Eltwise
  - Reorder
  - Softmax
  - Batch/layer norm
  - Binary
- acl_post_ops_t for matmul/ip/conv as a fallback for unfusable post ops
- “Fixed format” weights allows for reorders to be only done once
  - Using TF_ONEDNN_ASSUME_FROZEN_WEIGHTS=1

xbyak_aarch64

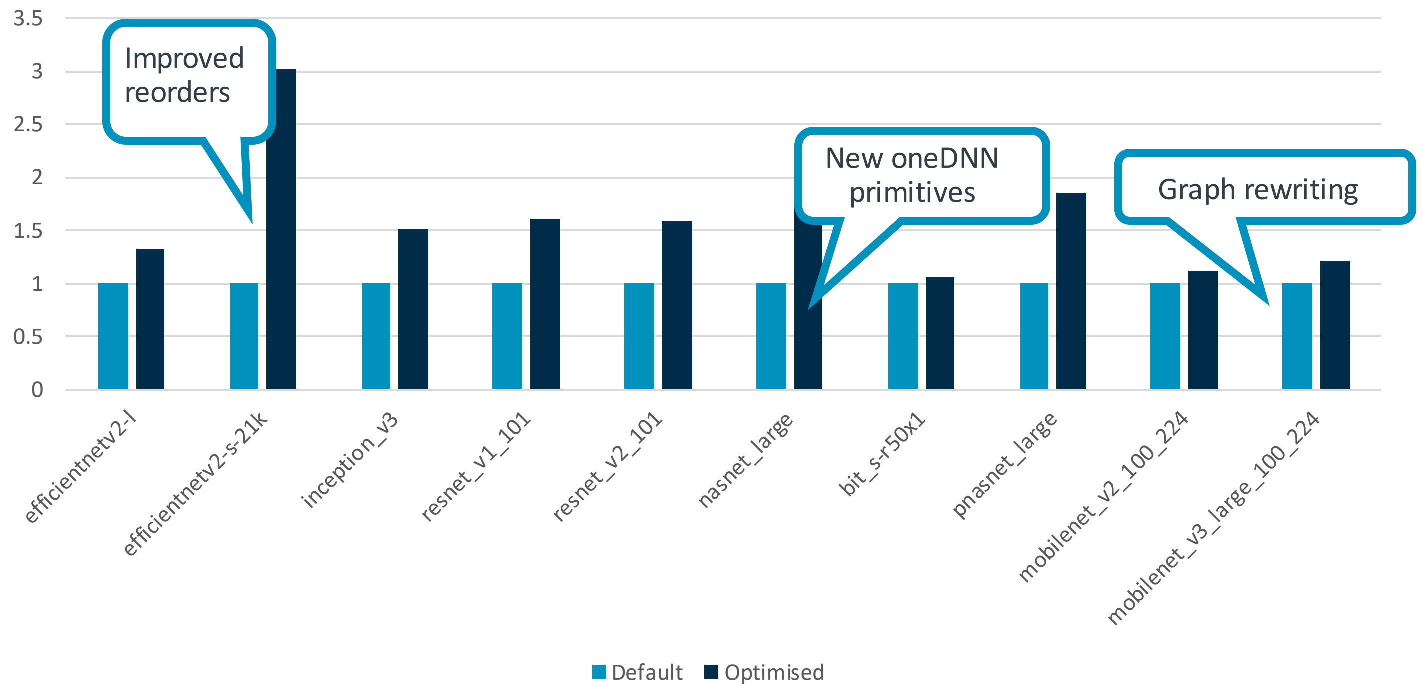
- JIT assembler for AArch64 CPUs by C++

  "Xbyak_aarch64 is a C++ library which enables run-time assemble coding with the AArch64 instruction set of Arm(R)v8-A architecture. Xbyak_aarch64 is based on Xbyak developed for x86_64 CPUs by MITSUNARI Shigeo."
- Use within oneDNN predominantly developed by Fujitsu to accelerate A64FX cores (512bit SVE)
- Accelerated primitives in oneDNN
  - Conv (only 512bit SVE)
  - Matmul (using brgemm)
  - Reorders
  - Softmax
  - Pooling
  - Eltwise

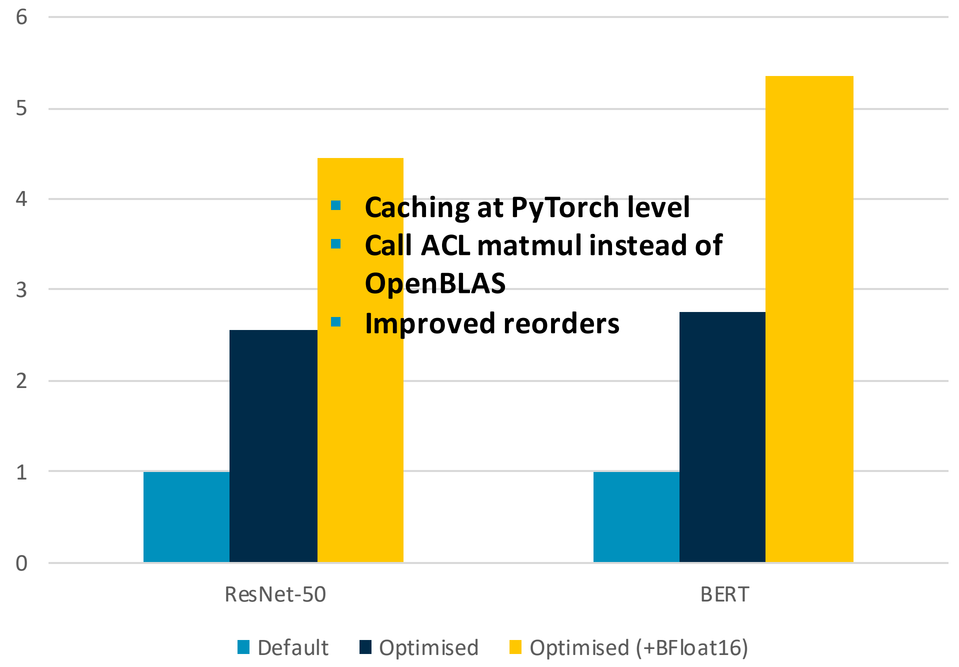
https://github.com/fujitsu/xbyak_aarch64

Performance improvements

Performance improvements on subset of image classification models
(normalized to baseline without optimisations)
c7g.8xlarge, 16 threads, batch 1



Performance progression on models from MLPerf Suite
(normalized to baseline without optimisation)
c7g.8xlarge, 16 threads, batch 8



Ongoing work

Caching and memory allocation: “stateless”

- The oneDNN + ACL operators currently
  - Allocate their own memory
  - Store pointers to tensors
- They have state which is **not captured by the problem descriptor**
- This is why we must use the resource mapper and a mutex
- Ongoing work
  - Work is ongoing to expose a “stateless” ACL
  - Move allocations to oneDNN scratchpad
- This should allow us
  - To make use of oneDNN caching => reduce tech debt
  - Allow concurrent use of a primitive => reduce startup cost
  - Allow sharing of temporary => reduce allocations

Dynamic quantization

- Quantize f32 to i8 at runtime based on actual inputs
- Suitable s8:s8:f32 matmul kernels in ACL 24.04 and oneDNN v3.5
  - >10x faster than gemm:jit
  - ~4x faster than acl F32
- Pending PRs in iDeep and PyTorch to enable their use
- Ongoing work to optimize for smaller problems and high thread counts

Static quantization

- Quantize f32 to i8 with scales and shifts fixed at init time based on input distributions
- Much harder than dynamic! User story for static quantization is still unclear in PyTorch
- Why bother? Allows you to keep inputs i8 between layers (no dequant/requant)
  - Is this a big difference in practice? Difference is not O(n^3). Even less difference if accumulator is f32
- To enable this in PyTorch we need u/s8:s8:u/s8 kernels in oneDNN
- oneDNN requires bias + post ops to be done in f32
- We are enabling s8:s8:s8 matmul using separate s8:s8:f32 + post ops + quantize
- We are working on a s8:s8:s8 convolution using fused requantization in ACL (no posts/bias yet)
- Still not sure if we can do s32 bias

Other work

- JITed reorder for bf16 reorders
- Pure bf16 matmul and convolution (for torch.autocast)
- Aarch64 brgemm (Fujistsu)
- Expanded support in JITed kernels (batch norm, pooling)

Future work

KlediAI

"KleidiAI is an open-source library that provides optimized performance-critical routines, also known as micro-kernels, for artificial intelligence (AI) workloads tailored for Arm® CPUs."

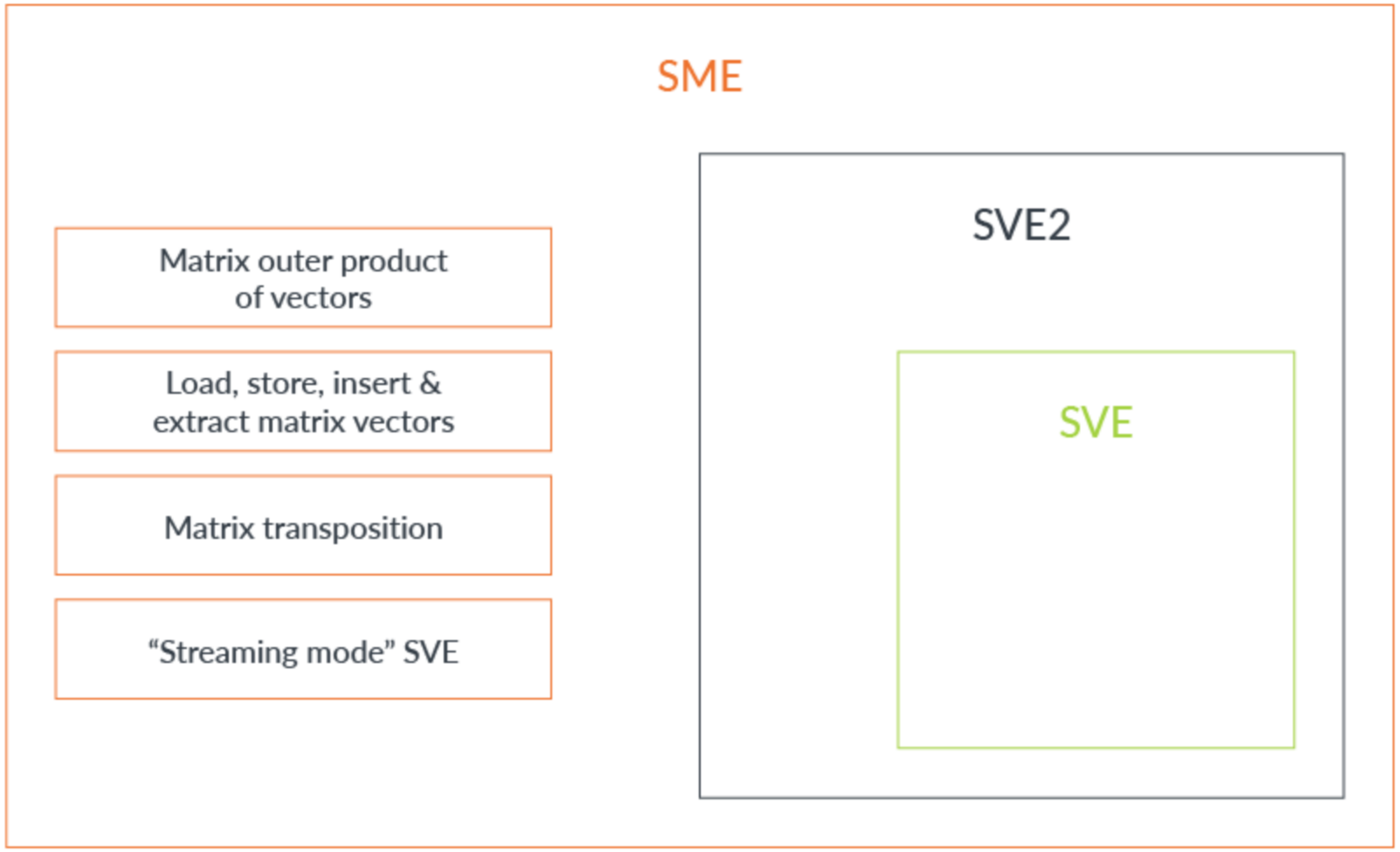
i.e. low level integration

- ACL was designed as a runtime library (manages threads and memory, has a narrow API)
- It has allowed us to quickly accelerate a lot of operations
- …but as we do more, the oneDNN-ACL API friction is growing
- Stateless ACL is a step in the right direction
- But we could go even lower

https://gitlab.arm.com/kleidi/kleidiai

https://newsroom.arm.com/blog/arm-kleidi

Scalable Matrix Extension (SME)


SME
Matrix outer product of vectors
Load, store, insert & extract matrix vectors
Matrix transposition
"Streaming mode" SVE
SVE2
SVE

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה
ధన్యవాదములు