LPGEMM Enhancements in AOCL BLAS

Bhaskar Nallani
Mithun Mohan
Meghana Vankadari

AMD
together we advance_

Agenda

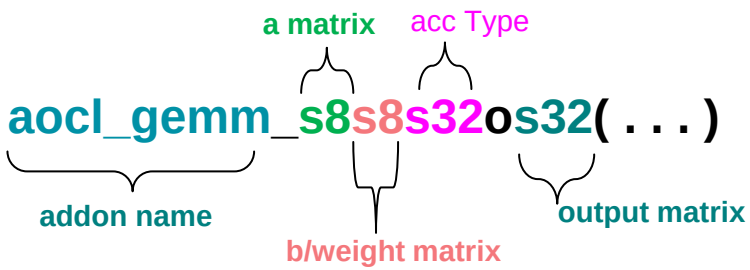
- Introduction to AOCL-BLAS and LPGEMM Addon
- LPGEMM API: Signature, Supported APIs and Usage
- Fusing Post-Ops with GEMM
- Standalone API for Elementwise Ops
- JIT based BF16 Kernel Generation
- Performance and Feature Improvements
- Q&A

Introduction to AOCL-BLAS and LPGEMM Addon

- AOCL (AMD optimizing CPU Libraries) is AMD's CPU Math Library tuned for AMD processors.
  - AOCL-BLAS is a fork of BLIS library optimized as part of AOCL.
  - AMD LPGEMM GitHub: https://github.com/amd/blis/tree/aocl-lpgemm
  - AMD Toolchain Support: toolchainsupport@amd.com
- **L**ow **P**recision **GEMM** (LPGEMM) was added as an addon named **aocl_gemm** in AOCL-BLAS 4.0.
  - Need for an efficient Low Precision GEMM has increased significantly in recent times in Deep Learning Inferences.
  - Uses reduced-precision data types like INT8, BF16, or even lower (e.g., INT4) instead of the standard FP32 or FP64.
  - While using LPGEMM APIs user should consider to use weights as B matrix and activations as A matrix, where B matrix data is expected reordered to use kernels with advanced instructions like AVX512_VNNI and AVX512_BF16.
  - Optimized for efficiency in terms of both computation and memory usage. Lower precision allows for faster computations and reduced memory bandwidth.
  - Often involves techniques like row-wise quantization and outlier-aware quantization to minimize accuracy loss while maintaining efficiency which needs mixed precision APIs and post-operations immediately after or before GEMM.

LPGEMM APIs: Signature and Support

- API Naming Conventions



- Supported API's

API Name	Data Type	ISA
aocl_gemm_<s8|u8>s8s32o<s32|s8>( )	INT8	AVX512_VNNI
aocl_gemm_<s8|u8>s8s16o<s16|s8|u8>( )	INT8	AVX2
aocl_gemm_bf16bf16f32o<f32|bf16>( )	BF16	AVX512_BF16
aocl_gemm_f32f32f32of32( )	Float	AVX2/AVX512
aocl_gemm_bf16s4f32o<bf16|f32>( )	Mixed Precision	AVX512_BF16

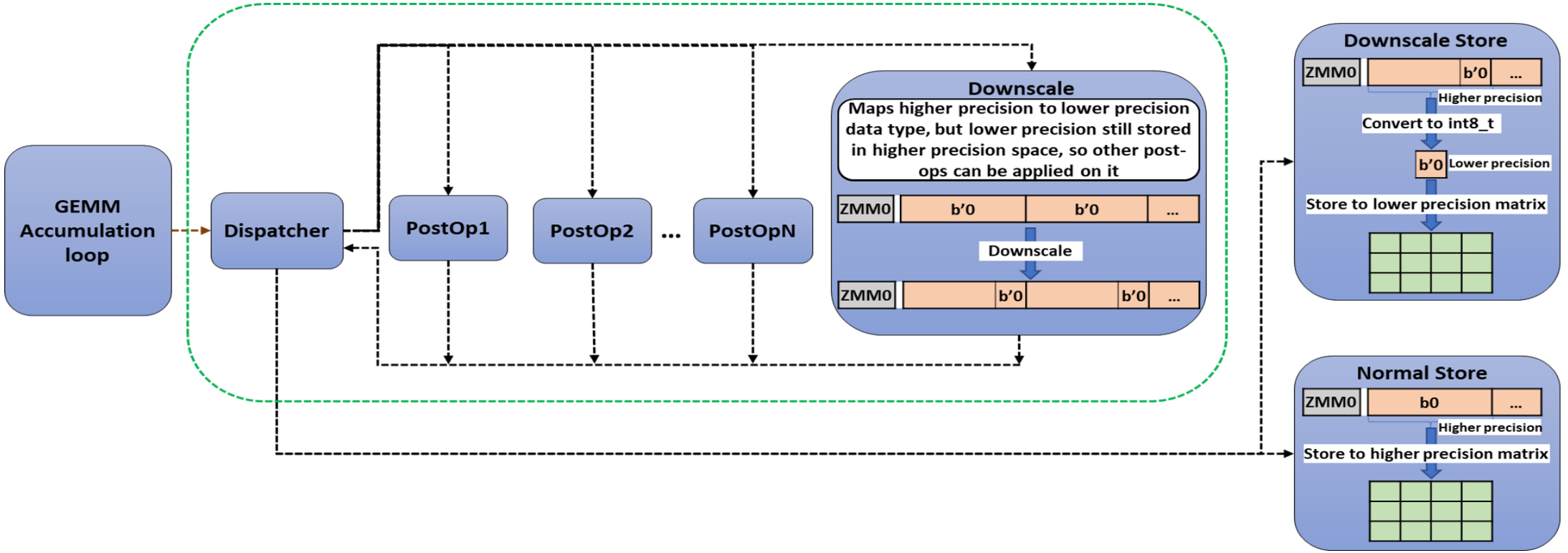
LPGEMM APIs: Usage

- Example usage of int8 aocl_gemm API along with reorder

```
dim_t size = aocl_gemm_get_reorder_size_s8s8s32os32(order, trans, mat_type, k, n);

char * b_reorder = (char *) aligned_malloc(size);

aocl_gemm_reorder_s8s8s32os32(order, trans, mat_type, *b, *b_reorder, n, k, ldb);

aocl_gemm_s8s8s32os32(order,  transa, transb,
                      m, n, k, alpha,
                      *a, lda,  mem_tag_a,
                      *b_reorder, ldb,  mem_tag_b,
                      beta, c, ldc,
                      *post_op);
```

Fusing Post-Ops with GEMM

- Fusing post-ops with GEMM at register level avoids multiple stores and loads to memory.
- Efficient post-ops dispatch with computed goto as illustrated in below diagram.



Fusing Post-Ops with GEMM Cont.

- Wide range of Post-ops are supported.
- Framework is enhanced to support applying same post-op multiple times with different ops data.
  - For example, scaling before activation and scaling as part of downscaling (before storing) with different scale factors.
- Supporting a max of 8 post-ops in fusing with all GEMM APIs.
- Post-ops are optimized for AVX2 and AVX512.

Eltwise ops	Description
BIAS	C (m x n) = [Beta*C + alpha*A*B] + bias_vector (1 x n) Adding bias per channel
ReLU	Rectified Linear Unit ReLU(x) = max(0,x)
PReLU	Parametric Rectified Linear Unit f(x) = (alpha*x) when x < 0 and x when x > 0
GeLU Tanh	Gaussian Error Linear Unit with approximation method as Tanh f(x) = 0.5* x * (1 + tanh (0.797884 * (x + (0.044715 * x^3))))
GeLU Erf	Gaussian Error Linear Unit f(x) = 0.5* x * (1 + erf (x * 0.707107))
Mat Add	C := (beta * C + alpha * A * B) + D Elementwise Addition
Mat Mul	C := (beta * C + alpha * A * B) **x** D **x** - Elementwise Multiplication
Scale	Scaling Supports Per Tensor/Channel
SWISH	Sigmoid Weighted Linear Unit when beta=1 swish(x) = x*sigmoid(beta*x)
CLIP	Clip the output to a given min and max values

Standalone API for Elementwise Ops

input matrix type

aocl_gemm_eltwise_ops_bf16o<bf16|f32>(order, transa, transb,
m , n,
a, lda,
b, ldb,
postops_struct)

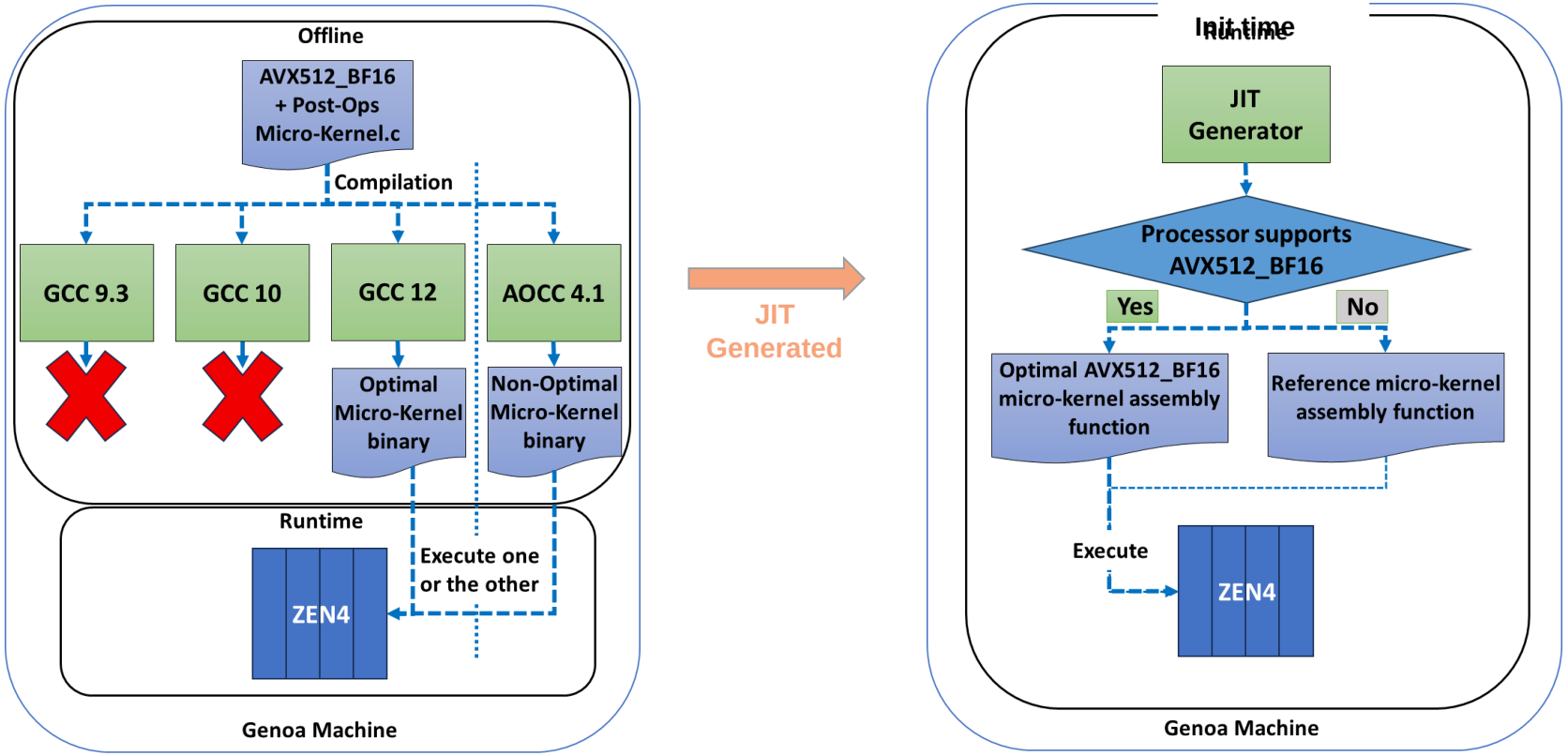
addon name

output matrix type

	NR	NR	NR	NR												
MR	ZMM0	ZMM1	ZMM2	ZMM3									ZMM0	ZMM1	ZMM2	ZMM3
	ZMM4	ZMM5	ZMM6	ZMM7									ZMM4	ZMM5	ZMM6	ZMM7
	ZMM8	ZMM9	ZMM10	ZMM11	→	PostOp1	→	PostOp2	→	...	→	PostOpN	=	ZMM8	ZMM9	ZMM10	ZMM11
	ZMM12	ZMM13	ZMM14	ZMM15		ZMM24		ZMM25				ZMM24		ZMM12	ZMM13	ZMM14	ZMM15
	ZMM16	ZMM17	ZMM18	ZMM19								ZMM25		ZMM16	ZMM17	ZMM18	ZMM19
	ZMM20	ZMM21	ZMM22	ZMM23								ZMM26		ZMM20	ZMM21	ZMM22	ZMM23

JIT based BF16 Kernel Generation

- GCC10.3 and below versions don’t support BF16 instructions, but Zen4 does support.
- Implemented BF16 kernels using JIT to provide support across variety of compilers/OS on Zen4.
- LPGEMM uses Xbyak (https://github.com/herumi/xbyak) to generate kernels Just-In-Time.



Performance Optimizations

- Downscale APIs where accumulation size is lesser than output size need an intermediate buffer to store.
  - BLIS_BUFFER_FOR_A_BLOCK was used in AOCL-BLAS4.2 which has a lock.
  - Multithread performance improved when changing the allocated buffer to BLIS_BUFFER_FOR_GEN_USE type

- Suboptimal code was generated by GCC from intrinsics for int8 fringe kernels where m<=4
  - Introduced some dummy instructions such that GCC generates the best code.
  - Performance improved by 15% for those individual kernels in the best case.

- Following optimizations done when n == 1 (LPGEMV) in all LPGEMM APIs
  - Extended MR from 6 to 16 to increase register usage.
  - Optimal parallelization is done only in m dimension
  - Avoided reorder of B matrix to eliminate NC, NR loops.

- Added support for Transpose and Column Major for all applicable LPGEM API’s

- When B matrix is reordered and post-ops are enabled column major is not supported!

Questions

COPYRIGHT AND DISCLAIMER

AMD, the AMD Arrow logo, EPYC and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate releases, for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.