



BLAS Extension APIs – GEMM Pack and Compute

**Arnav Sharma
Eashan Dash
Meghana Vankadari**

Agenda

Introduction

Problem Statement

BLAS Extension APIs – Pack and Compute

Usage

Q&A

Introduction

- AOCL (AMD Optimizing CPU Libraries) is a set of numerical libraries optimized for AMD processors based on the AMD “Zen” core architecture and generations.
 - AOCL-BLAS is a fork of BLIS library optimized as part of AOCL.
 - Github: <https://github.com/amd/blis>
 - AMD Toolchain Support: toolchainsupport@amd.com
- **GEMM (GEneral Matrix-Multiply)**
 - GEMM is a widely used linear algebra operation of the form $C := \text{beta} * C + \text{alpha} * \text{op}(A) * \text{op}(B)$.
 - The current approach to solve the GEMM operation involves a 5-loop algorithm which utilizes the concept of “packing”.
 - Packing aims to rearrange the matrices into blocks of contiguous memory aligned with the cache of the CPU therefore minimizing TLB and cache misses.

$$\begin{bmatrix} a_{11} & \cdots & a_{1k} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mk} \end{bmatrix} \longrightarrow \begin{bmatrix} a_{1,1} & \cdots & a_{1,k} \\ \vdots & \ddots & \vdots \\ a_{4,1} & \cdots & a_{4,k} \\ \vdots & & \\ a_{(m-3),1} & \cdots & a_{(m-3),k} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,k} \end{bmatrix}$$

Figure: Row-Major packing of A matrix.

Problem Statement

- The GEMM operation is widely used in various workloads and there exist use-cases wherein there are multiple GEMM invocations which have one or more common matrices.
- In such cases, with the current approach, the re-used matrix gets packed for each call thus resulting in packing overhead costs.

$$\begin{bmatrix} W_{00} & \cdots & W_{0n} \\ \vdots & \ddots & \vdots \\ W_{k0} & \cdots & W_{kn} \end{bmatrix}$$

W

$$\begin{bmatrix} X_{00} & \cdots & X_{0n} \\ \vdots & \ddots & \vdots \\ X_{k0} & \cdots & X_{kn} \end{bmatrix}$$

X

$$\begin{bmatrix} W_{0,0} & \cdots & W_{0,n} \\ \vdots & \ddots & \vdots \\ W_{3,0} & \cdots & W_{3,n} \end{bmatrix}$$

⋮

$$\begin{bmatrix} C_{X_{00}} & \cdots & C_{X_{0n}} \\ \vdots & \ddots & \vdots \\ C_{X_{m0}} & \cdots & C_{X_{mn}} \end{bmatrix}$$

C_x

$$\begin{bmatrix} W_{(m-3),0} & \cdots & W_{(m-3),n} \\ \vdots & \ddots & \vdots \\ W_{m,0} & \cdots & W_{m,n} \end{bmatrix}$$

Packed W

Problem Statement

- The GEMM operation is widely used in various workloads and there exist use-cases wherein there are multiple GEMM invocations which have one or more common matrices.
- In such cases, with the current approach, the re-used matrix gets packed for each call thus resulting in packing overhead costs.

$$\begin{bmatrix} w_{00} & \dots & w_{0n} \\ \vdots & \ddots & \vdots \\ w_{k0} & \dots & w_{kn} \end{bmatrix}$$

W

$$\begin{bmatrix} x_{00} & \dots & x_{0n} \\ \vdots & \ddots & \vdots \\ x_{k0} & \dots & x_{kn} \end{bmatrix} \times \begin{bmatrix} w_{00} & \dots & w_{0n} \\ \vdots & \ddots & \vdots \\ w_{k0} & \dots & w_{kn} \end{bmatrix} = \begin{bmatrix} c_{x_{00}} & \dots & c_{x_{0n}} \\ \vdots & \ddots & \vdots \\ c_{x_{m0}} & \dots & c_{x_{mn}} \end{bmatrix}$$

X **W** **C_x**

$$\begin{bmatrix} y_{00} & \dots & y_{0n} \\ \vdots & \ddots & \vdots \\ y_{k0} & \dots & y_{kn} \end{bmatrix}$$

Y

$$\begin{bmatrix} w_{0,0} & \dots & w_{0,n} \\ \vdots & \ddots & \vdots \\ w_{3,0} & \dots & w_{3,n} \end{bmatrix}$$

⋮

$$\begin{bmatrix} c_{y_{00}} & \dots & c_{y_{0n}} \\ \vdots & \ddots & \vdots \\ c_{y_{m0}} & \dots & c_{y_{mn}} \end{bmatrix}$$

C_y

$$\begin{bmatrix} w_{(m-3),0} & \dots & w_{(m-3),n} \\ \vdots & \ddots & \vdots \\ w_{m,0} & \dots & w_{m,n} \end{bmatrix}$$

Packed W

Problem Statement

- The GEMM operation is widely used in various workloads and there exist use-cases wherein there are multiple GEMM invocations which have one or more common matrices.
- In such cases, with the current approach, the re-used matrix gets packed for each call thus resulting in packing overhead costs.

$$\begin{bmatrix} W_{00} & \dots & W_{0n} \\ \vdots & \ddots & \vdots \\ W_{k0} & \dots & W_{kn} \end{bmatrix}$$

W

$$\begin{bmatrix} x_{00} & \dots & x_{0n} \\ \vdots & \ddots & \vdots \\ x_{k0} & \dots & x_{kn} \end{bmatrix} \times \begin{bmatrix} W_{00} & \dots & W_{0n} \\ \vdots & \ddots & \vdots \\ W_{k0} & \dots & W_{kn} \end{bmatrix} = \begin{bmatrix} c_{x_{00}} & \dots & c_{x_{0n}} \\ \vdots & \ddots & \vdots \\ c_{x_{m0}} & \dots & c_{x_{mn}} \end{bmatrix}$$

X **W** **C_x**

$$\begin{bmatrix} y_{00} & \dots & y_{0n} \\ \vdots & \ddots & \vdots \\ y_{k0} & \dots & y_{kn} \end{bmatrix} \times \begin{bmatrix} W_{00} & \dots & W_{0n} \\ \vdots & \ddots & \vdots \\ W_{k0} & \dots & W_{kn} \end{bmatrix} = \begin{bmatrix} c_{y_{00}} & \dots & c_{y_{0n}} \\ \vdots & \ddots & \vdots \\ c_{y_{m0}} & \dots & c_{y_{mn}} \end{bmatrix}$$

Y **W** **C_y**

$$\begin{bmatrix} Z_{00} & \dots & Z_{0n} \\ \vdots & \ddots & \vdots \\ Z_{k0} & \dots & Z_{kn} \end{bmatrix}$$

Z

$$\begin{bmatrix} W_{0,0} & \dots & W_{0,n} \\ \vdots & \ddots & \vdots \\ W_{3,0} & \dots & W_{3,n} \end{bmatrix}$$

⋮

$$\begin{bmatrix} C_{Z_{00}} & \dots & C_{Z_{0n}} \\ \vdots & \ddots & \vdots \\ C_{Z_{m0}} & \dots & C_{Z_{mn}} \end{bmatrix}$$

C_Z

$$\begin{bmatrix} W_{(m-3),0} & \dots & W_{(m-3),n} \\ \vdots & \ddots & \vdots \\ W_{m,0} & \dots & W_{m,n} \end{bmatrix}$$

Packed W

Problem Statement

$$\begin{aligned} \begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{k,1} & \cdots & x_{k,n} \end{bmatrix} \times \begin{bmatrix} w_{1,1} & \cdots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{k,1} & \cdots & w_{k,n} \end{bmatrix} &= \begin{bmatrix} c_{x_{1,1}} & \cdots & c_{x_{1,n}} \\ \vdots & \ddots & \vdots \\ c_{x_{m,1}} & \cdots & c_{x_{m,n}} \end{bmatrix} \\ \mathbf{X} & \quad \mathbf{W} & \quad \mathbf{C}_x \\ \\ \begin{bmatrix} y_{1,1} & \cdots & y_{1,n} \\ \vdots & \ddots & \vdots \\ y_{k,1} & \cdots & y_{k,n} \end{bmatrix} \times \begin{bmatrix} w_{1,1} & \cdots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{k,1} & \cdots & w_{k,n} \end{bmatrix} &= \begin{bmatrix} c_{y_{1,1}} & \cdots & c_{y_{1,n}} \\ \vdots & \ddots & \vdots \\ c_{y_{m,1}} & \cdots & c_{y_{m,n}} \end{bmatrix} \\ \mathbf{Y} & \quad \mathbf{W} & \quad \mathbf{C}_y \\ \\ \begin{bmatrix} z_{1,1} & \cdots & z_{1,n} \\ \vdots & \ddots & \vdots \\ z_{k,1} & \cdots & z_{k,n} \end{bmatrix} \times \begin{bmatrix} w_{1,1} & \cdots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{k,1} & \cdots & w_{k,n} \end{bmatrix} &= \begin{bmatrix} c_{z_{1,1}} & \cdots & c_{z_{1,n}} \\ \vdots & \ddots & \vdots \\ c_{z_{m,1}} & \cdots & c_{z_{m,n}} \end{bmatrix} \\ \mathbf{Z} & \quad \mathbf{W} & \quad \mathbf{C}_z \end{aligned}$$

W is being reused in each GEMM operation!!

BLAS Extension APIs – Pack and Compute

- From the problem statement, we can see that the **Matrix (W)** is being reused and thus, will have a major packing overhead as it is being packed for each inference.
- Thus, a set of 3 Extension APIs (*each for float and double types*) is implemented to handle this scenario:
 - `?gemm_pack_get_size(...)`
 - `?gemm_pack(...)`
 - `?gemm_compute(...)`
- This set of Pack and Compute Extension APIs are designed in such a way that they leverage the pre-existing optimized packing and GEMM SUP kernels. Thus, any new optimization (kernel dimensions, cache-blocking, etc.) done for these kernels will also provide performance uplift for these Extension APIs.
- Presently, this is enabled only for the AMD Zen™ code-paths and supports both Single-Threaded and Multi-Threaded implementations.

Usage

- Invoke the `?gemm_pack_get_size()` routine first to query the size of storage required for the packed matrix to be used in subsequent calls.
- Post this allocate a buffer whose size was determined using the `?gemm_pack_get_size()` routine and pass this buffer to the `?gemm_pack()` routine.
- The `?gemm_pack()` routine will scale by alpha and pack the specified matrix into the previously allocated buffer.
- Finally, invoke `?gemm_compute()` routine with this packed buffer to compute the GEMM operation ($C := \beta * C + \alpha * op(A) * op(B)$).
- *Note: If the users want to use packed buffers for both matrices, A and B, it is essential to use alpha scalar only for one of the matrices and unit-scalar for the other. Also, it is advised to use the same number of threads for both packing and compute operations.*

Usage - Snippet

```
// Assuming the reuse of B matrix.
// Calculate and get size of buffer for B
f77_char f77_identifB = 'B';

size_t b_buffer_size = sgemm_pack_get_size( &f77_identifB, &m, &n, &k );

// Allocate memory for B buffer
float* b_buffer = ( float* ) bli_malloc_user( b_buffer_size, &err );

// Pack B matrix
sgemm_pack( &f77_identifB, &f77_transB, &m, &n, &k, &alpha, &B, &ldb, b_buffer );

// Perform SGEMM operation using the above packed matrix
sgemm_compute( &f77_transA, &f77_packed, &m, &n, &k, &A1, &lda, b_buffer, &ldb1, &beta, &C1, &ldc1 );
sgemm_compute( &f77_transA, &f77_packed, &m, &n, &k, &A2, &lda, b_buffer, &ldb2, &beta, &C2, &ldc2 );
sgemm_compute( &f77_transA, &f77_packed, &m, &n, &k, &A3, &lda, b_buffer, &ldb3, &beta, &C3, &ldc3 );

// Free the memory for packed B buffer
bli_free( b_buffer );
```

Questions?

COPYRIGHT AND DISCLAIMER

©2024 Advanced Micro Devices, Inc. All rights reserved.

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.

AMD 