BLAS Extension APIs – GEMM Pack and Compute

Arnav Sharma
Eashan Dash
Meghana Vankadari

AMD
together we advance_

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 (GE**neral **M**atrix-**M**ultiply)
  - GEMM is a widely used linear algebra operation of the form C := beta * C + alpha * op(A) * 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.

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

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

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

$$\vdots$$

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

Packed W

$$\begin{bmatrix} c_{x_{00}} & \cdots & c_{x_{0\mathrm{n}}} \\ \vdots & \ddots & \vdots \\ c_{x_{\mathrm{m}0}} & \cdots & c_{x_{\mathrm{mn}}} \end{bmatrix}$$

C_x

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.

$$\underset{\mathbf{X}}{\begin{bmatrix} x_{00} & \cdots & x_{0n} \\ \vdots & \ddots & \vdots \\ x_{k0} & \cdots & x_{kn} \end{bmatrix}} \times \underset{\mathbf{W}}{\begin{bmatrix} w_{00} & \cdots & w_{0n} \\ \vdots & \ddots & \vdots \\ w_{k0} & \cdots & w_{kn} \end{bmatrix}} = \underset{\mathbf{C}_x}{\begin{bmatrix} c_{x_{00}} & \cdots & c_{x_{0n}} \\ \vdots & \ddots & \vdots \\ c_{x_{m0}} & \cdots & c_{x_{mn}} \end{bmatrix}}$$

$$\underset{\mathbf{W}}{\begin{bmatrix} w_{00} & \cdots & w_{0n} \\ \vdots & \ddots & \vdots \\ w_{k0} & \cdots & w_{kn} \end{bmatrix}}$$

$$\underset{\mathbf{Y}}{\begin{bmatrix} y_{00} & \cdots & y_{0n} \\ \vdots & \ddots & \vdots \\ y_{k0} & \cdots & y_{kn} \end{bmatrix}}$$

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

$$\vdots$$

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

Packed W

$$\underset{\mathbf{C}_y}{\begin{bmatrix} c_{y_{00}} & \cdots & c_{y_{0n}} \\ \vdots & \ddots & \vdots \\ c_{y_{m0}} & \cdots & c_{y_{mn}} \end{bmatrix}}$$

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.

$$\underset{\mathbf{X}}{\begin{bmatrix} x_{00} & \cdots & x_{0n} \\ \vdots & \ddots & \vdots \\ x_{k0} & \cdots & x_{kn} \end{bmatrix}} \times \underset{\mathbf{W}}{\begin{bmatrix} w_{00} & \cdots & w_{0n} \\ \vdots & \ddots & \vdots \\ w_{k0} & \cdots & w_{kn} \end{bmatrix}} = \underset{\mathbf{C}_x}{\begin{bmatrix} c_{x_{00}} & \cdots & c_{x_{0n}} \\ \vdots & \ddots & \vdots \\ c_{x_{m0}} & \cdots & c_{x_{mn}} \end{bmatrix}}$$

$$\underset{\mathbf{Y}}{\begin{bmatrix} y_{00} & \cdots & y_{0n} \\ \vdots & \ddots & \vdots \\ y_{k0} & \cdots & y_{kn} \end{bmatrix}} \times \underset{\mathbf{W}}{\begin{bmatrix} w_{00} & \cdots & w_{0n} \\ \vdots & \ddots & \vdots \\ w_{k0} & \cdots & w_{kn} \end{bmatrix}} = \underset{\mathbf{C}_y}{\begin{bmatrix} c_{y_{00}} & \cdots & c_{y_{0n}} \\ \vdots & \ddots & \vdots \\ c_{y_{m0}} & \cdots & c_{y_{mn}} \end{bmatrix}}$$

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

W

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

Z

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

Packed W

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

C_z

Problem Statement

$$\begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{k,1} & \cdots & x_{k,n} \end{bmatrix} \times \begin{bmatrix} w_{1,1} & \cdots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{k,1} & \cdots & w_{k,n} \end{bmatrix} = \begin{bmatrix} c_{x_{1,1}} & \cdots & c_{x_{1,\mathrm{n}}} \\ \vdots & \ddots & \vdots \\ c_{x_{\mathrm{m},1}} & \cdots & c_{x_{\mathrm{m,n}}} \end{bmatrix}$$

$$\mathbf{X} \qquad \mathbf{W} \qquad \mathbf{C}_{\boldsymbol{x}}$$

$$\begin{bmatrix} y_{1,1} & \cdots & y_{1,n} \\ \vdots & \ddots & \vdots \\ y_{k,1} & \cdots & y_{k,n} \end{bmatrix} \times \begin{bmatrix} w_{1,1} & \cdots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{k,1} & \cdots & w_{k,n} \end{bmatrix} = \begin{bmatrix} c_{y_{1,1}} & \cdots & c_{y_{1,n,}} \\ \vdots & \ddots & \vdots \\ c_{y_{\mathrm{m},1}} & \cdots & c_{y_{\mathrm{m,n}}} \end{bmatrix}$$

$$\mathbf{Y} \qquad \mathbf{W} \qquad \mathbf{C}_{\boldsymbol{y}}$$

$$\begin{bmatrix} z_{1,1} & \cdots & z_{1,n} \\ \vdots & \ddots & \vdots \\ z_{k,1} & \cdots & z_{k,n} \end{bmatrix} \times \begin{bmatrix} w_{1,1} & \cdots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{k,1} & \cdots & w_{k,n} \end{bmatrix} = \begin{bmatrix} c_{z_{1,1}} & \cdots & c_{z_{1,\mathrm{n}}} \\ \vdots & \ddots & \vdots \\ c_{z_{\mathrm{m},1}} & \cdots & c_{z_{\mathrm{m,n}}} \end{bmatrix}$$

$$\mathbf{Z} \qquad \mathbf{W} \qquad \mathbf{C}_{\boldsymbol{z}}$$

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_identifierB = 'B';

size_t b_buffer_size = sgemm_pack_get_size( &f77_identifierB, &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_identifierB, &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

AMD