Low Precision GEMM

Mithun Mohan K M, Nallani Bhaskar, Eashan Dash, HariharaSudhan S

Contents

Introduction

- 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.

- Problem Statement:
  - **L**ow **P**recision **GEMM** (LPGEMM) is not available as part of BLIS.
    - Used within this context as a blanket term for GEMM for integer (int8/int16/int32) and half precision floating point (bfloat16/float16) datatypes.
  - Performing math operations along with GEMM (as part of a sequence/pipeline) is inefficient due to separate calls to each operation.
    - Results in inefficient reuse of cached memory.

- Solution:
  - An addon (BLIS feature) based solution to enable multiple LPGEMM APIs/computations.
  - An extensive framework to support clubbing math operations along with LPGEMM.
    - Framework level support to setup the operations to perform along with the relative order after LPGEMM.
    - Supporting the math operation computation inside the LPGEMM micro-kernel.

LPGEMM APIs

LPGEMM API	typeof(A)	typeof(B)	Accumulation Type	typeof(C)	Instruction type
aocl_gemm_u8s8s32os32	uint8_t	int8_t	int32_t	int32_t	AVX512
aocl_gemm_u8s8s32os8	uint8_t	int8_t	int32_t	int8_t	AVX512
aocl_gemm_s8s8s32os32	int8_t	int8_t	int32_t	int32_t	AVX512
aocl_gemm_s8s8s32os8	int8_t	int8_t	int32_t	int8_t	AVX512
aocl_gemm_u8s8s16os16	uint8_t	int8_t	int16_t	int16_t	AVX2
aocl_gemm_u8s8s16os8	uint8_t	int8_t	int16_t	int8_t	AVX2
aocl_gemm_s8s8s16os16	int8_t	int8_t	int16_t	int16_t	AVX2
aocl_gemm_s8s8s16os8	int8_t	int8_t	int16_t	int8_t	AVX2
aocl_gemm_bf16bf16f32of32	bfloat16	bfloat16	float	float	AVX512
aocl_gemm_bf16bf16f32obf16	bfloat16	bfloat16	float	bfloat16	AVX512
aocl_gemm_f32f32f32of32	float	float	float	float	AVX512 / AVX2

Instruction friendly packing

- GEMM packing usually done as per the micro-kernel dimensions and 5 loop structure.
- In LPGEMM, packing further re-arranges data to leverage the fused operations provided by integer instructions.
- Example:
  - The aocl_gemm_u8s8s16os16 micro-kernel is implemented using VPMADDUBSW instruction.
  - VPMADDUBSW instruction works on 2 elements along the k dimension at a time.
  - B matrix packed further (apart from KCxNR in KCxNC panel) to have 2 elements along K dimension for each element in NR.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

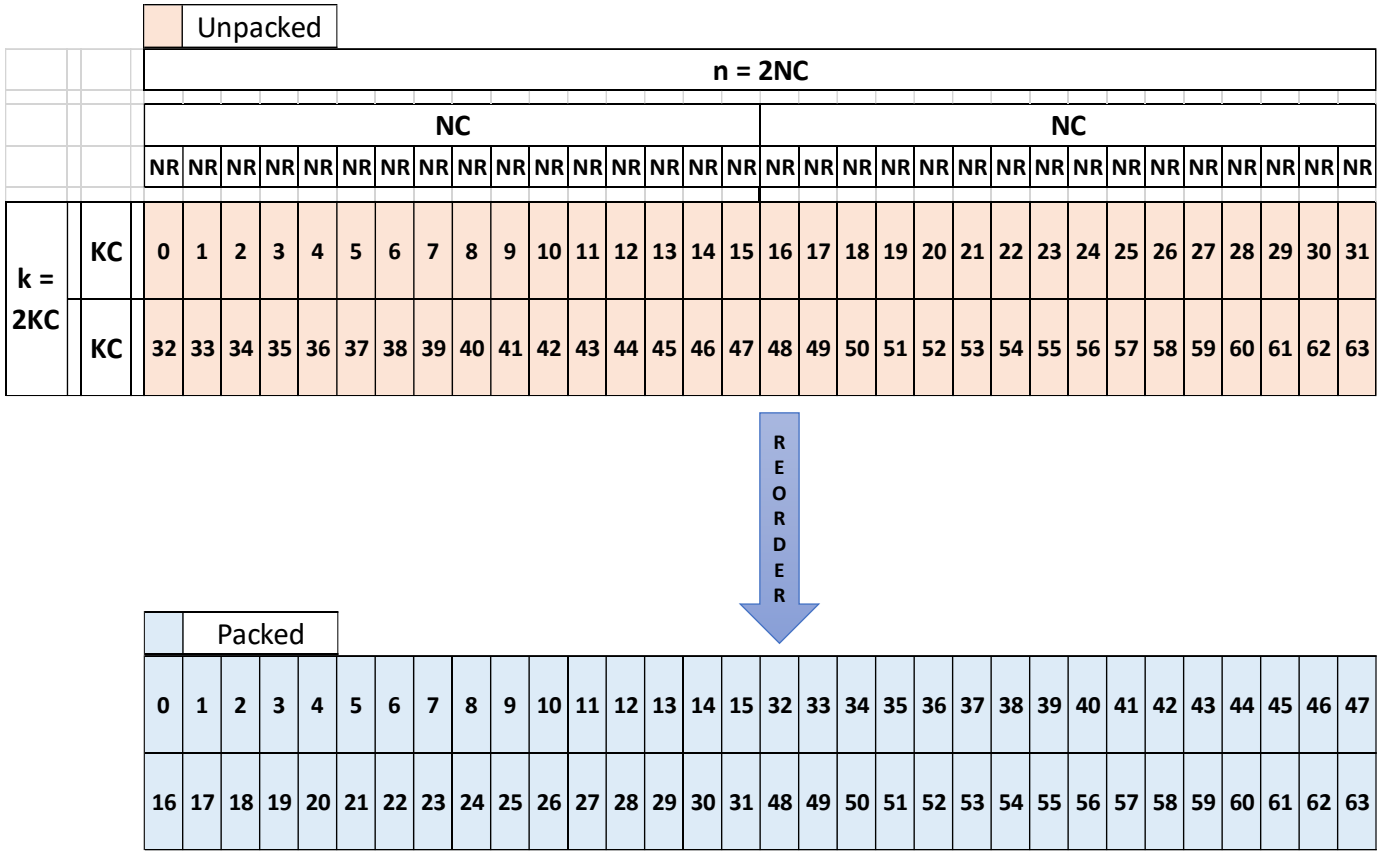
Original data access pattern is 0,1,2,3 …. 61,62,63

0	8	1	9	2	10	3	11
4	12	5	13	6	14	7	15
16	24	17	25	18	26	19	27
20	28	21	29	22	30	23	31
32	40	33	41	34	42	35	43
36	44	37	45	38	46	39	47
48	56	49	57	50	58	51	59
52	60	53	61	54	62	55	63

Packed data access pattern is 0,8,1,9 …. 62,55,63. Rearranged as 4 elements each along k dimension across NR columns till KC.

Matrix reordering

- Matrix reordering refers to packing the entire matrix (and then performing GEMM).
- One or both A & B matrix can be reordered and cached for reuse across multiple LPGEMM calls.
  - Only B reordering is supported for now.
  - Helps hide packing costs.



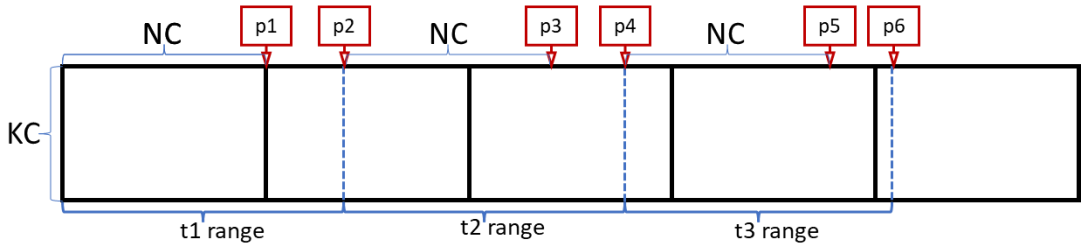
LPGEMM Reorder APIs

LPGEMM REORDER API	typeof(matrix)
aocl_get_reorder_buf_size_u8s8s32os32	
aocl_get_reorder_buf_size_s8s8s32os32	
aocl_get_reorder_buf_size_u8s8s16os16	
aocl_get_reorder_buf_size_s8s8s16os16	
aocl_get_reorder_buf_size_bf16bf16f32of32	
aocl_get_reorder_buf_size_f32f32f32of32	
aocl_reorder_u8s8s32os32	uint8_t/int8_t
aocl_reorder_s8s8s32os32	int8_t
aocl_reorder_u8s8s16os16	uint8_t/int8_t
aocl_reorder_s8s8s16os16	int8_t
aocl_reorder_bf16bf16f32of32	bfloat16
aocl_reorder_f32f32f32of32	float

Reordered matrix access in multi-threaded LPGEMM

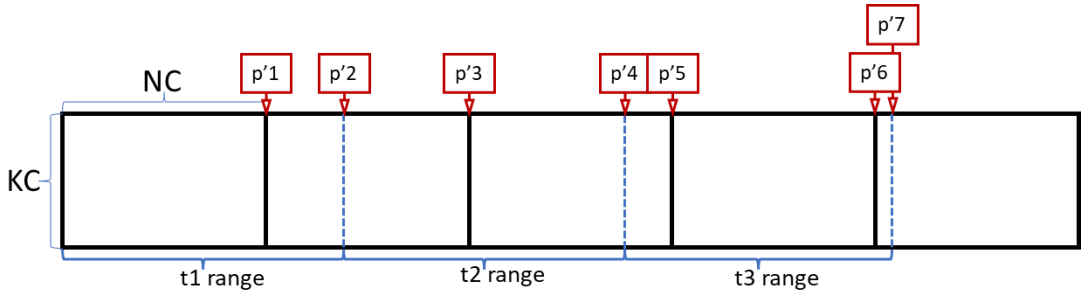
- Matrix reordering is implemented as a multi-threaded operation.
- Reordering can be done with **x** threads whereas LPGEMM can be computed using **y** threads:
  - Reordered matrix access (write in reorder API / read in LPGEMM 5 loop) should be consistent irrespective of the number of threads used.
- This invariant held by ensuring access to reorder matrix happens as if it is single threaded.

- Example using jc loop with 3 threads:



TID	jc iter 1	jc iter 2	jc iter 3
T1	0 –> p1 (=NC)	p1 –> p2 (<NC)	
T2	p2 –> p3 (=NC)	p3 –> p4 (<NC)	
T3	p4 –> p5 (=NC)	p5 –> p6 (<NC)	

Current: Per thread jc loop access when B matrix packed on the go inside 5 loop.



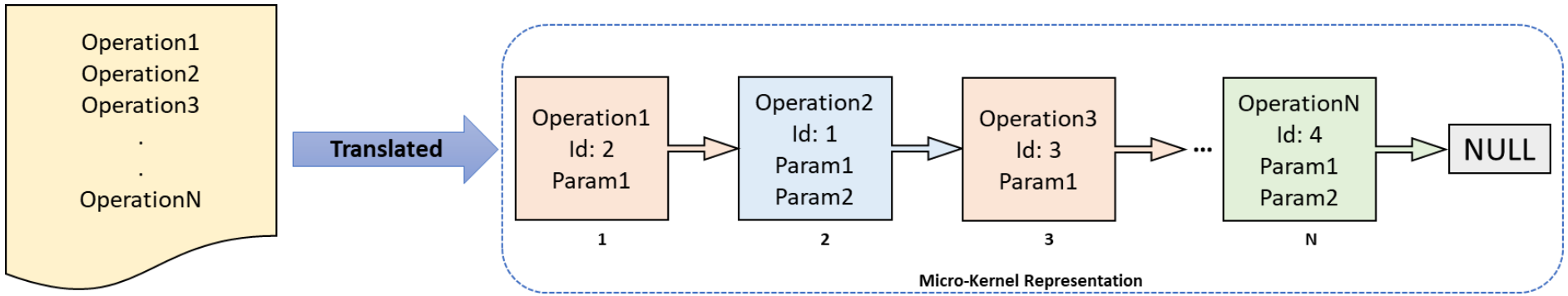
TID	jc iter 1	jc iter 2	jc iter 3
T1	0 –> p'1 (=NC)	p'1 –> p'2 (<NC)	
T2	p'2 –> p'3 (<NC)	p'3 –> p'4 (<NC)	
T3	p'4 –> p'5 (<NC)	p'5 –> p'6 (=NC)	p'6 –> p'7 (<NC)

Proposed: Per thread jc loop access when B matrix is reordered.

Clubbing math operations with LPGEMM

- 2 phase design used to support computing math operations after LPGEMM computation

- Phase 1 – Framework level:
  - User supplies a struct describing the operations to perform to the LPGEMM API.
  - The struct is converted to a linked list of operations, with each node corresponding to an operation.



Clubbing math operations with LPGEMM (cont.)

- 2 phase design used to support computing other operations after LPGEMM computation

- Phase 2 – Micro-Kernel level:
  - Math operations applied within micro-kernel on the registers immediately after LPGEMM results(C = alpha*A*B + beta*C) are computed.
  - Example for 6x64 micro-kernel (24 ZMM/512-bit registers) used in aocl_gemm_u8s8s32os32 with Op1 and Op2 applied.

	NR											
MR	ZMM0	ZMM1	ZMM2	ZMM3					ZMM0	ZMM1	ZMM2	ZMM3
	ZMM4	ZMM5	ZMM6	ZMM7					ZMM4	ZMM5	ZMM6	ZMM7
	ZMM8	ZMM9	ZMM10	ZMM11	**Op1**	ZMM24	**Op2**	ZMM25	= ZMM8	ZMM9	ZMM10	ZMM11
	ZMM12	ZMM13	ZMM14	ZMM15					ZMM12	ZMM13	ZMM14	ZMM15
	ZMM16	ZMM17	ZMM18	ZMM19					ZMM16	ZMM17	ZMM18	ZMM19
	ZMM20	ZMM21	ZMM22	ZMM23					ZMM20	ZMM21	ZMM22	ZMM23

questions?

COPYRIGHT AND DISCLAIMER

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