



Efficient TRSM Implementation for Small Dimensions

Bhaskar Nallani
Satish Kumar Nuggu

Agenda

Introduction to AOCL

TRSM Small Requirement

TRSM Small Algo

Optimization Techniques

Results

Q&A

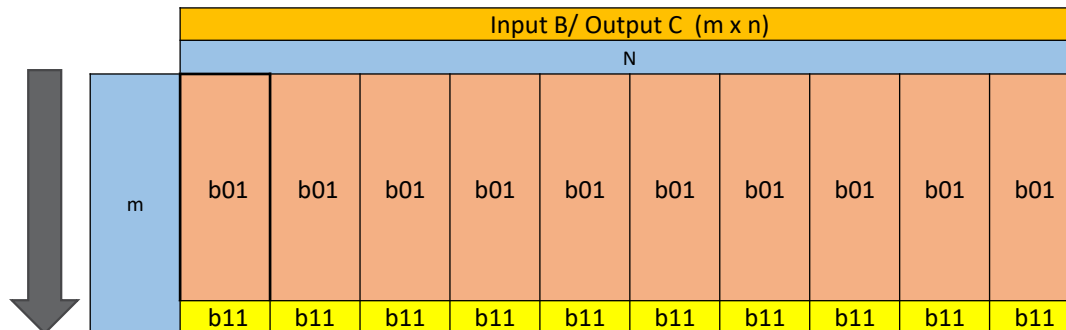
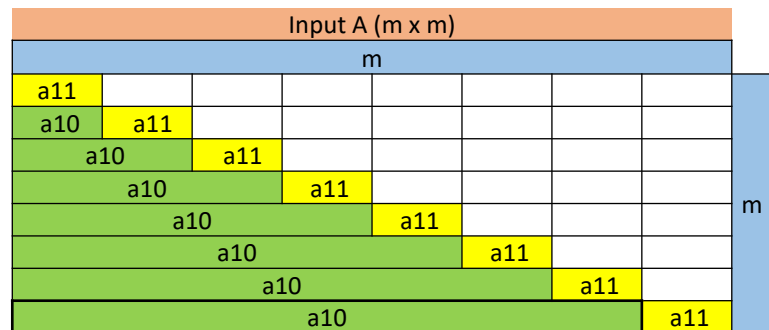
Introduction

- AOCL – AMD optimizing CPU Libraries
 - AOCL are a set of numerical libraries tuned specifically for AMD EPYC™ processor family
 - BLIS is part of this AOCL
 - <https://developer.amd.com/amd-aocl/>
 - Latest source code for BLIS is available on GitHub <https://github.com/amd/blis>
- For any issues or queries regarding the libraries, please contact aoclsupport@amd.com
- Latest version is AOCL 3.2
 - Zen3 Optimizations added

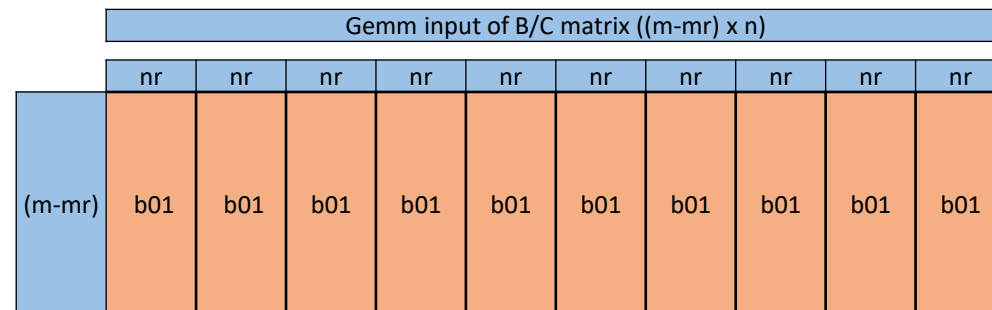
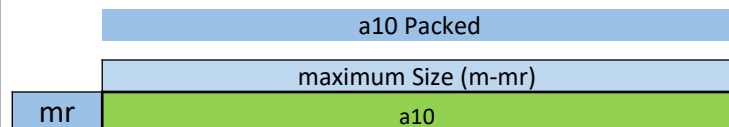
TRSM Small Requirement

- Problem Statement:
 - Existing native implementation of TRSM is suitable for larger inputs but less efficient for smaller input matrixes example range $\{m,n\} \leq 1000$ for dtrsm.
 - There is a need to implement efficient trsm implementation for small dimension inputs

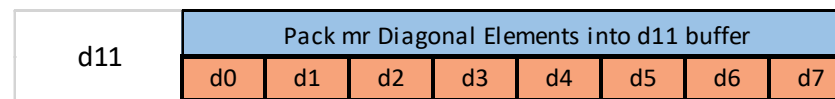
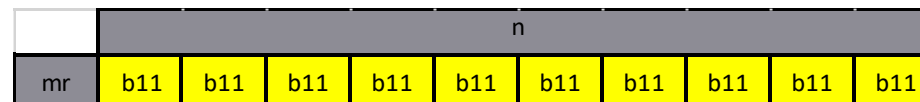
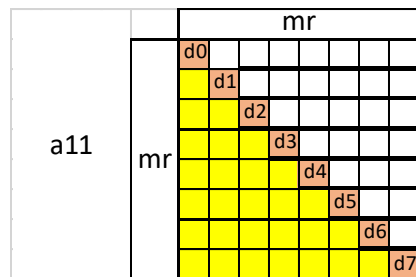
TRSM Small Algo



GEMM = (a10 * b01), which stored in ymm registers



TRSM = (b11 - ymm) / (d11)



TRSM Small Algo: Pseudo Code of Algorithm

Example: Left Cases of DTRSM (for DTRSM: D_MR : 8 and D_NR : 6)

for m cols of A ---> in steps of D_MR

Pack ($D_MR \times m'$) elements of A into a10 buffer where $0 \leq m' \leq m-8$

Invert and pack D_MR diagonal elements of a11 in d11 buffer

for n rows of B ---> in steps of D_NR

Initialize accumulator registers to zero

if(not diagonal block)

Perform GEMM using a10 and b01 at block sizes of ($D_MR \times D_NR$) and keep the GEMM output in registers

Load $D_MR \times D_NR$ block from B (b11) and multiply with alpha

Perform TRSM on b11 using a11 and GEMM output on the registers.

Update B matrix with output

end

Handle n fringe cases (where last the blocks are less than D_NR) with similar fashion with fringe kernels

end

Handle fringe cases (where last the blocks are less than D_MR and D_NR) similar fashion with fringe kernels

Optimization techniques used in this design

- Fusing GEMM and TRSM at register level
- Reuse of a10 packed buffer across micro-panels in B while performing GEMM operation
- Prefetching of next micro-panel in a10 and b01 while performing GEMM operation
- Packing diagonal elements alone into d0 buffer for better cache utilization
- Taking care of A transpose when packing

K loop in 8x6 DGEMM Kernel Without prefetch

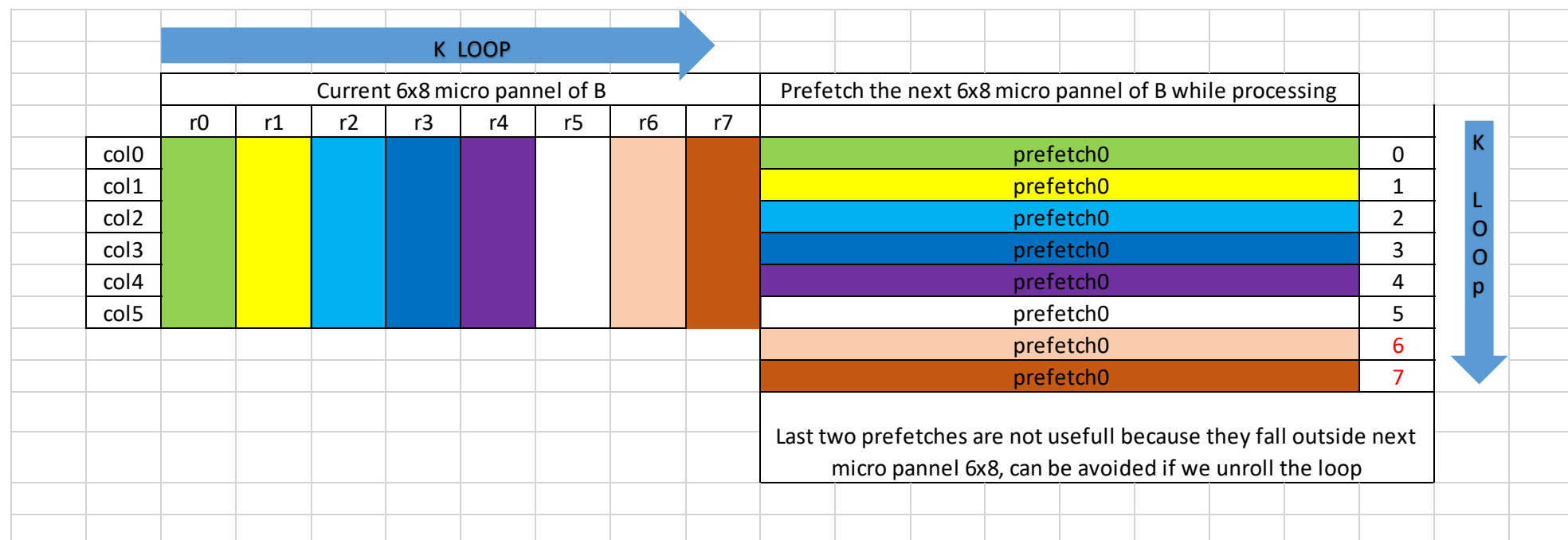
```
//LLNN
for(k = 0; k< m-8; k++)
{
    ymm0 = _mm256_loadu_pd((double const*)(pa));
    ymm1 = _mm256_loadu_pd((double const*)(pa + 4));
    ymm2 = _mm256_broadcast_sd((double const*)(pb + cs_b * 0));
    ymm8 = _mm256_fmadd_pd(ymm2, ymm0, ymm8);
    ymm12 = _mm256_fmadd_pd(ymm2, ymm1, ymm12);
    ymm2 = _mm256_broadcast_sd((double const*)(pb + cs_b * 1));
    ymm9 = _mm256_fmadd_pd(ymm2, ymm0, ymm9);
    ymm13 = _mm256_fmadd_pd(ymm2, ymm1, ymm13);
    ymm2 = _mm256_broadcast_sd((double const*)(pb + cs_b * 2));
    ymm10 = _mm256_fmadd_pd(ymm2, ymm0, ymm10);
    ymm14 = _mm256_fmadd_pd(ymm2, ymm1, ymm14);
    ymm2 = _mm256_broadcast_sd((double const*)(pb + cs_b * 3));
    ymm11 = _mm256_fmadd_pd(ymm2, ymm0, ymm11);
    ymm15 = _mm256_fmadd_pd(ymm2, ymm1, ymm15);
    ymm2 = _mm256_broadcast_sd((double const*)(pb + cs_b * 4));
    ymm4 = _mm256_fmadd_pd(ymm2, ymm0, ymm4);
    ymm6 = _mm256_fmadd_pd(ymm2, ymm1, ymm6);
    ymm2 = _mm256_broadcast_sd((double const*)(pb + cs_b * 5));
    ymm5 = _mm256_fmadd_pd(ymm2, ymm0, ymm5);
    ymm7 = _mm256_fmadd_pd(ymm2, ymm1, ymm7);
    pb += 1;          //move to next row of B
    pa += 8;          //move to next col of A
}
```

```
//RLNN
for(k = 0; k< m-8; k++)
{
    ymm0 = _mm256_loadu_pd((double const*)(pb));
    ymm1 = _mm256_loadu_pd((double const*)(pb + 4));
    ymm2 = _mm256_broadcast_sd((double const*)(pa + pda * 0));
    ymm8 = _mm256_fmadd_pd(ymm2, ymm0, ymm8);
    ymm12 = _mm256_fmadd_pd(ymm2, ymm1, ymm12);
    ymm2 = _mm256_broadcast_sd((double const*)(pa + pda * 1));
    ymm9 = _mm256_fmadd_pd(ymm2, ymm0, ymm9);
    ymm13 = _mm256_fmadd_pd(ymm2, ymm1, ymm13);
    ymm2 = _mm256_broadcast_sd((double const*)(pa + pda * 2));
    ymm10 = _mm256_fmadd_pd(ymm2, ymm0, ymm10);
    ymm14 = _mm256_fmadd_pd(ymm2, ymm1, ymm14);
    ymm2 = _mm256_broadcast_sd((double const*)(pa + pda * 3));
    ymm11 = _mm256_fmadd_pd(ymm2, ymm0, ymm11);
    ymm15 = _mm256_fmadd_pd(ymm2, ymm1, ymm15);
    ymm2 = _mm256_broadcast_sd((double const*)(pa + pda * 4));
    ymm4 = _mm256_fmadd_pd(ymm2, ymm0, ymm4);
    ymm6 = _mm256_fmadd_pd(ymm2, ymm1, ymm6);
    ymm2 = _mm256_broadcast_sd((double const*)(pa + pda * 5));
    ymm5 = _mm256_fmadd_pd(ymm2, ymm0, ymm5);
    ymm7 = _mm256_fmadd_pd(ymm2, ymm1, ymm7);
    pa += 1;          //move to next row of B
    pb += 8;          //move to next col of A
}
```

Prefetching next micro panel of loading matrix

	64 bytes					
kloop	Current working Pannel of A which is loading					
0	Load		Load			
1	Load		Load			
2	Load		Load			
3	Load		Load			
4	Load		Load			
5	Load		Load			
6	Load		Load			
7	Load		Load			
0	prefetch				Prefetch next 8x8 micro pannel while working on above micro	
1	prefetch					
2	prefetch					
3	prefetch					
4	prefetch					
5	prefetch					
6	prefetch					
7	prefetch					

Prefetching next micro panel of broadcasting matrix

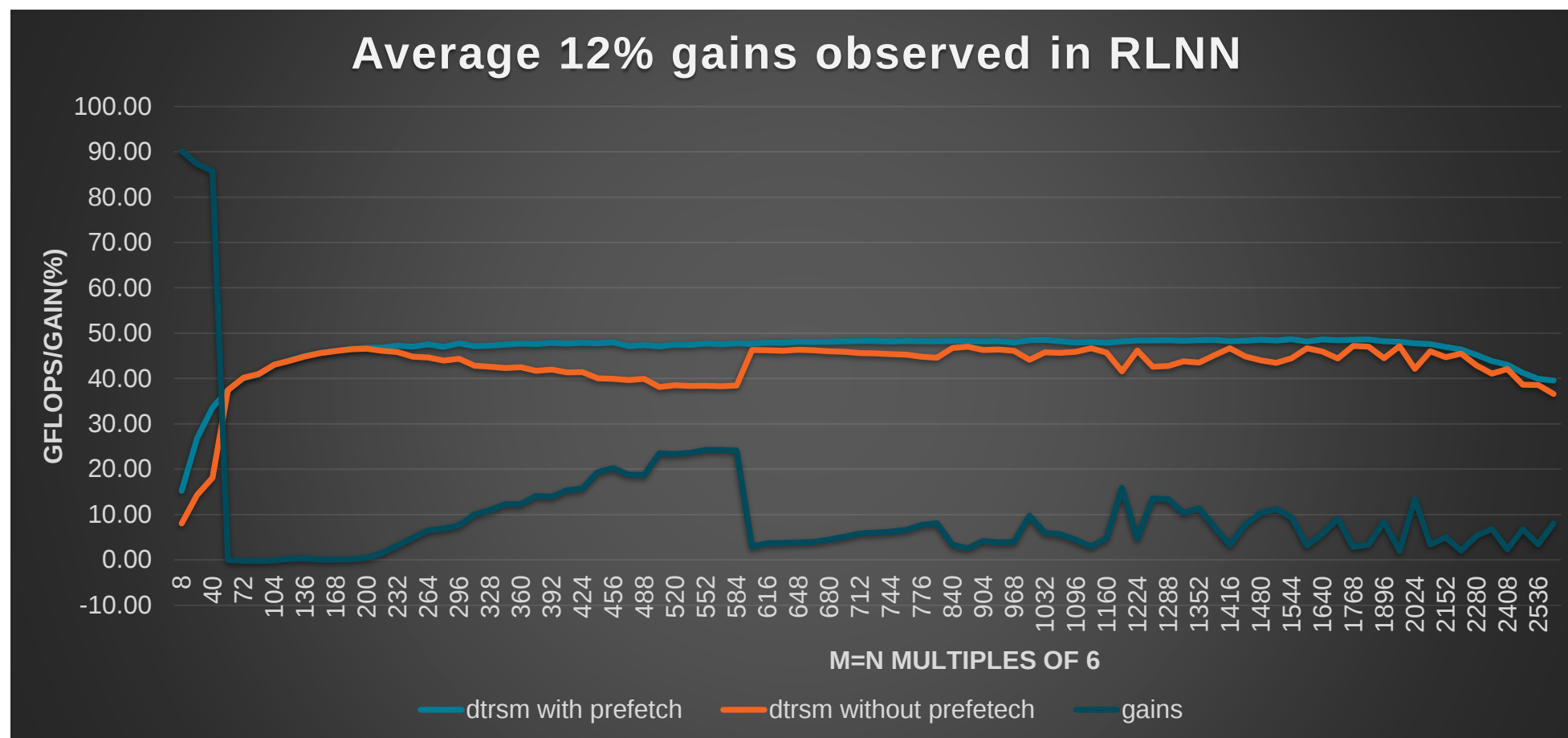


K loop in 8x6 DGEMM Kernel Without prefetch

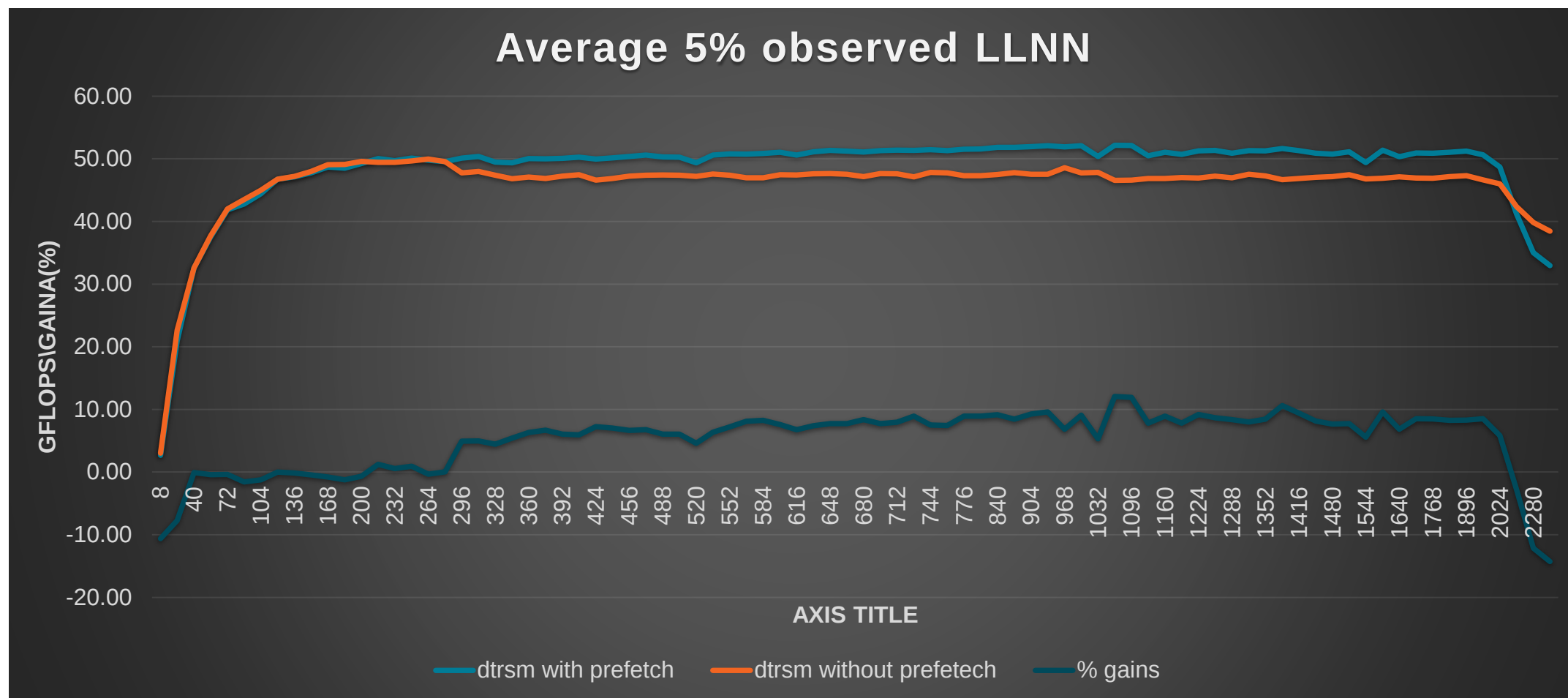
```
//LLNN
for(k = 0; k < K; k++)
{
    ymm0 = _mm256_loadu_pd((double const*)(pa));
    ymm1 = _mm256_loadu_pd((double const*)(pa + 4));
    _mm_prefetch((char*)( a10 + 8*8), _MM_HINT_T0);
    ymm2 = _mm256_broadcast_sd((double const*)(pb + cs_b * 0));
    ymm8 = _mm256_fmadd_pd(ymm2, ymm0, ymm8);
    ymm12 = _mm256_fmadd_pd(ymm2, ymm1, ymm12);
    ymm2 = _mm256_broadcast_sd((double const*)(pb + cs_b * 1));
    ymm9 = _mm256_fmadd_pd(ymm2, ymm0, ymm9);
    ymm13 = _mm256_fmadd_pd(ymm2, ymm1, ymm13);
    ymm2 = _mm256_broadcast_sd((double const*)(pb + cs_b * 2));
    ymm10 = _mm256_fmadd_pd(ymm2, ymm0, ymm10);
    ymm14 = _mm256_fmadd_pd(ymm2, ymm1, ymm14);
    if(k & 0x7) b01_prefetch += cs_b;
    else      b01_prefetch = b01 + 8;
    _mm_prefetch((char*)( b01_prefetch), _MM_HINT_T0);
    ymm2 = _mm256_broadcast_sd((double const*)(pb + cs_b * 3));
    ymm11 = _mm256_fmadd_pd(ymm2, ymm0, ymm11);
    ymm15 = _mm256_fmadd_pd(ymm2, ymm1, ymm15);
    ymm2 = _mm256_broadcast_sd((double const*)(pb + cs_b * 4));
    ymm4 = _mm256_fmadd_pd(ymm2, ymm0, ymm4);
    ymm6 = _mm256_fmadd_pd(ymm2, ymm1, ymm6);
    ymm2 = _mm256_broadcast_sd((double const*)(pb + cs_b * 5));
    ymm5 = _mm256_fmadd_pd(ymm2, ymm0, ymm5);
    ymm7 = _mm256_fmadd_pd(ymm2, ymm1, ymm7);
    pb += 1;          //move to next row of B
    pa += 8;          //pointer math to calculate next block of A for GEMM
}
```

```
//RLNN
for(k = 0; k < m-8; k++)
{
    ymm0 = _mm256_loadu_pd((double const*)(pb));
    ymm1 = _mm256_loadu_pd((double const*)(pb + 4));
    _mm_prefetch((char*)( pb + 8*cs_b), _MM_HINT_T0);
    ymm2 = _mm256_broadcast_sd((double const*)(pa + pda * 0));
    ymm8 = _mm256_fmadd_pd(ymm2, ymm0, ymm8);
    ymm12 = _mm256_fmadd_pd(ymm2, ymm1, ymm12);
    ymm2 = _mm256_broadcast_sd((double const*)(pa + pda * 1));
    ymm9 = _mm256_fmadd_pd(ymm2, ymm0, ymm9);
    ymm13 = _mm256_fmadd_pd(ymm2, ymm1, ymm13);
    ymm2 = _mm256_broadcast_sd((double const*)(pa + pda * 2));
    ymm10 = _mm256_fmadd_pd(ymm2, ymm0, ymm10);
    ymm14 = _mm256_fmadd_pd(ymm2, ymm1, ymm14);
    ymm2 = _mm256_broadcast_sd((double const*)(pa + pda * 3));
    ymm11 = _mm256_fmadd_pd(ymm2, ymm0, ymm11);
    ymm15 = _mm256_fmadd_pd(ymm2, ymm1, ymm15);
    ymm2 = _mm256_broadcast_sd((double const*)(pa + pda * 4));
    ymm4 = _mm256_fmadd_pd(ymm2, ymm0, ymm4);
    ymm6 = _mm256_fmadd_pd(ymm2, ymm1, ymm6);
    ymm2 = _mm256_broadcast_sd((double const*)(pa + pda * 5));
    ymm5 = _mm256_fmadd_pd(ymm2, ymm0, ymm5);
    ymm7 = _mm256_fmadd_pd(ymm2, ymm1, ymm7);
    pa += 1;          //move to next row of B
    pb += cs_b;       //move to next col of A
}
```

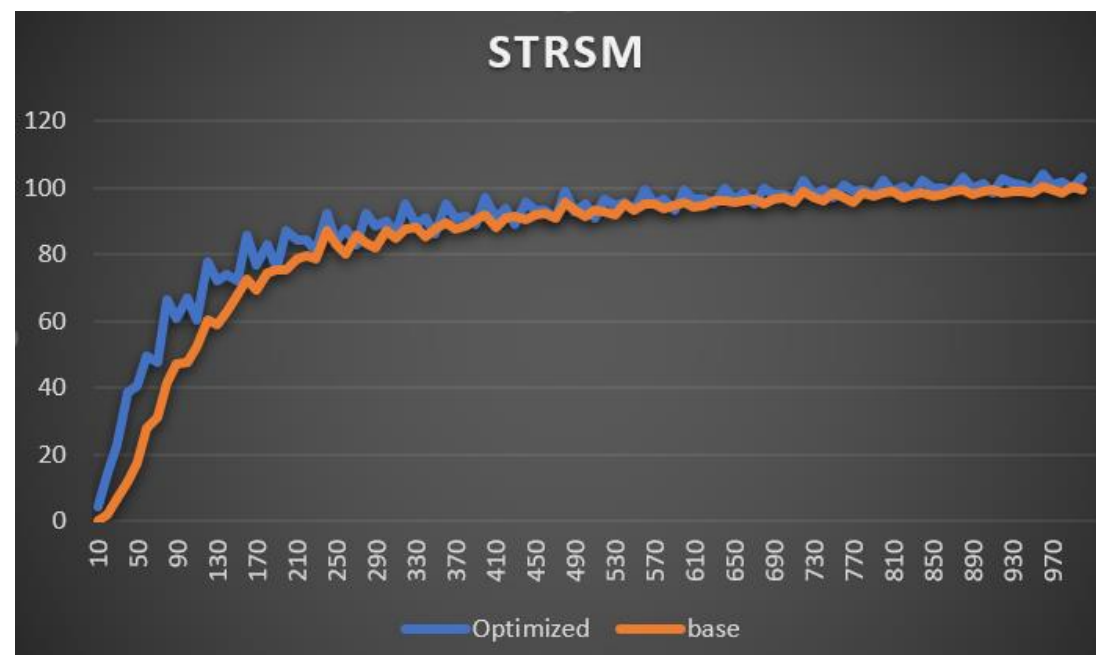
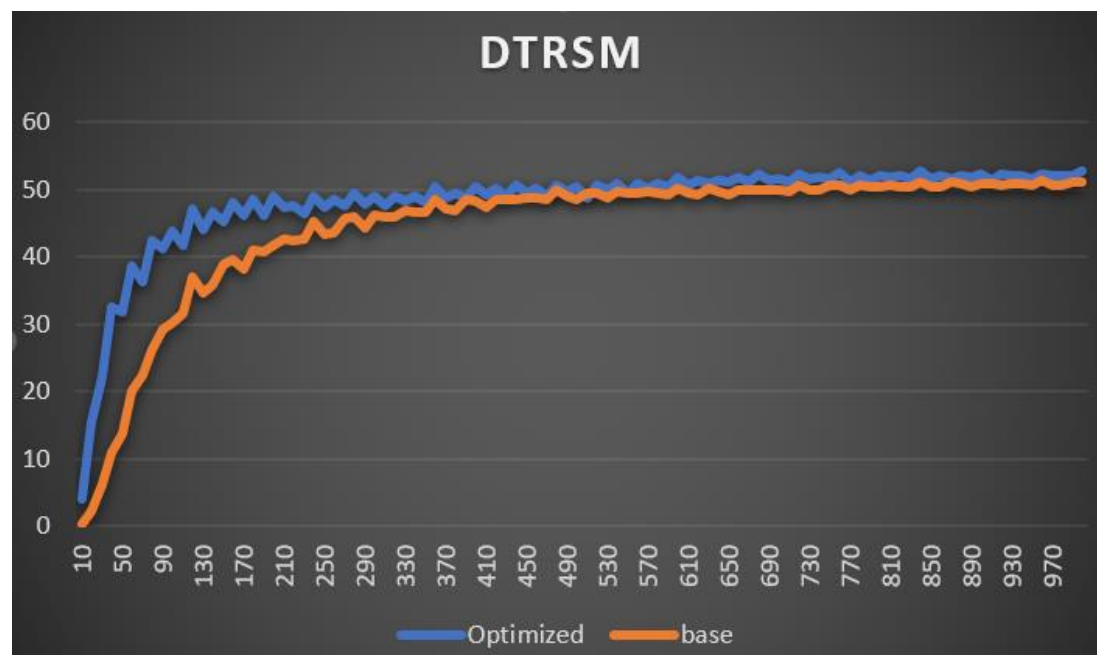
Gains observed in RLNN variant (XA=B) with Prefetch



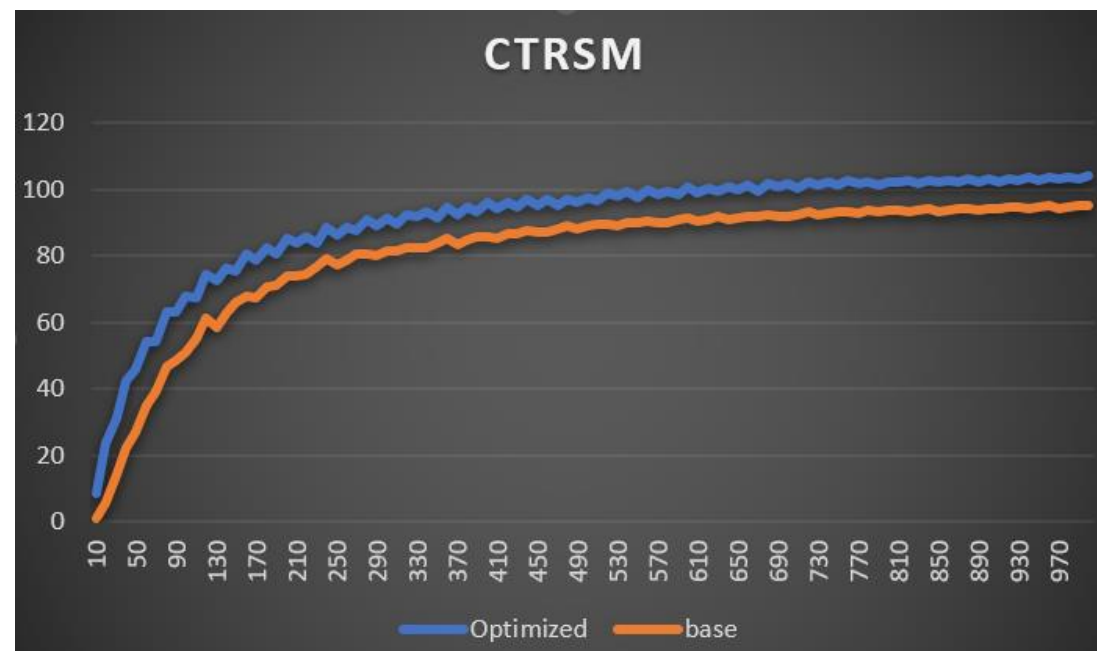
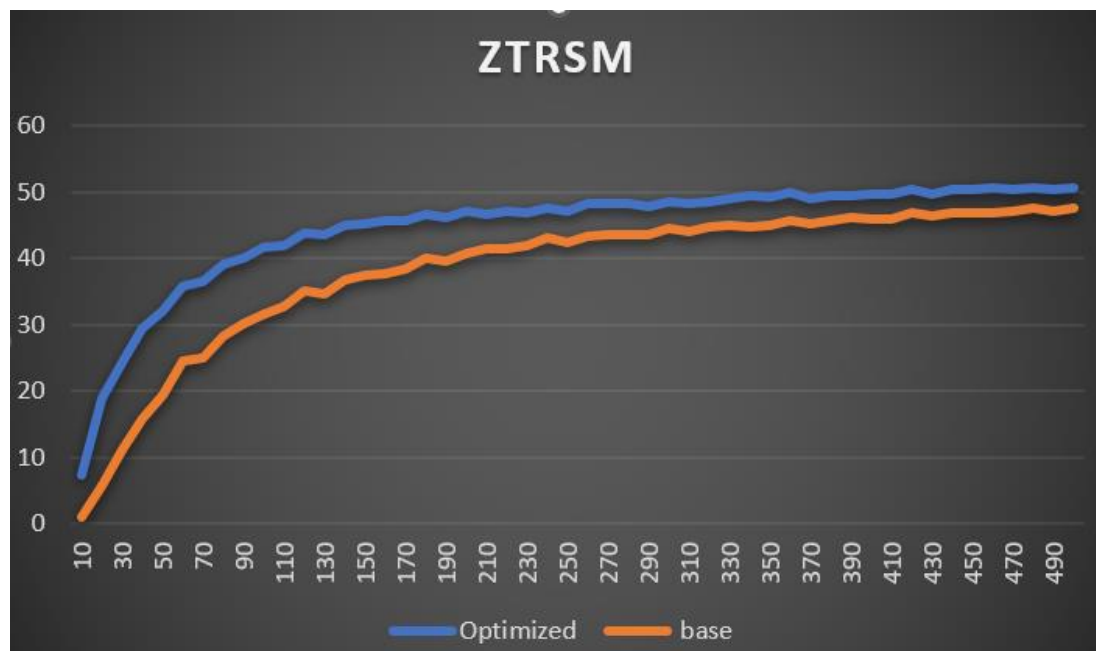
Gains observed in LLNN variant (AX=B) with Prefetch



DTRSM/STRSM Performance Report



ZTRSM/CTRSM Performance Report



Code Path

Code Link: https://github.com/amd/blis/blob/master/kernels/zen/3/bli_trsm_small.c

Thank you

Questions and suggestions?



COPYRIGHT AND DISCLAIMER

©2022 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.

