Empowering Ozaki Scheme with Hopper Architecture

Tao Wang
Graduate School of Engineering, The University of Tokyo

Takashi Shimokawabe
Information Technology Center, The University of Tokyo

I. Overview

Integer Ozaki Scheme utilizes **low-precision INT8 tensor core** (TC), a feature supported since Turing architecture, to reduce the running time of a large quantity of gemm kernels. Also, **each gemm kernel is followed by one accumulation (accum) kernel** to accumulate the gemm result into the final high-precision matrix, which leads to **extra memory footprint and kernel launch overhead**. However, previous implementations rely on Cublas library, which hides kernel details and disables implementation flexibility.

Compared to Matrix Multiply-Accumulate (MMA) instruction used in previous architectures, Hopper GPUs introduce new features **Warpgroup Matrix Multiply-Accumulate (WGMMA)** and **tensor memory accelerator (TMA),** which are favorable to implementations, i.e. warp specialization and persistent kernel, creating new possibilities for high-performance kernel.

This paper investigates how new features of Hopper architecture will affect INT8 gemm performance, utilizes them to **implement gemm accum-fusion integer Ozaki Scheme** and **compares it with Cublas for double-precision gemm (DGEMM).**

II. Ozaki Scheme

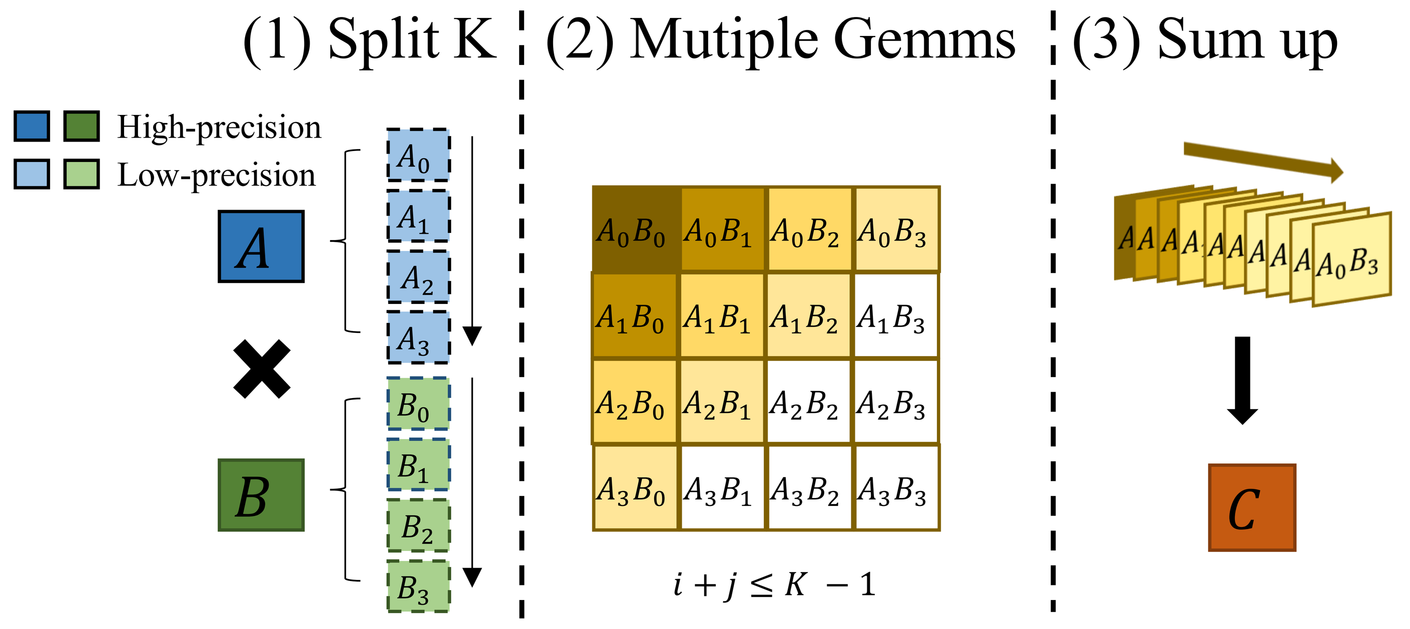


Fig.1 Diagram of Ozaki Scheme

III. Hopper Architecture

Warpgroup Matrix Multiply-Accumulate (wgmma):

1. Async operation → flexible control
2. Support larger tile computation → faster than *mma.sync*

Tensor Memory Accelerator (TMA):

1. PTX instruction: *cp.async.bulk.tensor, cp.reduce.async.bulk.tensor* and *cuTensorMap*
2. Async copy: *gmem* ↔ *smem*
3. Support multicast (one copy to multi-CTA in a cluster), store reduction
4. Less address computation and data transfer instruction → faster than *cp.async*

Warp Specialization:

1. decoupled warp role: producer and consumer → more flexible than multi-stage
2. regs allocation (*setmaxnreg*): more regs for consumer

Persistent Kernel:

1. All the CTAs in one wave → reduce CTA launch overhead; hide prologue and epilogue

IV. Methods

INT8 GEMM

1. Explore how INT8 gemm compute speed is affected by three new features: wgmma, TMA and warp specialization
2. Compare to cublas lib

Ozaki Scheme

1. Apply Hopper-empowered INT8 gemm to INT8 ozaki scheme
2. Fuse accum into gemm to reduce memory consumption and kernel launch overhead
3. Compare multiple INT8 ozaki scheme implementations on compute speed and memory consumption

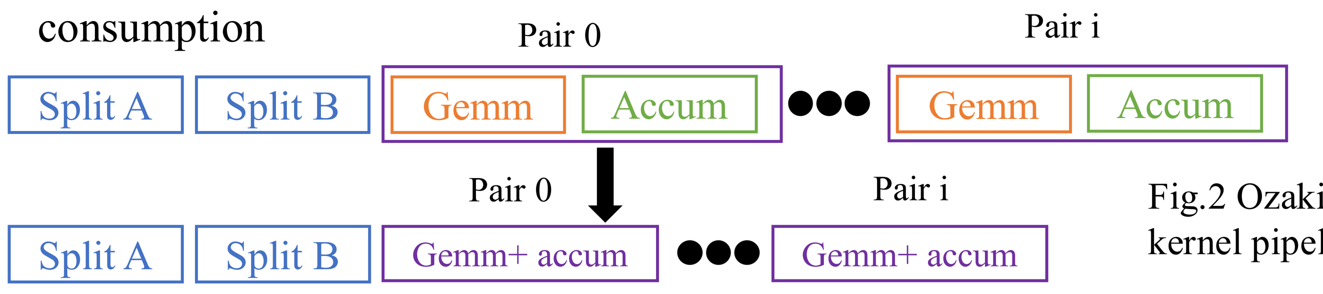


Fig.2 Ozaki scheme kernel pipeline

V. Experiment Setup

Machine

1. Grace cpu + H200 (GH200), cuda12.8, compute capability 9.0

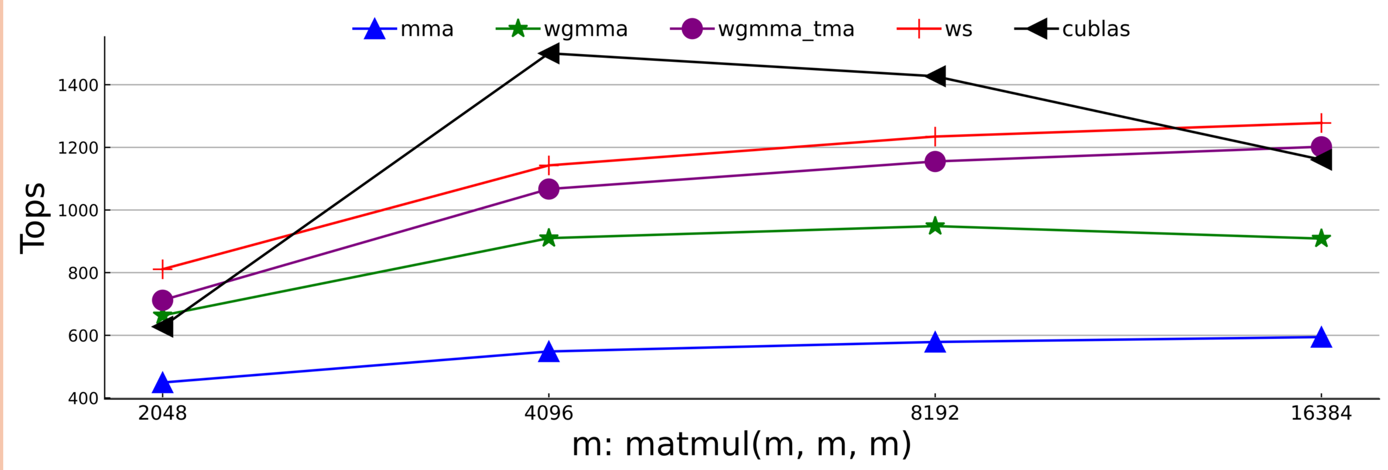
INT8 GEMM

1. **mma**: cute/cutlass; multi-stage(cp.async) + mma
2. **wgmma/wgmma_tma**: cute/cutlass; multi-stage(cp.async/TMA) + wgmma
3. **ws:** cute/cutlass; warp specialization + tma + wgmma, modified from FP16 gemm [3]
4. **cublas**: cublas lib

Ozaki Scheme

1. **mma**: mma from INT8 gemm section
2. **wgmma_tma**: wgmma_tma from INT8 gemm section
3. **ws_accum_fuse**: ws from INT8 gemm section + gemm-accum fusion
4. **ozaki_cublas_int8**: cublas from INT8 gemm section
5. **oziMMU**: code from [1], cublas gemm
6. **cuBLASDX**: mathDX lib

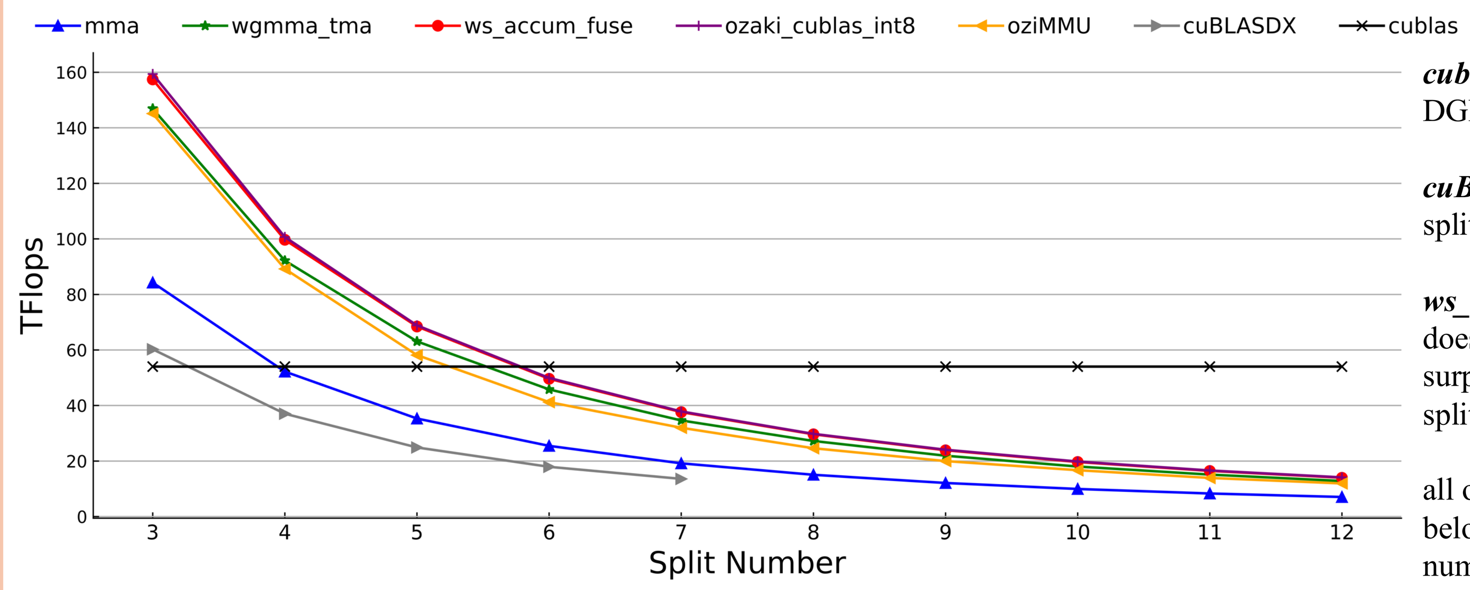
VI. Results on Compute Speed



Wgmma 👍
TMA 👍
Warp specialization 👍

ws wins in some cases over *cublas* when not specifically tuned!

Fig.3 Results of profiling INT8 MM kernels under different implementations



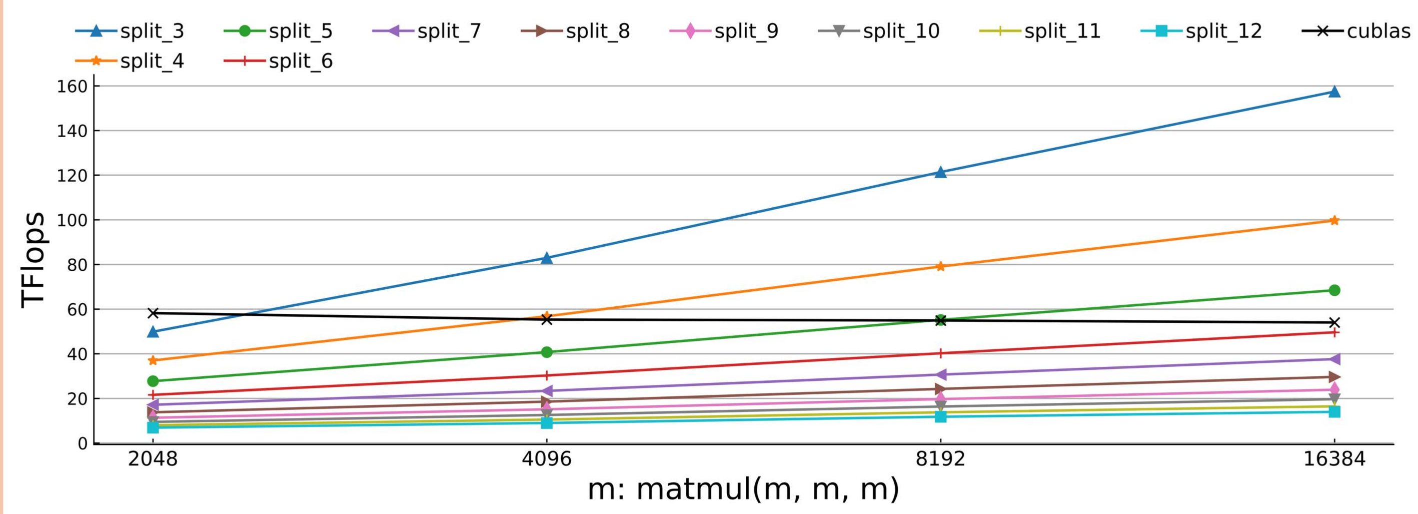
cublas denotes DGEMM from **cublas**

cuBLASDX OOM after split number 7…

ws_accum_fuse speed does not drop or even surpasses others when split number is small👍

all ozaki schemes drops below ***cublas*** after split number 6🤔

Fig.4 Results of profiling Ozaki Scheme under different implementations and Cublas for DGEMM. The input size is 16384 × 16384 × 16384



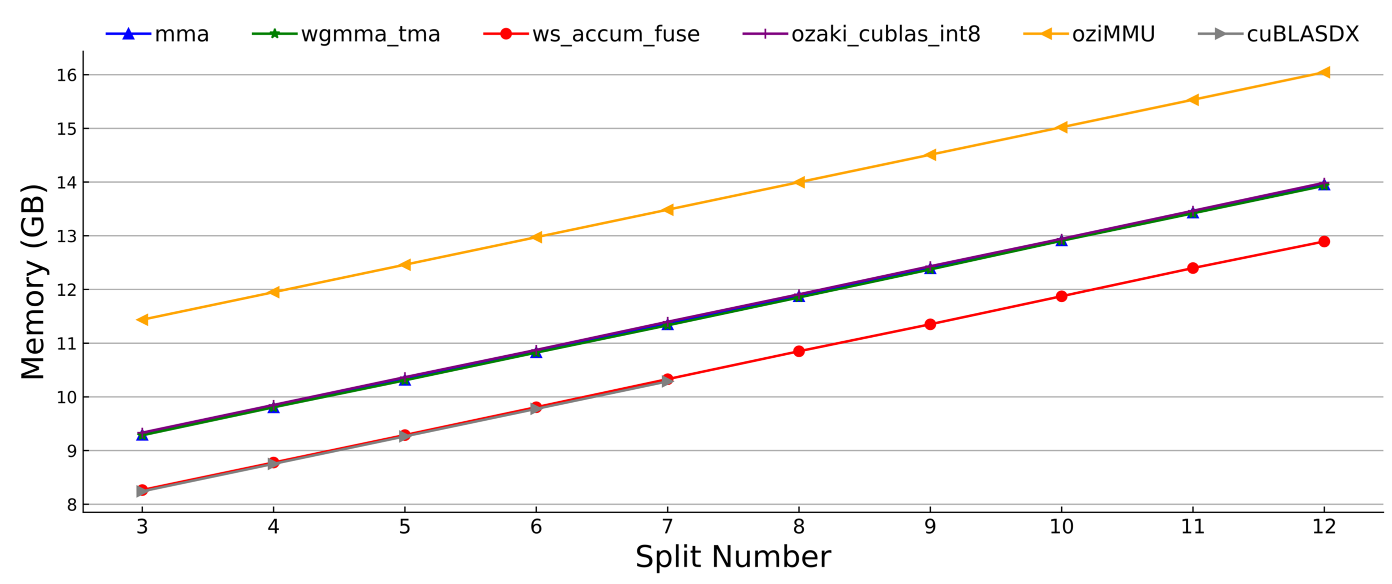
ws_accum_fuse is chosen as a representative here

Tradeoff between speed and split number (precision)

[1] shows split number ≥ 11 can steadily obtain better precision than cublas DGEMM across various exp range

Fig.5 Results of profiling ws_accum_fuse for different split numbers

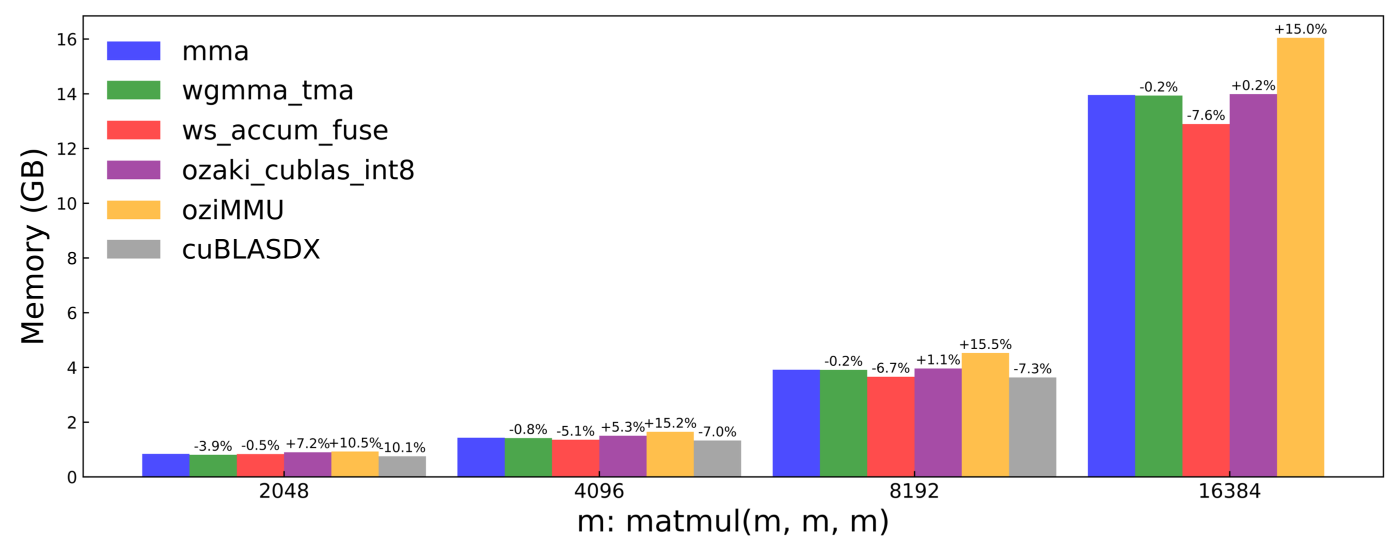
VII. Results on Memory Consumption



ws_accum_fuse enjoys smallest GPU memory footprint👍

cuBLASDX OOM after split number 7 when there is enough GPU…

Fig.6 GPU memory consumption of Ozaki Scheme under different implementations. The input size is 16384 × 16384 × 16384



Saved memory overhead are linear to product of m and n, so the benefit becomes more apparent as the input size increases.

Split number 12 is chosen because this is split number which can steadily towers over cublas on precision across various exp ranges

Fig.7 Memory consumption of Ozaki Scheme under different implementations for different input sizes. The split number is set to 12 here.

VIII Acknowledgements

This work was partially supported by JST SPRING Grant Number JPMJSP2108, JSPS KAKENHI Grant Number JP24K02947 and JHPCN project jh250037.

IX. Reference

[1] H. Ootomo et al., Dgemm on integer matrix multiplication unit. The International Journal of High Performance Computing Applications, 38 (4):297–313, 2024.
[2] K. Ozaki et al.. Error-free transformations of matrix multiplication by using fast routines of matrix multiplication and its applications. Numer. Algorithms, 59(1):95–118, Jan. 2012.
[3] https://github.com/CalebDu/Awesome-Cute