I. BACKGROUNDS: CUTLASS DETAILS

A. CUTLASS Tile Parameters

Figure 1 illustrates the relationship between the tiling parameters and GPU resources. In a detailed GEMM view, from the perspective of GEMM, two chunks of sizes $[T_m \times T_k]$, $[T_k \times T_n]$ are loaded from the global memory for a total of *gemm_it_k* times for each TBT. Next, from the perspective of threadblock, the chunks loaded from the global memory are stored in the shared memory. Since the CUTLASS GEMM operation uses software pipelining with a pipeline depth of 2, the size of data allocated to shared memory for a single TBT is doubled for each chunk. The size of data loaded from global memory to shared memory at a time for a single TBT is as follows.

$$TBTData_{G\to S} = T_m \times T_k + T_k \times T_n$$

Then, from the perspective of warp, GEMM elements of size W_m and W_n are loaded into registers for a total of *warp_mma_k* times for each WT. The amount of data loaded from shared memory to register at a time for a single warp is as follows.

$$WTData_{S\to R} = W_m + W_n$$

Finally, from the perspective of thread, each thread performs an outer-product-based GEMM operation using the candidate warp fragments (Th_m, Th_n fragments in Figure 1). The shape of the GEMM that each thread performs is determined by dividing the shape of the WT by the shape of the SIMD lane shape (L_m, L_n in Figure 3). As a result, the size of the GEMM performed in a single threadblock is as follows, where *ThPerTB* refers to the number of threads per threadblock.

$$Comp_{Core} = Th_m \times Th_n \times ThPerTB$$

B. Software Pipelining of CUTLASS GEMM

Figure 2 shows a simplified workflow of a CUTLASS GEMM operation using software pipelining for a single TBT. According to Figure 2, a tiled GEMM is performed in the order of *Prologue*, *Mainloop*, and then *Epilogue* for each TBT.

In the *Prologue* stage, the data of matrices A and B are loaded through the GLobal Stream (GLS) for the GEMM of the first iteration of the *Mainloop*, and then stored in shared memory. Next, in the *Mainloop* stage, three operations are performed: 1. loading the data of matrices A and B through the GLS for the next *Mainloop* iteration 2. loading the data stored in the shared memory through the Shared Load Stream (SLS) 3. performing GEMM operations with computing cores within the Computation Stream (CS)

Operations in the *Mainloop* stage can be overlapped with each other, and therefore, the latency of the *Mainloop* is determined by the stream that takes the longest time to finish. Based on this software pipelining concept, the latency of each stream (LAT_{Stage}) in the *Mainloop* is as follows, where BW_{res} , CNT_{SM} , and CNT_{WT} refer to each resource bandwidth, number of SMs, and number of WTs, respectively.

$$LAT_{GLS} = \frac{TBTData_{G\to S}}{BW_{DRAM}/CNT_{SM}}$$
$$LAT_{SLS} = \frac{WTData_{S\to R} \times warp\_mma\_k \times CNT_{WT}}{BW_{SMEM}}$$
$$LAT_{CS} = \frac{Comp_{Core} \times warp\_mma\_k}{BW_{Core}}$$

Finally, in the *Epilogue* stage, after all the *Mainloop*s are finished, the results stored in the register are transferred to the global memory through the GLS [1].

C. Split-K Optimization of CUTLASS GEMM

Figure 3 shows the overview of the Split-K parallel optimization of CUTLASS GEMM. The term "Split-K" refers dividing the GEMM operation based on the K dimension to allocate more TBTs into GPU SMs. When applying the Split-K parallel optimization, the target GEMM is divided into K sub-GEMMs, then sub-GEMMs are performed in parallel. After the sub-GEMMs are finished, the final result is generated through the reduction operation. The Split-K parallel optimization increases the number of TBTs by K times, so that the SM occupancy can be improved when SMs are not fully utilized. In general, the Split-K parallel optimization helps when the K dimension of the target GEMM is much larger than the M and N dimensions.

II. MOTIVATION

A. Building an Analytical Performance Model for CUTLASS GEMM

As an another method other than using general trends, to determine the appropriate TBT and WT parameters for the CUTLASS GEMM, we also considered a performance model based on the software pipelining method of the CUTLASS GEMM. Based on discussions above, latencies (LAT_{Stage}) of the software pipelined CUTLASS GEMM are defined as follows:

$$LAT_{Prol} = LAT_{GLS} + \frac{LAT_{SLS}}{warp\_mma\_k}$$
$$LAT_{Main} = max(LAT_{GLS}, LAT_{SLS}, LAT_{CS})$$
$$LAT_{Epil} = \frac{T_M \times T_N}{BW_{DRAM}}$$

According to our analytical model, we referenced to an open-source microbenchmarking tool [3] to compute several data bandwidth at various memory levels and check the commodity device specification. Especially, we considered the relationship between latencies of each pipelining stream when several number of active threadblocks are interleaved in the *MainLoop* stage [2] [1]. Note that we also considered the split-K parallel technique of CUTLASS GEMM by dividing the K dimension of GEMM by split-K and multiplying the M or N

[1]Because we don't consider *add_bias* or *activation* operations following the GEMM, we regard the *Epilogue* as the data copy operation of the GEMM result from register to global memory.

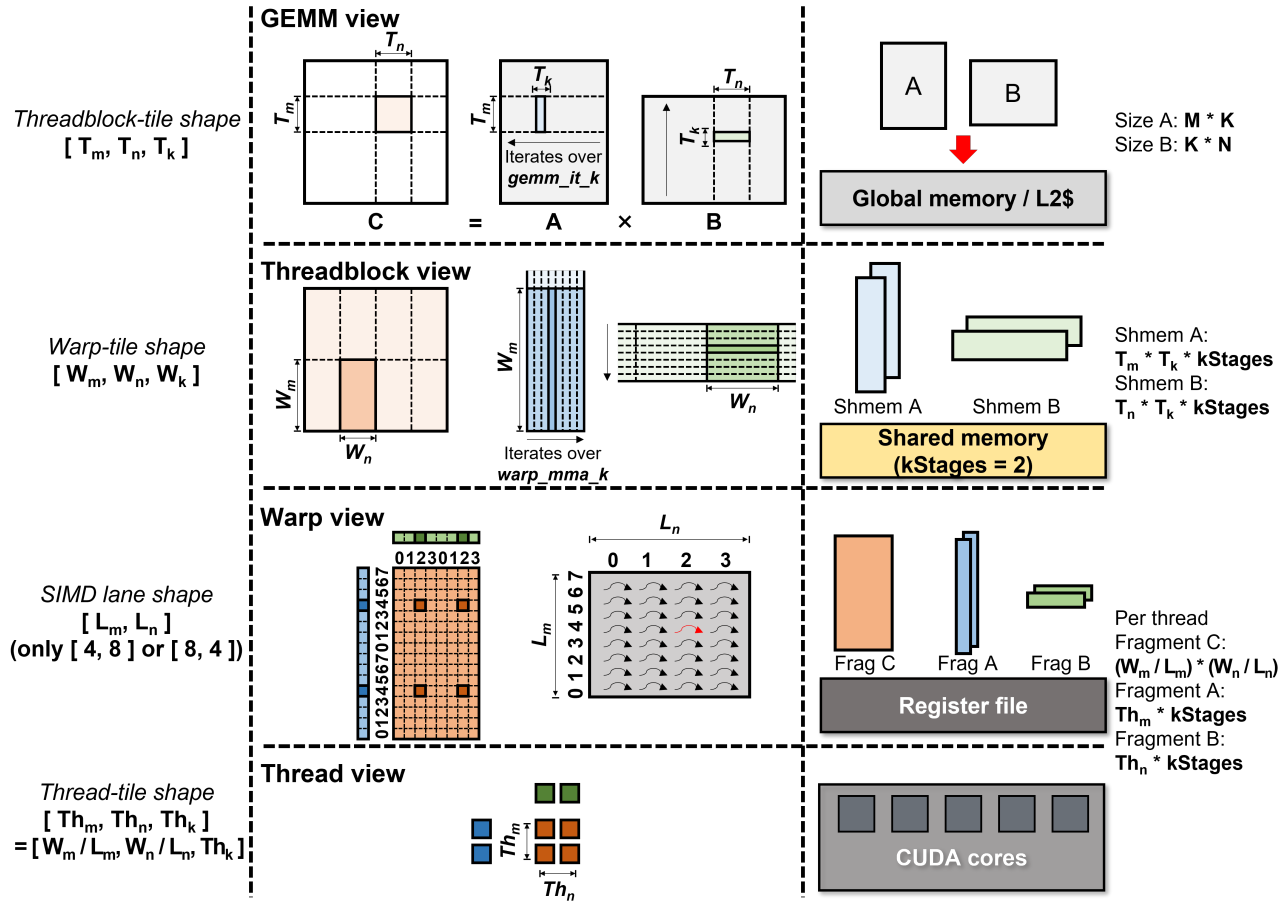

Fig. 1. A detailed overview of the CUTLASS tile hierarchy.

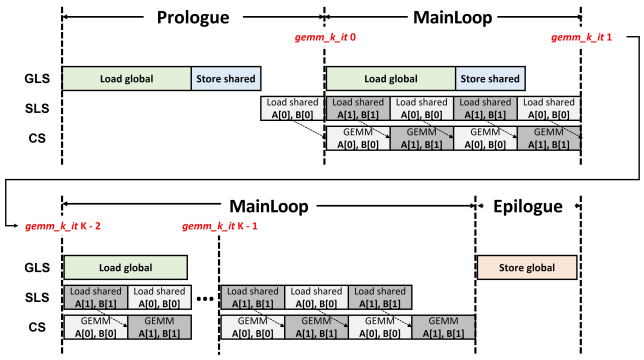

Fig. 2. An overview of a software pipelined CUTLASS GEMM operation for a single TBT (ThreadBlock Tile).

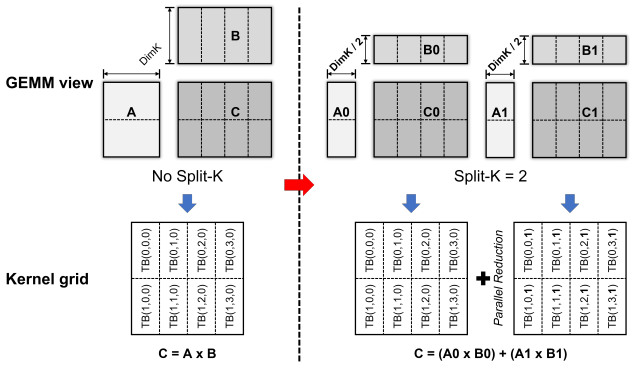

Fig. 3. An overview of the CUTLASS Split-K parallel optimization.

dimension by split-K as it seems like a batched GEMM. After merging considerations such as the number of allocable TBTs and the effect of various memory bandwidth, we derive the final latency for the GEMM as follows:

$$LAT_{GEMM} = LAT_{Prol} + LAT_{Main} \times gemm\_it\_k + LAT_{Epil}$$

Based on the formula above, the prediction model we constructed calculates latencies for all available tile parameters for the target GPU and GEMM, and then find the tile parameter that shows the lowest latency among them.

REFERENCES

[1] J. Huang, C. D. Yu, and R. A. v. d. Geijn. Strassen's algorithm reloaded on gpus. *ACM Trans. Math. Softw.*, 46(1), mar 2020.

[2] S. Lym, D. Lee, M. O'Connor, N. Chatterjee, and M. Erez. Delta: Gpu performance model for deep learning applications with in-depth memory system traffic analysis. In *2019 IEEE International Symposium*

on Performance Analysis of Systems and Software (ISPASS), pages 293–303, Los Alamitos, CA, USA, mar 2019.

[3] X. Mei and X. Chu. Dissecting gpu memory hierarchy through microbenchmarking. *IEEE Transactions on Parallel and Distributed Systems*, 28(1):72–86, 2017.