POSTER: Stream-K: Work-centric Parallel Decomposition for Dense Matrix-Matrix Multiplication on the GPU

Muhammad Osama[†], Duane Merrill[‡], Cris Cecka[‡], Michael Garland[‡], and John D. Owens[†]
mosama@ucdavis.edu,dumerrill@nvidia.com,ccecka@nvidia.com,mgarland@nvidia.com,jowens@ucdavis.edu
[†]University of California, Davis and [‡]NVIDIA Corporation

Abstract

We introduce *Stream-K*, a work-centric parallelization of matrix multiplication (GEMM) and related computations in dense linear algebra. Whereas contemporary decompositions are primarily tile-based, our method operates by partitioning an even share of the aggregate inner loop iterations among physical processing elements. This provides a near-perfect utilization of computing resources, regardless of how efficiently the output tiling for any given problem quantizes across the underlying processing elements.

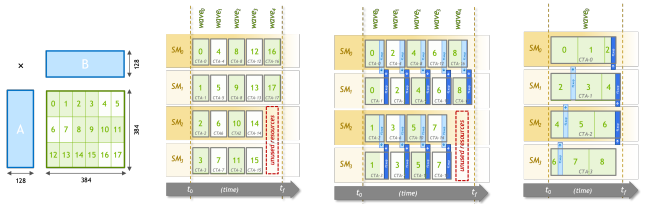
CCS Concepts: • **Computing methodologies** → **Parallel algorithms**.

Keywords: Matrix-Multiplication, GPU, Load-Balancing

1 Introduction

General matrix-matrix product (GEMM), convolution, and other similar computations constitute the dominant workloads in many deep learning and scientific computing applications. High-performance processors such as GPUs, for example, are designed to achieve nearly 100% of their theoretical peak math throughput when computing GEMM. Doing so, however, requires a work decomposition that perfectly occupies the underlying physical cores. Attaining such high levels of processor utilization across a broad landscape of problems shapes and sizes can be challenging.

Classically, GEMM implementations block their computation using a *data-parallel* tiling of the output matrix, assigning the independent production of output tiles among concurrent threads (or thread groups) [1–3]. A variant of *data-parallel* is known as *fixed-split*, where a given output tile is split with a constant splitting factor to better utilize the device. The work per output tile is regular, and tile production tends to dispatch across idle physical cores in "waves".



(a) *Data parallel* **(b)** *Fixed-split* **(c)** *Stream-K*

Figure 1. *Stream-K* vs. contemporary parallel decompositions across a hypothetical four-SM GPU. *Stream-K* achieves 100% GPU utilization.

However, such oversubscription has shrunk considerably as processors have grown in size. An increased core count will require fewer waves to produce a given tile count. Bigger cores will compel larger matrix blocking factors, leading to fewer waves of larger tiles. In general, execution schedules with fewer waves are much more likely to suffer from *quantization inefficiency*, i.e., the processor underutilization that occurs when the number of output tiles is not an even multiple of the number of processor cores. When the last wave is partially full, the unused cores must wait for the remaining threads to execute millions (if not billions) of multiply-accumulate (MAC) instructions before they are able to execute any dependent work.

2 Our *Stream-K* Decomposition

Our *Stream-K* decomposition addresses quantization inefficiency using a tile-splitting parallelization in which splitting seams are completely dissociated from the tiling structure itself. Although we employ familiar blocking and tiling strategies for data reuse, we instead quantize the GEMM computation into MAC-loop iterations, i.e., small volumes of CTA-wide BLK_M × BLK_N × BLK_K work. As presented in Figure 1, *Stream-K* evenly partitions the GEMM's aggregate workload of MAC-loop iterations across a constant-sized grid of g CTAs. Each CTA's range of MAC-loop iterations is mapped contiguously into the $m \rightarrow n \rightarrow k$ linearization of the GEMM shape, crossing output-tile boundaries as it may.

PPoPP '23, February 25–March 1, 2023, Montreal, QC, Canada

ACM ISBN 979-8-4007-0015-6/23/02.
https://doi.org/10.1145/3572848.3577479

	vs. CUTLASS $128 \times 128 \times 32$	vs. cuBLAS	vs. cuBLAS > 150 ops/B	vs. CUTLASS oracle
Average	1.63×	1.13×	1.15×	1.12×
StdDev	1.46	0.45	0.12	0.37
Min	0.80×	0.64×	0.98×	0.61×
Max	14.7×	6.74×	1.85×	4.63×

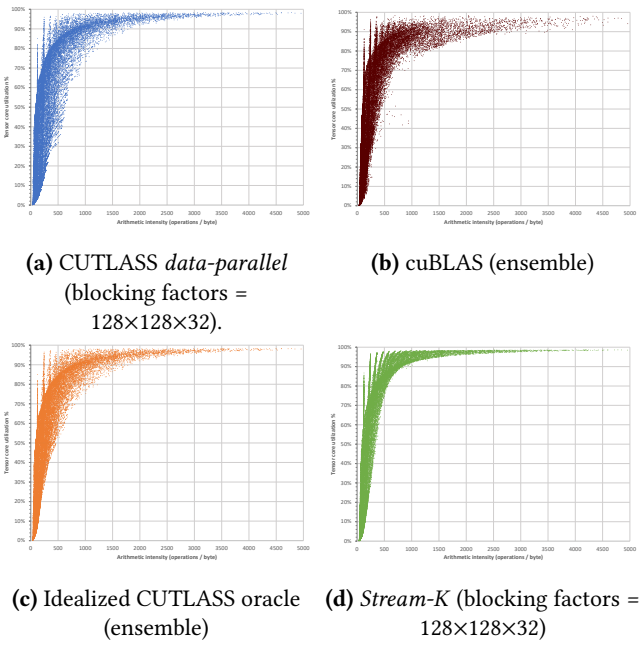
Table 1. *Stream-K* Relative Performance

Should a given CTA's starting and/or ending iterations not coincide with tile boundaries (as is expected to be the common case), it must consolidate its partial results with those of the other CTA(s) also covering that tile. In this basic implementation, each output tile in **C** is written by the CTA that performed that tile's $k = 0$ MAC-loop iteration. Before it can do so, however, it must accumulate any partial sums shared from other CTAs in temporary global storage. Notably, *Stream-K*'s communication, synchronization, and global storage overheads are independent of problem size, scaling instead with g, the number of CTAs. A secondary benefit of *Stream-K* is that synchronization-waiting is likely negligible when the number of output tiles is greater than the number of CTAs. In this regime, each output tile is covered by at most two CTAs, and the tile-processing skew ensures that the accumulating CTA will not need its peer contributions until well after those collaborators have finished producing them.

Additionally, the work volume of a single MAC-loop iteration is 32× smaller than that of an entire output tile. Consequently, a 32-way *fixed-split* decomposition would also provide a 100% quantization efficiency, but at the expense of an 8× larger "fixup" overhead. Furthermore, *Stream-K* is better able to hide the latency of inter-CTA synchronization due to the temporal skew between writers and readers when sharing partial sums. *Stream-K* also generalizes to both contemporary *fixed-split* and *data-parallel* decompositions. When the grid size g is an even multiple of the number of output tiles, *Stream-K* functions exactly as the *fixed-split* decomposition. Similarly, when g equals the number of output tiles, *Stream-K* behaves identically to the *data-parallel* decomposition.

3 Performance Evaluation

The "roofline" plot of Figure 2a-2c highlight the spread of performance produced by the CUTLASS and cuBLAS kernels. They plot the percentage of GEMM processor utilization as a function of computational intensity. Ideally, a GEMM implementation's performance response would manifest as a narrow band that adheres tightly to the machine's bandwidth- and compute-bound performance ceilings. Here, the *data-parallel* and *fixed-split* kernels exhibit a fairly large dynamic range for any given regime of arithmetic intensity. In contrast, the performance responses from the equivalent *Stream-K* kernels in Figure 2d are much tighter.



(a) CUTLASS *data-parallel* (blocking factors = 128×128×32).

(b) cuBLAS (ensemble)

(c) Idealized CUTLASS oracle (ensemble)

(d) *Stream-K* (blocking factors = 128×128×32)

Figure 2. GEMM "roofline" performance utilization landscapes on NVIDIA A100 across 32K GEMM problems.

These observations are corroborated by Table 1, which show the *Stream-K* kernels outperforming their *data-parallel* equivalents by an average of 1.63×. For extreme strong-scaling scenarios where $m \times n$ is small and k is large, our *Stream-K* kernels demonstrate up to 14.7× speedup. This is a significant improvement over the breadth of 32K GEMM problem shapes and sizes with 20× *less* executable code (a single kernel for each precision) than NVIDIA's vendor GEMM library, cuBLAS.

Stream-K is open-sourced within CUTLASS 2.11 [2] and the performance shown within this paper can be reproduced when compiled using CUDA 11.8 [4].

Acknowledgments

This material is based upon work supported by Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-18-3-0007. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the U.S. Government. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited). We would like to acknowledge Louis Feng, Valentin Andrei, Zhongyi Lin and Serban D. Porumbescu for their feedback on early drafts of the paper.

References

[1] Ahmad Abdelfattah, David Keyes, and Hatem Ltaief. 2016. KBLAS: An Optimized Library for Dense Matrix-Vector Multiplication on GPU Accelerators. *ACM Trans. Math. Software* 42, 3 (June 2016), 1–31. https://doi.org/10.1145/2818311

[2] Andrew Kerr, Duane Merrill, Julien Demouth, and John Tran. 2017. CUTLASS: Fast Linear Algebra in CUDA C++. (2017). https://devblogs.nvidia.com/cutlass-linear-algebra-cuda/

[3] Rajib Nath, Stanimire Tomov, and Jack Dongarra. 2010. An Improved Magma Gemm For Fermi Graphics Processing Units. *The International Journal of High Performance Computing Applications* 24, 4 (Nov. 2010), 511–515. https://doi.org/10.1177/1094342010385729

[4] NVIDIA Corporation. 2007–2022. CUDA C++ Programming Guide. (Dec. 2007–2022). https://docs.nvidia.com/cuda/ PG-02829-001_v12.0.