Tuning CUDA Applications for Ampere

Application Note

DA-09073-001_v12.0 | December 2022

Table of Contents

Chapter 1. NVIDIA Ampere GPU Architecture Tuning Guide

1.1. NVIDIA Ampere GPU Architecture

The NVIDIA Ampere GPU architecture is NVIDIA's latest architecture for CUDA compute applications. The NVIDIA Ampere GPU architecture retains and extends the same CUDA programming model provided by previous NVIDIA GPU architectures such as Turing and Volta, and applications that follow the best practices for those architectures should typically see speedups on the NVIDIA A100 GPU without any code changes. This guide summarizes the ways that an application can be fine-tuned to gain additional speedups by leveraging the NVIDIA Ampere GPU architecture's features.[1]

For further details on the programming features discussed in this guide, please refer to the CUDA C++ Programming Guide.

1.2. CUDA Best Practices

The performance guidelines and best practices described in the CUDA C++ Programming Guide and the CUDA C++ Best Practices Guide apply to all CUDA-capable GPU architectures. Programmers must primarily focus on following those recommendations to achieve the best performance.

The high-priority recommendations from those guides are as follows:

- Find ways to parallelize sequential code.
- Minimize data transfers between the host and the device.
- Adjust kernel launch configuration to maximize device utilization.
- Ensure global memory accesses are coalesced.
- Minimize redundant accesses to global memory whenever possible.

[1] Throughout this guide, *Maxwell* refers to devices of compute capability 5.x, *Pascal* refers to device of compute capability 6.x, *Volta* refers to devices of compute capability 7.0, *Turing* refers to devices of compute capability 7.5, and *NVIDIA Ampere GPU Architecture* refers to devices of compute capability 8.x

- Avoid long sequences of diverged execution by threads within the same warp.

1.3. Application Compatibility

Before addressing specific performance tuning issues covered in this guide, refer to the NVIDIA Ampere GPU Architecture Compatibility Guide for CUDA Applications to ensure that your application is compiled in a way that is compatible with the NVIDIA Ampere GPU Architecture.

1.4. NVIDIA Ampere GPU Architecture Tuning

1.4.1. Streaming Multiprocessor

The NVIDIA Ampere GPU architecture's Streaming Multiprocessor (SM) provides the following improvements over Volta and Turing.

1.4.1.1. Occupancy

The maximum number of concurrent warps per SM remains the same as in Volta (i.e., 64), and other factors influencing warp occupancy are:

- The register file size is 64K 32-bit registers per SM.
- The maximum number of registers per thread is 255.
- The maximum number of thread blocks per SM is 32 for devices of compute capability 8.0 (i.e., A100 GPUs) and 16 for GPUs with compute capability 8.6.
- For devices of compute capability 8.0 (i.e., A100 GPUs) shared memory capacity per SM is 164 KB, a 71% increase compared to V100's capacity of 96 KB. For GPUs with compute capability 8.6, shared memory capacity per SM is 100 KB.
- For devices of compute capability 8.0 (i.e., A100 GPUs) the maximum shared memory per thread block is 163 KB. For GPUs with compute capability 8.6 maximum shared memory per thread block is 99 KB.

Overall, developers can expect similar occupancy as on Volta without changes to their application.

1.4.1.2. Asynchronous Data Copy from Global Memory to Shared Memory

The NVIDIA Ampere GPU architecture adds hardware acceleration for copying data from global memory to shared memory. These copy instructions are asynchronous, with respect to computation and allow users to explicitly control overlap of compute with data movement from

global memory into the SM. These instructions also avoid using extra registers for memory copies and can also bypass the L1 cache. This new feature is exposed via the `pipeline` API in CUDA. For more information please refer to the section on Async Copy in the CUDA C++ Programming Guide.

1.4.1.3. Hardware Acceleration for Split Arrive/Wait Barrier

The NVIDIA Ampere GPU architecture adds hardware acceleration for a split arrive/wait barrier in shared memory. These barriers can be used to implement fine grained thread controls, producer-consumer computation pipeline and divergence code patterns in CUDA. These barriers can also be used alongside the asynchronous copy. For more information on the Arrive/Wait Barriers refer to the Arrive/Wait Barrier section in the CUDA C++ Programming Guide.

1.4.1.4. Warp level support for Reduction Operations

The NVIDIA Ampere GPU architecture adds native support for warp wide reduction operations for 32-bit signed and unsigned integer operands. The warp wide reduction operations support arithmetic `add`, `min`, and `max` operations on 32-bit signed and unsigned integers and bitwise `and`, `or` and `xor` operations on 32-bit unsigned integers.

For more details on the new warp wide reduction operations refer to Warp Reduce Functions in the CUDA C++ Programming Guide.

1.4.1.5. Improved Tensor Core Operations

The NVIDIA Ampere GPU architecture includes new Third Generation Tensor Cores that are more powerful than the Tensor Cores used in Volta and Turing SMs. The new Tensor Cores use a larger base matrix size and add powerful new math modes including:

- Support for FP64 Tensor Core, using new DMMA instructions.
- Support for Bfloat16 Tensor Core, through HMMA instructions. BFloat16 format is especially effective for DL training scenarios. Bfloat16 provides 8-bit exponent i.e., same range as FP32, 7-bit mantissa and 1 sign-bit.
- Support for TF32 Tensor Core, through HMMA instructions. TF32 is a new 19-bit Tensor Core format that can be easily integrated into programs for more accurate DL training than 16-bit HMMA formats. TF32 provides 8-bit exponent, 10-bit mantissa and 1 sign-bit.
- Support for bitwise `AND` along with bitwise `XOR` which was introduced in Turing, through BMMA instructions.

The following table presents the evolution of matrix instruction sizes and supported data types for Tensor Cores across different GPU architecture generations.

Instruction	GPU Architecture	Input Matrix format	Output Accumulator format	Matrix Instruction Size (MxNxK)
HMMA (16-bit precision)	NVIDIA Volta Architecture	FP16	FP16 / FP32	8x8x4
	NVIDIA Turing Architecture	FP16	FP16 / FP32	8x8x4 / 16x8x8 / 16x8x16
	NVIDIA Ampere Architecture	FP16 / BFloat16	FP16 / FP32 (BFloat16 only supports FP32 as accumulator)	16x8x8 / 16x8x16
HMMA (19-bit precision)	NVIDIA Volta Architecture	N/A	N/A	N/A
	NVIDIA Turing Architecture	N/A	N/A	N/A
	NVIDIA Ampere Architecture	TF32 (19-bits)	FP32	16x8x4
IMMA (Integer MMA)	NVIDIA Volta Architecture	N/A	N/A	N/A
	NVIDIA Turing Architecture	unsigned char/ signed char (8-bit precision)	int32	8x8x16
	NVIDIA Ampere Architecture	unsigned char/ signed char (8-bit precision)	int32	8x8x16 / 16x8x16 / 16x8x32
IMMA (Integer sub-byte MMA)	NVIDIA Volta Architecture	N/A	N/A	N/A
	NVIDIA Turing Architecture	unsigned u4/ signed u4 (4-bit precision)	int32	8x8x32
	NVIDIA Ampere Architecture	unsigned u4/ signed u4 (4-bit precision)	int32	8x8x32 / 16x8x32 / 16x8x64
BMMA (Binary MMA)	NVIDIA Volta Architecture	N/A	N/A	N/A
	NVIDIA Turing Architecture	single bit	int32	8x8x128
	NVIDIA Ampere Architecture	single bit	int32	8x8x128 / 16x8x128 / 16x8x256
DMMA (64-bit precision)	NVIDIA Volta Architecture	N/A	N/A	N/A
	NVIDIA Turing Architecture	N/A	N/A	N/A

Instruction	GPU Architecture	Input Matrix format	Output Accumulator format	Matrix Instruction Size (MxNxK)
	NVIDIA Ampere Architecture	FP64	FP64	8x8x4

For more details on the new Tensor Core operations refer to the Warp Matrix Multiply section in the CUDA C++ Programming Guide.

1.4.1.6. Improved FP32 throughput

Devices of compute capability 8.6 have 2x more FP32 operations per cycle per SM than devices of compute capability 8.0. While a binary compiled for 8.0 will run as is on 8.6, it is recommended to compile explicitly for 8.6 to benefit from the increased FP32 throughput.

1.4.2. Memory System

1.4.2.1. Increased Memory Capacity and High Bandwidth Memory

The NVIDIA A100 GPU increases the HBM2 memory capacity from 32 GB in V100 GPU to 40 GB in A100 GPU. Along with the increased memory capacity, the bandwidth is increased by 72%, from 900 GB/s on Volta V100 to 1550 GB/s on A100.

1.4.2.2. Increased L2 capacity and L2 Residency Controls

The NVIDIA Ampere GPU architecture increases the capacity of the L2 cache to 40 MB in Tesla A100, which is 7x larger than Tesla V100. Along with the increased capacity, the bandwidth of the L2 cache to the SMs is also increased. The NVIDIA Ampere GPU architecture allows CUDA users to control the persistence of data in L2 cache. For more information on the persistence of data in L2 cache, refer to the section on managing L2 cache in the CUDA C++ Programming Guide.

1.4.2.3. Unified Shared Memory/L1/Texture Cache

The NVIDIA A100 GPU based on compute capability 8.0 increases the maximum capacity of the combined L1 cache, texture cache and shared memory to 192 KB, 50% larger than the L1 cache in NVIDIA V100 GPU. The combined L1 cache capacity for GPUs with compute capability 8.6 is 128 KB.

In the NVIDIA Ampere GPU architecture, the portion of the L1 cache dedicated to shared memory (known as the *carveout*) can be selected at runtime as in previous architectures such as Volta, using `cudaFuncSetAttribute()` with the attribute `cudaFuncAttributePreferredSharedMemoryCarveout`. The NVIDIA A100 GPU supports shared memory capacity of 0, 8, 16, 32, 64, 100, 132 or 164 KB per SM. GPUs with compute capability 8.6 support shared memory capacity of 0, 8, 16, 32, 64 or 100 KB per SM.

CUDA reserves 1 KB of shared memory per thread block. Hence, the A100 GPU enables a single thread block to address up to 163 KB of shared memory and GPUs with compute

capability 8.6 can address up to 99 KB of shared memory in a single thread block. To maintain architectural compatibility, static shared memory allocations remain limited to 48 KB, and an explicit opt-in is also required to enable dynamic allocations above this limit. See the CUDA C++ Programming Guide for details.

Like Volta, the NVIDIA Ampere GPU architecture combines the functionality of the L1 and texture caches into a unified L1/Texture cache which acts as a coalescing buffer for memory accesses, gathering up the data requested by the threads of a warp prior to delivery of that data to the warp. Another benefit of its union with shared memory, similar to Volta L1 is improvement in terms of both latency and bandwidth.

1.4.3. Third Generation NVLink

The third generation of NVIDIA's high-speed NVLink interconnect is implemented in A100 GPUs, which significantly enhances multi-GPU scalability, performance, and reliability with more links per GPU, much faster communication bandwidth, and improved error-detection and recovery features. The third generation NVLink has the same bi-directional data rate of 50 GB/s per link, but uses half the number of signal pairs to achieve this bandwidth. Therefore, the total number of links available is increased to twelve in A100, versus six in V100, yielding 600 GB/s bidirectional bandwidth versus 300 GB/s for V100.

NVLink operates transparently within the existing CUDA model. Transfers between NVLink-connected endpoints are automatically routed through NVLink, rather than PCIe. The `cudaDeviceEnablePeerAccess()` API call remains necessary to enable direct transfers (over either PCIe or NVLink) between GPUs. The `cudaDeviceCanAccessPeer()` can be used to determine if peer access is possible between any pair of GPUs.

In the NVIDIA Ampere GPU architecture remote NVLINK accesses go through a Link TLB on the remote GPU. This Link TLB has a reach of 64 GB to the remote GPU's memory. Applications with remote random accesses may want to constrain the remotely accessed region to 64 GB for each peer GPU.

Appendix A. Revision History

Version 1.1

- Initial Public Release
- Added support for compute capability 8.6

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
www.nvidia.com