RWTH Aachen University
Department of Computer Science

Design and Performance Engineering of GPU-Accelerated Tensor Network Algorithms for Large-Scale Scientific Simulations

Master Thesis

submitted by

Daniel Sinkin
Matriculation Number: 367316

First Examiner:	Prof. Dr. TODO
Second Examiner:	Prof. Dr. TODO
Advisors:	Dr. Edoardo Di Napoli
External Institution:	Jülich Supercomputing Centre (JSC) Forschungszentrum Jülich

Aachen, February 17, 2026

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Acknowledgements

TODO

Contents

List of Figures

List of Tables

Listings

List of Symbols

$\mathcal{T}$	Tensor
$\mathbf{A}, \mathbf{B}$	Matrices
$\vec{v}$	Vector
χ	Bond dimension
d	Local (physical) dimension
N	Matrix/problem size
$\mathcal{O}(\cdot)$	Asymptotic upper bound

List of Abbreviations

BLAS	Basic Linear Algebra Subprograms
DFT	Density Functional Theory
GEMM	General Matrix Multiply
GPU	Graphics Processing Unit
HBM	High Bandwidth Memory
HPC	High Performance Computing
MPI	Message Passing Interface
MPS	Matrix Product State (tensor networks)
MPS	Multi-Process Service (Nvida CUDA)
SM	Streaming Multiprocessor
TN	Tensor Network

1. Introduction

1.1. Motivation

1.2. Problem Statement

1.3. Contributions

1.4. Outline

Chapter 2 introduces …Chapter 3 presents …Chapter 4 details …Chapter 5 evaluates …Chapter 6 summarises …

2. Background

2.1. Tensor Networks

2.1.1. Tensor Notation and Diagrams

2.1.2. Tensor Contraction

2.1.3. Tensor Network Structures

2.2. GPU Architecture

2.2.1. Streaming Multiprocessor and Warp Execution

2.2.2. Memory Hierarchy

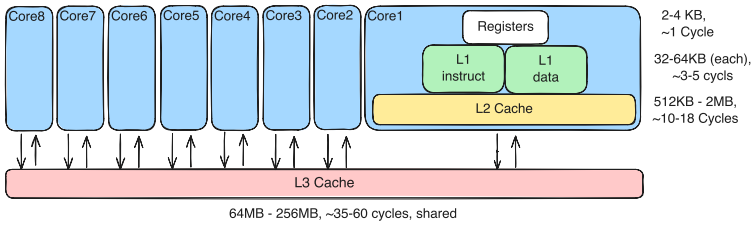

Figure 2.1.: Memory hierarchy of a modern CPU, showing the register file and cache levels (L1, L2, L3) before main memory (DRAM).

As shown in Figure 2.1, modern CPUs rely on a deep cache hierarchy to reduce effective memory latency.

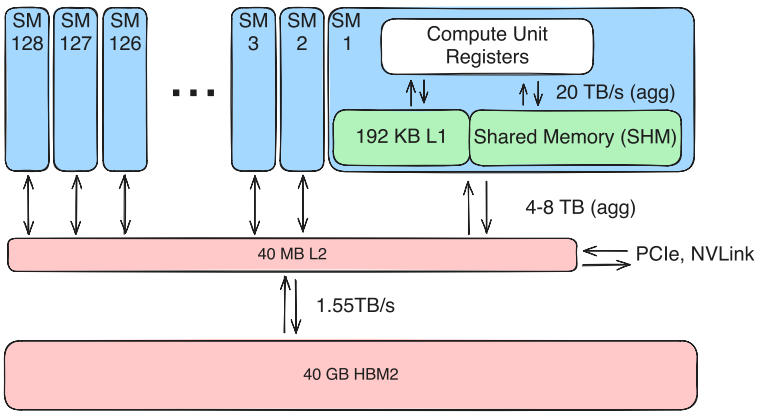

Figure 2.2.: Memory hierarchy of a modern CPU, showing the register file and cache levels (L1, L2, L3) before main memory (DRAM).

GPU Memory Hierarchy In contrast to CPUs, which rely heavily on large cache hierarchies to reduce memory latency, GPUs employ a memory hierarchy designed primarily to maximise memory bandwidth and support massive parallelism. The GPU memory hierarchy consists of several levels with different capacities, latencies, and scopes, ranging from fast on-chip storage to high-capacity off-chip main memory.

At the lowest level, each thread has access to a private register file located within the Streaming Multiprocessor (SM). Registers provide the fastest storage available on the GPU, with single-cycle access latency, and are used to hold temporary variables during computation. However, registers are a limited resource and are allocated per thread, which directly affects the maximum number of concurrent threads that can reside on an SM.

The next level is shared memory, which is also located on-chip and shared among all threads within a thread block. Shared memory has significantly lower latency than global memory and much higher bandwidth, making it suitable for explicitly managed data reuse and communication between threads. Efficient use of shared memory is essential for many high-performance GPU kernels, particularly those implementing tiled matrix multiplication and tensor contraction algorithms.

Beyond shared memory, modern GPUs include a unified L2 cache shared across all SMs. The L2 cache serves to reduce the average latency of global memory accesses and to improve effective memory bandwidth by caching frequently accessed data. Unlike CPU caches, GPU caches are generally smaller relative to the number of compute units, reflecting the design emphasis on throughput rather than latency optimisation.

The main device memory of the GPU is implemented using High Bandwidth Memory (HBM), a three-dimensional stacked DRAM technology integrated on the same package as the GPU. HBM achieves extremely high memory bandwidth by using a very wide memory interface with thousands of parallel data lines, in contrast to conventional DRAM technologies that rely on narrower interfaces and higher clock frequencies. The NVIDIA A100 GPU employs HBM2e memory, providing a capacity of 80 GB and a peak bandwidth of 2039 GB/s, as shown in Table 2.1.

Although HBM provides very high bandwidth, its access latency is substantially higher than that of on-chip memories such as registers and shared memory. Consequently, achieving high performance on GPUs requires structuring computations to maximise data reuse in fast on-chip memory and to access global memory in a bandwidth-efficient manner. In many scientific applications, including tensor network contractions, overall performance is often limited by the available memory bandwidth rather than by the peak floating-point throughput of the compute units.

This hierarchical organisation is summarised conceptually as follows:

- **Registers:** fastest storage, private to each thread, limited capacity
- **Shared memory:** fast on-chip memory shared within a thread block
- **L2 cache:** intermediate storage shared across all SMs
- **HBM global memory:** large-capacity, high-bandwidth main memory

Efficient utilisation of this memory hierarchy is a central factor in the performance of GPU kernels and plays a critical role in the optimisation of tensor network algorithms.

Table 2.1.: Key hardware specifications of the NVIDIA A100 (SXM4-80GB).

Property	Value
Streaming Multiprocessors (SMs)	108
CUDA cores per SM	64
Tensor cores per SM	4
Warp schedulers per SM	4
Maximum threads per SM	2048
Maximum warps per SM	64
Maximum blocks per SM	32
Register file size per SM (32-bit registers)	256 000
Shared memory per SM (KB)	164
L2 cache size (MB)	40
HBM memory capacity (GB)	80
Peak memory bandwidth (GB/s)	2039
Base clock frequency (GHz)	1.41

Table 2.2.: Theoretical peak floating-point throughput of the A100 GPU.

Precision	Peak TFLOPS	Relative speed
FP64 (CUDA cores)	9.7	1.0×
FP64 (Tensor cores)	19.5	2.0×
FP32	19.5	2.0×
TF32 (Tensor cores)	156.0	16.1×
FP16 (Tensor cores)	312.0	32.2×

2.2.3. NVIDIA A100 Ampere Architecture

Hardware Overview

The A100 employs HBM2e as its main device memory, providing the high memory bandwidth necessary to sustain its computational throughput.

Table 2.3.: Derived theoretical limits of the A100 architecture.

Metric	Value
Peak FP32 performance per SM (TFLOPS)	0.18
Peak FP64 performance per SM (TFLOPS)	0.09
Tensor core FP16 performance per SM (TFLOPS)	2.89
Memory bandwidth per SM (GB/s)	18.88
Total CUDA cores	6912
Total FP64 cores	3456
Maximum resident warps	6912
Maximum resident threads	221 184
Arithmetic intensity threshold FP32 (FLOPs/byte)	9.6
Arithmetic intensity threshold FP64 (FLOPs/byte)	4.8

Table 2.4.: Approximate memory access latency at different hierarchy levels.

Memory level	Latency (cycles)
Registers	1
Shared memory	20
L2 cache	200
HBM global memory	500

Theoretical Peak Performance

Derived Performance Limits

Memory Latency

2.3. CUDA Programming Model

2.3.1. Thread Hierarchy and Kernel Launch

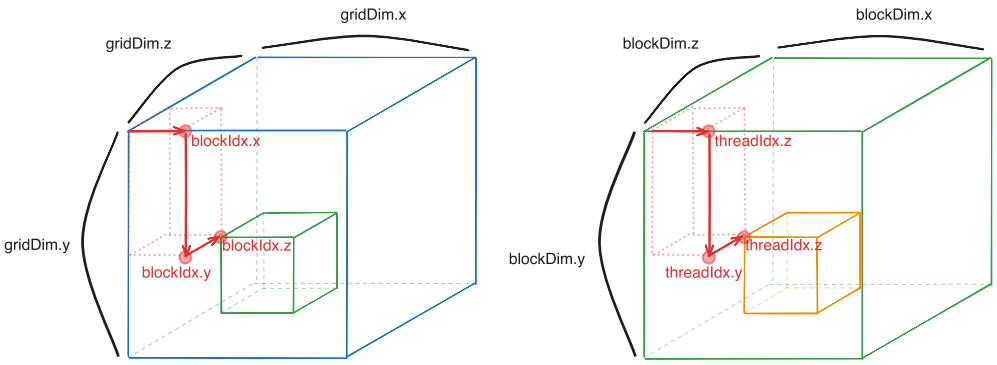

Figure 2.3.: CUDA execution hierarchy showing the mapping of grids to thread blocks and threads. Threads are organised in up to three dimensions and identified using the built-in variables `threadIdx` and `blockIdx`.

The hierarchical organisation of grids, thread blocks, and threads is illustrated in Figure 2.3.

2.3.2. Shared Memory and Synchronisation

2.3.3. Memory Coalescing and Bank Conflicts

2.3.4. Performance Profiling with Nsight Compute

2.4. Related Work

2.4.1. cuBLAS and cuTENSOR

2.4.2. ChASE Eigensolver

2.4.3. Existing GPU Tensor Network Implementations

3. Design and Methodology

3.1. Target Kernels

3.2. Algorithmic Approach

3.3. Data Layout and Memory Strategy

3.4. Baseline Selection

4. Implementation

4.1. Kernel Design

4.2. Shared Memory Tiling

4.3. Occupancy and Launch Configuration

4.4. Integration and Build System

5. Results

5.1. Experimental Setup

5.2. Single-GPU Performance

5.3. Profiling Analysis

5.4. Comparison with cuBLAS and cuTENSOR

5.5. Scaling Behaviour

5.6. Discussion

6. Conclusion

6.1. Summary

6.2. Limitations

6.3. Future Work

Bibliography

[NVI20] NVIDIA Corporation. *NVIDIA A100 Tensor Core GPU Architecture*, 2020. Whitepaper v1.0.

[NVI22] NVIDIA Corporation. *NVIDIA H100 Tensor Core GPU Architecture*, 2022. Whitepaper.

[NVI24a] NVIDIA Corporation. cuBLAS library, 2024. `https://developer.nvidia.com/cublas`.

[NVI24b] NVIDIA Corporation. *CUDA C++ Programming Guide*, 2024. `https://docs.nvidia.com/cuda/cuda-c-programming-guide/`.

[NVI24c] NVIDIA Corporation. cuTENSOR: A high-performance tensor primitives library, 2024. `https://developer.nvidia.com/cutensor`.

[NVI25] NVIDIA Corporation. cuBLASDx: Device side BLAS extensions, 2025. `https://docs.nvidia.com/cuda/cublasdx/`.

[SSK17] Paul Springer, Tong Su, and Tamara G. Kolda. Tensor contractions with extended BLAS kernels on CPU and GPU. In *IEEE 24th International Conference on High Performance Computing (HiPC)*. IEEE, 2017. `https://research.nvidia.com/sites/default/files/pubs/2017-10_Tensor-Contractions-with/tensors_hipc.pdf`.

[WDADN22] Xinzhe Wu, Davor Davidović, Sebastian Achilles, and Edoardo Di Napoli. ChASE: a distributed hybrid CPU-GPU eigensolver for large-scale hermitian eigenvalue problems. In *Proceedings of the Platform for Advanced Scientific Computing Conference (PASC '22)*, pages 1–12. ACM, 2022.

[WDN23] Xinzhe Wu and Edoardo Di Napoli. Advancing the distributed multi-GPU ChASE library through algorithm optimization and NCCL library. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W '23)*, pages 1688–1696. ACM, 2023.

[WSDN19] Jan Winkelmann, Paul Springer, and Edoardo Di Napoli. ChASE: Chebyshev accelerated subspace iteration eigensolver for sequences of Hermitian eigenvalue problems. *ACM Transactions on Mathematical Software*, 45(2):1–34, 2019.

[Wu19] Xinzhe Wu. *Contribution to the Emergence of New Intelligent Parallel and Distributed Methods Using a Multi-level Programming Paradigm for Extreme Computing*. PhD thesis, University of Lille, 2019.

A. Supplementary Benchmarks

TODO

Declaration of Authorship

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any quotes accordingly.

Aachen, February 17, 2026 Daniel Sinkin