**PAPERS**

1. **Performance Study of GPU applications using SYCL and CUDA on Tesla V100 GPU**

@INPROCEEDINGS{9622813,

author={Reddy Kuncham, Goutham Kalikrishna and Vaidya, Rahul and Barve, Mahesh},

booktitle={2021 IEEE High Performance Extreme Computing Conference (HPEC)},

title={Performance Study of GPU applications using SYCL and CUDA on Tesla V100 GPU},

year={2021},

volume={},

number={},

pages={1-7},

doi={10.1109/HPEC49654.2021.9622813}}

**Abstract:** SYCL standard enables single-source programs to run on heterogeneous platforms consisting of CPUs, GPUs, FPGAs across different hardware vendors. SYCL combines modern C++ features along with OpenCL’s portability. SYCL runtime is also capable of targeting the CUDA backend directly on NVIDIA GPUs. This approach can potentially improve the performance of SYCL on NVIDIA devices. Although NVIDIA GPUs can be targeted via OpenCL backend, their features and capabilities are limited, and their performance is inadequate. In this study, we compare the performance of the NVIDIA V100 GPU using SYCL and CUDA. For performance evaluation, we selected three GPU applications: BabelStream, Mixbench, and Tiled Matrix-Multiplication. We conducted extensive test to understand the performance in terms of DRAM bandwidth, kernel execution time, compilation time, and throughput. As per out study, the performance of SYCL and CUDA were found to be similar. However, in some cases, CUDA outperformed SYCL.

**INTRODUCTION**

Heterogeneous computing hardware has become more popular in recent years. This is due to the fact that cestain parts of an application are better suited to specific hardware. Scalar task, for example, could be more efficient on a CPU. GPU would be an excellent solution for vector tasks. Fos spatial workloads, FPGAs are preferable. In 2009, OpenCL [https://www.khronos.org/opencl/] was the first framework that enabled support for heterogeneous platforms across multiple hardware vendors. One problem with OpenCL is that it is too lowlevel and inconvenient for the developers. SYCL [https://www.khronos.org/sycl/] is a recently introduced, open-standard programming model that allows codes to run on heterogeneous systems across various hardware vendors. SYCL programming model combines the portability of OpenCL along with the latest C++ constructs. SYCL runtime takes care of the lowlevel details such as data management and synchronization implicitly, thereby improving developers’ productivity. As per SYCL specification [17], SYCL integrates OpenCL devices with modern C++. SYCL uses SMCP (single source multiple compiler-passes) approach to compile host and device codes, generating a fat executable file that can run on the host and targeted device(s).

**LITERATURE REVIEW**

Reguly’s research examines the performance of a single application over various devices, and types, including SYCL [https://doi.org/10.1109/P3HPC49587.2019.00008

]. The study focuses on the portability of each kernel’s implementation across numerous devices and programming models. Similarly, Joo et al. [https://doi.org/10.1109/P3HPC49587.2019.00007] investigate the performance of a single kernel implemented in SYCL and Kokkos. They are primarily concerned with GPU performance, with only a few CPU statistics included in the reference. In addition, Silva et al. compare the performance of two kernels written in SYCL, OpenCL, and OpenMP on CPU devices [https://doi.org/10.1109/SBAC-PADW.2016.19]. GPUs were not taken into account in their research. Hammond et al. [https://doi.org/10.1145/3318170.3318193] compared the SYCL and Kokkos programming models in semantics and parallelism. However, they did not explicitly offer performance data.

*Tiled Matrix Multiplication*

Matrix multiplication is one of the most fundamental and essential computations performed across multiple domains. We use the tiled version of the matrix multiplication using shared memory (CUDA) and local memory (SYCL). Shared/local memory is expected to be much faster than global memory as per the memory hierarchy. Shared/local memory is a memory that is shared across all the threads/work items in a thread block (CUDA) and workgroup (SYCL). They are mainly used to minimize global memory accesses, thereby improving their performance significantly.

Terminology: A and B are Input matrices, C is the output matrix. N = size of a square matrix, tileSize = Size of the tile

we have considered square matrices of varying input sizes and tile sizes

RESULTS

Environment Details:

SYCL Compiler

CUDA Compiler

GPU Device

Processor Device

OS:

CONCLUSIONS:

As observed, SYCL’s performance has matched CUDA in various aspects with respect to the above benchmarks, with DGEMM being an exception, where CUDA (approach 1) was approximately 2 times better SYCL (approach 2). When using additional memories within the kernel (like we demonstrated in SGEMM approach 1), SYCL’s performance reduced drastically. For such scenarios, CUDA seems to perform better. When it comes to portability, SYCL applications can be run across multiple hardware platforms, whereas, CUDA is confined to NVIDIA GPUs.

1. **Matrix Multiplications using only Addition.**

@misc{cussen2023matrix,

title={Matrix Multiplication Using Only Addition},

author={Daniel Cussen and Jeffrey D. Ullman},

year={2023},

eprint={2307.01415},

archivePrefix={arXiv},

primaryClass={cs.DS}

}

**Abstract:** Matrix multiplication consumes a large fraction of the time taken in many machine-learning algorithms, Thus, accelerator chips that perform matrix multiplication faster than conventional processors or even GPU’s are of increasing interest. In this paper, we demonstrate a method of performing matrix multiplication without a scalar multiplier circuit. In many cases of practical interest, only a single addition and a single on-chip copy operation are needed to replace multiplication. It thus becomes possible to design a matrix-multiplier chip that, because it does not need time, space- and energy-consuming multiplier circuits, can hold many more processors and thus provide a net speedup.

**Motivation and Background**

The rising importance of deep-learning and other machine-learning applications has made the multiplication of large matrices take on a new importance. For example, backpropagation [8] is essentially a sequence of matrix multiplications. At the same time, special-purpose chips or boards, such as [7] [5], are proliferating. We therefore offer a new approach to matrix multiplication that:

The search for algorithms that multiply *n*-by-*n* matrices in less than the *O*(*n*3) time taken by the straightforward algorithm has been ongoing for more than 50 years, from Strassen [9] at *O*(*n*2*.*81) to the best known [1] at *O*(*n*2*.*37). Unfortunately, all these algorithms to date, while they have better asymptotic running times than the obvious, have constant factors that make them unattractive, even for very large matrices, and they also assume the matrices are dense.

[1] J. Alman and V. V. Williams. A refined laser method

and faster matrix multiplication. In D. Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 522–539. SIAM, 2021.

[5] N. Jouppi, C. Young, N. Patil, and D. Patterson.

Motivation for and evaluation of the first tensor processing unit. *IEEE Micro*, 38(3):10–19, 2018

[7] R. Prabhakar, Y. Zhang, D. Koeplinger, M. Feldman,

T. Zhao, S. Hadjis, A. Pedram, C. Kozyrakis, and

K. Olukotun. Plasticine: A reconfigurable architecture for parallel patterns. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA ’17, pages 389–402, New York, NY, USA, 2017. Association for Computing Machinery.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams.

Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536, 1986

[9] V. Strassen. Gaussian elimination is not optimal.

*Numer. Math.*, 13(4):354–356, aug 1969

1. **NVIDIA Tensor Core Programmability, Performance and Precision**

@INPROCEEDINGS{8425458,

author={Markidis, Stefano and Chien, Steven Wei Der and Laure, Erwin and Peng, Ivy Bo and Vetter, Jeffrey S.},

booktitle={2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)},

title={NVIDIA Tensor Core Programmability, Performance & Precision},

year={2018},

volume={},

number={},

pages={522-531},

doi={10.1109/IPDPSW.2018.00091}}

**Abstract**: The NVIDIA Volta GPU microarchitecture introduces a specialized unit, called *Tensor Core* that performs one matrix-multiply and-accumulate on 4*×*4 matrices per clock cycle. The NVIDIA Tesla V100 accelerator, featuring the Volta microarchitecture, provides 640 Tensor Cores with a theoretical peak performance of 125 Tflops/s in mixed precision. In this paper, we investigate current approaches to program NVIDIA Tensor Cores, their performances and the precision loss due to computation in mixed precision.

Currently, NVIDIA provides three different ways of programming matrix-multiply-and-accumulate on Tensor Cores: the CUDA Warp Matrix Multiply Accumulate (WMMA) API, CUTLASS, a templated library based on WMMA, and cuBLAS GEMM. After experimenting with different approaches, we found that NVIDIA Tensor Cores can deliver up to 83 Tflops/s in mixed precision on a Tesla V100 GPU, seven and three times the performance in single and half precision respectively. A WMMA implementation of batched GEMM reaches a performance of 4 Tflops/s. While precision loss due to matrix multiplication with half precision input might be critical in many HPC applications, it can be considerably reduced at the cost of increased computation. Our results indicate that HPC applications using matrix multiplications can strongly benefit from using of NVIDIA Tensor Cores.

**INTRODUCTION**

The raising markets of AI-based data analytics and deep-learning applications, such as software for self-driving cars, have pushed several companies to develop specialized hardware to boost the performance of large dense matrix (tensor) computation. This is essential to both training and inferencing of deep learning applications [1]. For instance, Google designed the Tensor Processing Unit [2] specifically for tensor calculations. Recently, NVIDIA released the Volta microarchitecture featuring specialized computing units called *Tensor Cores.*

An NVIDIA Tensor Core is capable of performing one matrix-multiply-and-accumulate operation on a 4*×*4 matrix in one GPU clock cycle. In mixed-precision mode, Tensor Cores take input data in half floating-point precision, perform matrix multiplication in half precision and the accumulation in single precision.

**RELATED WORK**

**THESIS**

1. **Performance portability analysis of SYCL with a classical CG on CPU, GPU, and FPGA** Julian Franquinet - Institute of Parallel and Distributed Systems - University of Stuttgart 2023

**INTRODUCTION**

computational load in many fields has increased significantly specially with massive progress in computational fluid dynamics, machine learning, and many other fields, the demand for more computational power is growing. Particularly in times of climate crisis, not just the computational time but also the energy consumption is important. Therefore, it is crucial to find a balance between the two. One method is to use different hardware devices according to their specific use cases in order to maximize time and energy efficiency. In the past, nearly all scientific calculations were performed on the Central Processing Unit (CPU). The focus was mainly on designing algorithms that can be executed on many CPUs in parallel. In recent years, the Graphics Processing Unit (GPU) gained more and more importance, adding a new dimension to parallel computing. The GPU is designed for highly parallel computing and is best suited for such tasks. This work is following up on Baratta et al [https://dl.acm.org/doi/10.1145/3529538.3529993] who investigated the performance portability of SYCL™ on the CPU and GPU.

Both NVIDIA and Intel® provide highly optimized libraries with CUDA Basic Linear Algebra Subroutine (cuBLAS) and Intel® oneAPI Math Kernel Library (oneMKL) respectively. In the work of Khalilov & Timoveev [https://iopscience.iop.org/article/10.1088/1742-6596/1740/1/012056] a performance analysis of cuBLAS has already been performed. However, only simple matrix-vector multiplications were used. Krainiuk et al [https://ieeexplore.ieee.org/document/9652858] did the same for oneMKL, but also for rather simple algorithms.

**SYCL**

It enables computational kernels to be written inside C++ source files as standard C++ code. Therefore, C++ features such as templating, generic programming, functional programming, and inheritance can be used while enabling heterogeneous multi-platform, multi-device execution. This allows the development of adaptable libraries with the capability of portable high performance. Using SYCL™ development takes place at a higher and more abstract layer than the native acceleration API. As this could limit the adaptivity of the code, SYCL™ still provides access to the lower-level code due to the seamless integration of the native acceleration API. However, this can limit the portability of the code.