**ABSTRACT**

La multiplicación de matrices es una de las operaciones fundamental en muchos campos de la ciencia y la ingeniería y gracias al rápido crecimiento de la computación paralela, se han desarrollado varias arquitecturas enfocadas en acelerar esta operación. Dos modelos de programación populares hoy en día son CUDA y SYCL. Este estudio evalúa el rendimiento computacional de los algoritmos de multiplicación de matrices en estas dos modelos. Se realizan comparaciones entre los tiempos de ejecución, speedup, consumo de energía y la eficiencia de los algoritmos con diferentes tamaños de matrices. Los resultados mostraron que el uso de las librerías de NVIDIA e Intel presentaron los mejores resultados en cuanto a rendimiento y uso de los recursos en comparación con otras implementaciones. Sin embargo, este rendimiento se alcanza a costa de la perdida de precisión en los resultados. Por otro lado, SYCL mostro resultados aceptables en cuanto a rendimiento sin presentar errores de precisión a diferencia de varias de las otras implementaciones. Los algoritmos de SYCL utilizados en este trabajo presentan opciones de optimización que no fueron implementadas. Esto se hizo con el fin de mantener el Kernel de SYCL sin optimizaciones directamente relacionadas con las GPU de NVIDIA y así mantener su portabilidad entre la CPU y la GPU.

**INTRODUCTION**

En los últimos años, los sistemas heterogéneos para computación de alto rendimiento se han vuelto muy populares debido a que ciertas partes de las aplicaciones se adaptan mejor a cierto hardware especifico. Además, estos sistemas se han convertido en una de las soluciones en lo que se refiere a la eficiencia energética requerida establecida por las políticas actuales. Campos como la dinámica computacional de fluidos, el aprendizaje automático, entre otros, han incrementado su carga computacional requiriendo cada vez más poder de cómputo, lo que hace necesario poder orquestar de forma eficiente estos sistemas heterogéneos manteniendo un equilibrio entre rendimiento y eficiencia energética [Matrix Multiplication Using Only Addition].

Una de las operaciones fundamentales en varios de los campos mencionados anteriormente, es la multiplicación de grandes matrices. Áreas como el aprendizaje profundo (subárea del aprendizaje automático) han hecho que se adquiera un nuevo interés en la optimización de esta operación debido a que consume una gran fracción de tiempo durante el entrenamiento de los modelos. El algoritmo de backpropagation [D. E. Rumelhart, G. E. Hinton, and R. J. Williams] es esencialmente una secuencia de multiplicación de matrices.

La búsqueda de algoritmos de multiplicación de matrices NxN que reduzcan el tiempo que tarda el algoritmo sencillo (O(n^3)) se ha dado desde los años 50, desde Strassen [V. Strassen. Gaussian elimination is not optimal. Numer. Math., 13(4):354–356, aug 1969] con O(n^2.81) hasta el mejor conocido hoy en día con O(n^2.37)[ 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]. En computación de alto rendimiento estos algoritmos suelen agruparse en rutinas conocidas como GEMM (General Matrix Multiplication).

En cuanto a las cargas de trabajo de las redes neuronales profundas, los tamaños típicos de matrices en las operaciones GEMM se acercan al orden de 10000 [Qin E, Samajdar A, Kwon H, et al. SIGMA: A sparse and irregular GEMM accelerator with flexible interconnects for DNN training. Proceedings of the 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA); 2020: 58-70], por lo que ha requerido que las rutinas dividan efectivamente las matrices para encajar en la jerarquía de memoria y mejorar el rendimiento. En las librerías numéricas actuales se utiliza un enfoque en capas [Goto K, van de Geijn RA. Anatomy of high-performance matrix multiplication. ACM Trans Math Softw. 2008;34(3):12.] la cual reorganizan los datos para mejorar la localidad de estos a medida que avanzan desde la memoria principal a través de la jerarquía de memoria [DOI: 10.1002/spe.3214]. Librerías como Intel MKL, cuBLAS, IBM ESSL, OpenBLAS, Eigen entre otras, usan este enfoque de capas de una manera altamente optimizada. Sin embargo, todas presentan inconvenientes como: 1) cada librería debe ser instalada según la arquitectura; 2) cada librería debe escribirse en el código ensamblador que le pertenece a cada arquitectura; 3) se presentan desfaces entre la salida de nuevas arquitecturas y la creación de librerías optimizadas para dicha arquitectura.

Durante décadas el procesamiento de los cálculos científicos fue realizadas en la CPU. Esto motivo a que los diseños de los algoritmos se enfocaran principalmente en poder ejecutar estas rutinas de forma paralela en un gran número de CPUs. En los últimos años, con el surgimiento de las GPUs, tales diseños fueron adaptados a esta nueva arquitectura, la cual es mas adecuada para este tipo de tareas altamente paralelas [https://dl.acm.org/doi/10.1145/3529538.3529993]. Tanto NVIDIA como Intel proveen librerías altamente optimizadas para ejecutar estas rutinas en sus respectivas arquitecturas, CUDA Basic Linear Algebra Subroutine (cuBLAS) e Intel oneAPI Math Kernel Library (oneMKL). Trabajos como los de Khalilov & Timoveev [https://iopscience.iop.org/article/10.1088/1742-6596/1740/1/012056] y Krainiuk et al [https://ieeexplore.ieee.org/document/9652858] han evaluado el rendimiento de estas librerías con multiplicaciones simples matriz-vector y otros algoritmos bastante simples, mostrando una gran eficiencia y rendimiento en sus cálculos.

NVIDIA introdujo con su microarquitectura Volta una unidad especializada llamada Tensor Core, que logra realizar una multiplicación de matrices de 4x4 por cada ciclo de reloj. Las primeras versiones de esta arquitectura proporcionaban 640 Tensor Cores logrando un rendimiento teórico de 125 Tflops/s en precisión mixta. Actualmente, NVIDIA proporciona tres formas para realizar la multiplicación y acumulación de matrices (FMA) con el uso de los Tensor Cores: API CUDA Warp Matrix Multiply Accumulate (WMMA), CUTLASS - librería de plantillas basado en WMMA y cuBLAS GEMM [doi={10.1109/IPDPSW.2018.00091}].. En uso de la precisión mixta incurre en perdida de precisión en los cálculos al usar datos en half floating-point. Esto puede ser critico en muchas aplicaciones de HPC, sin embargo, reduce considerablemente el tiempo de ejecución [doi={10.1109/IPDPS47924.2020.00071}}].

La heterogeneidad creciente en arquitecturas HPC ha hecho que se busquen nuevas formas de reducir la complejidad de la programación y la curva de aprendizaje de las varias decenas de librerías y estándares existentes. En 2009 [https://www.khronos.org/opencl/] aparece OpenCL quien fue el primer framework de desarrollo con la capacidad de soportar este tipo de arquitecturas. Su principal inconveniente se debió a la programación de muy bajo nivel que este requería. En 2014 aparece SYCL [https://www.khronos.org/sycl/], un modelo de programación abierto que al igual que OpenCL permite la programación de sistemas heterogéneos presentando una gran portabilidad sin la complejidad de su predecesor. El runtime de SYCL es quien se encarga de los detalles de bajo nivel de manera implícita, tales como la gestión de memoria y la sincronización de datos. Su principal característica es que usa el enfoque SMPC (single source multiple compiler-passes) para compilar tanto los códigos del host como los del device generando un archivo ejecutable grueso que puede ser ejecutado en ambas arquitecturas. [ doi={10.1109/HPEC49654.2021.9622813}}] [https://doi.org/10.1109/SBAC-PADW.2016.19].

Existen varios trabajos que examinan el rendimiento de SYCL como es el caso de Reguly [https://doi.org/10.1109/P3HPC49587.2019.00008] en donde se evalúa un kernel en varios dispositivos. En el trabajo de Joo et al. [https://doi.org/10.1109/P3HPC49587.2019.00007] se evalúa el rendimiento de un Kernel SYCL en la GPU, pero la CPU no es un actor activo en este estudio. Existen varios otros como los de Silva et al [https://doi.org/10.1109/SBAC-PADW.2016.19] y Hammond et al [https://doi.org/10.1145/3318170.3318193] en donde se realizan comparaciones en arquitecturas específicas, sin embargo, no se ofrecen datos de rendimiento de forma explícita. En el trabajo de Kuncham et al [doi={10.1109/HPEC49654.2021.9622813}}] se muestra que SYCL puede igualar el desempeño de CUDA en varios aspectos, sin embargo, se muestra a DGEMM como una excepción en donde CUDA presento mejores resultados y concluyendo que CUDA presento mejor rendimiento y SYCL presenta gran portabilidad al no estar confinado a ninguna arquitectura.

El objetivo de este trabajo es realizar un estudio de la viabilidad de SYCL en el uso de aplicaciones científicas que dependan de las rutinas de multiplicación de matrices. Librerías como oneMKL o cuBLAS son altamente dependientes de sus correspondientes arquitecturas y restringen la portabilidad de dichas aplicaciones. SYCL por otro lado se presenta como un framework para sistemas heterogéneos, en donde sus kernels son altamente portables y pueden llegar a presentar un alto rendimiento computacional en comparación con otros estándares comúnmente usados en la computación de alto rendimiento. La evaluación se llevo a cabo mediante la comparación de un kernel SYCL de multiplicación de matrices frente a otras técnicas que abarcan openMP, Intrinsicos (AVX2 y AVX512), CUDA, el uso de las librerías oneMKL y cuBLAS. En cuanto al uso de CUDA y cuBLAS se realizaron pruebas con y sin Tensor Cores.

Este paper está organizado como de la siguiente manera: primero se presentó una introducción con los trabajos previos realizados correspondientes a la evaluación de la multiplicación de matrices y el uso de SYCL. En la sección II se presenta una descripción general XXXX en la sección III se presentan los resultados obtenidos, en la sección IV se realiza la evaluación y discusión detallada de ellos. Finalmente, en la última sección se presentan las conclusiones del trabajo.

**RESULTS**

Environment Details:

SYCL Compiler

CUDA Compiler

GPU Device

Processor Device

OS: