POLITECNICO
MILANO 1863

Challenge 2 - Parallel Computing
Convolution 2D in CUDA

10766381 - 10766863 - 10776157
Federico Pinto - Mattia Gotti - Michele Milani

Introduction

With this document, we share the work done on the second Parallel Computing challenge. It required implementing a 2D convolution using CUDA, leveraging GPU computing and tiling technique convered during the course.

Experimental Setup

The code was implemented in a single Python notebook file (`.ipynb`), executed using Google Colab. Thanks to the extension provided by the professor and the opportunity to utilize remote GPUs, we were able to complete the work.

Design Choices

For the implementation, we *separately* developed the algorithm using CUDA, including a version with the tiling technique, as requested. Both algorithms were tested in two variants, one utilizing `__constant__` memory for the convolution mask due to its significant impact on performance.

The input matrix is divided into smaller tiles, loaded into shared memory with padding for boundary overlaps. The convolution filter is stored in constant memory for faster access. Each thread computes the convolution for its assigned element through element-wise multiplication and summation, with boundary checks to prevent out-of-bound memory access. This approach minimizes global memory access and leverages CUDA's memory hierarchy for better performance.

Performance Measurements

We decided to evaluate performance based on four main aspects related to temporal performance, using a 5x5 convolution mask as a reference:

1. **Performance varying tile width**: we compared the two algorithms by testing them with different tile sizes.
2. **Tile vs non-tile performance**: we assessed how the algorithms behave, varying the matrix size, in both constant and non-constant mask versions separately, to observe the performance differences.
3. **Constant vs non-constant performance**: we compared the algorithms with constant and non-constant mask versions to highlight the significant performance differences.
4. **Overall performance**: we evaluated the general performance of all algorithms to compare them from a broader perspective.

Conclusion

The main and most significant aspects we observed are:

- **Tiled algorithm advantages**: The tiled algorithm is beneficial for large matrices, as the overhead of loading data into shared memory is compensated by performance gains from efficient memory usage.

- **Non-tiled algorithm with constant memory**: The constant memory version of the non-tiled algorithm reduces memory access latency, allowing it to perform similarly to the tiled version for large matrices. However, testing was limited to 16k×16k matrices on Colab.
- **General comparison of implementations**: Constant memory significantly improves performance, especially in the non-tiled version, but the tiled algorithm can still outperform or match the non-tiled one.
- **Impact of mask size variation**: Changing the mask size affects performance, but the tiled algorithm showed minimal improvement, likely due to overhead from small masks.