Exercise 09

Contents

[1 Atomic Add, Warp Shuffles and Thread Configuration 1](#_Toc90386556)

[1.1 Implementation 1](#_Toc90386557)

[1.2 Performance Comparisons 2](#_Toc90386558)

[2 Dot Product with OpenCL 5](#_Toc90386559)

[2.1 Implementation 5](#_Toc90386560)

[2.2 Performance Comparison 6](#_Toc90386561)

[3 OpenCL Sparse Matrix-Vector Product Kernels 6](#_Toc90386562)

[3.1 Implementation 7](#_Toc90386563)

[3.2 Performance Comparison 8](#_Toc90386564)

# Comparison CUDA / HIP

The code for this part can be found in the files **ex05\_1\_conj\_grad\_cuda.cpp** and **ex05\_1\_conj\_grad\_hip.cpp**.

## Conversion

The conjugate gradients implementation from exercise 4 was converted to HIP. The conversion was very easy using only three steps:

* String replacement: FindReplace(“cuda”, ”hip”)
* String manipulation: every identifier, that ended in .x had the .x replaced by \_x. The first letter is capitalized and then the substring hip is prefixed. E.g. blockDim.x -> hipBlockDim\_x
* Kernel calls had to be converted, which was the only semi-trivial step.

## Code Remarks

The original code used warp shuffles to calculate the dot product. This was replaced by a function, that uses block-level shared arrays instead. The reasoning was, that a quick internet search suggested, that while HIP is able to perform warp shuffles, potential differences in warp size and function names could lead to complications.

It should also be noted, that the original solution has an unsafe initialization of the return value of the dot product, since initialization is performed inside the kernel:

\_\_global\_\_ unsafeDotProduct(args){

**if** (thread\_id\_global == 0)

\*global\_result = 0;

// …

// do lots of work

// …

**if** (threadIdx.x == 0)

atomicAdd(\*global\_result, local\_result);

}

While this is unsafe, the implementation still gives convergence. There even is reason to believe, that it does not fail a single time, since the number of required iterations does not change between several runs of the program. However, the correct approach would be to initialize this variable outside the kernel. This problem was however not fixed, since the task was to compare CUDA with HIP, rather than implementing a correct dot product. This means both the CUDA and the HIP implementation have this error in the code.

## Performance Comparison

The comparison shows almost perfect performance equality between the two implementations.

Chart, line chart

Description automatically generated

Figure 1: Performance comparison between HIP and CUDA