Vienna University of Technology

360.252 Computational Science on Many Core Architectures

Institute for Microelectronics

Exercise 3

Authors:
Camilo Tello Fachin
12127084

Supervisor:
Dipl.-Ing. Dr.techn. Karl Rupp

November 2, 2022

TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

Abstract

Here documented the results of exercise 3.

Contents

1 Strided and Offset Memory Access (3 Points)

1.1 Task a

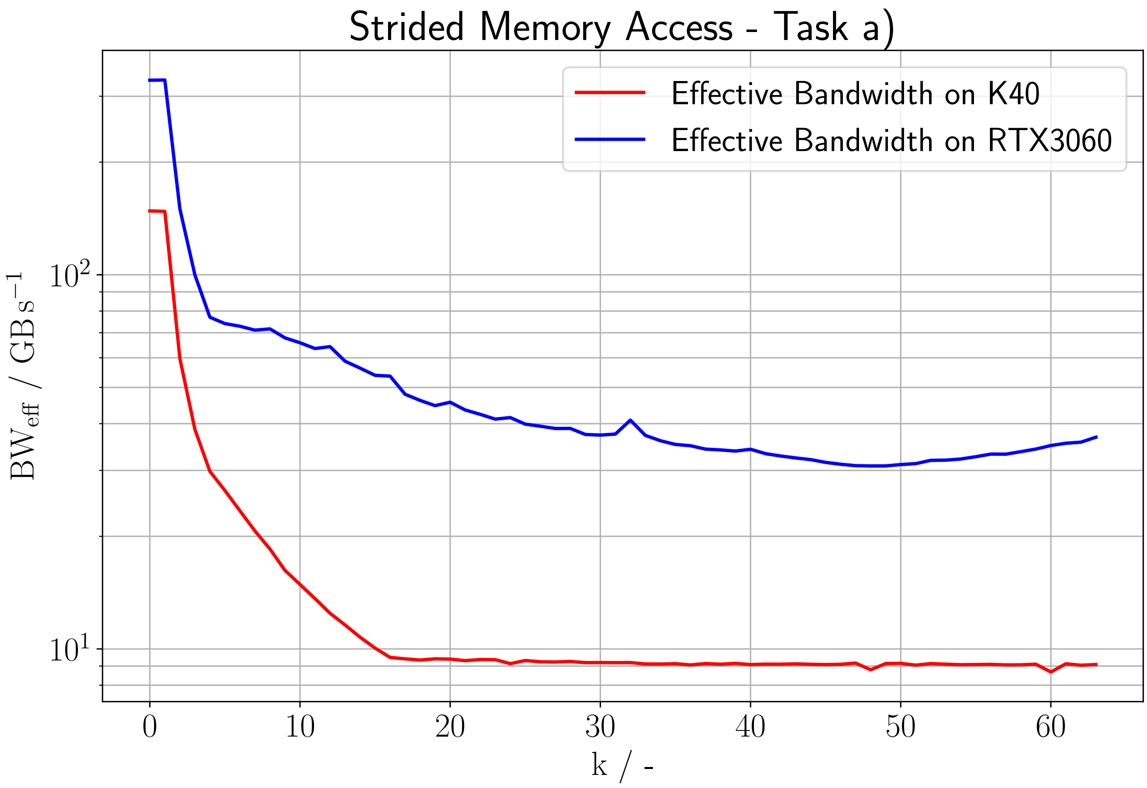


Figure 1: Plot for task 1a - see code in appendix A

1.2 Task b

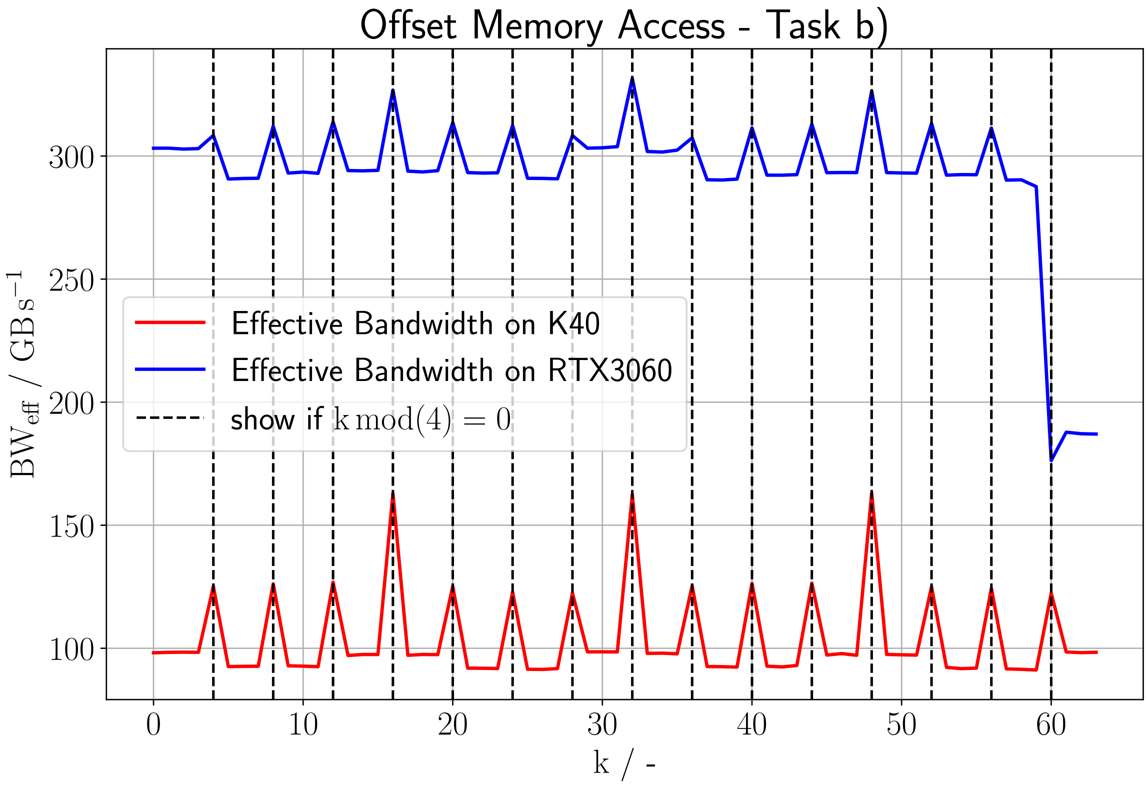
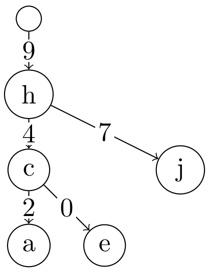


Figure 2: Plot for task 1b - see code in appendix B


9
h
4
7
c
j
2
0
a
e

A CPP CUDA Code - Strided and Offset Memory Access - Task 1a

C++ Listing for EX1 a)

```
#include <stdio.h>
#include "timer.hpp"
#include <algorithm>
#include <vector>

__global__ void addVec_kth(double *x, double *y, double *z, int N, int k) {
  unsigned int total_threads = blockDim.x * gridDim.x;
  unsigned int global_tid = blockIdx.x * blockDim.x + threadIdx.x;
  if (k==0) {
    k = 1;
  }
  for (unsigned int i = global_tid; i<N/k; i += total_threads) {
    z[i*k] = x[i*k] + y[i*k];
  }
}

// findMedian function for any vector lenghts, source geeksforgeeks.com
double findMedian(std::vector<double> a,
                  int n)
{
    if (n % 2 == 0) {
        std::nth_element(a.begin(),
                         a.begin() + n / 2,
                         a.end());
        std::nth_element(a.begin(),
                         a.begin() + (n - 1) / 2,
                         a.end());
        return (double)(a[(n - 1) / 2]
                        + a[n / 2])
               / 2.0;
    }
    else {
        std::nth_element(a.begin(),
                         a.begin() + n / 2,
                         a.end());
        return (double)a[n / 2];
    }
}

int main(void)
{
  // Task 1a//
  double *x, *y, *z, *gpu_x, *gpu_y, *gpu_z;
  double eff_BW;
  Timer timer;
  int N = pow(10.0, 8.0);
  std::vector<int> k_values(64, 0);
```

```
for(int i = 0; i<64; i++){
  k_values[i] = i;
}
std::vector<double> exec_timings = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
x = (double*)malloc(N*sizeof(double));
y = (double*)malloc(N*sizeof(double));
z = (double*)malloc(N*sizeof(double));
for (int i = 0; i < N; i++) {
  x[i] = (double)(i);
  y[i] = (double)(N-i-1);
}
cudaMalloc(&gpu_x, N*sizeof(double));
cudaMalloc(&gpu_y, N*sizeof(double));
cudaMalloc(&gpu_z, N*sizeof(double));
cudaMemcpy(gpu_x, x, N*sizeof(double), cudaMemcpyHostToDevice);
cudaMemcpy(gpu_y, y, N*sizeof(double), cudaMemcpyHostToDevice);
cudaMemcpy(z, gpu_z, N*sizeof(double), cudaMemcpyDeviceToHost);

for (int i = 0; i < 64; i++) {
  for (int m = 0; m < 11; m++) {
    timer.reset();
    addVec_kth<<<256, 256>>>(gpu_x, gpu_y, gpu_z, N, k_values[i]);
    cudaDeviceSynchronize();
    exec_timings[m] = timer.get();
  }
  if (k_values[i]==0) {
    eff_BW = 3 * N * sizeof(double) * pow(10,-9) / findMedian(exec_timings, 10);
  }
  else{
    eff_BW = 3 * floor((N/k_values[i])) * sizeof(double) * pow(10, -9) / findMedian(exec_timings, 10);
  }
  printf("%d,%g\n", k_values[i], eff_BW);
}

cudaFree(gpu_x);
cudaFree(gpu_y);
cudaFree(gpu_z);
free(x);
free(y);
free(z);
```

B CPP CUDA Code - Offset Memory Access - Task 1b

The code for the Offset Memory Access partial exercise is the same as for the Strided Memory Access partial exercise, except the `__global__` part where the offset is defined and the calculation of the effective bandwidth, where one can now also omit the case distinction for k=0.

C++ Listing for EX1 b)

```
__global__ void addVec_kth(double *x, double *y, double *z, int N, int k) {
  unsigned int total_threads = blockDim.x * gridDim.x;
  unsigned int global_tid = blockIdx.x * blockDim.x + threadIdx.x;
  if (k==0) {
    k = 1;
  }
  for (unsigned int i = global_tid; i<N-k; i += total_threads) {
    z[i+k] = x[i+k] + y[i+k];
  }
}

.
.
.

eff_BW = 3 * floor((N - k_values[i])) * sizeof(double) * pow(10, -9) / findMedian(exec_timings, 10);

.
.
.
```