



UNIVERSIDADE DE AVEIRO

DEPARTAMENTO DE ELECTRÓNICA, TELECOMUNICAÇÕES E
INFORMÁTICA

47022- ARQUITECTURA DE COMPUTADORES AVANÇADA

Home group assignment 2

Semi-Global Matching stereo processing using CUDA

8240 - MESTRADO INTEGRADO EM ENGENHARIA DE
COMPUTADORES E TELEMÁTICA

António Rafael da
Costa Ferreira
NMec: 67405

Rodrigo Lopes
da Cunha
NMec: 67800

Docentes: Nuno Lau e José Luís Azevedo

Janeiro de 2016
2015-2016

Conteúdos

1	Introdução	2
2	Exercício 1	3
	2.1 Cuda Kernel da função "determine_costs()"	3
3	Conclusão	5

1 Introdução

O trabalho proposto para a unidade curricular de Arquitetura de Computadores Avançada foi a implementação em CUDA para o processamento de um Semi-Global Matching.

Este programa tem como objetivo determinar a imagem de disparidade entre duas imagens idênticas mas de posições diferentes, como se de dois olhos se tratasse, uma vista com o olho da esquerda e outra com o olho da direita.

O relatório reflete todas as geometrias de kernel implementadas, formas de pensamento, métodos de como foram implementados os algoritmos, resultados, tutorial para correr o código elaborado, e por último a conclusão deste mesmo trabalho.

2 Exercício 1

2.1 Cuda Kernel da função "determine_costs()"

Neste primeiro exercício, era pedido que se desenvolvesse um kernel em CUDA que substituísse a função *determine_costs()*.

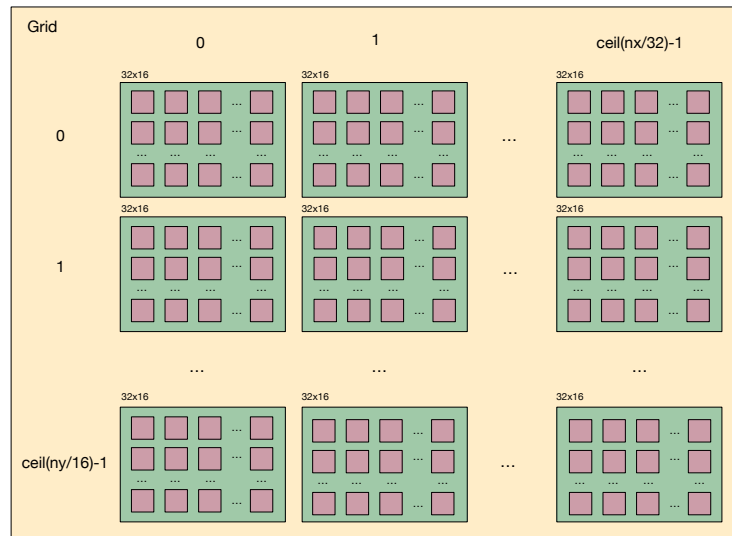


Figura 1:
Geometria do Kernel para a função *determine_costs()*

Neste kernel optou-se por uma geometria (Figura 1 constituída por uma grid de tamanho $(\text{ceil}(\text{nx}/32) \times \text{ceil}(\text{ny}/16))$ com blocos de 32×16 threads cada. Nesta função, cada thread corresponde a um pixel da imagem, e cada um calcula o valor de custo, sendo este a diferença entre as imagens num determinado pixel.

Este exercício foi ainda realizado de duas maneira, uma utilizando a *global memory*, e outra onde se coloca as imagens e o valor de COSTS na *texture memory*.

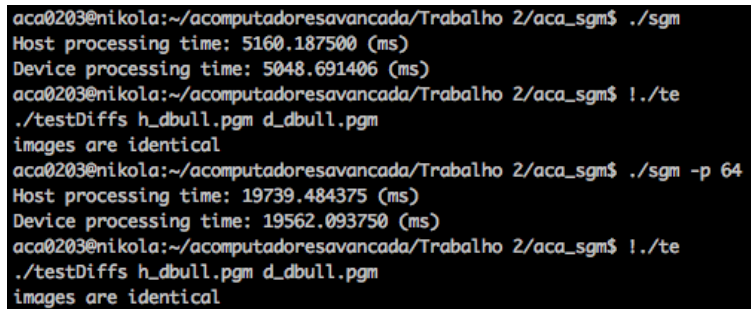
Para a *global memory* utilizou-se o seguinte algoritmo para desenvolver o kernel:

```
__global__ void determine_costs_device(const int *left_image, const int *right_image,
int *costs,
const int nx, const int ny, const int disp_range)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;

    if (i < nx && j < ny)
    {
```

```
for ( int d = 0; d < disp_range; d++ ) {  
    if(i >= d){  
        COSTS(i,j,d) = abs( LEFT_IMAGE(i,j) - RIGHT_IMAGE(i-d,j));  
    }  
}  
}
```

Com esta implementação obtiveram-se os seguintes resultados:



```
aca0203@nikola:~/acomputadoresavancada/Trabalho 2/aca_sgm$ ./sgm  
Host processing time: 5160.187500 (ms)  
Device processing time: 5048.691406 (ms)  
aca0203@nikola:~/acomputadoresavancada/Trabalho 2/aca_sgm$ !./te  
./testDiffs h_dbull.pgm d_dbull.pgm  
images are identical  
aca0203@nikola:~/acomputadoresavancada/Trabalho 2/aca_sgm$ ./sgm -p 64  
Host processing time: 19739.484375 (ms)  
Device processing time: 19562.093750 (ms)  
aca0203@nikola:~/acomputadoresavancada/Trabalho 2/aca_sgm$ !./te  
./testDiffs h_dbull.pgm d_dbull.pgm  
images are identical
```

Figura 2:
Resultados obtidos utilizando global memory

3 Conclusão

Este trabalho foi útil para assentar todos os conhecimentos que se foi obtendo ao longo destes anos, tanto em Arquitectura de Computadores Avançada como em Arquitectura de Computadores I e II. Ambas as entregas foram primeiro planeadas em papel antes de se avançar para a implementação o que resultou em bons resultados a nível de tempo despendido, para ambos as entregas fez-se uso de dois dias de trabalho.

Na primeira entrega houve uma falha na implementação, esqueceu-se de realizar o shift left 2 do program counter para calcular o endereço para onde é suposto saltar a instrução jump.

Já na segunda entrega teve-se os máximos cuidados para que tudo funcionasse perfeitamente.