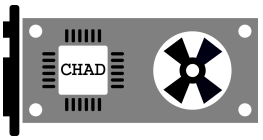


Computação Heterogénea de Alto Desempenho

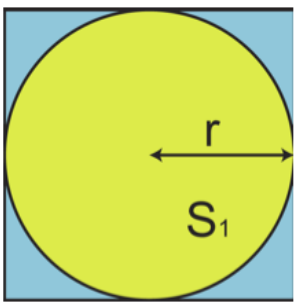
Mestrado em Engenharia Eletrotécnica e de Computadores

LAB 6

– CUDA –

1. Desenvolva um programa CUDA paralelo em GPU que calcule o desfasamento entre duas *streams* de vídeo. O objetivo é perceber quantas frames de atraso tem uma *feed* de vídeo em relação à outra. Para o efeito, use o código CUDA e os vídeo *feeds* fornecidos no UCstudent sob o nome 'video-feeds-SAD-code.zip'. calcule o desfasamento dos pares de vídeo que forem sendo fornecidos.
Nota: Durante a aula laboratorial será dado apoio à resolução desta questão.

2. a) Implemente um programa CUDA que use a biblioteca CURAND para calcular um valor aproximado do número π. A figura seguinte relaciona a área de um quadrado que contém o maior círculo que doe abarcar (respetivamente S_2 e S_1), e indica um caminho possível para a solução:



- Disk: $S_1 = \pi r^2$
- Square: $S_2 = 4r^2$
- $\pi = \frac{4S_1}{S_2}$

Nota: a equação que define a área de um círculo é dada por:

$(x - x_0)^2 + (y - y_0)^2 <= r^2$, em que `(x0, y0)` define as coordenadas do centro do círculo e `r` o seu raio.

b) Implemente uma nova versão que execute o programa até uma determinada precisão ser alcançada. Use todos os níveis de optimização que se adequarem.

3. Muitas aplicações de processamento de imagem usam filtros (e.g., para produzir *edge detection* ou *blur effects*), baseados na operação de convolução. Implemente um programa CUDA que aplica um filtro *blur* (de desfocagem ou passa-baixo). Neste exercício, o aluno é livre de escolher qualquer imagem ou filtro (pode usar um simples filtro de média, sem pesos, como o analisado na aula teórica que funciona como um filtro passa-baixo). Uma solução perfeita considerará as condições fonteira. Implemente duas versões do programa:
a) Uma versão simples (sem otimização).
b) Uma versão otimizada, em que o aluno pode implementar as estratégias que entender (opção mínima obrigatória: o uso de *shared memory*). Para ambos os casos, o aluno deve considerar:

- A dimensão do filtro;
- A operção de convolução na periferia da imagem:
    - Píxeis fora da imagem replicam o valor de píxeis vizinhos;
    - Píxeis fora da imagem são deifnidos a zero.

Compare ambas as versões e comente o desempenho de execução temporal.

4. Implemente um programa CUDA que calcula a seguinte operação usando **matrizes quadradas 2D** de dimensão 8000x8000 ou superior (**compile usando a flag `--default-stream per-thread`**):

$$M3[i] = M1[i] + \sqrt{3.14159^{M2[i]}},\ onde\ todos\ os\ elementos\ de\ M1[i],\ M2[i] > 0.$$

a) Implemente uma versão CUDA simples (sem otimizações). Use as funções `pow()` e `sqrt()` no kernel CUDA desenvolvido.

b) Implemente uma versão usando *streams* CUDA, considerando para o efeito as funções `cudaStreamCreate()`, `cudaMemcpyAsync()`, e `cudaStreamDestroy()`. Note que é fundamental definir o número de *streams* e o correspondente particionamento da *workload*.

c) Considere os *loops* para lançamento das chamadas ao CUDA. É mais vantajoso lançar todas as cópias Host → Device e depois chamar o kernel CUDA, ou é melhor lançar memcopy → kernel → memcopy? Compare as duas abordagens e comente.

d) Use as funções `cudaMallocHost()` e `cudaFreeHost()` para substituir a alocação / desalocação de host memory da alínea c). É mais vantajoso lançar todas as cópias Host → Device e depois chamar o kernel CUDA, ou é melhor lançar memcopy → kernel → memcopy? Compare com a versão sem *streams*.