COMPUTAÇÃO GRÁFICA

Rodando CUDA

Prof. Érick Oliveira Rodrigues

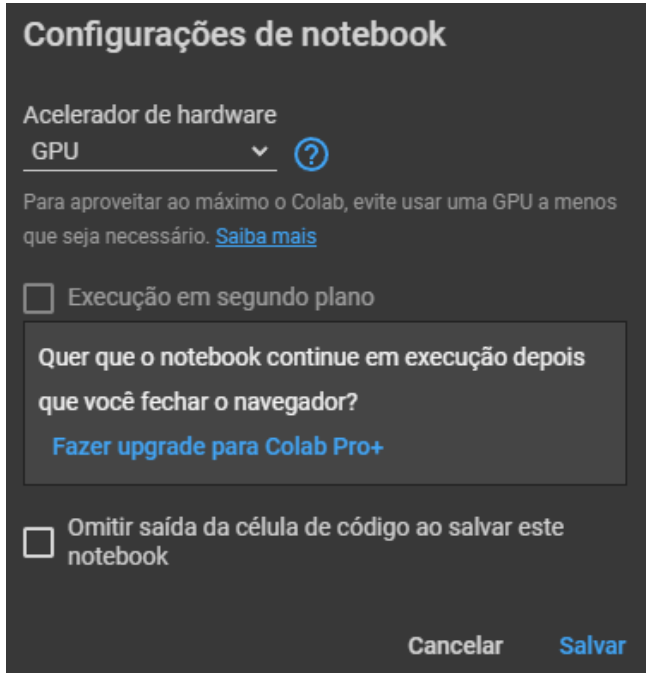
INSTALAÇÃO DO CUDA

- É possível instalar a linguagem CUDA no linux ou windows caso vocês tenham uma placa de vídeo da nvidia.
  - Não é possível instalar CUDA com uma placa da AMD. Embora a placa da AMD rode OpenCL, que é **bastante parecido** com CUDA. Não é difícil converter de um pro outro.

- Contudo, na disciplina, por ser mais fácil, vamos utilizar o Google Colab.
  - O colab é suficiente pra rodar as questões da disciplina.

PASSOS PARA INSTALAÇÃO

- Acesse o seguinte link:
  - https://colab.research.google.com/
- Clique em “novo notebook”.
- Vá em Editar → Configurações de Notebook e selecione GPU:



Esse passo é muito importante. Se for pulado, não vai funcionar!

PASSOS PARA INSTALAÇÃO

- Rode o seguinte comando para remover qualquer driver CUDA no notebook:
  - !apt-get --purge remove cuda nvidia* libnvidia-*
    !dpkg -l | grep cuda- | awk '{print $2}' | xargs -n1 dpkg --purge
    !apt-get remove cuda-*
    !apt autoremove
    !apt-get update

PASSOS PARA INSTALAÇÃO

- Rode o seguinte para instalar a versão 9 do CUDA:
  - !wget https://*developer.nvidia.com/compute/cuda/9.2/Prod/local_installers/cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64 -O cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.deb*
    !dpkg -i cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.deb
    !apt-key add /var/cuda-repo-9-2-local/7fa2af80.pub
    !apt-get update
    !apt-get install cuda-9.2

- Posteriormente, cheque a versão:
  - !nvcc --version

PASSOS PARA INSTALAÇÃO

- Instale o seguinte para compilar CUDA no notebook:
  - !git config --global url."https://".insteadOf git://
    !pip install git+git://*github.com/andreinechaev/nvcc4jupyter.git*

- Para carregar a extensão, rode:
  - %load_ext nvcc_plugin

- Pronto, baixe o seguinte código pra rodar no notebook (a resposta precisa ser 8):
  - MVD-DUT

    - Obs.: Não esqueça de adicionar *%%cu* antes do código

VECTOR ADD

- Para baixar uma modificação minha com comentários em português do exemplo da pasta “Samples” do CUDA, que realiza uma adição simples de vetores, utilize o seguinte código:
  - J2F-ZYV

SHARED MEMORY

- Exemplo com utilização de memória compartilhada:
  - WNZ-FOZ

DIRETIVAS

- O CUDA possui algumas diretivas, dentre as mais importantes, temos:

- __global__
  - Utilizada antes da função que será o kernel que é executado na GPU. Indica que a função pode ser disparada pela CPU.
- __device__
  - Utilizada antes da função. Indica que a função pode ser chamada por outra função da GPU.
- __shared__
  - Utilizado antes de alguma variável de memória. Indica que a memória alocada para aquela variável será da memória compartilhada.

CACHE DA HIERARQUIA DE MEMÓRIA

- Em basicamente qualquer sistema de computação, existe uma abordagem utilizada que pode te trazer problemas, principalmente programando em CUDA.

- Valores de memória são realocados em memórias mais rápidas, e a mudança na memória de nível mais global pode não ser instantânea.

- Qualquer operação que envolve memória global, por exemplo, pode não ser refletida automaticamente.
  - Para isso, por exemplo, existem as operações atômicas.

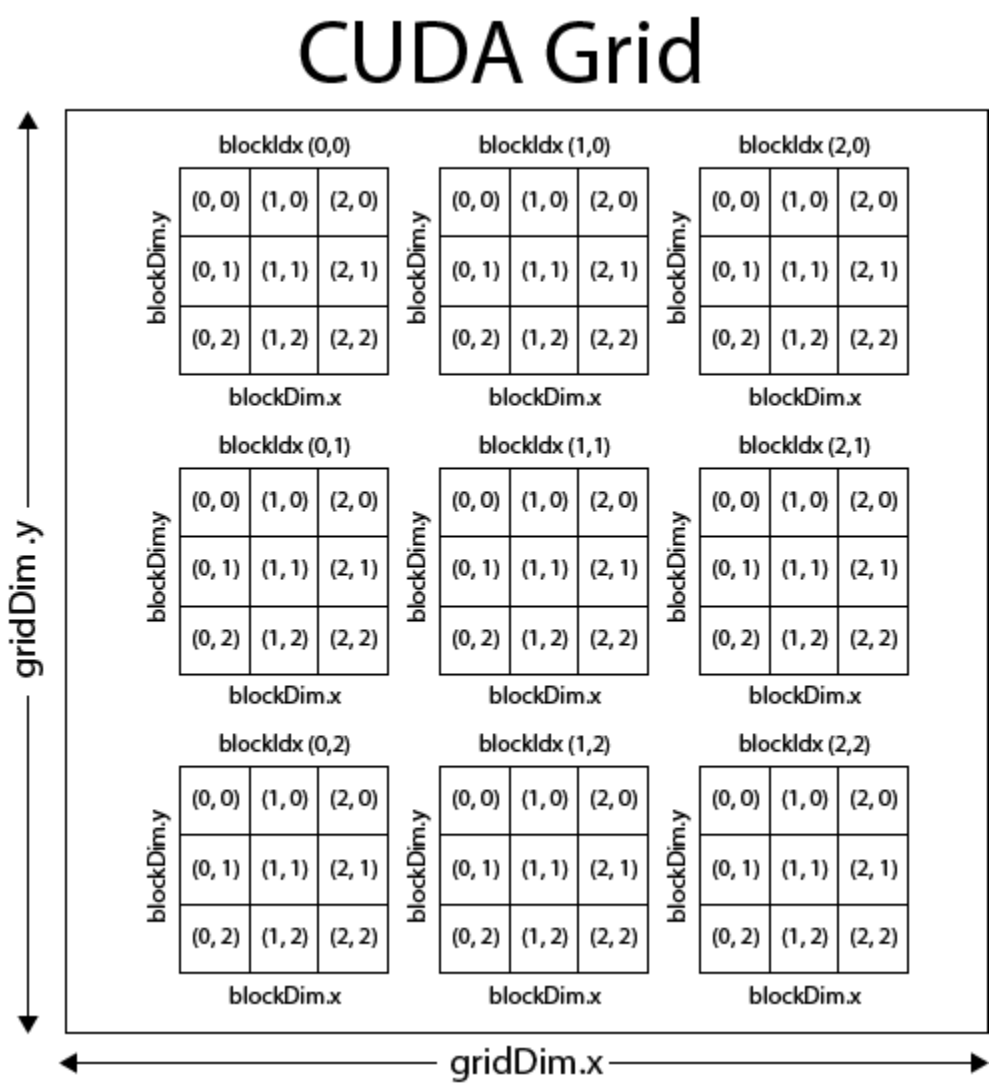
OUTROS DETALHES IMPORTANTES

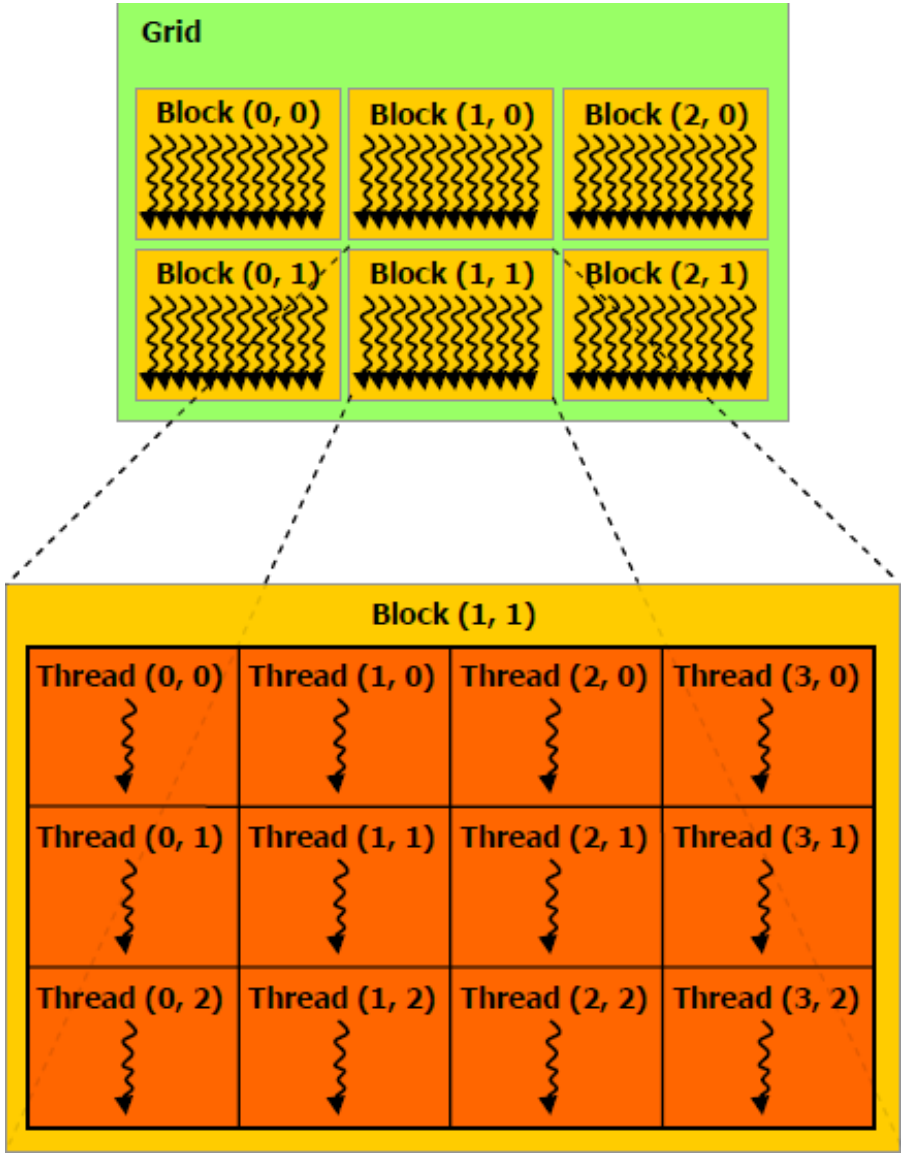
- Ainda, para uma boa utilização do CUDA, é necessário entender adequadamente a função:
  - __syncthreads();

- Ainda, na solução de alguns problemas, pode ser interessante entender as operações atômicas:
  - https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#atomic-functions

- É possível passar matrizes de 2 e 3 dimensões para um kernel em CUDA. O que pode ser mais viável na solução de alguns problemas.
  - Contudo, não veremos com detalhes aqui na disciplina porque pode complicar mais o entendimento, além de não ser necessário, já que qualquer matriz de 3 ou 2 dimensões pode ser convertida para um vetor.

EXEMPLO COM MATRIZ 2D





OUTROS DETALHES IMPORTANTES

- Ainda, memórias de textura podem ser mais eficientes na aplicação de alguns filtros 2D. Contudo, no geral, é melhor utilizar a memória global padrão.

- Sempre que possível, utilize a palavra constant, já que a memória constante vai ser mais rápida do que as outras, por conta dos caches que são feitos (isso é verdade basicamente em qualquer linguagem de programação e situação).

- Memória constante pode ser declarada da seguinte forma:
  - __device__ __constant__ **int** constNumber[4] = {1,2,3,4};
  - O __device__ caso pode ser suprimido em alguns casos.