CUDA

Programação Paralela

Mateus Zarth Seixas <mateus_seixas@hotmail.com.br>

Orientador: Marco Reis

Robótica e Sistemas Autônomos, Senai Cimatec

Novembro de 2021

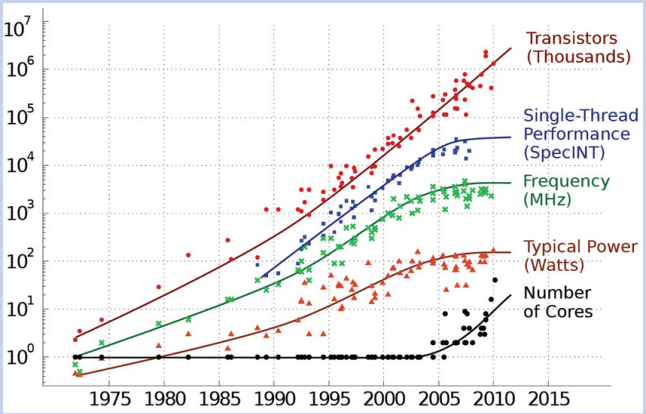
Sistema FIEB | SENAI CIMATEC

PELO FUTURO DA INOVAÇÃO

Lei de Moore

"A densidade de transistores em um chip dobra a cada 18 meses mantendo o mesmo custo de fabricação."
Gordon E. Moore, 1965

Lei de Moore



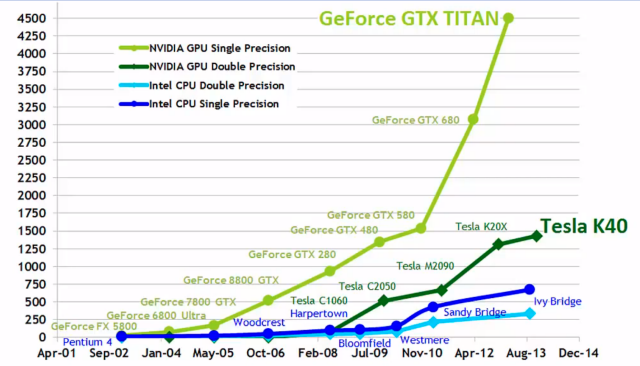
- O número de transistores por chip cresceu em escala logarítimica
- O aumento no número de transistores parou de refletir no aumento da performance
- O consumo energético se tornou muito alto

"Se um único computador (processador) consegue resolver um problema N segundos, podem N computadores (processadores) resolver o mesmo problema em 1 segundo?"

Programação em paralelo

Porquê usar GPUs?

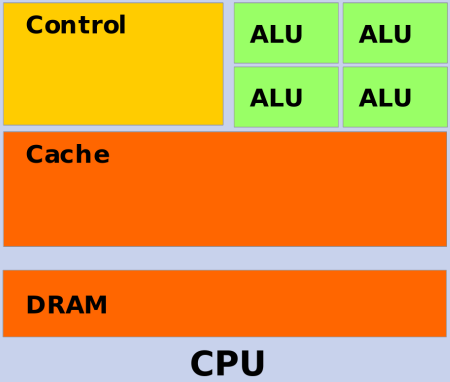
Performance das GPUs vs CPUs em GFLOPS/s

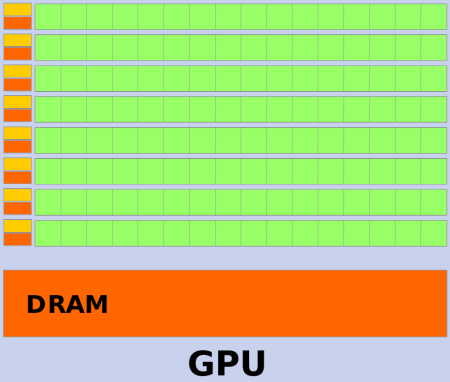


- Poder computacional muito superior das GPUs
- Reduzir o tempo de solução de um problema
- Resolver problemas mais complexos

CPU x GPU

Qual a diferença?

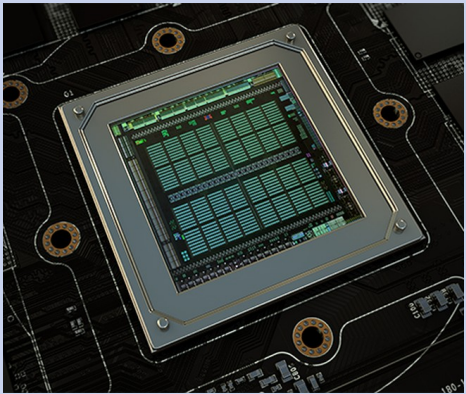




O que é CUDA?

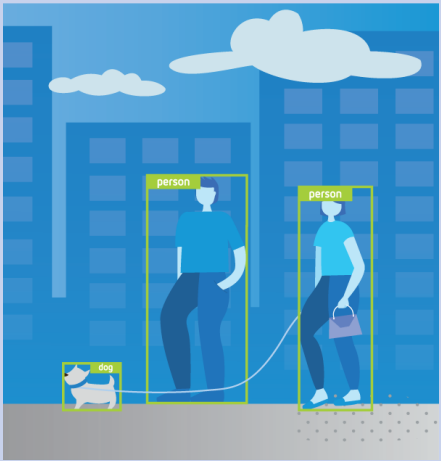
Compute Unified Device Architecture

- É um modelo de programação em paralelo que permite o uso das GPUs da Nvidia
- É basicamente C/C++ com algumas extensões
- Pode ser usado em outras linguagens como Fortran e Python



Aplicações

- Processamento de imagens
- Simulações
- Cálculos vetoriais e matriciais
- Algoritimos de buscas
- Química computacional
- Ordenação
- Inteligência computacional
- Deep learning



Nvidia G80

GeForce 8800

Conceitos Iniciais

Kernel
↪ É o código que é executado na GPU

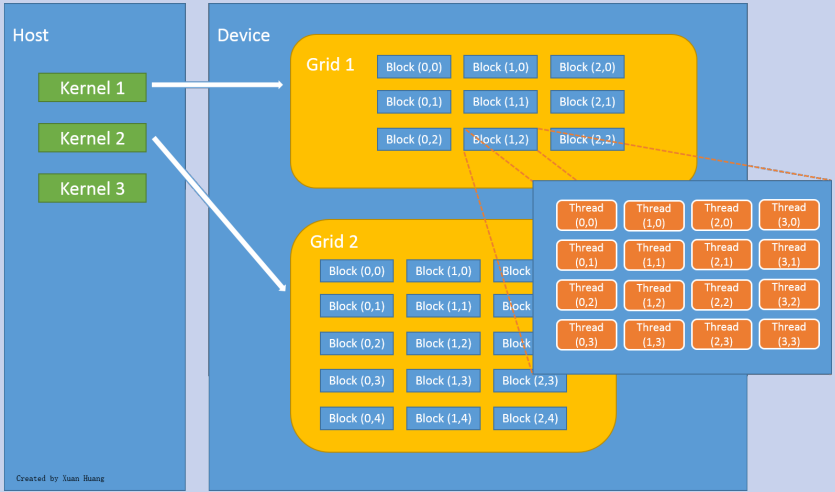
Thread
↪ É a execução do Kernel em um único pedaço dos dados que é executada em um CUDA Core

Thread Block
↪ É um conjunto de Threads executado em uma Streaming Multiprocessor (SM)

Grid
↪É um conjunto de Blocks que é executada em toda GPU apartir do launch do Kernel

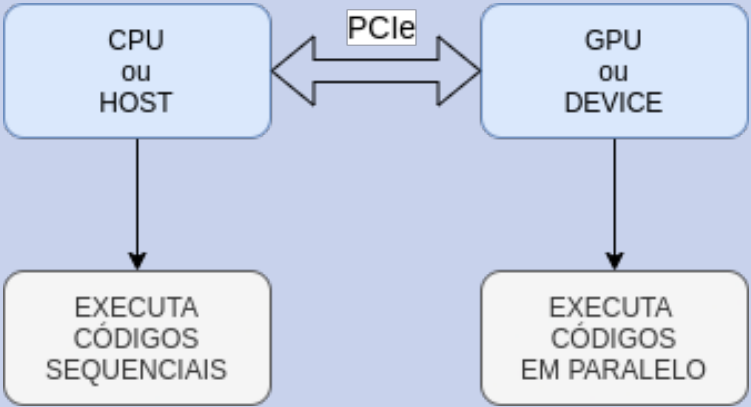
Arquitetura CUDA

Arquitetura CUDA



Programação Heterogênea

Host + Device



Block Index e Thread Index

ThreadIdx → Index da Thread

Cada thread tem um ID único no Block. Em um Block 3D, tem 3 componentes:

ThreadIdx.x

ThreadIdx.y

ThreadIdx.z

BlockIdx → Index do Block

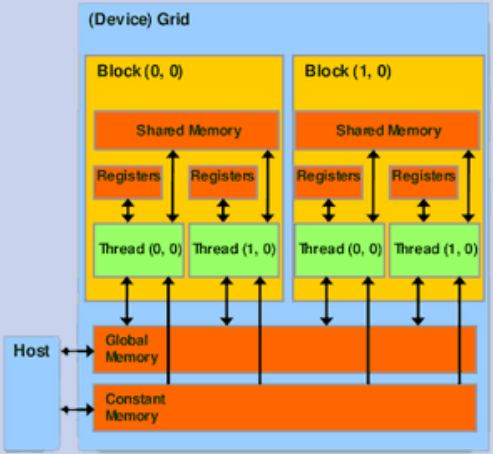
Cada Block tem um ID único no Grid. Em um Grid 3D, tem 3 componentes:

BlockIdx.x

BlockIdx.y

BlockIdx.z

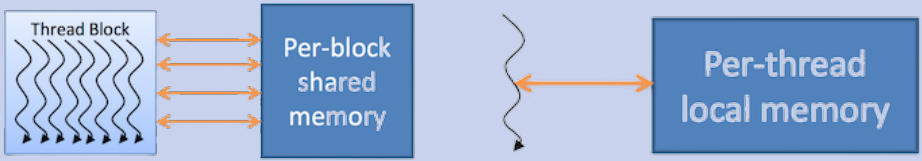
Modelo de Memória



- O host se comunica com o device através da memória global
- Apenas threads de um mesmo Block podem se comunicar e cooperar através da memória compartilhada
- Therds de blocks diferentes não se comunicam
- Cada thread possui um conjunto de registradores e uma memória local

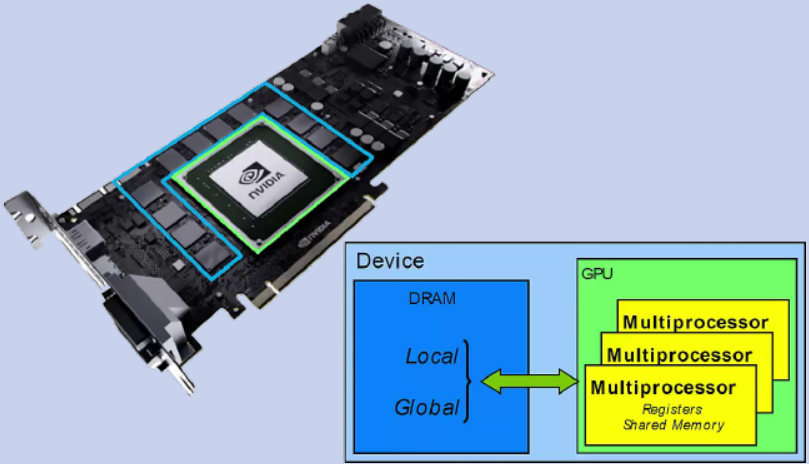
Modelo de Memória

Cooperação em Thread Blocks



- Threads podem compartilhar resultados entre si ou cooperar para produzir um resultado único através da memória compartilhada
- Threads podem se sincronizar umas com as outras
- A memória local é privada para cada thread
- A memória compartilhada é mais rápida que a memória global e a local

Modelo de memória



Kernel Mínimo

```
__global__void mykernel(...){
}
```

A keyword __global__ indica uma função que é executada no device. O compilador nvcc separa o código em componentes de host e componentes de device. Os componentes de device são compilados com o nvcc e os componentes de host por compiladores padrão como gcc.

Alocação de Memória

- cudaMalloc() → Aloca espaço para cópias no Device
- cudaMemcpy() → Copia entradas do Host pro Device ou do Device pro Host
- cudaFree() → Desaloca memória no Device

Alocação de Memória

cudaMalloc e cudaFree

cudaMalloc(LOCATION, SIZE);
cudaMalloc((void **)&d_a, sizeof(int));

- O primeiro argumento é a localização da memória no device para alocar dados
- O segundo argumento é o tamanho dos dados em bytes

cudaFree(d_a);→ Desaloca memória

Alocação de Memória

cudaMemcpy

```
cudaMemcpy(dst,src,size,direction);
cudaMemcpy(d_a, a, size,cudaMemcpyHostToDevice);
```

- O primeiro e o segundos argumento são os pointers do endereço que vai receber a cópia e do que vai enviar a cópia, respectivamente
- O terceiro argumento é o tamanho em bytes
- O quarto elemento é a direção, podendo ser:
  - ↪ cudaMemcpyHostToDevice
  - ↪ cudaMemcpyDeviceToHost

Launching do Kernel

```
dim3 grid_size(x,y,z);
dim3 block_size(x,y,z);
kernel<<<grid_size, block_size>>>(...);
```

O primeiro parâmetro e o segundo parâmetro se referem ao tamanho e a configuração do Grid e dos Blocks. Ambos parâmetros podem ser 1D, 2D ou 3D.

Instalando CUDA no Ubuntu 20.04

```
$ sudo apt update
$ sudo apt install build-essential
$ sudo apt install nvidia-cuda-toolkit
```

O segundo comando instala o compilador GCC, GNU Compiler Collection, o compilador de códigos em C/C++ e o terceiro comando instala o compilador NVCC, Nvidia CUDA Compiler.

Compilando Códigos e Executando

```
$ nvcc -o hello hello.cu
$ ./hello
```

O primeiro comando compila o código hello.cu e o segundo o executa o código binário compilado hello.

References (1)

Questions?

marco.a.reis@google.com