Aula 13
Paralelismo em GPU

Supercomputação

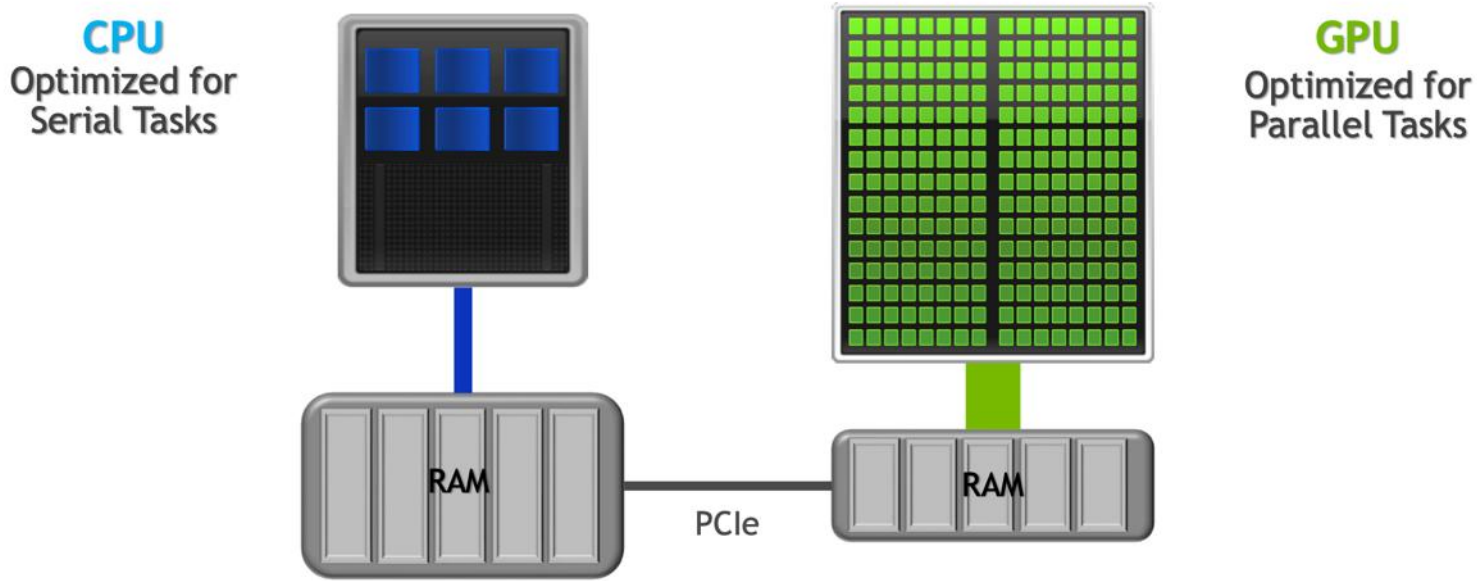
Prof. Lícia Sales Costa Lima

Na aula de hoje...

- Diferenciar dispositivos de latência (CPUs) e de throughput (GPUs)
- Compreender o layout de memória e transferência de dados em sistemas heterogêneos (CPU ⇔ GPU)
- Compilar primeiros programas em CUDA

Material adaptado do Deep Learning Institute NVIDIA e do NVIDIA Teaching Kit — Accelerated Computing

CPUs e GPUs


CPU
Optimized for
Serial Tasks
GPU
Optimized for
Parallel Tasks
RAM
PCIe
RAM

CPU vs GPU

A CPU

Rápida e ágil para tarefas individuais.

Carrega poucos por vez, com mínima.

A GPU (ÔNIBUS)

Carrega um grande volume de dados.

Focada em vazão (throughput).

A CPU é otimizada otimizada para latência

Execução rápida de de uma tarefa

GPU é otimizada para vazão

Execução de milhares de tarefas tarefas ao mesmo mesmo tempo

Speed vs Throughput

Speed

Throughput

Which is better depends on your needs...

Moto vs Ônibus

QUANDO A CPU VENCE

Cenário: Dois passageiros precisam atravessar a cidade o mais rápido possível.

- **Baixa Latência:** Ideal para tarefas que dependem do resultado anterior.
- **Lógica Complexa:** Algoritmos com muitos desvios (if/else) e recursão.
- **Acesso Aleatório:** Manipulação de estruturas de dados irregulares.
- **Exemplo:** Compiladores, sistemas operacionais, gerenciamento de interfaces

QUANDO A GPU VENCE

Cenário: 1000 passageiros precisam ir para o mesmo estádio ao mesmo tempo.

- **Alta Vazão (Throughput):** Ideal para processar milhões de dados independentes.
- **Paralelismo Massivo:** A mesma operação aplicada a diferentes dados (SIMD).
- **Cálculo Intensivo:** Operações matemáticas repetitivas e previsíveis.
- **Exemplo:** Processamento de imagem, IA, simulações físicas, criptografia.

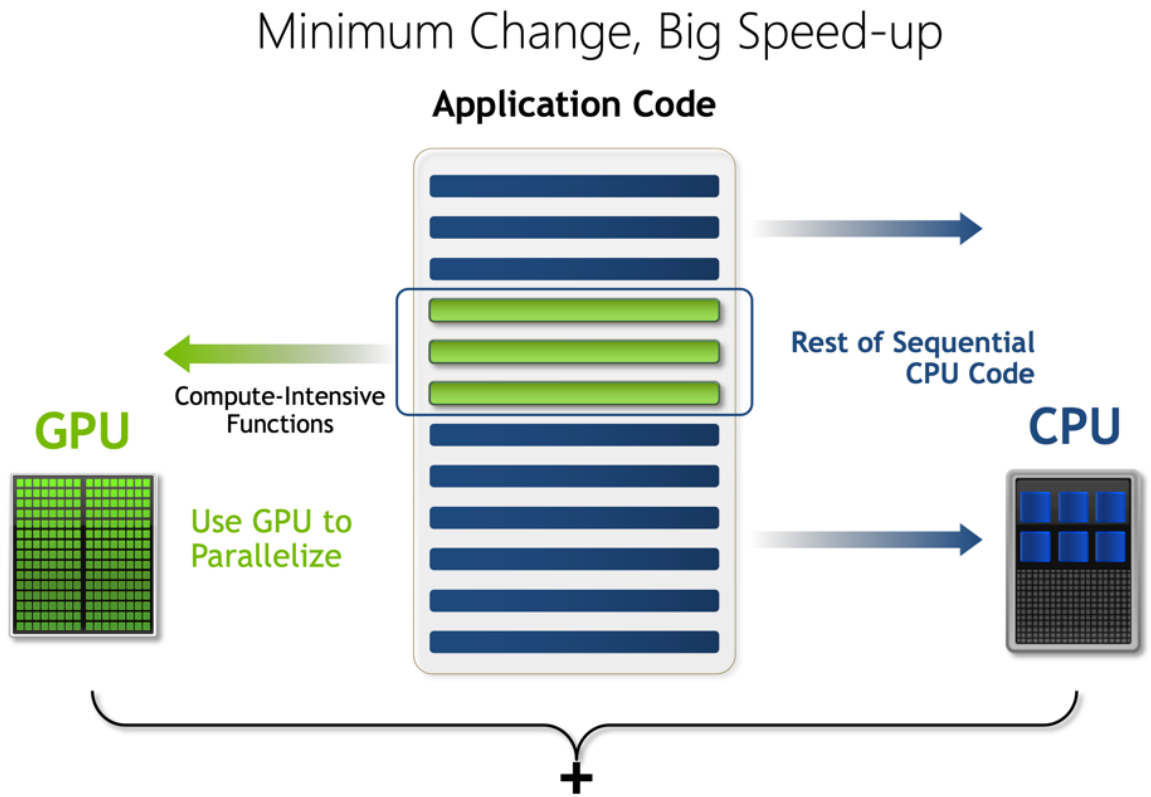
"A eficiência não depende apenas da velocidade, mas de escolher o veículo certo para a carga."

CPU vs GPU

- CPUs para partes sequenciais onde uma latência mínima é importante
  - CPUs podem ser 10X mais rápidas que GPUs para código sequencial

- GPUs para partes paralelas onde a taxa de transferência(throughput) bate a latência menor.
  - GPUs podem ser 10X mais rápidas que as CPUs para código paralelo

No fim das contas...



O que é CUDA?

CUDA (Compute Unified Device
modelo de programação paralela
pela NVIDIA que permite usar o poder das
para acelerar aplicações de alto

Principais características

- Executa **milhares de threads simultaneamente** na GPU
- Desenvolvido pela NVIDIA para suas
- Amplamente utilizado em HPC, IA e simulações científicas
- Integrado a clusters como **Franky** e **SDumont** via SLURM

"

A GPU não substitui substitui a CPU — —ela a complementa, assumindo tarefas tarefas paralelas.

Em um Cluster HPC com SLURM

Em sistemas HPC com SLURM, a execução de programas CUDA segue três etapas principais:

Etapa	Ação	Ferramenta
1	Carregar o módulo CUDA disponível no cluster	`module load cuda/12.8.1`
2	Compilar o código com o compilador NVIDIA	`nvcc`
3	Executar o binário solicitando uma GPU	`srun` ou `sbatch` com `--gres=gpu:1`

Preparando o Ambiente

```
# Listar módulos disponíveis
module avail cuda

# Carregar o módulo CUDA
module load cuda/12.8.1

# Verificar o compilador
nvcc –version
```

Configurar o ambiente CUDA cluster é o primeiro passo.

Após o carregamento, o o compilador **nvcc** estará estará disponível para compilar arquivos **.cu** (código CUDA).

Programando em CUDA

- Compilador especial: nvcc
- Endereçamento de memória separado
- Dados precisam ser copiados CPU <-> GPU
- Funções especiais (kernels) para trabalhar na GPU

Kernel CUDA

O kernel CUDA é onde a computação paralela em GPU acontece.

```
__global__ void add(int n, float *x, float *y) {
 int i = blockIdx.x * blockDim.x + threadIdx.x;

 if (i < n) y[i] = x[i] + y[i];
// cada thread processa UM elemento
}
```

Conceitos-chave

- **__global__**: função executada na GPU, chamada pela pela CPU
- **blockIdx.x** e **threadIdx.x**: identificam cada thread unicamente
- **blockDim.x**: número de threads por bloco
- Chamada do kernal:
  add<<<numBlocks, blockSize>>>(N, d_x, d_y)

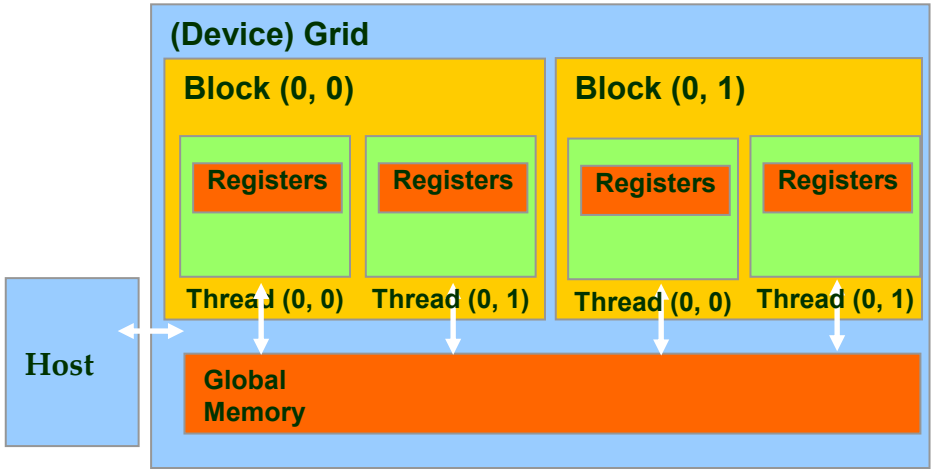
Memória em GPUs

Código da GPU (device) pode:

- Cada thread ler e escrever nos registradores
- Ler e escrever na memória global

Código da CPU (host) pode:

- Transferir dados de e para memória global

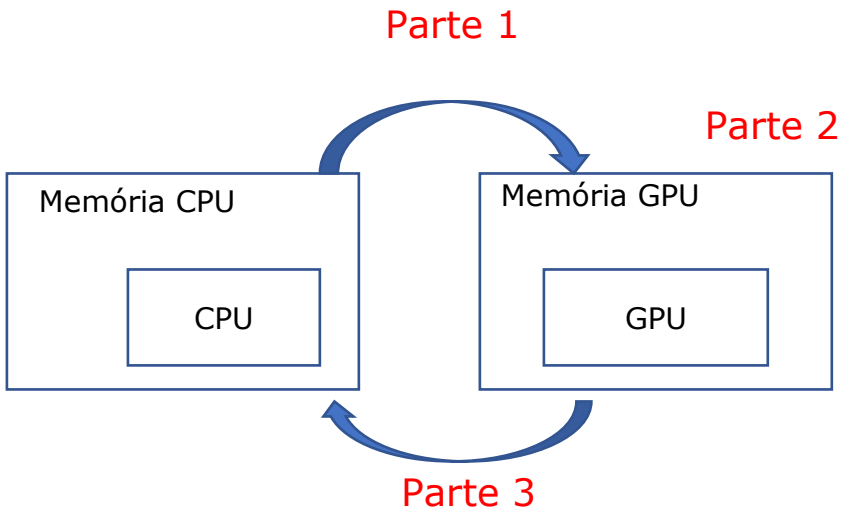


Fluxo dos programas

Parte 1: copia dados CPU → GPU

Parte 2: processa dados na GPU

Parte 3: copia resultados GPU → CPU




VAMOS VER
SE VOCÊ MANJA DOS NEGÓCIOS