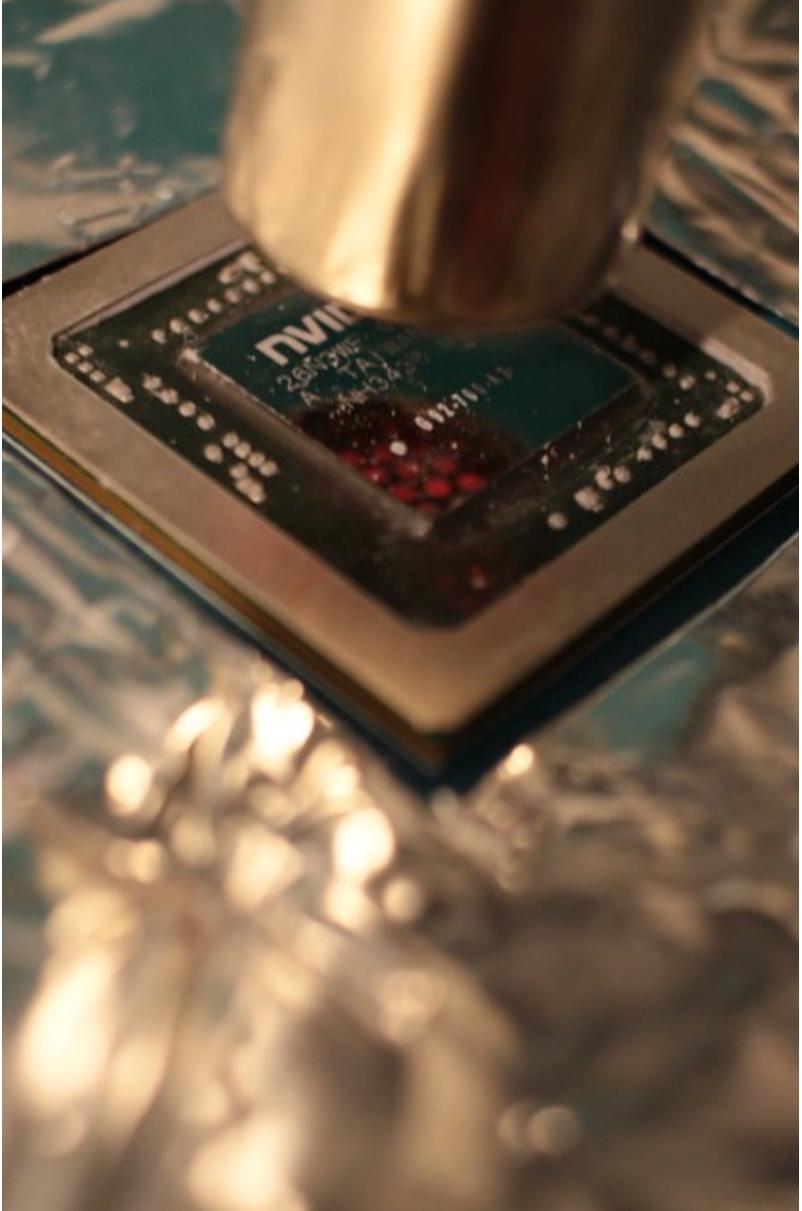




Insper Supercomputação



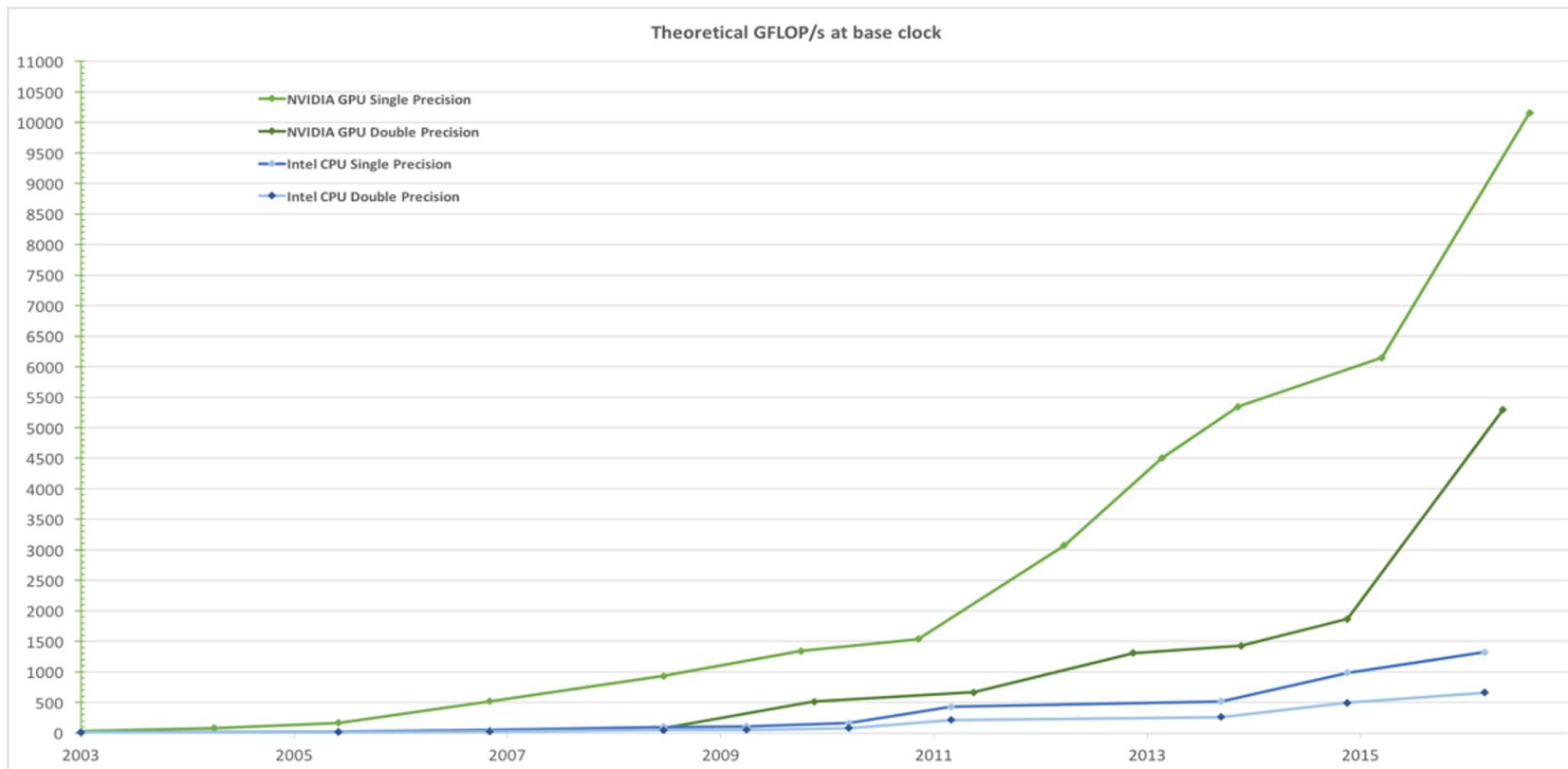
Aula - 13

- Introdução a GPU

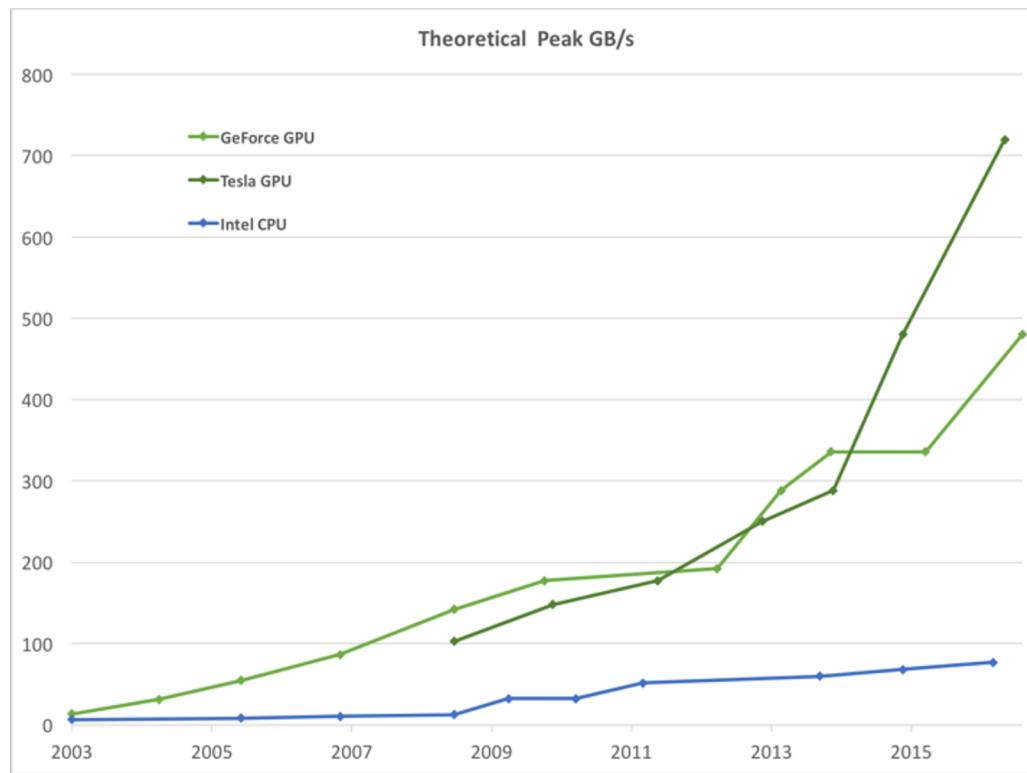
Nossos objetivos

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

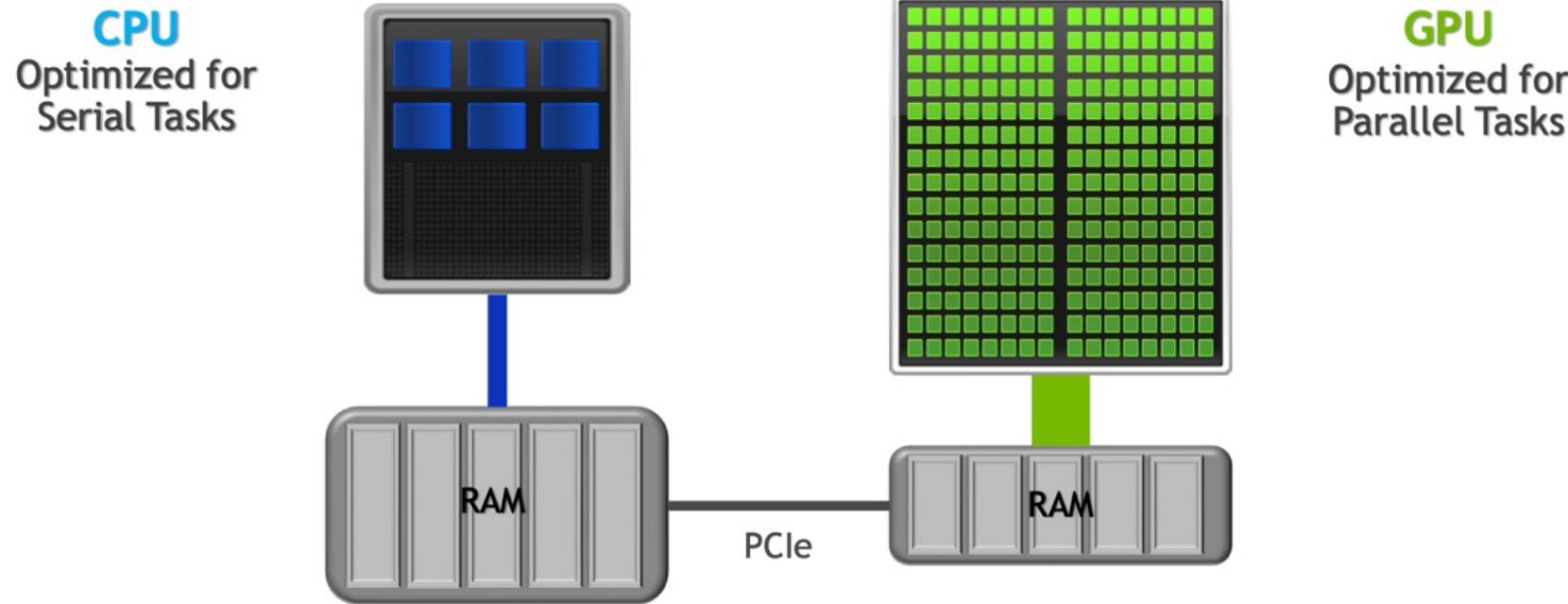
Desempenho em GFLOPS



Desempenho em GB/s



CPUs e GPUs



Speed vs Throughput

Speed



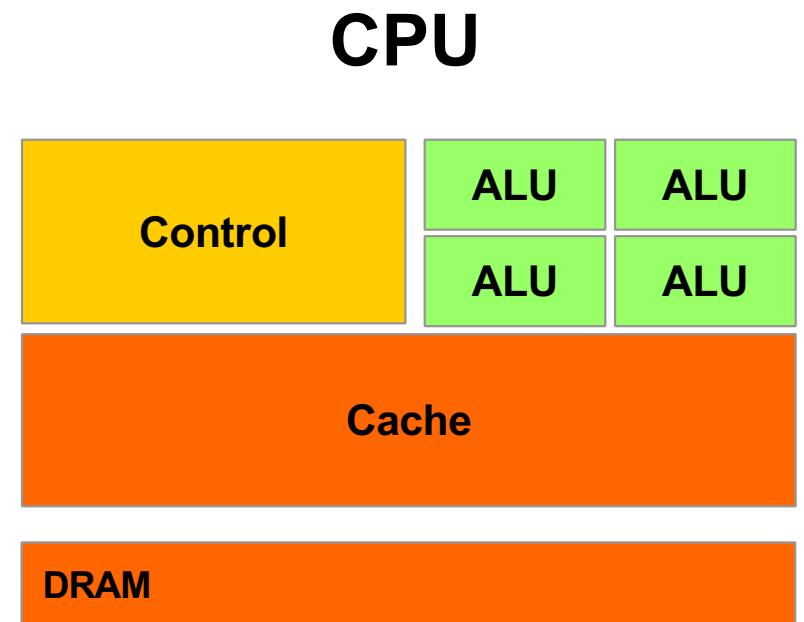
Throughput



Which is better depends on your needs...

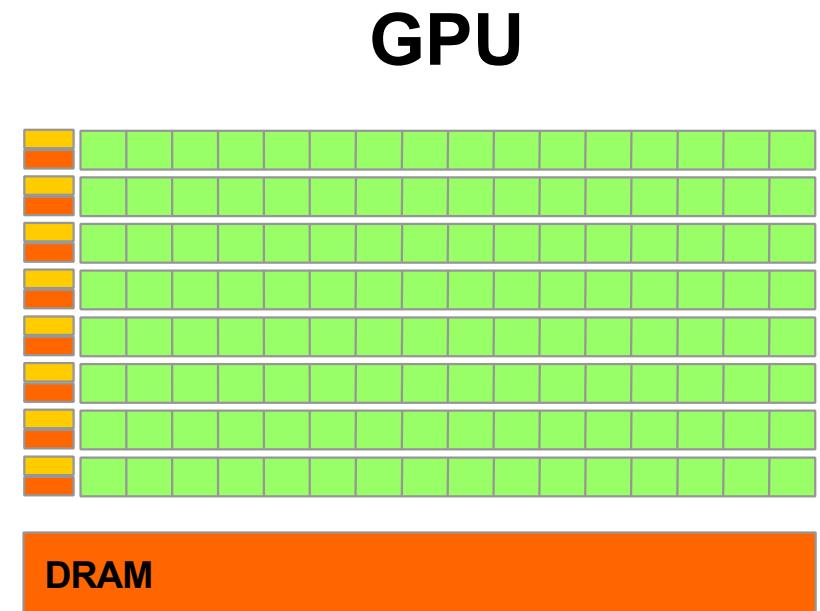
A CPU minimiza latência

- ULA potente, minimiza latência das operações
- Cache grande:
 - Acelera operações lentas de acesso à RAM
 - Mas cache-misses são custosos
 - Baixa performance / watt



A GPU maximiza throughput

- ULA simples
 - Eficiente energeticamente
 - Alta taxa de transferência
- Cache pequeno
 - Acesso contínuo a RAM
- Controle simples
- Número massivo de threads



CPU vs GPU

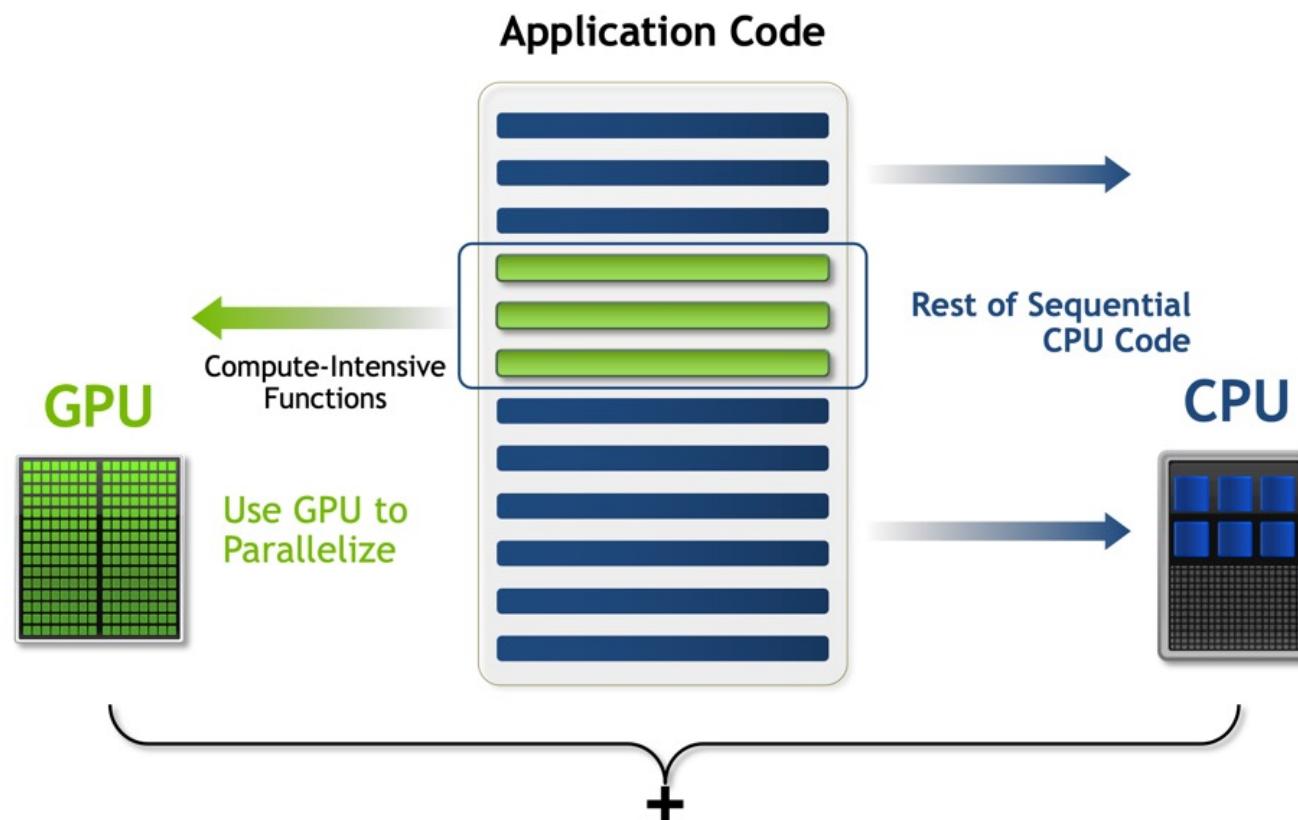
- CPUs para partes sequenciais onde uma latência mínima é importante
 - CPUs podem ser 10X mais rápidas que GPUs para código sequencial



- 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

CPU vs GPU

Minimum Change, Big Speed-up



Como usar a GPU?

Aplicações

Bibliotecas

Fácil de Usar
Alto Desempenho

Diretivas de
Compilação

Simples de Usar
Portabilidade de Código

Linguagens de
Programação

Maior Desempenho
Maior Flexibilidade

Bibliotecas aceleradas por GPU

Álgebra Linear
FFT, BLAS,
SPARSE, Matrix



cULA|tools

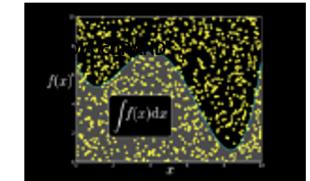


C U S P

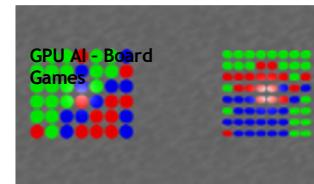
Numéricas/Math
RAND, Estatísticas



ArrayFire



Estrutura de Dados/IA
Sort, Scan, Zero Sum



Processamento Visual
Imagem & Video



Sundog™
Software

Bibliotecas aceleradas por GPU

- Facilidade de uso: O uso de bibliotecas permite a aceleração da GPU sem conhecimento aprofundado da programação da GPU
- Simplicidade: Muitas bibliotecas aceleradas por GPU seguem APIs padrão, permitindo aceleração com mudanças mínimas de código
- Qualidade: As bibliotecas oferecem implementações de alta qualidade de funções encontradas em uma ampla variedade de aplicativos

Linguagens de Programação

Numerical analytics ➤

MATLAB, Mathematica, LabVIEW

Fortran ➤

CUDA Fortran

C ➤

CUDA C

C++ ➤

CUDA C++

Python ➤

PyCUDA, Copperhead, Numba

Programando para GPU

- Compilador especial: nvcc
- Endereçamento de memória separado
 - Dados precisam ser copiados de/para GPU
 - Isto leva tempo
- Funções especiais (kernels) para rodar na GPU

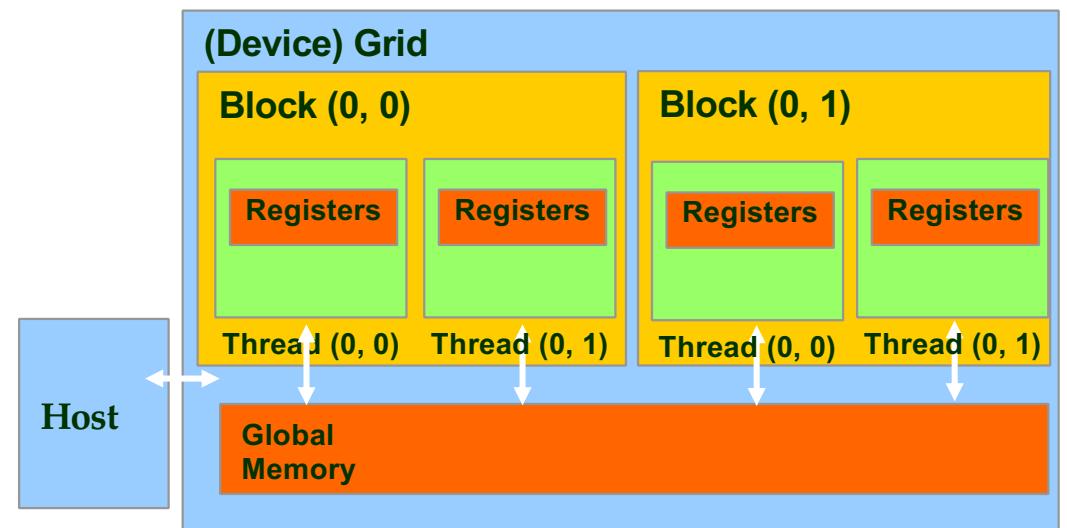
Memória em GPUs

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

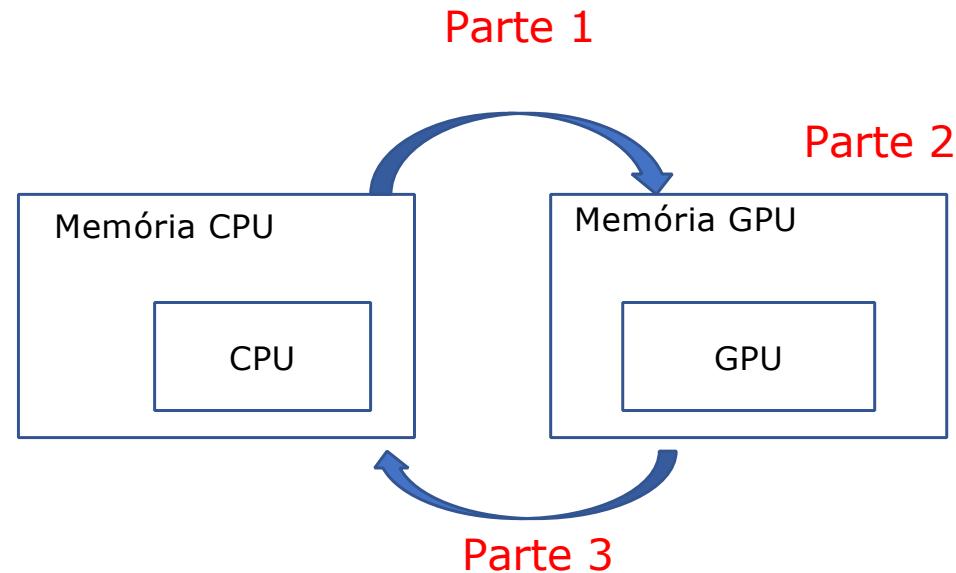


Fluxo dos programas

Parte 1: copia dados CPU → GPU

Parte 2: processa dados na GPU

Parte 3: copia resultados GPU → CPU



Biblioteca que usaremos - Thrust



Importante: infra

Se você tem uma GPU:

- pode usá-la diretamente na disciplina;
- instale o pacote nvidia-cuda-toolkit e os drivers compatíveis

Se você não tem GPU:

- compile código usando thrust/OpenMP
- solicite a conversão da sua VM para GPU com o Tiago (tiagoaodc@insper.edu.br)

A partir da próxima semana vamos supor que todos já tem acesso a uma GPU com compilador funcionando.

Nvidia Thrust

Vantagens:

- **Simplifica transferências de memória**
- Duas operações customizáveis (*reduce*, *transform*)
- Suporta OpenMP e CUDA

Desvantagens:

- Limitado: menos recursos e desempenho que CUDA C
- Só tem dois tipos de operações
- Baseado em *templates* – *difícil de debugar erros de compilação*

Nvidia Thrust – tipos de dados

Apenas dois tipos

`thrust::device_vector<T>`

- Vetor genérico de dados na GPU
- Automaticamente alocado e desalocado
- Cópia é feita usando atribuição

`thrust::host_vector<T>`

- Vetor genérico de dados na CPU
- Pode ser substituído em vários lugares por containers da STL ou ponteiros “normais”

Thrust - Exemplo

```
thrust::host_vector<double> vec_cpu(10); // alocado na CPU  
  
vec1[0] = 20;  
vec2[1] = 30;  
  
thrust::host_vector<double> vec_gpu (10); // alocado na GPU  
  
vec_gpu = vec_cpu; // copia o conteúdo da CPU para GPU  
thrust::device_vector<double> vec2_gpu (vec_cpu); // também transfere para GPU
```

Thrust - Iteradores

Funcionam igual aos iteradores de std::vector

```
v.begin() // primeiro elemento
```

```
v.end() // último elemento
```

```
v.begin()+2 // v[2]
```

```
i = v.begin() + 3; *i = 4; // v[3] = 4
```

Thrust - Exemplo

```
thrust::device_vector<int> v(5, 0); // vetor de 5 posições zerado
// v = {0, 0, 0, 0, 0}
thrust::sequence(v.begin(), v.end()); // inicializa com 0, 1, 2, ...
// v = {0, 1, 2, 3, 4}
thrust::fill(v.begin(), v.begin() + 2, 13); // dois primeiros elementos = 3
// v = {13, 13, 2, 3, 4}
```

Thrust – Redução

Resume o vetor para um escalar

- Soma todos elementos
- Máximo/mínimo do vetor
- Contagens
- Suporta iteradores, o que torna a operação bastante flexível.

Thrust - Exemplo

```
val = thrust::reduce(iter_comeco, iter_fim, inicial, op);
// iter_comeco: iterador para o começo dos dados
// iter_fim: iterador para o fim dos dados
// inicial: valor inicial
// op: operação a ser feita.
```

Thrust - Transformação

Operações elemento a elemento entre pares de vetores ou um só vetor.

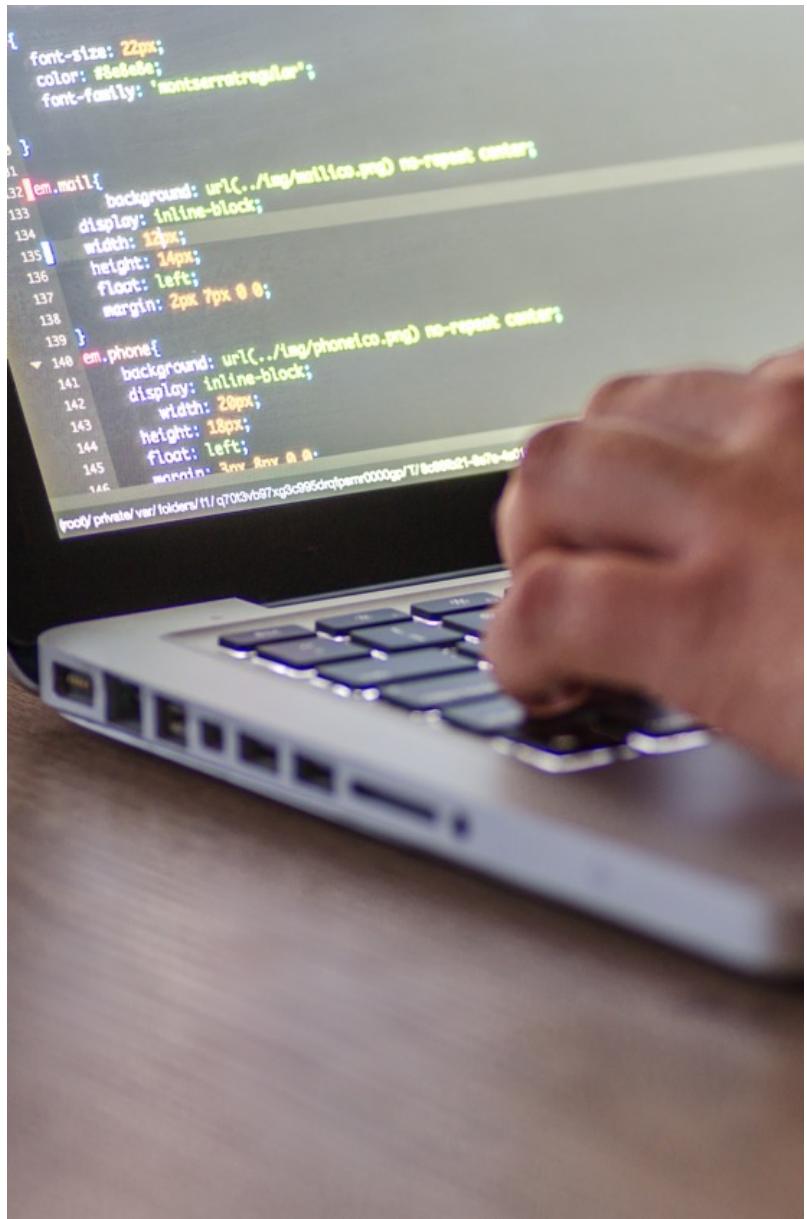
- Aritmética ponto a ponto
- Permite criação de operações customizadas
- Suporta iteradores de entrada e saída
- Funciona também para operações locais (imagens)

Thrust - Exemplo

```
thrust::device_vector<double> V1(10, 0);
thrust::device_vector<double> V2(10, 0);
thrust::device_vector<double> V3(10, 0);
thrust::device_vector<double> V4(10, 0);
// inicializa V1 e V2 aqui

// soma V1 e V2
thrust::transform(V1.begin(), V1.end(), V2.begin(), V3.begin(), thrust::plus<double>());

// multiplica V1 por 0.5
thrust::transform(V1.begin(), V1.end(),
                 thrust::constant_iterator<double>(0.5),
                 V4.begin(), thrust::multiplies<double>());
```



Thrust - Roteiro

Iniciando com Thrust

- Alocação de memória em CPU e GPU
- Utilização das operações de redução e transformação
- Operações elemento a elemento disponíveis na thrust