Insper

SuperComputação

Aula 18 – Iteradores especiais e operações customizadas

2021 – Engenharia

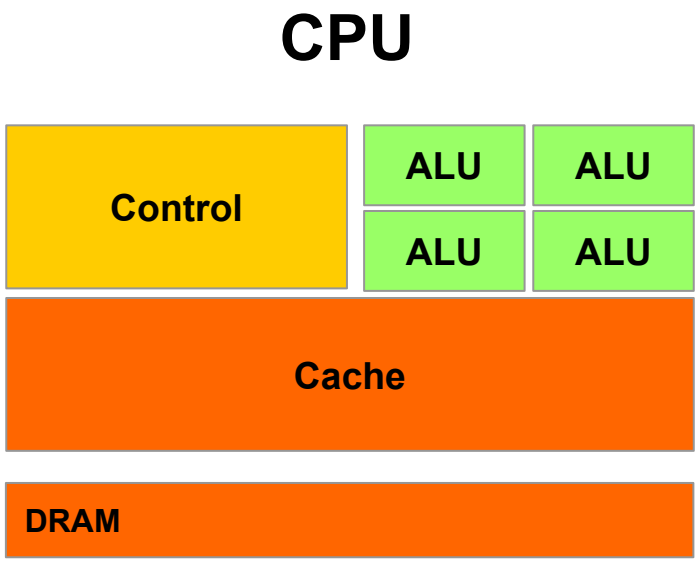
Igor Montagner <igorsm1@insper.edu.br>
Antônio Selvatici <antoniohps1@insper.edu.br>

Objetivos de aprendizagem

- Evitar alocação de memória usando iteradores especiais
- Rodar pequenas funções C++ em GPU

CPU minimiza latência

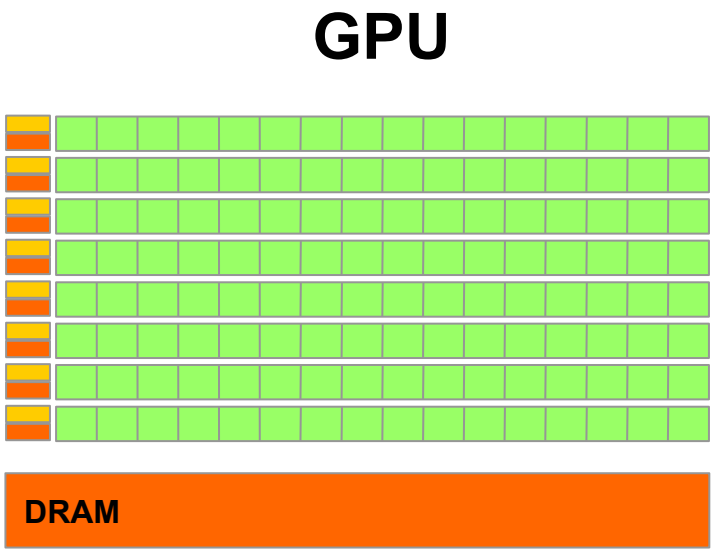
- ALU pontente minimiza latência das operações
- Cache grande:
  - Acelera operações lentas de acesso a RAM
- Controle sofisticado:
  - Branch prediction
  - Data forwarding



Insper

GPU minimiza throughput

- ALU 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

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

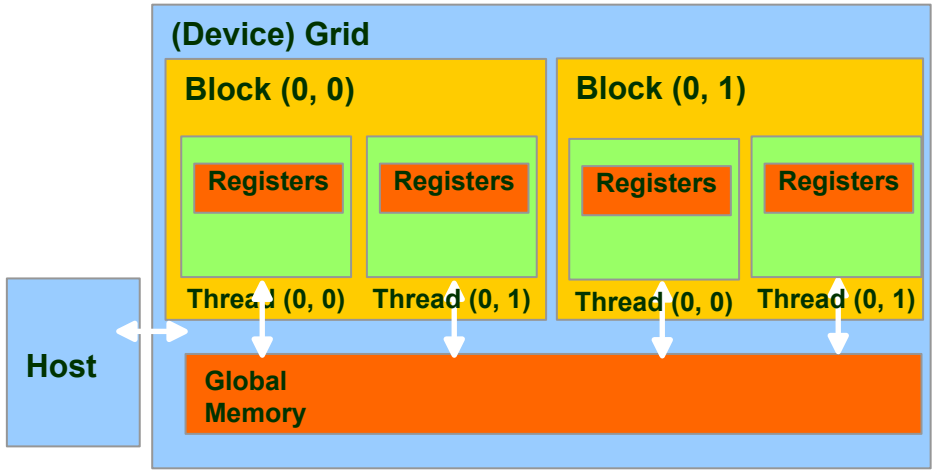
Memória em GPU

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



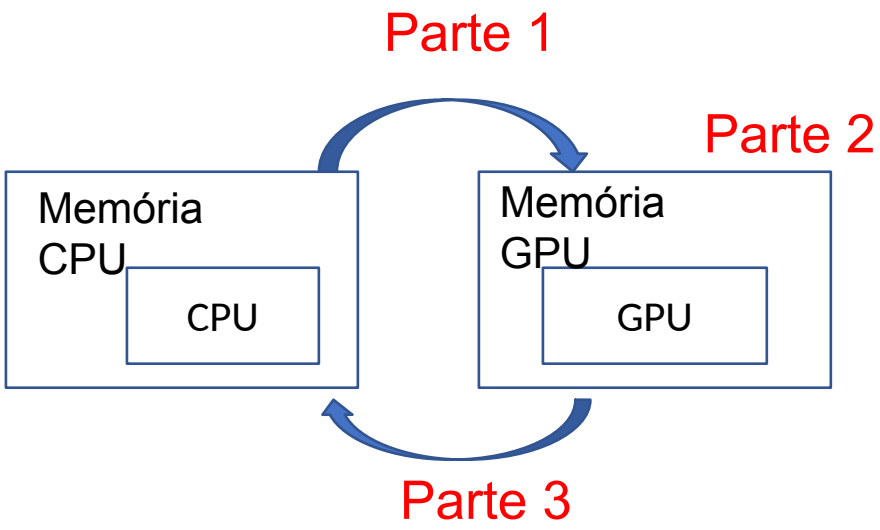
Insper

Fluxo de um programa

Parte 1: copia dados CPU → GPU

Parte 2: processa dados na GPU

Parte 3: copia resultados GPU → CPU



Insper

Programando para GPU - hoje

Nvidia Thrust: API simplificada em C++

Programando para GPU - 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 via formulário https://forms.office.com/r/zrNsetQdAH

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”

Nvidia Thrust – redução

```
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.
```

Nvidia Thrust – transformação

```
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>());
```

Memória em GPU

- Operação `transform` trabalha com vetores
- Alocamos dados na memória da GPU
- Às vezes não precisamos ter TUDO na memória
  - constantes
  - dados gerados automaticamente (range, contador)

Iteradores especiais

- Representam vetores que podem ser computados automaticamente
  - `thrust::counting_iterator<int> iter`
  - `thrust::constant_iterator`
- Ao invés de criar um vetor que ocupará GPU-RAM, criamos um iterador, que não ocupa GPU-RAM.

Atividade prática

Exercício Desvio Padrão (30 minutos)

1. Revisão de transform + reduce
2. Criar iterador para evitar consumir memória com dados estáticos

Operações customizáveis

- Podemos criar operações para usar em `reduce` e `transform`
- Criamos um struct com um método que pode ser compilado para GPU
- Anotações indicam onde o código roda
  - `__host__` é código CPU
  - `__device__` é código GPU

Atividade prática

Operações de contagem customizadas (30 minutos)

1. Reduções customizáveis
2. Compilação de código C++ para GPU

www.insper.edu.br

Nvidia 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
```

Nvidia Thrust – iteradores

```
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}
```


Insper