Estruturas de Dados e Computação Assíncrona em GPU

Otimizando o Desempenho de Algoritmos na GPU

Inspe

Objetivos da Aula

- Compreender o impacto das **estruturas de dados** no desempenho da GPU.
- Aprender estratégias de **execução assíncrona** para otimização.
- Dominar o formato **CSR** para matrizes esparsas.
- Implementar **CUDA Streams** para sobreposição de operações.

O Desafio das Matrizes Esparsas

Definição: Matrizes onde a grande maioria dos elementos possui valor zero.

O Problema: Armazenar todos os zeros é um desperdício massivo de memória e largura de banda.

Eficiência: Processar zeros consome ciclos de CPU/GPU sem gerar resultados úteis.

Robustez: Estratégias adequadas evitam erros comuns, como divisões por zero em algoritmos complexos.

99%

de elementos nulos em uma matriz típica de 1.000 x 1.000 com 10k valores úteis.

Formato CSR (Compressed Sparse Row)

O formato mais popular em ambientes de **High Performance Computing (HPC)**, especialmente para operações de multiplicação matriz-vetor.

NON-ZEROS (V)

Contém todos os **valores não nulos** da matriz, organizados sequencialmente linha por linha.

COLUMN INDICES (L)

Armazena o **índice da coluna** correspondente a cada elemento presente no vetor de valores.

ROW POINTERS (I)

Indica onde **começa e termina** cada linha dentro do vetor de valores. É o componente vital para a navegação.

Estrutura do Row Pointers

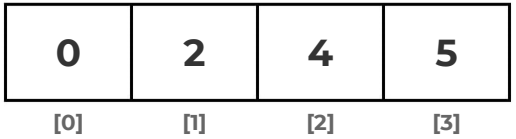
Tamanho do Vetor: Possui tamanho igual ao número de linhas mais um **(N + 1)**.

Função: O elemento `row_ptr[i]` indica exatamente onde a linha `i` começa.

Limite Final: O último elemento do vetor marca o fim da última linha.

Acesso Direto: Leia a linha `i` acessando índices entre `row_ptr[i]` e `row_ptr[i+1]`.

Exemplo de Vetor Row Pointers



Linha 0: Índices 0 a 1 (2 elementos)
Linha 1: Índices 2 a 3 (2 elementos)
Linha 2: Índice 4 (1 elemento)

Computação Síncrona vs. Assíncrona

Síncrona (Padrão)

1. **Host → GPU**: Transferência inicial de dados.
2. **GPU Executa**: Processamento do Kernel.
3. **GPU → Host**: Retorno dos resultados.

> *Hardware ocioso: A CPU espera a GPU terminar, e a GPU espera a CPU enviar dados.*

Assíncrona

- Permite enfileirar tarefas sem bloquear a CPU.
- **Sobreposição**: Transfere dados enquanto processa outros.
- Melhor aproveitamento dos recursos de hardware.

> *Redução do tempo total de execução através do paralelismo de tarefas.*

O que são CUDA Streams?

- Uma **fila de comandos** assíncronos enviada para execução na GPU.
- Operações na mesma stream seguem a ordem **FIFO** (First-In, First-Out).
- Operações em streams diferentes podem ser executadas de forma **concorrente**.

Exemplo de Filas Concorrentes:

Stream 0

```
Memcpy (H2D) → Kernel → Memcpy (D2H)
```

Stream 1

```
Memcpy (H2D) → Kernel → Memcpy (D2H)
```

Enquanto a Stream 0 executa um kernel, a Stream 1 pode estar transferindo dados.

Sobreposição de Operações (Overlapping)

Versão Sequencial (Síncrona)

Versão Assíncrona (Com Streams)

> Enquanto o **Kernel Engine** processa um conjunto de dados, o **Copy Engine** inicia a transferência dos dados seguintes. Essa simultaneidade reduz drasticamente o tempo total de execução.