Arquitetura e Organização de Computadores II

Unidades de Processamento Gráfico

Prof. Nilton Luiz Queiroz Jr.

Unidades de Processamento Gráfico

- No início as Unidades de Processamento Gráfico (Graphic processing units - GPUs) eram dedicadas a gerar gráficos 3D;
  - Continham unidades de ponto flutuante de alta performance;
  - Não eram programáveis;
- Com o passar do tempo foram se tornando cada vez mais programáveis;
  - GPUs produziam cores para cada pixel com unidades chamadas pixel shaders;
  - Os shaders recebiam dados adicionais;
    - Cores de entrada;
    - Coordenadas de texturas;
    - Outros atributos;

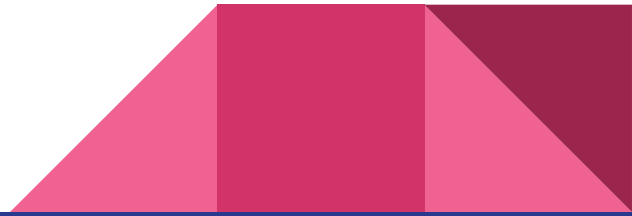
Unidades de Processamento Gráfico

- Toda aritmética aplicada sobre os dados de entrada era controlada pelo programador;
- Pesquisadores perceberam que as cores de entrada poderiam ser tratadas como dados;
  - Se tornou possível usar GPU para propósito geral com cálculos de shaders;

- Porém, ainda assim era algo um modelo de programação bem difícil;
- As GPUs passaram a ser mais usadas em computação de propósito geral quando surgiu uma linguagem de programação que as tornou mais fácil de programar;
  - Atualmente muitos programadores de aplicações científicas e multimídia tem a opção de usar tanto GPUs quanto CPUs;
- GPUs usadas em processamento de propósito geral são também chamadas de GPGPUs;
  - General Purpose Graphic Processing Units;

Unidades de Processamento Gráfico

- A ideia do uso de GPUs em propósito geral é:
  - Tomar vantagem da performance computacional e largura de banda da GPU para acelerar kernels de propósito geral;
- Porém, as GPUs ainda tem como objetivo principal a construção de gráficos;
  - Isso faz com que projetistas apenas tentem “suplementar” o hardware para suportar computação de propósito geral;



Modelo de programação

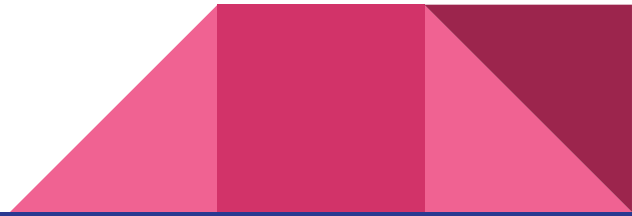
- Para programar uma GPU o programador deve coordenar o escalonamento da execução da GPU no processador;
- É necessário que o programador pense em grupos de operações SIMD para se ter um código eficiente em GPUs;
- Transferência de dados de dados entre processador e GPU também é responsabilidade do programador;
- Pode-se dizer que a GPU funciona como um co-processador para a CPU;
- GPUs englobam vários tipos de paralelismo:
  - Multithreading;
  - MIMD;
  - SIMD
  - ILP;

Modelo de programação

- Algumas linguagens foram desenvolvidas para facilitar a programação em GPUs:
  - Compute Unified Device Architecture (CUDA);
    - Arquitetura desenvolvida pela NVIDIA;
    - Como linguagem usa uma extensão de C/C++
      - C/C++ no processador;
      - Um dialeto de C/C++ para a placa de video;
  - OpenCL;
    - Desenvolvido por diversas empresas;
    - Tentativa de oferecer uma linguagem independente de fornecedor e para multiplas plataformas;

Modelo de programação

- Milhares de threads podem ser reunidas para utilizar estilos de paralelismo dentro de uma GPU;
- O modelo de programação CUDA é classificado pela NVIDIA como instrução única, múltiplos threads (Single Instruction Multiple Thread - SIMT);
- Esses threads são reunidos em blocos e executados em grupos de 32 threads;



Modelo de programação

- No modelo de programação CUDA é necessária a distinção de funções que serão executadas no CPU e na GPU;
  - Para GPU
    - __device__
    - __global__
  - Para CPU
    - __host__
- Variáveis declaradas nas funções CUDA são alocadas na memória da GPU;

Modelo de programação

- Em geral uma aplicação CUDA tem o seguinte fluxo:
  - Leitura de dados pela CPU;
  - Cópia de dados para memória da GPU;
  - Execução na GPU;
  - Cópia de dados para a memória da CPU;
  - Escrita dos dados pela CPU;

Modelo de programação

- Funções para GPU tem uma sintaxe estendida para sua chamada;
  - nome<<<dimGrid, dimBlock>>>(parametros);
    - dimGrid corresponde a dimensão do código (em blocos);
    - dimBlock corresponde a dimensão dos blocos (em threads);

DAXPY em CUDA

```
// Chamada de DAXPY em CUDA
daxpy<<<nblocks, nthreads>>>(n, 2.0, x, y);

// DAXPY em CUDA
__device__
void daxpy(int n, double a, double *x, double *y)
{
        int i = blockIdx.x*blockDim.x + threadIdx.x;
        if (i < n) y[i] = a*x[i] + y[i];
}
```

DAXPY em CPU

```
// Chamada de DAXPY em C
daxpy(n, 2.0, x, y);

// DAXPY em C
void daxpy(int n, double a, double *x, double *y)
{
        for (int i = 0; i < n; ++i)
                y[i] = a*x[i] + y[i];
}
```

Modelo de programação

- O hardware de uma GPU de arquitetura CUDA suporta execução em paralelo e o gerenciamento de threads;
  - O próprio hardware é capaz de fazer o escalonamento;
  - Blocos e threads devem ser capazes de ser executados independentemente e em qualquer ordem;
    - Tais blocos não podem se comunicar, porém são capazes de se coordenar usando operações atômicas na memória global;

Estrutura de uma GPU

- GPU possuem terminologias próprias;
  - NVIDIA possui alguns termos oficiais:
    - Abstrações de programa:
      - Grid: Agregado de um ou mais blocos de thread;
        - Executado na GPU;
        - Pode ser visto como um loop vetorizavel;
      - Bloco de thread: Agregado de threads;
        - Executado em uma pista SIMD;
        - Pode ser visto como o corpo de um loop vetorizado;
      - Thread CUDA: Corde vertical de um thread de instruções;
        - Um elemento executado dentro de uma pista SIMD;
        - Pode ser visto como uma iteração do loop escalar;

Estrutura de uma GPU

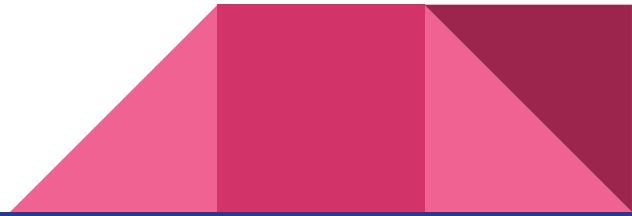
- Objeto de máquina
  - Warp: Thread que contém somente instruções SIMD;
    - Executado em um processador SIMD;
    - Pode ser visto como uma thread de instruções SIMD;
  - Instrução PTX: Uma única instrução SIMD;
    - Executada em pistas SIMD;
    - Pode ser vista como uma instrução assembly vetorial;
- Hardware de memória:
  - Registradores de processador de thread:registradores de uma única pista SIMD
  - Memória compartilhada: Memória de um processador SIMD;
    - Indisponível para os demais processadores SIMD;
  - Memória Local: memória privada para cada pista SIMD;
  - Memória global: memória da GPU;

Estrutura de uma GPU

- Hardware de processamento:
  - Multiprocessador de Streaming (Streaming multiprocessor - SMP): Processador SIMD multithreaded;
    - Executa threads de instruções SIMD de maneira independente de outros processadores SIMD;
  - Processador de thread: Uma pista SIMD;
    - Executa as operações em um thread de instruções simples em um único elemento;
    - Dentro da arquitetura cuda são comumente chamados de CUDA cores;
  - Engine Giga thread: Escalonador de Blocos de threads;
    - Designa múltiplos blocos de threads a processadores SIMD;
  - Escalonador de warp: Escalonador de Threads SIMD
    - Unidade de hardware para escalonar e despachar threads de instruções SIMD quando prontas;
    - Usa um scoreboard para rastrear a execução de threads SIMD;

Estrutura de uma GPU

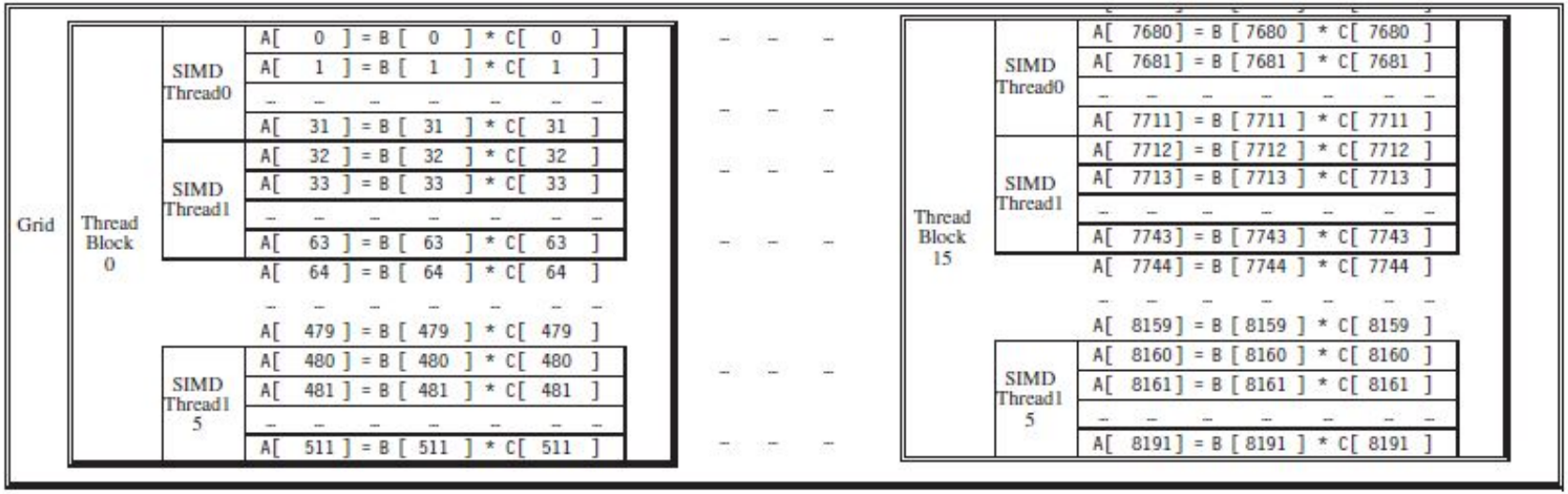
- As GPUs funcionam bem somente com problemas paralelos em nível de dados;
  - Assim como processadores vetoriais possuem transferências de dados gather-scatter e registradores de máscara;
    - GPUs possuem ainda mais registradores que processadores vetoriais;
- Algumas GPUs dependem de multithreading dentro de um único processador SIMD para ocultar latência de memória;



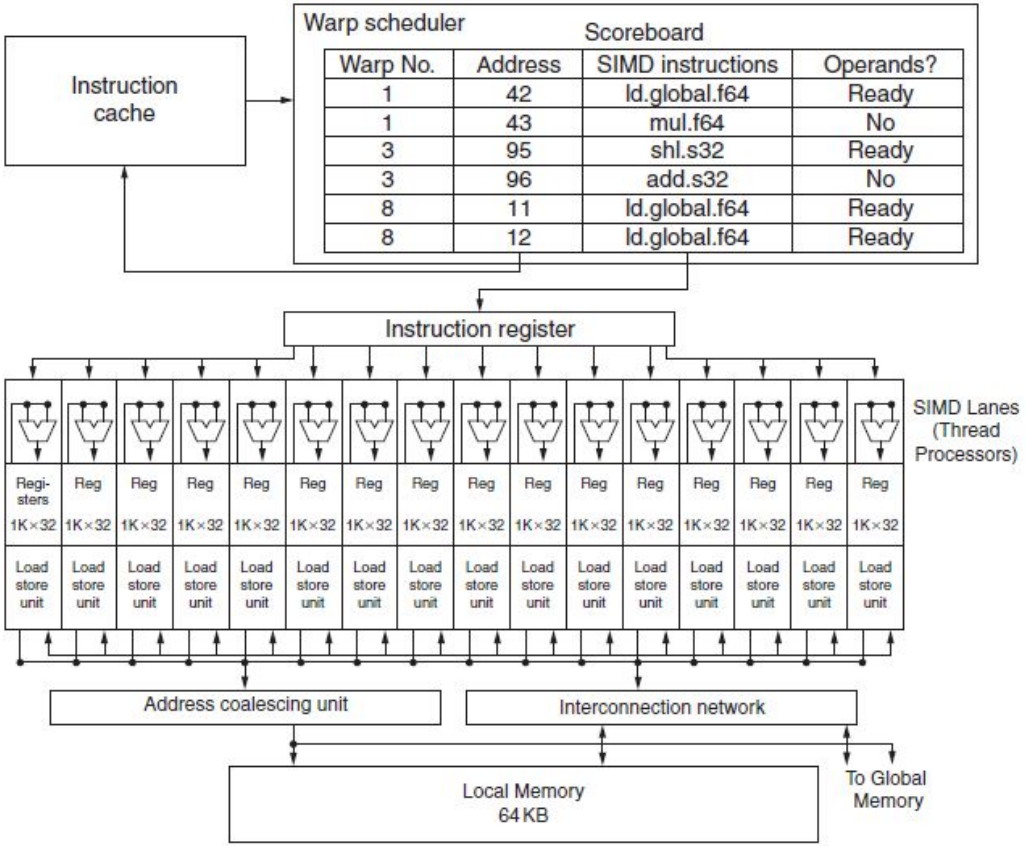
Execução na GPU

- O código executado na GPU é chamado de Grid;
- Um Grid é um conjunto de Blocos de threads;
- Tanto os grids quanto o blocos de threads são abstrações de programação implementadas no hardware da GPU;
  - Auxiliam o programador a organizar o código;
- Supondo que dois vetores de 8192 elementos serão multiplicados em uma GPU;
  - O grid trabalhará sobre as 8192 multiplicações;
  - Decompõe-se tal grid em partes menores chamadas de blocos (ou blocos de threads);
    - Existe um limite de elementos por bloco;
      - Limitado pela plataforma CUDA;
        - Versões mais atuais suportam até 1024;
        - Versões anteriores suportavam até 512;

Execução na GPU



Processador SIMD multithreaded



Estrutura de uma GPU

- O hardware da GPU contém uma coleção de processadores SIMD multithreaded que executam um grid de blocos de threads;
  - Em outras palavras, uma GPU um multiprocessador composto de processadores SIMD multithreaded;
- A quantidade de registradores em uma GPU é muito maior que a de uma CPU;
  - Um processador SIMD tem por volta de 32768 registradores de 32 bits;
    - Divididos em pistas, onde cada pista tem por volta de 64 registradores vetoriais de 32 elementos, e cada elemento contém 32 bits;

Escalonamento em GPU

- O escalonamento em GPUs é feito em dois níveis:
  - Escalonamento de bloco de threads;
    - Designa blocos de threads a processadores SIMD multithreaded;
    - Garante que os blocos de threads sejam designados para os processadores cujo as memórias locais tenham dados correspondentes;
  - Escalonamento de threads SIMD dentro de um processador SIMD;
    - Escalona quando os threads de instruções SIMD devem ser executados;

Conjunto de instruções da GPU NVIDIA

- O conjunto de instruções alvo dos compiladores NVIDIA é uma abstração do conjunto de instruções do hardware;
  - Esse conjunto é chamado de PTX (Parallel Thread Execution);
    - O PTX pode ser comparado as instruções x86, que na verdade são traduzidas para microinstruções no Hardware, porém no PTX essa tradução é feita no software e no tempo de carregamento;
- As instruções PTX são da seguinte forma:
  - opcode.tipo d, a, b, c;
    - Sendo d o operando destino, a,b e c operandos origem e o tipo indica qual o tipo de dado trabalho;
    - O tipo pode ser:
      - b8, b16, b32, b64 para bits sem tipo;
      - u8, u16, u32, u64 para inteiros sem sinal;
      - s8, s16, s32, s64 para inteiros com sinal;
      - f16,f32,f4 para ponto flutuante;

Conjunto de instruções da GPU NVIDIA

- As instruções são feitas com registradores virtuais de 32 ou 64 bits, com ou sem tipo;
  - Por exemplo: R0, R1, R2 … para valores de 32 bits e RD0, RD1, RD2 para registradores de 64 bits;
  - A designação de registradores virtuais para registradores físicos ocorre no momento do carregamento;

Conjunto de instruções da GPU NVIDIA

- Todas transferências de dados em GPU são gather-scatter;
  - Para transferências sequenciais existe hardware de aglutinação de endereço;
    - Esse hardware reconhece quando as pistas SIMD estão enviando coletivamente endereços sequenciais;
    - O hardware então notifica as unidades de memória, fazendo requisição de palavras de 32 bits sequenciais;
  - O programador deve tentar garantir que as threads adjacentes acessem endereços próximos ao mesmo tempo;
    - Desse modo se torna possível a aglutinação em um ou poucos blocos da memória ou da cache;

Desvios em GPU

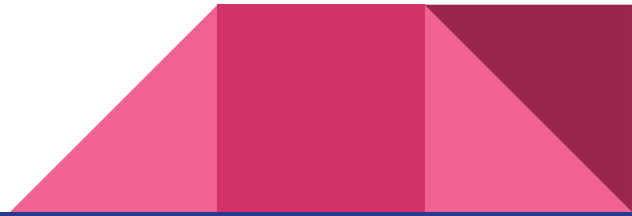
- GPUs possuem similaridades com arquiteturas vetoriais no tratamento de desvios;
  - Porém, o controle de desvios em arquiteturas vetoriais é feito em maior parte no software;
- GPUs possuem um hardware mais elaborado para desvios;
  - Registradores explícitos;
  - Máscaras internas;
  - Pilha de sincronização de desvios;
  - Marcadores de instrução;
    - Responsáveis por gerenciar quando um desvio diverge em multiplos caminhos de execução e quando os caminhos convergem;

Desvios em GPUs

- Cada thread SIMD tem sua própria pilha de sincronização de desvio;
  - Cada entrada da pilha contém:
    - Token identificador;
    - Endereço de instrução alvo;
    - Mascara-alvo de thread ativo;
- São usadas instruções especiais para:
  - Empilhar entradas na pilha;
  - Desempilhar entradas da pilha
  - Retornar a pilha para uma entrada específica e desviar a execução para o endereço da instrução alvo;
- Além disso existem também predicados individuais por pista;
  - 1 bit para cada pista

Desvios em GPUs

- Declarações if/then/else são otimizadas pelo assembler PTX;
  - Viram instruções com predicado, sem instruções de desvio;
- Porém fluxos de controle mais complexos misturam predicados e instruções de desvios em GPU;
  - Pode acontecer de pistas desviarem e outras caírem (ou seja, o desvio diverge);
    - A solução para isso é serializar a execução em dois caminhos até o ponto de convergência;



Desvios gerados em GPU

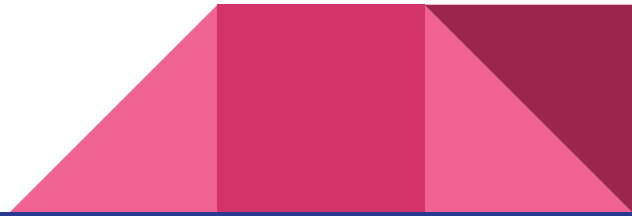
- Se forem usadas instruções com predicado sem instruções de desvio parte das pistas serão desabilitadas;
  - Enquanto as instruções dentro da parte *then* da condicional são executadas por suas respectivas pistas, as pistas com instruções da parte *else* não realizam operações e nem armazenam resultados;
  - O mesmo acontece com a parte *else;*
  - Para caminhos de comprimento igual um if-then-else opera com 50% de eficiência;
- Declarações if aninhadas fazem com que a eficiência fique mais baixa;
  - Dois ifs aninhados de mesmo tamanho executam com 25% de eficiência;
  - No caso de 3 ifs a eficiência é de 12,5%;
  - Assim por diante;

Desvios gerados em GPU

- O que determina se a pista será ou não desabilitada é um registrador de predicado;
- Quando o branch diverge o assembler:
  - Empilhada uma entrada e setada uma máscara interna que determina se a pista está ou não ativa, baseada na condição;
  - Após o término da execução do trecho pertencente ao then os bits de máscara são complementados (ou seja, invertidos);
  - No fim da declaração if é desempenhada a máscara ativa, para que a pilha volte ao estado que estava antes do início da execução da declaração if-then-else;
- Quando o branch converge todas instruções do then ou do else são puladas;

Desvios gerados em GPU

- Duas lanes SIMD não podem executar instruções diferentes simultaneamente
- Cada thread CUDA executa a mesma instrução que os demais threads no mesmo bloco;
  - Caso contrário estará ocioso;
- A eficiência que as GPUs executam declarações condicionais se resume a frequência com que os desvios divergem;



Referências

NVIDIA. CUDA Toolkit Documentation. 2017. Acessado em 24/07/2017. Disponíevel em: <http://docs.nvidia.com/cuda/index.html>

HENNESSY, John L.;PATTERSON, David A. Computer architecture: a quantitative approach. Elsevier, 2011.

Wentzlaff, David. Vector, SIMD, and GPUs. Notas de Aula.