Programação Paralela em CUDA

Nome: Arthur Arantes Faria

Sumário

- Introdução.
- Materiais e Métodos.
- Trabalhos Relacionados.
- Experimentos.
- Conclusões.

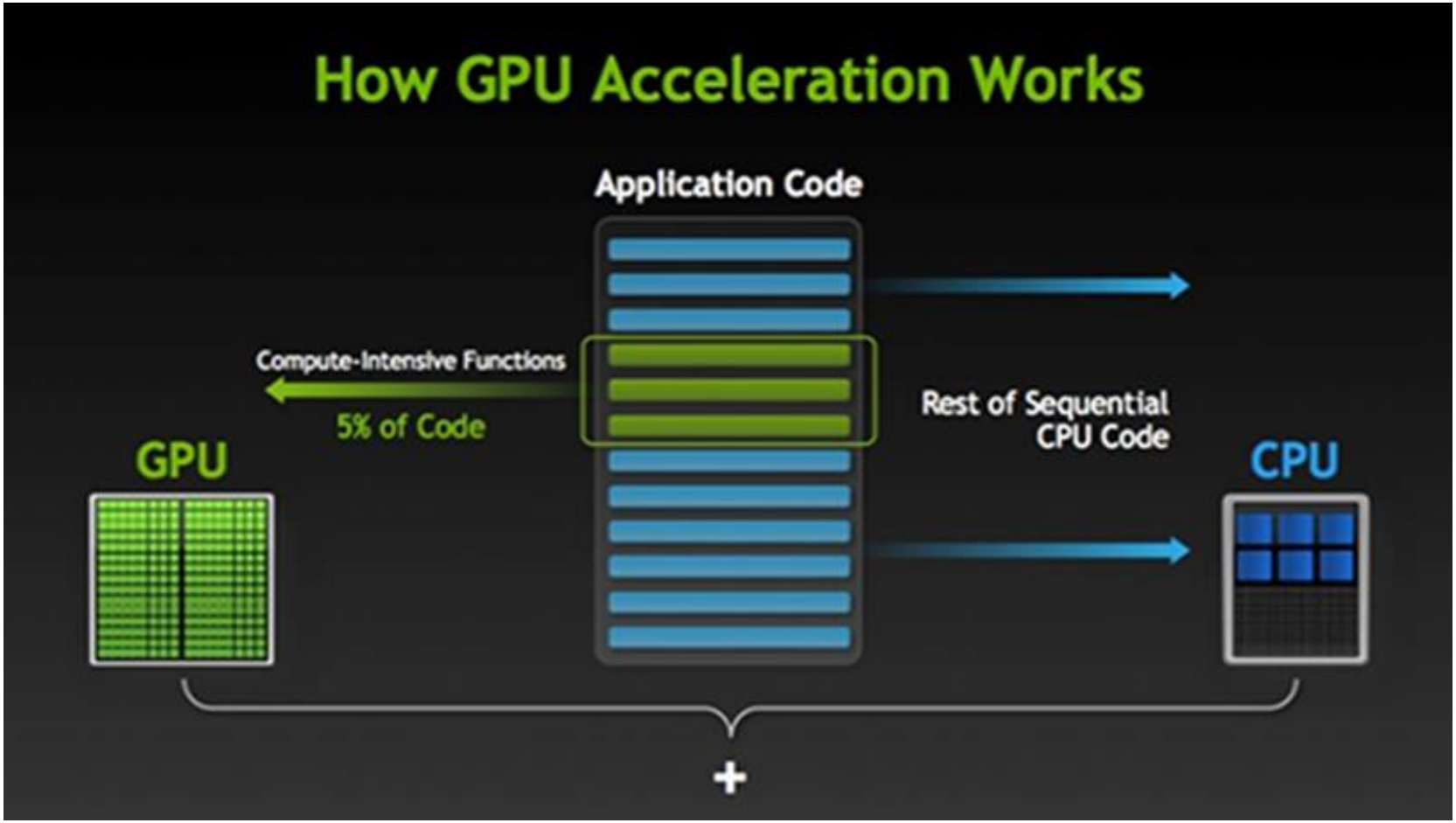
Evolução do Hardware e Software

- Hardware evoluiu em uma proporção mais rápida que o software.
  - CPUs *multicores*, memorias ram, armazenamento, gpus, etc.
- A grande maioria dos software não evoluíram na mesma proporção dos hardwares.
  - Editores de texto, navegadores de internet, etc.

GPUs

- Os principais fabricantes de GPUs (desktop) são a NVIDIA e a AMD.
  - Podem atuar em conjunto com CPUs Intel ou AMD.
- Tem sua própria hierarquia de memória e os dados são transferidos através de um barramento *PCI express*.
- Programa principal executa na CPU (*host*) e inicia as *threads* na GPU (*device*).

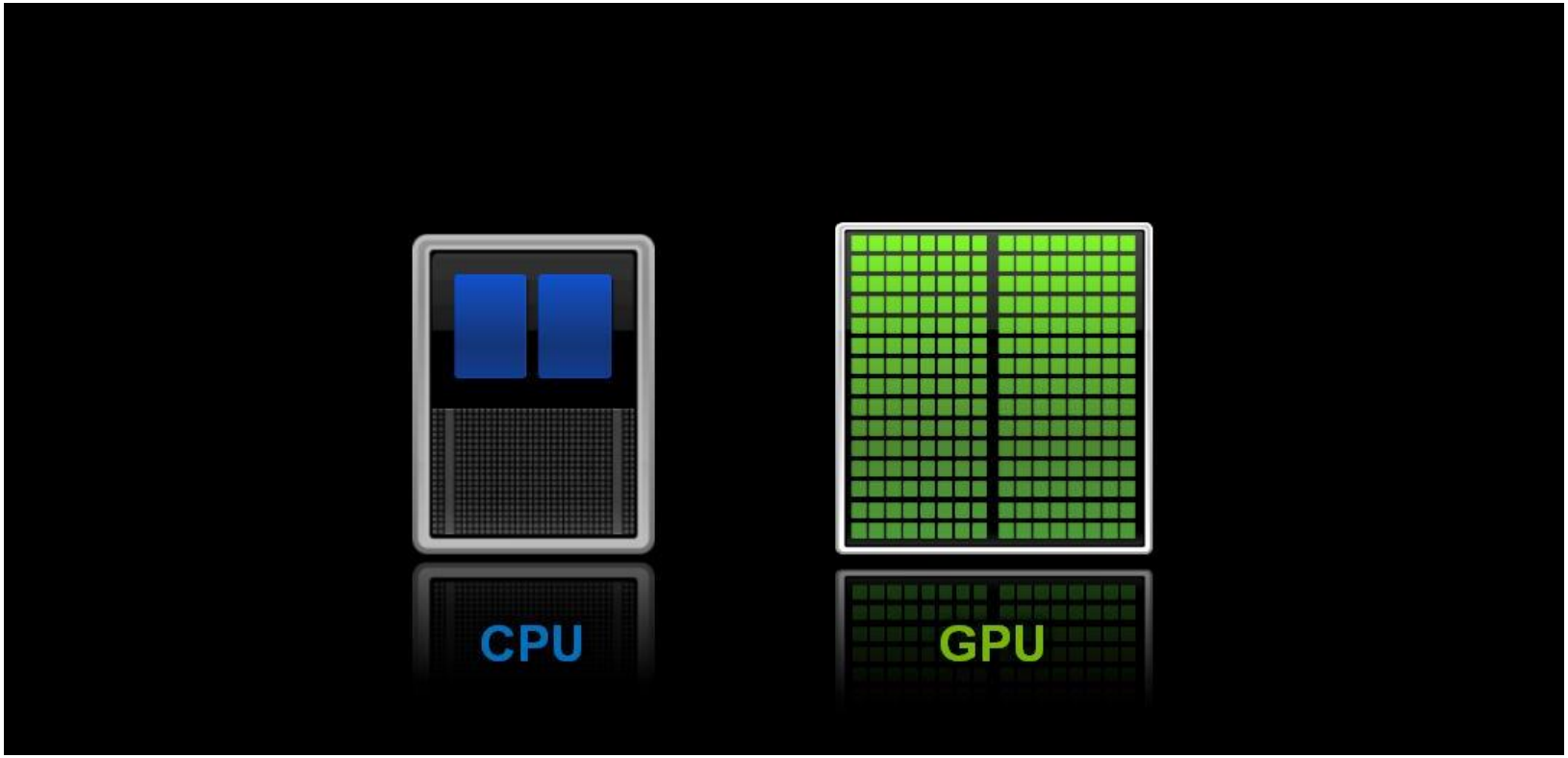
Programa principal executa na CPU (host) e inicia as threads na GPU (device).



O que é GPGPU ?

- *General Purpose computation using GPU*
  em outras aplicações que não sejam somente graficos 3D.
  - Usar a GPU para acelerar determinada aplicação.
- Algoritmos paralelos de dados aproveitam das características da GPU
  - Matrizes de dados grandes, alta taxa de transferência
  - Ganho de performance.
  - Baixa latência para tratamentos com *Floating point*

Diferença entre processadores CPU e GPU



Diferença entre processadores CPU e GPU

Característica	CPU i7-6700HQ	GPU NVIDIA 970M
Número de núcleos	4	**1280 CUDA**
Nº de threads	**8**	--
Frequência	**2,60 GHz**	924 Mhz
Frequência turbo max	**3,50 GHz**	--
TDP	**45 W**	100 W

Materiais e Métodos
Processos

- Um processo pode ser definido como uma das entidades de um programa que possui um contador que determina a próxima instrução a ser executada.
    - (SILBERSCHATZ; GALVIN; GAGNE, 2000)

- Tipos de processos:
    - Processos independentes,
    - Processos cooperativos.
        - (SILBERSCHATZ; GALVIN; GAGNE, 2000)

- Revisar a base de processos de um so pode reduzir a chance de, ao programar a paralelização de um programa, gerar um *deadlock*.

Materiais e Métodos
Programação Paralela

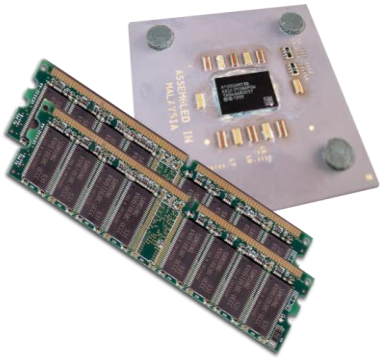
- Tipicamente, softwares tem sido desenvolvidos para serem executados sequencialmente.

- Com programação paralela é possível quebrar um problema em diversas partes independentes, assim, de tal forma cada elemento do processamento poderá executar sua parte do algoritmo simultaneamente com os outros em um núcleo diferente.

- Um dos objetivos de usar a paralelização é aumentar o desempenho da aplicação com a execução nos vários núcleos disponíveis na máquina

Materiais e Métodos
CUDA

- “Compute Unified Device Architecture”
  - API Desenvolvida pela NVIDIA
- Terminologia usada pelo CUDA :
  - *Host* O CPU e a sua memória (*host memory*)
  - *Device* A GPU e a sua memória (*device memory*)



Host

Device

Materiais e Métodos
CUDA

- Modelo de programação de uso geral
  - O usuário inicia lotes de threads na GPU
  - GPU = coprocessador paralelo de dados maciçamente super-encadeado
- Pilha de software direcionada
  - *Compute drivers* orientados, linguagem e ferramentas
- Driver para carregar programas de computação na GPU
  - Driver autônomo - otimizado para computação
  - Interface projetada para API livre de gráficos de computação
  - Compartilhamento de dados com objetos de buffer
  - Velocidades máximas de download e readback garantidas
  - Gerenciamento explícito de memória GPU
- (NVIDIA, 2018)

Materiais e Métodos
CUDA

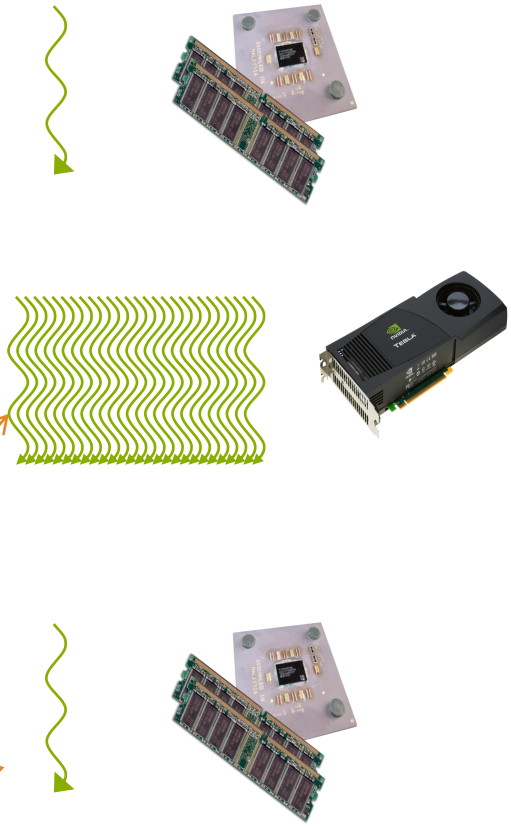
```
#include <iostream>
#include <algorithm>

using namespace std;

#define N          1024
#define RADIUS     3
#define BLOCK_SIZE 16

__global__ void stencil_1d(int *in, int *out) {
        __shared__ int temp[BLOCK_SIZE + 2 * RADIUS];
        int gindex = threadIdx.x + blockIdx.x * blockDim.x;
        int lindex = threadIdx.x + RADIUS;

        // Read input elements into shared memory
        temp[lindex] = in[gindex];
        if (threadIdx.x < RADIUS) {
                temp[lindex - RADIUS] = in[gindex - RADIUS];
                temp[lindex + BLOCK_SIZE] = in[gindex + BLOCK_SIZE];
        }

        // Synchronize (ensure all the data is available)
        __syncthreads();

        // Apply the stencil
        int result = 0;
        for (int offset = -RADIUS ; offset <= RADIUS ; offset++)
                result += temp[lindex + offset];

        // Store the result
        out[gindex] = result;
}

void fill_ints(int *x, int n) {
        fill_n(x, n, 1);
}

int main(void) {
        int *in, *out;          // host copies of a, b, c
        int *d_in, *d_out;      // device copies of a, b, c
        int size = (N + 2*RADIUS) * sizeof(int);

        // Alloc space for host copies and setup values
        in  = (int *)malloc(size); fill_ints(in,  N + 2*RADIUS);
        out = (int *)malloc(size); fill_ints(out, N + 2*RADIUS);

        // Alloc space for device copies
        cudaMalloc((void **)&d_in,  size);
        cudaMalloc((void **)&d_out, size);

        // Copy to device
        cudaMemcpy(d_in,  in,  size, cudaMemcpyHostToDevice);
        cudaMemcpy(d_out, out, size, cudaMemcpyHostToDevice);

        // Launch stencil_1d() kernel on GPU
        stencil_1d<<<N/BLOCK_SIZE,BLOCK_SIZE>>>(d_in + RADIUS,
d_out + RADIUS);

        // Copy result back to host
        cudaMemcpy(out, d_out, size, cudaMemcpyDeviceToHost);

        // Cleanup
        free(in); free(out);
        cudaFree(d_in); cudaFree(d_out);
        return 0;
}
```

Área paralela

Código em série

Código em paralelo

Codigo em série



Materiais e Métodos Comparação

- OpenMP
  - Paralelismo na CPU
  - Apresentado algumas aulas atrás.

- OpenCL
  - Outra opção de paralelismo na GPU
  - Também apresentado algumas aulas atrás.

Trabalhos Relacionados

- LEE, V. W. et al. Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu. **ACM SIGARCH computer architecture news**, ACM, v. 38, n. 3, p. 451–460, 2010.

- DU, P. et al. From cuda to opencl: Towards a performance-portable solution for multi-platform gpu programming. **Parallel Computing**, Elsevier, v. 38, n. 8, p. 391–407, 2012.

Experimentos

- Algoritmos testados:
  - Sequencia de *Fibonnaci*;
  - Multiplicação de matrizes.
- Métrica de desempenho
  - Menor tempo de execução de relógio.

Experimentos

- Sequencia de *Fibonnaci*;

N	Sequencial	OpenMP	CUDA
50	276.221968	273.421303	**0.000000**

- *Tempo em segundos*

Experimentos

- Multiplicação de matrizes.

N	Sequencial	OpenMP	CUDA
50	0.002010	0.000000	0.000034
500	0.929516	0.902617	0.000046
1000	7.933796	7.931795	0.000057

- *Tempo em segundos*

Conclusões
Observações sobre CUDA

- Pontos fortes
  - A técnica em GPU é altamente poderosa
    - Especialmente em aplicações altamente paralelizáveis.
    - Menor custo e espaço pelo hardware.
- Pontos fracos
  - Necessidade de um hardware habilitado para CUDA

    -- Baixa curva de aprendizagem.

    -- Em contrapartida existe padronizações.

    » OpenCL (Open Computing Language)

Conclusões

- Pode-se perceber uma vantagem clara em ganho de tempo de execução de um algoritmo quando é executado na GPU utilizando CUDA, quando comparado com os outros meios de execução.

- Acreditasse que a vantagem clara que a GPU tem diante a CPU é a quantidade de núcleos, logo, quanto mais núcleos a placa tiver, maior poder de processamento e melhor seu o desempenho será.

- Por outro lado, a vantagem da CPU é o seu relacionamento direto com os outros componentes do computador por ser um componente central indispensável.

Algumas fontes

- Principais fonte de aprendizado:
  - CUDA Programming Guide;
  - Udemy;
  - CUDA Zone – tools, training, webinars and more

    developer.nvidia.com/cuda

- Recomendam revisar antes de aprofundar:
  - Processos;
  - Programação paralela;
  - *Multi-dimensional indexing;*
  - *Textures.*

Executar Código fonte