TN741 - Computação de Alto Desempenho
Introdução ao CUDA

Marcelo Zamith

e-mail: mzamith@ufrrj.br
https://www.dcc.ufrrj.br/∼marcelo/

Universidade Federal Rural do Rio de Janeiro - DCC

UFRRJ
UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO

Roteiro

Introdução

A máquina que mudou o mundo !?

Introdução

- Computadores vistos como máquinas seqüências.
- Programas e algoritmos definidos como uma seqüência de passos.
- Execução do programa em uma seqüência de instruções, uma instrução por vez.
- Instruções:
  - Execução de microinstruções e sinais simultaneamente.
  - Uso de *pipelines* para execução de instruções.

Introdução

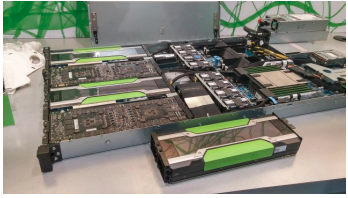
- Paralelismo de instrução super escalar.
- Várias unidades de execução dentro de um único processador.
- Execução de múltiplas instruções do mesmo programa em paralelo.
- Baixo custo de *hardware*.
- Melhora de desempenho.
- *Stream multiprocessors* (SMs).
- Compartilhamento de memória.
- Coerência de cache.

Introdução

1. O que é a GPU ?

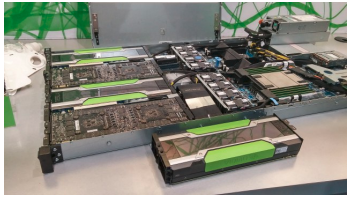
Introdução

1. O que é a GPU ?



Introdução

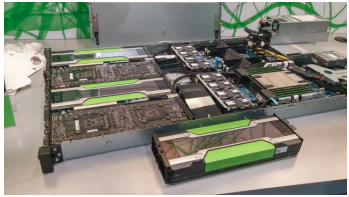
1. O que é a GPU ?



2. Por quê usar GPU ?

Introdução

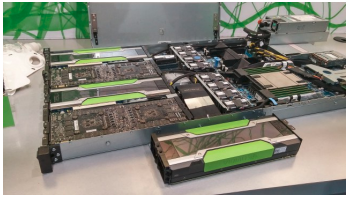
1. O que é a GPU ?



2. Por quê usar GPU ?

Introdução

1. O que é a GPU ?

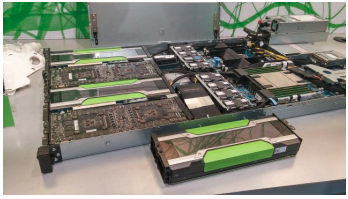


2. Por quê usar GPU ?

=

Introdução

1. O que é a GPU ?



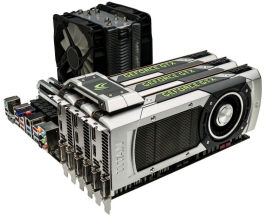
2. Por quê usar GPU ?

=

UFRRJ
UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO

Introdução

- Energia, calor, $$$



Introdução

Introdução

- Top 500 supercomputer (`http://www.top500.org/`)

R.	Computer	T. C.	Power	Mf./W.	Acc
1	TH-IVB-FEP	3120000	17808	1901.54	Xeon Phi 31S1P
2	Cray XK7	560640	8209	2142.77	NVIDIA K20x
3	BlueGene/Q	1572864	7890	2176.58	None
4	K computer	705024	12659.89	830.18	None
5	BlueGene/Q	786432	3945	2176.58	None
6	Cray XC30	115984	2325	2697.2	NVIDIA K20x
7	Cray XC40	196608	2834	1953.77	None
8	PowerEdge C8220	462462	4510	1145.92	Xeon Phi SE10P
9	BlueGene/Q	458752	2301	2176.82	None
10	BlueGene/Q	393216	1972	2177.13	None

Introdução

- Top 500 supercomputer (`http://www.top500.org/`)

R.	Computer	T. C.	Power	Mf./W.	Acc
1	TH-IVB-FEP	3120000	17808	1901.54	Xeon Phi 31S1P
2	Cray XK7	560640	8209	2142.77	NVIDIA K20x
3	BlueGene/Q	1572864	7890	2176.58	None
4	K computer	705024	12659.89	830.18	None
5	BlueGene/Q	786432	3945	2176.58	None
6	Cray XC30	115984	2325	2697.2	NVIDIA K20x
7	Cray XC40	196608	2834	1953.77	None
8	PowerEdge C8220	462462	4510	1145.92	Xeon Phi SE10P
9	BlueGene/Q	458752	2301	2176.82	None
10	BlueGene/Q	393216	1972	2177.13	None

Introdução

- The Green 500 (`http://www.green500.org/`)

R. G.	R. S.C.	Power	Mf./W.	T. C.	Computer	Acc.
1	160	50.31	7031.57	787968	ExaScaler	PEZY-SC
2	392	28.24	6842.32	263168	ExaScale	PEZY-SC
3	366	32.59	6217.04	262784	ExaScaler	PEZY-SC
4	215	57.15	5271.81	10976	ASUS	AMD S9150
5	469	39.83	4257.88	2992	LX	NVIDIA K20x
6	87	190	4112.10	14820	Cray	NVIDIA K80
7	437	44.54	3962.73	3080	Cray	NVIDIA K40m
8	301	52.62	3631.69	4864	Dell	NVIDIA K20
9	363	58.013	3614.70	3200	Bull	NVIDIA K80
10	395	54.6	3543.31	4318	iDataPlex	NVIDIA K20x

Introdução

- The Green 500 (`http://www.green500.org/`)

R. G.	R. S.C.	Power	Mf./W.	T. C.	Computer	Acc.
1	160	50.31	7031.57	787968	ExaScaler	PEZY-SC
2	392	28.24	6842.32	263168	ExaScale	PEZY-SC
3	366	32.59	6217.04	262784	ExaScaler	PEZY-SC
4	215	57.15	5271.81	10976	ASUS	AMD S9150
5	469	39.83	4257.88	2992	LX	NVIDIA K20x
6	87	190	4112.10	14820	Cray	NVIDIA K80
7	437	44.54	3962.73	3080	Cray	NVIDIA K40m
8	301	52.62	3631.69	4864	Dell	NVIDIA K20
9	363	58.013	3614.70	3200	Bull	NVIDIA K80
10	395	54.6	3543.31	4318	iDataPlex	NVIDIA K20x

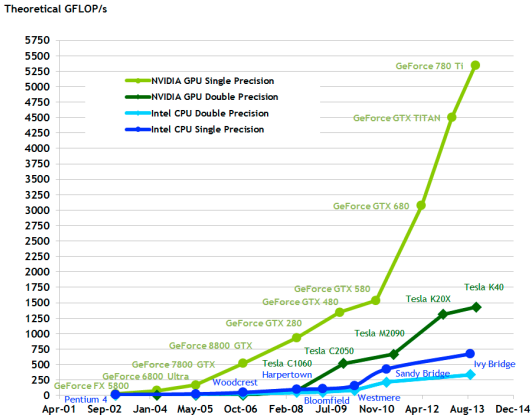
Introdução

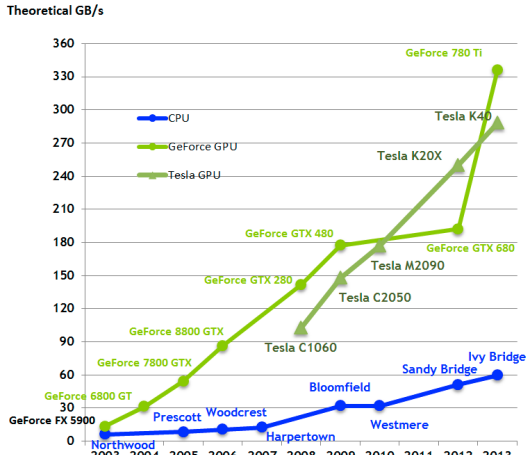
- As GPUs são usadas na industria:
  - Óleo e gás: processamento sísmico, simulação de reservatório, etc...
  - Pesquisa: astrofísica, dinâmica dos fluídos, previsão de tempo, etc...
  - Defesa e governo: processamento de imagem satélite, processamento de imagens e videos, etc...
  - Simulação bio-química, seqüência de DNA, etc...
  - Finanças: Método de Monte Carlo, análise de risco financeiro, etc...
  - Produção: Mecânica Estrutural, etc...
- Soluções de arquiteturas dedicadas e desenvolvimento de tecnologia.

Introdução

- As GPUs são usadas na industria:
  - Óleo e gás: processamento sísmico, simulação de reservatório, etc...
  - Pesquisa: astrofísica, dinâmica dos fluídos, previsão de tempo, etc...
  - Defesa e governo: processamento de imagem satélite, processamento de imagens e videos, etc...
  - Simulação bio-química, seqüência de DNA, etc...
  - Finanças: Método de Monte Carlo, análise de risco financeiro, etc...
  - Produção: Mecânica Estrutural, etc...
- Soluções de arquiteturas dedicadas e desenvolvimento de tecnologia.

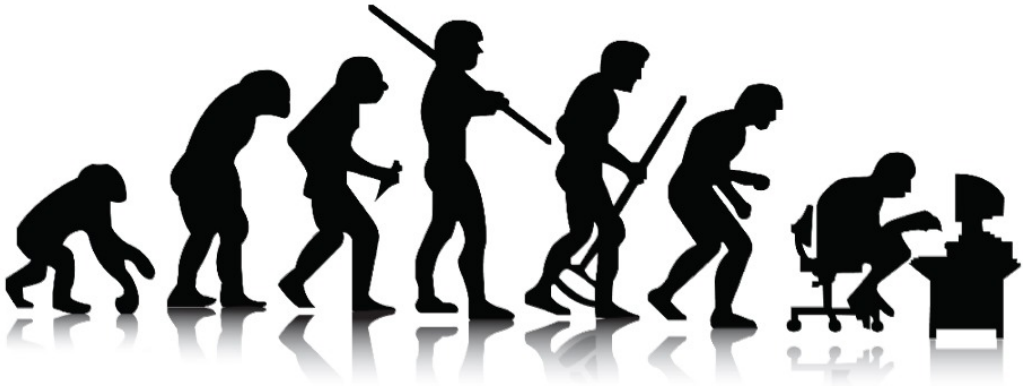
Introdução





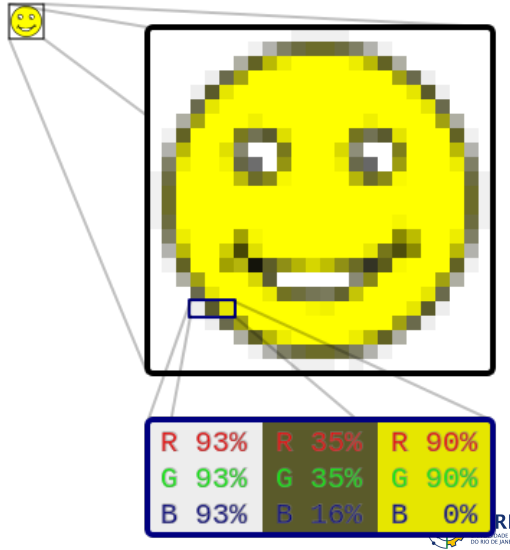
UFRRJ
UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO

De onde vieram as GPUs?



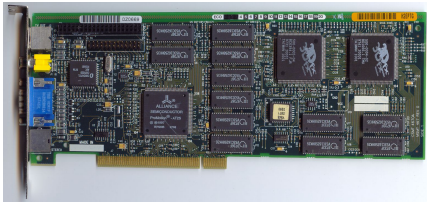
História: Pré-história

- Dispositivo de rasterização.
- Anos 60.
- Inventada por A. Michael Noll na Bell Labs.
- Usada para pintar polígonos.



História: GPUs de função fixa

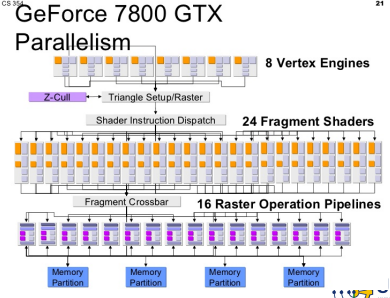
- 1996 VoodooFX.
- 1999 GeForce256 & ATI Radeon 7500 (*pipeline*).
- Não eram programadas.
- Processamento de vértices e pixel implementados em *hardware*.
- Apenas Computação Gráfico.
- Sem qualquer acesso ao processador.
- Utilização através de APIs.



UFRRJ
UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO

História: GPUs Programáveis

- 2000 até 2005.
- GeForce 3, FX, etc...
- ATI Radeon 8500, 9700, etc...
- Programação em linguagem Shader (GLSL, HLSL, CG).
- Processadores de fluxo (SIMD - Taxonomia Flynn)
- Programação dos estágio de vértice e pixel.
- GPGPU (*General-purpose computing on* GPU).
- Apenas Computação Gráfico.
- Utilização através de APIs.



História: Arquitetura Unificada

- 2006 - NVIDIA - Série 8 em diante.
- Um único conjunto de processadores.
- Surgimento do CUDA *Compute Unified Device Architecture*.
- Capacidade de programar a GPU para propósitos gerais.
- Biblioteca para diversas linguagens.

História: Arquitetura Unificada

- 2006 - Série 8:
  - recursos básicos do CUDA.
- 2008 - Tesla:
  - precisão dupla.
  - operações atômicas.
- 2010 - Fermi:
  - 64Kbytes de memória compartilhada.
  - suporte C++.
  - Caches L1, L2.
- 2012 - Kepler:
  - Nova geração dos SMs
  - Paralelismo dinâmico.
  - Cache L2 de 256Kbytes
  - GPUDirect
- 2014 - Maxwell:
  - Nova geração dos MPs
  - Cache L2 de 2048Kbytes

História: Arquitetura Unificada

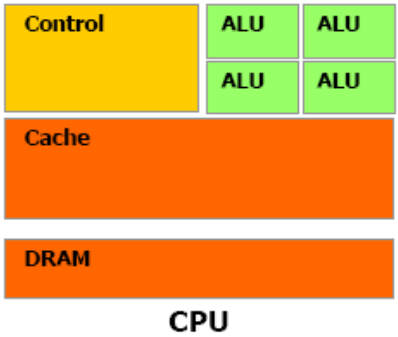
- 2006 - NVIDIA - Série 8 em diante.
- Um único conjunto de processadores.
- Surgimento do CUDA *Compute Unified Device Architecture*.
- Capacidade de programar a GPU para propósitos gerais.
- Biblioteca para diversas linguagens.

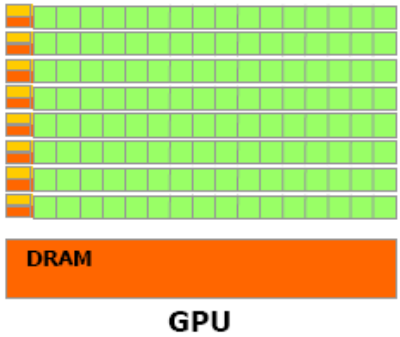
Modelo de Programação CUDA

- Capacidade de processamento de ponto flutuante.
- Uso da GPU para processamento não gráfico.
- GPGPU - *General Purpose computation on* GPU: Uso da GPU para programação de propósito geral usando alguma API gráfica.
- GPU *Computing*:Uso da GPU para computação paralela através de linguagem de programação e API, sem uso de APIs gráficas.
- CUDA: Modelo de programação paralela e plataforma de software para GPU.

GPU × CPU

- Unidade de controle de fluxo de instruções.
- Tamanho das memórias (Cache e MP).
- ALUs e cores.
- Gerência e criação de *threads*.



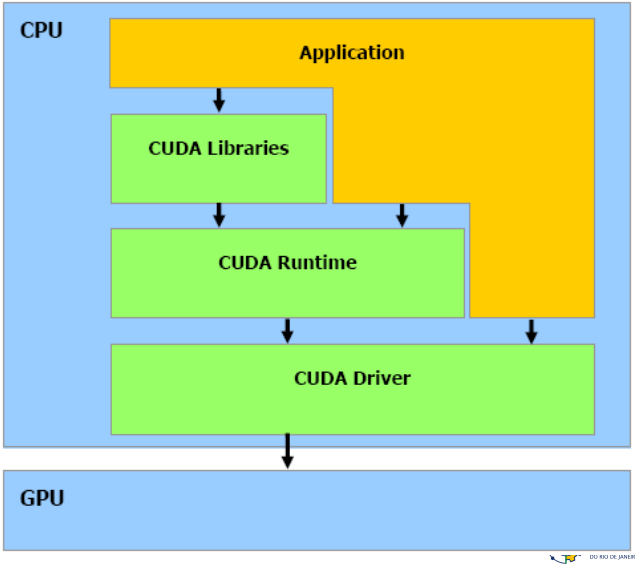


GPU × CPU

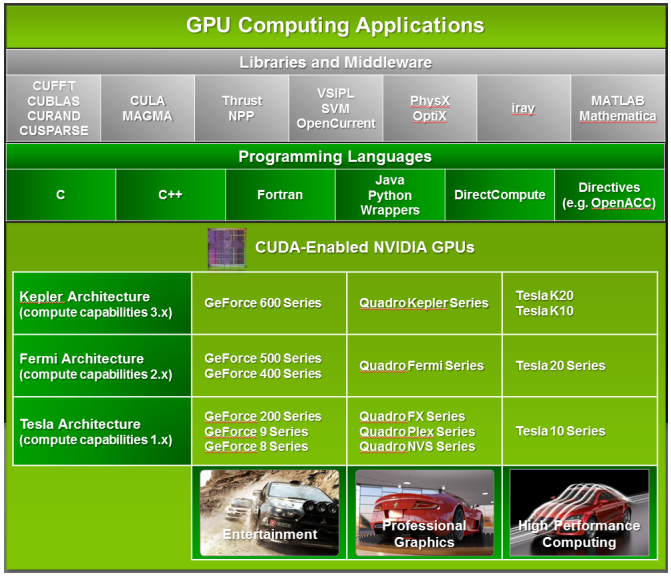
- Paralelismo extensivo de dados: milhares de computações feitas sobre os dados.
- Grupos de tarefas simples: grupo de *threads* podem executar diferentes programas.
- Aritmética intensiva de ponto flutuante.
- Latência e tolerância: quantidade de tarefas executas por instantes de tempo.
- Fluxo de dados: alta largura de banda para transferência dos dados com pouco reuso.
- Sincronismo entre *threads* de forma simples (barreira). *Threads* de um mesmo bloco.
- Custo zero para escalonamento das *threads*.

CUDA - *Compute Unified Device Architecture*

- Biblioteca (APIs).
  - Aplicação.
  - *runtime*.
  - *driver*.
- *toolkit*.
- *driver* da placa gráfica.



CUDA - *Compute Unified Device Architecture*

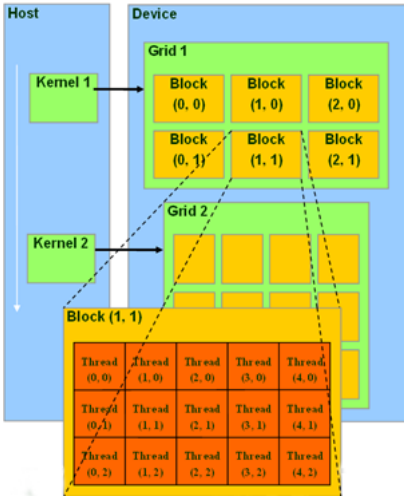


UFRRJ
UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO

CUDA - *Compute Unified Device Architecture*

- Conceitos:
  - GPU: Dispositivo de computação com capacidade de executar *threads* em paralelo.
  - CPU: *Host* que trabalha em conjunto com a GPU.
  - *Kernel*: instância do programa.
  - *Thread*: instância de um *Kernel*.
  - Memória de vídeo / *device*.
  - Memória principal / *host*.

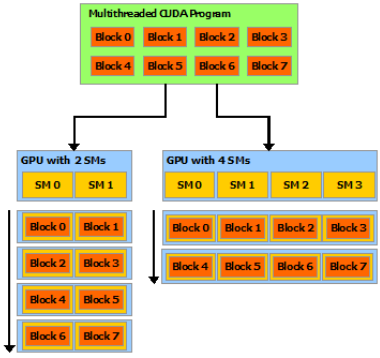
CUDA - *Compute Unified Device Architecture*



- *Threads* & blocos:
  - Organização em blocos de *threads*.
  - Tamanho do bloco dependente do *hardware*.
  - Memória compartilhada.
  - Um bloco por MP.
  - *threads*, blocos, cores e MPs.

UFRRJ
UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO

CUDA - *Compute Unified Device Architecture*



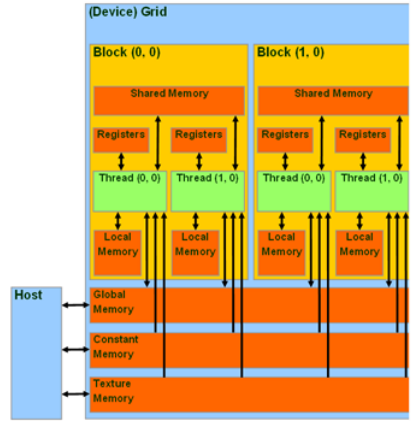
- *Array* de MPs.
- Programa divididos em blocos.
- Independência dos blocos.
- GPUs com diferentes quantidades de MPs (performance).

UFRRJ
UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO

CUDA - *Compute Unified Device Architecture*

- *Threads* & blocos:
  - Local.
  - Cache L1 e L2.
  - Compartilhada.
  - Constante.
  - Textura.
  - Global.

Hierarquia de memória da GPU:



UFRRJ
UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO

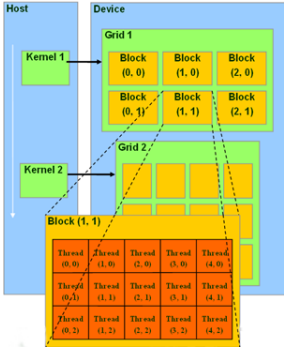
CUDA - *Compute Unified Device Architecture*

Hierarquia de memória da GPU:

CUDA - *Compute Unified Device Architecture*

Organização lógica e física:

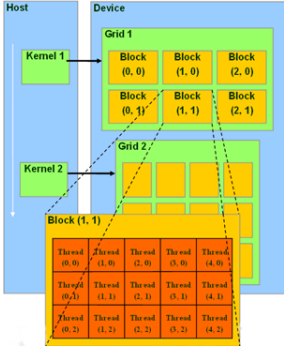
- Blocos × SMs.
- Threads × cores.

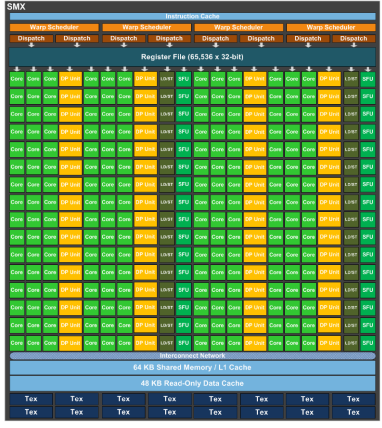


CUDA - *Compute Unified Device Architecture*

Organização lógica e física:

- Blocos × SMs.
- Threads × cores.





UFRRJ
UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO

Estrutura de um programa

1. Inicializações.
2. Alocação das memórias e outros recursos (CPU & GPU).
3. Cópia dos dados CPU $\rightarrow$ GPU.
4. Execução do *kernel*.
5. Cópia dos dados GPU $\rightarrow$ CPU.
6. Liberação da memória e outros recursos alocados (CPU & GPU).

CUDA - Primeiros passos

- Compilador NVCC:
  - Separação dos códigos (*host* e *device*).
  - Compila apenas o código do *device*.
  - Compila o código para formato assembly.
- Exemplo de compilação: **nvcc hello.cu -o hello**.

CUDA - Primeiros passos

Programação a nível de *driver* e/ou *runtime*:

- CUDA *DRIVER*.
- cuda.h
- Requer inicializações
- Prefixo da API (**cuXXX**)
  - **cuInit(...)**
  - **cuDeviceGet(...)**
  - **cuCtxCreate(...)**

- CUDA *RUNTIME*.
- cuda_runtime.h
- Inicializações automáticas
- Prefixo da API (**cudaXXX**)
  - **cudaDeviceReset(...)**
  - **cudaMemGetInfo(...)**
  - **cudaSetDevice(...)**

CUDA - Primeiros passos

- Código - *kernel*:
  - A CPU chama o código em GPU.
  - Definir quantidade de blocos e *threads*.
  - Qualificador de função **__device__** e **__global__**
  - Chamada do *kernel*: **MyKernel <<< blocos, threads >>> (...);**

CUDA - Primeiros passos

- Código - *kernel*:
  - A CPU chama o código em GPU.
  - Definir quantidade de blocos e *threads*.
  - Qualificador de função **__device__** e **__global__**
  - Chamada do *kernel*: **MyKernel <<< blocos, threads >>> (...);**

```
1  __device__ int max(...);
2
3  __global__  void MyKernel(...){
4   ...
5   int m = max(...)
6   ...
7  }
8  int main (){
9   ...
10  ...
11  ...
12  MyKernel<<<1, 10>>> (...);
13  ...
14 }
```

UFRRJ
UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO

CUDA - Primeiros passos

- Código - *kernel*:
  - A CPU chama o código em GPU.
  - Definir quantidade de blocos e *threads*.
  - Qualificador de função **__device__** e **__global__**
  - Chamada do *kernel*: **MyKernel <<< blocos, threads >>> (...);**
  - Blocos e *kernel* em 3D: **dim3 block(10, 10, 2);** e **dim3 threads(2, 2, 2);**.

```
1  __device__ int max(...);
2
3  __global__  void MyKernel(...){
4   ...
5   int m = max(...)
6   ...
7  }
8  int main (){
9   ...
10  dim3 block(10, 10, 2);
11  dim3 threads(2, 2, 2);
12  MyKernel<<<blocks, threads>>> (...);
13  ...
14 }
```

CUDA - Primeiros passos

- Função compilada para CPU & GPU:

```
1  __host__ __device__ void foo(int *out, int *in, int size){
2  #ifdef __CUDA_ARCH__
3      out[threadIdx.x] = in[threadIdx.x] + 1;
4  #else
5      for (int i = 0; i < size; i++){
6          out[i] = in[i] + 1;
7      }
8  #endif
9  }
```

CUDA - Primeiros passos

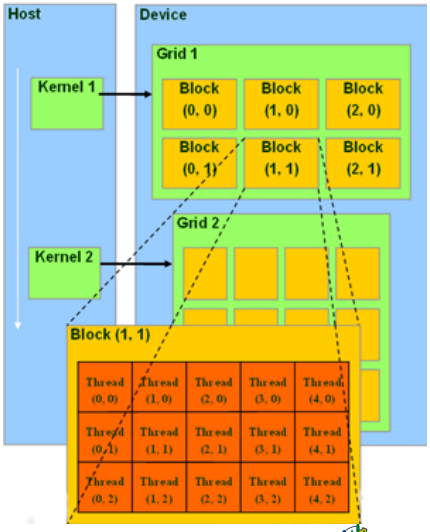
- Captura de erro:

```
#define CHECK_ERROR(call) do {                                  \
  if( cudaSuccess != call) {                                    \
   std::cerr << std::endl << "CUDA ERRO: " <<                   \
   cudaGetErrorString(call) <<  " in file: " << __FILE__ <<     \
   << " in line: " << __LINE__ << std::endl;                    \
   exit(0);                                                     \
 } } while (0)

int main (int argc, char **argv){
  ...
 CHECK_ERROR(cudaDeviceReset());
  ...
 MyKernel<<<1, 10>>> (...);
 CHECK_ERROR(cudaDeviceSynchronize());
}
```

Conhecendo o *kernel*

- Variáveis de ambiente:
  - **gridDim**: dimensão da *grid*.
  - **blockDim**: dimensão do bloco.
  - **blockIdx**: índice do bloco.
  - **threadIdx**: índice da *thread*.

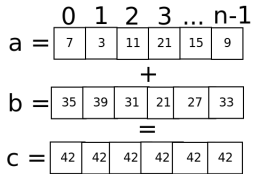


Conhecendo o *kernel*

Problema: somar o conteúdo de dois vetores de n posições, na forma $c = a + b$.

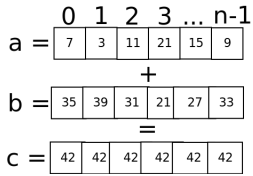
Conhecendo o *kernel*

Problema: somar o conteúdo de dois vetores de n posições, na forma $c = a + b$.



Conhecendo o *kernel*

Problema: somar o conteúdo de dois vetores de n posições, na forma $c = a + b$.

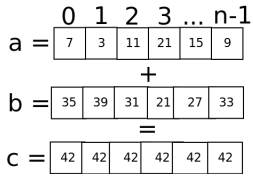


Código em C++:

```
1  void cpu (int *a, int *b, int *c, int n){
2     for (int  i = 0; i < n; i++){
3           c[i] = a[i] + b[i];
4     }
5
6  }
```

Conhecendo o *kernel*

Problema: somar o conteúdo de dois vetores de n posições, na forma $c = a + b$.



Código em C++:

```
1  void cpu (int *a, int *b, int *c, int n){
2     for (int  i = 0; i < n; i++){
3           c[i] = a[i] + b[i];
4     }
5
6  }
```

Código em CUDA: Tamanho vetor e a posição de cada elemento do vetor?!

```
1  __global__ void gpu(int *a, int *b, int *c){
2     const int i = blockDim.x * blockIdx.x + threadIdx.x;
3     c[i] = a[i] + b[i];
4  }
```

UFRRJ
UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO

Conhecendo o *kernel*

```
1  int main (int argc, char **argv){
2      int *h_a, *h_b, *h_c, *d_a, *d_b, *d_c, n = 8;
3      //Reset no device
4      CHECK_ERROR(cudaDeviceReset());
5      //Alocando memoria na GPU:
6      CHECK_ERROR(cudaMalloc((void**) &d_a, n * sizeof(int)));
7      CHECK_ERROR(cudaMalloc((void**) &d_b, n * sizeof(int)));
8      CHECK_ERROR(cudaMalloc((void**) &d_c, n * sizeof(int)));
9      //Alocando memoria na CPU
10     h_a = new int[n]; h_b = new int[n]; h_c = new int[n];
11     for (int i = 0; i < n; i++){ h_a[i] = (i+1); h_b[i] = (i+1) * 10;}
12     //Copia CPU --> GPU
13     CHECK_ERROR(cudaMemcpy(d_a, h_a, n * sizeof(int),  cudaMemcpyHostToDevice));
14     CHECK_ERROR(cudaMemcpy(d_b, h_b, n * sizeof(int),  cudaMemcpyHostToDevice));
15     gpu<<<1, n>>> (d_a, d_b, d_c);
16     CHECK_ERROR(cudaDeviceSynchronize());
17     //Copia dado da GPU para CPU (cudaMemcpyHostToDevice)
18     CHECK_ERROR(cudaMemcpy(h_c, d_c, n * sizeof(int),  cudaMemcpyDeviceToHost));
19     for (int i = 0; i < n; i++){cout << i << ": " << h_c[i] << endl;}
20     //Liberando memorias GPU e CPU
21     CHECK_ERROR(cudaFree(d_a)); CHECK_ERROR(cudaFree(d_b)); CHECK_ERROR(cudaFree(d_c));
22     delete[] h_a; delete[] h_b; delete[] h_c;
23     return EXIT_SUCCESS;
24 }
```

UFRRJ
UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO

UFRRJ
UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO