

# Super aceleração de algoritmos usando GPUs de última geração

**Dr. Mário Gazziro<sup>1</sup>**

mario.gazziro@ufabc.edu.br; mariogazziro@gmail.com

Monitores: Anderson G. Marco<sup>1</sup>, Eduardo M. Real<sup>1</sup> e Daniel M. Lima<sup>2</sup>

<sup>1</sup> Universidade Federal do ABC - UFABC/Santo André-SP

<sup>2</sup> Universidade de São Paulo - ICMC-USP/São Carlos-SP

15/05/2018

## Resumo

Curso prático de elaboração de algoritmos acelerados para GPUs com arquitetura Pascal – será utilizada Titan XP com 3500 núcleos, porém a maioria das técnicas abordadas se aplicam a todas as classes de GPU da fabricante NVIDIA. Após introdução sobre análise do paralelismo de algoritmos, serão apresentados estudos de caso aplicados em ressonância magnética, genética, além do processamento de grafos em escalas de bilhões. Em alguns dos casos apresentados a aceleração ultrapassa 300 vezes, e as técnicas apresentadas, como definição e ajuste fino da grade (GRID) de recursos, escolha dos tipos adequados de memória e técnicas para o escalonador de processos (WARP), podem ser facilmente adaptadas a outros problemas, logo os alunos inscritos são encorajados a trazer seus próprios problemas de pesquisa para paralelização em nossa oficina.

## CV resumido

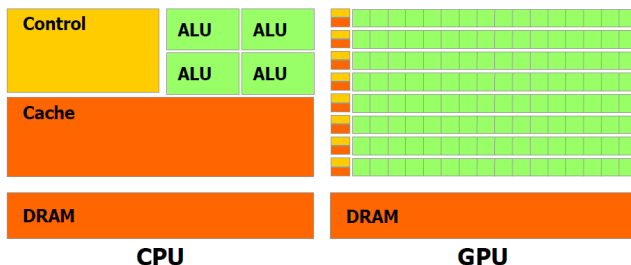
Mário Gazziro possui graduação e doutorado pela USP, com especialização na Toshiba Semiconductor, Kawasaki, Japão. Atualmente é Professor da Universidade Federal do ABC.

- GPU ou GPGPU: Graphics Processing Unit
- Speed-Up
- Speed-Down
- CUDA
- Pascal - arquitetura da titan xp
- Tesla - arquitetura da k20m
- TFlops - Teraflops: "Como a mais precisa e simplista medida de desempenho para capacidade de um processador, um Teraflop se refere ao processador sendo capaz de calcular um trilhão de operações de ponto flutuante por segundo."

# Introdução

- GPU (*Graphics Processing Unit*) [1, 2] é uma arquitetura *a priori* derivada do paradigma DATAFLOW [3], pesquisado na década de 80.
  - ▶ Ultrapassou o mercado de jogos, indo muito além da computação gráfica, sendo amplamente utilizada em computação científica e number crunching.
  - ▶ Tem características diferentes de CPU.
- Em comparação com CPU, o crescimento do número de núcleos é exponencial, embora sejam núcleos relativamente mais simples.
  - ▶ Causalidade e Paralelismo (nem tudo pode ser paralelizado)
  - ▶ Lei de Amdhal [4] (determinação do máximo speed-up teórico com múltiplos processadores).
- Supercomputação para o povão (série de artigos da revista Dr. Dobbs).
  - ▶ Se bem explorado, o poder computacional pode chegar a TFlops.
  - ▶ Porém, um ajuste adequado do problema e da sua paralelização são vitais para bons speed-ups.

# CPU x GPU



**Figure:** The GPU Devotes More Transistors to Data Processing [5]

- uma unidade de computação de GPU é muito mais simples que um moderno núcleo de CPU superescalar.
- uma unidade de computação de GPU não faz previsão de desvio.
- todos os núcleos da GPU executam as mesmas instruções, ao mesmo tempo, mas operam em dados diferentes (SIMD).
- um núcleo da CPU tem um cache grande, previsão de desvio e maior velocidade de clock.

- Uma arquitetura de computação paralela de propósito geral.
  - ▶ um modelo de programação paralela e arquitetura de conjunto de instruções que aproveita o mecanismo de computação paralela nas GPUs NVIDIA para resolver muitos problemas computacionais complexos de uma maneira mais eficiente do que em uma CPU.
- O CUDA vem com um ambiente de software que permite aos desenvolvedores usar o C como uma linguagem de programação de alto nível.
  - ▶ Ideia de manter uma curva de aprendizado baixa para programadores familiarizados com linguagens de programação padrão, como C.

Núcleo com 3 abstrações principais (como um conjunto mínimo de extensões de linguagem)

- 1 Uma hierarquia de grupos de threads.
- 2 Memórias compartilhadas.
- 3 Sincronização de barreiras.

- Particionar o problema em sub-problemas que podem ser resolvidos independentemente em paralelo por blocos de threads.
- Cada sub-problema em partes que podem ser resolvidas cooperativamente em paralelo por todos os threads dentro do bloco.

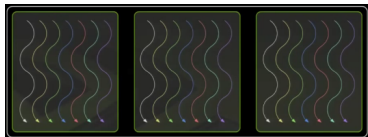
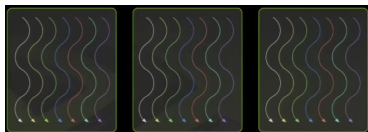
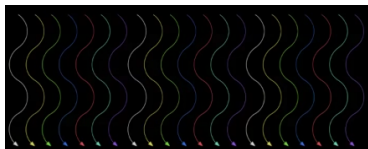
# Modelo de Programação

- host (CPU): executa a aplicação.
- device (GPU): executa **kernels**.
- host e device tem DRAMs próprias.
- host:
  - ▶ Aloca memória no device.
  - ▶ Transfere dados de entrada para o device.
  - ▶ Dispara a execução de kernels.
  - ▶ Transfere dados resultantes do device.
  - ▶ Libera memória no dispositivo.
- Kernel: executa no device N vezes em N threads em paralelo.



# Modelo de Programação

- Threads são organizadas em **blocos**.
- Um bloco é um arranjo 1D, 2D ou 3D (índices) de **threads**.
- Blocos são organizados em **grids**.
- Um grid é um arranjo 1D ou 2D (índices) de blocos.
  - ▶ Os blocos de um grid têm o mesmo número de threads.
- Hierarquia de threads:
  - ▶ Threads.
  - ▶ Bloco de threads (blocks).
  - ▶ Grade de blocos (grid).



# Modelo de Programação

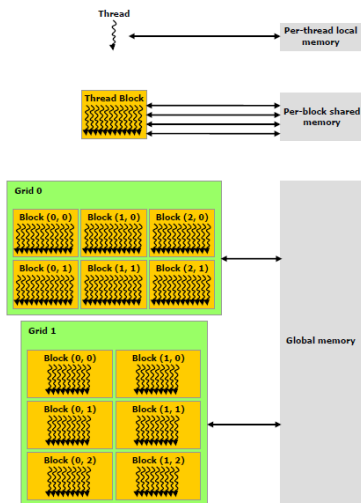


Figure: Memory Hierarchy [5]

# Modelo de Programação

Grid de 2 dimensões com blocos de threads com 2 dimensões:

- 1 Grid 2D com:
  - ▶  $3 \times 2 \times 1 = 6$  blocos
- 2 Blocos 2D com:
  - ▶  $4 \times 3 \times 1 = 12$  threads cada um.

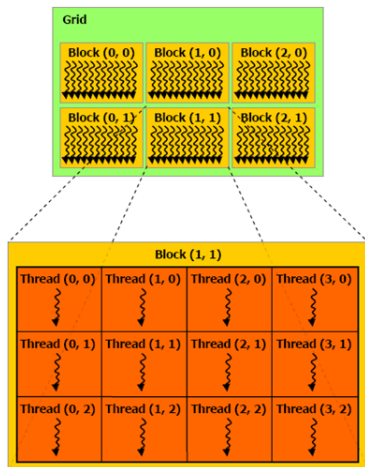


Figure: Grid of Thread Blocks [5]

# Modelo de Programação

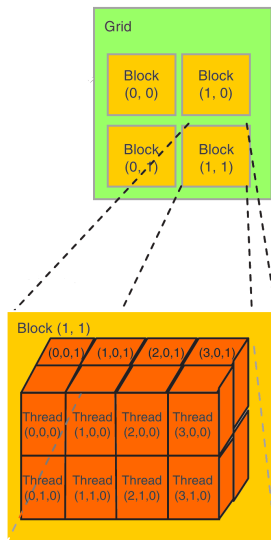
Grid de 2 dimensões com blocos de threads com 3 dimensões:

1 Grid 2D com:

▶  $2 \times 2 \times 1 = 4$  blocos

2 Blocos 3D com:

▶  $4 \times 2 \times 2 = 16$  threads cada um.



# Modelo de Programação

## Identificação das threads

- Block
  - ▶ Thread ( $tx, ty, tz$ )
  - ▶ `threadIdx.dim`, i.e., (`threadIdx.x`, `threadIdx.y`, `threadIdx.z`)
- Grid
  - ▶ Block ( $bx, by, bz$ )
  - ▶ `blockIdx.dim`, i.e., (`blockIdx.x`, `blockIdx.y`, `blockIdx.z`)
- Gerações CUDA com diferentes número máximo de threads (block) e de cada dimensão (Grid)!

# Modelo de Programação

## Identificação das threads

Especificação	Compute Capability					
	1.0	1.1	1.2	1.3	2.0	2.1
Número de dimensões da grade de blocos	2				3	
Tam. máx. de cada dimensão na grade	65535					
Número de dimensões do bloco de threads	3					
Tam. máx. das dimensões x e y no bloco	512				1024	
Tam. máx. da dimensão z no bloco	64					
Núm. máx. threads no bloco	512				1024	

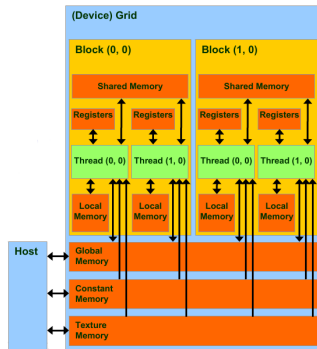
• e.g.

- ▶ Block, (threadIdx.x, threadIdx.y, threadIdx.z):
  - Tam. máx. cada dim: ( 512, 512, 64 ); máx. threads = 512.
  - Tam. máx. cada dim: ( 1024, 1024, 64 ); máx. threads = 1024.
- ▶ Grid, (blockIdx.x, blockIdx.y):
  - Tam. máx. cada dim: ( 65535, 65535 );
- ▶ Grid, (blockIdx.x, blockIdx.y, blockIdx.z):
  - Tam. máx. cada dim: ( 65535, 65535, 65535 );

# Modelo de Programação

Acesso da thread:

- compartilhada do bloco
  - ▶ Visível para todas e tempo de vida do bloco.
- local e registradores da thread.
- global, constante e leitura do Grid.

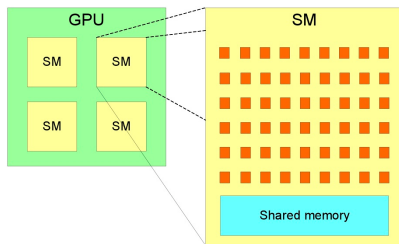


Tipo	Escopo	Acesso	Velocidade	Tempo de Vida
Registrador	Thread	R/W	Rápido	Kernel
Local	Thread	R/W	Lento	Kernel
Compartilhada	Bloco	R/W	Rápido	Kernel
Global	Grade	R/W	Lento	Aplicação
Constante	Grade	R/O	Rápido	Aplicação
Textura	Grade	R/O	Rápido	Aplicação

# Modelo de Programação

## SM:

- 1 core CUDA é 1 Streaming Processor (SP).
- 1 Streaming Multiprocessor (SM) é constituído por um número de SP.
- Quantidade de SP por SM (e.g. 8, 32, 48) depende da Compute Capability da GPU.
- As threads são atribuídas aos SMs em blocos.

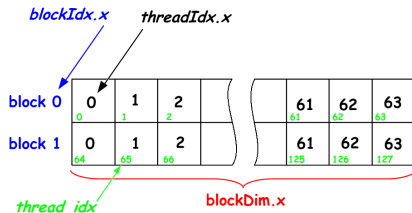




# Modelo de Programação

Mapeando o índice do thread no bloco (e.g.1):

- # blocos:  $blockDim.x$ 
  - ▶ o  $x$  indica que é possível mais de uma dimensão.
- $blockDim.x$  é a linha  
 $threadIdx.x$  é a coluna



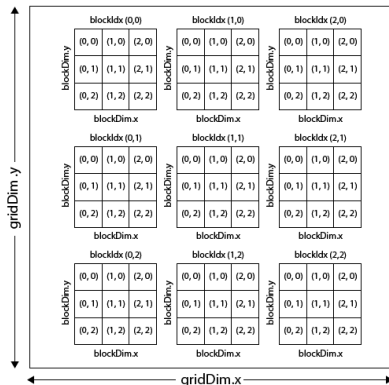
$$thread\_idx = blockDim.x * blockIdx.x + threadIdx.x$$

# Modelo de Programação

Mapeando o índice do thread no bloco (e.g.2):

- # elementos por dimensão: *blockDim.x* e *blockDim.y*
- id bloco: *blockDim.x*, *blockDim.y*
- Thread como uma matriz 2D: *thread[idx][idy]*

## CUDA Grid



```
idx = (blockIdx.x * blockDim.x) + threadIdx.x;  
idy = (blockIdx.y * blockDim.y) + threadIdx.y;  
thread_idx = (gridDim.x * blockDim.x * idy) + idx;
```

## CUDA Thread Indexing Cheatsheet

<https://cs.calvin.edu/courses/cs/374/CUDA/CUDA-Thread-Indexing-Cheatsheet.pdf>

```

Device 0: "TITAN Xp"
  CUDA Driver Version / Runtime Version      9.1 / 9.1
  CUDA Capability Major/Minor version number: 6.1
  Total amount of global memory:             12196 MBytes (12788498432
bytes)
  (30) Multiprocessors, (128) CUDA Cores/MP: 3840 CUDA Cores
  GPU Max Clock rate:                        1582 MHz (1.58 GHz)
  Memory Clock rate:                         5705 Mhz
  Memory Bus Width:                          384-bit
  L2 Cache Size:                             3145728 bytes
  Maximum Texture Dimension Size (x,y,z)     1D=(131072), 2D=(131072,
65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers 1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers 2D=(32768, 32768), 2048 layers
  Total amount of constant memory:            65536 bytes
  Total amount of shared memory per block:    49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                  32
  Maximum number of threads per multiprocessor: 2048
  Maximum number of threads per block:        1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                       2147483647 bytes
  Texture alignment:                           512 bytes
  Concurrent copy and kernel execution:       Yes with 2 copy engine(s)
  Run time limit on kernels:                   No
  Integrated GPU sharing Host Memory:          No
  Support host page-locked memory mapping:     Yes
  Alignment requirement for Surfaces:          Yes
  Device has ECC support:                      Disabled
  Device supports Unified Addressing (UVA):     Yes
  Supports Cooperative Kernel Launch:          Yes
  Supports MultiDevice Co-op Kernel Launch:    Yes
  Device PCI Domain ID / Bus ID / location ID: 0 / 1 / 0
  Compute Mode:
    < Default (multiple host threads can use ::cudaSetDevice() with device
simultaneously) >

```

- Programa básico em C para CUDA:

- ▶ Dispositivo a ser usado (`cudaSetDevice()`);
- ▶ Alocação de memória no host;
- ▶ Dados de entrada na memória do host;
- ▶ Alocação de memória no device (`cudaMalloc()`);
- ▶ Transferência de dados do host para device (`cudaMemcpy()`);
- ▶ Invocação do(s) kernel(s);
- ▶ Transferência de dados do device para host;
- ▶ Liberação de memória no host;
- ▶ Liberação de memória no device (`cudaFree()`).

# Exemplos

*Discutir e programar 1 ou 2 exemplos...*






# Estudo de caso 1

# Estudo de caso 2



# Estudo de caso 3

# Referências

-  NVIDIA Corporation. Graphics Processing Unit (GPU). Disponível em: <http://www.nvidia.com/object/gpu.html>. Acesso em: mai-2018.
-  NVIDIA Corporation. CUDA GPUs. Disponível em: <https://developer.nvidia.com/cuda-gpus>. Acesso em: mai-2018.
-  SOUSA, T. B. Dataflow Programming: Concept, Languages and Applications. In. Doctoral Symposium on Informatics Engineering. 2012.
-  RODGERS, D. P. Improvements in multiprocessor system design. In. Proceedings of the 12th annual international symposium on Computer architecture. Vol.13, 1985, p.225-231.
-  NVIDIA CUDA. NVIDIA CUDA C Programming Guide. Version 4.2, 2012.