Universidade Federal do Amazonas
Faculdade de Tecnologia
Departamento de Eletrônica e Computação

Computação Paralela (CUDA)

Hussama Ibrahim
hussamaibrahim@ufam.edu.br

Notas de Aula

Baseado nas Notas de Aula do Prof. Ricardo Rocha da Universidade do Porto, PT.
http://www.dcc.fc.up.pt/~ricroc/aulas/0607/cp/apontamentos/parte1.pdf

Baseado nas Notas de Aula do Prof. Marcelo Zamith da Universidade Federal Fluminense – UFF, BR.

Documentação NVIDIA CUDA
http://docs.nvidia.com/cuda/

Motivação

"Se uma mãe leva 9 meses para ter 1 bebe,
podem 9 mães podem ter um bebe em um mês?"

"Se um processamento leva 9 horas em um computador,
podem 9 computadores efetuar o processamento em 1 hora?"

- No que consiste a **Computação Paralela**?
  - Concorrência: Identificação das partes que podem ser executadas em qualquer ordem, sem alterar o resultado final
  - Scheduling: Distribuição do processamento de forma eficiente entre os nós de processamento disponíveis
  - Comunicação e Sincronização: Desenhar a computação para ser executada nos vários nós de processamento

Objetivo

- Conseguir reduzir o tempo de execução de um determinado processamento / programa, aproveitando melhor o *hardware* que temos a disposição

- Conceitos Fundamentais:
  - Potencial Paralelismo
  - Paralelismo
  - Paralelismo Explícito
  - Paralelismo Implícito
  - Comunicação Sincrona e Assincrona
  - Computação Paralela

Potencial Paralelismo

- Ocorre quando um programa possui tarefas que podem ser executadas em diferentes sequências, sem alterar o resultado final

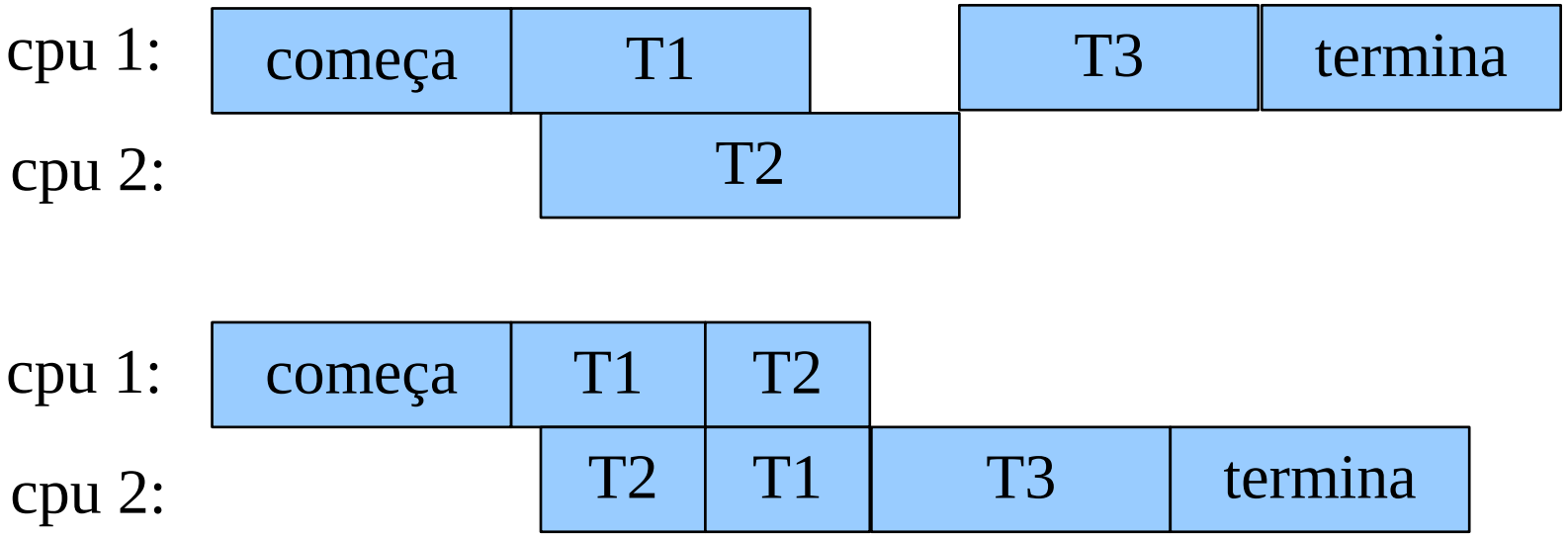
começa	T1	T2	T3	termina
começa	T2	T1	T3	termina
começa	T3	T2	T1	termina
começa	T3	T1	T2	termina

...

Paralelismo

- Diferente do pseudoparalelismo

- É uma execução paralela quando as tarefas de um programa são executadas ao mesmo tempo numa máquina de dois ou mais processadores.



Paralelismo Explícito

- É definido pelo **programador** e papel dele:
  - Definir as tarefas que executarão em paralelo
  - Atribuir as tarefas aos processadores
  - Controlar a Execução
    - Pontos de Sincronização
  - Efetuar "*tunnings*“ no código afim de obter um maior desempenho

(+) É possível obter melhores resultados

(-) O programador é responsável por todos os detalhes da implementação

(-) Pouco portável entre as diferentes arquiteturas

Paralelismo Implícito

- O paralelismo é responsabilidade do **compilador** e do próprio **sistema de execução**:
  - Encontra os potenciais paralelismos
  - Atribui as tarefas para execução em paralelo
  - Cuidar de toda a parte de sincronização

- Vantagens / Desvantagens:

  (+) Abstrai do programador os detalhes da execução paralela

  (-) Difícil de obter uma solução mais eficiente

Conceitos Fundamentais

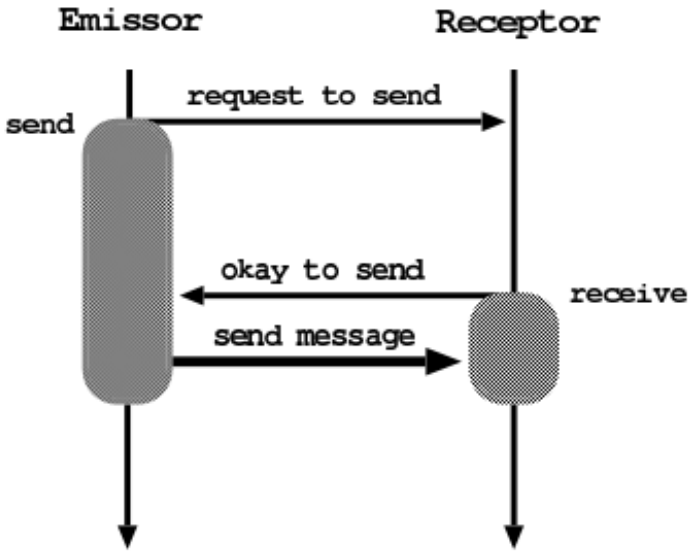
- Conceitos Fundamentais:
  - ~~Potencial Paralelismo~~
  - ~~Paralelismo~~
  - ~~Paralelismo Explícito~~
  - ~~Paralelismo Implícito~~

  } Implementação

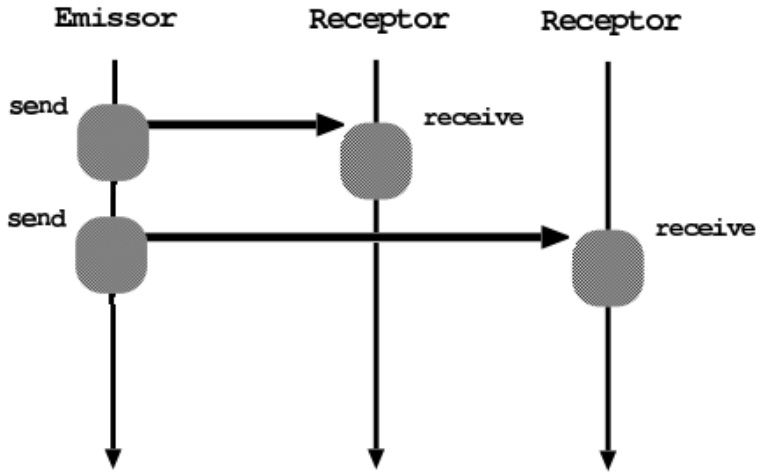
  - Comunicação Síncrona e Assíncrona
  - Computação Paralela
  - GPU ~ Computação Paralela utilizando CUDA

Comunicação Síncrona e Assíncrona

- Síncrona: É feita de maneira de maneira **simultânea**, **coordenada** e **sincronizada**
- Assíncrona: Ocorre de maneira não **sincronizada**.



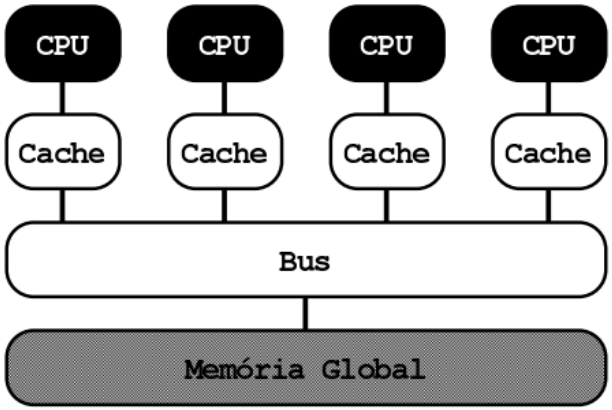
Comunicação Síncrona

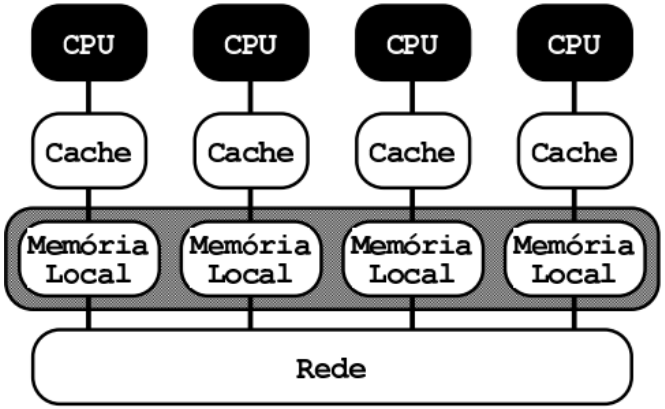


Comunicação Assíncrona

Computação Paralela

- É considerada computação paralela um programa que é executado sobre uma máquina com **vários processadores**, e esse **vários processadores** compartilham acesso ao mesmo bloco de memória
- Dividas em duas sub-classes:
  - *Centralized Multiprocessor* / *Uniform Memory Access*
  - *Distribuited Multiprocessor* / *Nonuniform Memory Access*





GPU

Graphic Processing Unit

Computação Paralela com GPU

- Atualmente a taxa de *clock* dos computadores modernos varia entre 4 GHz
  - Maior consumo de energia
  - Maior dissipação de calor
  - Alto custo em refrigeração

- Fabricantes como Intel e AMD vão na contra mão, gerando processadores com mais núcleo, e com um aumento gradativo do *clock.*
  - dual, tri, quad, 8, 12, 16, 32 *cores*.

Computação Paralela com GPU

- Uma empresa contendo *Clusters* com milhares de processadores trabalhando na execução de algoritmos complexos.

Computação Paralela com GPU

- Computação **Heterogênea**: Utilização do poder de processamento de uma CPU com uma GPU, trabalhando em conjunto e alcançando alta performance.

TOP 10 Sites for November 2014

For more information about the sites and systems in the list, click on the links or view the complete list.

RANK	SITE	SYSTEM	CORES	RMAX (TFLOP/S)	RPEAK (TFLOP/S)	POWER (KW)
1	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
2	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
3	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
5	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945

http://www.top500.org/lists/2014/11/

Computação Paralela com GPU

- Computação **Heterogênea**: Utilização do poder de processamento de uma CPU com uma GPU, trabalhadno em conjunto e alcançando alta performance.

The Green500 List

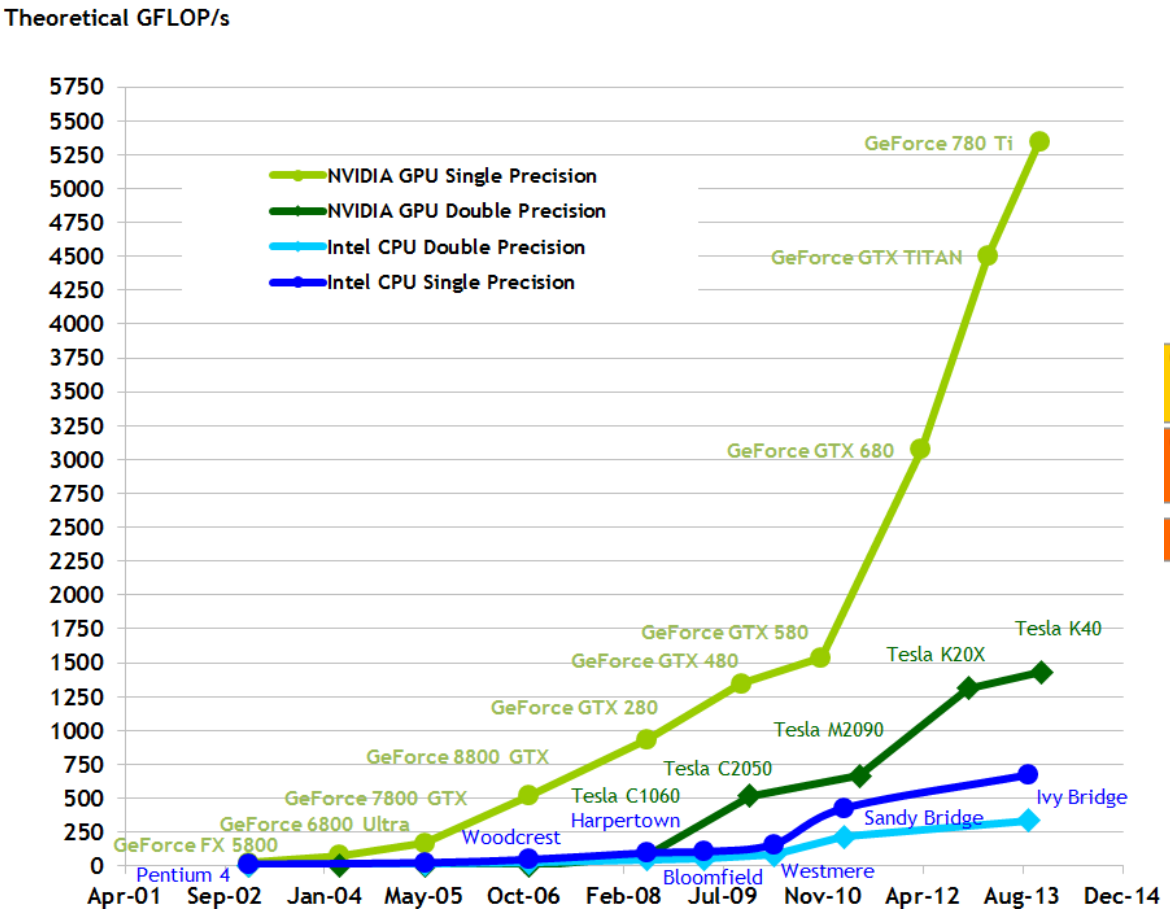
Listed below are the November 2014 The Green500's energy-efficient supercomputers ranked from 1 to 10.

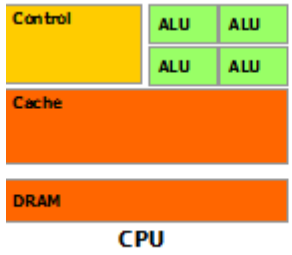
Green500 Rank	MFLOPS/W	Site*	Computer*	Total Power (kW)
1	5,271.81	GSI Helmholtz Center	L-CSC - ASUS ESC4000 FDR/G2S, Intel Xeon E5-2690v2 10C 3GHz, Infiniband FDR, AMD FirePro S9150 Level 1 measurement data available	57.15
2	4,945.63	High Energy Accelerator Research Organization /KEK	Suiren - ExaScaler 32U256SC Cluster, Intel Xeon E5-2660v2 10C 2.2GHz, Infiniband FDR, PEZY-SC	37.83
3	4,447.58	GSIC Center, Tokyo Institute of Technology	TSUBAME-KFC - LX 1U-4GPU/104Re-1G Cluster, Intel Xeon E5-2620v2 6C 2.100GHz, Infiniband FDR, NVIDIA K20x	35.39
4	3,962.73	Cray Inc.	Storm1 - Cray CS-Storm, Intel Xeon E5-2660v2 10C 2.2GHz, Infiniband FDR, Nvidia K40m Level 3 measurement data available	44.54
5	3,631.70	Cambridge University	Wilkes - Dell T620 Cluster, Intel Xeon E5-2630v2 6C 2.600GHz, Infiniband FDR, NVIDIA K20	52.62

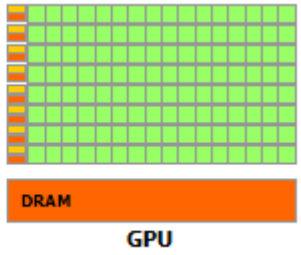
http://www.green500.org/lists/green201411

Computação Paralela com GPU

- GPU's possuem capacidade maior de processamento, tratando de **operações em ponto flutuante por segundo (TFLOP/s)**







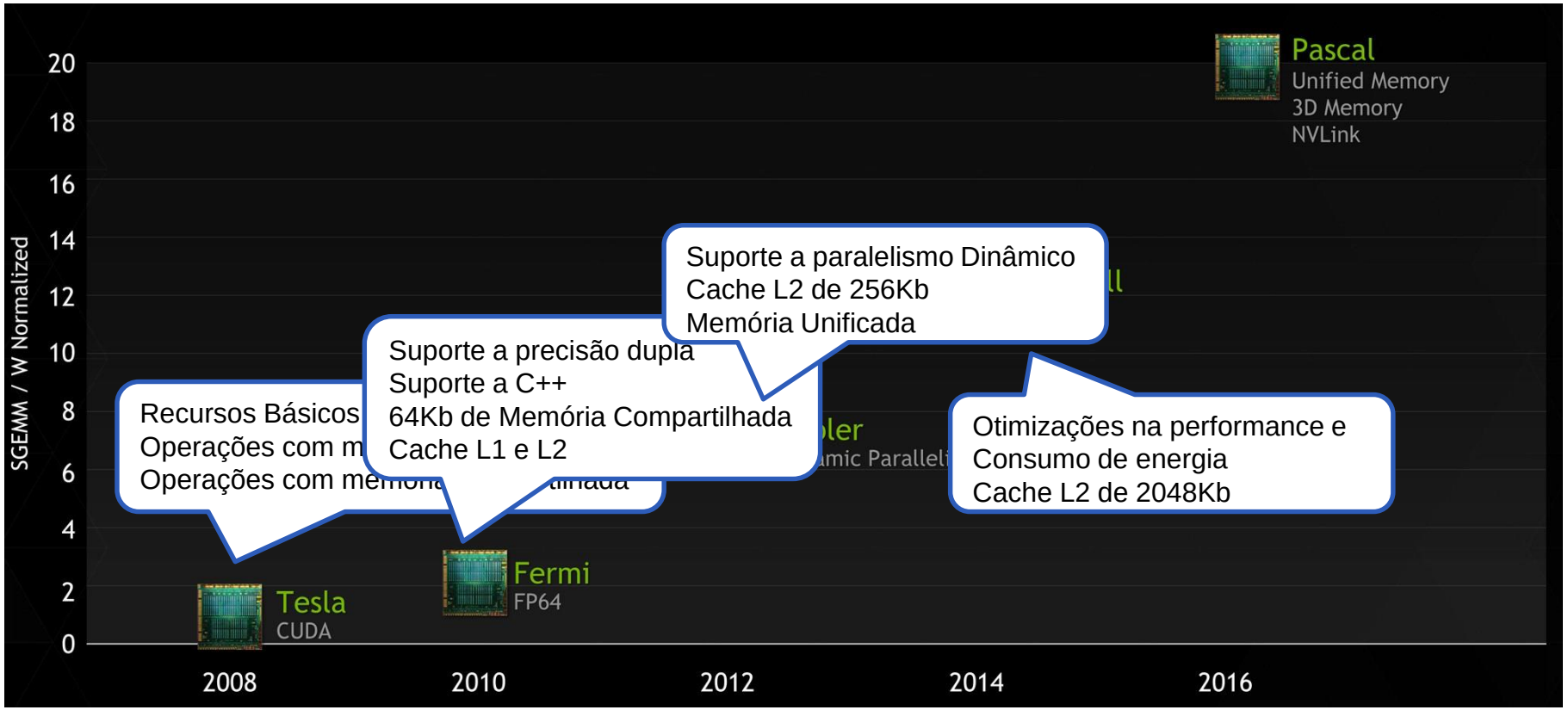
Computação Paralela com GPU

- **NVIDIA**
  - Vídeo (https://www.youtube.com/watch?v=8ZGBYod4kW0)
- **CUDA** (Compute Unified Device Architecture)
  - Capacidade programar a GPU para realizar propósitos gerais (i.e., não usar apenas para aplicações gráficas)
  - Biblioteca FORTRAN, **C/C++**
  - API gerenciar dispositivos, memórias e etc
  - Inicio na série 8 (2006)

Compilador **NVCC**

Compute Unified Device Architecture

- Suas características e funcionalidades são evolutivas e definidas por sua arquitetura, porém mantendo retrocompatibilidade.



Compute Unified Device Architecture

- Como saber a arquitetura da minha GPU com suporte ao CUDA? (*Compute Capability*)

```
lspci | grep –i nvidia
```

http://developer.nvidia.com/cuda-gpus

Versão	Recurso
1.0	Recursos básicos
2.0	Arquitetura Fermi
3.0	Arquitetura Kepler

- Posso ainda compilar um programa para uma determinada arquitetura passando os parametros **-arch** e -**code** para o compilador

```
nvcc <file>; ./a.out
```

Compute Unified Device Architecture

- **Threads:** Um pequeno programa que trabalha como um sub-sistema de um programa maior.
  - CPU's não são boas para trabalhar com um número grande de threads.
  - Custo de Gerenciamento
  - GPU's: feitas para trabalhar com grande número de threads, devido utilizar um outro paradigma.
- **GPU:** Dispositivo de computação capaz de executar muitas threads em paralelo (**device**)
- **CPU:** Dispositivo que trabalha em conjunto com a CPU (**host**)

Compute Unified Device Architecture

- **Kernel:** Definição de um programa que irá rodar na GPU
- **Thread:** Instância de um Kernel
- **Blocos:** Grupo ou agrupador de *Threads*
- **Grid:** Grupo ou agrupador de Blocos

- Vídeo (https://www.youtube.com/watch?v=-P28LKWTzrI)

- O “canhão” seria o *kernel*, pois nele estaria a definição do que se quer fazer.
- E cada tubo disparado em paralelo seriam as *threads*.

Problemas Unidimensionais

Problemas Unidimensionais com CUDA

- Primeiro programa em CUDA
- Soma de Vetores:

A		B		C
1		**6**		**7**
2		7		9
3	+	8	=	11
4		9		13
5		10		15

Host:

```
int i=0;
for (i=0; i<5; i++){
   c[i] = a[i] + b[i];
}
```

Device:

```
__global__ void my_kernel(int * a, int * b, int *c){
    int i = threadIdx.x;
    c[i] = a[i] + b[i];
}
```

- Visualizar Código no NSIGHT

Problemas Unidimensionais com CUDA

- O CUDA permite o número máximo **1024** threads por bloco
- E se meu vetor de soma possuísse 10240 posições?

```
my_kernel<<<10,1024>>>(a,b,c);
```

```
__global__ void my_kernel(int * a, int * b, int *c){
    int i = (blockDim.x * blockIdx.x) + threadIdx
    c[i] = a[i] + b[i];
}
```

- Visualizar Código no NSIGHT

Problemas bidimensionais

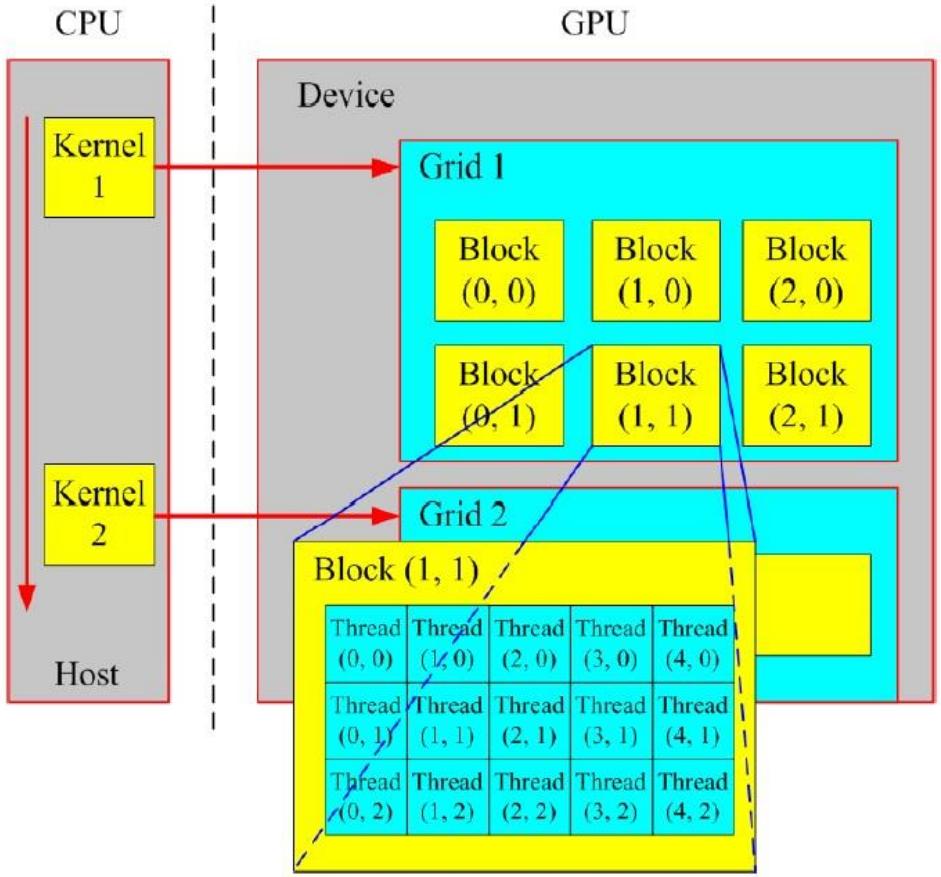
Problemas Bidimensionais com CUDA

- **Blocos:** Na GPU, podemos ter blocos em mais de uma dimensão (x,y)

- **Threads:** Podemos ter threads em 1, 2 ou 3 dimensões (x,y,z)

- Utilização do objeto **dim3(x,y,z)**

```
struct dim3 {
    int x;
    int y;
    int z; }
```



Problemas Bidimensionais com CUDA

- A dica é tratar um problema bidimensional como se fossemos trabalhar de forma unidimensional. Padronizando o acesso com uma função de normalização.
- Multiplicação de Matrizes:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

*

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

=

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

- Desenvolver o código para execução no HOST

Problemas Bidimensionais com CUDA

– Solução com 2 Blocos e 2 threads em cada eixo

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

*

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

=

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

```
__global__ void my_kernel(int * a, int * b, int *c){
    int i = (blockDim.x * blockIdx.x) + threadIdx .x;
    int j = (blockDim.y * blockIdx.y) + threadIdx.y;

    for(int k=0; k<dim; k++){
        c[i][j] += a[i,k] + b[k,j];
    }
}
```

Conclusões

- Podemos utilizar o CUDA para diversas aplicações visando melhor *performance* no tempo de execução.
- O recurso de paralelismo dinâmico do CUDA é interessante pois permite chamadas recursivas kernel, utilização de grafos, árvores.
- **SPIN** (verificador de modelos da NASA) apresentou uma redução de 7x do tempo de execução da verificação após a implementação em GPU.
- Obrigado! ☺