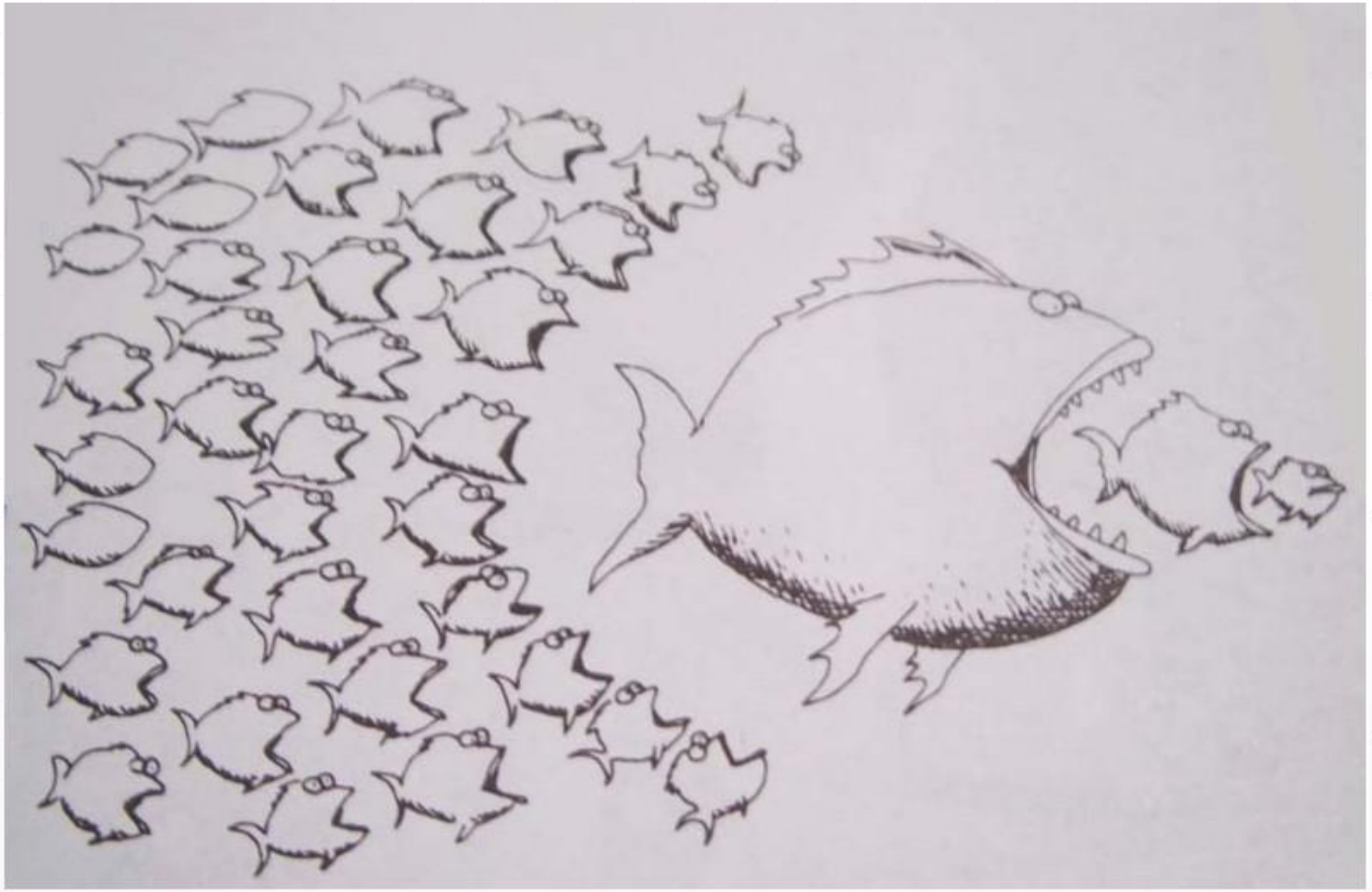
# Compute Unified Device Architecture - CUDA

Pargles Dall'Oglio

# Introdução

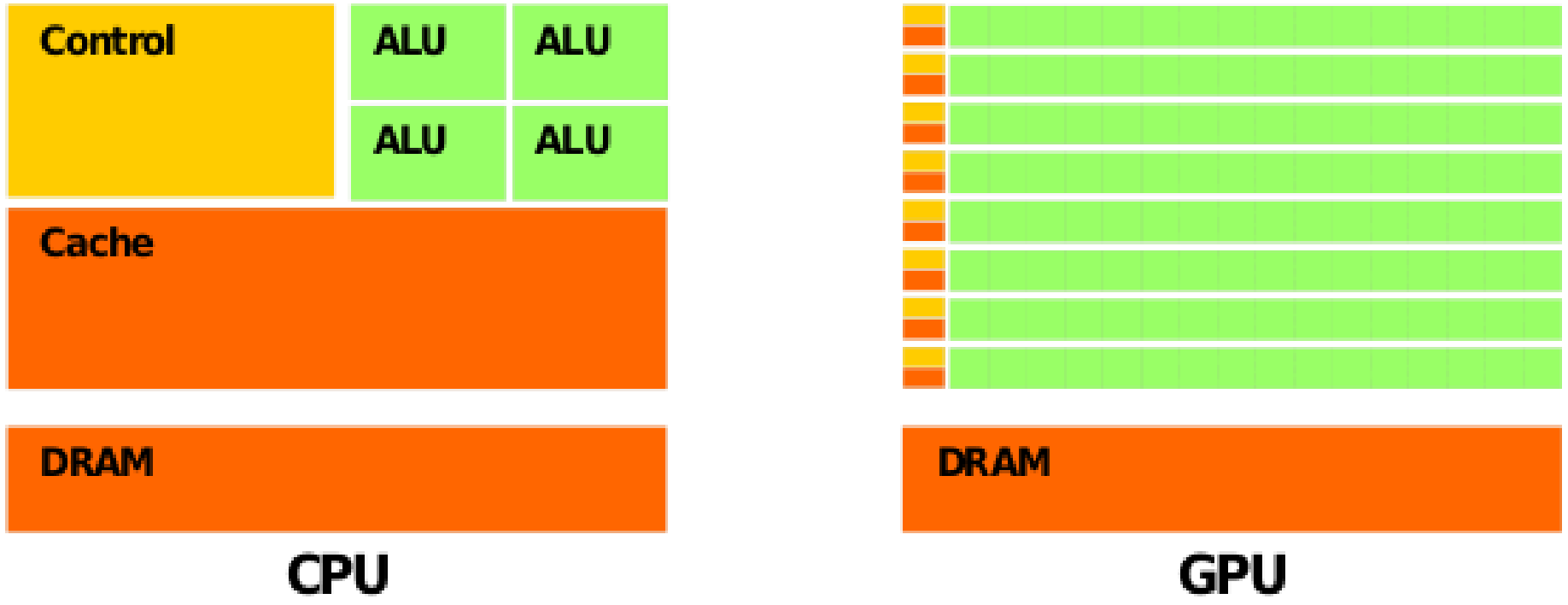
- Primeiras GPUs eram apenas aceleradores gráficos
- Cientistas começaram a utilizar a excelente performance de ponto flutuante
- 2006 - Primeira solução do mundo para computação de propósito geral em GPUs
- GPGPU - *General-Purpose computation on Graphics Processing Units*
- Utiliza diretivas de compilação, inclui bibliotecas e possui extensões para as principais linguagens de programação

# Benefícios



# Benefícios

- Desempenho
  - programa executa porção dos mesmos dados, unidade de controle de fluxo simples
  - mais transistores para processamento ao invés de cache e/ou controle de fluxo

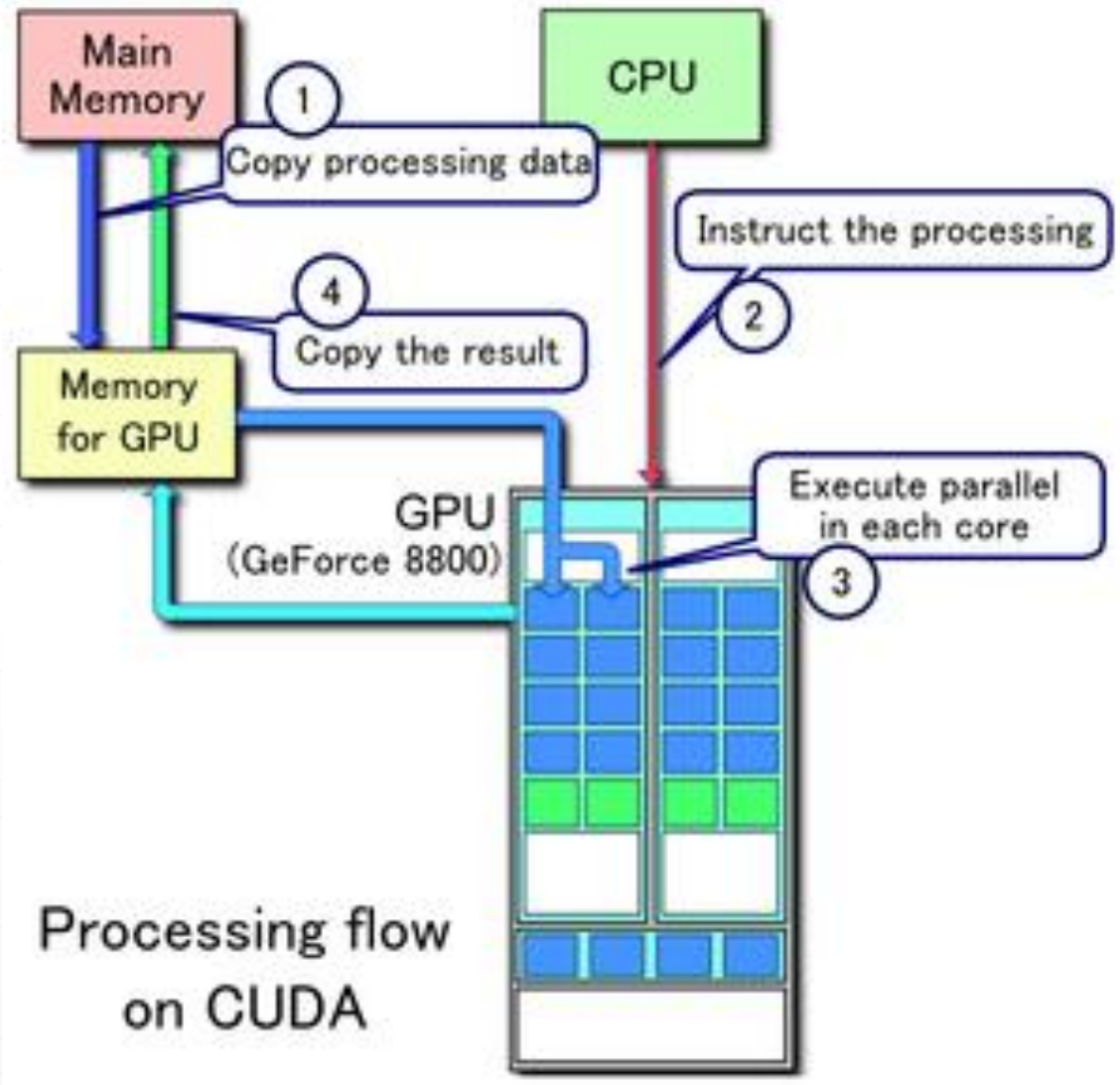


# Funcionamento

- Inicia a execução do programa

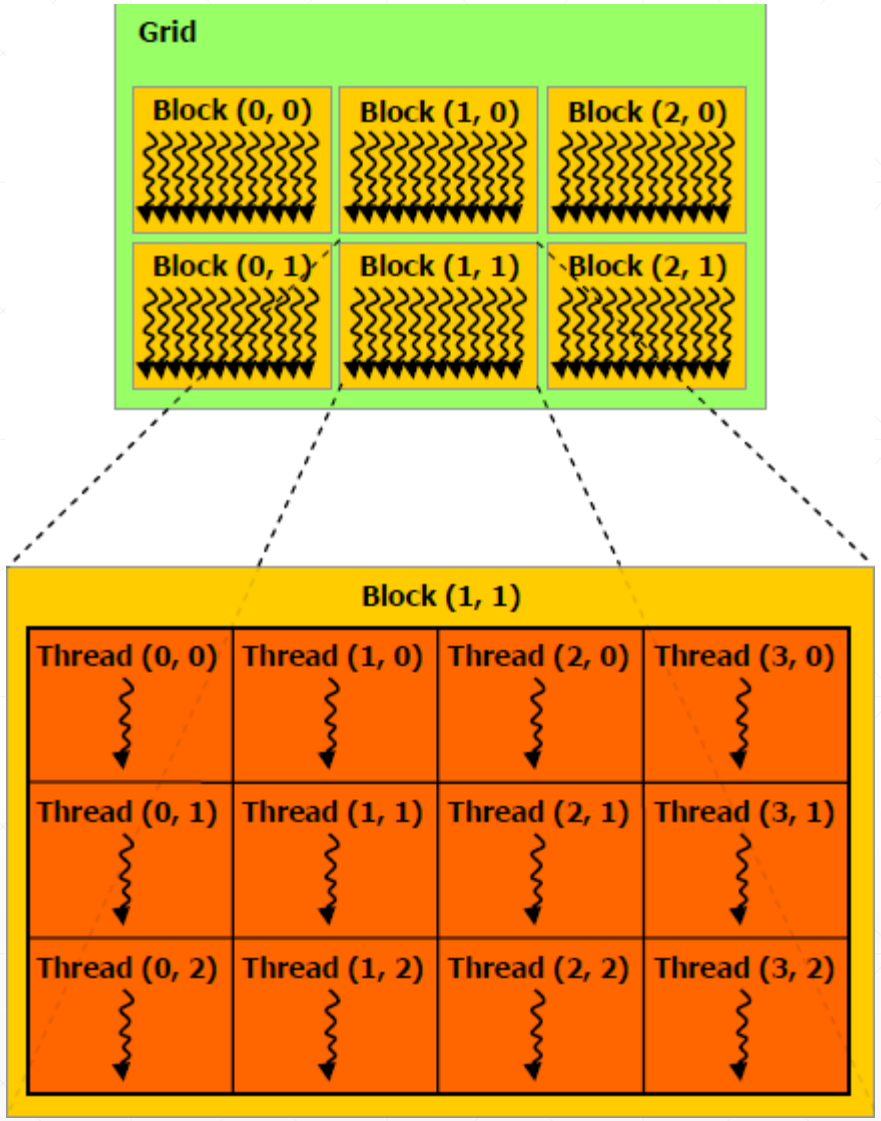
1. Realiza cópia dos dados a serem processados da memória do PC para a memória da GPU
2. Programa faz chamada a GPU
3. Processamento paralelo
4. Resultados são devolvido à memória do PC

- Retorna execução normal do programa



# Funcionamento

- Conceito de grade (grid) e blocos (block)
- A grade representa uma estrutura abstrata em forma de matriz, onde cada posição contém um bloco
- Cada bloco contem uma matriz de *threads*, limitada
- Cada bloco é mapeado a um multiprocessador automaticamente
- Quantidade de blocos independe da quantidade de multiprocessadores
- Um programa pode conter centenas de threads, existe um escalonador em cada multiprocessador da GPU



# Modelo de programação

- Utiliza diretivas de compilação
- <<< tamanho do grid , tamanho de cada bloco >>>
- __host__ = funções executadas na CPU
- __global__ = funções chamadas pela CPU e executadas pela GPU
- __device__ = funções chamadas e executadas apenas na GPU
- __shared__ = variáveis acessíveis a um bloco de threads
- __device__ = variáveis acessíveis tanto pela GPU quanto pela CPU
- __constant__ = parecido com variáveis __device__

# Modelo de programação

- __syncthreads()
- cudaMalloc(void **pointer, size_t nbytes)
- cudaMemset(void *pointer, int value, size_t count)
- cudaFree(void *pointer)
- cudaMemcpy(void *dst, void *src, size_t nbytes, enum cudaMemcpyKind direction);

# Modelo de programação

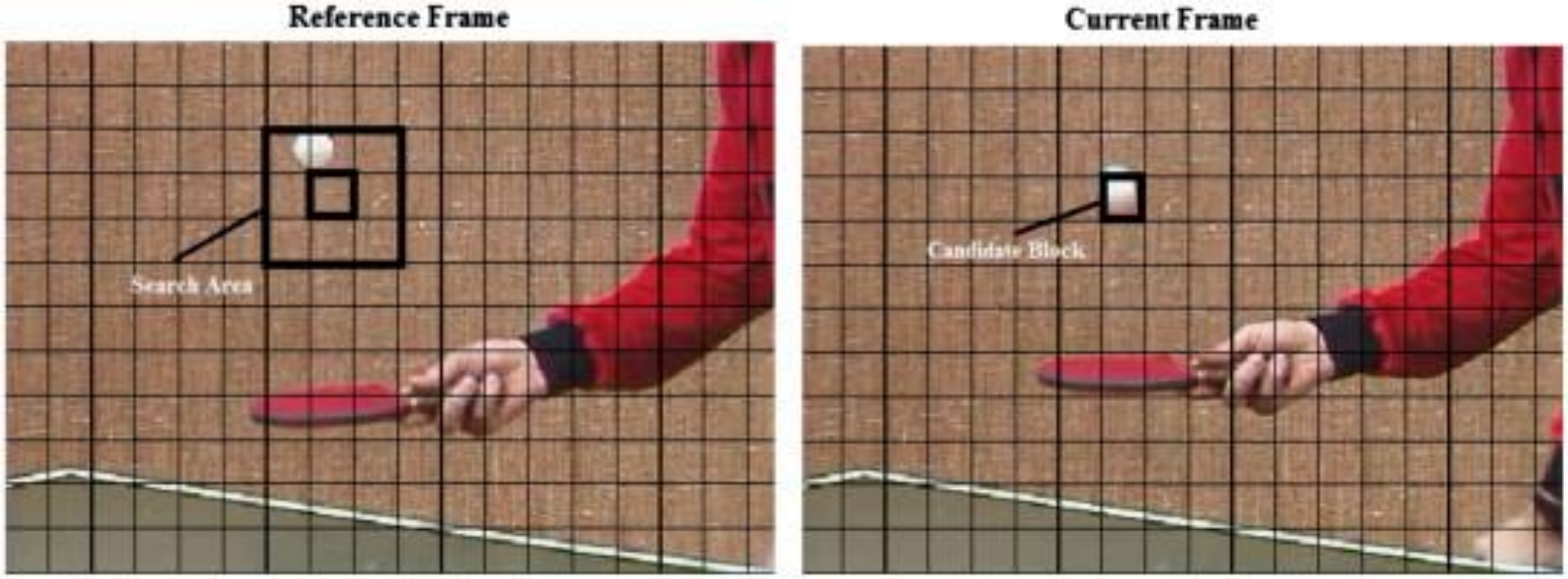
```
1.  #include "stdio.h"
2.  __global__ void add(int a, int b, int *c)
3.  {  *c = a + b;
4.   }
5.  int main() {
6.  int a,b,c;
7.  int *dev_c;
8.  a=3;
9.  b=4;
10. cudaMalloc((void**)&dev_c, sizeof(int));
11. add<<<1,1>>>(a,b,dev_c);
12. cudaMemcpy(&c, dev_c, sizeof(int), cudaMemcpyDeviceToHost);
13. printf("%d + %d is %d\n", a, b, c);
14. cudaFree(dev_c);
15. return 0;
16. }
```

# Aplicações

- Processamento de Imagens e Vídeo
- Criptografia
- Simulação de Dinâmica de Fluidos
- Análise Sísmica
- Visualização de moléculas
- Previsão do tempo
- Jogos, é claro

# Processamento de Vídeos

- Necessário comparar cada bloco de um frame de referência com os blocos do atual
- Alta definição = 1920x1080 = (2.073.600 pixeis / blocos de 8 pixeis) = (259.200 blocos * 30 frames por segundo) =  7.776.000 blocos em um segundo



# Deficiências

- CUDA  é uma tecnologia da NVidia, portanto, incompatível com demais arquiteturas
- Não faz milagres
- Muito utilizada em pesquisas, mas seu crescimento depende de empresas que forneçam serviços e soluções que utilizam essa tecnologia
- Ou seja, como o usuário comum pode obter alguma vantagem na utilização de potentes placas gráficas?

# Linguagens Suportadas

- C e C++
- Fortran
- Java
- Python
- Ruby
- Lua
- Haskell

# Quem já utiliza

- Adobe Premiere, AfterEffects, Photoshop (editor de vídeos, efeitos visuais e fotos)
- ANSYS (simulador de eventos)
- Autodesk ( Modelos 3D)
- MathWorks  MatLab (computacão matemática)
- Wolfram  (computacão matemática)

Obrigado

# Compute Unified Device Architecture

Pargles Dall’Oglio