Programação Paralela: das *threads* aos FPGAs

GPU's - *Graphic Processing Units*

Prof. Ricardo Menotti
menotti@ufscar.br

Prof. Maurício Acconcia Dias
macccdias@gmail.com

Prof. Helio Crestana Guardia
helio.guardia@ufscar.br

Departamento de Computação
Universidade Federal de São Carlos

Atualizado em: 10 de maio de 2020

Agenda

- Definição básica
- Histórico das GPUs
- Principais Fabricantes
- Arquitetura Básica
- Aplicações
- Programação de GPUs
- Principais problemas de utilização
- Comparação GPUs x FPGAs

Começando a nossa conversa...

- Uma GPU, como o próprio nome diz´, é uma Unidade de Processamento Gráfico

- Um hardware especializado em realizar operações que ocorrem com frequência em processamento de imagens e vídeos

- Atualmente também são utilizadas para processamento de dados, o que originou o GPGPU – *General Purpose* GPU

Começando a nossa conversa...

- Inicialmente a principal preocupação do hardware era o processamento de dados

- Porém com a evolução do hardware veio também o aumento das possibilidades de utilização do computador para as mais diversas aplicações

- Portanto a interface gráfica, cujo foco é totalmente no usuário e nas aplicações que podem ser desenvolvidas, criou a necessidade de desenvolvimento do mercado de placas gráficas

Histórico

- O início da trajetória das placas de vídeo é a época dos *arcades* em 1970

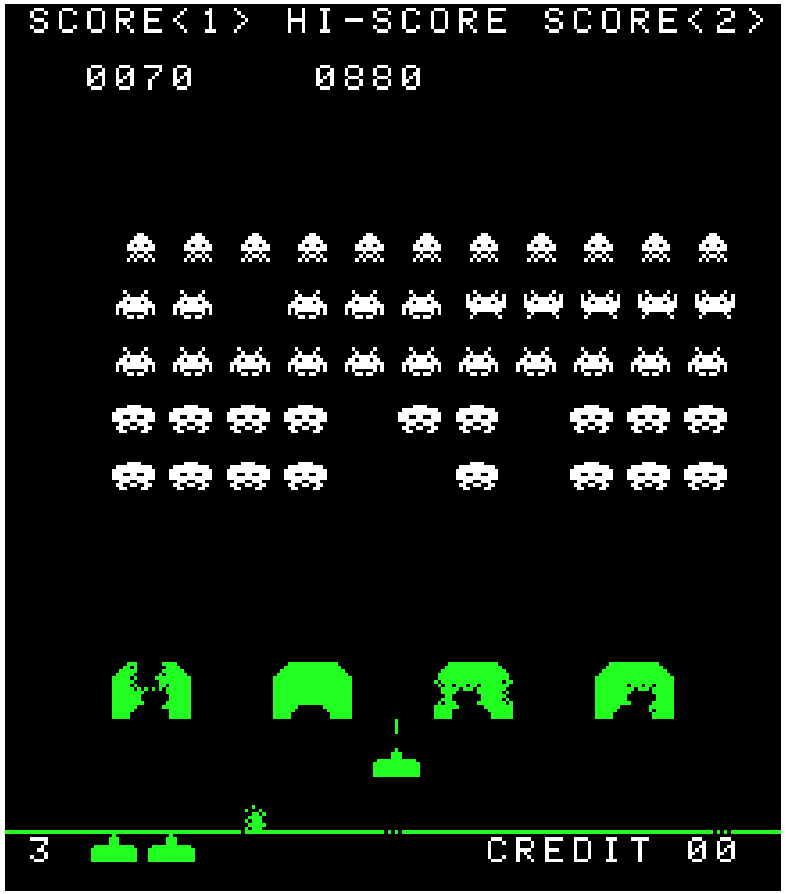
- Memória de vídeo na época era basicamente a memória RAM do computador cujo preço era proibitivo -> 4K = $125!!

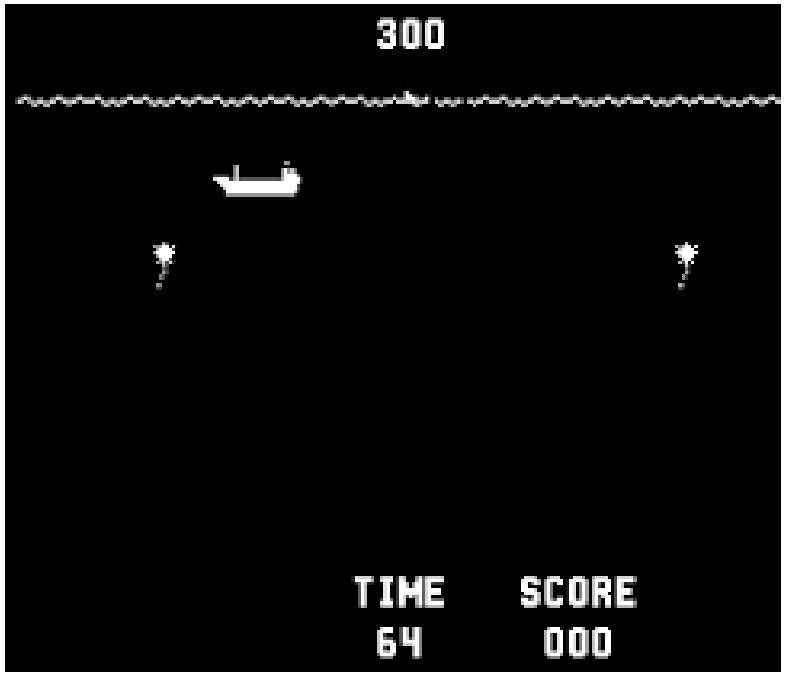
- Então para que a parte de vídeo não ficasse 5x o preço do restante do hardware, as primeiras interfaces eram desenvolvidas de outras maneiras

Histórico

- Um destas outras maneiras era uma tela monocromática que possuía apenas uma linha horizontal colorida.
  - Portanto para uma interface mais “bonita” era necessário reescrever a memória para cada uma das linhas da tv em sequencia criando o efeito de colorido

- Trabalhando as possibilidades com o hardware da época foi possível criar diversos jogos que fizeram muito sucesso

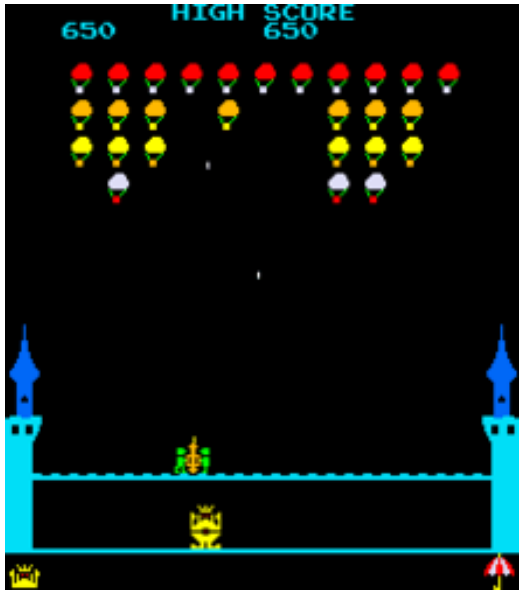




Fonte: imagens obtidas de emuladores dos jogos

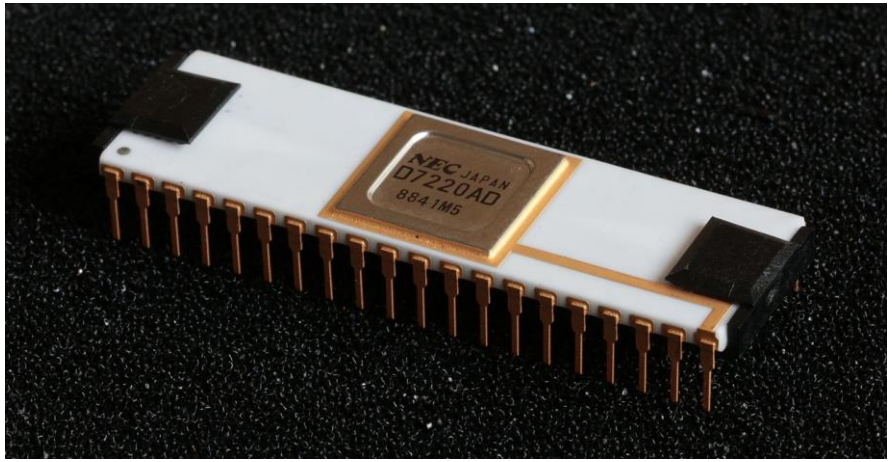
Histórico

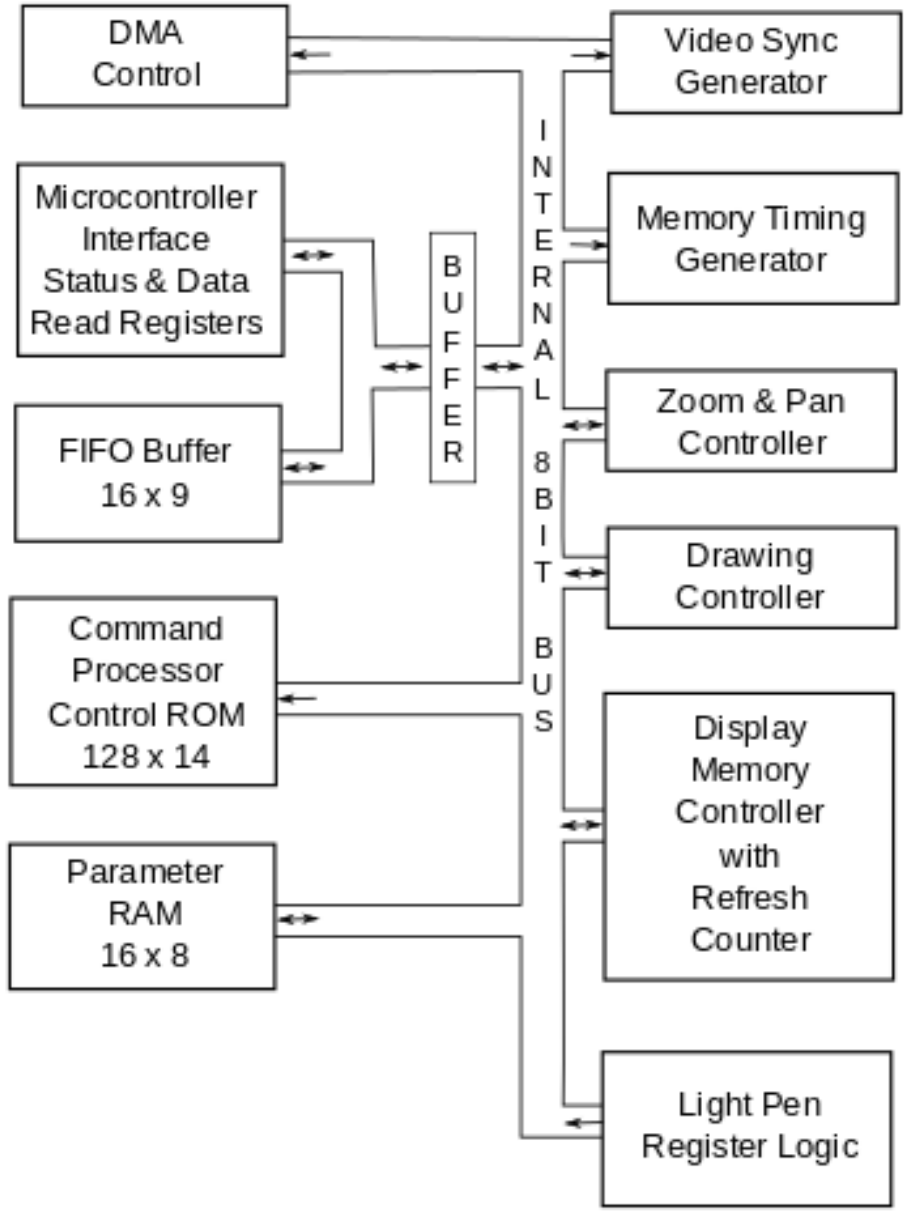
- Já em 1979 o Namco Galaxian tinha um hardware bem interessante
  - Zilog Z80
  - Sistema 8-bit
  - Hardware gráfico especializado com suporte a RGB, planos de fundo e spirtes multicoloridos
- Os backgrounds coloridos trouxeram uma diferença considerável em relação aos backgrounds monocromáticos anteriores



Histórico

- O primeiro processador gráfico desenvolvido foi o NEC uPD7220
  - Este hardware era capaz de desenhar círculos, arcos, linhas e caracteres

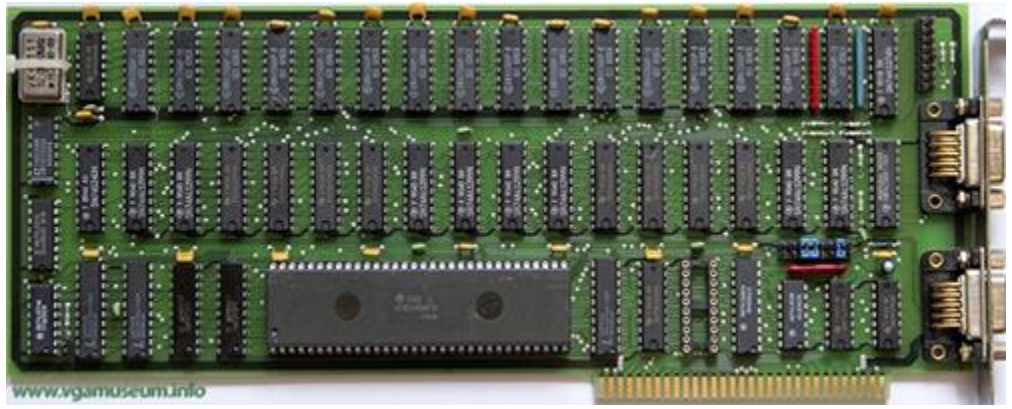


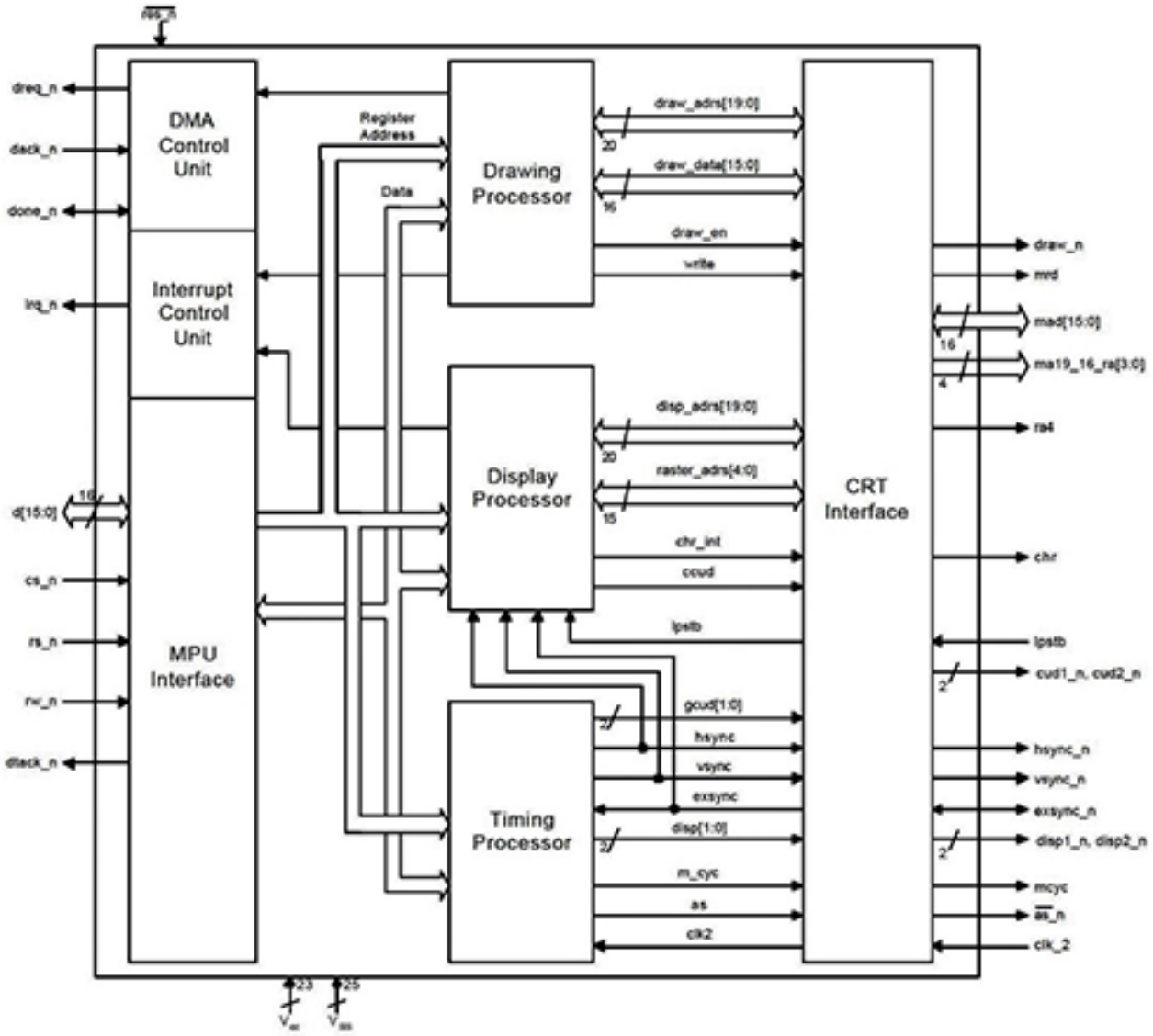


Fonte: https://en.wikipedia.org/wiki/NEC_%C2%B5PD7220

Histórico

- Em 1984 a Hitachi lançou o ARTC HD63484
  - Chip CMOS
  - 4K monocromático





Fonte: https://inn.tech.blog/tag/hd63484/

Histórico

- Em 1986 foi lançado pela Texas Instuments o TMS34010
  - Foi o primeiro chip gráfico programável do mercado
  - Processador de 32 bits com instruções gráficas

Fonte: https://en.wikipedia.org/wiki/TMS34010

Histórico

- As primeiras placas 3D aparecem em 1988 em arcades da Namco e Taito

Fonte: http://www.system16.com/hardware.php?id=536

Histórico

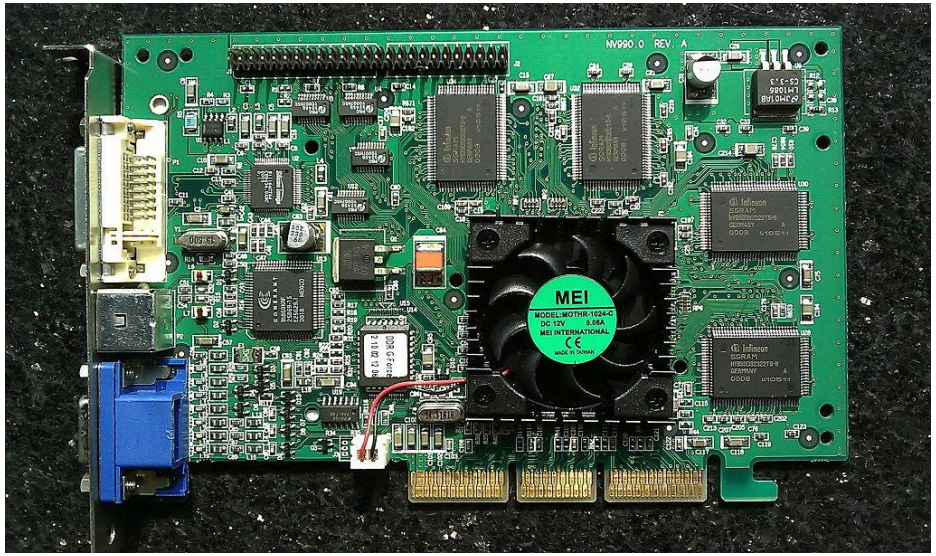
- No início e no meio dos anos 90 diversos *arcade*s já possuíam gráficos 3D em tempo real o que gerou uma demanda por hardware para esta finalidade
- Em 1993 a Namco possuía um simulador de voo que trabalhava transformações geométricas e iluminação. Nestes casos utilizava-se DSPs para acelerar o cálculos

Fonte:https://i.ytimg.com/vi/-4_qsIX9N5M/hqdefault.jpg

Histórico

- O termo GPU foi criado pela SONY 1994 com o lançamento do Playstation
- Em computadores pessoais as placas iniciais eram apenas aceleradoras de funções 3D
  - Power VR
  - 3dfx Voodoo
- No início dos anos 90 também foi criado o OpenGL, porém suas versões inciais sofriam de diversos problemas que iremos tratar mais a frente

Histórico



- O marco para o mercado consumidor foi o lançamento da G-Force FX 256
  - Outubro de 1999
  - Pixel Pipeline
  - Hardware transforming and lighting engine
  - Hardware para MPEG-2 vídeo
  - Primeiro hardware compatível com DirectX 3D 7
  - 220nm CMOS

Fonte: https://en.wikipedia.org/wiki/GeForce_256#/media/File:VisionTek_GeForce_256.jpg

Histórico

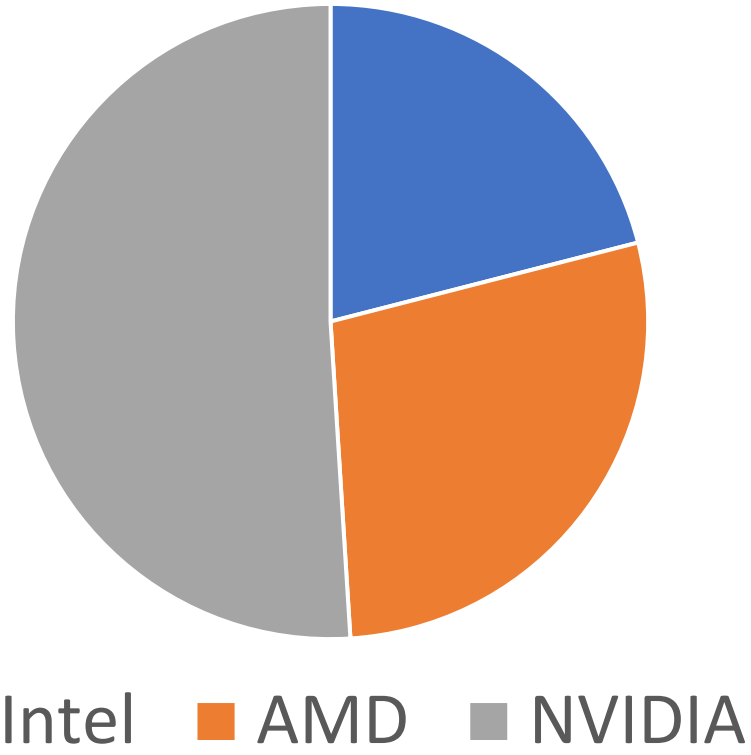
- Com o desenvolvimento da série Ge-Force 8 as placas gráficas começaram a ser utilizadas como GPGPU
  - Microarquitetura TESLA
  - Primeira arquitetura a implementar o Unified Shader Model
    - Desta maneira a placa pode se “adaptar” ao processamento necessário
  - Mudança de unidades de processamento funcionais para um sistema homogêneo de processadores de ponto flutuante
  - Feita para o processamento de streams

Histórico

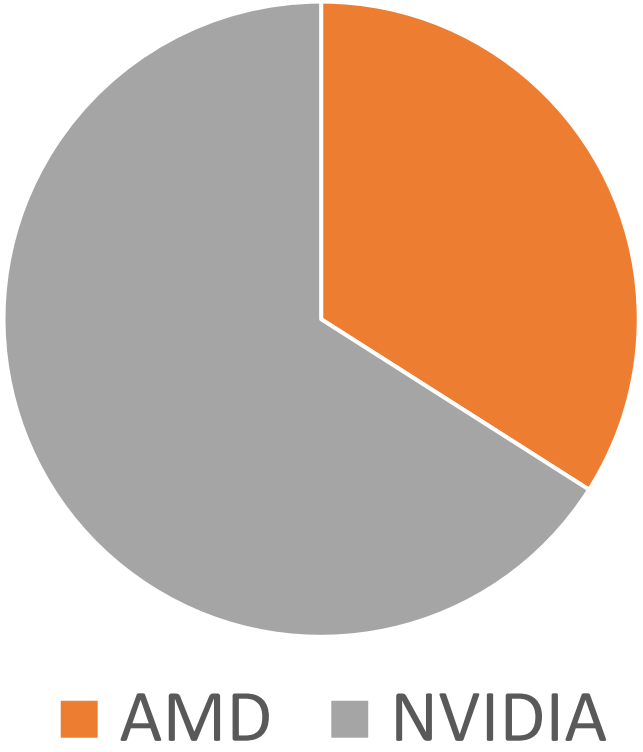
- Atualmente a diversidade de aplicações têm direcionado o desenvolvimento de hardware
  - NVIDIA TEGRA para veículos autônomos
  - AMD RADEON 6000M series para portáteis
  - A partir da microarquitetura Kepler da NVIDIA placas possuem autoajuste de clock para diminuir o consumo de energia

Fabricantes

2012 (com integrados)



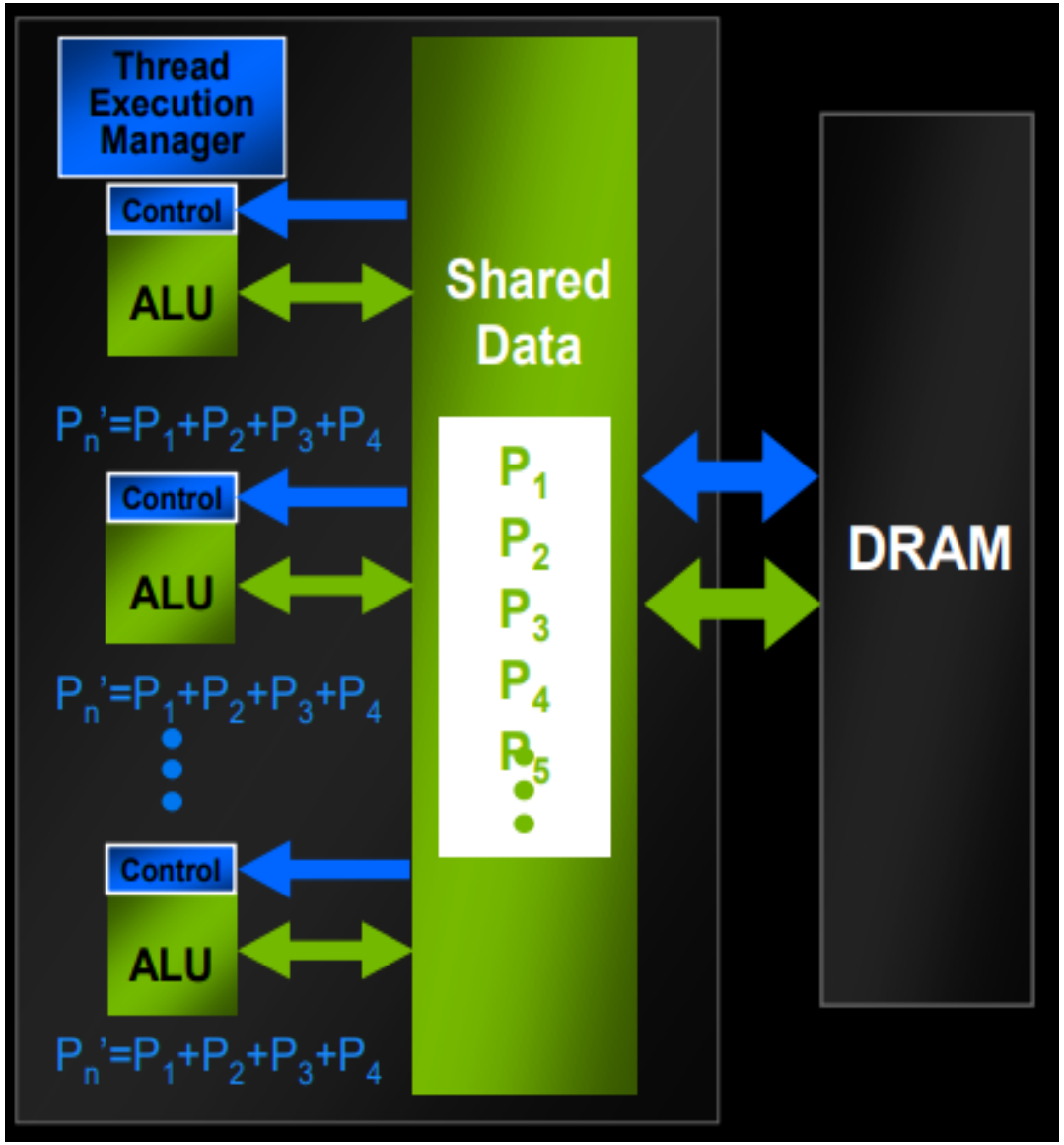
2018 (sem integrados)



Aquitetura

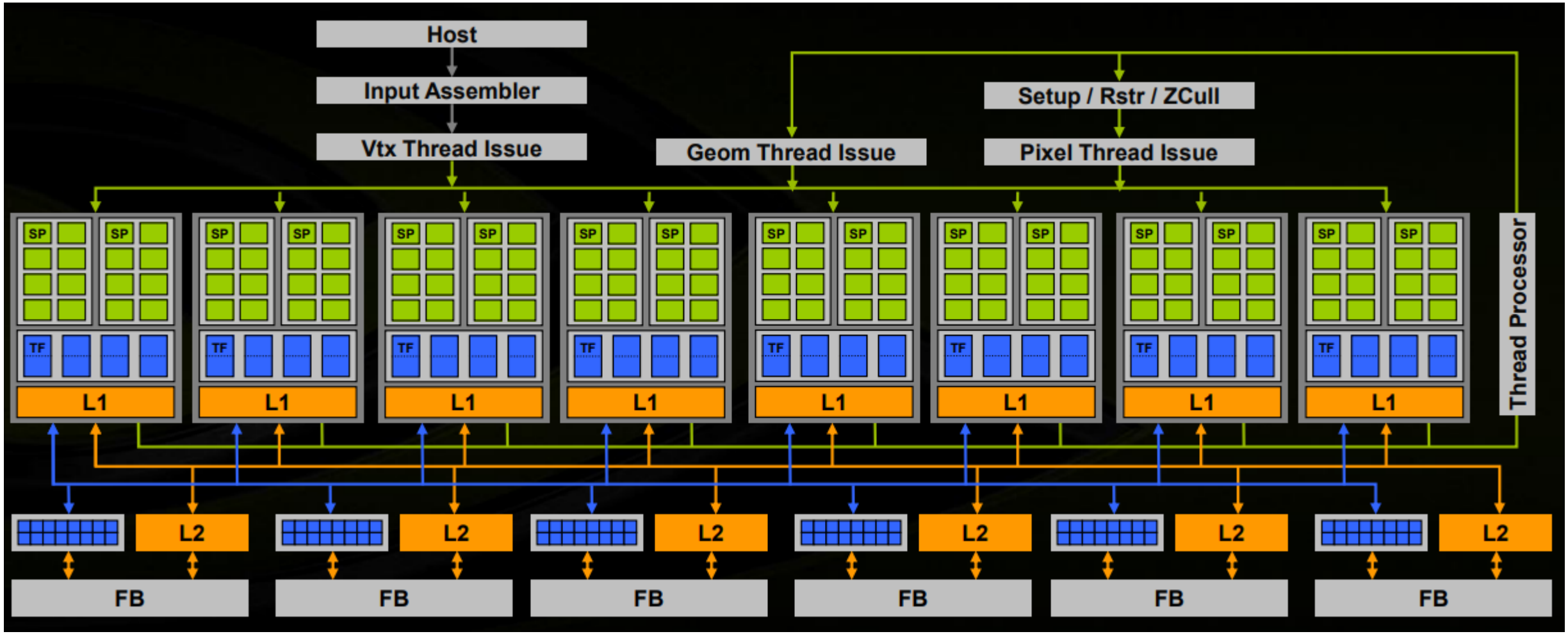
- Arquitetura básica de uma GPU atualmente
  - Unidade de execução de *threads*
  - Várias ULAs de ponto flutuante
  - Uma memória significativamente grande que se comunica com a RAM do computador

- Arquitetura mais próxima de SIMD e Vetorial



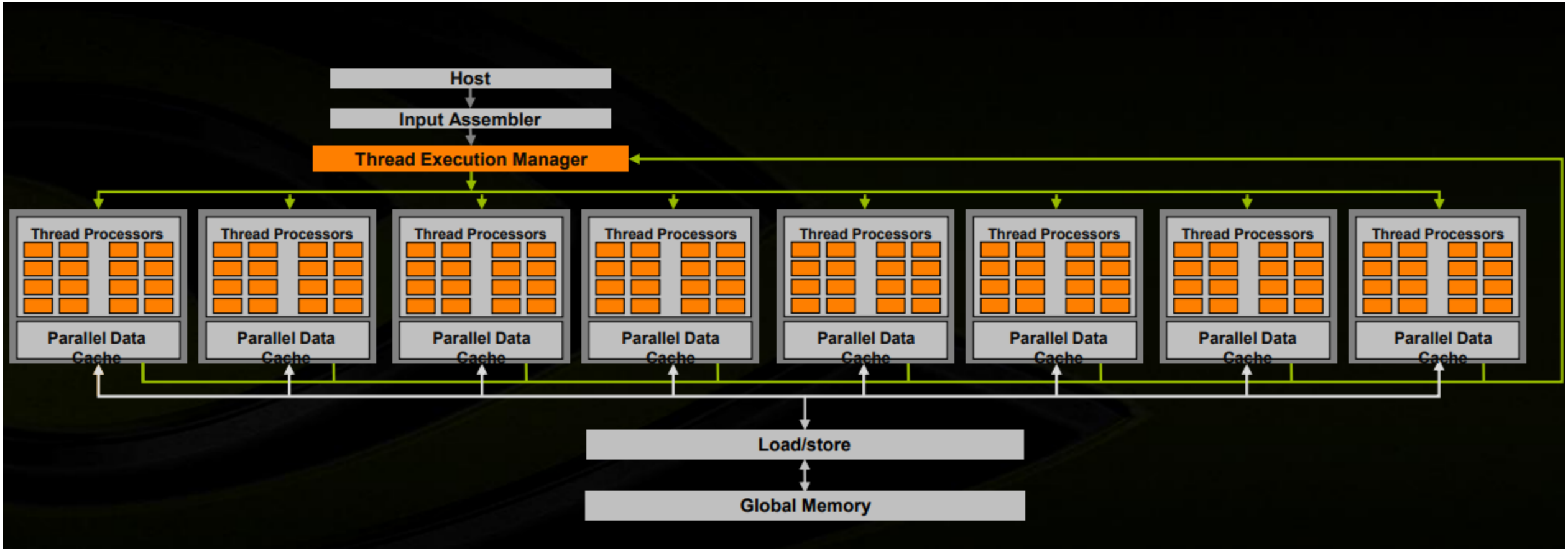
Fonte: https://www.nvidia.com/content/GTC-2010/pdfs/2275_GTC2010.pdf

Aquitetura



Fonte: https://www.nvidia.com/content/GTC-2010/pdfs/2275_GTC2010.pdf

Aquitetura



Fonte: https://www.nvidia.com/content/GTC-2010/pdfs/2275_GTC2010.pdf

Aplicações

- Aplicações de GPUs
  - Bioinformática – tarefas com muitos cálculos
  - Simulações financeiras – método de monte carlo
  - Dinâmica de Fluidos computacional
  - Data Science, Analytics
  - Inteligência computacional
  - Roteamento de circuitos eletrônicos
  - Processamento de imagens e visão computacional
  - Previsão do tempo
  - Cálculos estruturais

Programação

- Existem basicamente duas formas de se programar uma GPU
- Utilizando a solução proprietária da NVIDIA o CUDA
- Utilizando a solução opensouce atual que é o OpenCL
  - Que atualmente está “encapsulado” no SYCL

CUDA

- É uma sigla para Compute Unified Device Architecture
- Plataforma paralela para GPGPU
- Utiliza as linguagens C, C++ e Fortran, só é executada se você possuir uma placa de vídeo da NVIDIA
- Recomenda-se a utilização caso exista uma pequena tarefa com alto grau de paralelismo

CUDA

- No CUDA a execução pode ocorrer em dois modos
  - Host – que seria o seu processador, a CPU
  - Device – que seria então a placa de vídeo

- Um programa em cuda possui a extensão .cu e o compilador é o nvcc

- As funções a serem executadas de forma paralela precisam ter uma marcação no código

CUDA

- A chamada *kernel* em um código cuda é a parte a ser paralelizada na execução

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  __global__ void kernel (void){
5      printf("Hello world from GPU!!");
6  }
7
8  int main (){
9
10     kernel<<<1,1>>>();
11
12     printf("executado na gpu");
13
14     return 0;
15
16 }
```

CUDA

- A marcação __global__ alerta o compilador para a execução no device
- O parâmetro <<<1,1>>> é um pouco mais complexo...

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  __global__ void kernel (void){
5      printf("Hello world from GPU!!");
6  }
7
8  int main (){
9
10     kernel<<<1,1>>>();
11
12     printf("executado na gpu");
13
14     return 0;
15
16 }
```

CUDA

- A configuração da execução do kernel usa um parâmetro do tipo <<< Dg, Db, Ns, S >>>
  - Dg representa o tamanho do grid a ser alocado para a tarefa
  - Db representa o número de *threads* por bloco
  - Ns é um argumento opcional que irá determinar a alocação dinâmica de bytes em cada bloco
  - S, também um parâmetro opcional, que é do tipo cudaStream_t e indica os streams associados ao processamento

- Portanto <<<1,1>>> significa 1 bloco com 1 *thread*

CUDA

- Também é possível:
  - Passar parâmetros para a função que será executada no device
  - Alocar memória no device para que seja utilizada
  - **Gravar o resultado da GPU em variáveis locais**
- Vejamos o próximo exemplo

CUDA

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  __global__ void multiply (int a, int b, int *c){
5      *c = a * b;
6  }
7
8  int main (){
9
10 int c;
11 int *dev_c;
12
13     cudaMalloc(&dev_c, sizeof (int));
14     multiply<<<1,1>>>(2,7,dev_c);
15     cudaMemcpy( &c, dev_c, sizeof(int), cudaMemcpyDeviceToHost);
16     printf("Resultado = %d \n", c);
17     cudaFree(dev_c);
18     return 0;
19
20 }
```

CUDA

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  __global__ void multiply (int a, int b, int *c){
5      *c = a * b;
6  }
7
8  int main (){
9
10 int c;
11 int *dev_c;
12
13     cudaMalloc(&dev_c, sizeof (int));
14     multiply<<<1,1>>>(2,7,dev_c);
15     cudaMemcpy( &c, dev_c, sizeof(int), cudaMemcpyDeviceToHost);
16     printf("Resultado = %d \n", c);
17     cudaFree(dev_c);
18     return 0;
19
20 }
```

CUDA

- Para obter bons resultados com a execução de um vetor é interessante utilizar variáveis incorporadas
    - Cada bloco possui seu blockIdx.x, ou seja, uma variável incorporada do CUDA que contém o índice do bloco correspondente. Utilizando este índice de forma inteligente é possível resolver diversos problemas
- Vejamos o exemplo a seguir

CUDA

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  __global__ void multiply (int *vet){
5      int pos = blockIdx.x;
6      if(pos < N){
7          vet[pos] = pos*pos;
8      }
9  }
10 int main (){
11
12 int vet[N];
13 int *dev_vet
14     cudaMalloc(&dev_vet, N*sizeof (int));
15     cudaMemcpy(dev_vet, vet, N*sizeof(int), cudaMemcpyHostToDevice);
16     multiply<<<N,1>>>(dev_vet);
17     cudaMemcpy(vet, dev_vet, N*sizeof(int), cudaMemcpyDeviceToHost);
18     //coloque aqui seu for para imprimir o vetor vet =)
19     cudaFree(dev_vet);
20     return 0;
21
22 }
```

CUDA

- Como colocado anteriormente o CUDA é uma solução proprietária e depende de um hardware da NVIDIA para ser executado

- Então, desenvolver sua aplicação nesta linguagem pode prejudicar a portabilidade de algumas maneiras.

- A alternativa neste caso envolve a utilização de uma solução opensource que é capaz de gerenciar diversos dispositivos, porém não tão bem quanto CUDA gerencia as placas da NVIDIA.

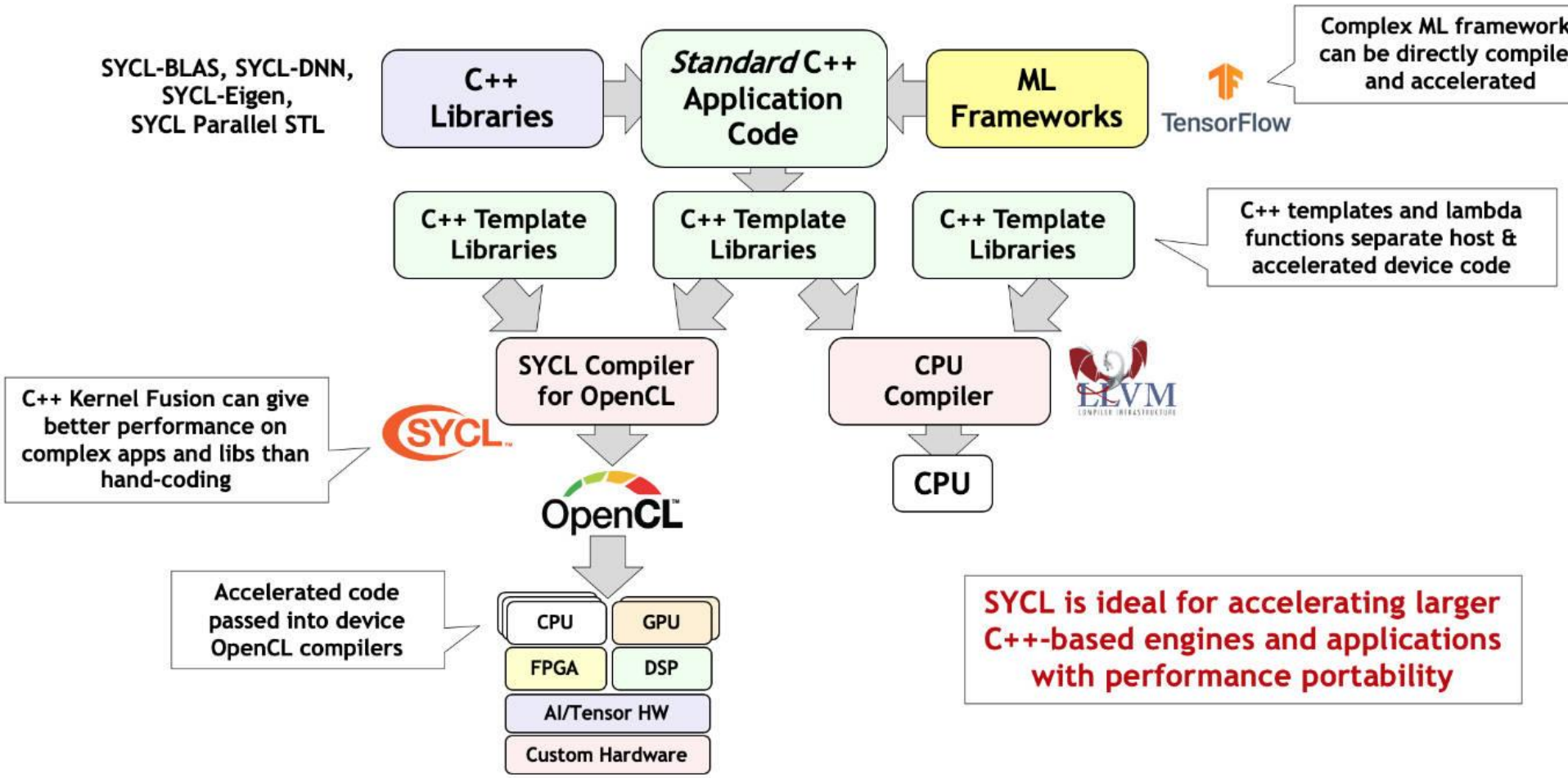
Solução...

- Neste caso é possível utilizar a linguagem OpenCl

- Linguagem *opensource* desenvolvida com objetivo de ser um *framework* que permite a programação para execução em plataformas heterogêneas
  - Dentre ela as GPUs

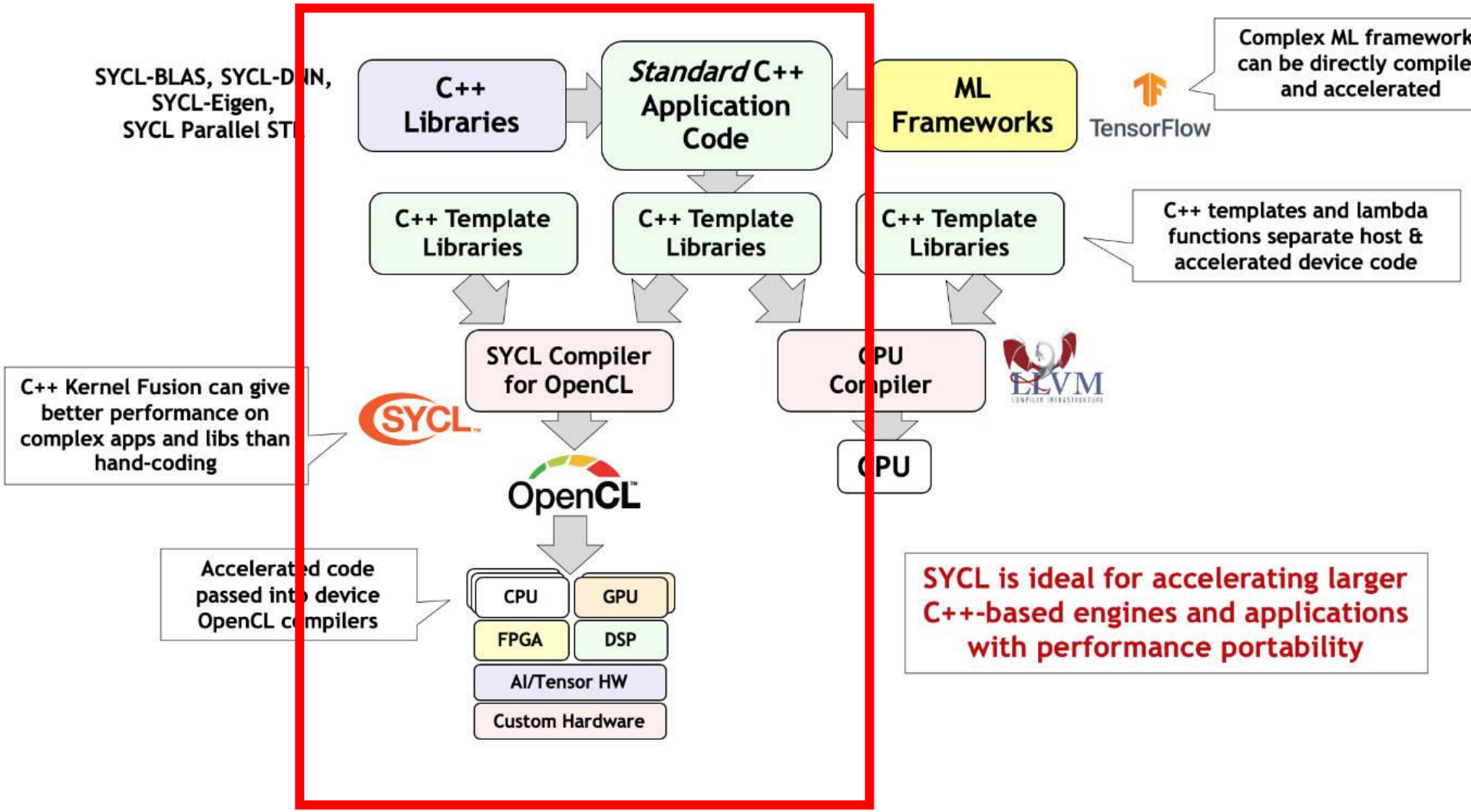
- Atualmente é mantida pelo Khronos group que criou SYCL para melhorar o desempenho das aplicações

Solução...


SYCL-BLAS, SYCL-DNN,
SYCL-Eigen,
SYCL Parallel STL
C++ Libraries
Standard C++ Application Code
ML Frameworks
TensorFlow
Complex ML frameworks can be directly compiled and accelerated
C++ Template Libraries
C++ Template Libraries
C++ Template Libraries
C++ templates and lambda functions separate host & accelerated device code
SYCL Compiler for OpenCL
CPU Compiler
LLVM
C++ Kernel Fusion can give better performance on complex apps and libs than hand-coding
SYCL
OpenCL
CPU
Accelerated code passed into device OpenCL compilers
CPU
GPU
FPGA
DSP
AI/Tensor HW
Custom Hardware
SYCL is ideal for accelerating larger C++-based engines and applications with performance portability


OpenCL

Solução...


SYCL-BLAS, SYCL-DNN,
SYCL-Eigen,
SYCL Parallel STL
C++ Libraries
Standard C++ Application Code
ML Frameworks
TensorFlow
Complex ML frameworks can be directly compiled and accelerated
C++ Template Libraries
C++ Template Libraries
C++ Template Libraries
C++ templates and lambda functions separate host & accelerated device code
SYCL Compiler for OpenCL
CPU Compiler
LLVM
C++ Kernel Fusion can give better performance on complex apps and libs than hand-coding
SYCL
OpenCL
CPU
Accelerated code passed into device OpenCL compilers
CPU
GPU
FPGA
DSP
AI/Tensor HW
Custom Hardware
SYCL is ideal for accelerating larger C++-based engines and applications with performance portability


OpenCL

Principais Problemas

- Apesar de utilizar linguagem C/C++/Fortran, é necessário aprender como programar
- O desempenho final não é totalmente previsível pelos aspectos críticos como o tempo de transferência de dados
- Caso a NVIDIA descontinue a tecnologia CUDA os códigos terão que ser reescritos
- Consumo de energia proibitivo para sistemas embarcados
- Tempo de transferência de dados proibitivo para sistemas de tempo-real

GPU x FPGA

- O estudo de dispositivos para programação paralela leva a uma comparação de certa forma inevitável

- Porém comparar GPUs e FPGAs é errado, pois são hardwares diferentes com escopos diferentes

- GPU é um processador voltado à instruções, FPGA é um hardware voltado à programação de bits

GPU x FPGA

- Mesmo considerando apenas o resultado final, de uma forma ou de outra há desvantagens
  - As GPUs consomem energia e possuem um tempo muito grande de transferência de dados
  - Os FPGAs podem atingir um desempenho excelente, porém o tempo de projeto pode ser muito longo e não há garantia alguma de que o hardware final tenha um desempenho mínimo
- O correto nestes casos é comparar a GPU com uma CPU multicore e não com um hardware específico de um FPGA, que deve ser comparado com um ASIC por exemplo.

Referencias Complementares

- Considerar todas as referências apresentadas anteriormente
- Apresentação NVIDIA https://www.nvidia.com/content/GTC-2010/pdfs/2275_GTC2010.pdf
- Para uma lista extensa de referências verificar: https://en.wikipedia.org/wiki/Graphics_processing_unit

Programação Paralela: das *threads* aos FPGAs

GPU's - *Graphic Processing Units*

Prof. Ricardo Menotti
menotti@ufscar.br

Prof. Maurício Acconcia Dias
macccdias@gmail.com

Prof. Helio Crestana Guardia
helio.guardia@ufscar.br

Departamento de Computação
Universidade Federal de São Carlos

Atualizado em: 10 de maio de 2020