Universidade Estadual do Rio Grande do Sul

Aula 10

Organização de Computadores

Processadores Paralelos

Profa. Débora Matos

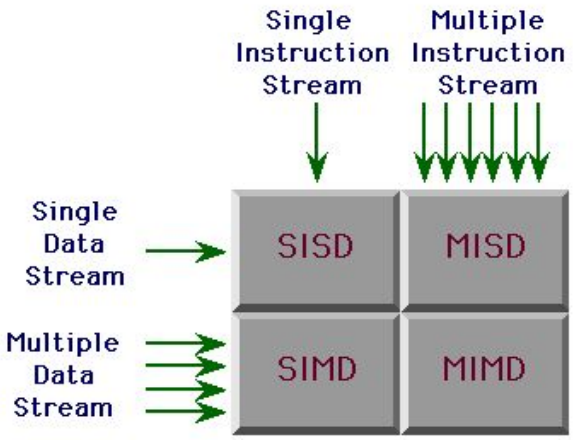
Taxonomia de Flynn

Taxonomia de Flynn

- Dois fatores são considerados:
  - Número de instruções
  - Número de sequência de dados passados para o processador
- Um processador pode receber várias sequências de dados;
- Podem ter 1 ou vários processadores trabalhando nestes dados;

Taxonomia de Flynn

- Quatro possíveis combinações:
  - **SISD** – single instruction, single data
  - **SIMD** – single instruction, multiple data
  - **MISD** – multiple instruction, single data
  - **MIMD** – multiple instruction, multiple data



- **Máquinas MISD:** possuem várias sequências de instruções operando sobre a mesma sequência de dados;
- **Máquinas MIMD:** adotam vários pontos de controle, possuem sequências de instruções e dados independentes;

Taxonomia de Flynn

- Os multiprocessadores e a maioria dos sistemas paralelos atuais são máquinas **MIMD**, no entanto estas máquinas são mais difíceis de projetar;
- Nos processadores SIMD, a mesma instrução é executada para diferentes dados;
- As máquinas MISD são as menos utilizadas. Não justifica o custo.
- A taxonomia de Flynn possui algumas falhas:
  - Assume que o paralelismo é homogêneo (sendo que a heterogeneidade pode estar em diferentes aspectos);

Quais seriam estes?

Taxonomia de Flynn

- Uma arquitetura MIMD pode ser classificada com relação a:
  - como os processadores estão conectados
    - Barramentos
    - Crossbar
    - Redes
  - como eles enxergam a memória;
    - Compartilhada
    - Vários níveis de cache

Taxonomia de Flynn

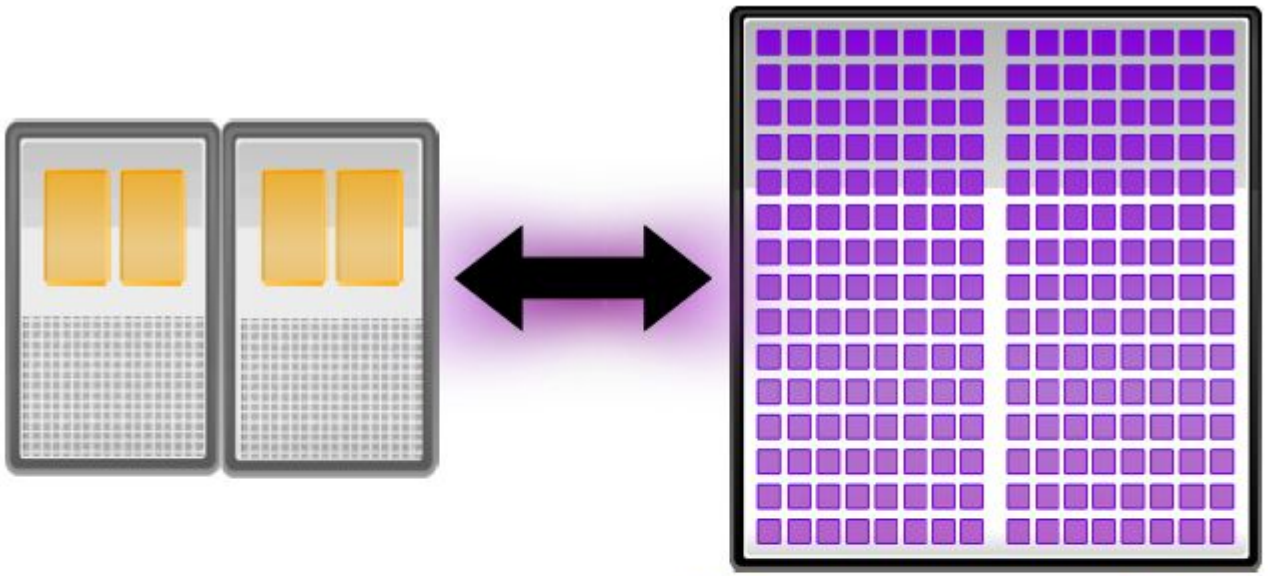
- Arquitetura Paralela MIMD:
  - SMP (ou CMP) – multiprocessadores simétricos
  - MPP (ou MPSoC) – processadores massivamente paralelos

- Computadores MPP são **difíceis de programar**
  - É preciso particionar o programa entre os processadores e garantir que eles se comuniquem corretamente;

Taxonomia de Flynn

- Computadores SMP apresentam sérios gargalos
  - Complica quando vários os processadores tentam acessar a memória compartilhada ao mesmo tempo
- A decisão de usar MPP ou SMP depende da aplicação:
  - Se o programa é facilmente particionável;
  - MPP têm sido uma tendência nos novos projetos de chips;

Multicore x Manycore



Multicore CPU

Fast Serial Processing

Latency-optimized cores

Manycore GPU

Scalable Parallel Processing

Throughput-optimized cores

Taxonomia de Flynn

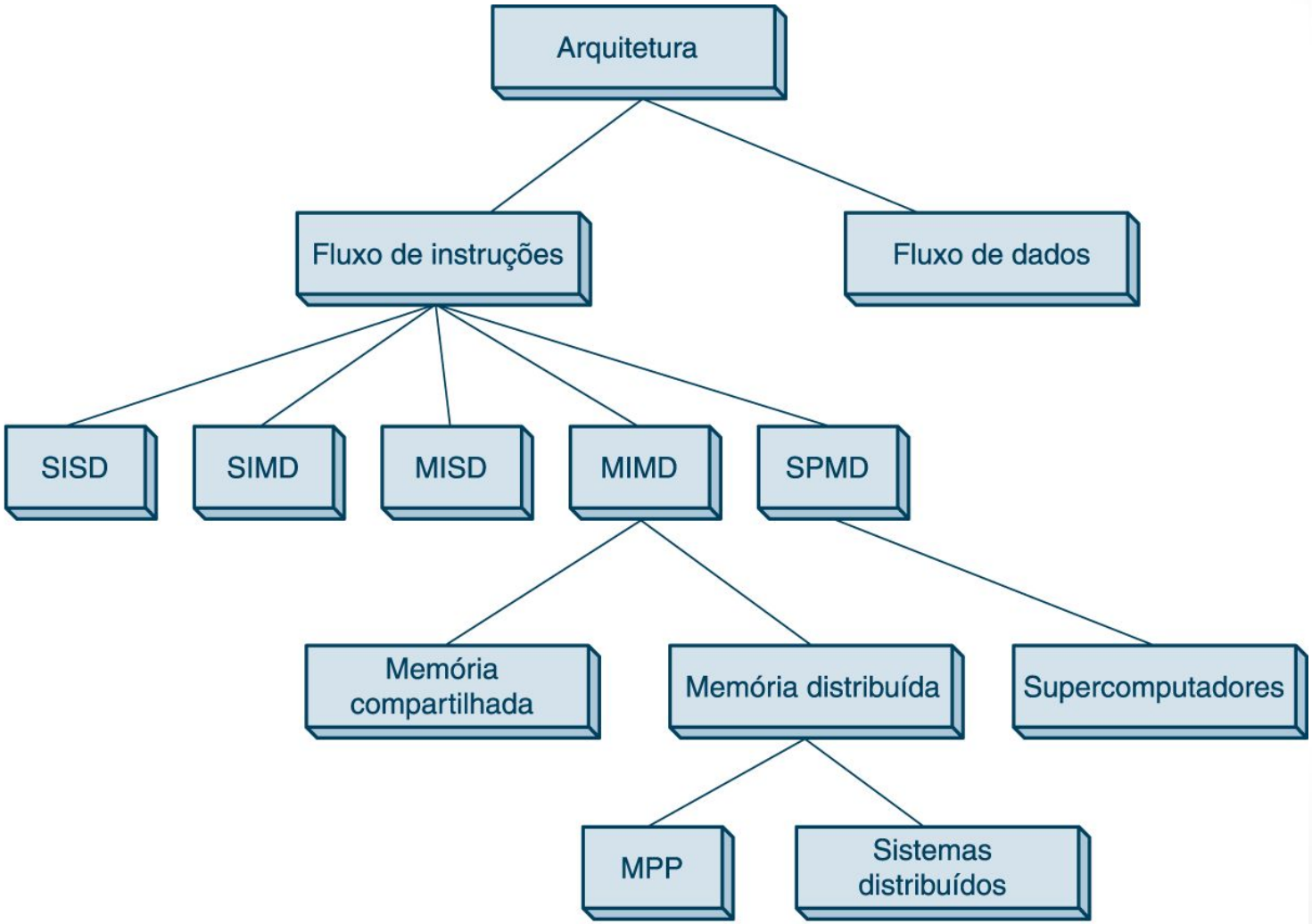
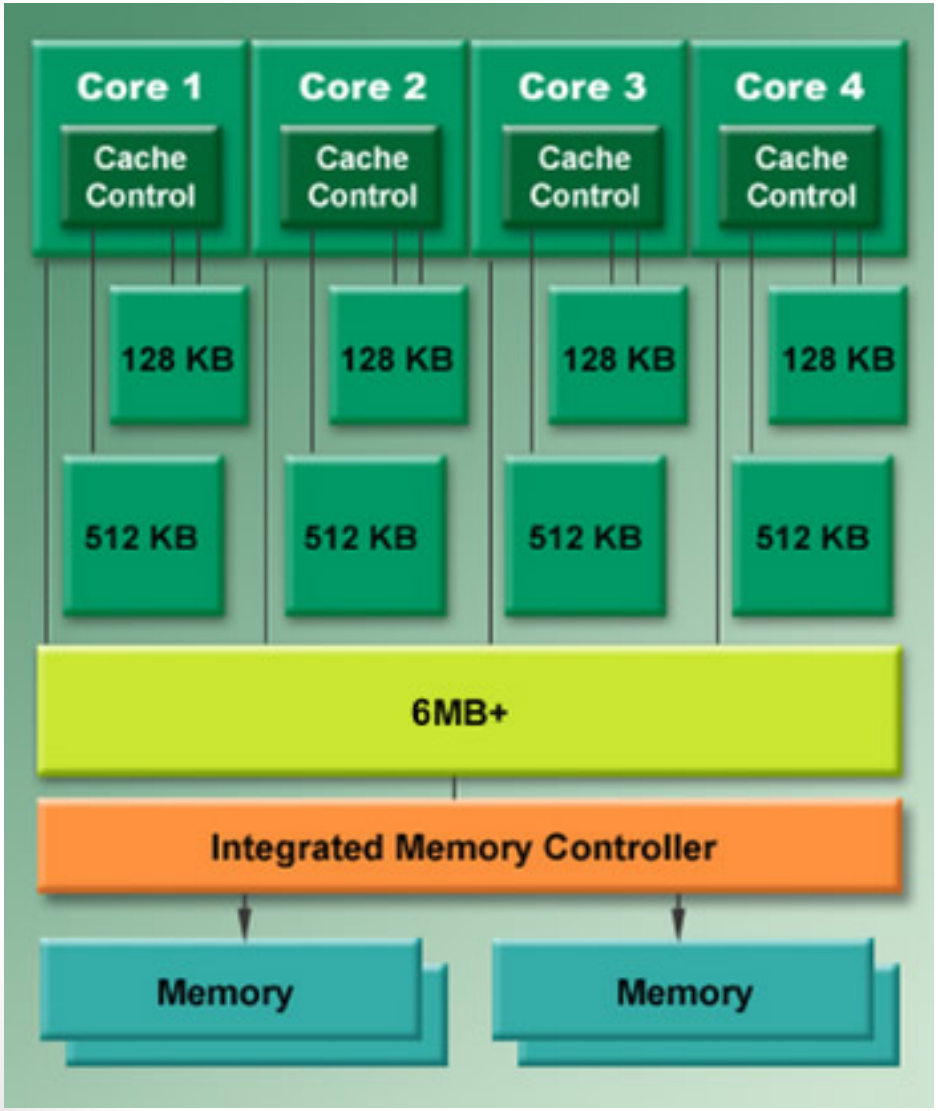
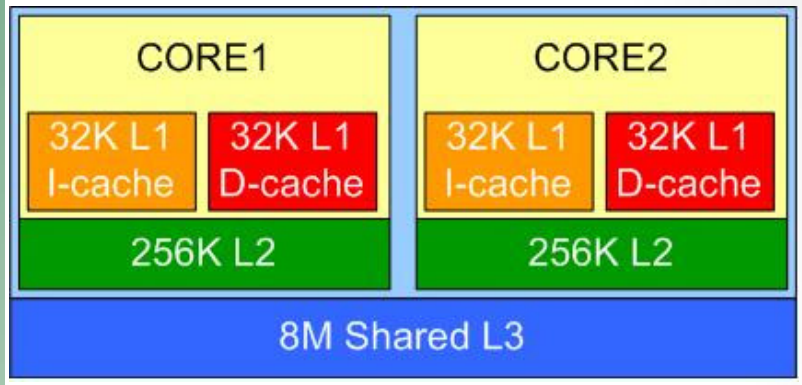


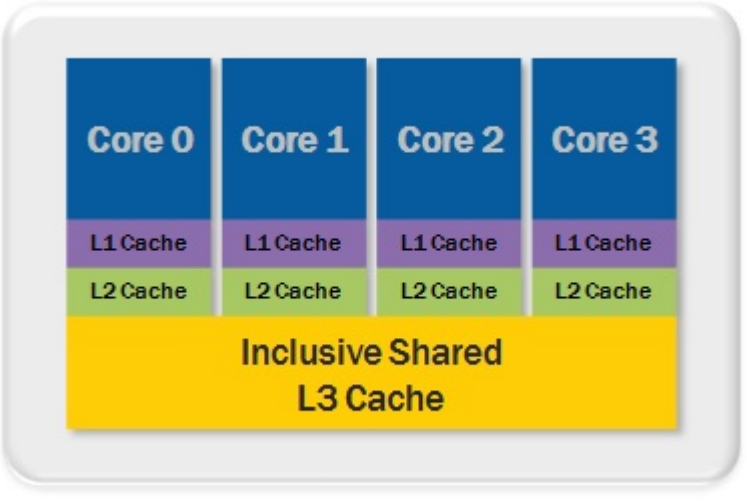
FIGURA 9.2 Uma taxonomia de arquiteturas de computadores.

Multiprocessadores e os níveis de cache





Core i7 - 820QM



CPU x GPU

CPU x GPU

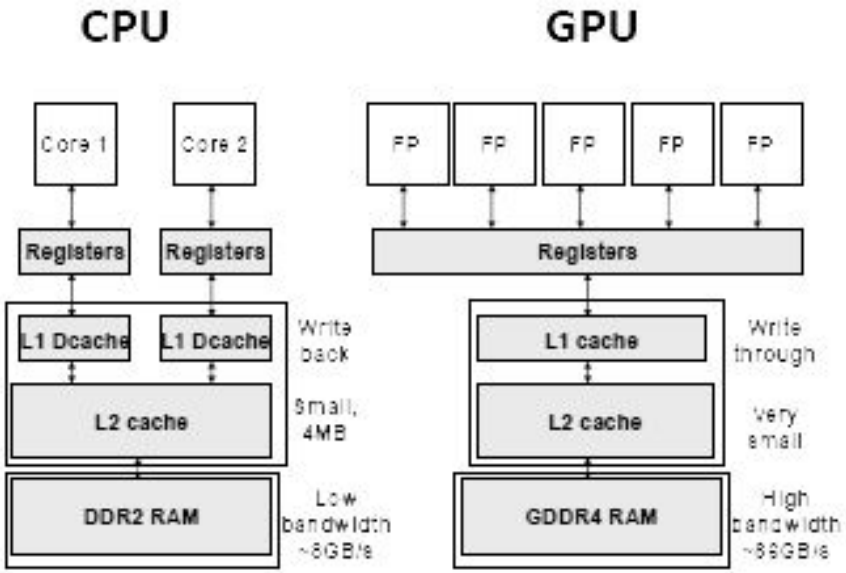
O que é uma GPU?

- GPU (Graphics Processing Unit) é um processador gráfico que realiza cálculos complexos necessários para criar efeitos de iluminação e transformar objetos em aplicações 3D;
- GPUs são projetados para executar cálculos matemáticos intensos;
- A GPU GeForce 256 é capaz de realizar bilhões de cálculos por segundo e processa um mínimo de 10 milhões de polígonos por segundo;
  - Polígonos representam as linhas e ângulos necessários para a produção de conteúdo visualizado em um computador.
- CPUs são capazes de uma vasta gama de velocidades de cálculo, mas em média, cerca de 100 milhões de cálculos por segundo.

CPU x GPU

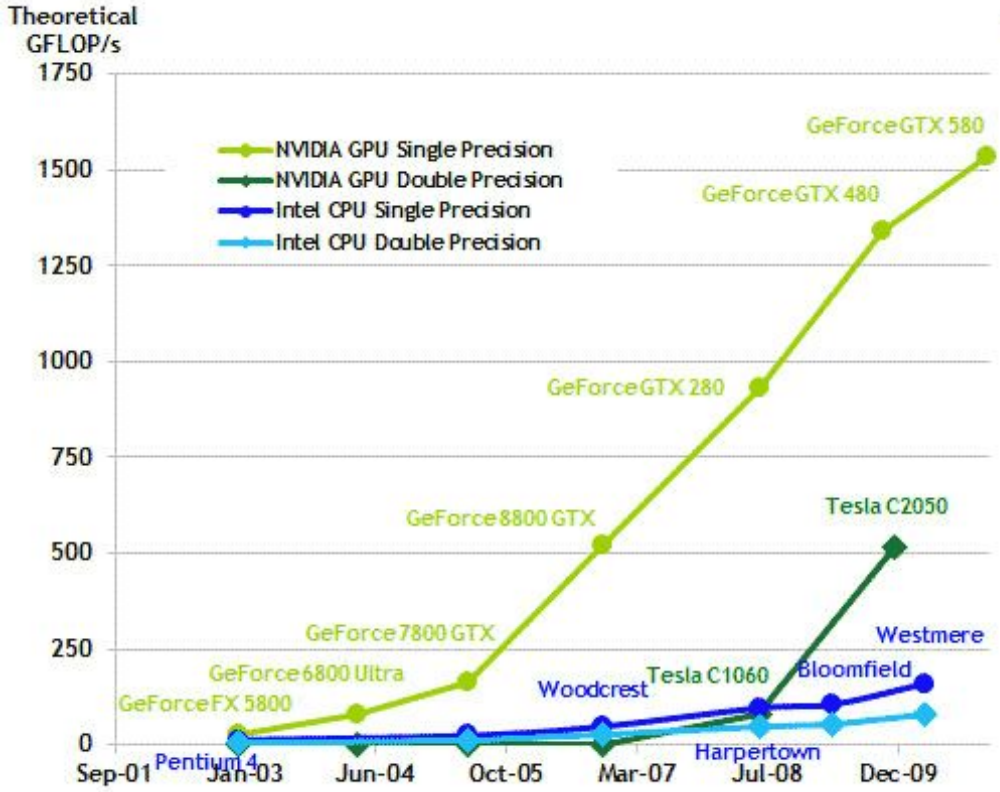
- **CPU**
  - Quantidade de Caches (L1, L2, L3)
  - Previsão de Salto
  - Alta-performance (pelas previsões)
- **GPU**
  - Muitas ALUs
  - Grande quantidade de tarefas paralelas
  - Executa programas em cada fragmento/vértice
  - GPUs são indicadas para paralelismo de dados

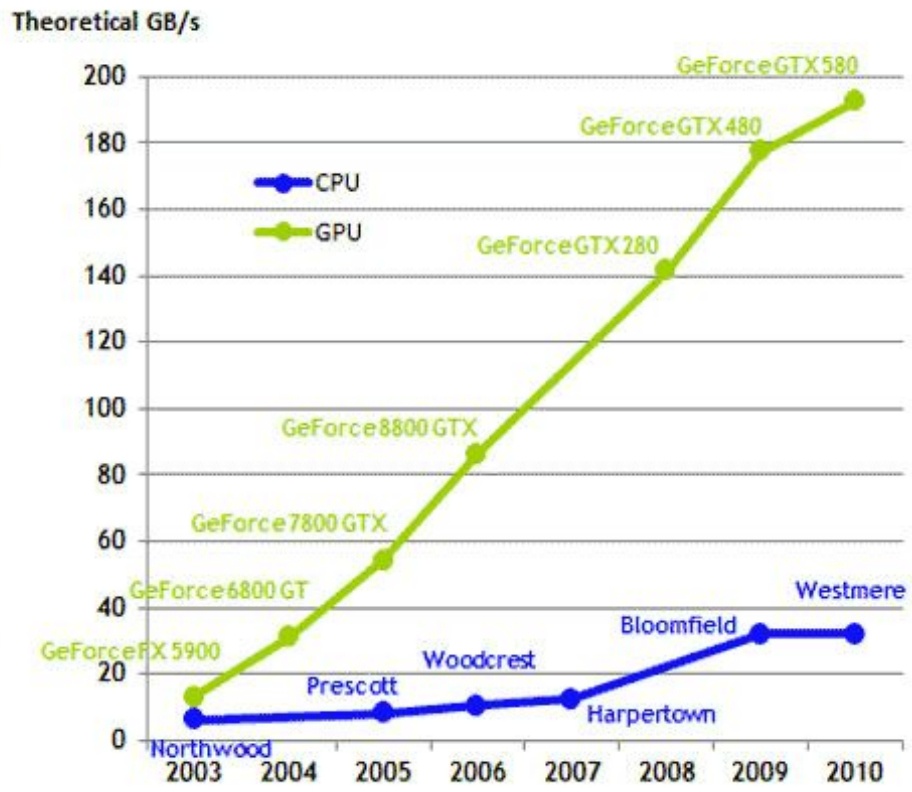
CPU x GPU


CPU
GPU
Core 1
Core 2
FP
FP
FP
FP
FP
Registers
Registers
Registers
L1 Dcache
L1 Dcache
Write back
L1 cache
Write through
L2 cache
Small, 4MB
L2 cache
Very small
DDR2 RAM
Low bandwidth ~6GB/s
GDDR4 RAM
High bandwidth ~88GB/s

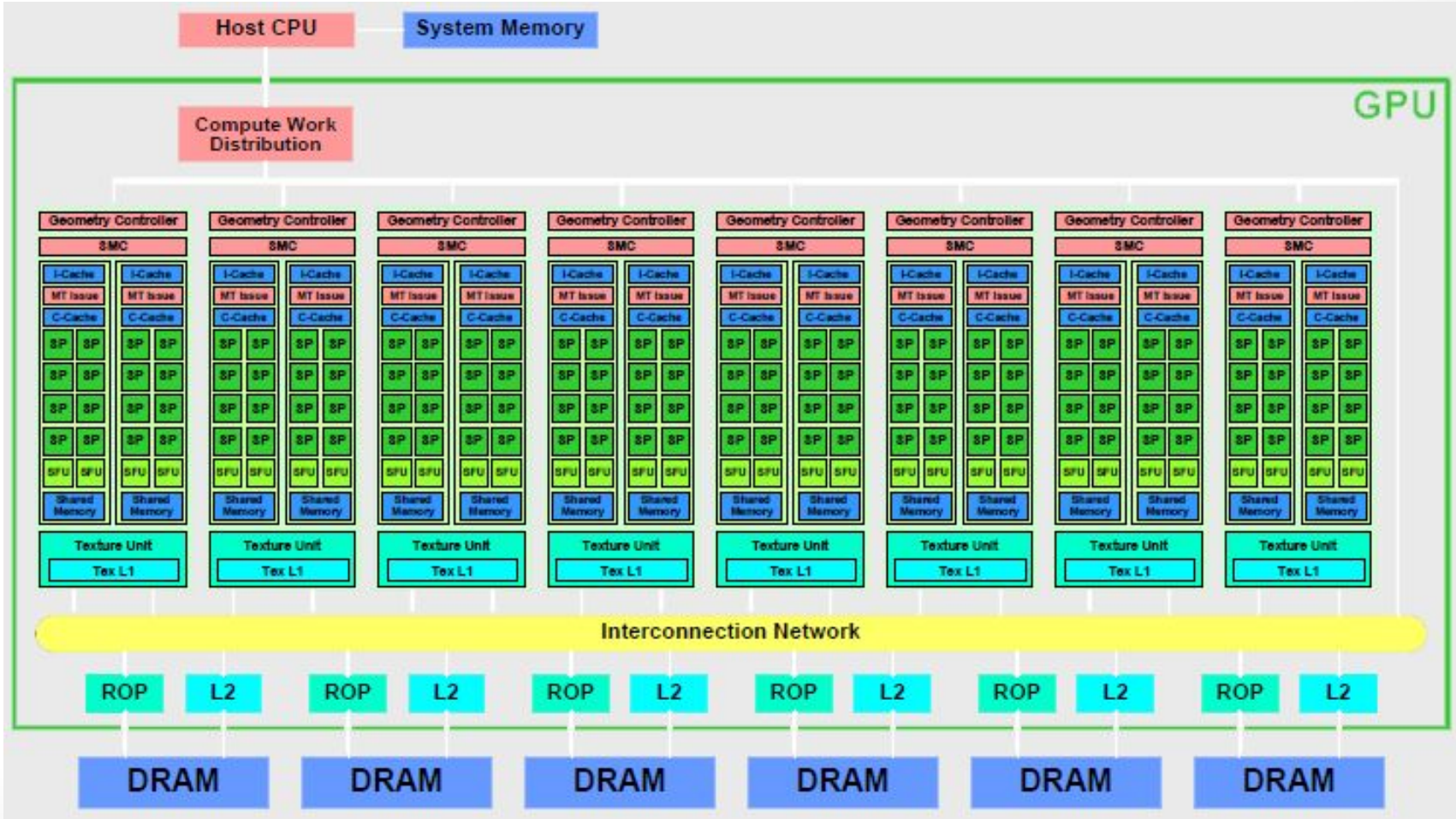

Control
ALU
ALU
ALU
ALU
Cache
DRAM
CPU
DRAM
GPU

CPU x GPU

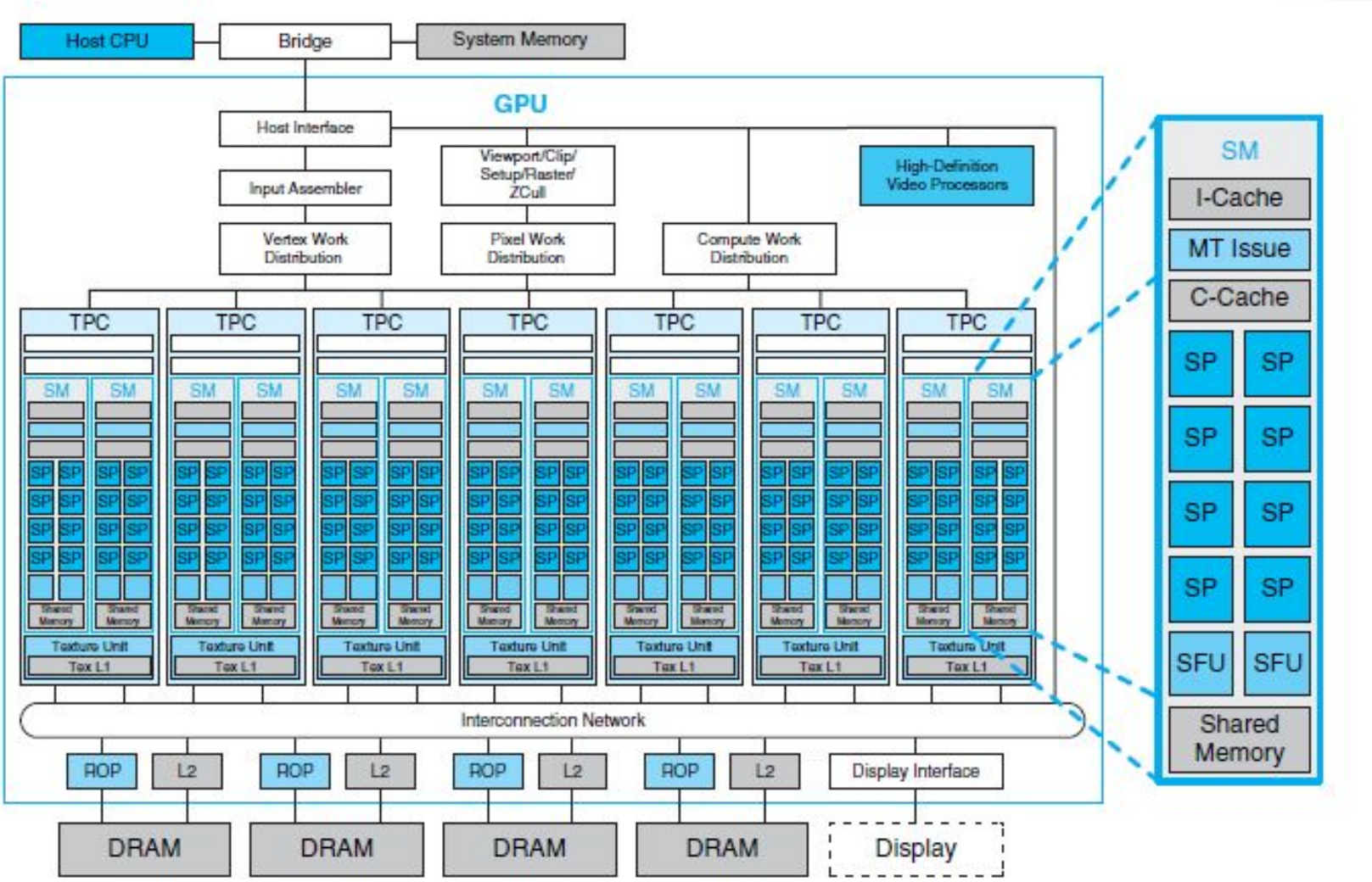

Theoretical GFLOP/s
1750
1500
1250
1000
750
500
250
0
NVIDIA GPU Single Precision
NVIDIA GPU Double Precision
Intel CPU Single Precision
Intel CPU Double Precision
GeForceGTX 580
GeForceGTX 480
GeForceGTX 280
GeForce8800 GTX
GeForce7800 GTX
GeForce6800 Ultra
GeForceFX 5800
Tesla C2050
Tesla C1060
Westmere
Bloomfield
Woodcrest
Harpertown
Pentium 4
Sep-01
Jan-03
Jun-04
Oct-05
Mar-07
Jul-08
Dec-09


Theoretical GB/s
200
180
160
140
120
100
80
60
40
20
0
CPU
GPU
GeForceGTX 580
GeForceGTX 480
GeForceGTX 280
GeForce8800 GTX
GeForce7800 GTX
GeForce6800 GT
GeForceFX 5900
Westmere
Bloomfield
Woodcrest
Prescott
Harpertown
Northwood
2003
2004
2005
2006
2007
2008
2009
2010

GPU


Host CPU
System Memory
GPU
Compute Work Distribution
Geometry Controller
SMC
I-Cache
MT Issue
C-Cache
SP
SFU
Shared Memory
Texture Unit
Tex L1
Interconnection Network
ROP
L2
DRAM

GPU



GeForce 8800

Arquitetura Fermi

- 3 Bilhões de transistores
- 4 GPC (Grap.Proc.Cluster)
- 16 SM
- 32 Elementos/SM
- 512 Cores
- 6 GB DDR5
- 64 bits float
- 32K Registradores/SM
- 64K Shared L1
- 768K Cache L2

CPU x GPU

- GPUs fornecem paralelismo de dados com alta largura de banda de memória;

- As arquiteturas de GPUs estão se tornando cada vez mais programáveis, oferecendo alto *speedup* para uma variedade de aplicações de uso geral, quando comparadas as CPUs.

- As GPUs apresentam melhor desempenho que as CPUs porque foram projetadas para computação intensiva – computação altamente paralelizável;

CPU x GPU

- Os projetos de GPUs tem diferentes objetivos comparada as CPUs:
  - CPUs devem ter bom desempenho para todos os tipos de aplicações, paralelas ou não;
  - GPUs assumem cargas de trabalho altamente paralelizáveis onde todas as threads da GPU executam a mesma sequência do programa;
- **CPU**: minimizam a latência de 1 thread:
  - Controle lógico sofisticado
  - Caches grandes
- **GPU**: maximizam o throughput de todas as threads
  - Número de threads é limitado pelo número de recursos;
  - Compartilham lógica de controle entre as treads.

CPU x GPU

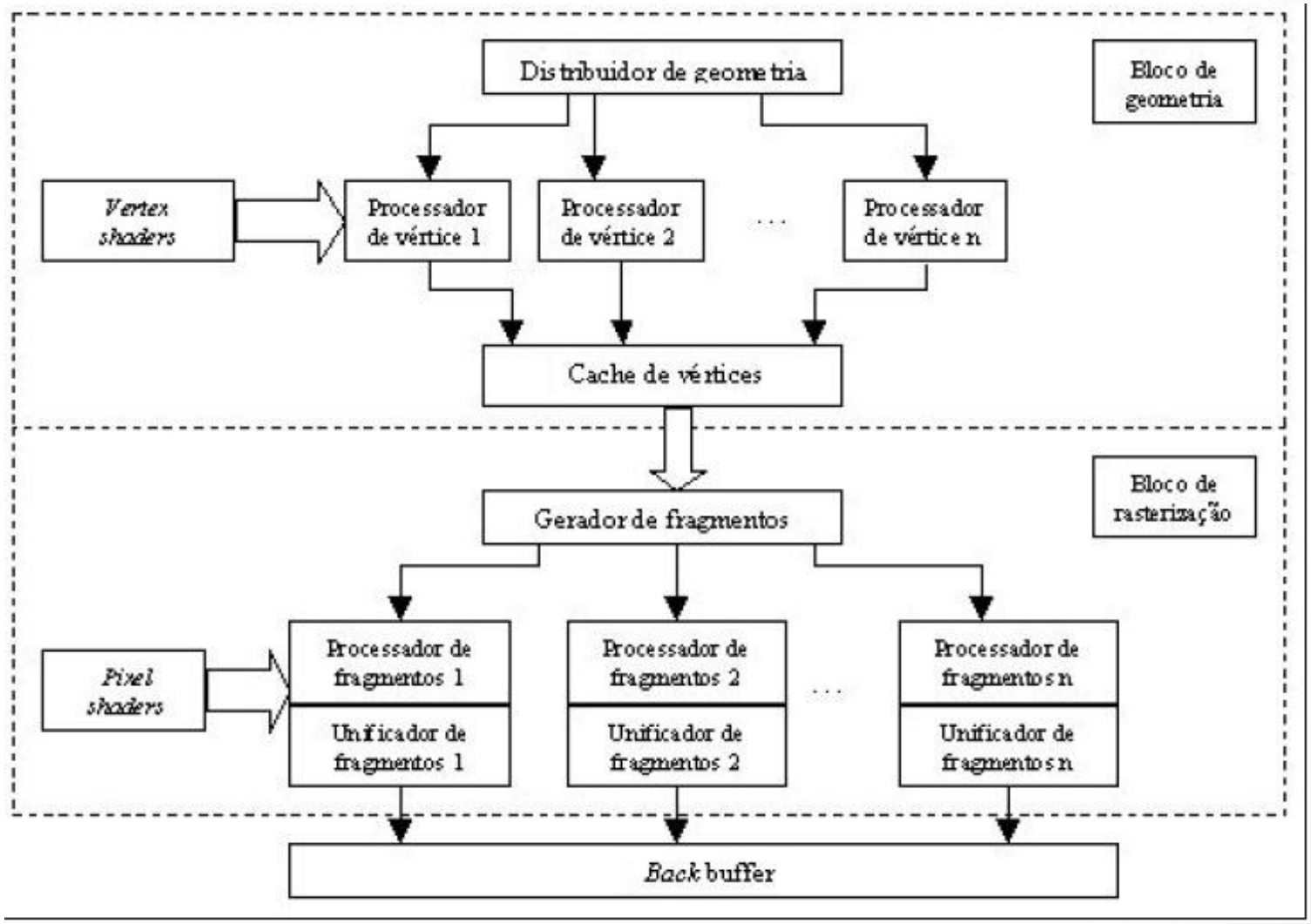
- Enquanto as GPUs tem benefícios, tais como:
  - maior poder de computação,
  - maior largura de banda de memória,
  - baixo consumo de potência comparada a alta habilidade de computação;

- Existem também alguns desafios:
  - maior dificuldade de *debugging*;
  - o desenvolvimento de códigos para GPU pode requerer mais tempo e conhecimento de um outro paradigma de programação ;
  - requerem partição de dados e técnicas de sincronismo;
  - nem todos os algoritmos podem ser executados em GPUs.

GPU – Quebra de paradigma radical

Você mudaria todo o teu paradigma de desenvolvimento e hardware apenas para ganhar 5% de desempenho?

Programmable GPU



GPGPU

- O que é GPGPU?
  - General-Purpose Computing on a Graphics Processing Unit (GPU)
  - Usando Hardware gráfico para computação não-gráfica
- O que é CUDA?
  - Compute Unified Device Architecture
  - Arquitetura de Software para gerenciar programação paralela orientada por dados

GPGPU

CUDA

NVIDIA > Tecnologias > CUDA Parallel Computing Platform

LINKS DE INTERESSE

- CUDA Zone
- CUDA Showcase
- Notícias da CUDA
- NVIDIA Tesla
- NVIDIA Quadro

O QUE É CUDA?

CUDA™ é uma plataforma de computação paralela e um modelo de programação inventados pela NVIDIA. Ela permite aumentos significativos de performance computacional ao aproveitar a potência da unidade de processamento gráfico (GPU).

Com milhões de GPUs habilitadas para CUDA já vendidas até hoje, os desenvolvedores de software, cientistas e pesquisadores estão descobrindo usos amplamente variados para a computação com GPU CUDA. Aqui estão alguns exemplos:

PARA DESENVOLVEDORES

Visite o CUDA Zone para começar a programar em paralelo e ver a tecnologia CUDA em ação

ACESSAR CUDA ZONE

GPGPU

O QUE AS PESSOAS ESTÃO DIZENDO

"As GPUs evoluíram ao ponto de permitirem que muitos aplicativos do mundo real sejam facilmente implementados e executados de forma significativamente mais rápida do que em sistemas com vários núcleos. As futuras arquiteturas de computação serão sistemas híbridos, com GPUs de núcleo paralelo operando em conjunto com CPUs de vários núcleos."

-- Jack Dongarra
Professor da Universidade do Tennessee

"A CUDA é uma plataforma computacional completa para C/C++/Fortran na GPU. Quando começamos a criar a CUDA, tínhamos várias opções para o que poderíamos construir. A informação principal que os clientes nos forneciam era que eles não queriam ter que aprender uma linguagem ou API totalmente nova. Alguns deles estavam contratando desenvolvedores de jogos porque sabiam que as GPUs eram rápidas mas não sabiam como usá-las. Fornecer uma solução que fosse fácil, que você pudesse aprender em uma única sessão e na qual você pudesse ver o ganho de performance em relação ao seu código para CPU era fundamental."

-- Ian Buck
Gerente Geral, NVIDIA

GPGPU

HISTÓRICO

Computação com GPU: A Revolução

Você enfrenta desafios: melhorar a performance, resolver problemas mais rapidamente. O processamento em paralelo seria mais rápido, mas a curva de aprendizado é alta, não é?

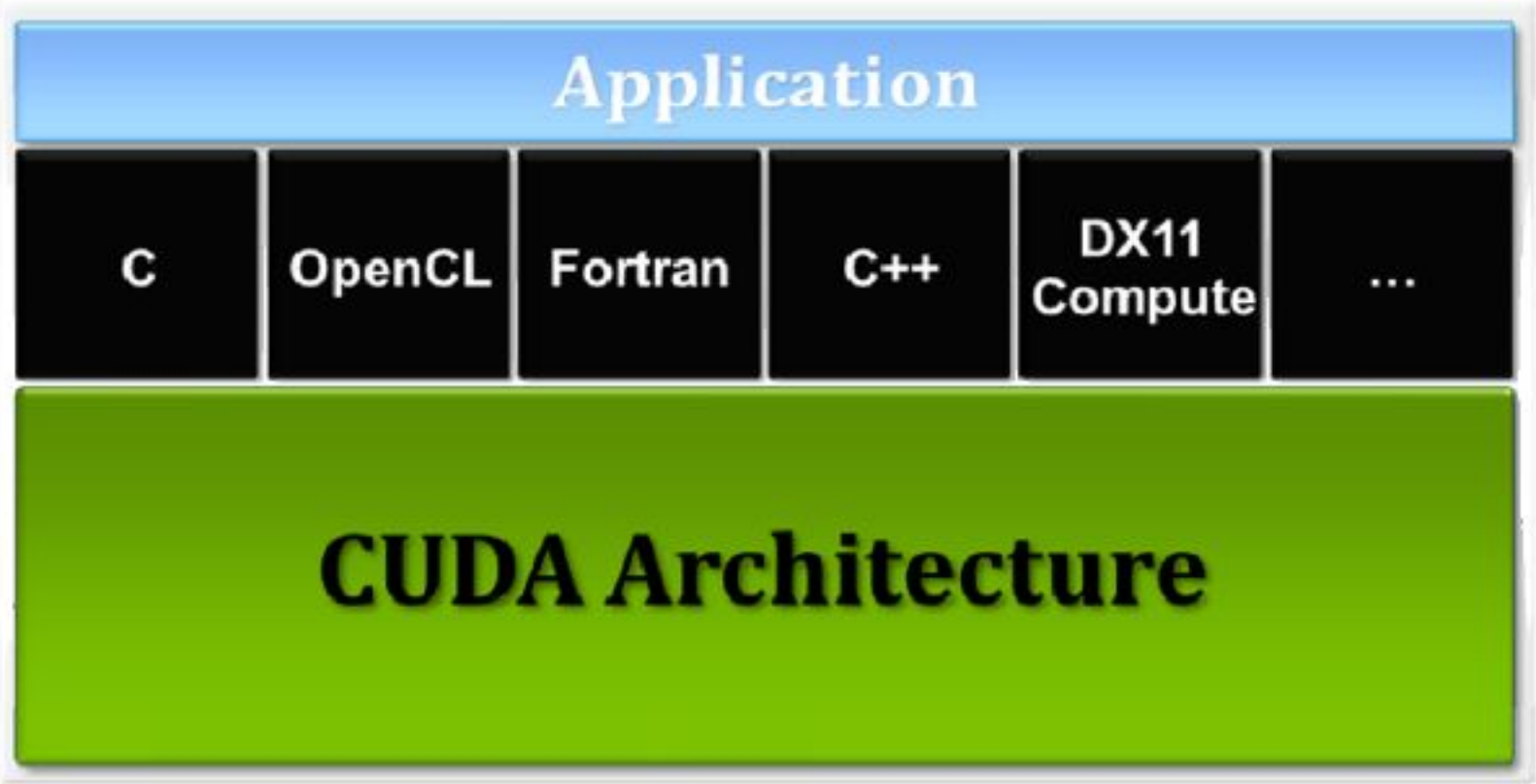
Não é mais. Com a CUDA, você pode enviar código em C, C++ e Fortran diretamente à GPU, sem precisar usar uma linguagem de compilação.

Desenvolvedores em empresas como Adobe, ANSYS, Autodesk, MathWorks e Wolfram Research estão acordando o gigante adormecido — a GPU — para executar computação científica de propósito geral e de engenharia em uma ampla variedade de plataformas.

Usando linguagens de alto nível, aplicativos acelerados por GPU executam as partes *sequenciais* de suas cargas de trabalho na CPU - que é otimizada para performance com um único segmento (thread) - ao mesmo tempo em que aceleram o *processamento em paralelo* da GPU. Isso é chamado de "computação com GPU".

A computação com GPU é possível porque a GPU de hoje faz muito mais do que processar imagens: ela lida com um teraflop de performance de ponto flutuante e processa tarefas de aplicativos projetados para tudo, desde finanças, até medicina.

Arquitetura Unificada - CUDA



Arquitetura Unificada - CUDA

Paralelismo sem esforço, baixo custo...

Aplicações	Endereço	Aumento de desempenho
Sismic Database	http://www.headwave.com	66 to 100X
Mobile Phone Antenna Simulation	http://www.acceleware.com	45X
Molecular Dynamics	http://www.ks.uiuc.edu/Research/vmd/	240X
Neuron Simulation	http://www.evolvedmachines.com	100X
MRI Processing	http://bic-test.beckman.uiuc.edu/	245 – 415X
Atmospheric Cloud Simulation	http://www.cs.clemson.edu/~jesteel/clouds.html	50X

Por que mais rápido? Tarefa 100 vezes mais rápido

Tarefa de 1 ano de duração cai para 3 dias

Tarefa de 1 dia cai para 15 minutos

Tarefa de 3 segundos cai para 30 vezes por segundo

Threads

Porque programar em Threads?

- Para fazer distribuição de carga em arquitetura single core

- Para fazer distribuição de carga entre múltiplos núcleos

- Desafio de ter que manter o máximo de uso de cada núcleo

Threads

- CUDA permite até 12 mil threads;

- CUDA é basicamente um cluster de threads.

Threads – Custo de gerenciamento

- Em CPU, como fazemos pouca troca de threads, podemos achar natural gastar 1000 instruções para fazer a troca de uma thread para outra. Em CUDA o paradigma é outro.

- Não é necessário gerenciar as threads, a priori.

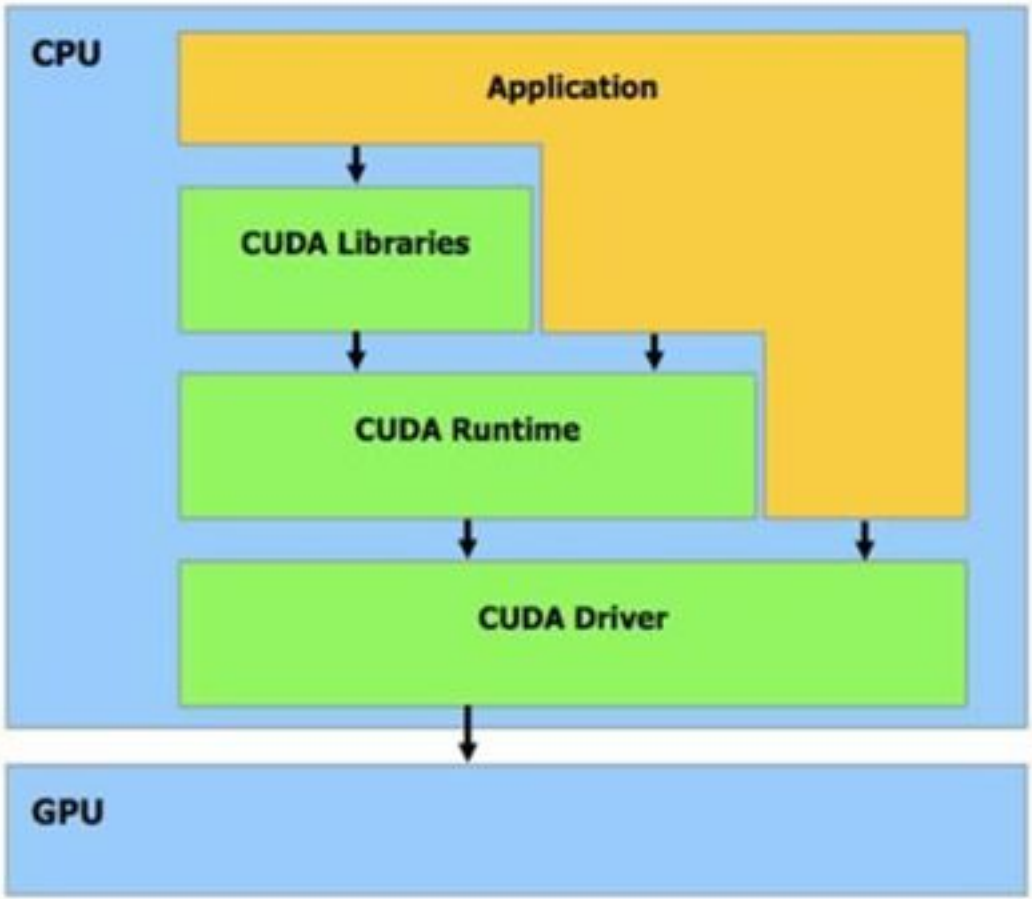
Modelo de Programação

CUDA estende a linguagem C através de kernels

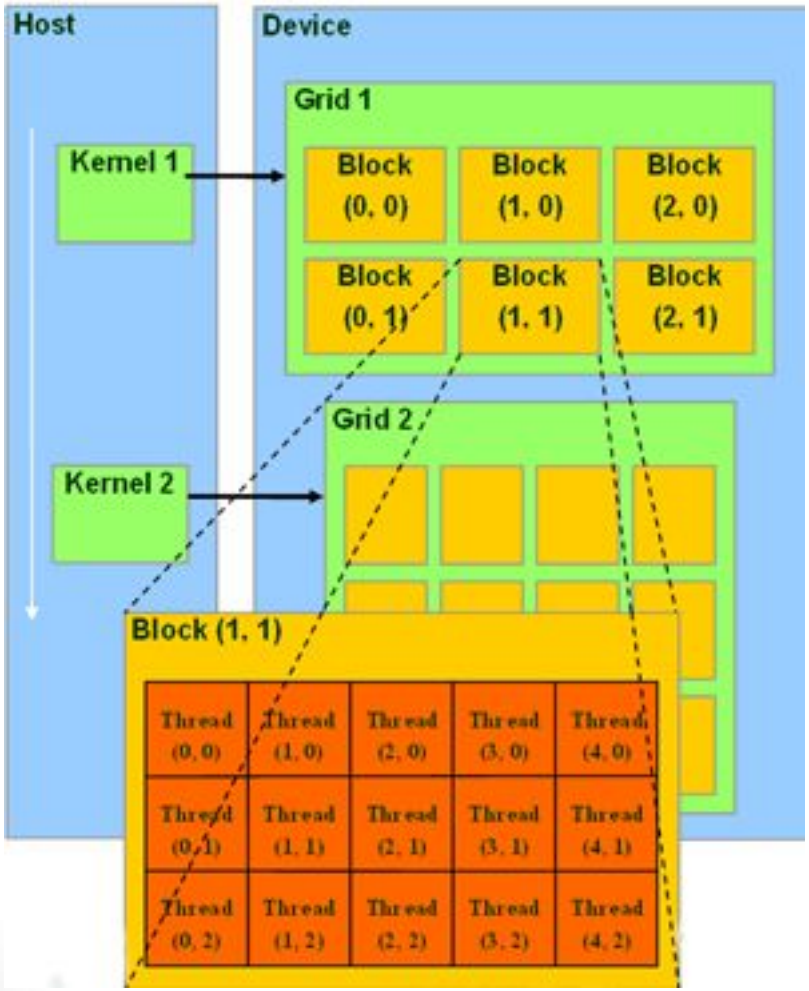
Kernels equivalem a funções, que serão executadas N vezes em paralelo

N é o número de threads

Arquitetura do Software

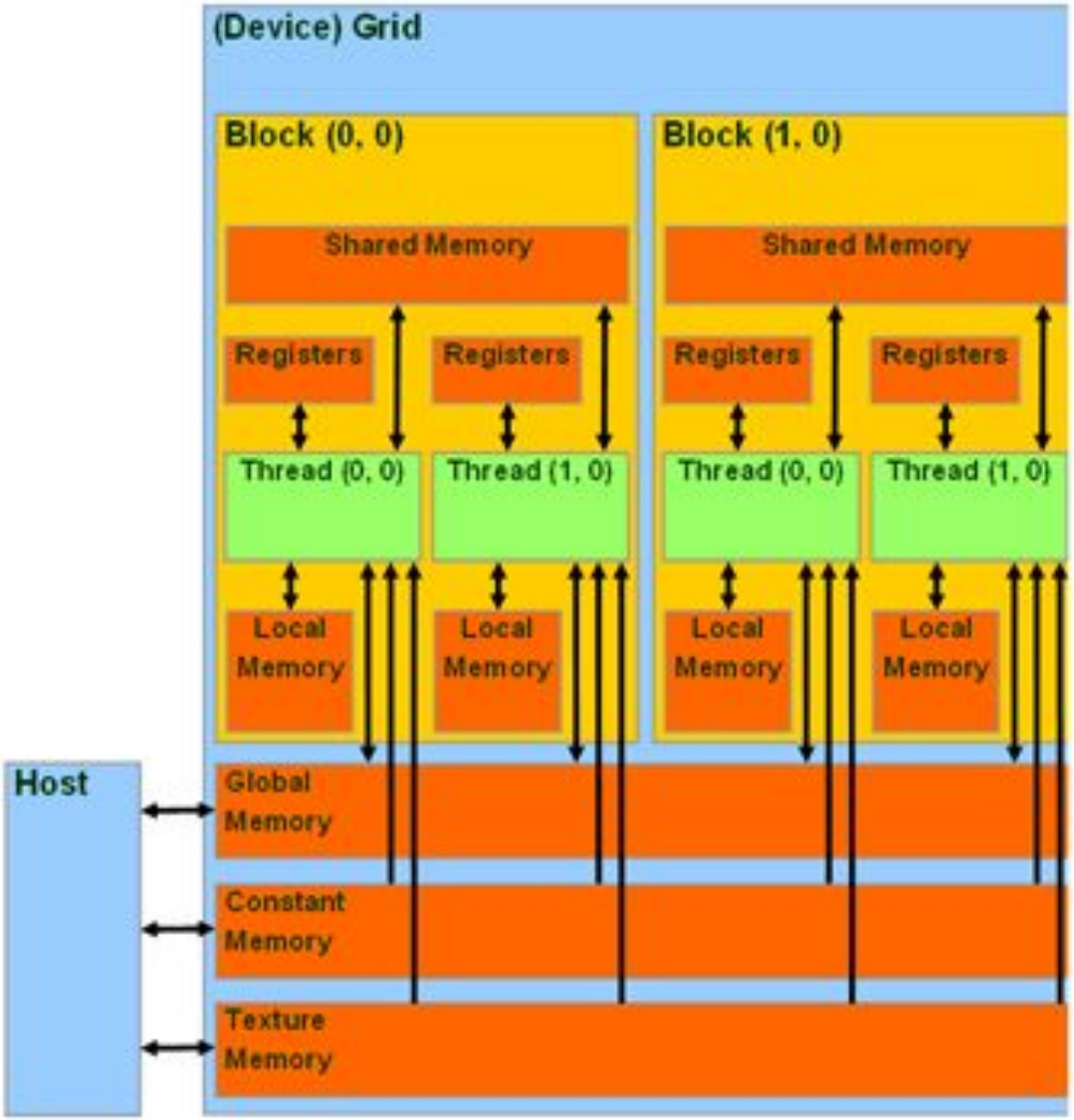


Threads, Blocos e Grids



- Um kernel corresponde a um grid de Blocos
- Cada Bloco é composto por threads
- Todas as threads compartilham a mesma área de memória
- As threads de um mesmo bloco podem compartilhar umas com as outras
- Threads de blocos diferentes não podem compartilhar memória entre si

Threads, Blocos e Grids - memórias



Hierarquia de Threads

- Todas as threads de um bloco usam da mesma memória compartilhada.
- O número de threads num bloco é limitado pela memória: GPUs possuem até 512 threads.