GPU a CUDA

Tomáš Oberhuber
tomas.oberhuber@fjfi.cvut.cz

18. března 2024

Video na Youtube

Historie GPU

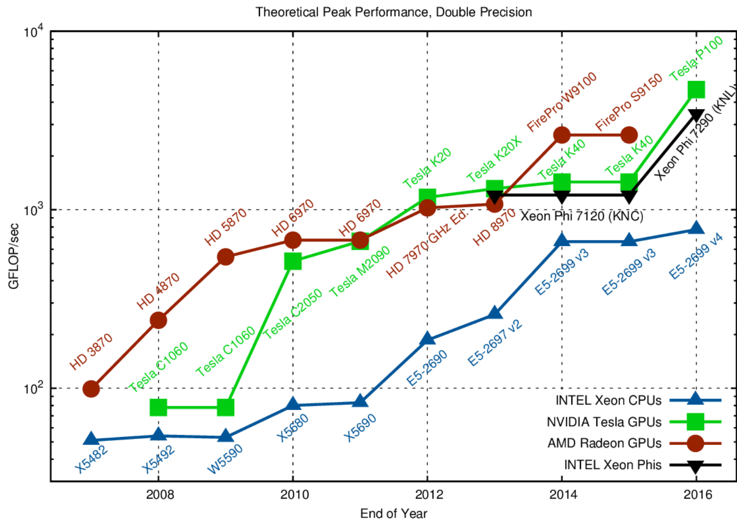
- **GPU = graphics processing unit**
- jde o akcelerátory pro algoritmy v 3D grafice a vizualizaci
- mnoho z nich původně vzniklo pro účely počítačových her
  - to byla často psychologická nevýhoda GPU
- typická úloha ve vizualizaci vypadá takto
  - transformování miliónů polygonů
  - aplikování textur o velikosti mnoha MB
  - projekce na framebuffer
- **žádná datová závislost**

Výhody GPU

Nvidia H100

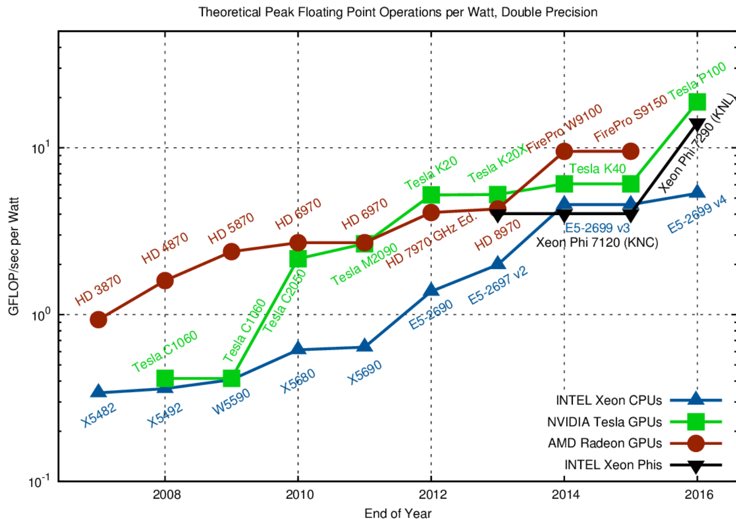
- GPU je navrženo pro **současný** běh až 16896 vláken - virtuálně až stovek tisíc vláken
- vlákna musí být **nezávislá** - není zaručeno, v jakém pořadí budou zpracována
- GPU je vhodné pro kód s intenzivními výpočty a s malým výskytem podmínek
- není zde podpora spekulativního zpracování
- GPU je optimalizováno pro **sekvenční přístup do paměti** rychlostí až 3 TB/s

Výhody GPU



Obrázek: Porovnání výkonu GPU a CPU - zdroj Karl Rupp.

Výhody GPU



Obrázek: Porovnání výkonu na watt GPU a CPU - zdroj Karl Rupp.

Porovnání CPU vs. GPU

	Nvidia Hopper H100	**AMD Epyc 9004 Genoa 96 Cores**
Výrobní proces	4 nm	5 nm
Transistory	80 miliard	90 miliard
Takt	1.78 GHz	2.4-3.7 GHz
Počet jader	16,896	96
Peak. Perf.	60 TFlops	$\approx$ 2.8 TFlops
Datová propustnost	3 TB/s	460 GB/s
RAM	80 GB	12$\times$6 TB
Příkon	700 W	360 W

Nvidia CUDA

CUDA = Compute Unified Device Architecture - Nvidia 15 February 2007

- výrazně zjednodušuje programování v GPGPU
- zcela odstraňuje nutnost pracovat s OpenGL a formulování úloh pomocí textur
- je založena na jednoduchém rozšíření jazyka C/C++
- funguje jen s kartami společnosti Nvidia
- obsahuje
  - překladač `nvcc`
  - profiler `computeprof`
  - sadu knihoven
    - cuBLAS, cuSPARSE, cuSOLVER, cuFFT, cuDNN, cuRAND, ...

Je velice snadné napsat kód pro CUDA ale je potřeba mít hluboké znalosti o GPU aby byl výsledný kód efektivní.

Architektura Hopper

Obrázek: Zdroj Nvidia

Architektura Hopper

H100

- skládá se ze 8 GPC = *Graphic Processing Cluster*
- každý GPC obsahuje 18 SMM = **Streaming Multiprocessors**
- celkem je zde 144 SMM
- všechny GPC mezi sebou sdílejí 60MB L2 cache
- dále obsahuje 80 GB HBM3 stacked memory
  - HBM = High Bandwidth Memory
  - 3D stack = návrh čipu se skládá z několika funkčních vrstev

Architektura Hopper - SMM

Architektura Hopper

Každý SMM architektury Hopper se skládá z:

- 4x16 jednotek pro výpočty s **celými čísly**
- 4x32 jednotek pro výpočty s **jednoduchou přesností** s pohyblivou des. čárkou
- 4x16 jednotek pro výpočty s **dvojitou přesností** s pohyblivou des. čárkou
- 256 kB velmi rychlé **sdílené paměti** nebo L1 cache
- sdílená paměť se dělí do 32 modulů (???)
- LD/ST jsou jednotky pro přístup do **globální paměti**
- SFU jsou jednotky pro výpočty složitých funkcí jako `sin`, `cos`, `tan`, `exp`
- **tenzorová jádra** pro rychlé násobení matic
- **DPX jednotky** pro urychlení algoritmů dynamického programování

Vlákna v CUDA

Od hardwarové architektury se odvíjí hierarchická struktura vláken:

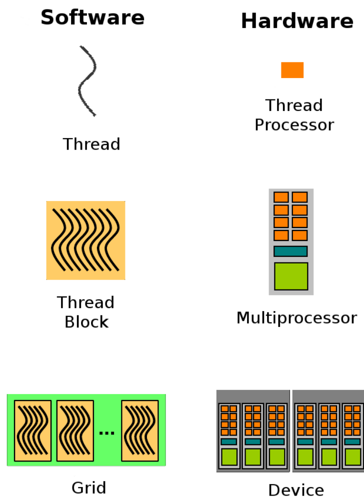
- **CUDA host** je CPU a operační paměť
- **CUDA device** je zařízení pro paralelní zpracování až stovek tisíc nezávislých vláken - threads
- **CUDA thread** je velmi jednoduchá struktura - rychle se vytváří a rychle se přepíná při zpracování
- komunikace mezi výpočetními jednotkami je hlavní problém v paralelním zpracování dat
- nemůžeme očekávat, že budeme schopni efektivně synchronizovat tisíce vláken
- CUDA architektura zavádí menší skupiny vláken zvané bloky - **blocks**

Bloky a gridy

- jeden blok je zpracován na jednom multiprocesoru
- vlákna v jednom bloku sdílejí velmi rychlou paměť s krátkou latencí
- vlákna v jednom bloku mohou být **synchronizována**
- v jednom bloku může být až 1024 vláken
  - multiprocesor přepíná mezi jednotlivými vlákny
  - tím zakrývá latence pomalé globální paměti
  - zpracovává vždy ta vlákna, která mají načtena potřebná data, ostatní načítají

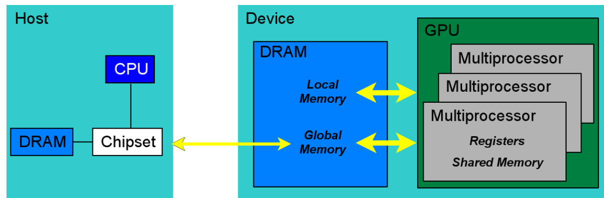
Bloky vláken jsou seskupeny do gridu - **grid**.

Model zpracování vláken



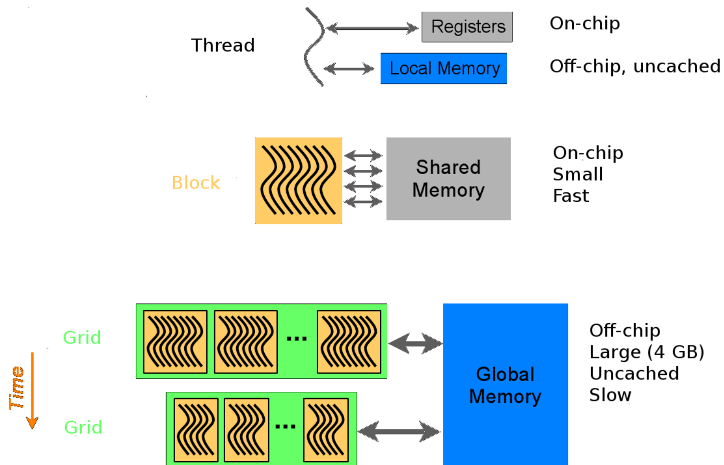
Obrázek: Zdroj Nvidia: Getting Started with CUDA

Paměťový model



Obrázek: Zdroj Nvidia: Getting Started with CUDA

Paměťová hierarchie



Obrázek: Zdroj Nvidia: Getting Started with CUDA

Programování v CUDA

- programování v CUDA spočívá v psaní kernelů - **kernels**
  - kód zpracovaný jedním vláknem
- kernely nepodporují rekurzi
- podporují větvení kódu, ale to může snižovat efektivitu
- nemohou vracet žádný výsledek
- jejich parametry nemohou být reference
- podporují šablony C++
- **od CUDA 2.0 podporují funkci** `printf` **!!!**

Příklad

Kód uložíme do `cuda-vector-addition.cu` a přeložíme pomocí `nvcc`.

Vývoj efektivního kódu

Pro získání efektivního kódu je nutné dodržet následující pravidla:

- redukovat přenos dat mezi CPU (CUDA host) a GPU (CUDA device)
- optimalizovat přístup do globální paměti
- omezit divergentní vlákna
- zvolit správnou velikost bloků

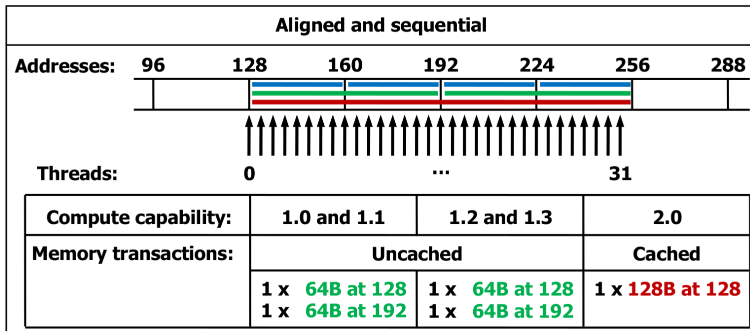
Komunikace mezi CPU a GPU

- komunikace přes PCI Express je velmi pomalá cca. 8 GB/s
- je nutné tuto komunikaci minimalizovat
  - ideálně provést jen na začátku a na konci výpočtu
- GPU se nevyplatí pro úlohy s nízkou aritmetickou intenzitou
- z tohoto pohledu mohou mít výhodu on-board GPU, které sdílí operační paměť
- pokud je nutné provádět často komunikaci mezi CPU a GPU pak je dobré jí provádět formou pipeliningu
- je možné provádět najednou
  - výpočet na GPU
  - výpočet na CPU
  - kopírování dat z CPU do GPU
  - kopírování dat z GPU na CPU

Sloučené přístupy do pamětí

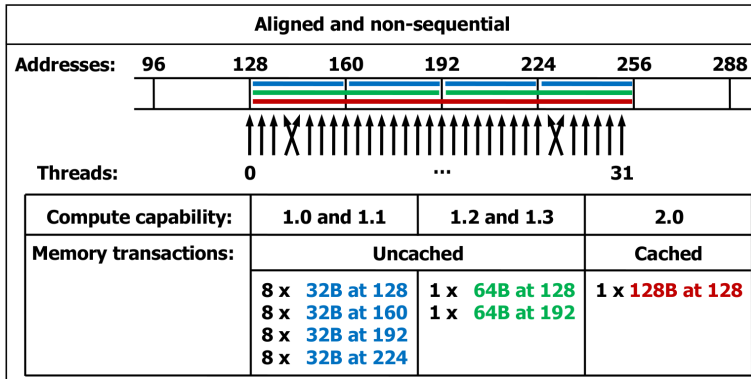
- většinu přístupů GPU do globální paměti tvoří načítání textur
- GPU je **silně optimalizováno pro sekvenční přístup do globální paměti**
- programátor by se měl vyhnout náhodným přístupům do globální paměti
- ideální postup je:
  - načíst data do sdílené paměti multiprocesoru
  - provést výpočty
  - zapsat výsledek do globální paměti
- sloučený přístup - **coalesced memory access** - může velmi výrazně snížit (až 32x) počet paměťových transakcí

Sloučené přístupy do paměti



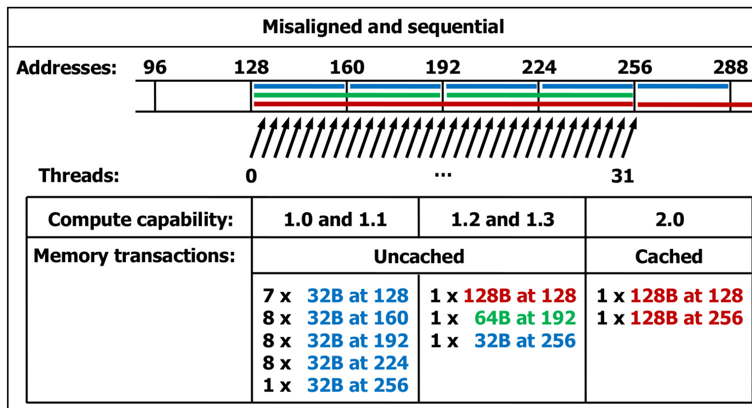
Obrázek: Zdroj Nvidia: Nvidia CUDA programming guide

Sloučené přístupy do paměti



Obrázek: Zdroj Nvidia: Nvidia CUDA programming guide

Sloučené přístupy do paměti



Obrázek: Zdroj Nvidia: Nvidia CUDA programming guide

Sdílená paměť multiprocesoru

- sdílená paměť může fungovat jako cache nebo může být řízena programátorem
- sdílená paměť multiprocesoru je rozdělena na 32 paměťových bank
- data se ukládají do jednotlivých bank vždy po 4 bajtech
- je potřeba se vyhnout situaci, kdy dvě vlákna ze skupiny 32 čtou z různých adres v jedné bance
- nevadí, když čte více vláken ze stejné adresy, použije se broadcast

Divergentní vlákna

- CUDA device umí zpracovávat současně různé kernely, ale jen na různých multiprocesorech
- Nvidia tuto architekturu nazývá SIMT = **Single Instruction, Multiple Threads**
- v rámci jednoho multiprocesoru jde ale o SIMD architekturu, tj. všechny jednotky provádějí stejný kód
- **warp** je skupina 32 vláken zpracovávaných současně
  - vlákna ve warpu jsou tedy implicitně synchronizovaná
  - všechna by měla zpracovávat stejný kód

Zpracování bloků vláken na multiprocesoru

- na mutliprocesoru většinou běží více bloků vláken
- scheduler mezi nimi přepíná a spouští vždy ty bloky vláken, které mají načteny potřebná data
  - tím se zakrývají velké latence globální paměti
- k tomu je ale potřeba, aby jeden blok nevyčerpal všechny registry a sdílenou paměť
  - pokud není dostatek registrů, ukládají se proměnné do *local memory* - to je pomalé
  - je potřeba dobře zvolit velikost bloku - násobek 32
  - minimalizovat počet proměnných a množství sdílené paměti použité jedním blokem
  - minimalizovat velikost kódu kernelu
- efektivnost obsazení multiprocesoru udává parametr zvaný **occupancy** (maximum je 1.0)
- za účelem optimalizace lze použít
  - CUDA occupancy calculator [1]
  - CUDA profiler
  - výpisy `nvcc -ptxas-options=-v`

[1] http://developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls

GPU od AMD a Intelu

Dalším dodavatelem výkonných GPU je firma AMD nebo Intel.

	Nvidia Hopper H100 NVL	**AMD Instinct MI300X**	**Intel GPU MAX 1350**
Výrobní proces	4 nm	5-6 nm	10 nm
Transistory	2x80 miliard	153 miliard	100 miliard
Takt	1.98 GHz	2.1 GHz	0.75-1.55 GHz
Počet jader	2x16,896	19,456	14,336
Peak. Perf.	2x67 TFlops	81.7 TFlops	44.44 TFlops
Tensor Peak. Perf.	2x1,980 TFlops	2,614 TFlops	???
Datová propustnost	2x3.9 TB/s	5.3 TB/s	2.45 TB/s
RAM	2x94 GB	192 TB	96 GB
Příkon	700 W	750 W	450 W

Programování GPU od AMD

ROCm = Radeon Open Compute platform

- softwarový open-source balík pro programování GPU od AMD, který obsahuje např.
- C++ překladač `hipcc`
- profiler `rocprof`
- obsahuje knihovny jako
  - rocBLAS, rocRAND, rocFFT, rocSPARSE, rocAI, rocMPI

Programování GPU od AMD

HIP = Heterogeneous-compute Interface for Portability

- HIP je rozhraní pro vývoj programů pro GPU
- je součástí ROCm
- obsahuje C++ překladač `hipcc`
- obsahuje knihovny jako
  - hipBLAS, hipRAND, hipFFT, hipSPARSE, hipAI, hipMPI
- umožňuje kompilaci programů pro GPU od AMD i NVIDIA
- rozhrání je inspirováno rozhraním CUDA
- místo prefixu `cuda` mají funkce prefix `hip`

Programování GPU od Intelu

Intel oneAPI

- Jde o jednotné rozhraní pro programování CPU, GPU ale i FPGA.
- Mělo by podporovat i GPU od AMD a Nvidia.

Multi-GPU systémy

- v rámci jednoho výpočetního uzlu lze instalovat až 8 GPU
- dostáváme tak systém s distribuovanou pamětí
  - jednotlivá GPU nemají přímý přístup do paměti těch ostatních
- více v CUDA Programming Guide