

Architektura počítačů

Paralelní zpracování

http://d3s.mff.cuni.cz/teaching/computer_architecture/



Lubomír Bulej

bulej@d3s.mff.cuni.cz

CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

Paralelní zpracování



● Dosavadní metody paralelizace

- Paralelismus na úrovni instrukčních kroků
 - Pipelining
- Paralelismus na úrovni instrukcí
 - Superskalární zpracování

● Další metody paralelizace

- Paralelismus na úrovni dat
 - Vektorové zpracování
 - Masivně paralelní zpracování
- Paralelismus na úrovni procesů



Proč je těžké psát paralelní programy?



- **Paralelní program musí být rychlejší než sekvenční**
 - Jinak nedává smysl ho psát
- **Očekává se, že bude škálovat s počtem procesorů**
 - S rostoucím počtem procesorů stále obtížnější
 - Plánování, synchronizace, load-balancing
- **Připomenutí: Amdahlův zákon**
 - Limit teoretického zrychlení sekvenčního algoritmu při paralelním zpracování
 - n ... počet paralelně běžících vláken
 - $B \in \langle 0, 1 \rangle$... poměrná část neparalelizovatelné části algoritmu

$$\text{Speedup}(n) = \frac{1}{B + \frac{(1-B)}{n}}$$



Co brání zrychlování?



- **Sekvenční část**

- Jaký program mohu 90x zrychlit na 100 procesorech?

$$Speedup(n) = \frac{1}{B + \frac{(1-B)}{n}}$$

- $n = 100$, $Speedup(n) = 90$

$$B = -\frac{Speedup(n) - n}{(n-1) \times Speedup(n)} = -\frac{90 - 100}{99 \times 90} = 0.0012$$

- Sekvenční část může být pouze 0.1% programu.



Co brání zrychlování?



• Velikost problému (weak/strong scaling)

- Součet 10 proměnných (sekvenční) a součet dvou matic 10x10 (paralelní), čas na součet t .
 - Na 1 procesoru: $100t + 10t = 110t$
 - Na 10 procesorech: $(100/10)t + 10t = 20t$
 - Zrychlení $110t/20t = 5.5$ (55% potenciálního zrychlení)
 - Na 40 procesorech: $(100/40)t + 10t = 12.5t$
 - Zrychlení $110t/12.5t = 8.8$ (22% potenciálního zrychlení)
- Součet 10 proměnných a součet dvou matic 20x20.
 - Na 1 procesoru: $400t + 10t = 410t$
 - Na 10 procesorech: $(400/10)t + 10t = 50t$
 - Zrychlení $410t/50t = 8.2$ (82% potenciálního zrychlení)
 - Na 40 procesorech: $(400/40)t + 10t = 20t$
 - Zrychlení $410t/20t = 20.5$ (51% potenciálního zrychlení)



Co brání zrychlování?



● Load-balancing

- Součet 10 proměnných a součet dvou matic 20x20 na 40 procesorech.
 - Předpoklad rovnoměrně rozdělené práce, tj. každý procesor vykonal 2.5% práce.
- Co kdyby jeden procesor udělal 2x více (5%)?
 - Jeden procesor udělá 20 součtů (5% ze 400), ostatních 39 se podělí o zbývajících 380.
 - Čas zpracování je $\text{Max} (380t/39, 20t/1) + 10t = 30t$
 - Zrychlení klesá z 20.5 na $410t/30t = 14$. Ostatních procesory jsou využity méně než polovinu času, počítají pouze $380t/39 = 9.7t$
- Co kdyby jeden procesor udělal 5x více (12.5%)?
 - Čas zpracování je $\text{Max} (350t/39, 50t/1) + 10t = 60t$
 - Zrychlení klesá z 20.5 na $410t/60t = 7$. Ostatních procesory jsou využity méně než 20% času.

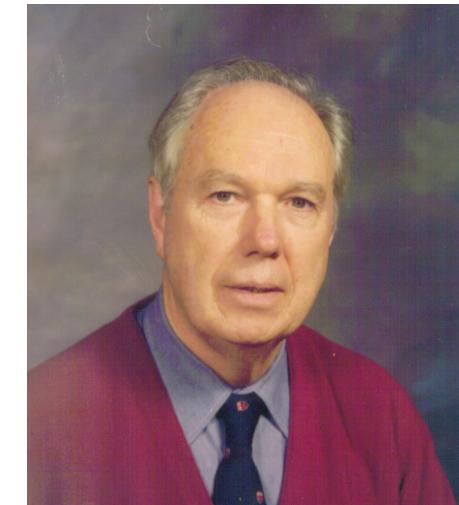


Flynnova taxonomie



● Architektura zpracování dat

- Michael J. Flynn (1966)
- **SISD:** *Single Instruction, Single Data stream*
 - Tradiční skalární procesor (von Neumann)
- **SIMD:** *Single Instruction, Multiple Data streams*
 - Vektorový procesor
- **MISD:** *Multiple Instruction, Single Data stream*
 - Potenciálně heterogenní systém odolný vůči výpadkům
- **MIMD:** *Multiple Instruction, Multiple Data streams*
 - Víceprocesorový systém
 - Programovací model pro MIMD: SPMD (Single Program Multiple Data)
- Superskalární procesory na pomezí SISD/SIMD



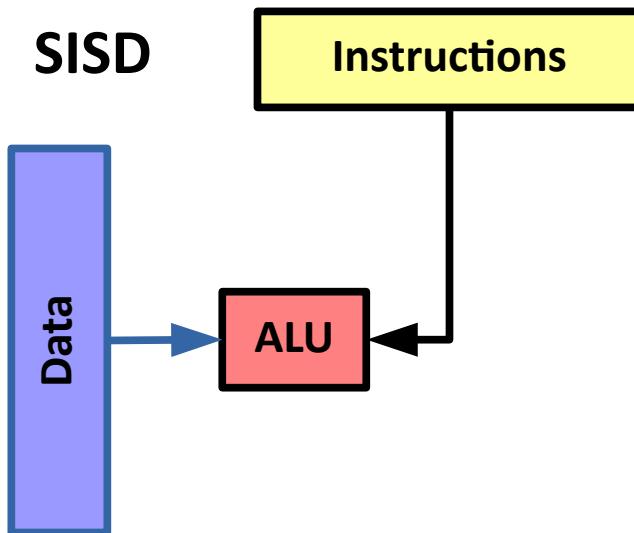
[1]



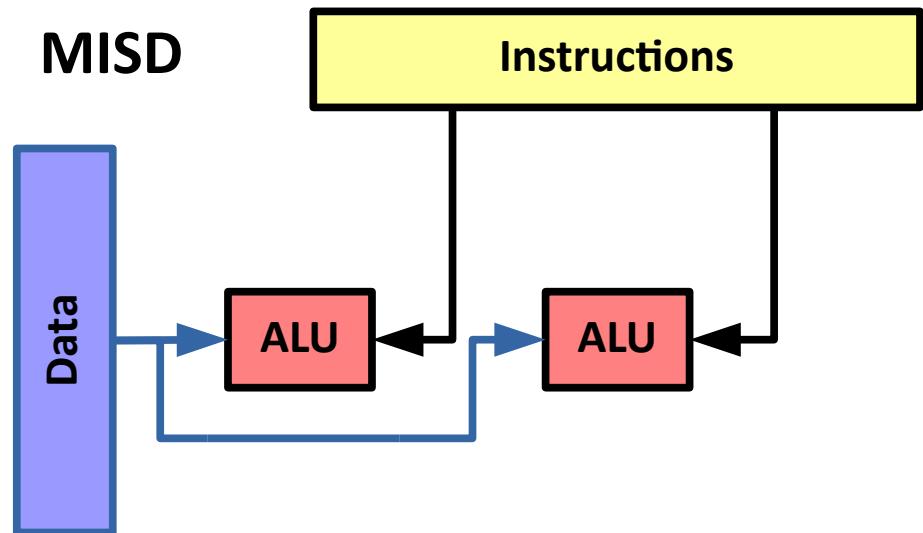
Flynnova taxonomie (2)



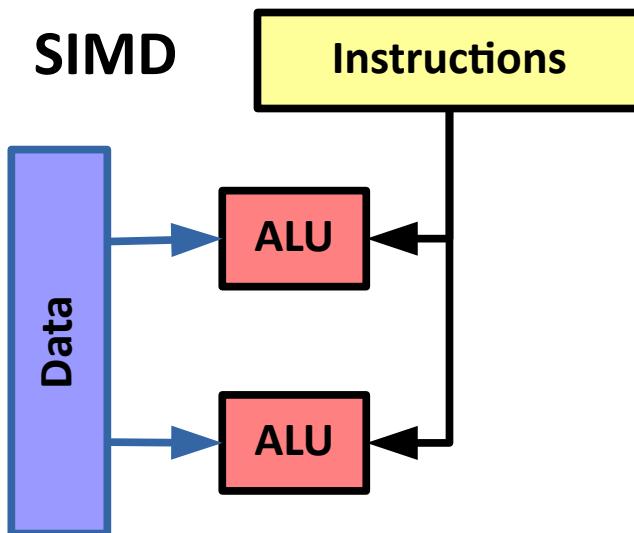
SISD



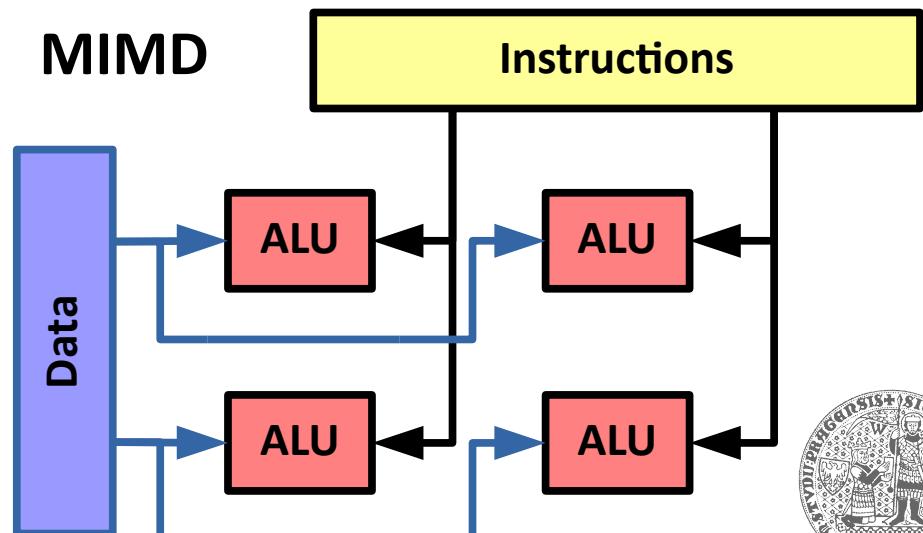
MISD



SIMD



MIMD



Vektorové zpracování



● Příklad: Sčítání tří dvojic jednobytových čísel

■ Skalární zpracování

for {0, 1, 2} → i:

mem_a[i] → r1 (*rozšíření na velikost slova*)

mem_b[i] → r2 (*rozšíření na velikost slova*)

add(r1, r2) → r3

r3 → mem_c[i] (*restrikce na 8 bitů*)

■ Vektorové zpracování

mem_a[0, 1, 2, 3] → vr1

mem_b[0, 1, 2, 3] → vr2

vector_add(vr1, vr2) → vr3 (*sčítání po složkách*)

vr3 → mem_c[0, 1, 2, 3]



Vektorové zpracování (2)



● Vektorová rozšíření skalární instrukční sady

■ IA-32

● MMX (*MultiMedia eXtensions*, od Pentia MMX)

- 64bitové registry MM0 až MM7 (aliasy na FPU registry)
 - 1× 64bitový integer (jen několik operací), 2× 32bitový integer, 4× 16bitový integer, 8× 8bitový integer
- Logické operace, bitové posuny, sčítání, odčítání, násobení, porovnávání (vše po složkách), konverze formátů
- Saturační aritmetika
 - Omezení výsledku na interval <min, max>

● 3DNow! (od K6-2)

- 2× 32bitový single precision
- Floating point aritmetika (po složkách), celočíselné průměrování, hledání maxima atd.



Vektorové zpracování (3)



● Vektorová rozšíření skalární instrukční sady

■ IA-32/AMD64

● SSE (*Streaming SIMD Extensions*, od Pentia III)

- SSE, SSE2, SSE3, SSSE3, SSE4, XOP, FMA4, CVT16
- 128bitové registry XMM0 až XMM7, později až XMM15
 - 4× 32bitový single precision
 - 2× 64bitový double precision, 2× 64bitový integer, 4× 32bitový integer,
 - 8× 16bitový integer, 16× 8bitový integer
- Aritmetické, logické, konverzní operace
- Kombinované aritmetické operace (aritmetika po složkách „do kříže“ apod.)
- Aritmetika s komplexními čísly
- Fused multiply-add, permutace složek
- Výpočet CRC32, počet jedničkových bitů



Vektorové zpracování (4)



● Vektorová rozšíření skalární instrukční sady

- IA-32/AMD64
 - **AVX** (*Advanced Vector Extensions*, od Sandy Bridge/Bulldozer)
 - AVX2, AVX-512 (od Haswell)
 - 256bitové registry YMM0 až YMM15
 - AVX-512: 512bitové registry ZMM0 až ZMM15
 - Tříoperandový formát instrukcí
- Power/PowerPC
 - **Altivec** (od G4, POWER6)
 - Přibližně srovnatelné s SSE



Vektorové zpracování (5)



● Vektorová rozšíření skalární instrukční sady

- SPARC
 - **VIS** (*Visual Instruction Set*, od UltraSPARC II)
 - RISCová filozofie, jednodušší vektorové operace
- MIPS
 - **MDMX** (*MIPS Digital Media eXtension*), **MIPS-3D**
 - Přibližně srovnatelné s MMX
- ARM
 - **Advanced SIMD/NEON**
 - Přibližně srovnatelné s SSE



Víceprocesorové systémy



● Široké spektrum implementací

■ Vícejádrové procesory

- Vlastně jen důsledněji oddělené pipelines superskalárního procesoru
- Některé části mohou být mezi jádry stále sdílené (L2 cache)

■ Víceprocesorové systémy

● *Symmetric/Shared Memory Multiprocessing (SMP)*

- Nezávislá obecná jádra v „oddělených pouzdrech“
- Sdílený přístup do hlavní paměti (*Uniform Memory Access, UMA*)

● Specializované koprocesory

- Nezávislá jádra, některá z nich velmi specializovaná (Cell)

● *Non-Uniform Memory Access (NUMA)*

- Nezávislá jádra s lokální hlavní pamětí
- Přístup do „nelokální“ paměti jiného jádra může být transparentní (ovšem dražší) nebo netransparentní (řízen softwarem)

● Clustery, některé typy cloudu (IaaS)



Masivně paralelní zpracování



● Základní myšlenka

- SIMD/MIMD architektura s (relativně) jednoduchými procesory a (relativně) malou lokální pamětí
- Omezená a řízená komunikace mezi procesory
 - **CM-1 Connection Machine (1983)**
 - 65536 1bitových procesorů, každý 4 kb RAM
 - Propojovací síť topologie hyperkrychle
 - Společná řídící logika, I/O procesor
 - Vhodné pro extrémně dobře paralelizovatelné úlohy
 - Umělá inteligence
 - Symbolické manipulace



[2]



Masivně paralelní zpracování (2)



● Evoluce grafických procesorů (GPU)

- Původně koprocesory pro urychlování konkrétních grafických operací
 - Kreslení grafických primitiv (čáry, kružnice)
 - Přesuny bloků obrazové paměti bez zásahu CPU (*blit*)
- Později implementace složitějších operací pro 3D grafiku
 - Efektivní pipelining na úrovni jednotlivých fází kreslení 3D scény
 - Geometrické transformace
 - Řešení viditelnosti
 - Mapování textur
 - Úpravy na úrovni pixelů
 - Algoritmy „zadrátovány“ do hardware



Masivně paralelní zpracování (3)

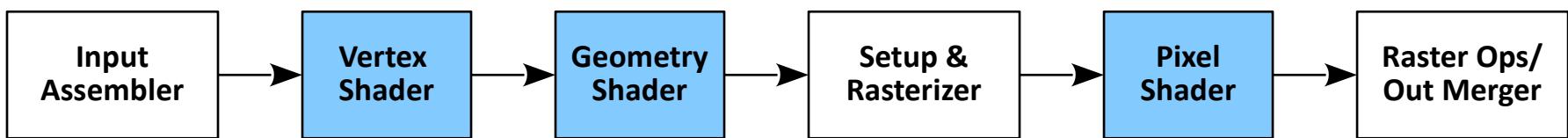


• Evoluce grafických procesorů (GPU)

■ Programovatelný zásah do grafické pipeline

• *Vertex shaders, pixel shaders*

- Původně jen jednoduchá a krátká sekvence vektorových operací (často bez možnosti cyklů, omezené větvení)
- Paralelní aplikace stejného algoritmu na všechny hrany scény (resp. všechny pixely) současně
- První pokusy o využití pro „negrafické“ výpočty
 - Víceméně jako vedlejší efekt velkého floating-point výkonu



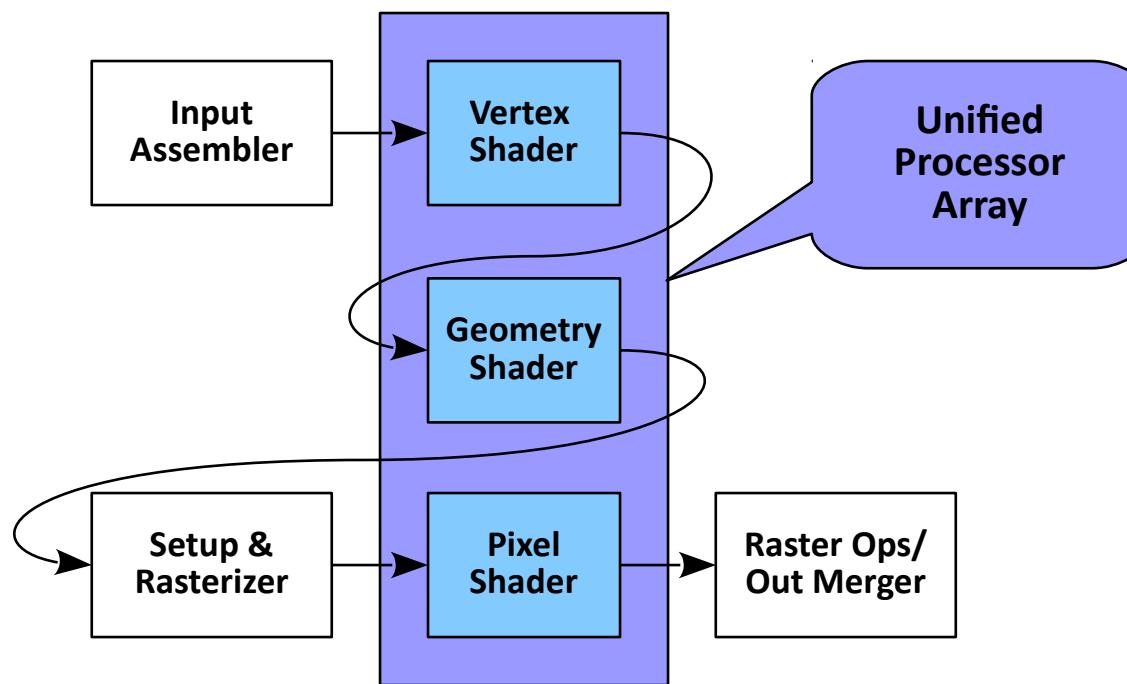
Masivně paralelní zpracování (4)

...

- **Evoluce grafických procesorů (GPU)**

- *General-Purpose Computing GPU (GPGPU)*

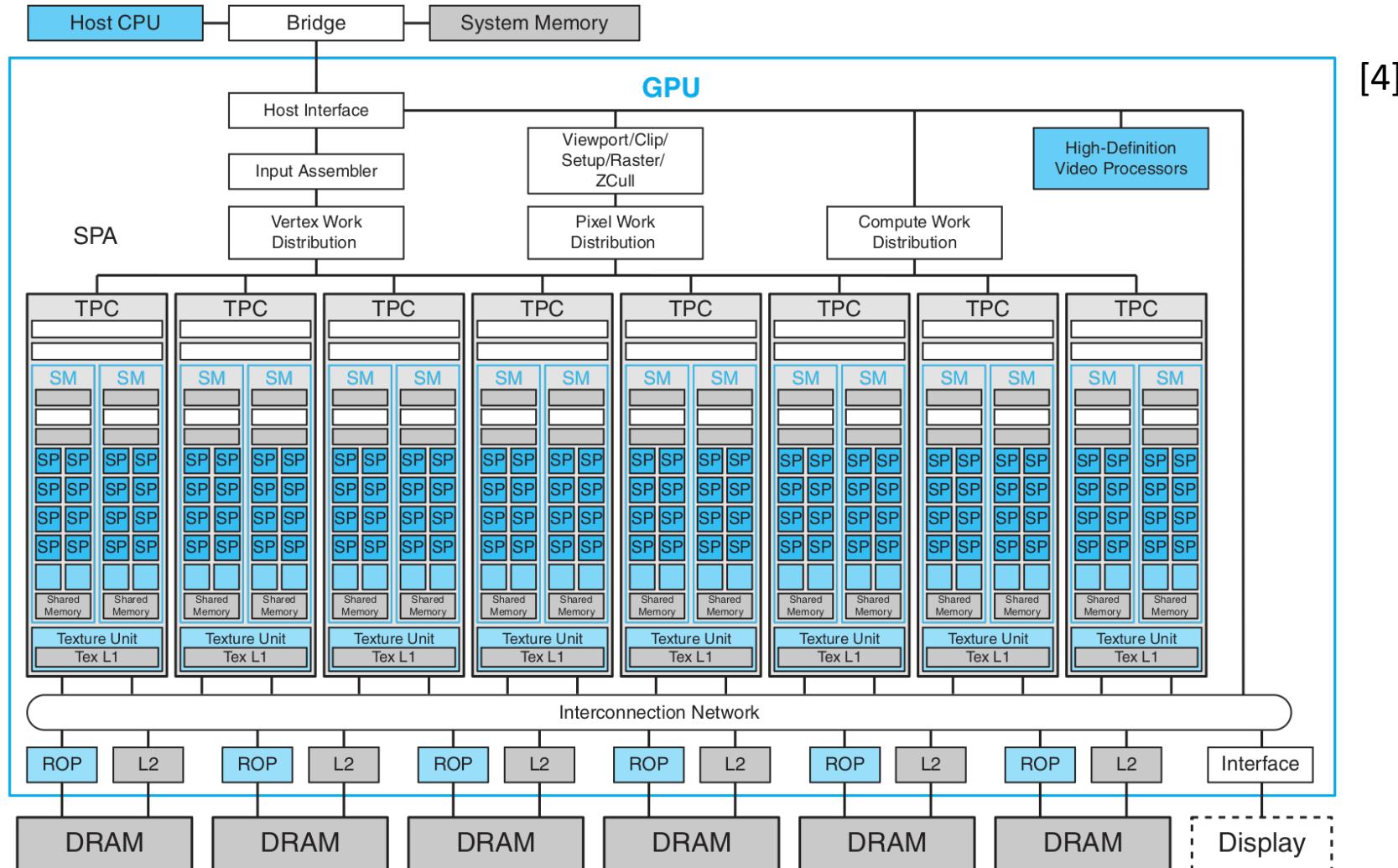
- Čím dál větší část „zadrátovaných“ grafických algoritmů řešena SIMD kódem
- Masivně paralelní architektura



Masivně paralelní zpracování (5)

...

• GeForce 8800 (NVIDIA Tesla)

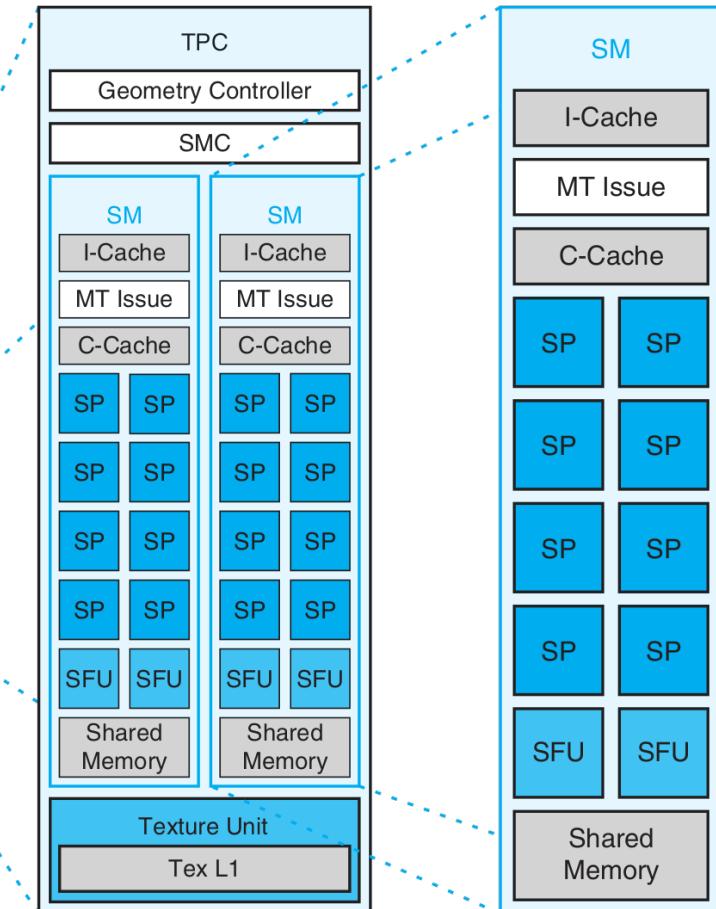
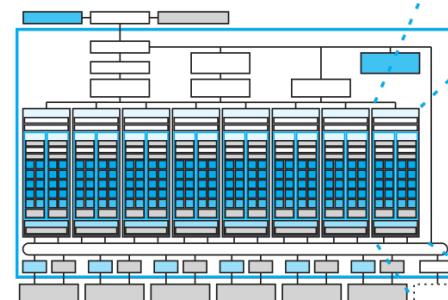


Masivně paralelní zpracování (7)

● GeForce 8800 (NVIDIA Tesla)

■ Texture/Processor Cluster

- 2x Streaming Multiprocessor
 - 8x Streaming Processor
 - 2x Special Function Unit
- Texture Unit (texture cache)



■ Streaming Processor

- 96 HW vláken
- 1024 32-bit FP registrů
- běžné FP operace

■ Special Function Unit

- transcendentní funkce
- interpolace atributů pixelů

[4]



Masivně paralelní zpracování (8)

[4]

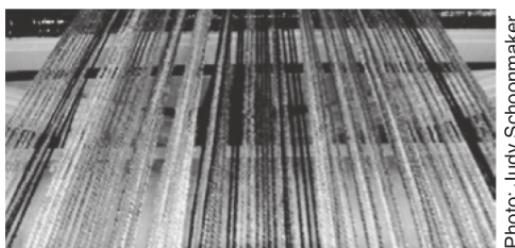
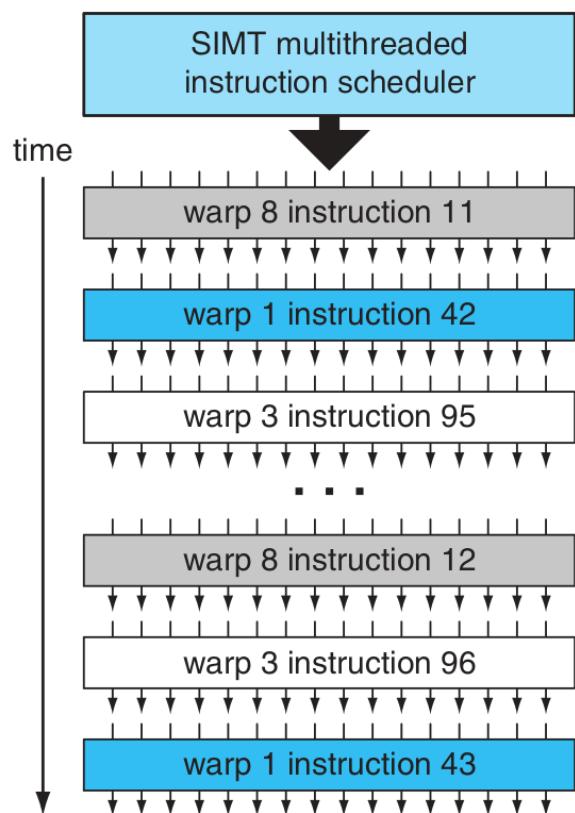


Photo: Judy Schoonmaker



- **GeForce 8800 (NVIDIA Tesla)**

- **warp = skupina 32 vláken**
- **instrukce vykonána všemi vlákny ve 4 taktech**
- **plánování warpů podle dostupnosti dat a priorit**



Masivně paralelní zpracování (9)



● Odlišnosti v cílech návrhu CPU a GPU

- CPU musí dobře zvládat různé typy úloh, ať už paralelních, nebo sekvenčních
- u GPU se předpokládá (ideálně) datově paralelní úloha, všechna vlákna vykonávají stejný program (kernel)
- GPU nepasuje do Flynnovy taxonomie, v případě GPU se hovoří o SIMT
 - datový paralelismus identifikován dynamicky, na rozdíl od vektorového/SIMD zpracování (analogické k dynamické identifikaci instrukčního paralelizmu u superskalárních procesorů na rozdíl od VLIW)

● CPU: minimalizace latence pozorovatelná jedním vláknem

- Velké cache, hit ratio > 99%, working set řádu jednotek MB
- Jednotky/desítky vláken, sofistikované řízení

● GPU: maximalizace propustnosti všech vláken

- Počet aktivních vláken omezen prostředky => velké množství prostředků
- Masivní multi-threading maskuje latenci => malé cache, hit ratio > 90%
- Working set v řádu stovek MB, nevykazuje silnou časovou lokalitu
- Stovky vláken, sdílená řídící logika



- **Aneb co si z předmětu odnést ...**

- ***Správné fungování vyšších vrstev abstrakce závisí na správném využití nižších vrstev abstrakce***

- Veškeré programy, ať už jsou napsány v čemkoliv, nakonec běží ve strojovém kódu na procesoru

- ***Základy vnitřního fungování procesoru se dají snadno pochopit***

- Skutečné procesory se od těch z přednášky liší jen tím, že několik set chytrých lidí mělo mnoho roků na přemýšlení, jak z nich dostat maximum (za danou cenu)

- ***Občas je dobré podívat se do historie***

- Nové inovativní myšlenky stále přicházejí, ale obvykle stojí na dřívějších zkušenostech, pokusech a omylech



- **Aneb co si z předmětu odnést ...**

- ***Moorův „zákon“ není zákon***

- Jeho udržování v platnosti je pracné
 - Neznamená automatické zvyšování hrubého sekvenčního výkonu
 - Je potřeba přemýšlet nad novými paradigmami (paralelní zpracování)

- ***Zachytit výkonnost počítače nelze jedním číslem***

- Občas ani dvě čísla nestačí
 - Vždy je důležité vědět, s jakým workloadem pracuji, zda jej můj benchmark správně approximuje, jakou metriku vlastně optimalizuji atd.



- **Aneb co si z předmětu odnést ...**

- **Počítače jsou složeny jako stavebnice**

- Sekvenční obvody, kombinační obvody, funkční bloky, logická hradla, logické operace

- **Applikační binární rozhraní (ABI)**

- Sada konvencí společná hardwaru a softwaru
 - Kódování celých a necelých čísel, big/little endian, kódování instrukcí, využití registrů, využití instrukční sady (volání podprogramů atd.)



Shrnutí (4)



- **Aneb co si z předmětu odnést ...**

- ***Optimalizace má největší účinek pro nejčastější případy***
 - Amdahlův zákon
 - Pozor na marná očekávání
- ***Když nelze zvýšit hrubý výkon, je možné zvyšovat propustnost***
 - Vícecyklové zpracování, pipelining, paralelní zpracování, zpracování mimo pořadí, spekulativní provádění, masivně paralelní architektury
 - Na výkonnosti se více či méně podílejí všechny části řetězce (programátor, překladač, procesor, paměťový subsystém atd.)
 - Výkonnost programu na dnešních počítačích může extrémně ovlivnit (ne)správné využití cache



Reference



- [1] http://upload.wikimedia.org/wikipedia/commons/0/0e/Michael_Flynn.jpg
- [2] <http://upload.wikimedia.org/wikipedia/commons/6/61/Frostburg.jpg>
- [3] D. A. Patterson, J. L. Hennessy. Computer Organization & Design. 5th ed., Elsevier, 2014
- [4] E. Lindholm et al. NVIDIA Tesla: A Unified Graphics and Computing Architecture. IEEE Micro, 28(2):39-55, 2008

