

# GPU akcelerace aplikací v METACentru

Jiří Filipovič  
fila@mail.muni.cz

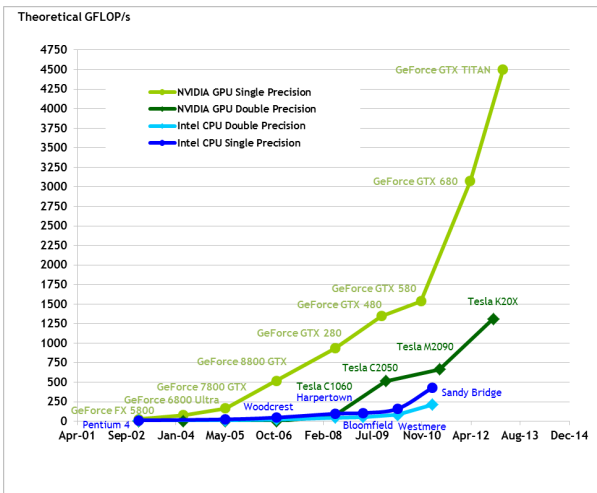
25. 11. 2013

# Proč GPU akcelerace?

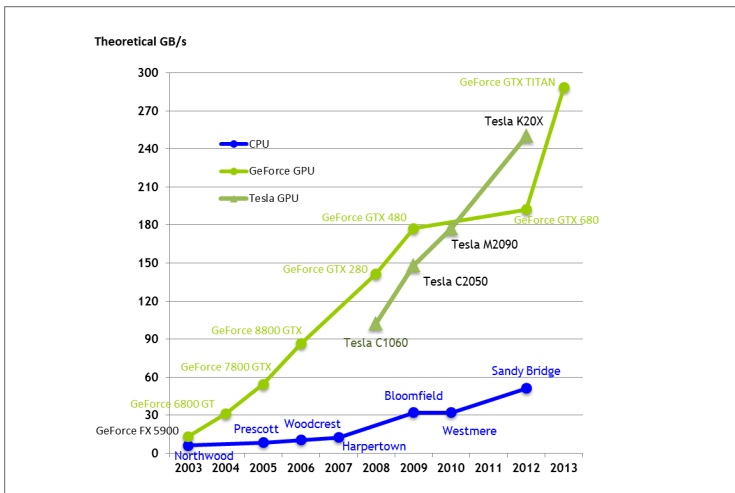
GPU jsou výkonné masivně paralelní procesory, ve srovnání s CPU

- nabízejí tisíce výpočetních jader, celkově řádově vyšší aritmetický výkon (jednotky TFlops)
- vyšší přenosová kapacita pamětí (stovky GB/s)

# Proč GPU akcelerace?



# Proč GPU akcelerace?



# Proč GPU akcelerace?

Výkon GPU lze využít v mnoha výpočetně náročných aplikacích

- mnoho aplikací využívaných v METACentru umí GPU akceleraci
- v METACentru máme 3 clustery osazeny GPU
  - doom.metacentrum.cz
  - gram.zcu.cz
  - konos.fav.zcu.cz

V přednášce se zaměřím na obecné vysvětlení, co lze na GPU akcelarovat a demonstruji GPU akceleraci u nejpoužívanějších aplikací v METACentru.

# Dostupná GPU

## GeForce 465

- 2x v každém uzlu konos.fav.zcu.cz (celkem 9 uzlů)
- architektura Fermi, 855 GFlops, 102,6 GB/s
- herní GPU, několik významných omezení
  - výkon v DP redukován na 1/8 výkonu SP
  - paměť 1 GB
  - není ECC, správa GPU, GPU Direct

# Dostupná GPU

## Tesla M2090

- 4x v každém uzlu gram.zcu.cz (celkem 10 uzlů)
- architektura Fermi, 1332 GFlops, 177 GB/s
- výpočetní GPU
  - výkon v DP redukován na 1/2 výkonu SP
  - paměť 6 GB
  - ECC (volitelně), správa GPU, GPU Direct

# Dostupná GPU

## Tesla K20M

- 2x v každém uzlu doom.metacentrum.cz (celkem 30 uzlů)
- architektura Kepler, 3.52 TFlops, 208 GB/s
- výpočetní GPU
  - výkon v DP redukován na 1/3 výkonu SP
  - paměť 5 GB
  - ECC (volitelně), správa GPU, GPU Direct



# Jak s GPU pracovat

Job si musí o GPU zažádat

- např. pomocí `-l nodes=1:gpu=4:ppn=16`
- specializované fronty (`gpu`, `gpu_long`)
- vidíte jen ty GPU, které vám systém přidělil

Manuální nastavení viditelnosti GPU

- používejte opatrně
- buď to lze zadat přímo spouštěné aplikaci, nebo pomocí proměnné `CUDA_VISIBLE_DEVICES` (seznam ID viditelných GPU)

# V čem jsou GPU jiná?

## Hlavní rozdíly oproti CPU

- masivně paralelní procesor, GPU kód běží v desítkách tisíc vláken
- určité skupiny vláken běží v lock-step módu
- koprocesor s vlastní pamětí, připojený přes PCI-E
- paměť GPU relativně malá

Nyní se podíváme na jejich důsledky...

# V čem jsou GPU jiná?

## Masivně paralelní procesor

- výpočetní problém musí být dostatečně „velký“
  - násobení matic  $10 \times 10$  vs.  $1000 \times 1000$
  - systém s 50 vs. systém s 100 000 atomy
- výpočetní problém musí jít dostatečně paralelizovat
  - velký systém vs. simulace dlouhého vývoje malého systému

# V čem jsou GPU jiná?

## Lock-step mód

- skupiny vláken by měly následovat stejnou cestu v kódu a přistupovat do spojitých paměťových oblastí
- efektivita GPU akcelerace horší pro „nepravidelné“ výpočty, např.
  - husté vs. řídké matice
  - dlouhý (či žádný) vs. krátký cutoff u ne vazebných sil
  - metoda konečných diferencí vs. metoda konečných prvků
  - graf s vrcholy stejného vs. různého stupně

# V čem jsou GPU jiná?

## Komunikace přes PCI-E

- GPU má velmi rychlou paměť, vstup a výstup je však třeba přenášet po PCI-E sběrnici
- má smysl akcelarovat jen takové algoritmy, které provedou dostatek výpočtů na přenesená data
  - násobení vs. sčítání matic
  - ne vazebné vs. vazebné síly
- problém může být zesílen v případě běhu na více GPU
  - citlivější než v případě více CPU

# V čem jsou GPU jiná?

## Malá paměť GPU

- velikost problému, který lze celý zpracovávat na GPU, je relativně omezená
- v případě, že data odkládáme do CPU paměti, jsme omezeni propustností PCI-E (viz předchozí slide)

# Akcelerace Amber

Valká část funkcionality pmemd je akcelerována (nepodporovány některá nastavení).

Simulovaný systém se musí vejít do GPU paměti

- větší systémy musíme počítat na CPU (v závislosti na konfiguraci simulace a paměti GPU okolo 2M atomů)
- neškáluje s počtem GPU

V METACentru v současné době není k dispozici kompilace pro více GPU.

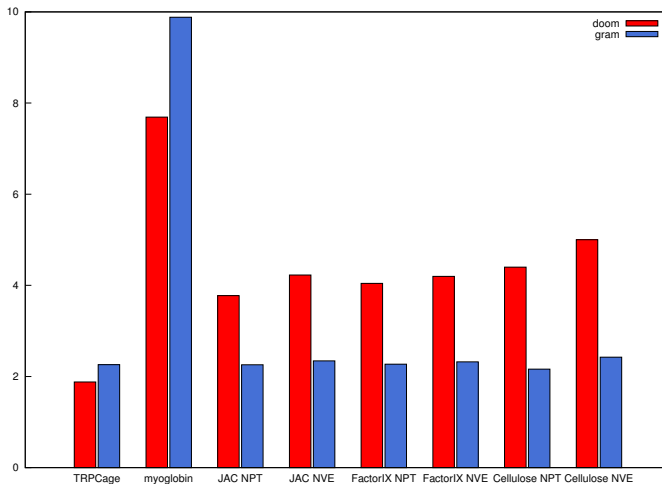
# Měření rychlosti

## Standardní testovací balík Amberu

- implicitní solvent: TRPCage (304 atomů), myoglobin (2 492 atomů)
- explicitní solvent: Cellulose (NPT/NVE, 23 558 atomů), FactorIX (NPT/NVE, 90 906 atomů), JAC (NPT/NVE, 408 609 atomů)



# Zrychlení oproti CPU\*



\*zrychlení měřeno oproti 2x 8-jádrovy Xeon 2.6GHz.

# Co jsme naměřili?

## Pozorování

- u menších molekul dosáhneme vyššího zrychlení na M2090 (zřejmě souvisí s nedostatečnou saturací K20M, implicitní solvent může být lépe optimalizován pro Fermi)
- u explicitního solventu Kepler citelně rychlejší ( $1.67 \times - 1.95 \times$ )
- nemohli jsme změřit škálování na více GPU, ale dle autorů Amberu není ideální (doporučují běh více nezávislých instancí)
- Amber používá téměř výhradně GPU
  - jednotlivé instance využívající rozdílná GPU se nespomalují (nevzniká úzké hrdlo na straně výkonu CPU a propustnosti PCI-E)
  - na clusteru gram můžeme tedy využít čtyřnásobek výkonu, na clusteru doom dvojnásobek (+ stále máme volných většinu CPU jader)

# Akcelerace Gromacs

Valká část funkcionality je akcelerována (je třeba používat Verlet scheme).

Při spouštění je třeba

- alespoň tolik MPI vláken, kolik chceme použít GPU
  - celkový počet vláken pomocí `-nt`, nebo `-ntmpi` a `-ntomp` pro explicitní nastavení MPI a OpenMP vláken na MPI vlákno
  - použité GPU definujeme pomocí `-gpu_id XYZ` (ID GPU pro každé MPI vlákno)
- příliš mnoho OpenMP vláken špatně škáluje (do 4 u AMD, do 8 u Intelu)

# Ladění vytížení CPU a GPU

Akcelerovány jsou ne vazebné síly do cutoffu.

- krátký cutoff vede k nevyužívání plného potenciálu GPU
  - Gromacs automaticky nastaví větší cutoff a zpřesní výpočet
- dlouhý cutoff vede k nevyužití CPU jader
  - můžeme ubrat CPU jádra
  - můžeme použít rychlejší GPU
  - můžeme ubrat cutoff

# Měření rychlosti

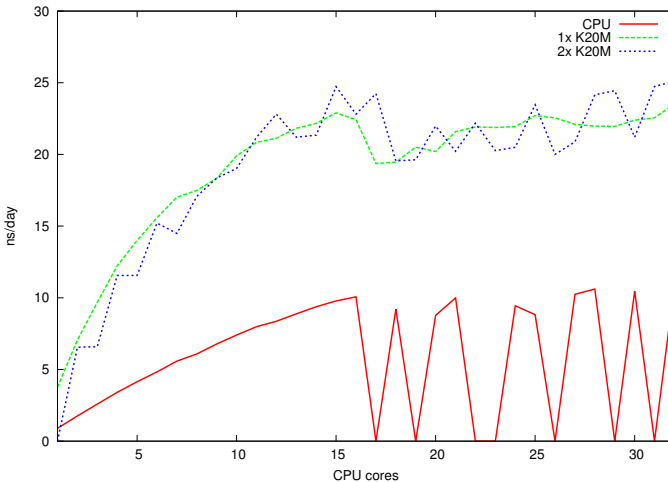
## Alkoholdehydrogenáza

- ve vodě, celkem 97 602 atomů.
- měřen výkon simulace v ns/den

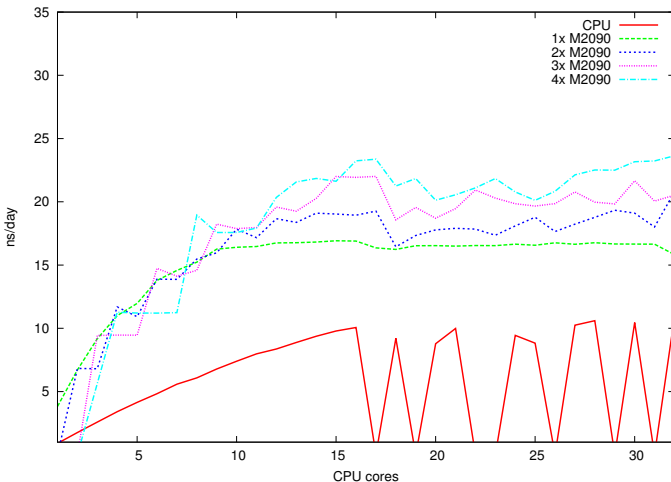
## RNáza ZF-1A

- ve vodě, celkem 16 948 atomů.
- měřen výkon simulace v ns/den

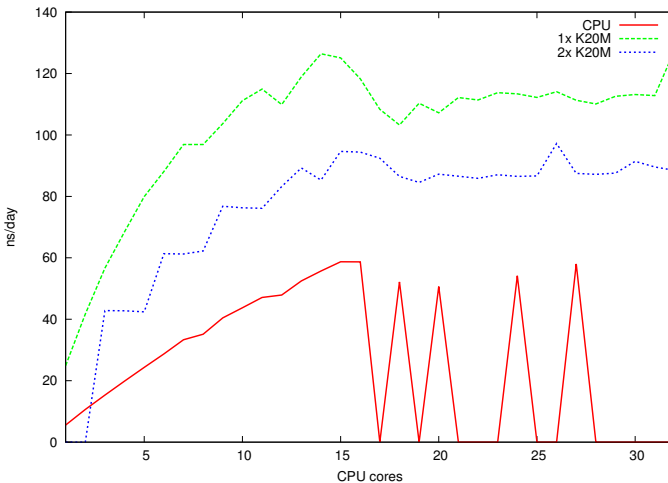
# doom.metacentrum.cz, ADH



# gram.zcu.cz, ADH

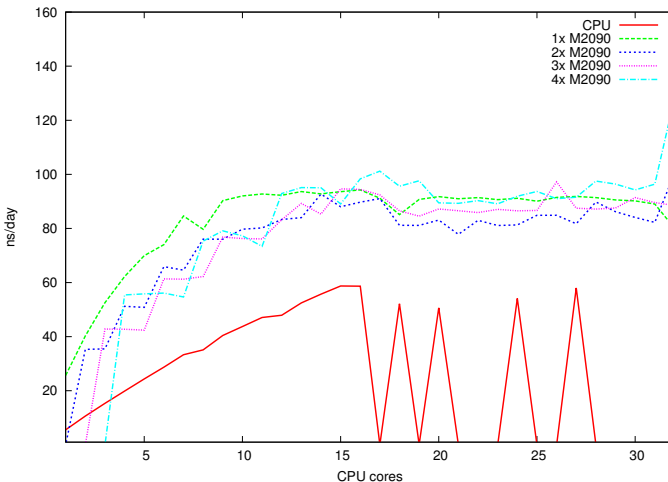


# doom.metacentrum.cz, RNáza





# gram.zcu.cz, RNáza



# Co jsme naměřili?

## Pozorování

- škálování pro více GPU není ideální
  - gram: pomalejší GPU, více zvýší výkon
  - doom: velmi slabé či záporné zvýšení rychlosti
  - u větší instance by mohlo být lepší
- i s použitím 1 GPU dokážeme dobře saturovat dostupná jádra
- Kepler neposkytuje očekávané zvýšení výkonu (1.39× ADH, 1.35 RNáza, běh s jedním GPU)

# Akcelerace NAMD

Valká část funkcionality je akcelerována.

Při spouštění je třeba

- nastavit `outputEnergies` na dostatečně vysokou hodnotu (alespoň 100, lépe více)
- nastavit parametr `+idlepoll` (aktivní doptávání-se GPU na výsledky)
- nakonfigurovat počet GPU a CPU jader (parametry `+p` `+devices`)
  - tím lze poměrně výrazně ovlivnit výkon
- pokud budete benchmarkovat, nechte při změně CPU jader proběhnout jeden krátký běh mimo měření (autotuning)

# Ladění počtu CPU jader k počtu GPU

Obecně je doporučováno 1-2 GPU na 1 CPU socket.

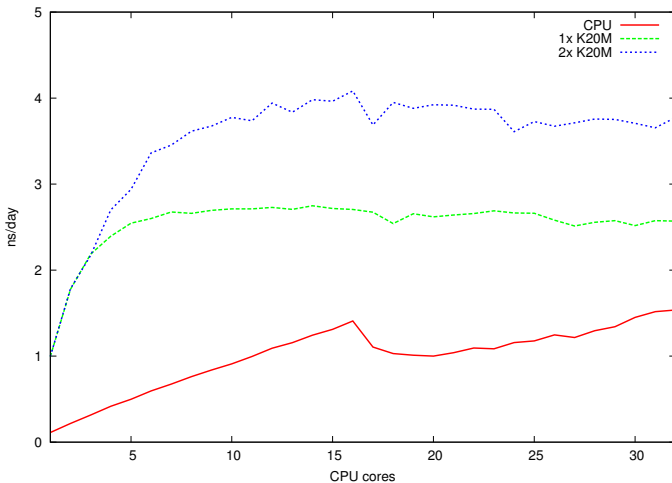
V praxi

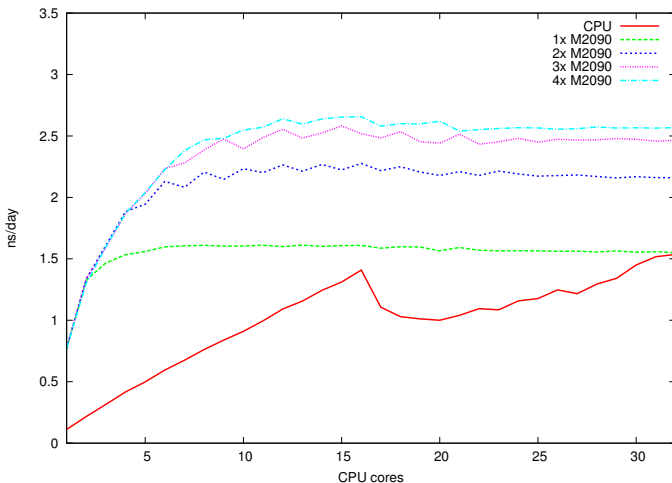
- využívání hyperthreadingu téměř nemá význam
- příliš mnoho CPU jader na GPU může zpomalit výpočet
- škálování pro více GPU není ideální

# Měření rychlosti

## Apolipoprotein A1

- ve vodě, celkem 92 224 atomů.
- outputEnergies 500
- měřen výkon v ns/den





# Co jsme naměřili?

## Pozorování

- škálování pro více GPU není ideální
  - gram:  $1.41 \times 1.57 \times 1.65 \times$
  - doom:  $1.49 \times$
  - u větší instance by mohlo být lepší
- výpočtu stačí relativně málo CPU jader
  - pro výkonnější GPU je třeba více
  - vhodný počet CPU jader neroste úměrně počtu GPU
  - při běhu více nezávislých simulací se vzájemně nespomalují
- Kepler poskytuje citelné zvýšení výkonu ( $1.71 \times$  běh s jedním GPU)



# Co jsme naměřili?

## Doporučení

- není vždy nutné (ani žádoucí) přiřazovat dostupným GPU všechna dostupná CPU jádra
- oddělené instance (pro každé GPU jedna, popř. jedna pro zbytek CPU jader) přinášejí lepší výkon, než jedna multi-GPU

# Matematický software

V METACentru je často používaný software Matlab a Mathematica

- oba umí v omezené míře využívat GPU
  - Mathematica má předdefinované funkce, lze použít uživatelský CUDA kód
  - Matlab má GPU akceleraci v Parallel Computing Toolbox, lze používat další toolboxy, předdefinované i uživatelské funkce
- nestačí jen „spustit GPU verzi“, je nutné GPU aktivně využívat

# Mathematica

## Předdefinované funkce

- práce se seznamy
- filtrování obrazu
- lineární algebra, FFT

```
In[1]:=Needs["CUDALink`"]  
In[2]:=CUDASort[Reverse@Range[10]]  
Out[2]={1,2,3,4,5,6,7,8,9,10}
```

Lze také explicitně zanechat v paměti GPU a tu vyzvednout až jsou data potřeba.

# Matlab

## Předdefinované funkce

- spousta funkcí se použije transparentně
- mnoho skalárních funkcí, lze je aplikovat na elementy polí

Explicitně říkáme, co je v GPU paměti.

```
>> A = rand(1024); gA = gpuArray(A);  
>> tic, C = A * A; toc  
Elapsed time is 0.075396 seconds.  
>> tic, gC = gA * gA; toc  
Elapsed time is 0.008621 seconds.
```

Děkuji za pozornost.