

# Paralelní programování na GPU (PCG 2021)

## Projekt č. 1: CUDA

---

Kristián Kadlubiak (ikadlubiak@fit.vutbr.cz)

### 1 ÚVOD

Cílem tohoto projektu bude implementovat částicový systém na grafické kartě pomocí technologie CUDA. Veškerý kód bude spouštěn na superpočítači Barbora.

### 2 CUDA NA SUPERPOČÍTAČI BARBORA

Pro připojení na superpočítač Barbora je potřeba mít vytvořený účet, se kterým je možné se připojit na tzv. čelní (login) uzel – `barbora.it4i.cz`. Tento uzel **neslouží** ke spouštění náročných úloh, veškeré experimenty je nutné provádět na výpočetních uzlech. Pro účely tohoto projektu je nejjednodušším řešením vytvořit interaktivní úlohu, např. pomocí následujícího příkazu:

```
[jarosjir@login1.barbora ~]$ qsub -A DD-21-22 -q qnvidia -l select=1:ncpus=24,walltime=1:00:00 -I
```

Příkaz `qsub` zadá požadavek na spuštění úlohy do fronty, jakmile bude v systému dostatek volných uzlů, dojde ke spuštění úlohy. Parametr `-A` určuje projekt, v rámci kterého máme alokované výpočetní hodiny (neměnit), `-q` určuje frontu, do které bude úloha zařazena. Jediná fronta, která vám umožní přístup na GPU je `qnvidia`. Parametr `-l` určuje zdroje, které budou úloze přiděleny (počet uzlů, počet procesorů, čas). Abyste předešli zkreslení výkonových statistik, vždy alokujte celý uzel, tj. 24 jader. Interaktivní úlohu pak získáte parametrem `-I`. Více o spouštění úloh na superpočítačích IT4I naleznete na stránce <https://docs.it4i.cz/barbora/job-submission-and-execution/>.

Software na superpočítači Barbora je dostupný pomocí tzv. *modulů*. Tyto moduly je potřeba před použitím načíst, jak pro kompilaci, tak po každém spuštění interaktivní úlohy. V tomto projektu budou potřeba moduly HDF5 a CUDA:

```
module load HDF5/1.12.0-impi-2020a
module load CUDA/11.0.2-GCC-9.3.0
```

### 3 ČÁSTICOVÝ SYSTÉM (20 BODŮ)

Cílem tohoto projektu bude nejprve implementovat a posléze optimalizovat výpočet vzájemného silového působení  $N$  těles. Každé těleso má jistou hmotnost, polohu v prostoru a rychlost. Gravitační síly působící na dané těleso od ostatních těles mají různé směry a jejich výslednice způsobuje změnu rychlosti pohybu tohoto tělesa. Pro vektory polohy  $\mathbf{r}$  a rychlosti  $\mathbf{v}$  platí:

$$\mathbf{r}^{i+1} = \mathbf{r}^i + \mathbf{v}^{i+1} \cdot \Delta t \quad (3.1)$$

$$\mathbf{v}^{i+1} = \mathbf{v}^i + \mathbf{v}_g^{i+1} + \mathbf{v}_c^{i+1} \quad (3.2)$$

kde  $\mathbf{v}_g^{i+1}$  je přírůstek rychlosti vzniklý gravitačním působením těles a  $\mathbf{v}_c^{i+1}$  je změna rychlosti vlivem kolize s některými tělesy.

Síla působící na těleso je dána vektorovým součtem dílčích sil způsobených gravitačním polem ostatních těles. Dvě tělesa na sebe působí gravitační silou danou:

$$F = \frac{G \cdot m_1 \cdot m_2}{r^2}, \quad (3.3)$$

kde  $G = 6.67384 \cdot 10^{-11} \text{Nm}^2\text{kg}^{-2}$  je gravitační konstanta,  $m_1$  a  $m_2$  jsou hmotnosti těles a  $r$  je jejich vzdálenost. Rychlost, kterou těleso obdrží díky této síle pak lze vyjádřit jako:

$$\mathbf{v}_g^{i+1} = \frac{\sum \mathbf{F}_j^{i+1}}{m} \cdot \Delta t \quad (3.4)$$

Pokud se tělesa dostanou do příliš blízké vzdálenosti, dané konstantou `COLLISION_DISTANCE`, dojde k jejich odrazu. Částice si můžete představit jako koule s poloměrem daným polovinou této konstanty. Pro jednoduchost mají všechna tělesa stejný poloměr. Rychlosti dvou těles po odrazu lze určit ze zákona zachování hybnosti a kinetické energie.

$$v_1 \cdot m_1 + v_2 \cdot m_2 = w_1 \cdot m_1 + w_2 \cdot m_2 \quad (3.5)$$

$$\frac{1}{2} \cdot v_1^2 \cdot m_1 + \frac{1}{2} \cdot v_2^2 \cdot m_2 = \frac{1}{2} \cdot w_1^2 \cdot m_1 + \frac{1}{2} \cdot w_2^2 \cdot m_2 \quad (3.6)$$

kde  $m_1$  a  $m_2$  jsou hmotnosti těles,  $v_1$  a  $v_2$  jsou rychlosti těles před kolizí a  $w_1$  a  $w_2$  jsou rychlosti těles po kolizi. Rovnice 3.5 je zákon o zachování hybnosti a rovnice 3.6 je zákon o zachování kinetické energie. Řešením těchto dvou rovnic o dvou neznámých pro  $w_1$  získáváme novou rychlost tělesa. Jelikož v daném kroku mohou na těleso působit i ostatní tělesa, je potřeba získat pouze rozdíl oproti původní rychlosti, který se na původní rychlost aplikuje později.

Změna rychlosti v daném kroku lze pak vyjádřit jako

$$v_c = w_1 - v_1 \quad (3.7)$$

Pro všechny elementy pak platí

$$\mathbf{v}_c^{i+1} = \sum \mathbf{v}_{c_j}^{i+1} \quad (3.8)$$

V každém kroku výpočtu je nutné spočítat změny rychlostí a poloh jednotlivých těles.

#### 3.1 KROK 0: ZÁKLADNÍ IMPLEMENTACE (5 BODŮ)

Kostra aplikace je připravena v adresáři `step0`.

1. Nejprve správně doplňte definici struktur `t_particles` a `t_velocities` v hlavičkovém souboru `nbody.h`. Použijte vhodné datové typy tak, aby se omezil počet přístupů do globální paměti.

Načtení a zápis hodnot zajišťují funkce, které se nacházejí v souboru `h5Helper.cpp`. Je pouze nutné vytvořit objekt typu `MemDesc` (nacházejí v hlavičkovém souboru `h5Helper.h`) který popisuje organizaci paměti.

2. Dalším krokem bude doplnění vyznačených míst v souboru `main.cu` – je třeba doplnit alokaci paměti na CPU a GPU, kopírování načtených dat z CPU do GPU a zpět a spouštění kernelů na GPU. Na GPU je také nutné alokovat strukturu pro uložení mezivýsledků vypočtených rychlostí (jako nápověda poslouží hlavičky kernelů).
3. Následně implementujte samotné kernely `calculate_gravitation_velocity`, `calculate_collision_velocity` a `update_particle` v souboru `nbody.cu` tak, aby kernely správně simulovaly pohyb jedné částice s časovým posuvem `dt` sekund. **Při implementaci se soustřeďte zejména na efektivitu práce s pamětí.**
4. V souboru `Makefile` nastavte počet vláken na blok a tuto proměnnou `thr_blc` společně s počtem částic `N` pak použijte při spouštění kernelu.
5. Program přeložíte příkazem `make` a spustíte pomocí `make run`.

**Správnost výpočtu je možné ověřit porovnáním výstupního souboru se vzorovým výstupem** `sampledata/sampleOutput.h5` **pomocí nástroje `h5diff` (viz. README ve složce `sampledata`, popřípadě `Makefile`), nebo pomocí testů ve složce `tests`.** Odchytky v hodnotách pozic částic v řádech desetinných značí, že je ve výpočtu významná chyba. Řádově menší chyby mohou být způsobeny i mírně odlišným výpočtem, dokonce i přeuspořádáním operací. **Průchod testy je nutnou, ne však postačující podmínkou pro udělení bodů z každého úkolu.**

Po ověření správnosti vyplňte tabulku v souboru `nbody.txt` a odpovězte na dotazy. Tabulka bude obsahovat naměřenou dobu běhu simulace pro různé velikosti dat při konstantním počtu vláken na blok (zvolte 512).

Pro ladění výkonosti použijte profilování, pomocí příkazu `make profile` spustíte profilovací nástroj `nvprof` s předpřipravenými metrikami. Seznam všech dostupných metrik získáte příkazem `nvprof -query-metrics`. Analyzujte přichystané i Vámi přidané metriky a na jejich základě optimalizujte svůj kód.

### 3.2 KROK 1: SLOUČENÍ KERNELŮ (1 BOD)

Zkopírujte celý adresář `step0` do nového adresáře `step1`. Vytvořte nový kernel `calculate_velocity` s vhodným rozhraním, který bude implementovat funkčnost všech předchozích kernelů. Zde na GPU alokujte vše dvakrát, v každém kroku výpočtu pak použijte jednu kopii dat jako vstupy (`p_in`) a druhou jako výstupy (`p_out`). Tím odpadne nutnost synchronizace vláken před zápisem do paměti. V každém dalším kroku pak tyto dvě kopie vždy prohod'te. Funkčnost řešení ověřte srovnáním výstupů simulací a testů!

Pomocí profilování zjistíte rozdíly mezi implementacemi v kroku 1 a 2 a tyto rozdíly popište v souboru `nbody.txt`.

### 3.3 KROK 2: SDÍLENÁ PAMĚŤ (4 BODŮ)

Zkopírujte celý adresář `step1` do nového adresáře `step2`. V tomto kroku využijte sdílené paměti, abyste omezili přístupy do globální paměti. Funkčnost řešení opět ověřte srovnáním výstupů simulací a testů! Porovnejte výkonost s předchozím krokem. Dochází ke zrychlení? Zdůvodněte.

### 3.4 KROK 3: IMPLEMENTACE VÝPOČTU TĚŽIŠTĚ NA GPU I (3 BODY)

Opět zkopírujte celý adresář `step2` do nového adresáře `step3`.

Zde je vaším úkolem doplnit kód pro výpočet těžiště částicového systému na GPU. Jako inspirace Vám může sloužit CPU varianta. Je zřejmé že daný výpočet vede na redukci:

1. Před samotnou redukcí je nutno vynulovat hodnoty těžiště v globální paměti. Jinak by docházelo k akumulaci z výsledků z různých iterací (projevilo by se až v kroku 4) nebo nepřesnému výsledku kvůli nenulovým počátečním hodnotám. Tuto operaci proveďte ze strany CPU vhodným voláním knihovní funkce.
2. Nejdříve implementujte redukci pomocí sdílené paměti v rámci bloku.
3. Následně pak obdržené mezivýsledky zredukujte do globální paměti (ukazatele jsou parametry kernelu). Zde je nutno zajistit vzájemné vyloučení, aby byl výsledek korektní. K tomu slouží ukazatel do globální paměti `lock`.

V této části počítejte těžiště jen z hodnot posledního kroku simulace. **Pro získání plného počtu bodů je nutné implementovat redukci tak, aby i menší tým vláken než je velikost vstupu byl schopen spočítat správný výsledek.** Správnost výpočtu určíte porovnáním s CPU verzí.

### 3.5 KROK 4: PARALELISMUS NA ÚROVNI KERNELŮ A SYNCHRONIZACE (5 BODŮ)

Opět zkopírujte celý adresář step3 do nového adresáře step4. Nejdříve doplňte kód pro zápis do souboru v každé `writeIntensity` iteraci. Funkce pro zápis jsou `writeParticleData` a `writeComData` z modulu `h5Helper`. Funkce `writeComData` vyžaduje explicitně předat hodnoty těžiště (spočteno na GPU) a číslo záznamu. Funkce `writeParticleData` si vystačí s číslem záznamu (interně využívá `MemDesc`). Několik pravidel pro zápis:

- zapisujete v každé  $n$ -té iteraci, kde  $n$  je dělitelné `writeIntensity`, a vždy v iteraci 0.
- V iteraci  $n$  počítejte těžiště a zapisujte hodnoty z  $t[n]$ . Současně počítejte nové hodnoty  $t[n+1]$  (v iteraci 0 počítejte těžiště a zapisujte vstupní hodnoty).
- `writeIntensity` rovno 0 znamená, že se vůbec nezapisuje (zápis jen po skončení simulace) a vyžaduje speciální ošetření.

Následně je vaším úkolem zajistit podmínky, aby správně fungoval (viz Obr. 3.1)

- současný výpočet těžiště a nových poloh částic - překrytí dvou kernelů
- současný výpočet a paměťové operací spojených se zápisem do souboru - překrytí výpočtu a komunikace
- překrytí latence paměťových operací zápisem do souboru - překrytí zápisu do souboru užitečnou prací.

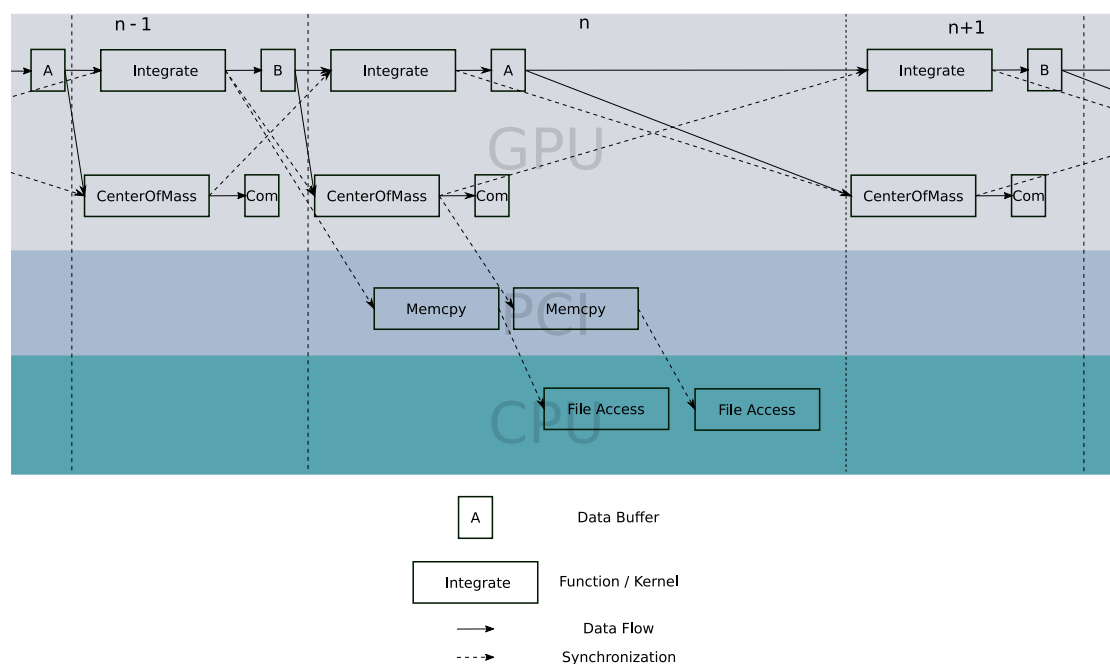


Figure 3.1: Schema výpočtu s vyznačenými souběžnými operacemi a synchronizací.

**Samotné události nemusejí nastat, pro bodový zisk je postačující umožnit jejich vznik. Důležitou součástí tohoto kroku je také ošetření případného souběhu zápisu a čtení.** Za tuhle část je možné získat plný počet bodů i když jste neimplementovali (nebo implementovali chybně) step3, jednoduše použijte kód z step3. Plný počet lze také získat spouštěním prázdných kernelů ve vhodných místech kódu, vše co se týče konkurence a synchronizace pak ale musí být v pořádku.

### 3.5.1 DETAILNĚJŠÍ POPIS SYNCHRONIZACE

- Aby vůbec mohl nastat překryv událostí, je nutné v plné míře využívat neblokuující verze knihovnických funkcí.
- V iteraci  $n$  (tj. iterace, kdy se má provést zápis na disk), se nashoduje výpočet nových hodnot částic pro  $t[n+1]$  z hodnot částic  $t[n]$ . **Je zde nutné počkat na dokončení výpočtu CenterOfMass z iterace  $n-1$  (vstupní buffer pro CenterOfMass v iteraci  $n-1$  je výstupní pro Integrate v iteraci  $n$ ).**
- V iteraci  $n$ , se nashoduje výpočet těžiště v  $t[n]$  z výstupu kernelu Integrate z iterace  $n-1$  (hodnoty částic v  $t[n]$ ). **V tomhle bodě je nutné zaručit korektnost dat čekáním na dokončení Integrate z iterace  $n-1$ .**
- V iteraci  $n$ , je pro zápis na disk potřeba nashodovat kopírování dat částic v  $t[n]$  na CPU. **Zde je nutné počkat na dokončení kernelu Integrate z iterace  $n-1$ .**
- V iteraci  $n$ , je také potřeba nashodovat kopírování data těžiště v  $t[n]$  na CPU. **Pozor, je nutné počkat na dokončení kernelu CenterOfMass z iterace  $n$ .**
- **Zápis hodnot částic na disk probíhá současně s kopírováním hodnot těžiště na CPU! Zápis musí počkat na dokončení kopírování hodnot částic na CPU.**
- Po dokončení kopírování hodnot těžiště nastává zápis těžiště na disk a simulace pokračuje iteraci  $n+1$ .
- **Výpočet CenterOfMass probíhá v každé iteraci.**
- Aby bylo možné souběžně vykonávat oba kernely, každý musí využívat vlastní stream. **Pozor, nepoužívejte výchozí stream je implicitně synchronizován!**
- Datové kopie využívají vlastní stream aby neblokovali výpočet. Zde postačí jeden, **opět ne výchozí**, stream.

### 3.6 KROK 5: ANALÝZA VÝKONU (2 + 1 BODY)

Pomocí programu gen generujte datové soubory různých velikostí (volte mocniny dvou). Např. pro vygenerování souboru s 4096 částicemi použijte následující příkaz:

```
./gen 4096 4096.h5
```

Pro každý počet částic stanovte ideální počet vláken na blok a zapište výsledný čas, dosaženou propustnost globální paměti a dosažený výkon do souboru nbody.txt. Naměřené časy porovnejte se sekvenční implementací CPU verze a spočítejte zrychlení. Od jakého počtu částic se vyplatí použít grafickou kartu (uvažujte, že paralelní verze na CPU bude cca 10× rychlejší než sekvenční verze)? Za tuto část je možné získat bonusový bod. Naměřte vaši aplikace také na superpočítači Karolína a srovnajte výkon grafických karet.

## 4 VÝSTUP PROJEKTU A BODOVÁNÍ

Výstupem projektu bude soubor xlogin00.zip obsahující všechny zdrojové soubory a textový soubor nbody.txt obsahující textový komentář k projektu. V každém souboru nezapomeňte uvést svůj login! Hodnotit se bude jak funkčnost a správnost implementace, tak textový komentář – ten by měl dostatečně popisovat rozdíly mezi jednotlivými kroky a odpovídat na otázky uvedené v zadání. Při řešení se soustřed'te především na správnost použití CUDA, přesnost výpočtu je závislá na mnoha okolnostech, např. zvoleném výpočtu, pořadí operací apod., a pokud bude v rozumných mezích, nebude hrát velkou roli při hodnocení. Projekt odevzdejte v uvedeném termínu do informačního systému.