Paralelní programování na GPU (PCG 2021)
Projekt č. 2: OpenACC

Jiří Jaroš (jarosjir@fit.vutbr.cz)

Termín odevzdání: 17. 12. 2021

1 Úvod

V prvním projektu jste si mohli vyzkoušet akceleraci částicového systému pomocí Nvidia CUDA. Cílem tohoto projektu bude tatáž úloha, ale implementace bude probíhat pomocí knihovny OpenACC[1] [2] [3] a kompilátoryu NVHPC[4].
Vypracování i opravování bude probíhat na Barboře.

2 OpenACC na superpočítači Anselm

Pro připojení na superpočítač Barbora postupujte dle návodu v IS. Pouze při vytváření úlohy zvolte frontu `qnvidia`:

```
[jarosjir@login1.barbora ~]$ qsub -A DD-21-22 -q qnvidia -l select=1:ncpus=24,walltime=1:00:00 -I -X
qsub: waiting for job 1307408.dm2 to start
qsub: job 1307408.dm2 ready

[jarosjir@cn197.barbora ~]$
```

Následně je nutné načíst moduly v tomto pořadí:

```
ml Python/3.8.6-GCCcore-9.3.0 NVHPC HDF5/1.10.6-gompi-2020a
```

[1] https://www.openacc.org/
[2] https://www.openacc.org/sites/default/files/inline-files/OpenACC_Programming_Guide_0.pdf
[3] https://www.openacc.org/sites/default/files/inline-files/OpenACC.2.7.pdf
[4] https://developer.nvidia.com/hpc-sdk

3 Částicový systém pomocí OpenACC(10 bodů)

Cílem tohoto projektu bude nejprve implementovat a posléze optimalizovat výpočet vzájemného silového působení N těles pomocí knihovny OpenACC. 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 tohoto tělesa. Pro vektory polohy $\mathbf{r}$ a rychlosti $\mathbf{v}$ platí:

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

$$\mathbf{v}^{i+1} = \mathbf{v}^{i} + \mathbf{v_g}^{i+1} + \mathbf{v_c}^{i+1} \tag{3.2}$$

kde $\mathbf{v_g}^{i+1}$ je přírustek 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:

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

kde $G = 6.67384 \cdot 10^{-11}\mathrm{Nm^2kg^{-2}}$ 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:

$$\mathbf{v_g}^{i+1} = \frac{\sum \mathbf{F}_j^{i+1}}{m} \cdot \Delta t \tag{3.4}$$

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 všechny tělesa stejný poloměr. Rychlosti dvou těles po odrazu lze určit ze dvou zákonů.

$$v_1 \cdot m_1 + v_2 \cdot m_2 = w_1 \cdot m_1 + w_2 \cdot m_2 \tag{3.5}$$

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

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

$$v_c = w_1 - v_1 \tag{3.7}$$

Pro všechny elementy pak platí

$$\mathbf{v_c}^{i+1} = \sum {v_c}_j^{i+1} \tag{3.8}$$

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

3.1 Krok 1: Implementace vzájemného působení těles (2 body)

Kostra aplikace je připravena v adresáři `step1` v souborech `main.cpp, nbody.cpp, nbody.h`. Zde máte vyznačeno, které části je nutné doplnit. V `Makefile` by nemělo být třeba nic měnit (můžete si zvýšit úroveň výpisů pomocí `-Minfo`, nebo přepnout na generování kódu pro CPU pomocí proměnné `ACC`, kde lze také měnit i další parametry akcelerace).
Při implementaci vycházejte ze základní procesorové verze, případně z naivní CUDA verze vytvořené v projektu 1. V tuto chvíli můžete použít pole struktur (AoS) pro jednodušší implementaci.
Zaměřte se na následující body (bude hodnoceno):

- Minimalizaci transferů mezi CPU a GPU mezi iteracemi či voláním kernelů. Doporučuji rozšířit strukturu Particles a Velocities tak, aby měla metody pro alokaci a dealokaci paměti, a pro transfery na/z GPU (viz cvičení 6).
- Zamyslete se, jak budete ukládat v paměti pozice, hmotnosti a rychlost. Pozor na to, že OpenACC nemá datové typy `float2, float3, float4`. Jejich tvorba v C++ je však triviální (`struct float4 {float x, y, z, w;}`)
- Navrhněte správně strukturu OpenACC smyček připravených metodách `calculate_gravitation_velocity`, `calculate_collision_velocity` a `update_particle`.
- Pokuste se kód optimalizovat (např. collapse, tile, gang, vector, ...)

Správnost výpočtu je možné ověřit porovnáním výstupního souboru se vzorovým výstupem `sampledata/sampleOutputA.h5`**, nebo pomoci testů ve složce** `tests`**.** Odchylky v řádech desetin 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 běhů simulace pro různé velikosti dat.
Pro ladění výkonnosti použijte profilování, pomocí příkazu `make profile` spusťte 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 2: Implementace vzájemného působení těles (2 body)

Zkopírujte celý adresář `step1` do nového adresáře `step2`. Vytvořte novou funkci `calculate_velocity` s vhodným rozhraním, který bude implementovat funkčnost všech předchozích kerelů. 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`). V každém dalším kroku pak tyto dvě kopie vždy prohoďte. Tím bude možné některé výpočty zjednodušit a dále optimalizovat.

Pomocí profilování zjistěte rozdíly mezi implementacemi v kroku 1 a 2 a tyto rozdíly popište v souboru nbody.txt. Popřemýšlejte jestli je výhodnější použít Pole struktur (AoS) nebo strukturu polí (AoS)
Cílem tohoto kroku je dostat co nejvyšší výkon, při udržení rozumně elegantního kódu (OpenACC je primárně o snadném návrhu, čitelnosti, udržovatelnosti...). Tedy, pokud potřebuji 5 pomocných proměnných, abyste různě otáčeli data v rámci jednoho kernelu, tak to asi není to pravé ořechové. Projekty opravuji ručně, proto mi komentáře mohou pomoci k správnému pochopení kódu.

3.3 Krok 3: Výpočet těžiště (2 body)

Opět zkopírujte celý adresář `step2` do nového adresáře `step3`. V tomto kroku je vašim úkolem doplnit kód pro výpočet těžiště částicového systému na GPU. Jako inspirace Vám může sloužit CPU varianta. V CUDA variantě jsme sázeli na redukci ve sdílené paměti a použití zámků. To v OpenACC nebude tak jednoduché, protože OpenACC je navrženo jako lockfree jazyk. Jaké tedy máte možnosti, pokud nechceme dělat výpočet sekvenčně:

- bariera - barieru lze udělat pouze mezi kernely - tedy není možné provést uvnitř `acc parallel`.
- zámky - nejsou v jazyce podporovány, ale můžete si je implementovat pomocí atomických operací s dovětkem `capture`.
- kritická sekce - neexituje, tedy update více položek atomicky nelze provést. Jedině vlastní zámek.
- redukce - funguje pouze nad základními datovými typy, ne nad float4.

Inspiraci můžete nalézt v paralelní redukci v CUDA, jen si představte, že nemáte sdílenou paměť ale pouze tu globální. Pokud to bude nutné, máte povoleno alokovat další paměť na GPU. Dělejte to ale pouze 1x v celém kódu, ne při každé iteraci.
Další zajímavá vlastnost je, že pokud vám běží jen jeden worker a ten má 32 vláken (jeden warp), pak tato vlákna běží synchronně (ale pozor, stejně musíte mít atomic do paměti... operace se ale nepromíchají (nemění pořadí) jako mezi workery nebo gangy).
Pokud se vám povede navrhnout více verzí, a nevíte která je nejlepší, klidně je nechtě v řešení, ale dejte jim nějaké jiné jméno např. `centerOfMassGPU_ver1`. Při hodnocení správnosti se budu dívat na tu první, ale mohu vám dát nějaký další bod za zajímavý návrh.
Pozor: Výpočet těžiště musí fungovat pro libovolný počet částic, nejenom mocninu dvou.

3.4 Krok 4: Překrytí výpočtu pozic, těžiště a ukládání dat na disk (2 body)

Opět zkopírujte celý adresář `step3` do nového adresáře `step4`. Nejdříve doplňte kód pro zápis do souboru v každé `writeIntesity` 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:

- zapisujte v každé n-té iteraci, kde n je rovno `writeIntesity`, a vždy v iteraci 0.
- V iteraci n počítejte těžiště a zapisujte hodnoty z kroku n. Současně počítejte nové hodnoty kroku$n+1$ (v iteraci 0 počítejte těžiště a zapisujte vstupy hodnoty).
- `writeIntesity` rovno 0 znamená že se vůbec nezapisuje (zápis jen po skončení simulace) a vyžaduje speciální ošetřeni.

Pokud se vám nepovede udělat vše, implementuje alespoň část (např. výpočet (polohy + těžiště) - transfer). Nápovědu hledejte v OpenACC 2.7 API, kapitola 2.16[5]

3.5 Krok 5: analýza výkonu (2 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
```

Naměřené časy porovnejte se paralelní implementací CPU (staří přeložit s `-ta:multicore`) verze a spočtěte zrychlení. Pokud máte naměřeny hodnoty i pro CUDA verzi, připište i tyto hodnoty. Od jakého počtu částic se vyplatí použít grafickou kartu?

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 a jméno (přepište ten můj)! 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 soustřeďte především na správnost použití OpenACC, 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. Projekty hodnotím ručně :) Projekt odevzdejte v uvedeném termínu do informačního systému.

[5] `https://www.openacc.org/sites/default/files/inline-files/OpenACC.2.7.pdf`