FAKULTA INFORMAČNÍCH TECHNOLOGIÍ, VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Architektura procesorů (ACH 2018)
Projekt č. 2: CUDA

Kristian Kadlubiak (ikadlubiak@fit.vutbr.cz)

Termín odevzdání: 16. 12. 2018

1 ÚVOD

V prvním projektu jste si mohli vyzkoušet optimalizaci sekvenčního kódu pomocí vektorizace na problému částicového systému. Cílem tohoto projektu bude implementovat částicový systém na grafické kartě pomocí technologie CUDA. Veškerý kód bude spouštěn na superpočítači Anselm.

2 CUDA NA SUPERPOČÍTAČI ANSELM

Pro připojení na superpočítač Anselm postupujte dle návodu v prvním projektu. Pouze při vytváření úlohy zvolte frontu `qnvidia`:

```
[ikadlubiak@login1.anselm ~]$ qsub -A DD-18-32 -q qnvidia -l select=1:ncpus=16,walltime=1:00:00 -I
qsub: waiting for job 1307408.dm2 to start
qsub: job 1307408.dm2 ready

[ikadlubiak@cn182.anselm ~]$
```

Následně je nutné načíst moduly `intel` a `cuda`:

```
module load intel/2016a
module load CUDA/8.0.44
```

3 Částicový systém (15 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 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 $G = 6.67384 \cdot 10^{-11}\mathrm{Nm}^2\mathrm{kg}^{-2}$ 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:

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

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 0: ZÁKLADNÍ IMPLEMENTACE (5 BODŮ)

Kostra aplikace je připravena v adresáři `step0`. 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. Doplňte také funkce `particles_read` a `particles_write` pro načítání a ukládání dat.
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 kernelu na GPU. Na GPU je taky nutné alokovat strukturu pro uložení mezivýsledků vypočtených rychlostí (ako nápověda poslouží hlavičky kernelů).
Následně implementujte samotné kernely `calculate_gravitation_velocity`, `calculate_colision_velocity` a `update_particle` v souboru `nbody.cu` tak, aby kernely správně simulovali pohyb jedné částice s časovým posuvem `dt` sekund. V souboru `Makefile` nastavte počet vláken na blok a tuto proměnnou thr_blc společně s počtem částic `N` pak použijte při spouštění kernelu. Program přeložíte příkazem `make` a spustíte pomocí `make run`.
Správnost výpočtu je možné ověřit porovnáním výstupního souboru se vzorovým výstupem `output.dat`. 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. Je také velmi vhodné použít pro testování upravenou sadu testů z prvního projektu!
Po ověření správnosti výpočtu najděte ideální nastavení `THREADS_PER_BLOCK` tak, aby byl výpočet co nejrychlejší. Do souboru `nbody.txt` zapište výsledné časy.
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 1: OPTIMALIZACE KÓDU (3 BODY)

Zkopírujte celý adresář `step0` do nového adresáře `step1`. Vztvořte nový kernel `calculate_velocity` s vhodným roshraním, 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`). Tím odpadne nutnost synchronizace vláken před zápisem do paměti. V každém dalším kroku pak tyto dvě kopie vždy prohoďte. Dále se pokuste optimalizovat výrazy a upravit kód tak, aby kernel využíval co nejméne registrů. Funkčnost řešení ověřte srovnáním výstupů simulací a testů!
Pomocí profilování zjistěte rozdíly mezi implementacemi v kroku 1 a 2 a tyto rozdíly popište v souboru `nbody.txt`.

3.3 KROK 2: SDÍLENÁ PAMĚŤ (5 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 výkonnost s předchozím krokem. Dochází ke zrychlení? Zdůvodněte.

3.4 KROK 3: 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.dat
```

Pro každý počet částic stanovte ideální počet vláken na blok a zapište výsledný čas, dosaženou propustnost globalní paměti a dosažený výkon do souboru `nbody.txt`. Naměřené časy porovnejte se sekvenční implementací z prvního projektu a spočtěte zrychlení. Od jakého počtu částic se vyplatí použít grafickou kartu (uvažujte, že paralelní verze na CPU bude cca 10× rychlejší než sekvenční verze)?

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 soustřeďte 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.