Paralelní programování na GPU (PCG 2024)
Projekt č. 2: OpenACC

David Bayer (ibayer@fit.vutbr.cz)

1 Úvod

Cílem tohoto projektu je implementovat simulaci částicového systému na grafické kartě pomocí technologie OpenACC. Projekt se skládá z několika částí, které na sebe navazují a postupně rozšiřují základní implementaci. V každé části je potřeba implementovat funkcionalitu vyznačenou v kódu pomocí komentáře `TODO`. Ke krokům se rovněž pojí otázky v souboru *nbody.md*, které je třeba zodpovědět.

2 Práce na superpočítači Karolina

Pro připojení na superpočítač Karolina je potřeba mít vytvořený účet, se kterým je možné se připojit na tzv. čelní (login) uzel – `karolina.it4i.cz`. Tento uzel **neslouží** ke spouštění náročných úloh, veškeré experimenty je nutné provádět na výpočetních uzlech. Software je na superpočítači dostupný pomocí modulů. Pro načtení modulů potřebných pro tento projekt slouží skript *loadModules.sh*, který je třeba spustit v shellu po každém přihlášení jako

```
[user@login1.karolina PCG-proj1]$ . loadModules.sh
```

Pro spuštění úlohy je nejjednodušší vytvořit interaktivní úlohu, např. pomocí následujícího příkazu:

```
[user@login1.karolina PCG-proj1]$ salloc -A DD-24-108 -p qgpu_exp -G 1 -t 01:00:00 -I
salloc: Granted job allocation 55125
salloc: Waiting for resource configuration
salloc: Nodes acn01 are ready for job
[user@acn01.karolina PCG-proj1]$
```

Příkaz `salloc` přidá požadavek na spuštění úlohy do fronty. Jakmile bude v systému dostatek volných uzlů, dojde k jejímu spuštění. Parametr `-A` určuje projekt, v rámci kterého máme alokované výpočetní hodiny (neměnit) a `-p` určuje frontu, do které bude úloha zařazena.

Fronta `qgpu_exp` má vysokou prioritu, úloha je ale omezena na maximálně 1 hodinu. Pro delší úlohy je třeba zvolit frontu `qgpu`. Jiné fronty neumožňují přístup na akcelerované GPU uzly. Parametr `-G` určuje počet GPU alokace, ponechte pouze 1, více v projektu nepoužijeme (na Barboře není povolena částečná alokace uzlu, parametr `G` tam tedy nepoužívejte). Parametr `-t` specifikuje délku alokace a parametr `-I`, že se jedná o interaktivní úlohu. Více o spouštění úloh na superpočítačích IT4I naleznete na stránce `https://docs.it4i.cz/general/job-submission-and-execution/`.

3 Překlad projektu

Pro překlad programu na osobním počítači je třeba pracovat na Linuxu a mít nainstalovány programy CMake, NVHPC Toolkit, knihovnu HDF5 a Python 3. NVHPC Toolkit lze stáhnout a nainstalovat z `https://developer.nvidia.com/hpc-sdk-downloads`.

3.1 Na superpočítači

Na superpočítačích Barbora i Karolina jsou všechny potřebné prerekvizity již nainstalovány, pouze je nutné nahrát jejich moduly, jak bylo popsáno v přechozí kapitole. Pro sestavení projektu postupujte stejně jako u linuxové varianty (viz další podkapitola), vynechejte však instalaci závislostí.

3.2 U vás na PC

Pokud vlastníte PC s NVIDIA GPU, můžete celý projekt vypracovat a ozkoušet u sebe na PC. V opačném případě máte i tak možnost pracovat u sebe na PC, nicméně přijdete o možnost projekt spustit.

3.2.1 Linux

Po instalaci CUDA Toolkitu a NVHPC Toolkitu je na linuxových systémech (včetně WSL) všechny potřebné závislosti nainstalovat pomocí příkazu:

```
$ sudo apt install -y gcc g++ cmake libhdf5-serial-dev hdf5-tools python3-dev python3-pip
  python3-venv
```

Poté je potřeba vytvořit složku `build` pro překlad a vygenerovat `Makefile` script pomocí programu CMake. Pro ověřování správnosti programu použijte konfiguraci `Debug`, pro měření výsledků pak konfiguraci `Release`. Proměnná `ACC_TARGET` určuje, pro jaký typ akcelerátoru je program překládán (*multicore* pro CPU, *gpu* pro GPU).

```
$ mkdir build
$ cmake -DCMAKE_CXX_COMPILER=nvc++ -DCMAKE_BUILD_TYPE=[Debug|Release] -S. -Bbuild -DACC_TARGET=[multicore|gpu]
```

Testování správnosti implementace lze ověřit pomocí skriptu *runTests.sh* s cestou k testovanému binárnímu souboru jako

```
$ ./runTests.sh build/nbody[Cpu|0-3]
```

4 Částicový systém (10 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:

$$\mathbf{r}^{i+1} = \mathbf{r}^{i} + \mathbf{v}^{i+1} \cdot \Delta t \tag{4.1}$$

$$\mathbf{v}^{i+1} = \mathbf{v}^{i} + \mathbf{v_g}^{i+1} + \mathbf{v_c}^{i+1} \tag{4.2}$$

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:

$$F = \frac{G \cdot m_1 \cdot m_2}{r^2}, \tag{4.3}$$

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:

$$\mathbf{v_g}^{i+1} = \frac{\sum \mathbf{F}_j^{i+1}}{m} \cdot \Delta t \tag{4.4}$$

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 zákonu zachování hybnosti a kinetické energie.

$$v_1 \cdot m_1 + v_2 \cdot m_2 = w_1 \cdot m_1 + w_2 \cdot m_2 \tag{4.5}$$

$$\frac{1}{2} \cdot v_1^2 \cdot m_1 + \frac{1}{2} \cdot v_2^2 \cdot m_2 = \frac{1}{2} \cdot w_1^2 \cdot m_1 + \frac{1}{2} \cdot w_2^2 \cdot m_2 \tag{4.6}$$

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 4.5 je zákon o zachování hybnosti a rovnice 4.6 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

$$v_c = w_1 - v_1 \tag{4.7}$$

Pro všechny elementy pak platí

$$\mathbf{v_c}^{i+1} = \sum {v_c}_j^{i+1} \tag{4.8}$$

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

5 ÚKOLY

Tato kapitola se věnuje implementaci, testování a vyhodnocení projektu. Obsahuje celkem 4 krokz, za které lze získat až 10 bodů. Důležité části implementace dokumentujte komentáři. K téměř všem krokům se také pojí otázky v souboru *nbody.md*, které je třeba zodpovědět. Při kontrole průchodu testy **Veškerá měření výsledků provádějte na superpočítači Karolina**.

5.1 KROK 0: ZÁKLADNÍ IMPLEMENTACE (2 BODY)

Kostra aplikace je připravena v adresáři `Step0`.

1. Nejprve správně doplňte definici struktur `Particles` a `Velocities` v hlavičkovém souboru *nbody.h*. Použijte vhodné datové typy tak, aby se omezil počet přístupů do globální paměti. Můžete využít datových struktur `float3` a `float4` definovaných v hlavičkovém souboru *Vec.h*. Můžete použít koncept *structure of arrays* nebo *array of structures*.

2. Pak implementujte jejich kostruktory, destruktory a metody pro kopírování mezi CPU a GPU v souboru *nbody.cpp*.

3. Poté upravte konstrukci deskriptoru paměti `MemDesc` tak, aby byla správně načtena data ze vstupního souboru. Mějte na paměti, že rozestupy a offsety dat jsou uváděny v počtech floatů, nikoli v bajtech.

4. Doplňte volání funkcí `calculateGravitationVelocity`, `calculateCollisionVelocity` a `updateParticle` v hlavní smyčce programu.

5. Nakonec implementujte samotené kernely v souboru *nbody.cpp* podle referenční CPU verze tak, aby by správně simulován pohyb částic s časovým posuvem `dt` sekund. K implementaci můžete využít přetížené operátory pro struktury `float3` a `float4`. Pokuste se kód optimalizovat pomocí OpenACC direktiv (např. collapse, tile, gang, vector, ...).

Správnost implementace ověřte pomocí testů skriptem *runTests* a porovnáním referenčního výstupu pomocí skriptu *checkOutput*. Odchylky v řádech desetin signalizují, že je ve výpočtu významná chyba. **Průchod testy je nutnou, ne však postačující podmínkou pro udělení bodů z každého úkolu**.
Po ověření správnosti vyplňte tabulku v souboru `nbody.md` a odpovězte na dotazy. Pro naměření časů simulace použijte skript *runProgressBenchmark*.

5.2 KROK 1: SLOUČENÍ KERNELŮ (2 BODY)

V kroku 1 je vaším úkolem sloučit všechny kernely do jednoho kernelu `calculateVelocity`. Zrychlí se tak výpočet simulace, sníží se počet přístupů do globální paměti a nadále nebude třeba používat dočasné úložiště pro rychlost. Aby nebylo nutné synchronizovat vlákna před zápisem do paměti, je struktura `Particles` dvakrát. V každém kroku výpočtu používejte jednu kopii jako vstup a druhou jako výstup. Pro identifikaci aktuálního indexu vstupní a výstupní kopie použijte proměnné `srcIdx` a `dstIdx` definované ve výpočetní smyčce. Po

skončení simulace budou aktuální data uložena v kopii na indexu `resIdx`. Nezapomeňte před začátkem simulace inicializovat daty obě kopie.
Po ověření správnosti vyplňte tabulku v souboru `nbody.md` a odpovězte na dotazy. Pro naměření časů simulace použijte skript *runProgressBenchmark*.

5.3 Krok 2: Implementace výpočtu těžiště na GPU (2 body)

V kroku 2 implementujte výpočet polohy a hmotnosti težiště v kernelu `centerOfMass`, který spusťte po dokončení simulace.

1. Nejprve je třeba v souboru *main.cpp* implementovat alokaci a dealokaci bufferu proměnné `comBuffer`. Ten bude sloužit jako úložiště lokálních pozic a hmotností těžišť. Velikost bufferu bude záviset na vaší konkrétní implementaci. Zajistěte, aby se paměť alokovala také na GPU a byla vynulována před zavoláním výpočtu.

2. Pak se můžete pustit do implementace výpočtu těžiště. OpenACC nedisponuje zámky, ani uživatelsky definovaných redukcemi. Synchronizace je možná pouze pomocí bariér mezi voláními kernelů. Výpočet je tedy implementovat iterativně. Inspiraci můžete nalézt v CUDA implementaci z 1. projektu, jenom si představte, že nemáte sdílenou paměť a máte jenom globální.

Pro zisk 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.

5.4 Krok 3: Překrytí výpočtu pozic, těžiště a ukládání dat na disk (2 bodů)

Při počítání simulací nás často nezajímá pouze její konečný stav, chceme znát i její průběh. Cílem tohoto kroku je tedy umožňit průběžné ukládání mezivýsledků simulace tak, abychom co nejlépe vykryli přenos dat užitečnou prací. K tomu bude potřeba využít vícestreamové zpracování a synchronizaci.
V každém kroku je z polohy částic v čase t (na indexu `srcIdx`) počítána nová poloha částic v čase $t+1$ (index `dstIdx`). Lambda funkce `shouldWrite` na základě kroku simulace a frekvence zápisu `writeFreq` určuje, zda se v dáném kroku má zapsat aktuální poloha částic (index `srcIdx`) do výstupního souboru. Je tedy nutné data překopírovat z GPU zpět na CPU a provést zápis pomocí metody `writeParticleData` objektu `h5Helper`. Číslo záznamu získejte pomocí lambda funkce `getRecordNum`. Zároveň s tím je také potřeba spočítat na GPU aktuální těžiště, překopírovat jej na CPU a uložit do výstupního souboru pomocí metody `writeComData` objektu `h5Helper`. Nezapomeňte před každým výpočtem těžiště vynulovat jeho hodnotu v globální paměti GPU.
Současně tedy bude ve vybraných krocích probíhat výpočet nové polohy částic, kopírování dat z GPU na CPU a výpočet těžiště. Pro každou z těchto úloh vytvořte vlastní stream. Po dokončení hlavní smyčky opět spočítejte finální těžiště stejně jako v kroku 3. Paměť pro výpočet těžiště alokujte pouze jednou, nikoliv při každém volání kernelu.

Ověření správnosti proveďte tak, že vygenerujete vstupní data programem `gen` libovolné velikosti, spustíte simulaci na CPU, poté na GPU a nakonec porovnejte jejich výstupy skriptem `compare`. Příklad pro data simulaci o 4096 částicích:

```
$ ./gen 4096 input4096.h5
$ ./nbodyCpu 4096 0.01f 100 5 input.h5 outputCpu.h5
$ ./nbody4 4096 0.01f 100 5 input.h5 outputGpu.h5
$ ../compare.sh outputCpu.h5 outputGpu.h5
```

V případě potřeby modifikujte signatury kernelů a rozhraní třídy `Particles`. Za tuhle část je možné získat plný počet bodů, i když jste neimplementovali (nebo implementovali chybně) krok 3, jednoduše použijte kód z kroku 2 a doplňte synchronizaci ve funkci `main.cpp` dle zadání. Pokud se vám nepodaří udělat vše, implementujte alespoň část.

5.5 Krok 4: Analýza výkonu (2 body)

Nad vyhotoveným binárním souborem z kroku 4 spusťte skript *runFinalBenchmark* a výslekdy zapište do tabulky v souboru *nbody.md*. Spočtěte dosažené zrychlení oproti naměřené referenční CPU implementaci, propustnost paměti a celkový výkon. Dále odpovězte na otázky v dokumentu. Referenční časy simulace na CPU byly naměřeny na CPU uzlech Karoliny (celkem 128 jader).

6 Výstup projektu a bodování

Výstupem projektu bude soubor `xlogin00.zip` obsahující celý původní vámi modifikovaný archiv bez binárních a vstup/výstupních souborů simulací. V každém souboru nezapomeňte změnit 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 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. Projekt odevzdejte v uvedeném termínu do informačního systému.