FAKULTA INFORMAČNÍCH TECHNOLOGIÍ, VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Architektura výpočetních systémů (AVS 2019)
Počítačové cvičení č. 3: Optimalizace n-body simulace

Gabriel Bordovský (ibordovsky@fit.vutbr.cz)
Filip Kuklis (ikuklis@fit.vutbr.cz)
Kristian Kadlubiak (ikadlubiak@fit.vutbr.cz)

Termín: 28. 10. 2019

1 ÚVOD

Cílem tohoto cvičení je optimalizace sekvenčního kódu zejména pomocí vektorizace. Vaším úkolem bude optimalizovat úlohu částicového systému (n-body). Pro analýzu výkonnosti je připravena knihovna PAPI umožňující přístup k zabudovaným hardwarovým *performance counterům* uvnitř procesoru.

2 ČÁSTICOVÝ SYSTÉM

Cílem cvičení je optimalizace vzájemného gravitačního 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:

$$\mathbf{r}^{i+1} = \mathbf{r}^{i} + \mathbf{v}^{i+1} \cdot \Delta t \tag{2.1}$$

$$\mathbf{v}^{i+1} = \mathbf{v}^{i} + \mathbf{v_g}^{i+1} + \mathbf{v_c}^{i+1} \tag{2.2}$$

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:

$$\mathbf{F} = \frac{G \cdot m_1 \cdot m_2}{\mathbf{r}^2}, \tag{2.3}$$

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 $\mathbf{r}$ je jejich vzdálenost. Rychlost, kterou těleso obdrží v daném kroku díky okolním tělesům je pak součet všech sil, které na těleso působí podělená jeho hmotnosti a vynásobena délkou kroku simulace:

$$\mathbf{v_g}^{i+1} = \frac{\sum \mathbf{F}_j^{i+1}}{m} \cdot \Delta t \tag{2.4}$$

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.

$$\mathbf{v}_1 \cdot m_1 + \mathbf{v}_2 \cdot m_2 = \mathbf{w}_1 \cdot m_1 + \mathbf{w}_2 \cdot m_2 \tag{2.5}$$

$$\frac{1}{2} \cdot \mathbf{v}_1^2 \cdot m_1 + \frac{1}{2} \cdot \mathbf{v}_2^2 \cdot m_2 = \frac{1}{2} \cdot \mathbf{w}_1^2 \cdot m_1 + \frac{1}{2} \cdot \mathbf{w}_2^2 \cdot m_2 \tag{2.6}$$

kde m_1 a m_2 jsou hmotnosti těles, $\mathbf{v}_1$ a $\mathbf{v}_2$ jsou rychlosti těles před kolizí a $\mathbf{w}_1$ a $\mathbf{w}_2$ jsou rychlosti těles po kolizi. Rovnice 2.5 je zákon o zachování hybnosti a rovnice 2.6 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 vlivem jednoho tělesa j lze pak vyjádřit jako:

$$\mathbf{v}_{cj} = \mathbf{w}_1 - \mathbf{v}_1 \tag{2.7}$$

Celková změna rychlosti $\mathbf{v_c}^{i+1}$ částice díky kolizím v daném kroku $i+1$ je pak součet všech dílčích změn $\mathbf{v}_{cj}$:

$$\mathbf{v_c}^{i+1} = \sum \mathbf{v}_c{}_j^{i+1} \tag{2.8}$$

2.1 Implementace a spuštění

Soubor `nbody.cpp` implementuje simulaci pohybu `N` částic `particles` v `STEPS` krocích s časovým posuvem `DT` sekund. Všechny potřebné parametry jsou definovány v souboru `Makefile`.
Soubor `velocity.cpp` implementuje výpočet změny rychlosti částic na základě výše popsaných rovnic.
Pro vygenerování vstupních hodnot použijeme připravený program *gen*. V datových souborech jsou vždy první tři hodnoty na řádku souřadnice částice, další tři tvoří vektor rychlosti dané částice a poslední je hmotnost.

```
$ make gen
$ ./gen 1000 ../input.dat
$ make run
```

2.2 Úkol 1: vektorizace funkcí

Většinu výpočtu při simulaci se provádí ve smyčce volající funkce pro výpočcet nové rychlosti.
Podívejte se na report `nbody.optrpt`.
Co udělal kompilátor?
Pomocí `#pragma omp simd` je možné vynutit vektorizaci.
Co radí kompilátor? Tuto modifikaci proveďte.
Je to optimální? Můžeme informovat kompilátor o vstupních datech?

2.3 Úkol 2: přístupy do paměti

Zkopírujte celý adresář `nbody/step1` do nového adresáře `nbody/step2`. V optimalizačním reportu `nbody.optrpt` byste měli vidět následující hlášení, nebo jejich obdobu:

```
scatter was emulated for the variable p: strided by 28 [ nbody.cpp(30,48)]
gather was emulated for the variable p: strided by 7 [ nbody.cpp(42,13) ]
```

Kompilátor tím dává najevo, že nemůže do vektorizačních registrů data přímo nakopírovat, ale musí je sesbírat a po výpočtu následně rozesílat. Vhodnou úpravou uložení dat v paměti tyto hlášení odstraňtě (ArrayOfStructures -> StructureOfArrays). Následně zarovnejte data v paměti a informujte o tomto zarovnání kompilátor.[1]
Po upravení struktur musíme upravit volání funkcí. Do funkce budeme předávat hodnotu `float` místo struktury.
Co se změnilo v reportu `nbody.optrpt`? Porovnejte s implementací z předešlých kroků.

[1] https://software.intel.com/en-us/articles/data-alignment-to-assist-vectorization