Architektura procesorů (ACH 2017)
Projekt č. 1: Optimalizace sekvenčního kódu

Gabriel Bordovský (ibordovský@fit.vutbr.cz)

Termín odevzdání: 26. 11. 2017

1 ÚVOD

Cílem tohoto projektu je vyzkoušet si optimalizaci sekvenčního kódu zejména pomocí vektorizace. Vaším úkolem bude optimalizovat úlohu částicového systému. Pro analýzu výkonnosti je připravena knihovna PAPI umožňující přístup k zabudovaným hardwarovým *performance counterům* uvnitř procesoru. Veškerý kód bude spouštěn na superpočítači Anselm.

2 SUPERPOČÍTAČ ANSELM

Superpočítač Anselm umístěný na VŠB v Ostravě je složen z celkem 209 uzlů, každý uzel disponuje 2 procesory Intel Xeon, většina (180) uzlů je založena na procesorech Intel Xeon E5-2665. Tyto procesory mají 8 jader s mikroarchitekturou Sandy Bridge, podporují tedy vektorové instrukce AVX. Pro připojení na superpočítač Anselm je potřeba mít vytvořený účet, s kterým je možné se připojit na tzv. čelní (login) uzel – `anselm.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:

```
[ibordovsky@login1.anselm ~]$ qsub -A DD-17-30 -q qexp -l select=1:ncpus=16,walltime=1:00:00 -I
qsub: waiting for job 262806.dm2 to start
qsub: job 262806.dm2 ready

[ibordovsky@cn117 ~]$
```

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 (pokud nebude úloha dlouhou dobu spuštěna, můžete použít frontu `qprod`, ale preferujte `qexp`), parametr `-l` určuje zdroje, které budou úloze přiděleny (počet uzlů, počet procesorů, čas). Abyste předešli zkreslení výkonových statistik, vždy alokujte celý uzel, tj. 16 jader. Interaktivní úlohu pak získáte parametrem `-I`.
Software na superpočítači Anselm 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 `intel` a `papi`:

```
ml intel/2016a papi
```

Příkaz `ml` zajišťuje práci s balíčky software, moduly. Modul `intel` zahrnuje `C/C++` kompilátor firmy Intel, který je možné vyvolat příkazy `icc` resp. `icpc`.
Modul `papi` pak obsahuje knihovnu PAPI, která usnadňuje přístup k hardwarovým performance counterům uvnitř procesoru. Každý procesor obsahuje několik (4-8) HW registrů, které jsou schopny počítat předem definované události. Mezi typické události, které nás zajímají, patří počet vykonaných FP/INT/LS instrukcí, IPC, počet přístupů do jednotlivých pamětí cache, propustnost paměti, přesnost predikce skoků atd. Knihovna PAPI obsahuje několik pomocných programů (`papi_avail`, `papi_native_avail`, `papi_mem_info`, ...), pomocí kterých je možné zjistit detaily o podpoře na daném procesoru. Pro zjednodušení práce s knihovnou PAPI je v projektu použita třída PapiCounter, která obaluje knihovnu PAPI. Její definice se nachází v souboru `papi_counter.h`. Kostra obou částí projektu již tuto třídu využívá. Seznam událostí, které chceme měřit, se předává přes proměnnou `PAPI_EVENTS`. Ta je již přednastavena v souboru `Makefile`. Po spuštění programu dojde k vypsání změřených hodnot do konzole.

3 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 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 (2 body)

Kostra aplikace je připravena v adresáři `nbody/step0`. 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`.
Vaším prvním úkolem je doplnění dvou funkcí pro výpočet změny rychlosti těles v souboru `velocity.cpp`.
Obě funkce zjišťují, jaký vliv má částice $p2$ na rychlost částice $p1$. Změna rychlosti je následně přičtena do pomocné proměnné vel. Na základě výše popsaných rovnic doplňte obě funkce. Není žádoucí upravovat jiné soubory ani rozhraní funkcí. Správnost výpočtu je možné ověřit spuštěním testovacího skriptu ve složce `stepX/test/`. Tyto testy zkompilují zdrojové soubory v nadřazeném adresáři s potřebnými parametry a následně porovnají výsledku běhu s referenčním.

```
[xbordo04@cn208.anselm tests]$ ./tests.sh
Two particles on circular trajectory...
Peak in possition difference:  1.57787e-05
*OK*
```

3.2 KROK 1: VEKTORIZACE FUNKCÍ (2 BODY)

Jakmile bude implementace funkční, zkopírujte celý adresář `nbody/step0` do nového adresáře `nbody/step1`. V tomto adresáři upravte `Makefile`. Řádek obsahující `REPORT` je potřeba odkomentovat a k řádku `OPT` přidat přepínače `-xavx` a `-qopenmp-simd`. Při kompilaci pak budou vygenerovány optimalizační reporty s koncovkou `.optrpt`.
Většinu výpočtu při simulaci se provádí ve smyčce volající funkce z nultého kroku. Podívejte se na report `nbody.optrpt`. Překladač zde usoudil, že není výhodné danou smyčku vektorizovat. Pomocí `#pragma omp simd` je možné vektorizaci vynutit. V reportu po kompilaci s touto pragmou sám kompilátor napoví, co by mohlo pomoci s efektivnější vektorizací. Tuto modifikaci proveďte. (Jedná se o variantu OpenMP pragmy.) V tomto kroku pracujte (krom úpravy `Makefile`) pouze s pragmy `omp simd`.

3.3 KROK 2: PŘÍSTUPY DO PAMĚTI (2 BODY)

Zkopírujte celý adresář `nbody/step1` do nového adresáře `nbody/step2`. V optimalizačním reportu nyní hlásí:

```
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 sbírat a 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] V reportu `nbody.optrpt` by se na konci tohoto kroku neměly vyskytovat hlášení o nezarovnaném přístupu do paměti či scatter/gatherer emulaci.
Neupravujte v tomto kroku nic v souborech `velocity.*`, pro ukládání dat již nepoužijete strukturu `particle`, ta bude použita jen pro předání dat do výpočetních funkcí.

3.4 KROK 3: REŽIE FUNKCÍ (1,5 BODY)

Tento krok se skládá ze tří postupně jdoucích částí a tak vzniknou postupně tři adresáře. Začněte tím, že zkopírujete celý adresář `nbody/step2` do nového adresáře `nbody/step3.1`.
Obě funkce v souboru `velocity.cpp` počítají vzdálenost dvou částic a určí změnu rychlosti částice p_1. Buď jedna nebo druhá funkce pak přičítá svůj výsledek. Spojte funkce do jedné a zjistěte jaký má toto spojení dopad na výkon.
Následně zkopírujte adresář `nbody/step3.1` do nového `nbody/step3.2` a upravte v tomto adresáři rozhraní funkce tak, aby na místo tří struktůr byly předávány pouze hodnoty $float$ či reference $float$ hodnoty. Zjistěte jak tato změna rozhraní ovlivňuje výkon.
Nyní zkopírujte adresář `nbody/step3.2` do nového `nbody/step3.3`. Volání funkcí uplně odstraňte a to tak, že kód výpočtu přesunete ze souboru `velocity.cpp` do těla smyčky `for` v souboru `nbody.cpp`

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

3.5 KROK 4: ODSTRANĚNÍ REDUNDANTNÍCH VÝPOČTŮ (1,5 BODY)

Nyní zkopírujte adresář `nbody/step3.3` do nového `nbody/step4`. Doposud výpočet simulace probíhal se složitostí $N \times N$. Gravitační síla mezi částicemi p_1 a p_2 má stejnou velikost, ale opačný směr a může z ní být odvozena změna rychlosti. Stejně tak při odrazu jsou všechny potřebné parametry pro výpočet nové rychlosti částice p_2 již k dispozici při výpočtu rychlosti částice p_1. Využijte toho a snižte složitost výpočtu na $N \times N/2$. Nezapomeňte správně ošetřit současný zápis na stejné paměťové místo. (Je potřeba o něco rozšířit OpenMP pragma z kroku 1.)
Pomocí programu `gen` generujte datové soubory různých velikostí. Např. pro vygenerování souboru s 20000 částicemi použijte následující příkaz:

```
./gen 20000 20k.dat
```

Pomocí PAPI sledujte míru výpadků jednotlivých úrovní cache a určete tak hranice, kdy velikost dat přesáhne velikost cache L1 a L2. Teoreticky určete hranici pro cache L3. Počet použitelných HW counterů je omezen. Použité HW countery zapište do taktéž zapište do `nbox.txt`. Výsledky zapište do souboru `nbody.txt`. Popište, jaký dopad na výkon mají výpadky v cache a jak byste tento vliv omezili.

4 VÝSTUP PROJEKTU A BODOVÁNÍ

Výstupem projektu bude soubor `xlogin00.zip` obsahující všechny zdrojové soubory a textový soubor `nbody.txt`. Do tohoto textového souboru zaznamenávejte v každém kroku výkon daného řešení naměřený pomoci knihovny PAPI a zodpovězte požadované otázky. V každém souboru (který jste změnili) 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. Hodnocení je uvedené u jednotlivých kroků a dohromady tvoří 9 bodů. Poslední desátý bod si hodnotící vyhrazuje k hodnocení čitelnosti odevzdaných souborů. Projekt odevzdejte v uvedeném termínu do informačního systému.