

Architektura procesorů (ACH 2018)

Projekt č. 1: Optimalizace sekvenčního kódu

Gabriel Bordovský (ibordovský@fit.vutbr.cz)

Termín odevzdání: 25. 11. 2018

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 spuštění 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-18-32 -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. Více o spouštění úloh na superpočítačích IT4I naleznete na stránce <https://docs.it4i.cz/anselm/job-submission-and-execution/>.

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 s kompilátorem z roku 2016 je vybrán z důvodu čitelnějších optimalizačních výstupů. Dobrovolně můžete porovnat výkon výsledného řešení kompilovaného novějším kompilátorem s tímto.

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 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 pomocí knihovny PAPI pro 1500 částic 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.

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 tohoto tělesa. Pro vektory polohy \mathbf{r} a rychlosti \mathbf{v} platí:

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

$$\mathbf{v}^{i+1} = \mathbf{v}^i + \mathbf{v}_g^{i+1} + \mathbf{v}_c^{i+1} \quad (4.2)$$

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:

$$\mathbf{F} = \frac{G \cdot m_1 \cdot m_2}{r^2}, \quad (4.3)$$

kde $G = 6.67384 \cdot 10^{-11} \text{Nm}^2\text{kg}^{-2}$ je gravitační konstanta, m_1 a m_2 jsou hmotnosti těles a 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 hmotností a vynásobena délkou kroku simulace:

$$\mathbf{v}_g^{i+1} = \frac{\sum_j \mathbf{F}_j^{i+1}}{m} \cdot \Delta t \quad (4.4)$$

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

$$\mathbf{v}_1 \cdot m_1 + \mathbf{v}_2 \cdot m_2 = \mathbf{w}_1 \cdot m_1 + \mathbf{w}_2 \cdot m_2 \quad (4.5)$$

$$\frac{1}{2} \cdot \mathbf{v}_1^2 \cdot m_1 + \frac{1}{2} \cdot \mathbf{v}_2^2 \cdot m_2 = \frac{1}{2} \cdot \mathbf{w}_1^2 \cdot m_1 + \frac{1}{2} \cdot \mathbf{w}_2^2 \cdot m_2 \quad (4.6)$$

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 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 \mathbf{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:

$$\mathbf{v}_{cj} = \mathbf{w}_1 - \mathbf{v}_1 \quad (4.7)$$

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} :

$$\mathbf{v}_c^{i+1} = \sum \mathbf{v}_{cj}^{i+1} \quad (4.8)$$

4.1 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`.

Vášim 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...
Maximum in position difference: 1.57787e-05
*OK*
```

V datových souborech jsou vždy první tři hodnoty na řádku souřadnice částice, další tři tvoří vektor rychlosti dané částice a poslední je hmotnost. Pro vizualizaci bodů v prostoru můžete použít například příkaz `splot` nástroje `gnuplot`. Modul `gnuplot` je před použitím potřeba nahrát explicitně.

```
[xbordo04@login1.anselm tests]$ gnuplot
%gnuplot> splot 'FILENAME.dat' using 1:2:3
```

4.2 KROK 1: VEKTORIZACE FUNKCÍ (2 BODY)

Jakmile bude implementace funkční, zkopírujte celý adresář `nbody/step0` do nového adresáře `nbody/step1`. V tomto adresáři upravit `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 OpenMP určenými k vektorizaci.

4.3 KROK 2: PŘÍSTUPY DO PAMĚTI (2 BODY)

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 odstraňte (`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.

4.4 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 struktur 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 úplně odstraňte a to tak, že kód výpočtu přesunete ze souboru `velocity.cpp` do těla smyčky `for` v souboru `nbody.cpp`

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

Opět 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 zajistit aby nedošlo k dělení nulou a 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 současně použitelných HW counterů je omezen. Použité HW countery zapište do souboru `nbody.txt`. Výsledky zapište do souboru `nbody.txt`. Popište, jaký dopad na výkon mají výpadky v cache a jak byste tento vliv omezili.

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