

Architektura procesorů (ACH 2016)

Projekt č. 1: Optimalizace sekvenčního kódu

Jakub Budiský (ibudisky@fit.vutbr.cz)

Termín odevzdání: 20. 11. 2016

1 ÚVOD

Cílem tohoto projektu je vyzkoušet si optimalizaci sekvenčního kódu zejména pomocí vektorizace. Projekt je rozdělen na dvě části – v první části bude úkolem dle zadaného postupu optimalizovat úlohu násobení matice a vektoru, v druhé části pak svými vlastními silami vylepšit 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:

```
[ibudisky@login1.anselm ~]$ qsub -A DD-16-43 -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

[ibudisky@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 pro tento projekt 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`:

```
module load intel/2016a papi
```

Modul `intel` zahrnuje C/C++ kompilátor firmy Intel, který je možné vyvolat příkazy `icc` resp. `icpc`. Číslo verze modulu za lomítkem je nepovinné, avšak v současnosti je výchozí Intel kompilátor `intel/13.5.192`, který ještě nedisponuje plnou podporou OpenMP 4. Proto je nutné verzi explicitně uvést.

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 SOUČIN MATICE A VEKTORU (4 BODY)

Začněte s naivní implementací v adresáři `matvec/step0` a postupujte dle následujících kroků. Pro každý další krok zkopírujte výsledek posledního kroku do nového adresáře (např. po nulovém kroku zkopírujte `matvec/step0` do `matvec/step1`) a pokračujte dalším krokem.

```
void mat_vec_mul(int rows, int cols, float** a, float* b, float* c)
{
    for (int i = 0; i < rows; i++)
    {
        c[i] = 0.0f;
        for (int j = 0; j < cols; j++)
            c[i] += a[i][j] * b[j];
    }
}
```

3.1 KROK 0: VYPNUTÉ OPTIMALIZACE

Výchozí nastavení kompilátoru používá optimalizace -O2. Nás však zajímá, jaký výkon má naivní kód bez využití optimalizací a proto optimalizace vypneme parametrem -O0. Prostudujte soubor Makefile, kód zkompilejte pomocí příkazu make a spusťte pomocí make run. Pro detailní pochopení práce kompilátoru prostudujte kód v symbolických instrukcích (matvec.s). Najděte kód odpovídající vnitřní smyčce a určete, jaká instrukce byla kompilátorem použita např. pro násobení.

3.2 KROK 1: ZAPNUTÍ OPTIMALIZACÍ (1 BOD)

Nyní zapněte optimalizace (-O2) a sdělte kompilátoru, že chcete využít instrukci AVX (-xavx), parametry запиšte do Makefile na řádek OPT=. Porovnejte výkonnost s předchozím krokem. Pro zobrazení informací o provedených optimalizacích použijte volbu -qopt-report=5¹. Parametr запиšte do souboru Makefile na řádek REPORT=, při kompilaci bude vytvořen soubor matvec.optrpt obsahující detailní informace o provedených optimalizacích.

Nyní odstraňte direktivu #pragma nounroll ve vnitřní smyčce mat_vec_mul() a spusťte kompilaci znovu. Najděte v reportu informaci o tom, zda byla smyčka vektorizována, a kolikrát byla rozbalena. Opět najděte kód odpovídající vnitřní smyčce a určete, jaké instrukce byly kompilátorem použity tentokrát.

3.3 KROK 2: DATOVÉ ZÁVISLOSTI (1 BOD)

Z optimalizačního reportu v předchozím kroku je patrné, že k vektorizaci sice došlo, ale kompilátor byl nucen smyčku transformovat:

```
remark #25228: Loop multiversed for Data Dependence
remark #25085: Preprocess Loopnests: Moving Out Load and Store
```

Kompilátor neví, jestli cyklus náhodou neobsahuje datové závislosti (pole a, b, c se mohou obecně překrývat). My však víme, že neobsahuje. Můžeme tedy kompilátoru pomoci tím, že mu tuto skutečnost poradíme. To lze provést více způsoby:

- OpenMP 4 direktiva #pragma omp simd
- parametr -fargument-noalias
- direktiva #pragma ivdep
- klíčové slovo restrict

Pro tenhle projekt použijte direktivu OpenMP verze 4², která podporuje vektorizaci. Nastudujte možné klauzule pro direktivu omp simd a nezapomeňte, že kód provádí redukci. Do Makefile na řádek OPT= přidejte přepínač -qopenmp-simd.

¹Více informací o formátu reportu lze nalézt zde: <https://software.intel.com/en-us/articles/getting-the-most-out-of-your-intel-compiler-with-the-new-optimization-reports>

²Referenční manuál k dispozici zde: <http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf>

Znovu zkompilejte kód a **zkontrolujte správnost** výsledku! Pomocí optimalizačního reportu ověřte, že smyčka byla vektorizována. Porovnejte výkon s předchozím krokem. Najděte kód symbolických instrukcí zodpovídající vnitřní smyčce a vysvětlete, proč je stále tak dlouhý.

3.4 KROK 3: ZAROVNÁNÍ DAT V PAMĚTI (1 BOD)

Zarovnáním dat v paměti lze dále optimalizovat kód. Úpravou alokace v souboru `main.cpp` vynuťte zarovnání řádků matice `a` a vektoru `b` v paměti. Informujte o této změně přidáním vhodné klauzule `aligned` do použité `OpenMP` direktivy.

Na kolik bytů musí být data v paměti zarovnána při použití instrukcí `AVX`? Ověřte, že vstupní data jsou zarovnána v optimalizačním reportu.

Opět porovnejte výkon oproti předchozímu kroku. Prohlídněte si také kód v symbolických instrukcích a vysvětlete změny ve vnitřní smyčce. Z hodnot `HW counterů` určete, zda program využívá vektorovou jednotku efektivně, nebo stále využívá i skalární instrukce.

3.5 KROK 4: PADDING (1 BOD)

Matice `c` má 32 řádků a 31 sloupců, řádky tedy nejsou zarovnané na velikost registrů `YMM`. Změňte parametr `PADDING` v souboru `Makefile` tak, aby byla velikost řádku pole `c` násobkem velikosti registrů `YMM`. Rozměry matice `c` zůstanou zachovány, každý řádek však bude obsahovat určitý okraj, se kterým se nebude počítat. Porovnejte výkon s předchozím krokem a rozdíl vysvětlete. Změnila se nějak vnitřní smyčka? Vysvětlete.

3.6 VÝSTUP PRVNÍ ČÁSTI PROJEKTU

Výstupem první části projektu budou zdrojové kódy jednotlivých kroků, tedy adresáře `step0`, `step1`, ..., `step5`, a soubor `matvec.txt` s textovým komentářem k jednotlivým krokům – u každého kroku vyplňte naměřené parametry výkonu a odpovězte na všechny otázky.

4 ČÁSTICOVÝ SYSTÉM (11 BODŮ)

Cílem této části 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, rychlosti a zrychlení platí:

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

$$\mathbf{v}^{i+1} = \mathbf{v}^i + \mathbf{a}^{i+1} \cdot \Delta t \quad (4.2)$$

$$\mathbf{a}^{i+1} = \frac{\sum \mathbf{F}_j^{i+1}}{m} \quad (4.3)$$

Síla \mathbf{F}^{i+1} působící na těleso je dána vektorovým součtem dílčích sil způsobených gravitačním polem ostatních těles. Dvě tělesa na sebe působí gravitační silou danou:

$$F = \frac{G \cdot m_1 \cdot m_2}{R^2}, \quad (4.4)$$

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. V každém kroku výpočtu je tedy nutné spočítat síly působící mezi jednotlivými tělesy a změny rychlostí a poloh jednotlivých těles.

4.1 KROK 0: ZÁKLADNÍ IMPLEMENTACE (3 BODY)

Kostra aplikace je připravena v adresáři `nbody/step0`. Prvním krokem bude doplnění funkce `particles_simulate` v souboru `nbody.cpp` tak, aby funkce správně simulovala pohyb N částic `particles` v STEPS krocích s časovým posuvem DT sekund. Všechny tyto parametry jsou definovány v souboru `Makefile`. Správnost výpočtu je možné ověřit porovnáním výstupního souboru se vzorovým výstupem `nbody/output.dat`.

4.2 KROK 1: OPTIMALIZOVANÁ IMPLEMENTACE (4 BODY)

Jakmile bude implementace funkční, zkopírujte celý adresář `nbody/step0` do nového adresáře `nbody/step1` a pokračujte optimalizací kódu. Porovnejte výkon základní implementace s optimalizovaným kódem. Do textového souboru `nbody.txt` popište provedené optimalizace a vysvětlete jejich dopad. Pokud nelze kód dále optimalizovat pomocí předvedených metod, zdůvodněte.

Pro zarovnání statických polí můžete využít klíčového slova `__declspec(align(N))`.

4.3 KROK 2: ANALÝZA PAMĚŤOVÉ HIERARCHIE (4 BODY)

Zkopírujte celý adresář `nbody/step1` do nového adresáře `nbody/step2`. Modifikujte proměnnou `PAPI_EVENTS` v `Makefile` tak, abyste mohli zjistit míru výpadků v jednotlivých úrovních cache (L1, L2, L3). Statistiky jednotlivých úrovní cache sledujte postupně, počet použitelných HW counterů je omezen. Použité HW countery zapište do `nbox.txt`. Pomocí programu `gen` generujte datové soubory různých velikostí. Např. pro vygenerování souboru s 20000 částicemi použijte následující příkaz:

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

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. Výsledky zapište do souboru `nbody.txt`. Popište, jaký dopad na výkon mají výpadky v cache a jak byste tento vliv omezili.

5 VÝSTUP PROJEKTU A BODOVÁNÍ

Výstupem projektu bude soubor `xlogin00.zip` obsahující všechny zdrojové soubory a textové soubory `matvec.txt` a `nbody.txt` obsahující textový komentář k oběma částem projektu. 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. Projekt odevzdejte v uvedeném termínu do informačního systému.