Profiling, Optimalizace a Paralelizace

Ing. Filip Vaverka

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2. 612 66 Brno - Královo Pole

ivaverka@fit.vutbr.cz

Návrh řešení problému

1. Hrubá formulace problému, způsobu řešení a požadavků
   - Prvotní analýza umožňující odhad výpočetní náročnosti řešení problému
2. Volba platformy vhodné pro zvolený způsob řešení
   - Bude CPU dostačující? Je úloha vhodná pro akcelerátory?
   - Kolik paměti je třeba?
3. Návrh a implementace řešení pro zvolenou platformu
   - Rozdělení úlohy mezi výpočetní jednotky
   - Volba/Návrh konkrétních algoritmů
4. Optimalizace implementovaného řešení
   - Hledání kritických míst v programu
   - Analýza využití HW prostředků
   - Modifikace implementace pro zlepšení využití HW

Software

- Intel Parallel Studio XE[1]
  - Komplexní nástroj pro analýzu vektorizace, práce s pamětí a paralelizace
  - Specializace na architektury Intel
- AMD μProf[2]
  - Jednoduchý nástroj pro základní analýzu programu
  - Architektury AMD
- PAPI[3]
  - Nízkoúrovňová knihovna pro přístup k "Performance Counters"
  - Pouze Linux-like systémy
- Překladače
  - Intel C++ Compiler, GCC, Clang, MSVC

[1] https://software.intel.com/en-us/parallel-studio-xe
[2] https://developer.amd.com/amd-uprof
[3] http://icl.utk.edu/papi

Základní přístupy k profilování

- Time sampling based
  - Pravidelně pozastavuje vykonávání programu a zaznamená aktuálně zpracovávanou funkci/instrukci
- Event based
  - Využívá hardwarové přerušení v kombinaci s čítači událostí v CPU
  - Případně události vistuálního stroje (Java, .NET, ...)
- Instrumentace kódu
  - Modifikuje program tak aby sám zaznamenával požadované informace
  - Např. ScoreP[4]

Pozor na délku profilovaného běhu!

[4] `https://www.vi-hps.org/projects/score-p`

Demonstrační příklady

- µ-Architektura
  - Predikce skoků (`ivs_demo_branch.cpp`)
  - Vektorizace (`ivs_demo_saxpy.cpp`)
- Paměťový sub-systém
  - Vliv využití cache (`ivs_demo_cache.cpp`)
- Vláknový paralelismus (OpenMP[5])
  - Paralelizace smyček (`ivs_demo_saxpy.cpp`)

[5] `https://www.openmp.org/specifications`

- Jaký vliv na výkon moderního CPU má predikce skoků?
  - Chybná predikce skoku způsobuje provedení práce, která bude později zahozena.

- Kód:

```
size_t count = 0;
for(size_t i = 0; i < size; ++i)
{
    if(pD[i] > 0.0f)
        count++;
}
```

- Pro `pD[i] = rand(-0.5, 0.5)` a `pD[i] = 1.0`
- Je nutné vypnout optimalizace!

- Jaký vliv na výkon CPU má práce s pamětí cache?

- Kód:

```
for(size_t i = 0; i < size; i += 16)
    pD[i % blockSize] += 1.0f;
```

- Pro "`blockSize`" = $\{8 \cdot 2^{10}, 9 \cdot 2^{10}, 96 \cdot 2^{10}, 6 \cdot 2^{20}\}$, tedy 32 kB, 36 kB, 384 kB, 24 MB
- Hodnoty jsou zvoleny tak, aby postupně docházelo k přístupům do L1, L2, L3 cache a následně do DRAM (systémové paměti)
- Program modifikuje pouze každý 16. prvek, což omezuje využití aritmetických jednotek, ale zachovává nutnost načtění každé "cache line" (64 B)

Numerický kód: Výchozí úloha

- SAXPY smyčka ($\mathbf{s} = a\mathbf{x} + \mathbf{y}$)

```
for(size_t i = 0; i < size; ++i)
{
    pC[i] = a * pB[i] + pC[i];
}
```

- Skalární součin vektorů ($d = \mathbf{b} \cdot \mathbf{c} = \sum_i b_i c_i$)

```
float dot = 0.0f;
for(size_t i = 0; i < size; ++i)
{
    dot += pB[i] * pC[i];
}
```

- SAXPY smyčka ($\mathbf{s} = a\mathbf{x} + \mathbf{y}$)

```
#pragma omp simd
for(size_t i = 0; i < size; ++i)
{
    pC[i] = a * pB[i] + pC[i];
}
```

- Skalární součin vektorů ($d = \mathbf{b} \cdot \mathbf{c} = \sum_i b_i c_i$)

```
float dot = 0.0f;
#pragma omp simd reduction(+:dot)
for(size_t i = 0; i < size; ++i)
{
    dot += pB[i] * pC[i];
}
```

- SAXPY smyčka ($\mathbf{s} = a\mathbf{x} + \mathbf{y}$)

```
#pragma omp parallel for simd schedule(static)
for(size_t i = 0; i < size; ++i)
{
    pC[i] = a * pB[i] + pC[i];
}
```

- Skalární součin vektorů ($d = \mathbf{b} \cdot \mathbf{c} = \sum_i b_i c_i$)

```
float dot = 0.0f;
#pragma omp parallel for schedule(static)
#pragma omp simd reduction(+:dot)
for(size_t i = 0; i < size; ++i)
{
    dot += pB[i] * pC[i];
}
```

Numerický kód: Práce s pamětí

- Dvojí průchod přes data omezuje znovupoužití dat (tzv. aritmetická intenzita)
- Smyčky lze snadno sloučit do jediného průchodu:

```
float dot = 0.0f;
#pragma omp parallel for schedule(static)
#pragma omp simd reduction(+:dot)
for(size_t i = 0; i < size; ++i)
{
    pC[i] = a * pB[i] + pC[i]; // SAXPY
    dot += pB[i] * pC[i]; // DOT (data reuse)
}
```

ivaverka@fit.vutbr.cz