IA039: Architektura superpočítačů a náročné výpočty

Profiling and Benchmarking

Luděk Matyska

Fakulta informatiky MU

Jaro 2018

Měření času výpočtu

- Optimalizace není možní bez znalosti, co optimalizovat
- Potřebujeme znát údaje o běhu programu
  - Čas výpočtu celého programu: příkaz **time**
  - Čas výpočtu částí programů: profilování
  - Srovnání systémů: benchmarking

Příkaz **time**

- User time
  - Čas procesoru strávený uživatelskými procesy
- System time
  - Čas procesoru strávený obsluhou funkcí jádra
- Elapsed time
  - Celkový čas výpočtu

CPU time = user time + system time

Příkaz **time** – další údaje

- Dostupné pro **time** v prostředí C shellu
- Sdílený paměťový prostor
- Privátní (unshared) paměťový prostor
- Počet block input operací
- Počet block output operací
- Počet page faultů
- Počet swapů

Profilování

- Snaha získat informace o částech programu
- Důraz na dynamické chování
  - statická analýza je součástí softwarového inženýrství
- Profilování ukazuje výsledek interakce mezi programem a výpočetním systémem, na kterým běží
  - Doba strávená v jednotlivých blocích
  - Doba strávená jednotlivými příkazy
  - Počet opakování bloků/příkazů
- Primární pozornost zaměřena na procedury
- Profil: graf
  - Osa X: jednotlivé procedury
  - Osa Y: Doba výpočtu

Základní principy

- Používá softwarové nástroje pro sběr dat nezbytných na konstrukci profilu
- Zpravidla podpora ze strany operačního systému
  - přístup k informacím, které má k dispozici pouze kernel
- Příklady běžných nástrojů pro profilování
  - gprof, oprofile, valgrind, pin
- Profile je sbírán při běhu programu – dynamický profil
- Analýza profilových dat se nazývá Performance Engineering

Typy sbíraných dat

- Graf volání (call graph) na úrovni procedur a funkcí
- Graf volání na úrovni základních bloků
- Výkon paměti
- Události související s architekturou, např. chybné předpovědi skoků, výjimky, výkon vyrovnávací paměti (hit/miss), ...
- Hodnoty výkonnostních čitačů (performance counters)

Typy profilů

- Ostrý profil
  - Píky odpovídají dominujícím blokům
  - Numerické aplikace, technické výpočty s maticemi atd.
  - „Snadno“ optimalizovatelné
- Plochý profil
  - Program tráví čas rovnoměrně ve všech procedurách
  - Zpravidla databáze, informační systémy, systémové programy
  - Obtížně optimalizovatelné
- Amdahlovo pravidlo platí i zde

Profilery

- Nástroje pro tvorbu profilů
  - Samozřejmě možné i ,,ručně“
- Procedurově orientované
  - **gprof**
- Blokově (řádkově) orientované
  - **pixie**
  - **tcov**
  - **lprof**

Použití profilerů

- Dvě fáze:
  - Instrumentovaný běh programu (s nebo bez opětného překladu)
  - Vlastní zpráva o výsledcích (report)
- Přístup ke zdrojovému kódu
  - Znalost struktury programu

Procedurově orientované profilery

- Typický představitel: **gprof**
- Instrumentace
  - Nový překlad programu
  - Zpravidla dostupný přes přepínač překladače **-pg**
- Výpočet
  - Instrumentovaný program vytváří záznam o výpočtu
  - Soubor **gmon.out**
- Vytvoření zprávy
  - Vlastní běh programu **gprof**

Typy profilů

SpeedShop jako příklad

- usertime – uživatelský čas
- [f]pcsamp[x] – vzorkování
- ideal (pixie) – blokový profiler
- fpe – pohyblivá řádová čárka
- prof_hwc – hardwarové čitače
  - f gi_hwc, [f]cy_hwc, [f]ic_hwc, [f]isc_hwc, [f]dc_hwc, [f]dsc_hwc, [f]tlb_hwc, [f]gfp_hwc
- Framework PAPI

Přesnost měření

- Měření času
  - Absolutní čas vstupu a výstupu procedury
    - Vnořené procedury
    - Krátké procedury
  - Sběr hodnoty čitače instrukcí v pravidelných intervalech
    - Přesnost výsledku závislá na vzorkování (sampling interval)

Blokově orientované profilery

- Poskytují informace o průchodech základními bloky
- Počet průchodů každým příkazem (řádkem)
- Počet cyklů procesorů strávený v každém příkazu

Příklad

```
 1: static void foo(), bar(),
 2:                  baz();
 3: main()
 4: { int l;
 5:   for (l=0; l <1000; l++)
 6:   { if ( l == 2*(l/2) )
 7:       foo();
 8:     bar();
 9:     baz();
10:   }
11: }
12:
```

```
13: void foo()
14: { int j;;
15:   for (j=0; j<200; j++
16: }
17: void bar()
18: { int j;
19:   for (j=0; j<200; j++
20: }
21: void baz()
22: { int j;
23:   for (j=0; j<300; j++
24: }
```

Usertime

```
usertime:
index   %Samples   self   descendents   total           name
[1]     100.0%     0.00          0.03       1           main
[2]     100.0%     0.03          0.00       1           bar
```

Vzorkování

```
pcsamp:
samples       time(%)       cum time(%)     procedure

      2     0.02s( 50.0)      0.02s( 50.0) foo
      1     0.01s( 25.0)      0.03s( 75.0) bar
      1     0.01s( 25.0)      0.04s(100.0) baz

fpcsamp:
samples       time(%)       cum time(%)     procedure
     18     0.02s( 41.9)      0.02s( 41.9) baz
     12     0.01s( 27.9)      0.03s( 69.8) bar
     12     0.01s( 27.9)      0.04s( 97.7) foo
```

Blokový přístup

```
ideal:
    cycles(%)       cum %      secs     instrns  alls procedu
  3918000(49.63)   49.63      0.03     2111000  1000 baz
  2618000(33.16)   82.80      0.02     1411000  1000 bar
  1309000(16.58)   99.38      0.01      705500   500 foo
    47024( 0.60)   99.98      0.00       25017     1 main
```

Blokový přístup II

```
ideal -h:
          cycles(%)    cum %  times  line procedure
    3907858(49.50%)  49.50% 300000    23 baz
    2607858(33.04%)  82.54% 200000    19 bar
    1303930(16.52%)  99.06% 100000    15 foo
      14000( 0.18%)  99.24%   1000     6 main
      13009( 0.16%)  99.40%   1000     5 main
       8000( 0.10%)  99.50%   1000     9 main
       8000( 0.10%)  99.60%   1000     8 main
       7000( 0.09%)  99.69%   1000    24 baz
       7000( 0.09%)  99.78%   1000    20 bar
       4000( 0.05%)  99.83%    500     7 main
       3500( 0.04%)  99.88%    500    16 foo
       3142( 0.04%)  99.92%   1000    18 bar
       3142( 0.04%)  99.96%   1000    22 baz
       1570( 0.02%)  99.98%    500    14 foo
```

Blokové (tcov)

```
tcov:
      1 ->   for (l=0; l <1000; l++)
   1000 ->   { if ( l == 2*(l/2) )
    500 ->       foo();
   1000 ->     bar();
               baz();

           void foo()
    500 ->   for (j=0; j<200; j++);

           void bar()
   1000 ->   for (j=0; j<200; j++);

           void baz()
   1000 ->   for (j=0; j<300; j++);
```

Blokové – pokračování

```
Top 10 Blocks
Line          Count
   6           1000
   8           1000
  19           1000
  23           1000
   7            500
  15            500
   5              1

  7    Basic blocks in this file
  7    Basic blocks executed
100.00 Percent of the file executed
  5001  Total basic block executions
714.43  Average executions per basic block
```

Profiling

viz též `http://www.site.uottawa.ca/~mbolic/elg6158/Subhasis_profiling.pdf`

Zjišťování výkonu – benchmarking

- Snaha o porovnání *systémů*
  - Hardware i software společně
- Neexistuje žádné ,,zázračné“ řešení
- Základní přístupy
  - Průmyslové (,,profesionální“) benchmarky
    - Porovnatelnost, nezávislost na výrobcích
  - ,,Privátní“ benchmarky
    - Konkrétní (specifické) požadavky

Mysteriózní MIPS a MFLOPS

- Srovnání na základě počtu instrukcích vykonaných za sekundu
- MIPS – milion celočíselných instrukcí za sekundu
- MFLOPS – milion operací s pohyblivou řádovou čárkou za sekundu
- Problémy
  - Jaké instrukce?
  - V jaké posloupnosti?
- Umělé, nevypovídající

Celočíslené benchmarky

- VAX MIPS
- Dhrystones

Benchmarky s pohyblivou řádovou čárkou

- Whetstone (umělý mix, skalární)
- Linpack (daxpy, vektorizace)
  - 100*100
  - 1000*1000

SPEC benchmarks

- Nezávislá organizace
  - Standard Performace Evaluation Corporation
- Standardizované benchmarky pro různé architektury
- Vychází z tzv. *kernel kódů*
  - Celý nebo část existujícího programu
  - Dostupné ve zdrojovém kódů
    - Je možno ,,doladit“

SPEC skupiny

- Open Systems Group (OSG)
- High Performance Group (HPG
- Graphics Performance Characterization Group (GPC)

SPEC OSG podskupiny

- CPU
  - SPECmarks a CPU benchmarks
- JAVA
  - JVM98, JBB2005, Java client a server benchmarky
- MAIL
  - SPECmail2001
- SFS
  - Systémy souborů (SFS97)
- WEB
  - WEB99, WEB99_SSL, WEB2005

CPU2006

- Aktuální CPU benchmark
- Dělení
  - CINT2006 – celočíselné výpočty
  - CFP2006 – výpočty s pohyblivou řádovou čárkou

CINT2000

- Jednotlivé součásti

164.gzip	Compression
175.vpr	FPGA Circuit Placement and Routing
176.gcc	C Programming Language Compiler
181.mcf	Combinatorial Optimization
186.crafty	Game Playing: Chess
197.parser	Word Processing
252.eon	Computer Visualization
253.perlbmk	PERL Programming Language
254.gap	Group Theory, Interpreter
255.vortex	Object-oriented Database
256.bzip2	Compression
300.twolf	Place and Route Simulator

CINT2006

- Jednotlivé součásti

400.perlbench	C	PERL Programming Language
401.bzip2	C	Compression
403.gcc	C	C Compiler
429.mcf	C	Combinatorial Optimization
445.gobmk	C	Artificial Intelligence: go
456.hmmer	C	Search Gene Sequence
458.sjeng	C	Artificial Intelligence: chess
462.libquantum	C	Physics: Quantum Computing
464.h264ref	C	Video Compression
471.omnetpp	C++	Discrete Event Simulation
473.astar	C++	Path-finding Algorithms
483.xalancbmk	C++	XML Processing

CFP2000

- Jednotlivé součásti

168.wupwise	Physics / Quantum Chromodynamics
171.swim	Shallow Water Modeling
172.mgrid	Multi-grid Solver: 3D Potential Field
173.applu	Parabolic / Elliptic Partial Differential Equations
177.mesa	3-D Graphics Library
178.galgel	Computational Fluid Dynamics
179.art	Image Recognition / Neural Networks
183.equake	Seismic Wave Propagation Simulation
187.facerec	Image Processing: Face Recognition
188.ammp	Computational Chemistry
189.lucas	Number Theory / Primality Testing
191.fma3d	Finite-element Crash Simulation
200.sixtrack	High Energy Nuclear Physics Accelerator Design
301.apsi	Meteorology: Pollutant Distribution

CFP2006

410.bwaves	Fortran	Fluid Dynamics
416.gamess	Fortran	Quantum Chemistry
433.milc	C	Physics: Quantum Chromodynamics
434.zeusmp	Fortran	Physics/CFD
435.gromacs	C/Fortran	Biochemistry/Molecular Dynamics
436.cactusADM	C/Fortran	Physics/General Relativity
437.leslie3d	Fortran	Fluid Dynamics
444.namd	C++	Biology/Molecular Dynamics
447.dealII	C++	Finite Element Analysis
450.soplex	C++	Linear Programming, Optimization
453.povray	C++	Image Ray-tracing
454.calculix	C/Fortran	Structural Mechanics
459.GemsFDTD	Fortran	Computational Electromagnetics
465.tonto	Fortran	Quantum Chemistry
470.lbm	C	Fluid Dynamics
481.wrf	C/Fortran	Weather Prediction

Transakční benchmarky

- Výkon databází
  - TPC-A
    - Testuje interakci s ATM (6 požadavků za minutu)
    - 1 TPS znamená, že 10 ATM současně vydá požadavek a výsledek dostane do 2 s (s 90 % účinností)
  - TPC-B
    - Jako TPC-A, ale testuje se přímo, ne přes (pomalou) síť
  - TPC-C
    - Komplexní, transakce jsou objednávky, platby, dotazy s jistým procentem záměrných chyb vyžadujících automatickou korekci

Síťové benchmarky

- netperf
- iperf
- End to end měření
- Pozor na to, co skutečně v síti měříme

Vlastní benchmarky

- Specifické (konkrétní) požadavky
- Důležité parametry:
  - Co testovat
  - Jak dlouho testovat
  - Požadavky na velikost paměti
- Typy benchmarků
  - Jednoduchý proud (opakování)
  - Propustnost (benchmark stone wall)

Kontroly

- Nezbytná součast používání benchmarků
  - Měříme opravdu to, co chceme?
- Možné příčiny ovlivnění
  - Použitá optimalizace
  - Velikost paměti
  - Přítomnost jiných procesů
- Co je třeba explicitně kontrolovat
  - CPU čas a čas nástěných hodin
  - Výsledky!
  - Srovnání se ,,známým“ standardem