Měření výkonu počítače

Břetislav Bakala

Měření výkonu počítače

Co je myšleno, že jeden počítač je rychlejší než druhý? Je možno **porovnat čas na vykonání určité úlohy**. Každá úloha muže být specificky rozdílná (transakce za hodinu, čas provedení výpočtu, propustnost systému)

Systém X je n krát rychlejší, než systém Y:

$$\frac{\check{c}as\ vykon\acute{a}n\acute{i}_y}{\check{c}as\ vykon\acute{a}n\acute{i}_x} = n$$

Vykonávaný čas je převrácená hodnota vyjádření výkonnosti :

$$n = \frac{\check{c}as\ vykon\acute{a}n\acute{i}_{\ y}}{\check{c}as\ vykon\acute{a}n\acute{i}_{\ x}} = \frac{\frac{1}{v\acute{y}konnost_y}}{\frac{1}{v\acute{y}konnost_x}} = \frac{v\acute{y}konnost_x}{v\acute{y}konnost_y}$$

Např. propustnost systému X je 1,3x vyšší než Y znamená, že počet úloh provedených za jednotku času na počítači X je 1,3x větší než počet úloh provedených za stejný čas na počítači Y

Stanovení metody měření výkonu

Při stanovení času pro vykonání určité úlohy záleží na tom, co je do této doby zahrnuto. (čas odezvy, zpoždění přístupu k datům na disku při provedení úlohy, ostatní I/O operace …)

Uživatel nemůže tedy hodnotit rychlost pouze podle CPU time – čas, po který procesor pracuje na dané úloze (nejsou započítávány ostatní časy). Nejlepší volba by byla změřit výkon na reálné aplikaci.

Dříve používané metody

Nelze objektivně změřit výkonnost pomocí jednoduchých programů.

Např. dříve používané metody vedly k úskalím při měření výkonu:

- U malých jader se používaly při měření klíčové **části reálných aplikací**
- Malé jednoduché programy (do 100 řádků) demonstrující **elementární úlohu** (např. Quicksort)
- Uměle vytvořené testovací programy snažící se přizpůsobit profilu a chování skutečných aplikací (např. **Dhrystone** – reprezentativní mix vybraný z běžných konstrukcí programu např. volání procedury, nepřímá adresace, přiřazení hodnoty pro různé programovací jazyky)

Vlivy na provádění Benchmarku

Uvedené dřívější metody mohly být ovlivněny návrhem a architekturou kompilátoru a počítač se jevil v těchto měřeních rychlejší než v reálných aplikacích. Přesto je Dhrystone stále nejpoužívanější Benchmark pro embedded procesory.

Nevhodné nastavení příznaků překladu testovacího programu mohou způsobit nedovolené překlady a zpomalení výkonnosti. (požadavek - jeden překladač a jedna sada příznaků pro všechny programy ve stejném jazyce)

Vlivy na provádění Benchmarku

Vliv modifikace zdrojového kódu, existují tři přístupy:

- nejsou povoleny žádné úpravy zdrojového kódu
- úpravy zdrojového kódu jsou povoleny, ale jsou v podstatě neuskutečnitelné. Např. databázové benchmarky se spoléhají na standardní databázové programy, které představují desítky miliónů řádků kódu. Je vysoce nepravděpodobné, že by databázové společnosti chtěly zvýšit výkonnost jednoho konkrétního počítače.
- Změny zdrojového kódu jsou povoleny, pokud upravená verze produkuje stejný výstup.

Struktura Benchmarku

Při měření výkonosti počítače existuje široká škála tříd charakteristických pro různé typy úloh. Benchmarky lze proto sestavit z jednotlivých typových oblastí do testovací aplikace nazvané **suita**, jejichž cílem je vytvořit testovací program s podobným chováním jako program, který zákazník pravděpodobně spustí.

SPEC (Standard Performance Evaluation Corporation) – úspěšně využívaná Benchmarková sada pro mnoho aplikačních tříd (www.spec.org).

Tento standard vyvinul řadu referenčních hodnot pro počítače s operačním systémem Windows.

Desktop Benchmark

Benchmarky vychází z reálného programu modifikovaného pro možnost přenosu na jinou architekturu a minimalizovaný na vliv I/O obvodů.

Processor-intensive Benchmarks, sada SPEC CPU:

- Benchmarky v pevné řádové čárce obsahují programy od C kompilace přes šachový program až po simulaci kvantových počítačů.
- Floating-point benchmarky obsahují programy zahrnující výpočty strukturované mřížky, částicové nebo molekulární modelování (např. proudění tekutých látek)

Sada je určena jak pro benchmarking procesorů tak pro desktopové systémy, ale i pro servery s jedním procesorem.

Server Benchmark

Poněvadž je mnoho funkcí serverových služeb, je rovněž mnoho typů Benchmarků.

- Benchmark orientovaný na propustnost, např. spuštění více kopií SPEC CPU na multiprocesoru (SPECrate).
- Benchmark zahrnující I/O operace vznikající při síťovém a diskovém provozu, např. file server, web server, databáze, …(SPECWeb, SPECSFS, SPECjbb, SPECvirt).
- Benchmark Transaction-processing (TP) měří schopnost systému řešit transakce při přístupu k databázi a její aktualizaci. (www.tpc.org)

Výsledky Benchmarku

- Základním principem vykazování výsledků výkonu je jejich reprodukovatelnost (uvedení podrobného popisu počítače a nastavení podmínek kompilace).
- Výsledek vychází z velkého počtu ohodnocení výkonu napříč testovací sadou, aby mohl být věrohodný.
- Dílčí výsledky Benchmarkových testů dané sady jsou sumarizovány do jednoho čísla. Je případně možné přidat váhový faktor na jednotlivé dílčí výsledky.

Hodnocení výkonu počítače I.

- SPECRatio – poměr času provedení Benchmarkové úlohy na referenčním počítači k času na sledovaném počítači A.

$$SPECRatio_A = \frac{\text{Čas vykonání}_{reference}}{\text{Čas vykonání}_A}$$

- Porovnání výkonnosti počítačů A, B:

$$\frac{Výkonnost_A}{Výkonnost_B} = \frac{\frac{\text{Čas vykonání}_{reference}}{\text{Čas vykonání}_A}}{\frac{\text{Čas vykonání}_{reference}}{\text{Čas vykonání}_B}} = \frac{\text{Čas vykonání}_B}{\text{Čas vykonání}_A}$$

Hodnocení výkonu počítače II.

- Určení průměrné hodnoty SPECRatio pomocí geometrického průměru z jednotlivých *n* vzorků:

$$geometrický\ průměr = \sqrt[n]{\prod_{i=1}^{n} vzorek_i}$$

$vzorek_i$ je SPECRatio pro program *i*

- Porovnání výkonnosti počítače A, B:

$$\frac{geometrický\ průměr\ výkonnosti_A}{geometrický\ průměr\ výkonnosti_B} = \sqrt[n]{\prod_{i=1}^{n} \frac{Výkonnost_{A_i}}{Výkonnost_{B_i}}}$$

Využití paralelismu

- Nejdůležitější metoda zvyšování výkonu
- Využití paralelismu na systémové úrovni – server Benchmark (SPECWeb – paralelní žádosti, TPC-C – paralelní vláknové zpracování, multiprocesor, více disků – diskové pole). Navýšení kapacity paměti, počtu procesorů a disků se nazývá škálovatelnost (scalability).
- Na úrovni samotných procesorů je využívání paralelnosti mezi instrukcemi rozhodující pro dosažení vysokého výkonu - pipelining
- Využití vícecestných vyrovnávacích pamětí s možností paralelního vyhledávání ve více paměťových blocích.
- Využití paralelní sčítačky v ALU

Pipelining – zřetězení instrukcí

IF = Instruction Fetch

ID = Instruction Decode

EX = Execute

MEM = Memory access

WB = Register write back

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

Podmínkou pro využití zřetězení instrukcí je, že instrukce na sobě závislé mají stejný počet hodinových cyklů a probíhá sekvenční zpracování bez skoků.

Princip lokality

- Charakteristickou vlastností programů je **opětné využití** dat a instrukcí (např. cykly).
- Důsledkem lokality je to, že můžeme **předvídat** s rozumnou přesností, jaké instrukce a program bude v blízké budoucnosti využívat na základě svých přístupů v nedávné minulosti.
- Zásada lokality se vztahuje také na **přístup k datům**, i když ne tak silně jako **k instrukcím**.
- **Dočasná lokalita** – opětovný přístup k nedávno použitým položkám
- **Prostorová lokalita** – položky s blízkou adresou jsou volány v blízkém časovém sledu

Principy optimalizace výkonu

- Při vytváření návrhového kompromisu je **upřednostňována optimalizace časté úlohy před občasnou úlohou**. Jedná se o vytížení systémových zdrojů, kde dopad zlepšení je vyšší, pokud je výskyt častý.
- Zaměření na **společné využití energii, stejně jako na alokaci zdrojů a výkonu.** Jednotka načítání a dekódování instrukcí procesoru je **využívána nejčastěji**, proto se nejprve optimalizuje její činnost.
- **Častý případ** je také často **jednodušší** a může se **provádět rychleji**. Např. při sčítání dvou čísel můžeme očekávat, že přetečení nebude časté a optimalizovat běžnější případ bez přetečení. Obezřetnostní se výkon pro normální případ optimalizací zlepší.
- Pro kvantifikaci tohoto principu lze použít Amdahlův zákon.

Amdahlův zákon I.

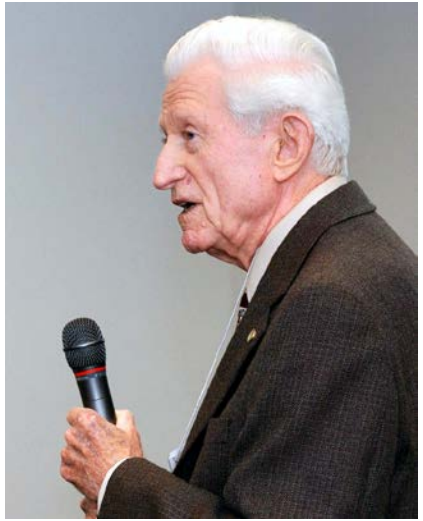
- Zvýšení výkonnosti dosažitelné nějakým zlepšením je omezené mírou používání tohoto zlepšení

Definuje zvýšení výkonnosti S (Speedup) jako poměr výkonnosti při použití zlepšení ku výkonnosti bez použití zlepšení:

$$S = \frac{výkonnost\ při\ použití\ zlepšení}{výkonnost\ bez\ použití\ zlepšení}$$

Nebo také vyjádření pomocí času vykonání výpočtu:

$$S = \frac{čas\ výpočtu\ bez\ použití\ zlepšení}{čas\ výpočtu\ při\ použití\ zlepšení}$$



Amdahlův zákon II.

- Pro výpočet celkového zlepšení zavedeme dílčí proměnné: F_E (Fraction$_{enhanced}$), S_E (Speedup$_{enhanced}$)

$$F_E = \frac{\text{původní čas výpočtu zlepšené části}}{\text{původní celkový čas výpočtu}} \leq 1$$

$$S_E = \frac{\text{původní čas výpočtu zlepšené části}}{\text{čas výpočtu zlepšené části úlohy}} > 1$$

- Doba výpočtu po zlepšení:

$$T_{po\ zlepšení} = T_{před\ zlepšením} * \left((1 - F_E) + \frac{F_E}{S_E} \right)$$

Amdahlův zákon a jeho důsledky

$$S_{celkové} = \frac{T_{před\ zlepšením}}{T_{po\ zlepšení}} = \frac{1}{(1-F_E)+\frac{F_E}{S_E}}$$

Shrnutí

- Optimalizace má největší užitek pro nejčastější případy
- Nejčastější případy jsou často mnohem jednodušší než speciální případy, takže se lépe optimalizují
- I výrazné optimalizace speciálních případů mohou přinést menší užitek než drobné optimalizace nejčastějších případů

Špatná měřítka výkonnosti I.

- Použití podmnožiny faktorů (taktovací frekvence, CPI, počet instrukcí) jako samostatné metriky
- Použit jen jednoho faktoru je téměř vždy špatně
- Použit dvou faktorů může být ve specifických případech v pořádku, ale často tomu tak není (mezi jednotlivými faktory může existovat závislost)
- Jiné metriky, které jsou jen převlečené známé faktory

Špatná měřítka výkonnosti II.

MIPS (Million Instructions Per Second) = $\frac{počet\ instrukcí}{10^6 \times čas\ instrukce}$

- Rychlost vykonávání instrukcí
- Intuitivní (čím vyšší, tím rychlejší)

Problémy:

- Nebere v úvahu možnosti instrukcí, dobu vykonávání jednotlivých instrukcí atd. Nelze porovnávat počítače s různou instrukční sadou
- Liší se podle konkrétního instrukčního mixu daného programu (jedna hodnota nereprezentuje výkon počítače)

Výkonnost procesoru

- Hodinový takt procesoru – nejmenší oddělená časová událost v procesoru vyjádřená délkou periody (např. 1ns) nebo frekvencí (např. 1 GHz).
- CPU time - čas pro zpracování programu procesorem se dá vyjádřit:

$$CPU\ time = počet\ hodinových\ cyklů\ v\ programu\ x\ délka\ periody\ hodinového\ cyklu$$

nebo

$$CPU\ time = \frac{počet\ hodinových\ cyklů\ v\ programu}{taktovací\ frekvence}$$

Výkonnost procesoru II.

- Z počtu provedených instrukcí – IC (instruction count) a počtu hodinových cyklů je možné vyjádřit průměrný počet hodinových cyklů na provedení jedné instrukce – CPI (clock cycles per instruction).

$$CPI = \frac{počet\ hodinových\ cyklů\ v\ programu}{počet\ instrukcí}$$

- V případě více procesorového systému se uvádí počet provedených instrukcí v jednom hodinovém cyklu - IPC (instruction per clock cycle)
- Toto obecné měřítko umožňuje použití pro různé instrukční sady

Výkonnost procesoru III.

- Čas zpracování programu CPU time můžeme také vyjádřit :

$$CPU\ time = IC\ x\ CPI\ x\ délka\ periody\ hodinového\ cyklu$$

- Výkon procesoru závisí na třech charakteristických parametrech:
  - Doba trvání hodinového cyklu
  - Počet hodinových cyklů na instrukci
  - Počet instrukcí v programu

$$CPU\ time = \frac{počet\ sekund}{program} = \frac{počet\ instrukcí}{program} \times \frac{počet\ hodinových\ cyklů}{instrukce} \times \frac{počet\ sekund}{hodinový\ cyklus}$$

Výkonnost procesoru IV.

- Celkový počet hodinových cyklů procesoru při provádění programu:

$$počet\ hodinových\ cyklů\ v\ programu = \sum_{i=1}^{n} IC_i \times CPI_i$$

IC_i průměrný počet typů instrukcí *i* provedených v programu

CPI_i průměrný počet hodinových cyklů instrukce typu *i*

n počet typů instrukcí použitých v programu

Výkonnost procesoru V.

- Celkový čas provádění programu:

$$CPU\ time = (\sum_{i=1}^{n} IC_i \times CPI_i) \times \textit{doba trvání hodinového cyklu}$$

- Průměrný počet hodinových cyklů na provedení jedné instrukce:

$$CPI = \frac{\sum_{i=1}^{n} IC_i \times CPI_i}{\textit{počet instrukcí}} = \sum_{i=1}^{n} \frac{IC_i}{\textit{počet instrukcí}} \times CPI_i$$

Hodnota CPI_i by měla být měřena a ne pouze vypočtena z tabulky, protože musí obsahovat efekty pipeline, cache misses a ostatní vlastnosti paměťového systému.