

# Úvod do organizace počítače

---

Měření výkonnosti počítačů,  
zkušební úlohy  
(benchmarks)

# Přehled

---

- Vyhodnocení výkonu
- Omezení
- Metriky
- Rovnice pro výkon procesoru
- Reporty o vyhodnocení výkonu
- Amdahlův zákon

# Přehled (pokr.)

---

- Zkušební úlohy
- Populární zkušební úlohy
  - Linpack
  - Intelský iCOMP
- SPEC
- Chyby a omyly

# Co to je „výkon“?

---

- Uvažujme tři automobily. Porsche Boxster S, Honda Civic s hybridním pohonem a Ford E450 – velkoprostorový vůz
  - Boxster S má největší cestovní rychlost.
  - Honda Civic má největší dojezd.
  - E450 má největší kapacitu.
- Nyní budeme definovat výkon pomocí rychlosti.
  - Chcete-li dopravit jednoho cestujícího z jednoho místa do druhého, největší výkon má Boxster S.
  - Chcete-li dopravit 15 cestujících, největší výkon má E450.
- Co je to vlastně výkon?

# Jak definovat výkon?

---

| Letadlo             | Kapacita<br>[počet<br>pasažerů] | Dolet<br>[míle] | Cestovní<br>rychlost<br>[m.p.h.] | "Propustnost"<br>pasažerů<br>[počet pasažerů x<br>m.p.h.] |
|---------------------|---------------------------------|-----------------|----------------------------------|---|
| Boeing 777          | 370                             | 4630            | 610                              | 228,750   |
| Boeing 747          | 470                             | 4150            | 610                              | 286,700   |
| Concord             | 132                             | 4000            | 1350                             | 178,200   |
| Douglas DC-<br>8-50 | 146                             | 8720            | 544                              | 79,424  |

# Dvě klíčové metriky výkonu

---

- Doba běhu úlohy
  - doba výpočtu, doba odezvy, spotřebovaný čas, latence
- Počet úloh za jednotku času
  - rychlost výpočtu, šířka pásma, propustnost

| Letadlo    | Washington D.C.<br>Paris | Rychlost | Cestující | Propustnost<br>(#cestujících x mph) |
|------------|--------------------------|----------|-----------|-------------------------------------|
| Boeing 747 | 6.5 hodiny               | 610mph   | 470       | 286,700                             |
| Concord    | 3 hodiny                 | 1350mph  | 132       | 178,200                             |

# Latence versus propustnost

---

- Latence

- “realný” čas potřebný pro vykonání úlohy
- důležité, pokud se soustředujeme na **jednotlivou úlohu**
  - uživatel, který zpracovává jedinou aplikaci
  - kritická úloha ve vestavných systémech reálného času

- Propustnost (šířka pásma)

- # úloh, vykonaných za jednotku času
- metrika nezávislá na přesném počtu provedených úloh
- důležité, jestliže se soustředujeme na běh **mnoha úloh**
  - manažera datového centra zajímá hlavně celkový objem práce, vykonané během dané doby

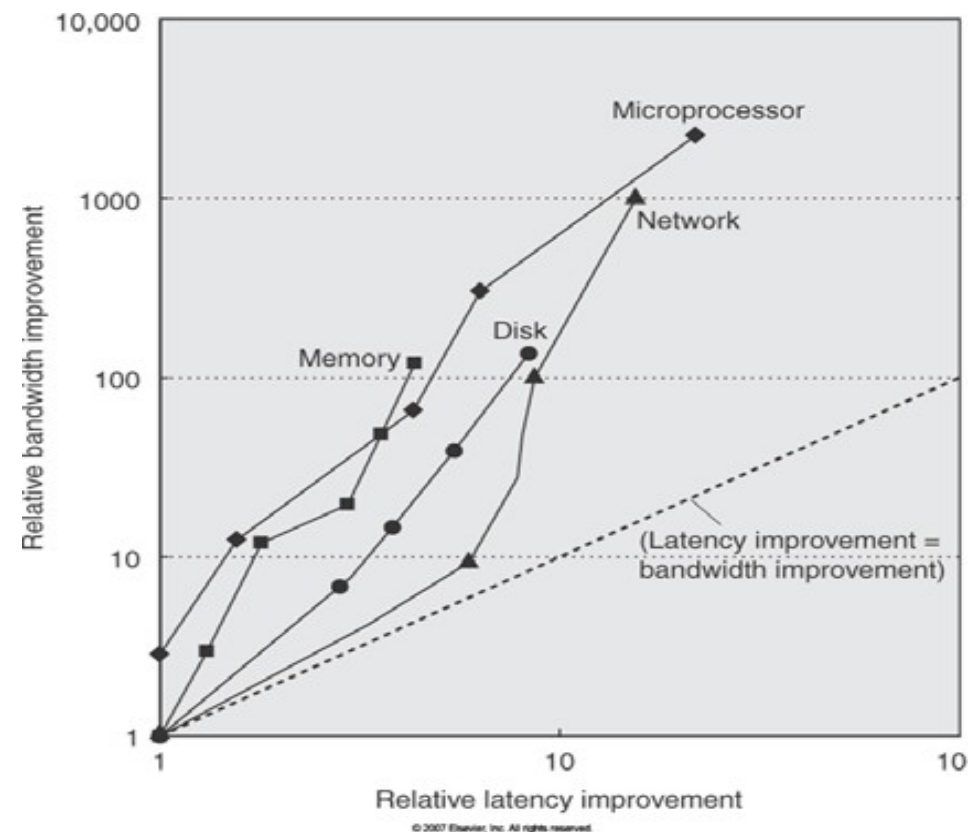
# Latence zaostává za šířkou pásma

- Šířka pásma „přeběhla“ latenci ve všech hlavních počítačových technologiích

*„Existuje staré rčení:*

- Bandwidth problems can be cured with money. Latency problems are harder because the speed of light is fixed—you can't bribe God.”*

*[Anonymous]*



Obrázek vyjadřuje **poměrné** zlepšování latence a propustnosti



# Výkon počítače

---

- Hlavním kritériem uživatele může být *doba odezvy* (nebo *doba výpočtu*) – kolik času uplyne od startu do ukončení úlohy?
  - Důležité pro jednotlivé uživatele.
- Šéf výpočetního střediska se naopak bude asi zajímat nejvíce o *propustnost* – kolik celkové práce vykoná počítač za určitou dobu?
  - Důležité pro šéfa výpočetního centra.
- Co je tedy vlastně výkon?
- Potřebujeme prostředky pro měření výkonu výpočetních prostředků od systémů třídy „*embedded*“, přes *stolní počítače*, až po *výkonné servery*.

# Doba výpočtu

---

- Nejjednodušší měření výkonu se zakládá na měření *doby výpočtu* (*execution time* nebo jinak *wall-clock time*).
- Ta se vztahuje na celkové množství času, potřebného k provedení úlohy včetně přístupů na disk, přístupů do paměti, režie operačního systému, I/O aktivit, doby aktivity CPU a dalších složek.
- Odstartujeme-li úlohu v 7:20 a úloha skončí v 7:26, je doba výpočtu 6 minut.

# Hodnocení systémů se sdílením času?

---

- Většina moderních počítačů dnes pracuje v režimu sdílení času - *timesharing* (uživatel zpracovává současně více úloh).
- Systém je většinou navržen tak, aby se maximalizovala propustnost. Nesleduje se tedy minimální doba výpočtu jednotlivého programu.
- *Doba CPU* – doba, kterou procesor věnuje zpracování jedné úlohy a nezahrnuje I/O aktivity, ani čas, který procesor věnuje zpracování jiných úloh.
- *Doba CPU* se dále dělí na *dobu CPU uživatele* a *dobu CPU systému*. Doba CPU uživatele je čas procesoru, věnovaný jen dané úloze, doba CPU systému se vztahuje na dobu, kdy procesor provádí akce operačního systému, vázající se na provádění uživatelské úlohy.

# Hodinové cykly

---

- Většina moderních počítačů používá hodinový signál s konstantní periodou. Perioda hodin je diskrétní interval, během něhož se v HW odehraje nějaký elementární krok.
- Frekvence hodin se udává v MHz nebo GHz.
- *Frekvence hodin* a *perioda hodin* jsou inverzní veličiny – jestliže systém používá hodiny s frekvencí 2.5 GHz, perioda hodin činí 400 ps.
- Nyní se můžeme pokusit vymezit pojem *výkon CPU*.

# Koncepce hodnocení výkonnosti

- Hodnocení výkonu počítače je založeno na:
  - Propustnosti
  - Době výpočtu (*execution time, elapsed time*)
- Hodnocení výkonu komponent a vrstev
- Vyhodnocení výkonu procesoru
  - Založeno na **době výpočtu** (*execution time*) programu:

Doba od startu do ukončení.  
(nezapočítává se doba I/O a zpracování jiných programů)

$$\text{výkon}_x = 1 / \text{doba výpočtu}_x$$

Relativní výkon:  $n = \text{výkon}_x / \text{výkon}_y$

$n > 1 \Rightarrow X$  je  $n$  krát **rychlejší než**  $Y$

- Terminologie
  - **Zlepšit** výkon: = **zvýšit** výkon
  - **Zlepšit** dobu výpočtu: = **zkrátit** dobu výpočtu

# Příklad

---

- Důraz na propustnost a dobu odezvy:
  - Použití rychlejšího procesoru
    - *Snížení doby odezvy*
    - *Vyšší propustnost*
  - Zvýšení počtu procesorů v systému
    - Vyšší propustnost
    - Snížení doby odezvy (*pokud je nízká režie*)
      - U masivně paralelních procesorů (MPP)
      - U symetrických „multiprocessorových“ procesorů (SMP)
        - » Pouze když další procesory redukují čas čekání ve frontách

# Měření výkonnosti

---

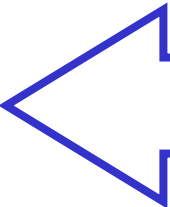
Složky doby výpočtu programu:

1. Čas práce CPU
  - čas CPU věnovaný uživatelskému programu
  - čas CPU spotřebovaný operačním systémem
2. Čas pro I/O operace
3. Čas věnovaný výpočtu jiných programů

Příkaz Unixu **time**

```
time cc prog.c
```

```
9.7u 1.5s 20 56%
```



9.7u = 9.7 sec User CPU time  
1.5s = 1.5 sec System CPU time  
20 = Total Elapsed Time  
56% = Percent CPU time

# Výpočet výkonu CPU

---

- **CPU time** = počet cyklů CPU \* doba cyklu  
= počet cyklů CPU / frekvence hodin
  - počet cyklů CPU = IC \* CPI
    - IC: počet instrukcí (počet instrukcí na program)
    - CPI: střední hodnota počtu cyklů na instrukci

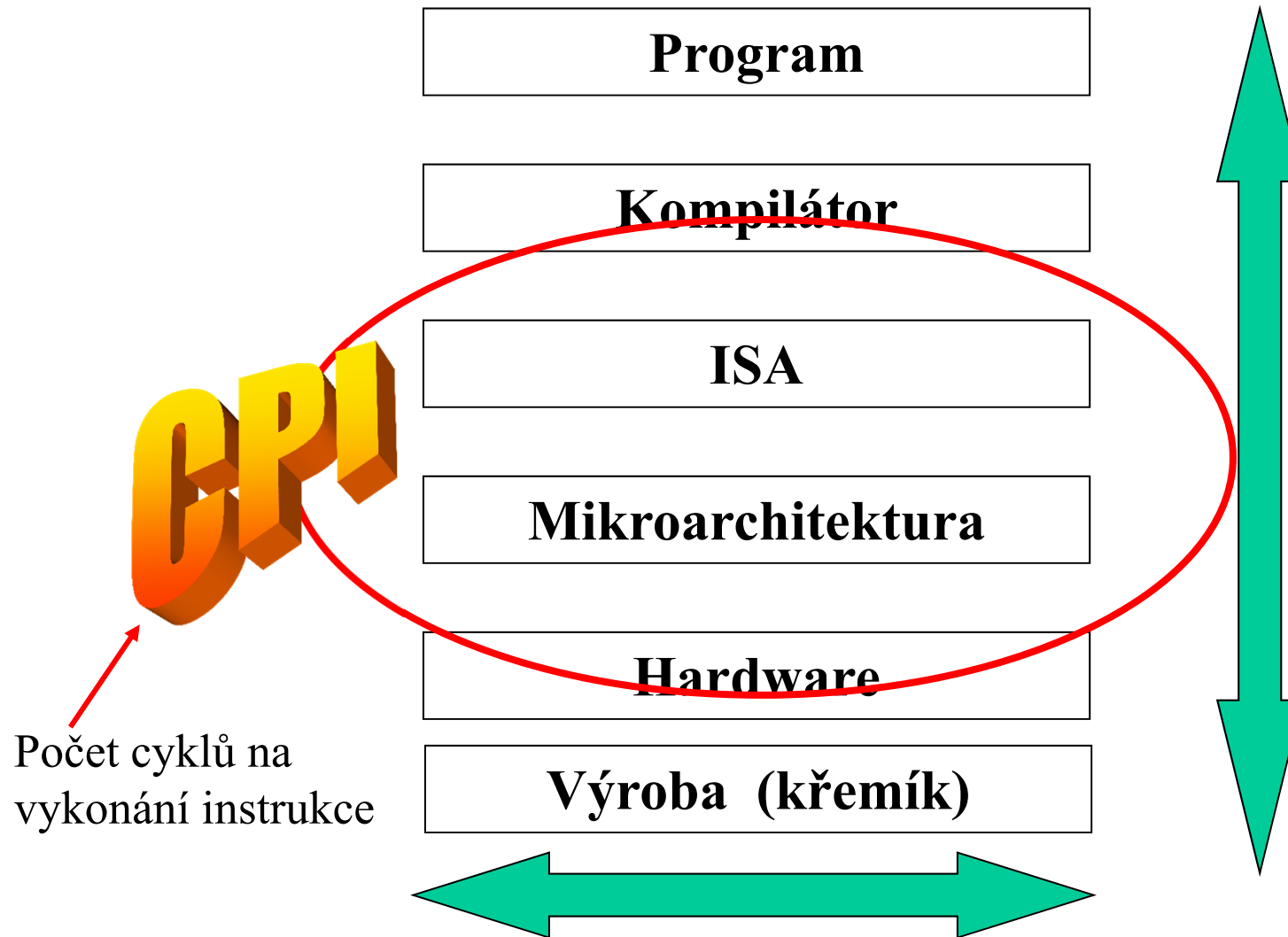
- **CPU time = IC \* CPI \* doba cyklu**

$$\frac{\text{sekundy}}{\text{program}} = \frac{\text{instrukce}}{\text{program}} * \frac{\text{cykly hodin}}{\text{instrukce}} * \frac{\text{sekundy}}{\text{cykly hodin}}$$

- **Cykly hodin CPU** =  $\sum_i (\text{CPI}_i * \text{IC}_i)$ 
  - $\text{IC}_i$ : počet instrukcí třídy  $i$
  - $\text{CPI}_i$ : cykly, které jsou třeba k provedení instrukcí třídy  $i$

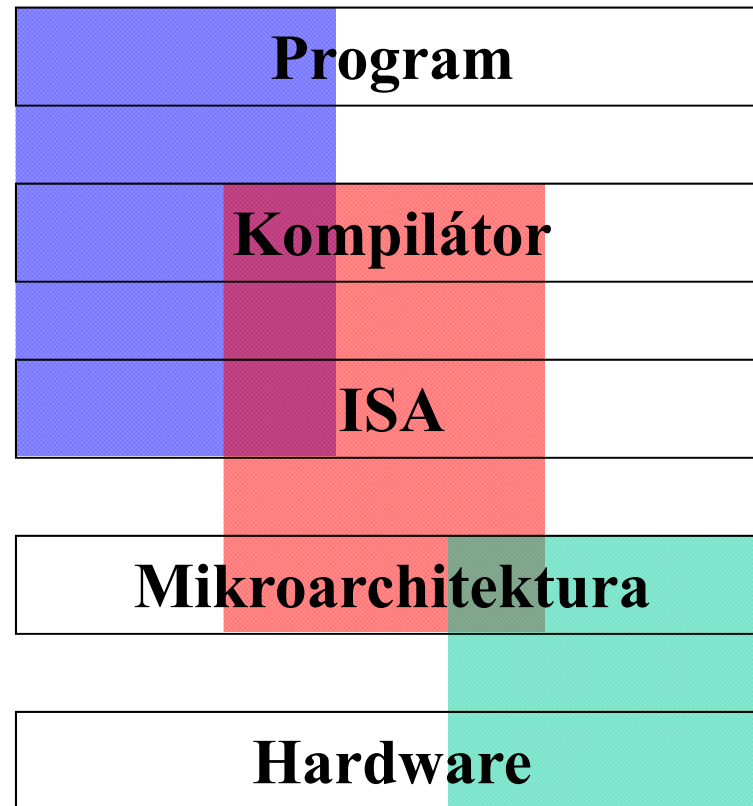


# Co ovlivňuje CPI



# Vliv jednotlivých složek na výkon

$$\text{CPU time} = \text{IC} * \text{CPI} * \text{Cycle time}$$



**vzájemná závislost úrovní**

# Měření uvedených parametrů

---

- Dobu CPU lze získat provedením programu (moderní operační systémy tyto statistiky zaznamenávají).
- Hodinová frekvence je snadno dostupný parametr.
- Počet instrukcí se nezískává snadno. Tyto statistiky se získávají použitím simulátorů, emulátorů nebo přímo HW čítači v procesoru (HW ladicí prostředky). Protože počet instrukcí závisí na architektuře a nikoliv na vnitřní organizaci nižších úrovní, může být měřen nezávisle na implementaci (počet instrukcí je stejný jak pro Pentium, tak pro Pentium 4 - pro stejný program).

# Měření uvedených parametrů (pokr.)

---

- Nejobtížněji se získává parametr CPI, který je závislý na konkrétní implementaci.
- CPI je ovlivňován všemi volbami při návrhu organizace počítače, včetně paměťového systému a dalších faktorů. Například určitá instrukce může být provedena během 27 period hodin na Pentiu, ale během 9 period hodin na Pentiu 4.
- CPI také závisí na typu instrukčního mixu (program s 80% 1- cyklových instrukcí bude mít nižší CPI než program s 80% 9 - cyklových instrukcí).

# Příklad

---

- Program proběhne za 10 sekund na 2 GHz procesoru. Návrhář chce postavit počítač, který provede stejný program za 6 sekund zvýšením hodinové frekvence.
- Vlivem změny časových poměrů ale naroste také střední hodnota CPI 1.2 krát.
- Jaká frekvence hodin má být použita ?

$$\frac{10}{6} = \frac{IC * CPI / (2 \text{ GHz})}{IC * 1.2 \text{ CPI} / (X \text{ GHz})} \quad \begin{array}{l} \text{(stávající doba výpočtu)} \\ \text{(cílová doba výpočtu)} \end{array}$$

Řešením pro  $X$  získáme  **$X = 4 \text{ GHz}$**

# Třídy instrukcí

---

- Instrukce lze s určitým zjednodušením rozdělit do *instrukčních tříd* – instrukce se společnými výkonnostními charakteristikami (počtem cyklů/instrukci).
- Například instrukce, které přistupují do paměti obvykle vyžadují větší počet cyklů než jednoduchá instrukce součtu.
- Jestliže pro určitý procesor existuje  $n$  instrukčních tříd, potom můžeme určit počet cyklů hodin, které vyžaduje zpracování specifického programu ...

$$\# \text{ hodinových cyklů CPU} = \sum_{i=1}^n (CPI_i \times C_i)$$

|         |   |
|---------|---|
| $C_i$   | # instrukcí třídy $i$ , které byly provedeny      |
| $CPI_i$ | střední počet cyklů na instrukci (pro třídu $i$ ) |
| $N$     | # instrukčních tříd                               |

# Příklad – třídy instrukcí

Porovnání dvou kódových sekvencí kompilátoru

| Třída instrukcí $i$ | $CPI_i$ pro třídu instrukcí $i$ |
|---------------------|---------------------------------|
| A                   | 1                               |
| B                   | 2                               |
| C                   | 3                               |

| Kódová<br>sekvence | Počet instrukcí ( $IC_i$ ) pro třídu instrukcí |   |   |
|--------------------|--|---|---|
|                    | A  | B | C |
| 1                  | 2  | 1 | 2 |
| 2                  | 4  | 1 | 1 |

- Která kódová sekvence provede největší počet instrukcí?
- Která je rychlejší?  $S_1 = 2 \cdot 1 + 1 \cdot 2 + 2 \cdot 3 = 10$

$$S_2 = 4 \cdot 1 + 1 \cdot 2 + 1 \cdot 3 = 9$$

Druhá sekvence instrukcí je rychlejší, i když jich obsahuje více.

# Co určuje výkon CPU

---

CPU time = počet instrukcí x CPI x doba\_cyklu

|                    | počet instrukcí | CPI | doba_cyklu |
|--------------------|-----------------|-----|------------|
| Algoritmus         |                 |     |            |
| Programovací jazyk |                 |     |            |
| Kompilátor         |                 |     |            |
| ISA                |                 |     |            |
| Organizace jádra   |                 |     |            |
| Technologie        |                 |     |            |



# Co určuje výkon CPU

---

CPU time = počet instrukcí x CPI x doba\_cyklu

|                    | počet instrukcí | CPI | doba_cyklu |
|--------------------|-----------------|-----|------------|
| Algoritmus         | X               | X   |            |
| Programovací jazyk | X               | X   |            |
| Kompilátor         | X               | X   |            |
| ISA                | X               | X   | X          |
| Organizace jádra   |                 | X   | X          |
| Technologie        |                 |     | X          |

# Příklad

| Operace | Četnost | CPI <sub>i</sub> | Četnost x CPI <sub>i</sub> |
|---------|---------|------------------|----------------------------|
| ALU     | 50%     | 1                |                            |
| Load    | 20%     | 5                |                            |
| Store   | 10%     | 3                |                            |
| Branch  | 20%     | 2                |                            |
|         |         |                  | Σ =                        |

- Jak se zrychlí počítač, zredukuje-li rychlejší cache dobu cyklu na dva takty?
- Jakého výsledku dosáhneme využitím predikce, která zkrátí cykl skoků?
- Co kdyby bylo možno vykonat dvě ALU instrukce současně?

# Příklad

| Operace | Četnost | CPI <sub>i</sub> | Četnost x CPI <sub>i</sub> |     |     |      |
|---------|---------|------------------|----------------------------|-----|-----|------|
| ALU     | 50%     | 1                | .5                         | .5  | .5  | .25  |
| Load    | 20%     | 5                | 1.0                        | .4  | 1.0 | 1.0  |
| Store   | 10%     | 3                | .3                         | .3  | .3  | .3   |
| Branch  | 20%     | 2                | .4                         | .4  | .2  | .4   |
| Σ =     |         |                  | 2.2                        | 1.6 | 2.0 | 1.95 |

- Jak se zrychlí počítač, zredukuje-li rychlejší cache dobu cyklu na dva takty?

CPU time new = 1.6 x IC x CC proto 2.2/1.6 dává o 37.5% vyšší rychlost

- Jakého výsledku dosáhneme využitím predikce, která zkrátí cykl skoků na polovinu?

CPU time new = 2.0 x IC x CC proto 2.2/2.0 dává o 10% vyšší rychlost

- Co kdyby bylo možno vykonat dvě ALU instrukce současně?

CPU time new = 1.95 x IC x CC pak 2.2/1.95 a dostaneme o 12.8% vyšší rychlost

# MIPS

---

- Kdykoliv se jednalo o měření výkonu počítače, bylo snahou použít čas jako měřítko výkonu. Velmi často to však vedlo ke špatným výsledkům a chybné interpretaci.
- Nejpopulárnějším měřítkem je počet instrukcí za jednotku času (měřeno v MIPS).
- MIPS je velmi jednoduchá koncepce ... (příliš jednoduchá!)

$$MIPS = \frac{Počet\_instrukcí}{Doba\_výpočtu \times 10^6}$$

# Hlavní problémy s měřítkem „MIPS“

---

- MIPS je jednoduché měřítko – rychlejší stroje mají vyšší ohodnocení MIPS.
- Existují tři hlavní problémy s MIPS jako měřítkem pro hodnocení výkonu ...
  - měřítko MIPS nezahrnuje výkonnost instrukcí. Nelze tedy porovnávat počítače s rozdílným instrukčním souborem.
  - měřítko MIPS se mění i u jednoho počítače (různé instrukce mají různý CPI).
  - měřítko MIPS se dokonce může měnit inverzně vzhledem k výkonu.

# MIPS - příklad

- Necht' má počítač následující třídy instrukcí...

| Třída instrukcí | CPI pro tuto třídu instrukcí |
|-----------------|------------------------------|
| Třída A         | 1 cykl / instrukci           |
| Třída B         | 2 cykly / instrukci          |
| Třída C         | 3 cykly / instrukci          |

- Program bude přeložen dvěma kompilátory:
  - **Kompilátor 1:** 5 instrukcí A, 1 instrukci B a 1 instrukci C
  - **Kompilátor 2:** 10 instrukcí A, 1 instrukci B a 1 instrukci C
- Celkový počet cyklů pro každou sekvenci...
  - $\text{Cykly}_1 = (5 \times 1 + 1 \times 2 + 1 \times 3) \times 10^9 = 10 \times 10^9$  cyklů
  - $\text{Cykly}_2 = (10 \times 1 + 1 \times 2 + 1 \times 3) \times 10^9 = 15 \times 10^9$  cyklů
- Předpokládejme, že počítač běží na 1 GHz, kód kompilátoru 1 zabere 10 sekund, kód kompilátoru 2 zabere 15 sekund. (Počet cyklů pro každou sekvenci instrukcí, dělený počtem cyklů za sekundu.)

# MIPS – příklad (pokr.)

---

- Nyní můžeme počítat MIPS pro každou sekvenci instrukcí. MIPS určíme...

$$MIPS = \frac{Počet\_instrukcí}{Doba\_výpočtu \times 10^6}$$

- Pro každou posloupnost dostaneme ...

$$MIPS_1 = \frac{(5+1+1) \times 10^9}{10 \times 10^6} = 700 \quad MIPS_2 = \frac{(10+1+1) \times 10^9}{15 \times 10^6} = 750$$

- To znamená, že kompilátor 2 má vyšší ohodnocení MIPS, ale prakticky se sekvence 1 provádí rychleji. To je hlavní problém s použitím MIPS jako metriky pro výkon.

# Hodnocení měřítka „MIPS“

---

- Porovnávání jablek s pomeranči
- Nedostatek: 1 MIPS jednoho procesoru neznamena stejnou práci jako 1 MIPS jiného
  - *Podobné určení vítěze běžeckého závodu podle toho, kdo udělá méně kroků*
  - Některé procesory mají FP v software (asi 1FP = 100 INT)
  - Různé instrukce trvají různě dlouho
- Vhodné pouze pro porovnání 2 procesorů stejného výrobce se stejnou ISA a se stejným kompilátorem. (např. Intel iCOMP benchmark)



# MFLOPS

---

- Jiným alternativním měřítkem k době výpočtu jsou **MFLOPS** (*million floating-point operations per second*)  
Tato metrika se určí ...

$$MFLOPS = \frac{\text{Počet FP operací v programu}}{\text{Doba výpočtu} \times 10^6}$$

- *Floating-point operace* - sem patří sčítání, odčítání násobení a dělení čísel v pohyblivé řádové čárce.
- Evidentně MFLOPS závisí na zpracovávaném programu. Například kompilátor (který skoro nepoužívá FP operace) by měl nulové ohodnocení na každém stroji.

# Problémy s MFLOPS

---

- Metrika MFLOPS je založena na aktuálních operacích, nikoliv na instrukcích. Poskytuje tedy věrnější hodnocení než MIPS.
- Přesto existují vážné problémy ...
  - ❖ Soubor FP operací se u různých strojů liší. Například Motorola 68882 má FP instrukce dělení, odmocniny, sinu a kosinu. Cray 2 je nemá. Motorola provádí operaci kosinus během jedné instrukce, kdežto Cray 2 jich musí udělat celou řadu.

# Problémy s MFLOPS

---

- ❖ Hodnocení pomocí MFLOPS je ovlivněno nejen poměrem FP operací k non-FP operacím, ale také poměrem „rychlých“ FP operací ku „pomalým“ FP operacím. Například program, obsahující 100% FP dělení bude prováděn pomaleji než program, který obsahuje 100% FP sčítání. Byl by hodnocen výš.
- ❖ To lze kompenzovat zavedením vah jednotlivých typů instrukcí do výpočtu podle jejich složitosti. Takové měřítko se pak nazývá *normalizované MFLOPS*.

# MIPS a MFLOPS

---

- Metriku MIPS nebo MFLOPS nelze zobecnit tak, aby se dala použít jako jediná pro hodnocení výkonu počítače.
  - Nejvyšší MIPS lze získat výběrem sekvence instrukcí, která minimalizuje dobu výpočtu (výběrem instrukcí, které minimalizují CPI).
  - Nejvyšší MFLOPS lze získat výběrem sekvence instrukcí, která bude obsahovat jen “rychlé” FP operace.
- Takové měřítko neříká o počítači nic, protože nikdo takový program nebude spouštět!

# Složky rovnice pro výkon CPU

---

- Doba cyklu hodin
  - Odhady časování a verifikace (kompletní návrh)
  - Cílová doba cyklu
- Počet instrukcí
  - Kompilátor
    - Simulátor instrukčního souboru
    - Monitorování (execution-based monitoring, *profiling*)
- Je-li uplatněn princip pipeline, potom parametr CPI je třeba určit s ohledem na hloubku „pipeline“.

# Amdahlův zákon

---

- Běžným problémem, který je spojen s měřením výkonu je myšlenka zlepšit určitý aspekt stroje, při čemž se očekává, že ve stejném poměru, ve kterém bude provedeno dílčí zlepšení se zvýší i celkový výkon stroje.
- Amdahlův zákon říká...

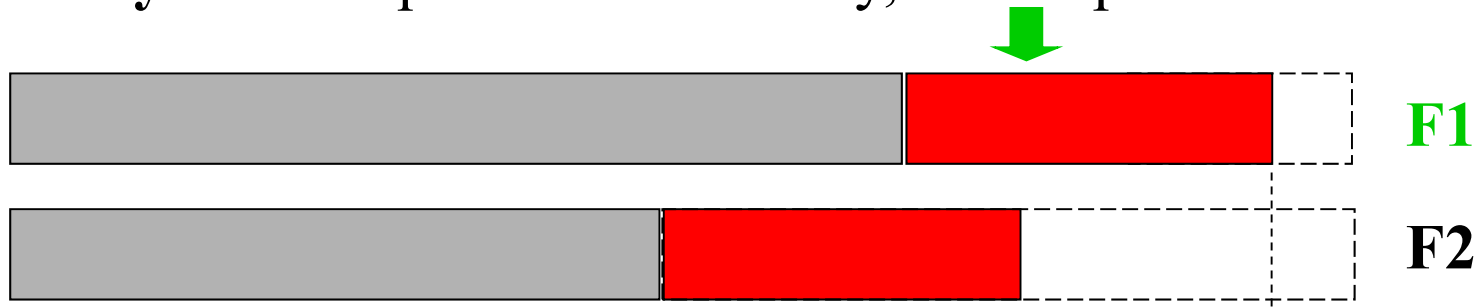
$$\begin{array}{ccccc} & & \text{Doba výpočtu ovlivněná} & & \\ & & \text{zlepšením} & & \\ \text{Doba výpočtu} & = & \frac{\text{-----}}{\text{Poměr zlepšení}} & + & \text{Doba výpočtu} \\ \text{po zlepšení} & & & & \text{neovlivněná} \end{array}$$

# Amdahlův zákon

- Zákon o klesajícím zisku (návratnosti investic)

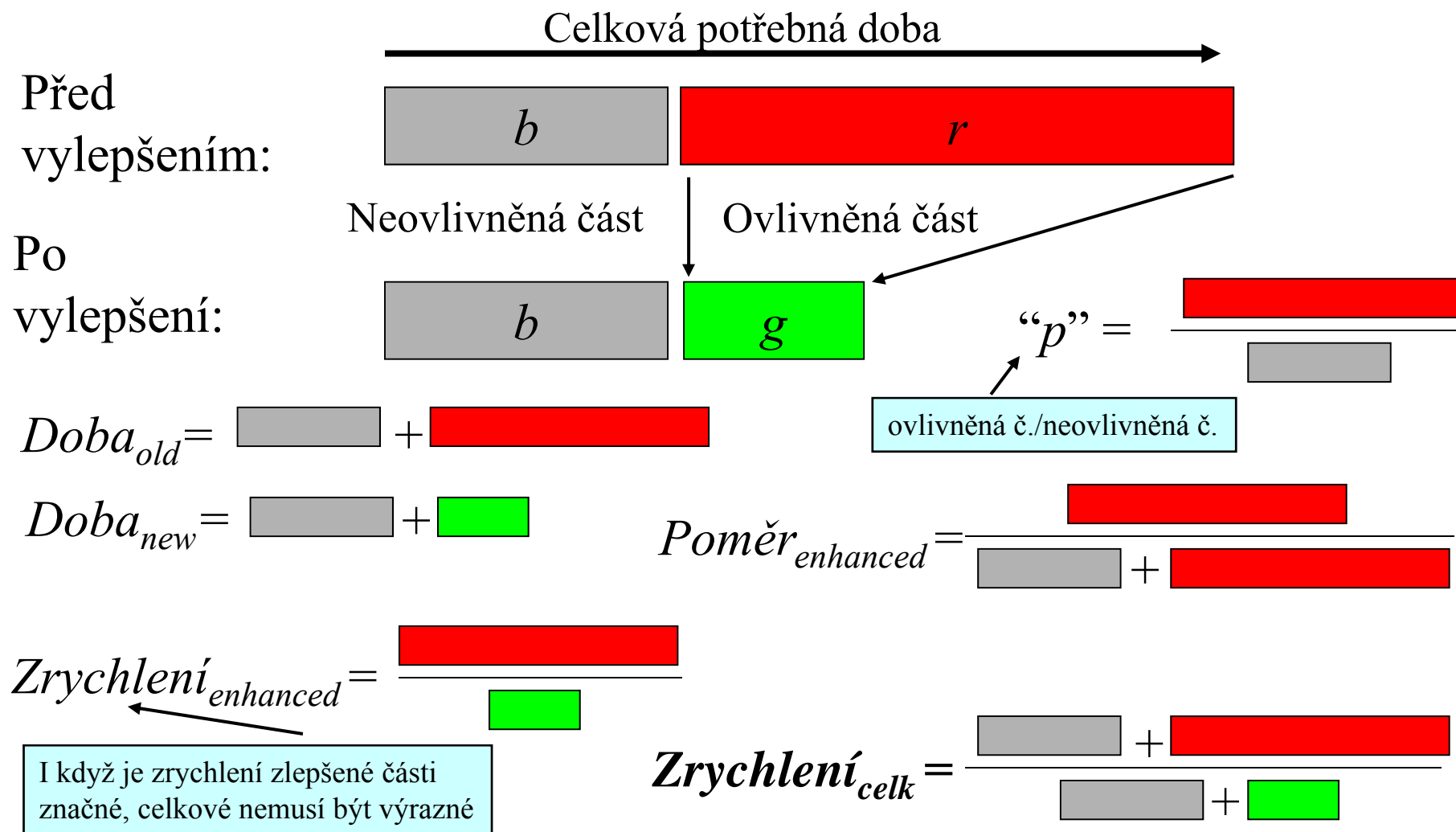
$$\text{Zrychlení} = \frac{\text{Doba výpočtu před zlepšením}}{\text{Doba výpočtu po zlepšení}}$$

**Příklady:** **F1**: zlepšení na 75% doby, **F2**: zlepšení na 50% doby



$$\text{Zrychlení} = \frac{1}{(1 - \text{zlepšená část}) + (\text{zlepšená část} / \text{koeficient zlepšení})}$$

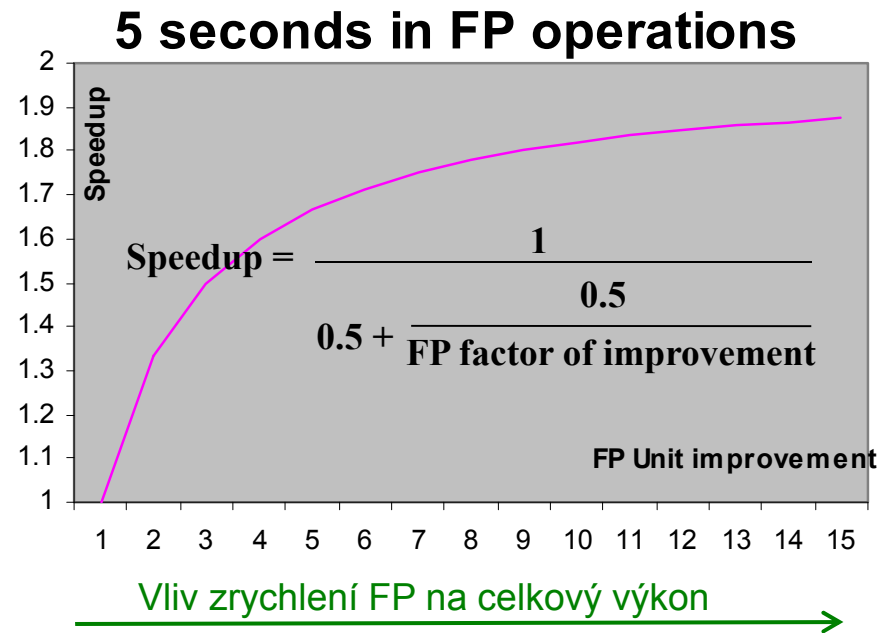
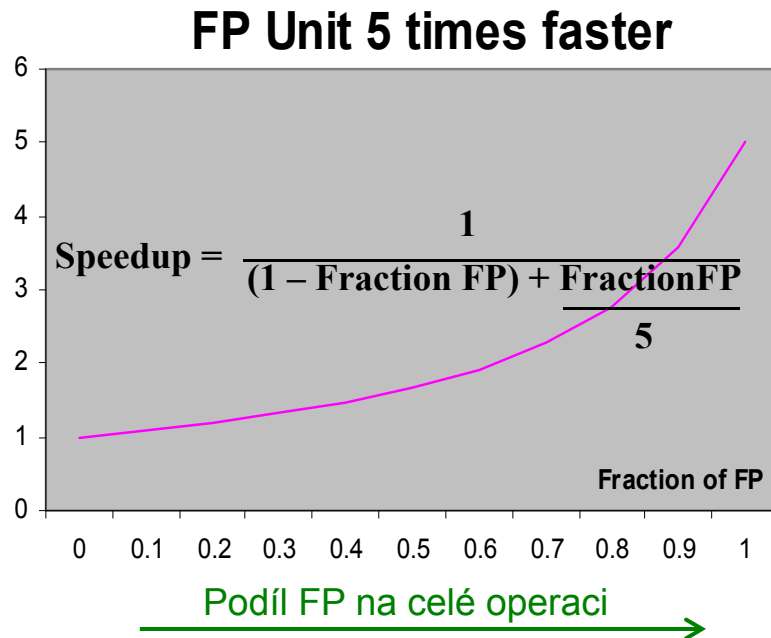
# Grafické znázornění Amdahlova zákona





# Příklad

- Doba zpracování programu je 10 sekund
- Jaké bude zrychlení, bude-li procesor doplněn rychlejší floating point jednotkou?



# Programy pro vyhodnocení výkonu

---

- Benchmarky měří různé stránky výkonu systému a jeho složek
- Ideální situace: známé programy (*workload*)
- Benchmarky
  - **Reálné programy**
  - Jádra
  - Hry-benchmarky
  - **Syntetické benchmarky**
- Risk: návrh může být podřízen požadavkům benchmarku
  - (částečné) řešení: použití **reálných programů**
    - Inženýrské nebo vědeckotechnické aplikační programy
    - Softwarové vývojové prostředky
    - Provádění transakcí
    - Aplikační programy (Office, .... )



# Zkušební úlohy - benchmarky

---

- *Benchmark* je program nebo soubor programů určených pro měření výkonu počítače.
- Čím lépe vystihuje benchmark typickou zátěž počítače, tím přesnější výsledky může poskytnout.
  - Kdyby většinu uživatelů tvořili inženýři, dobrým benchmarkem by byla sada typických inženýrských a vědeckotechnických aplikací.
  - Kdyby většina uživatelů byli vývojáři SW, potom dobrým benchmarkem by byly kompilátory a programy pro zpracování zdrojových textů.

# Kritéria hodnocení výkonu

---

- **Reprodukovatelnost**

- Zahrnuje konfiguraci hardware / software
- Podmínky vyhodnocovacího procesu

- **Shrnutí výkonových parametrů**

- Celkový čas: čas CPU + čas I/O + ostatní doby

- *Aritmetický průměr:*

$$AM = (1/n) * \sum \text{exec\_time}_i$$

- *Vážený aritmetický průměr:*

$$WM = \sum w_i * \text{exec\_time}_i$$

- *Harmonický průměr:*

$$HM = n / \sum (1/\text{rate}_i)$$

- *Geometrický průměr:*

$$GM = (\prod \text{exec\_time\_ratio}_i)^{1/n}$$

$$\frac{GM(X_j)}{GM(Y_j)} = \left[ \frac{X_j}{Y_j} \right]$$

***Podstatné !!!***

# Poznámka k harmonickému průměru

---

**Harmonický průměr** je podíl počtu prvků a součtu převrácených hodnot prvků. Používá se tam, kde má smysl sčítat převrácené hodnoty. Například máme vypočítat střední hodnotu v případě, že víme, že jeden zemědělec zorá pole za 5 dní a druhý za 2 dny. Jaký je jejich průměrný výkon. Můžeme uvažovat tak, že použijeme prostý aritmetický průměr, který má hodnotu 3.5. Ale tento údaj je chybný. Jak je to možné. Představte si, že mají zorat dvě pole. Každý začne na svém poli. Ke konci prvního dne má jeden zoraný jednu pětinu svého pole (celé pole zorá za pět dní, proto za jeden den zorá jednu pětinu) a druhý jednu polovinu. Druhý den má první zorané dvě pětiny, ale druhý už má zorané celé pole a může tedy pomoci prvnímu. Zbývají pouze 0.6 pole, a my víme, že oba dohromady zorají za jediný den 0.7 pole (jedna pětina a jedna polovina dávají jako výsledek právě 0.7). To tedy znamená, že zbytek musejí zorat za méně než jeden den, přesně je to 0.86 dne. Průměr nás však informuje, že by to mělo být ještě jeden a půl dne. Pokud bychom použili harmonický průměr, dostaneme přesně 2.86 dne, což souhlasí s naší úvahou. V tomto případě je tedy nutné použít právě harmonický průměr. (zdroj [www.tiscali.cz](http://www.tiscali.cz))

# Aritmetický a geometrický průměr

|    | A    | B   | C  |
|----|------|-----|----|
| P1 | 1    | 10  | 20 |
| P2 | 1000 | 100 | 20 |

Doba výpočtu (v sekundách)  
strojů: A, B a C  
programů: P1 a P2

|                 | Normalized to A |      |       | Normalized to B |     |      | Normalized to C |      |     |
|-----------------|-----------------|------|-------|-----------------|-----|------|-----------------|------|-----|
|                 | A               | B    | C     | A               | B   | C    | A               | B    | C   |
| Program P1      | 1.0             | 10.0 | 20.0  | 0.1             | 1.0 | 2.0  | 0.05            | 0.5  | 1.0 |
| Program P2      | 1.0             | 0.1  | 0.02  | 10.0            | 1.0 | 0.2  | 50.0            | 5.0  | 1.0 |
| Arithmetic mean | 1.0             | 5.05 | 10.01 | 5.05            | 1.0 | 1.1  | 25.03           | 2.75 | 1.0 |
| Geometric mean  | 1.0             | 1.0  | 0.63  | 1.0             | 1.0 | 0.63 | 1.58            | 1.58 | 1.0 |
| Total time      | 1.0             | 0.11 | 0.04  | 9.1             | 1.0 | 0.36 | 25.03           | 2.75 | 1.0 |

# Shrnutí benchmarků

---

- Předpokládejme, že už jsou vybrány zkušební úlohy. Jak získáme výsledek?
- Například, jestliže dva benchmarky spuštěné na dvou počítačích dávají následující výsledky....

|  | Počítač A | Počítač B |
|--|-----------|-----------|
| Program 1 – doba výpočtu (v sekundách) | 1         | 10        |
| Program 2 – doba výpočtu (v sekundách) | 1000      | 100       |
| Celková doba výpočtu (v sekundách)     | 1001      | 110       |

Z měření vyplývá:

- A je 10 krát rychlejší než B pro program 1.
- B je 10 krát rychlejší než A pro program 2.
- Poměr výkonů počítačů A a B je nejasný. Lze vůbec některý prohlásit za rychlejší ?

# Přístup 1: Celková doba výpočtu

---

- Nejjednodušším přístupem je porovnat celkovou dobu výpočtu pro všechny programy ve zkušební úloze.

- Tímto postupem dostaneme ...

$$\frac{Výkon_B}{Výkon_A} = \frac{DobaVýpočtu_A}{DobaVýpočtu_B} = \frac{1001}{110} = 9.1$$

- Pro uvedenou skladbu programů dostaneme, že B je 9.1 krát rychlejší než A pro programy 1 a 2 dohromady.

Poznámka: poměr 9.1x platí POUZE pro uvedené dva programy, každý spuštěný pouze jednou. O poměru výkonu počítačů A a B nelze říci vlastně nic.



# Přístup 2: Aritmetická střední hodnota

---

- Jinou možností jak vyhodnotit benchmark je určení střední doby výpočtu.
- Tímto způsobem dostaneme...

$$AM = \frac{1}{n} \sum_{i=1}^n Doba_i$$

$$AM_A = \frac{1}{2} \sum_{i=1}^2 Doba_i = \frac{1}{2}(1001) = 500.5 \quad AM_B = \frac{1}{2} \sum_{i=1}^2 Doba_i = \frac{1}{2}(110) = 55$$

- Pro uvedenou kombinaci programů dostaneme, že střední doba výpočtu pro B je 9.1 krát rychlejší než pro A, což souhlasí s předchozím výpočtem celkové doby výpočtu.

# Přístup 3: Vážená aritmetická střední hodnota

---

- Aritmetická střední hodnota předpokládá, že počet spuštění každého programu je stejný (přispívají ke střední hodnotě stejnou měrou). Není-li to pravda, lze každému programu přiřadit váhu.
- Tak dostaneme ...

$$WAM = \frac{1}{n} \sum_{i=1}^n w_i * Doba_i$$

- Jestliže program 1 představuje 20% celkové zátěže a program 2 80%, bude váhový koeficient programu 1 roven 0.2 a váhový koeficient programu 2 bude 0.8.

# Cílené benchmarky

---

- Vývoji benchmarkových programů už bylo věnováno mnoho úsilí.
- Základním problémem při takovém vývoji („výzkumu“) je podvádění ...
  - Kompilátory mohou obsahovat velmi účinné sekvence instrukcí pro specifické benchmarky (zadané ve zdrojovém kódu).
  - Procesory mohou být optimalizovány na provádění určitých instrukčních proudů, protože tyto jsou použity ve specifických benchmarkcích.

# Problémy cílených benchmarků

---

- Hlavním problémem takových benchmarků jsou „podvody“.
- Verze benchmarku SPEC z r. 1989, obsahovala program matrix300, který obsahoval operace násobení matic.
- 99% doby výpočtu tohoto benchmarku probíhalo v jediné řádce zdrojového textu.
- Důsledkem bylo, že mnoho výrobců kompilátorů se soustředilo na extrémní optimalizaci této jediné řádky. To zvýšilo rychlost provádění až na 700 násobek v porovnání se situací bez optimalizace. Takový benchmark ale dává velmi „matnou“ představu o celkovém výkonu systému.

# „Z historie zkušebních úloh“

---

1. Vytvoříte benchmark zvaný *bmark* 😊
2. Ověříte jej na řadě počítačů
3. Publikujete výsledky na [www.bmark.org](http://www.bmark.org)
4. *bmark* and [www.bmark.org](http://www.bmark.org) se stanou populární 😊
  - Uživatelé začnou nakupovat podle výsledků *bmarku*
  - Stanete se populární i u výrobců
5. Výrobci analyzují *bmark* a upraví kompilátory, popř. i mikroarchitekturu tak, aby *bmark* dával dobré výsledky
6. Tímto krokem byl *bmark* znehodnocen ☹️
7. Vytváříte *bmark* 2.0 😊

# Linpack Benchmark

---

- „Matka“ všech benchmarků
- Doba potřebná pro řešení systému lineárních rovnic

```
DO I = 1, N
  DY(I) = DY(I) + DA * DX(I)
END DO
```
- Metriky
  - $R_{\text{peak}}$ : výkonová špička v Gflops
  - $N_{\text{max}}$ : velikost matice dávající nejvíce Gflops
  - $N_{1/2}$ : velikost matice dávající polovinu  $R_{\text{max}}$  Gflops
  - $R_{\text{max}}$ : Gflops dosažené pro velikost matice  $N_{\text{max}}$
- Použito <http://www.top500.org>

# Intelský iCOMP Index 3.0

---

- Nová verze (3.0) zahrnuje:
  - Instrukční mix pro existující software.
  - Nárůst 3D, multimédia a internetový software.
- Benchmarky
  - 2 aplikace integer (20 % každá)
  - Výpočty geometrie v 3D a osvětlení (20 %)
  - FP inženýrské a finanční programy, hry (5 %)
  - Multimediální a internetové aplikační programy (25%)
  - Aplikační programy v jazyce JAVA (10 %)
- Vážený relativní výkon GM (geometrický průměr)
  - Jednotkový“ procesor: Pentium II procesor na 350MHz

$$iCOMP\ Index\ baseline = \left( \frac{BM1}{Base\_BM1} \right)^{P_1} * \left( \frac{BM2}{Base\_BM2} \right)^{P_2} * \dots * \left( \frac{BM6}{Base\_BM6} \right)^{P_6}$$

# Benchmarky Př.3: SPEC2000

---

- **S**ystem **P**erformance **E**valuation **C**orporation
- Je třeba provádět update/upgrade benchmarků
  - Delší doba běhu
  - Rozsáhlejší problémy
  - Různorodost aplikací
- Pravidla běhu a report
  - Základní a optimalizovaný
  - Geometrický průměr normalizovaných dob výpočtu
  - Referenční stroj: Sun Ultra5\_10 (300-MHz SPARC, 256MB)
- CPU2000: poslední SPEC CPU benchmark (4. verze)
  - 12 integer a 14 floating point programů
- Metriky: doba odezvy a propustnost



# Benchmarky SPEC CINT2000

---

- |     |             |     |                                    |
|-----|-------------|-----|------------------------------------|
| 1.  | 164.gzip    | C   | Compression                        |
| 2.  | 175.vpr     | C   | FPGA Circuit Placement and Routing |
| 3.  | 176.gcc     | C   | C Programming Language Compiler    |
| 4.  | 181.mcf     | C   | Combinatorial Optimization         |
| 5.  | 186.crafty  | C   | Game Playing: Chess                |
| 6.  | 197.parser  | C   | Word Processing                    |
| 7.  | 252.eon     | C++ | Computer Visualization             |
| 8.  | 253.perlbnk | C   | PERL Programming Language          |
| 9.  | 254.gap     | C   | Group Theory, Interpreter          |
| 10. | 255.vortex  | C   | Object-oriented Database           |
| 11. | 256.bzip2   | C   | Compression                        |
| 12. | 300.twolf   | C   | Place and Route Simulator          |

# Benchmarky SPEC CFP2000

---

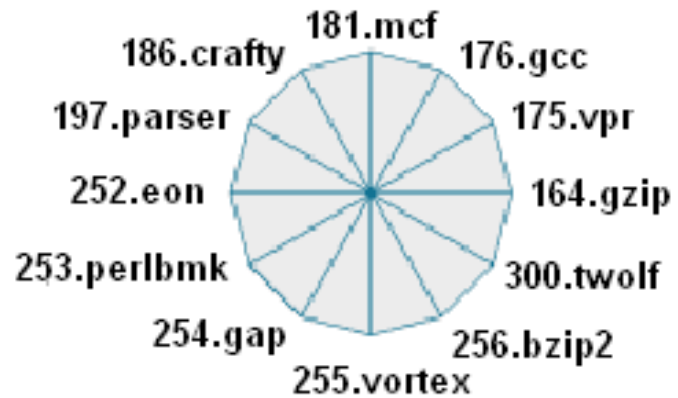
|     |              |     |   |
|-----|--------------|-----|---|
| 1.  | 168.wupwise  | F77 | Physics / Quantum Chromodynamics                    |
| 2.  | 171.swim     | F77 | Shallow Water Modeling                              |
| 3.  | 172.mgrid    | F77 | Multi-grid Solver: 3D Potential Field               |
| 4.  | 173.applu    | F77 | Parabolic / Elliptic Partial Differential Equations |
| 5.  | 177.mesa     | C   | 3-D Graphics Library                                |
| 6.  | 178.galgel   | F90 | Computational Fluid Dynamics                        |
| 7.  | 179.art      | C   | Image Recognition / Neural Networks                 |
| 8.  | 183.quake    | C   | Seismic Wave Propagation Simulation                 |
| 9.  | 187.facerec  | F90 | Image Processing: Face Recognition                  |
| 10. | 188.amp      | C   | Computational Chemistry                             |
| 11. | 189.lucas    | F90 | Number Theory / Primality Testing                   |
| 12. | 191.fma3d    | F90 | Finite-element Crash Simulation                     |
| 13. | 200.sixtrack | F77 | High Energy Nuclear Physics Accelerator Design      |
| 14. | 301.apsi     | F77 | Meteorology: Pollutant Distribution                 |

# Metriky SPECINT2000

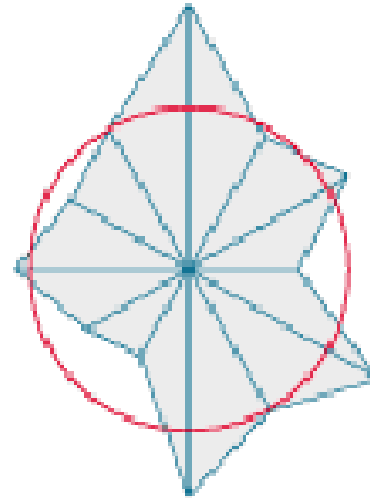
---

- **SPECint2000**: Geometrický průměr 12 normalizovaných poměrů (pro každý integer benchmark jeden), každý benchmark kompilován s "agresivní" optimalizací
- **SPECint\_base2000**: Geometrický průměr 12 normalizovaných poměrů, kompilováno s „konzervativní“ optimalizací
- **SPECint\_rate2000**: Geometrický průměr 12 normalizovaných poměrů pro **propustnost**, kompilováno s "agresivní" optimalizací
- **SPECint\_rate\_base2000**: Geometrický průměr 12 normalizovaných poměrů pro **propustnost**, kompilováno s „konzervativní“ optimalizací

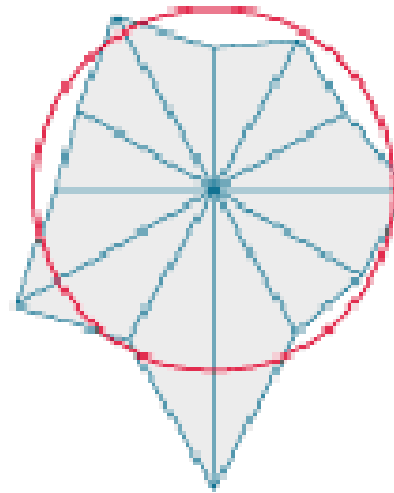
# Výsledky SPECint\_base2000



**Alpha/Tru64**  
**21264 @ 667 MHz**

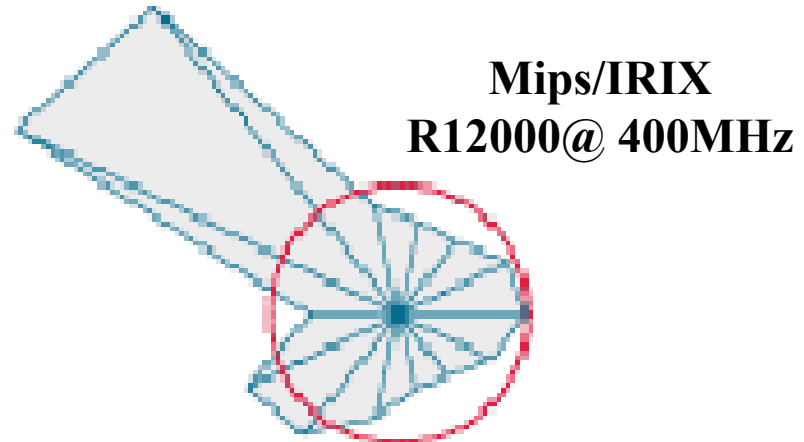
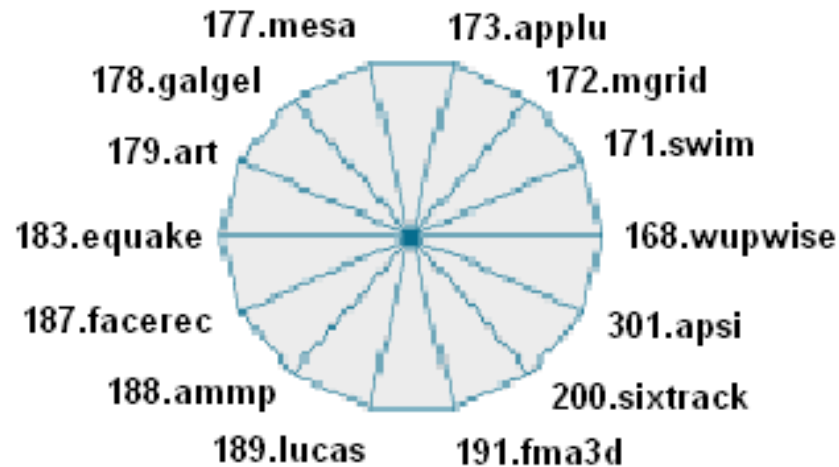


**Mips/IRIX**  
**R12000@ 400MHz**



**Intel/NT 4.0**  
**PIII @ 733 MHz**

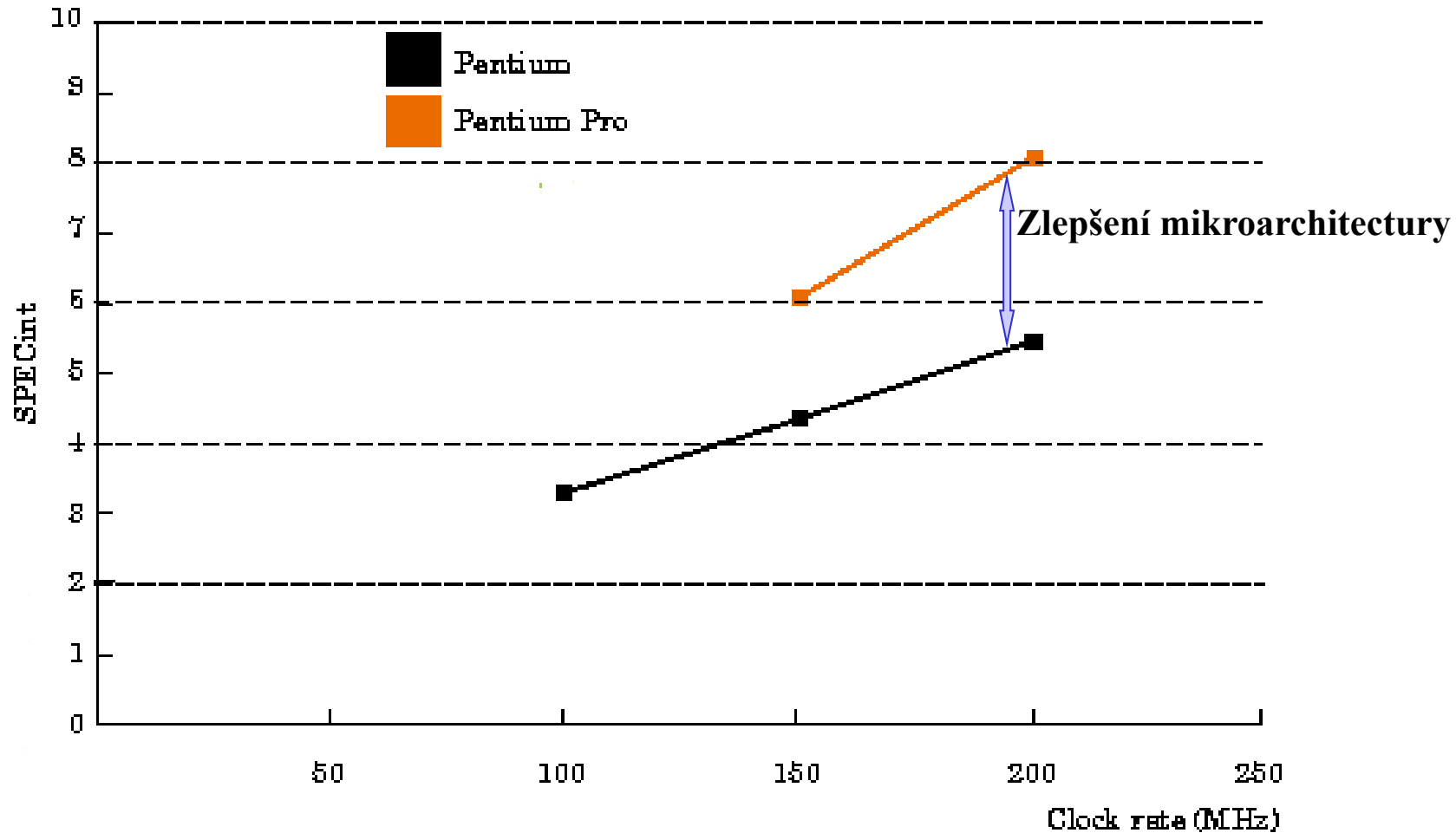
# Výsledky SPECfp\_base2000



**Alpha/Tru64  
21264 @ 667 MHz**

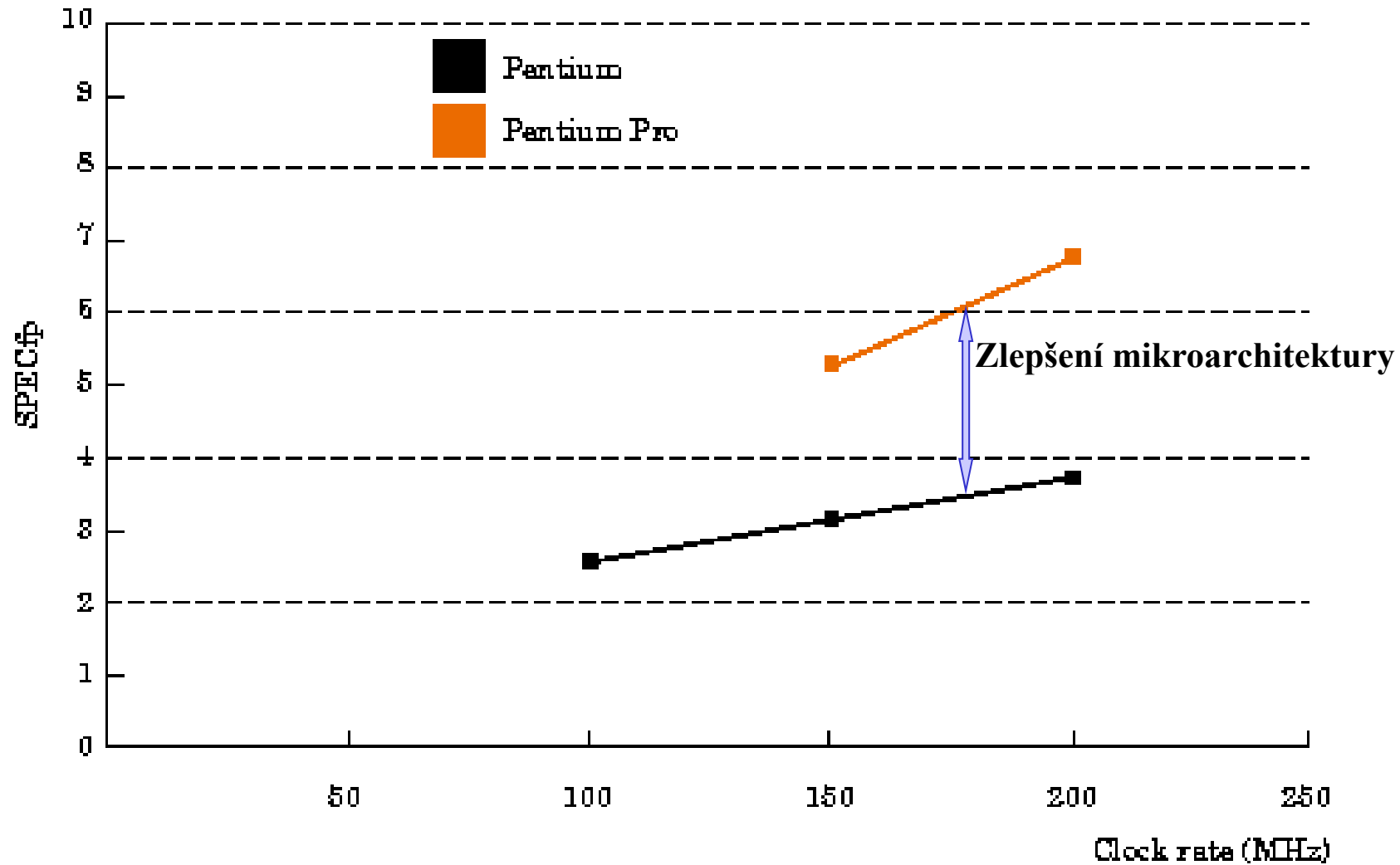
**Intel/NT 4.0  
PIII @ 733 MHz**

# Hodnocení SPECint95



$$\text{CPU time} = \text{IC} * \text{CPI} * \text{clock cycle}$$

# Hodnocení SPECfp95



# Benchmark SPECint2006

---

SPECint2006 test obsahuje 12 benchmarkových programů, navržených exkluzivně pro test výkonu systému v integer oblasti.

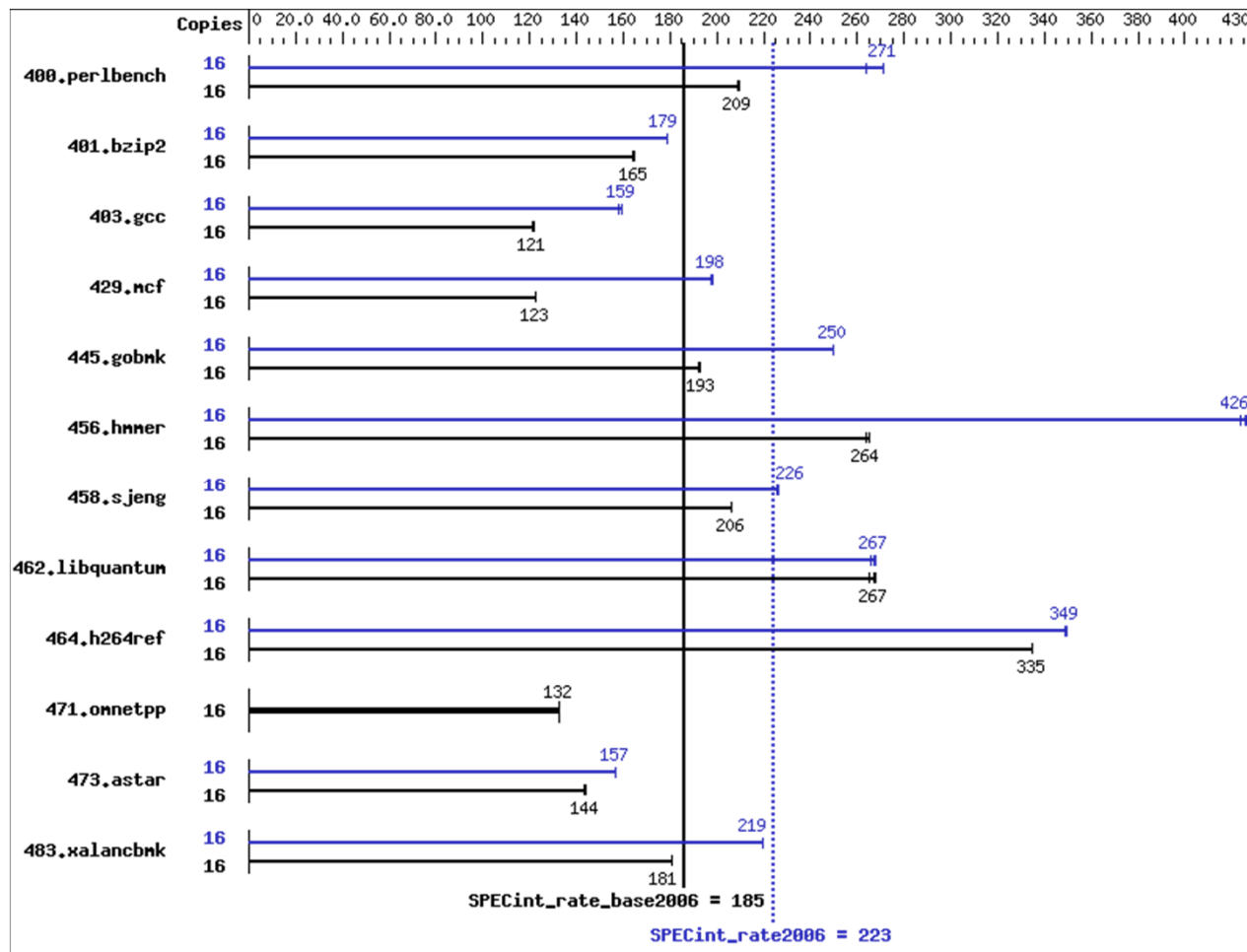
| Benchmark      | Language | Category                    |
|----------------|----------|-----------------------------|
| 400.perlbench  | C        | Programming Language        |
| 401.bzip2      | C        | Compression                 |
| 403.gcc        | C        | Compiler                    |
| 429.mcf        | C        | Combinatorial Optimization  |
| 445.gobmk      | C        | Artificial Intelligence     |
| 456.hmmer      | C        | Search Gene Sequence        |
| 458.sjeng      | C        | Artificial Intelligence     |
| 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              |



# SPEC® CINT2006 Result

Copyright 2006-2009 Standard Performance Evaluation Corporation

Dell Inc., PowerEdge M905 (AMD Opteron 8378, 2.40 GHz)



## Hardware

CPU Name: AMD Opteron 8378  
CPU Characteristics:  
CPU MHz: 2400  
FPU: Integrated  
CPU(s) enabled: 16 cores, 4 chips, 4 cores/chip  
CPU(s) orderable: 4 chips  
Primary Cache: 64 KB I + 64 KB D  
on chip per core  
Secondary Cache: 512 KB I+D on chip per core  
L3 Cache: 6 MB I+D on chip per chip  
Other Cache: None  
Memory: 64 GB (16 x 4 GB DDR2-800)  
Disk Subsystem: 1 x 73 GB 10000 RPM SAS  
Other Hardware: None

## Software

Operating System: SUSE Linux  
Enterprise Server 10 (x86\_64) SP2,  
Kernel 2.6.16.60-0.21-smp  
Compiler: PGI Server Complete Version 7.2  
PathScale Compiler Suite Version 3.2  
Auto Parallel: No  
File System: ReiserFS  
System State: Run level 3 (multi-user)  
Base Pointers: 32/64-bit  
Peak Pointers: 32/64-bit  
Other Software: binutils 2.18  
32-bit and 64-bit libhugetlbfs libraries  
SmartHeap 8.1 32-bit Library for Linux

# Chyby a omyly

---

- Ignorování Amdahlova zákona
- Použití MIPS jako metriky pro výkon
- Použití aritmetického průměru (AM) normalizovaných dob CPU (ratios)
- Užití hardwarově nezávislých metrik
  - Velikost kódu jako měřítko rychlosti
- Syntetické benchmarky predikují výkon
  - Nerespektují chování reálných programů
- Geometrický průměr poměrných dob CPU je proporcionální celkové době výpočtu [Nikoliv!!]

# Závěr

---

- Přesné měření výkonu počítače je vlastně obtížné !
- Benchmarky musí respektovat předpokládanou zátěž. Jedině tak jsou použitelné.
- Různá kritéria
  - čas CPU, čas I/O, ostatní doby, celkový čas
- Vyberte nejlepší metodu prezentace
  - Aritmetická stř. hodnota pro dobu výpočtu
  - Geometrická stř. hodnota pro poměrný výkon
- Nezapomeňte na Amdahlův zákon
  - Velké náklady  $\Leftrightarrow$  a často jen malá zlepšení
  - Je třeba zahrnout i náklady na vývoj

# Závěr (pokr.)

---

- Výkonnost je vázána na konkrétní programy !
- Čas CPU: jediný adekvátní parametr pro měření výkonu. Všechny ostatní metriky jsou nepřesné. Buď nerespektují dobu výpočtu a nebo se soustřeďují jen na málo významný aspekt výkonosti a chybně jej interpretují jako celkový výkon.
- Pro danou ISA lze výkon zvyšovat:
  - nárůstem hodinové frekvence (bez zhoršení CPI)
  - zlepšením organizace procesoru, která snižuje CPI
  - zlepšením kompilátoru, které snižuje CPI a/nebo IC
- *Vaše úlohy (workload) = Váš ideální benchmark*
- Nesmíte vždy věřit všemu co se píše, ani v technické oblasti!