Faktory ovlivňující rychlost vykonávání programového kódu.

Thursday, May 30, 2013 8:32 AM

Motivace

• State of the Art schopností procesorů

- Superskalární architektura
- CISC instrukce se překládají na mikrokód, aby i procesory s CISCovou instrukční sadou mohly těžit z výhod RISCové architektury
- Pipelining
- Vykonávání instrukcí mimo pořadí
- Přejmenovávání registrů
- Spekulativní vykonávání kódu dopředu
- Odhadování výsledků podmínek skoků
- Vektorové instrukce

Ve výsledku může jedno jádro procesoru vykonávat celé instrukce, nebo alespoň části instrukcí, paralelně

- Tj. co programátor píše jako kód jednoho vlákna, se ve skutečnosti vykonává paralelně
- Tj. uvedené optimalizace spadají do rámce PPR
  - Více vláken = principiální pohled na PPR
  - +Optimalizace = reakce na aktuální stav
  - Co bývalo výhodné paralelizovat, to dnes může rychleji spočítat „sériový kód“
- Dobrý překladač může vytvořit takový strojový kód, který umí využít paralelizačních schopností procesoru
  - Proto se liší výkonnost kódu v závislosti na překladači a zvolené míře optimalizace
  - Ale ani ten nejlepší překladač neumí odhadnout vše
- Proto je třeba mu umět pomoci takovým zápisem kódu, ze kterého toho pozná co nejvíce
  - A proto je třeba znát, jak to vypadá z pohledu procesoru
  - Low-level techniky, pokud pro to není zvláštní důvod, se vyplatí nechat na překladači
  - Ale algoritmy na vyšší úrovni jsou záležitosti pro programátora

Základní myšlenky

- Co lze, to se vypočítá pouze jednou
- Co lze, to se vypočítá paralelně
  - Je však třeba zvolit správnou granularitu výpočtu
  - Paralelizace totiž není vždy výhodná
- V některých případech je sériový kód rychlejší, má menší režii – nevytváří a nesynchronizuje vlákna
- Paměť se alokuje co nejméně, využívá se již alokovaná paměť
- Program se píše tak, aby operační systém co nejméně swappoval
- Redukovat náklady na vytvoření a synchronizaci vláken
- Běžet co nejvíce v uživatelském adresovém prostoru
- Ale hlavně, psát efektivní kód už samotného vlákna
- Paralelizovaný neefektivní kód bude s největší pravděpodobností pomalejší než efektivně napsaný sériový kód

Skoky

- Eliminace skoků má zásadní vliv na zvýšení výkonu (jump je hodně náročnej)
- Využit instrukce cmov – bytecode pro to nemá ekvivalent
  - Cílem je uspořádat podmínky, aby si procesor správně tipnul, která instrukce bude další
- Při adresování paměti přes pointery *m musí překladač předpokládat, že *m může ukazovat kamkoli a nemá moc možností, jak optimalizovat. Proto vzhledem k optimalizacím je dobré použít zápis přes indexy, kdy takový zápis obsahuje jakousi nápovědu pro překladač, jak to celé optimalizovat.

Eliminace nepotřebných sekvencí Store-Load

- Používání dočasných proměnných, které mohou být realizovány s pomocí registru
- Programátor by měl pomoci překladači jejich použitím – namísto co nejkratšího zápisu kódu

Konverze operandů

- Nemíchat operandy různé velikosti, konverze to zpomaluje

Loop Unrolling

- Nápověda pro překladač s autovektorizací
- V nejhorším dojde k většímu využití pipelines procesoru

- Loop Unrolling
  - Nápověda pro překladač s auto-vektorizací
  - V nejhorším dojde k většímu využití pipelines procesoru

Naivně	Lépe

`double a[100], sum;`
`int i;`
 	
`sum = 0.0f;`
 	
`for (i = 0;`
`  i < 100; i++) {`
`  sum += a[i];`
`}`
`//Překladač to může,`
`//ale i také nemusí`
`//správně pochopit.`
 	
`//Loop-unrolling`
`//také snižuje počet`
`//porovnávání, tj. i`
`//případných skoků.`
	`double a[100], sum;`
	`double sum1, sum2, sum3, sum4;`
	`int i;`
	
	`sum1 = 0.0f;`
	`sum2 = 0.0f;`
	`sum3 = 0.0f;`
	`sum4 = 0.0f;`
	
	`for (i = 0; i < 100; i + 4) {`
	`  sum1 += a[i];`
	`  sum2 += a[i+1];`
	`  sum3 += a[i+2];`
	`  sum4 += a[i+3];`
	`}`
	`sum = (sum4 + sum3) +`
	`  (sum1 + sum2);`

  - Použití i++ v a[…] by mohlo vnést závislost, tj. zhoršit výkon při využití pipelines

Vyhodnocování podmínek

- Podmínky jsou obvykle vyhodnocovány ve zkrácené formě – cílem je seřadit podmínky tak, aby při jejich vyhodnocování bylo zapotřebí provést co nejméně úkonů, tzn. Bylo co nejrychleji vyřazeno co nejvíce možností a aby byly co nejmenší nároky na vyhodnocení podmínky.
- Sekvence AND končí prvním FALSE, sekvence OR končí prvním TRUE.

Branch prediction – předpovídání skoků

- procesor obsahuje tzv. Branch prediction, kdy spekulativně vykonává kód dopředu podle toho, jaký předpokládá výsledek skoku. Uvedená konverze nemění stav branch-prediction, tj. Neovlivňuje urychlování volající funkce, a kdyby se alokovala paměť, např. Pro nový string, tak už to stav branch-prediction ovlivní.

Kopírování paměti

- kopírování celého bloku paměti místo prvek po prvku

Garbage Collector

- způsobuje náhlé, nedeterministické vytěžování systému

Synchronizace

- jinak než kritickou sekcí, např. Pomocí InterlockedIncrement apod. Kritická sekce má velkou režii.

From <https://d.docs.live.net/e3534876709763a3/Dokumenty/ZCU/Statnice/Statnice.docx> fq