Paralelizace cyklů.

Thursday, May 30, 2013 8:31 AM

Paralelizace cyklů

- Lze provést dekompozici dat, můžeme cykly paralelizovat a tím urychlit výpočet => datový paralelismus SPMD
- Příklady: nalezení minima/maxima v poli, součet prvků v poli

Rozdělení proměnných v paralelním výpočtu

- **Lokální proměnné** – inicializovány uvnitř smyčky
- **Sdílené proměnné**
  - **Nezávislé** – pokud jsou využívány jen pro čtení, nebo pokud v případě pole jde pouze o jeden prvek, se kterým je pracováno v jedné iteraci
  - **Závislé** – dále se dělí na:
    - **Redukční** – nejprve se čte a pak zapisuje (vše v jedné iteraci) – např. suma - stačí mi lokální kopie proměnné a dám to dohromady nakonec - viz concurrency runtime
    - **Uzamykané** – může být čtena i zapisována v několika iteracích (v rámci jedné iterace), nezáleží na pořadí iterací – např. max - musim pouzit zamky, jinak to nejde
    - **Uspořádané** – závisí na pořadí iterací - nepomuze mi nic (jen nejake chytre prevedeni na jiny problem, nelze paralelizovat)

Paralelizaci součtu pole lze provést snadno, protože se v cyklu vyskytuje nejvýše redukční proměnná, kterou stačí jen ošetřit mutexem.

Paralelizace cyklu se závislou uspořádanou proměnnou

S = urychlení = čas_neparalelizovaný/čas_paralelní

E = účinnost = urychlení/počet_cpu

Paralelizace sčítání prefixů (výsledek pole součtů) již nelze provést jednoduchým rozdělením iterací vláknům, protože se v cyklu nachází uspořádaná proměnná.

Uspořádaná proměnná nám brání zapisovat do pole v jiném než určeném pořadí. Proto zavedeme ještě jednu kopii pole. S i-1 pak přistupujeme do pole, do kterého se v cyklu již nezapisuje (zrušeno uspořádání). Poté můžeme přeuspořádat pořadí počítání prefixů do stromu s vrcholem uprostřed sčítané posloupnosti. Jestliže lze paralelizovat výpočet posledního prvku, lze paralelizovat i výpočet ostatních prvků. Pomocí úprav a optimalizací se můžeme dostat až na složitost O(n/m) u paralelní verze.

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

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