Týden 2
Programování zasíláním zpráv

1

Paralelní zpracování

3 typy operací v paralelním programu:

- výpočetní operace (aritmeticko-logické, přenosy dat L/S, řízení průběhu) jako v sekvenčních programech
- operace paralelismu (správa procesů - vytváření a rušení, přepínání kontextu, tvorba skupin procesů)
- interakční operace - meziprocesorová komunikace a synchronizace

- implicitní nebo explicitní v textu paralelního programu
- režie paralelního zpracování

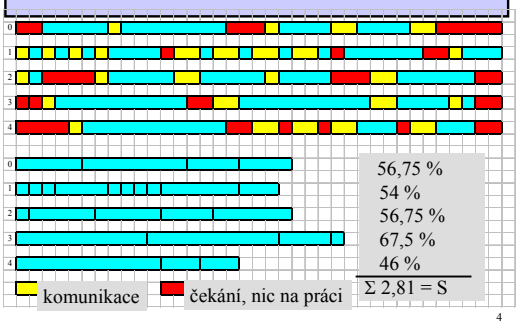
2

Režie paralelního zpracování

1. Nestejné využití procesorů; režie čekání na jiné procesory až dokončí jistou práci, stejný výpočet na několika procesorech k omezení komunikace. (idle)
2. Režie paralelismu - správa procesů, přepínaní kontextu, někdy i výpočty navíc související s počátečním rozdělením úlohy na procesory nebo se zpracováním dílčích výsledků do celkového (par)
3. Režie interakcí procesů - komunikace a synchronizace se u sekvenčního programu nevyskytují. (inter)

Model PRAM zohledňuje pouze režii pod bodem 1.

Reálné paralelní zpracování



4

Vztah zrychlení a využití procesorů

Všechny procesory i = 1, 2, …, P končí ve stejném čase

$$T(n,P) = t_{compi} + (t_{interi} + t_{idlei} + t_{pari}) = t_{compi} + t_{overi}$$

$$PT(n,P) = \sum T(n,P) = \sum (t_{compi} + t_{overi}) = T_S + H$$

$$S(n,P) = \frac{T_s}{T(n,P)} = \sum \frac{t_{compi}}{(t_{compi} + t_{overi})} = \sum E_i$$

$$\frac{1}{P}\sum E_i = E = \frac{S(n,P)}{P}$$

celková režie $H(n,P)$

průměrné využití procesorů E

5

Příklad: globální minimum na EREW PRAM

Sekvenčně: n čísel na jednom procesoru: $n*c$ taktů

Paralelně: P procesorů, n je dělitelné P, $n >> P$, $P = 2^k$

- každý procesor lokální minimum za $c*n/P$ taktů
- redukce na globální minimum za $c*\log P$ taktů

Zrychlení S paralelního zpracování proti sekvenčnímu:

$$S = \frac{sekvenční\ čas}{paralelní\ čas} = \frac{cn}{cn/P + c\log P} = P\frac{n}{n + P\log P}$$

Ideálně $S = P$; reálně $S < P$;
$P \log P$ = režie paralelního zpracování

6

Příklad: naivní mergesort (EREW, hyperkostka)

- n čísel na P procesorech, n je dělitelné P, $n >> P$, $P = 2^k$
- každý CPU: sekvenční Quicksort za $\frac{n}{P}\log\frac{n}{P}$ kroků
- spojení seřazených posloupností délky $n/P, \ldots, n/4, n/2$:

$$\frac{n}{P} + \frac{2n}{P} + \ldots + \frac{n}{4} + \frac{n}{2} = n(1 - \frac{1}{P})$$

kroků, každý 2 takty (komunikace + sečtení)

$$S(n,P) = \frac{n \log n}{\frac{n}{P}\log\frac{n}{P} + 2n(1-\frac{1}{P})} = P\frac{\log n}{\log n + 2P - 2 - \log P}$$

$2P - 2 - \log P$ ..... režie

Numerické hodnoty pro naivní mergesort

- $S(n, P) = P \log_2(n)/(\log_2(n/P) + 2P - 2)$.
- $S(n, \infty) = \log_2(n)/2$.
- $E(n,P) = S(n, P)/P = \log_2(n) /(\log_2(n/P) +2P - 2)$,
- $E(n, \infty) = 0$.
- Příklad:

$S(10^8, 2)$	=	1.9 (96%),
$S(10^8, 4)$	=	3.5 (88%),
$S(10^8, 8)$	=	5.7 (71%),
$S(10^8, 16)$	=	8.1 (51%),
$S(10^8, 32)$	=	10.2 (32%),
$S(10^8, 64)$	=	11.6 (18%).

Komunikace zasíláním zpráv

1. **Tvorba procesů:**
   - 1 proces/procesor
   - několik procesů/ procesor - multitasking, podpora v HW nebo distribuovaném OS
     a. statické, nemění se během zpracování
     b. dynamické, tvořeny a rušeny během zpracování
2. **Zasílání zpráv:**
   - udáním id cílového procesu (kam) nebo
   - pojmenovanými **kanály** (kudy);

   kanály *interní* v rámci procesoru, kopie z paměti do paměti
   externí mezi procesory, mapovány na fyzické linky statické nebo dynamické.

Modely komunikace

- Kanály = programovací abstrakce
  jednosměrné (simplex) nebo
  obousměrné (half-duplex, full-duplex).
- Technika přepínaní:
  ulož a pošli dál (SF), červí (WH)
- Počet bran

Brána je hw jednotka (DMA) schopná komunikovat současně s prací CPU. Počet bran účastnících se komunikace současně může být jedna, všechny (all-port) nebo obecně k.

- Schopnost manipulace se zprávami (pakety):

Uzly mohou vytvářet větší zprávy poskládáním vstupních zpráv (combining model) nebo jen přenášet dál původní zprávy (non-combining model).

Modelování komunikace - časový model

Zpoždění při komunikaci je funkcí
- *vzdálenosti* d, počtu kanálů po cestě a
- *délky zprávy* m [slov (byte)].

Vnitřní kanály:
$t_{INT} = t_s + m\, t_w$ „lineární model“

Vnější kanály SF: citlivé na vzdálenost
$t_{SF} = t_s + d\,(m\, t_w + t_r) + t_{CD}$ $t_r << m\, t_w$

Vnější kanály WH: necitlivé na vzdálenost
$t_{WH} = t_s + m\, t_w + d\, t_r + t_{CD}$ $d = 1$: $SF \approx WH$

t_s, pevná doba nastartování komunikace
t_r, směrovací zpoždění, zahrnuje dobu přepínání ve směrovači
t_w, doba přenosu slova t_{CD}, zpoždění kvůli obsazeným linkám

Parametry komunikačního modelu (Transim)

		Default values:
• CPU speed SPD [MHz]	real	20
• time slice TSL [CPU clock cycles]	int	2048
• link speed LS [Mbit/s]	real	10
• external memory factor EF [ ]	int	5
(additional CPU cycles required per external memory cycle)		
• memory ratio MR [ ]	real	0
(0 /1 = process' code & data entirely in internal / external memory)		

Int./ext.channel speed: real
- ICS [Mbyte/s] 2.0 * SPD= 40 Mbyte/s = 1 byte/25 ns
- ECS [Mbyte/s] LS /11.25 bit/byte = 0.89 Mbyte/s

Int./ext.channel delay: real = 1 byte/1.12µs
- ICD [µs] 64 / SPD = 3.2 µs
- ECD [µs] 82/ SPD = 4.1 µs

Komunikační model - příklady

[μs]	t_s	t_w [4 bytes]
Transim (Transputer T8)	4,1	4,5
IBM SP2	46	0,14
Pracovní stanice na Ethernetu	1500	5
Pracovní stanice na FDDI	1150	1,1
Unix pipes Linux 1.1 na Pentiu	157	0,31

13

Zlepšení účinnosti paralelního zpracování

- překrytí výpočtů a komunikací, komunikací jinými komunikacemi
- komunikace větších bloků dat
- předvýběr dat před jejich použitím
- eliminace komunikační SW režie
- zrychlení komunikačního HW
- lepší mapování algoritmu na architekturu
- dynamické vyrovnávání zátěže

14

Způsoby zasílání zpráv a implementace

1. Synchronní
2. (Asynchronní) blokující
3. (Asynchronní) neblokující

Implementace:

a. speciální paralelní programovací jazyk

b. rozšíření syntaxe a vyhrazených slov sekvenčního programovacího jazyka pro zasílání zpráv

c. použití stávajícího sekvenčního jazyka plus knihovna externích procedur pro zasílání zpráv

15

Synchronní zasílání zpráv

Proces P:

```
s =10;
...
send (&s, Q) ;
s = 20;
```

s = send buffer

Proces Q:

```
r = -100;
...
recv(&r, P) ;
x = r + 1;
```

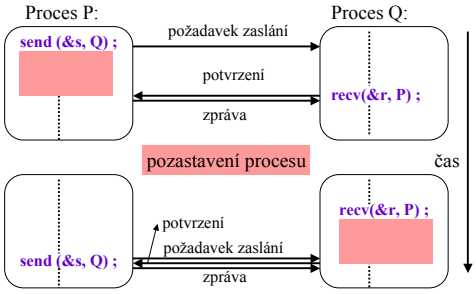
r = receive buffer

Procesy čekají jeden na druhého až jsou oba u **send** nebo **receive**; oba příkazy skončí až je zpráva odeslána a přijata. s lze hned přepsat a r hned číst, x = 11. Žádný další buffer.

Uváznutí komunikace pokud partner nedojde ke správnému komunikačnímu příkazu.

16

Synchronní send a receive, třícestný protokol



17

(Asynchronní) blokující send/receive

Proces P:

```
s =10;
...
send (&s, Q);
s = 20;
```

Proces Q:

```
r = -100;
...
recv(&r, P);
x = r + 1;
```

send/receive začne hned jakmile k němu proces dojde, nečeká na partnera (bude potřeba *dočasný buffer zprávy*).

Blokující **send skončí** jakmile je zpráva odeslána a s je volné. (Dokud s nelze přepsat, **send** „blokuje“). Při tom receive ale nemusí být hotovo, nemuselo ani začít.

Blokující **receive skončí** až je zpráva přijata, x = 11.

18

(Asynchronní) neblokující send/receive

Proces P:

```
s =10;
send (&s, Q);
wait (sh,ss);
s = 20;
```

Proces Q:

```
r = -100;
recv(&r, P);
wait (rh,rs);
x = r + 1;
```

send/receive začne hned jakmile k němu proces dojde, nečeká na partnera (bude potřeba *dočasný buffer*).
Neblokující **send skončí**, jakmile uvědomí systém, že se má poslat zpráva. Data mohou být ještě v s a nelze je přepsat.
Neblokující **receive skončí**, když uvědomí systém, že má být přijata zpráva. (Zpráva při tom může být na cestě nebo ještě ani nebyla odeslána). x = 11, 21, -99 ! → **wait(handle,stav)**

19

Srovnání 3 typů komunikací

Cílem je <u>překrytí komunikace a výpočtů.</u>

- u **synchronního** způsobu je možné pro každou komunikaci vytvořit vlákno (hw nebo OS) a dobu čekání využít pro práci na jiném vláknu; nebo vytvořit vlákno (proces) buffer.

- u **asynchronního** způsobu se redukuje čekání na partnera, ale jsou nutné dočasné buffery. Jejich alokace, přepisování zprávy a testování pomocí wait( ) jsou ale nová režie! Neměla by být větší než úspora z překrytí.

20