Návrh paralelních algoritmů

Tomáš Oberhuber
tomas.oberhuber@fjfi.cvut.cz

25. února 2021

Video na Youtube

Úvod I.

Vývoj paralelního algoritmu je nutno chápat jako **optimalizaci**.

1. nejprve vždy vyvíjíme co **nejjednodušší sekvenční algoritmus bez optimalizací**
   - **za každou cenu se vyhýbáme předčasné optimalizaci**
   - ta může zcela zbytečně poničit čistý návrh algoritmu
   - bez základní jednoduché verze kódu nemůžeme poměřit přínos paralelizace
2. máme-li základní funkční kód, který není dostatečně výkonný, přistupujeme k optimalizacím
   - pokud je to možné, optimalizujeme jen sekvenční kód
   - pokud to nepostačuje, přikročíme k paralelizaci
3. během implementace optimalizací provádíme průběžné testy a kontrolujeme, zda optimalizovaný kód dává stále správné výsledky
4. nakonec poměřujeme přínos optimalizace tj. výslednou **efektivitu paralelizace**

Úvod II.

Budeme se tedy zabývat:

1. návrhem paralelních algoritmů
2. analýzou paralelních algoritmů

Návrh paralelních algoritmů

Metodiky návrhu paralelních algoritmů:

- **task/channel model** - Ian Foster
  - naprosto nejběžnější postup při návrhu paralelních algoritmů
- **bulk synchronous parallel model** - `www.bsp-worldwide.org`

Task/channel model I.

Tento model reprezentuje paralelní výpočet jako množinu úloh (*tasks*), které mezi sebou komunikují pomocí komunikačních kanálů (*channels*).

Task/channel model I.

Tento model reprezentuje paralelní výpočet jako množinu úloh (*tasks*), které mezi sebou komunikují pomocí komunikačních kanálů (*channels*).
Task představuje:

- program
- lokální paměť
  - instrukce a soukromá data
- vstupně/výstupní porty
  - task je používá ke komunikaci s ostatními tasky

Task/channel model II.

Channel je modelován jako:

- fronta zpráv spojující výstupní port jednoho tasku se vstupním portem nějakého jiného tasku
- data se na vstupu příjemce objevují ve stejném pořadí, v jakém bylo odeslána odesílatelem
- task nemůže přijímat data, která nebyla ještě odeslána
  - přijímání zpráv je vždy **blokující**
- task, který zrávu odesílá nemusí čekat, až druhý task zprávu přijme
  - odesílání zpráv je vždy **neblokující**
- přijímání zpráv je **synchronní**
- odesílání zpráv je **asynchronní**

Task/channel model III.

Výhodou *task/channel* modelu je, že jasně odlišuje přístup do lokální paměti a komunikaci mezi tasky.

- to je nezbytné pro návrh algoritmů pro architektury s distribuovanou pamětí
- umožňuje to efektivnější návrh algoritmů pro architektury se sdílenou pamětí

Task/channel model III.

Výhodou *task/channel* modelu je, že jasně odlišuje přístup do lokální paměti a komunikaci mezi tasky.

- to je nezbytné pro návrh algoritmů pro architektury s distribuovanou pamětí
- umožňuje to efektivnější návrh algoritmů pro architektury se sdílenou pamětí

Definition

Doba běhu paralelního algoritmu je pak definována jako čas, po který byl aktivní alespoň jeden task.

Fosterova metodika návrhu paralelních algoritmů

Ian Foster navrhl čtyři kroky vedoucí k návrhu paralelního algoritmu:

1. *partitioning* - rozdělení úlohy
   - celou úlohu rozdělíme na malé kousky - prvotní tasky - *primitive tasks*
   - snažíme se o co nejjemnější rozdělení
2. *communication* - komunikace
   - odvodíme, jak spolu musí jednotlivé podúlohy komunikovat
3. *agglomeration* - aglomerace
   - slučujeme malé podúlohy do větších za účelem redukce komunikace (počtu kanálů)
   - dostáváma tak vlastní tasky
4. *mapping* - mapování
   - jednotlivé tasky přidělujeme procesorům/výpočetním jednotkám

Analýza paralelních algoritmů I.

- použití dvou procesorů místo jednoho prakticky nikdy nevede k ukončení výpočtu v polovičním čase
- paralelizace s sebou vždy nese určitou režii navíc:
  - interakce a komunikace mezi jednotlivými procesy
  - prostoje procesorů
    - nerovnoměrné rozdělení práce
    - čekání na ostatní procesy
  - některé výpočty navíc oproti sekvenčnímu algoritmu

Naší snahou nyní bude odvození teorie pro ohodnocení úspěšnosti paralelizace dané úlohy.
Poznámka: p opět označuje počet procesorů, které se účastní paralelního výpočtu a n je velikost řešené úlohy (podle vstupních dat).

Analýza paralelních algoritmů II.

Definition

Sériový (sekvenční) čas běhu algoritmu (*serial runtime*) - $T_S(n)$ - je doba mezi spuštěním a ukončením výpočtu sekvenčního algoritmu na úloze o velikosti n.

Definition

Paralelní čas běhu algoritmu (*parallel runtime*) - $T_P(n, p)$ - je doba mezi spuštěním algoritmu a okamžikem, kdy poslední proces ukončí svůj výpočet.

Analýza paralelních algoritmů II.

Definition

Sériový (sekvenční) čas běhu algoritmu (*serial runtime*) - $T_S(n)$ - je doba mezi spuštěním a ukončením výpočtu sekvenčního algoritmu na úloze o velikosti n.

Definition

Paralelní čas běhu algoritmu (*parallel runtime*) - $T_P(n, p)$ - je doba mezi spuštěním algoritmu a okamžikem, kdy poslední proces ukončí svůj výpočet.

Poznámka: Pokud porovnáváme sekvenční a paralelní čas, měříme sekvenční čas na nejrychlejším známém sekvenčním algoritmu. Navíc požadujeme nejrychlejší známý sekvenční algoritmus pro danou velikost úlohy, ne asymptoticky nejrychlejší sekvenční algoritmus.

Analýza paralelních algoritmů II.

Definition

Sériový (sekvenční) čas běhu algoritmu (*serial runtime*) - $T_S(n)$ - je doba mezi spuštěním a ukončením výpočtu sekvenčního algoritmu na úloze o velikosti n.

Definition

Paralelní čas běhu algoritmu (*parallel runtime*) - $T_P(n, p)$ - je doba mezi spuštěním algoritmu a okamžikem, kdy poslední proces ukončí svůj výpočet.

Poznámka: Pokud porovnáváme sekvenční a paralelní čas, měříme sekvenční čas na nejrychlejším známém sekvenčním algoritmu. Navíc požadujeme nejrychlejší známý sekvenční algoritmus pro danou velikost úlohy, ne asymptoticky nejrychlejší sekvenční algoritmus.

Analýza paralelních algoritmů III.

Definition

Čas čistě sekvenční části algoritmu (*time of inherently sequential part*) - $P_S(n)$ je doba, za kterou proběhne výpočet neparalelizovatelné části algoritmu.

Definition

Čas paralelizovatelné části algoritmu (*time of parallelisable part*) - $P_P(n)$ je doba, za kterou proběhne výpočet paralelizovatelné části algoritmu při sekvenčním zpracování.

- $P_P(n) = T_S(n) - P_S(n)$

Analýza paralelních algoritmů IV.

Definition

Celková režie (*total overhead*) - $T_O(n, p)$ - je definována jako

$$T_O(n, p) = pT_P(n, p) - T_S(n).$$

Analýza paralelních algoritmů IV.

Definition

Celková režie (*total overhead*) - $T_O(n, p)$ - je definována jako

$$T_O(n, p) = pT_P(n, p) - T_S(n).$$

Definition

Urychlení (*speedup*) - $S(n, p)$ - je definováno jako

$$S(n, p) = T_S(n)/T_P(n, p).$$

Analýza paralelních algoritmů V.

- platí, že $S(n,p) \leq p$
  - kdyby $S(n,p) > p$, pak by žádný procesor nesměl běžet déle než $T_S(n)/p$
  - potom bychom mohli vytvořit sekvenční algoritmus, který bude emulovat paralelní výpočet a dostaneme menší $T_S(n)$
- v praxi lze ale často pozorovat **superlineární urychlení**, kdy je $S(n,p) > p$
  - rozdělená úloha se může vejít do vyrovnávacích pamětí procesorů, datová komunikace je tak rychlejší
  - dekompozice při prohledávání
    - tím, že prohledáváme současně více větví stavového stromu, můžeme řešení najít dříve
    - sekvenčně lze toto napodobit prohledáváním stromu do šířky - to je ale těžší k implementování

Analýza paralelních algoritmů VI.

Definition

Efektivita (*efficiency*) - $E(n)$ - je definována jako

$$E(n,p) = S(n,p)/p \leq 1.$$

- je-li $S(n,p)$ lineární funkcí vůči p, pak $E(n,p) = E(n)$, máme algoritmus s efektivitou nezávislou na počtu procesorů, což by byl ideální stav
- s rostoucím počtem procesorů efektivita ve většině případů klesá

Analýza paralelních algoritmů VI.

Definition

Náklady (*cost*) - $C(n, p)$ - jsou definovány jako

$$C(n, p) = pT_P(n, p).$$

Definition

Řekneme, že algoritmus je nákladově optimální, pokud je $C(n, p) = \Theta\left(T_S(n)\right)$.

Analýza paralelních algoritmů VI.

Definition

Práce (*work*) - $W(n, p)$ - je definována jako

$$W(n, p) = \sum_{i=0}^{p-1} t_i,$$

kde t_i je čistý čas výpočtu i-tého procesoru.

Amdahlův zákon

- tento zákon popisuje jak maximálně lze urychlit úlohu o pevně dané velikosti n
- známe tedy sekvenční čas

$$T_S(n) = P_S(n) + P_P(n)$$

- odvodíme paralelní čas jako

$$T_P(n,p) = P_S(n) + P_P(n)/p + T_O(n,p)$$

- potom je

$$\begin{aligned} S(n,p) &= \frac{T_S(n)}{T_P(n,p)} = \frac{P_S(n) + P_P(n)}{P_S(n) + P_P(n)/p + T_O(n,p)} \\ &\leq \frac{P_S(n) + P_P(n)}{P_S(n) + P_P(n)/p} \end{aligned}$$

Amdahlův zákon

- označme
$$s = P_S(n)/(P_S(n) + P_P(n))$$
čistě sekvenční část algoritmu **při sekvenčním výpočtu**.
- pak je
$$P_S(n) + P_P(n) = \frac{P_S(n)}{s}, \tag{1}$$
a
$$P_P(n) = (1/s - 1)P_S(n). \tag{2}$$
- dosazením (1) a (2) do
$$S(n,p) \leq \frac{P_S(n) + P_P(n)}{P_S(n) + P_P(n)/p},$$
- dostáváme
$$\begin{aligned} S(n,p) &\leq \frac{\frac{P_S(n)}{s}}{P_S(n) + \left(\frac{1}{s} - 1\right)\frac{P_S(n)}{p}} = \frac{\frac{1}{s}}{1 + \left(\frac{1}{s} - 1\right)\frac{1}{p}} \\ &= \frac{\frac{1}{s}}{\frac{s + \frac{1-s}{p}}{s}} = \frac{1}{s + \frac{1-s}{p}} \end{aligned}$$

Amdahlův zákon

Theorem

Amdahlův zákon: *Buď* $0 \leq s \leq 1$ *část výpočtů, které musí být prováděny čistě sekvenčně. Maximální urychlení* $S(n,p)$ *dosažitelné při použití p procesorů pro úlohu fixní velikosti (tj. fixního sekvenčního času* $T_S(n)$*) je*

$$S(n,p) \leq \frac{1}{s + (1-s)/p}.$$

- z Amdahlova zákona lze snadno získat asymptotický odhad pro urychleni $S(n,p)$
- $$\lim_{p\to\infty} S(n,p) \leq \lim_{p\to\infty} \frac{1}{s + (1-s)/p} = \frac{1}{s}.$$
- to znamená, že výpočet nemůžeme nikdy urychlit více než $1/s$-krát.

Amdahlův zákon

Příklad: Odhady říkají, že 90% našeho algoritmu lze paralelizovat a zbývajících 10% musí být zpracováno jen na jednom procesoru. Jakého urychlení dosáhneme při použití 8 procesorů?

$$S(n,p) \leq \frac{1}{0.1 + (1 - 0.1)/8} \approx 4.7$$

To je výrazně méně než požadované urychlení 8. Minimálně ze tří procesorů nemáme žádný užitek.

Amdahlův efekt

U rozumně navržených paralelních algoritmů platí, že paralelní režie ma asymptoticky nižší složitost než výpočet paralelizovatelné části

$$T_O(n, p) = o(P_P(n))$$

- navyšování velikosti výpočtu způsobí výraznější růst $P_P(n)$ než $T_O(n, p)$
- při pevném počtu procesorů p je urychlení $S(n, p)$ rostoucí funkcí proměnné n

Amdahlův efekt

- Amdahlův zákon zkoumá možnost maximálního urychlení výpočtu úlohy fixní velikosti
- výsledek Amdahlova zákona je dost pesimistický pro paralelizaci
  - pokud čistě sekvenční část algoritmu tvoří 10%, pak nikdy nedosáhneme většího než desetinásobného urychlení
- paralelizace tedy neumožňuje řešit úlohu dané velikosti v libovolně krátkém čase
- ale už z Amdahlova efektu vidíme, že přínos paralelizace je spíše v tom, že dokážeme řešit větší úlohy
- paralelizace tedy umožňuje provádět přesnější výpočty, kvalitnější vizualizaci apod.

Gustavsonův-Barsiho zákon

- nyní se tedy nebudeme snažit zkracovat čas výpočtu
- místo toho se pokusíme v daném čase vypočítat paralelně co největší úlohu
- to znamená, že nyní bude pevně daný paralelní čas

$$T_P(n,p) = P_S(n) + P_P(n)/p + T_O(n,p)$$

Gustavsonův-Barsiho zákon

- onačme jako s časový podíl, který zabere zpracování čistě sekvenční části **při paralelním výpočtu**, tj.

$$s = \frac{P_S(n)}{P_S(n) + \frac{P_P(n)}{p} + T_O(n,p)}. \tag{3}$$

- tj.

$$P_S(n) = \left(P_S(n) + \frac{P_P(n)}{p} + T_O(n,p)\right) s, \tag{4}$$

- z (3) dostáváme

$$1 - s = \frac{\frac{P_P(n)}{p} + T_O(n,p)}{P_S(n) + \frac{P_P(n)}{p} + T_O(n,p)},$$

tj.

$$P_P(n) = \left(P_S(n) + \frac{P_P(n)}{p} + T_O(n,p)\right)(1-s)\,p - pT_O(n,p). \tag{5}$$

Gustavsonův-Barsiho zákon

- získané vztahy

$$P_S(n) = \left(P_S(n) + \frac{P_P(n)}{p} + T_O(n,p) \right) s,$$

$$P_P(n) = \left(P_S(n) + \frac{P_P(n)}{p} + T_O(n,p) \right) (1-s)\, p - pT_O(n,p).$$

- a dosadíme do vztahu pro urychlení

$$\begin{aligned} S(n,p) &= \frac{P_S(n) + P_P(n)}{P_S(n) + P_P(n)/p + T_O(n,p)} \\ &= \frac{(P_S(n) + P_P(n)/p + T_O(n,p))\,(s + (1-s)\,p)}{P_S(n) + P_P(n)/p + T_O(n,p)} - \\ &\quad \frac{pT_O(n,p)}{P_S(n) + P_P(n)/p + T_O(n,p)} \end{aligned}$$

Gustavsonův-Barsiho zákon

- celkem tedy dostáváme

$$S(n,p) = s + (1-s)p - \frac{pT_O(n,p)}{P_S(n) + P_P(n)/p + T_O(n,p)}.$$

- předpokládáme-li

$$T_O(n,p) = o(P_P(n)) \text{ a } P_S(n) = o(P_P(n)),$$

- pak dostáváme

$$S(n,p) = s+(1-s)p-O(P_P(n)^{-1}) = p+(1-p)s-O(P_P(n)^{-1}).$$

Gustavsonův-Barsiho zákon

Theorem

Gustavsonův-Barsiho zákon: *Mějme paralelní program řešící problém velikosti n na p procesorech. Buď s část z celkového času výpočtu potřebná ke zpracování čistě sériové části při paralelním výpočtu. Předpokládejme*

$$T_O(n,p) = o(P_P(n)) \text{ a } P_S(n) = o(P_P(n)),$$

Pro maximální dosažitelné urychlení pak platí

$$S(n,p) = p + (1-p)s - O(P_P(n)^{-1}).$$

Poznámka: Mnoho textů o paralelizaci uvádí jen $S(n,p) \leq p + (1-p)s$.

Gustavsonův-Barsiho zákon

- Amdahlův zákon vychází ze sekvenčního výpočtu a odvozuje, kolikrát rychlejší může být tento výpočet s využitím paralelizace
- Gustavsonův-Barsiho zákon vychází z paralelního výpočtu a vyvozuje, kolikrát déle by trval tento výpočet bez paralelizace
  - nebere ale v úvahu superlineární urychlení, tedy fakt, že paralelní architektura může mít výhodu ve větším množství rychlejší paměti

Gustavsonův-Barsiho zákon

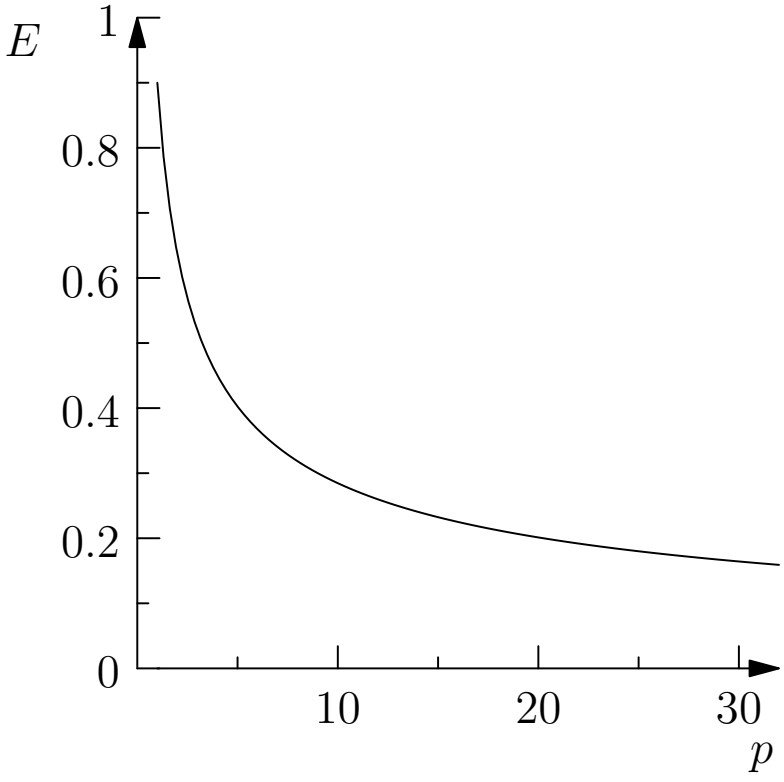
Příklad: Výpočet běžící na 64 procesorech trvá 220 sekund. Měření ukazuje, že 5% z celého času výpočtu zabere čistě sekvenční část algoritmu. Jakého urychlení bylo dosaženo?

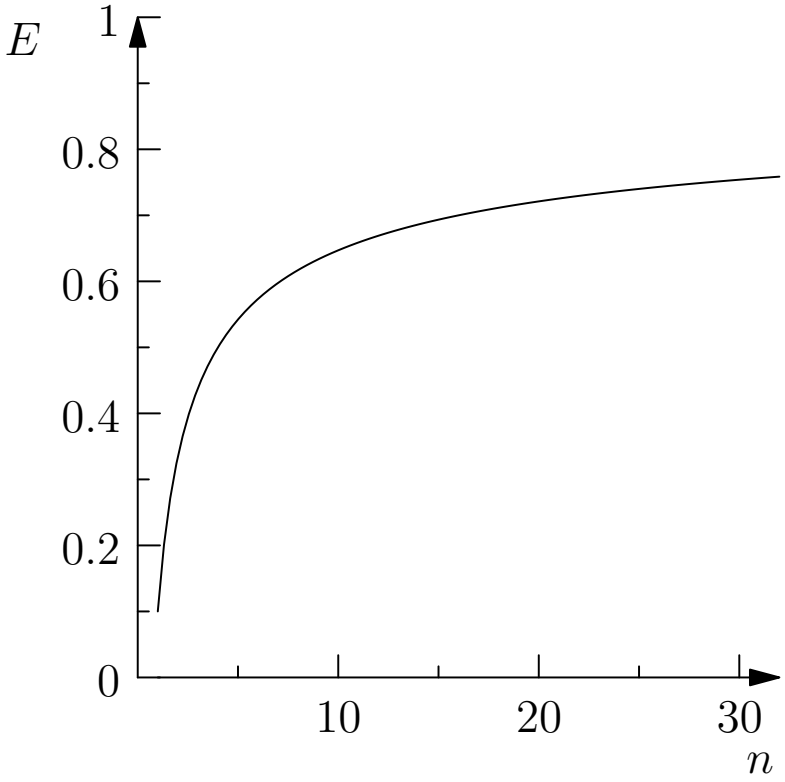
$$S(n,p) = 64 + (1 - 64) \cdot 0.05 = 64 - 3.15 = 60.85.$$

Efektivita a škálovatelnost

Závislost efektivity na

- rostoucím počtu procesorů p (plyne z Amdahlova zákona)
- a velikosti úlohy W (plyne z Gustavsonova-Barsiho zákona)





Škálovatelnost paralelních algoritmů

- jde o vlastnost paralelního algoritmu využít efektivně velký počet procesorů
- otázka je, o kolik musíme zvětšit danou úlohu, abychom po přidání určitého počtu procesorů zachovali zvolenou efektivitu
- čím méně je nutné velikost úlohy zvětšovat, tím lépe

Nákladově optimálního algoritmus

Definition

Algoritmus je nákladově optimální, právě když platí

$$pT_P(n,p) = \theta(T_S(n)).$$

Platí

$$T_P(n,p) = \frac{T_S(n) + T_O(n,p)}{p}.$$

Tedy

$$pT_P(n,P) = T_S(n) + T_O(n,p) = \theta(T_S(n)) \Leftrightarrow T_O(n,p) = O(T_S(n)).$$

Nákladově optimálního algoritmus

Definition

Algoritmus je nákladově optimální, právě když platí

$$pT_P(n,p) = \theta(T_S(n)).$$

Platí

$$T_P(n,p) = \frac{T_S(n) + T_O(n,p)}{p}.$$

Tedy

$$pT_P(n,P) = T_S(n) + T_O(n,p) = \theta(T_S(n)) \Leftrightarrow T_O(n,p) = O(T_S(n)).$$

Theorem

Algoritmus je nákladově optimální, právě když paralelní režie nepřevyšuje řádově velikost úlohy.

Modely ideálních paralelních architektur

- když analyzujeme složitost paralelních algoritmů, je často potřeba předpokládat určité vlastnosti paralelní architektury, pro kterou je navržený
- teoreticky se paralelní architektury popisují pomocí PRAM

Modely ideálních paralelních architektur

PRAM = Parallel Random Access Machine

- jde o model architektury se sdílenou pamětí
- stroj má p procesorů a globální paměť neomezené kapacity se stejně rychlým přístupem na jakoukoliv adresu pro všechny procesory
- modely PRAM se dělí podle ošetření přístupu více procesorů na stejnou adresu

Modely ideálních paralelních architektur

- **EREW PRAM** = Exclusive Read Exclusive Write PRAM
  - ani čtení ani zápis nelze provádět současně
  - je to nejslabší PRAM
- **CREW PRAM** = Concurrent Read Exclusive Write PRAM
  - možné současné čtení více procesorů z jedné adresy
  - to umí dnešní GPU
- **ERCW PRAM** = Exclusive Read Concurrent Write PRAM
  - umožňuje současný zápis
- **CRCW PRAM** = Concurrent Read Concurrent Write PRAM
  - umožňuje současné čtení i zápis

PRAM CW protokoly pro zápis

Současný zápis je možné ošetřit následujícími protokoly:

- **common** (obyčejný)
  - všechny zapisované hodnoty musí být stejné
- **arbitrary** (náhodný)
  - náhodně se vybere jeden proces, který zápis provede
  - u ostatních zápis selže
- **priority** (prioritní)
  - procesy mají dané priority, podle kterých se určí, kdo provede zápis
- **sum** (součet)
  - zapíše se součet všech hodnot