Vysoké učení technické v Brně
Fakulta informačních technologií

Paralelní a distribuované algoritmy

Implementace algoritmu pipeline merge sort

8. dubna 2015 Jan Wrona

Obsah

1 Úvod

Cílem projektu je pomocí knihovny *OpenMPI* implementovat v jazyce C/C++ algoritmus *pipeline merge sort*. Vstupem programu je soubor *numbers*, který obsahuje libovolná (tedy binární) data. Ta jsou programem interpretována jako čísla tak, že jeden bajt je chápán jako číslo v rozsahu $< 0-255 >$. To jsou čísla určená k seřazení. Je požadováno tuto posloupnost vypsat na standardní výstup, čísla budou v vzájemně oddělena mezerou. Druhou částí předpokládaného výstupu je správně vzestupně seřazená vstupní posloupnost, čísla budou vzájemně oddělena novým řadkem.

2 Rozbor a analýza algoritmu

Sekvenční verze algoritmu *merge sort* pracuje se sekvencemi čísel. Vstupní posloupnost n čísel je rozdělena do n subsekvencí délky 1. Prvním průchodem je vytvořeno $\frac{n}{2}$ seřazených subsekvencí délky 2, obecně je i-tým průchodem vytvořeno $\frac{n}{2^i}$ seřazených subsekvencí délky 2^i. Každý průchod se skládá z n kroků a seřazení celé posloupnosti vyžaduje $\log(n)$ průchodů. Celková časová složitost sekvenčního algoritmu merge sort je tak $O(n \log(n))$.

Jedna z možností jak tento proces paralelizovat je pomocí pipelines, vzniká tak paralelní algoritmus pipeline merge sort. Tento algoritmus vyžaduje lineární pole procesorů délky $\log(n)+1$, kde n je délka vstupní posloupnosti čísel taková, že $n = 2^r, r \in \mathbb{N}$. Procesory P_1 až P_{r+1} jsou rozděleny do tří skupin. První skupinu tvoří pouze procesor P_1, který má na vstupu jednu pipeline obsahující vstupní posloupnost, ze které v každém kroku přečte jedno číslo a střídavě jej pošle na jednu ze svých dvou výstupních pipelines (vytváří tak subsekvence délky 1 pro procesor P_2). Druhou skupinu tvoří procesory P_2 až P_r, které přijímají na svých dvou vstupních pipelines subsekvence délky 2^{i-2}, kde i je číslo procesoru. Výstup této skupiny procesorů tvoří subsekvence, vytvořené sloučením dvou vstupních subsekvencí, které jsou postupně umisťovány do jedné z dvou výstupních pipelines. Třetí skupinu tvoří pouze procesor P_{r+1}. Ten zpracovává vstup stejně jako procesory předchozí skupiny, výstup v podobě seřazené vstupní posloupnosti však umisťuje do jediné pipeline. Procesory jsou pomocí dvou pipelines propojeny v pořadí, v jakém jsou očíslovány, tedy P_1 předává subsekvence P_2, P_2 je předává P_3 atd.

Při analýze časové složitosti se jeden výpočetní krok skládá z porovnání dvou čísel z vrcholů vstupních pipelines, vytažení většího/menšího z nich (v závisloti na vzestupném/sestupném řazení) a jeho umístění do výstupní pipeline. Z předchozího popisu lze vidět, že procesory nemohou pracovat vždy všechny současně, protože nemusí mít ve svých vstupních pipelines dostatek dat. Procesor P_1 začíná pracovat okamžitě po startu, ostatní procesory až ve chvíli, kdy mají celou vstupní sekvenci v jedné z pipelines a jedno číslo z následující sekvence v druhé pipeline. Matematicky zapsáno, procesor P_1 začíná pracovat v kroku 1. Krok, kterým začína procesor P_i, je dán součtem délky sekvence plus 1 všech procesorů $P_j, j \leq i$, tedy $1+\sum_{j=0}^{i-2}(2^{i-2}+1)$. Tato posloupnost lze zapsat jako $2^n + n$, tento vztah je ale potřeba upravit s ohledem na číslování procesorů od jedničky následovně:

$$1 + \sum_{j=0}^{i-2}(2^{i-2} + 1) = 2^{i-1} + (i\ - 1).$$

Obdobná situace je při ukončování výpočtu. Procesory svou práci končí spolu s poslední zpracovanou sekvencí, nikoliv všechny současně. V kroku, kdy procesor začal pracovat, bylo zpracováno jedno číslo. Zbývá jich tedy ještě $n-1$. Krok, kdy procesor ukončí svou práci, lze vyjádřit

následovně:

$$2^{i-1} + (i\ -1) + (n-1).$$

Časová složitost je u paralelních algoritmů rovna času mezi startem výpočtu prvního procesoru a koncem výpočtu posledního procesoru. U zkoumaného algoritmu první procesor P_1 začíná pracovat v čase 1 a poslední procesor P_{r+1} končí výpočet v čase $2^{r+1-1} + (r+1-1) + (n-1) = 2^r + r + n - 1$. Z předpokladu, že $n = 2^r$ můžeme odvodit $r = \log(n)$ a po dosazení do předchozího výrazu vznikne $2n + \log(n) - 1$. Časová složitor algoritmu je tedy lineání

$$O(2n + \log(n) - 1) = O(n).$$

Cena je definována jako součin časové složitosti a počtu procesorů.

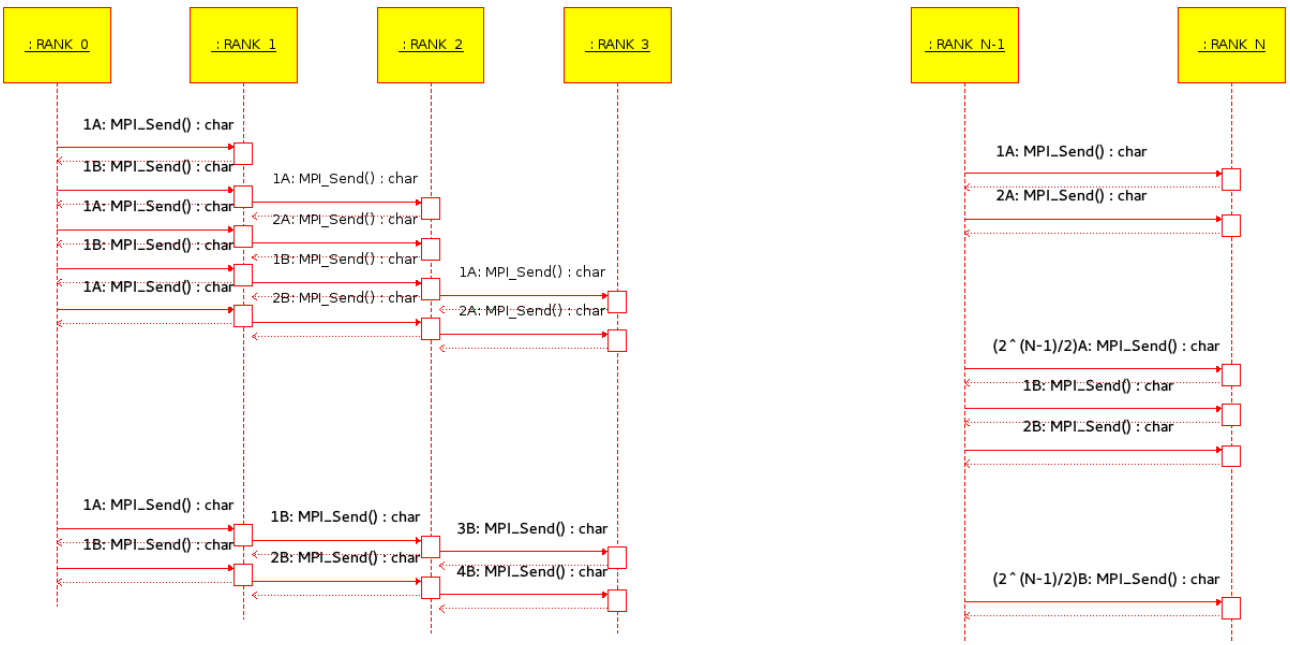
$$C(n) = O(n) * (\log(n) + 1) = O(n\log(n) + n) = O(n\log(n)).$$

Nejrychlejší známé sekvenční řadící algoritmy dosahují časové složitosti $O(n\log(n))$, cena pipeline merge sort je tedy optimální.

Paměťová složitost je dána potřebnou velikostí všech pipelines. Vstupní a výstupní pipelines mají velikost stejnou jako počet prvků řazené sekvence čísel n. Každý procesor P_2 až P_{r+1} disponuje dvěma vstupními pipelines, každou o velikosti délky vstupní sekvence daného procesoru plus jedna, tedy $2^{i-2} + 1$. Celková paměťová složitost je tedy lineární

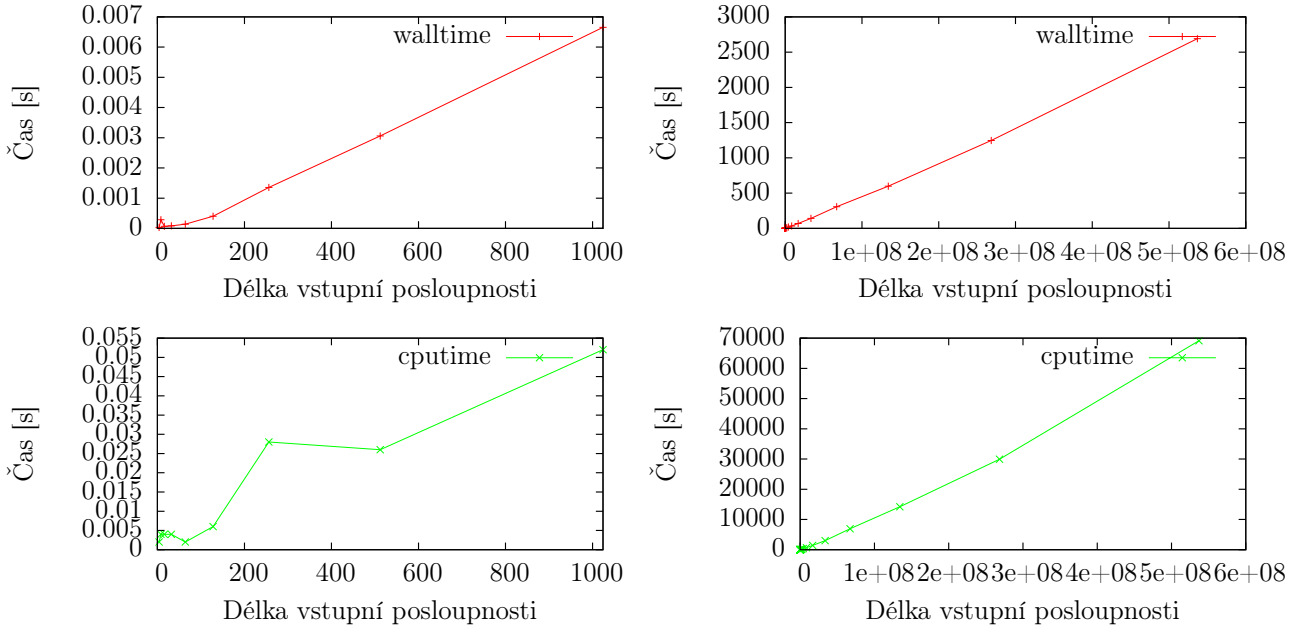
$$O(2n + \sum_{i=2}^{\log(n)+1} (2 * (2^{i-2} + 1))) = O(4n + 2\log(n) - 2) = O(n).$$

3 Komunikační protokol



4 Implementace, testování a experimenty

Algoritmus jsem implementoval jak v jazce C s využitím vlastní implementace fronty, tak i v C++, kde byla použita `queue` z STL. Dvojí implementace jsem následně využil k sérii expe-



Obrázek 1: Výsledky měření časové složitosti. Reálný čas nahoře, procesorový čas dole.

rimentů. Program v C byl mírně rychlejší, zároveň vlastní impelemtace fronty byla paměťově méně náročná (oboje pouze v řádu jednotek procent).

Ověření časové složitosti jsem prováděl měřením reálného času běhu části programu. Tato část obsahovala pouze reálné výpočetní jádro bez inicializací, výpisu do terminálu/souboru, atp. Začátek měření času byl spuštěn v jednom procesu po návratu z funkce `MPI_Barrier()`, stejným způsobem bylo měření ukončeno, aby bylo zajištěno, že všechny procesory již ukončily svůj výpočet. Spolu s reálným časem probíhalo také měření spotřebovaného procesorového času všech procesorů, který byl následně sečten promocí `MPI_Reduce()`.

Měření, jehož výsledky lze vidět na obrázku 1, bylo provedeno na jediném výpočetním uzlu[1], na kterém byl alokován počet procesorů větší nebo roven počtu MPI procesorům. Tímto by měl být minimalizován vliv ostatních procesů a plánovače operačního systému, které mohou dobu běhu programu výrazně ovlivnit. Ze stejných důvodů bylo měření pro každou velikost vstupu opakováno pětkrát, z výsledných časů byl zaznamenán ten nejmenší.

5 Závěr

Rozbor algoritmu v sekci 2 ukazuje, že teoretická časová i paměťová složitost algoritmu jsou lineární, což při použití daného počtu procesorů dělá algoritmus také optimálním. Ve stejné sekci je také popsána vzájemná komunikace procesorů a v kapitole 3 je tento princip znázorněn pomocí sekvenčního diagramu. Po implementaci algoritmu byla provedena řada testů a experimentů, které si kladly za cíl ověřit teoretickou časovou složitost. Jak lze vidět v sekci 4, naměřené časy pro různé délky vstupních posloupností odpovídají lineárnímu průběhu a teoretickou složitos tak potvrzují i prakticky.

[1] ramdal.ics.muni.cz, `http://metavo.metacentrum.cz/pbsmon2/machine/ramdal.ics.muni.cz`