Enumeration sort

Petr Dvořáček – xdvora0n@stud.fit.vutbr.cz

Tento dokument pojednává o paralelním řadícím algoritmu Enumeration sort. Text se zaměřuje především na jeho implementaci a experimentování s časovou složitostí.

1 Popis algoritmu a jeho implementace

Paralelní řadicí algoritmus Enumeration sort má dvě verze, které se liší v použité topologii. Po nás byl vyžadován Enumeration sort s lineární topologií. Takže pro seřazení n hodnot potřebujeme n procesorů. Všechny procesory jsou navíc propojeny sběrnicí.

Průběh řazení se dá shrnout v algoritmu uvedeného na přednáškách:

1. Inicializace proměnné $C := 1$ na všech procesorech.
2. Následující činnosti se opakují $2n$ krát $1 <= k <= 2n$:
   - Pokud vstup není vyčerpán, vstupní prvek x_i se vloží do X_i (sběrnicí) a do Y_1 (lineárním spojením) a obsah všech registrů Y se posune doprava.
   - Každý procesor s neprázdnými registry X a Y je porovná, a je-li $X > Y$ inkrementuje C.
   - Je-li $k > n$ (t.j. po vyčerpání vstupu) procesor P_{k-n} pošle sběrnicí obsah svého registru X procesoru P_{C_k-n}, který jej uloží do svého registru Z.
3. V následujících n cyklech procesory posouvají obsah svých registrů Z doprava a procesor P_n produkuje seřazenou posloupnost od největšího prvku po nejmenší.

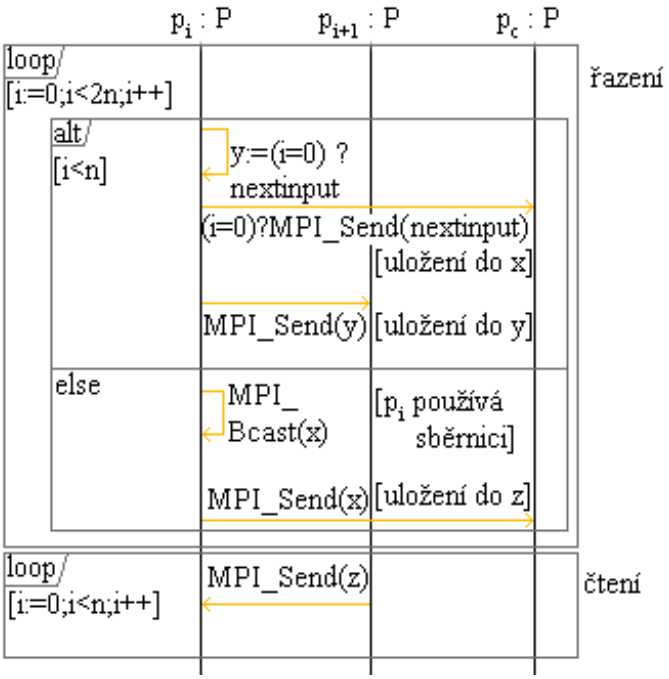
Tento algoritmus však není schopen porovnávat stejné hodnoty. Toto lze vyřešit takovým způsobem, že změníme vstupní abecedu řazených symbolů. V zadání je specifikováno, že máme očekávat hodnoty v intervalu od 0 do 255. Tuto hodnotu před řazením bitově posuneme doleva o 24 bitů a provedeme logický součin s indexem načítaného znaku. Jinými slovy: `value = (value << 24) idx;`. Po řazení pak stačí posunout hodnotu o 24 bitů doprava.

Časová složitost prvního kroku je konstantní – c. Druhý krok se provede po $2n$ krocích. Tisknutí a posouvaní hodnot zabere n kroků. Z teorie složitosti víme, že $O(2n + n + c) = O(n)$. Důkaz tohoto tvrzení je zřejmý. Teoretická časová složitost algoritmu je tedy lineární: $t(n) = O(n)$. V úvodu této kapitoly bylo řečeno, že bude potřeba n procesorů: $p(n) = O(n)$. Algoritmus není optimální, protože $t(n)p(n) = n^2$.

Chtěl bych poznamenat, že implementaci druhého kroku jsem z důvodů zadání trochu pozměnil. Posuv registrů Z provádím doleva, aby procesor P_0 tiskl hodnoty od nejmenšího po největší. Jiný způsob řešení by byl, kdybychom před inkrementací C porovnávali $X < Y$.

2 Komunikační protokol

V implementaci je předávání zpráv mezi sousedy řešeno následujícím způsobem. Pokud procesor P s indexem i chce poslat svému pravému resp. levému sousedovi zprávu, pak jí pošle procesoru P s indexem $i + 1$ resp. $i - 1$. Implementace sběrnice je řešena broadcastem (místo konkrétního procesoru se použije konstanta `MPI_ANY_SOURCE`) a s jiným TAGem než je použito u předávaní zpráv mezi sousedy. Princip této komunikace znázorňuje sekvenční diagram uvedený níže.



3 Experimenty s posloupnostmi různých délek

Pro ověření časové složitosti algoritmu byla použita funkce `gettimeofday()` z modulu `<sys/time.h>`. Začátek respektive konec měření se prováděl po inicializaci procesů respektive před jejich ukončením. Bylo zvoleno vždy 10 běhů pro pole o délkách 10, 20, 30 a 40, jejichž časové hodnoty se pak vypisovaly na standardní chybový výstup (stderr). Měření probíhalo na serveru Merlin. Následující tabulka a graf zachycuje naměřené hodnoty, z nichž lze vyčíst, že algoritmus se chová lineárně.

Velikot pole	Průměrný čas (us)
10	11089
20	21473
30	31121
40	42921