Paralelní a distribuované algoritmy
Enumeration sort

Lukáš Dibďák, xdibda00

1 Rozbor a analýza algoritmu

Popis algoritmu

Algoritmus Enumeration sort pracuje s lineárním polem procesorů **p(n) = n**, doplněných společnou sběrnicí, která je schopná v každém kroku přenést právě jednu hodnotu. Procesory, nacházející se za sebou jsou propojeny lineárním spojením. Algoritmus není schopen řadit vstup, který obsahuje více stejných hodnot, z tohoto důvodu byla navržena modifikace popsaná níže.

Každý procesor obsahuje čtyři registry **C**, **X**, **Y**, **Z**, kde

C	počet prvků menších než x_i
X	prvek x_i
Y	postupně prvky x_1 .. x_n
Z	seřazený prvek z_i

Rozbor algoritmu

- Nechť algoritmus řadí **n** prvků.
- Každý procesor nastaví svůj registr **C** na inicializační hodnotu 1
- Celkový počet kroků k tohoto algoritmu odpovídá **2n**. V každém kroku se provede:
  - V první polovině kroků, kde **k <= n**, není vstup vyčerpán. V každém kroku se tedy prvek $\mathbf{x_i}$ vloží na sběrnici, odkud se přesune do $\mathbf{X_i}$, a na lineární spojení, kde se vloží do registru $\mathbf{Y_1}$ a obsah všech registrů se posune doprava
  - Pokud registry **X** a **Y** daného procesu nejsou prázdné, porovnají a platí-li **X > Y** a pokud jsou tyto podmínky splněny, pak se inkrementuje registr **C**
  - V druhé polovině, kde **k > n**, dochází k vyčerpání vstupů a v každém kroku procesor $\mathbf{P_{k-n}}$ pošle sběrnicí procesoru, který je na jeho pozici **C,** obsah svého registru **X**. Procesor, který hodnotu **X** přijme si ji uloží do svého registru **Z**
- V následujících **n** cyklech posouvají procesory obsahy svých registrů **Z** vpravo a produkují seřazenou posloupnost **n** prvků

2 Teoretická složitost

Algoritmus lze rozdělit na tři části podle výše zmíněného rozboru.

inicializace registrů C všech n procesorů	**konstantní**	**O(n)**
samotný proces řazení	**konstantní**	**O(2n)**
produkování sežazené posloupnosti z registrů Z všech n procesorů	**konstantní**	**O(n)**

Celková složitost je pak vypočítána následujícím způsobem a není optimální.

p(n)	**n**	
t(n)	**O(n)**	
c(n)	**$O(n^2)$**	výpočet: t(n) * p(n)

3 Implementace

Nejdříve se pomocí příkazů **MPI_Init**, **MPI_Comm_size** a **MPI_Comm_Rank** inicializuje prostředí paralelního procesu.

V další části řídící procesor načte ze souboru *numbers* náhodně vygenerovanou posloupnost čísel, které se budou dále zpracovávat. Každý z procesorů si inicializuje registry na hodnotu **-1**, která reprezentuje prázdný registr.

V hlavní smyčce programu řídící procesor posílá hodnoty ze souboru ostatním procesorům, dokud na vstupu nezbývá žádná další hodnota. Veškeré odesílání a přijímání prvků procesorami je implementováno pomocí blokujících příkazů **MPI_Send** a **MPI_Recv**. Každý procesor provádí **2n** smyček ve kterých podle pravidel z části **Rozbor algoritmu** inkrementuje svůj registr **C,** přeposílá číslo svému pravému sousedovi a zároveň přijímá číslo od levého souseda a v závěru naplňuje svůj registr **Z**. Specifikace registrů při odesílání pomocí **MPI** funkcí je zařízena unikátním tagem registru.

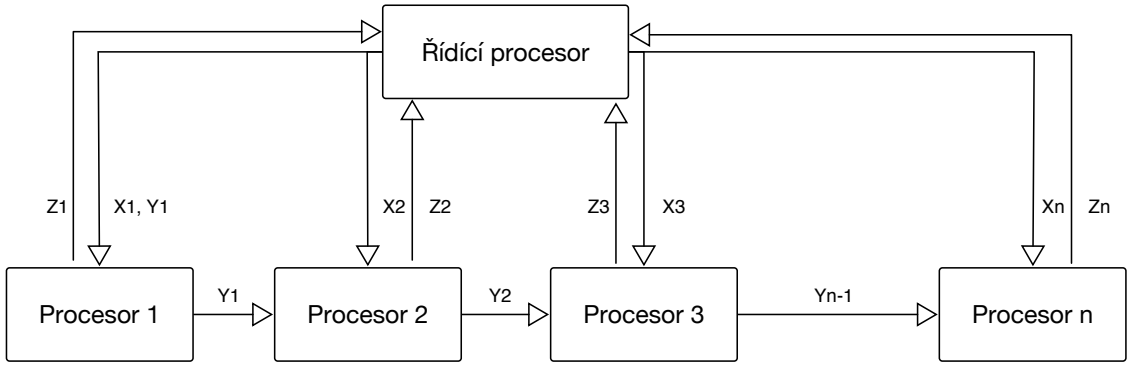
Jelikož algoritmus neumí řadit posloupnosti se stejnými prvky, byl tento problém ošetřen pomocí kontroly prázdnosti registru **Z** a vypsání obsahu registru **Z** předchozího procesoru (předchozí procesor musel logicky obržet i volání následujícího procesoru).

Měření bylo prováděno explicitně podle rozdílu mezi UNIXovým časem na začátku vykonávání programu a na konci tohoto programu. Synchronizovanost procesorůů kvůli konečnému času měření byla zajištěna vytvořením synchronizační bariéry příkazem **MPI_Barrier.**

4 Experimenty s různě velkými vstupy

5 Komunikační protokol

Komunikační protokol je implementován pomocí následujícího obecného sekvenčního diagramu.



6 Závěr

Dosažené výsledky hodnotím jako přijatelné. Nelze v reálném prostředí získat výsledky odpovídající teoretické složitosti daného algoritmu. Důvodem tohoto zkreslení může být režie procesorů při provádění programu, zpomalení na serveru, testovací stroj či neoptimálnost implementace algoritmu v programu.