Paralelné a distribuované algoritmy

Enumeration sort

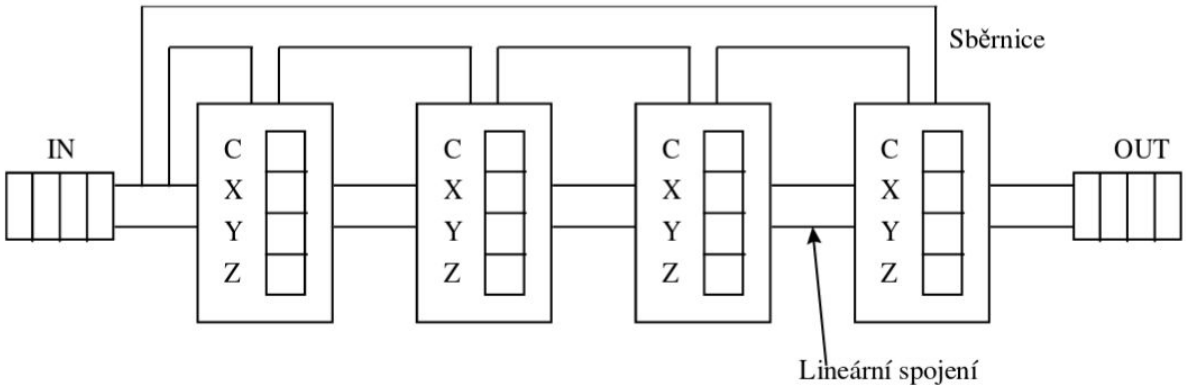
xmarus05 2016/2017

Úvod

V tejto dokumentácii je popísaná implementácia a analýza algoritmu Enumeration sort na lineárnom poli o n procesoroch. Algoritmus je implementovaný pomocou C++ a Open MPI knižnicou. Uvedená je analýza algoritmu, experimenty a meriania časovej zložitosti algoritmu. V neposlednom rade je uvedený komunikačný protokol MPI správ medzi jednotlivými procesormi.

Analýza algoritmu

Radiaci algoritmus Enumeration na lineárnom poli n procesorov beží na lineárnej architektúre, v ktorej sa navyše nachádza spoločná zbernica (viz. Obrázok 1). Táto zbernica musí byť schopná preniesť v jednom kroku 1 hodnotu.



Obrázok 1: Architektúra Enumeration sort(slidy PRL)

Algoritmus je možné neformálne popísať nasledovne:

1. Všetky registre C sa nastavia na hodnotu 1 (zložitosť: **O(1)**)
2. Následujúce činnosti sa opakujú 2n krát (zložitosť: **O(2n)**)
    a. ak vstup nie je vyčerpaný xi sa vloží do Xi (zbernicou) a do Y1 (linearnym spojením) a obsah všetkých registrov Y sa posunie doprava.
    b. Každý procesor s neprázdnymi registrami X a Y ich porovná a ak X > Y inkrementuje C.
    c. Ak k > n procesor Pk-n pošle zbernicou obsah X procesoru Pck-n, ktorý ho uloží do svojho registru Z.
3. V následujúcich n cykloch procesory posúvajú svoj obsah registrov Z doprava a procesor Pn produkuje zoradenú postupnosť. (zložitosť: **O(n)**)

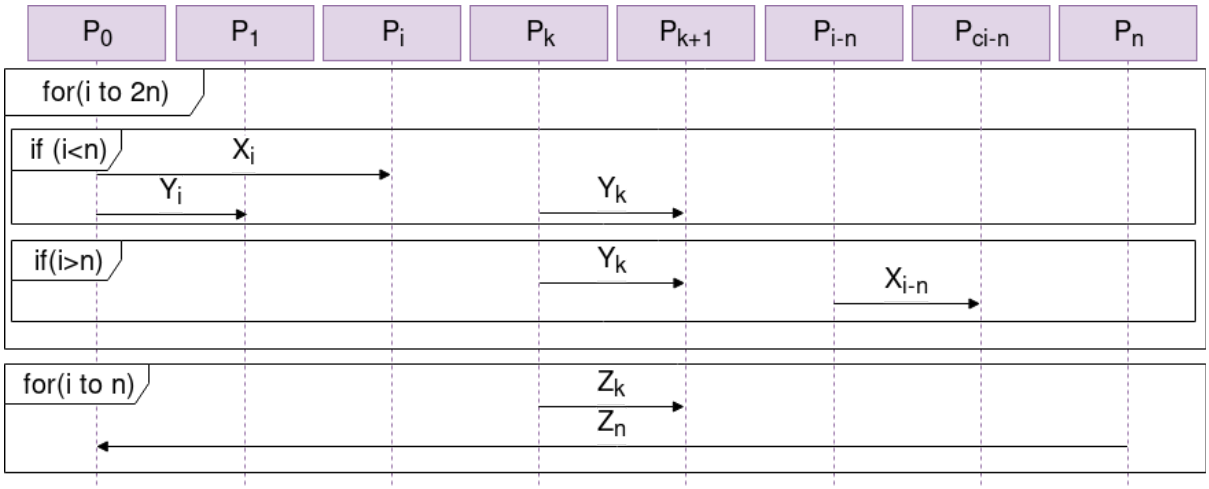
Z algoritmu je možné vidieť že jeho časová zložitosť je lineárna (1. krok trvá konštantný čas, 2. krok trvá 2n a 3. krok trvá n).

- Časová zložitosť: $t(n) = O(n)$
- Počet procesorov pre výpočet n prvkov: $p(n) = n$
- Celková cena výpočtnu: $c(n) = t(n) * p(n) = O(n^2)$

Cena algoritmu nie je optimálna. Tento výpočet platí len za predpokladu, že prenos hodnoty zbernicou tvrá konštantný čas bez ohľadu na vzdialenosť procesorov.

Implementácia

Algoritmus je implementovaný pomocou knižnice Open MPI v jazyku C++. Každý procesor obsahuje register X, Y, Y_{index}, Z. Register X obsahuje hodnotu procesoru, register *Y* obsahuje hodnotu, ktorú si procesory posúvajú smerom do prava. Register Y_{index} obsahuje index procesoru, ktorý má posúvanú hodnotu *Y* v registre *X* a register *Z* obsahuje výslednú hodnotu procesoru. Spojenie medzi jednotlivými procesormi boli implementované pomocou funkcii *MPI_Send()* a *MPI_Recv()*. *MPI_Send()* umožňuje zasielať správy medzi jednotlivými procesormi a funkcia *MPI_Recv()* umožňuje prijatie správy. Ako zbernica slúži master procesor (procesor P_0), ktorý načíta dáta, rozpošle jednotlivé hodnoty x_i a na koniec prijme všetky zoradené hodnoty do registru *Z*. Po prijatí všetkých hodnôt do *Z* registru ich vypíše na štandardný výstup. Typy správ sú rozlíšené pomocou tagov *tagX*, *tagY*, *tagYIndex* a *tagZ*, ktoré boli zvolené pre prenos hodnôt z registrov *X*, *Y*, Y_{index} (číslo procesoru, ktorý vlastní dané číslo) a hodnoty registru *Z*. Komunikačný protokol je možné vidieť na Obrázku 2, kde je pre zjednodušenie uvažované zasielanie hodnôt registrov *Y* a Y_{index} v rovnakej správe. Keďže algoritmus nedokáže zoradiť postupnosti, v ktorých sa opakujú prvky, bolo nutné s hodnotou *Y* posielať aj index procesora Y_{index}, ktorému táto hodnota patrí. V prípade, že hodnoty registrov *X* = *Y* porovnávame aj index procesora s Y_{index} až v prípade že je Y_{index} menší inkrementujeme hodnotu v registry *C*. Po zoradení si procesory posúvajú hodnoty registru *Z* do prava a posledný procesor (P_n) zasiela hodnotu svojho registra Z master procesoru. Master procesor potom vypíše zoradenú postupnosť na výstup.



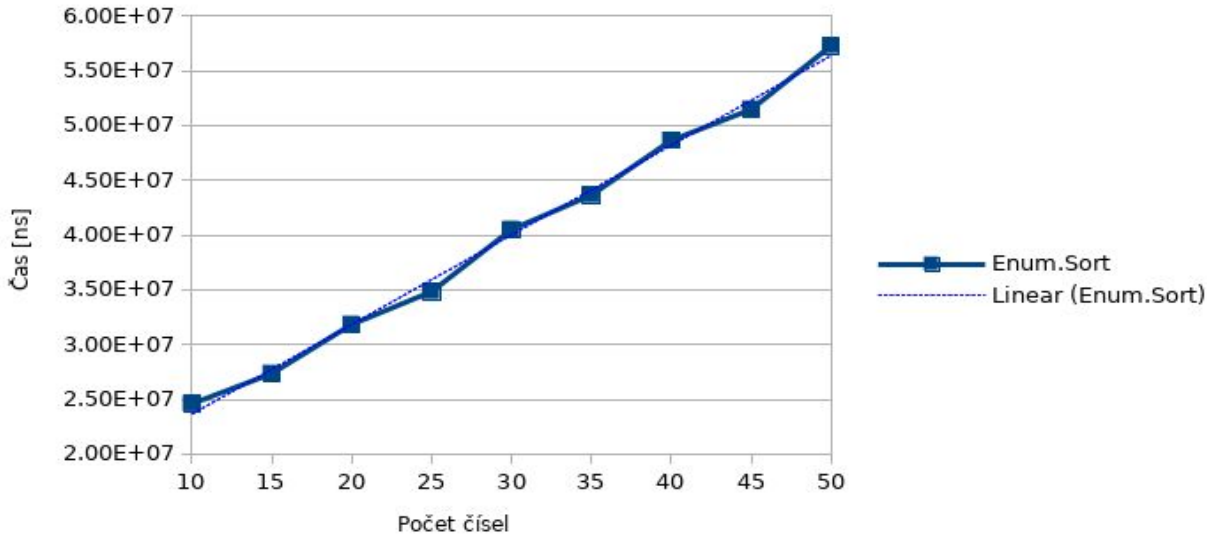
Obrázok 2: Komunikačná protokol

Experimenty

Meranie času behu algoritmu bolo realizované pomocou funkcie clock_settime(), ktoroá vynulovala stopky *tS* a *clock_gettime()*, ktorá vráti čas behu *tS* v nanosekundách. Tieto časy zapisoval program *es* do súboru *times.csv* v csv formáte, kde jednotlivé stĺpece obsahovali veľkosť postupnosti radených čísel, čas v sekundách a čas v nanosekundách. Pomocou skriptu bolo najskôr vygenerované 50 rôznych vstupných súborov s dĺžkami postupností 10, 15, 20, 25, 30, 35, 40, 45, 50. Ďalším skriptom prebehlo spustenie *es* programu so všetkými vygenerovanými vstupnými súbormi. Z nameraných časov bol potom vypočítaný priemer pre jednotlivé počty radených prvkov (viz. Obrázok 3). Z nameraných hodnôt bol vytvorený graf (viz Obrázok 4), kde je možné vidieť lineárny rast času s nárastom radených hodnôt. Tento graf potvrdzuje, že lineárne rastie čas výpočtu so zvyšovaním počtu zoraďovaných čísel.

Počet čísel	Enum.sort
10	24588377
15	27348220
20	31831068
25	34833077
30	40503759
35	43651126
40	48638769
45	51476232
50	57272613

Obrázok 3: Výsledky meraní



Obrázok 4: Rast času s rastom počtu čísel

Záver

Časová zložitosť implementovaného algoritmu Enumeration Sort na lineárnom poli n procesorov má lineárnu časovú, čo bolo potvrdené pomocou meraní s rôznymi počtami čísel. Na grafoch je vidieť že s rastom radených čísel lineárne rastie aj čas behu programu. Cena výpočtu je $O(n^2)$ čo nieje optimálne. Algoritmus sám o sebe nedokáže radiť postupnosti čísel, v ktorých sa opakujú rovnaké čísla. Preto bolo nutné použiť vylepšenie algoritmu a zohľadniť aj indexy procesorov, v prípade zoraďovania rovnakých čísel.