2. projekt - Bucket sort

Bc. Michal Ormoš
Paralelné a distribuované algoritmy (PRL) 2018/2019

22. marca 2019

Abstrakt

Dokumentácia k druhému projektu do predmetu Praralelné a distribuované algoritmy (PRL). Obsahuje popis zadania, rozbor a analýzu algoritmu Bucket sort, jeho implementáciu pomocou OpenMPI, zhrnutie časovej a pamäťovej náročnosti, výsledky experimentov.

1 Analýza algoritmu

1.1 Teoretický rozbor

Algoritmus Bucket sort spracujeme tak ako bol uvedený na prednáškach [1]. Algoritmus vykonáva radenie stromom procesorov s m listovými procesormi. Strom obsahuje $2^n - 1$ procesorov, takže počet potrebných procesov bude $(2 * log_2(n) - 1)$. Každý listový procesor obsahuje n/m radených prvkov a vie ich zoradiť optimálnym sekvenčným algoritmom. Každý nelistový procesor vie spojiť dve zoradené postupnosti svojich dvoch synov, optimálným sekvenčným algoritmom.

1.2 Praktický rozbor

Demonštrujme na príklade:
Príprava behu.

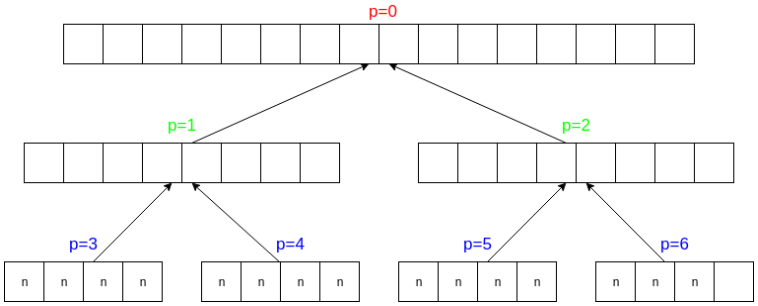
- Majme 15 prvkov, ktoré chceme zoradiť, $n = 15$. Vypočítame si teda počet potrebných procesov $(2 * log_2(n) - 1)$. Medzikrokom bude vypočítanie $log_2(n) = m = 3.9$. Uvažujme možnosť, že chceme zachovať strom vždy úplný. Teda počet procesorov vždy zaokrúhlime k najbližiej mocnine čisla 2. V našom prípade $m = 4$, teda $2 * m - 1 = 7$. Veľkosť listového uzlu vypočítame ako počet všetkých prvkov delene čislo m, $b = n/m = 15/4 = 3.75$. Majme na mysli, že každý listový procesor bude mať počet prvkov rovnaký, teda čislo vždy zaokrúhlime smerom nahor, $b = 4$. Pre počet čisel 15 potrebujeme teda 7 procesorov. Grafický načrt implementácie vyzerá nasledovne, obrázok 1.

Implementácia paralelného Bucket sort v C++:

- Vieme počet procesov, môžme teda úspešne spustiť OpenMPI, s počtom procesov 7. Každý proces si spustí rovnaký program paralelene a my vieme o aký proces sa jedná, identifikovaný svojim čislom od 0 do 7. Proces $p = 0$ bude náš nultý proces, ktorého ulohou bude načítať čisla zo súboru *numbers* a potom ich následne pomocou `MPI_Send` rozoslať listovým procesom, každé číslo inému procesu, zaradom po jednom. Následne sa z neho stáva rodič, ktorý už len čaká na správy od svojich sinov a to $p = 1$ a $p = 2$. Listové procesy $p3$ až $p6$ čakajú na

[1] https://www.fit.vutbr.cz/study/courses/PDA/private/www/h003.pdf

Obr. 1: Príklad stromu s 15 prvkami.

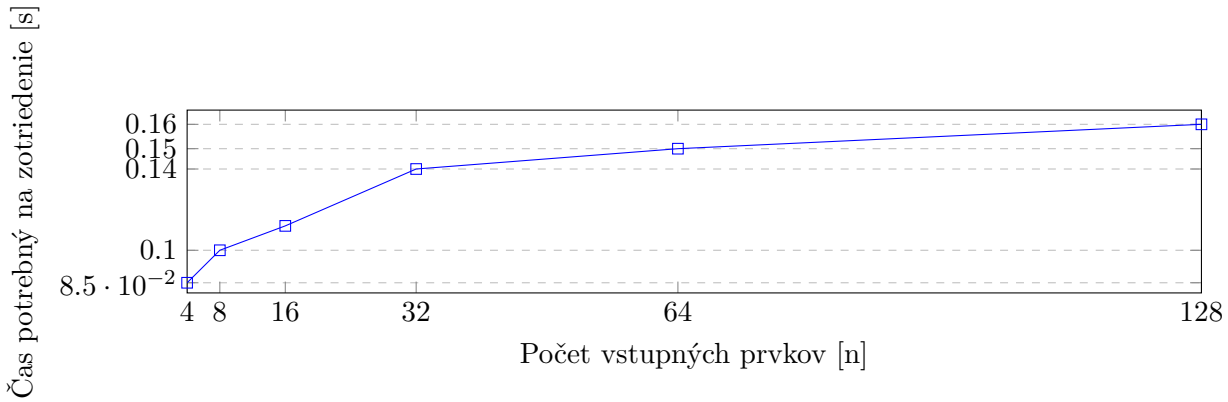


správu od procesu $p = 0$ pomocu `MPI_Recv`. Vedia, že im bude čísla posielať po jednom a keď skončí, pošle im hodnotu -1, ktorá signalizuje ukončenie zasielania prúdu dát. Teda proces $p = 3$ príjma čisla, a ukladá si ich postupne do vektoru, až príde číslo -1, zahodí ho a posunie sa v programe ďalej, kde tieto čísla pomocou sekvenčného algoritmu potriedi a následne si vypočíta, aký proces je jeho otec. Ak je daný list nepárny ako $p = 3$, tak vie, že jeho otec bude index neho samotného delené dvomi a zaokrúhlené smerom nadol, $3/2 = 1.5 = 1$. Ak je proces nepárny tak vie, že jeho rodič je index jeho samotného deleno 2, mínus hodnota 1. List teda posiela čisla rovnakým spôsobom ako ich prijal ďalej svojemu rodičovi pomocou `MPI_Send`, po jednom a ako posledné mu pošlu číslo -1, aby ich rodič vedel, že z posielaným skončili. Rodič čisla od ľavého a pravého syna prijme, sekvenčne ich spojí a pošle ďalej svojemu rodičovi rovnakým spôsobom, až sa dostanú k procesu $p = 0$ a on ich po spojení vypiše na *stdout*.

- (poznámka) možnosť posielať čisla po jednom sa možno javí ako pomalá, ale v praxi keď som si robil porovnania so spolužiakmi, ktorý posielali čisla ako celý vektor pohromade, sme vo výsledku dostali veľmi podobné časy, pre menši počet čisel, to bolo rýchlejsie a pre väčší počet čísel porovnateľne rýchle.
  (poznámka) 0 vstupných čisel považujem za chybu na vstupe, 1 vstupné číslo sa rieši jedným procesorom, 2 vstupné čísla rovnako jedným procesorom, 3 vstupné čísla a viac počitaním počtu procesorov ako bolo uvedené na prednáškach.

2 Experimenty

Časová zložitosť bola meraná pre počet prvkov 4, 816, 32, 64 a 128. Pre každé číslo bolo vykonaných 40 meraní, kde krejných 5 hodnôt bolo odstránených a zvyšných 30 bolo spriemerovaných. Tento priemer je nanesený v grafe. Experimenty boli vykonané na VUT FIT servery merlin. Ich čas behu bol meraný pomocou funkcií `OopenMPI` – `MPI_Barrier` a `MPI_Wtime` a to od spustenia behu progrmau C++, teda v potaz nemeria čas práce skriptu `text.sh`. Namerané teda bolo samotné spracovania súboru, rozdeľovanie hodnôt a triedenie prvkov spolu so spájaním prvkov.



3 Zložitosť algoritmu

Časová zložitosť: $t(n) = O(n) + O(n * log(n)) * O(log(n))$ //vstup + sort * hlbka stromu
Priestorová zložitosť: $p(n) = O(n)$

4 Záver

Výsledky merania neodpovedajú predpokladanej časovej zložitosti z predášok. To z dôvodu nedokonalej implementácie algoritmu. Výsledná časová zložitosť je ovplyvnená načitávaním čisel zo súboru a samotným triediacim algoritmom, ktorým je v jazyku C++ pre funckiu `sort` quicksort. Avšak zodpovedajú časovej zložitosti, ktorá vyplýva z experimentov.

A Sekvenčný diagram

Obr. 2: Sekvenčný diagram programu