2. projekt - Bucket sort

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

31. 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, spracovaný tak ako bol uvedený na prednáškach [1], 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 Rozbor zložitosti

- Z teoretického rozboru vyplýva počet procesorov **p(n)=O(log(n))**.
- Každý listový procesor číta $n/(log(n))$ prvkov. Triedenie prvkov listovými procesormi, pri použití optimálneho algoritmu $O(r.log(r)) = O((n/log(n))) * log(n/(log(n))) = O(n)$.
  Pri x-tej iterácií posielania prvkov rodičovi každý procesor na úrovni $i = (log(m)) - j$ spojí dve postupnosti o dĺžke $n/2i$, posledný krok teda trvá $O(n)$. Vo výsledku **t(n)=O(n)**.
- Cena **c(n)**= $t(n) * p(n)$ = **O(n*log(n))**.

2 Implementácia

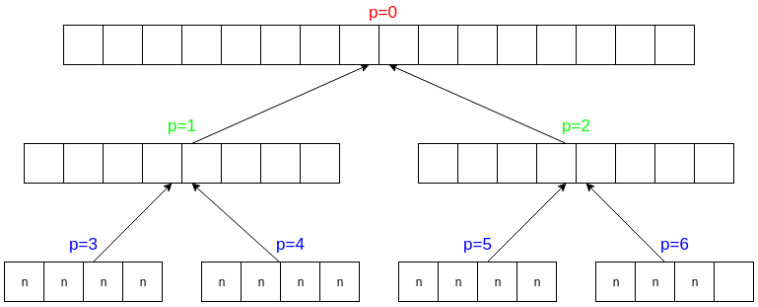
Príprava behu (demonštrujme na príklade):

- 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 vyššej 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

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

mysli, že každý listový procesor bude mať počet prvkov rovnaký, teda číslo vždy zaokrúhlime smerom nahor, $b = 4$. Pre počet čisel 15 potrebujeme teda 7 procesorov. Grafický načrt implementácie vyzerá nasledovne, obrázok 1.

Obr. 1: Príklad stromu s 15 prvkami.



Implementácia paralelného Bucket sort v C++:

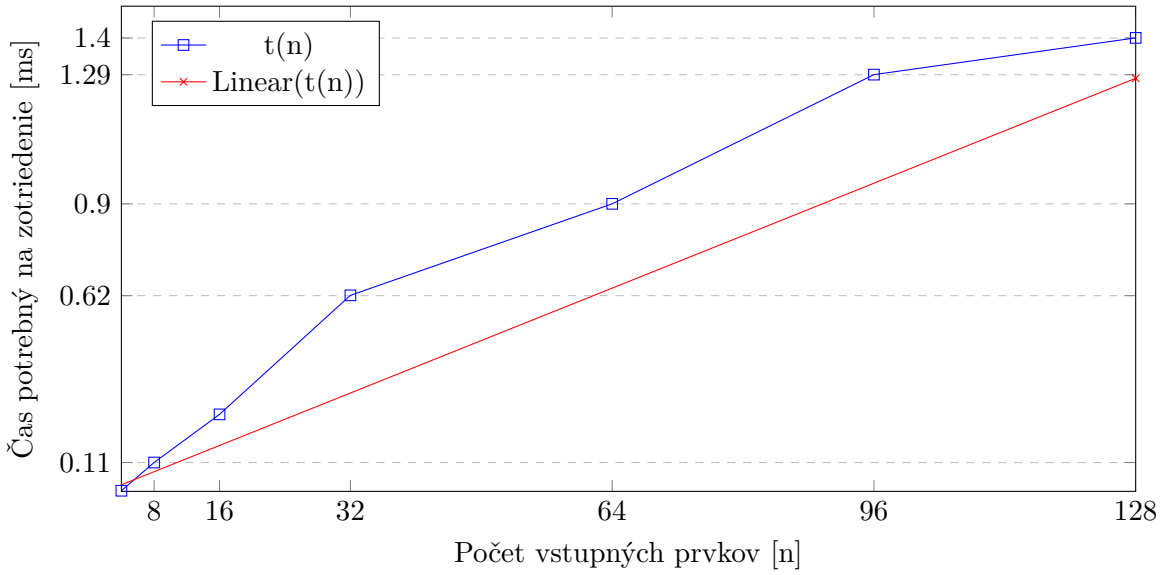
- Vieme počet procesov, môžme teda úspešne spustiť OpenMPI, s počtom procesov 7. Každý proces si spustí rovnaký program paralelene a my vieme o aký proces sa jedná, identifikovaný svojim číslom od 0 do 7. Proces $p = 0$ bude náš nultý proces, ktorého ulohou bude načítať čisla zo súboru *numbers* a potom ich následne pomocou `MPI_Send` rozoslať listovým procesom, každé číslo inému procesu, zaradom a po jednom. Následne sa z neho stáva rodič, ktorý už len čaká na správy od svojich synov a to $p = 1$ a $p = 2$. Listové procesy $p3$ až $p6$ čakajú na 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, $4/2 = 2 - 1 = 1$. List teda posiela čísla 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 si obdobným spôsobom indexy ľavého a pravého syna dopočíta a ich čísla 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 pri vykonávaní porovnanií 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.

3 Experimenty

Časová zložitosť bola meraná pre počet prvkov 4, 8, 16, 32, 64, 98 a 128. Pre každé číslo bolo vykonaných 20 meraní, kde krajných 5 hodnôt bolo odstránených a zvyšných 10 bolo šatisticky

vhodne upravených a spriemerovaných. Tento priemer je nanesený v grafe ako $t(n)$. Experimenty boli vykonané na VUT FIT servery `Merlin`. Ich čas behu bol meraný pomocou funkcie `OopenMPI` – `MPI_Wtime` a to od chvíle radenia / spojovanie prvkov po koniec behu procesu. Snažili sme sa teda nameraťbeh samotného algorimut Bucket sort, bez ostatných častí programu (počítanie prvkov, posielanie synom, atd.). Výsledky reprezentované graficky:



4 Záver

V rámci tohto projektu sa podarilo úspešne implementovať paralelný algoritmus na radenie postupností – Bucker Sort. Výsledkom je funkčný paralelný program schopný radiť postupnosti do maximálnej dĺžky 100 000 prvkov (tento limit je určený serverom Merlin, ktorý nám viac procesorov neposkytol). Uskutočnené merania ukázali, že reálna časová zložitosť programu je skutočne približne lineárna (ako sa očakávalo).

A Sekvenčný diagram

Obr. 2: Sekvenčný diagram programu