Dokumentácia projektu do predmentu Paralelné a distribuované algoritmy: Algoritmus Bucket Sort

Juraj Ondrej Dúbrava (xdubra03)

30. marca 2019

1 Úvod

Cieľom projektu bolo implementovať v jazyku C/C++ paralelný radiaci algortimus Bucket Sort a riadiaci skript na riadenie výpočtu.

2 Rozbor a analýza algoritmu

Algoritmus Bucket Sort je paralelný radiaci algoritmus, ktorý pracuje na stromovej topológií. Na najvyššej úrovni sa nachádza koreňový procesor a na najnižšej úrovni listové procesory.

V prvej fáze algoritmu sa rozošle príslušné množstvo dát jednotlivým listovým procesorom. Tie v druhom kroku algoritmu dáta zoradia optimálnym sekvenčným algoritmom a posielajú ich rodičovskému procesoru. V treťom kroku prebieha cyklus od 1 do $log(m)$, kde nelistové procesory na aktuálnej úrovni zoradia prijaté postupnosti, spoja ich do jednej optimálnym sekvenčným algoritmom a pošlú rodičovskému procesoru. Koreňový procesor nakoniec prijme dáta, spojí ich a zapíše do pamäti. Pre algoritmus je dôležitý údaj o počte listových uzlov, získame ho zo vzťahu $n = 2^m$, kde n je počet vstupných prvkov a m je počet listových procesorov. Z údaju o počte listov sa zo vzťahu $p = (2 * m) - 1$ určí počet procesorov v strome.

Asymptotická časová zložitosť algoritmu je $O(n)$. Táto zložitosť vychádza z toho, že každý listový procesor číta $n/(\log n)$ prvkov, zložitosť je $O(n/(\log n))$. Ďalej pri radení postupnosti použitím optimálneho sekvenčného algoritmu so zložitosťou $O(r.\log r) = O((n/\log n)).log(n/\log n)$ dostávame zložitosť $O(n)$. Tretí krok spájania postupností má zložitosť $O(n)$, posledná fáza zápisu má zložitosť $O(n)$. Cena algoritmu je $O(n.\log n)$, je teda optimálna. Počet použitých procesorov je $O(\log n)$ [1].

3 Implementácia algoritmu

Algoritmus bol implementovaný v jazyku C++ pomocou knižnice na paralelné výpočty *openMPI*. Pred spustením programu je potrebné vypočítať na základe počtu radených prvkov počet použitých procesorov. Výpočet je realizovaný v riadiacom skripte *test*. V prvom kroku výpočtu sa podľa vzorca $n = 2^m$ vypočíta počet listových procesorov. Výpočet je realizovaný počítaním mocnín čísla 2, výsledkom je najbližšia celá mocnina čísla 2 v prípade, že vstup nie je mocninou dvojky. Keďže v treťom kroku algoritmu chceme pracovať s počtom listov pre ktorý bude strom úplný, druhý krok výpočtu listových procesorov spočíva v tom, že sa zistí najbližšia väčšia mocnina čísla 2, tak aby hodnota m bola mocninou dvojky. Samotný počet procesorov sa získa dosadením získanej hodnoty do vzorca $p = (2 * m) - 1$.

Na začiatku programu je najskôr potrebné inicializovať prostredie *MPI* funkciou *MPI_Init()* a naplniť premenné pre počet procesorov a *rank* aktuálneho procesoru. Program je ďalej rozdelený do častí, ktoré vykonáva procesor s rankom 0 alebo ostatné procesory. Procesor s rankom 0, reprezentujúci koreňový procesor, najprv funkciou *count_numbers()* zistí počet radených hodnôt zo súboru *numbers*. Následne je z celkového počtu procesorov vypočítaný počet listov a veľkosť tzv. bucketu, ktorý sa má poslať listom. Veľkosť bucketu je zaokrúhlená na najbližšie celé číslo.

Ďalej je potrebné poslať dáta jednotlivým listom. Keďže algoritmus nepočíta s tým, že strom musí byť úplný, táto situácia bola vyriešená výpočtom počtu listov tak, aby tento počet bol mocninou 2. Následne chceme listom

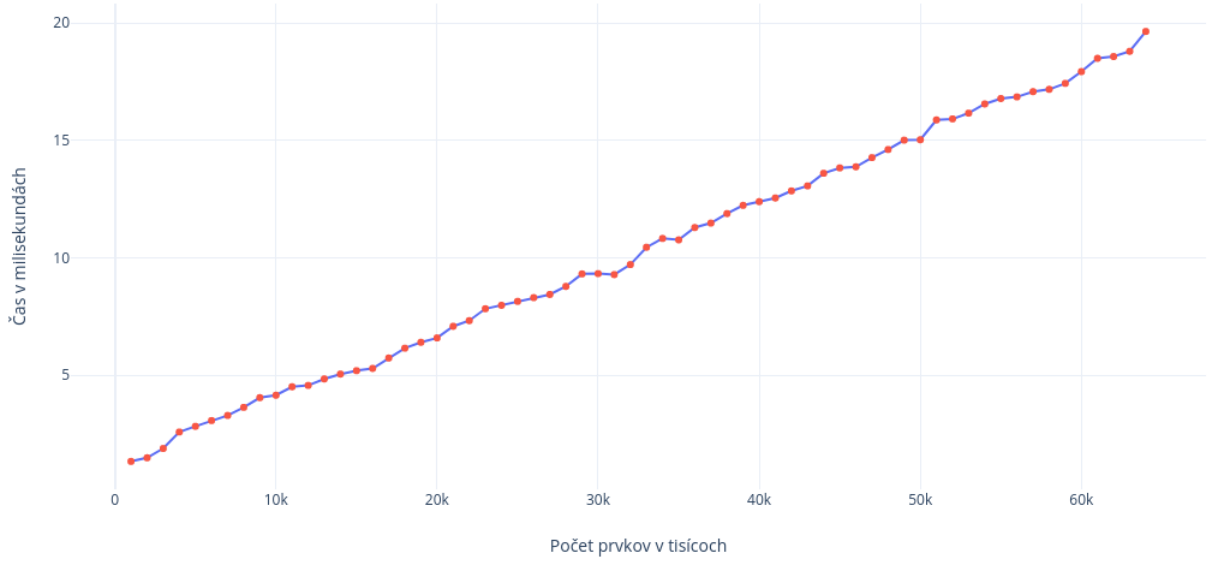
poslať rovnaké množstvo dát. Tým, že sme upravili strom, sme dostali väčší počet listov, ale podľa hodnoty veľkosti bucketu by dáta nebolo možné rovnomerne rozdeliť medzi listy. Táto situácia je vyriešená tak, že sa spočíta nový počet radených prvkov ako $pocet\_listov * velkost\_bucketu$. Podľa tejto veľkosti sa alokuje pole inicializované hodnotou 0, kde sú načítané vstupné dáta. Tým dosiahneme, že každý list dostane bucket rovnakej veľkosti, avšak niektoré buckety môžu mať naviac doplnené nuly. Radené prvky sú načítané funkciou *load_numbers()*. Pri 1 a 2 prvkoch sa použije len koreňový procesor. Inak procesor vykoná 2 *for* cykly, v prvom pošle všetkým procesorom novú veľkosť počtu radených prvkov a v druhom posiela dáta listovým procesorom. Odosielanie je realizované funkciou *MPI_Send()*, indexovanie synov je realizované tak, že ľavý syn má rank $2 * rank + 1$ a pravý syn $2 * rank + 2$. Po odoslaní všetky procesory prijímajú veľkosť radeného poľa funkciou *MPI_Recv()* a každý si vypočíta počet listov. Jednotlivé procesory overia, či sú listové a ak áno, z veľkosti vstupu si vypočítajú veľkosť bucketu a alokujú pomocné pole danej veľkosti. Následne prijímajú dáta funkciou *MPI_Recv()* a uložia ich do pomocného poľa. Každý list pole zoradí optimálnym sekvenčným algoritmom. Na radenie je použitá funkcia *sort* zo štandardnej knižnice, algoritmus implementovaný touto funkciou má podľa manuálu asymptotickú časovú zložitosť $O(n.log\,n)$. Po zoradení listy posielajú zoradenú postupnosť rodičovskému procesoru funkciou *MPI_Send()*.

Po odoslaní dát listovými procesormi nasleduje *for* cyklus od hodnoty 1 po hodnotu $log(m)$, ktorým sa prechádzajú jednotlivé úrovne stromu. V každej iterácií sa makrom *GET_LEVEL(x)* zistí, či sa procesor s daným rankom nachádza na aktuálnej úrovni stromu. Ak áno, zistí počet procesorov na aktuálnej úrovni ako $1 << uroven$, z tejto a z hodnoty nového počtu radených prvkov získa veľkosť svojho bucketu. Následne si alokuje 3 pomocné polia, dva o polovičnej veľkosti bucketu a jedno o veľkosti bucketu. Nasleduje prijatie dát od oboch synov funkciou *MPI_Recv()*, zoradenie dvoch pomocných polí funciou *sort* a spojenie do jednej zoradenej postupnosti funkciou *merge* zo štandardnej knižnice s asymptotickou časovou zložitosťou $O(n1 + n2)$, kde $n1, n2$ sú dĺžky spojovaných postupností. Keď je dosiahnutá úroveň s koreňovým procesorom, ten prijme dáta od synov, zoradí ich, spojí a následne vypíše.

4 Experimenty

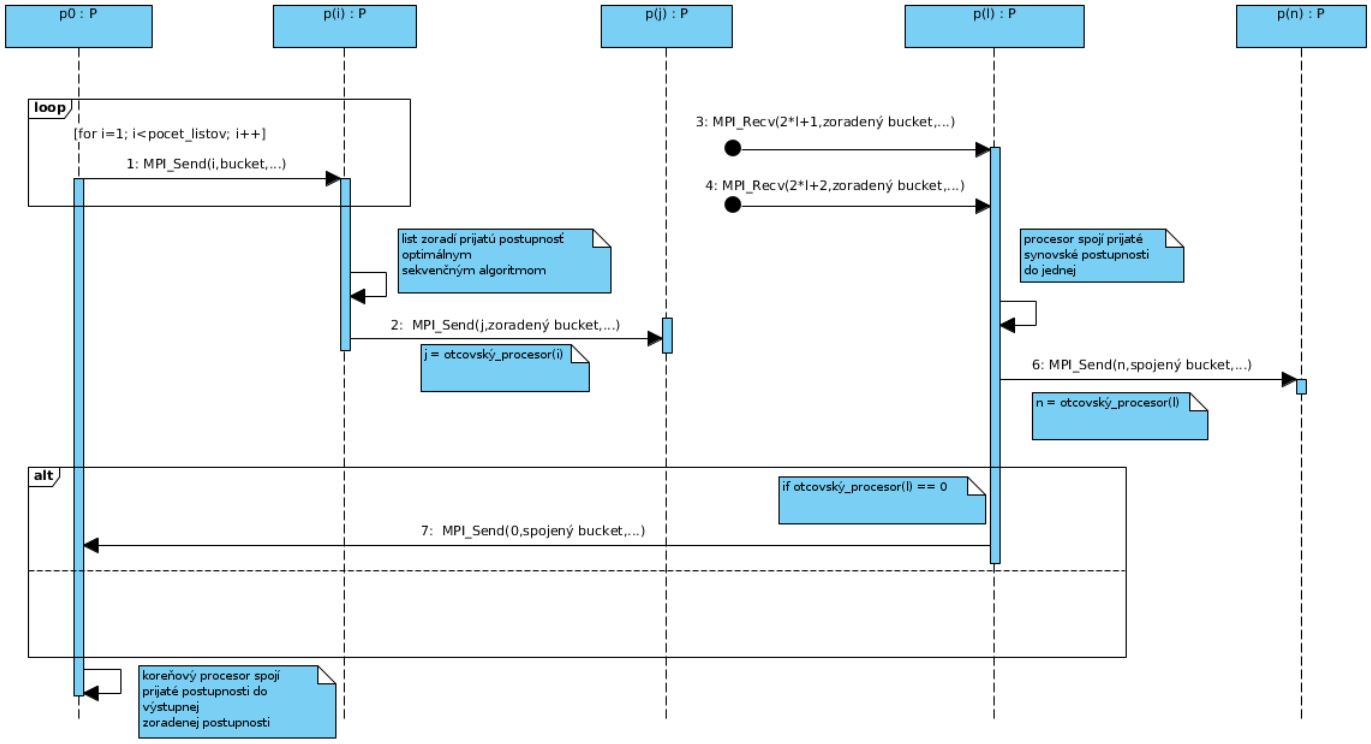
Algoritmus bol testovaný na overenie časovej zložitosti na vstupoch o veľkosti od 1 do 64 tisíc s krokom 1000 prvkov. Pre každú hodnotu počtu prvkov bolo vykonaných 100 meraní na tej istej postupnosti hodnôt, z ktorých bol následne vybraný medián ako výsledná hodnota. Generovanie náhodných čísel bolo uskutočnené pomocou utility *urandom* na urýchlenie výpočtu. Meranie času behu algoritmu bolo realizované funkciou *MPI_Wtime()*, meranie začalo pred rozoslaním dát listom a skončilo po prijatí dát koreňovým procesorom.

Získané hodnoty merania sú zobrazené v grafe na obrázku 2.



Obr. 1: Graf merania pre jednotlivé počty vstupných prvkov.

5 Komunikačný protokol



Obr. 2: Sekvenčný diagram reprezentujúci vzájomnú komunikáciu medzi procesmi.

6 Záver

Bucket Sort je paralelný radiaci algortimus s optimálnou cenou a časovou zložitosťou $O(n)$. Túto zložitosť mali overiť experimenty s rôznymi počtami prvkov. Všetky merania boli uskutočnené na serveri Merlin. Z grafu na obrázku 2 je vidieť, že výsledky jednotlivých meraní nemajú úplne stopercentný lineárny priebeh, pretože server Merlin nemá potrebný počet logických procesorov použitých pri meraniach a takisto je server rôzne vyťažený, čo ovplyvní jednotlivé behy programu. Takéto výkyvy je možné v grafe pozorovať, aj keď bolo snahou minimalizovať ich vplyv uskutočnením 100 meraní pre každú hodnotu počtu radených prvkov.

Referencie

[1] HANÁČEK, Petr: Distribuované a paralelní algoritmy a jejich složitost, algoritmy řazení, select. [cit. 2019-03-28]. Dostupné z `http://www.fit.vutbr.cz/study/courses/PDA/private/www/h003.pdf`