Paralelní a distribuované algoritmy

Projekt 2 - Merge-splitting sort

Ľubomír Gallovič - xgallo03

1 Úvod

Cieľom projektu bolo implementovať algoritmus Merge-splitting sort v jazyku C/C++ s použitím knižnice Open MPI a otestovať jeho funkcionalitu.

2 Rozbor a analýza

Merge-splitting sort pracuje na lineárnom poli procesorov. Algoritmus je podobný algoritmu Odd-even transposition sort, rozdiel je v tom, že každý procesor môže mať na starosti viac ako jeden prvok. Na algoritme pracuje ľubovoľný počet procesorov p, pričom každému je na začiatku odoslané rovnomerné množstvo prvkov ktoré bude zoradzovať. Teda pri poli veľkosti n dostane každý procesor n/p prvkov. V prípade, že počet prvkov a procesorov je nesúdeliteľný sú na koniec doplnené falošné prvky. Po prijatí každý procesor zoradí svoje prvky optimálnym sekvenčným algoritmom (napr. Quick sort, Merge sort), čo má časovú zložitosť O((n/p)*log(n/p)). Po zoradení prechádza algoritmus do fázy paralelného radenia. Najprv všetky procesy s párnym číslom odošlú ich zoradené prvky svojmu ľavému susedovi, čo trvá lineárny čas, teda O(n/p). Nepárne procesy tieto čísla príjmu a zoradia dve zoradené postupnosti (svoju a susedovu) do jednej 2x dlhšej postupnosti. Tento krok trvá 2*n/p, teda opäť lineárny čas. Druhú polovicu tejto novej postupnosti odošlú späť svojmu pravému susedovi. Táto fáza sa opakuje znova, avšak tento krát svoje prvky odosielajú nepárne procesy a párne robia výpočet. Po n/2 iteráciách týchto dvoch fází sú všetky prvky zoradené. Po ukončení dostane procesor 0 od ostatných procesorov ich prvky, ktoré spojí do kontinuálnej postupnosti.

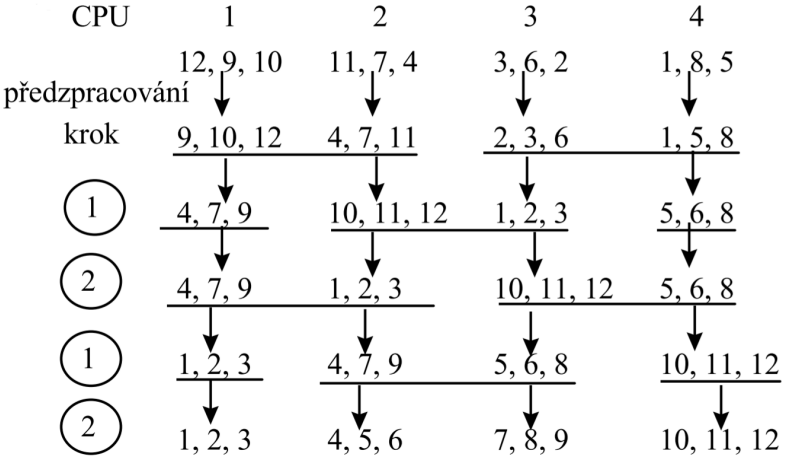

Figure 1: Princíp Algoritmu

Algoritmus má teda časovú zložitosť O((n/p)*log(n/p)) + O(n). Znamená to, že zložitosť závisí od pomeru počtu procesorov a počtu prvkov. Pri veľkom počte procesorov vypadne prvá časť zložitosti a ostane lineárna časová zložitosť. Naopak pri málo procesoroch bude zložitosť linearitmická. Cena algoritmu je O(n*log(n)) + O(n*p), čo je optimálne pre $p \leq \log(n)$.

3 Implementácia

Po spustení programu je potrebné načítať jednotlivé čísla zo vstupného súboru. Túto činnosť má na starosti proces 0, ktorý si prvky uloží do poľa a zavrie súbor. Následne vypočíta koľko prvkov bude prináležiť každému procesoru a v prípade nesúdeliteľnosti n a p doplní na koniec poľa falošné prvky. Falošné prvky majú hodnotu 300, pretože rozmedzie radených čísel je len 0-255. Proces 0 pošle ostatným procesom hodnotu n/p, aby vedeli aké veľké pole si alokovať. Použije na to funkciu `MPI_Bcast()`. Následne rozošle jednotlivé prvky poľa pomocou funkcie `MPI_Scatter()`. Na sekvenčné zoradenie svojich prvkov použije každý procesor funkciu `sort()` z knižnice `<algorithm>`, ktorá má optimálnu linearitmickú časovú zložitosť. Po sekvenčnom zoradení si začnú procesory predávať navzájom svoje prvky a zoradzovať ich do väčšej postupnosti. V prvej fáze pošlú párne procesy svoje zoradené pole svojim ľavým susedom pomocou funkcie `MPI_Send()` a čakajú na zoradené prvky funkciou `MPI_Recv()`. Nepárne procesy zoradia svoje a susedove prvky funkciou `sortAndSplit()`, ktorá má lineárnu časovú zložitosť a funguje podobne ako v algoritme Merge sort. Po zoradení dvoch zoradených polí do jedného väčšieho poľa je druhá časť tohto poľa poslaná spätne pravému susedovi. Druhá fáza algoritmu pracuje analogicky. Po n/2 iteráciách má každý procesor vo svojom poli uložené správne zoradené prvky, ktoré pošle procesoru 0 pomocou funkcie `MPI_Gather()`. Falošné prvky sú odstránené a zoradené prvky sú vypísané na výstup. Pre ukončenie je volaná funkcia `MPI_Finalize()`.

Komunikácia medzi procesmi je popísaná v sekvenčnom diagrame:

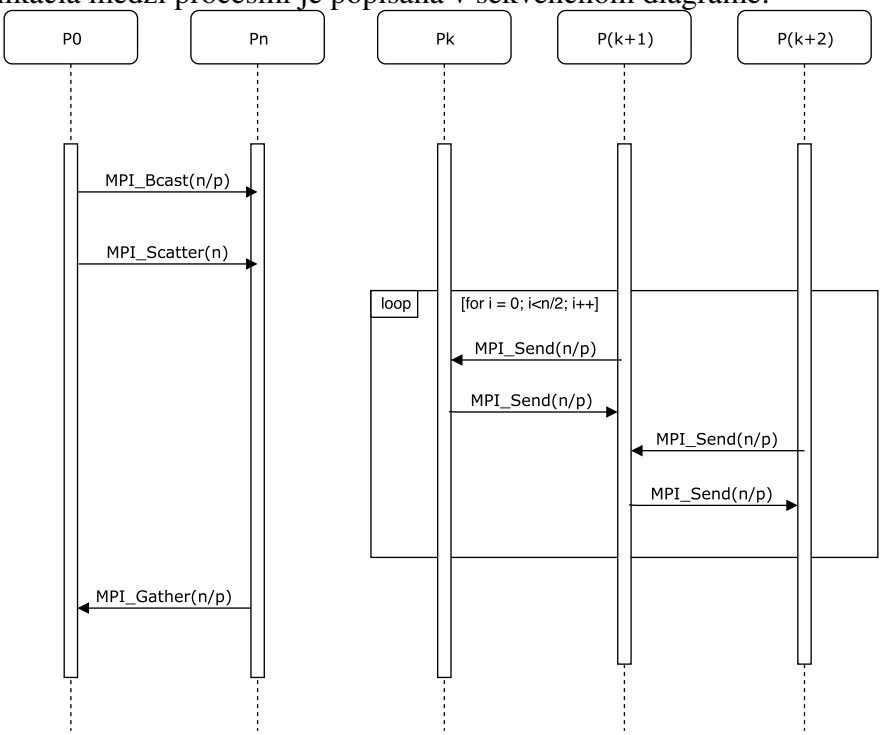

Figure 2: Princíp Algoritmu

4 Experimenty

Program bol testovaný na serveri merlin, a meraný bol čas od načítania znakov zo súboru do získania zoradeného poľa procesom 0. Počiatočné a koncové časy boli zaznamenané s využitím knižnice `<chrono>`. Cieľom meraní bolo zistiť ako sa mení doba výpočtu v závislosti od počtu prvkov, ktoré majú byť zoradené, čím experimentálne zistíme časovú zložitosť algoritmu. Ďalej sa snažíme zistiť ako sa tieto časy menia v závislosti od počtu procesov. Zisťovná bola časová zložitosť pre 1, 2, 3, 6, 8 a 10 procesorov. Pre každý počet procesorov bol meraný čas výpočtu pre veľkosti poľa 1 až 250 prvkov. Meranie v každej konfigurácii procesorov a prvkov prebehlo 100 krát, pričom z nameraných hodnôt bol vypočítaný aritmetický priemer. V grafe sa nachádzajú trendové čiary pohyblivého priemeru s veľkosťou okna 5 prvkov, aby bola oscilácia v nameraných časoch menej výrazná.

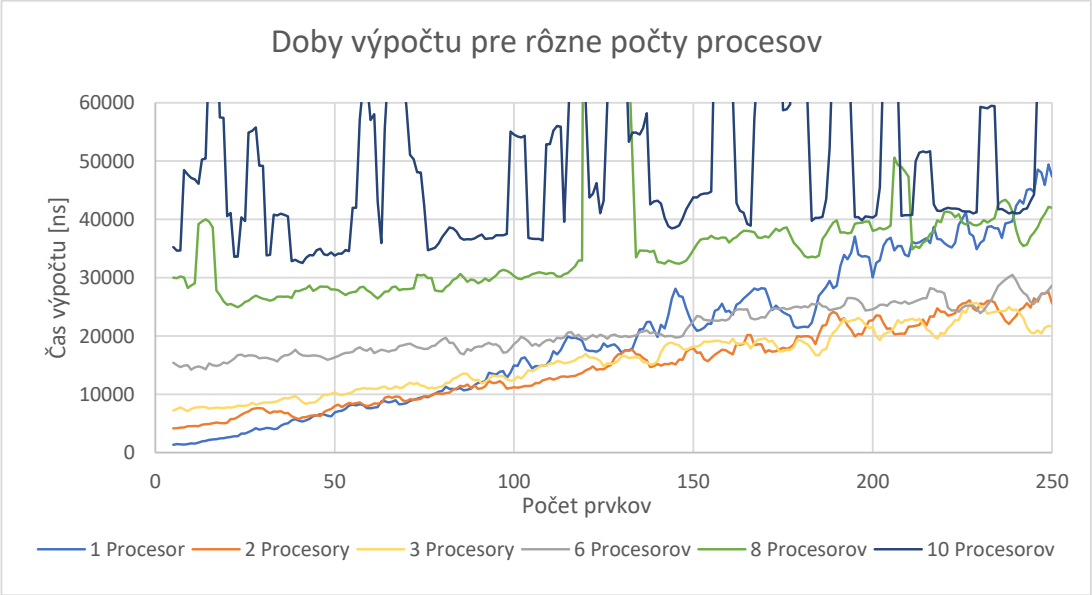

Figure 3: Namerané výsledky

5 Záver

Na nameraných výsledkoch môžeme vidieť, že pri lineárnom raste počtu prvkov približne lineárne rastie aj časová zložitosť algoritmu. U sekvenčného radenia (1 procesor) je rast strmší, a viac pripomína linearitmickú zložitosť. Riešenia s viacerými procesormi dosahujú lepšie výsledky ako sekvenčné riešenie, avšak individuálne rozdiely paralelných riešení sú relatívne nízke. U nízkeho počtu prvkov je však lepšie sekvenčné riešenie, čo môže byť spôsobené vyššou réžiou komunikácie procesorov. Namerané časy u ôsmych a desiatich procesoroch sú horšie a majú vysoký rozptyl v časoch výpočtu, čo indikuje možné nesprávne fungovanie paralelizmu. Tento trend zhoršovania času a zvyšovania rozptylu sa zvýrazňuje aj u vyšších počtoch procesorov (nie sú zahrnuté v grafe) a môže to byť dôsledkom toho, že nie každému procesoru je pridelené vlastné jadro. Namerané výsledky teda čiastočne potvrdili teoretickú zložitosť algoritmu, hlavne pri menšom počte procesorov, pri mnohých procesoroch (10+) sa namerané výsledky s teoretickými nezhodujú.

References

[1] P. Hanáček, *Distribuované a paralelní algoritmy a jejich složitost, algoritmy řazení*. 2005, [Online]
https://www.fit.vutbr.cz/study/courses/PDA/private/www/h003.pdf