PRL - Merge-splitting sort

Patrik Segedy, xseged00

8.4.2017

1 Analýza algoritmu

Algoritmus Merge-splitting-sort slúži na zoradenie postupnosti čísel pomocou lineárneho poľa procesorov $p(n) < n$, kde p je počet procesorov a n počet radených prvkov.

Je variantou algoritmov Odd-even transposition sort a Odd-even merge sort. V porovnaní s algoritmom Odd-evevn transposition sort je porovnanie a výmena prvkov nahradená operáciami spojenia a rozdelenia postupnosti.

Každý procesor sa stará o viac prvkov, konkrétne o n/p prvkov, kde n je počet prvkov a p je počet procesorov. Postupnosť je zoradená po p/2 iteráciách.

Postup algoritmu:

1. Každý procesor zoradí svoju postupnosť optimálnym sekvenčným algoritmom.
2. V p/2 iteráciách:
    a. Všetky nepárne procesory paralelne spoja svoju zoradenú postupnosť so zoradenou postupnosťou pravého suseda do jednej zoradenej postupnosti optimálnym sekvenčným algoritmom.
    b. Prvú (menšiu) polovicu si procesor nechá, druhú polovicu pošle pravému susedovi.
    c. Všetky párne procesory vykonajú rovnaké kroky ako nepárne procesory.
3. Na procesoroch sa nachádza zoradenú postupnosť

1.1 Teoretická zložitosť

Časová zložitosť

- zoradenie optimálnym sekvenčným algoritmom $O((n/p) \times log\ (n/p))$
- prenos postupnosti od pravého suseda $O(n/p)$
- spojenie 2 zoradených postupnosti $O(2 \times n/p)$
- prenos dát pravému susedovi $O(n/p)$

$$t(n) = O((n / p) \times log\ (n / p)) + O(n) = O((n \times log\ n) / p) + O(n)$$

Priestorová zložitosť

- Každý procesor potrebuje pole o veľkosti n / p - $p \times O(n / p) = O(n)$
- Každý procesor potrebuje pole o veľkosti $2 \times n / p$ pre uloženie spojenej postupnosti $p \times O(2 \times n / p) = O(2 \times n)$

$$s(n) = O(2 \times n) + O(n)$$

Cena algoritmu

$c(n) = t(n) \times p = O(n \times log\ n) + O(n \times p)$

Cena je optimálna pre $p \leq log\ n$

2 Implementácia

Pre implementáciu bol použitý jazyk C++ a knižnica OpenMPI. Táto knižnica umožňuje riešenie paralelných algoritmov pomocou posielania správ medzi procesormi.

Najprv procesor s rankom 0 načíta obsah vstupného súboru. Ak počet čísel nie je deliteľný počtom procesorov, pole sa doplní hodnotami 256, tak aby počet čísel bol deliteľný počtom procesorov. Pri výpise len počet čísel zadaný na vstupe. Následne tento procesor rozošle dáta na jednotlivé procesory pomocou funkcie `MPI_Scatter()`. Po tomto kroku má každý procesor `n/p` prvkov.

Potom každý procesor zoradí svoju postupnosť optimálnym sekvenčným algoritmom, použitím `std::sort()`.

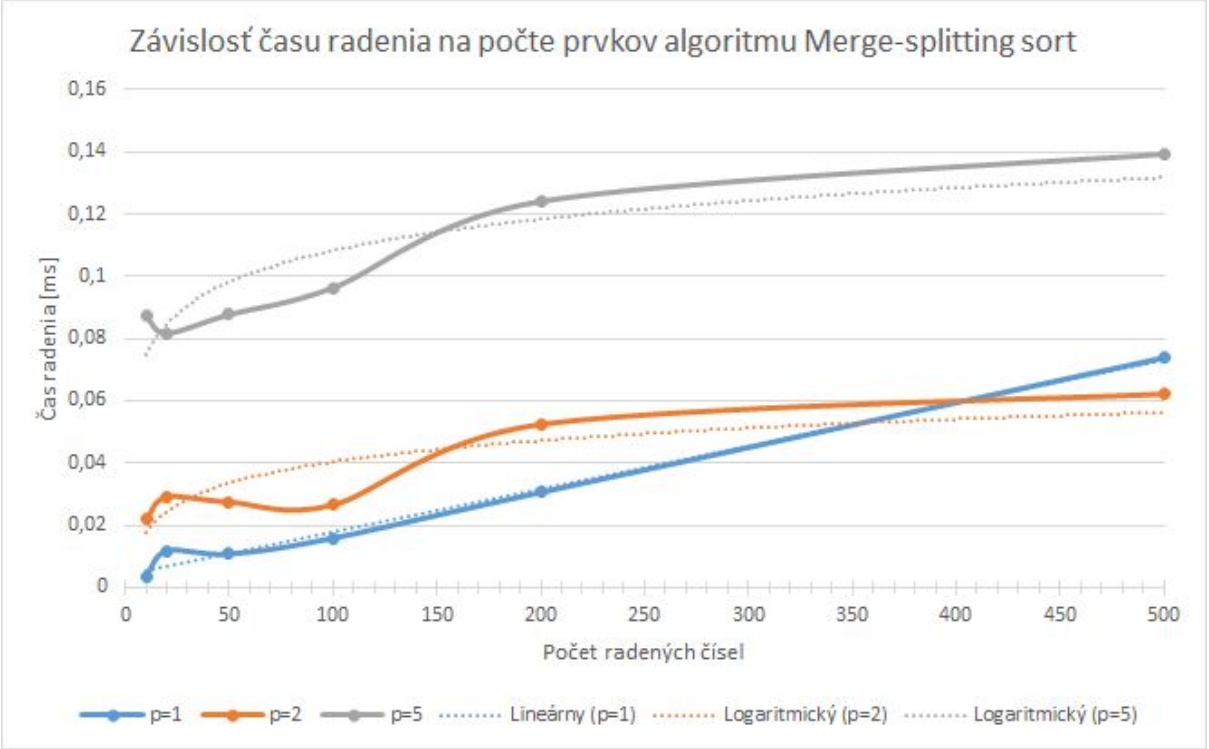
Následne algoritmus pracuje samostatne pre párne a nepárne procesory. Pričom v obidvoch prípadoch sa vykonáva rovnaké telo programu, preto opíšeme proces len pre párne procesory. Každý párny procesor, ktorý nie je posledný, pošle svoje pole pravému susedovi pomocou `MPI_Send()` a funkciou `MPI_Recv()` očakáva od neho jeho hodnotu, teda menšiu polovicu spojených postupnosti. Nepárny procesor príjme dáta od ľavého suseda (párneho procesoru), túto postupnosť pomocou `std::merge` spojí so svojou postupnosťou do jednej zoradenej postupnosti. Polovicu spojenej postupnosti, ktorá obsahuje menšie hodnoty pošle ľavému susedovi.

Posledným krokom je spojenie polí z každého procesoru do jedného poľa v procesore s rankom 0. Pre tento účel je využitá funkcia `MPI_Gather()`. Procesor s rankom 0 vypíše zoradenú postupnosť na výstup.

3 Experimenty

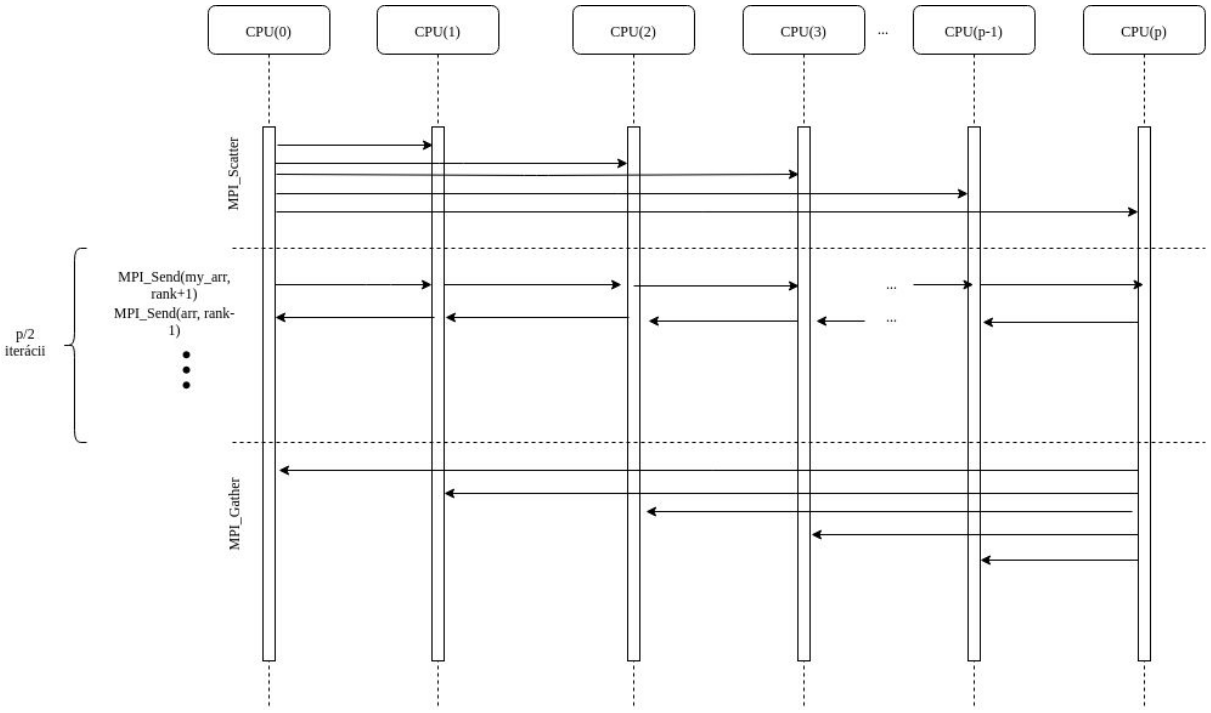
Pre overenie časovej zložitosti je v kóde meraný čas behu algoritmu bez načítania súboru, výpisov, rozoslania a spájania dát. K tomuto účelu bola použitá knižnica `std::chrono`. Čas behu bol meraný v milisekundách.

Kvôli zrýchleniu získania dát bol pri experimentoch zmenený skript `test.sh` tak, aby pre vytvorenie náhodných dát nepoužíval `/dev/random` ale pseudonáhodné dáta z `/dev/urandom`.



4 Komunikačný protokol

Na nasledujúcom sekvenčnom diagrame je zobrazený komunikačný protokol, ako si procesy zasielajú správy.



5 Záver

Výsledkom je implementovaný algoritmus Merge-splitting sort v jazyku C++ s použitím knižnice OpenMPI. Experimenty nad implementovaným algoritmom potvrdili teoretickú zložitosť algoritmu.