Merge-splitting sort

Popis algoritmu

Algoritmus Merge-splitting sort pracuje nad linearní topologií procesorů. Počet procesorů musí být menší než počet vstupních znaků a každý prvek obsluhuje právě n/p čísel. Algoritmus pomocí operace merge-split porovnává a vyměňuje prvky jednolivých procesorů ve variantě sudý/lichý procesor. Po p/2 iteracích je posloupnost seřazena.

Obecný kód algoritmu

1. Řídící proces rozdělí vstupní posloupnost na n/p posloupností a rozešle jednotlivým procesům
2. Každý procesor seřadí paralelně svojí posloupnost sekvenčním algoritmem
3. Střídají se paralelně varianty sudá/lichá, dokud není celá posloupnost seřazena (po p/2 iteracích)
   - 3.1. Varianta sudá
     - 3.1.1. Každý procesor, který není dělitelný 2 (je lichý), zašle svojí posloupnost procesoru pi-1
     - 3.1.2. Každý procesor, který je dělitelný 2 (je sudý), přijme posloupnost z procesoru pi+1 (pokud takový proces existuje)
     - 3.1.3. Sudý procesor spojí posloupnost Si a Si+1 do setříděné sekvence
     - 3.1.4. Setříděnou sekvenci rozdělí, první polovinu si ponechá, druhou polovinu odešle na procesor pi+1
   - 3.2. Varianta lichá
     - 3.2.1. Každý procesor, který je dělitelný 2 (je sudý), zašle svojí posloupnost procesoru pi-1
     - 3.2.2. Každý procesor, který není dělitelný 2 (je lichý), přijme posloupnost z procesoru pi+1 (pokud takový proces existuje)
     - 3.2.3. Lichý procesor spojí posloupnost Si a Si+1 do setříděné sekvence
     - 3.2.4. Setříděnou sekvenci rozdělí, první polovinu si ponechá, druhou polovinu odešle na procesor pi+1

Analýza algoritmu

Dle výše uvedeného algoritmu je vidět že časová složitost algoritmu je lineární (1. Krok lze udělat v čase logaritmickém *O(log n)*, 2. krok je možné provést v čase linearitmickým *O(n log n),* 3. krok je možné provést v čase lineárním *O(n).*

- Časová složitost - $t(n)$= *O(log n)+ O(n log n) + O(*n*) = O((n log n)) + O(n)*
- Prostorová složitost – *p(n) = O(n)/p*
- Cena algoritmu - $c(n)$= *O((n log n)) + O(n.p)*
- Cena algoritmu je tedy optimální pro p ≤ log n

Implementace

Pro implementaci byl využit jazyk C++ a knihovna OpenMPI.

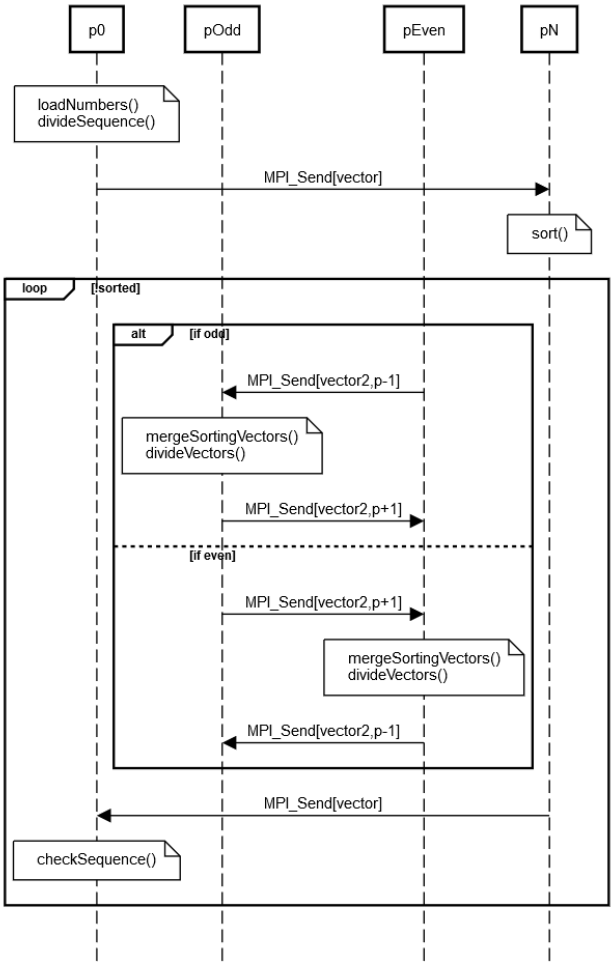
Po spuštění programu začne řídící procesor (procesor 0) číst vstupní soubor a hodnoty v souboru postupně přidává do vektoru (jednorozměrné pole). Následně na základně vstupních čísel a počtu procesů rozdělí posloupnost na n/p vektorů (pokud existuje zbytek po dělení, tak přiděluje vektorům o 1 číslo navíc, dokud není zbytek 0). Tyto vektory následně rozešle všem procesorům (včetně sebe). Nejdříve však zašle velikost vektoru (velikosti se mohou lišit, kvůli zbytku po dělení) a až poté vektor samotný.

Poté už začínají pracovat ostatní procesory, nejdříve přijmou vektor z řídícího procesoru, ten si seřadí sekvenčně pomocí algoritmu *std::sort* třídy *algorithm*. Následně začíná cyklus, který se opakuje do té

doby, dokud nejsou všechna čísla seřazená. Cyklus má 2 režimy Sudý/lichý tyto režimy se po jedné iteraci střídají. Pokud je režim sudý, tak liché procesory zasílají své vektory prvku p-1. Sudé prvky zase tyto vektory přijímají (za předpokladu že existuje p+1), následně ze 2 seřazených posloupností vytvoří jednu posloupnost (postupně se porovnávají nejnižší prvky obou vektorů a ten menší je zaslán na nový vektor). Tento nový seřazený vektor se rozdělí opět na 2 části (podle původních velikostí) a 2. část (s vyššími hodnotami) zašle zpátky na procesor p+1. V režimu Lichý funguje algoritmus obdobně, akorát sudé procesory zasílají prvky na liché (p-1) a ty zpracovávají zbytek.

Poté co proběhne jedna iterace, zašlou všechny procesory svoje vektory na řídící procesor, ten spojí všechny vektory do jednoho společného vektoru a kontroluje, zda jsou již prvky seřazené. Pokud ano, je algoritmus ukončen, jinak přepne režim cyklu a pustí další iteraci. Zároveň pošle zprávu všem dalším procesům o stavu výpočtu (zda je hotový, případně jaký bude následující režim).

Komunikační protokol



Obrázek 1 - Sekvenční diagram algoritmu

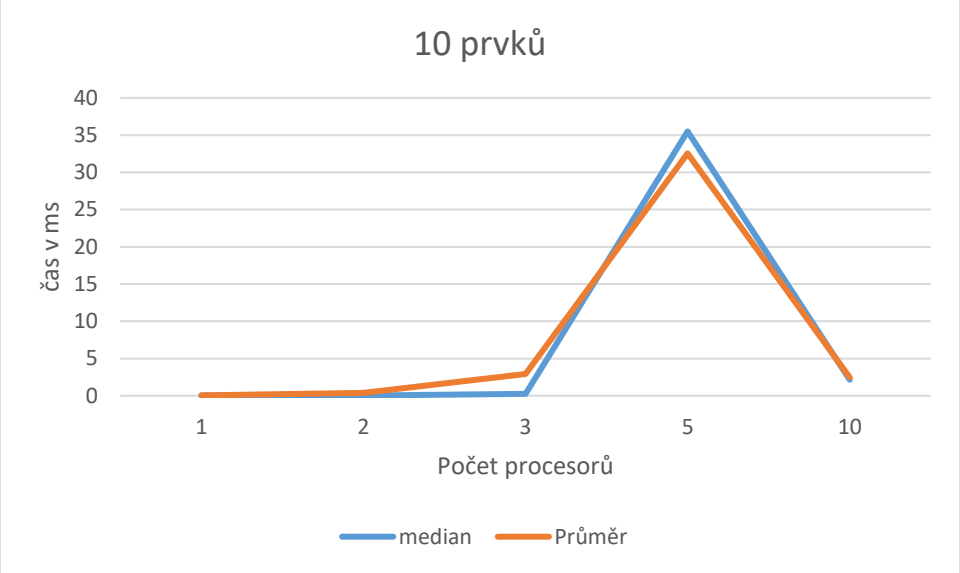
Naměřené výsledky

Pro měření výsledků jsem použil rozdíl proměnných typu *MPI_Wtime()*. V programu je měřen čistě čas měření, tím pádem čas čtení souboru řídícím procesem, výpis seřazených hodnot atd. není v měření uvedeno.

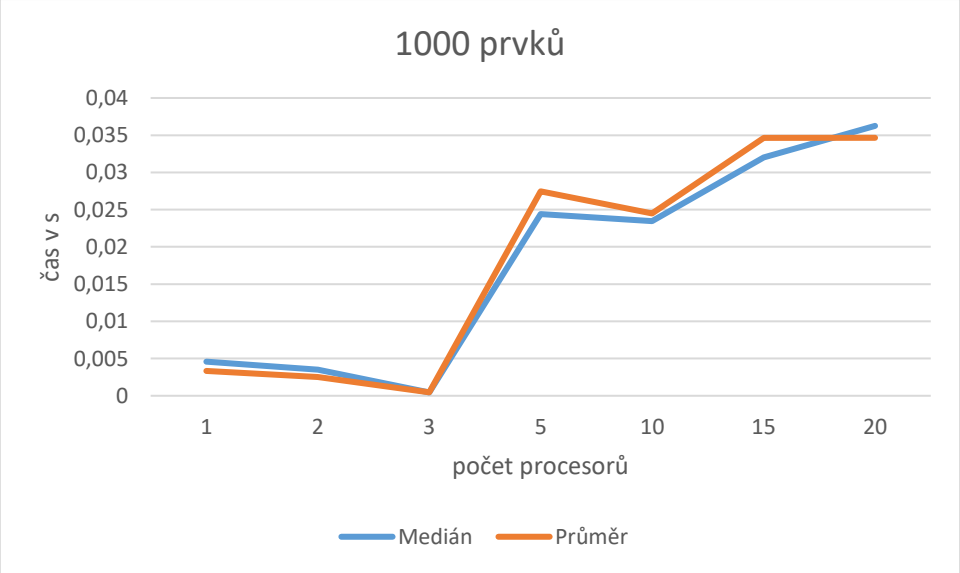
V měření jsem se zaměřil na měření 10 a 1000 prvků, každou hodnotu jsem měřil 10x na několika procesorech, z kterých jsem vytvořil průměr a medián. Navíc některé hodnoty, které byly několik

řádů od svého průměru jsem ani do měření nezapočítával. Celé měření probíhalo na školním portálu Merlin.

Při řazení 10 prvků, bylo podle mediánu nejrychlejší využití 2 procesorů. Když bylo využito 5 procesů, tak průměrný čas měření byl téměř 1600x větší. Časové složitosti jsou tedy v tomto případě extrémně rozdílné



Řazení 1 000 prvků probíhalo nejrychleji na 3 procesorech, nejpomaleji na 20 procesorech, ovšem rozdíl je „pouze“ 80 ti násobný, není tedy tak markantní jako v přechozím případě.



Závěr

Podle naměřených hodnot se zdá, že mnou vypracovaný algoritmus pracuje nejrychleji na menším počtu procesorů. V případě zapojení dalších procesorů je časová složitost vyšší, než když se měří pouze na 1 procesoru (sekvenční zpracování). Předpokládám že je to způsobeno „pomalým“ předáváním hodnot mezi procesory.

Bohužel naměřené hodnoty jsou, dle mého názoru výrazně zkreslené, jelikož někdy měří algoritmus výrazně rychleji, než v jinou dobu, přesto se zdá že časová složitost je částečně splněna.