Implementace algoritmu *Merge-splitting sort*

František Šumšal

April 8, 2018

1 Merge-splitting sort

Paralelní řadící algoritmus, který je řízen lineárním polem procesorů $p(n) < n$, kde každý procesor obsahuje několik čísel, jež jsou řazena. Předzpracování dat spočívá v rovnoměrném rozdělení čísel mezi procesory, kde každý procesor seřadí svou posloupnost sekvenčním řadícím algoritmem.

Poté se v $\lceil p/2 \rceil$ iteracích střídavě provádí následující fáze:

1. Každý sudý procesor získá posloupnost ze sousedního procesoru, kterou spojí v jednu a seřadí. Seřazenou sekvenci opět rozdělí a druhou polovinu odešle zpět sousednímu procesoru.
2. Totéž pro každý lichý procesor.

Po provedení předcházejících kroků a následném sesbírání posloupností ze všech procesorů získáme seřazenou posloupnost čísel.

1.1 Analýza

- Předzpracování: ($\mathcal{O}(n \log n)$ pro std::sort) $\Rightarrow \mathcal{O}((n/p) \log (n/p))$
- Přenos z procesoru P_{i+1} do P_i: $\mathcal{O}(n/p)$
- Spojení dat z P_i a P_{i+1}: $2 \cdot n/p$
- Přenos druhé poloviny dat zpět do P_{i+1}: $\mathcal{O}(n/p)$
- Fáze 1 nebo 2: $\mathcal{O}(n/p)$

$t(n) = \mathcal{O}((n/p) \log (n/p)) + \mathcal{O}(n) = \mathcal{O}((n \log n)/p) + \mathcal{O}(n)$
$c(n) = t(n) \cdot p = \mathcal{O}(n \log n) + \mathcal{O}(n \cdot p)$

2 Implementace

Algoritmus je implementován v jazyce C++ s využitím knihovny OpenMPI.

Načítání, rozesílání a následné sesbírání dat z procesorů spravuje *"kořenový"* (root) procesor, v tomto případě s ID 0. Data jsou načtena z předdefinovaného souboru *numbers* a uložena do vektoru. Tento vektor je poté *"vycpán"* předdefinovaným číslem, které se mezi řazenými čísly nemůže vyskytnout – vzhledem k zadání projektu jsou očekávána pouze čísla 0-255, proto bylo vhodným kandidátem číslo -1. Tímto *vycpáním* je dosaženo toho, že počet řazených čísel je dělitelný počtem procesorů a není nutné ošetřovat případné odchylky. Pro tento projekt je to řešení dostačující, ale existuje několik dalších vhodnějších řešení (například mapování počtu prvků pro každý procesor). Navíc aktuální řešení může mít vliv na výsledky experimentálního měření.

Samotné rozeslání dat ostatním procesorům je rozděleno do dvou kroků – nejprve se rozdistribuuje velikost posloupnosti pro každý procesor pomocí metody `MPI::Comm::Bcast`, která se získá vydělením počtu řazených čísel počtem procesorů. Dalším krokem je distribuce posloupností samotných – to obstarává metoda `MPI::Comm::Scatter`, která každému procesoru pošle předem dohodnutých n čísel.

Před samotným řazením si každý procesor seřadí svou posloupnost pomocí funkce `std::sort` a vyčká na bariéře na zbytek procesorů (bariéru obstarává metoda `MPI::Comm::Barrier`).

Následně se vykoná samotné jádro algoritmu – v $\lceil p/2 \rceil$ krocích se provádí dvě fáze:

1. Každý sudý procesor p získá posloupnost od svého souseda $p+1$ pomocí metody `MPI::Comm::Recv` (soused tuto posloupnost odešle metodou `MPI::Comm::Send`). Procesor p spojí tyto posloupnosti v jednu (merge), seřadí ji (opět pomocí funkce `std::sort`), uloží si její první polovinu a druhou odešle zpět procesoru $p+1$ s využitím výše zmíněných metod.

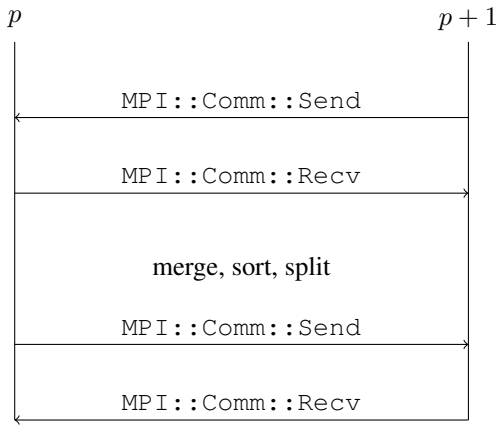
2. Totéž pro každý lichý procesor.

Mezi každou fází je opět bariéra, aby nedošlo k přeposílání chybných údajů.

Po provedení algoritmu jsou data opět pomocí kořenového procesoru sesbírána do výsledného vektoru metodou `MPI::Comm::Gather`, ze kterého je odstraněno číslo použito k *vycpání* v předzpracování. Výsledný vektor je poté vypsán na standardní výstup.

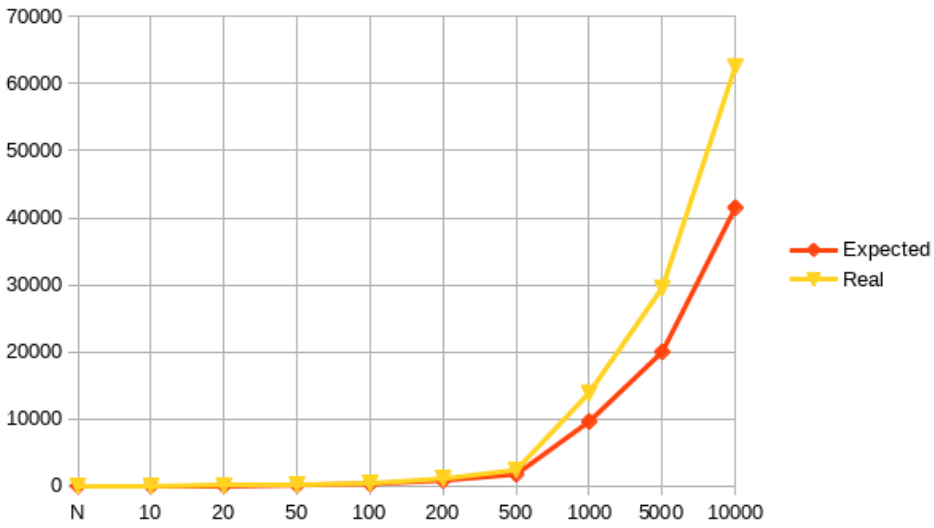
3 Komunikační protokol

Komunikační protokol je velmi jednoduchý, jak je popsáno v předchozí sekci. Vynecháme-li počáteční distribuci hodnot pomocí metod `Bcast` a `Scatter`, a závěrečeného sesbírání dat metodou `Gather`, zůstává nám pouze komunikace jádra algoritmu, která probíhá vždy mezi dvěma sousedními procesy.



4 Experimenty

Pro výpočet časové složitosti bylo do programu vloženo několik instrukcí, které průběžně počítaly složitost dle vztahů ze sekce 1. Počet procesorů byl pro všechny testy nastaven na 4.



5 Závěr

Výsledkem projektu je funkční implementace algoritmu Merge-splitting sort. Dle provedených experimentů se časová složitost odchyluje více při větším počtu řazených čísel, což může být následkem řešení nerovnoměrného rozložení čísel pomocí *vycpávání*, případně chybou při samotném zjišťování časové složitosti.