Vysoké učení technické v Brně
Fakulta informačních technologií

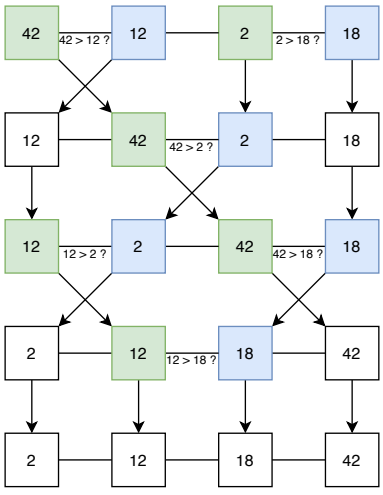
Paralelní a distribuované algoritmy
2. Projekt – Odd-Even Transposition Sort

Dominik Harmim (xharmi00)
xharmi00@stud.fit.vutbr.cz
27. března 2020

1 Rozbor algoritmu

Algoritmus *Odd-Even Transposition Sort* je *paralelní řadící algoritmus*, který předpokládá *lineární pole procesů*. Dále se budeme bavit o verzi algoritmu, který řadí prvky *vzestupně*.

Pro seřazení n hodnot je potřeba n procesů. Předpokládá se, že sousední procesy spolu mohou komunikovat. Na počátku každý proces p_i obsahuje jednu z řazených hodnot x_i. V prvním kroku algoritmu každý *lichý proces* p_i porovná svoji hodnotu x_i s hodnotou x_{i-1} svého levého souseda p_{i-1} a pokud je jeho hodnota menší než hodnota souseda ($x_i < x_{i-1}$), procesy své hodnoty vymění. V druhém kroku provedou analogickou operaci všechny *sudé procesy*. Řazené hodnoty budou seřazeny nanejvýše po n takovýchto krocích, kde se střídají sudé a liché procesy. Obrázek 1 demonstruje fungování tohoto algoritmu.



Obrázek 1: Demonstrace řazení prvků algoritmem

1.1 Analýza algoritmu

Jak již bylo řečeno výše, počet procesů pro seřazení n prvků je n, tedy $p(n) = n$.

Každý proces si vždy pamatuje hodnotu právě jednoho prvku. *Prostorová složitost* je proto n, tedy $s(n) = n$.

V každém kroku algoritmu se provádí paralelně na každém druhém procesu jedno porovnání a dva přenosy hodnot. *Časová složitost* je tedy konstantní, $t(n) = \mathcal{O}(n)$.

Celková cena tohoto algoritmu je následující: $c(n) = t(n) \cdot p(n) = \mathcal{O}(n) \cdot n = \mathcal{O}(n^2)$. Tato cena *není optimální*, protože $\mathcal{O}(n^2) \neq \mathcal{O}(n \cdot \log n)$, kde $\mathcal{O}(n \cdot \log n)$ je cena optimálního sekvenčního řadícího algoritmu.

Zrychlení tohoto algoritmu oproti optimálnímu sekvenčnímu řeší, který má časovou složitost $t(n) = \mathcal{O}(n \cdot \log n)$, je následující: $\frac{\mathcal{O}(n \cdot \log n)}{\mathcal{O}(n)} = \mathcal{O}(\log n)$.

2 Implementace

Algoritmus je implementován v programovacím jazyce C++. Je využito knihovny *Open MPI* pro zasílání zpráv mezi procesy. Implementace se nachází v souboru `ots.cpp`. Po spuštění programu se hned po inicializaci knihovny Open MPI zjistí číslo procesu, které mu knihovna přidělila. Podle tohoto čísla se potom rozhoduje, jak se bude daný proces chovat a se kterými procesy bude komunikovat (rozdělení na sudé/liché procesy). Procesy se číslují přirozenými čísly včetně čísla 0. Proces s číslem 0, tzv. hlavní proces, je proces, který provádí vstupní a výstupní operace a který iniciuje komunikace s ostatními procesy a celou komunikaci řídí. Lze říci, že program je rozdělen do 3 částí, které budou dále blíže popsány.

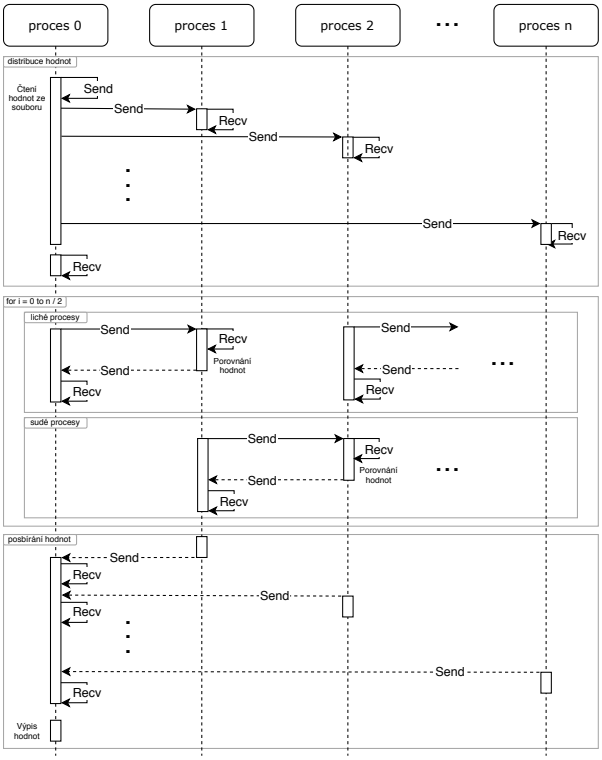
V 1. části programu provádí hlavní proces funkci `read_numbers`, která čte řazené prvky (konkrétně přirozená čísla včetně čísla 0 v rozsahu 0-255) ze vstupního souboru `numbers` a tyto vypisuje na standardní výstup a následně je postupně posílá všem procesům včetně sebe samotného (počet procesů musí být roven počtu čísel ve vstupním souboru, což je zajištěno při spouštění programu připraveným testovacím skriptem `test.sh`). Zasíláním a přijímáním prvků od procesů se bude dále myslet dvojice knihovních funkcí `MPI_Send`/`MPI_Recv`. Všechny ostatní procesy volají funkci `receive_number` (i hlavní proces po zpracování vstupního souboru), kde přijímají ča-

zený prvek od hlavního procesu, který se tímto stane jejich prvkem.

Ve 2. části programu probíhá samotné řazení prvků. Toto je implementováno funkcí `ots`. Zde se m-krát, kde $m = \frac{\text{počet procesů}}{2}$, provádí paralelní porovnávání nejdříve lichých procesů a následně paralelní porovnávání sudých procesů. Porovnání probíhá tím způsobem, že proces zašle sousednímu procesu prvek a čeká na přijetí nového prvku. Sousední proces nový prvek spočítá tím způsobem, že přijatý prvek porovná se svým prvek a v případě potřeby je prohodí a prvek zašle zpátky sousedovi.

Ve 3. části programu všechny procesy kromě hlavního procesu posílají své prvky hlavnímu procesu. Hlavní proces všechny prvky přijímá a ukládá si je do pomocného pole. Toto se děje ve funkci `receive_all_numbers`. Po přijetí všech prvků je postupně tiskne na standardní výstup funkcí `print_numbers`.

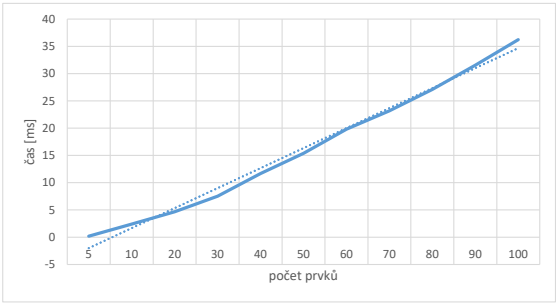
Na obrázku 2 je sekvenční diagram, který znázorňuje výše popsanou komunikaci mezi procesy.



Obrázek 2: Komunikační protokol procesů

3 Experimenty

Pro ověření teoretické časové složitosti byly provedeny experimenty, kde byl měřen reálný čas provádění řazení prvků funkcí `std::chrono::high_resolution_clock::now`. Měření bylo prováděno na stroji, kde je možné spustit nanejvýše přibližně 100 procesů, proto bylo měření provedeno nejvýše se 100 prvky. Měření bylo provedeno pro různý počet prvků od 5 až do 100 prvků. Každé měření pro určitý počet prvků bylo prováděno několikrát a výsledky byly průměrovány, aby nebyly vidět případné odchylky. Výsledky tohoto měření jsou k vidění na obrázku 3. Je vidět, že reálná časová složitost roste přibližně lineárně, stejně jako teoretická.



Obrázek 3: Výsledky měření experimentů

4 Závěr

V rámci tohoto projekty byla úspěšně vytvořena implementace algoritmu Odd-Even Transposition Sort. Teoretický časová složitost tohoto algoritmu byla ověřena reálnými experimenty.