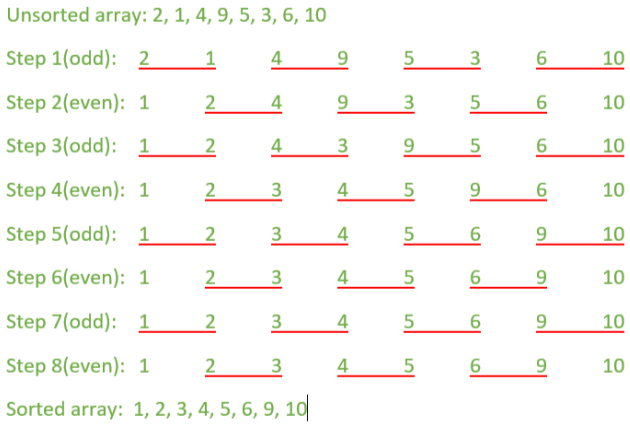
Paralelní a distribuované algoritmy - Projekt č.2

Buchal Petr, xbucha02

27. března 2020

1. Rozbor a analýza algoritmu

Odd-even transportation sort je řadící algoritmus podobný bubble sortu, který je ale na rozdíl od něj možné implementovat paralelně. Vstupem algoritmu je sekvence čísel, na výstupu algoritmu je tato sekvence seřazena. Vizuální ukázka algoritmu je na obrázku 1.



Obrázek 1: Vizuální příklad použití algoritmu Odd-even transportation sort. Převzato z shorturl.at/orPY8.

Při paralelní implementaci je počet procesů roven počtu seřazovaných hodnot. Na počátku obsahuje každý proces jednu vstupní hodnotu. V prvním kroku algoritmu se každý lichý proces spojí s následujícím sudým procesem. Pokud je hodnota lichého procesu větší než sudého procesu, tak si čísla vymění. Následně se spojí každý sudý proces se svým následujícím lichým procesem a proběhne stejná kontrola a případná záměna hodnot. Nejpozději po N krocích jsou hodnoty seřazeny. Pseudokód algoritmu je na obrázku 2.

```
for k = 1 to ⌈n/2⌉ do
  for i = 1, 3, ..., 2.(n/2)-1 do in parallel
    if y_i > y_i+1 then y_i ↔ y_i+1 endif
  endfor
  for i = 2, 4, ..., 2.((n-1)/2) do in parallel
    if y_i > y_i+1 then y_i ↔ y_i+1 endif
  endfor
endfor
```

Obrázek 2: Pseudokód algoritmu Odd-even transportation sortu. Převzato z shorturl.at/rtzW8.

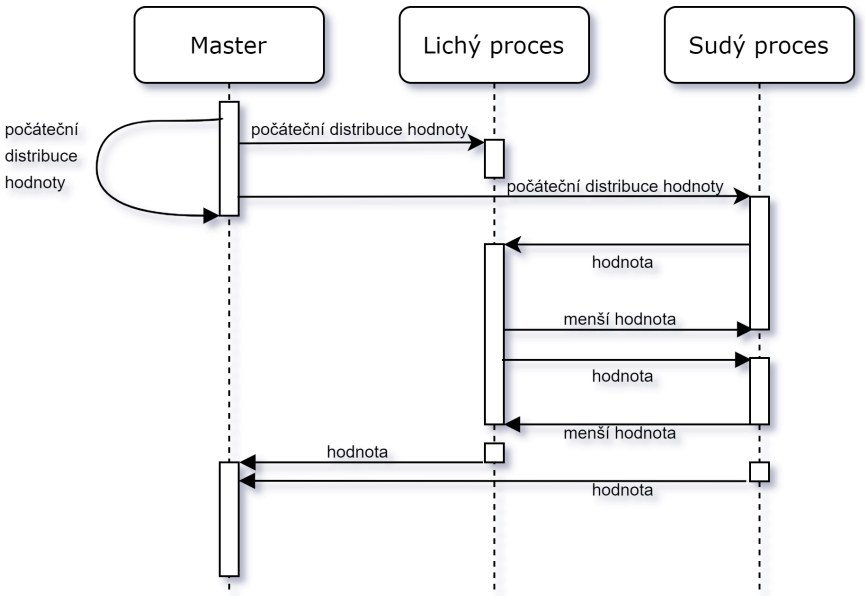
2. Implementace

Algoritmus jsem implementoval v jazyce c++ s využitím knihovny Open MPI. Zdrojový kód se nachází ve dvou souborech. V souboru *ots.cpp* se nachází implementace algoritmu. Soubor *test* slouží ke spouštění algoritmu.

Program v souboru *ots.cpp* nejprve načte čísla ze souboru *numbers*. Následně program pomocí funkce *MPI_Send* rozešle všem procesům konkrétní načtenou hodnotu. Ty je přijmou pomocí funkce *MPI_Recv*. Poté program přejde k vykonávání samotného algoritmu. Nejprve proběhne liché prohození hodnot a následně sudé prohození hodnot. Část prohození hodnot má na starost funkce *recieve_and_compare*, která přijímá hodnotu od procesu s nižším číslem, posílá mu zpět nižší hodnotu a vrací větší hodnotu procesu s větším číslem, ze kterého je volána.

2.1 Komunikační protokol

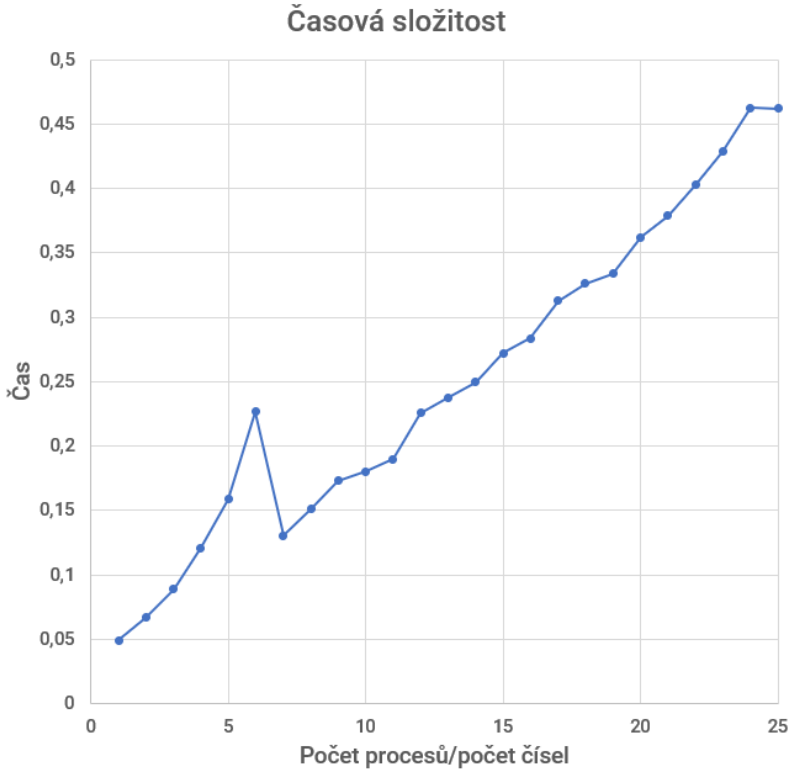
Vedoucí proces (tzv. master, viz obrázek 3) na začátku programu odešle všem procesům (včetně sebe) zprávu obsahující načtenou hodnotu k seřazení. Ostatní procesy čekají na příjem zprávy od nultého procesu (master). V okamžiku kdy ji obdrží, tak liché procesy začnou zasílat data procesům s *id*, které je o 1 větší od nich samotných. Tyto procesy zase čekají na zprávu od procesů s *id* o jedna menší. Obdobně probíhá komunikace mezi procesy během zasílání zpráv sudými procesy. Po skončení algoritmu všechny procesy odešlou svou hodnotu masterovi, který čeká na hodnotu od všech procesů. Během komunikace se mění jen obsah zprávy a políčko zdroj/cíl. Argument TAG zůstává po celou dobu nastaven na hodnotu 0. Zjednodušený sekvenční diagram použitého protokolu je na obrázku 3.



Obrázek 3: Zjednodušený sekvenční diagram komunikace procesů.

3. Experimenty

Pro počet hodnot od 1 do 25 bylo pro každou variantu provedeno 10 testů a jejich čas byl zprůměrován. Na obrázku 4 jsou vidět výsledky experimentu, které prokazují lineární časovou složitost algoritmu.



Obrázek 4: Graf zobrazující časovou složitost algoritmu vzhledem k počtu prvků.

4. Závěr

Na programování algoritmu Odd-even transportation sort jsem si vyzkoušel práci s knihovnou pro paralelní programování Open MPI. Na základě experimentů jsem potvrdil lineární složitost algoritmu.