1 Rozbor a analýza algoritmu

Algoritmus *Pipeline-Merge Sort* [1] je paralelný usporadúvací algoritmus pre lineárne pole procesorov. Vstup o veľkosti n celočíselných hodnôt postupne vstupuje do poľa procesorov o veľkosti $k = log_2 n + 1$. Procesory sú prepojené dvomi frontami. Prvý procesor P_1 odosiela hodnoty zo vstupnej fronty striedavo do front nasledujúceho procesora. Potom každý procesor $P_i, i \in \{2..k\}$, spojuje dve, už usporiadané postupnosti hodnôt dĺžky 2^{i-2} do jednej o dĺžke 2^{i-1}. Predtým než začnú procesory pracovať musí prvá fronta obsahovať 2^{i-2} hodnôt a druhá aspoň jednu hodnotu. Po skončení algoritmu sa usporiadaná postupnosť hodnôt nachádza na výstupe posledného procesora P_k.

1.1 Zložitosť algoritmu

Procesor P_1 začína v cykle 1, a každý ďalší procesor P_i, začína, až keď je splnená podmienka naplnenia vstupných front procesora, čo je $2^{i-2} + 1$ cyklov po tom, čo začal predchádzajúci procesor P_{i-1}. Procesor P_i teda začína v cykle

$$1 + \sum_{j=0}^{i-2} 2^j + 1 = 2^{i-1} + 1$$

a končí, keď spracuje zvyšných $n - 1$ hodnôt, teda v cykle

$$(n - 1) + 2^{i-1} + i - 1.$$

Keďže procesor P_k je procesor, ktorý spojuje posledné dve postupnosti hodnôt do finálnej usporiadanej postupnosti, celý algoritmus končí v cykle

$$n + 2^{k-1} + k = 2n + log_2 n - 1, k = log_2 n + 1,$$

z čoho vyplýva, že časová zložitosť algoritmu $t(n)$ je $O(n)$. Vieme, že priestorová zložitosť algoritmus $p(n)$, tj. počet potrebných procesorov je $log_2(n) + 1$, a teda celková cena algoritmu $c(n)$ je

$$c(n) = t(n) * p(n) = O(n) * (log_2(n) + 1) = O(nlog_2 n),$$

čo znamená, že algoritmus je optimálny.

2 Implementácia

Na implementáciu algoritmu bol použitý jazyk C++ spolu s open-source knižnicou *OpenMPI* a štandardnou knižnicou *std::queue* na implementáciu front medzi procesmi. Keďže zadanie uvažuje iba veľkosť vstupu o šestnástich hodnotách, program pracuje striktne s touto veľkosťou a s počtom procesov $log_2 16 + 1 = 5$. Procesy sú indexované od 0. Program má dve hlavné časti zjednodušene popísané algoritmami 1, pre proces 0, a 2 pre zvyšné procesy. Proces 0 načíta vstupné hodnoty, vypisuje ich na štandardný výstup a posiela nasledujúcemu procesu.

Algoritmus 1: Činnosť procesu 0

```
while je číslo na vstupe do
    načítaj a vypíš číslo;
    pošli číslo nasledujúcemu procesu;
end
```

Všetky ostatné procesy, tj. procesy 1..4, najprv čakajú na potrebné naplnenie front, kde si každý proces určí veľkosť vstupných postupností podľa svojho *id* ako 2^{id-1}. Ďalej každý proces využíva vlastné riadiace premenné,

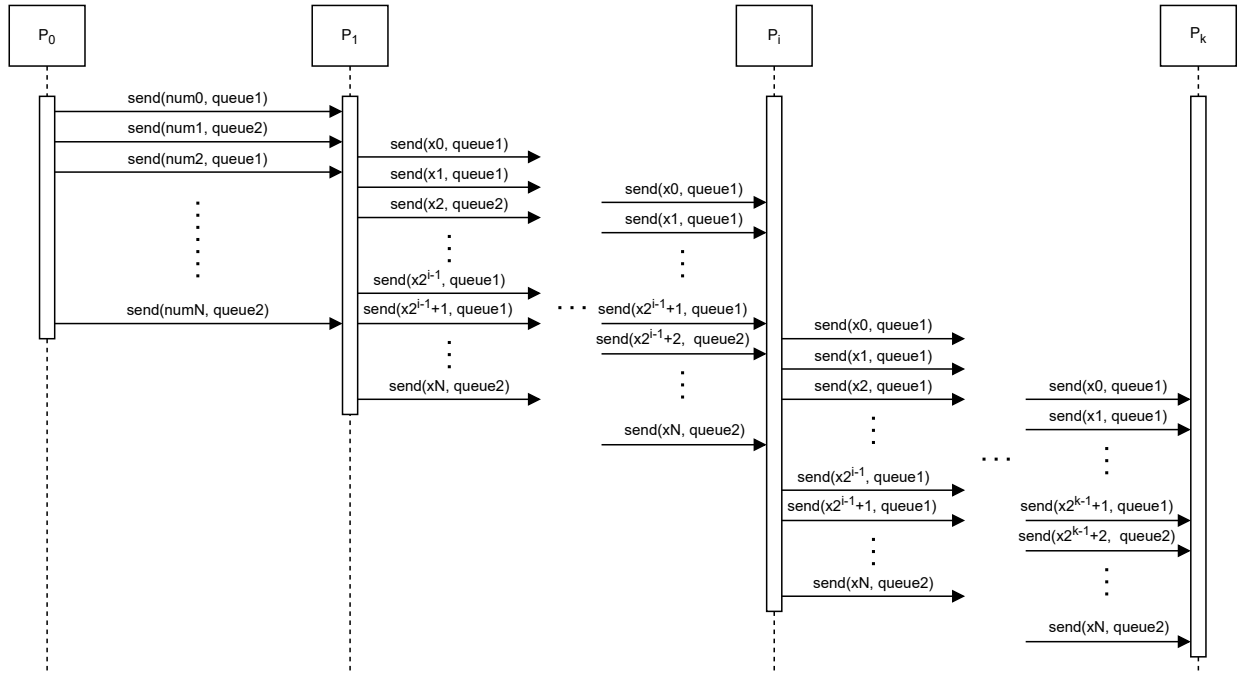
- `received` – počet prijatých hodnôt,
- `received1` – počet prijatých hodnôt prvou frontou,
- `received2` – počet prijatých hodnôt druhou frontou,
- `size1` – počet hodnôt na odoslanie z prvej fronty,
- `size2` – počet hodnôt na odoslanie z druhej fronty,

na zaistenie správneho fungovania programu, tj. na určenie, či už boli procesom prijaté všetky hodnoty, ak nie, tak do ktorej fronty sa prijíma a koľko hodnôt z konkrétnych vstupných postupností ešte ostáva na spojenie výstupnej postupnosti. Všetky procesy, okrem posledného procesu 4, ktorý vypisuje hodnoty výslednej usporiadanej postupnosti na štandardný výstup, odosielajú hodnoty nasledujúcemu procesu a to tak, že vždy je odoslaná menšia hodnota z aktuálneho čela front.

Algoritmus 2: Činnosť procesov 1..4

počkaj na potrebné naplnenie front;
while *je číslo v niektorej fronte* **do**
  porovnaj čísla na vrchole front;
  if *je číslo z fronty 1 menšie* **then**
    pošli číslo z fronty 1 nasledujúcemu procesu;
  else
    pošli číslo z fronty 2 nasledujúcemu procesu;
  end
  načítaj nové číslo do správnej fronty;
end

2.1 Komunikačný protokol procesov



Obr. 2.1: Komunikačný protokol procesov

Literatúra

[1] Akl, S. G. *Parallel Sorting Algorithms*. 1. vyd. Academic Press, Inc., 1985. 48–51 s. ISBN 0-12-047680-0.