

Pipeline Merge Sort

Paralelní a distribuované algoritmy

Autor: Jakub Sadílek

Login: xsadil07

Datum: 9.4.2021

1. Úvod

Cílem tohoto projektu je implementovat paralelní řadící algoritmus Pipeline Merge Sort v jazyce C/C++ pomocí knihovny Open MPI. Pro jednoduchost je velikost vstupního pole čísel podle zadání nastaveno na 16. Níže v této dokumentaci je podrobněji popsán princip a analýza algoritmu, implementace a na závěr spuštění programu.

2. Popis algoritmu

Algoritmus Pipeline Merge Sort je paralelní řadící algoritmus, který pracuje nad lineární topologií procesorů a funguje na principu spojování posloupností. Počet procesorů potřebných pro správnou funkci algoritmu je pevně daný rovnicí $p(n) = \log_2 n + 1$ a závisí na počtu řazených čísel n . První procesor (P_1) obsahuje jednu frontu, na které je uložena vstupní posloupnost čísel. Ostatní procesory (P_i) obsahují dvě fronty a spojují dvě seřazené posloupnosti (z každé fronty jednu) o délce 2^{i-2} do jedné výstupní posloupnosti o délce 2^{i-1} , kterou postupně posílají dalšímu procesoru. Poslední procesor bude mít na výstupu tedy kompletní seřazenou vstupní sekvenci.

Algoritmus začíná první procesor, který pouze posílá prvky z čela vstupní fronty druhému procesoru, který tyto prvky střídavě ukládá na svoji první nebo druhou frontu. Jakmile procesor P_i má na svoji první frontě posloupnost o délce 2^{i-2} a na druhé 1 prvek, začne tyto posloupnosti spojovat a posílat dále.

Spojování posloupností funguje tak, že procesor vezme z čela první nebo druhé fronty menší prvek a ten pošle na výstup. Zároveň je zde nutné, aby si daný procesor pohlídal, kolik prvků, z jaké fronty už vzal, protože musí spojovat právě dvě posloupnosti o délce 2^{i-2} a nemůže přebrat více. Tedy pokud z první fronty už poslal celou posloupnost dále a nachází se zde menší číslo, musí prvně dokončit sekvenci – poslat dále zbylé prvky z posloupnosti na druhé frontě, protože ono malé číslo v první frontě patří již do další výstupní sekvence.

3. Teoretická složitost algoritmu

V předchozí kapitole bylo řečeno, že procesor P_i začíná řadit prvky na výstup, jakmile má v první frontě 2^{i-2} prvků a v druhé 1. Lze tedy spočítat, že procesor P_i začne pracovat $2^{i-2} + 1$ cyklů potom, co procesor P_{i-1} začal pracovat. Procesor P_i začne pracovat od začátku algoritmu v cyklu podle následujícího vztahu:

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

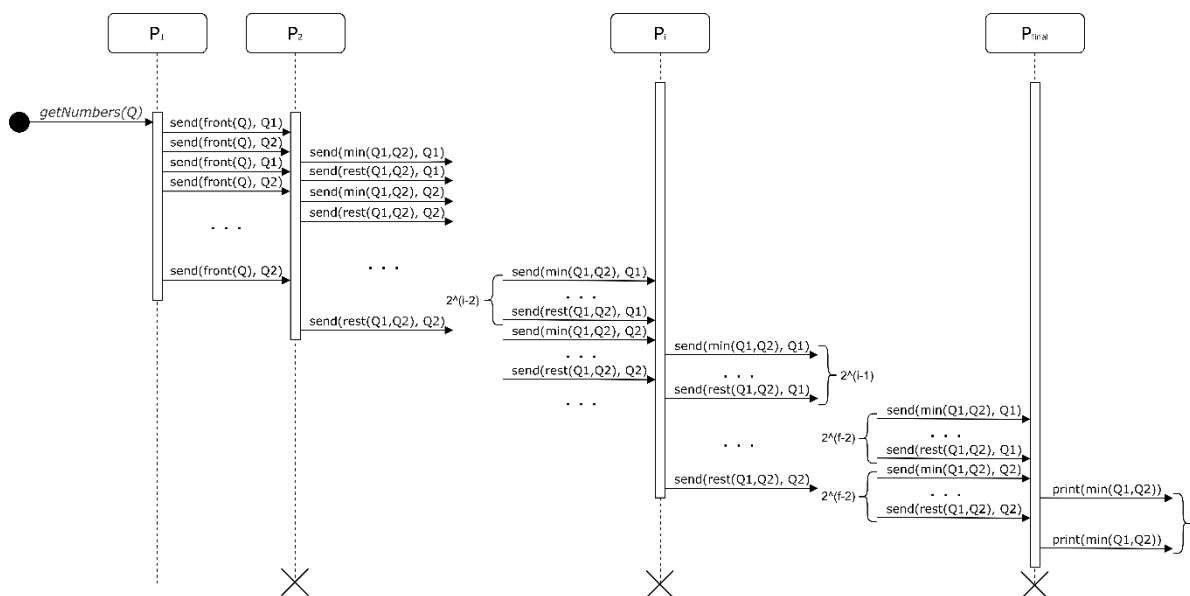
Procesor skončí potom, co seřadí zbylých $n - 1$ prvků, tedy v $(n - 1) + 2^{i-1} + i - 1$ cyklu. Celý algoritmus skončí potom, co skončí poslední procesor P_f , tedy v cyklu:

$$(n - 1) + 2^{f-1} + f - 1 = (n - 1) + 2^{(\log_2 n + 1 - 1)} + \log_2 n + 1 - 1 = 2n + \log_2 n - 1$$

Z rovnice $2n + \log_2 n - 1$ vidíme, že dominantní zastoupení hodnoty tvoří lineární část $2n$, tedy časová složitost je $t(n) = O(n)$. Počet procesorů je zadán $p(n) = \log_2 n + 1$, potom celkovou cenu spočítáme jako $c(n) = t(n) * p(n) = O(n) * (\log_2 n + 1) = O(n * \log_2 n)$.

4. Komunikační protokol

Níže na obrázku 1 je zobrazena sekvenční komunikace mezi prvním a druhým procesorem, obecným procesorem v lince a také se zde nachází poslední procesor, který na výstup produkuje kompletně seřazenou posloupnost.



Obrázek 1: Sekvenční diagram komunikace procesorů

5. Implementace

Program je podle zadání implementován v jazyce C/C++ za použití knihovny Open MPI. Na začátku programu je inicializováno MPI prostředí a první procesor binárně načte posloupnost čísel do své vstupní fronty. Dále jsou každému procesoru podle jeho ID spočítané limity podle rovnic z analýzy algoritmu (viz kapitola 2 a 3) s rozdílem, že číslování procesorů je zde prováděno od $i = 0$.

Veškeré počítání limit, které ovlivňují chování procesorů a jejich synchronizaci při posílání dat mezi sebou je provedeno ve funkci `procConfig()`. Zejména je zde spočítána délka vstupní sekvence nebo cyklus, kdy procesor začne přijímat data od předchozího procesoru, tj. kdy předchozí procesor začne data zpracovávat. Déle je spočítán cyklus kdy procesor začne sám data řadit a posílat dále či tisknout na standardní výstup jedná-li se o poslední procesor.

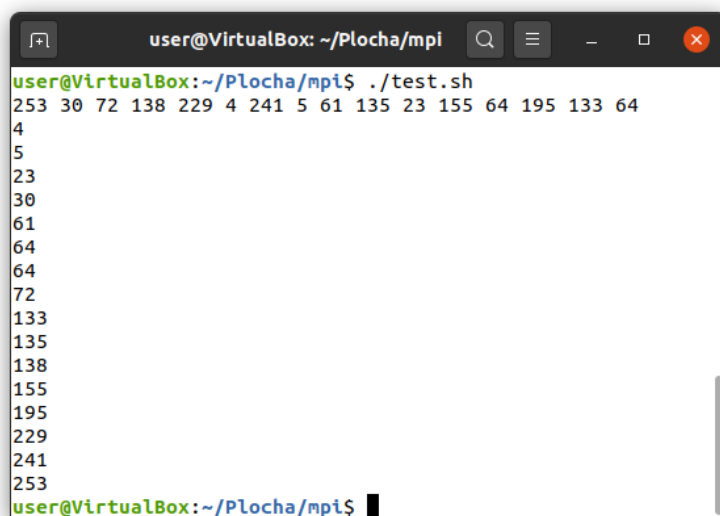
Poté co má každý procesor spočítané hodnoty, které definují jeho chování. Následuje smyčka s cykly algoritmu, kde cyklus, ve kterém algoritmus končí je spočítán podle rovnice v kapitole 3 (viz časová složitost). V této smyčce jsou úkoly procesorů rozlišeny podle toho, zda se jedná o první, poslední či jiný procesor v lineární topologii. První procesor pouze načítá data ze vstupní fronty a posílá je druhému procesoru pomocí funkce `MPI_Send()` dokud vstupní fronta není prázdná.

Ostatní procesory data přijímají od předchozího procesoru pouze pokud aktuální cyklus algoritmu se nachází mezi spočtenými hranicemi na začátku programu. Pokud v aktuálním cyklu má procesor hodnotu přijmu, je hodnota přijata od přechozího procesoru pomocí funkce `MPI_Recv()` a je

uložena pomocí funkce `storeNumber()` do odpovídající fronty. Pokud aktuální cyklus je mezi spočtenými hranicemi pro odeslání prvku dalšímu procesoru, může procesor začít řadit. Načtení správné hodnoty z front je implementováno ve funkci `loadNumber()` a hodnota je následně poslána pomocí `MPI_Send()`. V případě posledního procesoru je hodnota vypsána. Po posledním cyklu algoritmu je pak zavolána funkce `MPI_Finalize()`, která korektně ukončí prostředí MPI.

6. Spuštění

Program je možné snadno spustit pomocí přiloženého skriptu `test.sh`, který provede překlad programu, vygeneruje vstupní soubor s náhodnými čísly a spustí program (viz obrázek 2). Po doběhnutí programu smaže vygenerovaný binární soubor a také soubor s čísly.



```
user@VirtualBox: ~/Plocha/mpl
user@VirtualBox:~/Plocha/mpl$ ./test.sh
253 30 72 138 229 4 241 5 61 135 23 155 64 195 133 64
4
5
23
30
61
64
64
72
133
135
138
155
195
229
241
253
user@VirtualBox:~/Plocha/mpl$
```

Obrázek 2: Výpis programu

7. Závěr

V této dokumentaci jsme zmínili princip fungování paralelního algoritmu Pipeline Merge Sort a odvodili jeho teoretickou časovou složitost. Dále je znázorněna komunikace mezi procesory v tomto algoritmu pomocí sekvenčního diagramu a v kapitole o implementaci jsou zmíněny důležité úseky programu pro jeho snadnější pochopení. Testování rychlosti řazení aplikace zde není uvedeno, jelikož program pracuje pouze nad krátkou sekvencí čísel, tudíž výsledky by byly velmi zkreslené režii open MPI nebo samotnými procesy a velmi těžko bychom došli k rozumnému závěru. Přesto program funguje dle očekávání.