Paralelní a distribuované algoritmy (PRL)

2020/2021

František Horázný – xhoraz02

Pipeline merge sort

Úvod k algoritmu

Pipeline merge sort je paralelní řadící algoritmus, který vychází z algoritmu paralelního merge sortu, který vychází z merge sortu, který řadí dvě seřazené posloupnosti. Paralelní merge sort rozděluje posloupnost na *n* jednoprvkových (tedy seřazených) posloupností, paralelně spojí všechny dvojice těchto posloupností vedle sebe a v dalším kroku spojí tyto vzniklé dvouprvkové posloupnosti stejným způsobem a tak dále. Tímto způsobem lze seřadit posloupnosti o délce 2^i, i ∈ N. Při pipeline merge sortu se využívá možnosti překrývání jednotlivých řazení, tedy pokud algoritmus řadí 2 n prvkové posloupnosti, stačí mu alespoň první prvky z každé posloupnosti. Při hlubší analýze bychom zjistili, že optimální je začít ve chvíli, kdy jedna posloupnost je k dispozici celá a z druhé posloupnosti alespoň nejmenší, respektive největší prvek.

Teoretická realizace algoritmu

Pro tento algoritmus je využito $\log_2(n)+1$ procesorů, které jsou propojeny za sebou. Vstup celého systému je jedna fronta do prvního procesoru a výstup je jedna fronta z posledního procesoru. Výstup všech ostatních procesorů jsou 2 fronty, které jsou zároveň vstup dalšího procesoru.

První procesor čte po jedné hodnotě ze vstupu a zasílá hodnoty střídavě na výstupní fronty.

Prostřední procesory (P_i, kde i ∈ N) čekají, až se alespoň jedna jejich vstupní fronta naplní 2^{i-1} prvky a v druhé se objeví 1 prvek. Pokud je podmínka splněna začnou Postupně brát vždy menší, respektive větší prvek z vrcholů vstupních front, dokud nepřijmou stejný počet hodnot z obou front. Těchto hodnot vždy pošle na první frontu 2^i a poté přepne výstup na druhou výstupní frontu (stále dokola).

Poslední procesor funguje stejně, pouze nepřepíná mezi výstupy.

Analýza složitosti algoritmu

Procesor P_0 začíná v 1. cyklu a končí po n krocích. Ovšem procesory za ním začínají až se zpožděním. Procesor P_i začíná $2^{i-1}+1$ cyklů po P_{i-1}. Z toho plyne, že procesor P_i začíná v cyklu:

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

Po zpracování vstupu o délce n tedy skončí v cyklu

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

Pro výpočet doby celého algoritmu je třeba zjistit kdy skončí poslední procesor P_r, kde označení r je počet procesorů -1, tedy r=$\log_2(n)$ a tedy P_r skončí v cyklu:

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

Implementace

Pro implementaci jsem zvolil jazyk C. Kromě na pochopení triviálního test.sh, který se příliš neliší od vzorového příkladu na wiki stránkách je stěžejní pms.c. Byla využita knihovna mpi.h a využití paralelních procesorů.

Popis procesorů:

0. procesor otevře soubor a načítá čísla po jednom a rovnou rozesílá ostatním procesorům pomocí MPI_Send(). Následně čeká, až se mu vrátí seřazená posloupnost od posledního procesoru.

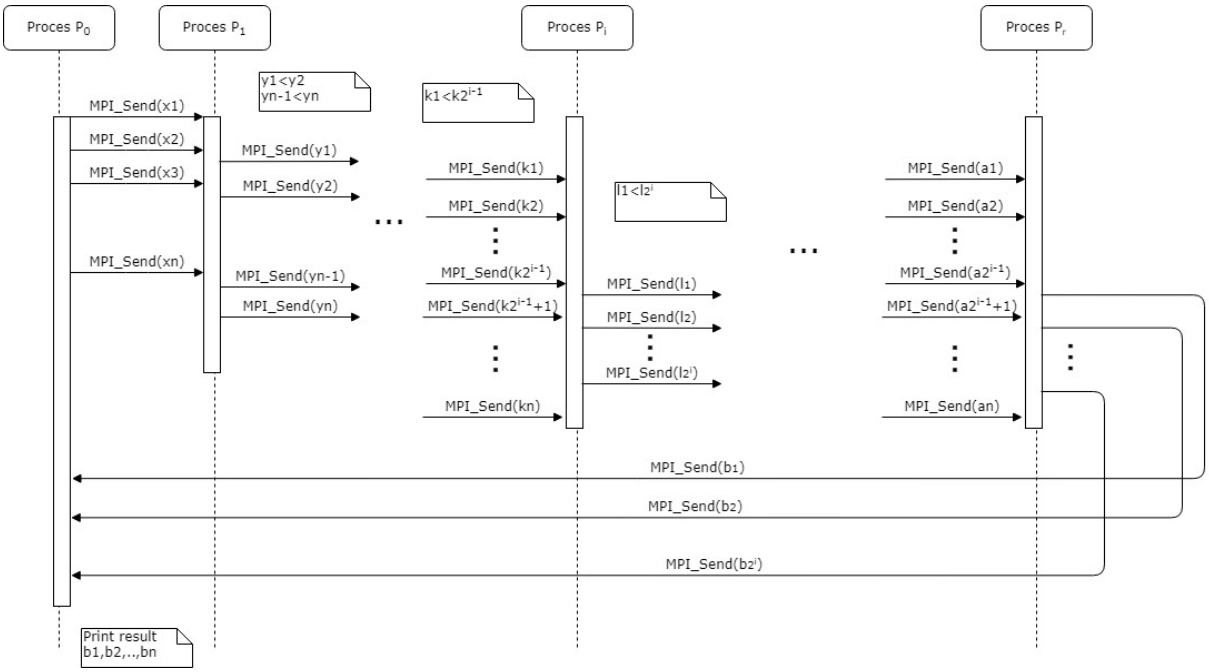
1. procesor je velmi podobný všem dalším, ale z důvodu jednodušší implementace vstupní fronty má mírně odlišnou implementaci. Obsahuje dvourozměrné pole o velikosti 2x2 se kterým se pracuje jako s cyklickou frontou. Pro tuto potřebuje jsou k oběma frontám zapotřebí označení začátku a konce fronty (tops a bottom). Následuje cyklus, ve kterém se po skupinách načtou data buď do jedné nebo do druhé fronty. Při dostatečném množství prvků se aktivuje výpočet proměnnou start, který sestává pouze z podmínky, odeslání dat a upravení fronty.

Ostatní procesory mají všechny stejnou implementaci, která se od 1. procesoru liší pouze výpočtem dimenzí a odpočítáváním hodnot které patří k sobě v rámci posloupností. V tomto případě již nebinárně – jsou potřeba 2 countery, které zajišťují čtení ze správné fronty i když je na druhé frontě menší číslo.

Z důvodu startování procesorů po určité době je nutné odeslat zbylé hodnoty z front, k tomu slouží druhý cyklus pro 2^{i-1} hodnot.

Komunikační protokol

Procesy si posílají pouze jednotlivé hodnoty lineárně za sebou a řadí si je samy podle toho, v jakém stavu se právě nachází. Nebylo tedy nutné ani používat tagy u zpráv. Všechny procesy dostávají zprávy pouze od předchozího procesu a posílají pouze dalšímu, a proto je komunikační protokol velmi jednoduchý. V diagramu je znázorněno, kdy proces začíná a jak seřazená posloupnost do něj vchází a jak seřazená z něj vychází.



Závěr

Největší problém bylo pochopit samotný algoritmus a zjistit, že se mají vždy dobrat posloupnosti patřící k sobě. Po pochopení tohoto principu už stačilo naimplementovat jednoduché cyklické fronty a neudělat chybu. Projekt byl testován na Merlinovi, kde fungoval i pro seřazené posloupnosti, opačně seřazené posloupnosti, náhodné posloupnosti a posloupnosti s obsahem duplicit.
S projektem jsem spokojen.

Brno, 9.4.2021