

Dokumentace k prvnímu projektu do PRL – Pipeline Merge Sort

Kateřina Fořtová (xforto00)

duben 2020

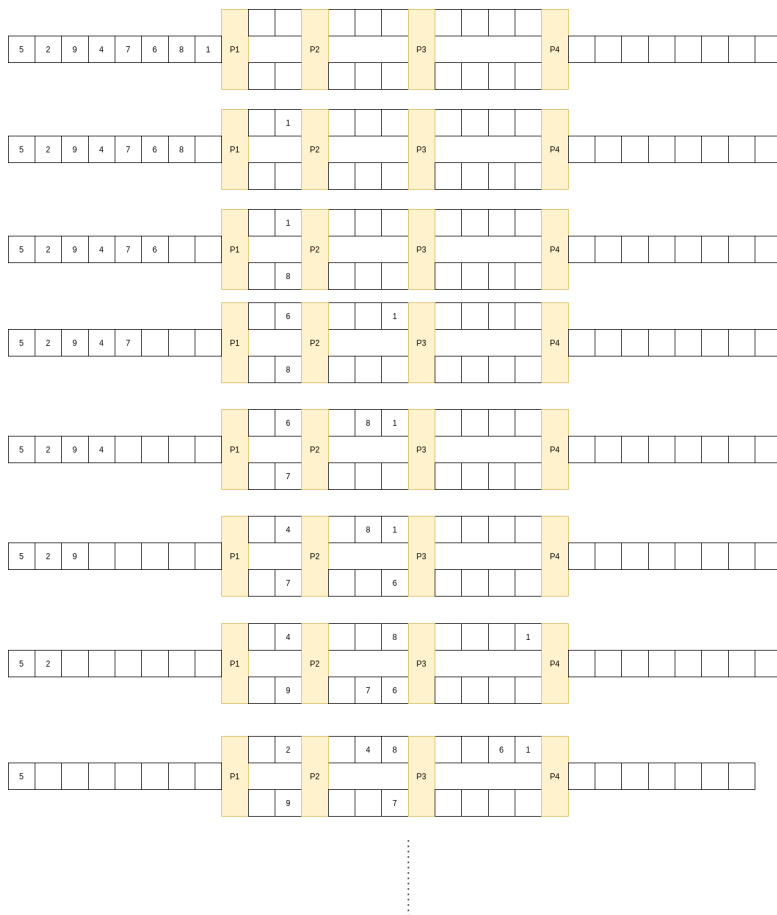
1 Rozbor a analýza algoritmu

Úkolem projektu byla implementace řadícího paralelního algoritmu Pipeline Merge Sort. Tento algoritmus pracuje s fixním počtem použitých procesorů, který se řídí vzorcem $p(n) = \log_2 n + 1$, kde n je délka vstupní posloupnosti. Využívá se tedy pevné lineární sítě procesorů pro optimální cenu.

První procesor má za úkol načítat vstupní posloupnost a zasílat je dalšímu procesoru, kdy střídavě probíhá přepínání výstupní linky a hodnoty jsou tedy střídavě uloženy do jedné z daných dvou front. Tento procesor neprovádí žádné porovnávání.

Prostřední procesory P_i začnou pracovat, když mají na jednom vstupu posloupnost délky 2^{i-2} a na druhém vstupu posloupnost délky 1. Při tomto potřebném naplnění začne probíhat proces slučování, kdy se spojují dvě posloupnosti do jedné. Hodnoty na stejných pozicích v obou linkách jsou porovnávány a menší z nich je následně zasílána dalšímu z procesorů.

Poslední z procesorů má za úkol porovnávat vstupy od předposledního procesoru a řadit je do finální výstupní posloupnosti. Výstupní seřazená posloupnost se tiskne uživateli.



Obrázek 1: Příklad řazení posloupnosti pomocí Pipeline Merge Sort – prvních 7 kroků algoritmu

Jak již bylo uvedeno, procesor P_i začne pracovat, když má na jednom vstupu posloupnost délky 2^{i-2} a na druhém vstupu posloupnost délky 1. Tento procesor tedy začne svoji práci $2^{i-2} + 1$ cyklů po předchozím procesoru P_{i-1} . Procesor P_i začne v cyklu, který je dán následujícím vzorcem:

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

Procesor P_i poté následně skončí v cyklu:

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

Proměnná i značí index konkrétního procesoru a $p(n)$ počet procesorů. Právě počet procesorů byl získán na základě vzorce výše. Algoritmus skončí když skončí činnost poslední z procesorů. Procesor P_i skončí svoji činnost v cyklu $(n - 1) + 2^{i-1} + i - 1$, kde n je délka vstupní posloupnosti.

Celý algoritmus skončí za $2n + \log n - 1$ cyklů – z tohoto plyne, že časová složitost se dá vyjádřit následovně:

$$t(n) = O(n)$$

Prostorová složitost lze vyjádřit následovně:

$$s(n) = p(n)$$

Cenu algoritmu lze spočítat jako:

$$c(n) = t(n) \cdot (n) = O(n) \cdot (\log n + 1) = O(n \cdot \log n)$$

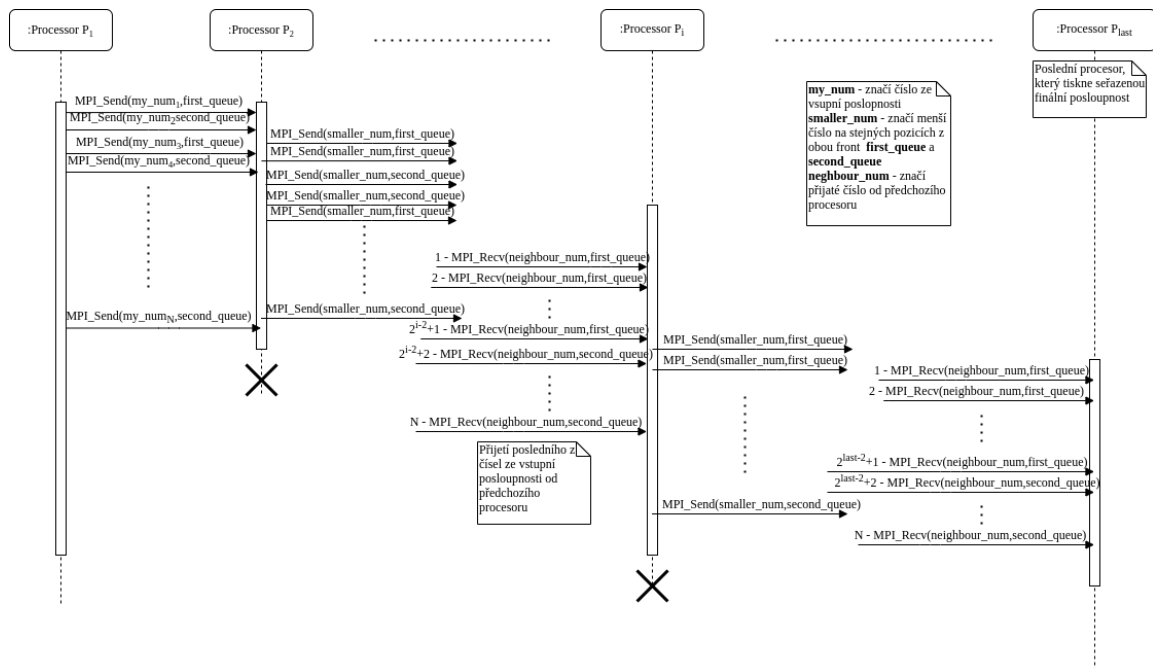
Časová složitost optimálního řadícího sekvenčního algoritmu lze vyjádřit jako $t_{seq}(n)$. Tyto hodnoty jsou řádově stejné, proto se jedná se o optimální cenu při porovnání s časovou složitostí optimálního sekvenčního řadícího algoritmu [1].

2 Implementace

Pipeline Merge Sort byl implementován v jazyce C++ s využitím knihovny OpenMPI, která implementuje funkce pro meziprocessorovou komunikaci. Dle zadání je vstupní počet čísel fixně nastaven na číslo 16, je proto nutné spustit algoritmus se správným počtem procesorů dle vzorce $p(n) = \log_2 n + 1 = \log_2 16 + 1 = 5$. Tento počet procesorů je zadán ve spouštěcím skriptu `test.sh`. Při načtení je uložena vstupní posloupnost do fronty a až poté je vytisknuta uživateli. Posloupnost je načtena do vstupní fronty a následně ji začne zpracovávat první procesor, který pouze vybírá první číslo z fronty vstupní posloupnosti a zasílá ji druhému procesoru. Bylo nutné správně rozdělit daný prostor pomocí indexů, které určovali, kdy může daný procesor začne pracovat, skončí svoji práci nebo kdy započne porovnávání. Také bylo nutné analyzovat, kdy je nutné přepínat dvě fronty, do kterých procesory ukládají vstupní čísla. Pro komunikaci mezi procesory byly využity pouze funkce `MPI_Send` a `MPI_Recv`. Poslední procesor již funkce `MPI_Send` nevyužívá, ale přímo tiskne na výstup postupně seřazená čísla. Bylo nutné zajistit, aby byla zachována vstupní pořadí čísel a aby se tedy zpracovávala popořadě. Funkce `MPI_Recv` slouží pro přijímání čísla od předchozího procesoru a následně je toto číslo zařazeno korektně do jedné ze dvou front.

3 Komunikační protokol

Komunikační protokol znázorňuje princip zasílání zpráv mezi procesory. Pro vizualizaci bylo využito sekvenčního diagramu. Zasílání zpráv je znázorněno pomocí využití funkcí knihovny OpenMPI `MPI_Send` a `MPI_Recv`. Procesor P_1 má za úkol přijímat vstupní neseřazenou posloupnost. Procesor P_{last} pak jakožto poslední procesor tiskne seřazenou posloupnost na výstup.



Obrázek 2: Obecný sekvenční diagram pro n procesorů

4 Závěr

Úkolem projektu byla implementace a dokumentace jednoho z paralelních řadících algoritmů Pipeline Merge Sort. Počet vstupních čísel byl pevně nastaven na 16 a na základě příslušného vzorce byl nastaven před spuštěním programu příslušný počet procesorů na 5. Při programování bylo potřeba vytvořit indexy pracovního prostoru pro jednotlivé pracující procesory. Při implementaci jsem vycházela z vlastností uvedených na přednáškách do předmětu PRL, avšak bylo potřeba upravit obecně indexování, protože na podkladech k přednášce, kde byl Pipeline Merge Sort vysvětlen, se indexuje od 1 a nikoliv od 0.

Z funkcí umožňující meziprocessorovou komunikaci bylo v programu použito úplné minimum – pouze funkce `MPI_Send` pro zaslání dat dalšímu procesoru a `MPI_Recv` pro přijetí dat od předchozího procesoru.

Celkově implementaci považuji za úspěšnou, nejdůležitější částí bylo vytvořit správné indexování pro prostor algoritmu a správné pochopení samotného principu Pipeline Merge Sort. Projekt byl řádně otestován jak na lokálním Ubuntu systému, využívající novější knihovny OpenMPI, tak i v referenčním prostředí `merlin.fit.vutbr.cz`.

Reference

- [1] HANÁČEK, P. *Paralelní a distribuované algoritmy – Řazení* [online]. Poslední změna 15. května 2020 [cit. 5. dubna 2021]. Dostupné na: <https://bit.ly/3t4oavq>.