Paralelní a distribuované algoritmy
Bucket sort

Bc. David Kolečkář, xkolec07
xkolec07@stud.fit.vutbr.cz

30. března 2019

1 Úvod

V této dokumentaci je popsán řadící algoritmus Bucket sort, který byl vytvořen v rámci projektu Paralelní a distribuované algoritmy. Dokumentace obsahuje analýzu algoritmu včetně odvození složitosti, popis implementace, rozbor experimentu a vizualizaci komunikačního protokolu.

2 Rozbor a analýza algoritmu

Bucket sort je paralelní algoritmus na seřazení množiny prvků. Samotný algoritmus pracuje nad binárním stromem. Strom obsahuje celkem $2 * \log(n) - 1$ procesorů. Kde každý listový procesor obsahuje n/m řazených prvků (m je počet listových procesorů, který lze vypočítat jako $\log(n)$). Na začátku první procesor rovnoměrně rozdělí vstupní množinu prvků mezi listové procesory. Listové procesory následně seřadí svou množinu prvků optimálním sekvenčním algoritmem. Následně na každé úrovni stromu, procesory spojí seřazenou posloupnost svých synů. Nakonec kořenový procesor uloží výslednou posloupnost do paměti.

Asymptotická časová složitost tohoto algoritmu je $t(n) = \mathcal{O}(n)$, kde n je počet řazených prvků. Časová složitost vychází ze 3 částí:

- Každý listový procesor čte $n/log(n)$ prvků $\rightarrow \mathcal{O}(n/log(n))$
- Použití optimálního řadícího algoritmu $\mathcal{O}(r * log(r)) = O(n/log(n)) * log(n/log(n)) = \mathcal{O}(n)$
- Každý procesor na úrovni stromu $i = log(m)–j$ kde j je iterace mezi úrovněmi stromu, spojí dvě posloupnosti o délce $n/2i \rightarrow \mathcal{O}(n)$

Počet procesorů je $2 * log(n) - 1 \rightarrow p(n) = \mathcal{O}(log(n))$. Celkově tedy cena algoritmu je $c(n) = t(n) * p(n) \rightarrow \mathcal{O}(n * log(n))$, což je optimální.

3 Implementace algoritmu

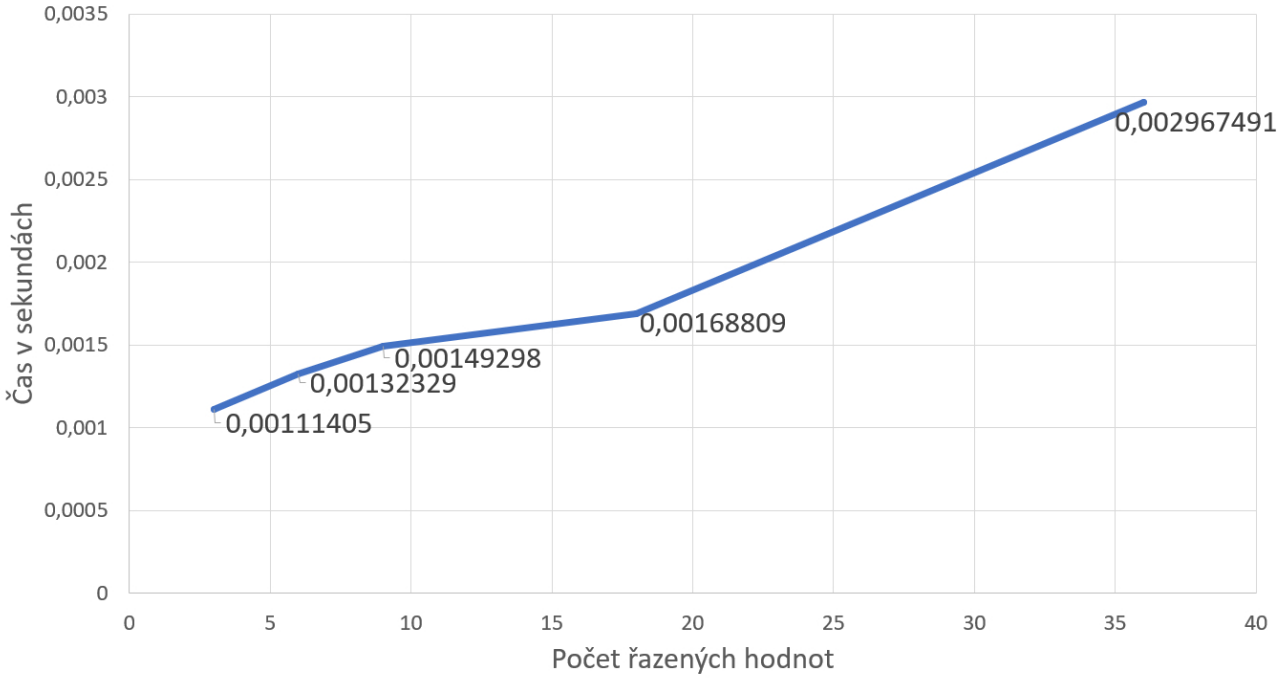
Pro spuštění aplikace byl vytvořen skript *test.sh*, který aplikací přeloží, připraví vstupní soubor *numbers* s náhodnými čísly a spustí aplikaci. Parametrem skriptu je počet hodnot, které se mají vygenerovat do vstupního souboru. Na základě počtu hodnot je vypočítán počet procesorů, se kterými je spuštěn samotný algoritmus. Při výpočtu se nejprve vypočítá počet listových procesorů $m = log(n)$, toto číslo je zaokrouhleno na nejbližší mocninu dvojky, kvůli úplnosti stromu. Následně je dopočítán počet procesorů jako $p = 2 * m - 1$. V tomto skriptu je také provedena kontrola parametrů. Pokud je vstupní parametr (počet hodnot) roven 0 vyhodí skript chybu, stejně tak při zadání většího počtu parametrů. Pokud není zadán žádný vstupní parametr, bere se výchozí hodnota 32.

Program byl napsán v jazyce C++ s využitím knihovny Open MPI. Vstupem programu je posloupnost náhodných čísel uložená v binárním souboru, kde jeden byte představuje číslo s rozsahem hodnot 0-255. Program tuto posloupnost načte a v téže pořadí ji vypíše v jednom řádku na standartní výstup. Jednotlivé části této posloupnosti o velikosti n/m (kde n je počet hodnot a m je počet listových procesorů) procesor s rankem 0 odešle listovým procesorům. V případě, že n/m není celé číslo, je doplněna posloupnost číslem -1 na takové n, kde počet prvků v listech i hloubka stromu je celé číslo. Následně listové procesory seřadí svou posloupnost čísel, pomocí funkce *std::sort()*, jejíž časová složitost je $\mathcal{O}(n * log(n))$. Poté každý listový procesor odešle svou seřazenou posloupnost svému rodiči. Pro nelistové procesory je zde cyklus, který pro každou úroveň stromu (počet úrovní stromu je roven $log(m)$) spojí posloupnosti svých synů. Ke spojení těchto posloupností je využita funkce *std::merge()*.

Výstupem programu, v případě správného chování programu, je na standartní výstup vzestupně vypsána seřazená posloupnost, kde každý prvek je oddělen novým řádkem. V případě že na začátku program doplnil posloupnost, jsou doplněná čísla odstraněna.

4 Experiment

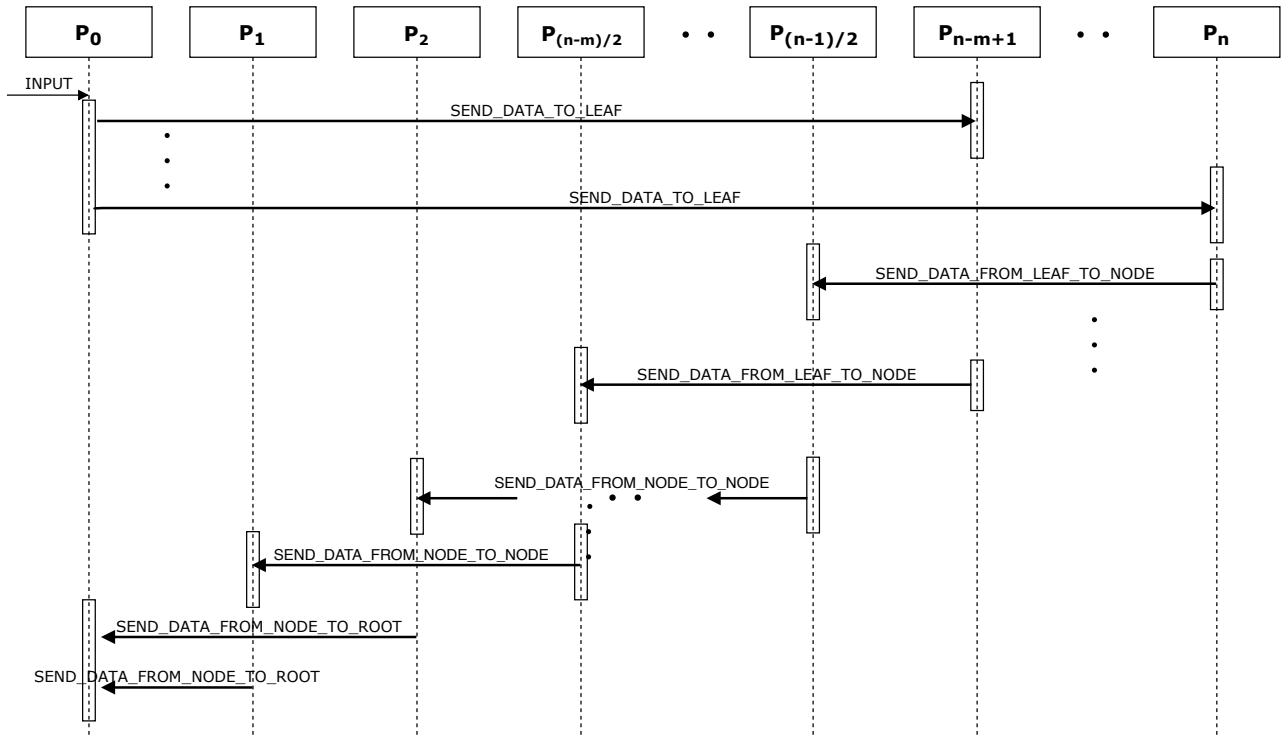
Algoritmus byl testován na různě velké vstupy (respektive pro n o velikosti 3, 6, 9, 18 a 36, kde n udává počet řazených hodnot) pro ověření časové složitosti. K tomuto účelu byl vytvořen jednoduchý skript, který pro každé n spustil 25 iterací algoritmu a změřené časy zapisoval do souboru. Následně byl udělán z těchto hodnot průměr, který lze vidět v grafu 1. Měření v algoritmu bylo spuštěno v momentě odesílání prvků k listovým procesorům a skončilo, jakmile byly prvky seřazeny. Načítání hodnot, inicializace knihovny openMPI a samotný výpis hodnot nejsou ve výsledném čase zahrnuty. Pro měření času byla využita funkce *MPI Wtime()*.



Obrázek 1: Výsledky měření zanesené do grafu

5 Komunikační protokol

Na obrázku 2 je znázorněna komunikace mezi procesory, jedná se o obecný model pro n procesorů (proměnná m, uvedená v diagramu, znamená počet listových procesorů). Komunikace simuluje funkce MPI Send() a MPI Recv().



Obrázek 2: Komunikační protokol

6 Závěr

Teoretická časová složitost algoritmu uvedená v kapitole 2, odpovídá grafu získaného pomocí experimentu v kaptiole 4. Algoritmus byl úspěšně otestován na školním serveru merlin (CentOS) a na privátním počítači s operačním systémem Ubuntu 18.