1. Úvod

Cílem projektu je pomocí knihovny Open MPI implementovat v jazyce C/C++ algoritmus Minimum Extraction Sort.

1.1 Vstup programu

Vstupem programu je posloupnost náhodných čísel uložená v binárním souboru. Tento soubor obsahuje posloupnost bajtů, kde jeden bajt představuje číslo s rozsahem hodnot 0-255. Délka této vstupní posloupnosti nemusí být nutně 2^n, kde n je přirozené číslo.

Program tuto posloupnost čísel načte a pomocí algoritmu Minimum Extraction Sort ji seřadí vzestupně podle hodnoty.

1.2 Výstup programu

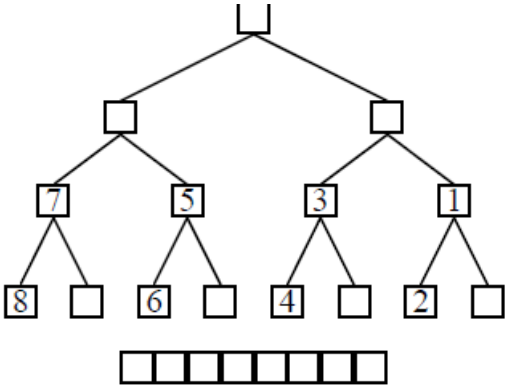
Výstup programu je buď na standardní výstup nebo na standardní chybový výstup. V případě korektního chování programu je na standardní výstup vypsána na prvním řádku posloupnost hodnot ze vstupního souboru a následně je na každém novém řádku vzestupně vypsán jeden prvek ze vstupní posloupnosti.

2. Analýza algoritmu

2.1 Popis algoritmu

Samotný algoritmus pracuje nad binárním stromem. Princip algoritmu spočívá v tom, že pokud budeme mít vstupní posloupnost o velikosti n, pak je třeba vytvořit binární strom s n listy, ($\log n$) + 1 úrovněmi a $2n - 1$ procesory.

Řazení pak probíhá tím způsobem, že každý listový procesor obsahuje jeden řazený prvek a každý nelistový procesor umí dva prvky porovnat. Díky takovému uspořádáni prvků ve stromu může každý nelistový procesor porovnat hodnoty svých dvou synů a menší z nich poslat svému otci. Po ($\log n$) + 1 krocích se minimální prvek dostane do kořenového procesoru. Výhodou je, že s každým dalším krokem se získá další nejmenší prvek.



Obrázek 2.1 Ukázka řazení pomocí binárního stromu

2.2 Analýza algoritmu

Jak bylo zmíněno v kapitole 2.1, výsledný strom má ($\log n$) + 1 úrovní, tedy první prvek se získá právě po ($\log n$) + 1 krocích. Kořenový procesor pak potřebuje jeden krok na porovnání a jeden na uložení výsledku do paměti. Dále každý ze zbylých $n - 1$ prvků spotřebuje 2 kroky.

Tedy výsledné složitosti jsou následující:

$$t(n) = 2*n + (\log n) - 1 = O(n)$$

$$p(n) = 2*n - 1$$

Pokud známe tyto dvě složitosti jsme dále schopni si odvodit celkovou cenu algoritmu.

$$c(n) = t(n)*p(n) = O(n^2)$$

Z čehož vyplývá, že algoritmus Minimum Extraction Sort není optimální.

3. Implementace

K implementaci algoritmu bylo použito jazyka C++ a knihovny Open MPI. Celá implementace algoritmu je rozdělena do dvou větších celků.

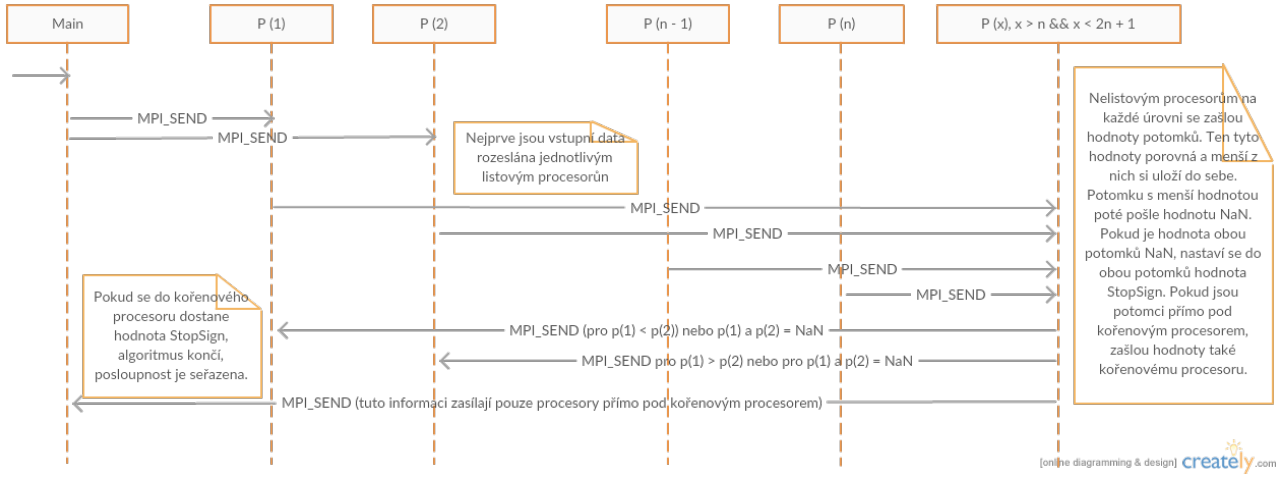
3.1 Příprava vstupní posloupnosti

V první části algoritmu bylo nejprve třeba připravit si vstupní posloupnost pro následné seřazení. Jednotlivé hodnoty jsou tedy načteny ze vstupního souboru a pomocí knihovny MPI rozeslány do jednotlivých listových procesorů. Vzhledem k tomu, že vstupní posloupnost nemusí být o velikosti mocnin čísla dvě, je v případě, že se tak stane do zbylých listových uzlů odeslána hodnota –1.

3.2 Implementace algoritmu

Samotné řazení probíhá tak, že nelistové procesory na nejnižší úrovni nejprve čekají dokud nebudou naplněny listové procesory daty a následně začínají porovnávat data v synovských uzlech.
Data jsou vždy na nejvyšší možné úrovni porovnána a každý otcovský procesor si do sebe uloží nižší hodnotu ze synovských uzlů, pokud ovšem tato hodnota není konstanta *NaN* nebo-li námi definovaná hodnota –1, která by se neměla objevit ve výstupní posloupnosti.

Z důvodu nepoužití cyklu for bylo nutné do algoritmu implementovat zarážku, která v případě, že obě hodnoty v synovských uzlech jsou *NaN*, ukládá do těchto uzlů druhou speciální hodnotu, a to *StopSign* a dále nastavuje příznak *stop* na *true*. V případě, že na takovýto uzel narazí kořenový procesor, algoritmus končí. Funkci samotného algoritmu, lze vyčíst z obrázku 3.2.1.



Obrázek 3.2.1 Sekvenční diagram Minimum Extraction Sort

4. Experimenty a testování

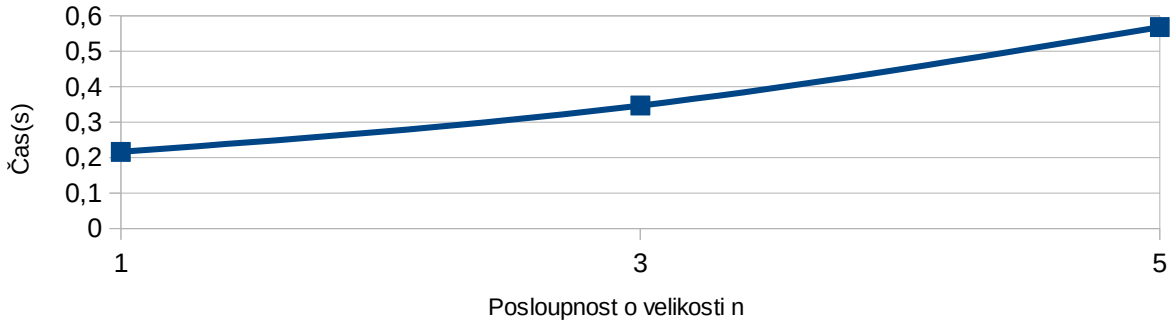
Veškeré experimenty byly prováděny na školním serveru *Merlin* s unixovým operačním systémem. Testy byly prováděny pro vstupní posloupnost o velikosti 1, 3, 7 hodnot. Více hodnot bohužel server *Merlin* neumožňuje z důvodu limitu na počet přidělených procesorů. Veškeré testy byly opakovány celkem desetkrát se stejnými parametry, přičemž pro zlepšení výsledků testování byly ve finále vyloučeny nejrychlejší

a nejpomalejší výsledky řazení. Čas byl měřen pomocí unixové utility *time*.

	Posloupnost $n = 1$	Posloupnost $n = 3$	Posloupnost $n = 5$
Pokus č. 1	0.25s	0.345s	0.509s
Pokus č. 2	0.195s	0.28s	0.476s
Pokus č. 3	0.204s	0.293s	0.491s
Pokus č. 4	0.247s	0.393s	0.763s
Pokus č. 5	0.212s	0.375s	0.659s
Pokus č. 6	0.198s	0.285s	0.466s
Pokus č. 7	0.209s	0.455s	0.577s
Pokus č. 8	0.216s	0.349s	0.606s
Průměr	0.216375s	0.346875s	0.568375

Tabulka 4.1 Naměřené hodnoty implementovaného algoritmu

Naměřené hodnoty jsou zanesené do grafu na obrázku 4.1, ze kterého lze vidět téměř lineární průběh měřeného algoritmu. Nicméně taktéž lze na grafu vidět, že se čas výpočtu algoritmu může zvyšovat exponenciálně, tuto anomálii by šlo pravděpodobně eliminovat testováním algoritmu na více procesorech a s delší vstupní posloupností, případně tato odchylka může být způsobena využitím utility *time*, která nezměří pouze průběh algoritmu, ale taktéž načítání vstupní posloupnosti dat a práci se standardním výstupem. V poslední řadě může být mírná odchylka způsobená chybou/neoptimalizací algoritmu.



Obrázek 4.1 Graf znázorňující vztah velikosti řazené posloupnosti a doby řazení.

5. Závěr

V projektu jsem implementoval algoritmus Minimum Extraction Sort a následně provedl ověření časové složitosti. Z výsledků získaných testováním a experimenty lze odvodit, že výsledná časová složitost implementovaného algoritmu odpovídá teoreticky spočtené.