1 Odd-even transposition sort

Tento algoritmus je v podstatě variace na známý *bubble-sort* algoritmus. Liší se v tom, že algoritmus *odd-even sort* pracuje v každé iteraci ve dvou fázích. V první fázi řadí liché elementy, a až poté řadí elementy sudé. Tedy teoretická časová i prostorová složitost je rovna složitosti algoritmu *bubble-sort*, a tedy:

- Časová složitost: $O(n)$, kde n je počet prvků pole.
- Celková složitost: $O(n^2)$, kde n je počet prvků pole.
- Prostorová složitost: $O(1)$ – konstantní, protože se pouze přehazují prvky v rámci pole, ve kterém bylo od počátku zapsáno.

Algoritmus je tedy založen na porovnávání sousedních prvků a pokud jsou v nesprávném pořadí, prvky jsou prohozeny. Neporovnávají se vždy dva sousední prvky jako u bubble-sortu, ale v každé fázi pouze liché se sousedními sudými, či sudé se sousedními lichými. Průchody jsou opakovány až do chvíle, kdy se žádný další prvek neprohodí, a pole je tedy správně seřazené.

Tento jednoduchý algoritmus byl původně navržen pro paralelní zpracování na procesorech s lokálním propojením, kde každý proces má přiřazenu jednu hodnotu (t.j. jeden prvek pole) a cyklicky kontroluje a (v případě špatného pořadí) prohazuje hodnotu se sousedovou hodnotou.

2 Implementace

Algoritmus je dle zadání implementován pomocí knihovny `OpenMPI`. Po počáteční inicializaci jsou ve funkci `masterReadAndDistribute()` (volané pouze master procesorem) čteny jednotlivé prvky pole (byty) ze vstupního souboru a ty jsou postupně distribuovány mezi jednotlivé procesory pomocí funkce `MPI_Send()`.

Každý procesor volá po spuštění (a master po distribuci čísel ze vstup. souboru) funkci `receiveNumberAndProcess()`, kde pomocí blokujícího volání `MPI_Recv()` čeká na jeden byte od master procesoru.

Po obdržení tohoto bytu (čísla 0-255) vstupuje každý procesor do cyklu, který je rozdělený na dvě části - t.j. fáze algoritmu.

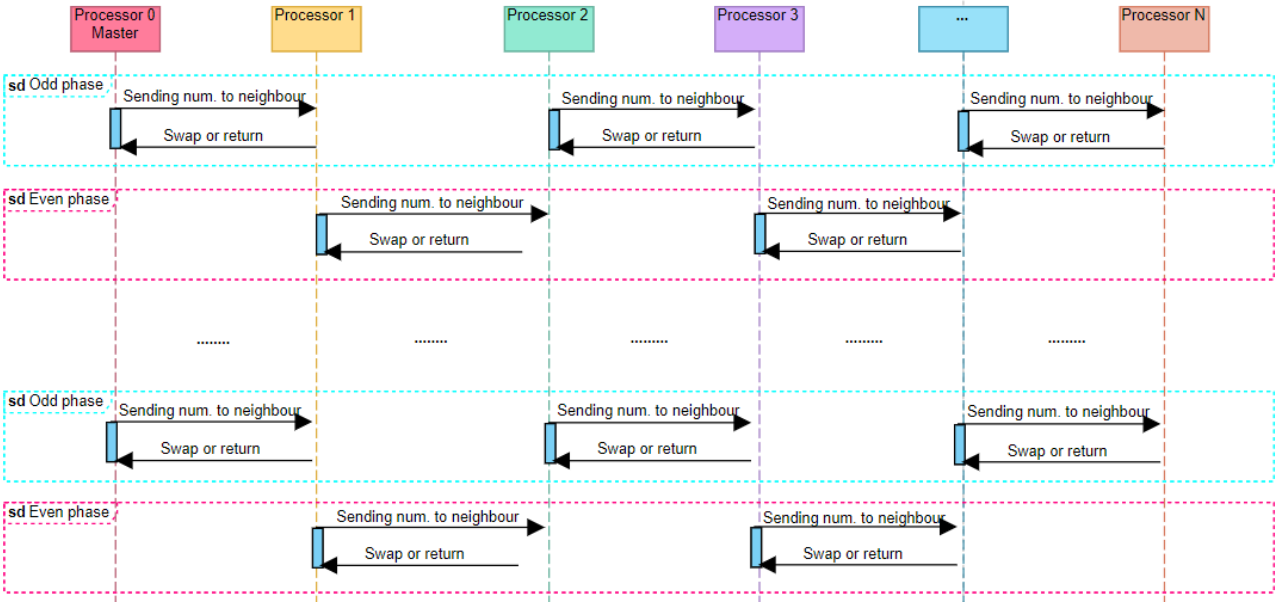
V první části (kroku) procesory se sudým identifikátorem (t.j. včetně master procesoru) odesílají své číslo procesoru vpravo, tedy s identifikátorem o jedna větším a tedy lichým. Poté vstupují do blokujícího čtení od souseda, jemuž zrovna odeslali své číslo. Jejich sousedi a tedy procesory s lichým identifikátorem okamžitě po vstupu do smyčky vstupují do blokujícího čtení od svého levého souseda, tedy procesoru s identifikátorem o jedna menším. Po obdržení čísla od souseda provedou porovnání se svým číslem. Pokud je sousedovo číslo větší, vrací sousedovi své číslo a sousedovo ukládají jako vlastní. V opačném případě sousedovi jeho číslo vrátí zpět, protože by porušilo pořadí.

Ve druhé části (kroku) se úlohy procesorů se sudými a lichými identifikátory prohodí.

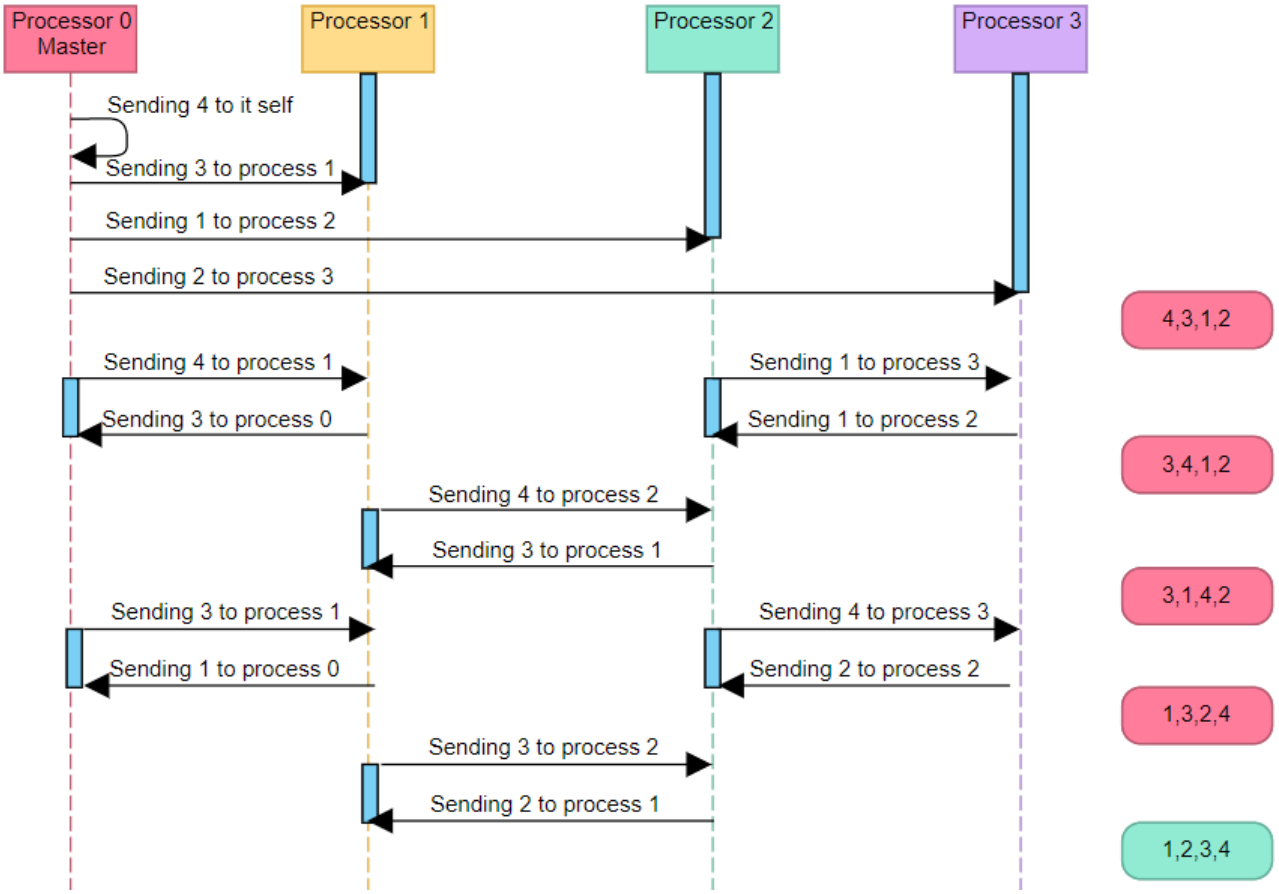
Toto je prováděno pro všechny procesory kromě nejpravějšího (s největším identifikátorem), protože ten by neměl svého pravého souseda, se kterým by se spojil. Nakonec, po (maximálně) n krocích je pole seřazeno.

Po tom co je celé pole seřazeno, procesory (kromě master procesoru) distribuují svá aktuálně přiřazená čísla zpět do master procesoru pomocí volání `MPI_Send()`. Master procesor tato čísla přijímá pomocí blokujícího volání `MPI_Recv()` a ukládá je do seřazovacího bufferu. Poté co přijme čísla od všech ostatních procesorů vypíše seřazené pole na standardní výstup.

Testovací skript ve formě shell skriptu přijímá právě jeden parametr (jinak končí chybou), a to je počet čísel (bytů) které má vygenerovat do cílového souboru `numbers`. Provádí také kompilaci a spuštění algoritmu odd-even pomocí `mpic++, mpirun` a tzv. *cleanup*, tedy smazání binárního souboru a souboru s vygenerovanými čísly.



Obrázek 1: Obecný sekvenční graf zasílání zpráv při provádění algoritmu `odd-even sort` pro N procesorů. Modré obdelníky znázorňují blokující čekání, ale nejsou použity všude kde by měly – pro přehlednost. Na obrázku lze vidět střídající se fáze pro liché a suché indexy pole (procesory).[1]



Obrázek 2: Příklad zasílání zpráv při použití algoritmu `odd-even sort` pro čtyři procesory se vstupními daty `4, 3, 1, 2`. Příklad končí až je pole prvků vzestupně seřazené `1, 2, 3, 4`.[2]

[1] Vytvořeno pomocí `https://online.visual-paradigm.com/diagrams`.
[2] Vytvořeno pomocí `https://online.visual-paradigm.com/diagrams`.

3 Měření

Vzhledem k tomu, že knihovna OpenMPI je navržena pro praktické použití a tedy neobsahuje žádné metody pro měření výkonnosti a zadáním nebylo měřit počet kroků, nýbrž rychlost, byla využita linuxová utilita jménem `time`. Všechny měřené časy jsou z referenčního stroje `merlin`. Měření bylo provedeno pro velikosti vstupu (a tomu odpovídajícímu počtu procesorů) 1 až 25, což je horní omezení na serveru merlin.

Obrázek 3: Měření výkonnosti implementace algoritmu `odd-even sort` na serveru merlin.[3]

V grafu se vyskytují tři křivky a význam jednotlivých křivek je následující:

- **`Real`** – Je celkový čas od počátku spuštění procesu až do jeho ukončení, a to včetně času kdy je proces blokovaný (např. čeká na dokončení vstupně-výstupní operace) a běh jiných procesů.
- **`User`** – Je množství času stráveného v uživatelské části kódu daného procesu. Je co čas strávený procesorem při zpracovávání kódu procesu a nepočítají se sem blokující operace a ostatní procesy.
- **`Sys`** – Je množství času stráveného v jádru systému (kernel) v rámci daného procesu (t.j. čas strávený systémovým voláním). Opět se sem nepočítá čas strávený blokováním a jinými procesy.

Jak lze z grafu vyčíst, reálná doba běhu řazení se nějak značně neliší pro 1 a pro 25 prvků (téměř lineárně rostoucí), zato doba strávená v uživatelské části kódu a systémových voláních se značně zvýšila, protože je tam započteno všech až 25 procesorů. Při větších velikostech vstupu se projevuje výkonnostní zisk daný větším počtem použitých procesorů. Doba pro větší počet prvků není stejná jako pro 1 prvek (ačkoli má každý prvek vlastní procesor), protože se v kódu vyskytuje blokující čekání `MPI_Recv()`, kdy procesor čeká na jiný procesor, než dokončí práce a pošlu/vrátí mu číslo.

4 Závěr

Z měření vyplývá, že ačkoli je algoritmus `odd-even sort` relativně jednoduchý, lze s ním dosáhnout velmi dobrých výsledků – téměř stejný reálný čas běhu pro 1 a pro 25 prvků, pouze malý, téměř lineární nárůst s drobnými výkyvy, které však mohly být způsobeny vyšší zátěží serveru.

[3] Vytvořeno pomocí `https://plot.ly`.