PRL - Odd-Even Merge Sort

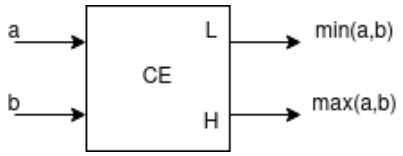
Natália Holková
xholko02@stud.fit.vutbr.cz
8. apríla 2022

1 Rozbor algoritmu

Odd-even merge sort je paralelný riadiaci algoritmus vytvorený K.E.Batcherom v 1968. Je založený na merge sort algoritme, ktorý spája dokopy dve zoradené polovice postupnosti. Avšak na rozdiel od klasického merge sort nie je tento algoritmus závislý na dátach - rovnaké porovnanie sa vykonáva bez ohľadu na skutočné dáta. Vďaka tomuto môže byť odd-even merge sort implementovaný ako *sorting network*.

Sorting network je typ porovnávačovej siete (comparator network), ktorá je určená na zoradenie pevne daného počtu hodnôt. Táto sieť sa skladá z množstva drôtov nesúcich hodnoty a porovnávacích modulov (comparison element), ktoré zamieňajú hodnoty na drôtoch ak nie sú v správnom poradí.

Algoritmus Odd-even merge sort sa teda riadi špeciálnou sieťou procesorov, ktoré označujeme ako `CE`. Jej schéma je zobrazená na obrázku 1. Každý procesor obsahuje dva vstupné a výstupné kanály. Procesor porovnáva hodnoty na vstupoch, menšiu dá na `L` výstup a väčšiu na `H` výstup. Väčšie siete vytvárame spájaním menších sietí.



Obr. 1: Schéma CE

2 Analýza algoritmu

Radíme postupnosť o dĺžke $n = 2^m$. V našom prípade $n = 8$, a preto $m = 3$. V 1. fáze potrebujeme 2^{m-1} CE, konkrétne 4 procesory. V 2. fáze potrebujeme $2^{m-2} * 3$ CE, teda 6 CE. V 3. a poslednej fáze potrebujeme $2^{m-3} * 9$ CE, čo predstavuje 9 procesorov. Celkovo je potrebných **19** procesorov.

Pre zoradenie n hodnôt je potrebných *p(n)* porovnaní, kde

$$p(n) = \mathcal{O}(n * \log(n)^2)$$

Časová zložitosť algoritmu $t(n)$ je pri n hodnotách na zoradenie:

$$t(n) = \mathcal{O}(m^2) = \mathcal{O}(\log^2 n)$$

Celková *cena* algoritmus $c(n)$ je závislá na počte porovnaní $p(n)$ a časovej zložitosti $t(n)$. Je určená podľa vzorca:

$$\begin{aligned} c(n) &= p(n) \times t(n) \\ &= \mathcal{O}(n * \log(n)^2) \times \mathcal{O}(\log^2 n) \\ &= \mathcal{O}(n * \log(n)^4) \end{aligned}$$

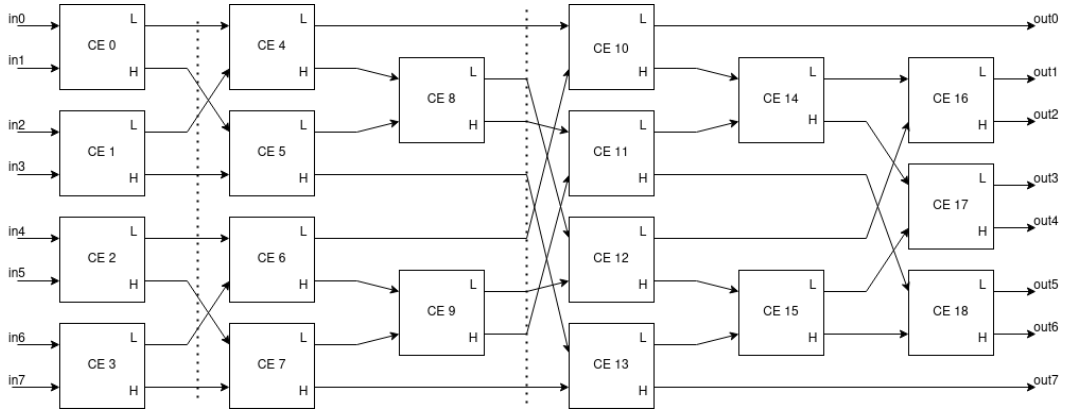
3 Implementácia

Algoritmus Odd-even merge sort implementujem v programovacom jazyku C++ s použitím knižnice *Open MPI* pre paralelizáciu. Implementácia je v súbore `oems.cpp`. Kompiláciu a spustenie rieši skript `test.sh`. Tento skript napevno vygeneruje 8 náhodných čísel do súboru `numbers` a vždý spúšťa preložený kód s 19 procesormi.

Po inicializácii knižnice pomocou `MPI_Init` je každému procesoru priradené číslo `rank` funkciou `MPI_Comm_rank`, ktoré slúži na identifikáciu. Procesory nadobúdajú rank od 0 po 18. Procesor s `rank` 0 predstavuje hlavný ROOT

procesor zodpovedný navyše za načítanie a výpis vstup a na konci výpis už zoradenej postupnosti. Okrem svojho ranku si každý procesor pamätá dve čísla `low` a `high`.

Je ďalej vytvorená matica `neighbour_matrix` podľa schémy 2. Táto matica obsahuje informácie o tom, komu má každý procesor posielať ďalej svoj `low` a `high` výstup. Procesory 10, 13, 16, 17 a 18 obsahujú spolu finálnu zoradenú postupnosť, a preto zasielajú správy ROOT procesoru.



Obr. 2: Schéma procesorov

Načítanie vstupu je riešené vo funkcii `distribute_numbers`. Túto akciu vykonáva iba procesor `ROOT`. Postupne je načítaných 8 čísel a tie sú rozoslané vždy po dvoch do procesorov 0 až 3, ako je možné vidieť na schéme 2. Čísla sú načítané zo súboru ako `unsigned char`, ale v programe je s nimi nárabané ako `int`. Na zaslanie čísel slúži funkcia `MPI_Send` s konkrétnym rankom príjemcu. Zároveň táto funkcia vypisuje nezoradené čísla na štandartný výstup.

Po rozdistribuovaní vstupných hodnôt každý procesor prijme dve hodnoty pomocou `receive_inputs`. Táto funkcia využíva `MPI_Recv` s parametrami pre rank odosielateľa `MPI_ANY_SOURCE` a tag `MPI_ANY_TAG`. Nemusíme kontrolovať odosielateľa nakoľko matica `neighbours_matrix` nám zaručuje, že správy budú poslané správnemu príjemcovi. Hodnoty sú načítané do premenných procesoru `low` a `high` a v prípade, že $low > high$ vymenené.

Následne každý procesor posiela svoje už porovnané hodnoty `low` a `high` ďalej podľa matice `neighbours_matrix`, ktorá obsahu rank príjemnu na základe ranku odosielateľa. Toto vykonáva funkcia `send_value_forward`.

Medzitým ROOT procesor vyčkáva na prijatie výslednej postupnosti. Pre usporiadanú postupnosť je pripravené pole `numbers`. Vo funkcii `collect_sorted_values` konštrukcia `switch` riadi ukladanie hodnôt prijatých zo špecifických procesorov na správny index vopred vytvoreného poľa. V závere je už len vypísané pole zoradených hodnôt na štandartný výstup.

4 Záver

V rámci projektu bol úspešne implementovaný algoritmus *Odd-even merge sort* jazyku C++ pre 8 čísel.