Autor: Jan Lorenc
(xloren15)

Datum: 20.03.2022

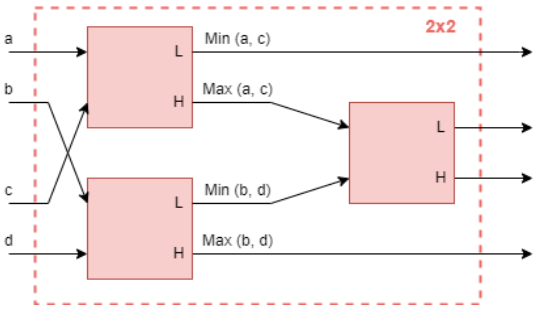
PRL - Paralelní a distribuované algoritmy

1. projekt – Odd-even merge sort

Algoritmus

Odd-even merge sort provádí řazení slučováním již seřazených posloupností. K tomu využívá síť komparačních jednotek, jejichž správné propojení zaručí korektně seřazenou výstupní posloupnost. Základní jednotkou je klasický binární komparátor (obrázek 1), který své vstupy porovná a vloží výsledky na své high/low výstupy. Samostatně tvoří síť 1x1. Pomocí těchto sítí lze kaskádově tvořit větší sítě, jako ukazuje obrázek 2.

Obr 1. Jednotka komparátoru (sítě 1x1)



Obr 2. Síť 2x2 poskládaná ze základních 1x1 sítí

Algoritmus se jmenuje odd-even, neboť pro vstup délky *n* vezme pozičně liché prvky a vloží je na vstup své první ze svých úvodních *n/2* jednotek a stejně tak sudé prvky předá druhé *n/2* jednotce. Demonstrovat to lze na obrázku 2, kde jsou 4 vstupy (*n=4*). Posloupnost se rozdělí na liché {a, c} a sudé {b, d} a ty jsou spolu dále porovnány v 1x1 sítích, které berou *n/2=2* vstupy.

Princip algoritmu je velmi jednoduchý a prakticky spočívá jen ve správném propojení komparátorů. Nevýhoda však tkví v tom, že nejsou vždy všechny jednotky vytíženy (např. na obr. 2 pracují nejprve první dvě jednotky, třetí čeká a poté pracuje jen třetí). Dále algoritmus vyžaduje příliš velké množství komparátorů, které rychle narůstá se zvětšující se vstupní posloupností.

Analýza složitosti

Studijní literatura[1] k předmětu odvozuje složitost následovně. Nechť $s(2^i)$ je čas potřebný v i-té fázi na spojení 2 seřazených posloupností o délce 2^{i-1} prvků, pak:

$$s(2) = 1 \qquad \text{pro } i = 1,$$
$$s(2^i) = s(2^{i-1}) + 1 \qquad \text{pro } i > 1,$$

řešením čehož je $s(2^i) = i$. Proto čas potřebný pro řazení odd-even merge sort sítí je

$$t(n) = \sum_{i=1}^{logn} s(2^i) = O(\log^2 n).$$

Dále nechť $q(2^i)$ je počet řadiček potřebných v i-té fázi na spojení 2 seřazených posloupností o délce 2^{i-1} prvků, pak:

$$q(2) = 1 \qquad \text{pro } i = 1,$$
$$q(2^i) = 2q(2^{i-1}) + 2^{i-1} - 1 \qquad \text{pro } i > 1,$$

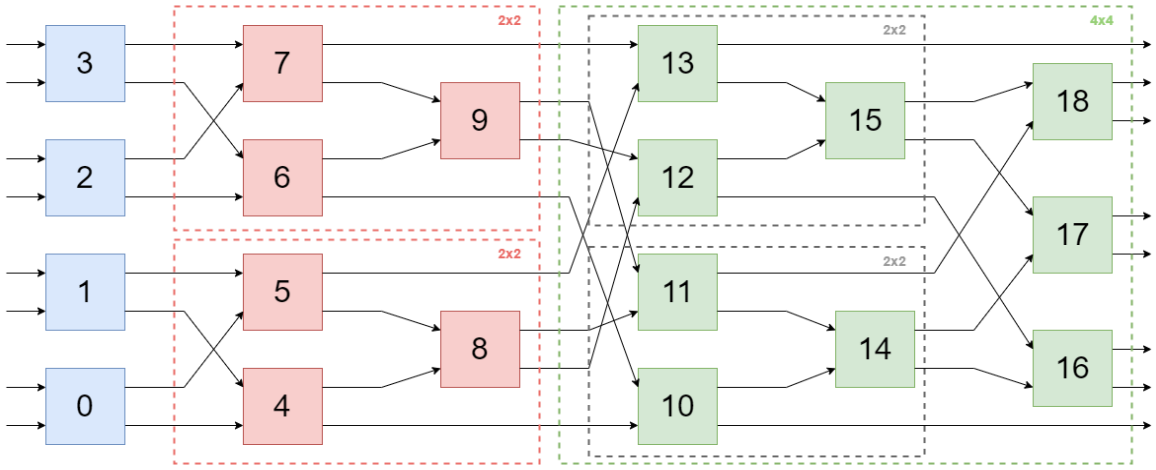
řešením čehož je $q(2^i) = (i-1)\,2^{i-1} + 1$. Počet řadiček pro řazení je poté

[1] Selim G. Akl. 1989. *The design and analysis of parallel algorithms*. Prentice-Hall, Inc., USA.

$$p(n) = \sum_{i=1}^{logn} 2^{(logn)-i} q(2^i) = O(n \log^2 n).$$

Celková cena je tedy: $O(\log^2 n)$ $O(n \log^2 n)$ = $\underline{O(n \log^4 n)}$, což není optimální.

Pro úkol v projektu k seřazení posloupnosti 8 čísel je potřeba tří úrovní. V první je zapotřebí čtyř sítí 1x1, v druhé pak 2 sítě 2x2 a v poslední jedna síť 4x4. Tuto hierarchii lze pozorovat i na obrázku 3. Celkem je tedy zapotřebí 19 komparátorů (z pohledu projektu procesů) a výsledku je dosaženo v 6 krocích, přičemž je však vidět, že nikdy nepracují zároveň více než 4 komparátory/procesy. To demonstruje neefektivní vytížení procesů a existuje prostor pro optimalizace.



Obr. 3 Struktura sítě pro řazení vstupní posloupnosti délky 8. Dle tohoto schématu byl projekt implementován a jednotlivé komparátory reprezentují procesy daného ranku. Dále lze vidět hierarchické členění na 4x 1x1, 2x 2x2 a 1x 4x4 bloky.

Implementace

Algoritmus je implementován v jazyce C++ s využitím knihovny Open MPI. Implementace počítá s 19 procesy, kde každý reprezentuje jeden určitý komparátor. O který komparátor se jedná proces zjistí z globálního pole struktur `Comparer` s názvem `Net`. Toto pole reprezentuje celou síť tak, jak je zobrazeno na obrázku 3, a je na pevno nastaveno. Index do pole značí komparátor a proměnné struktury odpovídají jeho vstupům a výstupům, tedy ke kterým komparátorům je napojen. Z pohledu zasílání zpráv MPI to značí, na které procesy se čeká a kterým se pošlou výstupy.

Při zahájení provádění programu si každý proces zjistí své číslo = `rank`. Toto udává, jak se bude daný proces chovat. Jedná-li se o proces 0, tedy master proces, přečte ze souboru `numbers` funkcí `read_input()` vstupní posloupnost a po dvou je zašle procesům 0-3, čímž zahájí řazení. Provádění programu pro všechny procesy pokračuje funkcí `simulate_comparer(int rank)`. Zde si proces z globálního pole `Net` zjistí, od koho má přijmout vstupy a komu poslat výstupy. Dále na oba vstupy čeká, po přijmutí je porovná a pošle dál. Následně běžné procesy končí a master čeká, než obdrží všechny výstupy, které pak vytiskne na standardní výstup.

Závěr

Byl naimplementován algoritmus odd-even merge sort pro vstupní posloupnost 8 čísel využívající 19 procesů, kde každý koresponduje s řadící jednotkou. Pro řešení bylo klíčové si uvědomit roli jednotlivých procesů a správně sestavit síť. Implementace dále nabízí možnosti pro optimalizaci procesů, což může být námětem další práce.