Dokumentácia k projektu z predmetu PRL : *Odd Even Transposition Sort*

Anton Firc (xfirca00)

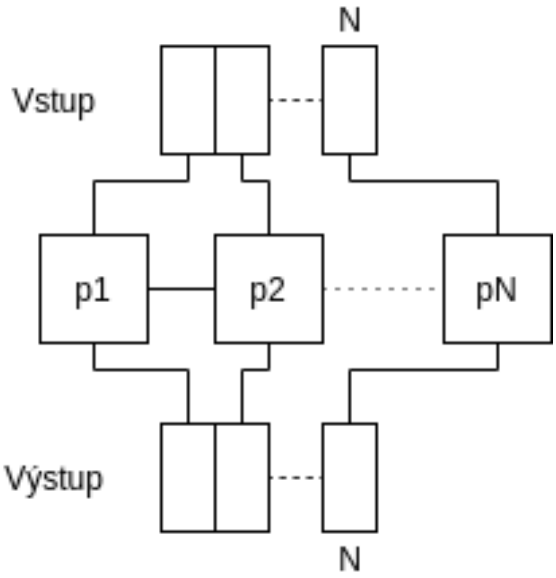
23. marca 2020

1 Úvod

Cieľom projektu je implementácia algoritmu *Odd Even Transposition Sort* použitím knižnice Open MPI a zmerať a odôvodniť jeho časovú zložitosť.

2 Rozbor a analýza algoritmu

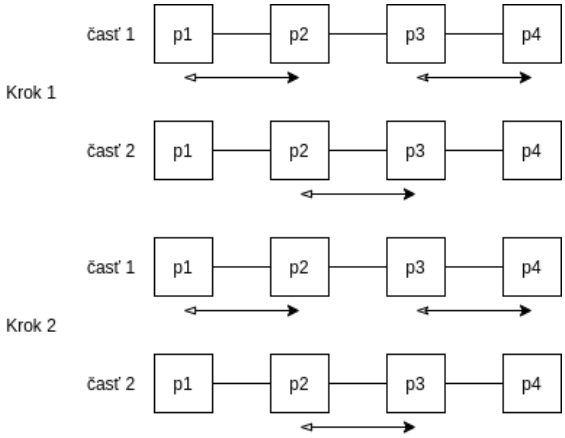
Odd Even Transposition Sort je paralelný radiaci algoritmus využívajúci lineárne pole o N procesoroch. Všetky procesory (okrem prvého a posledného) sú prepojené so svojim pravým a ľavým susedom. Prvý a posledný procesor sú prepojené len s pravým resp. ľavým susedom. Takéto prepojenie zabezpečuje, že si každý procesor dokáže vymeniť hodnotu s pravým susedom.



Obr. 1: Architektúra pre N čísel

Alogritmus najprv distribuuje N vstupných hodnôt medzi všetkých N procesorov. Každý procesor obdrží práve jednu hodnotu. Radenie následne prebieha v maximálne $n/2$ krokoch pozostávajúcich z dvoch častí:

1. Každý nepárny procesor p_n sa spojí so svojím susedom p_{n+1} a porovnajú svoje hodnoty. Následne, pokiaľ hodnota $p_n >$ hodnota p_{n+1} dôjde k výmene hodnôt procesorov p_n a p_{n+1}.
2. Každý párny procesor p_p sa spojí so svojím susedom p_{p+1} a porovnajú svoje hodnoty. Následne, pokiaľ hodnota $p_p >$ hodnota p_{p+1} dôjde k výmene hodnôt procesorov p_p a p_{p+1}.



Obr. 2: Vizualizácia dvoch krokov radenia

2.1 Zložitosť

Asymptotická časová zložitosť algoritmu *Odd Even Transposition Sort* je určená ako $O(n)$, kde n značí počet radených hodnôt[1]. Časová zložitosť je odvodená nasledovne:

- inicializácia procesorov vykonaná za konštantný čas $O(1)$
- distribúcia hodnôt procesorom vykonaná za lineárny čas $O(n)$
- radenie hodnôt vykonané za lineárny čas $O(n)$
- distribúcia výsledku vykonaná za lineárny čas $O(n)$

Po dosadení týchto čiastočných zložitostí do vzorca dostávame $t(n) = O(1 + n + n + n) = O(3n + 1) = O(n)$. Cena algoritmu je daná časovou zložitosťou a počtom procesorov ako $n * n$, takže $c(n) = O(n^2)$[1].

3 Implementácia

Algoritmus je implementovaný v jazyku C++, s využitím knižnice Open MPI podporujúcej paralelné výpočty. Beh programu začína inicalizáciou knižnice Open MPI volaním funkcie `MPI_Init()`. Následne sú prostredníctvom funkcií `MPI_Comm_size()` a `MPI_Comm_rank()` získané počet dostupných (bežiacich) procesov a vlastný identifikátor (ID) každého procesu. Hlavný (riadiaci) proces má ID 0, pre ostatné procesy je ID nenulové kladné celé číslo. Procesor s ID 0, následne načíta vstup zo súboru *numbers* a volaním funkcie `MPI_Send()` odošle ostatným procesom ich počiatočnú hodnotu. V prípade, že by došlo k načítaniu viac hodnôt než je dostupných procesov program skončí chybou. Všetky ostatné procesy[1] následne čakajú na príjem hodnoty, volaním funkcie `MPI_Recv()`. Každý proces po obdržaní svojej hodnoty pokračuje na fázu radenia.

Radenie je vykonávané vo *for* cykle, bežiacom v $n/2$ opakovaniach. Najprv dostávajú riadenie procesy s nepárnym ID, tie odošlú svojmu pravému susedovi (pokiaľ nejakého majú) svoju hodnotu, a následne čakajú na príjem novej hodnoty. Všetky párne procesy okrem procesu s ID 0 čakajú na prijatie hodnoty od ich ľavého suseda. Následne porovnajú prijatú hodnotu s ich vlastnou, menšiu z týchto dvoch hodnôt odošlú naspäť ľavému susedovi a väčšiu nastavia ako svoju. Následne dostávajú riadenie procesy s párnym ID, a rovnakým spôsobom predávajú hodnoty svojim pravým susedom, ktorí porovnajú prijatú a svoju vlastnú a odošlú menšiu z nich naspäť.

Po dobehnutí cyklu radenia každý proces, okrem procesu s ID 0, odošle svoju hodnotu procesu s ID 0. Proces s ID 0 následne vo *for* cykle najprv vypíše svoju hodnotu, následne postupne prijíma hodnoty od ostatných procesov a vypisuje ich na štandardný výstup.

Posledne je volaná funkcia `MPI_Finalize()` ktorá ukončí prostredie pre beh MPI a beh programu je úspešne ukončený.

[1]Proces s ID 0 nečaká na prijatie hodnoty, priamo si uchová prvú načítanú hodnotu zo súboru *numbers*

4 Experimenty

Implementácia algoritmu bola podrobená experimentom s rôzne veľkými vstupmi za účelom overenia teoretickej časovej zložitosti. Experimenty prebiehali na školskom serveri *merlin*, so vstupmi o veľkosti 4, 8, 12, 16, 20 a 24 hodnôt. Pre vstupy každej veľkosti bol meraný počet vykonaných inštrukcií všetkými procesmi dokopy, a čas behu algoritmu.

Pre merania výkonnosti algoritmu bola využitá knižnica PAPI[2], ktorá poskytuje rozhranie pre využitie počítadiel výkonnosti[3] dostupných vo väčšine procesorov.

Meranie počtu vykonaných inštrukcií zabezpečovala funkcia `PAPI_ipc()`[4]. Meranie počtu vykonaných inštrukcií prebiehalo v každom procese samostatne a následne hlavný proces spočítal počty inštrukcií od všetkých procesov a seba samého a vypísal celkovú hodnotu. Meranie bolo spustené ihneď po tom ako každý proces obdržal svoju hodnotu, a ukončené ihneď pred odoslaním finálnej hodnoty hlavnému procesu. Takto bolo zaistené aby počet inštrukcií zahŕňal iba samotnú časť radenia hodnôt, nie ich načítavania alebo distribúcie do a z procesov. Meranie celkového počtu vykonaných inštrukcií bolo zvolené pre demonštrovanie výpočtovej náročnosti algoritmu, bez ohľadu na to, ako dlho procesu trvá jednotlivé inštrukcie spracovať. Počet vykonaných inštrukcií sa samozrejme môže líšiť v závislosti od použitých procesorov, dôležitý je však vzťah medzi počtami pre jednotlivé veľkosti vstupov, podľa ktorého dokážeme odvodiť rast výpočtovej resp. časovej náročnosti výpočtu v závislosti na veľkosti vstupu.

Meranie času behu algoritmu takisto zabezpečovala vyššie uvedená funkcia `PAPI_ipc()`, ktorá okrem počtu vykonaných inštrukcií od jej prvého volania dokáže získať aj čas ubehnutý od jej prvého volania. Narozdiel od počtu vykonaných inštrukcií je čas behu algoritmu zaznamenávaný len v riadiacom procese, nie vo všetkých procesoch dokopy. Začiatok a koniec merania času sú vykonané v rovnaký čas ako začiatok a koniec merania počtu vykonaných inštrukcií. Takto dostávame ešte pohľad na reálnu dobu spracovania daného počtu inštrukcií na konkrétnom systéme. Znovu nás nezaujímajú konkrétne hodnoty namerané pre jednotlivé vstupy, ale vzťah medzi hodnotami pre jednotlivé veľkosti vstupov.

Pre vstup každej veľkosti bolo vykonaných 100 meraní. Následne bol získaný priemerný počet vykonaných inštrukcií a priemerný čas behu algoritmu pre každú veľkosť vstupu. Priemerný počet vykonaných inštrukcií v závislosti na veľkosti vstupu je uvedený v tabuľke 3 ktorej dáta sú vyobrazené v grafe 4. Priemerný čas behu algoritmu v závislosti na veľkosti vstupu je uvedený v tabuľke 5 ktorej dáta sú vyobrazené v grafe 6.

Režim merania výkonu je možné v programe zapnúť definovaním makra `BENCHMARK` a vytvorením zložky `benchmark`.

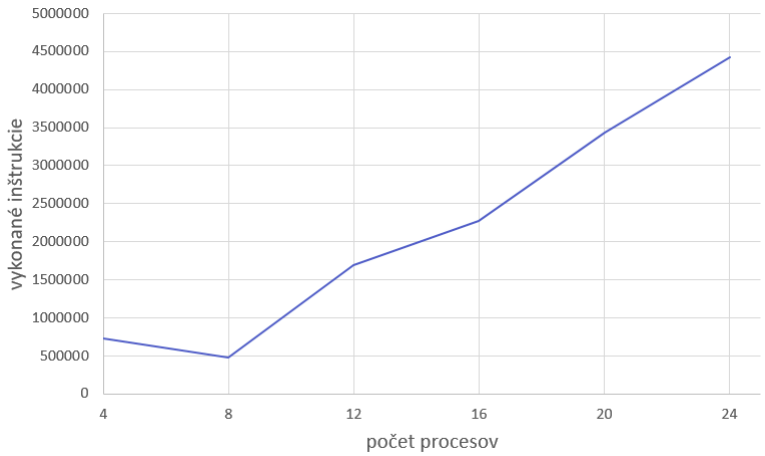
[2] https://icl.utk.edu/papi/

[3] z angličtiny, *performance counters*

[4] http://parallel.vub.ac.be/education/computerarchitectuur/man/html/papi_ipc.html

veľkosť vstupu	4	8	12	16	20	24
vykonané inštrukcie	733215,8	479832,23	1691908,77	2271768,21	3439663,88	4426561,92

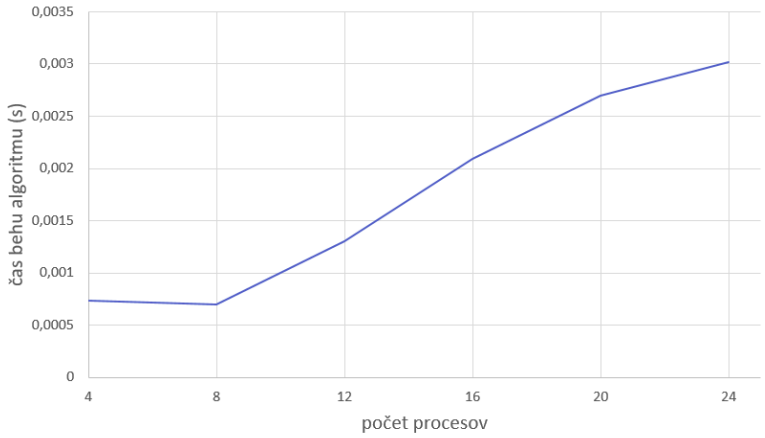
Tabuľka 3: Priemerný počet vykonaných inštrukcií v závislosti na veľkosti vstupu



Obr. 4: Priemerný počet vykonaných inštrukcií v závislosti na veľkosti vstupu

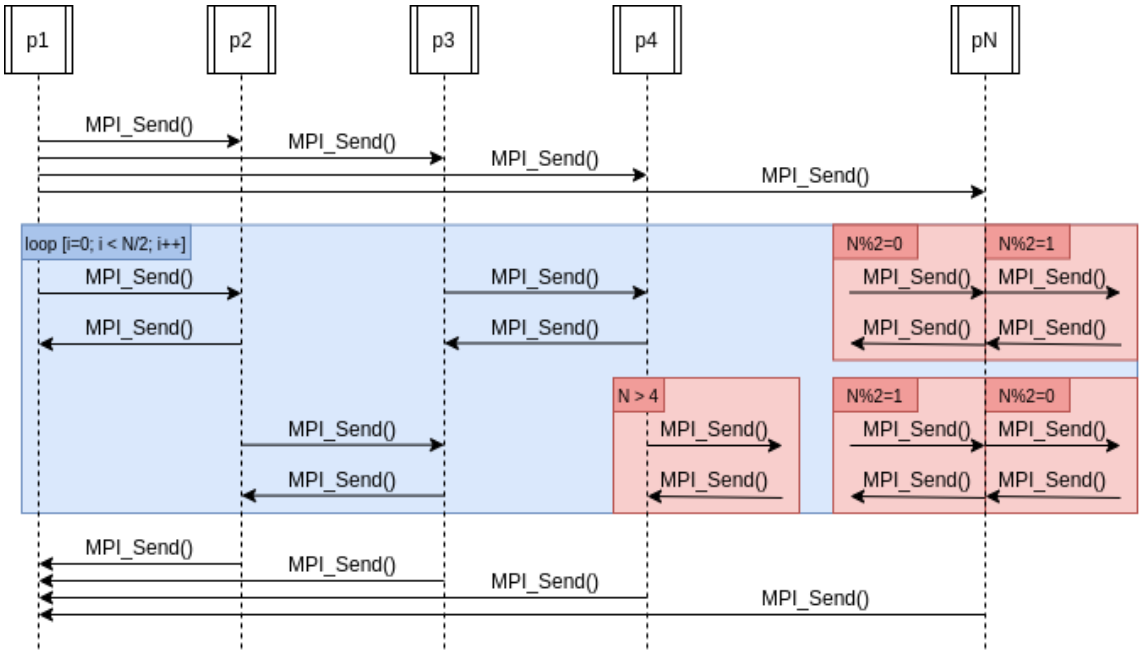
veľkosť vstupu	4	8	12	16	20	24
čas (s)	0,000736	0,000704	0,001309	0,002092	0,0026996	0,0030176

Tabuľka 5: Priemerný čas behu algoritmu v závislosti na veľkosti vstupu



Obr. 6: Priemerný čas behu algoritmu v závislosti na veľkosti vstupu

5 Komunikačný protokol



Obr. 7: Sekvenčný diagram komunikácie medzi procesmi

6 Záver

Experimenty so vstupmi rôznych veľkostí potvrdili teoretickú zložitosť algoritmu. Z grafov je vidno, že množstvo vykonaných inštrukcií takisto ako čas behu algoritmu lineárne rastie s počtom radených hodnôt. Odchýlku od lineárneho rastu vykazuje algoritmus pri 4 radených hodnotách. Domnievam sa, že táto odchýlka je spôsobená poklesom efektivity algoritmu pri radení malého počtu hodnôt[2].

Literatúra

[1] Hanáček, P.: Distribuované a paralelní algoritmy a jejich složitost. 2005, [Online; 22.03.2020]. URL `https://www.fit.vutbr.cz/study/courses/PDA/private/www/h003.pdf`

[2] Qureshi, K.: A Practical Performance Comparison of Parallel Sorting Algorithms on Homogeneous Network of Workstations. 07 2006, s. 615–619.