Dokumentácia k projektu do predmetu Pararelné a Distribuované algoritmy: Implementácia algoritmu *Enumeration sort*

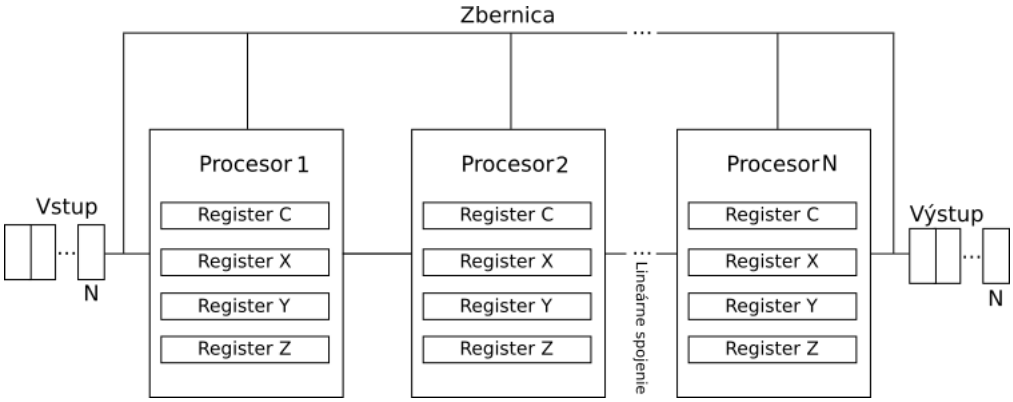
Autor: Filip Gulán (xgulan00)

1 Úvod

Úlohou tohoto projektu bolo implementovať v jazyku C/C++ pomocou knižnice Open MPI algoritmus *Enumeration sort* na lineárnom poli o n procesoroch. Okrem samotnej implementácie algoritmu bolo nutné vytvoriť riadiaci skript, ktorý riadi testovanie.

2 Rozbor a analýza algoritmu

Enumeration sort s lineárnou topológiou je pararelný radiacii algoritmus, ktorý pracuje na lineárnej architektúre. Všetky procesory sú spojené zbernicou, ktorá je schopná preniesť v každom kroku jednu hodnotu. Okrem toho sú všetky procesory prepojené pomocou lineárneho spojenia. Toto spojenie umožňuje presun hodnoty z registrov medzi susednými procesormi. Všetky procesory obsahujú 4 registre. Register *X* slúži na uchovanie hodnoty, ktorá bola pridelená zbernicou tomuto procesoru. Register *Y* slúži na postupné uchovávanie hodnôt súsediacich procesorov. Register *C* slúži na uchovanie hodnoty, ktorá predstavuje počet hodnôt menších, než hodnota v registre *X*. Teda hodnoty, ktorá sa inkrementuje vždy, v prípade, že hodnota v registre *X* je väčšia ako hodnota v *Y*. Register *Z* slúži ku koncu na uchovanie finálnej zoradenej hodnoty. Architektúra pre *N* čísel je je znázornená na obrázku 1.



Obrázok 1: Architektúra pre *N* čísel

Algoritmus *Enumeration sort* funguje tak, že sa v prvom kroku všetky registre *C* nastavia na hodnotu 1. V kroku 2 nasleduje cyklus od 1 do 2n ($1 <= k <= 2n$), v ktorom sa vykoná to, že sa najprv, pokiaľ vstup nie je vyčerpaný, vstupný prvok x_i pomocou zbernice vloží do registru X_i a lineárnym spojením do Y_1. Následne sa obsah všetkých registrov Y

posunie doprava. Potom každý procesor s neprázdnym registrom *X* a *Y* porovná, či je *X* väčšie ako *Y* a v kladnom prípade inkrementuje svoj register *C*. Ku koncu po vyčerpaní vstupu ($k > n$) procesor P_{k-n} pošle zbernicou obsah svojho registru *X* procesoru P_{Ck-n}, ktorý ho uloží do svojho registru *Z*. V poslednom treťom kroku v následujúcích *n* cykloch procesory posúvajú obsah svojich registrov *Z* doprava a procesor P_n produkuje zoradenú postupnosť.

Asymptotická časová zložitosť tohoto algoritmu je *O(n)*, kde *n* je počet radených prvkov. Časová zložitosť *O(n)* vychádza z toho, že inicializácia registrov *C* je vykonaná za konštantný čas *O(1*), distribúcia hodnôt a porovnávanie sú vykonané za lineárny čas *O(2n)* a distribúcia výsledku je vykonaná za lineárny čas *O(n)*. Pre výpočet je nutné využiť *p* procesorov a cena algoritmu je teda kvadratická $O(n^2)$. [1]

3 Implementácia algoritmu

Algoritmus je implementovaný v jazyku *C++* za použitia knižnice pre pararelné výpočty *openMPI*. V prvom rade je na začiatku programu nutné volať funkciu *MPI_Init()*, ktorá inicializuje *MPI* prostredie, a zároveň je treba naplniť premenné, ako je počet procesov a *id* aktuálneho procesu. Ďalej je program už vykonávaný podľa toho, či ide o proces s *id* 0, teda v tomto prípade o zbernicu/riadiaci proces, alebo o procesy s *id* väčším ako 0, v tomto prípade o procesory, ktoré obsahujú registre. Zbernica, alebo lepšie povedané procesor s *id* 0 na začiatku načíta hodnoty zo súboru *numbers* a vypíše ich na štandardný výstup na jeden riadok. Ďalej sa riadiaci procesor stará o to, že posiela hodnoty priamo do registrov X daných procesorov vo *for* cykle (pomocou funkcie *MPI_Send()*) a zároveň posiela hodnotu vždy prvému procesoru do registra *Y*. Procesory, zase tieto hodnoty prijímajú (pomocou *MPI_Recv()*) a následne si susedské procesory predávajú hodnotu *Y* tak, že procesor *k* pošle hodnotu registra *Y* procesoru *k+1*. Vždy keď procesor prijme hodnotu registru *Y*, tak ju porovná so svojou hodnotou *X* a v prípade, že je *X* väčšie ako *Y*, tak inkrementuje hodnotu registru *C*. Toto predávanie hodnoty Y je implementované vo *for* cykle, ktorý vykonáva každý procesor. Procesor najprv teda príjme hodnotu registra *Y* od suseda, porovná ju a následne prepošle ďalšiemu susednému procesoru. Keď prebehne tento cyklus *for*, tak sa v registri *C* nachádza pozícia, do ktorého procesora patrí daná hodnota *X*. Každý procesor potom najprv odošle svoju hodnotu *X* *C*-temu procesoru a zároveň po odoslaní každý procesor príjme hodnotu *X* od hociktorého procesoru (zabezpečené pomocou *MPI_ANY_SOURCE*) a uloží ju do registra *Z*. Po tom, ako sa v registroch *Z* nachádzajú zoradené hodnoty, tak sa začnú tieto hodnoty postupne vysúvať (zabezpečené pomocou *for* cyklu, ktorý vykonáva každý procesor od 1 do hodnoty aktuálneho *id* procesu). Posledný procesor vždy pošle svoju hodnotu X procesu s *id* 0. Proces s *id* 0 tieto hodnoty od posledného procesoru prijíma a ukladá do poľa a následne vypisuje na výstup, každú hodnotu na samostatný riadok. Algoritmus sám o sebe nedokáže pracovať s duplicitnými hodnotami. Tento problém bol vyriešený podľa článku [2], kedy sa iba mierne upraví podmienka, inkrementácie registru *C*.

4 Popis experimentov

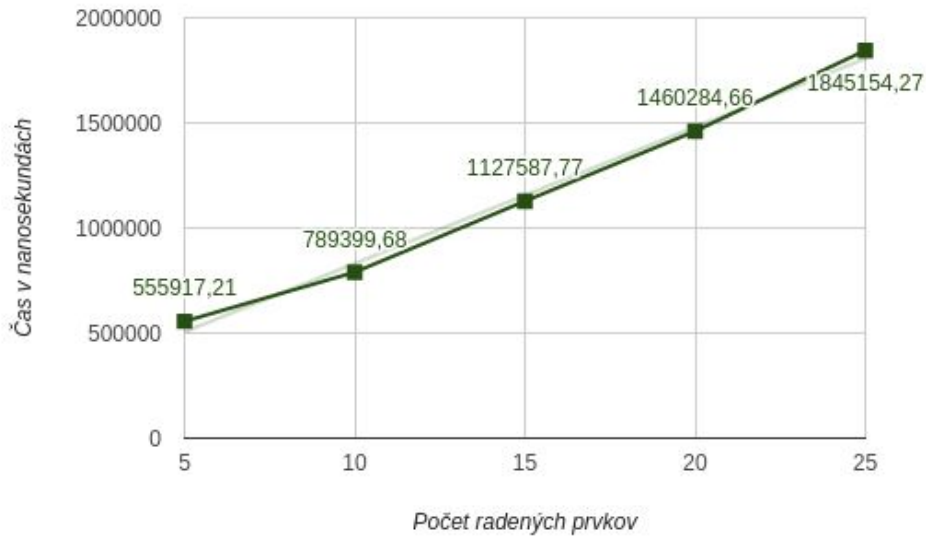
Implementovaný algoritmus bol testovaný experimentmi, na rôzne veľké vstupy pre overenie časovej zložitosti. Bol vytvorený špeciálny *bash* skript, ktorého výstupom boli časy behu programu. Tento skript testoval 10 rôznych vstupov o 5, 10, 15, 20 a 25 prvkoch a každý z týchto vstupov bol minimálne 10x zaraďovaný. Napríklad pre vstupy o veľkosti 5 prvkov bolo získaných 100 hodnôt (10x10).

Keďže záujmom merania bol samotný algoritmus a réžia okolo je v tomto prípade nezaujímavá, tak sa meranie spustilo pred tým, ako sa začali odosielať hodnoty jednotlivým procesorom a skončilo po tom, ako boli hodnoty zoradené (uložené do pomocného poľa). Načítanie hodnôt, inicializácia *openMPI*, výpis hodnôt nie sú v tomto čase zahrnuté. Na meranie času boli použité knižné funkcie *time.h*, presný popis použitej metódu je možné nájsť v [3]. Mód časového meranie je možné spustiť nastavením makra *MEASURE* na *true* v kóde programu.

Získané namerané hodnoty je možné vidieť v tabuľke 1 a pre názornosť boli vynesené do grafu zobrazeného na obrázku 2.

Počet hodnôt	Priemerný čas v ns
5	555917,21
10	789399,68
15	1127587,77
20	1460284,66
25	1845154,27

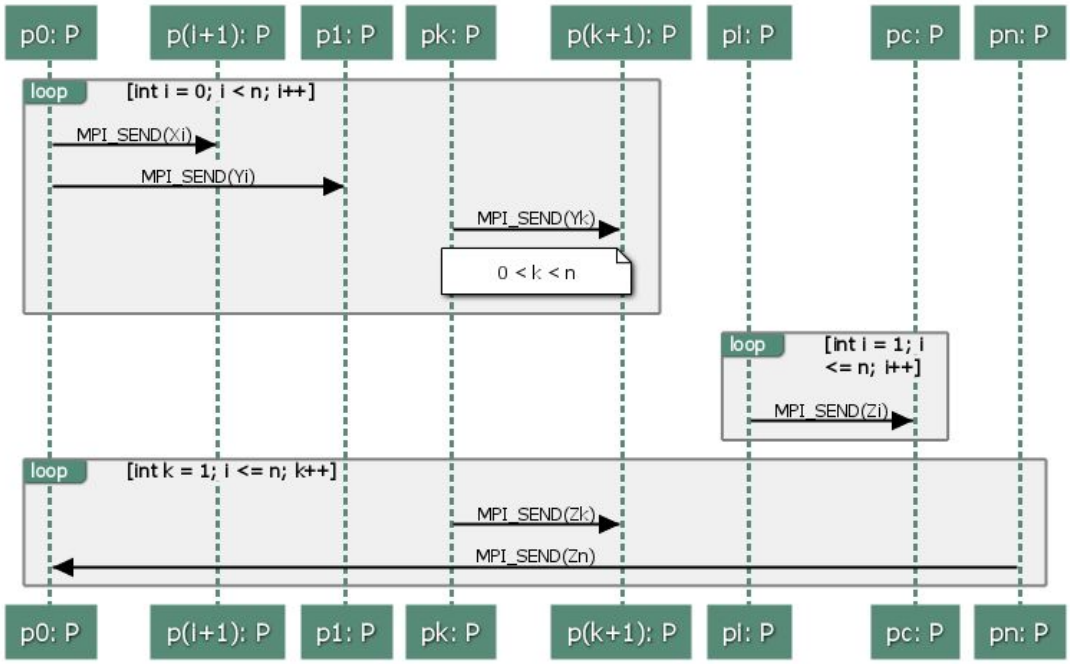
Tabuľka 1: Časy získané z experimentov



Obrázok 2: Výsledky merania zanesené do grafu

5 Komunikačný protokol

Komunikačný protokol je možné vidieť na obrázku 3. Komunikácia simuluje funkcie *openMPI*, a to konkrétne *MPI_Send()* a *MPI_Recv()*.



Obrázok 3: Sekvenčný diagram znázorňujúci komunikáciu medzi procesmi

6 Záver

Teoretická časová zložitosť algoritmu, uvedená v kapitole 2, odpovedá grafu získaného pomocou experimentov na obrázku 3. Z grafu je jasne vidieť, že počtom hodnôt, ktoré sa zoraďujú rastie čas lineárne.

Algoritmus bol riadne otestovaný na operačnom systéme *CentOS* (merlin) a na *Ubuntu 12* virtuálnom privátnom serveri pomocou mnou navrhnutej testovacej sady. Všetky testy dopadli úspešne.

Referencie

[1] HANÁČEK, Petr. *Distribuované a paralelní algoritmy a jejich složitost, algoritmy řazení, select* [online]. 2005 [cit. 2017-04-01]. Dostupné z: https://www.fit.vutbr.cz/study/courses/PDA/private/www/h003.pdf

[2] YASUURA, TAKAGI a YAJIMA. The Parallel Enumeration Sorting Scheme for VLSI. *IEEE Transactions on Computers* [online]. 1982, **C-31**(12), 1192-1201 [cit. 2017-04-01]. DOI:

10.1109/TC.1982.1675943. ISSN 0018-9340. Dostupné z: http://ieeexplore.ieee.org/document/1675943/

[3] RUTENBERG, Guy. *Profiling Code Using clock_gettime* [online]. 2007-9-26 [cit. 2017-04-05]. Dostupné z: http://www.guyrutenberg.com/2007/09/22/profiling-code-using-clock_gettime/