Dokumentácia k projektu do predmetu Paralelné a Distribuované algoritmy: Implementácia algoritmu *Mesh Multiplication*

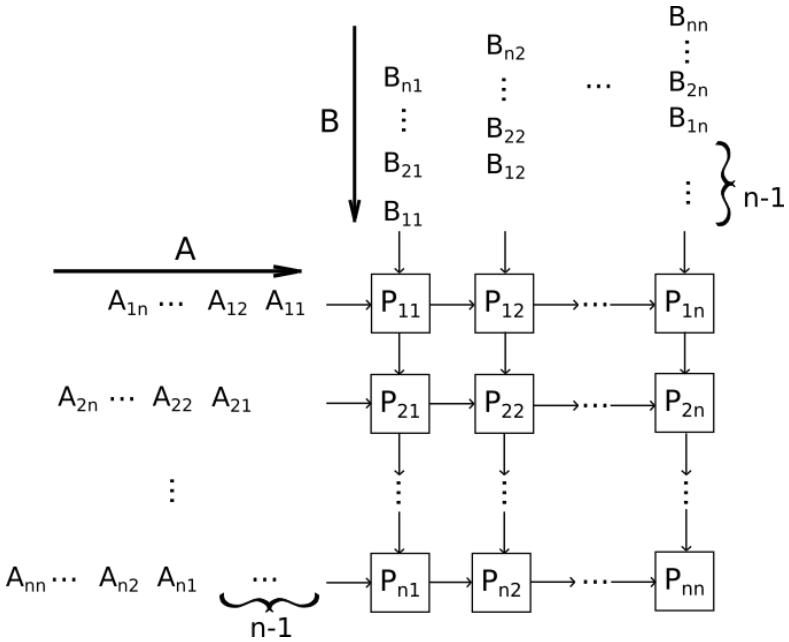
Autor: Filip Gulán (xgulan00)

1 Úvod

Úlohou tohto projektu bolo implementovať v jazyku *C/C++* pomocou knižnice *Open MPI* algoritmus *Mesh Multiplication*. Okrem samotnej implementácie algoritmu bolo nutné vytvoriť riadiaci skript, ktorý riadi testovanie.

2 Rozbor a analýza algoritmu

Mesh Multiplication je algoritmus na násobenie matíc, ktorý využíva pre svoj beh mriežku procesorov, kde táto mriežka obsahuje výsledné hodnoty súčinov matíc a hodnoty matíc, ktoré sa násobia, sa nachádzajú (matica *A* a matica *B*) mimo mriežku a sú postupne do mriežky privádzané. Každý z procesorov sa stará o jeden prvok výstupnej matice. Matica *A* je privádzaná do mriežky zprava a matica *B* zhora. Prvky nie sú privádzané zároveň, ale musia byť vždy o jedno miesto posunuté. Architektúra je znázornená na obrázku 1.



Obrázok 1: Architektúra mriežky *N x N*

Všetky procesory vykonávajú paralelne to, že na začiatku inicializujú register *C* na 0 a potom každý z procesorov v mriežke, pokiaľ dostáva hodnoty na vstupe, tak vykonáva ďalšie operácie. Procesor najprv získava na svojich 2 vstupoch *A* (zľava) a *B* (zhora) hodnoty, ktoré

vynásobí a pričíta k registru *C*. Nakoniec pošle hodnotu zo vstupu *A* pravému susediacemu procesoru a hodnotu *B* dolnému susediacemu procesoru.

Asymptotická časová zložitosť je lineárna *O(n)*, pretože prvky a_{m1} a b_{1k} potrebujú *m+k+n-2* krokov aby sa dostali k poslednému procesoru $P_{m,k}$. Počet potrebných procesorov je p^2 a cena algoritmu je teda n^3. [1]

3 Implementácia algoritmu

Algoritmus je implementovaný v jazyku *C++* za použitia knižnice pre paralelné 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. Potom nasleduje načítanie hodnôt matíc a ich rozmerov zo súborov do pomocného vektora. Toto načítanie vykonáva iba procesor s *id* 0. Tento proces zároveň distribuuje rozmery matíc ostatným procesorom, aby vedeli, koľko hodnôt majú prijímať od svojich susedov. Podľa týchto rozmerov si potom každý procesor vypočíta svoj i a j index, kvôli ľahšej a intuitívnejšej implementácii. Následne procesor s *id* 0 ešte rozdistribuuje hodnoty riadkov matice 1 procesorom nachádzajúcich sa v stĺpci 0, alebo presnejšie s *i* indexom 0. To isté sa urobí pre hodnoty stĺpcov matice 2, ktoré sa posielajú procesorom nachádzajúcim sa v riadku 0, alebo presnejšie s *j* indexom 0. Tieto krajné procesory si získané hodnoty uložia do svojich pomocných vektorov, z ktorých ich vyberajú a následne posielajú ďalej. Po tejto distribúcii hodnôt už môže začať samotné vykonávanie algoritmu. Tento algoritmus pozostáva z toho, že susedné procesory si navzájom posúvajú hodnoty. Presnejšie sa posúvajú hodnoty tak, že procesor hodnotu, ktorú dostal zľava pošle pravému susedovi, hodnotu, ktorú dostal od horného suseda zase pošle dolu. Počas posúvania hodnôt si procesory počítajú svoju aktuálnu hodnotu registra *C* tak, že procesor vynásobí svoju hodnotu získanú zľava s hodnotou získanou zhora a pripočíta ju k tomuto registru. Krajné procesory hodnoty získavajú zo svojich pomocných vektorov a následne ich preposielajú ďalej. Po tom, ako sa výpočet skončí, tak procesor 0 ešte zozbiera hodnoty od ostatných procesorov a následne výsledok vypíše na štandardný výstup.

4 Popis experimentov

Implementovaný algoritmus bol testovaný experimentmi na rôzne veľké matice pre overenie časovej zložitosti. Bol vytvorený špeciálny *bash* skript, ktorého výstupom boli časy behu programu. Tento skript testoval násobenie matíc rozmeru *2x2, 3x3, 4x4, 5x5, 6x6*. Každý rozmer bol násobený *100x* a následne bol na týchto hodnotách vykonaný orezaný priemer.

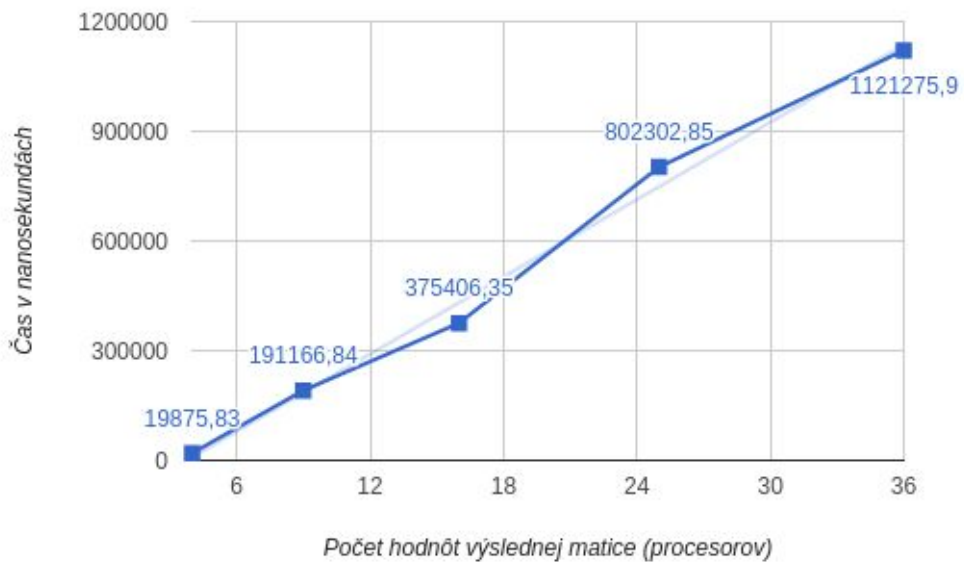
Keďže záujmom merania bol samotný algoritmus a réžia okolo je v tomto prípade nezaujímavá, tak sa meranie spustilo vtedy, keď procesor P_{00} prijal prvú hodnotu a skončilo, keď procesor P_{nn} prijal poslednú hodnotu. Načítanie hodnôt prvým procesorom, distribúcia hodnôt procesorom, inicializácia *openMPI*, získanie výsledných hodnôt a ich výpis nie sú v

tomto čase zahrnuté. Na meranie času boli použité knižné funkcie *time.h*, presný popis použitej metódy 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 procesorov	Orezaný priemer
4	19875,83
9	191166,84
16	375406,35
25	802302,85
36	1121275,9

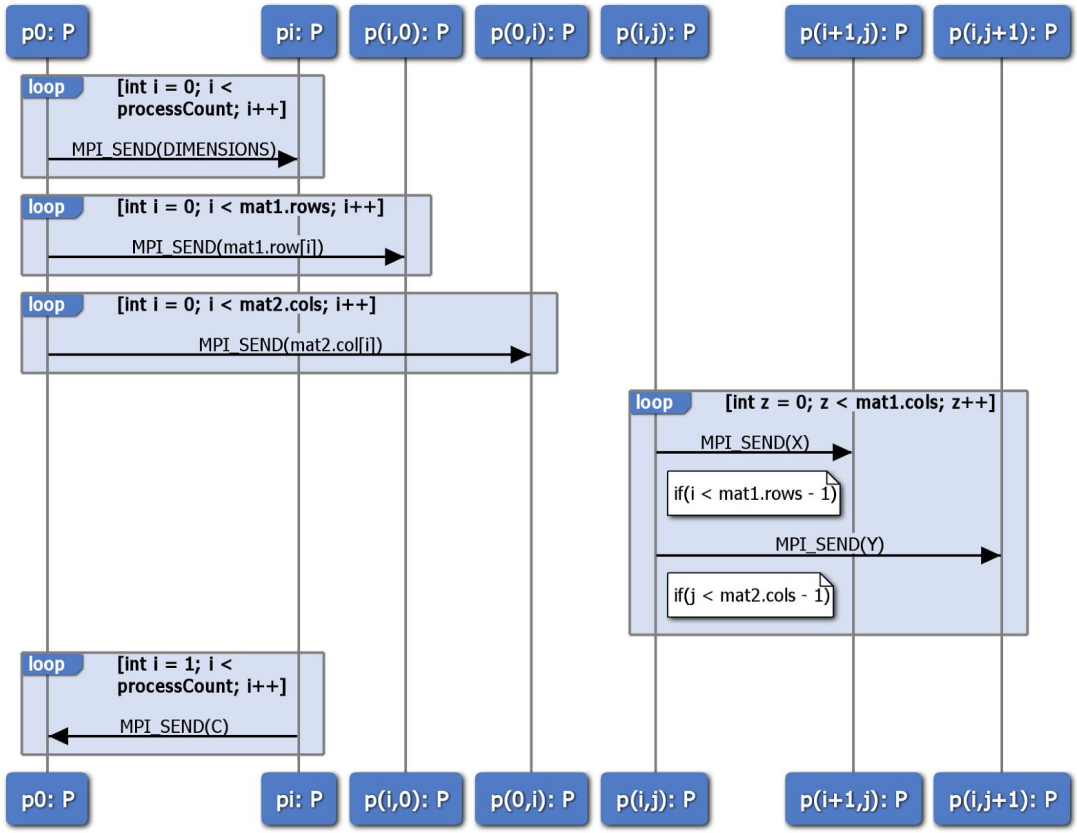
Tabuľka 1: Časy získané z experimentov



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 obsahuje okrem samotného algoritmu aj nutnú réžiu distribúcie hodnôt a ich konečné získavanie od procesorov.



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 zväčšovaním matíc, ktoré sa násobia, 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. *Vektorové a maticové algoritmy* [online]. 2005 [cit. 2017-04-01]. Dostupné z: https://www.fit.vutbr.cz/study/courses/PDA/private/www/h003.pdf

[2] 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/