تمرین درس سیستم های توزیع شده – مبحث MPI

نیمسال تحصیلی ۹۵۱

مرتضی ذاکری - ۹۵۷۲۳۰۸۸

تعداد مسئله ها: ۲

۱ تعیین مقدار π با استفاده از روش مونت کارلو

با اســتفاده از روش مونت کارلو که یک روش انتگرال عددی می باشــد مقدار عدد PI را تعیین نمایید. N نقطه تصــادفی در مربع با مشــخصــات x بین ۱- تا ۱ و y بین ۱- تا ۱ در نظر بگیرید و مقدار پی را از طریق رابطه زیر محاسبه نمایید.

$$\pi = 4\,n/N$$

n تعداد نقاطی است که درون دایره ای به شعاع یک قرار می گیرند.

a. در ابتدا یک کد سریال بنویسید که مقدار پی را محاسبه نماید.
b. سپس با استفاده از MPI کدی موازی بنویسید و مطمئن شوید که کدی که نوشته اید همان نتیجه کد سریال را باز میگرداند.
c. زمان را برای $N=10^8$ محاسبه نمایید هنگامی که از ۱ و ۲ و ۴ و ۱۶ پردازنده استفاده می نمایید. (برای این کار میتوانید از MPI_WTIME استفاده نمایید و یا از دستور موجود در UNIX به صورت time ./mc_pi استفاده نمایید.). در گزارش خود مطرح نمایید که آیا زمان initialization را در نظر گرفته اید یا خیر؟

۱ـ۱ روش مونت کارلو

در ابتدا روش مونت کارلو برای محاسبه عدد PI را مطرح می کنیم. در این روش دایره ای را در نظر گرفته که داخل یک مربع محاط شده است. اگر طول ضلع مربع را 2R در نظر بگیریم آن گاه شعاع دایره برابر R خواهد بود و نسبت مساحت دایره به مساحت مربع برابر خواهد بود با:

$$\pi R^2 / (2R)^2 = \pi / 4$$

تعدادی نقطه تصادفی تولید می کنیم به طوری که در محدوده ی داخلی مربع قرار گیرند. حال اگر نسبت محاسبه شده در رابطه فوق را برابر با نسبت تعداد نقاط تصادفی که داخل دایره قرار گرفته اند (n) به کل نقاط تولید شده (N) قرار دهیم، می توانیم مقدار π را از رابطه زیر با تقریب خوبی محاسبه کنیم. بدیهی هر چه تعداد نقاط تصادفی تولید شده بیش تر باشد آن گاه دقت محاسبه عدد π نیز بیش تر خواهد بود.

$$\pi / 4 = n / N \Rightarrow \pi = 4n / N$$

۱ـ۲ راه حل سریال

کد شکل ۱-۱ مقدار π را به صورت سریال حساب می کند و باز می گرداند. چند خروجی نمونه حاصل از اجرای کد با مقادیر مختلف برای تعداد نقاط نیز در زیر آمده است.

```
printf("pi = %lf ", cal_pi_serial(1000));
pi = 3.208000

printf("pi = %lf ", cal_pi_serial(10000));
pi = 3.146400

printf("pi = %lf ", cal_pi_serial(100000));
pi = 3.146280

printf("pi = %lf ", cal_pi_serial(100 000 000));
pi = 3.141019  // total execute time = 24 second
```

```c
#include <iostream>
#include <stdio.h>
#include <stdlib.h> // include rand function

#define sqr(x)  ((x)*(x))
using namespace std;

double cal_pi_serial(int darts)
{
    double x_coord,     /* x coordinate, between -1 and 1  */
        y_coord,        /* y coordinate, between -1 and 1  */
        pi,             /* pi  */
        r;              /* random number scaled between 0 and 1  */

    int score,          /* number of darts that hit circle */
        n;

    score = 0;

    for (n = 1; n <= darts; n++)
    {
        /* generate random numbers for x and y coordinates */
        r = (double)rand() / (double)RAND_MAX;
        x_coord = (2.0 * r) - 1.0;

        r = (double)rand() / (double)RAND_MAX;
        y_coord = (2.0 * r) - 1.0;

        /* if dart lands in circle, increment score */
        if ((sqr(x_coord) + sqr(y_coord)) <= 1.0)
            score++;
    }

    /* calculate pi */
    pi = 4.0 * (double)score / (double)darts;
    return(pi);
}
```

شکل ۱-۱محاسبه PI به صورت سریال توسط روش مونت کارلو در زبان C

علاوه بر این می توان چندین بار تابع فوق را در یک حلقه فراخوانی کرد و از مقادیر تولید شده میانگین گرفت تا دقت افزایش یابد. در زیر این کد آمده است:

```c
#define DARTS 1000
#define ROUNDS 1000

int main()
{
    double pi = 0;              /* average of pi after "darts" is thrown */
    double pi_avg = 0;          /* average pi value for all iterations */

    srandom (5);                /* seed the random number generator */

    for (int i = 0; i < ROUNDS; i++)
    {
       /* Perform pi calculation on serial processor */
       pi = cal_pi_serial(DARTS);

       pi_avg = ((pi_avg * i) + pi) / (i + 1);

       //printf(" After %3d throws, average value of pi = %10.8f\n", (DARTS * (i + 1)), avepi);
    }
    printf(" Final PI = %10.8f\n", pi_avg);

    printf("\nReal value of PI: 3.1415926535897 \n");

    return 0;
}
```

۱ـ۳ راه حل موازی با استفاده از MPI

برای حل به صـــورت موازی می توان از این ایده اســـتفاده کرد که هر یک از Slave ها مقدار PI را جداگانه محاســـبه کنند. Master نیز خود یک بار این مقدار را محاســـبه کند و سپس Slaveها مقدار محاسبه شده را به Master ارسال نمایند، Master این مقادیر را دریافت کرده، میانگین آن ها را محاسبه و نتیجه را در خروجی چاپ کند. کد راه حل ذکر شده در فایل parallel_pi_calc.c آمده است. برای اجرای برنامه ابتدا آن را کامپایل و سپس اجرا می کنیم:

- **$ sudo mpicc** parallel_pi_calc.c -o parallel_pi_calc
- **$ mpirun -f hosts -n 2** ./parallel_pi_calc

خروجی نهایی برنامه نیز برای N=10^8 و 100 دور محاسبه، به صورت زیر است:

- PI = 3.14225600

که با دقت دو رقم اعشار دقیقا برابر با مقدار خروجی کد سریال و نیز مقدار واقعی عدد PI است. همچنین زمان اندازه گیری شده برای N=10^8 هنگامی که از ۱ و ۲ پردازنده استفاده می کنیم به ترتیب برابر است با:

- **$ time ./parallel_pi_calc**
- real 0m1.075s
- user 0m0.744s
- sys 0m0.000s

- **$ time mpirun -f hosts -n 1 ./parallel_pi_calc**
- real 0m0.981s
- user 0m0.720s
- sys 0m0.008s

- **$ time mpirun -f hosts -n 2 ./parallel_pi_calc**
- real 0m2.134s
- user 0m1.064s
- sys 0m0.064s

بدیهی است که زمان Initialization نیز در اجراهای فوق در نظر گرفته شده است. با وجود این که سیستم بنده امکان اجرا بر روی تعداد ۴ و یا بیش تر پردازنده را نداشت ولی به راحتی می توان زمان این اجرا را نیز مشابه آن چه در بالا بیان شده به دست آورد.

*** فایل کد هر دو بخش سریال و موازی پیوست شده است.**

۲ پردازش آرایه با استفاده از استراتژی Master-Slave

۱_۲

ابتدا یک کد سریال بنویسید که این کار را انجام دهد. سعی کنید یک بار به صورت سطری و بار دیگر به صورت ستونی آرایه را پیمایش نمایید. در هر دو روش مقدار زمان لازم برای انجام اینکار را محاسبه نمایید و دلایل اختلاف را بیان نمایید.

پیمایش سطری

real 0m0.002s

user 0m0.000s

sys 0m0.000s

پیمایش ستونی

real 0m0.011s

user 0m0.000s

sys 0m0.000s

چون ماتریس ها در زبان C به صورت سطری در حافظه ذخیره می شوند. پیمایش سطری زمان کمتری نسبت به پیمایش ستونی می گیرد.

۲_۲

با استفاده از روش Master-Salve کد موازی بنویسید به طوری که Master آرایه را مقداردهی اولیه می نماید و سپس آرایه را بین slave ها توزیع می نماید. سپس slave ها محاسبات را انجام می دهند و داده ها در نهایت توسط master جمع آوری می شود تا خروجی نهایی پدید آید. در این قسمت به سوالات زیر نیز پاسخ دهید.

۱. آیا بهتر نیست Master قسمتی از آرایه را نگه دارد تا خود قسمتی از پردازش را انجام دهد؟

با توجه به این که Master در حین محاسبات Slave ها، کاری برای انجام دادن ندارد در صورت کم بودن تعداد Salveها بهتر است این کار انجام شود.

۲. شـما چگونه آرایه را بین پردازنده ها تقسـیم می کنید؟(روش خود را به طور مختصـر شرح دهید)

با ا ستفاده از تابع MPI_Scatter و از روی تعداد Slaveها یعنی world_size -1 هر سطر آرایه را بین Slaveها تقسیم می کنیم. سپس با استفاده از تابع MPI_Gather نتایج محاسبه شده را توسـط Master جمع آوری می نماییم. این کار را برای هر سـطر ماتریس انجام می دهیم و نهایتا در پردازه Master نتیجه را چاپ می کنیم.

۳. شما چگونه می توانید کاری کنید که تمامی پردازنده ها به یک میزان بار اجرایی داشته باشند؟ (Load Balance Between Processors)

با توجه به روش موازی سازی که در ق سمت ۲ مطرح شد، می توان ادعا نمود بار کاری تمامی پردازنده ها (لااقل تمامی Slaveها به یک میزان است) زیرا ارایه را به صورت مساوی بین همه Slaveها تقسیم نموده ایم.

*** فایل کد هر دو بخش سریال و موازی پیوست شده است.**

***