Параллельная реализация метода Монте-Карло
ЛАБОРАТОРНАЯ РАБОТА

http://www.kimrt.ru
Онлайн-курс
Суперкомпьютерные технологии в задачах моделирования

Как уже говорилось в одной из предыдущих лабораторных работ [1], методы Монте-Карло связаны с генерацией большого количества случайных (псевдослучайных) чисел. Таким образом, первое, что приходит в голову для написания параллельных версий таких алгоритмов, является распараллеливание генерации случайных величин. На примере рассмотренной ранее задачи о нахождении числа π проанализируем возможные варианты распараллеливания данного метода. В листинге 1 используется одномерный массив *NumPointInCircArr[numThreads]*, количество ячеек которого соответствует числу используемых потоков. В цикле генерируются случайные координаты точек. С помощью директивы *#pragma* библиотеки OpenMP [2, 3] этот цикл распараллеливается на некоторое количество потоков (*numThreads*). Каждый поток пишет информацию по количеству точек, попавших в круг, в свою ячейку массива *NumPointInCircArr*, номер которой соответствует номеру потока. После генерации случайных координат информация по количеству точек в круге суммируется по всем ячейкам массива *NumPointInCircArr* в отдельном цикле.

Листинг 1. Параллельная версия алгоритма

```
#define _USE_MATH_DEFINES
#include <iostream>
#include <math.h>
#include <random>
#include <ctime>
#include <omp.h>
#define a 2.0 //сторона квадрата
#define numThreads 4 //число потоков
using namespace std;
int main() {
```

```
    cout << "Program calculates PI by Monte Carlo
method.\n";
    cout<<"Maximal threads
number="<<omp_get_max_threads()<<endl;
    if (omp_get_max_threads() < numThreads){
        cout << "Threads number is not enough." << endl;
        cout << "Program is finished." << endl;
        system("pause");
        return 1;
    }
    omp_set_num_threads(numThreads);
    //общее количество точек
    long int NumPoint = 0;
    cout << "Enter number of points: ";
    cin >> NumPoint;
    //проверка ввода
    if ((cin.good()==false)||(NumPoint < 100)) {
        cout<<"\nWrong number or it's less than 100 or
too big!"
            <<endl;
        cout << "Program is finished." << endl;
        system("pause");
        return 1;
    }
    clock_t jobTime = clock();//время работы
    //число точек внутри четверти круга
    long int NumPointInCirc = 0;
    long int NumPointInCircArr[numThreads] = {0};
    double R2 = 0.25 * a * a;//квадрат радиуса круга
    std::mt19937 gen(time(0));//генератор случайных чисел
```

```
    std::uniform_real_distribution<> rnd(0, 1);
    #pragma omp parallel for
    for (long int i=0; i < NumPoint; i++){
        double x = rnd(gen); double y = rnd(gen);
        if ((x*x + y*y)<= R2)
            NumPointInCircArr[omp_get_thread_num()]+=1;
    }
    for (int j=0; j<numThreads; j++)
        NumPointInCirc += NumPointInCircArr[j];
    double pi = 4.0*NumPointInCirc/NumPoint;
    cout << "Calculated PI= " << pi << endl;
    cout << "Absolute error= " << abs(pi-M_PI)/M_PI <<
endl;
    //время в тактах микропроцессора
    jobTime = clock() - jobTime;
    cout<<"Execution time:
"<<(double)jobTime/CLOCKS_PER_SEC
        << " seconds" << endl;
    system("pause");
    return 0;
}
```

Второй вариант (листинг 2) заключается в использовании опции *reduction (+:NumPointInCirc)* при распараллеливании цикла с помощью *#pragma omp parallel for*. В этом случае у каждого потока создается своя локальная версия переменной *NumPointInCirc*. После окончания цикла значения этих локальных переменных суммируются автоматически и сохраняются в обобщенной переменной *NumPointInCirc*.

Листинг 2. Параллельная версия алгоритма №2

```
#define _USE_MATH_DEFINES
```

```
#include <iostream>
#include <math.h>
#include <random>
#include <ctime>
#include <omp.h>
#define a 2.0 //сторона квадрата
#define numThreads 12 //число потоков
using namespace std;
int main() {
    cout << "Program calculates PI by Monte Carlo method.\n";
    cout<<"Maximal threads number="<<omp_get_max_threads()<<endl;
    if (omp_get_max_threads() < numThreads){
        cout << "Threads number is not enough." << endl;
        cout << "Program is finished." << endl;
        system("pause");
        return 1;
    }
    omp_set_num_threads(numThreads);
    //общее количество точек
    long int NumPoint = 0;
    cout << "Enter number of points: ";
    cin >> NumPoint;
    //проверка ввода
    if ((cin.good()==false)||(NumPoint < 100)) {
        cout<<"\nWrong number or it's less than 100 or too big!"
            <<endl;
        cout << "Program is finished." << endl;
```

```
        system("pause");
        return 1;
    }
    clock_t jobTime = clock();//время работы
    //число точек внутри четверти круга
    long int NumPointInCirc = 0;
    double R2 = 0.25 * a * a;//квадрат радиуса круга
    std::mt19937 gen(time(0));//генератор случайных чисел
    std::uniform_real_distribution<> rnd(0, 1);
    #pragma omp parallel for reduction(+:NumPointInCirc)
    for (long int i=0; i < NumPoint; i++){
        double x = rnd(gen); double y = rnd(gen);
        if ((x*x + y*y)<= R2)
            NumPointInCirc++;
    }
    double pi = 4.0*NumPointInCirc/NumPoint;
    cout << "Calculated PI= " << pi << endl;
    cout << "Absolute error= " << abs(pi-M_PI)/M_PI <<
endl;
    //время в тактах микропроцессора
    jobTime = clock() - jobTime;
    cout<<"Execution time:
"<<(double)jobTime/CLOCKS_PER_SEC
        << " seconds" << endl;
    system("pause");
    return 0;
}
```

Задание

Напишите параллельную программу приближенного определения объема шара с радиусом $R = 1$, вписанного в куб со стороной $a = 2$, методом Монте-Карло. Программа запрашивает количество случайных чисел N и выводит на экран приближенный ответ и погрешность. Подумайте, как можно улучшить алгоритм, например, используя генерацию большого количества случайных чисел на видеокарте [4]. Сравните время выполнения последовательной и параллельных версий программы, постройте графики и напишите отчет.

http://www.kimrt.ru/index/course_stm/0-24

Проект реализуется победителем Конкурса на предоставление грантов преподавателям магистратуры 2020/2021 благотворительной программы «Стипендиальная программа Владимира Потанина» Благотворительного фонда Владимира Потанина.

Источники

1. Метод Монте-Карло.
http://www.kimrt.ru/courses/stm/LabWork/MonteCarlo.pdf
2. Настройка OpenMP проекта в Visual Studio.
http://www.kimrt.ru/courses/stm/self_work/OpenMP.pdf
3. Настройка OpenMP проекта в Code::Blocks под операционной системой Linux Debian 11.
http://www.kimrt.ru/courses/stm/self_work/openmpInDebian.pdf
4. Установка CUDA toolkit.
http://www.kimrt.ru/courses/stm/self_work/Cuda_Toolkit.pdf