РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

Отчет по лабораторной работе № 3

Дисциплина: Параллельное программирование

Студент: Логинов Сергей Андреевич

Группа: НФИбд-01-18

МОСКВА 2021г

Задание

1.
    - Используйте подпрограмму random_number и напишите функцию, которая возвращает случайное целое число *i* такое, что $1 \le i \le N$
    - Проверьте на примере $n = 6$, что вероятность выпадения чисел 1, 2, 3, 4, 5, 6 одинакова
    - Проверку осуществите проведя минимум $N=10^6$ испытаний методом Монте-Карло
    - Программу ускорьте с помощью параллельных потоков OpenMP

Для начала создадим модуль rand_f и в нем создадим функцию rand_func.

Данная функция принимает граничное значение N для интервала 1;N и возвращает случайное целое число из этого интервала. Код модуля:

```
module rand_f
    implicit none
    contains
    integer function rand_func(N) ! функция, возвращающая случайное целое число
    integer, intent(in) :: N       ! в промежутке от 0 до N
    real :: u
    call random_number(u)
    rand_func = 1 + FLOOR(N*u) ! пеходим с промежутка 0;1 к промежутку 1;N
end function
end module rand_f
```

С помощью функции random_number мы получаем случайное число из промежутка 0;1. Далее приводим его к нашему интервалу 1:N

В самой программе мы создадим массив, в который будем записывать количество выпавших чисел на нашем интервале:

```
program main
    use rand_f
    implicit none
    include "omp_lib.h"
    integer :: j, i, th
    integer, dimension(1:6) :: test ! массив для записи количества выпадения чисел
    test = 0
```

Вычисления выполняем параллельно:

```
!$omp parallel private(i, th) num_threads(6)
    th = omp_get_thread_num() + 1
    !$omp do
    do i = th, 10**6, 6 ! каждый поток будет выполнять свою последовательность итераций
        j = rand_func(6) ! случайное целое число от 1 до 6
        test(j) = test(j) + 1 ! каждый элемент массива - количество выпадений числа от 1 до 6
    end do
    !$omp end do
    print *, th, " имеет вероянтость", real(test(th))/sum(test) ! каждый поток выводит собственную информацию
    !$omp end parallel
end program main
```

Мы пользуемся параллельным циклом, в котором каждый поток вычисляет свою последовательность итераций цикла.

Каждую итерацию цикла мы получаем число, соответствующее ему значение в массиве увеличиваем на 1.

В конце каждый поток печатает результаты своей работы.

```
(base) sergejloginov@MacBook-Air-Sergej src % gfortran -fopenmp lab03_ex1.f90
(base) sergejloginov@MacBook-Air-Sergej src % ./a.out
           5  имеет вероятность  0.167323694
           3  имеет вероятность  0.166414812
           2  имеет вероятность  0.166173890
           1  имеет вероятность  0.167290851
           4  имеет вероятность  0.166546211
           6  имеет вероятность  0.166250542
```

Мы получили очень близкие друг к другу значения вероятности, погрешность в пределах тысячных.

2. Смоделируйте задачу про парадокс Монти Холла методом Монте-Карло для двух стратегий игрока:

   - Игрок никогда не меняет свое первое решение
   - Игрок всегда меняет первое решение

   Программа должна работать в параллельном режиме. Вычислите частоту побед для обеих стратегий

Моделируем задачу следующим образом:

Имеется три числа: 1, 2 и 3. Победить можно выбрав правильное число. Имеется 1/3 - вероятность правильного выбора. То есть вероятность того, что одно из невыбранных чисел верное 2/3. Далее узнаем, что одно из невыбранных чисел неверное. Предлагается изменить решение. Начальные вероятности сохраняются. Решим данную модель методом Монте-Карло.

Для начала создадим массив, в который запишем количество выпавших 1, 2 и 3:

```
program main
    use iso_fortran_env
    implicit none
    include "omp_lib.h"
    real :: u, y
    real(Real64) :: t1, t2, t3, t4, z1, z2
    integer i, j, k, q
    real, dimension(1:3) :: test, test1
    test = [0.0, 0.0, 0.0]
```

Далее используем параллельный цикл и проводим 10^6 проверок. В итоге наш массив заполняется количеством выпавших чисел. Добавим проверку времени выполнения:

```
t1 = omp_get_wtime()
    !$omp parallel private(i, j, u) shared(test) num_threads(8)
    !$omp do
    do i = omp_get_thread_num() + 1, 10**6, 8
        call random_number(u) !получаем случайное число от 0 до 1
        j = 1 + FLOOR(3*u) !    и переводим его в пределы от 1 до 3
        test(j) = test(j) + 1.0 ! записываем количество выпавших
единиц, двоек и троек
    end do
    !$omp end do
    !$omp end parallel
    t2 = omp_get_wtime()
    z1 = t2 - t1
    print *, "Вероятность выиграть без изменения выбора: ",
test(1)/sum(test)
    print *, "Вероятность выиграть с изменением выбора: ", (test(2) +
test(3))/sum(test)
    print *, "Время выполнения через потоки: ", z1
```

Далее пишем аналогичный код для последовательных вычислений без использования потоков (используются другие переменные и другой массив для чистоты эксперимента):

```
test1 = [0.0, 0.0, 0.0]
    t3 = omp_get_wtime()
    do q = 1, 10**6
        call random_number(y)
        k = 1 + FLOOR(3*y)
        test1(k) = test1(k) + 1.0
    end do
    t4 = omp_get_wtime()
    z2 = t4 - t3
    print *, "Время выполнения без использования потоков: ", z2
    print *, " Проверка ", z2 / z1
end program main
```

Результат выполнения программы:

```
(base) sergejloginov@MacBook-Air-Sergej src % gfortran -fopenmp lab03_ex2.f90
(base) sergejloginov@MacBook-Air-Sergej src % ./a.out
 Вероятность выиграть без изменения выбора:   0.337150931
 Вероятность выиграть с изменением выбора:   0.662849069
 Время выполнения через потоки:    2.0520000252872705E-003
 Время выполнения без использования потоков:    1.2177999946288764E-002
  Проверка    5.9346977564407686
```

Результаты совпадают с теоретическими. Параллельные вычисления дали выигрыш в производительности в 5,93 раза.

3. Смоделируйте задачу про парадокс дней рождения методом Монте-Карло. Вычислите частоту совпадения дней рождения для 10^6 испытаний. Постройте график изменения этой частоты в зависимости от количества людей в группе (от 2 до 100 человек). Программа должна работать в параллельном режиме.

В данной программе мы выясняем вероятность совпадений дней рождения у групп людей количеством от 2 до 100 человек. Используем метод Монте-Карло.

Суть заключается в выборе начального значения и дальнейшем поиске равного ему среди случайно выбранных в цикле (все числа в промежутке от 1 до 365, что можно считать датой рождения)

Для начала создадим нужные переменные и зададим значение нашему числу и создадим массив вероятностей :

```
include "random.f90"
program main
    use random
    implicit none
    include "omp_lib.h"
    integer :: j, i, k, q
    integer :: test
    real, dimension(1:100) :: p
    test = 56 ! зададим начальное значение, к которому хотим искать соотвествие среди случайных чисел
    p = 0.0
```

Далее в параллельном цикле мы пройдемся сначала по размерности группы (от 2 до 100 человек) и для каждой размерности проведем 10^6 экспериментов с выбором случайных чисел. Количество совпадений с начальным значением записываем в соответствующие элементы массива:

```
!$omp parallel private(i, q, k) num_threads(8)
    !$omp do
    do i = 2, 100 ! группы от 2 до 100 человек
        do q = 1, 10**6    ! 10**6 экспериментов методом Монте-Карло
            do k = 2, i ! находим вероятность того, что наше выбранное число совпадает со случайно сгенерированным функцией randomint
                j = randomint(365)
                if (j == test) then
                    p(i) = p(i) + 1.0 ! считаем совпадения
```

```
                end if
            end do
        end do
    end do
    !$omp end do
    !$omp end parallel
```

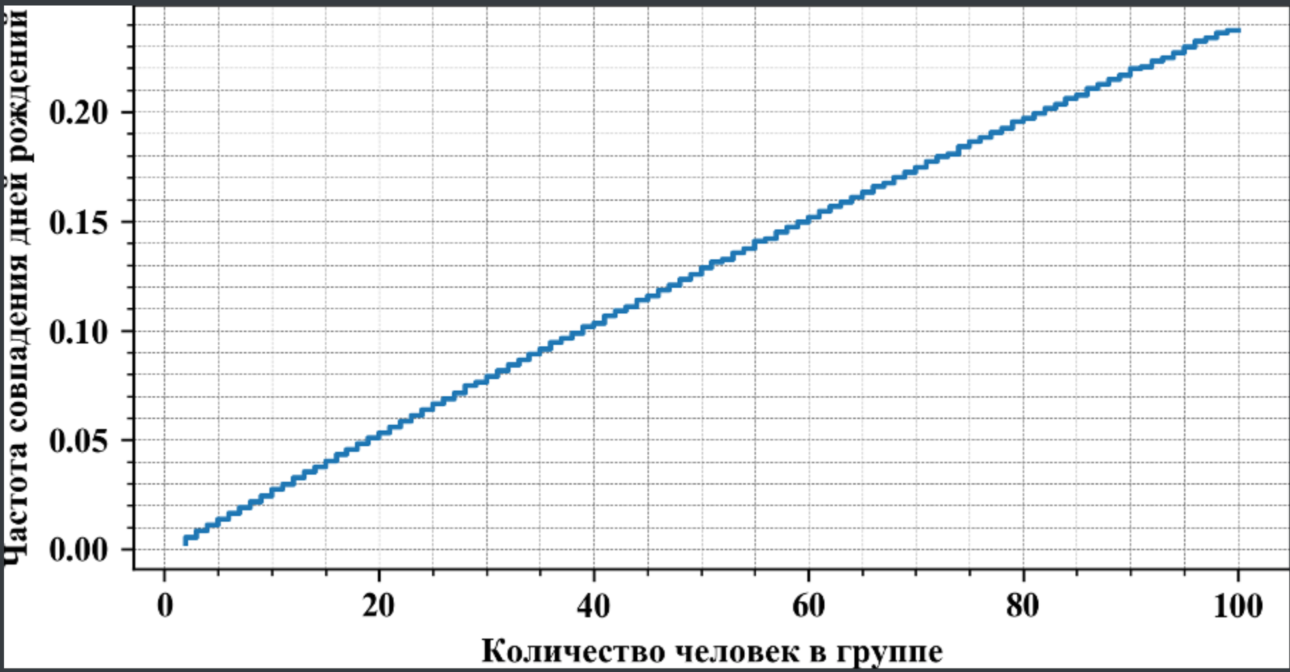
Далее от количества встреч перейдем к вероятностям а также запишем вероятности в файл (понадобится для построения графика):

```
do i = 2, 100 ! с двойки тк вероятность совпадения при одном человеке
ноль
        p(i) = (p(i)) / 10**6
        print '(f5.3)', p(i)
    end do
    open(1, file = "birthday.txt", status = "old") ! запишем
вероятности в файл
    do i = 1, 100
        write(1, *) p(i)
    end do
    close(1)
end program main
```

Результат выполнения программы:

```
(base) sergejloginov@MacBook-Air-Sergej src % gfortran -fopenmp lab03_ex3.f90
(base) sergejloginov@MacBook-Air-Sergej src % ./a.out
0.003
0.006
0.008
0.011
0.014
0.017
0.019
0.022
0.024
0.027
0.030
0.033
0.036
0.038
0.041
0.044
0.046
0.049
0.052
0.055
0.057
0.060
0.063
0.066
0.068
0.071
0.074
0.076
0.080
0.082
0.085
0.087
0.090
0.094
0.096
0.099
0.101
0.104
0.107
0.110
0.113
0.115
0.118
0.121
0.123
0.126
0.129
0.131
0.134
0.137
```

Для найденных значений построим график:



4. Смоделируйте задачу о разорении игрока методом Монте-Карло. Вычислите частоту побед и проигрышей. Вычислите также среднее число раундов в каждой игре.

В данном случае мы пользуемся предоставленным модулем random и его функцией random_walk_1d, которая представляет собой модель разорения.

Для начала задаем стартовые значения:

```
include "random.f90"
program main
    use iso_fortran_env
    use random
    implicit none
    include "omp_lib.h"
    integer(int64) :: r, rounds = 0
    real(real64) :: t1, t2, t3, t4
    logical(1) :: win
    real :: p, winrate
    integer :: i, start, end, n, wins = 0
    p = 0.5 ! вероятность выигрыша
```

```
        r = 0 ! кол-во раундов
        start = 0 ! минимальное значение(проигрыш)
        end = 10 ! конечное значение(выигрыш)
        n = 9 ! текущее значение
```

Далее используем обычный цикл, в котором считаем количество побед с нашими условиями из 10^6 итераций и количество раундов. Также производим замер времени и подсчет вероятности победы и среднего количества раундов:

```
    t1 = omp_get_wtime()
        do i = 1, 10**6
            call random_walk_1d(start, end, n, p, r, win)
            if(win) then
                wins = wins + 1 ! считаем количество попед
            end if
            rounds = rounds + r ! и количество раундов
        end do
        t2 = omp_get_wtime()
        t3 = t2 - t1
        winrate = real(wins) / 10**6 !частота побед
        rounds = rounds / 10**6 ! среднее количество раундов в игре
        print *,"Вероятность победы ", winrate, " Среднее количество
    раундов ", rounds
        print *, " time = ", t3
```

Обнуляем значения побед и раундов и выполняем те же самые действия с использованием потоков:

```
        wins = 0
        rounds = 0
```

```
    t1 = omp_get_wtime()
    !$omp parallel private(i) num_threads(8)
    !$omp do reduction(+ : wins, rounds)
    do i = 1, 10**6
        call random_walk_1d(start, end, n, p, r, win)
        if(win) then
            wins = wins + 1
        end if
        rounds = rounds + r
    end do
    !$omp end do
    !$omp end parallel
    t2 = omp_get_wtime()
    t4 = t2 - t1
    winrate = real(wins) / 10**6
    rounds = rounds / 10**6
    print *, "Вероятность победы ", winrate, " Среднее количество раундов ", rounds
    print *, " time = ", t4
    print *, "Выигрыш в производительности составил", t3 / t4 ! проверим, что параллельное вычисление дало выигрыш в производительности
end program main
```

Результат выполнения:

```
(base) sergejloginov@MacBook-Air-Sergej src % ./a.out
 Вероятность победы   0.899892986      Среднее количество раундов                    9
  time =   0.15086699998937547
 Вероятность победы   0.899613023      Среднее количество раундов                    9
  time =   0.11728499992750585
 Выигрыш в производительности составил   1.2863281756629299
```

Получаем результаты, аналогичные таблице (с учетом небольшой погрешности).

5. Смоделируйте игру в жетоны тремя игроками методом Монте-Карло. Вычислите частоту побед и проигрышей. Вычислите среднюю продолжительность раундов в каждой игре и сравните ее с теоретическим значением.

Моделируем следующим образом: игроки выбирают число от 1 до 3, далее получаем случайное целое из этого промежутка, победитель +2 жетона, проигравшие -1 жетон каждый. Играем до разорения какого-либо игрока.

Для начала объявим необходимые переменные и находим решение по формуле:

```
include "random.f90"
program main
    use iso_fortran_env
    use random
    implicit none
    include "omp_lib.h"
    integer :: p, i, j, round, x = 10, y = 10, z = 10
    real :: test
    round = 0
    test = x * y * z / (x + y + z - 2)
```

Далее делаем 10^6 проверок методом Монте-Карло, то есть во внутреннем цикле моделируем отдельно взятую игру. При банкротстве хотя бы одного игрока выходим из цикла и фиксируем значение счетчика цикла - оно и будет количеством раундов.

В конце делим количество раундов на количество итераций внешнего цикла проверок:

```
    do j = 1, 10**6 ! метод Монте-Карло
        x = 10  ! начальное количество жетонов у каждого игрока
        y = 10
        z = 10
        do i = 1, 100 ! воспроизводим отдельно взятую игру
            p = randomint(3) ! если выпадает 1, побеждает первый(x), 
забирает два жетона, у второго и третьего -1 жетон и тд
            select case(p)
                case(1)
```

```fortran
                    x = x + 2
                    y = y - 1
                    z = z - 1
                case(2)
                    x = x - 1
                    y = y + 2
                    z = z - 1
                case(3)
                    x = x - 1
                    y = y - 1
                    z = z + 2
            end select
            if (x == 0 .or. y == 0 .or. z == 0) then
                round = round + i ! фиксируем количество раундов в данной игре
                exit ! выходим из цикла
            end if
        end do
    end do
    print *,"Среднее количество раундов программно = ", real(round) / 10**6 ! находим среднее количество раундов в играх
    print *,"Среднее количество раундов по формуле = ", test
    print *,"Погрешность = ", test - real(round) / 10**6
end program main
```

Результат выполнения программы:

```
(base) sergejloginov@MacBook-Air-Sergej src % gfortran -fopenmp lab03_ex5.f90
(base) sergejloginov@MacBook-Air-Sergej src % ./a.out
 Среднее количество раундов программно =    32.8623734
 Среднее количество раундов по формуле =    35.0000000
 Погрешность =    2.13762665
```

К сожалению была получена достаточно значительная погрешность.