Лабораторная работа №1
Молодцов Владислав

1 Задача 1

Описание циклов представлены в виде комментариев в программе (см. рис. 1). В данном случае в первой паре циклов, где определяются значения a, никаких зависмостей нет, а в третьей циклов, где происходит вывод значений, используется один и тот же файловый дескриптор, поэтому их распараллеливание невозможно. Во второй паре циклов, где происходят сами вычисления, мы имеем loop-carried dependency, поэтому можно параллелить оба цикла, а также переставлять их местами, если очень хочется.

```
#pragma omp parallel shared(a) private(i, j)
{
    double start = omp_get_wtime();

    int num = omp_get_thread_num();
    int num_threads = omp_get_num_threads();

    // there are no dependencies here therefore we are able to parallel both loops
    #pragma omp for schedule(static) collapse(1)
    for (i=0; i<ISIZE; i++)
    {
        for (j=0; j<JSIZE; j++)
        {
            a[i][j] = 10*i +j;
        }
    }

    // there are only loop carried dependencies here therefore we are able to parallel both loops
    // distance vector (0,0), direction vector (=,=)
    #pragma omp for schedule(static) collapse(1)
    for (i=0; i<ISIZE; i++)
    {
        for (j = 0; j < JSIZE; j++)
        {
            a[i][j] = sin(0.00001*a[i][j]);
        }
    }

    double end = omp_get_wtime();
    if (num == 0)
    {
        printf("%d\t%lf\n", num_threads, end-start);
    }
}

ff = fopen("program_results/par_result.txt","w");

// we are unable to parallel that since it uses the same file descriptor
for(i=0; i < ISIZE; i++)
{
    for (j=0; j < JSIZE; j++)
    {
        fprintf(ff,"%f ",a[i][j]);
    }

    fprintf(ff,"\n");
}

fclose(ff);
```

Рис. 1: Параллельная версия программы (файл par_1.c)

На рис. 2 представлена зависимость времени исполнения первых двух пар циклов от количества исполнителей. Заметно явное ускорение при увеличении количества исполнителей до 4, а далее происходит насыщение и начинаются осцилляции, из-за чего прироста не наблюдается. Важно отметить. что эксперименты проводились на 4-х ядерном процессоре с 8-ми потоками, что объясняет такое поведение.

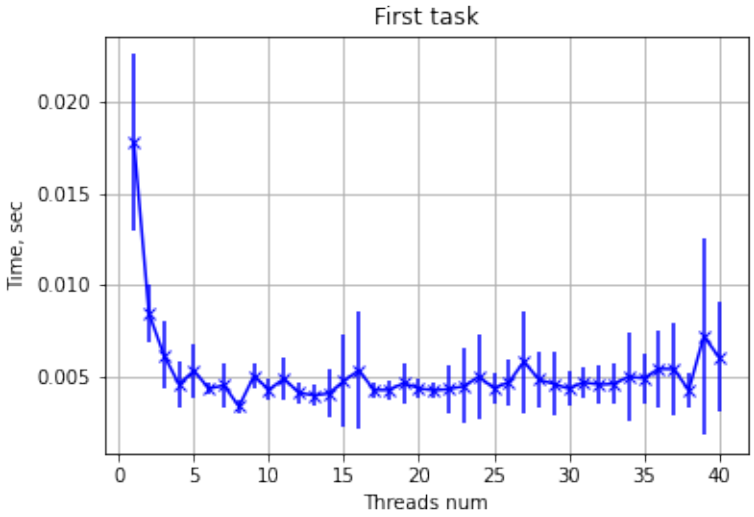

Рис. 2: Результаты для задачи 1

2 Задача 2 (1д)

Описание циклов представлены в виде комментариев в программе (см. рис. 3). Первая пара циклов та же самая, за исключением того, что теперь там мы производим также резервное копирование массива a в массив b, что пригодится нам в дальнейшем. Третья пара циклов без изменений, а вот вторая пара циклов теперь уже имеет loop independent dependencies. Вектор расстояний (-1,6), вектор направлений (>,<), поэтому имеем антизависимость: данные сначала используются, а потом определяются. Внутренний цикл зависимостей не содержит и по нему можно параллелить без ограничений. А вот по внешнему циклу распараллеливание возможно при условии предварительного резервирования данных. Отметим, что в данном случае менять циклы местами нельзя: это изменит граф алгоритма.

На рис. 4 представлена зависимость времени исполнения первых двух

пар циклов от количества исполнителей. В данном случае также заметно явное ускорение при увеличении количества исполнителей до 4, а далее также происходит насыщение и начинаются осцилляции, из-за чего прироста не наблюдается, что объясняется количество ядер процессора.

```
// there are no dependencies here therefore we are able to parallel both loops
#pragma omp for schedule(static) collapse(1)
for (i=0; i<ISIZE; i++)
{
    for (j=0; j<JSIZE; j++)
    {
        a[i][j] = 10*i +j;
        b[i][j] = 10*i +j;
    }
}

// there are loop independent dependencies here
// distance vector (-1,6), direction vector (>,<) - anti dependency
// we are able to parallel inner loop without constraints
// it is also possible to parallel outer loop with preliminary data backup (a->b)
#pragma omp for schedule(static) collapse(1)
for (i=1; i<ISIZE-1; i++)
{
    for (j = 6; j < JSIZE-1; j++)
    {
        a[i][j] = sin(0.00001*b[i+1][j-6]);
    }
}
```

Рис. 3: Параллельная версия программы (файл par_2.c)

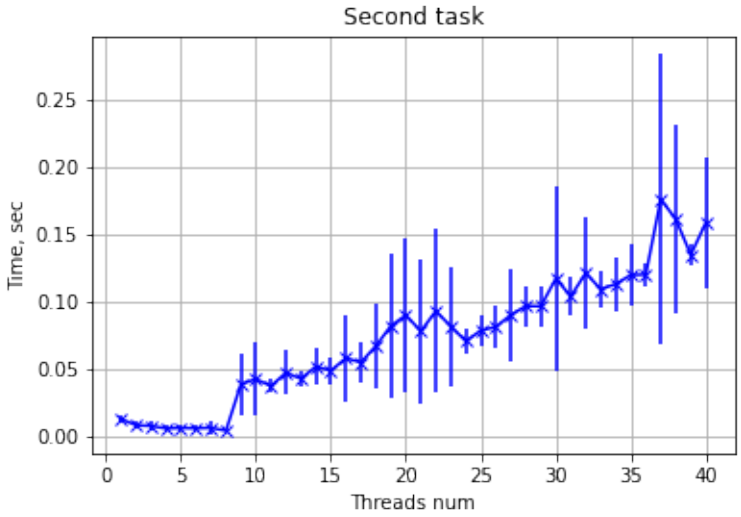

Рис. 4: Результаты для задачи 2

3 Задача 3 (2д)

Описание циклов представлены в виде комментариев в программе (см. рис. 5). Первая и третья пара циклов как и в задании 1).

Вторая пара циклов теперь имеет loop independent dependencies: вектор расстояний (8,-3), вектор направлений (<,>), поэтому имеем истинную зависимость: данные сначала определяются, а потом используются. При это внутренний цикл не содержит зависимостей и может быть распараллелен без ограничений, однако требуется барьерная синхронизация между различными итерациями внешнего цикла. Распараллелить по внешнему циклу или по двум направлениям не представляется возможным. Отметим, что в данном случае менять циклы местами также нельзя.

```
// there are loop independent dependencies here
// distance vector (8,-3), direction vector (<,>) - true dependency
// we are able to parallel inner loop without constraints, but we have to do
// barrier syncronization between outer loop iterations:w
for (i=8; i<ISIZE; i++)
{
    #pragma omp for schedule(static)
    for (j = 0; j < JSIZE-3; j++)
    {
        a[i][j] = sin(0.00001*a[i-8][j+3]);
    }
    #pragma omp barrier
}
```

Рис. 5: Параллельная версия программы (файл par_3.c)

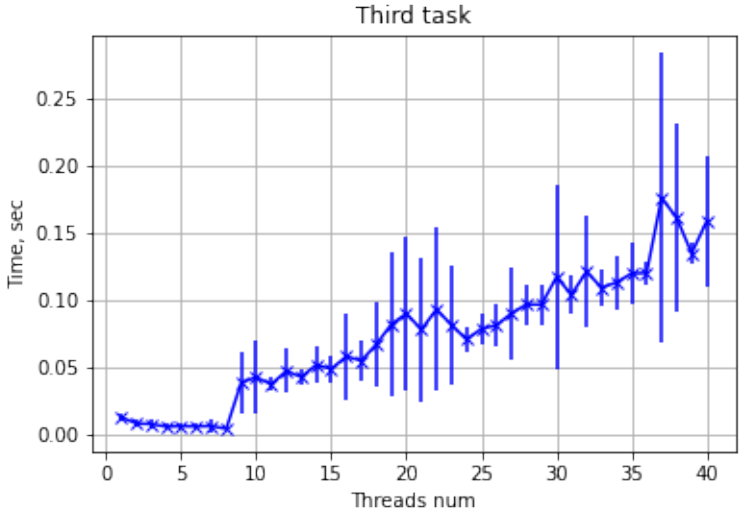

Рис. 6: Результаты для задачи 3

На рис. 4 представлена зависимость времени исполнения первых двух циклов от количества исполнителей.

В данном случае также заметно небольшое ускорение при увеличении количества исполнителей до 4. Ускорение незначительное, поскольку необходимо использовать большой количество барьеров. При дальнейшем увеличении числа исполнителей происходит резкое увеличение во времени исполнения программы, поскольку при большом количестве потоков частые переключения контекста и барьерные синхронизации сильно портят дело.