Санкт-Петербургский
политехнический университет Петра Великого

Лабораторная работа №1:

*«Распараллеливание вложенных циклов
на примере вычислительных операций с трехмерной
декартовой расчетной сеткой»*

Группа: 3640301/00201
Студент: Машаев Н.К.
Преподаватель: Абрамов А.Г.

Санкт-Петербург 2020

Содержание

1 О работе

1.1 Цель работы

Целью работы является изучение возможностей технологии OpenMP применительно к распараллеливанию вложенных циклов, в которых производятся независимые операции с многомерными массивами

1.2 Задание к работе

Разработать распараллеленную средствами OpenMP программу, в которой производится сдвиг расчетной области и покрывающей ее сетки в каждом из трех координатных направлений с независимой для каждого направления величиной сдвига.

1.3 Ход выполнения работы

1. Разработать и отладить последовательную версию программы;

2. Исследовать влияние на время работы последовательной программы порядка вложенности циклов по индексам массива, а также опций оптимизации компилятора;

3. Выполнить параллелизацию программы двумя способами: с использованием OpenMP-директивы распараллеливания циклов (omp for) и с использованием независимых параллельных секций (omp sections), в каждой из которых выполняются вычислительные операции с сеткой только в одном из координатных направлений;

4. Для способа распараллеливания с использованием директивы параллелизации циклов (omp for) и наилучшего по времени исполнения программы порядка вложенности циклов выполнить исследование влияния на эффективность распараллеливания месторасположения директивы параллелизации, располагая ее перед каждым из трех витков цикла и измеряя время работы программы (исследования проводить для фиксированного числа нитей, например для четырех);

5. Для способа распараллеливания с использованием директивы параллелизации циклов (omp for) и наилучшего по времени исполнения программы порядка вложенности циклов выполнить исследование влияния на эффективность распараллеливания способа распределения итераций цикла между нитями (STATIC, DYNAMIC, GUIDED), а также размера блока распределяемых итераций для статического (STATIC) способа (исследования проводить для фиксированного числа нитей, например для четырех; способ распределения итераций следует менять с помощью переменной окружения OMP_SCHEDULE, к директиве omp for в коде программы необходимо при этом добавить атрибут schedule(runtime)).

6. Для способа распараллеливания с использованием директивы параллелизации циклов (omp for) изучить логику работы и эффективность применения атрибута collapse(N), задавая различные значения параметра N (1, 2, 3) и меняя при запуске значение размера блока при статическом способе распределения итераций с помощью переменной окружения OMP_SCHEDULE (исследования проводить для фиксированного числа нитей, например для четырех, вычислительные эксперименты провести при включенной и отключенной опции оптимизации компилятора -O0, -O2)

7. Провести исследование эффективности распараллеливания программы для обоих реализованных вариантов (задавая при запуске программы разное число нитей, в диапазоне от 1 до 8), а также сопоставительное сравнение вариантов (для варианта с использованием директивы автоматического распараллеливания циклов задействовать наиболее эффективный код по результатам проведенных численных испытаний). Результаты представить в графическом виде, в форме зависимости ускорения программы от числа нитей.

2 Исследование возможностей OpenMP

Программы написаны на языке программирования C++. Вычислительные операции производились на сетке 100x100x100 (файл cube.msh). В качестве величины сдвига по осям X (x_shift), Y (y_shift) и Z (z_shift) были выбраны величины 1.0, 2.0 и 3.0 соответственно. Для считывания, хранения и записи сетки реализован модуль grid_reader.h. Время выполнения вычислений измерялось при помощи стандартного модуля <chrono>.

2.1 Отладка последовательной версии программы

Последовательная версия программы для сдвига расчетной области и покрывающей сетки в каждом из трех координатных направлений представлен ниже:

Листинг 2.1: *Последовательная версия программы* ***consistent_ijk***

```
for (size_t i = 0; i < x_size; ++i)
{
  for (size_t j = 0; j < y_size; ++j)
  {
    for (size_t k = 0; k < z_size; ++k)
    {
      xc[i][j][k] += x_shift;
      yc[i][j][k] += y_shift;
      zc[i][j][k] += z_shift;
    }
  }
}
```

Вычислительная сложность приведенного выше алгоритма составляет $O(n^3)$. Интересным представляется изучение влияния порядка вложенности цикла на

время выполнения программы. Для исследования данного вопроса поменяем местами цикл по i и k. Таким образом, получим:

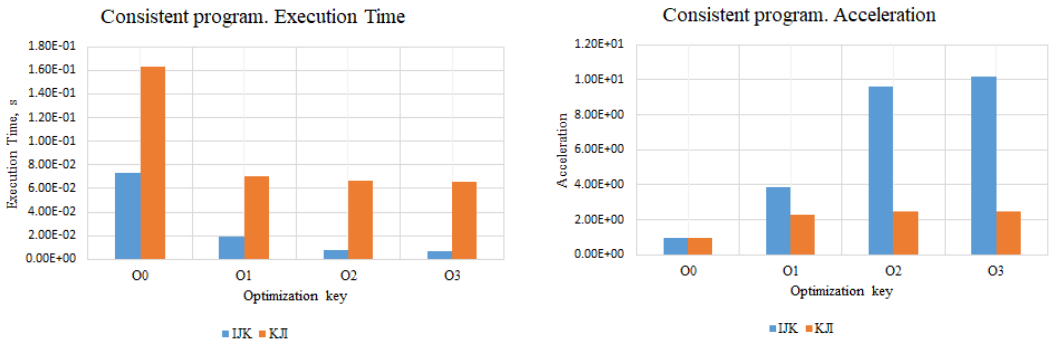
Листинг 2.2: *Последовательная версия программы* ***consistent_kji*** *с обратным порядком вложенности циклов*

```
for (size_t k = 0; k < z_size; ++k)
{
  for (size_t j = 0; j < y_size; ++j)
  {
    for (size_t i = 0; i < x_size; ++i)
    {
      xc[i][j][k] += x_shift;
      yc[i][j][k] += y_shift;
      zc[i][j][k] += z_shift;
    }
  }
}
```

При компиляции использовались ключи -O0, -O1, -O2, -O3. Для каждого из случаев ускорение вычислялось как отношение времени выполнения программы при отсутствии оптимизации (-O0) к времени выполнения программы при рассматриваемом ключе оптимизации. Время, затрачиваемое на сдвиг сетки, а также ускорение в зависимости от ключа оптимизации приведено в таблице 1 и на рис.1.

	Время выполнения,с		**Ускорение**	
Оптимизация	**IJK**	**KJI**	**IJK**	**KJI**
O0	0.0735	0.1634	1.0000	1.0000
O1	0.0190	0.0702	3.8721	2.3288
O2	0.0076	0.0662	9.6223	2.4694
O3	0.0072	0.0657	10.2106	2.4891

Таблица 1: Исследование порядка вложенности циклов



(a) Зависимость времени выполнения от ключа оптимизации

(b) Зависимость ускорения от ключа оптимизации

Рис. 1: Влияние порядка вложенности циклов

Из рис.1 видно, что вложенных порядок вложения крайне важен, а значит, необходимо помнить как хранятся массивы в памяти. В C/C++ массивы хранятся

построчно, поэтому и порядок IJK вложения циклов является наиболее оптимальным.

Согласно рис.1 использовании оптимизации значительным образом ускоряет выполнение программы. В данном случае наихудший результат показывают вычисления без применения оптимизации, когда как применение ключа -O2 и -O3 в случае порядка вложенности циклов IJK ускоряют вычисления примерно в 10 раз. Для программы с порядком вложенности циклов KJI применение оптимизации показывают более скромные результаты: ускорение вычислений для каждой из оптимизации примерно вдвое. В данном случае такой результат ожидаем, поскольку здесь происходит неэффективное обращение к памяти.

2.2 Параллелизация циклов при помощи директивы omp for

Директива omp for является специальной директивой параллелизации циклов: она указывает компилятору на то, что итерации следующего непосредственно за ним цикла должны быть выполнены в параллельном режиме группой работающих нитей.

Программа для вычисления сдвига расчетной сетки с применением директивы omp for приведена ниже:

Листинг 2.3: *Применение директивы omp for (pure_for_1)*

```
#pragma omp parallel \
  shared(x_shift, y_shift, z_shift, \
         x_size, y_size, z_size, \
         xc, yc, zc)
{
  #pragma omp for schedule(runtime)
  for (int i = 0; i < x_size; ++i)
  {
    for (int j = 0; j < y_size; ++j)
    {
      for (int k = 0; k < z_size; ++k)
      {
        xc[i][j][k] += x_shift;
        yc[i][j][k] += y_shift;
        zc[i][j][k] += z_shift;
      }
    }
  }
}
```

Поскольку рассматриваемая директива применяется лишь к стоящему за ней циклу for интересным представляется исследование месторасположения директивы параллелизации.

Для изучения данного вопроса будем располагать директиву omp for перед каждым из трех витков цикла и измерять время работы программы.

Листинг 2.4: *Применение директивы omp for (pure_for_2)*

```
#pragma omp parallel \
  shared(x_shift, y_shift, z_shift, \
       x_size, y_size, z_size, \
       xc, yc, zc)
{
  for (int i = 0; i < x_size; ++i)
  {
    #pragma omp for schedule(runtime)
    for (int j = 0; j < y_size; ++j)
    {
      for (int k = 0; k < z_size; ++k)
      {
        xc[i][j][k] += x_shift;
        yc[i][j][k] += y_shift;
        zc[i][j][k] += z_shift;
      }
    }
  }
}
```

Листинг 2.5: *Применение директивы omp for (pure_for_3)*

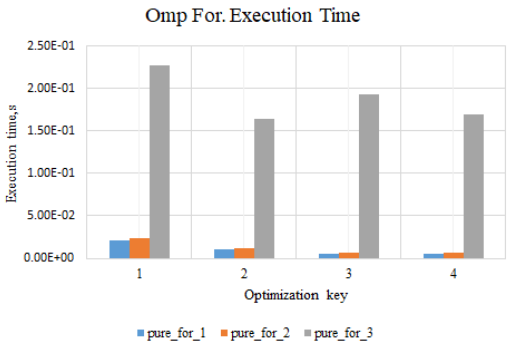
```
#pragma omp parallel \
shared(x_shift, y_shift, z_shift, \
     x_size, y_size, z_size, \
     xc, yc, zc)
{
  for (int i = 0; i < x_size; ++i)
  {
    for (int j = 0; j < y_size; ++j)
    {
      #pragma omp for schedule(runtime)
      for (int k = 0; k < z_size; ++k)
      {
        xc[i][j][k] += x_shift;
        yc[i][j][k] += y_shift;
        zc[i][j][k] += z_shift;
      }
    }
  }
}
```

Исследование эффективности выполнения программы в зависимости от месторасположения директивы omp for для числа нитей равного четырем приведено ниже(в качестве OMP_SCHEDULE использовалось STATIC,1). Здесь loop1(pure_for_1), loop2(pure_for_2), loop3(pure_for_3) - программы с месторасположением директивы omp for перед первым, вторым и третьим циклами соответственно. Стоит заметить, что ускорение для каждого из случаев вычислялось относительно случая отсутствия ключа оптимизации: ускорение вычислялось как отношение времени выполнения программы при отсутствии оптимизации (-O0) к времени выполнения

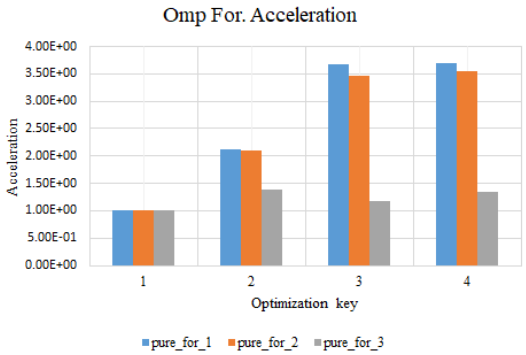
программы при рассматриваемом ключе оптимизации.

	Время выполнения,с			Ускорение		
Оптимизация	**loop1**	**loop2**	**loop3**	**loop1**	**loop2**	**loop3**
O0	0.0214	0.0240	0.2274	1.0000	1.0000	1.0000
O1	0.0100	0.0114	0.1640	2.1293	2.1083	1.3863
O2	0.0058	0.0069	0.1934	3.6828	3.4611	1.1759
O3	0.0058	0.0068	0.1693	3.6983	3.5448	1.3428

Таблица 2: Исследование месторасположение директивы omp for



(a) Зависимость времени выполнения от ключа оптимизации



(b) Зависимость ускорения от ключа оптимизации

Рис. 2: Влияние месторасположения директивы for

Из рис.2 видно, наиболее эффективный результат по времени вычисления и ускорения получается если расположить директиву omp for перед внешним циклом. Это объясняется тем, что при таком расположении достигается более равномерная загрузка нитей. При этом вычисления ускоряются примерно в 3.5 раз при использовании оптимизации с ключом -O2 или -O3.

2.3 Исследование применения OMP_SCHEDULE

Атрибут omp for директивы schedule позволяет управлять распределением итераций цикла между нитями. Он имеет следующие значения(с CHUNK_SIZE параметром, который устанавливает размер блока итераций, назначаемого нитям, и по умолчанию равным единице):

1. STATIC(статический, блочно-циклический) - распределение блоками по размерами CHUNK_SIZE следующих итераций в порядке увеличения номеров нитей;

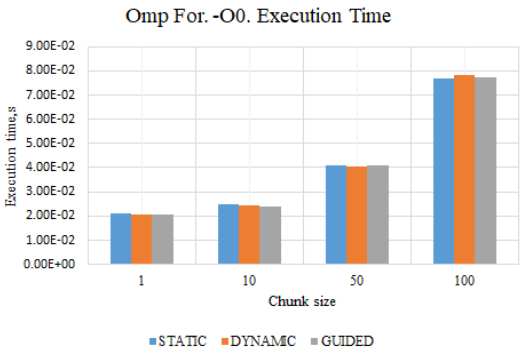
2. DYNAMIC(динамический) - распределение блоками по CHUNK_SIZE следующих итераций, при этом освободившаяся нить принимает к исполнению первый по порядку из еще не обработанных блоков;

3. GUIDED(управляемый) - динамическое распределение блоками следующих подряд итераций с постепенным уменьшением размера блока вплоть до величины CHUNK_SIZE

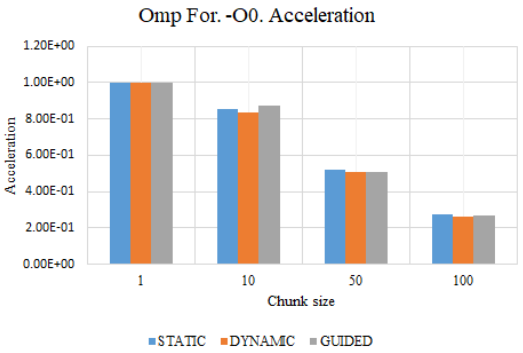
Данные при изучении применения omp schedule получены при вычислении на четырех нитей без оптимизации(-O0).

Chunk size	Время выполнения,с			Ускорение		
	Static	Dynamic	Guided	Static	Dynamic	Guided
1	0.0212	0.0205	0.0207	1.0000	1.0000	1.0000
10	0.0248	0.0245	0.0237	0.8569	0.8379	0.8716
50	0.0409	0.0406	0.0407	0.5191	0.5054	0.5068
100	0.0767	0.0781	0.0774	0.2768	0.2627	0.2667

Таблица 3: Исследование применения различных атрибутов schedule



(a) Зависимость времени выполнения от размера блока



(b) Зависимость ускорения от размера блока

Рис. 3: Влияние применения атрибута schedule

Из рис.3 видно, что использование различных атрибутов schedule дает практически одинаковый результат: с увеличение размера блока скорость выполнения расчетов уменьшается. Это может быть объяснено тем, что при выбранных размерах блока нарушается равномерность загруженности нитей, а значит хуже используется кеш-память, что и ведет к ухудшению результатов.

2.4 Применение атрибута collapse

Атрибут collapse(n) является одним из атрибутов директивы omp for. Он определяет особенности распараллеливания последовательных вложенных целов с прямоугольным индексным пространством, для которых не используются директивы вложенного параллелизма. n - параметр атрибута - целое положительное число, указывающее на то, сколько петель цикла будут образовывать общее пространство итераций, совместно распределяемое между нитями.

Таким образом, применение collapse(1) равносильно исходной конструкции из трех вложенных циклов, в то время как использование collapse(3) привлечет к работе с общим индексным пространством как с одним циклом.

Листинг 2.6: *Применение атрибута collapse(collapse_3)*

```
#pragma omp parallel \
  shared(x_shift, y_shift, z_shift, \
         x_size, y_size, z_size, \
       xc, yc, zc)
{
  #pragma omp for schedule(runtime) \
           collapse(3)
  for (int i = 0; i < x_size; ++i)
  {
    for (int j = 0; j < y_size; ++j)
    {
      for (int k = 0; k < z_size; ++k)
      {
        xc[i][j][k] += x_shift;
        yc[i][j][k] += y_shift;
        zc[i][j][k] += z_shift;
      }
    }
  }
}
```

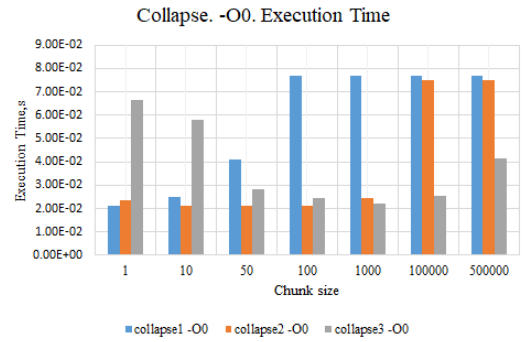
Применение директивы collapse будем исследовать для значений n от одного до трех. Число нитей задается равным четырем, атрибут OMP_SCHEDULE=STATIC.

Chunk size	**Время выполнения,с**			**Ускорение**		
	collapse_1	**collapse_2**	**collapse_3**	**collapse_1**	**collapse_2**	**collapse_3**
1	0.0212	0.0236	0.0663	1.0000	1.0000	1.0000
10	0.0248	0.0214	0.0580	0.8570	1.1034	1.1423
50	0.0409	0.0210	0.0281	0.5195	1.1199	2.3572
100	0.0767	0.0210	0.0243	0.2769	1.1227	2.7227
1000	0.0768	0.0245	0.0220	0.2768	0.9628	3.0129
100000	0.0768	0.0749	0.0252	0.2768	0.3145	2.6284
500000	0.0768	0.0749	0.0416	0.2768	0.3145	1.5936

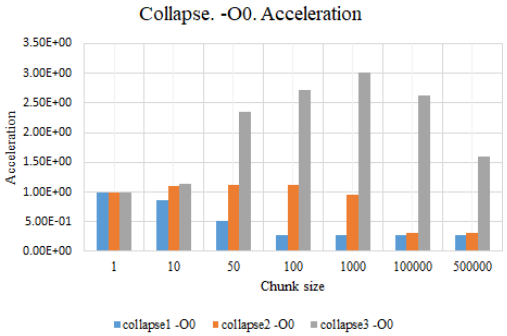
Таблица 4: Исследование применения атрибута collapse в отсутствие оптимизации

Как видно из таблицы 4 при увеличении размера блока CHUNK_SIZE атрибута schedule значительным образом увеличивается скорость выполнения программы с использованием collapse(3). Это объясняется тем, что с использованием collapse(3) мы назначаем размер блока для каждой нити для общего пространства итераций размером 1000000, в то время как, к примеру, при collapse(1) лишь для 100. Естественно, при достижении размера блока 100 в случае collapse(1) весь цикл выполняет лишь одна нить, в результате чего эффективность падает. Для collapse(3) эффективность растет вплоть до размера блока 1000. При размере блоков 100000 и 500000 равномерность загруженности нитей нарушается, из-за чего производительность падает.

Более наглядно это видно на рис.4, построенном по данным таблицы 4.



(a) Зависимость времени выполнения от размера блока



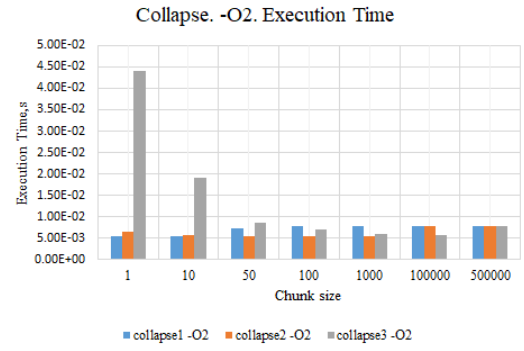
(b) Зависимость ускорения от размера блока

Рис. 4: Применение collapse без оптимизации

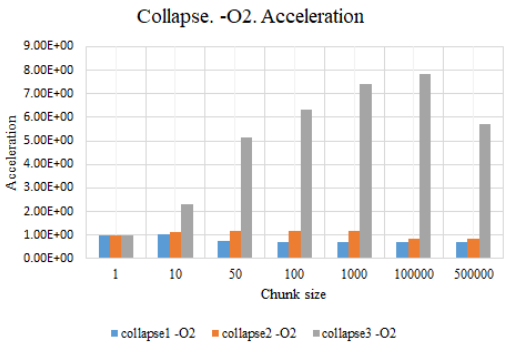
При выполнении расчетов с ключом оптимизации -O2 результаты по времени выполнения с ростом размера блоков примерно одинаковые, в отличие от случая -O0.

Chunk size	**Время выполнения,с**			**Ускорение**		
	collapse_1	**collapse_2**	**collapse_3**	**collapse_1**	**collapse_2**	**collapse_3**
1	0.0056	0.0065	0.0440	1.0000	1.0000	1.0000
10	0.0054	0.0057	0.0192	1.0236	1.1285	2.2912
50	0.0072	0.0056	0.0086	0.7726	1.1576	5.1410
100	0.0078	0.0056	0.0069	0.7154	1.1632	6.3438
1000	0.0079	0.0054	0.0059	0.7079	1.1879	7.4054
100000	0.0078	0.0078	0.0056	0.7164	0.8285	7.8434
500000	0.0078	0.0078	0.0077	0.7144	0.8272	5.7144

Таблица 5: Исследование применения атрибута collapse -O2



(a) Зависимость времени выполнения от размера блока



(b) Зависимость ускорения от размера блока

Рис. 5: Применение collapse -O2

2.5 Параллелизация циклов с использованием omp sections

Директива sections применяется для объявления фрагментов кода внутри параллельной области программы, выполнение которых (в виде независиых подзадач) будет распределено между нитями. Данная директива представляет собой неитеративную параллельную конструкцию, определяющую внутри параллельной области набор независимо и однократно исполняемых частей кода (параллельных секций).

Листинг 2.7: *Параллелизация с применением omp sections*

```
#pragma omp parallel \
  shared(x_shift, y_shift, z_shift, \
      xc, yc, zc)
{
  #pragma omp sections
  {
    #pragma omp section
    {
      axis_i_shift(xc, x_shift);
    }
    #pragma omp section
    {
      axis_i_shift(yc, y_shift);
    }
    #pragma omp section
    {
      axis_i_shift(zc, z_shift);
    }
  }
}
```

Листинг 2.8: *Реализация axis_i_shift*

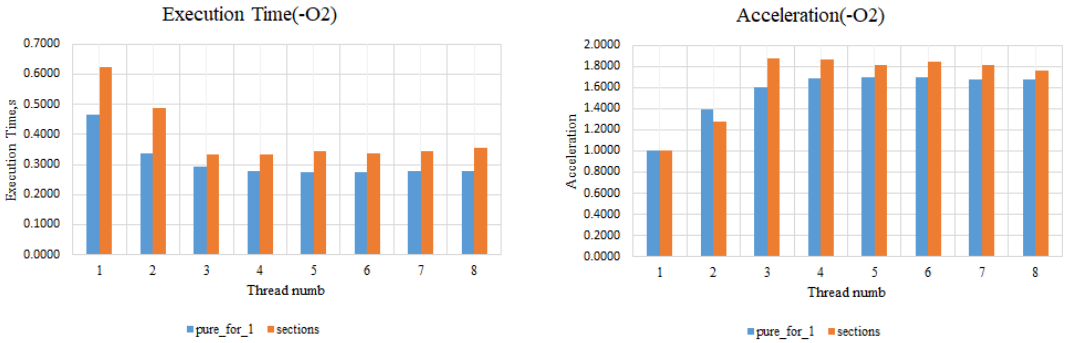
```
void axis_i_shift(GR::msh3d & ic,
          double ishift)
{
  int x_size = static_cast<int>(ic.size());
  int y_size = static_cast<int>(ic[0].size());
  int z_size = static_cast<int>(ic[0][0].size());
  for (int i = 0; i < x_size; ++i)
  {
    for (int j = 0; j < y_size; ++j)
    {
      for (int k = 0; k < z_size; ++k)
      {
        ic[i][j][k] += ishift;
      }
    }
  }
}
```

Исследуем эффективность параллелизации при помощи директивы omp sections и сравним ее с применением директивы omp for (pure_for_1) для внешнего цикла при разном количестве нитей. В данном разделе для исследований использовалась сетка размером 400x400x400.

К качестве ключа оптимизации будем использовать -O2. При этом атрибут schedule равен STATIC,1

Число	**Время выполнения,с**		**Ускорение**	
нитей	**pure_for_1**	**sections**	**pure_for_1**	**sections**
1	0.4671	0.6221	1.0000	1.0000
2	0.3360	0.4869	1.3901	1.2775
3	0.2918	0.3325	1.6007	1.8707
4	0.2763	0.3339	1.6907	1.8632
5	0.2746	0.3440	1.7011	1.8083
6	0.2757	0.3366	1.6946	1.8478
7	0.2789	0.3431	1.6749	1.8133
8	0.2788	0.3544	1.6752	1.7553

Таблица 6: Сравнение omp for и omp sections



(a) Зависимость времени выполнения от числа нитей

(b) Зависимость ускорения от числа нитей

Рис. 6: Сравнение omp for и omp sections

Из рис.6 видно, что программа с применением директивы omp_for для внешнего цикла выполняется быстрее. Причем, в виду того, что в случае применения omp sections используются лишь три секции, то при достижении числа нитей трех, дальнейшее увеличение числа нитей не сказывается на производительности. Для omp for наблюдается подобный результат, но в данном случае этот эффект можно объяснить тем, что нагрузка между нитями начинает распределяться неравномерно, из-за чего дальнейшее увеличение числа нитей влечет слабый прирост производительности.

3 Вывод

Проведено исследование особенностей различных директив параллельного программирования для работы с циклами. Были получены следующие результаты:

- При работе с вложенными циклами необходимо помнить как хранятся массивы в памяти. При неправильном порядке вложенности циклов снижается эффективность использования кеш памяти, что значительным образом повышает время выполнения программы;
- При параллелизации работы с вложенными циклами директивой for наилучший результат по скорости выполнения программы достигается при применении данной директивы к внешнему циклу;
- При работе с директивой omp for важно определить то, как нити будут разделять между собой итерации. В случае рассматриваемой задачи сдвига существенным оказывается выбор размера блока итераций;
- При использовании директивы collapse(3) наиболее оптимальным размеров блока является 1000 из рассмотренных в данной работе. В отсутствие данного атрибута(или же collapse(1)) увеличение размера блока влечет к ухудшению производительности;
- Применение оптимизации способно существенным образом ускорить выполнение программы;
- В данном случае применение директивы omp sections является неэффективным. При числе нитей равным трем каждая нить выполняет свою секцию и дальнейшее увеличение числа нитей не повлечет за собой рост производительности.

4 Приложение

https://github.com/nkmashaev/OpenMP_Lab1 - код программы