МГУ им. М.В.Ломоносова

Шувалов Антон Юрьевич

4 курс, группа 425, кафедра вычислительной механики

Параллельная реализация алгоритма численного решения уравнений Навье-Стокса.

Курсовая работа

Научный руководитель:

доктор физ.-мат. наук,

профессор Луцкий А.Е.

Механико-математический факультет МГУ, 2019

Оглавление

Оглавление

Введение 3
Описание последовательного кода 4
Реализация параллельного алгоритма 7
Примеры расчетов 17
Заключение 22
Список использованных источников 23

Введение

Расчетная сетка - совокупность точек (узлов сетки), заданных в области определения некоторой функции. Расчетные сетки используются при численном решении дифференциальных и интегральных уравнений. Качество построения расчетной сетки в значительной степени определяет успех численного решения уравнения. Точность решения может быть повышена путем уменьшения размеров элементов сетки. Однако, уменьшение размеров ячеек во всей области значительно увеличивает вычислительную сложность задачи. Большинство задач математической физики требуют огромной вычислительной мощности. Отчасти эта проблема решается, если перейти к параллельным вычислениям. Параллельная программа – это ансамбль взаимодействующих слабосвязанных последовательных процессов. Создание комплексов программ, которые могут выполнять расчеты на параллельных компьютерах является достаточно сложной и трудоемкой задачей. При переходе от последовательных программ к параллельным требуется не только добавить в последовательную программу обмены данных между процессорами, но и значительно перестроить всю структуру используемых данных. Именно в этом состояла моя задача в данной работе с конкретной программой.

Описание последовательного кода

Уравнения Навье-Стокса и Рейнольдса с двумя пространственными переменными при использовании гипотезы Буссинеска могут быть записаны в единообразной форме.

$$\frac{\partial U}{\partial t}+\frac{\partial F}{\partial x}+\frac{\partial G}{\partial y}=H$$

$$U=(\rho,\rho u,\rho v,e)^T$$

$$F=F^i+F^v, G=G^i+G^v, H=\frac{\omega}{y}(H^i+H^v)$$

$$F^i=(\rho u,\rho u^2+p,\rho u\upsilon,(e+p)u)^T,$$

$$F^v=(0,-\tau_{xx},-\tau_{xy},-u\tau_{xx}-\upsilon\tau_{xy}-q_x)^T$$

$$G^i=(\rho\upsilon,\rho u\upsilon,\rho\upsilon^2+p,(e+p)\upsilon)^T,$$

$$G^v=(0,-\tau_{xy},-\tau_{yy},-u\tau_{xy}-\upsilon\tau_{yy}-q_y)^T$$

$$H^i=(-\rho\upsilon,-\rho u\upsilon,-\rho\upsilon^2,-(e+p)\upsilon)^T,$$

$$H^v=(0,\tau_{xy},\tau_{yy}-\tau_{\varphi\varphi},u\tau_{xy}+\upsilon\tau_{yy}+q_y)^T$$

В нашем случае решается плоская задача, поэтому $\omega=0$.

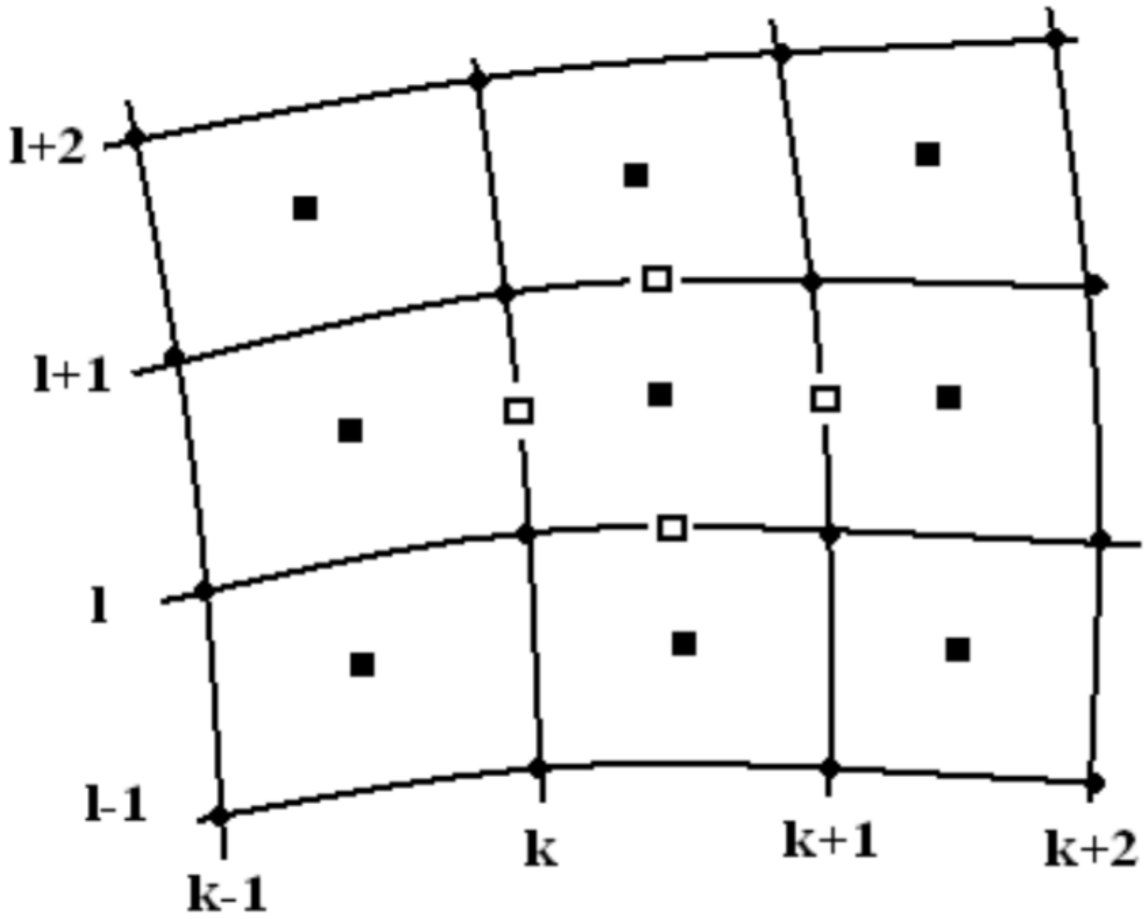


Схема расчета ячейки

Рассмотрим аппроксимацию потоков на примере ребра с номером k+1,l+1/2 (см. рис) . Пусть $U_{\mathrm{k+1/2,l+1/2}}$ - величины, отнесенные к центрам ячеек, $U_{k,l} = (U_{k-1/2,l-1/2} + U_{k+1/2,l-1/2} + U_{k+1/2,l+1/2} + U_{k-1/2,l+1/2})/4$ - величины в узлах сетки.

Производные $(U_x, U_y)_{\mathrm{k+1,l+1/2}}$, входящие в выражения потоков, вычисляются через разности $(U_{k+3/2,l+1/2} - U_{k+1/2,l+1/2})$ и $(U_{k+1,l+1} - U_{k+1,l})$.

Значения функций $U_{\mathrm{k+1,l+1/2}}$ на ребре определяются из решения задачи Римана с начальными данными

$$U^{+}_{k+1} = U_{k+1/2,l+1/2} + 1/2\Delta x U_{x,k+1/2,l+1/2} + 1/2\Delta y U_{y,k+1/2,l+1/2}$$

$$U_{k+1}^{-} = U_{k+3/2,l+1/2} - 1/2\Delta x U_{x,k+3/2,l+1/2} - 1/2\Delta y U_{y,k+3/2,l+1/2}$$

Для обеспечения монотонности разностной схемы производные в ячейках определяются в соответствии с принципом минимума модуля производных на противоположных ребрах

$$U_{x,k+1/2,l+1/2} = \min mod(U_{x,k,l+1/2}, U_{x,k+1,l+1/2})$$

$$U_{y,k+1/2,l+1/2} = \min mod(U_{y,k,l+1/2}, U_{x,k+1,l+1/2})$$

Таким образом, аппроксимация конвективных членом аналогична разностной схеме В.П.Колгана [6].

Структура последовательного кода:

1. Input:

1.1. Размер сетки

1.2. Структура сетки

1.3. Список всех ячеек с указанием вершин

1.4. Начальные значения газодинамических величин в углах сетки

1.5. Остальные параметры, характеризующие данную задачу (максимальное число шагов, шаг по времени, начальное время)

2. Step (в цикле по времени) :

2.1. Проводятся расчеты газодинамических величин с учетом типа потока и геометрии изучаемого объекта для следующего момента времени

2.2. При необходимости производится вывод промежуточных параметров

2.3. Осуществляется адаптация расчетной сетки

3. Output:

3.1. Результаты вычислений газодинамических величин

Реализация параллельного алгоритма

MPI является библиотекой функций обмена данными между процессами, реализованная для языков C, C++, Fortran и Java. Головной организацией проекта MPI является Аргоннская национальная лаборатория США. После появления первой версии стандарта MPI в мае 1994 года он получил широкое распространение. В настоящее время MPI является наиболее распространенным стандартом интерфейса обмена данными в параллельном программировании, существуют его реализации для большого числа компьютерных платформ. Библиотека используется для разработки программ для кластеров и суперЭВМ. Благодаря простоте технической реализации кластеров на базе локальных сетей сотни университетов используют MPI для учебных и научных целей.

Стандарт MPI-1 использует статическое размещение процессов и данных по процессорам, а стандарт MPI-2 предназначен для динамического распределения работ.

Базовым механизмом связи между MPI процессами является передача и прием сообщений. Сообщение несет в себе передаваемые данные и информацию, позволяющую принимающей стороне осуществлять их выборочный прием. Например:

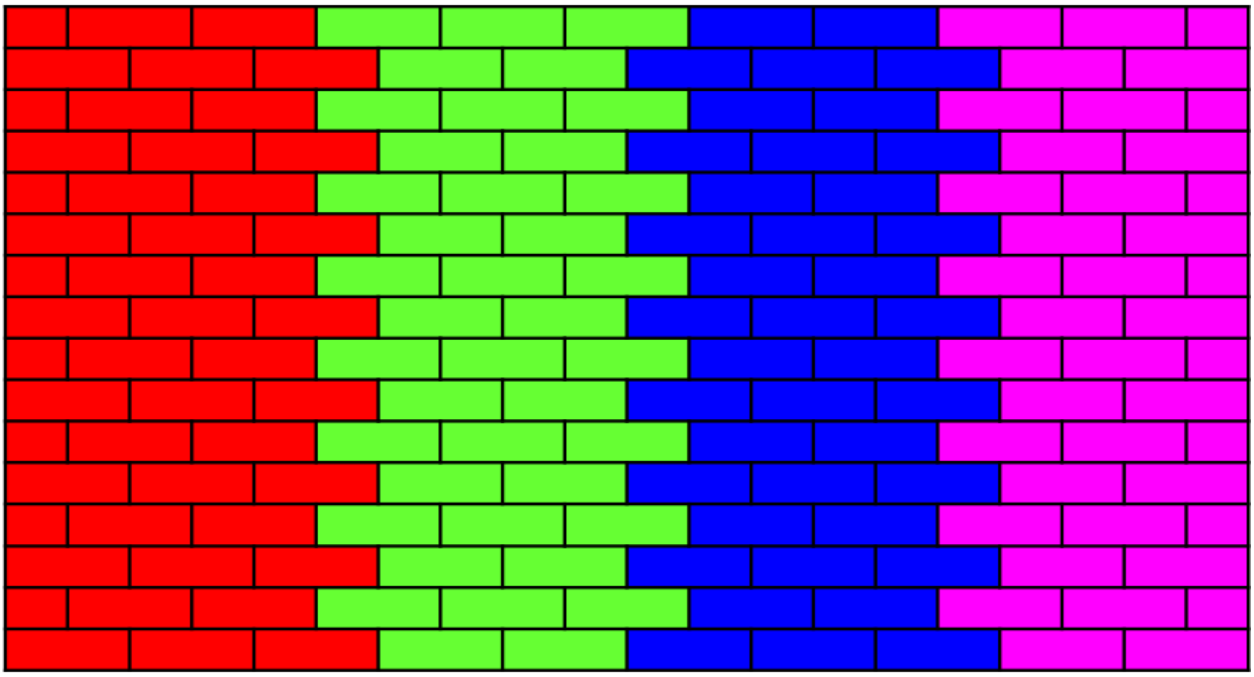
- отправитель — ранг (номер в группе) отправителя сообщения;
- получатель — ранг получателя;
- признак — может использоваться для разделения различных видов сообщений;
- коммуникатор — код группы процессов.

Для распараллеливания данного мне кода я использовал метод геометрического параллелизма. Он позволяет получать высокие ускорения, если все процессоры работают синхронно, с одинаковой скоростью. Метод

используется при решении задач математической физики, при решении систем уравнений и т.д. Он заключается в том, что исходную задачу мы можем разбить на группу областей, независимых друг от друга на каждом расчетном шаге и пересекающихся только по границе разбиения. Т.е. мы рассчитываем n-ый временной слой в каждой области, затем согласуем границы и переходим к расчету следующего слоя.

Однако, при таком подходе, когда мы делим расчетную область на непересекающиеся подобласти, у нас возникают проблемы с пересчетом значений на границах между данными областями, поэтому предлагается следующий достаточно логичный шаг, - делить исходную область на взаимно перекрывающиеся подобласти.

Появятся по две "фиктивные" точки слева для первой области и справа для последней области. Таким образом, мы получаем четыре независимых на каждом шаге по времени процесса. Для перехода к следующей итерации необходимо согласование границ, так как первая область должна передать второй ее левую границу для следующего шага по времени, в свою очередь, вторая область должна предать первой ее правую границу, и т.д.



Графическая интерпретация метода геометрического параллелизма

Пусть n - ширина стены, k - высота стены. Если τ_c - время на прием и передачу данных, то показатели времени выполнения, ускорения и эффективности метода конвейерного параллелизма определяются соотношениями:

$$T_1(kn) = \tau_c kn \qquad T_p(kn) = \tau_c \frac{kn}{p} + 4k\tau_s$$

$$S_p(kn) = p\frac{1}{1+4\frac{p}{n}\frac{\tau_s}{\tau_c}} \qquad E_p(kn) = \frac{1}{1+4\frac{p}{n}\frac{\tau_s}{\tau_c}}$$

Для распараллеливания предоставленного кода мною были реализованы следующие функции:

1. void Sendgdf(int size, int rank, int beg, int end, double *gdfSend, double *gdfRecv, double **gdf);

   Описание аргументов:

Int size – количество процессов выполняющих расчет

Int rank – номер процесса в группе

Int beg – начало области за расчет которой ответствен данный процесс

Int end – конец области за расчет которой ответствен данный процесс

double *gdfSend – массив граничных ячеек для пересылки соседним процессам

double *gdfRecv – массив принятых ячеек

double **gdf – массив газодинамических параметров

Описание алгоритма работы функции:

Данная функция осуществляет пересылку ячеек, расположенных на границе областей, принадлежащих процессам, для продолжения расчетов.

На первом шаге в одномерный массив gdfSend записываются газодинамические параметры со столбцов под номерами beg, beg+1, end-2, end-1. Далее все процессы выполняют пересылку данных соседним процессам. Принятые данные записываются в массив gdfRecv. После этого полученные переменные распределяются на места beg-2, beg-1, end, end+1.

Ниже приведено тело функции:

```
void Sendgdf (int size, int rank,int IT1, int IT2, double *gdfSend,
double *gdfRecv, double **gdf){
      MPI_Status st;
      //gdf[0][IC(IT1,3)]=2.3;
      for (int i=0;i<2;i++){
            for (int j=0;j<JT;j++)
                  for (int k=0;k<6;k++){
                        gdfSend[(j)*6+k+6*(JT)*i]=gdf[k][IC(IT1+i,j)];
                        gdfSend[(j)*6+k+6*(JT)*(i+2)]=gdf[k][IC(IT2-(2-1-
i)-1,j)];
                        }
                  }
      if (rank!=(size-1))
```

```
            MPI_Send((gdfSend+2*6*(JT)), 2*6*(JT), MPI_DOUBLE, rank+1,
rank+1, MPI_COMM_WORLD);
      if (rank!=0)
            MPI_Recv(gdfRecv, 2*6*(JT), MPI_DOUBLE, rank-1, rank,
MPI_COMM_WORLD, &st);
      if (rank!=0)
            MPI_Send(gdfSend, 2*6*(JT), MPI_DOUBLE, rank-1, rank-1,
MPI_COMM_WORLD);
      if (rank!=(size-1))
            MPI_Recv((gdfRecv+2*6*(JT)), 2*6*(JT), MPI_DOUBLE, rank+1,
rank, MPI_COMM_WORLD, &st);
      if (rank!=0 && rank != (size-1))
            for (int i=0;i<2;i++){
                  for (int j=0;j<JT;j++)
                        for (int k=0;k<6;k++){
                              gdf[k][IC(IT1-(1-i)-
1,j)]=gdfRecv[(j)*6+6*(JT)*i+k];

      gdf[k][IC(IT2+i,j)]=gdfRecv[(j)*6+k+6*(JT)*(i+2)];
                              }
                        }
      if (rank==0)
            for (int i=0;i<2;i++){
                  for (int j=0;j<JT;j++)
                        for (int k=0;k<6;k++){

      gdf[k][IC(IT2+i,j)]=gdfRecv[(j)*6+k+6*(JT)*(i+2)];
                              }
                        }

      if (rank==(size-1))
            for (int i=0;i<2;i++){
                  for (int j=0;j<JT;j++)
                        for (int k=0;k<6;k++){
                              gdf[k][IC(IT1-(2-1-i)-
1,j)]=gdfRecv[(j)*6+6*(JT)*i+k];
                              }
                        }
}
```

2. void send_to_print();

   Описание алгоритма работы функции:

   Данная функция осуществляет пересылку информации о газодинамических параметрах процессу с номером 0 для записи результатов вычисления в файл.

На первом этапе работы функции создаются и заполняются все переменные, необходимые для пересылки. После в одномерный массив gdf_send записываются газодинамические параметры из всех ячеек области, за расчеты параметров на которых ответсвеннен каждый процесс. Далее все процессы выполняют пересылку данных процессу с номером 0. Принятые данные записываются в массив gdf_recv, после чего из gdf_recv заполняется массив газодинамических параметров всей области на процессе 0.

Ниже приведено тело функции:

```
void send_to_print(){
int rank, size, IT1, IT2;
MPI_Status st;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
IT1=(rank==0) ? 2 :  2+rank*(IT-4)/size;
IT2=(rank!=(size-1)) ? 2+(rank+1)*(IT-4)/size : IT-2;
double *gdf_send;
double *gdf_recv;
gdf_recv=new double [(IT-2-(2+(size-1)*(IT-4)/size))*(JT-4)*6];
gdf_send=new double [(IT2-IT1)*(JT-4)*6];

if (rank!=0)
      for (int i=IT1;i<IT2;i++){
            for (int j=2;j<JT-2;j++)
                  for (int k=0;k<6;k++)
                        {
                        gdf_send[(j-2)*6+k+6*(JT-4)*(i-
IT1)]=gdf[k][IC(i,j)];
                        }
                  }
if (rank!=0)
{
      MPI_Send(gdf_send, 6*(JT-4)*(IT2-IT1), MPI_DOUBLE, 0, rank,
MPI_COMM_WORLD);
}
if(rank==0)
      for (int count=1;count<size;count++)
      {
            IT1=2+count*(IT-4)/size;
            IT2=(count!=(size-1)) ? 2+(count+1)*(IT-4)/size : IT-
2;
            MPI_Recv(gdf_recv, 6*(JT-4)*(IT2-IT1), MPI_DOUBLE,
count, count, MPI_COMM_WORLD, &st);
            for (int i=IT1;i<IT2;i++){
                  for (int j=2;j<JT-2;j++)
```

```
                              for (int k=0;k<6;k++)
                                    {
                                    gdf[k][IC(i,j)]=gdf_recv[(j-
2)*6+k+6*(JT-4)*(i-IT1)];
                                    }
                              }
            }
}
```

3. void test_send_gdf (int n_step);

   Описание алгоритма работы функции:

   Данная функция написана для проверки работоспособности передачи данных между процессами.

   Функция имеет в качестве входного параметра переменную, определяющую количество шагов по времени, расчитанных программой.

   В начале функции определяются все переменные, необходимые для обмена данными между процессами. Далее переданные и отправленные данные записываются в файлы для последующей сверки.

   Ниже приведено тело функции:

```
void test_send_gdf (int n_step){
      if (n_step==20)
      {   printf("%d\n",JT);
            FILE *testS1, *testR1,*testS2, *testR2;
            testS1=fopen("test/test_send1.txt","w");
            testR1=fopen("test/test_recv1.txt","w");
            testS2=fopen("test/test_send2.txt","w");
            testR2=fopen("test/test_recv2.txt","w");
            int rank, size, IT1, IT2;
            MPI_Comm_rank(MPI_COMM_WORLD, &rank);
            MPI_Comm_size(MPI_COMM_WORLD, &size);
            IT1=(rank==0) ? 2 :  2+rank*(IT-4)/size;
            IT2=(rank!=(size-1)) ? 2+(rank+1)*(IT-4)/size : IT-2;
            for (int k=0;k<6;k++)
            {
                  if ( rank==1)
                  {
                        for (int j=0;j<JT;j++)
                              fprintf(testS1,"%13lf\n",gdf[k][IC(IT2-
1,j)]);
                        fprintf(testS1, "\n\n\n\n");
```

```
                }
                if ( rank==2)
                {
                      for (int j=0;j<JT;j++)
                            fprintf(testR1,"%13lf\n",gdf[k][IC(IT1-
1,j)]);
                      fprintf(testR1, "\n\n\n\n");
                }
            }

            for (int k=0;k<6;k++)
           {
                if ( rank==2)
                {
                      for (int j=0;j<JT;j++)

     fprintf(testS2,"%13lf\n",gdf[k][IC(IT1,j)]);
                      fprintf(testS2, "\n\n\n\n");
                }
                if ( rank==1)
                {
                      for (int j=0;j<JT;j++)

     fprintf(testR2,"%13lf\n",gdf[k][IC(IT2,j)]);
                      fprintf(testR2, "\n\n\n\n");
                }
            }
           fclose(testR1);
           fclose(testR2);
           fclose(testS1);
           fclose(testS2);
           printf("%d\n",rank);
     }
 }
```

А также непосредственно в теле программы реализован обмен значениями d_t для синхронизации шага по времени.

Ниже предоставлен фрагмент кода, осуществляющий передачу значений:

```
d_tMu_max[0]=d_t;
   d_tMu_max[1]=mu_max;
   MPI_Gather( d_tMu_max, 2, MPI_DOUBLE , d_tMu_max, 2,
MPI_DOUBLE,0,MPI_COMM_WORLD);
   if (rank==0){
    for (int k=1;k<size;k++){
      d_t=min(d_t, d_tMu_max[k*2]);
```

```
      mu_max=max(mu_max, d_tMu_max[k*2+1]);}
    d_tMu_max[0]=d_t;
    d_tMu_max[1]=mu_max;
   }
   MPI_Bcast(d_tMu_max, 2, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

Кроме того использованы следующие функции библиотеки MPI:

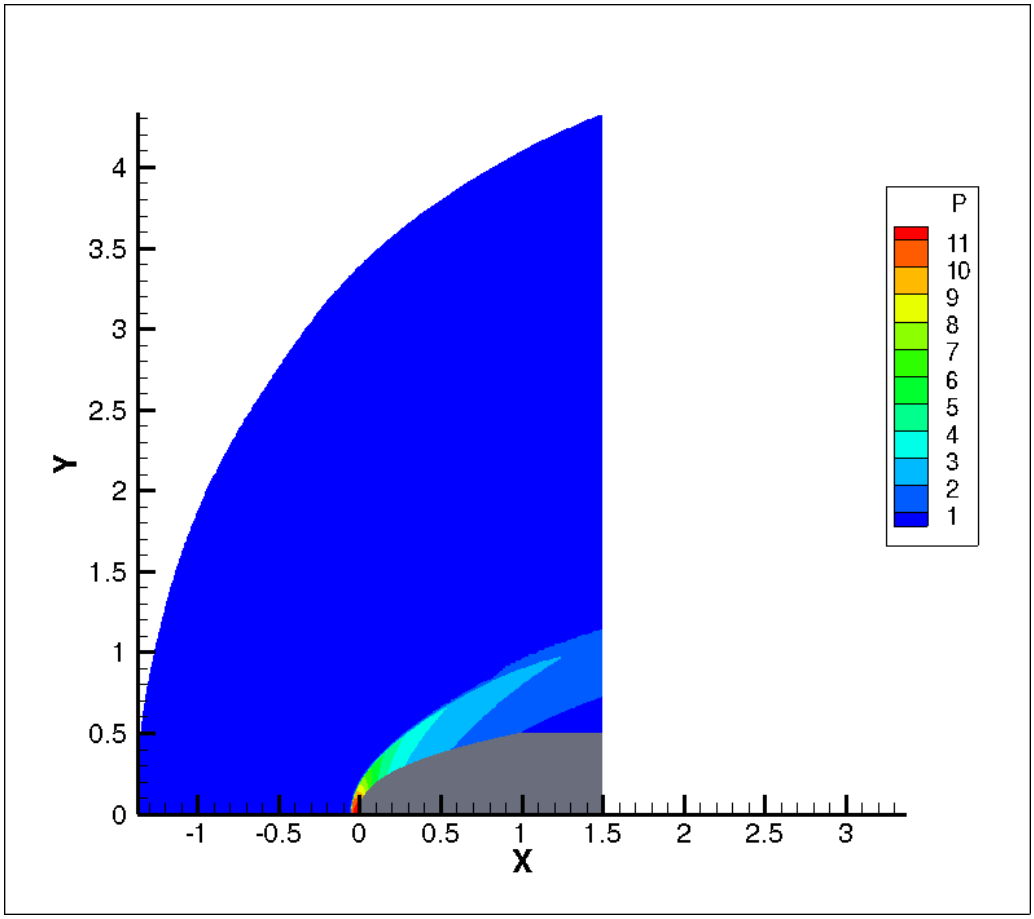
1. int MPI_Init(int *argc, char ***argv) и int MPI_Finalize(void). В результате работы первой функции создается группа процессов и область связи, описываемая предопределенным коммуникатором MPI_COMM_WORLD, в которую помещены все процессы. Вторая функция закрывает все MPI-процессы и ликвидирует все области связи.
2. int MPI_Comm_size(MPI_Comm comm, int *size) — Функция определения числа процессов в области связи.
3. int MPI_Comm_rank(MPI_Comm comm, int *rank) — Функция определения номера процесса.
4. int MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm) — Функция передачи сообщения.
5. int MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status) — Функция приема сообщения.
6. int MPI_Barrier(void) — Функция синхронизации процессов, которая блокирует работу вызвавшего ее процесса до тех пор, пока все другие процессы группы также не вызовут эту функцию. Завершение работы этой функции возможно только всеми процессами одновременно.
7. double MPI_Wtime(void) — Функция, которая возвращает количество секунд, представляя полное время по отношению к некоторому моменту времени в прошлом. Гарантируется, что этот момент в прошлом не изменяется на протяжении времени жизни процесса.

8. int MPI_Bcast(void* buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm) — Функция для рассылки одинакового сообщения от одного процесса всем остальным в области связи.
9. int MPI_Gather(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm) — Функция, которая производит сборку блоков данных, посылаемых всеми процессами группы, в один массив процесса с номером root.
10. int MPI_Scatter(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm) — Функция, которая разбивает сообщение из буфера посылки процесса root на равные части размером sendcount и посылает i-ю часть в буфер приема процесса с номером i.

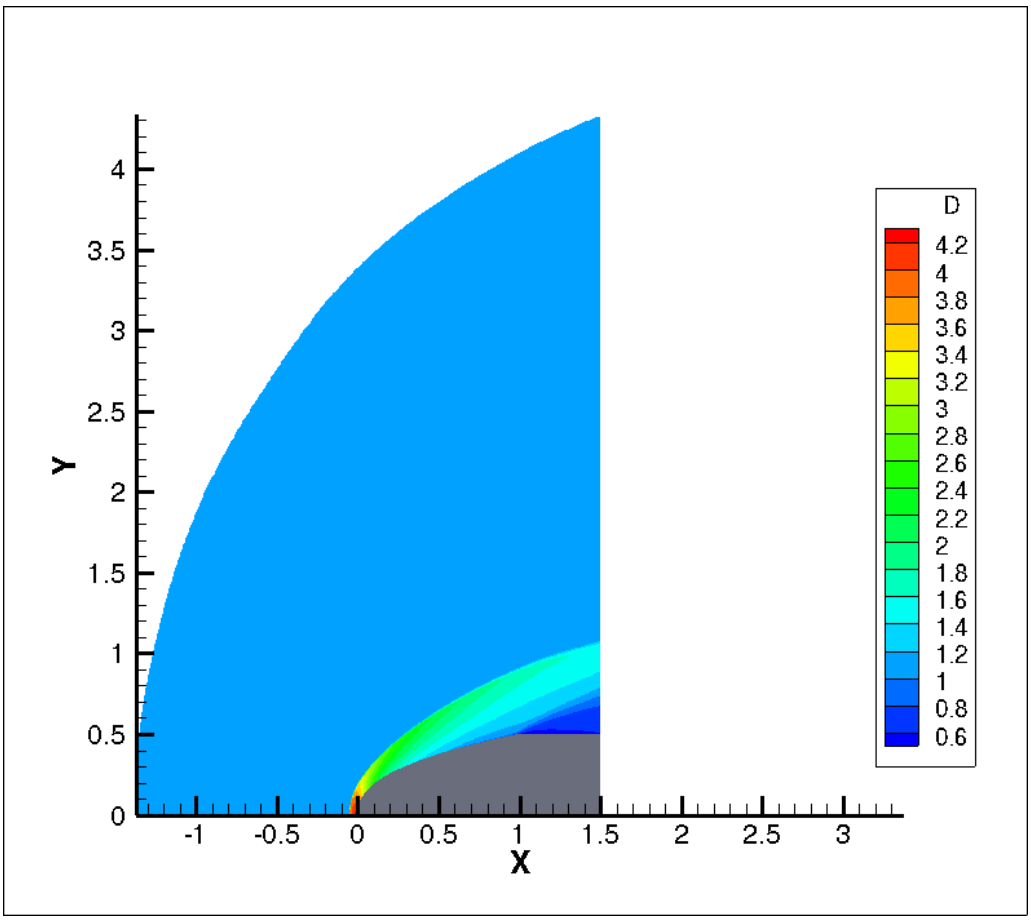
Примеры расчетов

После распараллеливания были проведены несколько расчетов на оригинальном и на моем коде с целью сравнения результатов работы и времени выполнения (для уменьшения затрат времени количество шагов по времени было ограничено 5000).

Далее предоставлен результат расчета для симметричной задачи обтекания цилиндра потоком. В связи с симметричностью задачи на графиках продемонстрирована только область, находящаяся выше плоскости симметрии.



Обтекание цилиндра, поля давления.



Обтекание цилиндра, поля плотности.

Получены следующие результаты замеров времени при запуске на:

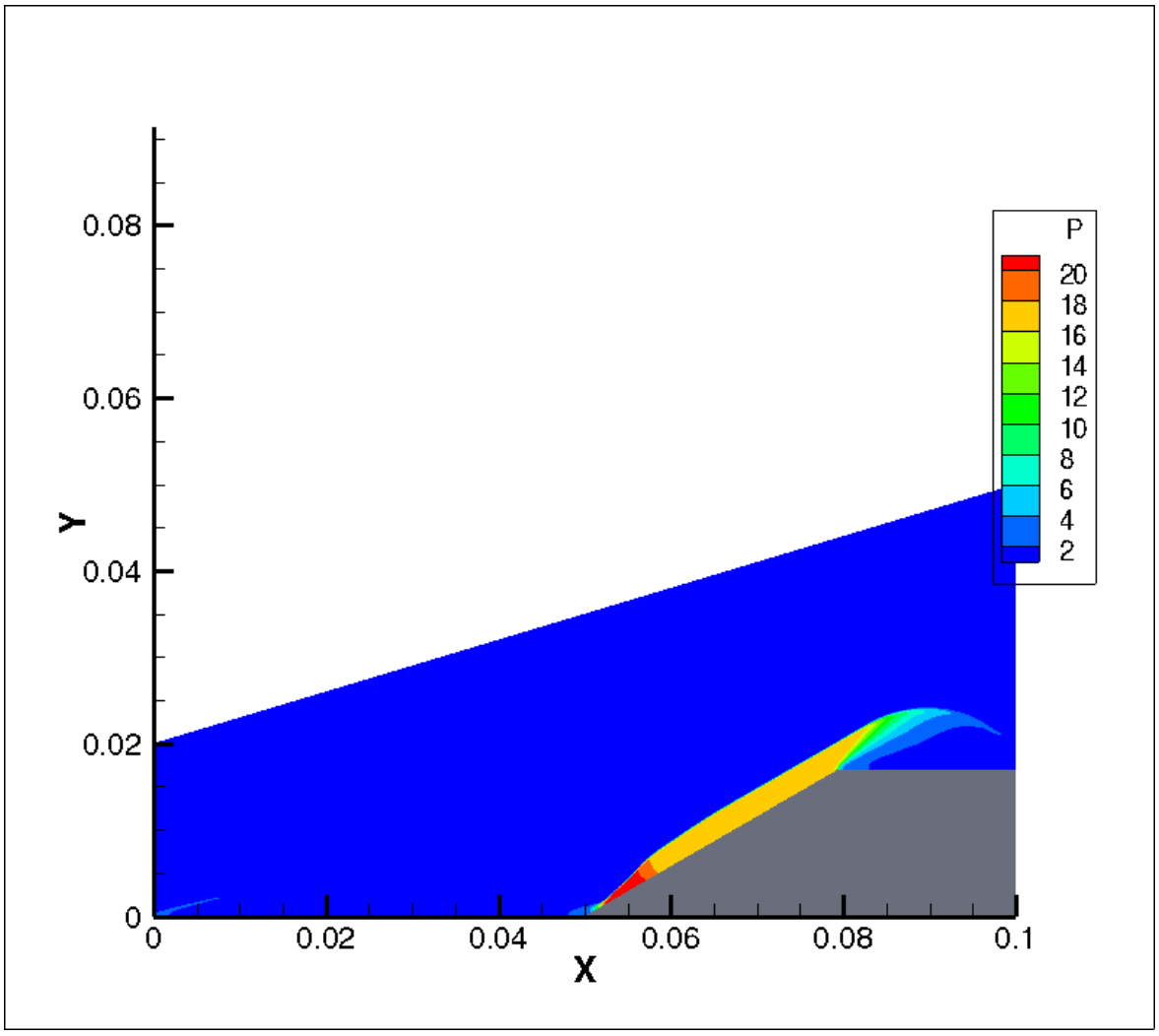
Одном процессе	Двух процессах	Четырех процессах
731,7809	410,1031	385,2567

Передача данных при расчете на двух процессах	
Передача d_t для вычисления оптимальной длинны шага	Передача граничных ячеек между процессами
0,0168	0,7061
Передача данных при расчете на четырех процессах	

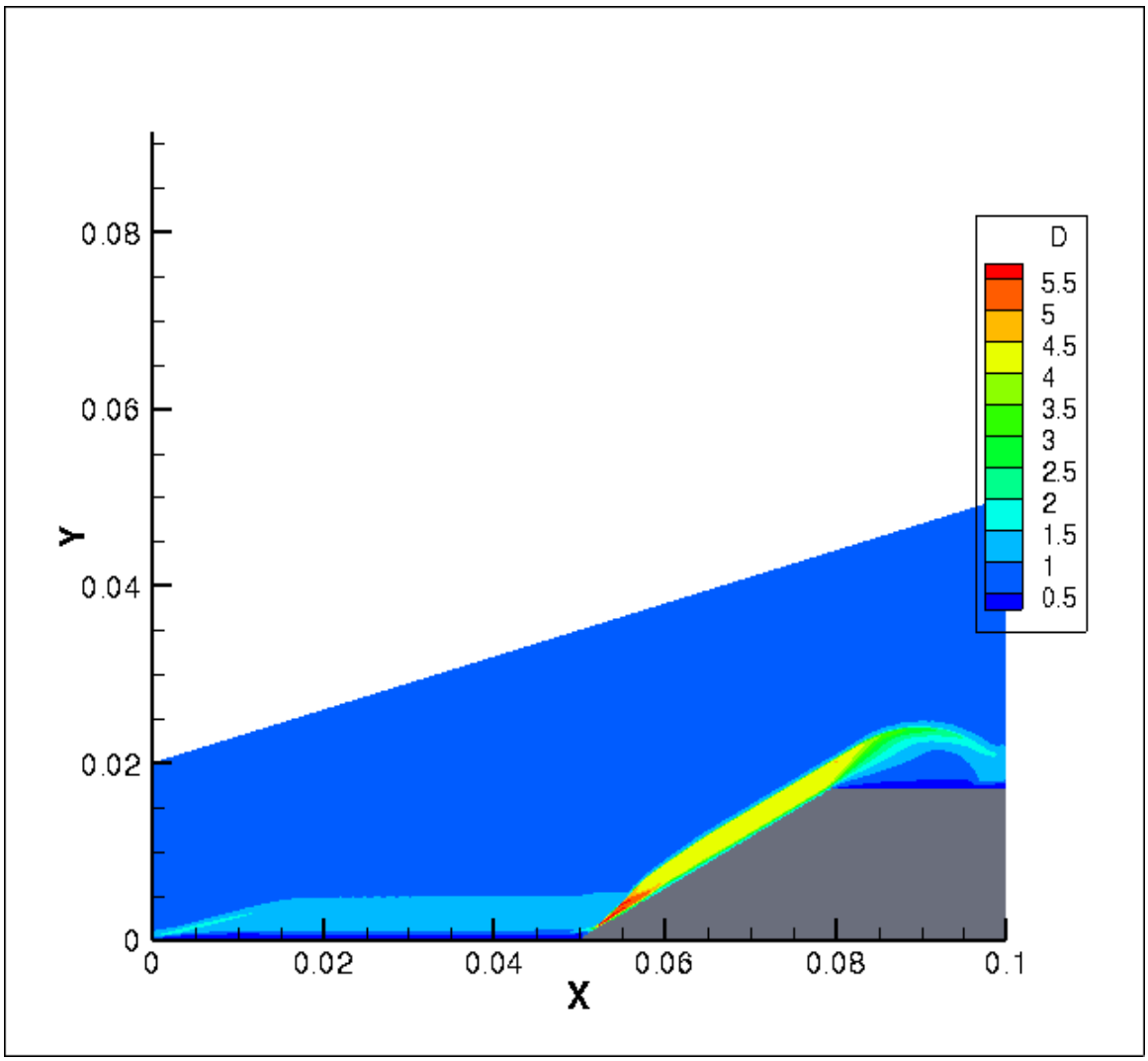
Передача d_t для вычисления оптимальной длинны шага	Передача граничных ячеек между процессами
0,1815	1.1965

Также были произведены замеры времени, потраченного на передачу данных между процессами:

Кроме того был произведен расчет для задачи обтекания угла на полоской пластине.



Обтекание угла, пола давления.



Обтекание угла, поля плотности.

Получены следующие результаты замеров времени при запуске на:

Одном процессе	Двух процессах	Четырех процессах
5335,3797	3114,2256	2756,4223

Передача данных при расчете на двух процессах	
Передача d_t для вычисления оптимальной длинны шага	Передача граничных ячеек между процессами

0,0172	0,9097
Передача данных при расчете на четырех процессах	
Передача d_t для вычисления оптимальной длинны шага	Передача граничных ячеек между процессами
0,0312	1,5725

Также были произведены замеры времени, потраченного на передачу данных между процессами:

Заключение

1. Изучена библиотека функций для обмена данными между процессами и написана пробная программа для закрепления навыков работы с MPI.

2. Разобраны основные методы параллелизации кода.

3. Написана параллельная реализация последовательного кода для решения задач газовой динамики.

Список использованных источников

1. Алгоритм многоуровневой адаптации сеток по критериям на основе вейвлет-анализа для задач газовой динамики, А.Л.Афендиков [и др.] Препринты ИПМ им. М.В.Келдыша. 2015. № 97.
2. Богачев К.Ю. Основы параллельного программирования. – М.: БИНОМ. Лаборатория знаний, 2003.
3. Антонов А.С. Параллельное программирование с использованием технологии MPI: Учебное пособие. – М.: Издательство МГУ, 2004.
4. Шпаковский Г.И., Серикова Н.В. Программирование для многопроцессорных систем в стандарте MPI. – М.: Издательство БГУ, 2002
5. http://parallel.ru/tech/tech_dev/MPI/examples/
6. Колган В.П. Применение принципа минимальных значений производных к построению конечно-разностных схем для расчета разрывных решений газовой динамики // Ученые записки ЦАГИ, 1972, 3, №6, с. 68-77.
7. Якобовский М.В. Введение в параллельные методы решения задач: Учебное пособие / Предисл.: В. А. Садовничий. – М.: Издательство Московского университета, 2013. – 328 с., илл. – (Серия «Суперкомпьютерное образование»), список исправлений: http://lira.imamod.ru
8. Кудряшов И.Ю., Луцкий А.Е., Северин А.В. Численное исследование отрывного трансзвукового обтекания моделей с сужением хвостовой части // Препринты ИПМ им. М.В.Келдыша. 2010. № 7. 12 с. URL: http://library.keldysh.ru/preprint.asp?id=2010-7