Второе практическое задание по OpenMP: решение дифференциальных уравнений в частных производных

Введение

Дифференциальные уравнения в частных производных представляют собой широко применяемый математический аппарат при разработке моделей в самых разных областях науки и техники. К сожалению, явное решение этих уравнений в аналитическом виде оказывается возможным только в частных простых случаях. Решение сложных уравнений обеспечивается при помощи приближенных численных методов решения. Объем выполняемых при этом вычислений обычно является достаточно большим, и использование высокопроизводительных вычислительных систем является традиционным для данной области вычислительной математики.

В данном задании вам надо будет разработать параллельную программу решения задачи Дирихле для уравнения Пуассона. Перед началом выполнения данного задания мы вам настоятельно рекомендуем просмотреть все видео второго модуля нашего курса, если вы еще этого не сделали, и пройти тест второго модуля, а также внимательно прочитать описание самой задачи.

Для начала выполнения данного задания вам необходимо скачать архив со стартовым проектом и распаковать его в директорию, в которой вы будете выполнять практическое задание. Для выполнения данного задания вам понадобится Microsoft Visual Studio 2015. Инструкцию по установке данного программного продукта вы можете найти в вводном модуле нашего курса.

Файлы входящие в задание

Папки:
Ex2 – основная папка всего решения;
GaussSeidel – папка, содержащая исходные файлы.

Файлы с исходным кодом:
main.cpp – файл, содержащий основную функцию main;
init.cpp – файл, содержащий функцию, задающую краевые и начальные значения;
calc_seq.cpp– файл, содержащий функцию, реализующую последовательный алгоритм Гаусса-Зейделя.
calc_par.cpp – файл, содержащий заготовку функции calc_par, в которой вам надо реализовать параллельный алгоритм Гаусса-Зейделя.

Заголовочные файлы:
GaussSeidel.h – заголовочный файл, содержащий описание основных функций.

1. Постановка задачи Дирихле

В этом задании рассматривается проблема численного решения задачи Дирихле для уравнения Пуассона, определяемая как задача нахождения функции $u = u(x, y)$, удовлетворяющей в области определения D уравнению:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y),\ (x, y) \in D$$

Подобная модель может быть использована для описания установившегося течения жидкости, стационарных тепловых полей, процессов теплопередачи с внутренними источниками тепла и деформации упругих пластин и др.

Для упрощения в качестве области задания функции будет использоваться единичный квадрат:

$$D = \{(x, y) \in R^2 : 0 \le x, y \le 1\}.$$

Для решения поставленной задачи воспользуемся методом конечных разностей. Представим область решения в виде дискретного равномерного набора точек (узлов). Так, например, прямоугольная сетка в нашей области D может быть задана в следующем виде

$$\begin{cases} D_h = \{(x_i, y_j) : x_i = ih, y_i = jh, 0 \le i, j \le N+1\}, \\ h = 1/(N+1), \end{cases}$$

где величина N задает количество внутренних узлов по каждой из координат области D (рисунок 1).

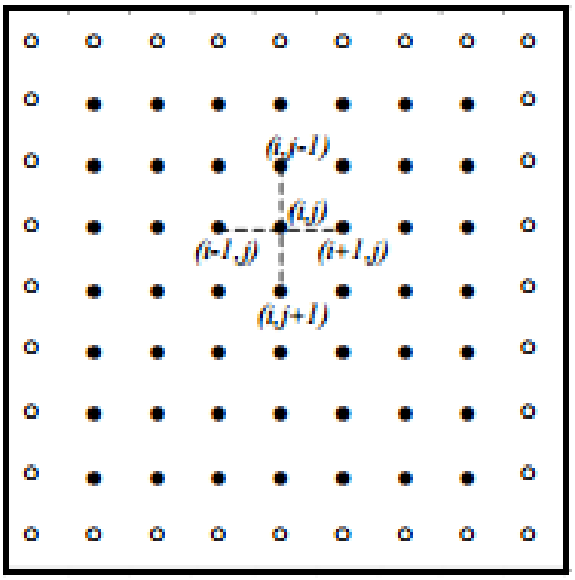

Рисунок 1 – Прямоугольная сетка в области D (темные точки представляют внутренние узлы сетки, нумерация узлов в строках слева направо, а в столбцах – сверху вниз)

Обозначим оцениваемую при подобном дискретном представлении аппроксимацию функции $u(x, y)$ в точках через u_{ij}. Тогда, используя пятиточечный шаблон (рисунок 1) для вычисления значений производных, мы можем представить уравнение Пуассона в конечно-разностной форме

$$\frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{ij}}{h^2} = f_{ij} \quad (1)$$

Данное уравнение может быть разрешено относительно u_{ij}

$$u_{ij} = 0.25(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - h^2 f_{i,j}) \quad (2)$$

Разностное уравнение, записанное в подобной форме, позволяет определять значение по известным значениям функции в соседних узлах используемого шаблона. Данный результат служит основой для построения различных итерационных схем решения задачи Дирихле, в которых в начале вычислений формируется некоторое приближение для значений, а затем эти значения последовательно уточняются в соответствии с приведенным соотношением. Воспользуемся методом Гаусса-Зейделя, в котором для уточнения использует следующая формула

$$u_{i,j}^k = 0.25(u_{i-1,j}^k + u_{i+1,j}^{k-1} + u_{i,j-1}^k + u_{i,j+1}^{k-1} - h^2 f_{i,j}) \quad (3)$$

по которому очередное k-ое приближение значения u_{ij} вычисляется по последнему k-ому приближению значений $u_{i-1,j}, u_{i,j-1}$ и предпоследнему (k-1)-ому приближению значений $u_{i+1,j}$ и $u_{i.j+1}$. Выполнение итераций обычно продолжается до тех пор, пока получаемые в результате итераций изменения значений u_{ij} не станут меньше некоторой заданной величины (требуемой точности вычислений). Последовательность решений, получаемых методом сеток, равномерно сходится к решению задачи Дирихле, а погрешность решения имеет порядок h^2.

Последовательный код решения задачи Дирихле для уравнения Пуассона с использованием метода Гаусса-Зейделя вы можете найти в исходных кодах начального проекта.

2. Компиляция и запуск последовательной версии программы

В первой части задания вам нужно задать количество узлов, равное 600 (за количество узлов по каждой из координат отвечает переменная N), и точность вычислений, равную 0,0001 (за точность вычислений отвечает переменная eps).

После того как вы выполните задание, раскомментируйте вызов функции `Submit_Part1()` в файле main.cpp, перекомпилируйте программу и запустите ее. После завершения программы в папке ex1 будет создан файл submit1.bin, который надо загрузить на платформу Coursera для проверки первой части задания.

3. Реализация параллельной версии программы

В данном пункте Вам необходимо провести распараллеливание вычислений.

Как вы видите из формулы расчета 3, при вычислении используются как старые, так и новые значения. В связи с этим распараллелить вычисления, выполняемые в функции `Calc_serial`, просто добавлением директив parallel и for не получится, так как вы не можете гарантировать, какой элемент в какой момент времени будет рассчитываться без синхронизации. Из рисунка 2 видно, что при вычислении нового значения $u_{i,j}$ в зависимости от скорости выполнения операций потоками используются разные значения (от предыдущей или текущей итерации) $u_{i,j}$. Так в первом случае второй поток использует значения предшествующей итерации $u_{i,j}$, во втором случае второй поток использует значения текущей итерации $u_{i,j}$, в третьем случае второй поток использует значения предшествующей и текущей итерации $u_{i,j}$. Тем самым данный алгоритм приводит к тому, что получаемые значения $u_{i,j}$ будут отличаться от получаемых значений в последовательном алгоритме.

Рисунок 2

Поэтому вам необходимо реализовать параллельный алгоритм, который бы на промежуточных шагах давал точно такой же результат, как и последовательный.

Такой параллельный алгоритм состоит в реализации волновой схемы вычислений. В последовательном алгоритме метода Гаусса-Зейделя каждое k-ое приближение вычисляется по последнему k-ому приближению значений и предпоследнему (k-1)-ому

приближению значений. При требовании совпадения результатов вычислений при последовательной и параллельной реализациях в начале каждой итерации может быть вычислено только одно значение $u_{1,1}$. Затем, зная $u_{1,1}$ на текущей итерации, могут быть вычислены значения $u_{1,2}$ и $u_{2,1}$, затем, зная $u_{1,2}$ и $u_{2,1}$, можно вычислить значение $u_{1,3}$,$u_{2,2}$ и $u_{3,1}$ и т.д. Таким образом, выполнение итерации Гаусса-Зейделя можно разбить на последовательность шагов, на каждом из которых к вычислениям окажутся подготовленными узлы вспомогательной диагонали сетки с номером, определяемом номером этапа (рисунок 3). Такая вычислительная схема получила наименование волны (или фронта) вычислений. Если реализовать такой алгоритм для отдельных элементов матрицы, то он будет не эффективный (вы можете это проверить самостоятельно), поэтому такой подход необходимо реализовать совместно с блочными вычислениями. Т.е. на рисунке 3 каждый квадрат надо рассматривать не как отдельный элемент, а как блок элементов. Блочный последовательный алгоритм реализован в функции `Calc_seq_block`. Ваша задача – реализовать параллельный блочный алгоритм с использованием волны вычислений.

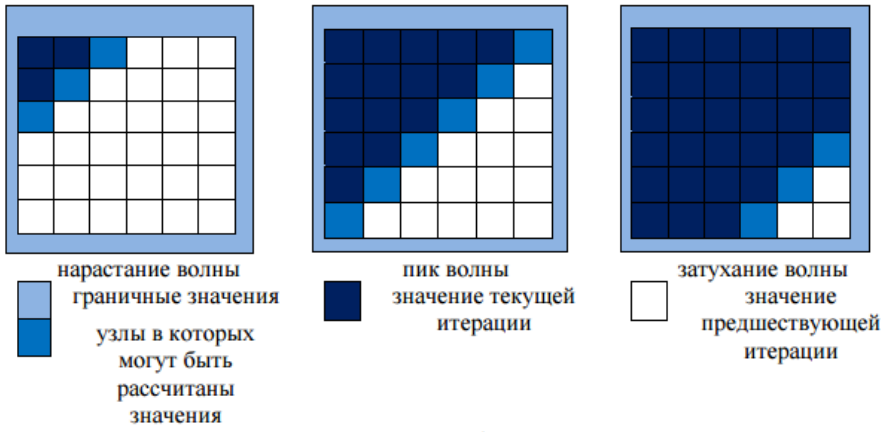

Рисунок 3

Подсказки

- Удобнее реализовать такой алгоритм в виде двух циклов, первый проходит до главной побочной диагонали, а после главной проход выполняется вторым циклом.
- Для повышения эффективности алгоритма изменяйте размер блока.

После того как выполните задание, проверите корректность работы и получите ускорение, раскомментируйте вызов функции `Submit_Part2()` в файле main.cpp, перекомпилируйте программу и запустите ее. После завершения программы в папке ex1 будет создан файл submit2.bin, который надо загрузить на платформу Coursera для проверки во второй части задания.

Оценка заданий

После выполнения всех частей убедитесь, что вы загрузили все файлы в систему проверки заданий и все задания у вас были засчитаны.
В таблице 1 представлены названия заданий и баллы за каждую часть.

Таблица 1

Название части	**Баллы**
Компиляция и запуск последовательной версии программы	15
Реализация параллельной версии программы	85
Итого	100