УДК 519.6

СРАВНЕНИЯ ЭФФЕКТИВНОСТИ РАЗЛИЧНЫХ ПАРАЛЛЕЛЬНЫХ РЕАЛИЗАЦИЙ МЕТОДА ГАУССА-ЗЕЙДЕЛЯ ДЛЯ ТЕХНОЛОГИИ OPENMP

Замятина А.В.
Научный руководитель – к.ф.-м.н. доцент Карепова Е.Д.

Сибирский федеральный университет

Рассмотрим задачу Дирихле для уравнения Пуассона. Пусть D – квадрат на плоскости $D = \{(x,y) \in D: 0 \le x,y \le \pi\}$ с границей ∂D. Найти функцию $u = u(x,y)$, удовлетворяющую в D уравнению Пуассона

$$\partial^2 u/\partial x^2 + \partial^2 u/\partial y^2 = f(x,y), (x,y) \in D \quad (1)$$

и принимающей на границе ∂D известные значения:

$$u(x,y)=g(x,y), (x,y) \in \partial D \quad (2)$$

Здесь $f(x,y)$ и $d(x,y)$ функции, известные в области D и на границе ∂D ссоответственно.

Для численного решения задачи (1) – (2) используем метод конечных разностей. Для этого в области D введем равномерную разностную сетку размерности $N \times N$ с шагом $h=\pi / n$:

$$\overline{D}_h = \{(x_i,y_j): x_i=ih, y_j=ih, 0 \le i,j \le N\}.$$

Используя для аппроксимации вторых производных центральные разности второго порядка, можно переписать задачу (1) – (2) в конечно-разностной форме:

$$\begin{cases} (u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j})/h^2 = f_{i,j} \\ u_{i,0}=\varphi_{i,0}, \; u_{i,N}=\varphi_{i,N}, \; 0 \le i \le N \\ u_{0,j}=\varphi_{0,j}, \; u_{N,j}=\varphi_{N,j}, \; 0 \le j \le N. \end{cases} \quad (3)$$

Здесь для значений функций φ приняты следующие обозначения: $\varphi_{i,j}= \varphi(x_i,y_j)$. Уравнение (3) можно переписать в следующем виде:

$$u_{i,j}= 0.25*(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - h^2 f_{i,j}).$$

Применим для решения задачи (3) итерационный метод Гаусса – Зейделя, который покомпонентно можно записать следующим образом:

$$\begin{cases} u_{i,j}^{k} = 0.25*(u_{i-1,j}^{k} + u_{i+1,j}^{k-1} + u_{i,j-1}^{k} + u_{i,j+1}^{k-1} - h^2 f_{i,j}) \\ u_{i,j}^{k}=0, \; 0<i,j<N \\ u_{i,0}=\varphi_{i,0}, \; u_{i,N}=\varphi_{i,N}, \; 0 \le i \le N \\ u_{0,j}=\varphi_{0,j}, \; u_{N,j}=\varphi_{N,j}, \; 0 \le j \le N. \end{cases} \quad (4)$$

Здесь верхний индекс k означает номер итерации. Выполнение итераций обычно продолжается до тех пор, пока получаемые в результате итераций изменения значений $u_{i,j}$ не станут в некоторой норме меньше требуемой точности вычислений. В нашем случае для останова итерационного процесса используется дискретный аналог равномерной нормы:

$$\max_{i,j} | u_{i,j}^{k} - u_{i,j}^{k-1} | \le \varepsilon$$

Таким образом, значение функции u на текущей итерации рассчитывается по четырем значениям функции, два из которых берется с предыдущей итерации, а два уже насчитаны на текущей и, соответственно, используются при расчетах.

Рассмотрим различные способы организации параллельных вычислений для задачи Дирихле уравнения Пуассона методом Гаусса-Зейделя на многопроцессорных вычислительных системах с общей памятью с использованием технологии OpenMP. Все тестовые расчеты будем проводить на одном восьмиядерном узле малого кластера СФУ.

Сравнивать различные параллельные реализации алгоритмов будем на основе сравнения ускорения параллельной программы относительно последовательной.

Ускорение S алгоритма оценивается как отношение времени вычисления на одном процессоре T_1 к времени вычисления на p процессорах T_p:

$$S_p=T_1/T_p.$$

При наличии достаточного количества ядер время выполнения программы на p нитях складывается из времени вычислений T_{calc}, которое при условии сбалансированного распределения нагрузки близко к T_1/p и времени T_{sinch}, затрачиваемого нитями на синхронизацию:

$$T_p = T_{calc} + T_{sinch} = T_1/p + T_{sinch}. \qquad (5)$$

Первый вариант параллельного алгоритма исходной задачи может быть получен, если разрешить произвольный порядок пересчета значений $u_{i,j}$. Данный параллельный алгоритм был получен из последовательного кода программы путем добавления соответствующих директив и функций технологии OpenMP и с использованием ленточной схемы организации параллельных вычислений. Из рис. 3–8 видно что данный алгоритм обеспечивает наилучшее по сравнению с другими вариантами ускорение. Так, в случае размерности сетки 800x800 достигается почти линейное ускорение. Однако данный параллельный алгоритм не является строгим аналогом последовательного – порождаемая при вычислениях последовательность обработки данных при одних и тех же исходных параметрах решаемой задачи может различаться при разных запусках программы. Данный эффект может проявляться в силу изменения каких-либо условий выполнения программы (вычислительной нагрузки, алгоритмов синхронизации потоков и т.п.). В результате некоторые значения новой итерации u будут пересчитываться по значениям предыдущей итерации, а не текущей, как было бы в случае последовательной версии кода. Это приводит к несколько более медленной сходимости метода

Из рис.1 видно что при вычислении нового значения $u_{i,j}$ в зависимости от скорости выполнения операций потоками используются разные значение (от предыдущей или текущей итерации) $u_{i,j}$. Так в первом случае 2 поток использует значения предшествующей итерации $u_{i,j}$, во втором случае 2 поток использует значения текущей итерации $u_{i,j}$, в третьем случае 2 поток использует значения предшествующей и текущей итерации $u_{i,j}$. Тем самым данный алгоритм приводит к тому, что получаемые значения $u_{i,j}$ будет сходиться к точному решению исходной задачи с заданной точности, но могут быть различными.

Рис.1 Возможные различные варианты взаиморасположения параллельных потоков

Второй вариант параллельного алгоритма состоит в реализации волновой схемы параллельных вычислений [*В.П. Гергель, Р.Г. Стронгин. Основы параллельных вычислений для многопроцессорных вычислительных систем. Учебное пособие. Издательство Нижегородского госуниверситета, Нижний Новгород, 2003*]. В последовательном алгоритме метода Гаусса-Зейделя каждое *k*-ое приближение вычисляется по последнему *k*-ому приближению значений и 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 (рис.2). Такая вычислительная схема получила наименование *волны* или *фронта вычислений.*

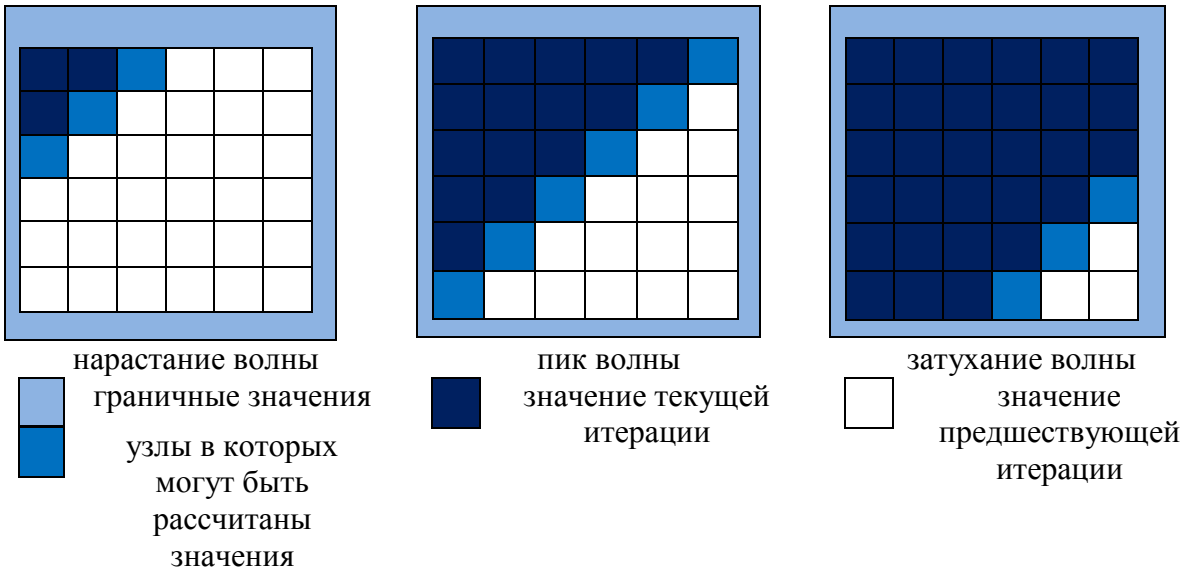

Рис2. Распространение фронта вычислений

Из рис. 3–8 видно, что данный алгоритм показывает неплохие результаты только на сетках больших размерностей (*n*=1600 и *n*=3200). Низкое ускорение данного алгоритма объясняется тем, что большую часть времени только часть нитей участвует в вычислениях, то есть велика доля последовательных операций в параллельной реализации или, другими словами, затраты на синхронизацию нитей. В начале каждой итерации может быть вычислено только одно значение одной нитью, а все остальные нити ожидают, на втором шаге проводить вычисления могут уже две нити, а остальные опять ждут, на третьем – три и т.д.

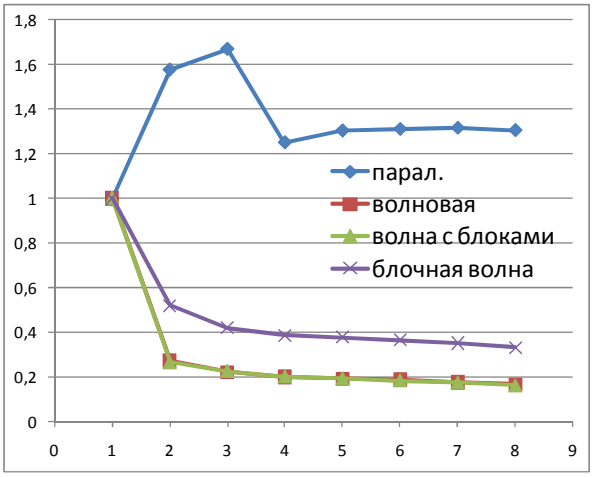

Рис3. Зависимость ускорения от различных способов организации параллельных вычислений на сетке размерности 100x100

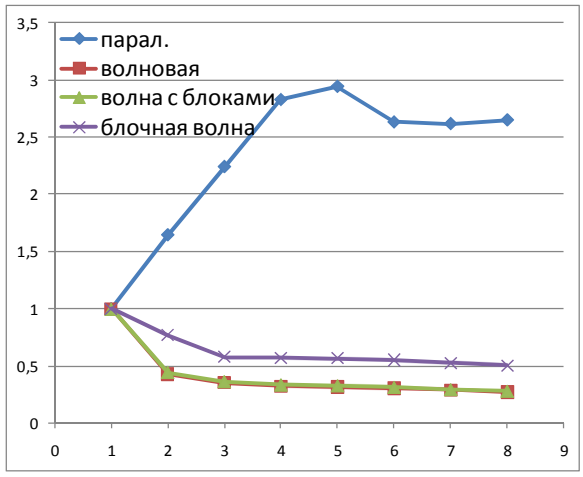

Рис4. Зависимость ускорения от различных способов организации параллельных вычислений на сетке размерности 200x200

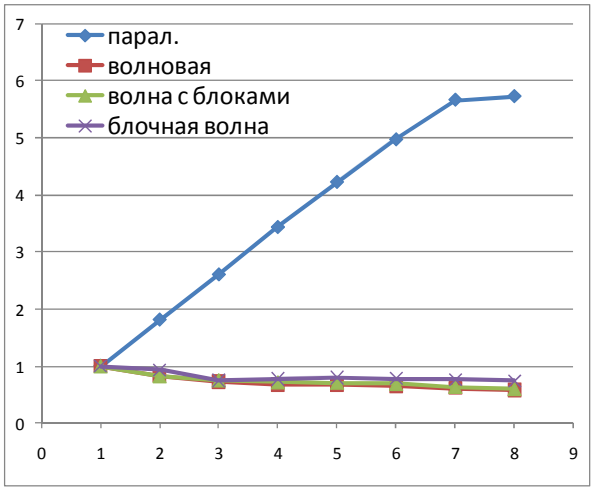

Рис5. Зависимость ускорения от различных способов организации параллельных вычислений на сетке размерности 400x400

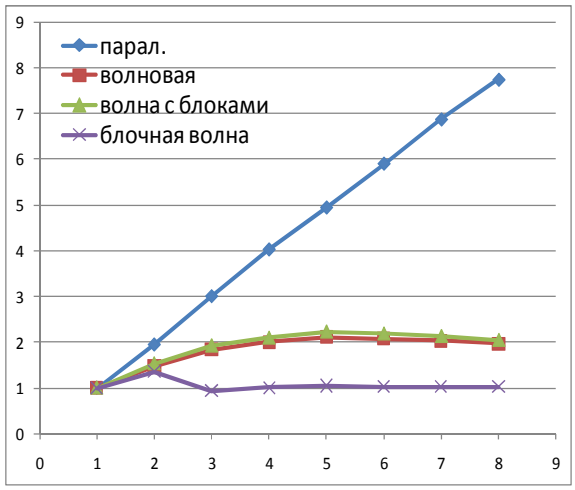

Рис6. Зависимость ускорения от различных способов организации параллельных вычислений на сетке размерности 800x800

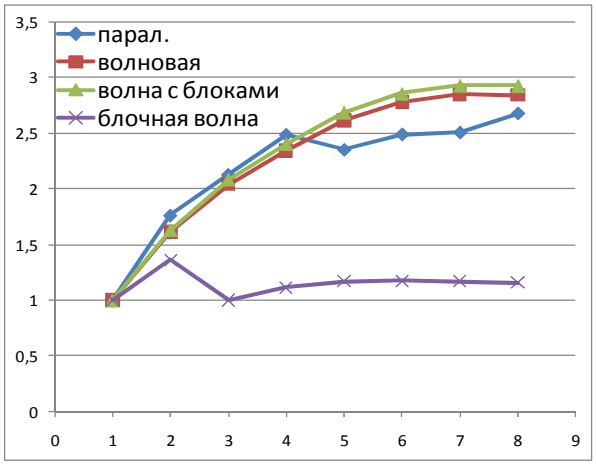

Рис7. Зависимость ускорения от различных способов организации

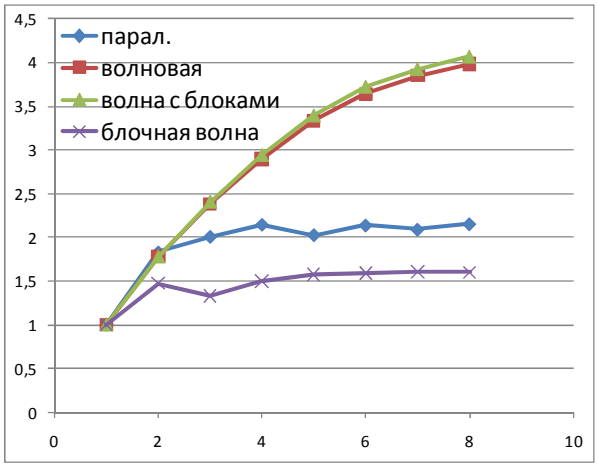

Рис8. Зависимость ускорения от различных способов организации

параллельных вычислений на сетке размерности 1600x1600

параллельных вычислений на сетке размерности 3200x3200

Третий вариант параллельного алгоритма (на рис. 3–8 обозначен как «волна с блоками») характеризуется увеличением параллельности при расчете критерия останова итерационного процесса. Действительно вычисление максимума в предыдущем варианте суть параллельная операция только в пределах строки сетки, для разных строк нахождение максимума во втором алгоритме принципиально последовательно. Увеличить параллельность можно увеличив размеры участков, которые можно вычислять параллельно, то есть осуществлять поиск максимума по блокам строк. Однако, из рис. 3–8 видно, что данный алгоритм дает лишь небольшое увеличение ускорения по сравнению с параллельным алгоритмом волновой схемы.

Четвертый вариант параллельного алгоритма («блочная волна») состоит в том чтобы разбить сетку на блоки и реализовать волновую схему второго варианта алгоритма по блокам. Этот вариант параллельной реализации метода направлен на попытку использования кэш-памяти для повышения производительности программы. Вычисления в предлагаемом подходе происходят в соответствии с волновой схемой обработки данных – вначале вычисления выполняются только в левом верхнем блоке с координатами (0,0), далее для обработки становятся доступными блоки с координатами (0,1) и (1,0). Из рис. 3–8 видно, что данный параллельный алгоритм не дает хорошего ускорения, и затраты на написание данного кода программы не оправдали себя.

Таким образом, наши исследования показывают, что самым выгодным является первый вариант параллельной программы, единственным недостатком которого является его не полное соответствие последовательной программе. Среди всех точных параллельных реализаций следует констатировать, что каждое новое повышение параллелизма предполагает существенное усложнение кода, но не дает видимого увеличения эффективности, а иногда приводит и к ее падению.