Отчёт по практическому заданию №3.

Ю. Н. Лукашкина

julialukashkina@gmail.ru

МГУ имени М. В. Ломоносова, Москва

19 декабря 2016 г.

Аннотация

Данный документ содержит отчет по практической работе №3 курса «Суперкомпьютерное моделирование и технологии» ВМК МГУ.

1 Математическая постановка задачи

В прямоугольной области

$$\Pi = [A_1, A_2] \times [B_1, B_2]$$

требуется найти дважды гладкую функцию $u = u(x, y)$, удовлетворяющую дифференциальному уравнению

$$-\Delta u = F(x, y), \quad A_1 < x < A_2, B_1 < y < B_2 \tag{1}$$

и дополнительному условию

$$u(x, y) = \varphi(x, y) \tag{2}$$

во всех граничных точках $(x,\ y)$ прямоугольника. Оператор Лапласа Δ определён равенством:

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

$$F(x, y) = (x^2 + y^2)\sin(xy), \quad \varphi(x, y) = 1 + \sin(xy),$$

Прямоугольник $\Pi = [0, 2] \times [0, 2]$.

Было подобрано точное решение задачи Дирихле: $u(x, y) = 1 + \sin(xy)$.

2 Численный метод решения задачи

В расчётной области Π определяется прямоугольная сетка

$$\bar{\omega}_h = \{(x_i, y_j), i = 0, 1, \ldots, N_1, j = 0, 1, \ldots, N_2\},$$

где $A_1 = x_0 < x_1 < x_2 < \cdots < x_{N_1} = A_2$ — разбиение отрезка $[A_1, A_2]$ оси (ox), $B_1 = y_0 < y_1 < y_2 < \cdots < y_{N_2} = B_2$ — разбиение отрезка $[B_1, B_2]$ оси (oy).

Через $omega_h$ обозначим множество внутренних, а через γ_h — множество граничных узлов сетки $\bar{\omega}_h$. Пусть $h_i^{(1)} = x_{i+1} - x_i, i = 0, 1, \ldots, N_1 - 1$, $h_j^{(2)} = y_{j+1} - y_j, j = 0, 1, \ldots, N_2 - 1$ — переменный шаг сетки по оси абсцисс и ординат соответственно. Средние шаги сетки определяются равенствами:

$$\hbar_i^{(1)} = 0.5\big(h_i^{(1)} + h_{i-1}^{(1)}\big), \quad \hbar_j^{(2)} = 0.5\big(h_j^{(2)} + h_{j-1}^{(2)}\big)$$

Рассмотрим линейное пространство H функций, заданных на сетке ω_h. Будем считать, что в пространстве H задано скалярное произведение и максимум норма

$$\big(u,v\big)=\sum_{i=1}^{N_1-1}\sum_{j=1}^{N_2-1}\hbar_i^{(1)}\hbar_j^{(2)}u_{ij}v_{ij},\quad \|u\|=\max_{\substack{0<i<N_1\\0<j<N_2}}|u_{i,j}| \tag{3}$$

где $u_{ij}=u(x_i,y_i), v_{ij}=v(x_i,y_j)$ — любые функции из пространства H.

Для аппроксимации уравнения Пуассона (1) воспользуемся пятиточечным разностным оператором Лапласа, который во внутренних узлах сетки определяется равенством:

$$-\Delta_h p_{ij}=\frac{1}{\hbar_i^{(1)}}\left(\frac{p_{ij}-p_{i-1j}}{h_{i-1}^{(1)}}-\frac{p_{i+1j}-p_{ij}}{h_i^{(1)}}\right)+\frac{1}{\hbar_j^{(2)}}\left(\frac{p_{ij}-p_{ij-1}}{h_{j-1}^{(2)}}-\frac{p_{ij+1}-p_{ij}}{h_j^{(2)}}\right).$$

Здесь предполагается, что функция $p=p(x_i,y_i)$ определена во всех узлах сетки $\bar{\omega}_h$.

Приближенным решением задачи (1), (2) называется функция $p=p(x_i,y_j)$, удовлетворяющая уравнениям

$$-\Delta_h p_{ij}=F(x_i,y_j),\ (x_i,y_j)\in\omega_h,\qquad p_{ij}=\varphi(x_i,y_j),\ (x_i,y_j)\in\gamma_h. \tag{4}$$

Эти соотношения представляют собой систему линейных алгебраических уравнений с числом уравнений равным числу неизвестных и определяют единственным образом неизвестные значения p_{ij}. Совокупность уравнений (4) назвается разностной схемой для задачи (1), (2).

Приближенное решение системы уравнений (4) может быть получено методом сопряжённых градиентов. Начальное приближение $p^{(0)}$ и первая итерация $p^{(1)}$ вычисляются так:

$$p_{ij}^{(0)}=\varphi(x_i,y_j),\quad (x_i,y_j)\in\gamma_h,$$

во внутренних узлах сетки $p_{ij}^{(0)}$ — любые числа. Метод является одношаговым. Итерация $p^{(k+1)}$ вычисляется по итерации $p^{(k)}$ согласно равенствам:

$$p_{ij}^{(k+1)}=p_{ij}^{(k)}-\tau_{k+1}g_{ij}^{(k)},\quad k=1,2,\ldots$$

Здесь

$$\tau_{k+1}=\frac{\big(r^{(k)},g^{(k)}\big)}{\big(-\Delta_h g^{(k)},g^{(k)}\big)},$$

вектор

$$g_{ij}^{(k)}=r_{ij}^{(k)}-\alpha_k g_{ij}^{(k-1)},\quad k=1,2,\ldots,$$

$$g_{ij}^{(0)}=r_{ij}^{(0)},$$

коэффициент

$$\alpha_k=\frac{\big(-\Delta_h r^{(k)},g^{(k-1)}\big)}{\big(-\Delta_h g^{(k-1)},g^{(k-1)}\big)}.$$

Вектор невязки $r^{(k)}$ вычисляется по следующим формулам:

$$r_{ij}^{(k)} = -\Delta_h p_{ij}^{(k)} - F(x_i, y_j),\ (x_i, y_j) \in \omega_h, \qquad r_{ij}^{(k)} = 0,\ (x_i, y_j) \in \gamma_h.$$

Итерационный процесс останавливается, как только

$$||p^{(n)} - p^{(n-1)}|| < \varepsilon,$$

по евклидовой норме, где $epsilon$ — заранее выбранное положительное число ($epsilon = 10^{-4}$).

3 Описание работы по созданию MPI / CUDA реализации

Все операции, кроме вычисления разностного оператора Лапласа, производятся независимо для каждого элемента матрицы. Это позволяет эффективно распараллелить работу с матрицами по процессорам, каждый из которых будет работать только с своей частью исходной матрицы. Для вычисления величин, которые требуют всю матрицу (таких как норма, скалярное произведение двух матриц и т.д.), процессоры будут синхронизироваться. Распараллеливание матрицы будет происходить по блокам/клеткам.

Чтобы корректно производить вычисление разностного оператора Лапласа каждый процессор кроме своей части матрицы будет хранить ещё граничные строки и столбцы. То есть одну предыдущую и одну последующую строчки относительно выделенных ему собственных строк (для процессоров, работающих с граничными частями матрицы соответствующие строки всегда будут пустыми). Аналогично будут храниться и дублирующие столбцы. Перед и после каждой операции подсчёта разностного оператора Лапласа процессоры будут обмениваться этими пограничными строками и столбцами.

3.1 Разделение матрицы

Исходная матрица делится между процессорами на клетки примерно одинакового размера. Для вычисления количества клеток по каждой из размерностей матрицы была использована функция `SplitFunction`, прилагающаяся к заданию. Число процессоров p является входным параметром и должно быть степенью двойки $p = 2^{power}$. Данную степень можно представить в следующем виде: $power = p_1 + p_2$, где $2^{p_1}, 2^{p_2}$ — число процессоров по первой и второй размерностям матрицы соотвественно.

3.2 MPI

Были использованы следующие функции:

— Для вычисления максимум нормы и подсчёта суммы.

```
MPI_Allreduce(...);
```

— Для обмена строками матрицы.

```
MPI_Sendrecv(...);
```

— Для подсчёта времени.

```
MPI_Wtime();
```

3.3 CUDA

— Для вычисления матричных операций и приближенных операций использовалась $2D$ сетка.
— Для вычисления reduce операций (максимума и суммы) использовалась $1D$ сетка.

3.4 Вывод результатов программы

Результатом работы программы является вывод в консоль времени работы программы и погрешности решения. Также каждый процессор сохраняет в отдельные файлы свои части получившейся приближенной матрицы решения.

Таблица 1. Таблица с результатами расчётов на ПВС «Ломоносов»

Число процессоров N_p	Число точек сетки N^3	Время решения T	Ускорение S	Эффективность
1	1000×1000	379.135		
2	1000×1000	192.87	1.97	0.99
4	1000×1000	99.3778	3.82	0.96
8	1000×1000	56.7781	6.78	0.85
1	2000×2000	2979.66		
2	2000×2000	1505.57	1.98	0.99
4	2000×2000	761.021	3.91	0.98
8	2000×2000	478.44	6.23	0.78

Таблица 2. Таблица с результатами расчётов на ПВС «Ломоносов», CUDA

Число процессоров N_p	Число точек сетки N^3	Время решения T	Ускорение S	Эффективность
1	1000×1000	20.3783		
2	1000×1000	11.9322	1.70	0.85
4	1000×1000	7.62324	2.67	0.66
8	1000×1000	5.1586	3.83	0.48
1	2000×2000	144.604		
2	2000×2000	76.009	1.90	0.95
4	2000×2000	41.346	3.50	0.88
8	2000×2000	24.0867	6.01	0.75

4 Результаты расчётов

4.1 MPI ПВС «Ломоносов»

По таблице 1 видно, что реализованная программа распараллеливается хорошо. На обеих сетках для небольшого числа процессоров эффективность примерно одинаковая. Значения ускорения на сетках в 1000 и 2000 узлов отличаются незначительно. При конфигурации запуска с 8 процессорами эффективность ощутимо уменьшается. Это связано с тем, что затраты на передачу информации между процессорами становятся слишком высокими.

4.2 CUDA/MPI «Ломоносов»

По таблице 2 видно, что для сетки из 1000 точек значения эффективности в среднем ниже, чем для сетки из 2000 точек. На сетке большего размера достигается большее ускорение и эффективность в среднем выше. На обеих сетках эффективность уменьшается с ростом числа процессоров. Это, скорее всего, связано с тем, что размер клетки, выделяемой каждому процессору становится всё меньше и накладные расходы на управление потоками не компенсируются полученным ускорением. Этот эффект особенно хорошо заметен на сетке из 1000 точек.

4.3 CUDA/MPI и MPI «Ломоносов»

При одинаковом числе процессоров время работы на ПВС «Ломоносов» с использованием CUDA значительно меньше, чем без. Для 1 процессора ≈ 20 и 144 против ≈ 379 и 2979. Для 8 процессоров ≈ 5 и 24 против ≈ 56 и 478. Значение эффективности в среднем ниже с использованием CUDA, чем без.

5 Выводы

Выполненная программная реализация хорошо масштабируется. Однако при росте числа процессоров и уменьшении размера клетки накладные расходы на распараллеливание уменьшают эффективность параллельной программы. Ускорение CUDA версии по сравнению с обычной значительно.