Московский Государственный Университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики

Отчёт по первому практическому заданию в рамках курса «Суперкомпьютерное моделирование и технологии»: решение задачи Дирихле с использованием MPI и CUDA

Выполнил: Роор Даниил Дмитриевич, 616 группа

Математическая постановка

Задача Дирихле в двумерной области *D* для функции *u(x,y)* записывается следующим образом:

$$-\Delta u = f(x,y),$$

В левой части уравнения применяется оператор Лапласа, определяющийся следующей формулой:

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

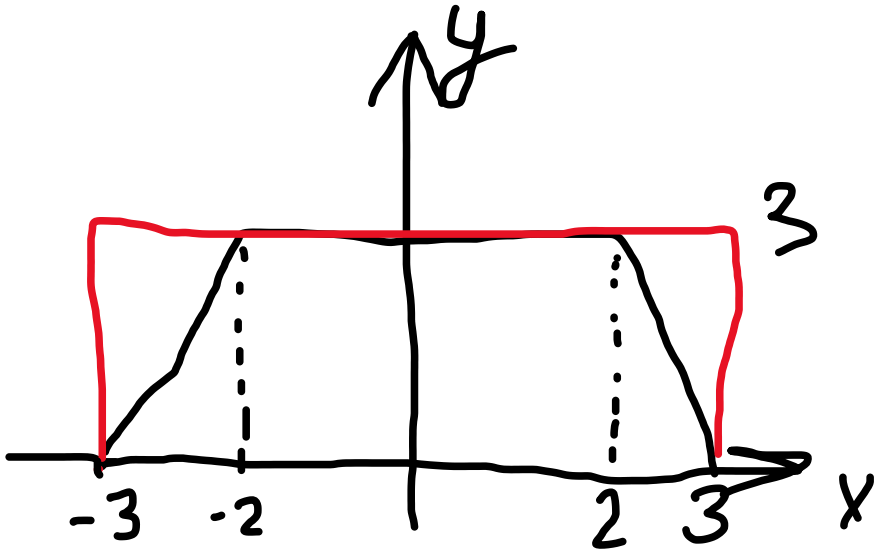
На границе области выполняется граничное условие *u(x,y) = 0*

Пусть область *D* - трапеция с вершинами в точках *A(−3,0),B(3,0),C(2,3),D(−2,3)* и *f(x,y)=1*

Эта трапеция лежит полностью внутри прямоугольника П, ограниченного прямыми

$x = A_1 = -3$, $x = B_1 = 3$, $y = A_2 = 0$, $y = B_2 = 3$

и симметрична относительно оси y:



Введём на прямоугольнике П двумерную сетку

$$\bar{\omega}_1 = \{x_i = A_1 + ih_1,\ i = \overline{0,M}\},\ \bar{\omega}_2 = \{y_j = A_2 + jh_2,\ j = \overline{0,N}\}.$$

где

$$h_1 = (B_1 - A_1)/M,\ h_2 = (B_2 - A_2)/N.$$

- шаги сетки, *M, N* - размеры сетки по осям x и y соответственно.

Значения сеточной функции, аппроксимирующей функцию *u* в узлах этой сетки обозначим матрицей *ω*

Задача может быть численно решена на данной сетке с помощью метода фиктивных областей.

Система уравнений может быть записана следующим образом

$$-\frac{1}{h_1}\left(a_{i+1j}\frac{w_{i+1j}-w_{ij}}{h_1} - a_{ij}\frac{w_{ij}-w_{i-1j}}{h_1}\right) - \frac{1}{h_2}\left(b_{ij+1}\frac{w_{ij+1}-w_{ij}}{h_2} - b_{ij}\frac{w_{ij}-w_{ij-1}}{h_2}\right) = F_{ij},$$
$$i = \overline{1, M-1},\ j = \overline{1, N-1},$$

В этой системе для левых частей уравнений:

$$a_{ij} = \frac{1}{h_2}\int\limits_{y_{j-1/2}}^{y_{j+1/2}} k(x_{i-1/2}, t)dt, \quad b_{ij} = \frac{1}{h_1}\int\limits_{x_{i-1/2}}^{x_{i+1/2}} k(t, y_{j-1/2})dt$$

$$i = \overline{1, M},\ j = \overline{1, N}.$$

где

$$k(x,y) = \left[\begin{array}{ll} 1, & (x,y) \in D, \\ 1/\varepsilon, & (x,y) \in \hat{D} \end{array}\right.$$

- кусочно-постоянная функция

$$\hat{D} = \Pi \setminus \overline{D}$$

- область прямоугольника П, лежащая за пределами трапеции

$$x_{i\pm1/2} = x_i \pm 0.5h_1, \quad y_{j\pm1/2} = y_j \pm 0.5h_2.$$

- полуцелые узлы сетки

Интеграл в формуле для a_{ij} есть длина части вертикального отрезка между полуцелыми узлами сетки, расположенной внутри трапеции, плюс длина части, расположенной вне трапеции, поделённая на некоторую малую константу.

Интеграл в формуле для b_{ij} имеет такой же смысл, но в нём отрезок горизонтальный.

Коэффициенты в правой части рассчитываются по формуле:

$$F_{ij} = \frac{1}{h_1 h_2} \iint\limits_{\Pi_{ij}} F(x,y)dxdy, \quad \Pi_{ij} = \{(x,y) : x_{i-1/2} \leqslant x \leqslant x_{i+1/2}, y_{j-1/2} \leqslant y \leqslant y_{j+1/2}\}$$

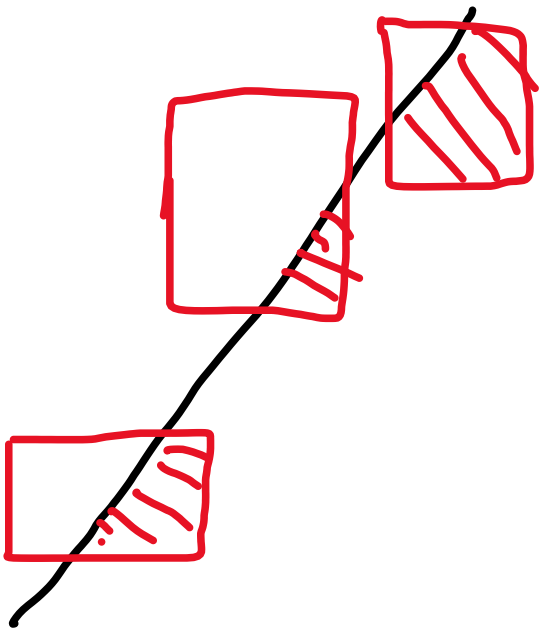
$$F(x,y) = \left[\begin{array}{ll} f(x,y), & (x,y) \in D, \\ 0, & (x,y) \in \hat{D}. \end{array} \right.$$

при

$$i = \overline{1, M-1}, \ j = \overline{1, N-1}.$$

Здесь двойной интеграл представляет собой площадь той части прямоугольника Π_{ij}, которая находится внутри трапеции.

Прямоугольника Π_{ij} может не лежать целиком ни внутри, ни вне трапеции, если его пересекает боковая сторона трапеции. В этом случае сторона трапеции отсекает от него либо трапецию, либо треугольник, либо пятиугольник. Площадь этой фигуры и есть искомое значение интеграла. Вид фигуры зависит от того, как боковая сторона трапеции пересекает прямоугольник и может быть определён по значениям полуцелых узлов, ограничивающих прямоугольник, и по координатам точек пересечения боковой стороны со сторонами прямоугольника.



Система может быть записана в матричном виде *Aω=B,* где оператор из левой части соответствует левым частям уравнений системы, а *B* - матрица из элементов F_{ij}.

Граничные условия для сеточной функции при *i = 0, I = M, j = 0, j = N* задаются условием

$$w_{ij} = w(x_i, y_j) = 0,$$

Численное решение будем находить итерационно методом наименьших невязок, в котором на каждой итерации новая сеточная функция вычисляется по следующему алгоритму:

$$r^{(k)} = Aw^{(k)} - B,$$

$$\tau_{k+1} = \frac{\left(Ar^{(k)}, r^{(k)}\right)}{\left\|Ar^{(k)}\right\|_E^2}.$$

$$w_{ij}^{(k+1)} = w_{ij}^{(k)} - \tau_{k+1} r_{ij}^{(k)},$$

Алгоритм начинается с произвольного приближения сеточной функции. Мы возьмём нулевую сеточную функцию.

Остановка алгоритма производится по условию:

$$\|w^{(k+1)} - w^{(k)}\|_E < \delta,$$

где заданная константа $\delta = 10^{-6}$

Здесь скалярное произведение и норма определены на пространстве сеточных функций, заданных на введённой нами сетке, следующим образом:

$$\left(u, v\right) = \sum_{i=1}^{M-1} \sum_{j=1}^{N-1} h_1 h_2 u_{ij} v_{ij}, \quad \|u\|_E = \sqrt{(u, u)}.$$

Программная реализация

Программа начинается с чтения каждым процессом из общего коммуникатора своего номера и количества процессов. Далее каждый процесс определяет граничные индексы подобласти прямоугольника П, которую он должен обработать, и номера своих соседей с четырёх сторон путём бинарных разбиений подобластей (начиная со всей области - прямоугольника П) попеременно по горизонтали и по вертикали, при которых анализируются биты номера процесса от младшего до старшего, и от каждого бита зависит положение подобласти, отведённой процессу, относительно текущей прямой разбиения. На первом шаге вся область разбивается на две равные подобласти вертикалью y=0, и процессы с чётным номером (младший бит 0) обрабатывают подобласть из левой части, а с нечётным - из правой. На втором шаге две подобласти разбиваются пополам горизонталью x=1.5, и процессы со вторым справа битом 0 получают на счёт нижнюю подобласть (¼ прямоугольника П), а с битом 1 - верхнюю. Далее разбиение продолжается аналогичным образом (на каждом шаге в разбиении с предыдущего шага каждая подобласть делится пополам). Этот алгоритм разбивает П на N близких по площади подобластей и каждой из них назначается процесс, который будет производить вычисления в её пределах. Число N может быть произвольной степенью числа 2. Условие 4 для числа узлов в подобластях выполняется за счёт чередования горизонтального и вертикального разбиений при условии, что ему удовлетворяет исходное разбиение прямоугольника П.

MPI применяется в программе следующим образом:

- Каждый процесс производит вычисления в отведённой ему подобласти, а не на всём прямоугольнике П
- Перед применением оператора A к сеточной функции и невязке между процессами осуществляется обмен их значениями в граничных точках подобластей, осуществляемый с помощью средств MPI. Это необходимо, поскольку оператор A вычисляется со сдвигом элементов по сетке и для граничных элементов некоторой подобласти задевает другую подобласть, обсчитываемую другим процессом.
- При вычислении скалярных произведений (Ar,r), (Ar, Ar) и нормы разности текущей и предыдущей сеточной функции каждый процесс вычисляет локальную часть скалярного произведения, представляющего собой поэлементное произведение матриц с последующим суммированием всех элементов матрицы, после чего локальные части суммируются с помощью MPI_Allreduce. Полученные значения используются для расчёта итерационного параметра и проверки критерия остановки итерационного алгоритма в каждом процессе.

Все вычисления производятся на GPU. Размер блока нитей threads_per_block установлен равным 1024. Вычисления в пределах подобласти производятся путём вызова функций ядра, между которыми происходят MPI-обмены, в следующей последовательности:

- Вычисление невязки (step1)
- Копирование границ невязки с GPU на хост
- Обмен границами невязки между процессами (на хосте)
- Копирование полученных от соседних подобластей границ невязки с хоста на GPU
- Вписывание полученных от соседей границ в матрицу на GPU (fill_borders)

- Вычисление Ar (step2)
- Вычисление локальных скалярных произведений (Ar,r), (Ar, Ar) с помощью dot_reduce. В первом вызове функции каждый блок нитей вычисляет скалярное произведение части массивов, индексы которой соответствуют линейным индексам входящих в блок нитей. Во втором вызове в качестве массива для суммирования (умножение осуществляется лишь при первом суммировании) рассматриваются начальные элементы каждой из уже просуммированных частей массивов с индексами 0, threads_per_block, 2*threads_per_block и т.д., и каждый блок нитей суммирует threads_per_block таких элементов, то есть вычисляет сумму части массива длиной не более $threads\_per\_block^2$. Далее суммирование может быть продолжено по такому же принципу (на первой итерации каждый блок считает сумму части массива, на второй - сумму сумм частей, на третьей - сумму сумм сумм частей и т.д. по принципу бинарного дерева), но фактически с учётом фигурирующих в задаче размеров массивов для вычисления будет хватать двух вызовов ядра.

  Вызовы ядра для вычисления каждого из двух скалярных произведений осуществляются паралелльно в разных потоках команд CUDA, результаты сохраняются в отдельные буферы

- Копирование вычисленных локальных скалярных произведений на хост
- Суммирование локальных скалярных произведений с помощью MPI_Allreduce
- Вычисление новых значений сеточной функции и их разности со старыми (step3)
- Вычисление локальной части квадрата нормы разности с помощью dot_reduce
- Копирование локальной части квадрата нормы разности на хост
- Суммирование локальных частей с помощью MPI_Allreduce
- Копирование границ новой сеточной функции с GPU на хост
- Обмен границами сеточной функции между процессами (на хосте)
- Копирование полученных от соседних подобластей границ сеточной функции с хоста на GPU

Результаты расчётов

Были проведены вычисления на разных значениях числа процессов (1,2,4) для архитектур Kepler (sm_35) и Pascal (sm_60), вычислены ускорения относительно последовательной программы на сетке (40,40).

Время работы программы и обменов MPI измерялось с помощью функции стандартной библиотеки gettimeofday. Время работы паралелльных циклов на CUDA (вызовов ядер) и копирований между хостом и устройством вычислялось с помощью объектов cudaEvent_t отдельно от вычисления общего времени работы программы, отдельным запуском (поскольку накладные расходы на создание и уничтожение событий сильно замедляли программу в целом)

Оценка корректности паралелльной программы производилась путем расчета покомпонентной разности итоговой сеточной функции в паралелльной и последовательной программах с последующим усреднением всех элементов полученной матрицы. Было получено достаточно малое значение.

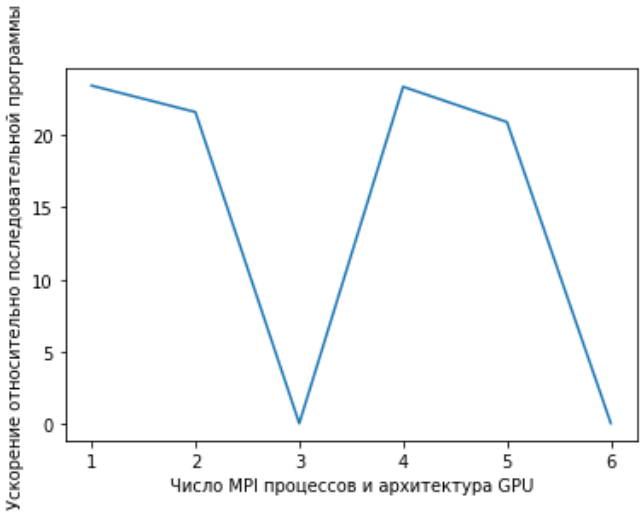
Для случая множества процессов в качестве времени работы выбиралось максимальное из всех процессов.

Архитектура GPU	Число MPI процессов	Число OpenMP-нитей	Число точек сетки M x N	Время решения, с	Ускорение относительно последовательной программы
Без CUDA	Без MPI	Без OpenMP (последовательная программа)	10 x 10	0.0024956	
Без CUDA	Без MPI	Без OpenMP (последовательная программа)	20 x 20	0.1502776	
Без CUDA	Без MPI	Без OpenMP (последовательная программа)	40 x 40	6.7815706	
Без CUDA	Без MPI	1	40 x 40	6.7821554	0.99991
Без CUDA	Без MPI	2	40 x 40	11.316978	0.5592
Без CUDA	Без MPI	2	80 x 80	100.91779	
Без CUDA	Без MPI	4	40 x 40	0.337126116	20.1158
Без CUDA	Без MPI	4	80 x 80	2.09664763	
Без CUDA	Без MPI	4	160 x 160	9.0053362775	
Без CUDA	Без MPI	8	40 x 40	0.753791276	8.9966
Без CUDA	Без MPI	8	80 x 80	2.188923054	
Без CUDA	Без MPI	8	160 x 160	9.0105074975	
Без CUDA	Без MPI	16	40 x 40	11.7042626	0.57941
Без CUDA	Без MPI	16	80 x 80	56.16378517	
Без CUDA	Без MPI	16	160 x 160	12.6444255	
Без CUDA	Без MPI	32	160 x 160	11.9983471	

Без CUDA	1	Без OpenMP	40 x 40	2.446649	2.772018
Без CUDA	2	Без OpenMP	40 x 40	1.587341	4.272651
Без CUDA	4	Без OpenMP	40 x 40	1.367768	4.958557
Без CUDA	1	4	40 x 40	0.351037	19.320343
Без CUDA	2	4	40 x 40	1.256924	5.395836
Без CUDA	2	1	80 x 80	131.295835	
Без CUDA	2	2	80 x 80	30.878037	
Без CUDA	2	4	80 x 80	1.040367	
Без CUDA	2	8	80 x 80	2.623257	
Без CUDA	4	1	80 x 80	129.106531	
Без CUDA	4	2	80 x 80	1.871344	
Без CUDA	4	4	80 x 80	2.062783	
Без CUDA	4	8	80 x 80	2.524404	
Без CUDA	4	1	160 x 160	150.673852	
Без CUDA	4	2	160 x 160	7.242101	
Без CUDA	4	4	160 x 160	8.271759	
Без CUDA	4	8	160 x 160	9.870419	
Kepler	1	Без OpenMP	40 x 40	0.290035	23.38392
Kepler	2	Без OpenMP	40 x 40	7.314653	21.554396
Kepler	4	Без OpenMP	40 x 40	132.864729	0.051046
Kepler	1	Без OpenMP	80 x 80	8.014897	
Kepler	1	Без OpenMP	160 x 160	9.001459	
Pascal	1	Без OpenMP	40 x 40	0.291034	23.303653
Pascal	2	Без OpenMP	40 x 40	9.324975	20.869776
Pascal	4	Без OpenMP	40 x 40	98.569132	0.068806
Pascal	1	Без OpenMP	80 x 80	7.935401	
Pascal	1	Без OpenMP	160 x 160	8.985792	

```
1 - 1 процесс, Kepler
2 - 2 процесса, Kepler
3 - 4 процесса, Kepler
4 - 1 процесс, Pascal
5 - 2 процесса, Pascal
6 - 4 процесса, Pascal

[<matplotlib.lines.Line2D at 0x277e5a478b0>]
```



Для 2 и 4 процессов были получены плохие результаты, что можно объяснить неоправданно большими затратами на обмены между процессами и сопутствующие копирования между хостом и GPU. Таким образом, оптимально использовать CUDA без MPI, чтобы избежать копирований границ областей.

Вычислим время выполнения отдельных секций (суммарное время операций каждого вида)

Архитектура GPU	Число MPI процессов	Число точек сетки M x N	Время инициализации программы, с	Время завершения программы, с	Время паралелльных циклов, с	Время копирований GPU, с	Время обменов MPI, с
Без CUDA	Без MPI	40 x 40	0.000326	0.000015	6.781225	0	0
Kepler	1	40 x 40	0.010456	0.000314	0.123881	0.154067	0
Kepler	2	40 x 40	0.051203	0.036573	2.340689	4.069873	0.804612
Kepler	4	40 x 40	0.037835	0.010445	42.649575	66.830954	14.482254
Kepler	1	80 x 80	0.040476	0.03249	3.17416	4.36447	0

Kepler	1	160 x 160	0.050956	0.000314	4.025356	4.670983	0
Pascal	1	40 x 40	0.011043	0.001043	0.131256	0.14639	0
Pascal	2	40 x 40	0.049924	0.029768	3.661166	5.408486	0.931999
Pascal	4	40 x 40	0.032012	0.024324	41.875026	56.155787	11.039742
Pascal	1	80 x 80	0.043456	0.031354	3.164506	4.36447	0
Pascal	1	160 x 160	0.051456	0.000794	4.127956	4.670983	0

В целях более корректного сравнения времени работы MPI и MPI+CUDA программ проведем дополнительное исследование на большой сетке: возьмем размер сетки 6000x6000 (тогда объемы массивов с сеточной функцией и невязкой будут примерно по 0.27 ГБ), проведем вычисление первых 20 шагов итерационного алгоритма на чистом MPI без GPU и на MPI+CUDA и посчитаем время. Проведем эксперимент отдельно для случая 1,2,4 процессов.

Программа	Общее время, с	Инициализация, с	Завершение, с	Копирования GPU, с	Обмены MPI, с	Циклы, с
MPI 1 proc	78.847178	0.347598	0.006508	0	0.000367	72.695
MPI 2 proc	147.1682	0.667079	0.01434713	0	0.000740	142.50242
MPI 4 proc	137.5322	0.625656	0.011618	0	0.000595	129.92352
MPI+CUDA 1 proc	0.572405	0.096073	0.002483	0.0009115	0.0001	0.469599
MPI+ CUDA 2 proc	1.074427	0.158838	0.004049	0.001687	0.000161	0.846483
MPI+ CUDA 4 proc	1.028943	0.172498	0.004830	0.001623	0.000185	0.83604

Измерения показывают, что добавление GPU при больших объемах данных дает выигрыш во времени.

График полученной сеточной функции на сетке 160 x 160: