Отчет по заданию №1

студента 620 группы Бугаевского Владимира.

Вариант 5

1 Постановка задачи и описание алгоритма

В данном варианте предлагалось исследовать масштабируемость алгоритма решения СЛАУ $Ax = b$ после LU-факторизации. Элементы матрицы A и вектора правой части b являются действительными числами.

Отметим, что матрица A должна быть не вырожденной для того, чтобы система была совместной при любой правой части b (см. т. Кронекера – Капелли). Для генерации такой матрицы воспользуемся алгоритмом, предложенным в условии задания:

1. сгенерируем диагональную матрицу D со случайными ненулевыми элементами;
2. домножим дигональную матрицу D симметричным двусторонним умножением на ортогональную матрицу Q: $A = Q'DQ$.

Напомним, что LU-разложение – представление матрицы в виде:

$$A = PLU \tag{1}$$

где P – перестановочная матрица, L – нижняя треугольная матрица с единичной диагональю, U – верхняя треугольная матрица с ненулевыми элементами на диагонали.

Для удобства, не ограничивая общности, будем полагать, что матрица P является единичной. Таким образом, задача сводится к последовательному решению двух СЛАУ с треугольными матрицами:

$$Ax = LUx = Ly = b$$

$$Ux = y \tag{2}$$

$$Ly = b \tag{3}$$

Рассмотрим алгоритм решения СЛАУ с верхней треугольной матрицей (2):

$$\begin{pmatrix} u_{11} & u_{12} & \dots & u_{1(n-1)} & u_{1n} \\ 0 & u_{22} & \dots & u_{2(n-1)} & u_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & u_{(n-1)(n-1)} & u_{(n-1)n} \\ 0 & 0 & \dots & 0 & u_{nn} \end{pmatrix} * \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix}$$

Удобно вычислить сначала x_n, затем x_{n-1} и т.д. Тогда легко видеть, что решение системы выглядит следующим образом:

$$x_n = y_n / u_{nn}$$

$$x_i = \left(y_i - \sum_{j=i+1}^{n} u_{ij} x_j \right) / u_{ii} \; ; i = \overline{1, n-1}$$

Алгоритм решения выглядит следующим образом:

Листинг 1: "Псевдокод алгоритма решения СЛАУ (2)"

```
for i in range(n, 1, -1):
    s = 0
    for j in range(n, i+1, -1):
        s = s + U[i, j] * x[j]
    x[i] = (y[i] - s) / U[i, i]
```

На рисунке 1 изображена информационная структура для задачи СЛАУ с верхней треугольной матрицей.

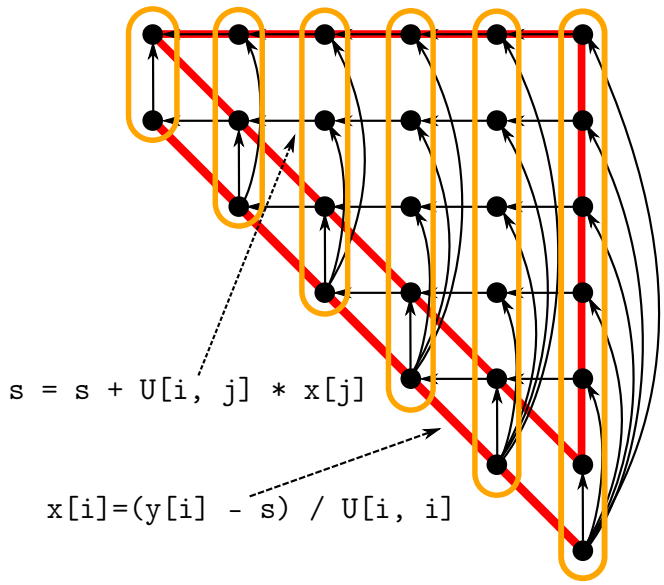

Рис. 1: Информационная структура для задачи (2)

Вычисление суммы s можно осуществлять паралелльно, а вычисление решения x_i – последовательно. Критический путь графа алгоритма имеет длину $o(N)$, поэтому алгоритм обладает хорошим ресурсом параллелизма.

Рассуждения для алгоритма решения СЛАУ с нижней треугольной матрицей (3) аналогичны. Таким образом решение СЛАУ после LU-факторизации потребует выполнения $O(N^2)$ операций.

2 Масштабируемость алгоритма

В данном задании требуется измерить масштабируемость реализации алгоритма, а именно функции `magma_sgetrs_gpu`.

N	**Итерация**					**mean**	**median**
	1	**2**	**3**	**4**	**5**		
32	0.00184	0.00219	0.00278	0.00103	0.00271	0.00211	0.00219
64	0.00234	0.00173	0.00125	0.00242	0.00171	0.00189	0.00173
128	0.00133	0.00170	0.00176	0.00101	0.00252	0.00166	0.00170
256	0.00205	0.00199	0.00159	0.00106	0.00169	0.00168	0.00169
512	0.00076	0.00097	0.00100	0.00078	0.00076	0.00085	0.00078
1024	0.00119	0.00156	0.00190	0.00122	0.00118	0.00141	0.00122
2048	0.00229	0.00170	0.00183	0.00152	0.00150	0.00177	0.00170
4096	0.00168	0.00182	0.00201	0.00209	0.00216	0.00195	0.00201
8912	0.00283	0.00216	0.00218	0.00219	0.00218	0.00231	0.00218
10240	0.00376	0.00422	0.00283	0.00284	0.00282	0.00329	0.00284

Таблица 1: Измерения работы функции `magma_sgetrs_gpu` в секундах

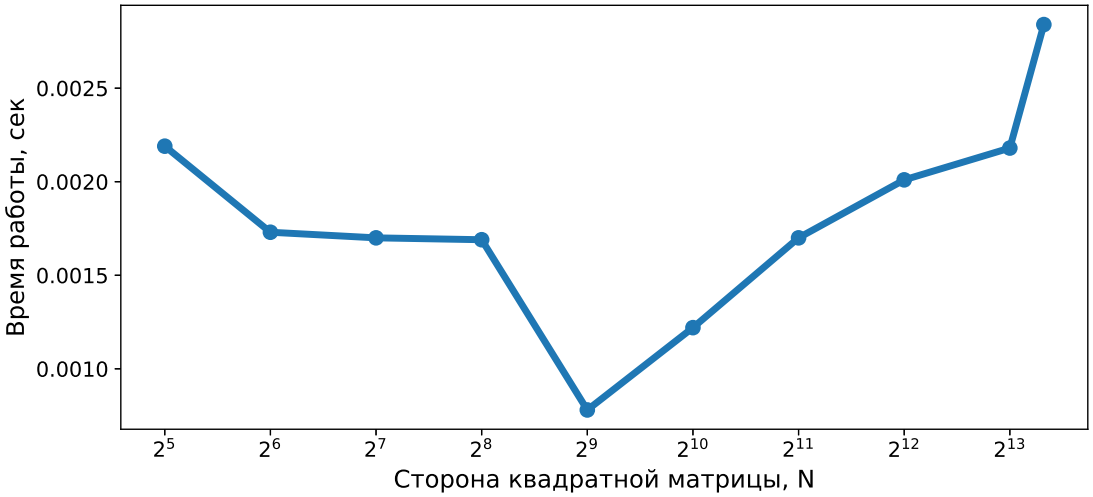

Рис. 2: Зависимость медианного времени работы функции от размеров матрицы

К сожалению, напрямую средствами библиотеки Magma измерить производительность суперкомпьютера не удалось. Однако, т.к. ассимптотика задачи известна, можно вычислить производительность с точностью до константы, поделив ассимптотическое число операций на среднее время работы алгоритма. Так как тип данных, предусмотренный в варианте, `float`, число операций по-прежнему остается $O(N^2)$.

N	32	64	128	256	512
mean(ts), s	0.00211	0.00189	0.00166	0.00168	0.00085
perfomance, Gflops/s	0.00049	0.00217	0.00985	0.03910	0.30696

N	1024	2048	4096	8912	10240
mean(ts), s	0.00141	0.00177	0.00195	0.00231	0.00329
perfomance, Gflops/s	0.74367	2.37234	8.59489	34.41237	31.83291

Таблица 2: Оценка производительности функции `magma_sgetrs_gpu` в Gflops/s

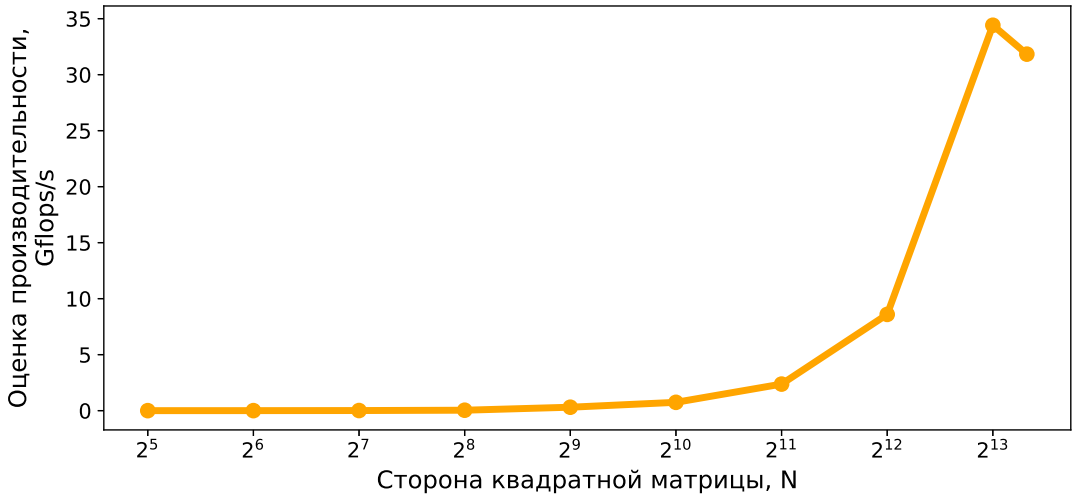

Рис. 3: Зависимость производительности работы функции от размеров матрицы

Результаты были получены на вычислительном комплексе Polus с использованием библиотек: Magma и АЛГЛИБ. На узлах суперкомпьютера Polus установлены по 2 графических ускорителя NVIDIA Tesla P100 GPU. Компиляция программ проходила с использованием компилятора `g++ 4.8.5` и оптимизацией `-O3`.

Из графика 2 видно, что при маленьких размерах матрицы система деградирует – время работы уменьшается, несмотря на увеличение размеров матрицы. Это связано с тем, что накладные расходы на пересылку данных и синхронизацию потоков выполнения превышают полезную нагрузку.

Из графика 3 видно, что при увеличении размеров матрицы производительность постепенно растет, достигает своего пика, а затем немного убывает и выходит на плато. Это связано с тем, что с определенного момента начинает увеличиваться количество обменов данными между RAM и DRAM, поэтому производительность падает.

При конфигурации, описанной выше, оптимальным значением выбрано: $N = 8912$.

3 Проверка корректности алгоритма

Исходные диагональная матрица D и вектор правой части b инициализируются случайными значениями из отрезка $[-5; 5]$.

Для проверки того, что полученный $\hat{x}$ действительно является решением x, подставим его в исходную систему:

$$A\hat{x} = \hat{b} \tag{4}$$

и рассмотрим норму невязки $\|b - \hat{b}\|_2$. Важно понимать, что при такой проверке ошибка достаточно сильно накапливается: первый раз – при решении СЛАУ (получении приближенного решения $\hat{x}$), второй раз – при вычислении $\hat{b}$.

N	**Итерация**					**mean**	**median**
	1	**2**	**3**	**4**	**5**		
32	0.00002	0.00002	0.00002	0.00003	0.00001	0.00002	0.00002
64	0.00004	0.00005	0.00003	0.00003	0.00008	0.00005	0.00004
128	0.00009	0.00008	0.00188	0.00023	0.00046	0.00055	0.00023
256	0.00196	0.00575	0.00160	0.00269	0.00174	0.00275	0.00196
512	0.00382	0.01699	0.00300	0.01847	0.00139	0.00873	0.00382
1024	0.05221	0.03051	0.01324	0.01660	0.00738	0.02399	0.01660
2048	0.08376	0.09140	0.05595	0.17878	0.06547	0.09507	0.08376
4096	0.23069	0.24875	0.21867	0.23858	0.22686	0.23271	0.23069
8912	1.63736	1.60046	1.48046	1.58926	1.74936	1.61138	1.60046
10240	3.21780	2.80067	3.28120	2.81901	2.84314	2.99234	2.84310

Таблица 3: Измерения l_2-нормы невязки

В таблице 3 видим, что значение нормы невязки увеличивается с ростом N, при этом норма принимает досаточно маленькие допустимые значения. В связи с этим можно считать, что алгоритм реализован корректно.