Индивидуальное задание
Вариант №10 “Решение СЛАУ методом Гаусса”

Постановка задачи

В данной работе необходимо:
1. Реализовать параллельную программу, решающую СЛАУ методом Гаусса, с использованием библиотеки MPI.
2. Реализовать последовательную программу решения СЛАУ.
3. Провести эксперименты по сравнению скорости работы для СЛАУ разного размера и сделать выводы.

Теоретическая справка:

Суть классического метода Гаусса заключается в следующем. Пусть в системе уравнений

$$\begin{aligned} &a_{11}^{(0)}x_1 + a_{12}^{(0)}x_2 + \ldots, + a_{1n}^{(0)}x_n = a_{1,n+1}^{(0)} \\ &a_{21}^{(0)}x_1 + a_{22}^{(0)}x_2 + \ldots, + a_{2n}^{(0)}x_n = a_{2,n+1}^{(0)} \\ &\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ &a_{n1}^{(0)}x_1 + a_{n2}^{(0)}x_2 + \ldots, + a_{nn}^{(0)}x_n = a_{n,n+1}^{(0)} \end{aligned} \qquad (1)$$

первый элемент $a_{11}^{(0)} \neq 0$ и назовем его ведущим элементом первой строки. Поделим все элементы этой строки на $a_{11}^{(0)}$ и исключим x_1 из всех последующих строк, начиная со второй, путем вычитания первой (преобразованной), умноженной на коэффициент при x_1 в соответствующей строке. Получим

$$\begin{aligned} &x_1 + a_{12}^{(1)}x_2 + \ldots, + a_{1n}^{(1)}x_n = a_{1,n+1}^{(1)} \\ &0 + a_{22}^{(1)}x_2 + \ldots, + a_{2n}^{(1)}x_n = a_{2,n+1}^{(1)} \\ &\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ &0 + a_{n2}^{(1)}x_2 + \ldots, + a_{nn}^{(1)}x_n = a_{n,n+1}^{(1)}. \end{aligned}$$

Если $a_{22}^{(1)} \neq 0$, то, продолжая аналогичное исключение, приходим к системе уравнений с верхней треугольной матрицей

$$\begin{aligned} &x_1 + a_{12}^{(1)}x_2 + \ldots, + a_{1n}^{(1)}x_n = a_{1,n+1}^{(1)} \\ &0 + x_2 + \ldots, + a_{2n}^{(2)}x_n = a_{2,n+1}^{(2)} \\ &0 + 0 + x_3 + \ldots, + a_{3n}^{(3)}x_n = a_{3,n+1}^{(3)} \\ &\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ &0 + 0 + \ldots \quad \ldots, + x_n = a_{n,n+1}^{(n)}. \end{aligned}$$

Далее из нее в обратном порядке аналогичным образом исключим с помощью последней строки все x_n во всех строках, расположенных выше последней. Продолжая этот процесс, получим на месте исходной матрицы единичную матрицу, а в столбце правых частей будут находиться значения искомых компонент решения x_i:

$$x_n = a^{(n)}_{n,n+1}$$
$$x_{n-1} = a^{(n-1)}_{n-1,n} - a^{(n-1)}_{n-2,n} x_n$$
$$\dots \quad \dots \quad \dots \quad \dots \quad \dots$$
$$x_1 = a^{(1)}_{1,n+1} - a^{(1)}_{1,2} x_2 - \dots - a^{(1)}_{1,n} x_n .$$

Процесс приведения исходной системы к системе с треугольной матрицей называется прямым ходом метода Гаусса, а нахождение неизвестных - обратным.

1) Постановка параллельного метода:

Рассмотрим один из вариантов организация параллельных вычислений при решении системы уравнений $A\vec{x} = \vec{y}$ методом Гаусса, где A – квадратная матрица размерности M, $\vec{x}$ – вектор-столбец неизвестных, $\vec{y}$ – вектор-столбец правых частей системы. К матрице A добавляется вектор правых частей и эта расширенная матрица $\overline{A}$

$$\overline{A} = \begin{pmatrix} a_{11}\ a_{12} \dots a_{1n}\ a_{1,n1} \\ a_{21}\ a_{22} \dots a_{21}\ a_{2,n+1} \\ \dots \quad \dots \quad \dots \quad \dots \\ a_{n1}\ a_{n2} \dots a_{nn}\ a_{n.n+1} \end{pmatrix}$$

разрезается на N полос равной ширины (для удобства выберем значение N кратным M), каждая из которых загружается в свой процессор. Далее в нулевом (активном) процессоре первая (ведущая) строка матрицы $\overline{A}$ делится на первый (ведущий) элемент и она передается всем остальным процессорам. Во всех процессорах с помощью этой строки элементарными преобразованиями все элементы первого столбца матрицы A (за исключением первой строки активного процессора) преобразовываются в нулевые. Затем в активном процессоре ведущей становится следующая строка, ведущим – ее первый ненулевой элемент и процесс продолжается до исчерпания строк в активном процессоре. После этого активным становится следующий (первый) процессор и все повторяется снова до тех пор, пока не исчерпаются все ведущие строки во всех процессорах. В результате в расширенной матрица $\overline{A}$ матрица A приведется к верхней треугольной матрице, распределенной по всем процессорам. На этом прямой ход метода Гаусса заканчивается. Далее активным становится (N-1) – й процессор и аналогичным образом организуется обратный ход, в результате которого матрица A приводится к единичной. На этом этапе каждая ведущая строка состоит только из одного ненулевого элемента и после приведения матрицы A к единичной на месте последнего столбца расширенной матрицы $\overline{A}$ оказываются значения вектора-столбца искомого решения $\vec{x}$.

Параллельный алгоритм состоит из прямого и обратного ходов.
На этапе прямого хода:

```
/* Цикл p - цикл по компьютерам. Все ветви, начиная с нулевой,
    последовательно приводят к
    диагональному виду свои строки. Ветвь, приводящая свои строки к диагональному
    виду,
    назовем активной, строка, с которой производятся вычисления, так же назовем
    активной. */

for(p = 0; p < size; p++)
```

```
    {
        /* Цикл k - цикл по строкам. (Все ветви "крутят" этот цикл). */
        for(k = 0; k < N; k++)
        {
            if(MyP == p)
            {
                /* Активная ветвь с номером MyP == p приводит свои строки к диагональному
                виду.
                Активная строка k передается ветвям, с номером большим чем MyP */
                MAD = 1.0/MA[k][N*p+k];
                for(j = M; j >= N*p+k; j--)
                    MA[k][j] = MA[k][j] * MAD;
                for(m = p+1; m < size; m++)
                    MPI_Send(&MA[k][0], M+1, MPI_DOUBLE, m, 1,MPI_COMM_WORLD);
                for(i = k+1; i < N; i++)
                {
                    for(j = M; j >= N*p+k; j--)
                        MA[i][j] = MA[i][j]-MA[i][N*p+k]*MA[k][j];
                }
            }
            /* Работа принимающих ветвей с номерами MyP > p */
            else if(MyP > p)
            {
                MPI_Recv(V, M+1, MPI_DOUBLE, p, 1,MPI_COMM_WORLD, &stat);
                    for(i = 0; i < N; i++)
                    {
                        for(j = M; j >= N*p+k; j--)
                            MA[i][j] = MA[i][j]-MA[i][N*p+k]*V[j];
                    }
            }
        }
    }
```

На этапе обратного хода:

```
/* Циклы по p и k аналогичны, как и при прямом ходе. */
    for(p = size-1; p >= 0; p--)
    {
        for(k = N-1; k >= 0; k--)
        {
            /* Работа активной ветви */
            if(MyP == p)
            {
                for(m = p-1; m >= 0; m--)
                    MPI_Send(&MA[k][M], 1, MPI_DOUBLE, m,1,MPI_COMM_WORLD);
                for(i = k-1; i >= 0; i--)
                    MA[i][M] -= MA[k][M]*MA[i][N*p+k];
            }
```

```
            /* Работа ветвей с номерами MyP < p */
            else
            {
                if(MyP < p)
                {
                    MPI_Recv(&R, 1, MPI_DOUBLE, p, 1,MPI_COMM_WORLD, &stat);
                    for(i = N-1; i >= 0; i--)
                        MA[i][M] -= R*MA[i][N*p+k];
                }
            }
        }
    }
```

2) Постановка последовательного метода

Решение задачи осуществляется с помощью функции solveSystem:

```
double* solveSystem(double** matrix, double* vector, int N) {
    /*Прямой ход */
    double forward;
    for (int i = 0; i < N; i++) {
        forward = matrix[i][i];   //Ведущий элемент строки
        for (int j = i; j < N; j++) {
            matrix[i][j] /= forward;  //Каждый элемент в строке делится на ведущий..
        }
        vector[i] /= forward; //..и правый столбец тоже
        for (int j = i + 1; j < N; j++) {
            forward = matrix[j][i];   //выбираем новый ведущий элемент
            for (int k = i; k < N; k++)
                matrix[j][k] -= forward * matrix[i][k]; //"исключаем" старый ведущий элемент
из системы
            vector[j] -= forward * vector[i];
        } //На выходе получаем верхнетреугольную матрицу
    }
    /*Обратный ход*/
    double* answer = new double[N];
    answer[N - 1] = vector[N - 1];
    for (int i = N - 2; i >= 0; i--) {
        answer[i] = vector[i];  //Крайний элемент ведущий
        for (int j = i + 1; j < N; j++) {
            answer[i] -= matrix[i][j] * answer[j]; //Исключаем крайний элемент из всех
строк, выше текущей
        }
    } //На выходе имеем матрицу E, у к-ой в правой части искомые x
    return answer;
}
```

3) Сравнение скорости работы двух алгоритмов для разного количества потоков и различных размеров исходной матрицы

Порядок матрицы (N)	Последовательный алгоритм	Параллельный алгоритм (2 потока)	Параллельный алгоритм (4 потока)	Параллельный алгоритм (8 потоков)
160	0.008223	0.001204	0.003873	0.003136
640	0.416630	0.338770	0.255397	0.144902
1600	6.269390	4.954079	2.815972	2.088334
2400	23.614288	17.723499	9.943506	7.756652

В приведенной выше таблице время каждого опыта измеряется в секундах. Также стоит отметить, что каждое измерение является усредненным значением, с количеством опытов для каждого случая, равным 10.

Исходя из результатов, стоит отметить, что оптимальным для реализации задачи решения СЛАУ методом Гаусса является параллельный алгоритм, использующий 8 потоков. Однако, в тех случаях, когда порядок исходной матрицы невелик, например, при N=160 преимущественен параллельный алгоритм с 2-мя потоками или в отдельных случаях (опытах) даже последовательный. То есть, можем заключить, что наибольшую выгоду из параллельного алгоритма с 8-ю потоками мы можем извлечь при больших N.

Характеристики используемого устройства:

```
gitpod /workspace/SupercomputersPMI_2018/MPI $ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   46 bits physical, 48 bits virtual
CPU(s):                          16
On-line CPU(s) list:             0-15
Thread(s) per core:              2
Core(s) per socket:              8
```