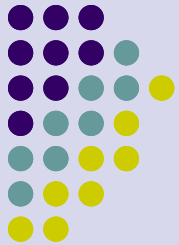
Использование модифицированного преобразования Хаусхолдера для выполнения собственного и сингулярного разложений на параллельных вычислительных системах с общей памятью



Авторы: А.Е.Андреев, В.А.Егунов

Solving of Eigenvalue and Singular Value Problems via Modified Householder Transformations on Shared Memory Parallel Computing Systems

Докладчик: В.А.Егунов

Суперкомпьютерные дни в России
Москва, 23-24 сентября 2019 г.

Постановка задачи

Собственное разложение

$$Av = \lambda v$$

$$A_{k+1} = U_k A_k U_k^*$$

Сингулярное разложение

$$Av = \lambda u$$

$$A^*u = \lambda v$$

$$A_{k+1} = U_k A_k V_k^*$$

Ускорение процесса вычислений

Хессенбергова форма

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1,n-2} & a_{1,n-1} & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2,n-2} & a_{2,n-1} & a_{2n} \\ 0 & a_{32} & a_{33} & \cdots & a_{3,n-2} & a_{3,n-1} & a_{3n} \\ 0 & 0 & a_{43} & \cdots & a_{4,n-2} & a_{4,n-1} & a_{4n} \\ \hline 0 & 0 & 0 & \cdots & 0 & a_{n,n-1} & a_{nn} \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{12} & 0 & \cdots & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & \cdots & 0 & 0 & 0 \\ \hline a_{n-2,1} & a_{n-2,2} & a_{n-2,3} & \cdots & a_{n-2,n-2} & a_{n-2,n-1} & 0 \\ a_{n-1,1} & a_{n-1,2} & a_{n-1,3} & \cdots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n-2} & a_{n,n-1} & a_{nn} \end{bmatrix}$$

Двухдиагональная форма

$$\begin{bmatrix} a_{11} & a_{12} & 0 & \cdots & 0 & 0 & 0 \\ 0 & a_{22} & a_{23} & \cdots & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & \cdots & a_{n-2,n-2} & a_{n-2,n-1} & 0 \\ 0 & 0 & 0 & \cdots & 0 & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & 0 & \cdots & 0 & 0 & a_{nn} \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & 0 & 0 & \cdots & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & \cdots & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & \cdots & 0 & 0 & 0 \\ 0 & 0 & a_{43} & \cdots & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & \cdots & 0 & a_{n,n-1} & a_{nn} \end{bmatrix}$$

Отражение Хаусхолдера

$$A = QRQ^T$$

$$s_k = -sign(a_{kk})\left(\sum_{i=k}^{n} a_{ik}^2\right)^{\frac{1}{2}},$$

$$\varphi_k = \left(s_k^2 - s_k a_{kk}\right)^{-1}$$

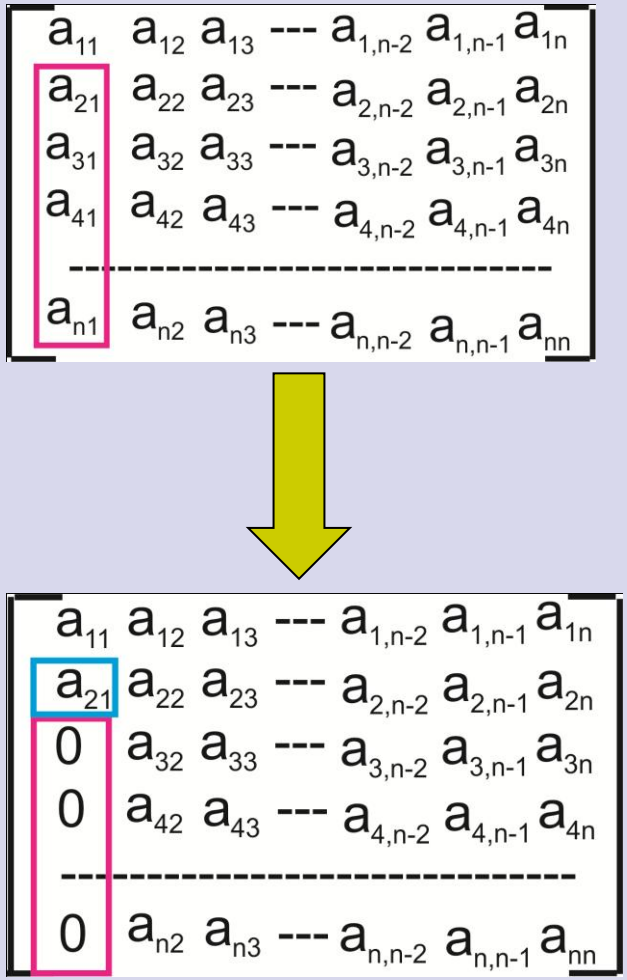
$$\boldsymbol{u}_k^T = \left(0, \dots, 0, a_{kk} - s_k, a_{k+1,k}, \dots, a_{nk}\right)$$

$$a_{kk} = s_k$$

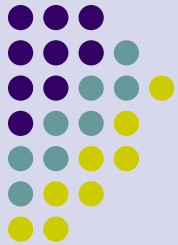
Для = k+1 до n

$$\lambda_j = \varphi_k \boldsymbol{u}_k^T \boldsymbol{a}_j$$

$$\boldsymbol{a}_j = \boldsymbol{a}_j - \lambda_j \boldsymbol{u}_k$$



Приведение матрицы произвольного вида к Хессенберговой форме



$$s_k = -sign(a_{k+1,k})\left(\sum_{i=k+1}^{n} a_{ik}^2\right)^{\frac{1}{2}},$$

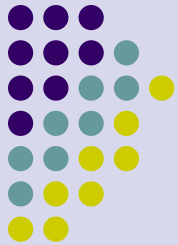
$$\varphi_k = \left(s_k^2 - s_k a_{k+1,k}\right)^{-1}$$

$$\boldsymbol{u}_k^T = \left(0, \dots, 0, a_{k+1,k} - s_k, a_{k+2,k}, \dots, a_{nk}\right)$$

$$\left.\begin{aligned} \lambda_j &= \varphi_k \boldsymbol{u}_k^T \boldsymbol{a}_j \\ \boldsymbol{a}_j &= \boldsymbol{a}_j - \lambda_j \boldsymbol{u}_k \end{aligned}\right| j = \overline{k,n}$$

$$\left.\begin{aligned} \lambda'_j &= \varphi_k \boldsymbol{b}_j \boldsymbol{u}_k \\ \boldsymbol{b}_j &= \boldsymbol{b}_j - \lambda'_j \boldsymbol{u}_k^T \end{aligned}\right| j = \overline{k,n}$$

Приведение матрицы произвольного вида к двухдиагональной форме



$$s_k = -sign(a_{k,k})\left(\sum_{i=k}^{n} a_{ik}^2\right)^{\frac{1}{2}},$$

$$\varphi_k = \left(s_k^2 - s_k a_{k,k}\right)^{-1}$$

$$\boldsymbol{u}_k^T = (0, \dots, 0, a_{k,k} - s_k, a_{k+1,k}, \dots, a_{nk})$$

$$\left.\begin{aligned} \lambda_j &= \varphi_k \boldsymbol{u}_k^T \boldsymbol{a}_j \\ \boldsymbol{a}_j &= \boldsymbol{a}_j - \lambda_j \boldsymbol{u}_k \end{aligned}\right| j = \overline{k,n}$$

$$\left.\begin{aligned} s'_k &= -sign(a_{k,k+1})\left(\sum_{i=k+1}^{n} a_{ki}^2\right)^{\frac{1}{2}}, \\ \varphi'_k &= \left(s_k'^2 - s'_k a_{k,k+1}\right)^{-1} \\ \boldsymbol{u}'^T_k &= (0, \dots, 0, a_{k,k+1} - s'_k, a_{k,k+2}, \dots, a_{kn}) \\ \lambda'_j &= \varphi_k \boldsymbol{b}_j \boldsymbol{u}'_k \\ \boldsymbol{b}_j &= \boldsymbol{b}_j - \lambda'_j \boldsymbol{u}'^T_k \end{aligned}\right| j = \overline{k,n} \Bigg\| \; k < n-1$$

«Узкие» места алгоритмов

$$s_k = -sign(a_{k+1,k})\left(\sum_{i=k+1}^{n} a_{ik}^2\right)^{\frac{1}{2}},$$

$$\varphi_k = \left(s_k^2 - s_k a_{k+1,k}\right)^{-1}$$

$$\boldsymbol{u}_k^T = \left(0, \dots, 0, a_{k+1,k} - s_k, a_{k+2,k}, \dots, a_{nk}\right)$$

1 этап

элементы вектора отражения

последовательное выполнение

!!! синхронизация

$$\left.\begin{aligned} \lambda_j &= \varphi_k \boldsymbol{u}_k^T \boldsymbol{a}_j \\ \boldsymbol{a}_j &= \boldsymbol{a}_j - \lambda_j \boldsymbol{u}_k \end{aligned}\right| j = \overline{k,n}$$

2 этап

обработка столбцов

неэффективная реализация

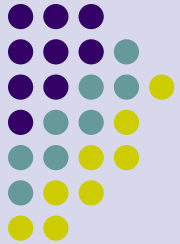
!!! синхронизация

$$\left.\begin{aligned} \lambda_j' &= \varphi_k \boldsymbol{b}_j \boldsymbol{u}_k \\ \boldsymbol{b}_j &= \boldsymbol{b}_j - \lambda_j' \boldsymbol{u}_k^T \end{aligned}\right| j = \overline{k,n}$$

3 этап

обработка строк

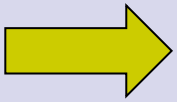
«Узкие» места алгоритмов



- обработка данных вдоль столбцов матрицы, что существенно снижает эффективность по сравнению с вариантом вычислений, основанном на обработке строк матрицы;
- последовательный этап вычисления элементов вектора отражения;

Ускорение обработки столбцов матриц

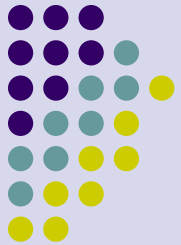
- определяются значения скалярных произведений столбцов и вектора отражения;
- на основе полученных значений скалярных произведений вычисляются новые значения элементов столбцов.



- на первом этапе формируется матрица специального вида;

- на втором этапе вычисляются новые значения элементов столбцов, обработка всех элементов матрицы можно вести параллельно,

Ускорение обработки столбцов матриц



$$\boldsymbol{sc} = \varphi_k \boldsymbol{u}_k^T \boldsymbol{A}$$

$$\boldsymbol{D} = diag(sc_0, sc_1, \dots, sc_n)$$

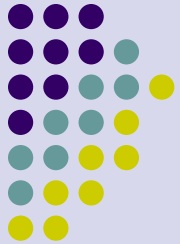
$$\boldsymbol{U}[:j] = [\boldsymbol{u_k}]$$

$$\boldsymbol{SCU} = \boldsymbol{UD}$$

$$\boldsymbol{b_j} = \boldsymbol{b_j} - \boldsymbol{scu_j} \Big| j = \overline{k,n}$$

$$\left.\begin{aligned} \lambda'_j &= \varphi_k \boldsymbol{b_j} \boldsymbol{u_k} \\ \boldsymbol{b_j} &= \boldsymbol{b_j} - \lambda'_j \boldsymbol{u_k^T} \end{aligned}\right| j = \overline{k,n}$$

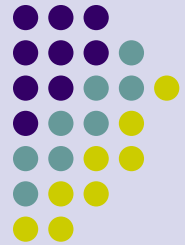
Снижение доли последова-тельных вычислений



- в базовом варианте алгоритма, а также в модификациях, рассмотренных выше, элементы вектора отражения формируются в последовательной части программы;

- перенос этапа определения элементов вектора отражения из последовательной части вычислений в параллельную;

Снижение доли последовательных вычислений



$$s_{k\mid j+1} = 0$$

$$\left.\begin{aligned} \lambda_j &= \varphi_k \boldsymbol{u_k^T a_j} \\ \boldsymbol{a_j} &= \boldsymbol{a_j} - \lambda_j \boldsymbol{u_k} \end{aligned}\right| \; j = \overline{k,n}$$

$$\left.\begin{aligned} \lambda'_j &= \varphi_k \boldsymbol{b_j u_k} \\ \boldsymbol{b_j} &= \boldsymbol{b_j} - \lambda'_j \boldsymbol{u_k^T} \\ \boldsymbol{u_{k+1}^T}[j] &= \boldsymbol{b_j}[k+1] \mid \; j > k+2 \\ s_{k\mid j+1} &= s_{k\mid j+1} + \boldsymbol{b_j}[k+1]^2 \mid \; j \geq k+2 \end{aligned}\right| \; j = \overline{k,n}$$

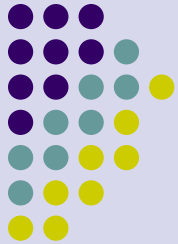
$$s_{k\mid j+1} = -\,sign\left(a_{k+2,k+1}\right) s_{k\mid j+1}{}^{\frac{1}{2}},$$

$$\varphi_{k\mid j+1} = \left(s^2_{k\mid j+1} - s_{k\mid j+1} a_{k+2,k+1}\right)^{-1}$$

$$\boldsymbol{u_{k+1}^T}[0{:}\,(\mathrm{k}+1)] = 0$$

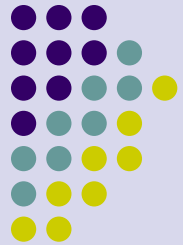
$$\boldsymbol{u_{k+1}^T}[\mathrm{k}+2] = a_{k+2,k+1} - s_{k\mid j+1}$$

Снижение доли последовательных вычислений



```
#pragma omp parallel for
for(int row = step; row < N; row++){
        double scalar = 0;
        for(int i = (step + shift); i < N; i++)
                scalar += matr[row * N + i] * vect[i];
        scalar *= gamma;
        for(int i = (step + shift); i < N; i++){
                matr[row * N + i] -= scalar * vect[i];
                / !!! БОЛЬШОЕ ЧИСЛО ПРОВЕРОК !!!
                if (i == (step + shift)){
                        if (row > (step + 1 + shift))
                                vectAdd[row] = matr[row * N + i];
                        if (row >= (step + 1 + shift))
#pragma omp atomic
                                s += matr[row * N + i] * matr[row * N + i];
                }
        }
}
// Корректировка …..
```

Снижение доли последова-тельных вычислений



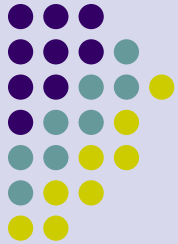
-разворачиваем цикл;

-убираем проверку if (row > (step + 1 + shift)); вектор отражения скорректируем в последовательной части программы;

-убираем проверку if (row >= (step + 1 + shift)); значение модуля скорректируем в последовательной части программы;

- убираем синхронизацию #pragma omp atomic, каждый поток накапливает модуль в своей переменной;

Снижение доли последова-тельных вычислений



```
#pragma omp parallel for
for(int row = step; row < N; row++){
        double scalar = 0;
        for(int i = (step + shift); i < N; i++)
                scalar += matr[row * N + i] * vect[i];
        scalar *= gamma;
        matr[row * N + step + shift] -= scalar * vect[step + shift];
        vectAdd[row] = matr[row * N + step + shift];
        sAr[numT] += matr[row * N + step + shift] * matr[row * N + step +
shift];
        for(int i = (step + shift + 1); i < N; i++)
                matr[row * N + i] -= scalar * vect[i];
}
```

Снижение доли последовательных вычислений

$$s_{k\mid j+1} = 0$$

$$\left.\begin{aligned} \lambda_j &= \varphi_k \boldsymbol{u_k^T a_j} \\ \boldsymbol{a_j} &= \boldsymbol{a_j} - \lambda_j \boldsymbol{u_k} \end{aligned}\right| \; j = \overline{k,n}$$

$$\left.\begin{aligned} \lambda'_j &= \varphi_k \boldsymbol{b_j u_k} \\ \boldsymbol{b_j} &= \boldsymbol{b_j} - \lambda'_j \boldsymbol{u_k^T} \\ \boldsymbol{u_{k+1}^T}[j] &= \boldsymbol{b_j}[k+1] \\ s_{k\mid j+1} &= s_{k\mid j+1} + \boldsymbol{b_j}[k+1]^2 \end{aligned}\right| \; j = \overline{k,n}$$

$$s_{k\mid j+1} = s_{k\mid j+1} - \mathbf{A}[k,k+1]^2 - \mathbf{A}[k+1,k+1]^2$$

$$s_{k\mid j+1} = -sign\left(a_{k+2,k+1}\right) s_{k\mid j+1}^{\frac{1}{2}},$$

$$\varphi_{k\mid j+1} = \left(s^2_{k\mid j+1} - s_{k\mid j+1} a_{k+2,k+1}\right)^{-1}$$

$$\boldsymbol{u_{k+1}^T}[0:(k+1)] = 0$$

$$\boldsymbol{u_{k+1}^T}[k+2] = a_{k+2,k+1} - s_{k\mid j+1}$$

Результирующий алгоритм

$$s_{k\mid j+1} = 0$$

$$\boldsymbol{sc} = \varphi_k \boldsymbol{u}_k^T \boldsymbol{A} \quad \boldsymbol{D} = diag(sc_0, sc_1, \dots, sc_n) \quad \boldsymbol{U}[:j] = [\boldsymbol{u_k}]$$

$$\boldsymbol{SCU} = \boldsymbol{UD}$$

$$\boldsymbol{b_j} = \boldsymbol{b_j} - \boldsymbol{scu_j} \Big| j = \overline{k,n}$$

$$\left.\begin{aligned} \lambda'_j &= \varphi_k \boldsymbol{b_j} \boldsymbol{u_k} \\ \boldsymbol{b_j} &= \boldsymbol{b_j} - \lambda'_j \boldsymbol{u_k^T} \\ \boldsymbol{u_{k+1}^T}[j] &= \boldsymbol{b_j}[k+1] \\ s_{k\mid j+1} &= s_{k\mid j+1} + \boldsymbol{b_j}[k+1]^2 \end{aligned}\right| j = \overline{k,n}$$

$$s_{k\mid j+1} = s_{k\mid j+1} - \mathbf{A}[k,k+1]^2 - \mathbf{A}[k+1,k+1]^2$$

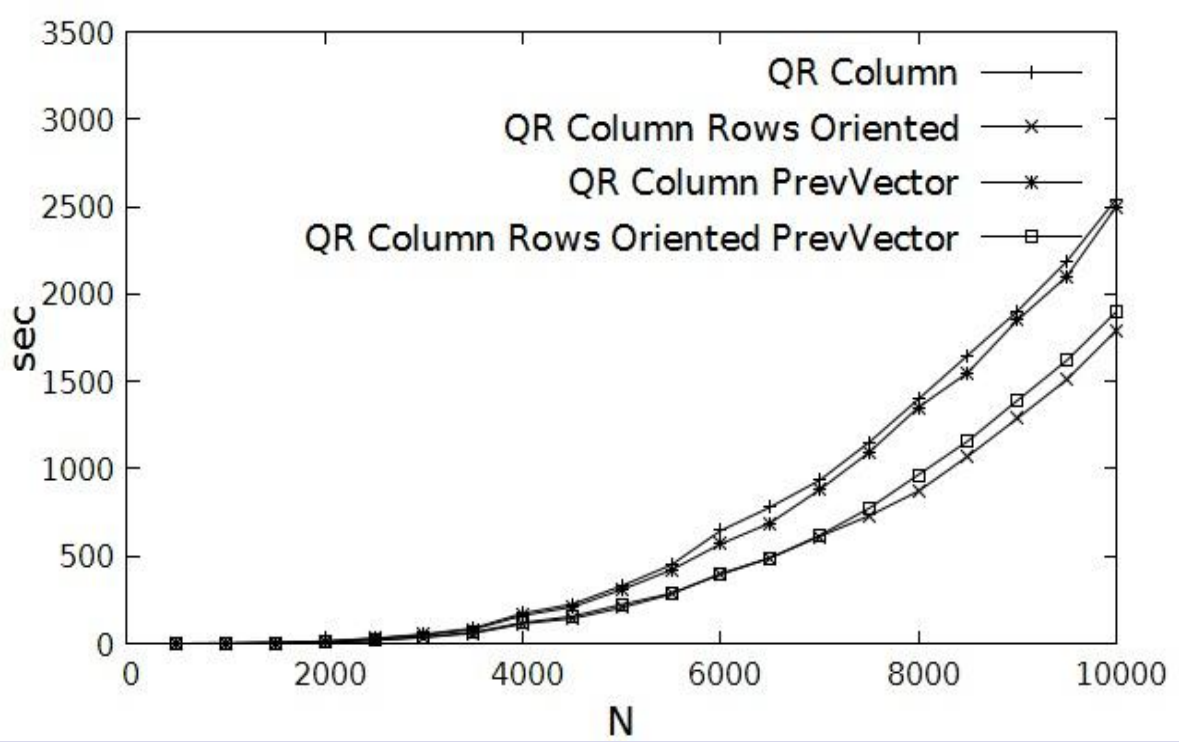
$$s_{k\mid j+1} = -sign(a_{k+2,k+1}) s_{k\mid j+1}{}^{\frac{1}{2}},$$

$$\varphi_{k\mid j+1} = \left(s^2_{k\mid j+1} - s_{k\mid j+1} a_{k+2,k+1}\right)^{-1}$$

$$\boldsymbol{u_{k+1}^T}[0:(k+1)] = 0$$

$$\boldsymbol{u_{k+1}^T}[k+2] = a_{k+2,k+1} - s_{k\mid j+1}$$

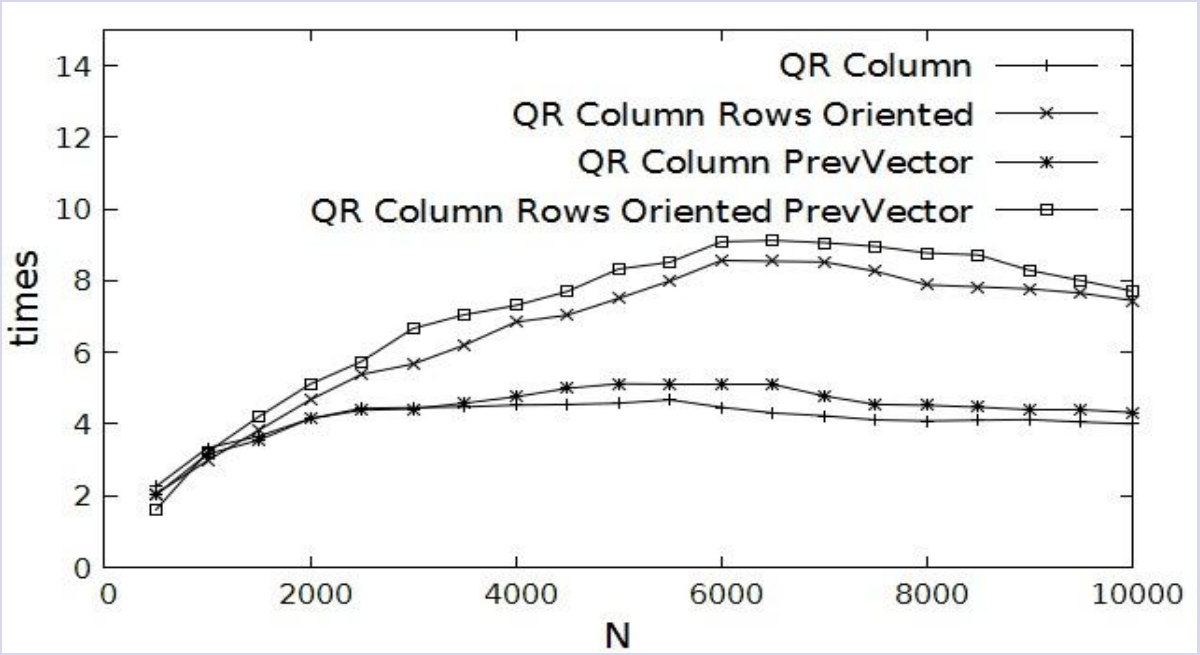
Время работы последовательных вариантов программ приведения матрицы к верхней Хессенберговой форме



- наименее эффективна реализация классического преобразования;

- самым быстрым является вариант с использованием строчно – ориентированный схемы;

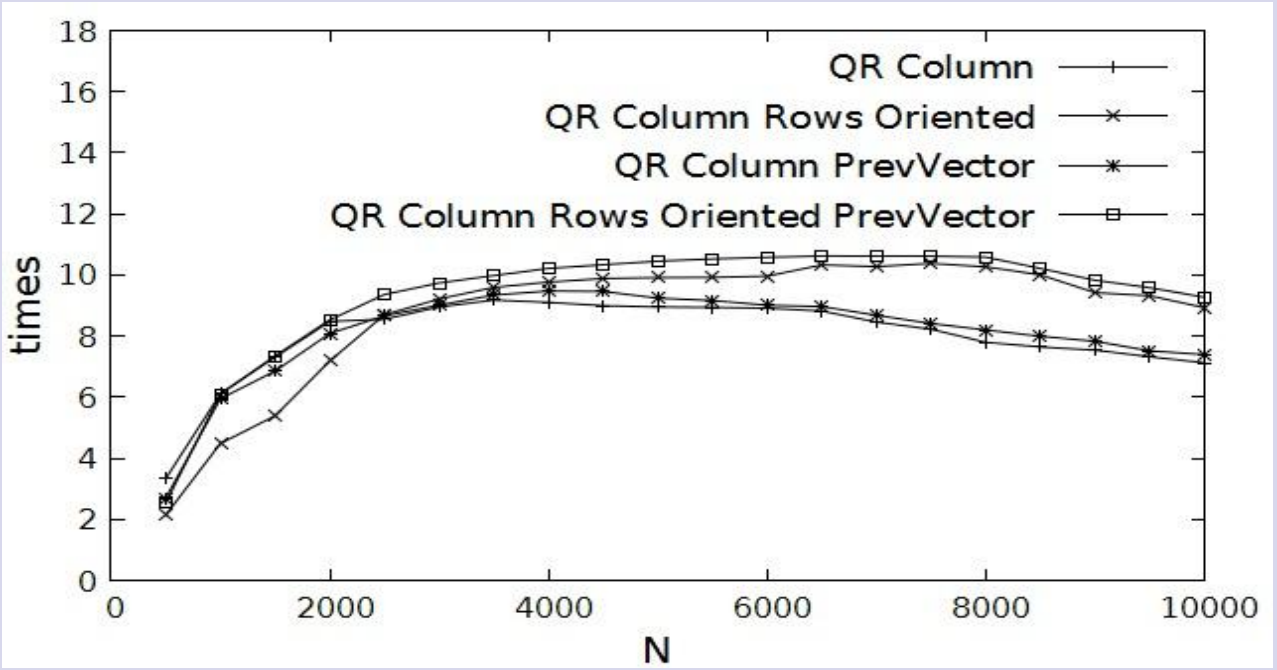
Значения ускорения параллельных вариантов программ приведения матрицы к верхней Хессенберговой форме по отношению к базовому алгоритму, число потоков равно 5



- лучшим оказывается вариант, в котором использовались обе модификации;

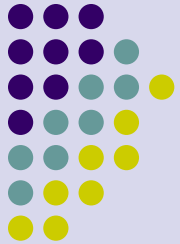
- на 5 параллельных потоках работа базового варианта алгоритма была ускорена более, чем в 9 раз;

Значения ускорения параллельных вариантов программ приведения матрицы к верхней Хессенберговой форме по отношению к базовому алгоритму, число потоков равно 10



- предложенные модификации базового алгоритма обладают лучшими показателями ускорения, однако, их эффективность у увеличением числа параллельных потоков несколько снизилась;

Дальнейшая оптимизация



-векторизация вычислений…

Исследование выполнено при финансовой поддержке РФФИ и Администрации Волгоградской области в рамках научного проекта № 18-47-340010 р_а.