Спецкурс кафедры «Вычислительной математики»

Параллельные алгоритмы вычислительной алгебры

Александр Калинкин

Сергей Гололобов

Часть 5: Разделение переменных

- Разделение переменных для оператора Лапласа
- Вычислительный алгоритм
- OMP и MPI реализация

Сеточная аппроксимация оператора Лапласа

$$\begin{cases} -\dfrac{\partial^2 u}{\partial x^2} - \dfrac{\partial^2 u}{\partial y^2} = f; \quad 0 < x < 1; \quad 0 < y < 1; \\ u_{x=0} = u_{x=1} = u_{y=0} = u_{y=1} = 0; \end{cases}$$

Кусочно-линейная аппроксимация, квадратная сетка

$$\frac{1}{h^2}\begin{bmatrix}
4 & -1 & & & -1 & & & & & & & \\
-1 & 4 & -1 & & & -1 & & & & & & \\
 & -1 & 4 & -1 & & & -1 & & & & & \\
 & & -1 & 4 & & & & -1 & & & & \\
-1 & & & & 4 & -1 & & & -1 & & & \\
 & -1 & & & -1 & 4 & -1 & & & -1 & & \\
 & & -1 & & & -1 & 4 & -1 & & & -1 & \\
 & & & -1 & & & -1 & 4 & & & & -1 \\
 & & & & -1 & & & & 4 & -1 & & \\
 & & & & & -1 & & & -1 & 4 & -1 & \\
 & & & & & & -1 & & & -1 & 4 & -1 \\
 & & & & & & & -1 & & & -1 & 4
\end{bmatrix} (u)_i = (f)_i$$

Сеточная аппроксимация оператора Лапласа

Как найти собственные вектора и собственные значения матрицы A?

$$\frac{1}{h^2}\begin{bmatrix}
4 & -1 & & & -1 & & & & & & & & & & & \\
-1 & 4 & -1 & & & -1 & & & & & & & & & & \\
 & -1 & 4 & -1 & & & -1 & & & & & & & & & \\
 & & -1 & 4 & & & & -1 & & & & & & & & \\
-1 & & & & 4 & -1 & & & -1 & & & & & & & \\
 & -1 & & & -1 & 4 & -1 & & & -1 & & & & & & \\
 & & -1 & & & -1 & 4 & -1 & & & -1 & & & & & \\
 & & & -1 & & & -1 & 4 & & & & -1 & & & & \\
 & & & & -1 & & & & 4 & -1 & & & -1 & & & \\
 & & & & & -1 & & & -1 & 4 & -1 & & & -1 & & \\
 & & & & & & -1 & & & -1 & 4 & -1 & & & -1 & \\
 & & & & & & & -1 & & & -1 & 4 & & & & -1 \\
 & & & & & & & & -1 & & & & 4 & -1 & & \\
 & & & & & & & & & -1 & & & -1 & 4 & -1 & \\
 & & & & & & & & & & -1 & & & -1 & 4 & -1 \\
 & & & & & & & & & & & -1 & & & -1 & 4
\end{bmatrix} = A$$

Базисные функции оператора Лапласа

Пусть $\mu_k(i,j) = \mu_{k_1}^{(1)}(i) * \mu_{k_2}^{(2)}(j)$ - собственный вектор матрицы A.

Тогда оказывается:

$$\lambda_{k_\alpha}^{\alpha} = \frac{4}{h_\alpha^2}\sin^2\left(\frac{k_\alpha \pi h}{2}\right), \qquad k_\alpha = 1,2,\ldots, N-1$$

$$\mu_{k_1}^{(1)}(i) = \sqrt{2}\sin\left(\frac{k_1 \pi i}{N}\right), \qquad k_1 = 1,2,\ldots, N-1$$

$$\mu_{k_2}^{(2)}(j) = \sqrt{2}\sin\left(\frac{k_2 \pi j}{N}\right), \quad k_2 = 1,2,\ldots, N-1$$

Следовательно для нашей матрицы A:

$$\mu_k(i,j) = 2\sin\left(\frac{k_1 \pi i}{N}\right)\sin\left(\frac{k_1 \pi j}{N}\right); \qquad 0 < i, j < N$$

$$\lambda_k = \lambda_{k_1}^{(1)} + \lambda_{k_2}^{(2)} = \sum_{\alpha=1}^{2}\frac{4}{h^2}\sin^2\left(\frac{k_\alpha \pi h}{2}\right)$$

Базисные функции оператора Лапласа

Разложим теперь решение задачи и правую часть по базисным функциям матрицы A

$$u(i,j)=\sum_{k_1=1}^{N_1-1}\sum_{k_2=1}^{N_2-1}u_{k_1k_2}\mu_{k_1}^{(1)}(i)\mu_{k_2}^{(2)}(j)$$

$$f(i,j)=\sum_{k_1=1}^{N_1-1}\sum_{k_2=1}^{N_2-1}f_{k_1k_2}\mu_{k_1}^{(1)}(i)\mu_{k_2}^{(2)}(j)$$

Подставим полученные выражения в изначальное уравнение получим и учтя собственные значения матрицы A:

$$\sum_{k_1=1}^{N_1-1}\sum_{k_2=1}^{N_2-1}\left(\lambda_{k_1}^{(1)}+\lambda_{k_2}^{(2)}\right)u_{k_1k_2}\mu_{k_1}^{(1)}(i)\mu_{k_2}^{(2)}(j)=\sum_{k_1=1}^{N_1-1}\sum_{k_2=1}^{N_2-1}f_{k_1k_2}\mu_{k_1}^{(1)}(i)\mu_{k_2}^{(2)}(j)$$

$$u_{k_1k_2}=\frac{f_{k_1k_2}}{\lambda_{k_1}^{(1)}+\lambda_{k_2}^{(2)}};\ 1\le k_1\le N-1,\ 1\le k_2\le N-1$$

$$u(i,j)=\sum_{k_1=1}^{N_1-1}\sum_{k_2=1}^{N_2-1}\left(\frac{f_{k_1k_2}}{\lambda_{k_1}^{(1)}+\lambda_{k_2}^{(2)}}\right)u_{k_1k_2}\mu_{k_1}^{(1)}(i)\mu_{k_2}^{(2)}(j);\ 0\le i,j\le N$$

Вычислительный алгоритм (двойной ряд)

$$\varphi_{k_2}(i) = \sum_{j=1}^{N-1} f(i,j)\sin\left(\frac{k_2 \pi j}{N}\right); \qquad 1 \le i, k_2 \le N-1;$$

$$\varphi_{k_1 k_2} = \sum_{i=1}^{N-1} \varphi_{k_2}(i)\sin\left(\frac{k_1 \pi i}{N}\right); \qquad 1 \le k_1, k_2 \le N-1;$$

$$u_{k_2}(i) = \sum_{k_1=1}^{N_1-1}\left(\frac{\varphi_{k_1 k_2}}{\lambda_{k_1}^{(1)} + \lambda_{k_2}^{(2)}}\right)\sin\left(\frac{k_1 \pi i}{N}\right); \quad 1 \le i, k_2 \le N-1;$$

$$u(i,j) = \frac{4}{N^2}\sum_{k_2=1}^{N-1} u_{k_2}(i)\sin\left(\frac{k_2 \pi j}{N}\right); \qquad 1 \le i, j \le N-1;$$

$$y_k = \sum_{j=1}^{N-1} a_i \sin\left(\frac{k \pi i}{n}\right);$$ - Тригонометрическое разложение, трудоемкость C*N*log(N)

Вычислительный алгоритм (двойной ряд)

$$\varphi_{k_2}(i) = \sum_{j=1}^{N-1} f(i,j)\sin\left(\frac{k_2 \pi j}{N}\right); \qquad 1 \le i, k_2 \le N-1;$$

$$\varphi_{k_1 k_2} = \sum_{i=1}^{N-1} \varphi_{k_2}(i)\sin\left(\frac{k_1 \pi i}{N}\right); \qquad 1 \le k_1, k_2 \le N-1;$$

$$u_{k_2}(i) = \sum_{k_1=1}^{N_1-1}\left(\frac{\varphi_{k_1 k_2}}{\lambda_{k_1}^{(1)} + \lambda_{k_2}^{(2)}}\right)\sin\left(\frac{k_1 \pi i}{N}\right); \quad 1 \le i, k_2 \le N-1;$$

$$u(i,j) = \frac{4}{N^2}\sum_{k_2=1}^{N-1} u_{k_2}(i)\sin\left(\frac{k_2 \pi j}{N}\right); \qquad 1 \le i, j \le N-1;$$

$$y_k = \sum_{j=1}^{N-1} a_i \sin\left(\frac{k \pi i}{n}\right);$$

- Тригонометрическое разложение, трудоемкость C*N*log(N)

Вычислительный алгоритм (однократный ряд)

$$\varphi_{k_1 k_2} = \sum_{i=1}^{N-1} \varphi_{k_2}(i) \sin\left(\frac{k_1 \pi i}{N}\right); \qquad 1 \le k_1, k_2 \le N-1;$$

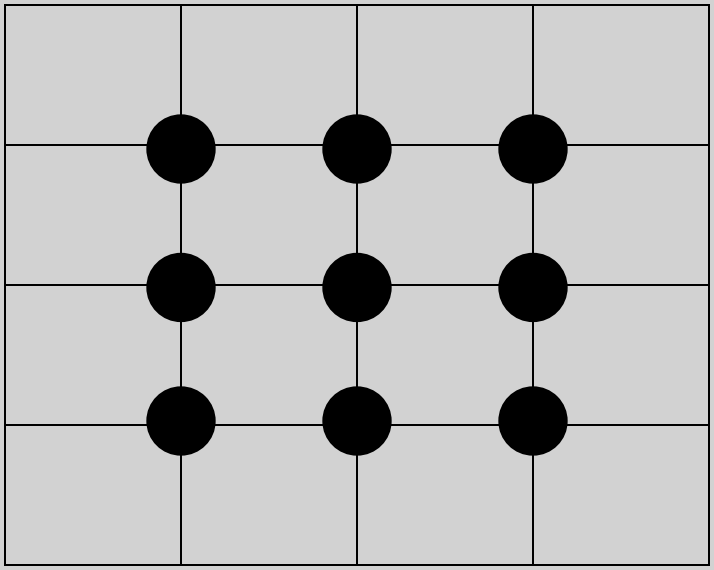
$$u_{k_2}(i) = \sum_{k_1=1}^{N_1-1} \left(\frac{\varphi_{k_1 k_2}}{\lambda_{k_1}^{(1)} + \lambda_{k_2}^{(2)}}\right) \sin\left(\frac{k_1 \pi i}{N}\right); \quad 1 \le i, k_2 \le N-1;$$

$$\Lambda_2 u_{k_2} - \lambda_{k_2}^{(2)} u_{k_2}(i) = \varphi_{k_2}(i); \qquad 1 \le i, k_2 \le N-1;$$

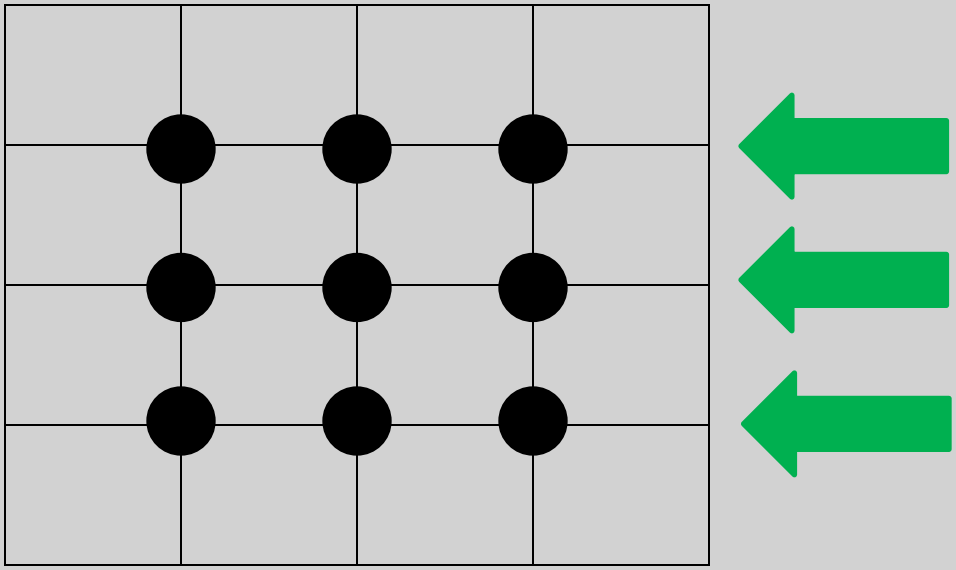
$$u_{k_2}(0) = u_{k_2}(N) = 0;$$

- Серия из трехдиагональных матриц, трудоемкость решения 6*N

Вычислительный алгоритм (однократный ряд)

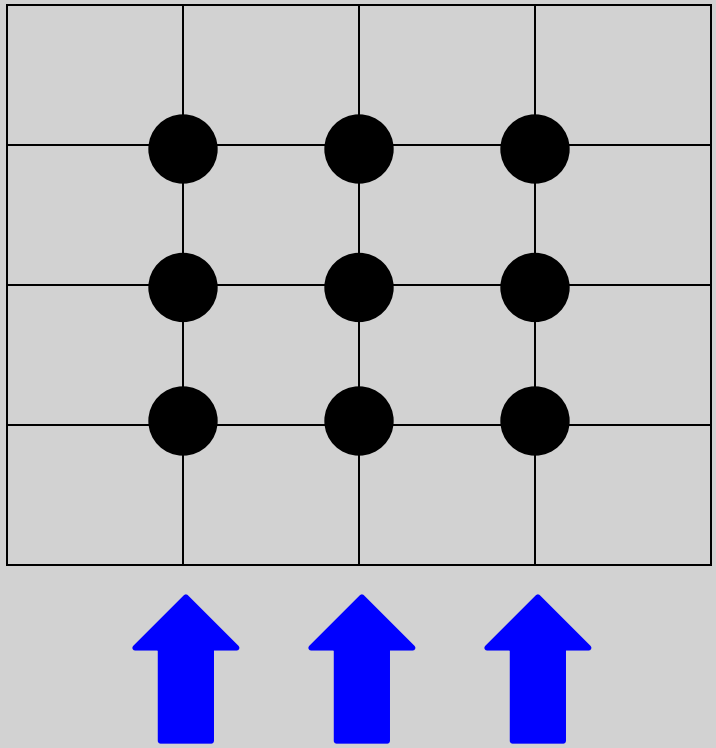


Вычислительный алгоритм (однократный ряд)



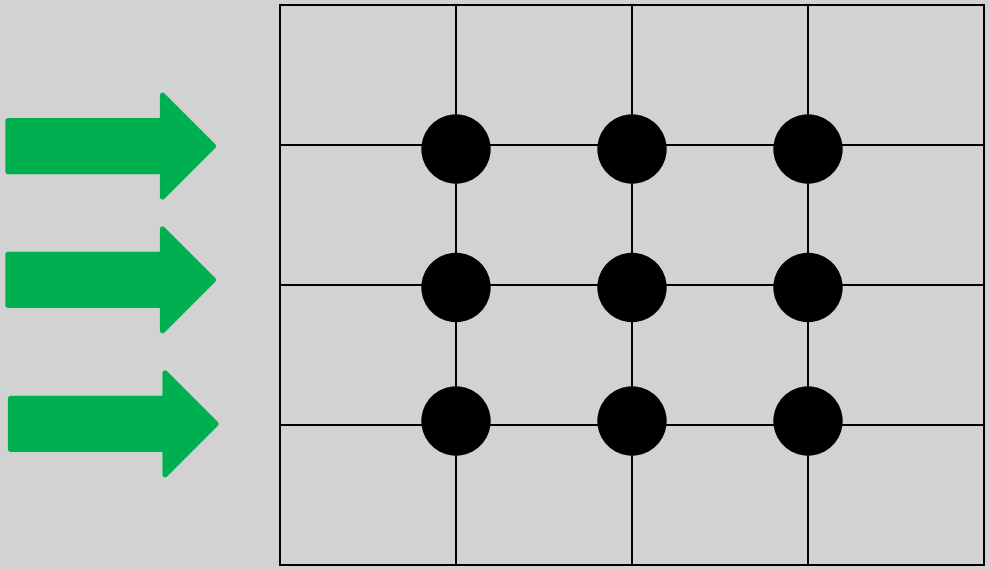
Разложение по синусам горизонтальных компонент.
Присутствует в библиотеках Intel MKL, Netlib

Вычислительный алгоритм (однократный ряд)



Решение трехдиагональных систем (каждая система разная!!! к диагонали добавляютя различные собственные числа)

Вычислительный алгоритм (однократный ряд)



Обратное разложение по синусам горизонтальных компонент. Присутствует в тех же библиотеках

Вычислительный алгоритм (однократный ряд)

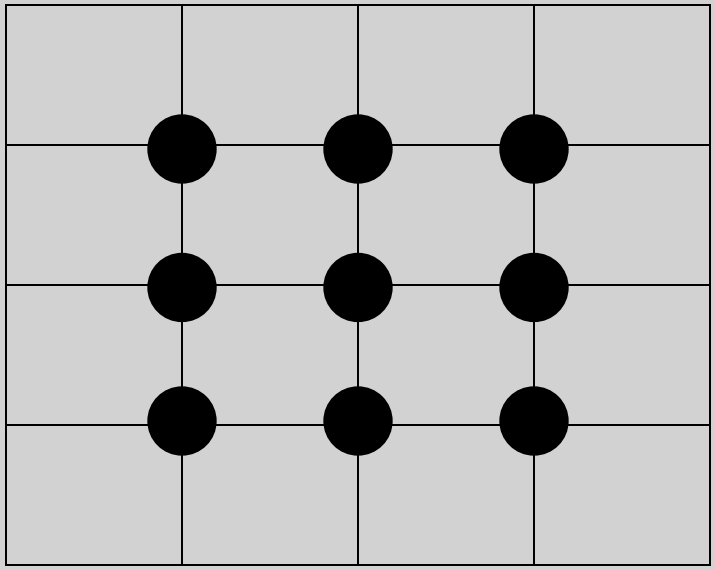
```
subroutine laplas_solution(f(*,*),N)
……

Do i=1,N
  Call forward_trig_transform(f(i,*),….)
enddo



Do j=1,N
  Call three_diagonal_lu(f(*,j),lambda(j),…)
enddo


Do i=1,N
  Call backward_trig_transform(f(i,*),….)
enddo

…..
End subroutine
```

Вычислительный алгоритм (однократный ряд) (OMP)

```
subroutine laplas_solution(f(*,*),N)
……
C$OMP PARALLEL DO
Do i=1,N
  Call forward_trig_transform(f(i,*),….)
enddo
C$OMP END PARALLEL DO

C$OMP PARALLEL DO
Do j=1,N
  Call three_diagonal_lu(f(*,j),lambda(j),…)
enddo
C$OMP END PARALLEL DO

C$OMP PARALLEL DO
Do i=1,N
  Call backward_trig_transform(f(i,*),….)
enddo
C$OMP END PARALLEL DO

…..
End subroutine
```

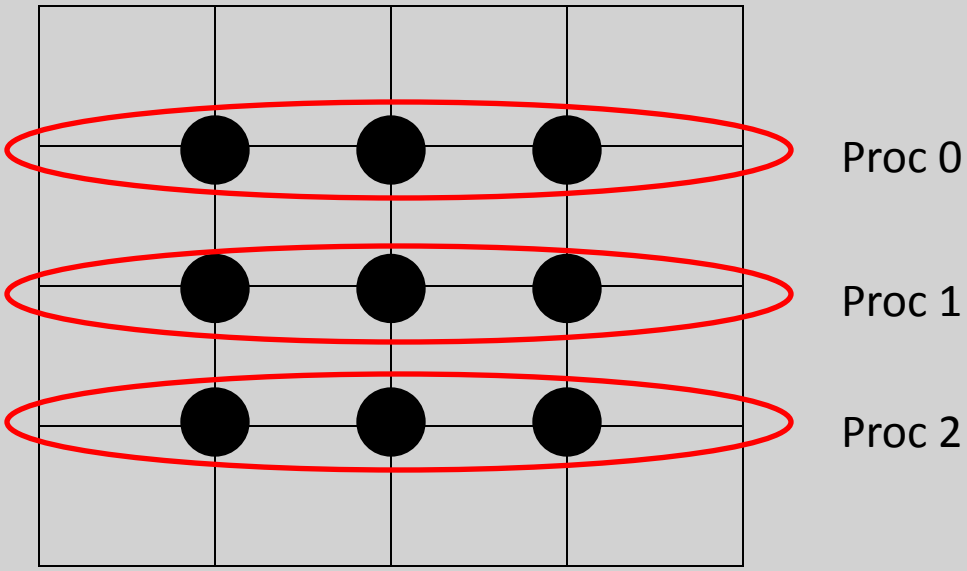
Вычислительный алгоритм (однократный ряд) (MPI)

Для MPI идеалогии такое неподходит, так как данные распределены по различным процессам, следовательно невозможно иметь либо LU на одном процессе (если данные распределены по строкам), либо разложение по синусам (если данные распределены по столбцам), либо оба (если данные распределены “шахматкой”).



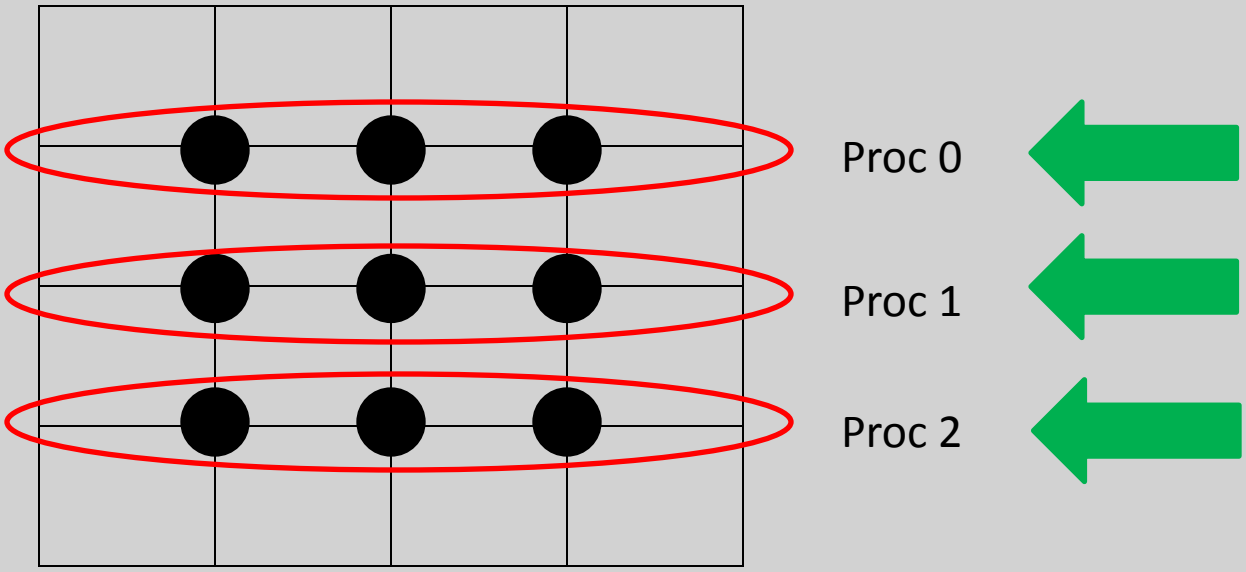
Вариант 1 (транспонирование, MPI)

Пусть данные распределены по строкам



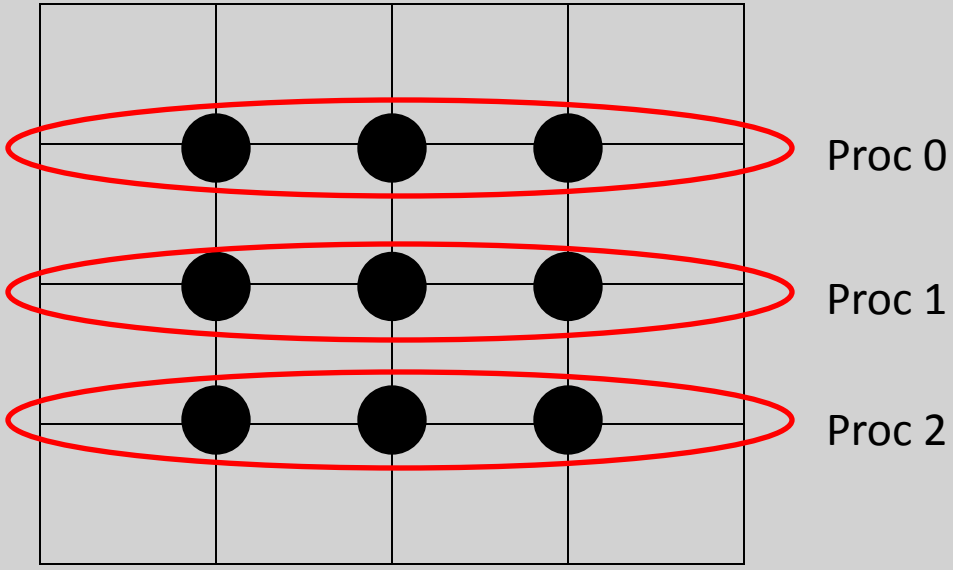
Вариант 1 (транспонирование, MPI)

Так как каждая строка целиком лежит на одном процессе, то каждый процесс независимо может сделать разложение по синусам



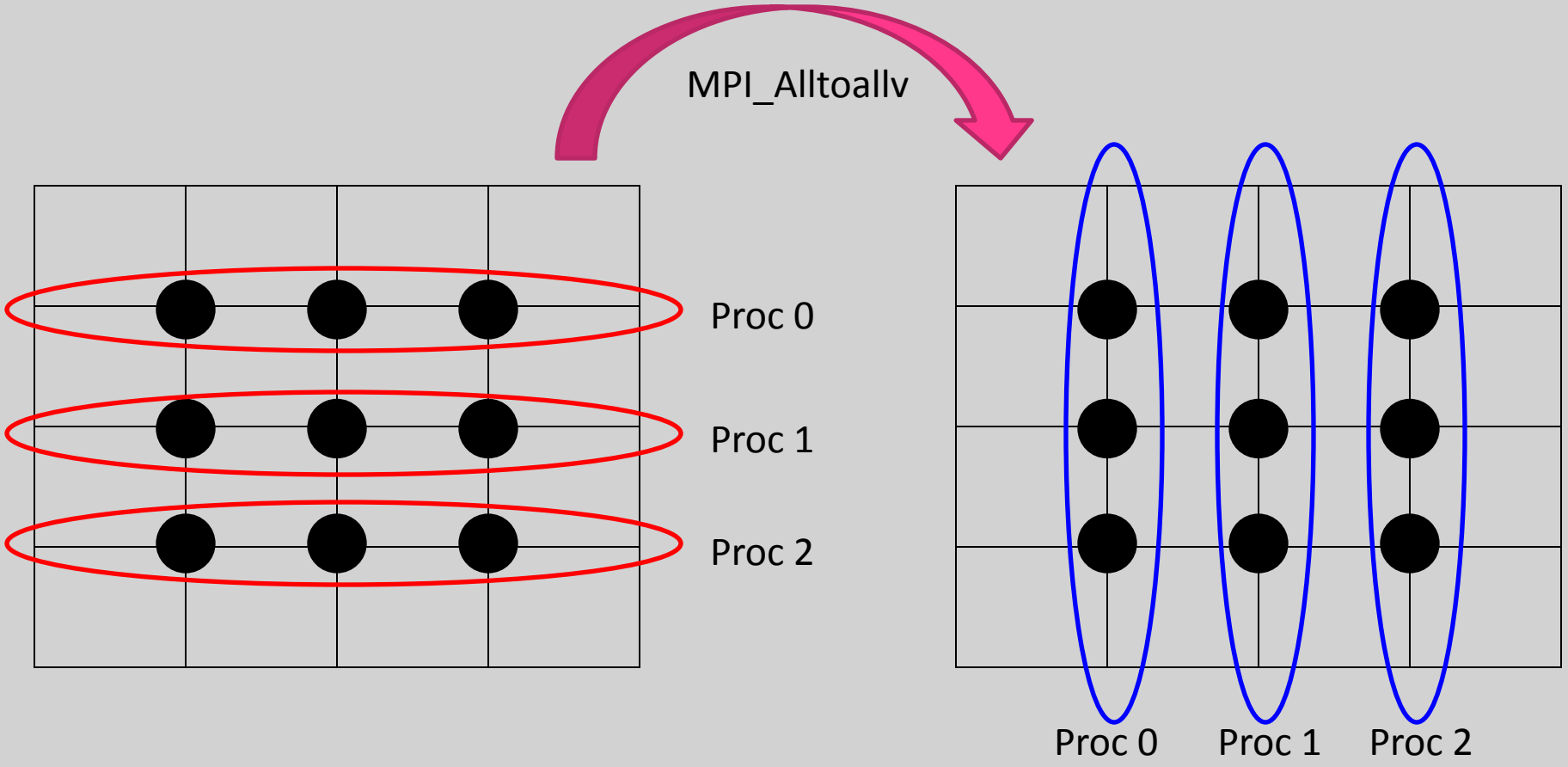
Вариант 1 (транспонирование, MPI)

Обычный алгоритм прогонки не может быть реализован, так как компоненты правых частей лежат на разных процессах



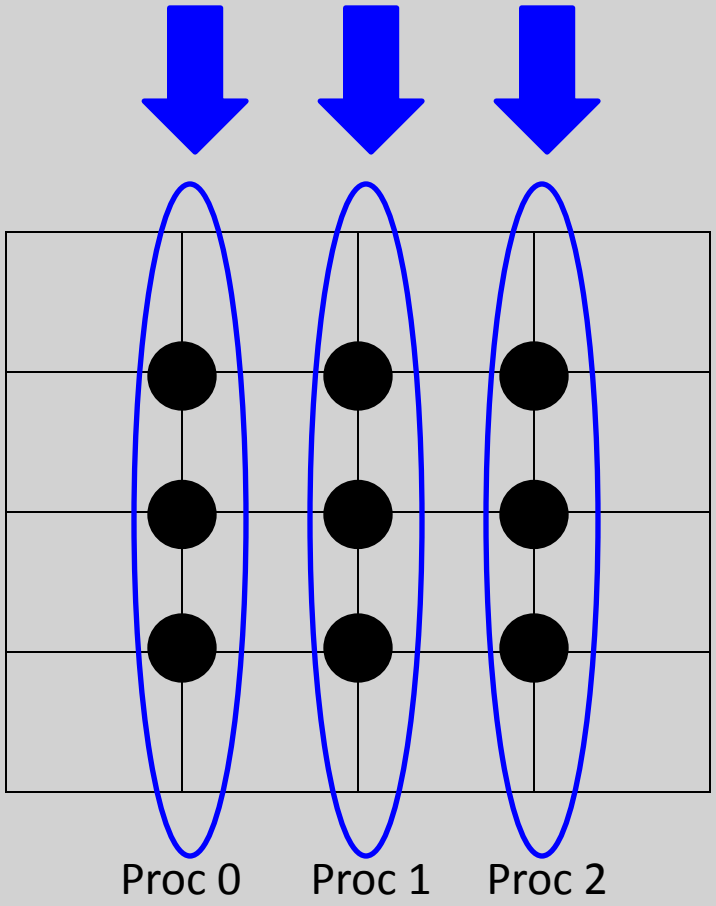
Вариант 1 (транспонирование, MPI)

Данные можно транспонировать между процессами.
ВАЖНО! Не забыть про порядок нумерации на процессах



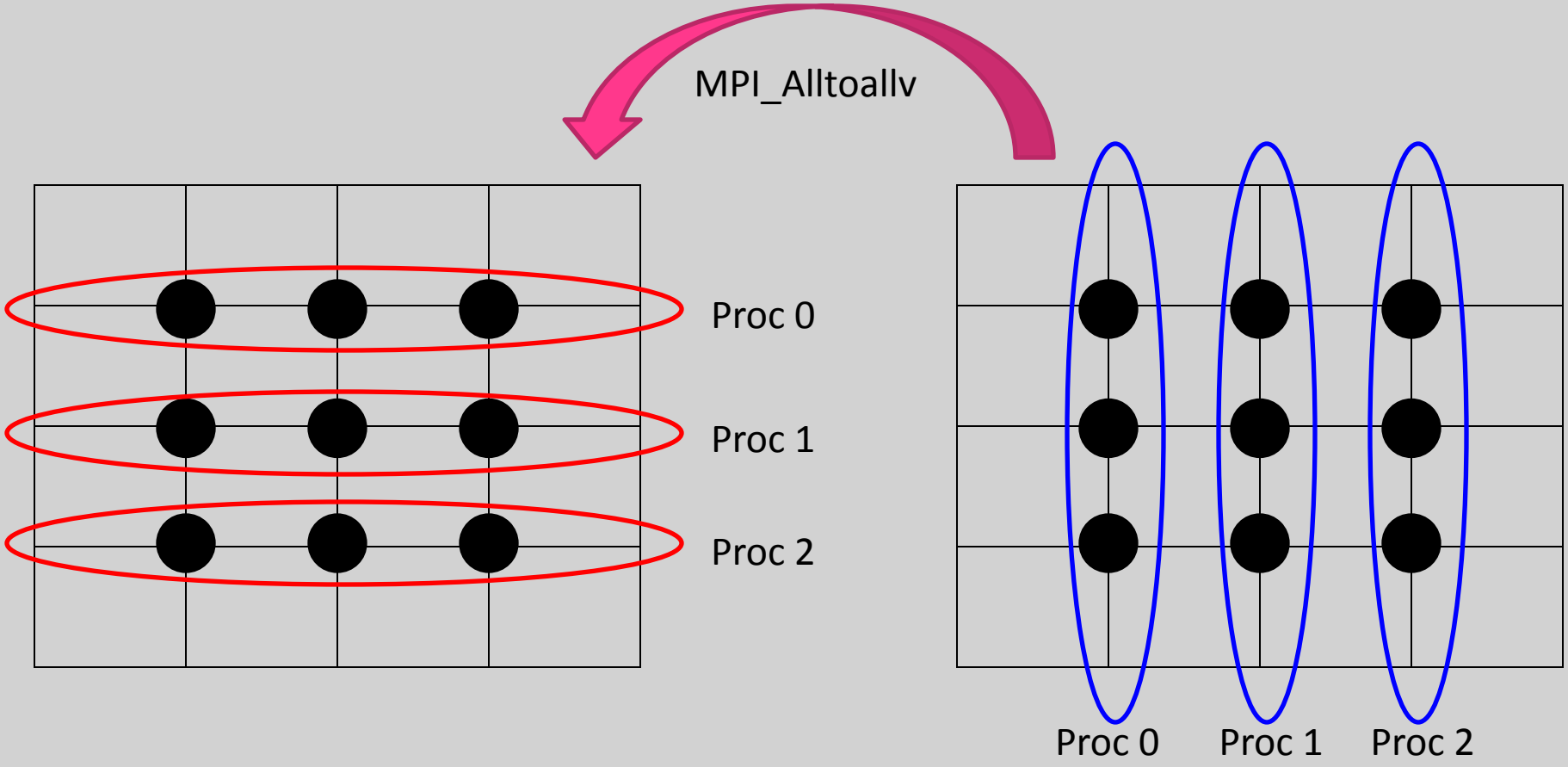
Вариант 1 (транспонирование, MPI)

После транспонирования данных правые части для прогонок лежат на одном процессе, следовательно может быть выполнен обычный алгоритм



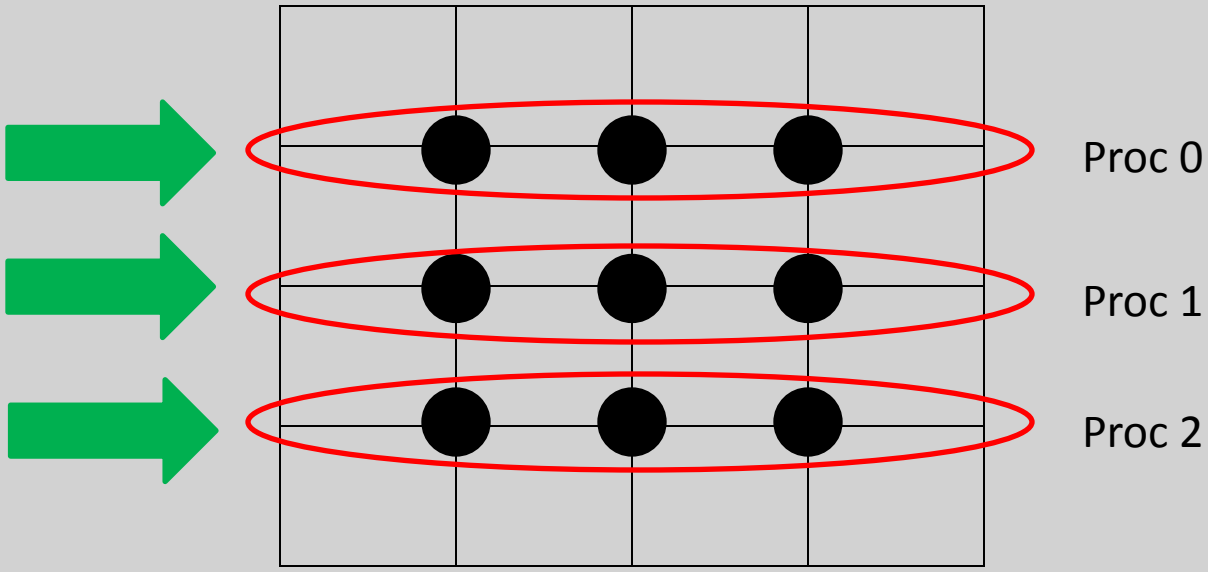
Вариант 1 (транспонирование, MPI)

Данные необходимо вернуть в изначальное расположения для релизации обратного разложения по синусам



Вариант 1 (транспонирование, MPI)

Теперь каждая строка опять целиком лежит на одном процессе, и каждый процесс независимо может сделать обратное разложение по синусам

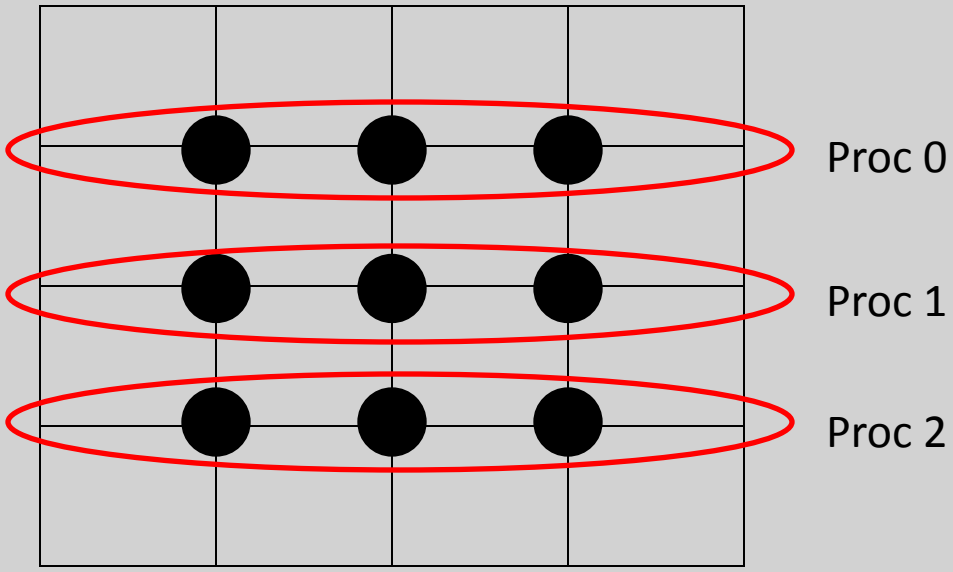


Вариант 1 (транспонирование, MPI)

```
subroutine laplas_solution(f(*,*),N)
……
Call MPI_Init();

Do i=ny_first,ny_last
  Call forward_trig_transform(f(i-ny_first+1,*),….)
enddo

call transpose _y_to_x(f,f_work,…)
call MPI_All_to_allv(f_work,f,…,MPI_COMM_WORLD)

Do j=nx_first,nx_last
  Call three_diagonal_lu(f(*,j-nx_first+1),lambda(j),…)
enddo

call MPI_All_to_allv(f,f_work,…,MPI_COMM_WORLD)
call transpose _x_to_y(f_work,f,…)

Do i=ny_first,ny_last
  Call backward_trig_transform(f(i-ny_first+1,*),….)
Enddo

…..
Call MPI_Finalize();
End subroutine
```

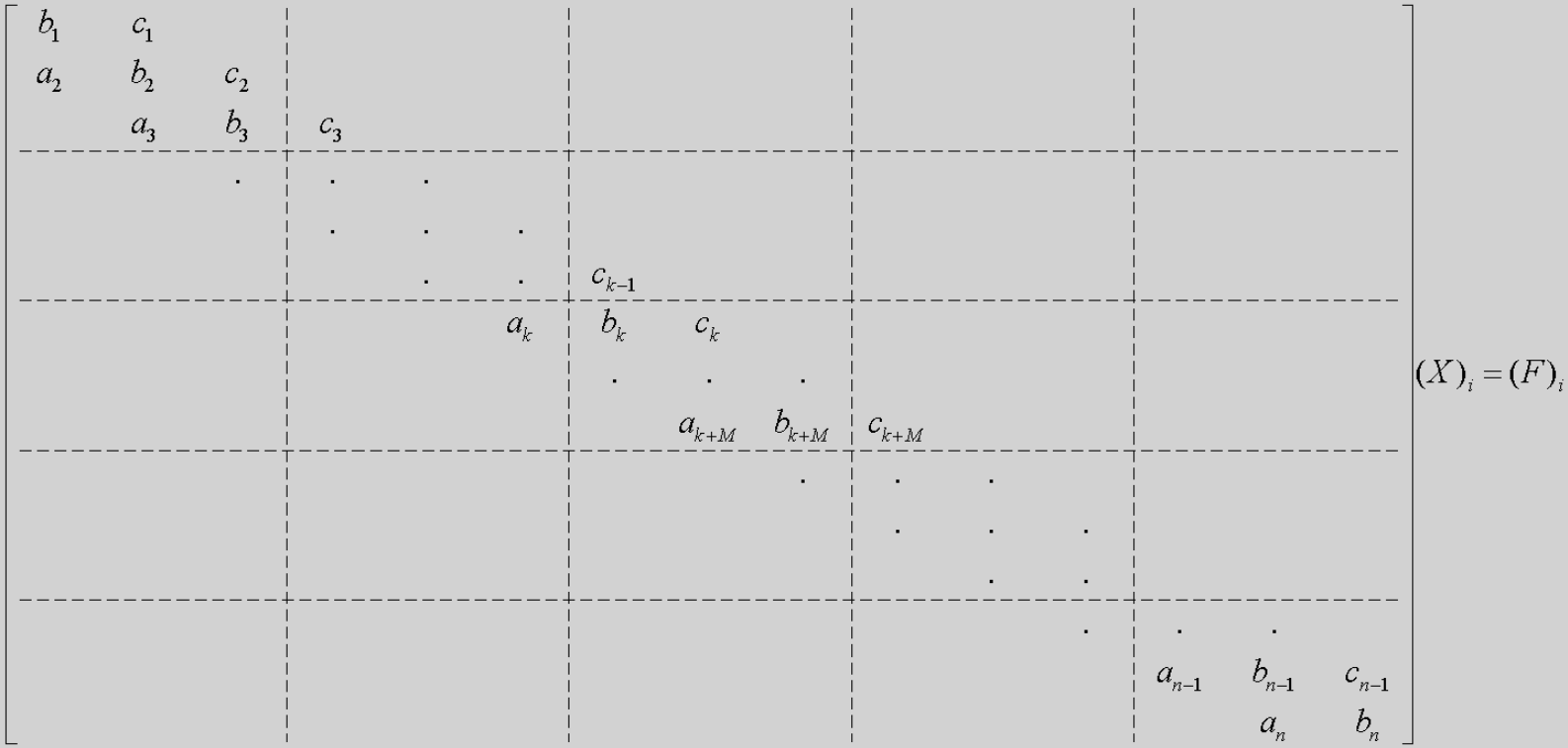
Вариант 2 (параллельная прогонка, MPI)

Так как обмены данных может быть достаточно затратной операций, предлагается аглоритм параллельной прогонки (прогонка для правой части, распределенной между процессами)



Вариант 2 (параллельная прогонка, MPI)

Разобьем компоненты вектора решения $(X)_i$ и правой части $(F)_i$ между процессами таким образом, чтоб на каждом процессоре лежала часть вектора с компонентами [k,k+M], каждая компонента хранилась только на одном процессе.



Вариант 2 (параллельная прогонка, MPI)

Теорема: Если вектор решения записать в следующем виде:

$$X = \left(x_0^1,..,x_M^1,x_0^2,...,x_0^{ya},...,x_M^{ya},...,x_M^{\max\_proc}\right) = (X^1,..,X^{ya},...,X^{\max\_proc})$$

,где max_proc – число процессов, то решение задачи с трехдиагональной матрицей записывается в следующем виде:

$$X^{ya} = R^{ya} + P^{ya} \cdot W_{ya-1} + Q^{ya} \cdot W_{ya}; \quad W_i = x_0^{i+1}$$

$$a_i P_{j-1}^{ya} + b_i P_j^{ya} + c_i P_{j+1}^{ya} = 0; \;\; P_0^{ya} = 1, P_M^{ya} = 0;$$

$$a_i Q_{j-1}^{ya} + b_i Q_j^{ya} + c_i Q_{j+1}^{ya} = 0; \;\; Q_0^{ya} = 0, P_M^{ya} = 1;$$

$$a_i R_{j-1}^{ya} + b_i R_j^{ya} + c_i R_{j+1}^{ya} = f_j; \;\; R_0^{ya} = 0, R_M^{ya} = 0;$$

$$j = \overline{0,M}; i = (ya-1)M + j$$

Вариант 2 (параллельная прогонка, MPI)

Теорема: Если вектор решения записать в следующем виде:

$$X = \left(x_0^1,..,x_M^1,x_0^2,...,x_0^{ya},...,x_M^{ya},...,x_M^{\max\_proc}\right) = (X^1,..,X^{ya},...,X^{\max\_proc})$$

,где max_proc – число процессов, то решение задачи с трехдиагональной матрицей записывается в следующем виде:

$$X^{ya} = R^{ya} + P^{ya} \cdot W_{ya-1} + Q^{ya} \cdot W_{ya}; \quad W_i = x_0^{i+1}$$

$$a_i P_{j-1}^{ya} + b_i P_j^{ya} + c_i P_{j+1}^{ya} = 0; \;\; P_0^{ya} = 1, P_M^{ya} = 0;$$

$$a_i Q_{j-1}^{ya} + b_i Q_j^{ya} + c_i Q_{j+1}^{ya} = 0; \;\; Q_0^{ya} = 0, P_M^{ya} = 1;$$

$$a_i R_{j-1}^{ya} + b_i R_j^{ya} + c_i R_{j+1}^{ya} = f_j; \;\; R_0^{ya} = 0, R_M^{ya} = 0;$$

$$j = \overline{0,M}; i = (ya-1)M + j$$

Относительно W_k записывается трехдиагональная система уравнений, с коэффициентами, зависящими от a,b,c,P,Q.

Вариант 2 (параллельная прогонка, MPI)

Алгоритм:

1. Решаем на каждом процессора 3 системы уравнений методом прогонки с одной и той же матрицей, разными правыми частями

2. Рассылаем с каждого процесса малый объем данных на один процесс для того, чтоб собрать трехдиагональную систему уравнений размерностью=число процессов

3. Полученные компоненты решения рассылаем на каждый процесс (по два значения на процесс) и собираем на них требуемые компоненты решения

Вариант 1 (транспонирование, MPI)

```
subroutine laplas_solution(f(*,*),N)
……
Call MPI_Init();

Do i=ny_first,ny_last
  Call forward_trig_transform(f(i-ny_first+1,*),….)
enddo


Do j=1,nx
  Call MPI_three_diagonal_lu(f(*,j),lambda(j),…)
enddo


Do i=ny_first,ny_last
  Call backward_trig_transform(f(i-ny_first+1,*),….)
Enddo

…..
Call MPI_Finalize();
End subroutine
```

Вопросы и Ответы